

Programação VBA no Excel



Autor: Júlio Battisti

site: www.juliobattisti.com.br

e-mail: webmaster@juliobattisti.com.br

Nota sobre direitos autorais:

Este ebook é de autoria de Júlio Battisti, sendo comercializado diretamente através do site www.juliobattisti.com.br ou através do site de leilões Mercado Livre: www.mercadolivre.com.br.

Ao adquirir este ebook você tem o direito de lê-lo na tela do seu computador e de imprimir quantas cópias desejar. É vetada a distribuição deste arquivo, mediante cópia ou qualquer outro meio de reprodução, para outras pessoas. Se você recebeu este ebook através do e-mail ou via ftp de algum site da Internet, ou através de um CD de Revista, saiba que você está com uma cópia pirata, não autorizada. O valor cobrado por este arquivo é praticamente simbólico pelas horas e horas de trabalho que ele representa. Novos cursos somente podem ser desenvolvidos pela honestidade de pessoas que adquirem o arquivo do curso e não o distribuem livremente para outras pessoas. Se você recebeu uma cópia deste arquivo sem tê-la adquirido diretamente com o autor, seja honesto, entre em contato com o autor, através do e-mail webmaster@juliobattisti.com.br, para regularizar esta cópia.

Ao regularizar a sua cópia você estará remunerando, mediante uma pequena quantia, o trabalho do autor e incentivando que novos trabalhos sejam disponibilizados.




Se você tiver sugestões sobre novos cursos que gostaria de ver disponibilizados, entre em contato pelo e-mail: webmaster@juliobattisti.com.br.

Visite periodicamente o site www.juliobattisti.com.br para ficar por dentro das novidades:

- Cursos de informática.
- Artigos e dicas sobre Certificações da Microsoft.
- Artigos sobre Carreira e Trabalho.
- Dicas de livros e sites sobre diversos assuntos.
- Simulados gratuitos, em português, para os exames da Microsoft.

PIRATARIA É CRIME, COM PENA DE CADEIA. EU AGRADEÇO PELA SUA HONESTIDADE. SE VOCÊ COMPROU UMA CÓPIA DESTA CURSO, DIRETAMENTE COM O AUTOR, NÃO DISTRIBUA CÓPIAS PARA OUTRAS PESSOAS. SE VOCÊ RECEBEU UMA CÓPIA ILEGAL DESTA ARQUIVO, NÃO ADQUIRIDA DIRETAMENTE COM O AUTOR JÚLIO BATTISTI, ENTRE EM CONTATO E REGULARIZE A SUA CÓPIA.

Conheça outros livros do autor Júlio Battisti:

	<p>Windows Server 2003 – Curso Completo – 1568 páginas</p> <p>O livro ensina desde os fundamentos básicos do Active Directory, passando pela instalação do Windows Server 2003 e por dicas sobre o projeto, implementação e migração do Windows 2000 Server para o Windows Server 2003. Você aprenderá, em detalhes, sobre os serviços de compartilhamento de arquivos e impressoras, segurança, como tornar o Windows Server 2003 um servidor Web, aprenderá sobre os serviços de rede: DNS, DHCP, WINS, RRAS, IPSec, Análise de Segurança, Group Policy Objects e muito mais. Confira, vale a pena.</p>
	<p>Manual de Estudos Para o Exame 70-216 - 712 páginas</p> <p>Neste aguardado lançamento da Axcel Books Editora – Certificação Microsoft – Guia de Estudos Para o MCSE – Exame 70-216, o autor Júlio Battisti descreve, de forma detalhada e com exemplos passo-a-passo, todos os tópicos que fazem parte do programa oficial da Microsoft para o exame de certificação. A obra apresenta e explica desde os princípios básicos, incluindo os fundamentos do protocolo TCP/IP; passando por instalação, configuração e administração do DNS, DHCP, WINS e RRAS; além de ainda tratar de questões quanto ao roteamento, NAT, Certificados Digitais, IPSec, entre outros.</p>
	<p>Windows XP Home & Professional – 840 páginas</p> <p>O novo mundo do Windows XP, que representa a nova era do sistema operacional para usuários e administradores está reunido nesta obra. Júlio Battisti apresenta a nova interface do sistema, completamente redesenhada e com a experiência de um profissional certificado da Microsoft. Na obra, os leitores irão aprender a implementar, configurar e utilizar o Windows XP, desvendando as funcionalidades, além das configurações de segurança, de desempenho e de estabilidade do sistema. O livro aborda ainda toda a parte de Internet do Windows XP – conectando e usando a Internet; configurando o firewall de conexão; além dos novos recursos do correio eletrônico. Veja também os detalhes sobre o Active Directory, as configurações de rede e protocolo TCP/IP, criptografia, registry do Windows, entre tantos outros assuntos. O leitor ainda vai poder contar com um capítulo exclusivo e um simulado com 100 questões/respostas destinados aos interessados no exame de Certificação 70-270 da Microsoft.</p>

	<p>ASP.NET: Uma Nova Revolução Na Criação de Sites e Aplicações Web – 730 páginas</p> <p>Conheça o ASP.NET, a mais nova versão do ASP, que representa uma mudança no modelo de desenvolvimento de aplicações Web. O livro traz todas as informações necessárias sobre o assunto, inclusive os detalhes da iniciativa .NET, o CLR, o MSIL e o C#, a nova linguagem da Microsoft. Aprenda os novos controles do ASP.NET e como utilizar o Visual Studio.NET para criar páginas ASP.NET. Veja ainda como criar formulários avançados para edição de dados, configurar as opções de segurança do Windows 2000, do IIS e do ASP.NET, além de aprender como criar páginas ASP.NET para as mais diversas funções.</p>
	<p>SQL Server 2000: Administração & Desenvolvimento Curso Completo – 816 páginas</p> <p>O lançamento é destinado aos usuários/leitores da versão anterior do SQL Server, o SQL 7, além de redes de computadores em geral, Windows 2000 Server, TCP/IP, Bancos de Dados em geral, do Microsoft Access e do Visual Basic. O leitor aprenderá na obra destinada do iniciante ao avançado detalhes sobre o modelo de dados relacional, como instalar o SQL Server 2000 em diferentes plataformas, além da criação e administração de bancos de dados, tabelas e outros objetos. Aprenda ainda Como criar páginas ASP que acessam os dados do SQL Server 2000.</p>

PRÉ-REQUISITOS PARA O CURSO:

Para que você possa acompanhar as lições deste curso é necessário que você já tenha preenchido os seguintes pré-requisitos:

- Curso básico de Excel em 120 Lições, disponível no seguinte endereço: (<http://www.juliobattisti.com.br/excel120/excel120.asp>) ou conhecimento equivalente.
- Curso de Excel Avançado em 120 Lições, disponível no seguinte endereço: (<http://www.juliobattisti.com.br/excel120avancado/excel120avancado.asp>) ou conhecimento equivalente.
- Conhecimento dos aspectos básicos do Modelo Relacional de banco de dados. Para saber mais sobre o Modelo Relacional de dados, consulte as seguintes lições do curso básico de Access, disponível no meu site, no seguinte endereço:
 - <http://www.juliobattisti.com.br/accbasico/modulo1/licao2.htm>
 - <http://www.juliobattisti.com.br/accbasico/modulo1/licao3.htm>
 - <http://www.juliobattisti.com.br/accbasico/modulo1/licao4.htm>
 - <http://www.juliobattisti.com.br/accbasico/modulo1/licao5.htm>
 - <http://www.juliobattisti.com.br/accbasico/modulo1/licao6.htm>

Estes conhecimentos serão fundamentais para acompanhar os tópicos apresentados no curso. Muitos dos tópicos dependem destes pré-requisitos.

Orientações sobre os arquivos de Exemplos: Antes de iniciar o curso, crie uma pasta chamada Excel Avançado e descompacte os arquivos de exemplos do curso, dentro desta pasta.

Algumas palavras do autor:

Este curso foi criado com o objetivo de ajudá-lo a entender e a utilizar no seu dia-a-dia, a programação VBA – Visual Basic for Applications, para implementar soluções com o Microsoft Excel. O curso é composto de 6 Módulos, com 25 lições por módulo.

Em cada lição são apresentados conceitos teóricos, seguidos por exemplos práticos, passo-a-passo, para que você possa consolidar os conceitos teóricos apresentados.

Um bom estudo a todos e espero, sinceramente, que este curso possa ajudá-los a utilizar melhor a programação VBA no Microsoft Excel.

ÍNDICE DO CURSO

Introdução.....	9
Módulo 1 – Introdução às Macros e ao VBA no Excel.....	11
Lição 01: Uma Introdução às Macros.....	12
Lição 02: O que são exatamente Macros???	15
Lição 03: Conhecendo do que é feita uma Macro	18
Lição 04: Operações com Macros	21
Lição 05: Associando botões com macros	23
Lição 06: Introdução ao VBA.....	27
Lição 07: O Ambiente de programação – o Editor VBA – Parte I	30
Lição 08: O Ambiente de programação – o Editor VBA – Parte 2	33
Lição 09: VBA - Declaração de Variáveis.....	36
Lição 10: VBA - Cálculos, Operadores Aritméticos e Exemplos.....	40
Lição 11: VBA - Estrutura If...Then e os Operadores de Comparação	45
Lição 12: Estruturas For...Next, Do...While e Do...Until.....	50
Lição 13: VBA - Funções do VBA – Funções de Tipo – Parte 1	56
Lição 14: VBA - Funções do VBA – Funções de Tipo – Parte 2.....	61
Lição 15: VBA - Funções do VBA – Funções de Conversão de Tipo – Parte 1	64
Lição 16: VBA - Funções do VBA – Funções de Conversão de Tipo – Parte 2	68
Lição 17: VBA - Funções do VBA – Funções Para Tratamento de Texto	70
Lição 18: VBA - Funções do VBA – Data, Hora e Funções Matemáticas.....	74
Lição 19: VBA - O Conceito de Módulos, Procedimentos e Funções – Parte I	82
Lição 20: VBA - O Conceito de Módulos, Procedimentos e Funções – Parte II	89
Lição 21: VBA – Criando Funções Personalizadas – Parte I.....	92
Lição 22: VBA – Um exemplo prático – calculando o DV do CPF - Algoritmo.....	96
Lição 23: Usando o VBA Para Criar uma Função de Validação do DV do CPF.....	98
Lição 24: Usando a Função ValidaCPF, Criada na Lição 21.....	102
Lição 25: Mais Alguns Exemplos de Funções Personalizadas.....	103
Lição 26: Mais Alguns Exemplos de Funções Personalizadas.....	109
Lição 27: Conclusão do Módulo 1	113
Módulo 2 – O Modelo de Objetos do Excel	114
Lição 01: O que é um Modelo de Objetos?	115
Lição 02: Descrição dos Principais Objetos do Modelo de Objetos do Excel.....	119
Lição 03: O Microsoft Object-Browser: Navegando pela Hierarquia de Objetos	125
Lição 04: Objeto Application – O Pai de Todos - Introdução	128
Lição 05: Objeto Application – Como Declarar e Utilizar	131
Lição 06: Objeto Application – Propriedades que Retornam Objetos Filho	135
Lição 07: Objeto Application – Exibindo/Ocultando itens do Excel.....	139
Lição 08: Objeto Application – Habilitando/Desabilitando Recursos do Excel.....	142
Lição 09: Objeto Application – Associando Macros à Teclas Especiais	144
Lição 10: Objeto Application – Operações com Arquivos.....	148
Lição 11: Objeto Application – Recálculo da Planilha	153
Lição 12: Conceitos Avançados na Criação de Funções e Procedimentos.....	155
Lição 13: Conceitos Avançados na Criação de Funções e Procedimentos.....	158
Lição 14: Conceitos Avançados na Criação de Funções e Procedimentos.....	163
Lição 15: Conceitos Avançados na Criação de Funções e Procedimentos.....	166
Lição 16: Conceitos Avançados na Criação de Funções e Procedimentos.....	170
Lição 17: Conceitos Avançados na Criação de Funções e Procedimentos.....	172
Lição 18: A função MsgBox em Detalhes.....	178
Lição 19: A função InputBox em Detalhes.....	181
Lição 20: O Tratamento de Erros no VBA – Parte 1	185
Lição 21: O Tratamento de Erros no VBA – Parte 2.....	189
Lição 22: O Tratamento de Erros no VBA – Parte 3.....	192
Lição 23: O Tratamento de Erros no VBA – Parte 4.....	194
Lição 24: O Tratamento de Erros no VBA – Parte 5.....	198
Lição 25: O Tratamento de Erros no VBA – Parte 6.....	201

Lição 26: Conclusão do Módulo 2	204
Módulo 3 – O Objeto Range e Exemplos Práticos.....	205
Lição 01: Apresentação do Objeto Range.....	206
Lição 02: Objeto Range – Outras Maneiras de Criar um Objeto Range.....	211
Lição 03: Objeto Range – Outras Maneiras de Criar um Objeto Range.....	216
Lição 04: Objeto Range – Column, Columns, Row e Rows – Parte 1.....	219
Lição 05: Objeto Range – Column, Columns, Row e Rows – Parte 2.....	223
Lição 06: Objeto Range – Principais Métodos e Propriedades – Parte 1.....	226
Lição 07: Objeto Range – Principais Métodos e Propriedades – Parte 2.....	231
Lição 08: Objeto Range – Principais Métodos e Propriedades – Parte 3.....	235
Lição 09: Objeto Range – Principais Métodos e Propriedades – Parte 4.....	237
Lição 10: Objeto Range – Principais Métodos e Propriedades – Parte 5.....	242
Lição 11: Objeto Range – Principais Métodos e Propriedades – Parte 6.....	246
Lição 12: Objeto Range – Principais Métodos e Propriedades – Parte 7.....	249
Lição 13: Objeto Range – Principais Métodos e Propriedades – Parte 8.....	253
Lição 14: Objeto Range – Principais Métodos e Propriedades – Parte 9.....	257
Lição 15: Objeto Range – Principais Métodos e Propriedades – Parte 10.....	262
Lição 16: Exemplos Práticos de Uso dos Objetos do Excel – Parte 1.....	265
Lição 17: Exemplos Práticos de Uso dos Objetos do Excel – Parte 2.....	269
Lição 18: Exemplos Práticos de Uso dos Objetos do Excel – Parte 3.....	272
Lição 19: Exemplos Práticos de Uso dos Objetos do Excel – Parte 4.....	274
Lição 20: Exemplos Práticos de Uso dos Objetos do Excel – Parte 5.....	277
Lição 21: Exemplos Práticos de Uso dos Objetos do Excel – Parte 6.....	281
Lição 22: Exemplos Práticos de Uso dos Objetos do Excel – Parte 7.....	285
Lição 23: Exemplos Práticos de Uso dos Objetos do Excel – Parte 8.....	289
Lição 24: Exemplos Práticos de Uso dos Objetos do Excel – Parte 9.....	292
Lição 25: Resumo do Módulo.....	295
Módulo 4 – Estudo dos Objetos Workbook e Worksheet.....	296
Lição 01: Apresentação dos Objetos Workbook e Worksheet.....	297
Lição 02: Objeto Workbook e Coleção Workbooks – Métodos e Propriedades	301
Lição 03: Objeto Workbook e Coleção Workbooks – Métodos e Propriedades	304
Lição 04: Objeto Workbook e Coleção Workbooks – Métodos e Propriedades	308
Lição 05: Mais Métodos e Propriedades dos Objetos Workbook e Worksheets	311
Lição 06: Mais Métodos e Propriedades dos Objetos Workbook e Worksheets	314
Lição 07: O Objeto WorkSheet – Métodos e Propriedades – Parte 1	317
Lição 08: O Objeto WorkSheet – Métodos e Propriedades – Parte 2	320
Lição 09: O Objeto WorkSheet – Métodos e Propriedades – Parte 3	323
Lição 10: O Objeto WorkSheet – Métodos e Propriedades – Parte 4	327
Lição 11: Eventos – Conceitos e Definições	330
Lição 12: Eventos – Eventos do Objeto Worksheet	334
Lição 13: Eventos – Eventos do Objeto Worksheet	338
Lição 14: Eventos – Eventos do Objeto Workbook	341
Lição 15: Eventos – Eventos do Objeto Workbook	344
Lição 16: Eventos – Eventos do Objeto Application	348
Lição 17: Eventos – Eventos do Objeto Application	351
Lição 18: Exemplos Práticos	354
Lição 19: Exemplos Práticos	357
Lição 20: Exemplos Práticos	360
Lição 21: Exemplos Práticos	363
Lição 22: Exemplos Práticos	366
Lição 23: Exemplos Práticos	369
Lição 24: Exemplos Práticos	372
Lição 25: Resumo do Módulo.....	375
Módulo 5 – Criação de Aplicações Usando UserForms	376
Lição 01: User Form – Introdução e Conceito	377
Lição 02: User Form – Criando um Novo User Form	380
Lição 03: User Form – Propriedades e Eventos.....	384
Lição 04: User Form – Trabalhando com a Caixa de Ferramentas.....	389

Lição 05	User Form – Trabalhando com a Caixa de Ferramentas	392
Lição 06:	User Form – Trabalhando com Controles – Parte 1	395
Lição 07:	User Form – Trabalhando com Controles – Parte 2	398
Lição 08:	User Form – Trabalhando com Controles – Parte 3	402
Lição 09:	User Form – Trabalhando com Controles – Parte 4	405
Lição 10:	User Form – Caixa de Combinação	409
Lição 11:	User Form – Propriedades dos Controles – Parte 1	412
Lição 12:	User Form – Propriedades dos Controles – Parte 2	415
Lição 13:	User Form – Propriedades dos Controles – Parte 3	418
Lição 14:	User Form – Propriedades dos Controles – Parte 4	421
Lição 15:	User Form – Propriedades dos Controles – Parte 5	424
Lição 16:	User Form – Propriedades dos Controles – Parte 6	428
Lição 17:	User Form – Propriedades dos Controles – Parte 7	431
Lição 18:	User Form – Propriedades dos Controles – Parte 8	434
Lição 19:	User Form – Propriedades dos Controles – Parte 9	437
Lição 20:	User Form – Propriedades dos Controles – Parte 10	440
Lição 21:	User Form – Propriedades dos Controles – Parte 11	443
Lição 22:	User Form – Propriedades dos Controles – Parte 12	446
Lição 23:	User Form – Propriedades dos Controles – Parte 13	449
Lição 24:	User Form – Propriedades dos Controles – Parte 14	452
Lição 25:	Resumo do Módulo	455
Módulo 6 –	Controles e Exemplos Práticos	456
Lição 01:	User Forms – O controle Caixa de Listagem	457
Lição 02:	User Forms – O controle Caixa de Seleção	460
Lição 03:	User Forms – O controle Botão de Opção - OptionButton	464
Lição 04:	User Forms – O controle Botão de ativação - ToggleButton	468
Lição 05:	User Forms – Os controles Botão de Comando e Frame	472
Lição 06:	User Forms – O controle Barra de Rolagem	475
Lição 07:	User Forms – O controle Botão de Rotação	478
Lição 08:	User Forms – O controle Image	481
Lição 09:	Exemplos práticos para o seu dia-a-dia	484
Lição 10:	Exemplos práticos para o seu dia-a-dia	487
Lição 11:	Exemplos práticos para o seu dia-a-dia	490
Lição 12:	Exemplos práticos para o seu dia-a-dia	493
Lição 13:	Exemplos práticos para o seu dia-a-dia	496
Lição 14:	Exemplos práticos para o seu dia-a-dia	499
Lição 15:	Exemplos práticos para o seu dia-a-dia	502
Lição 16:	Exemplos práticos para o seu dia-a-dia	505
Lição 17:	Exemplos práticos para o seu dia-a-dia	508
Lição 18:	Exemplos práticos para o seu dia-a-dia	511
Lição 19:	Exemplos práticos para o seu dia-a-dia	513
Lição 20:	Exemplos práticos para o seu dia-a-dia	515
Lição 21:	Exemplos práticos para o seu dia-a-dia	518
Lição 22:	Exemplos práticos para o seu dia-a-dia	521
Lição 23:	Exemplos práticos para o seu dia-a-dia	523
Lição 24:	Exemplos práticos para o seu dia-a-dia	525
Lição 25:	Resumo do Módulo	527

Introdução

Este é um curso sobre programação VBA no Microsoft Excel. Neste curso você aprenderá a utilizar uma série de comandos e objetos do Excel, para implementar soluções sofisticadas, as quais somente são possíveis de serem implementadas com o uso de programação. As telas e exemplos foram criados usando o Excel 2000, porém a quase totalidade dos comandos são válidos também para o Excel 97 e para o Excel XP.

A programação é utilizada para solucionar problemas no Excel, os quais não teriam como ser solucionados usando funções e comandos de planilha. Por exemplo, não existe uma função no Excel para a validação do DV de um CPF ou CNPJ. Neste caso, somente usando programação, o usuário poderá criar uma função personalizada, a qual faz a validação do CPF e do CNPJ. Neste curso você aprenderá, dentre outras coisas, a criar funções personalizadas, as quais atendem necessidades específicas. A seguir uma breve descrição do que será abordado em cada módulo do Curso.

Módulo 1: Neste módulo você aprenderá os fundamentos sobre Macros e VBA. Mostrarei exatamente o que é uma macro, o que é programação VBA, o ambiente de programação, as principais funções do VBA e como criar os primeiros programas. Os conceitos apresentados neste módulo serão fundamentais para os demais módulos do curso. Em todos os exemplos do curso, você irá utilizar um ou mais conceito apresentado neste módulo.

Módulo 2: Nas lições do Módulo 2 você aprenderá sobre o conceito de funções, sub-rotinas e módulos. Mostrarei como criar funções que podem ser utilizadas em várias planilhas de uma pasta de trabalho. Também apresentarei o conceito mais importante quando se trata de programação VBA; A Hierarquia de Objetos do Excel. Você verá que com o uso dos Objetos do Excel é possível acessar qualquer elemento de uma planilha. Apresentarei o primeiro objeto na Hierarquia de Objetos: O Objeto Application.

Módulo 3: Neste módulo continuarei o estudo sobre os objetos do Excel. Você aprenderá sobre os objetos Workbook e Worksheet, dois importantes objetos, utilizados em uma série de situações. Além da sintaxe, dos métodos e propriedades dos objetos, apresentarei uma série de exemplos práticos, os quais você poderá, facilmente, adaptar para uso nas rotinas que você desenvolver.

Módulo 4: Este módulo é sobre o objeto Range, sem dúvidas o objeto mais utilizado em programação VBA no Excel. O objeto Range representa uma ou mais faixas de células em um planilha do Excel. Qual problema que não envolve acessar dados de uma faixa de células e fazer cálculos ou pesquisas em uma faixa? Praticamente todos. Por isso que o objeto Range é, sem dúvidas, o objeto mais utilizado na programação VBA no Excel. Você aprenderá sobre os principais métodos e propriedades deste objeto, além de uma série de exemplos práticos, os quais ilustram detalhadamente como funciona e como pode ser utilizado o objeto Range.

Módulo 5: Neste módulo você aprenderá sobre UserForms. Com a utilização de UserForms você pode criar formulários personalizados, semelhantes aos que podem ser criados usando linguagens de programação como o Visual Basic ou Delphi. Mostrarei como criar um UserForm, quais os controles disponíveis e como utilizá-los, como funciona o modelo de eventos do Windows e uma série de exemplos práticos para que você possa começar a criar seus próprios formulários personalizados.

Módulo 6: Neste último módulo do curso continuarei o estudo de UserForms, apresentando um estudo dos diversos controles que podem ser utilizados em um UserForm. Na sequência apresentarei uma série de exemplos práticos, os quais você certamente utilizará no seu trabalho do dia-a-dia. Apresentarei exemplos de uso dos principais objetos da Hierarquia de objetos do Excel.

Este curso foi especialmente projetado para torná-lo mais produtivo com o Microsoft Excel. O domínio da programação VBA permite que você implemente soluções para problemas que não teriam solução, somente com o uso de Funções de Planilha e comandos do Excel. O domínio da programação VBA exige dedicação, estudo e muita experimentação. Sem nenhuma dúvida é um esforço que vale a pena.

É o meu mais sincero desejo que este curso possa ser de grande utilidade para você, ajudando-o a utilizar todos os recursos disponíveis nesta fantástica ferramenta que é a Programação VBA no Excel. Um bom estudo e muito sucesso.

Para enviar suas dúvidas referentes aos assuntos e exemplos abordados neste curso, para enviar sugestões de alterações/correções, para sugerir novos cursos, para criticar e para elogiar (porque não?), é só entrar em contato pelo e-mail: batisti@hotmail.com.

Módulo 1 – Introdução às Macros e ao VBA no Excel

Neste módulo você aprenderá os fundamentos sobre Macros e VBA. Mostrarei exatamente o que é uma macro, o que é programação VBA, o ambiente de programação, as principais funções do VBA e como criar os primeiros programas. Os conceitos apresentados neste módulo serão fundamentais para os demais módulos do curso. Em todos os exemplos do curso, você irá utilizar um ou mais conceito apresentado neste módulo.

Vou iniciar o módulo mostrando como criar uma macro e como verificar o Código VBA que é criado, o qual na prática é quem “faz o trabalho” da macro. Também mostrarei como associar teclas de atalho e botões de comando com uma Macro.

O próximo passo será aprender a utilizar o Ambiente de Programação do VBA, também chamado de Editor do VBA. Você aprenderá a criar código, a “navegar” através dos objetos disponíveis e a utilizar os vários recursos de ajuda fornecidos pelo Editor. Mostrarei as diversas partes que compõem o Editor do VBA e como utilizá-las.

Em seguida passarei ao estudo da linguagem VBA propriamente dita. Estes tópicos são a base da linguagem, os quais serão utilizados em todos os exemplos práticos do curso. Você aprenderá sobre os fundamentos do VBA, tais como:

- ➔ Declaração de variáveis
- ➔ Tipos de dados
- ➔ Operadores aritméticos
- ➔ Valores lógicos
- ➔ Operadores lógicos
- ➔ Estrutura de controle
- ➔ Estruturas de decisão
- ➔ Estruturas de repetição
- ➔ Exemplos de utilização

Seguindo o nosso estudo, apresentarei as principais funções internas do VBA. Farei a apresentação dividindo as funções em categorias, tais como funções de Data/Hora, funções de texto e assim por diante. Para encerrar o módulo você aprenderá sobre o conceito de Módulos, procedimentos e funções. Este conceito é muito importante para que você aprenda a criar código que possa ser reaproveitado. Isso aumenta, e muito, a sua produtividade no uso do VBA,

Então mãos à obra. Chega de apresentações e rodeios e vamos iniciar o nosso estudo da programação VBA no Excel. Um bom estudo a todos e não esqueça: em caso de dúvidas sobre os exemplos apresentados neste curso, entre em contato pelo e-mail: webmaster@juliobattisti.com.br

Lição 01: Uma Introdução às Macros

Introdução:

Existem situações onde não conseguimos resolver um determinado problema, simplesmente utilizando os comandos e fórmulas do Excel (embora existam milhares de funções disponíveis no Excel). Nessas situações temos que fazer o uso de recursos como Macros e Programação. A linguagem de programação do Excel é o **VBA – Visual Basic for Applications**. O VBA é a linguagem de programação para todos os aplicativos do Microsoft Office: Word, Excel, Access e PowerPoint.

Nas lições desse módulo você aprenderá sobre Macros e sobre os fundamentos da linguagem VBA. Mostrarei o que é uma Macro, para que serve, quando devemos usar Macros, como criar e alterar Macros. Em seguida você aprenderá os fundamentos básicos da linguagem VBA.

Nas lições desse módulo veremos os conceitos teóricos da linguagem VBA. Nas lições do próximo módulo, veremos exemplos de aplicação do VBA para a solução de problemas práticos, os quais não poderiam ser solucionados sem o uso de programação.

O que são Macros??

Nesse tópico apresentaremos uma visão geral sobre Macros. Nas próximas lições iremos detalhar os vários aspectos relacionados à Macros.

Caso você execute uma tarefa várias vezes no Microsoft Excel, é possível automatizá-la com uma macro. Uma macro é uma sequência de comandos e funções armazenados em um módulo de código do VBA e pode ser executada sempre que você precisar executar a tarefa. Quando você grava uma macro, o Excel armazena informações sobre cada etapa realizada à medida que você executa uma sequência de comandos. Em seguida, você executa a macro para repetir, ou "reproduzir", os comandos.

Por exemplo, vamos supor que, seguidamente, você precisa formatar uma célula com Negrito, cor de fonte Vermelha, Itálico, Fonte Verdana de Tamanho 13, com quebra automática de linha. Ao invés de ter que executar todos os comandos de formatação em cada célula, você pode criar uma Macro que aplica todos os comandos de formatação. Após criada a Macro, cada vez que você tiver que aplicar o conjunto de comandos de formatação, basta executar a Macro, o que normalmente é feito através da associação de uma combinação de teclas com a Macro, como por exemplo Ctrl+L. No nosso exemplo, cada vez que você quisesse formatar uma célula com os formatos descritos, bastaria clicar na célula e pressionar Ctrl+L. Bem mais fácil do que aplicar cada comando individualmente.

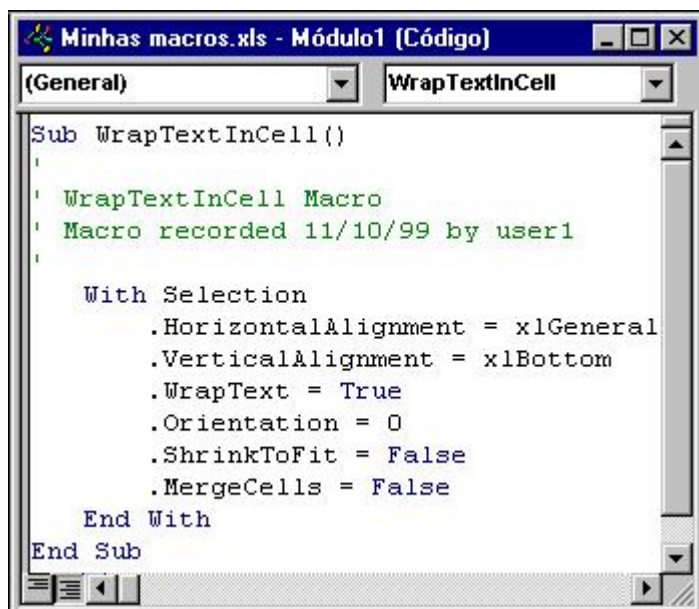
Você pode Gravar uma macro para realizar uma tarefa em uma etapa: Antes de gravar uma macro, planeje as etapas e os comandos que você deseja que a macro execute. Se cometer um erro durante a gravação da macro, as correções feitas também serão gravadas como comandos da macro. Ao gravar macros, o VBA armazena cada macro em um novo módulo de código VBA, anexado a uma pasta de trabalho (arquivo .xls).

Por exemplo, se você insere com frequência seqüências de caracteres de texto extensas nas células, você pode gravar uma macro para formatar essas células de maneira que o texto retorne automaticamente. Selecione a célula em que deseja inserir o retorno automático de texto e inicie a gravação da Macro. Clique em Células no menu Formatar, clique na guia Alinhamento, marque a caixa de seleção Retorno automático de texto, clique em OK e, em seguida, clique em Parar gravação (você verá exemplos práticos nas próximas lições).

Como Tornar uma macro fácil de ser executada: Você pode executar uma macro escolhendo-a de uma lista na caixa de diálogo Macro. Para que uma macro seja executada sempre que você clicar em um botão específico ou pressionar determinada combinação de teclas, você pode atribuir a macro a um botão da barra de ferramentas, a um atalho no teclado ou um objeto gráfico em uma planilha. Veremos como fazer essas atribuições nas próximas lições.

Como Exibir e alterar macros: Depois de gravar uma macro, você poderá exibir o código da macro com o Editor do VBA para corrigir erros ou alterar a função da macro. O Editor do VBA é um programa criado para facilitar a escrita e a edição de código de macro para principiantes e fornece bastante Ajuda on-line. Você não precisa aprender a programar ou a usar a linguagem do Visual Basic para fazer alterações simples nas suas macros. Nas próximas lições veremos como exibir e editar macros.

Na Figura a seguir temos um exemplo de código associado com uma Macro. Esse é um exemplo de código VBA:



Como Gerenciar suas macros Com o Editor do VBA: Você pode editar macros, copiar macros de um módulo para outro, copiar macros entre pastas de trabalho diferentes, renomear os módulos que armazenam as macros ou renomear as macros. Por exemplo, se você quisesse que a macro de retorno automático de texto, do exemplo anterior, também deixasse o texto em negrito, você poderia gravar outra macro para aplicar negrito a uma célula e copiar as instruções dessa macro para a macro de retorno automático de texto.

Segurança da macro: O Microsoft Excel 2000 fornece proteção contra vírus que podem ser transmitidos através das macros. Se você compartilha macros com outros usuários, você pode certificá-las com uma assinatura digital de forma que os outros usuários possam verificar que as macros são de origem confiável. Sempre que você abrir uma pasta de trabalho que contenha macros, poderá verificar a origem das macros antes de ativá-las.

Programação no Excel – A linguagem VBA:

As Macros são uma excelente solução quando queremos automatizar uma tarefa que é realizada através de uma série de cliques de mouse ou digitações no teclado. Porém existem situações mais complexas, que envolvem cálculos ou uma lógica mais apurada, onde não é possível encontrar a solução do problema, simplesmente usando os comandos ou fórmulas prontas do Excel.

Nessas situações temos que fazer uso de programação. Um programa (ou módulo como é chamado no Excel) é uma sequência de comandos VBA, onde cada comando executa um passo específico, necessário à resolução do problema.

Nota: Para um melhor aproveitamento e entendimento do VBA é importante que o amigo leitor já tenha uma noção básica de Lógica de Programação. Você encontra um excelente curso de Lógica de Programação no seguinte endereço: www.webaula.com.br. O curso é gratuito, apenas é necessário fazer um cadastro no site, cadastro esse que também é gratuito.

Por exemplo, vamos supor que você precisasse fazer a verificação do CPF que é digitado em uma célula. O cálculo do DV do CPF, o qual é de domínio público, envolve uma série de operações aritméticas. Para implementar uma função que faz a verificação do DV do CPF, você terá que fazer uso de programação.

Por isso que, conforme descrito anteriormente, nas lições desse módulo veremos os comandos básicos da linguagem VBA, para aplicá-los em alguns exemplos práticos nas lições dos demais módulos deste curso.

Lição 02: O que são exatamente Macros???

Conforme descrito anteriormente, uma macro é uma sequência de comandos (cliques de mouse ou toques de teclado) que são gravados em um Módulo VBA e podem ser executados, sempre que necessário. A grande vantagem de gravarmos uma sequência de comandos é que poderemos utilizá-la sempre que necessário. Para isso basta executar a macro na qual foi gravada a sequência de comandos.

As Macros são uma excelente opção para automatizar tarefas repetitivas. Com o uso de Macros temos um ganho de produtividade considerável, ao evitar que tenhamos que executar manualmente, os diversos passos de uma tarefa, passos estes que podem ser automatizados através do uso de uma macro. Cada comando do Excel ou clique na planilha é representado por um ou mais comandos VBA, na Macro.

Existem duas maneiras distintas para a criação de uma macro:

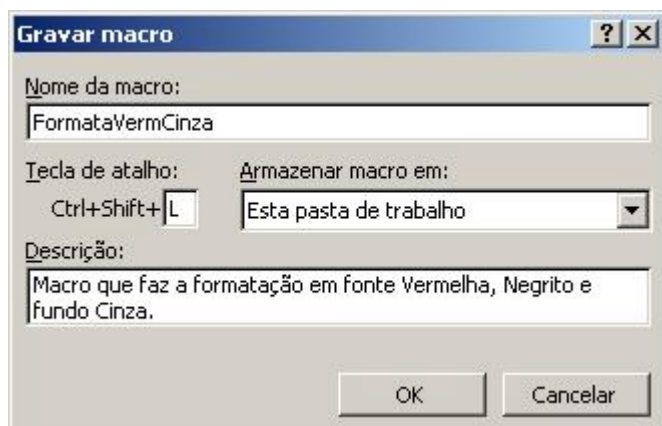
- ➔ **Podemos usar o gravador de Macros:** Nesse caso o Excel grava cada uma das ações que farão parte da Macro e transforma essas ações nos comandos VBA equivalentes. Quando a macro for executada, os comandos VBA é que serão efetivamente executados. Cada comando VBA corresponde a uma ação efetiva da macro.
- ➔ **Criar a Macro usando VBA:** A partir do momento em que você domina a linguagem VBA, poderá criar a macro digitando os comandos VBA necessários. Isso é feito usando o Editor de VBA, conforme veremos nas próximas lições.

Melhor do que definições é a prática!!

Melhor do que uma série de definições é ver uma Macro em ação. Vamos a um exemplo simples, onde criaremos uma Macro. Em seguida vamos executá-la. Na próxima lição analisaremos o código VBA criado pelo gravador de macros.



Exemplo 1: Criar uma macro usando o Gravador de Macros. A macro deverá formatar a célula atual com Negrito, cor de fonte Vermelha, com fundo cinza. Gravar a macro com o nome de FormataVermCinza.

1. Abra o Excel.
2. Abra a Planilha C:\Programação VBA no Excel\Módulo 1 – Exercício 01.xls.
3. Clique na célula A4.
4. Agora vamos iniciar a gravação da Macro.
5. Selecione o comando Ferramentas -> Macro -> Gravar nova macro.
6. Será exibida a janela Gravar Macro.
7. No campo Nome da macro digite: FormataVermCinza.
8. No campo Tecla de atalho digite L. Observe que o Excel troca para Ctrl+Shift+L. Isso acontece porque a combinação Ctrl+L já deve estar associada com algum comando do Excel. Com isso estamos associando a combinação Ctrl+Shift+L com a macro FormataVermCinza, ou seja, cada vez que quisermos executar essa macro basta pressionar Ctrl+Shift+L.
9. O campo descrição é simplesmente uma descrição da funcionalidade da macro. Digite o texto indicado na Figura a seguir:





10. Clique em OK. A gravação da Macro será iniciada. Todos os comandos que você executar, durante a gravação da Macro, farão parte da Macro.



11. Uma nova barra () é exibida na planilha do Excel. Essa barra é utilizada para parar a gravação da Macro. Agora devemos escolher os comandos que farão parte da macro. Após ter executado os comandos que farão parte da macro, basta clicar no botão () para encerrar a gravação da Macro.

12. Clique no botão () para aplicar Negrito.

13. Na lista de Cores da fonte () selecione Vermelho.

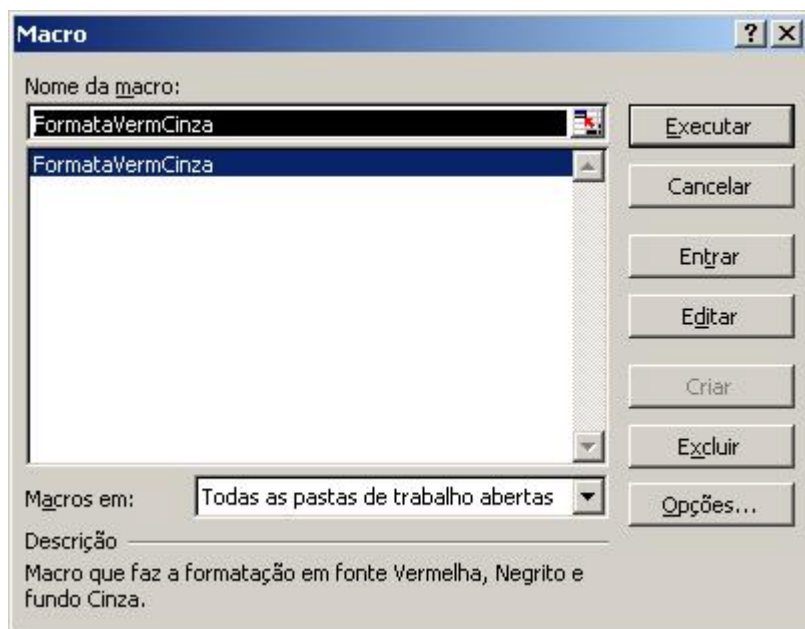
14. Na lista de Cores de fundo () selecione Cinza.

15. Clique no botão () para encerrar a gravação da Macro.

16. Agora a macro FormataVermCinza foi criada e podemos utiliza-la sempre que necessário.

Para executar a macro FormataVermCinza faça o seguinte:

1. Clique na Célula B7.
2. Pressione Ctrl+Shift+L.
3. A macro FormataVermCinza é executada e as formatações definidas pela macro (Negrito, fonte Vermelha e fundo Cinza) são automaticamente aplicadas na Célula B7. Veja que com um simples comando de teclado, executo uma série de comandos (nesse exemplo: três comandos de formatação). Esse exemplo nos dá uma pequena idéia do poder e facilidade do uso das macros.
4. Também é possível executar a macro usando o comando Ferramentas -> Macro -> Macros.
5. Clique na célula B5.
6. Selecione o comando Ferramentas -> Macro -> Macros.
7. Será exibida a janela Macro, onde são listadas todas as macros existentes na pasta de trabalho atual (no arquivo carregado no Excel), conforme indicado na Figura a seguir:



8. Clique na Macro FormataVermCinza para selecioná-la.
9. Clique no botão Executar.
10. A Macro será executada e as respectivas formatações serão aplicadas à célula B5.
11. A sua planilha deve estar conforme indicado na Figura a seguir:

	A	B	C	D	E
1	PRIMEIRO EXEMPLO DE MACRO				
2					
3	Ano	Vendas			
4	1999	23250			
5	2000	15623			
6	2001	35260			
7	2002	45123			
8					

12. Mantenha a planilha aberta, pois iremos utilizá-la na próxima lição.

Lição 03: Conhecendo do que é feita uma Macro

Na Lição anterior podemos ver uma macro em ação. Criamos uma macro chamada FormataVermCinza. Essa macro é composta por três comandos de formatação. Cada vez que a macro é executada, os três comandos de formatação são novamente executados e aplicados à célula onde está o cursor ou na faixa de células selecionadas. Até aqui tudo OK. Nenhuma novidade. A pergunta que pode surgir é:

Como é que o Excel faz isso?

Ou de outra forma:

Do que é feita uma macro?

Conforme veremos nessa lição, **uma macro é gravada no Excel como uma sequência de comandos VBA**. Por exemplo, ao clicarmos no botão (**N**), para a aplicação de negrito, o Excel gera um comando VBA que faz a formatação em negrito. Nessa lição aprenderemos a acessar o código VBA gerado pelo Excel (embora ainda não sejamos capazes de entender esse código).

Para acessar os comandos VBA associado com uma macro, siga os seguintes passos:

1. Você deve estar com a planilha C:\Programação VBA no Excel\Módulo 1 – Exercício 01.xls aberta, se não estiver, abra-a.
2. Selecione o comando Ferramentas -> Macro -> Macros.
3. Será exibida a janela Macro.
4. Clique na macro FormataVermCinza para selecioná-la.
5. Clique no botão Editar.
6. O Editor do VBA será carregado e serão exibidas as seguintes linhas de código:

```
Sub FormataVermCinza()  
,  
,  
    ' FormataVermCinza Macro  
    ' Macro que faz a formatação em fonte Vermelha, Negrito e  
    ' fundo Cinza.  
,  
,  
    ' Atalho do teclado: Ctrl+Shift+L  
,  
  
    Selection.Font.Bold = True  
    Selection.Font.ColorIndex = 3  
    With Selection.Interior  
        .ColorIndex = 15  
        .Pattern = xlSolid  
    End With  
End Sub
```

Esses são os comandos VBA (por enquanto não se preocupe se você não entender o que significa cada comando, pois este será exatamente o objeto de estudo deste curso, ou seja,

Programação VBA) que formam a macro FormataVermCinza. Apenas para adiantar um pouco o assunto, a seguir descrevo o que faz cada um dos principais comandos dessa Macro:

```
Selection.Font.Bold = True
```

Esse comando aplica a formatação em Negrito para a célula onde está o cursor (ou no conjunto de células selecionadas), quando a macro é executada.

```
Selection.Font.ColorIndex = 3
```

Esse comando aplica cor de fonte Vermelha para a célula onde está o cursor (ou no conjunto de células selecionadas), quando a macro é executada.

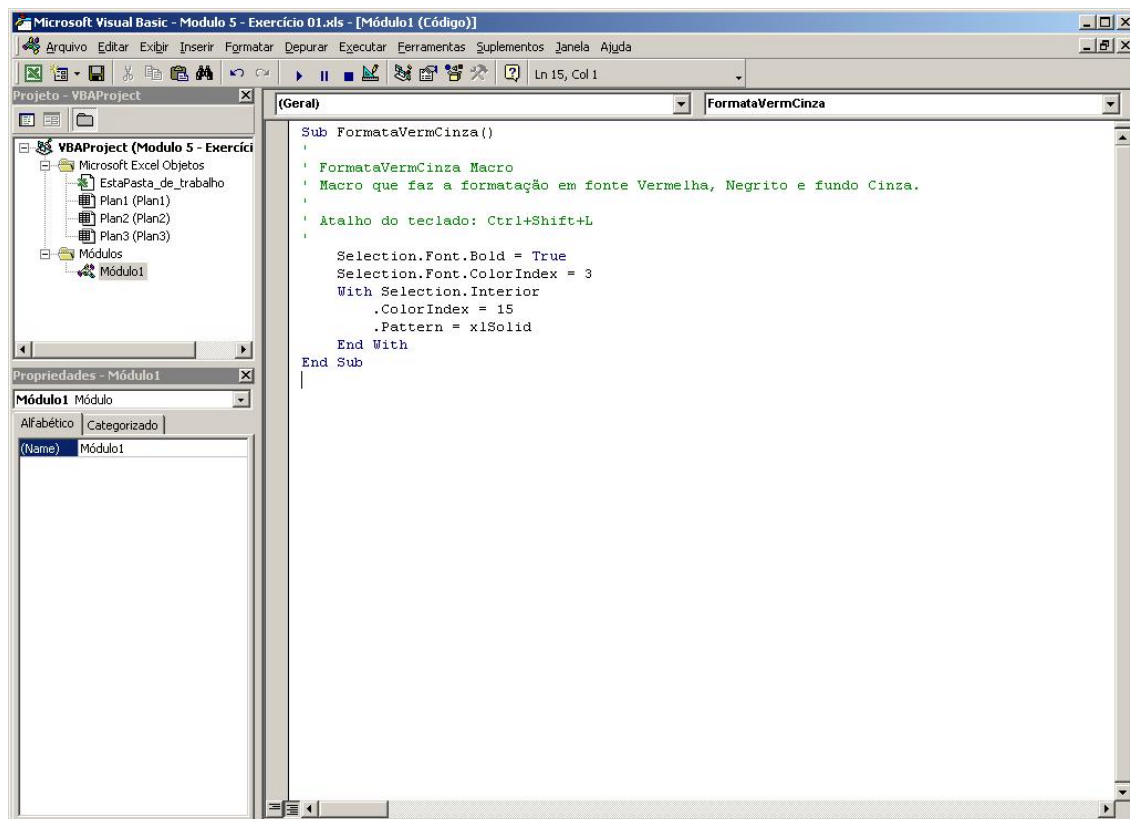
```
With Selection.Interior  
    .ColorIndex = 15  
    .Pattern = xlSolid  
End With
```

Esses comandos aplicam a cor de fundo cinza, na célula onde está o cursor (ou no conjunto de células selecionadas), quando a macro é executada.

Esses são comandos da linguagem VBA. Com o uso do VBA temos acesso a todos os comandos e funções do Microsoft Excel, além de muitos outros recursos não diretamente disponíveis através dos menus de comandos e das funções de planilha do Excel, ou seja, recurso que somente estão disponíveis através da programação VBA. Tudo o que você faz usando o teclado e o mouse, também é possível de ser feito com o uso do VBA, porém de uma maneira automatizada. O uso de macros é especialmente indicado naquelas situações em que temos um conjunto de comandos que precisam ser executados, repetidamente, em diferentes situações. Nesses casos é muito mais prático criar uma macro composta pelo conjunto de comandos e, cada vez que os comandos precisarem ser executados, executar a macro.

Na Figura a seguir, apresento uma visão do Editor do Visual Basic. Veremos mais detalhes sobre esse editor nas próximas lições, quando começarmos a trabalhar com o VBA.

7. Selecione o comando Arquivo -> Fechar e Voltar para o Microsoft Excel.
8. Você estará de volta à planilha C:\Programação VBA no Excel\Módulo 1 – Exercício 01.xls.
9. Salve e feche a planilha.



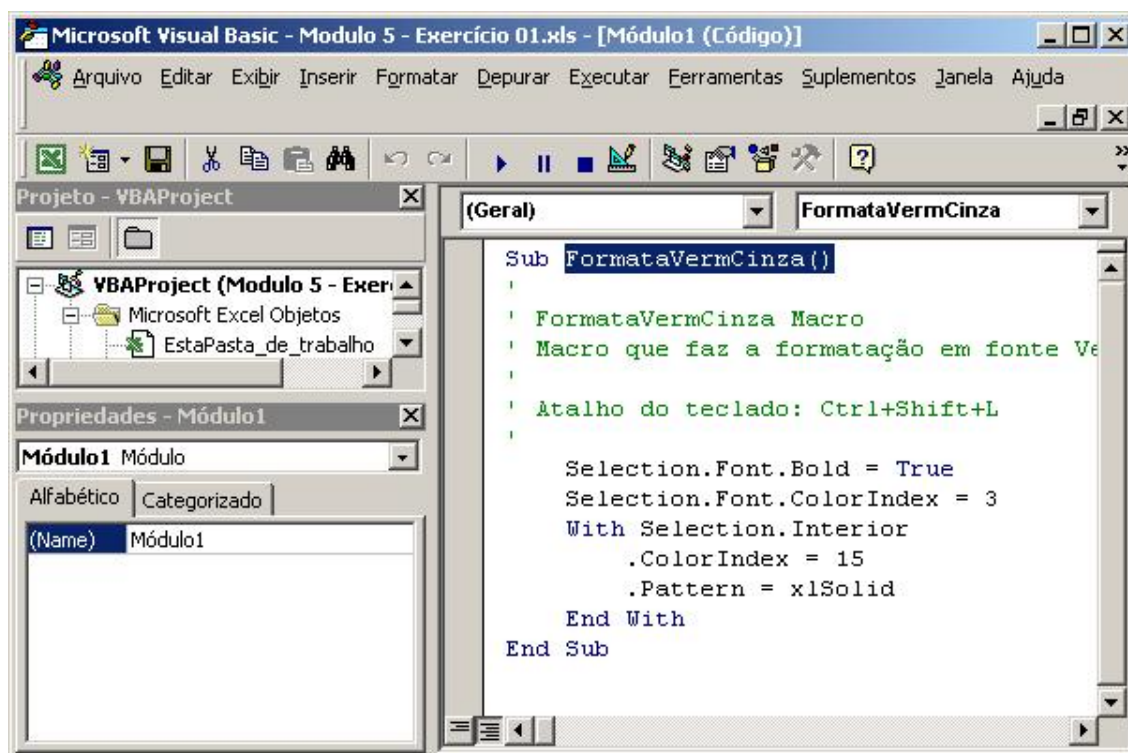
Lição 04: Operações com Macros

Nessa lição aprenderemos a renomear, excluir e fazer outras alterações em macros. Também aprenderemos a alterar outras opções associadas com a Macro, tal como o comando de teclado (tecla de atalho) para executar a macro.

Existem algumas operações que podem ser feitas com uma macro, após a sua criação. A mais óbvia (e o motivo pelo qual uma macro é criada) é para executar a macro. Além da execução é possível executar outras operações com uma macro, conforme descrito a seguir:

Para renomear uma macro siga os seguintes passos:

1. Abra o arquivo onde está a macro a ser renomeada.
2. Selecione o comando Ferramentas -> Macro -> Macros.
3. Será exibida a janela Macro.
4. Clique na macro a ser renomeada para selecioná-la.
5. Clique no botão Editar.
6. Será aberto o editor do VBA. O nome da macro vem logo após o comando Sub da primeira linha, conforme destacado na Figura a seguir:

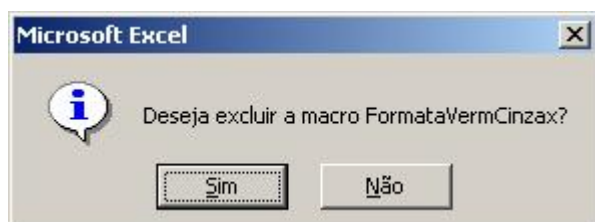


Renomeando uma Macro.

7. Para renomear a macro basta alterar o nome após o comando Sub e depois clicar no botão (💾) para salvar as alterações.
8. Em seguida é só fechar o Editor de VBA.

Para excluir uma macro siga os seguintes passos:

1. Abra o arquivo onde está a macro a ser excluída.
2. Selecione o comando Ferramentas -> Macro -> Macros.
3. Será exibida a janela Macro.
4. Clique na macro a ser excluída para selecioná-la.
5. Clique no botão Excluir.
6. O Excel emite um aviso solicitando que você confirme a exclusão, conforme indicado na Figura a seguir:

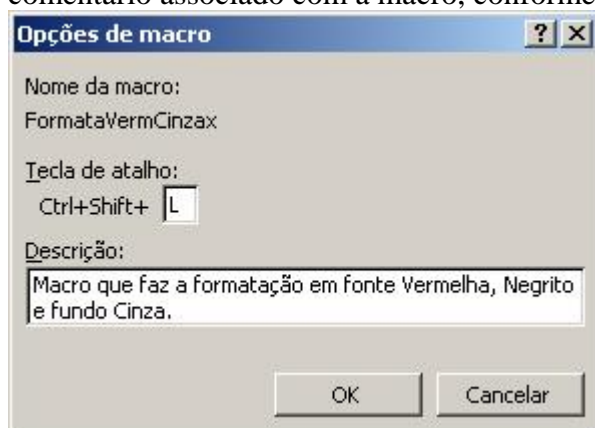


Confirmando a exclusão da macro.

7. Clique em Sim para confirmar a exclusão ou em Não para cancelar a exclusão da macro.

Para alterar a tecla de atalho e o comentário associado com uma macro, siga os seguintes passos:

1. Abra o arquivo onde está a macro a ser alterada.
2. Selecione o comando Ferramentas -> Macro -> Macros.
3. Será exibida a janela Macro.
4. Clique na macro a ser alterada para selecioná-la.
5. Clique no botão Opções...
6. Será exibida a janela Opções de Macro, onde você pode alterar a tecla de atalho e o comentário associado com a macro, conforme indicado na Figura a seguir:



Janela de Opções da Macro.

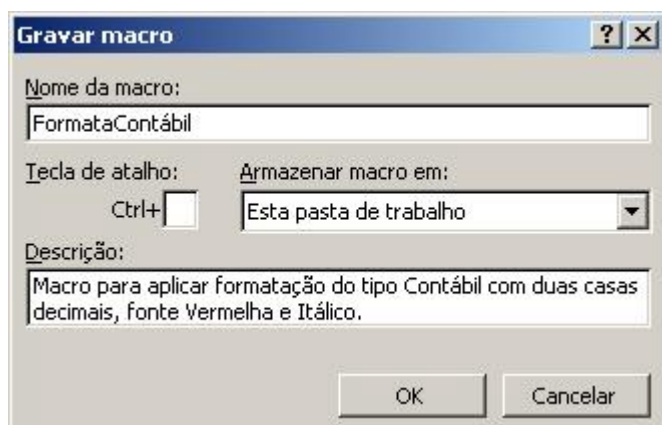
7. Faça as alterações desejadas e clique em OK.

Lição 05: Associando botões com macros

Nessa lição criaremos mais uma macro de exemplo. Além da criação da macro aprenderemos a criar um novo botão, na Barra de Ferramentas do Excel e a associar esse botão com a macro. Dessa forma toda vez que precisarmos executar a macro, bastará clicar no botão associado com a macro.

Exemplo 2: Criar uma macro usando o Gravador de Macros. A macro deverá formatar As células selecionadas com formato Contábil, com duas casas decimais, cor de fonte Vermelha e Itálico. Gravar a macro com o nome de **FormataContábil**. Criar um botão de comando na Barra de Ferramentas padrão e associar esse botão com a macro FormataContábil.




1. Abra o Excel.
2. Abra a Planilha C:\Programação VBA no Excel\Módulo 1 – Exercício 02.xls.
3. Clique na célula F2.
4. Agora vamos iniciar a gravação da Macro.
5. Selecione o comando **Ferramentas -> Macro -> Gravar nova macro**.
6. Será exibida a janela Gravar Macro.
7. No campo Nome da macro digite: **FormataContábil**.
8. Na lista Armazenar macro em, selecione a opção **Esta pasta de trabalho**.
9. O campo descrição é simplesmente uma descrição da funcionalidade da macro. Digite o texto indicado na Figura a seguir:




Definição do nome e da descrição da macro FormataContábil.

10. Clique em OK. A gravação da Macro será iniciada. Todos os comandos que você executar, durante a gravação da Macro, farão parte da Macro.




11. Uma nova barra () é exibida na planilha do Excel. Essa barra é utilizada para parar a gravação da Macro. Agora devemos escolher os comandos que farão parte da macro. Após ter executado os comandos que farão parte da macro, basta clicar no botão () para encerrar a gravação da Macro.
12. Clique no botão () para aplicar Itálico.

13. Na lista de Cores da fonte () selecione Vermelho.
14. Selecione o comando **Formatar -> Células**. Clique na opção Contábil e selecione duas casas decimais, conforme indicado na Figura a seguir:




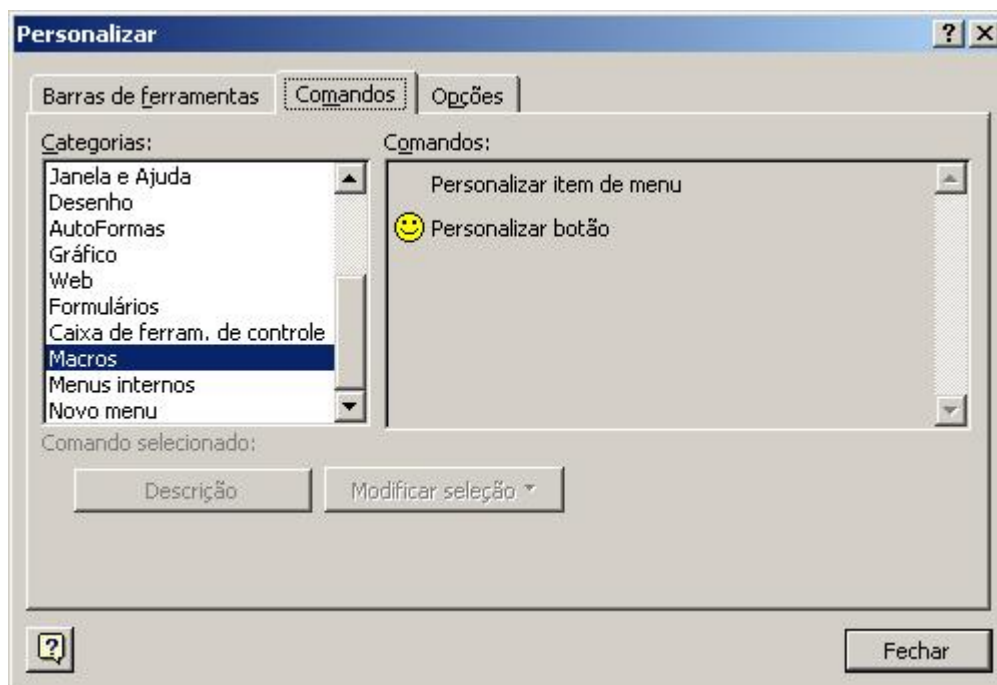
Formatação Contábil com duas casas decimais.

15. Clique em OK.
16. Clique no botão () para encerrar a gravação da Macro.
17. Agora a macro **FormataContábil** foi criada e podemos utiliza-la sempre que necessário.

Agora vamos aprender como associar um botão com a macro FormataContábil.

Para associar um botão com uma macro siga os seguintes passos:

1. Clique com o botão direito do mouse em uma das barras de ferramentas do Excel. Por exemplo, clique com o botão direito do mouse na área cinza, ao lado do botão ().
2. No menu que é exibido clique na opção Personalizar.
3. Será exibida a janela Personalizar.
4. Clique na guia Comandos.
5. Na lista de Categorias que é exibida dê um clique na opção Macros conforme indicado na Figura a seguir:



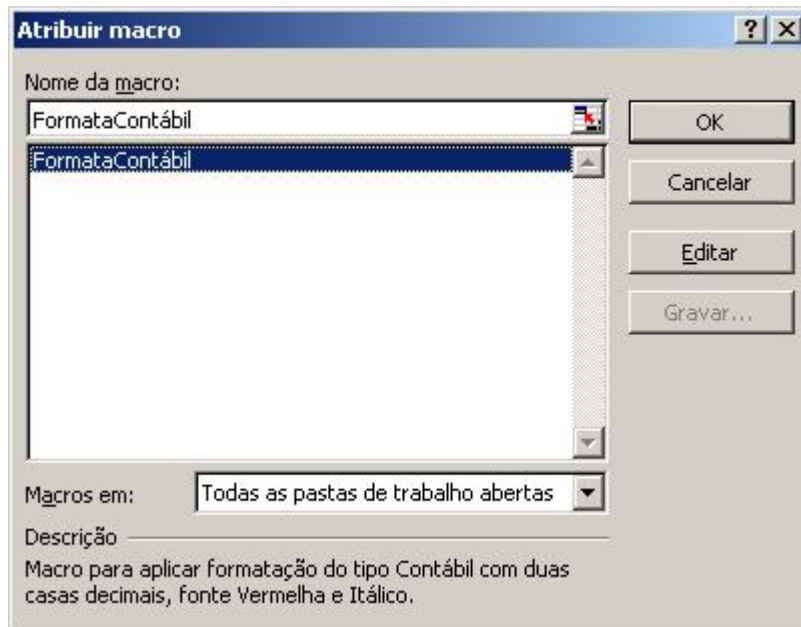
A opção Macros da guia Comandos.

6. Clique no botão (😊) e arraste-o para a barra de ferramentas padrão, ao lado esquerdo do botão (📊), conforme indicado na Figura a seguir:



Arrastando o botão que será associado com a Macro.

7. A janela Personalizar será fechada e o botão (😊) será adicionado à barra de ferramentas padrão.
8. Dê um clique no botão (😊).
9. Será exibida a janela Atribuir macros. Nessa janela você pode selecionar a macro que será associada com o botão, isto é, a macro que será executada quando você clicar no botão.
10. Clique na macro **FormataContábil** para selecioná-la, conforme indicado na Figura a seguir.
11. Clique em OK.
12. Pronto, agora o botão (😊) está associado com a macro FormataContábil.



Associando a macro FormataContábil com o botão.

13. Vamos testar se o botão está funcionando.
14. Clique na célula F10.
15. Clique no botão (😊).
16. Observe que a macro é executada e as respectivas formatações são aplicadas à célula F10. Isso comprova que o botão (😊) está associado à macro FormataContábil.
17. Salve e Feche a planilha.

Muito bem, sobre macros é basicamente isso: O que são Macros?, Como criar, como editar, como excluir e como associar um botão de comando. A partir da próxima lição vamos iniciar o estudo da linguagem de programação VBA.

Lição 06: Introdução ao VBA

Nas lições iniciais desse módulo aprendemos a criar macros simples, as quais reproduzem uma série de comandos de mouse e teclado. Para que possamos criar macros mais sofisticadas e resolver problemas mais complexos com o Excel, precisamos utilizar programação VBA. Conforme descrito anteriormente, a linguagem de programação do Excel (e de todos os aplicativos do Office) é o VBA: **Visual Basic for Application**.

Uma linguagem de programação, basicamente, é um conjunto de comandos, rotinas, objetos e funções que executam tarefas específicas. Considere o exemplo genérico a seguir, onde são utilizados comandos para acessar uma tabela do access a partir de uma planilha do Excel:

```
' Comentários iniciais do Programa.  
' Acessa dados da tabela pedidos do banco de dados  
' C:\Meus documentos\vendas.mdb
```

```
Acessar o banco de dados  
Acessar a tabela Pedidos  
Aplicar um filtro para País='Brasil'  
Exibir os dados obtidos na planilha atual  
Formatar a primeira linha com negrito  
Formatar a primeira linha com fonte azul
```

```
Encerrar a macro
```

Por que eu preciso aprender a usar o VBA?

A utilização de Macros em conjunto com os recurso do VBA nos oferece um grande número de opções na busca por soluções para os problemas mais complexos. Porém existem situações em que, por mais que saibamos utilizar todos os recursos, comandos e funções do Excel, essa utilização não é capaz de solucionar o problema proposto. Nestas situações temos que utilizar programação. Muitos usuários acreditam que é possível solucionar todo e qualquer problema usando somente os comandos e funções do Excel. Isso não é verdade. Existem situações onde você terá que criar suas próprias rotinas e funções, para solucionar um determinado problema. E isso só pode ser feito com o uso de programação.

Muitos usuários entram em contato via email, dizendo: “Por favor, me dê uma solução que não envolva programação, não quero saber de programação”. **Respeito a opinião de todos, mas “fugir” da programação, significa abrir mão dos recursos mais poderosos que o Excel disponibiliza, significa ficar sem poder solucionar problemas mais complexos. A programação é uma ferramenta realmente útil. E você verá, no decorrer deste curso, que programação não é nenhum bixo de sete cabeças.**

A linguagem de programação utilizada pelo Microsoft Excel é o **VBA - Visual Basic for Applications**. Conforme veremos a partir de agora esta é uma linguagem, ao mesmo tempo, extremamente simples e poderosa. Com o VBA temos acesso completo a todos os elementos de todos os objetos de uma planilha do Excel. Também temos acesso a elementos externos, tais como bancos de dados do Access.

Com código VBA podemos criar uma rotina para validação do dígito verificador de uma célula que contém um valor de CPF, CNPJ ou de um campo NúmeroDoProcesso; podemos criar código que percorre todas as linhas de uma planilha, alterando os valores de uma ou mais colunas, com base em uma ou mais condições, podemos automatizar rotinas para importação e exportação de dados e assim por diante.

Nesta introdução a linguagem VBA, trataremos dos seguintes assuntos:

1. Programação com o Microsoft Excel.
2. Introdução a linguagem VBA - **Visual Basic For Applications**.
3. Aprendendo VBA:
 - Uma visão geral.
 - O Ambiente de Programação.
 - Anatomia dos Módulos do VBA.
 - Tipos de dados.
 - Variáveis.
 - Escopo de Variáveis.
 - Estruturas de controle, etc.
4. Exercícios e Exemplos.
5. Funções e procedimentos.
6. Funções de Data e Hora.

Programação com o Microsoft Excel - Por que utilizar?.

O VBA nos oferece possibilidades de controle e personalização para criar aplicativos que vão além das ações de macro.

O VBA é uma linguagem de programação interna do Microsoft Excel (na prática é a linguagem de programação para todos os aplicativos do Office: Access, Word, Excel e PowerPoint). Usamos o VBA pelo mesmo motivo que utilizamos macros - para automatizar tarefas e rotinas repetitivas, envolvendo os diversos elementos do banco de uma planilha (células, faixas de células, gráficos, planilhas, funções, realizar cálculos, etc.). No entanto, o VBA oferece maior poder e controle mais detalhado do que as ações de macro.

Na prática as ações de macro duplicam as operações que podemos realizar manualmente, usando o mouse para executar comandos nos menus e o teclado. O VBA vai além da simples automação de seqüências de ações. Ele oferece um conjunto de ferramentas que lhe permite criar aplicações personalizadas com base nos elementos do Excel e nos objetos de planilha do Excel. Por exemplo, podemos criar uma rotina em VBA em uma planilha do Excel. Esta rotina pode acessar dados em uma segunda planilha que está na rede, em um drive mapeado em um servidor. A mesma rotina além de acessar os dados pode fazer cálculos, consolidações, gerar os resultados no formato de uma planilha do Excel e salvar a planilha na rede ou publicar os dados no formato HTML, no servidor da Intranet da empresa. Este é apenas um pequeno exemplo do que pode ser feito como o VBA.

Vantagens em utilizarmos o VBA:

Nas primeiras lições deste módulo, já utilizamos macros e aprendemos a automatizar algumas tarefas como a aplicação de formatos personalizados. O VBA apresenta, em relação as macros, as seguintes vantagens:

- **Acessando dados de uma ou mais planilhas:** Com ações de macros estamos limitados a operar com os dados atualmente sendo exibido na pasta de trabalho atual. O VBA permite trabalhar com qualquer conjunto de dados, quer seja da pasta de trabalho atual, quer seja de outra pasta de trabalho na rede ou com outros formatos de dados, como por exemplo de arquivos .txt ou bancos de dados do Microsoft Access.
- **Manipulação de objetos:** O VBA oferece métodos de criação e modificação dos objetos de uma planilha no Excel (assunto que será abordado nas lições dos Módulos 2,3, 4, além de uma série de exemplos práticos, apresentados no Módulo 6). Chamamos de objeto qualquer elemento do Microsoft Excel, por exemplo: uma planilha, uma faixa de células, um gráfico, etc.
- **Criação de funções definidas pelo usuário:** Este é um dos maiores benefícios do VBA. Podemos criar funções que executam cálculos repetitivos. Por exemplo, várias planilhas podem conter um campo CPF ou CNPJ. Poderíamos criar, em cada planilha, o código necessário para a validação do DV do CPF ou do CNPJ. Porém este procedimento não é o mais indicado, pois além da duplicação do código necessário a validação, teríamos dificuldades para fazer atualizações neste código, pois cada alteração necessária teria que ser feita em vários locais. O ideal é criarmos uma função para validação do DV (uma função deve ser criada dentro de um módulo. Trataremos disso mais adiante.). Em cada planilha, onde for necessária a utilização da função, chamamos a função, passando o valor do CPF como parâmetro. A função calcula o DV e retorna o resultado para a planilha. Desta maneira precisamos criar uma única função. Quando forem necessárias alterações, basta alterar a função (em um único local, ou seja, no módulo onde a função foi criada) e todos os formulários passarão a utilizar a versão atualizada da função.
- **Definição de condições e controle de fluxo:** O VBA oferece uma variedade de comandos e estruturas para a realização de testes condicionais e para a repetição de um conjunto de comandos. Aprenderemos a utilizar todas as estruturas de controle de fluxo e testes condicionais, mais adiante.
- **Realização de cálculos complexos e solução de problemas que envolvem uma lógica complexa:** Com macros é impossível a realização de cálculos mais complexos, simplesmente através da automação de comandos de teclado e mouse. Também não é possível a resolução de problemas que envolvem uma lógica complexa, como por exemplo cálculo do imposto de renda, recolhimentos de tributos, etc.

Muito bem, esta lição foi para fazer uma apresentação do VBA e convencê-lo de quão útil é este recurso.

Lição 07: O Ambiente de programação – o Editor VBA – Parte I

O Microsoft Excel fornece um ambiente de programação bastante poderoso, com uma série de recursos que facilitam a criação de código VBA. Neste tópico vamos aprender a utilizar os aspectos básicos do Ambiente de Programação do VBA. O ambiente de programação é um editor que facilita a criação de código. Dentro do ambiente de programação, são oferecidas uma série de facilidades e dicas para que o Programador possa encontrar, facilmente, os objetos disponíveis, bem como os métodos e propriedades de cada objeto.

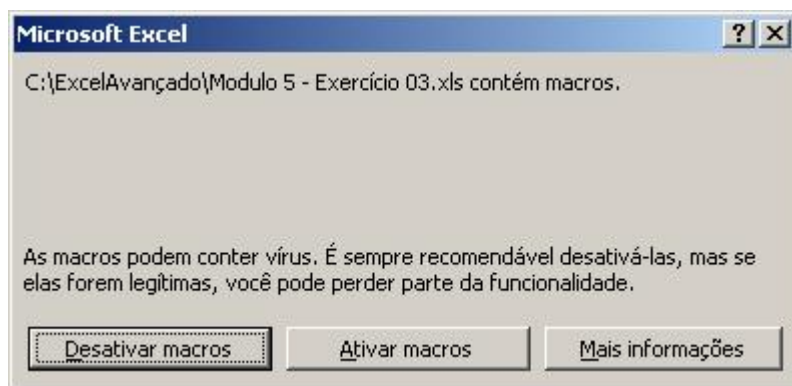
Se não tivéssemos disponível um Ambiente de Programação, teríamos que lembrar da sintaxe de todos os comandos, dos métodos e propriedades dos objetos. Convenhamos que isso é praticamente impossível, pois com o VBA temos acesso a milhares de objetos (é isso mesmo: milhares de objetos, comandos e funções. Por enquanto estou utilizando bastante o termo objeto, sem tê-lo explicado ainda. Mais adiante detalharei o conceito de classes, módulos e objetos). Cada objeto pode ter dezenas de propriedades, métodos e coleções (alguns tem centenas de propriedades e métodos).

O ambiente de Desenvolvimento fornece uma série de facilidades para a criação de código VBA. Por exemplo, ao digitar o nome de um objeto e um ponto será aberta, automaticamente, uma lista com todos os métodos e propriedades deste objeto. Ao invés de lembrar do nome dos métodos/propriedades, basta selecioná-los em uma lista. Se selecionarmos um método, ao digitarmos o parênteses de abertura, será exibida uma lista com os argumentos esperados pelo método, bem como o tipo (texto, número, data, etc) de cada argumento. Se digitarmos um comando incorretamente, o Ambiente de Desenvolvimento emite uma mensagem e coloca em destaque o comando que foi digitado incorretamente. Estas são apenas algumas das facilidades fornecidas pelo Ambiente de Desenvolvimento do VBA.

Para conhecermos melhor o referido ambiente, vamos a um exemplo prático. Vamos abrir uma planilha, que contém uma macro chamada AplicaNegrito e editar essa macro. Lembre que para editar uma macro, temos que acessar o código VBA associado à macro. Isso é feito no Editor do VBA, ou seja, no Ambiente de Desenvolvimento do VBA.

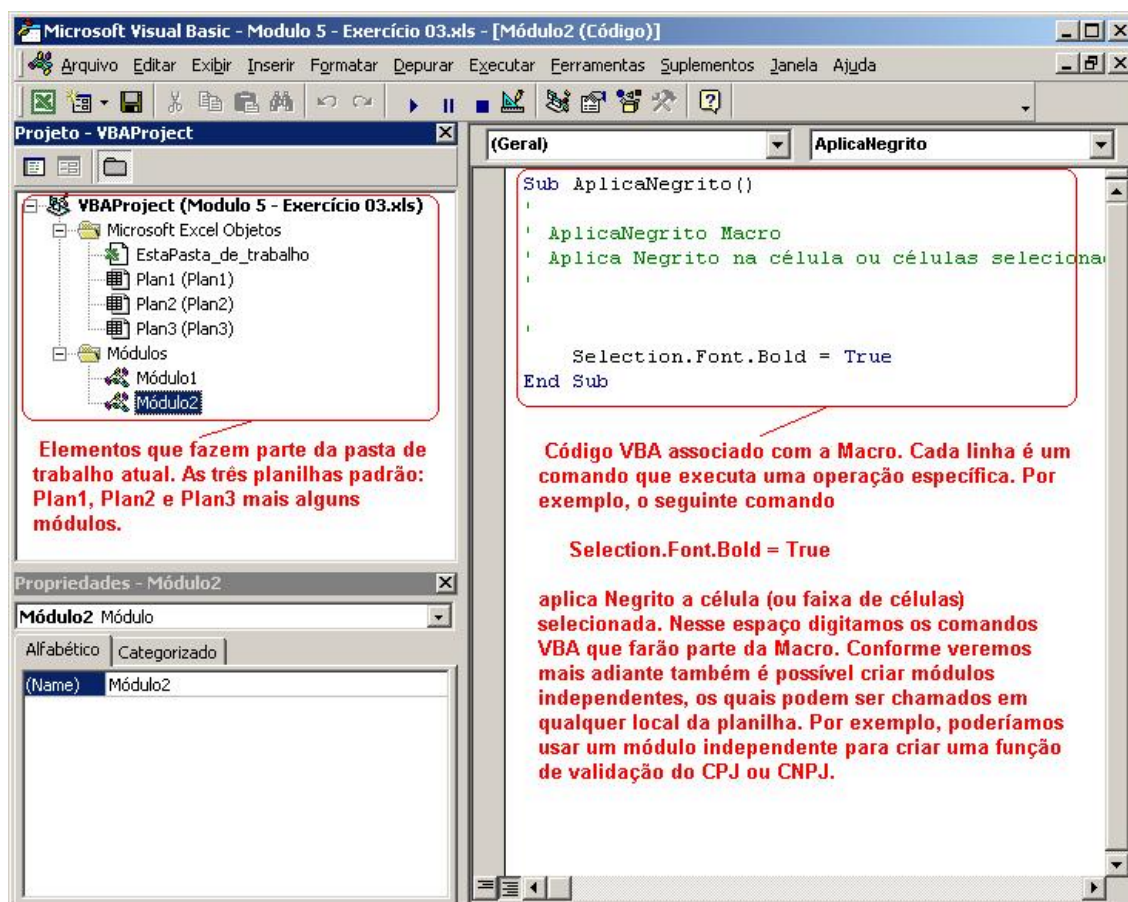
Exemplo 3: Acessar o Editor VBA para alterar a macro AplicaNegrito, da planilha C:\Programação VBA no Excel\Módulo 1 – Exercício 03.xls.

1. Abra o Excel.
2. Abra a Planilha C:\Programação VBA no Excel\Módulo 5 – Exercício 03.xls.
3. Como já existem macros nesta planilha, o Excel pede uma confirmação, perguntando se você deseja ativar as macros existentes, conforme indicado na próxima figura. Essa confirmação é solicitada devido ao perigo representado pelos vírus de Macro, que podem afetar os aplicativos do Microsoft Office.
4. Clique no botão Ativar Macros.
5. A planilha será aberta.
6. Nessa planilha existe uma macro chamada AtivaNegrito. Vamos editar essa macro. O nosso objetivo é conhecer alguns detalhes sobre o Ambiente de Desenvolvimento do VBA.



Confirmação para ativação de macros, na abertura da planilha.

7. Selecione o comando Ferramentas -> Macro -> Macros...
8. Será exibida a janela Macro.
9. Clique na macro AplicaNegrito para selecioná-la.
10. Clique no botão Editar.
11. Será aberto o editor do VBA e o código associado à macro AplicaNegrito.
12. Na Figura a seguir, temos a descrição de algumas partes da janela do editor VBA.



O Editor VBA.

No lado esquerdo da tela temos duas janelas:

Autor: Júlio Cesar Fabris Battisti

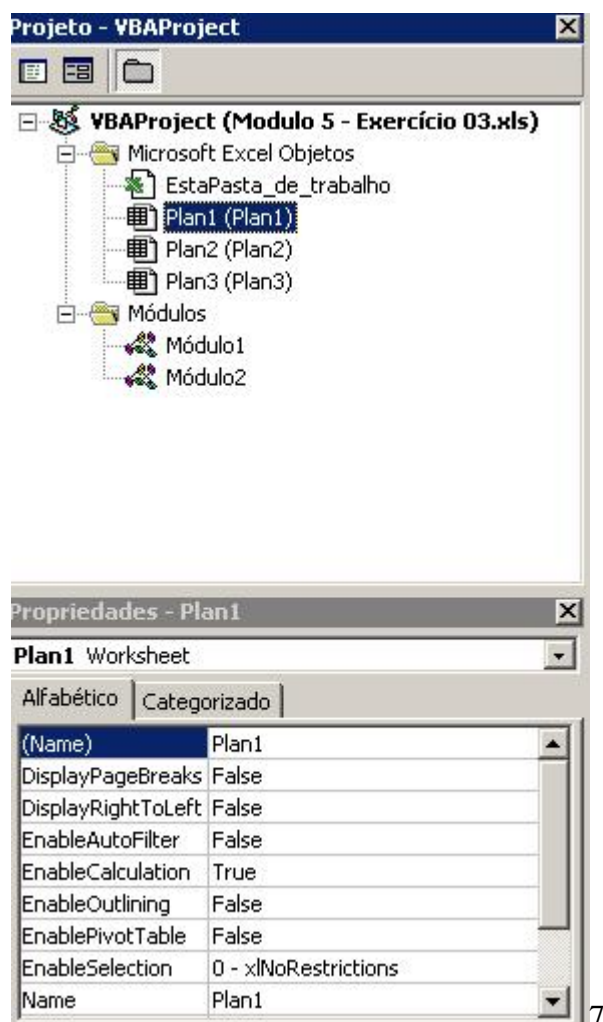
Site: www.juliobattisti.com.br

Confira também o livro: "Windows Server 2003 – Curso Completo, 1568 páginas"

Página 31 de 527

Project – VBA Project: Nessa janela são exibidos os vários elementos que fazem parte da Pasta de trabalho atual. Aqui são exibidas as planilhas e módulos da pasta de trabalho (arquivo .xls) carregado no Excel. Ao criarmos uma macro podemos criá-la em uma determinada planilha. Os módulos são utilizados para criar funções e procedimentos que podem ser chamados em todas as planilhas da pasta de trabalho atual. Aprenderemos mais sobre funções e procedimentos nas próximas lições.

Janela na parte de baixo: A janela abaixo da janela Project, exibe as propriedades do elemento selecionado na janela Project. Por exemplo, ao selecionar Plan1, na janela Project, na janela de baixo serão exibidas as propriedades de Plan1, conforme indicado na Figura a seguir:



Propriedades do objeto selecionado em VBAProject.

13. Feche o Editor do VBA e a planilha.

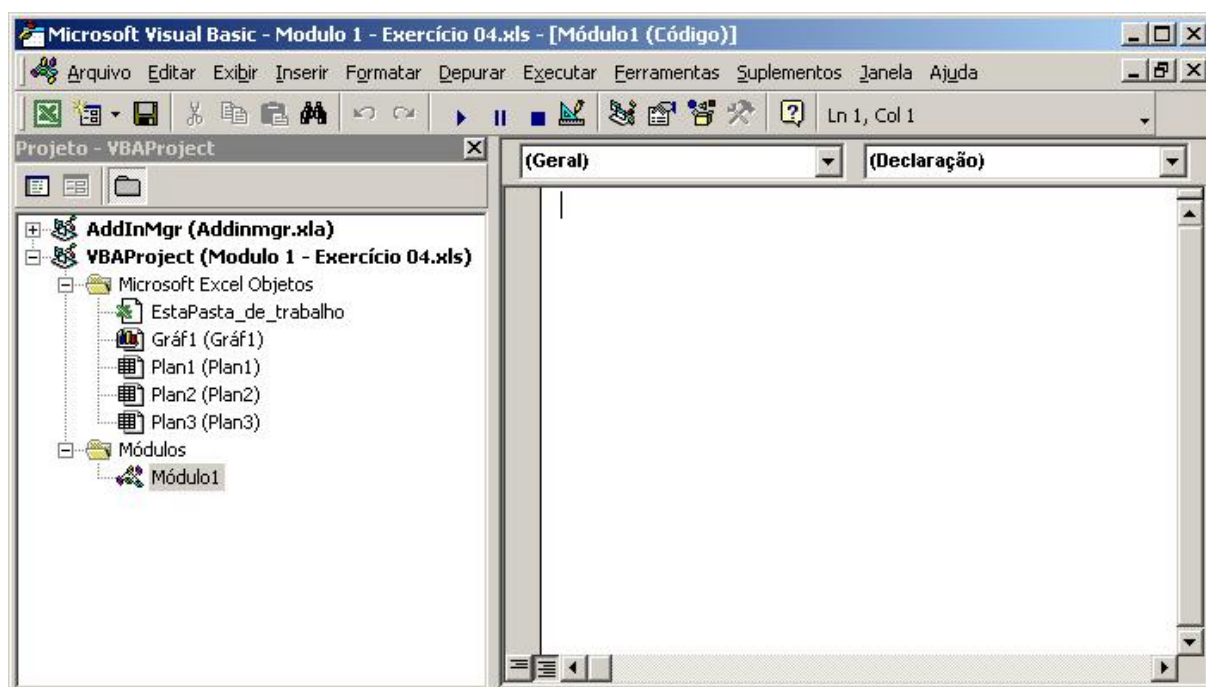
Nas próximas lições veremos mais algumas funcionalidades do Editor VBA.

Lição 08: O Ambiente de programação – o Editor VBA – Parte 2

Nesta lição mostrarei mais alguns importantes recursos do Ambiente de Programação do VBA. Mostrarei como o ambiente procura ajudar o programador, à medida que este digita o seu código. Também descreverei a estrutura de código contida em uma planilha do Excel. Para mostrar as funcionalidades do ambiente de programação, utilizarei um exemplo prático.

Exemplo 4: Acessar o Editor VBA para aprender sobre as funcionalidades do ambiente de programação e sobre a estrutura de código VBA em uma planilha.

1. Abra o Excel.
2. Abra a Planilha C:\Programação VBA no Excel\Módulo 1 – Exercício 04.xls.
3. Para abrir o Editor do VBA pressione Alt+F11 ou selecione o comando Ferramentas -> Macro -> Editor do Visual Basic.
4. O ambiente de programação do VBA será aberto, conforme indicado na figura a seguir:

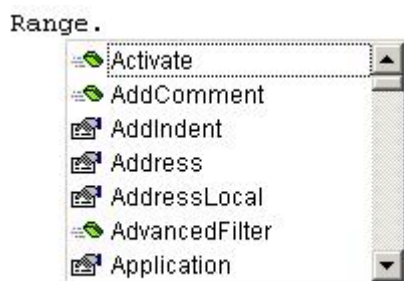


O Editor do VBA.

5. No lado esquerdo da janela, no painel VBAProject, são exibidas entradas para as diversas planilhas da pasta de trabalho atual (Gráf1, Plan1, Plan2 e Plan3 para o nosso exemplo). Você pode criar códigos, funções e rotinas associadas com uma determinada planilha da pasta de trabalho. Uma função ou rotina criada em uma planilha, poderá ser utilizada somente dentro da planilha onde foi criada. As rotinas que devem estar disponíveis para todas as planilhas da pasta de trabalho, devem ser criadas em Módulos independentes de código. No exemplo da Figura anterior, temos um módulo chamado Módulo 1. Para planilhas mais complexas, é possível criar vários módulos de código, dividindo os procedimentos em categoriais, tais como funções para cálculos de data, funções de validação de dados, funções

de cálculos financeiros e assim por diante. Esta divisão facilita a localização, manutenção e alteração das funções existentes. É muito importante que você entenda este ponto. Vamos repetir para fixar bem. As rotinas de código VBA em uma pasta de trabalho do Excel, podem ser criadas associadas com uma planilha ou com um Módulo de código. As rotinas que fazem parte de um módulo de código, poderão ser utilizadas em qualquer planilha da pasta de trabalho onde o Módulo foi criado.

6. Conforme você aprenderá nas lições do Módulo 2, a programação VBA no Excel, faz uso de um grande conjunto de objetos. Cada objeto contém um grande número de métodos e propriedades. O Editor do VBA facilita a utilização dos métodos e propriedades de um objeto, pois quando você digita o nome do objeto e depois um ponto, o Excel exibe uma lista de todas as propriedades e métodos do respectivo objeto. Esse recurso é um dos que eu considero mais importantes, pois evitam de o programador ter que decorar o nome de todos os métodos e propriedades e reduzem o número de vezes que o programador tem que consultar a Ajuda do Excel ou o manual de programação. No exemplo da Figura a seguir, estou utilizando o objeto Range (que você estudará em detalhes neste curso). Ao digitar Range. , o Editor do VBA exibe uma lista de métodos e propriedades do objeto Range, conforme indicado na Figura a seguir:



7. Após selecionar um método e digitar o parênteses de abertura, o Editor VBA apresenta a lista de argumentos que deve ser fornecida para o método selecionado, bem como o tipo de cada argumento, conforme indicado na Figura a seguir. Esta também é uma dica valiosa, que evita erros e evita que o programador tenha que fazer uma série de tentativas, até conseguir informar todos os parâmetros corretamente:

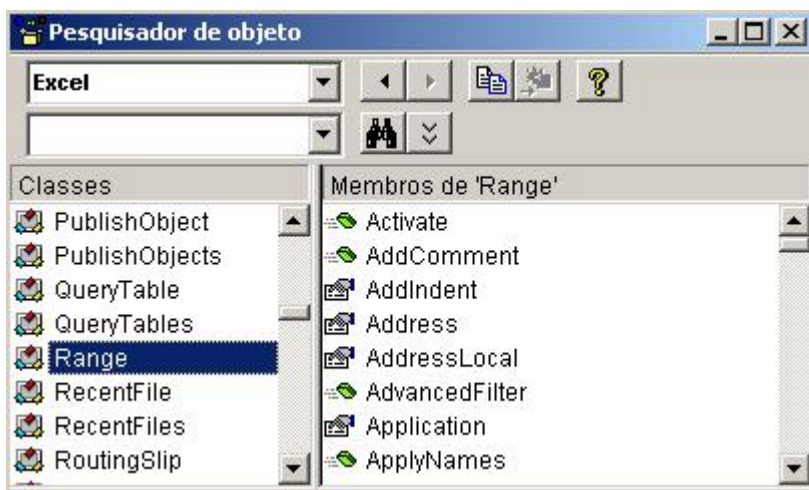
```
Range.Address(|
  Address([RowAbsolute], [ColumnAbsolute], [ReferenceStyle As XlReferenceStyle = xlA1], [External],
  [RelativeTo]) As String
```

8. Outro recurso muito útil do Editor VBA é a janela Pesquisador de Objetos. Esta janela exibe uma listagem de todas as bibliotecas disponíveis, bem como a lista de cada objeto de cada biblioteca, juntamente com todas as propriedades e métodos de cada objeto. É uma verdadeira “jóia”, que facilita muito o trabalho do Programador. Para exibir a janela Pesquisador de Objetos, pressione a tecla F2 ou selecione o comando Exibir -> Pesquisador de objeto. Será exibida a janela indicada na Figura a seguir:

Nota: No Módulo 2, apresentarei os conceitos de Biblioteca, Objetos, Métodos e Propriedades, em mais detalhes.



9. Na lista Todas as Bibliotecas você pode selecionar uma biblioteca específica, para que o Pesquisador de objeto, use somente os objetos da Biblioteca selecionada. Por exemplo, selecione a biblioteca Excel. Serão exibidos apenas os objetos do Excel. Na lista de objetos, localize o objeto Range e clique para marcá-lo. No painel da direita, será exibida uma lista dos métodos e propriedades do objeto Range, conforme indicado na figura a seguir:



10. Para obter ajuda sobre um método ou propriedade, clique no método ou propriedade desejado e depois clique no botão (?). A ajuda do Excel será aberta, e será carregada a página de ajuda sobre o método ou propriedade selecionada.

11. Para fechar o Pesquisador de objeto, clique no botão x, da janela do Pesquisador de objeto.

12. Para fechar o Editor do VBA, selecione o comando Arquivo -> Fechar e voltar para o Microsoft Excel. Muito bem, agora você já conhece os recursos do editor do VBA. O próximo passo é aprender os fundamentos da linguagem VBA, para criar suas primeiras rotinas de programação.

Lição 09: VBA - Declaração de Variáveis

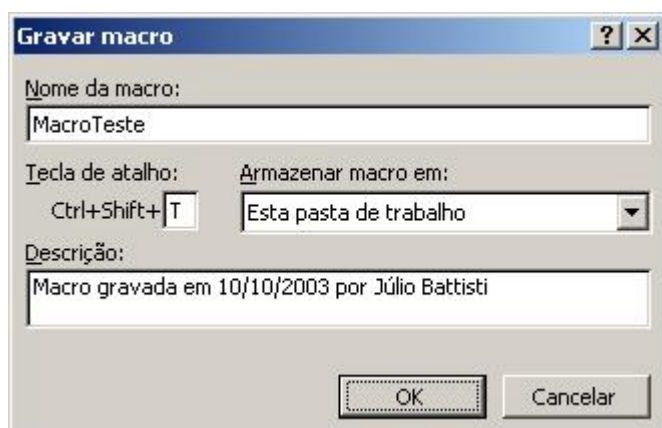
A partir desta lição, você iniciará o estudo da linguagem VBA. O VBA é composto por uma série de comandos básicos, os quais fazem parte de praticamente toda linguagem de programação. São comandos para declaração de variáveis, para realização de operações aritméticas e lógicas e comandos para fazer testes lógicos e para executar a repetição de um conjunto de comandos, com base em uma ou mais condições. Esses comandos básicos serão utilizados em praticamente todos os exemplos deste curso e em qualquer rotina de programação que você venha a desenvolver na prática.

Iniciaremos o nosso estudo de VBA pela definição do conceito de variáveis a aprendendo a declarar variáveis no VBA.

Nota: A medida que os conceitos forem sendo apresentados, faremos alguns testes. Para os testes criarei uma macro chamada MacroTeste, a qual irei associar um combinação de teclas de Atalho: Ctrl+Ç. Para testar os comandos, a medida que estes forem sendo apresentados, vamos acessar o código VBA da macro MacroTeste (usando o Editor VBA) e inserir os comandos a serem testados. Em seguida voltaremos para a planilha e executaremos a Macro para testar os comandos. Para facilitar o processo de execução da Macro usaremos a combinação de teclas Ctrl+T, a qual está associada à macro. Vou utilizar uma planilha chamada Módulo 1 – Exemplos Básicos VBA.xls. A seguir descrevo os passos para criação da MacroTeste e para associar a combinação Ctrl+T com esta macro. Você criará a macro sem nenhum comando. Os comandos serão inseridos e testados, nos exemplos práticos, das lições deste módulo.

Exemplo: Para criar a macro MacroTeste, siga os passos indicados a seguir:

1. Abra o Excel.
2. Abra a Planilha C:\Programação VBA no Excel\ Módulo 1 – Exemplos Básicos VBA.xls.
3. Selecione o comando Ferramentas -> Macro -> Gravar nova macro...
4. Será exibida a janela Gravar Macro. No campo Nome da macro:, digite MacroTeste. No campo Tecla de atalho, ao lado do Ctrl, digite T. Sua janela deve estar conforme indicado na Figura a seguir:



5. Clique em OK.
6. Será exibida a barra de gravação da Macro. Como queremos criar uma macro em branco, vamos parar a gravação da macro sem ter executado nenhum comando.



7. Clique no botão (■), da barra de ferramentas da macro (■), para encerrar a gravação da macro MacroTeste.
8. Pronto, agora temos uma macro chamada MacroTeste, a qual não tem nenhum comando. A cada exemplo, você irá alterar esta macro, inserindo comandos do VBA na Macro. Depois você usará a combinação de teclas Ctrl+T, para testar o funcionamento da macro. A cada novo exemplo, os comandos do exemplo anterior serão apagados e os comandos do novo exemplo digitados.

Declaração de variáveis e “tipos” de dados:

Uma variável é um espaço na memória do computador, reservado para armazenar um ou mais valores. Fazemos referência a este espaço utilizando nomes, ou seja, atribuímos nomes as variáveis utilizadas no VBA. Como o valor armazenado pode variar, a medida que o código VBA é executado, estas estruturas são chamadas de variáveis.

No VBA, não é obrigatório a declaração de variáveis. Porém é recomendável que declaremos todas as variáveis, de tal forma que o código fique mais claro e de fácil compreensão. Para declararmos uma variável, utilizamos o comando Dim, conforme exemplificado abaixo:

```
Dim x
Dim nome
Dim teste
```

Neste caso estamos apenas declarando o nome da variável, sem declarar de que tipo (texto, número inteiro, número com decimal, data, etc) é a variável. Uma variável declarada sem tipo é considerada do tipo Variant, o que na prática significa que a variável pode conter qualquer tipo de valor. Pode parecer uma prática interessante a não declaração do tipo da variável, porém isso é altamente desaconselhável. Se não declararmos o tipo, conforme descrito anteriormente, a variável poderá conter qualquer valor. Neste caso o que impede de um campo numérico conter valores de texto ou vice-versa??

A sintaxe para o comando Dim é a seguinte:

```
Dim nome_da_variável As tipo_da_variável
```

Também podemos declarar mais do que uma variável, com um único comando Dim. Para isto, basta separar os nomes das variáveis, com vírgula, conforme exemplificado abaixo:

```
Dim x, y, z As String
Dim nome as Double
Dim teste1, teste2 As Integer
```

IMPORTANTE: Observe que definimos o “tipo” de cada variável. O Tipo define quais dados podem ser armazenados em uma variável. Por exemplo, variáveis que armazenam valores numéricos, não devem aceitar caracteres de texto. Variáveis que armazenam datas, não devem aceitar datas inválidas, como por exemplo 30/02/2001. Toda variável no VBA, é do tipo Variant, isto significa que a variável pode ser de qualquer tipo. O que define o tipo da variável é o valor que está armazenado no momento. Existem funções que conversão de tipo, conforme veremos mais adiante.

Também podemos utilizar variáveis que não foram, explicitamente, declaradas com o comando Dim. Com isso, a variável é criada na memória, no momento da sua utilização. Para fazer com que toda variável tenha que ser, explicitamente, declarada, antes de ser utilizada, devemos utilizar o seguinte comando na seção de declaração do módulo:

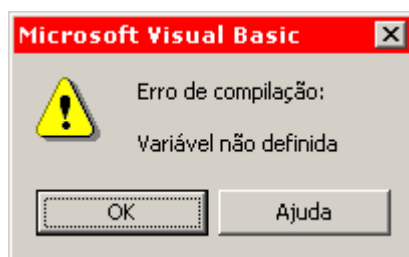
Option Explicit

Ao colocarmos este comando na seção de declaração do módulo, estamos definindo que toda variável deve ser declarada, antes de ser utilizada em uma expressão. Se tentarmos utilizar uma variável não declarada, será gerado um erro de compilação e a execução dos comandos é suspensa. Na Listagem 1, temos um exemplo simples de utilização de variáveis não declaradas explicitamente.

Listagem 1 – Utilização de variáveis não declaradas.

```
Dim a As Integer
Dim b As Integer
a=5
b=2
c=a+b
Msgbox "A variável C vale: " & c
```

Ao tentarmos executar este código, tendo sido definida a opção "Option Explicit", obteremos a mensagem de erro indicada na próxima figura. Este erro acontece porque tentamos utilizar uma variável c, variável esta que não foi declarada.



O tipo Variant é formado de pequenas unidades, chamadas subtipos. Cada subtipo, identifica de que maneira os dados são armazenados em uma variável do tipo Variant. Por exemplo, variáveis do subtipo Integer são armazenadas de uma maneira diferente de variáveis do subtipo Long. Na Tabela a seguir temos uma descrição dos principais subtipos.

Tabela - Subtipos do tipo Variant disponíveis no VBA:

Subtipo	Subtipo
Empty	O Valor é zero para variáveis numéricas ou uma String de tamanho zero (" "), para variáveis de texto.
Null	A variável não contém dados válidos.
Boolean	Contém variáveis que somente podem assumir dois valores: Verdadeiro ou Falso (True ou False).
Byte	Valor inteiro, na faixa de 0 até 255.
Integer	Valor inteiro, na faixa de -32768 até 32767.
Currency	Valores na faixa de -923.337.203.685.447,5808 até 922.337.203.685.447,5807
Long	Valor inteiro, na faixa de -2.147.483.648 até 2.147.483.647.
Date(Time)	É um número que representa a data entre 01 de Janeiro do ano 100, até 31 de Dezembro de 9999 (Olha o bug do ano 10000 chegando).
String	Texto de tamanho variável, pode conter, aproximadamente, 2 bilhões de caracteres.
Object	Pode conter um objeto qualquer, como um Controle Activex, ou um Objeto COM+
Error	Pode conter um número de erro.

Antes de fazermos alguns exemplos práticos, vamos aprender um pouco mais sobre o uso de variáveis no VBA. Vamos falar sobre operadores aritméticos e de comparação. Mas isso já é assunto para a próxima lição.

Lição 10: VBA - Cálculos, Operadores Aritméticos e Exemplos

Fazendo cálculos e comparações com o VBA - Operadores Aritméticos.

Para realizarmos cálculos e comparações entre variáveis, utilizamos operadores. Neste item trataremos sobre operadores aritméticos e operadores de comparação.

Fazendo cálculos com os Operadores aritméticos

Podemos realizar cálculos no VBA, utilizamos operadores aritméticos. Na Tabela a seguir, temos uma descrição dos operadores que podemos utilizar:

Operadores Aritméticos do VBA

Operador	Símbolo	Descrição
Adição	+	Soma o valor de duas ou mais variáveis.
Subtração	-	Subtração entre duas ou mais variáveis.
Multiplicação	*	Multiplica os valores de duas ou mais variáveis.
Divisão	/	Divide o valor de duas ou mais variáveis.
Inteiro da Divisão entre dois números	\	Retorna a parte inteira, da divisão entre dois números.
Exponenciação	^	x^y -> É o valor do número x, elevado na potência y
Modulo	Mod	Retorna o resto de uma divisão de 2 números.

Considere o pequeno trecho de código a seguir, onde declaramos as variáveis x, y e z. Atribuímos valores a essas variáveis e em seguida fazemos algumas operações. As linhas que iniciam com um apóstrofe são simplesmente comentários. Toda linha que iniciar com um apóstrofe será desconsiderada pelo VBA. Os comentários são importantes para documentação do código.

Listagem – Exemplo de uso de operadores aritméticos:

```
` Exemplo de utilização de variáveis e operadores aritméticos
` Curso: Programação VBA no Excel.
` Autor: Júlio Battisti
` Site: www.juliobattisti.com.br
```

```
` Declaração das variáveis.
```

```
Dim x, y, z As Integer
```

```
x = 10
```

```
y = 25
```

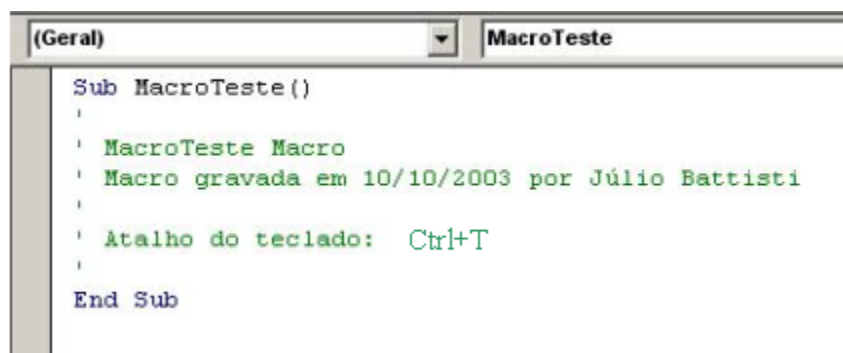
```
z = x*y
```

```
` Nesse exemplo a variável z conterà o valor 250
```

Vamos fazer um pequeno teste com os comandos para declaração de variáveis.

Exemplo: Acessar o Editor VBA para alterar a macro MacroTeste, da planilha Módulo 1 – Exemplos Básicos VBA.xls.

1. Abra o Excel.
2. Abra a Planilha C:\Programação VBA no Excel\Módulo 1 – Exemplos Básicos VBA.xls.
3. Como existem macros já criadas, o Excel pede uma confirmação, perguntando se você deseja ativar as macros existentes. Essa confirmação é solicitada devido ao perigo representado pelos vírus de Macro, que podem afetar os aplicativos do Microsoft Office.
4. Clique no botão Ativar Macros.
5. A planilha será aberta.
6. Nessa planilha existe uma macro chamada MacroTeste. Vamos editar essa macro. O nosso objetivo é utilizar o comando Dim para declarar algumas variáveis e os operadores aritméticos para fazer alguns cálculos. Os valores dos cálculos serão exibidos quando a macro for executada.
7. Selecione o comando Ferramentas -> Macro -> Macros...
8. Será exibida a janela Macro.
9. Clique na macro MacroTeste para selecioná-la.
10. Clique no botão Editar.
11. Será aberto o editor do VBA. Observe que como não gravamos nenhum comando na Macro (veja lição anterior), existe apenas a declaração da Macro e alguns comentários, conforme indicado na Figura a seguir:



MacroTeste somente com comentários

12. Abaixo da linha Atalho de Teclado: Ctrl+T, digite o trecho de código indicado a seguir:

```
` Exemplo de utilização de variáveis e operadores aritméticos
` Curso: Excel Avançado em 120 Lições.
` Autor: Júlio Battisti
` Site: www.juliobattisti.com.br

` Declaração das variáveis.

Selection.Font.Bold = True

Dim a As Integer
Dim b As Integer
```

```
Dim som, subtr, divis, mult, intdivs, expo, modul As Double

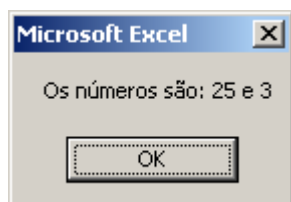
a = 25
b = 3
' Uso os operadores aritméticos para efetuar cálculos

som=a+b
subtr=a-b
divis=a/b
mult=a*b
intdivs=a\b
expo=a^b
modul= a mod b

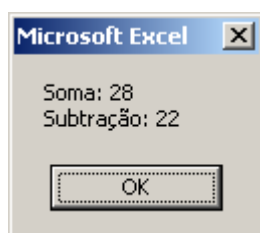
'Uso de MsgBox para exibir os resultados.

MsgBox "Os números são: " & a & " e " & b & Chr(13)
MsgBox "Soma: " & som & Chr(13)& "Subtração: " & subtr & Chr(13)
MsgBox "Divisão: " & divis & Chr(13)& "Multiplicação: " & mult & Chr(13)
MsgBox "Divisão inteira: " & intdivs & Chr(13)& "Exponenciação: " & expo & Chr(13)
MsgBox "Resto da divisão: " & modul
```

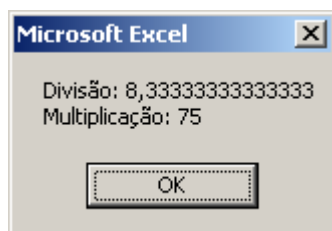
13. Após digitar o código feche o Editor do VBA.
14. Salve a Planilha.
15. Clique em qualquer célula da planilha.
16. Pressione Ctrl+T para executar a macro **MacroTeste**.
17. Será exibida a seguinte mensagem:



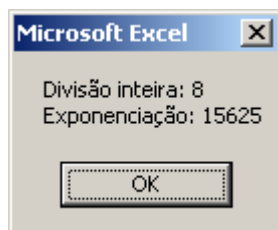
18. Dê um clique no botão OK.
19. Será exibida a seguinte mensagem:



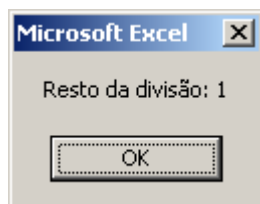
20. Dê um clique no botão OK.
21. Será exibida a seguinte mensagem:



- 22; Dê um clique no botão OK.
- 23. Será exibida a seguinte mensagem:



- 24. Dê um clique no botão OK.
- 25. Será exibida a seguinte mensagem:



- 26. Dê um clique no botão OK.
- 27. Você estará de volta à planilha.
- 28. Mantenha a planilha aberta pois iremos utiliza-la nas demais lições desse módulo.

Neste exemplo utilizamos os operadores aritméticos, para realizar uma série de operações com os valores atribuídos às variáveis “a” e “b”. Depois utilizamos a função MsgBox (que é um comando interno do próprio VBA), para exibir os resultados obtidos.

O uso da função MsgBox, merece alguns comentários. Primeiro, utilizamos a função MsgBox, para exibir mensagens em uma pequena janela, chamada de Janela Pop-Up (que é a janela exibida nas figuras desse exemplo). Podemos exibir texto e também valores de variáveis. Quando temos que exibir diversos componentes, como um pouco de texto, em seguida o valor de uma variável, depois mais texto, e assim por diante, devemos utilizar o **operador de concatenação: &**. Este operador permite que as diversas partes sejam exibidas como uma única mensagem. Considere o exemplo da seguinte linha de código:

MsgBox "Os números são: " & a & " e " & b & Chr(13)

Neste exemplo, primeiro é exibido o texto: "**Os números são:**", logo em seguida o valor da variável **a**. Observe que as duas partes são concatenadas com o operador **&**. Depois concatenamos o valor da variável **b**, e finalmente utilizamos a função **Chr(13)**. A função

Chr(número), envia o caractere correspondente ao número digitado entre parênteses. Este número, é o número do caractere no padrão ASCII (American Standard Character Interchange Information). No código ASCII, cada caracter possui um número associado. No caso o número 13, é associado ao <ENTER>. Com isso estamos enviando um <ENTER>, para a caixa de mensagem, o que equivale a uma troca de linha. Assim utilizamos o Chr(13), para simular um <ENTER>, de tal maneira que os dados não saiam todos “emendados”, na mesma linha.

Com o uso dos operadores aritméticos , realizamos uma série de operações sobre as variáveis existentes no código da macro AplicaNegrito.

Vamos avançar um pouco mais no nosso estudo sobre operadores. Vamos tratar dos operadores de comparação. Esse é justamente o assunto da próxima lição.

Lição 11: VBA - Estrutura If...Then e os Operadores de Comparação

Além de declaração de variáveis e cálculos básicos, o VBA fornece uma série de comandos conhecidos como estruturas de controle e repetição. As estruturas de controle são utilizadas para a realização de testes. Por exemplo: **“se o valor de x for maior do que y, execute estes comandos, caso contrário execute os seguintes comandos”**. Os testes condicionais são intensamente utilizados em programação. Eu até diria que é impossível não utilizá-los, conforme você mesmo constatará nos diversos exemplos deste curso.

Em um primeiro grupo, temos as chamadas **estruturas de decisão**. São comandos/estruturas que realizam um teste lógico, e executam determinados comandos quando o teste resultar verdadeiro, ou um conjunto diferente de comandos, quando o teste resultar falso. Agora passaremos a analisar as estruturas de decisão, disponíveis no VBA.

A estrutura If ... Then

A estrutura If...Then, é uma das estruturas de decisão mais conhecidas. Toda linguagem de programação implementa esta estrutura. É utilizada para executar determinados comandos, caso uma condição seja verdadeira. A forma geral desta estrutura é a seguinte:

```
If condição Then
    Comando1
    Comando2
    ...
    Comandon
End IF.
```

Uma condição é testada, caso a condição seja verdadeira, um ou mais comandos podem ser executados. Considere o exemplo a seguir:

```
Dim x, y

x=10
y=15

If x<y Then
    MsgBox "x é menor do que y"
End If
```

Neste exemplo, a mensagem **“x é menor do que y”**, será exibida, uma vez que o teste **x<y** é verdadeiro, conforme podemos constatar pelos valores atribuídos as variáveis x e y.

A estrutura If ... Then...Else

A estrutura If...Then...Else, acrescenta mais uma possibilidade a estrutura If...Then. É utilizada para executar determinados comandos, caso uma condição seja verdadeira, ou um conjunto diferente de comandos, caso a condição seja falsa. A forma geral desta estrutura é a seguinte:

```
If condição Then
    Comando1
    Comando2
    ...
    Comandon
Else
    Comando1
    Comando2
    ...
    Comandon
End IF
```

Uma condição é testada, caso a condição seja verdadeira, um determinado comando, ou conjunto de comandos será executado; caso a condição seja falsa, um comando, ou conjunto de comandos diferentes, será executado. Considere o exemplo a seguir:

```
Dim x, y

x=15
y=10

If x<y Then
    MsgBox "x é menor do que y"
Else
    MsgBox "x é maior do que y"
End If
```

Neste exemplo, a mensagem “**x é maior do que y**”, será exibida, uma vez que o teste **x<y** é **falso**, conforme podemos constatar pelos valores atribuídos as variáveis x e y.

IMPORTANTE: Observe que os comandos são colocados em uma posição mais “para dentro”, endentados em relação ao If e ao Else. Esta colocação, que chamamos de endentação de código, não é obrigatório, mas é altamente recomendada. A endentação facilita a leitura do código. Observe o código a seguir, onde não foi utilizada endentação e compare-o com o exemplo anterior, onde foi utilizada a endentação:

```
If condição Then
Comando1
Comando2
...
Comandon
Else
Comando1
Comando2
...
Comandon
End IF
```

Fica claro que o uso de endentação facilita, e muito, a leitura e interpretação do código VBA. Por isso vai aqui a minha recomendação: **Sempre utilize endentação no código que você criar.**

A estrutura If ... Then...ElseIf...Else

A estrutura **If...Then...ElseIf**, nos dá um poder maior, para testarmos diversas possibilidades, isto é, para fazermos vários testes e executarmos diferentes comandos com base no resultado dos testes. Esta estrutura é utilizada quando precisamos realizar mais do que um teste lógico. Neste caso, para cada novo teste que se faça necessário, utilizamos um ElseIf. A forma geral desta estrutura é a seguinte:

If condição Then

Comandos a serem executados, caso a condição seja verdadeira.

Comando1
Comando2
...
Comandon

ElseIf condição-2

Comandos a serem executados, caso a condição2 seja verdadeira.

Comando1
Comando2
...
Comandon

ElseIf condição-3

Comandos a serem executados, caso a condição3 seja verdadeira.

Comando1
Comando2
...
Comandon

...

ElseIf condição-n

Comandos a serem executados, caso a condição n seja verdadeira.

Comando1
Comando2
...
Comandon

Else

Comandos a serem executados, caso nenhuma das condições anteriores seja verdadeira.

Comando1
Comando2
...
Comandon

End IF.

Uma condição é testada, caso a condição seja verdadeira, um determinado comando, ou conjunto de comandos será executado; caso a condição seja falsa, podemos fazer um segundo teste (condição-2). Caso a segunda condição seja verdadeira, um determinado comando, ou conjunto de comandos será executado, assim por diante, para n condições. Caso nenhuma das condições seja verdadeira, os comandos após a cláusula Else, serão executados.

NOTA: É importante observar que somente um dos conjuntos de comandos será executado. Quando uma condição verdadeira for encontrada, o conjunto de comandos associado a essa condição será executado e a execução segue a partir do End If, isto é, as demais condições da estrutura não serão testadas.

Considere o exemplo a seguir:

```
Dim x, y
Dim z, k
Dim w, p

x=35
y=30
z=25
k=20
w=15
p=10

If x<y Then
    MsgBox "x é menor do que y"
ElseIf x<z Then
    MsgBox "x é menor do que z"
ElseIf x<k Then
    MsgBox "x é menor do que k"
ElseIf x<w Then
    MsgBox "x é menor do que w"
ElseIf x<p Then
    MsgBox "x é menor do que p"
Else
    MsgBox "x é o maior dos números"
End If
```

Neste exemplo, a mensagem **"x é maior dos números"**, será exibida, uma vez que todos os testes anteriores falham (pois x é o maior número), com isso somente será executado o último MsgBox, que faz parte do Else.

Mas o que acontece, quando um dos ElseIf é verdadeiro?

Os teste vão sendo feitos. Quando o teste de um dos ElseIf for verdadeiro, os comandos abaixo do ElseIf verdadeiro, serão executados e o laço será encerrado. **Em resumo, quando um dos ElseIf apresentar um teste verdadeiros, os comandos relacionados serão executados, e os demais não serão avaliados, seguindo a execução para o primeiro comando, após o End If.**

A estrutura Select...Case

Quando precisamos realizar uma série de testes, é mais eficiente utilizarmos uma única estrutura **Select...Case**, do que utilizarmos uma série de testes utilizando a estrutura **If...Then...ElseIf**.

O funcionamento da estrutura Select...Case, é bastante intuitivo.

Considere o exemplo a seguir:

```
Dim x
x=10

Select Case x
    Case 2
        MsgBox "X vale 2 !"
    Case 4
        MsgBox "X vale 4 !"
    Case 6
        MsgBox "X vale 6 !"
    Case 8
        MsgBox "X vale 8 !"
    Case 10
        MsgBox "X vale 10 !"
    Case Else
        MsgBox "X não é um número par, menor do que 12 "
End Select
```

A estrutura **Select Case x**, vai testar o valor de x. Em cada um dos Case, o valor de x está sendo testado. Quando for encontrado um valor coincidente com o de x, os comandos abaixo deste Case serão executados. No nosso exemplo, o comando **MsgBox “X vale 10 !”**, abaixo de Case 10, será executado. **O comando abaixo do Case Else somente será executado, se todos os testes anteriores falharem.**

O uso da estrutura Select...Case, torna o código mais eficiente e de mais fácil leitura.

Na próxima lição veremos mais algumas estruturas de controle, disponíveis no VBA.

Lição 12: Estruturas For...Next, Do...While e Do...Until.

Estruturas de repetição

Em determinadas situações, precisamos repetir um ou mais comandos, um número específico de vezes, ou até que uma determinada condição torne-se verdadeira ou falsa. Por exemplo, pode ser que haja a necessidade de percorrer todas as linhas de dados de uma determinada planilha, até que o último registro seja alcançado. Para isso, utilizamos as chamadas estruturas de repetição, ou Laços. A partir de agora, aprenderemos as estruturas de repetição disponíveis.

A estrutura For...Next

Utilizamos o laço For...Next, para repetir um ou mais comandos, um número determinado de vezes. Utilizamos esta estrutura, quando já sabemos o número de vezes que uma determinada seção de código deve ser repetida ou quando o número de repetições é baseado no valor de uma variável. Neste tipo de estrutura, normalmente, utilizamos uma variável como contador. Este contador varia de um valor inicial até um valor final. O Formato geral desta estrutura é o seguinte:

```
For contador = inicio to fim incremento  
    Comando1  
    Comando2  
    ...  
    Comandon  
Next
```

No início a variável contador tem o valor definido. Em cada passagem do laço, a variável contador é incrementada pelo valor definido em incremento. Caso não seja definido o incremento, será utilizado o padrão 1.

Considere o exemplo a seguir:

```
Dim x  
x=10  
Soma=0  
  
' Faz a soma dos 10 primeiros números maiores do que zero  
  
For i=1 to x  
    Soma = Soma +i  
Next  
  
MsgBox "Valor da Soma = " & Soma
```

Neste exemplo, a variável i inicia com o valor 1. Em cada passo, o valor de i é acrescentado à variável Soma. Como o incremento não foi definido, será utilizado o padrão que é 1. Com isso, ao final do laço For...Next, a variável Soma, terá armazenado o valor da soma dos 10 primeiros números inteiros.

Poderíamos utilizar um valor de incremento diferente de 1, para, por exemplo, obter a soma somente dos números ímpares menores do que 10:

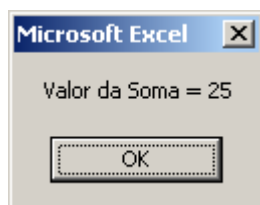
```
Dim x
x=10
Soma=0

' Faz a soma dos 10 primeiros números ímpares

For i=1 to x Step 2
    Soma = Soma +i
Next

MsgBox "Valor da Soma = " & Soma
```

Com este código, obtemos o resultado indicado na Figura a seguir:



A estrutura Do...Loop

Esta estrutura pode ser utilizada para repetir um trecho de código, **enquanto uma determinada condição for verdadeira, ou até que uma determinada condição torne-se verdadeira**. Podemos utilizar dois operadores condicionais diferentes: **While** ou **Until**. Os operadores While ou Until, podem ser utilizados de duas maneiras diferentes: No início do laço, ou no final do laço. Com isso temos quatro situações distintas, vamos analisar cada uma delas, a partir de agora.

A estrutura Do While Condição...Loop

Neste caso, estamos utilizando o operador condicional While, no início do laço. O formato geral, neste caso é o seguinte:

```
Do While condição/teste
    Comando1
    Comando2
    ...
    Comandon
Loop
```

Nesta estrutura, enquanto a condição for verdadeira, o código dentro do laço continuará a ser executado. Quando a condição tornar-se falsa, o primeiro comando após o final do laço, será executado. Neste caso, se a condição for falsa já na primeira vez, o laço não será executado nenhuma vez.

IMPORTANTE: O Código dentro do laço deve ser capaz de alterar a condição para que essa se torne Falsa, em algum momento, pois caso contrário a condição será sempre verdadeira, e os comandos dentro do laço ficarão em execução, infinitamente, ou até o programa travar. A criação de laços infinitos, devido a erros de programação, é uma causa comum de erros e travamentos.

Considere o exemplo a seguir:

```
Dim x
x=10
Contador=1
Soma=0

' Faz a soma dos 10 primeiros números maiores do que zero

Do While Contador <= x
    Soma = Soma + Contador
    Contador = Contador + 1
Loop

MsgBox "Valor da Soma = " & Soma
```

Observe que dentro do laço, vamos Incrementando o valor da variável Contador, uma unidade para cada passagem do laço. Com isso, quando o valor de Contador, atingir 11, o teste do início do laço torna-se falso, e o laço é encerrado.

A estrutura Do... Loop While Condição

Neste caso, deslocamos o teste de condição para o final do laço. Com o teste no final do laço, o código dentro do laço, será executado, pelo menos uma vez, pois o teste somente é feito no final, e continuará sendo executado, enquanto a condição for verdadeira. O formato geral, neste caso é o seguinte:

```
Do
    Comando1
    Comando2
    ...
    Comandon
Loop While condição/teste
```

Nesta estrutura, enquanto a condição for verdadeira, o código dentro do laço é executado. Quando a condição tornar-se falsa, o primeiro comando após o final do laço, será executado. Neste caso, se a condição for falsa já na primeira vez, o laço será executado uma única vez.

IMPORTANTE: O Código dentro do laço deve ser capaz de alterar a condição para Falso, quando for necessário, pois caso contrário a condição será sempre verdadeira, e os comandos dentro do laço ficarão em execução, infinitamente, ou até o programa travar. A criação de laços infinitos, devido a erros de programação, é uma causa comum de erros e travamentos.

Considere o exemplo a seguir:

```
Dim x
x=10
Contador=1
Soma=0

' Faz a soma dos 10 primeiros números maiores do que zero

Do
    Soma = Soma + Contador
    Contador = Contador + 1
Loop While Contador <= x

MsgBox "Valor da Soma = " & Soma
```

Vamos modificar um pouco o nosso exemplo. Considere o exemplo mostrado a seguir:

```
Dim x
x=10
Contador=11
Soma=0

' O laço será executado uma única vez, pois a condição
' Contador < x é falsa.

Do
    Soma = Soma + Contador
    Contador = Contador + 1
Loop While Contador <= x

MsgBox "Valor da Soma = " & Soma
```

Qual o valor será exibido para a variável Soma?

Muito simples. A condição **Contador<x** é falsa, pois x=10 e Contador=12 (Lembre que o Contador foi incrementado de uma unidade dentro do laço, antes do teste ser realizado). Neste caso, o laço será executado uma única vez, pois o teste de condição está no final do laço. Quando o laço é executado, é atribuído o valor 11 para a variável Soma. Com isso, o valor exibido para a variável Soma, será 11.

A estrutura Do Until Condição...Loop

Neste caso, estamos utilizando o operador condicional Until, no início do laço. O formato geral, neste caso é o seguinte:

```
Do Until condição
    Comando1
    Comando2
    ...
    Comandon
Loop
```

Nesta estrutura, enquanto a condição for falsa, o código dentro do laço é executado. Quando a condição tornar-se verdadeira, o primeiro comando após o final do laço, será executado. Neste caso, se a condição for verdadeira, já na primeira vez, o laço não será executado nenhuma vez.

IMPORTANTE: O Código dentro do laço deve ser capaz de tornar a condição Verdadeira, quando for necessário, pois caso contrário a condição será sempre Falsa, e os comandos dentro do laço ficarão em execução, infinitamente, ou até o programa travar. A criação de laços infinitos, devido a erros de programação, é uma causa comum de erros e travamentos.

Considere o exemplo a seguir:

```
Dim x
x=10
Contador=1
Soma=0

' Faz a soma dos 10 primeiros números maiores do que zero

Do Until Contador > x
    Soma = Soma + Contador
    Contador = Contador + 1
Loop

MsgBox "Valor da Soma = " & Soma
```

Observe que dentro do laço, vamos Incrementando o valor da variável Contador, uma unidade para cada passagem do laço. Com isso, quando o valor de Contador, atingir 11, o teste do início do laço torna-se Verdadeiro, e o laço é encerrado.

A estrutura Do... Loop Until Condição

Neste caso, deslocamos o teste de condição para o final do laço. Com o teste no final do laço, o código dentro do laço, será executado, pelo menos uma vez, pois o teste somente é feito no final, e continuará sendo executado, enquanto a condição for Falsa. O formato geral, neste caso é o seguinte:

```
Do
    Comando1
    Comando2
    ...
    Comandon
Loop Until condição
```

Nesta estrutura, enquanto a condição for Falsa, o código dentro do laço é executado. Quando a condição tornar-se Verdadeira, o primeiro comando após o final do laço, será executado. Neste caso, se a condição for Verdadeira, já na primeira vez, o laço será executado uma única vez.

IMPORTANTE: O Código dentro do laço deve ser capaz de tornar a condição Verdadeira, quando for necessário, pois caso contrário a condição será sempre Falsa, e os comandos dentro do laço ficarão em execução, infinitamente, ou até o programa travar. A criação de laços infinitos, devido a erros de programação, é uma causa comum de erros e travamentos.

Considere o exemplo a seguir:

```
Dim x
x=10
Contador=1
Soma=0

' Faz a soma dos 10 primeiros números maiores do que zero

Do
    Soma = Soma + Contador
    Contador = Contador + 1
Loop Until Contador > x

MsgBox "Valor da Soma = " & Soma
```

Vamos modificar um pouco o nosso exemplo. Considere o exemplo a seguir:

```
Dim x
x=10
Contador=11
Soma=0

' O laço será executado uma única vez, pois a condição
' Contador >= x é verdadeira.

Do
    Soma = Soma + Contador
    Contador = Contador + 1
Loop Until Contador >= x

MsgBox "Valor da Soma = " & Soma
```

Qual o valor será exibido para a variável Soma?

Muito simples. A condição `Contador >= x` é falsa, pois `x=10` e `Contador=12` (Lembre que o Contador foi incrementado de uma unidade, na linha 11, na primeira passagem do laço). Neste caso, o laço será executado uma única vez, pois o teste de condição está no final do laço. Quando o laço é executado, é atribuído o valor 11 para a variável Soma. Com isso, o valor exibido para a variável Soma, será 11.

Lição 13: VBA - Funções do VBA – Funções de Tipo – Parte 1

Além dos comandos básicos do VBA, tais como declaração de variáveis, operadores aritméticos, estruturas de decisão e laços, estão disponíveis para uso no VBA, as centenas de funções de planilha disponíveis no Excel. Por exemplo, você pode usar no código VBA, funções tais como SOMA, Média e assim por diante. Um detalhe importante é que no código VBA, devem ser utilizadas as funções com o nome em Inglês: SUM ao invés de SOMA, Avg ao invés de Média e assim por diante.

A partir dessa lição veremos uma série de aspectos que vão além do básico do VBA. Inicialmente vamos estudar uma série de funções disponíveis no Excel. Vamos dividir estas funções em categorias para facilitar o seu entendimento. Iniciaremos falando um pouco mais sobre tipos de dados e funções para conversão de tipos de dados, ou seja, funções que fazem a conversão de um tipo de dado para outro, como por exemplo de texto para número. Este tópico é de grande importância, principalmente quando criamos código que efetua cálculos, tais como uma folha e pagamentos ou cálculos de impostos.

Depois passaremos a analisar uma série de funções internas do VBA. São funções que fazem parte do Microsoft Excel, como por exemplo a função Date(), que retorna a data do sistema. Na sequência do módulo, aprenderemos a criar nossas próprias funções e Sub-rotinas. Aprenderemos as diferenças entre uma Sub-rotina e uma função, e quando utilizar uma ao invés da outra.

Tipos de dados e funções para conversão de tipos

Neste item, aprenderemos a determinar qual o tipo de um dado que está armazenado em uma variável, bem como a converter valores de um tipo para outro, utilizando as funções para conversão de tipos.

Determinando o Tipo de Dados contido em uma variável.

Existem diversas funções, que permitem que seja determinado o tipo de valor contido em uma variável. Existem diversas aplicações para este tipo de função. Por exemplo, ao digitar dados em um formulário do Excel (nos próximos módulos aprenderemos a criar formulários), podemos utilizar uma função para determinar se os valores digitados pelo usuário, não apresentam problemas. Por exemplo, o usuário pode ter digitado, por engano, texto em um campo que deve conter valores numéricos.

A função IsArray

Um Array é um tipo especial de variável, a qual pode armazenar diversos valores em uma única variável. De uma forma simples, um Array é um conjunto ou como aprendi nos bons e velhos tempos da faculdade de Engenharia Elétrica na UFSM: Um vetor. Por exemplo, poderíamos ter uma variável Array na qual são armazenados todos os códigos de tributo, válidos para o recolhimento de impostos. Cada valor armazenado no Array é um elemento do conjunto. Um outro exemplo: Poderíamos criar um Array para armazenar os nomes dos meses do ano. Com isso teríamos um Array de 12 elementos.

Cada elemento de um Array, é acessado através do nome da variável Array e de um índice. **O índice inicia em zero e não em um.** Por isso se tivermos um Array de 10 elementos, teremos o elemento 0, o elemento 1, o elemento 2, e assim por diante, até o elemento 9. O fato do índice começar com 0, influencia na hora que formos declarar um Array. Para declarar um Array chamado produtos, com 20 elementos, utilizaríamos o seguinte comando:

Dim produtos(19)

O 19 significa que temos 20 elementos (sempre um a mais do que o número que aparece na declaração), isto é, do elemento 0, indicado por produtos(0), até o elemento 20, indicado por produtos(19).

No exemplo a seguir, temos um exemplo simples de utilização de Array:

```
'Declara um Array de 7 posições
'Como a primeira posição é a posição zero,
'Indicamos o tamanho como 6, o que significa
'da posição 0 até a posição 6 = 7 elementos.

Dim Dias(6)

'Atribuimos valores para os dias da semana

Dias(0)= "Segunda-feira"
Dias(1)= "Terça-feira"
Dias(2)= "Quarta-feira"
Dias(3)= "Quinta-feira"
Dias(4)= "Sexta-feira"
Dias(5)= "Sábado"
Dias(6)= "Domingo"

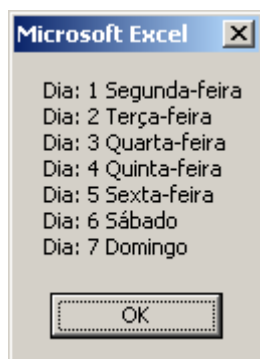
'Agora utilizamos um laço For...Next
'Para criar uma String, com o conteúdo
'Do Array Dias.

For i=0 to 6
    mensagem = mensagem & "Dia: " & i+1 & " " & Dias(i)& Chr(13)
Next

'Utilizamos uma MsgBox, para exibir a mensagem
'com o conteúdo do Array Dias.

MsgBox mensagem
```

Esse trecho de código, ao ser executado, produziria a seguinte saída:



Utilizamos a função IsArray, para determinar se uma variável do tipo Variant está armazenando um Array (lembre que todas as variáveis do VBA, são do tipo Variant e que possuem um subtipo definido pelo valor armazenado na variável).

Utilizamos a seguinte sintaxe:

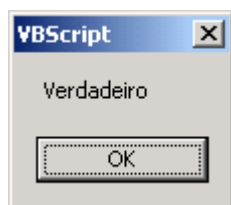
IsArray(NomeDaVariável)

A função IsArray retorna verdadeiro se a variável for um Array, e falso caso contrário.

Vamos alterar um pouco o exemplo anterior, adicionando o seguinte comando, ao final da listagem.:

MsgBox IsArray(Dias)

o resultado seria o indicado na Figura a seguir:



A função VarType

Podemos utilizar a função VarType, para determinar o subtipo de uma variável. Como todas as variáveis são do tipo Variant, o subtipo pode ser utilizado para determinar o tipo de dado armazenado na variável. Passamos para a função, o nome de uma variável ou expressão. A função retorna um número inteiro que indica o subtipo da variável.

A sintaxe da função é a seguinte:

VarType(NomeDaVariável)

ou

VarType(expressão)

Na Tabela a seguir temos os códigos de retorno da função VarType.

Tabela - Valores de retorno da função VarType

Valor	Descrição
0	Vazio (não inicializado)
1	Nulo (dados não válidos)
2	Inteiro
3	Inteiro longo
4	Número de precisão simples
5	Número de precisão dupla
6	Monetário.
7	Data
8	Texto
9	Objeto de automação
10	Erro
11	Booleano
12	Variant (somente é utilizado com Arrays de variantes)
13	Um objeto para acesso a dados.
17	Byte
8192	Array

No exemplo a seguir, temos um exemplo de utilização da função VarType.

```
Dim x,y,z
Dim a, b
Dim c(20)
Dim mensagem As String

x=12
y=23.456
y=123456789
a="Este é um valor de texto !"

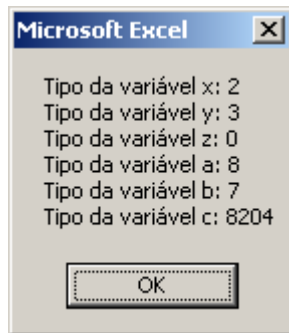
'Utilizamos a função Date( ), para capturar a data do sistema.

b=Date()

mensagem = "Tipo da variável x: " & Vartype(x) & Chr(13)
mensagem = mensagem & "Tipo da variável y: " & Vartype(y) & Chr(13)
mensagem = mensagem & "Tipo da variável z: " & Vartype(z) & Chr(13)
mensagem = mensagem & "Tipo da variável a: " & Vartype(a) & Chr(13)
mensagem = mensagem & "Tipo da variável b: " & Vartype(b) & Chr(13)
mensagem = mensagem & "Tipo da variável c: " & Vartype(c) & Chr(13)

MsgBox mensagem
```

Ao executar este exemplo, obteremos o resultado indicado na Figura a seguir:



Observe, principalmente, no tipo da variável z, o qual retornou 0. Isto significa que a variável z está vazia, ou seja, não foi inicializada. Este resultado está coerente com a Tabela anterior. Também podemos salientar o retorno para a variável b, o qual retornou 7, indicando que o valor armazenado nesta variável é do tipo Data, o que também está coerente com a Tabela anterior, uma vez que usamos a função `Date()`, para armazenar a data do sistema, na variável b. Finalmente observe o valor de retorno para a variável c, o qual retornou 8192, indicando que a variável c é um Array. Este resultado está coerente com a Tabela anterior, e com a declaração: `Dim c(20)`, a qual declarou a variável c como sendo um Array.

Muitas são as aplicações práticas para a função `VarType`, dentre as quais podemos destacar a validação dos dados digitados em um formulário.

Lição 14: VBA - Funções do VBA – Funções de Tipo – Parte 2

Nesta lição continuaremos o estudo das funções para determinação e conversão de tipos no VBA.

A função IsDate

A função IsDate recebe uma variável ou expressão como argumento, e determina se a variável ou expressão é uma data válida, ou pode ser convertida para uma data válida. Caso o argumento passado seja uma data válida, a função retorna Verdadeiro, caso contrário, retorna Falso. Podemos utilizar esta função, por exemplo, para verificar se o usuário digitou uma data válida, em um campo de um formulário.

A sintaxe da função IsDate é a seguinte:

IsDate(NomeDaVariável)

ou

IsDate(expressão)

A seguir temos um exemplo de utilização da função IsDate.

```
If IsDate(x) Then
    MsgBox "Você digitou uma data válida !"
Else
    MsgBox "Data inválida, digite novamente !"
End If
```

A função IsEmpty

A função IsEmpty recebe uma variável ou expressão como argumento, e determina se, em algum momento, foi atribuído algum valor para a variável ou expressão. Caso tenha sido atribuído algum valor, a função retorna Verdadeiro, caso contrário, retorna Falso. Podemos utilizar esta função, por exemplo, para verificar se um campo de digitação obrigatória, como por exemplo o nome, não foi deixado em branco.

A sintaxe da função IsEmpty é a seguinte:

IsEmpty(NomeDaVariável)

ou

IsEmpty(expressão)

A seguir temos um exemplo de utilização da função IsEmpty.

Vamos declarar uma variável x, que nunca
Será utilizada no nosso Script

```
Dim a, b, c  
Dim x
```

```
a=10  
b=23  
c=a+b
```

```
If IsEmpty(x) Then  
    MsgBox "A variável x, não foi utilizada !"  
End If
```

A variável x foi declarada porém não foi inicializada, com isso está vazia, logo a função IsEmpty(x) irá retornar Verdadeiro.

A função IsNull

A função IsNull recebe uma variável ou expressão como argumento, e determina se, em algum momento, foi atribuído o valor Null para a variável ou expressão. Caso tenha sido atribuído o valor Null, a função retorna Verdadeiro, caso contrário, retorna Falso. Para atribuirmos Null para uma variável, utilizamos a seguinte sintaxe:

NomeDaVariável = Null

IMPORTANTE: Uma variável com valor Null, não é a mesma coisa que uma variável com valor zero, ou uma variável de texto com tamanho zero. Quando é atribuído o valor Null para a variável, esta continua existindo na memória, porém sem nenhum valor definido.

A sintaxe da função IsNull é a seguinte:

IsNull(NomeDaVariável)
ou
IsNull(expressão)

A seguir temos um exemplo de utilização da função IsNull.

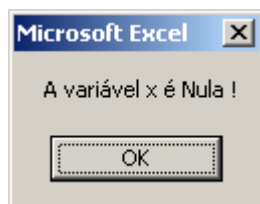
```
' Vamos declarar uma variável x, e atribuir  
' Null, para a variável.
```

```
Dim x
```

```
x = Null
```

```
    If IsNull(x) Then  
        MsgBox "A variável x é Nula !"  
    End If
```

Este exemplo produz o seguinte resultado:



A função IsNumeric

A função IsNumeric recebe uma variável ou expressão como argumento, e determina se o valor atribuído à variável ou expressão é numérico, ou pode ser convertido para numérico. Caso o valor seja numérico, ou possa ser convertido, a função retorna Verdadeiro, caso contrário, retorna Falso.

A sintaxe da função IsNumeric é a seguinte:

IsNumeric(NomeDaVariável)

ou

IsNumeric(expressão)

A seguir temos um exemplo de utilização da função IsNumeric.

```
Dim x,y,z

x=123

'Atribuo um valor que não pode ser convertido
'para numérico

y = "Riachuelo - 80"

z = Date()

mensagem = "Valor de x: " & x & " É numérico ? " & IsNumeric(x)& Chr(13)
mensagem = mensagem & "Valor de y: " & y & " É numérico ? " & IsNumeric(y)& Chr(13)
mensagem = mensagem & "Valor de z: " & z & " É numérico ? " & IsNumeric(z)
MsgBox mensagem
```

Cabe salientar a utilização da função Date(), para capturar a data do sistema, e atribuir esta data à variável z. Observe que esta data não foi considerada um valor numérico para a função IsNumeric.

Lição 15: VBA - Funções do VBA – Funções de Conversão de Tipo – Parte 1

Nesta lição aprenderemos a utilizar as principais funções para conversão de tipos. Existem situações em que um determinado tipo de dado, deve ser convertido para outro. Por exemplo, se tivermos um número, armazenado na forma de texto, precisamos convertê-lo para inteiro ou double, para que possamos realizar cálculos. Em um dos exemplos no final deste módulo, faremos um exemplo de cálculo do DV do CPF, onde o CPF é um valor do tipo texto. Ao extraírmos cada dígito do CPF, estes serão extraídos como caracteres de texto. Precisaremos utilizar uma função de conversão, para convertê-los para números, a fim de que possamos efetuar os cálculos necessários.

Na sequência, apresento as principais funções de conversão, bem como um pequeno fragmento de código, exemplificando a utilização de cada uma delas.

Função Cbool

A função Cbool converte uma variável ou resultado de uma expressão, para o subtipo Boolean (Verdadeiro ou Falso). **Qualquer número, com exceção do zero, é automaticamente convertido para Verdadeiro. O valor zero é sempre convertido para Falso.** O argumento desta função, não pode ser Texto, caso contrário será gerado um erro em tempo de execução.

Observe este linha de código:

```
MsgBox cbool(10>25) & chr(13) & cbool(3)
```

Esta linha gera a mensagem indicada na Figura a seguir:



A expressão 10>25 é avaliada, como a expressão é falsa, a função Cbool retorna Falso. Já no segundo uso da função Cbool, foi passado o parâmetro 3 para a função. Qualquer valor diferente de zero (com exceção de texto), a função interpreta como Verdadeiro, o que é comprovado pela Figura anterior.

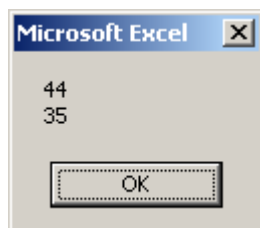
Função CByte

A função CByte converte uma variável ou resultado de uma expressão, para o subtipo Byte. O valor a ser convertido, deve estar na faixa aceitável para o tipo byte, que vai de 0 à 255. Caso o número esteja fora desta faixa, será gerada uma mensagem de erro, em tempo de execução. O argumento desta função, não pode ser Texto, caso contrário será gerada uma mensagem de erro, em tempo de execução.

Observe esta linha de código:

```
MsgBox CByte(10+34) & chr(13) & CByte(35)
```

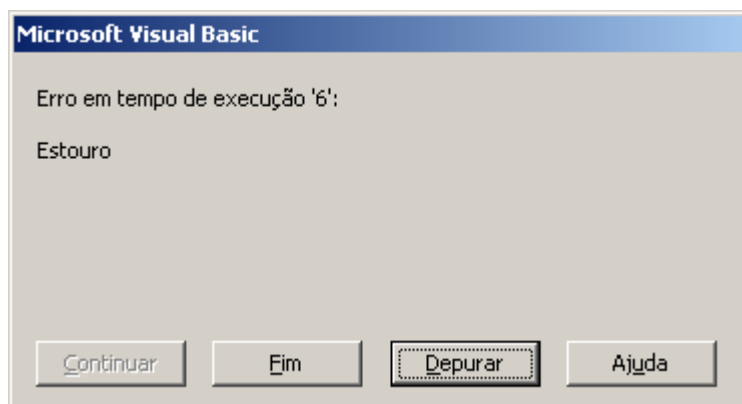
Esta linha gera a mensagem indicada na Figura a seguir:



A expressão 10+34 é calculada, e o resultado (44), é convertido para o tipo byte. A linha de código abaixo, irá gerar uma mensagem de erro, em tempo de execução, pois o valor a ser convertido para byte, está fora da faixa de 0 à 255.

```
MsgBox CByte(100+200)
```

Essa linha de código irá gerar a seguinte mensagem de erro:



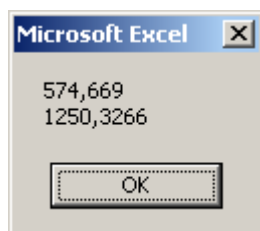
Função CCur

A função CCur converte uma variável ou resultado de uma expressão, para o subtipo Currency (semelhante ao formato Moeda, porém sem o símbolo do real: R\$). O argumento desta função, não pode ser Texto, caso contrário será gerado um erro.

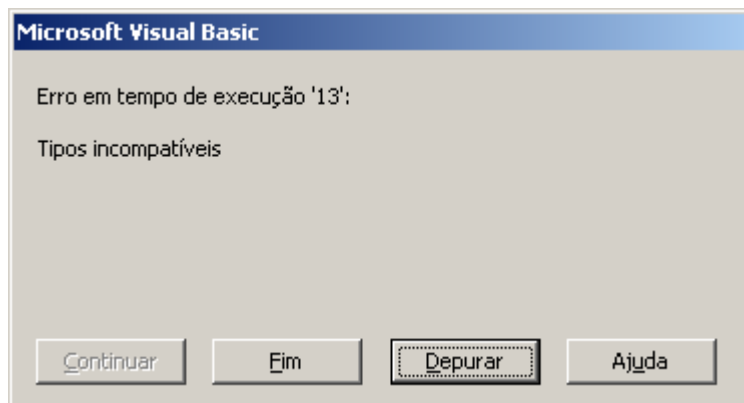
Observe esta linha de código:

```
MsgBox CCur(250.335677+324.3333) & chr(13) & CCur(1250.326582)
```

Esta linha gera a mensagem indicada na Figura a seguir:



A expressão $250.335677 + 324.3333$ é calculada, e o resultado é convertido para o tipo Currency. Se passarmos um argumento de texto para a função CCur, será gerado um erro de execução, conforme indicado na Figura a seguir:



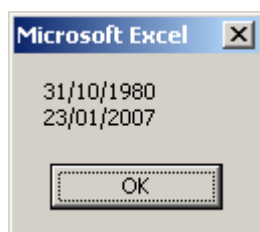
Função CDate

A função CDate converte uma variável ou resultado de uma expressão, para o subtipo Date. O argumento desta função deve estar em um formato que seja aceitável para datas, caso contrário será gerada uma mensagem de erro, em tempo de execução.

Observe esta linha de código:

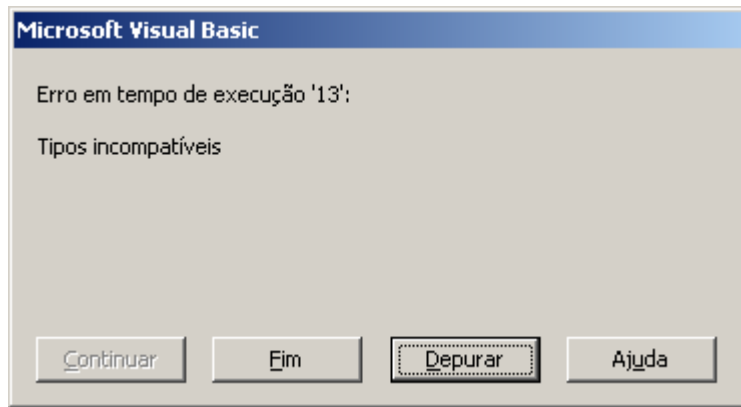
```
MsgBox CDate("31/10/1980") & chr(13) & CDate("23-01-2007")
```

Esta linha gera a mensagem indicada na Figura a seguir:



A linha de código abaixo, irá gerar uma mensagem de erro, em tempo de execução, conforme indicado na próxima figura:

```
MsgBox CDate("31/02/1980")
```



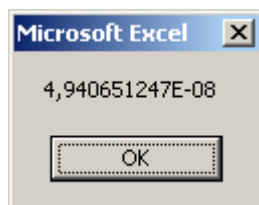
Função CDbI

A função CDbI converte uma variável ou resultado de uma expressão, para o subtipo Double. O tipo Double é utilizado para números grandes com casas decimais. O argumento desta função não pode ser do tipo texto, caso contrário será gerada uma mensagem de erro, em tempo de execução.

Observe esta linha de código:

```
MsgBox CDbI("4.940651247E-17")
```

Esta linha gera a mensagem indicada na Figura a seguir:



Na próxima lição estudaremos mais algumas funções para conversão de tipos.

Lição 16: VBA - Funções do VBA – Funções de Conversão de Tipo – Parte 2

Vamos apresentar mais algumas funções do VBA, para a conversão de tipos de dados.

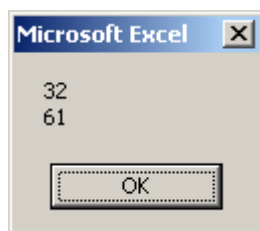
Função CInt

A função CInt converte uma variável ou resultado de uma expressão/fórmula, para o subtipo Integer. O argumento desta função não pode ser do tipo texto, caso contrário será gerada uma mensagem de erro, em tempo de execução.

Observe este linha de código:

```
MsgBox CInt(32.36) & Chr(13) & CInt(20.35+40.45)
```

Esta linha gera a mensagem indicada na Figura a seguir:



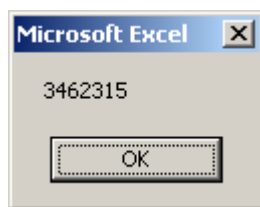
Função CLng

A função CLng converte uma variável ou resultado de um expressão, para o subtipo Long. O argumento desta função não pode ser do tipo texto, caso contrário será gerada uma mensagem de erro, em tempo de execução. O argumento não pode estar fora da faixa admitida pelo subtipo Long, caso contrário será gerada uma mensagem de erro, em tempo de execução.

Observe este linha de código:

```
MsgBox CLng("3462315")
```

Esta linha gera a mensagem indicada na Figura a seguir:



Função CSng

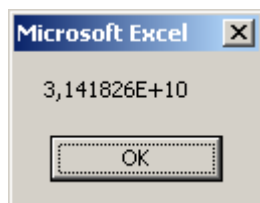
A função CSng converte uma variável ou resultado de um expressão, para o subtipo Single. O argumento desta função não pode ser do tipo texto, caso contrário será gerada uma mensagem de erro, em tempo de execução. O argumento também não pode estar fora da faixa admitida

pelo subtipo Single, caso contrário será gerada uma mensagem de erro, em tempo de execução.

Observe este linha de código:

```
MsgBox CSng("3.1418256927")
```

Esta linha gera a mensagem indicada na Figura a seguir:



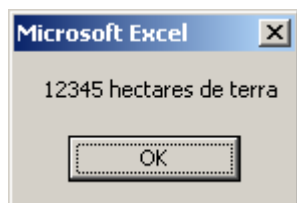
Função CStr

A função CStr converte uma variável ou resultado de um expressão, para o subtipo String.

Observe este linha de código:

```
MsgBox CStr("12345" & " hectares de terra")
```

Esta linha gera a mensagem indicada na Figura a seguir:

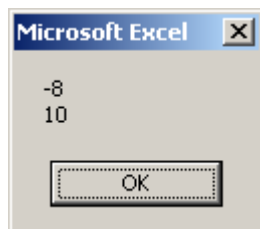


Função Int

A função Int retorna somente a parte inteira de um determinado número. A linha de código a seguir:

```
MsgBox Int(-7.35) & Chr(13) & Int(10.35)
```

Esta linha gera a mensagem indicada na Figura a Seguir:



Com isso, terminamos a nossa apresentação sobre as principais funções para a conversão de tipos de dados. Na próxima lição, analisaremos mais algumas funções do VBA, para operações com dados do tipo String e do tipo Data/Hora.

Lição 17: VBA - Funções do VBA – Funções Para Tratamento de Texto

Nessa lição veremos as principais funções para tratamento de Texto, via programação VBA no Excel.

Função Asc

A função Asc, retorna o valor numérico do código ASCII, para a primeira letra de uma String. Considere o exemplo:

Asc("Ainda chovia")

Este exemplo de uso da função retorna o valor 65, o qual é o código ASCII, para a letra **A** maiúscula. Caso fosse a letra "a" minúscula, o código retornado seria 97 e assim por diante.

Função Chr

A função Chr(número), recebe um número como parâmetro, e retorna o caracter ASCII, associado ao número passado como parâmetro.

Considere o exemplo:

Chr(65)

Este exemplo de uso da função retorna o caractere "A" maiúsculo.

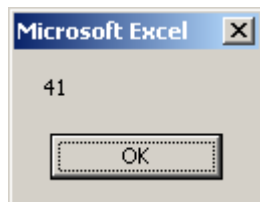
NOTA: Até agora, utilizamos a função Chr em diversos exemplos. Utilizamos o Chr(13), para simular um ENTER, o que faz uma quebra de linha, nas mensagens montadas com o MsgBox.

Função Len

Esta função determina o tamanho da String que foi passada como parâmetro para a função. Considere o exemplo:

MsgBox Len("Este é um exemplo de uso da função Len !!")

Este exemplo de uso da função, retorna 41, conforme indicado na Figura a seguir:



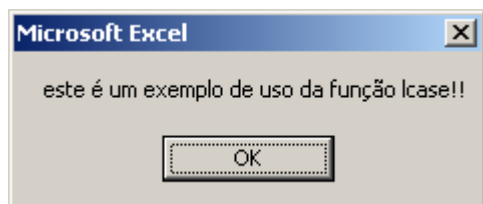
É importante observar que os espaços em branco também “contam” para o tamanho da string.

Função LCase

Esta função converte para minúsculas, a String que foi passada como parâmetro para a função. Considere o exemplo:

MsgBox Lcase("ESTE É UM EXEMPLO DE USO DA FUNÇÃO LCASE!!")

Este exemplo de uso da função, converte o parâmetro passado, para letras minúsculas, conforme indicado na Figura a seguir:



Função UCase

Esta função converte para MAIÚSCULAS, a String que foi passada como parâmetro para a função. Considere o exemplo:

MsgBox Lcase("este é um exemplo do uso da função ucase !!")

Este exemplo de uso da função, converte o parâmetro passado, para letras maiúsculas.

Função Left

Esta função retorna um número especificado de caracteres, a partir do início (Left – Esquerda) de uma String. É importante salientar que espaços em branco também devem ser considerados.

O formato geral da função é o seguinte:

Left(String, n)

onde n é o número de caracteres a retornar.

Considere o exemplo:

MsgBox Left("Júlio Cesar Fabris Battisti",5)

Este exemplo de uso da função, retorna as cinco primeiras letras da String passada, neste caso, retornará Júlio.

Vamos alterar um pouco o nosso exemplo:

MsgBox Left("Júlio Cesar Fabris Battisti",7)

Irá retornar: Júlio C

Observe que o espaço em branco também é considerado.

Função Right

Esta função retorna um número especificado de caracteres, a partir do final (Right – Direita) de uma String. É importante salientar que espaços em branco também devem ser considerados.

O formato geral da função é o seguinte:

Right(String, n)

onde n é o número de caracteres a retornar.

Considere o exemplo:

MsgBox Right("Júlio Cesar Fabris Battisti",6)

Este exemplo de uso da função, retorna as seis ultimas letras da String passada, neste caso, retornará "ttisti".

Vamos alterar um pouco o nosso exemplo:

MsgBox Right("Júlio Cesar Fabris Battisti",10)

Irá retornar: s Battisti

Observe que o espaço em branco também é considerado.

Função Mid

Esta função retorna um número especificado de caracteres, a partir de uma posição especificada, dentro da String. É importante salientar que espaços em branco também devem ser considerados.

O formato geral da função é o seguinte:

Mid(String, posicao_inicio, n)

onde:

posicao_inicio: é a posição a partir da qual devem ser retornados caracteres

n: é o número de caracteres a retornar.

Considere alguns exemplos:

Mid("Júlio Cesar Fabris Battisti",7,5)

Este exemplo de uso da função, retorna, a partir da posição 7, 5 caracteres, neste caso, retornará Cesar.

Mid("SANTA MARIA",3,7)

, irá retornar NTA MAR. Observe que o espaço em branco também é considerado.

Um detalhe interessante, é que podemos utilizar o valor retornado por uma função, como parâmetro para outra função. Considere o seguinte exemplo:

LCase(Mid("SANTA MARIA",3,7))

Este exemplo retorna **nta mar**.

A função Mid retira os caracteres NTA MAR, os quais são passados como parâmetros para a função LCase, a qual converte os caracteres para minúsculos.

Função String

Esta função retorna um determinado caractere, um número especificado de vezes.

O formato geral da função é o seguinte:

String(n, Caractere)

onde n é o número de vezes que Caractere deve ser repetido.

Considere o exemplo:

MsgBox String(35,"*")

A Figura a seguir mostra o resultado deste comando:



Lição 18: VBA - Funções do VBA – Data, Hora e Funções Matemáticas

Nesta lição veremos uma série de funções, relacionadas ao tratamento de valores de data e hora e a realização de cálculos matemáticos.

Funções para tratamento de Data e Hora

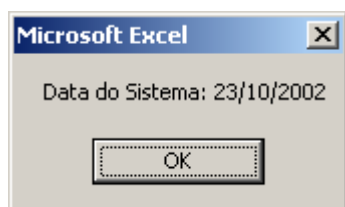
Veremos as principais funções para tratamento de Data e Hora.

Função Date

Retorna a data corrente do sistema. Não precisamos passar parâmetros para esta função. Considere o exemplo abaixo:

MsgBox “Data do Sistema: “ & Date()

O resultado deste comando, está indicado na Figura a seguir:

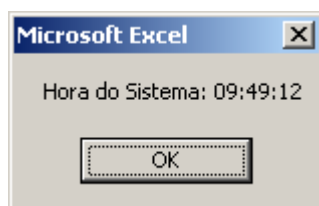


Função Time

Retorna a hora corrente do sistema. Não precisamos passar parâmetros para esta função. Considere o exemplo abaixo:

MsgBox “Hora do Sistema: “ & Time()

O resultado deste comando, está indicado na Figura a seguir:



Função Day

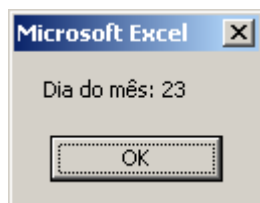
Recebe como parâmetro uma data, e retorna um número entre 1 e 31, indicando o dia do mês. O formato geral é o seguinte:

Day(data)

Considere o exemplo abaixo:

MsgBox “Dia do mês: “ & Day(Date())

A função Date() captura a data do sistema e passa como parâmetro para a função Day, a qual por sua vez, retorna apenas o dia do mês.



Função Month

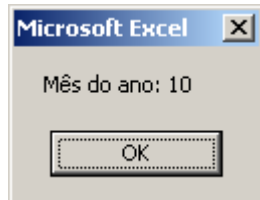
Recebe como parâmetro uma data, e retorna um número entre 1 e 12, indicando o mês do ano. O formato geral é o seguinte:

Month(data)

Considere o exemplo abaixo:

MsgBox “Mês do ano: “ & Month(Date())

O resultado deste comando, está indicado na Figura a seguir:



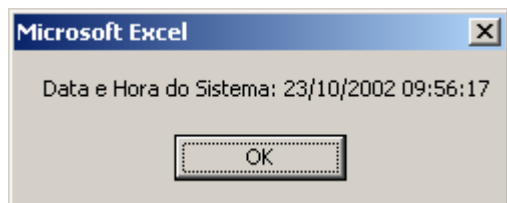
A função Date() captura a data do sistema e passa como parâmetro para a função Month, a qual por sua vez, retorna apenas o mês do ano

Função Now

Retorna a hora e a data corrente do sistema. Não precisamos passar parâmetros para esta função. Considere o exemplo abaixo:

MsgBox “Data e Hora do Sistema: “ & Now()

O resultado deste comando, está indicado na Figura a seguir:



Função MonthName

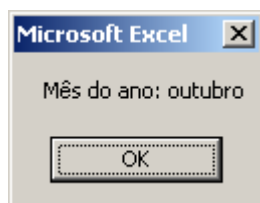
Recebe como parâmetro um número, indicativo do mês do ano (1 – Janeiro, 2 – Fevereiro, e assim por diante), e um segundo parâmetro que pode ser Verdadeiro ou Falso. Se o segundo parâmetro for verdadeiro, o nome do mês será exibido abreviadamente. O formato geral é o seguinte:

MonthName(número_do_mês, abreviar)

Considere o exemplo abaixo:

MsgBox "Mês do ano: " & MonthName(Month(Date))

O resultado deste comando, está indicado na Figura a seguir:



A função Date() captura a data do sistema e passa como parâmetro para a função Month. A função Month retorna o número do mês retornado pela data. Este número é passado como primeiro parâmetro para a função MonthName.

Função Hour

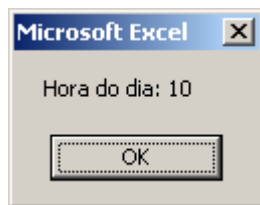
Recebe como parâmetro uma hora, e retorna um número entre 0 e 23, indicando o hora do dia. O formato geral é o seguinte:

Hour(horário)

Considere o exemplo abaixo:

MsgBox "Hora do dia: " & Hour(Time())

O resultado deste comando, está indicado na Figura a seguir:



A função Time() captura a hora do sistema e passa como parâmetro para a função Hour(), a qual por sua vez, retorna apenas o hora do dia.

A função DateDiff

Esta função pode ser utilizada para determinar o número de intervalos (em dias, trimestres, semestres, anos, etc), entre duas datas. A sintaxe desta função é o seguinte:

DateDiff(intervalo, data1, data2)

O parâmetro intervalo é uma String que diz que tipo de intervalo vamos calcular. Por exemplo, é este parâmetro que define se queremos calcular o número de dias, ou o número de meses entre duas datas. Na Tabela a seguir, temos os valores possíveis para o parâmetro intervalo.

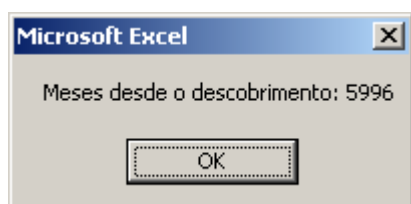
Valores para o parâmetro intervalo.

Valor	Descrição
yyyy	Anos
q	Trimestres
m	Meses
y	Dias do ano (o mesmo que dias)
d	Dias
w	Semanas
ww	Semanas do ano (o mesmo que semanas)
h	Horas
n	Minutos
s	Segundos

A título de exemplo, vamos calcular o número de meses, desde o descobrimento do Brasil, até 31 de Dezembro de 1999. Para isso, utilizaríamos o seguinte comando

MsgBox “Meses desde o descobrimento: “ & DateDiff(“m”,”22/04/1500”,”31/12/1999”)

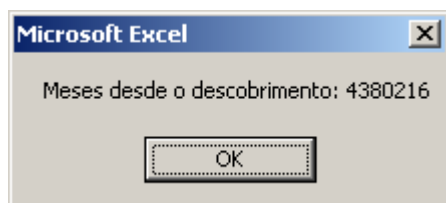
O resultado deste comando, pode ser conferido na Figura a seguir:



Vamos modificar um pouco o nosso exemplo, e calcular o número de horas desde o descobrimento. Para isto, podemos utilizar o seguinte comando:

MsgBox “Meses desde o descobrimento: “ & DateDiff(“h”,”22/04/1500”,”31/12/1999”)

O resultado deste comando, pode ser conferido na Figura a seguir:



A função DateAdd

Esta função pode ser utilizada para determinar uma data futura, com base em uma data fornecida, o tipo de período a ser acrescentado (dias, meses, anos, etc), e o número de períodos a serem acrescentados. A sintaxe desta função é o seguinte:

DateAdd(intervalo, número_de_intervalos, data)

O parâmetro intervalo é uma String que diz que tipo de intervalo vamos acrescentar. Por exemplo, é este parâmetro que define se queremos acrescentar um número especificado de dias, meses, anos, etc. Na Tabela a seguir, temos os valores possíveis para o parâmetro intervalo.

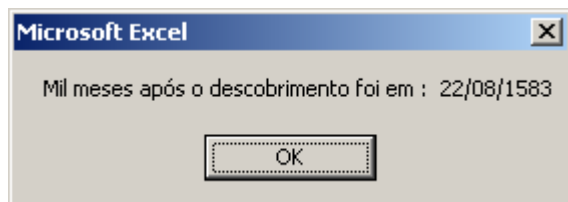
Valores para o parâmetro intervalo.

Valor	Descrição
yyyy	Anos
q	Trimestres
m	Meses
y	Dias do ano (o mesmo que dias)
d	Dias
w	Semanas
ww	Semanas do ano (o mesmo que semanas)
h	Horas
n	Minutos
s	Segundos

A título de exemplo, vamos calcular a data em que tivemos um período de 1000 meses, após o descobrimento do Brasil. Para isso, utilizaríamos o seguinte comando:

MsgBox “Mil meses após o descobrimento foi em : “ & DateAdd(“m”,1000,”22/04/1500”)

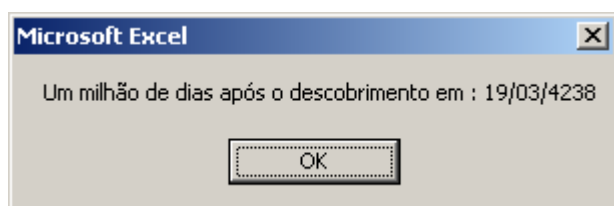
O resultado deste comando, pode ser conferido na Figura a seguir:



Vamos modificar um pouco o nosso exemplo, e calcular em que data teremos passado um milhão de dias após o descobrimento.

MsgBox “Um milhão de dias após o descobrimento em : “ & DateAdd(“d”,1000000,”22/04/1500”)

O resultado deste comando, pode ser conferido na Figura a seguir:



Função Year

Recebe como parâmetro uma data, e retorna um número indicativo do ano. O formato geral é o seguinte:

Year(data)

Considere o exemplo abaixo:

MsgBox “Ano atual: “ & Year(Date())

O resultado deste comando, está indicado na Figura a seguir:



A função Date() captura a data do sistema e passa como parâmetro para a função Year, a qual por sua vez, retorna apenas o ano.

Função WeekDay

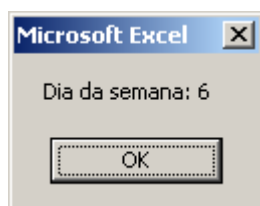
Recebe como parâmetros uma data, e um parâmetro opcional, que indica qual o primeiro dia da semana. Se este parâmetro for omitido, o primeiro dia da semana será considerado Domingo. O valor para o primeiro dia da semana é numérico: 1 – Domingo, 2 – Segunda-feira, e assim por diante.

WeekDay(data, prim_dia_semana.)

Considere o exemplo abaixo:

MsgBox “Dia da semana: “ & WeekDay(“31/12/1999”)

O resultado deste comando, está indicado na Figura a seguir:

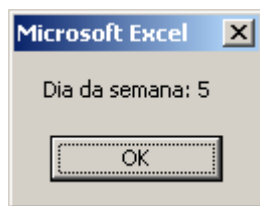


O valor 6, indica que foi uma Sexta-feira, o que confere com o calendário.

Poderíamos determinar que o primeiro dia da semana é a Segunda-feira (2 para o último parâmetro). Com isso o nosso exemplo, ficaria assim:

MsgBox “Dia da semana: “ & WeekDay(“31/12/1999”,2)

O resultado deste comando, está indicado na Figura a seguir:



O valor 5, indica que foi uma Sexta-feira, pois agora a Segunda-feira passou a ser o dia 1, a Terça-feira o dia 2, e assim por diante. Novamente confere com o calendário.

Função WeekDayName

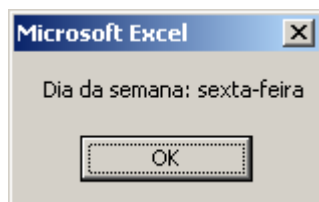
Recebe como parâmetro um número, indicativo do dia da semana, e um segundo parâmetro que pode ser Verdadeiro ou Falso. Se o segundo parâmetro for verdadeiro, o nome do dia da semana será exibido abreviadamente. O formato geral é o seguinte:

WeekDayName(número_do_dia, abreviar)

Considere o exemplo abaixo:

MsgBox “Dia da semana: “ & WeekDayName(6,False)

O resultado deste comando, está indicado na Figura a seguir:



Funções para Cálculos matemáticos

Veremos as principais funções para efetuar cálculos matemáticos.

Função	Descrição
Abs(n)	Retorna o valor absoluto (sem sinal), do número n.
Atn(n)	Retorna o valor do arco, cuja tangente é o número n. O número n deve ser fornecido em radianos.
Cos(n)	Retorna o cosseno do número n. O número n deve ser fornecido em radianos.
Exp(n)	Retorna o número e (logaritmo neperiano e=2,7183), elevado no número n.
Log(n)	Retorna o logaritmo natural de um número n.
Rnd(n)	Retorna um número aleatório entre 0 e 1.
Sgn(n)	Retorna um número inteiro, indicando o sinal do número n. Retorna -1 para números negativos e 1 para números positivos.
Sin(n)	Retorna o seno do número n. O número n deve ser fornecido em radianos
Sqr(n)	Retorna a Raiz quadrada do número n.
Tan(n)	Retorna a tangente do número n. O número n deve ser fornecido em radianos.

NOTA: Para converter graus para radianos, multiplique o valor em graus por pi (3.14), e divida o resultado por 180.

Na Tabela a seguir temos alguns exemplos de utilização das funções matemáticas do VBA.

Alguns exemplos de utilização das funções matemáticas

Exemplo	Valor de retorno.
Abs(-2350)	2350
Atn(2)	1,1071487177
Cos(0)	1
Exp(1)	2,71828182845905
Log(1000)	6,90775527898214
Rnd*10	Gera um número aleatório entre 0 e 10
Sgn(-235)	Retorna -1, pois o número é negativo
Sin(0)	Retorna 0
Sqr(400)	Retorna 20
Tan(0)	Retorna 0

Lição 19: VBA - O Conceito de Módulos, Procedimentos e Funções – Parte I

Nas lições anteriores, aprendemos a utilizar as funções prontas, que já fazem parte do Excel. Porém podemos criar nossas próprias funções. Neste item, aprenderemos a criar nossas próprias funções e Sub-rotinas .

O princípio por trás da criação de funções e sub-rotinas, é o de reaproveitamento de código e facilidade de manutenção do programa. Imagine que estejamos criando uma planilha para cálculos financeiros, por exemplo, depreciação contábil. Vamos supor que em diversos locais, seja necessária a realização de um determinado cálculo de depreciação. Poderíamos colocar o código/fórmulas que faz o cálculo, em cada um dos locais, onde o cálculo seja necessário. Porém esta não é a melhor maneira de criarmos nossos programas. Imagine, por exemplo, quando fosse necessário alterar a maneira de realizar os cálculos. Teríamos que repassar todos os locais onde o cálculo é feito, procurando os pontos onde o código está , e fazer as alterações.

Para resolver estes problemas, poderíamos criar uma função ou sub-rotina que faz os cálculos de depreciação. A função/sub-rotina seria criada dentro de um módulo de código do VBA, na própria planilha. Depois, em cada local onde precisamos fazer os cálculos, é só chamar a função (ou sub-rotina), para fazer os cálculos. Quando fosse necessária a alteração da metodologia de cálculo, era só alterar a função (ou sub-rotina) e pronto, os novos cálculos passarão a ser feitos com base na nova metodologia. Isto **poupa esforço, reduz o número de linhas de código, e facilita a manutenção, além de reduzir a possibilidade de erros.**

Agora é chegada a hora de aprendermos a criar funções e sub-rotinas. Estes procedimentos/funções são criados em módulos de código VBA associados a planilha. Quando uma destas funções/procedimentos for necessária, basta chamar a respectiva função/procedimento que o Microsoft Excel se encarrega de localizar a função/procedimento, passar os parâmetros necessários (se for o caso) e receber os resultados retornados, caso seja uma função.

Vamos a outro exemplo. O Excel tem centenas de funções internas, para realiar os mais variados tipos de cálculos. Mas evidentemente que mesmo com um grande número de funções, não estão disponíveis funções para todos os cálculos que você necessitará na prática. Por exemplo, não existe uma função simples para cálculo de Imposto de Renda com base na faixa de salário. Usando programação VBA, você pode criar uma função para cálculo do IRPF e utilizá-la em suas planilhas.

NOTA: Vamos falar um pouco mais sobre o termo Procedimento. Um procedimento é um grupo de instruções que pode ser chamado pelo nome atribuído ao procedimento. Neste contexto, funções e sub-rotinas, são tipos diferentes de procedimentos. Nesta e nas próximas lições você aprenderá mais sobre funções e sub-rotinas e verá alguns exemplos práticos.

Criando e utilizando Sub-rotinas

Uma sub-rotina é um grupo de comandos que podem ser executados, simplesmente através da chamada do nome da Sub-rotina. Podemos passar um ou mais argumentos para uma Sub-rotina. Quando uma sub-rotina é chamada, a execução desloca-se para dentro da Sub-rotina, depois de executados todos os comandos dentro da Sub-rotina, a execução do código continua, com o comando seguinte ao que chamou a Sub-rotina. Os argumentos são valores que são passados para a sub-rotina, os quais são utilizados internamente para a realização de cálculos. Por exemplo, se você desenvolve uma sub-rotina para importação de dados de um arquivo de texto, você poderia definir dois argumentos para esta sub-rotina: um informa a pasta e o nome do arquivo a ser importado e o outro o nome da planilha para a qual os dados serão importados.

A sintaxe para a criação de uma Sub-rotina é a seguinte:

Sub Nome_da_Sub-rotina(argumento1, argumento2, ..., argumenton)

```
Comando1
Comando2
...
Comandon
End Sub
```

Uma Sub-rotina pode, ou não, conter argumentos. Caso sejam necessários argumentos, estes serão passados quando a Sub-rotina for chamada, e devem ser passados, na mesma ordem em que foram definidos.

DICA: Quando você criar Sub-rotinas, procure utilizar nomes que descrevam a função da Sub-rotina. Com isso você torna o entendimento do código mais fácil para quem for utilizá-lo.

Considere o seguinte exemplo de declaração de uma Sub-rotina:

```
Sub Calcula_imposto(salario, desconto, extras)

Comando1
Comando2
...
Comandon
End Sub
```

Neste caso, declaramos uma rotina chamada Calcula_imposto, a qual espera receber 3 parâmetros: salario, desconto e extras. Os parâmetros devem ser fornecidos nesta ordem, para que a Sub-rotina funcione corretamente.

Uma vez criada a Sub-rotina, podemos chamá-la, simplesmente digitando o nome da Sub-rotina, ou utilizando a palavra **Call**, mais o nome da Sub-rotina. Nos exemplos abaixo, temos dois métodos de chamada da Sub-rotina Calcula_imposto:

Calcula_imposto(1500,23,125)

ou

Call Calcula_imposto(1500,23,125)

Observe que os parâmetros são passados dentro do parênteses.

Também poderíamos declarar uma Sub-rotina, sem parâmetros. Neste caso posso simplesmente não utilizar os parênteses após o nome da Sub-rotina, ou utilizar um par de parênteses, sem nada dentro, conforme indicado abaixo:

Sub Nome_da_Sub-rotina

Comando1

Comando2

...

Comandon

End Sub

ou

Sub Nome_da_Sub-rotina()

Comando1

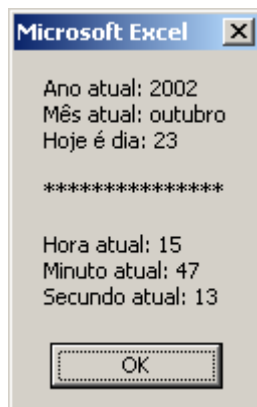
Comando2

...

Comandon

End Sub

Por exemplo, poderíamos criar uma Sub-rotina que exibe uma Caixa de mensagem com a Data do sistema e a hora, no Formato indicado na Figura a seguir:



Poderíamos criar todos os comandos necessários, para exibir a hora e a data neste formato, cada vez que a exibição fosse necessária. Porém é mais prático criar uma Sub-rotina que faz este trabalho. Cada vez que precisarmos exibir a data e a hora neste formato, é só chamarmos a Sub-rotina.

Na Listagem a seguir temos o código da Sub-rotina **exibe_data_hora()**, a qual faz a exibição da data e da hora, no formato proposto pela Figura anterior.

Listagem– A Sub-rotina **exibe_data_hora**

```
Sub exibe_data_hora

mes_atual=Month(Date())
mes_nome=MonthName(mes_atual,False)

mensagem = "Ano atual: " & Year(Date()) & Chr(13)
mensagem = mensagem & "Mês atual: " & mes_nome & Chr(13)
mensagem = mensagem & "Hoje é dia: " & Day(Date())& Chr(13)
mensagem = mensagem & Chr(13) & "*****" & Chr(13)& Chr(13)

hora_atual = Hour(Time())
minuto_atual = Minute(Time())
segundo_atual = Second(Time())

mensagem = mensagem & "Hora atual: " & hora_atual & chr(13)
mensagem = mensagem & "Minuto atual: " & minuto_atual & chr(13)
mensagem = mensagem & "Segundo atual: " & segundo_atual

MsgBox mensagem

End Sub
```

Agora, cada vez que for necessário exibir a data e a hora nos formatos do exemplo, basta chamar a Sub-rotina **exibe_data_hora**, utilizando um dos seguintes comandos:

Call exibe_data_hora

ou

exibe_data_hora

Criando e utilizando Funções

Uma Função é um grupo de comandos que podem ser executados, simplesmente através da chamada do nome da Função. Podemos passar um ou mais argumentos para uma Função. Quando uma Função é chamada, a execução desloca-se para dentro da Função, depois de executados todos os comandos dentro da Função, a execução do código continua, com o comando seguinte ao que chamou a Função. **A diferença da Função para a Sub-rotina, é que a Função sempre retorna um ou mais valores para o comando que a chamou.**

A sintaxe para declaração de uma Função é a seguinte:

Function Nome_da_Função(argumento1, argumento2, ..., argumenton) As Tipo

```
    Comando1
    Comando2
    ...
    Comandon
End Function
```

Uma Função pode, ou não, conter argumentos. Caso sejam necessários argumentos, estes serão passados quando a Função for chamada, e devem ser passados, na mesma ordem em que foram definidos.

DICA: Quando você criar Função, procure utilizar nomes que descrevam os objetivos da Função. Com isso você torna o entendimento do código mais fácil para quem for utilizá-lo.

Considere o seguinte exemplo de declaração de uma Função:

Função Converte_para_dolar(valor_em_real, cotacao_dolar)

```
    Comando1
    Comando2
    ...
    Comandon
End Function
```

Neste caso, declaramos uma Função chamada Converte_para_dolar, a qual espera receber 2 parâmetros: um valor em real e a cotação do dólar.

Uma vez criada a Função, devemos chamá-la, em um comando de atribuição, isto é, o valor retornado pela função, deve ser atribuído a uma variável, ou exibido através de um comando como MsgBox. No exemplo abaixo, estamos atribuindo o valor de retorno da função, à variável valor_dolar.

```
valor_dolar = Converte_para_dolar (1500,1.81)
```

Também poderíamos exibir o valor retornado pela função, utilizando o comando MsgBox, conforme indicado a seguir:

```
MsgBox "Valor em dólar: " & Converte_para_dolar (1500,1.81)
```

Observe que os parâmetros são passados dentro dos parênteses, e na mesma ordem definida quando da criação da função.

Também poderíamos declarar uma Função, sem parâmetros. Neste caso posso simplesmente não utilizar os parênteses após o nome da Função, ou utilizar um par de parênteses, sem nada dentro, conforme indicado abaixo:

```
Function Nome_da_Função
```

```
    Comando1
```

```
    Comando2
```

```
    ...
```

```
    Comandon
```

```
End Function
```

ou

```
Function Nome_da_Função( )
```

```
    Comando1
```

```
    Comando2
```

```
    ...
```

```
    Comandon
```

```
End Function
```

Por exemplo, vamos criar uma função que converte um valor de um ângulo de Graus para Radianos. Depois utilizaremos a função dentro de um laço For...Next, para exibir o valor em radianos, para os ângulos de 0 à 20 graus.

No exemplo a seguir temos o código onde foi criada a função CRad, que converte um valor em graus para radianos. Depois utilizamos um laço for para exibir, através de uma Caixa de mensagem os valores em radianos, para os ângulos de 0 à 20 graus.

```
'Criação da função CRad.
```

```
Function CRad(valor_graus)
```

```
    CRad = (valor_graus*3.14)/180
```

```
End Function
```

```
'Agora utilizamos a função dentro do laço For/Next.
```

```
For i=0 to 20
```

```
    mensagem = mensagem & "Angulo: " & i & "Valor em Radianos: "
```

```
    mensagem = mensagem & FormatNumber(CRad(i),5)& Chr(13)
```

```
Next
```

```
MsgBox mensagem
```

IMPORTANTE: Observe que dentro da função, atribuímos a variável CRad um determinado valor. É isso que caracteriza uma função. Dentro do código da função, devemos atribuir a uma variável que tenha o mesmo nome da função, um determinado valor. Este valor é que será o valor de retorno da função.

Cabe salientar o uso da função `FormatNumber`, dentro do laço `For`. A função `FormatNumber` é utilizada para formatar a maneira como um número é exibido. Neste caso, utilizamos a função `FormatNumber`, para limitar o número de casas decimais, a 5 casas depois da vírgula.

Em cada "passada" do laço `For...Next`, chamamos a função `CRad(i)`, para fazer a conversão de Graus para radianos. O Valor retornado pela função `CRad`, é passado para a função `FormatNumber`, para ser formatado com apenas 5 casas decimais.

Lição 20: VBA - O Conceito de Módulos, Procedimentos e Funções – Parte II

Nesta lição trataremos de um assunto de grande importância: Escopo de variáveis. Quando você cria sub-procedimentos e funções personalizadas é muito importante conhecer bem o conceito de Escopo de variáveis. Conforme será visto neste módulo, o escopo define em eu locais do código VBA uma determinada variável é válida e possui um valor associado a ela.

O Escopo das variáveis, no VBA

O escopo de uma variável, define em que partes do código a variável pode ser utilizada. Em VBA, podemos ter os seguintes escopos para as variáveis:

Escopo de Módulo: Uma variável declarada dentro de um Módulo (um módulo pode conter um ou mais sub-procedimento ou função, cada sub-procedimento ou função começa com um Sub ou Function e termina com um End Sub ou End Function, respectivamente. Você pode declarar uma variável na seção de declarações do módulo, isto é, fora de qualquer Procedimento. Com isso a variável pode ser utilizada dentro de todo o bloco de código do Módulo, inclusive dentro dos Procedimentos, caso exista algum. Um módulo de uma planilha pode conter uma ou mais macros (sub-procedimentos ou funções), sendo que cada macro é um procedimento, isto é, inicia com um Sub e termina com um End Sub. Tudo o que estiver entre Sub e End Sub, faz parte da respectiva macro (sub-procedimento ou função). Uma variável declarada ao nível de Módulo, existe enquanto o Módulo estiver sendo executado – dizemos que tem um escopo de módulo. São as variáveis declaradas na seção de Declarações do Módulo. Uma variável declarada ao nível de Módulo, poderá ser utilizada em todos os procedimentos/funções do módulo, isto é, em todas as macros que fazem parte do respectivo módulo.

Escopo de sub-procedimento/função/macro: É uma variável que foi declarada dentro de um sub-procedimento ou função. A variável somente pode ser utilizada, dentro do sub-procedimento ou função onde ela foi declarada. Aqui o conceito de sub-procedimento ou função se confunde com o de macro, ou seja, cada macro é criada como um sub-procedimento ou função separado, dentro do módulo de código da planilha. Se tentarmos utilizar a variável fora do procedimento onde a ela foi declarada, não teremos acesso ao valor da variável. Uma variável declarada ao nível de procedimento, existe enquanto o procedimento estiver sendo executado.

Vamos considerar alguns trechos de código para entendermos melhor este conceito de escopo de uma variável.

Considere o seguinte trecho de código, dentro de um módulo VBA de uma planilha do Excel:

```
'Seção geral do módulo.  
' Variáveis declaradas nessa seção tem o escopo de módulo,  
' isto é, podem ser utilizadas em qualquer procedimento dentro do  
módulo.  
  
Dim x, y, aux1 As Integer
```

```
'Agora vamos criar um procedimento
'E declarar duas variáveis dentro deste procedimento.
'Estas variáveis somente serão visíveis,
'Dentro deste procedimento.

Sub proc1()
    Dim a, b

    x=10
    y=20

    a=5
    b=7
    MsgBox "x= " & x & " y= " & y & " a= " & a & " b= " & b
End Sub

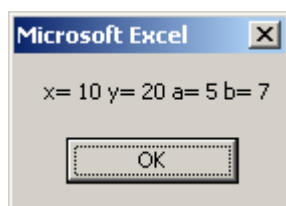
    Call proc1

    ' Agora vamos tentar utilizar os valores de a e b
    ' Fora do procedimento proc1.
    ' Observe que a variável aux1 não é calculada
    ' corretamente, uma vez que os valores de a e b
    ' não estão disponíveis, fora do procedimento proc1

    Aux1 = a + b
    MsgBox "aux1= " & Aux1
```

O comando Call faz o chamado ao procedimento proc1. Este procedimento deve ser definido no mesmo módulo, caso contrário um erro será gerado. Ao encontrar esse comando o Excel desloca a execução para o procedimento proc1, executa os comandos deste procedimento e após encerrar continua a execução a partir do comando seguinte ao comando Call.

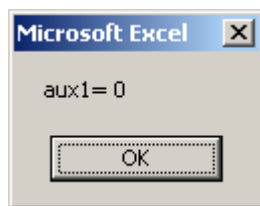
Ao executar esse código será exibida a janela indicada na Figura a seguir:



O comando Call Proc1 é executado. O excel executa os comandos dentro de Proc1. As variáveis x e y foram declarados com escopo de módulo, por isso podem ser utilizadas corretamente dentro do procedimento Proc1, embora não tenham sido declaradas dentro desse procedimento. As variáveis x e y tem escopo de módulo, ou seja, podem ser utilizadas em qualquer procedimento, dentro do módulo. Os valores definidos para x e y, dentro do procedimento Proc1, permanecerão disponíveis, mesmo se usarmos x e y em outros procedimentos desse módulo.

Como a e b estão declaradas dentro de Proc1, o seu escopo está ativo, isso é, o Excel tem acesso aos valores de a e b. Isso ocorre porque a e b foram declarados. Por isso os valores de a e b também são exibidos corretamente. Agora vamos fazer uma tentativa de usar as variáveis a e b fora do procedimento Proc1. Como elas foram declaradas dentro de Proc1, somente são “visíveis”, isto é, mantêm o seu valor quando Proc1 estiver sendo executada. Em outras palavras, o escopo das variáveis a e b, declaradas dentro de Proc1 é o próprio procedimento Proc1. Fora de Proc1, as variáveis a e b terão valores nulos, conforme comprovaremos logo a seguir:

Dê um clique no botão OK. Será exibida a janela indicada na figura a seguir:



Como é possível se $aux1 = a + b$ e $a=5$ e $b=7$, logo $aux1$ deveria ser igual a 12. O que está acontecendo?? Observe que o valor da variável aux1 (foi calculado incorretamente), pois a variável aux1 depende dos valores de "a" e "b". Como as variáveis a e b foram declaradas dentro do procedimento Proc1, elas não podem ser acessadas de fora do procedimento proc1, ou seja, após a execução do procedimento ter sido encerrada.

Dê um clique no botão OK e mantenha a planilha aberta.

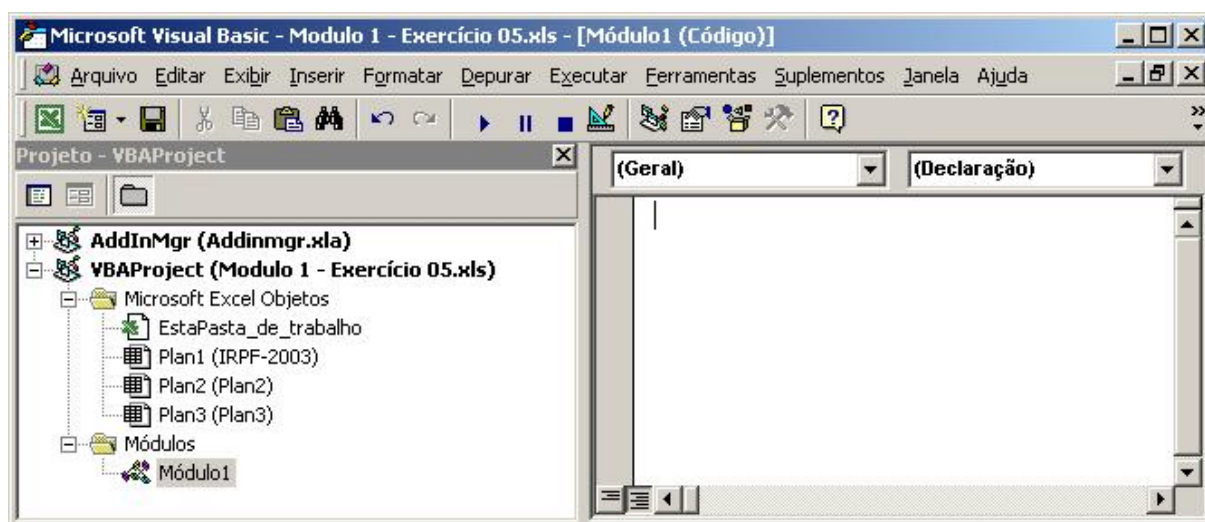
Com esse exemplo, foi possível verificar, na prática, o conceito de escopo (algumas vezes chamado de visibilidade) das variáveis. Na próxima lição apresentarei um exemplo prática de criação de funções e de uso destas funções em uma planilha do Excel.

Lição 21: VBA – Criando Funções Personalizadas – Parte I

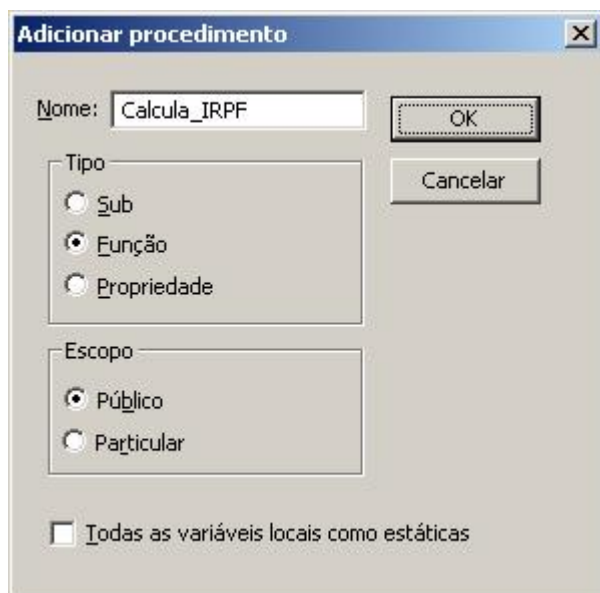
Nesta lição você aprenderá a criar funções personalizadas e a utilizá-las em suas planilhas. Vou mostrar como criar uma função bastante simples. Será uma função para cálculo do imposto de renda Pessoa Física, com base na “Tabela Progressiva Anual Para Cálculo do Imposto”, publicada na página da Receita Federal: www.receita.fazenda.gov.br.

Exemplo: Criando e utilizando uma função personalizada para o cálculo do imposto de renda pessoa física. A função será chamada `Calcula_IRPF` e receberá como parâmetro o valor da Base de Cálculo para o imposto. A função retornará o valor do imposto a ser pago.

1. Abra o Excel.
2. Abra a Planilha `C:\Programação VBA no Excel\Módulo 1 – Exercício 01.xls`.
3. Agora vamos iniciar a criação da função `Calcula_IRPF`.
4. Pressione `Alt+F11` para exibir o Editor do VBA.
5. Para criar uma função que possa ser utilizada em qualquer planilha da sua pasta de trabalho (arquivo .XLS), você deve criar a função em um Módulo de Código separado, não pode ser em um dos módulos de código de uma das planilhas ou da pasta de Trabalho. Clique com o botão direito do mouse em `VBAProject (Modulo 1 – Exercício 05.xls)`, no painel da esquerda. No menu que é exibido, clique em `Inserir -> Módulo`. Será criado um módulo chamado `Módulo 1`, conforme indicado na Figura a seguir:



6. Dê um clique duplo em `Módulo 1`, para selecioná-lo. Agora vamos criar a função `Calcula_IRPF`, dentro do `Módulo 1`. Para criar uma nova função selecione o comando: `Inserir -> Procedimento`. Será exibida a janela `Adicionar procedimento`. Digite o nome da função – `Calcula_IRPF` no campo `Nome` e marque a opção `Função`, no grupo `Tipo`. Sua janela deve estar conforme indicado na figura a seguir:



7. Clique em OK.
8. Será inserido o que chamamos de “esqueleto da função”, ou seja, a declaração da função e o comando End Sub, conforme indicado a seguir:

```
Public Function Calcula_IRPF()  
  
End Function
```

9. Agora vamos alterar este código. O primeiro passo é definir os parâmetros que a função receberá e o tipo que será retornado pela função. Vamos definir um parâmetro chamado BaseDeCálculo, do tipo Currency. O tipo de retorno da função também será definido como sendo do tipo Currency. Altere o código para que fique conforme indicado a seguir:

```
Public Function Calcula_IRPF(BaseDeCálculo As Currency) As Currency  
  
End Function
```

Nota: As variáveis Currency são armazenadas como números de 64 bits (8 bytes) em um formato de número inteiro, em escala de 10.000 para fornecer um número de ponto fixo com 15 dígitos à esquerda da vírgula decimal e 4 dígitos à direita. Essa representação fornece um intervalo de -922.337.203.685.477,5808 até 922.337.203.685.477,5807. O tipo de dados Currency é útil para cálculos que envolvem dinheiro e cálculos de ponto fixo, nos quais a precisão é especialmente importante.

10. Até aqui fizemos a declaração da função, dos seus argumentos e do tipo de retorno da função. A próxima etapa é o cálculo do valor do imposto, com base no valor do parâmetro BaseDeCálculo. Na tabela a seguir, apresento a tabela oficial para cálculo do imposto de renda, para o ano de 2003. Você verá que apenas utilizando os comandos básicos do VBA e alguns testes If...Then, será possível implementar a função Calcula_IRPF.

Base de Cálculo	Alíquota	Parcela a deduzir
Até R\$ 12.696,00	Isento	-
De R\$ 12.696,01 a R\$ 25.380,00	15 %	R\$ 1.904,40
Acima de R\$ 25.380,00	27,5 %	R\$ 5.076,90

11. A seguir coloco o código da função `Calcula_IRPF`. No próprio código coloquei comentários que explicam cada parte da função. Esta função foi desenvolvida com objetivos didáticos. Na prática, os valores a deduzir não devem ser colocados diretamente no código da função e sim serem lidos a partir de células na própria planilha. Pois se colocarmos estes valores no código da função, sempre que a tabela de cálculo do imposto for modificada, teremos que alterar o código da função. Já se os valores forem informados em células da planilha e a função ler estes valores, sempre que a tabela de cálculo do imposto for alterada, bastará alterar estes valores na planilha. Mas, didaticamente, o exemplo como foi implementado, atinge os objetivos propostos.

```
Public Function Calcula_IRPF(BaseDeCálculo As Currency) As Currency

' Teste para ver se a Base de Cálculo está na faixa de isenção,
' ou seja, menor do que: R$ 12696,00

' Se estiver nesta faixa, o valor do imposto a pagar será R$ 0,00
' Observe que para fazer com que a função retorne um valor,
' atribuímos o valor a ser retornado para uma variável com o mesmo
' nome da função.

If BaseDeCálculo <= 12696 Then
    Calcula_IRPF = 0
End If

' Teste para ver se a Base de Cálculo está na faixa acima de R$ 12696,00
' até R$ 25380,00. Se estiver nesta faixa, o valor do imposto a pagar
' será de 15% da Base de Cálculo, deduzindo a parcela de R$ 1904,40.

If (BaseDeCálculo > 12696) And (BaseDeCálculo <= 25380) Then
    Calcula_IRPF = (BaseDeCálculo * 0.15) - 1904.4
End If

' Teste para ver se a Base de Cálculo está acima de R$ 25380. Se estiver
' o valor do imposto a pagar será de 27,5% da Base de Cálculo, deduzindo
' a parcela de R$ 5076,90.

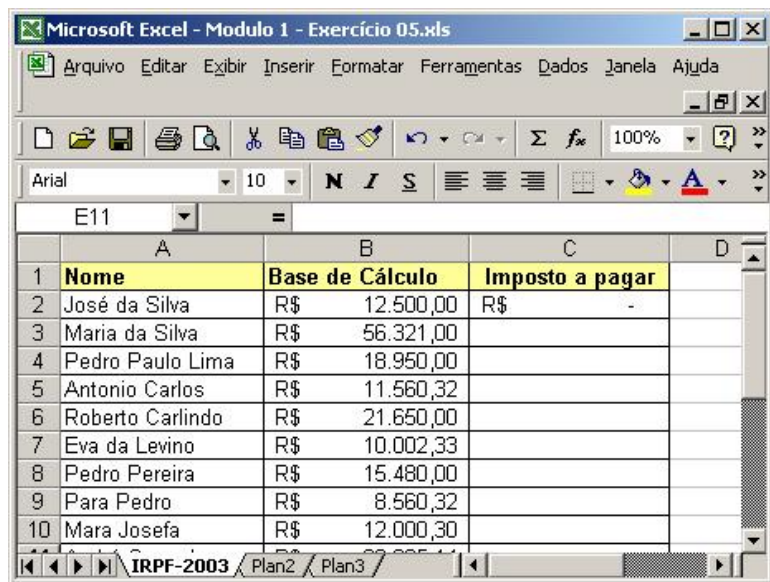
If (BaseDeCálculo > 25380) Then
    Calcula_IRPF = (BaseDeCálculo * 0.275) - 5076.9
End If

End Function
```

12. As linhas que iniciam com um apóstrofe são linhas de comentários, que serão ignoradas pelo VBA. Pressione **Alt+Q**, para fechar o Editor do VBA e voltar ao Excel. Agora vamos testar a função `Calcula_IRPF`. Se for emitido um aviso para salvar o módulo clique em **Sim**.

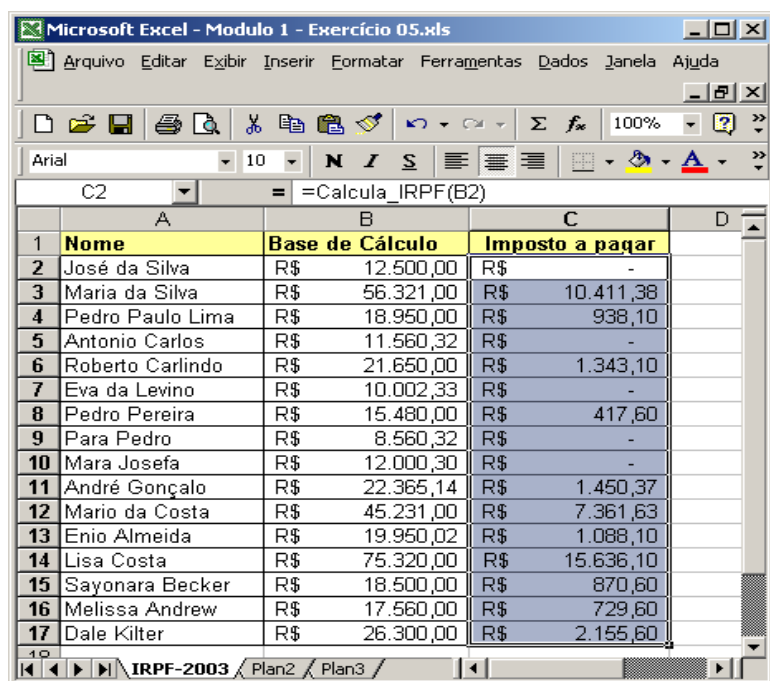
13. Clique na Célula C2 e digite a seguinte fórmula: **=Calcula_IRPF(B2)**

14. Pressione Enter e observe que o Excel usa a função personalizada Calcula_IRPF, para determinar o valor do imposto, conforme indicado na Figura a seguir:



	A	B	C	D
1	Nome	Base de Cálculo	Imposto a pagar	
2	José da Silva	R\$ 12.500,00	R\$ -	
3	Maria da Silva	R\$ 56.321,00		
4	Pedro Paulo Lima	R\$ 18.950,00		
5	Antonio Carlos	R\$ 11.560,32		
6	Roberto Carlindo	R\$ 21.650,00		
7	Eva da Levino	R\$ 10.002,33		
8	Pedro Pereira	R\$ 15.480,00		
9	Para Pedro	R\$ 8.560,32		
10	Mara Josefa	R\$ 12.000,30		

15. O tracinho significa zero. Esta é a formatação padrão aplicada ao formato Contábil (utilizado na célula), ou seja, ao invés de 0, exibe o tracinho. Utilize o mouse para estender a fórmula para as demais células da coluna C. O imposto será calculado, conforme indicado na Figura a seguir:



	A	B	C	D
1	Nome	Base de Cálculo	Imposto a pagar	
2	José da Silva	R\$ 12.500,00	R\$ -	
3	Maria da Silva	R\$ 56.321,00	R\$ 10.411,38	
4	Pedro Paulo Lima	R\$ 18.950,00	R\$ 938,10	
5	Antonio Carlos	R\$ 11.560,32	R\$ -	
6	Roberto Carlindo	R\$ 21.650,00	R\$ 1.343,10	
7	Eva da Levino	R\$ 10.002,33	R\$ -	
8	Pedro Pereira	R\$ 15.480,00	R\$ 417,60	
9	Para Pedro	R\$ 8.560,32	R\$ -	
10	Mara Josefa	R\$ 12.000,30	R\$ -	
11	André Gonçalo	R\$ 22.365,14	R\$ 1.450,37	
12	Mario da Costa	R\$ 45.231,00	R\$ 7.361,63	
13	Enio Almeida	R\$ 19.950,02	R\$ 1.088,10	
14	Lisa Costa	R\$ 75.320,00	R\$ 15.636,10	
15	Sayonara Becker	R\$ 18.500,00	R\$ 870,60	
16	Melissa Andrew	R\$ 17.560,00	R\$ 729,60	
17	Dale Kilter	R\$ 26.300,00	R\$ 2.155,60	

Muito bem, a função Calcula_IRPF foi criada, está funcionando e poderá ser utilizada em qualquer planilha da pasta de trabalho Módulo 1 – Exercício 05.xls.

Lição 22: VBA – Um exemplo prático – calculando o DV do CPF - Algoritmo

Nessa e nas próximas duas lições, veremos o uso do VBA para solucionar mais um exemplo prático. Vamos criar uma função personalizada, chamada ValidaCPF. Em seguida usaremos essa função para fazer o cálculo do DV de um conjunto de CPFs. Observe que para a criação desta função, usaremos apenas os comandos e funções internas básicas do VBA, vistas nas lições anteriores.

Importante: O algoritmo de cálculo do DV de CPFs e CNPJs é de domínio público, já tendo sido publicado no diário oficial da união e em diversas revistas de informática, de circulação nacional.

Nessa lição explicarei como funciona o cálculo do DV do CPF.

Como calcular o DV do CPF

Para entender o algoritmo de cálculo do CPF vamos utilizar um exemplo prático.

Considere o seguinte CPF (sem o DV): **333.444.555**

Posição	1	2	3	4	5	6	7	8	9
Número	3	3	3	4	4	4	5	5	5

Começamos a multiplicar os dígitos do CPF, a partir da posição 9, ou seja, de trás para frente, por 2, 3, 4, 5 e assim por diante, conforme indicado na tabela a seguir:

Posição	1	2	3	4	5	6	7	8	9
Número	3	3	3	4	4	4	5	5	5
Multiplica por:	10	9	8	7	6	5	4	3	2
Resultado	30	27	24	28	24	20	20	15	10

Somo os resultados obtidos na quarta linha da tabela anterior:

$$\begin{aligned}\text{Soma1} &= 30+27+24+28+24+20+20+15+10 \\ \text{Soma1} &= 198\end{aligned}$$

Faço a divisão desta soma por 11 e determino o resto da divisão:

198/11 Resulta em uma divisão exata, com **resto 0**

Regra: Quando o **resto** é **zero** ou **um**, o **DV** é **0**
Quando o **resto** é **diferente** de **zero** ou **um**, o **DV** é
obtido fazendo-se: **11-resto**

Neste caso como o resto foi zero, o primeiro DV é zero:

DV1=0

O DV1 calculado passa a fazer parte do CPF, conforme indicado pela tabela a seguir:

Posição	1	2	3	4	5	6	7	8	9	10
Número	3	3	3	4	4	4	5	5	5	0

Agora repetimos o processo anterior, porém já considerando o DV1 como parte integrante do CPF, conforme indicado pela tabela a seguir:

Posição	1	2	3	4	5	6	7	8	9	10
Número	3	3	3	4	4	4	5	5	5	0
Multiplica por:	11	10	9	8	7	6	5	4	3	2
Resultado	33	30	27	32	28	24	25	20	15	0

Somo os resultados obtidos na quarta linha da tabela anterior:

$$\begin{aligned}\text{Soma2} &= 33+30+27+32+28+24+25+20+15+0 \\ \text{Soma1} &= 234\end{aligned}$$

Faço a divisão desta soma por 11 e determino o resto da divisão:

234/11 Resulta em (21), com **resto 3**

Regra : Quando o **resto** é **zero** ou **um**, o **DV** é **0**.
Quando o **resto** é **diferente** de **zero** ou **um**, o **DV** é
obtido fazendo-se: **11-resto**

Neste caso como o resto foi 3, o segundo DV é :

$$\begin{aligned}\text{DV2} &= 11 - 3 \\ \text{DV2} &= 8\end{aligned}$$

Com isso o CPF, já com os dois DVs fica conforme indicado na tabela a seguir:

Posição	1	2	3	4	5	6	7	8	9	10	11
Número	3	3	3	4	4	4	5	5	5	0	8

Ou seja: **333.444.555-08**

Os algoritmos para cálculo dos DVs do CNPJ são praticamente iguais. A única diferença é a quantidade de dígitos do CNPJ é diferente do CPF e no CNPJ após chegar a multiplicação por 10, inicia novamente a multiplicação por 2, depois por 3 e assim por diante.

Lição 23: Usando o VBA Para Criar uma Função de Validação do DV do CPF

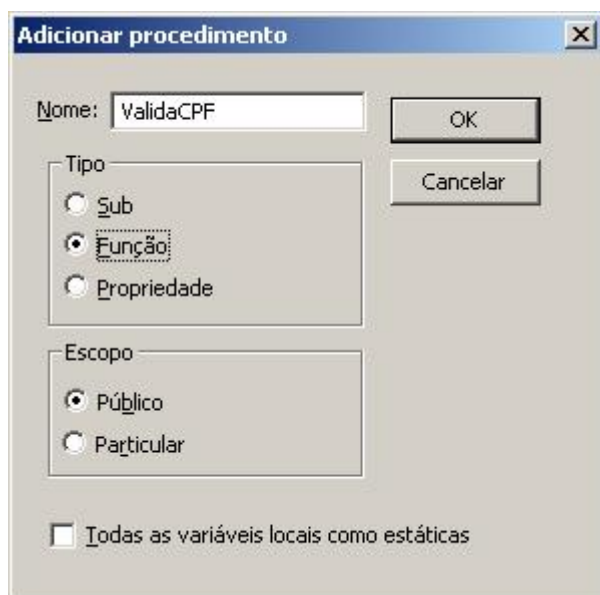
Nessa lição vamos criar uma função chamada ValidaCPF. Essa função recebe, como parâmetro, um valor de CPF no formato 111.111.111-11 ou o endereço de uma célula que contém um CPF nesse formato. A função retorna a palavra Válido se o CPF for Válido e Inválido se o CPF for inválido.

Para criar uma função que possa ser utilizada na planilha, devemos criar a função dentro de um Módulo do VBA, conforme descrito no exemplo de criação da função Calcula_IRPF, na Lição 19, deste módulo.

Criaremos a função ValidaCPF, dentro do Módulo1 da planilha **Números de CPF.xls**, a qual está na pasta C:\Programação VBA no Excel\.

Exercício: Criar uma função chamada ValidaCPF, no Módulo 1 da planilha Números de CPF. Em seguida utilizar essa função para verificar se os CPFs da planilha são ou não válidos.

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\Números de CPF.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Vamos criar a função ValidaCPF de tal maneira que ele possa ser utilizada em qualquer local da planilha. Para isso vamos cria-la como uma função Pública, dentro de um módulo do VBA.
5. Para inserir um módulo, selecione o comando **Inserir -> Módulo**. Será criado o Módulo 1. Agora vamos inserir uma função, dentro desse módulo. Clique em Módulo 1, no painel da esquerda.
6. Selecione o comando **Inserir -> Procedimento**.
7. Será aberta a janela Adicionar procedimento. Preencha os dados conforme indicado na Figura a seguir:



8. Clique em OK.
9. Será inserido o código básico para a criação de uma função pública (que pode ser utilizada em qualquer parte da planilha), conforme indicado a seguir:

```
Public Function ValidaCPF()  
  
End Function
```

10. A função ValidaCPF deverá receber um parâmetro: o número do CPF no formato 111.111.111-11. A definição dos parâmetros que serão recebidos pela função é feito dentro dos parênteses. Fora dos parênteses, definimos o tipo da função, isto é, que tipo de valor a função irá retornar: inteiro, real, texto, data, etc. No nosso exemplo a função retornará um valor do tipo Texto: Válido ou Inválido. Altere o código para definir o parâmetro a ser recebido e o tipo da função, conforme indicado a seguir:

```
Public Function ValidaCPF(CPF As String) As String  
  
End Function
```

Observe que o “**As String**” dentro do parênteses, define que o parâmetro CPF é do tipo texto. Já o “**As String**” fora do parênteses, define o tipo da função, isto é, a função irá retornar um valor do tipo Texto.

11. Agora vamos digitar o código de validação do CPF. Esse código é digitado entre os comandos “Public Function ValidaCPF(CPF As String) As String” e “End Function”.

Nota: Cada comando do VBA deve ser digitado em uma única linha. Se você precisa “quebrar” uma linha, deve ser colocado um caractere de sublinhado no final da linha: _

12. Digite o código da listagem a seguir. As linhas que iniciam com um apóstrofe – ‘ – são comentários e servem para descrever cada parte da função, ou seja, o que faz cada trecho do código da função. Observe que a função implementa exatamente (como não poderia deixar de ser), os passos do algoritmo de validação, descrito na Lição 20:

```
Public Function ValidaCPF(CPF As String) As String  
  
'Função para cálculo do dígito verificador do CPF  
  
'Iniciamos a função com a declaração das variáveis que serão utilizadas.  
  
' As variáveis d1 até d11, conterão os dígitos individuais  
' do CPF. Por exemplo, ao digitar o CPF: 123.456.789-11, essas  
' variáveis conterão os seguintes valores:  
' d1=1 d2=2 d3=3 d4=4 d5=5 d6=5 d7=7 d8=8 d9=9 d10=1 d11=1  
  
Dim d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11 As Integer  
  
' Demais variáveis que serão utilizadas para o cálculo do DV.  
  
Dim Soma1, Soma2, Resto As Integer  
Dim Resto1, Resto2 As Integer  
Dim DV1, DV2 As Integer
```


Macros e Programação VBA no Excel

```
' Em primeiro lugar testo se a célula com o CPF contém um valor
' válido, isto é, um valor Não Nulo.

If Not (IsNull(CPF)) Then

' *****
' Os comandos a seguir desmembram o CPF um a um , atribuindo os valores *
' de d1 ... d11 , usando as funções Mid$ e Val *
' Como o CPF está no formato de Texto, vamos extrair os dígitos do CPF *
' um a um, converter o respectivo valor de texto para número e atribuir *
' esse valor para as variáveis d1 até d11. *
' *****
    d1 = Val(Mid$(CPF, 1, 1))
    d2 = Val(Mid$(CPF, 2, 1))
    d3 = Val(Mid$(CPF, 3, 1))
    d4 = Val(Mid$(CPF, 5, 1))
    d5 = Val(Mid$(CPF, 6, 1))
    d6 = Val(Mid$(CPF, 7, 1))
    d7 = Val(Mid$(CPF, 9, 1))
    d8 = Val(Mid$(CPF, 10, 1))
    d9 = Val(Mid$(CPF, 11, 1))
    d10 = Val(Mid$(CPF, 13, 1))
    d11 = Val(Mid$(CPF, 14, 1))
' *****
' A partir de agora passo a utilizar os valores anteriores para cálculo *
' do dígito verificador do CPF *
' *****

' Cálculo do primeiro DV

Soma1 = ((d1*10)+(d2*9)+(d3*8)+(d4*7)+(d5*6)+(d6*5)+(d7*4)+(d8*3)+(d9*2))
Resto1 = (Soma1 Mod 11)

    If (Resto1 <= 1) Then
        DV1 = 0
    Else
        DV1 = 11 - Resto1
    End If

' Agora inicio o cálculo do segundo DV, já incorporando
' o segundo DV como parte do CPF, para o cálculo.

Soma2=(d1*11)+(d2*10)+(d3*9)+(d4*8)+(d5*7)+(d6*6)+(d7*5)+(d8*4)+(d9*3)+(DV1* 2)
Resto2 = (Soma2 Mod 11)

    If (Resto2 <= 1) Then
        DV2 = 0
    Else
        DV2 = 11 - Resto2
    End If


' Agora faço o teste para saber se os DVs calculados (DV1 e DV2)
' conferem com os DVs do CPF - d10 e d11

If ((DV1 <> d10) Or (DV2 <> d11)) Then

    ' Atribuo a palavra "Inválido" para uma variável com o mesmo
    ' nome da função - ValidaCPF.
    ' Essa é a maneira de fazer com que a função retorne um valor,
    ' ou seja, atribuindo o valor a ser retornado, à uma variável
    ' com o mesmo nome da função.
    ValidaCPF = "Inválido"
Else
    ' Atribuo a palavra "Válido" para uma variável com o mesmo
    ' nome da função - ValidaCPF.
```



```
' Essa é a maneira de fazer com que a função retorne um valor,  
' ou seja, atribuindo o valor a ser retornado, à uma variável  
' com o mesmo nome da função.  
  
ValidaCPF = "Válido"  
End If  
  
End If  
  
End Function
```

13. Clique no botão () para salvar a função ValidaCPF.
14. Feche a janela do Editor do Visual Basic.
15. Você estará de volta à planilha Números de CPF. a Próxima lição aprenderemos a utilizar a função ValidaCPF para verificar se os CPFs da planilha são válidos ou não.
16. Salve e feche a planilha.

Na próxima lição mostrarei como utilizar a função ValidaCPF, criada nesta lição.

Lição 24: Usando a Função ValidaCPF, Criada na Lição 21

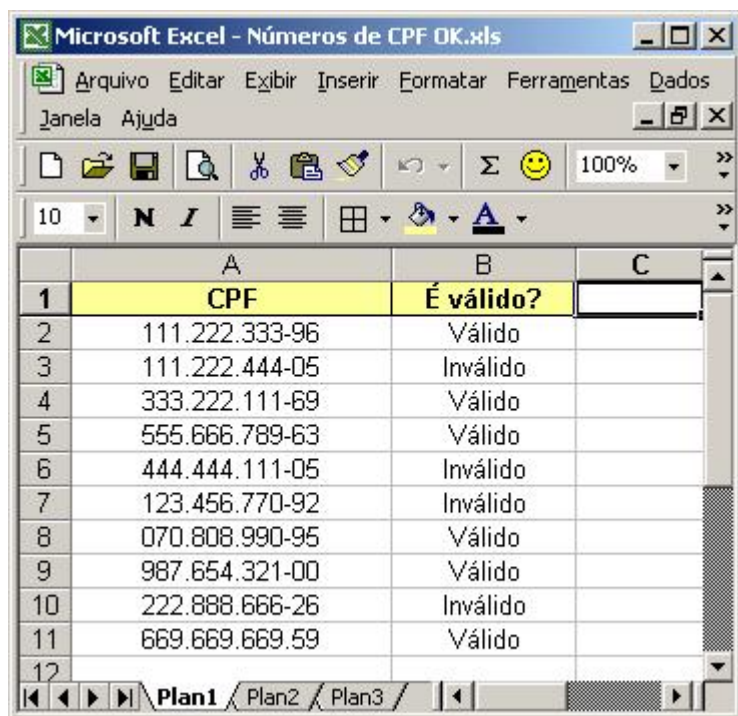
Nessa lição veremos como utilizar a função ValidaCPF, criada na lição Anterior.

Exercício: Utilizar a função ValidaCPF criada na lição anterior.

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\Números de CPF.xls.
3. Clique na célula B2 e digite a seguinte fórmula:

=ValidaCPF(A2)

4. Estenda essa fórmula para o intervalo de A2 até A11.
5. Observe que, rapidamente, o Excel usa a função ValidaCPF (criada na lição anterior), para informar se o CPF é válido ou inválido, conforme indicado na Figura a seguir:



The screenshot shows a Microsoft Excel window titled "Números de CPF OK.xls". The spreadsheet has three columns: A (CPF), B (É válido?), and C. The data is as follows:

	A	B	C
1	CPF	É válido?	
2	111.222.333-96	Válido	
3	111.222.444-05	Inválido	
4	333.222.111-69	Válido	
5	555.666.789-63	Válido	
6	444.444.111-05	Inválido	
7	123.456.770-92	Inválido	
8	070.808.990-95	Válido	
9	987.654.321-00	Válido	
10	222.888.666-26	Inválido	
11	669.669.669.59	Válido	
12			

6. O endereço da célula onde está o CPF é passado como parâmetro para a função ValidaCPF. Esse valor é utilizado para cálculo do DV e definir se o CPF é válido ou não.
7. Observe que uma vez criada a função podemos utilizá-la sempre que necessário, nas pastas de trabalho onde foi criado o módulo no qual está a função.
8. Salve e feche a planilha.

Lição 25: Mais Alguns Exemplos de Funções Personalizadas

Nesta lição apresentarei mais alguns exemplos de criação de funções personalizadas. É importante lembrar que para que uma função personalizada, possa ser utilizada em todas as planilhas de uma pasta de trabalho, esta função deve ser criada em um módulo de código do VBA, conforme exemplificado nas lições anteriores.

Nesta lição vamos criar duas funções que serão bastante úteis e que, com certeza, você utilizará nas planilhas que criar no Excel. Para criar estas funções vou utilizar o objeto Range, para fazer referência a faixa de células onde estão os valores de dados a serem utilizados pelas funções. Você ainda não aprendeu sobre o Objeto Range, mas não se preocupe, pois esta objeto será detalhadamente estudado, nos demais módulos deste curso. Na tabela a seguir descrevo as funções que iremos criar nesta lição:

Nome da função	Descrição	Parâmetro 1	Parâmetro 2	Parâmetro 2
Conta_Intervalo	Retorna o número de valores, de uma faixa de células, que estão dentro de um determinado intervalo.	Faixa onde estão os valores, onde será feita a contagem	Valor do limite inferior da faixa	Valor do limite superior da faixa
Soma_Intervalo	Soma os valores, de uma faixa de células, que estão dentro de um determinado intervalo.	Faixa onde estão os valores, onde será feita a soma	Valor do limite inferior da faixa	Valor do limite superior da faixa

A seguir coloco como seria a sintaxe de uso destas funções que estamos criando:

= **Conta_Intervalo(A1:A50;10;50)**: Conta quantos valores estão entre 10 e 50, na faixa de A1 até A50

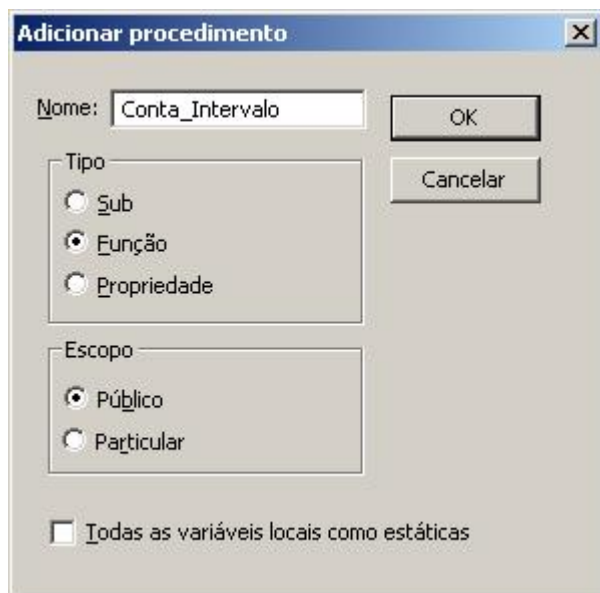
= **Soma_Intervalo(A1:A50;10;50)**: Soma os valores que estão entre 10 e 50, na faixa de A1 até A50

Muito bem, agora é hora de criarmos as funções Conta_Intervalo e Soma_Intervalo e depois utilizá-las em um exemplo prático.

Exemplo - 06: Criar as funções Conta_Intervalo e Soma_Intervalo, descritas na tabela anterior.

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 1 - Exercício 06.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Vamos criar a função Conta_Intervalo de tal maneira que ele possa ser utilizada em qualquer local da planilha. Para isso vamos criá-la como uma função Pública, dentro de um módulo do VBA.
5. Para inserir um módulo, selecione o comando **Inserir -> Módulo**. Será criado o Módulo 1. Agora vamos inserir uma função, dentro desse módulo. Clique em Módulo 1, no painel da esquerda para selecioná-lo.

6. Selecione o comando **Inserir -> Procedimento**.
7. Será aberta a janela Adicionar procedimento. Preencha os dados conforme indicado na Figura a seguir:



8. Clique em OK. Será criada a declaração da função. Agora altere a declaração para incluir os parâmetros, conforme indicado no exemplo a seguir:

Public Function Conta_Intervalo(Faixa As Range, ValInicial As Double, ValFinal As Double) As Long

End Function

9. O código da função, basicamente irá declarar uma variável contador, e percorrer todas as células do intervalo fornecido testando. Se o valor da célula estiver no intervalo definido pelos parâmetros ValInicial e ValFinal, o contador será incrementado, caso contrário, será lido o próximo valor da faixa informada no parâmetro Faixa. No final o valor da variável contador será atribuído ao nome da função, para que este valor seja retornado pela função.
10. Digite o código da função, conforme indicado na listagem a seguir:

```
Public Function Conta_Intervalo(Faixa As Range, ValInicial As Double, ValFinal As Double) As Long
```

```
    ' Declaração da variável contador
```

```
    Dim ContaQuantos As Long
```

```
    ' Inicializo a variável contador em zero
```

```
    ContaQuantos = 0
```

```
    ' Faça um loop através de todas as células da faixa passada
```

```
    ' como parâmetro. Para isso uso a estrutura For...Each, a qual
```

```
    ' será detalhada nas lições do Módulo 2.
```

For Each Célula In Faixa.Cells

' Texto para ver se o valor está dentro da faixa definida
' pelos parâmetros ValInicial e ValFinal. Se estiver, incremento
' o contador.

If (Célula.Value >= ValInicial) And (Célula.Value <= ValFinal) Then
 ContaQuantos = ContaQuantos + 1
End If

Next

' Atribuo o valor da variável ContaQuantos ao nome da função,
' para que este valor possa ser retornado pela função

Conta_Intervalo = ContaQuantos

End Sub

11. Pressione Ctrl+B para salvar o código da função e pressione Alt+Q para fechar o editor do VBA e voltar à planilha Modulo 1 - Exercício 06.xls.

12. Agora vamos contar quantos salários estão em cada faixa. Isso será feito nas células de B18 a B20. Nestas células, utilize as fórmulas indicadas na tabela a seguir:

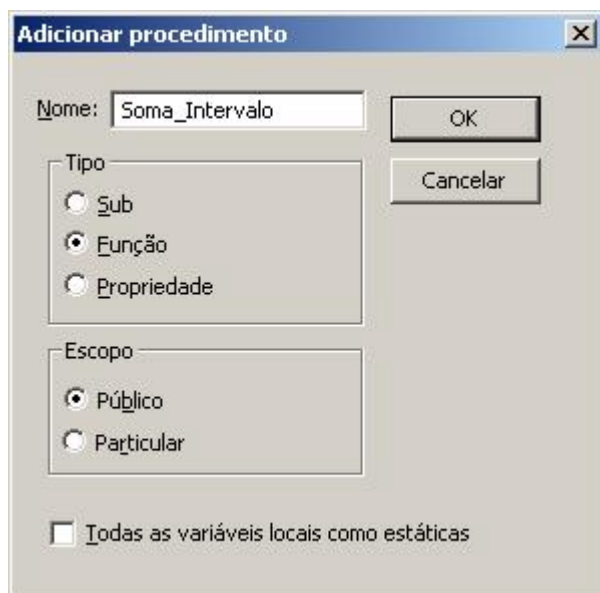
Célula	Fórmula	Descrição
B18	= CONT.SE(E4:E15;"<500")	Quantos salários estão com valor menor do que 500
B19	= Conta_Intervalo(E4:E15;500;1000)	Quantos salários estão na faixa entre 500 e 1000.
B20	= CONT.SE(E4:E15;">1000")	Quantos salários estão com valor maior do que 1000

13. Você deve obter os resultados indicados na Figura a seguir:

The screenshot shows an Excel window titled 'Microsoft Excel - Modulo 1 - Exercício 06.xls'. The active sheet is 'Plan1'. The data is organized as follows:

	A	B	C	D
12	Adão da Silva	620,00	75	65
13	Eva Costa Moreira	280,00	50	35
14	Pedrolino Santos	330,00	50	35
15	José Mariano	1000,00	75	65
16				
17		Quantos ?	Soma dos Salários por faixa	
18	Salários <500	6		
19	Salários entre 500 e 1000	4		
20	Salários >1000	2		
21				
22				

14. Observe que agora temos uma função personalizada – Conta_Intervalo, a qual pode ser utilizada para contar quantos valores estão dentro de um determinado intervalo, em uma faixa de valores, o que não era possível de ser feito diretamente com a função CONT.SE, a qual aceita um único teste como critério.
15. Seguindo os passos indicados anteriormente, vamos criar a nova função Soma_Intervalo.
16. Pressione Alt+F11 para exibir novamente o editor do VBA e clique em Módulo 1, no painel da esquerda, para selecioná-lo. Posicione o cursor no final do código já existente, na primeira linha após o último End Sub. Selecione o comando **Inserir -> Procedimento**.
17. Será aberta a janela Adicionar procedimento. Preencha os dados conforme indicado na Figura a seguir:



18. Clique em OK. Será criada a declaração da função. Agora altere a declaração para incluir os parâmetros, conforme indicado no exemplo a seguir:

Public Function Soma_Intervalo(Faixa As Range, ValInicial As Double, ValFinal As Double) As Long

End Function

19. O código da função, basicamente irá declarar uma variável SomaValores, e percorrer todas as células do intervalo fornecido, testando. Se o valor da célula estiver no intervalo definido pelos parâmetros ValInicial e ValFinal, o valor da célula será somado à variável SomaValores, caso contrário, será lido o próximo valor da faixa informada no parâmetro Faixa. No final o valor da variável SomaValores será atribuído ao nome da função, para que este valor seja retornado pela função.
20. Digite o código da função, conforme indicado na listagem a seguir:

```
Public Function Soma_Intervalo(Faixa As Range, ValInicial As Double, ValFinal As Double) As Long

' Declaração da variável de soma

Dim SomaValores As Double

' Inicializo a variável de soma em zero

SomaValores = 0

' Faça um loop através de todas as células da faixa passada
' como parâmetro. Para isso uso a estrutura For...Each, a qual
' será detalhada nas lições do Módulo 2.

For Each Célula In Faixa.Cells

    ' Texto para ver se o valor está dentro da faixa definida
    ' pelos parâmetros ValInicial e ValFinal. Se estiver, somo
    ' o valor da célula na variável SomaValores.

    If (Célula.Value >= ValInicial) And (Célula.Value <= ValFinal) Then
        SomaValores = SomaValores + Célula.Value
    End If

Next

' Atribuo o valor da variável SomaValores ao nome da função,
' para que este valor possa ser retornado pela função

Soma_Intervalo = SomaValores

End Sub
```

21. Pressione Ctrl+B para salvar o código da função e pressione Alt+Q para fechar o editor do VBA e voltar à planilha Modulo 1 - Exercício 06.xls.
22. Agora vamos somar o total de salários em cada faixa. Isso será feito nas células de C18 a C20. Nestas células, utilize as fórmulas indicadas na tabela a seguir:

Célula	Fórmula	Descrição
C18	= SOMASE(E4:E15;"<500";E4:E15)	Soma dos salários estão com valor menor do que 500.
C19	= Soma_Intervalo(E4:E15;500;1000)	Soma dos salários estão na faixa entre 500 e 1000.
C20	=SOMASE(E4:E15;">1000";E4:E15)	Soma dos salários estão com valor maior do que 1000.

23. Você deve obter os resultados indicados na Figura a seguir:

The screenshot shows the Microsoft Excel interface with the following data:

	A	B	C	D	E
12	Adão da Silva	620,00	75	65	480,00
13	Eva Costa Moreira	280,00	50	35	195,00
14	Pedrolino Santos	330,00	50	35	245,00
15	José Mariano	1000,00	75	65	860,00
16					
17		Quantos ?	Soma dos Salários por faixa		
18	Salários <500	6	1910		
19	Salários entre 500 e 1000	4	2843		
20	Salários >1000	2	2190,3		
21					

24. Observe que agora temos uma função personalizada – Soma_Intervalo, a qual pode ser utilizada para Somar os valores que estão dentro de um determinado intervalo, em uma faixa de valores, o que não era possível de ser feito diretamente com a função SOMASE, a qual aceita um único teste como critério.

Importante: É claro que existem diversas outras maneiras de implementar estas funções, até mesmo é possível resolver estes problemas sem a criação de funções personalizadas. O objetivo destes exemplos é ilustrar a possibilidade da criação de funções personalizadas, como criá-las e como utilizá-las. Na próxima lição, apresentarei mais alguns exemplos de criação de funções personalizadas.

Lição 26: Mais Alguns Exemplos de Funções Personalizadas

Nesta lição mostrarei mais alguns exemplos de códigos de funções personalizadas que poderão ser úteis em suas planilhas.

Exemplo 1: Uma função para calcular uma Média Condicional. A função Média_SE, irá calcular a média aritmética em uma faixa de valores, mas serão contabilizados somente os valores que estão dentro de uma determinada faixa de valores. A função receberá como parâmetros a faixa de valores, o valor inferior da faixa e o valor superior da faixa. A sintaxe da função está indicada a seguir:

=Média_SE(A1:A50;100;500)

este exemplo calcula a média dos valores que estão entre 100 e 500, na faixa de A1 até A500.

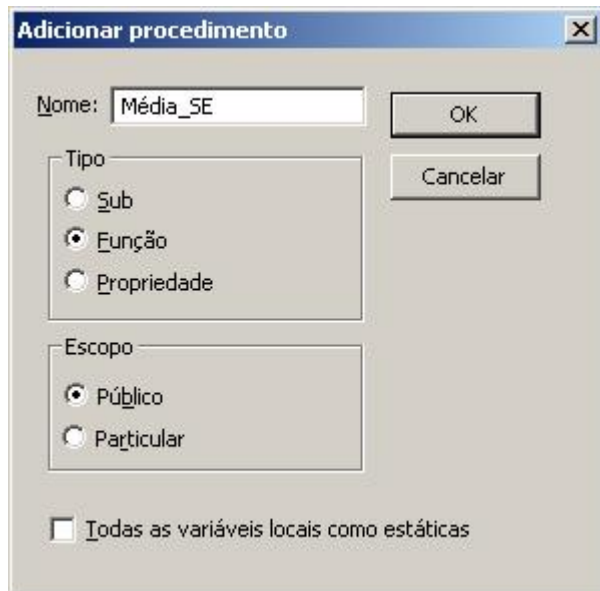
Vamos criar a função Média_Se no Módulo 1, da planilha C:\Programação VBA no Excel\Modulo 1 - Exercício 06.xls, com a qual já trabalhamos nas lições anteriores. Um detalhe interessante é que para criar a função Média_SE, utilizaremos as funções Soma_Intervalo e Conta_Intervalo, criadas na Lição 23. A média condicional nada mais é do que o resultado da função Soma_Intervalo (na mesma faixa, com os mesmos valores), dividido pelo resultado da função Conta_Intervalo (na mesma faixa, com os mesmos valores).

Para criar a função Média_SE, siga os passos indicados a seguir:

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 1 - Exercício 06.xls. Se for emitido uma aviso de Macros, clique em Ativar Macros
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Vamos criar a função Média_SE, de tal maneira que ele possa ser utilizada em qualquer local da planilha. Para isso vamos cria-la como uma função Pública, dentro de um módulo do VBA.
5. Clique em Módulo 1 (já criado nas lições anteriores), no painel da esquerda para selecioná-lo.
6. Selecione o comando **Inserir -> Procedimento**.
7. Será aberta a janela Adicionar procedimento. Preencha os dados conforme indicado na Figura a seguir.
8. Clique em OK. Será exibido o código básico da função, com a declaração da função e o comando End Sub. Altere o código para definir os parâmetros da função e o tipo de dados a ser retornado pela função. O código deve ficar conforme indicado a seguir:

Public Function Média_SE(Faixa As Range, LimInf As Double, LimSup As Double) As Double

End Function



9. Agora vamos alterar o código da função para que sejam feitos os cálculos necessários. O código da função será uma única linha, conforme indicado a seguir:

Public Function Média_SE(Faixa As Range, LimInf As Double, LimSup As Double) As Double

Média_SE = Soma_Intervalo(Faixa, LimInf, LimSup) / Conta_Intervalo(Faixa, LimInf, LimSup)

End Function

10. Conforme descrito anteriormente, com uma única linha, usando as funções já criadas anteriormente, foi possível implementar a função Média_SE. Na figura a seguir, temos um exemplo do uso desta função, para cálculo da média dos Salários, na faixa de E4:E15, para os salários entre 500 e 1000:

	A	B	C	D	E
12	Adão da Silva	620,00	75	65	480,00
13	Eva Costa Moreira	280,00	50	35	195,00
14	Pedrolino Santos	330,00	50	35	245,00
15	José Mariano	1000,00	75	65	860,00
16					
17		Quantos ?	Soma dos Salários		
18	Salários <500	6	1910		
19	Salários entre 500 e 1000	4	2843	710,75	
20	Salários >1000	2	2190,3		
21					

Mais alguns exemplos de funções personalizadas:

Podemos, facilmente, modificar um pouco o código das funções criadas anteriormente, para criar novas funções. Por exemplo, você pode criar uma função que faça a soma de valores em um intervalo, porém ignorando valores negativos. Ou você pode criar uma função que faz a média aritmética dos valores em um intervalo, ignorando valores iguais a zero. A seguir apresento o código destas duas funções: Soma_Positivo, para somar valores ignorando os negativos e Média_Nzero, para fazer a média de valores em uma faixa, ignorando os zeros.

Função Soma_Positivo:

```
Public Function Soma_Positivo(Faixa As Range) As Double

    ' Declaração da variável de soma
    Dim SomaValores As Double

    ' Inicializo a variável de soma em zero

    SomaValores = 0

    ' Faça um loop através de todas as células da faixa passada
    ' como parâmetro. Para isso uso a estrutura For...Each, a qual
    ' será detalhada nas lições do Módulo 2.

    For Each Célula In Faixa.Cells

        ' Testo para ver se o valor é diferente de zero

        If (Célula.Value >= 0) Then
            SomaValores = SomaValores + Célula.Value
        End If
    Next

    ' Atribuo o valor da variável SomaValores ao nome da função,
    ' para que este valor possa ser retornado pela função

    Soma_Positivo = SomaValores

End Function
```

A sintaxe para usar esta função no Excel é indicada a seguir:

= Soma_Positivo(A1:A50)

Neste exemplo, será retornada a soma das células com valores positivos, na faixa de A1 até A50.

Função Média_NZero:

```
Public Function Média_NZero(Faixa As Range) As Double

' Declaração da variável de soma e da variável que conta os itens

Dim SomaValores As Double
Dim ContaItens As Long

' Inicializo a variável de soma e de contar itens em zero

SomaValores = 0
ContaItens = 0

' Faça um loop através de todas as células da faixa passada
' como parâmetro. Para isso uso a estrutura For...Each, a qual
' será detalhada nas lições do Módulo 2.

For Each Célula In Faixa.Cells

' Testo para ver se o valor é diferente de zero

If (Célula.Value > 0) Or (Célula.Value < 0) Then
    SomaValores = SomaValores + Célula.Value
    ContaItens = ContaItens + 1
End If

Next

' Atribuo o valor do cálculo da média ao nome da função,
' para que este valor possa ser retornado pela função

Média_NZero = SomaValores / ContaItens

End Function
```

A sintaxe para usar esta função no Excel é indicada a seguir:

= Média_NZero(A1:A50)

Neste exemplo, será retornada a média das células com valores diferentes de zero, na faixa de A1 até A50.

Lição 27: Conclusão do Módulo 1

Neste módulo você aprendeu os fundamentos sobre Macros e VBA. Mostrei exatamente o que é uma macro, o que é programação VBA, o ambiente de programação, as principais funções do VBA e como criar os primeiros programas e funções personalizadas. Os conceitos apresentados neste módulo serão fundamentais para os demais módulos do curso. Em todos os exemplos do curso, você irá utilizar um ou mais conceito apresentado neste módulo.

Módulo 1 – Introdução às Macros e ao VBA no Excel

- Lição 01: Uma Introdução às Macros
- Lição 02: O que são exatamente Macros???
- Lição 03: Conhecendo do que é feita uma Macro
- Lição 04: Operações com Macros
- Lição 05: Associando botões com macros
- Lição 06: Introdução ao VBA
- Lição 07: O Ambiente de programação – o Editor VBA – Parte I
- Lição 08: O Ambiente de programação – o Editor VBA – Parte 2
- Lição 09: VBA - Declaração de Variáveis
- Lição 10: VBA - Cálculos, Operadores Aritméticos e Exemplos
- Lição 11: VBA - Estrutura If...Then e os Operadores de Comparação
- Lição 12: Estruturas For...Next, Do...While e Do...Until
- Lição 13: VBA - Funções do VBA – Funções de Tipo – Parte 1
- Lição 14: VBA - Funções do VBA – Funções de Tipo – Parte 2
- Lição 15: VBA - Funções do VBA – Funções de Conversão de Tipo – Parte 1
- Lição 16: VBA - Funções do VBA – Funções de Conversão de Tipo – Parte 2
- Lição 17: VBA - Funções do VBA – Funções Para Tratamento de Texto
- Lição 18: VBA - Funções do VBA – Data, Hora e Funções Matemáticas
- Lição 19: VBA - O Conceito de Módulos, Procedimentos e Funções – Parte I
- Lição 20: VBA - O Conceito de Módulos, Procedimentos e Funções – Parte II
- Lição 21: VBA – Criando Funções Personalizadas – Parte I
- Lição 22: VBA – Um exemplo prático – calculando o DV do CPF - Algoritmo
- Lição 23: Usando o VBA Para Criar uma Função de Validação do DV do CPF
- Lição 24: Usando a Função ValidaCPF, Criada na Lição 23
- Lição 25: Mais Alguns Exemplos de Funções Personalizadas
- Lição 26: Mais Alguns Exemplos de Funções Personalizadas
- Lição 27: Conclusão do Módulo 1

No próximo módulo iniciaremos o estudo do Modelo de Objetos do Excel. Com o uso dos objetos do Modelo de Objetos do Excel, você será capaz de criar rotinas de programação muito mais poderosas, as quais poderão solucionar uma série de problemas avançados no Excel.

Bibliografia recomendada:

Confira as dicas de livros de Excel no seguinte endereço:

<http://www.juliobattisti.com.br/indicados/excel.asp>

Autor: Júlio Cesar Fabris Battisti

Site: www.juliobattisti.com.br

Confira também o livro: "Windows Server 2003 – Curso Completo, 1568 páginas"

Página 113 de 527

Módulo 2 – O Modelo de Objetos do Excel

Nas lições deste módulo apresentarei o conceito mais importante quando se trata de programação VBA no Excel: “**A Hierarquia de Objetos do Excel**”. Você verá que com o uso dos Objetos do Excel é possível acessar qualquer elemento de uma planilha. Apresentarei o primeiro objeto na Hierarquia de Objetos: O Objeto Application.

Um objeto representa um elemento da planilha do Excel ou o próprio Excel. Por exemplo, o objeto mais “alto” na hierarquia de objetos é o objeto Application. Este objeto representa o próprio Excel, ou seja, o aplicativo Excel. Descendo um pouco mais na hierarquia temos o objeto Workbook. O objeto Workbook representa uma pasta de trabalho do Excel. Dentro de uma pasta de trabalho (arquivo .xls), podemos ter uma ou mais planilhas. As planilhas são representadas por objetos Worksheet. Dentro de uma planilha posso ter dados em várias faixas da planilha. Uma faixa de células é representada pelo objeto Range.

O parágrafo anterior serve para dar uma pequena idéia do que é possível de ser feito, usando a Hierarquia de Objetos do Excel na programação VBA. Literalmente é possível ter acesso a qualquer elemento de uma planilha do Excel. Por exemplo, você poderia criar uma rotina que percorre todos os dados de uma ou mais planilhas de uma pasta de trabalho e coloca em negrito e fonte vermelha, todos os dados que atendam a determinadas condições. Ou você pode criar uma rotina que percorre uma faixa de células (Range) e exclui todas as linhas em branco.

Cada objeto da Hierarquia de objetos do Excel tem métodos e propriedades. Os métodos são utilizados para executar determinadas ações. Por exemplo, abrir uma planilha, ativar uma planilha, selecionar uma faixa de células, excluir uma faixa de células e assim por diante. Os métodos estão sempre associados a ações e são representados, por isso, por verbos. As propriedades descrevem características dos objetos. Por exemplo, o objeto Range tem uma propriedade chamada Count, a qual retorna o número de células da faixa. As propriedades representam características dos objetos.

Neste módulo farei uma descrição do modelo de Objetos do Excel, apresentarei uma visão geral do modelo, bem como uma descrição da sintaxe para uso de métodos e propriedades de um objeto.

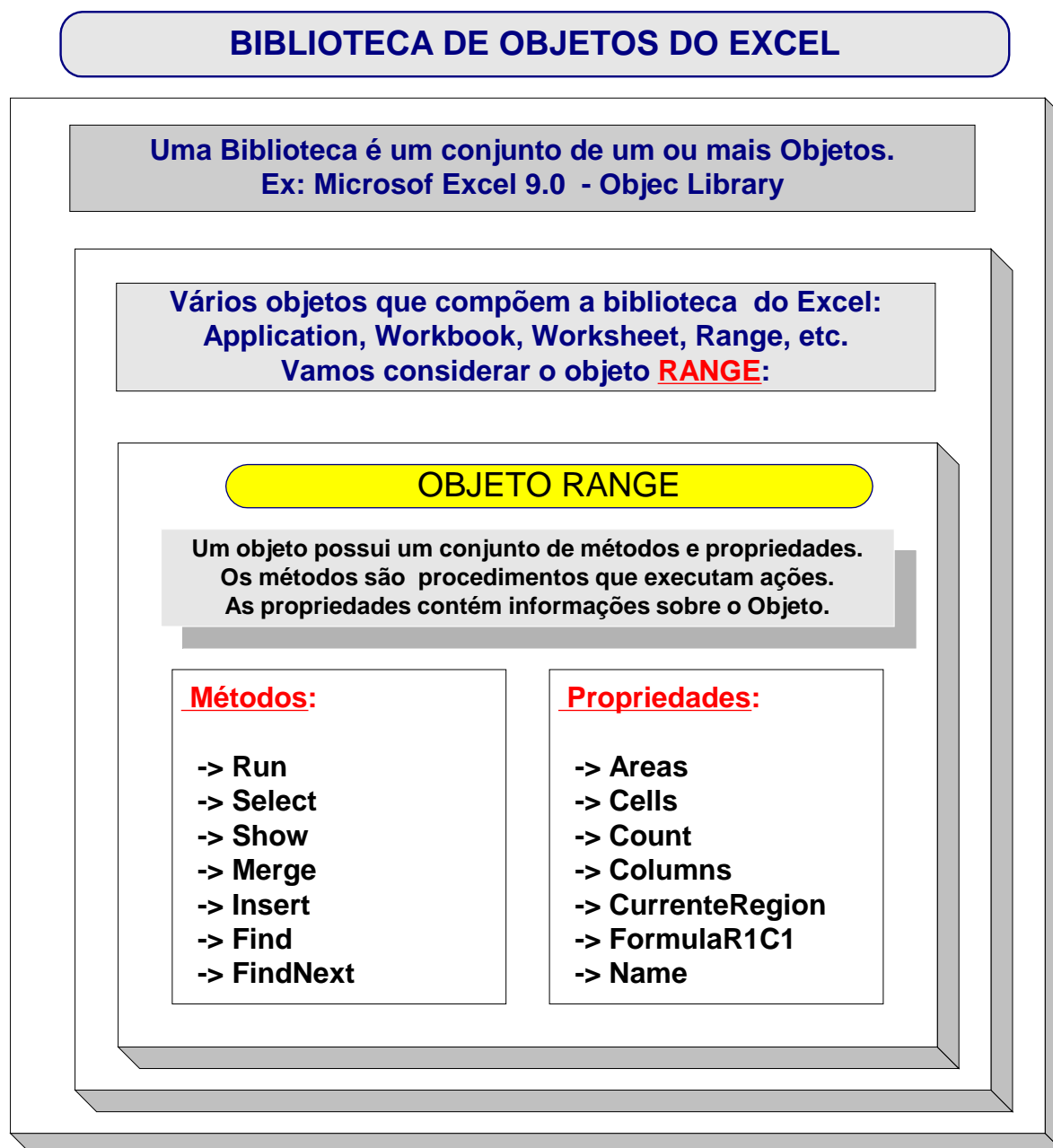
Nas lições deste módulo, faremos um estudo detalhado do objeto Application. Você aprenderá a utilizar uma série de métodos e propriedades do objeto Application. Também mostrarei o conceito de coleções e como utilizar as coleções do objeto Application. Além dos exemplos teóricos, para ilustrar o uso dos métodos e propriedades do objeto Application, também apresentarei uma série de exemplos práticos, para ilustrar a utilização do objeto Application.

Se eu tivesse que resumir a importância deste módulo, resumiria no fato de entender o que é um modelo de Objetos, principalmente o modelo de objetos do Excel. Não se preocupe em decorar métodos e propriedades, pois, conforme mostrarei, o próprio Excel fornece ferramentas para que possamos encontrar ajuda, rapidamente, sobre os métodos e propriedades de qualquer objeto.

Lição 01: O que é um Modelo de Objetos?

Uma Hierarquia de Objetos, como a do Excel, formada por vários objetos, cada um deles com dezenas (alguns com centenas) de métodos e propriedades, também é conhecida como uma Biblioteca. Nas lições do Módulo 1 tratamos dos aspectos básicos do VBA. Com os aplicativos do Office (Word, Excel, Access, PowerPoint e Outlook), temos acesso a um vasto conjunto de Bibliotecas; cada biblioteca com dezenas/centenas de objetos, cada objeto com muitos métodos, propriedades e coleções. Com a utilização dos objetos disponibilizados pelo Office, podemos criar soluções bastante sofisticadas. Nesta lição vamos entender exatamente o que são e como se relacionam, os seguintes itens: Bibliotecas, Objetos, Propriedades, Métodos, Coleções.

Para início de conversa, considere a figura a seguir:



Os diversos objetos disponíveis estão agrupados em Bibliotecas. Uma Biblioteca é um conjunto de objetos que são utilizados para uma determinada função/atividade. Por exemplo, todos os objetos para acesso a dados são agrupados em uma biblioteca chamada DAO - Data Access Objects. Existe uma outra biblioteca para acesso a dados, conhecida como ADO - Activex Data Objects. Existe a biblioteca com os diversos objetos do **Excel – Excel 9.0 Object Library**, representada no diagrama da figura anterior e assim por diante. Existem dezenas de bibliotecas disponíveis. Isso demonstra bem o poder da utilização do VBA em conjunto com os Objetos/Bibliotecas disponíveis.

Em cada Biblioteca estão disponíveis dezenas/centenas de objetos. Cada objeto é utilizado para um conjunto de ações específico. Por exemplo: O objeto RecordSet é utilizado para acessar dados de uma tabela de um banco de dados. Uma vez criado um objeto RecordSet, podemos realizar uma série de operações sobre os diversos registros da tabela. Um objeto Range é utilizado para fazer referência a uma faixa de células em uma planilha do Excel. Uma vez criado o objeto Range, podemos utilizar os diversos métodos e propriedades deste objeto.

Cada objeto possui um conjunto de métodos, propriedades e coleções. Um método realiza uma operação específica, como por exemplo o método Calculate, do objeto Range. Este método é utilizado para recalcular as células na faixa representada pelo objeto Range.

No código VBA, utilizamos a seguinte sintaxe, para fazer acesso aos métodos e propriedades de um objeto:

```
NomeDoObjeto.NomeDoMétodo(par1, par2, ..., parn)
```

Por exemplo, para utilizar o método Quit, de um objeto Application, atribuído a variável ExApp, utilizaríamos a seguinte sintaxe:

```
ExApp.Quit
```

Uma propriedade descreve uma característica do objeto. Por exemplo, temos uma propriedade do objeto Range, chamada Count, a qual retorna o número de células de um objeto Range. Por exemplo, para atribuir à variável quantos, o número de células de um objeto Range, associado a variável ExRange, utilizamos a seguinte sintaxe:

```
quantos = ExRange.Count
```

Uma coleção é um conjunto de elementos do mesmo tipo. Por exemplo, todo banco de dados do Microsoft Access, possui uma coleção chamada Forms. Através desta coleção podemos ter acesso a todos os Formulários do banco de dados. Toda planilha do Excel tem uma coleção chamada Worksheets. Através dessa coleção temos acesso a todas as planilhas de uma pasta de trabalho do Excel (arquivo .xls).

Podemos percorrer todos os elementos de uma coleção, utilizando a estrutura de Controle For...Each, que será tratada no próximo tópico.

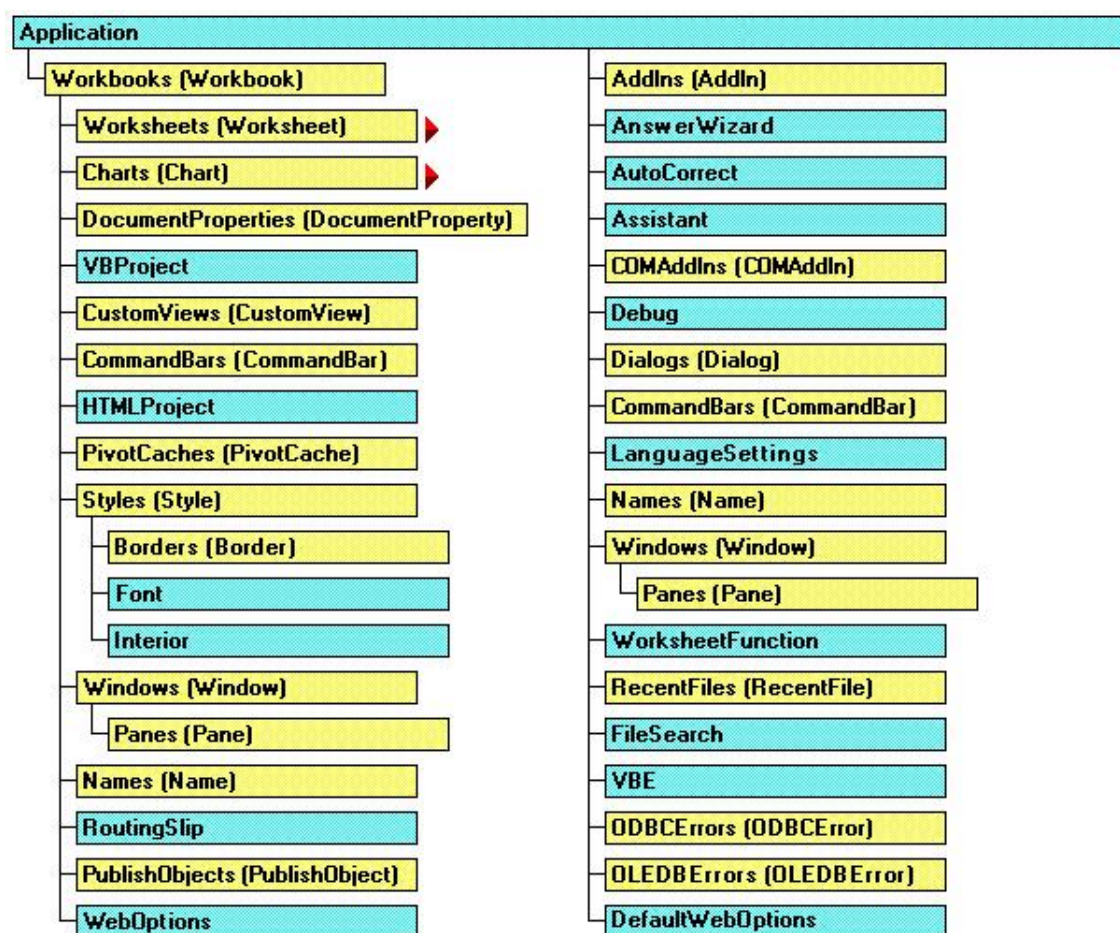
A Estrutura For...Each.

A estrutura For...Each é utilizada para "percorrer" todos os elementos de uma coleção. Por exemplo, se quisermos percorrer todos os elementos da coleção Worksheets de uma pasta de trabalho do Excel, exibindo o nome de cada planilha, devemos utilizar a estrutura For/Each, para percorrer todos os elementos da coleção Worksheets, exibindo o nome de cada um dos elementos.

Nota: Nos exemplos de código das próximas lições, veremos o funcionamento da estrutura For...Each em detalhes.

Uma visão geral da Biblioteca de Objetos do Excel:

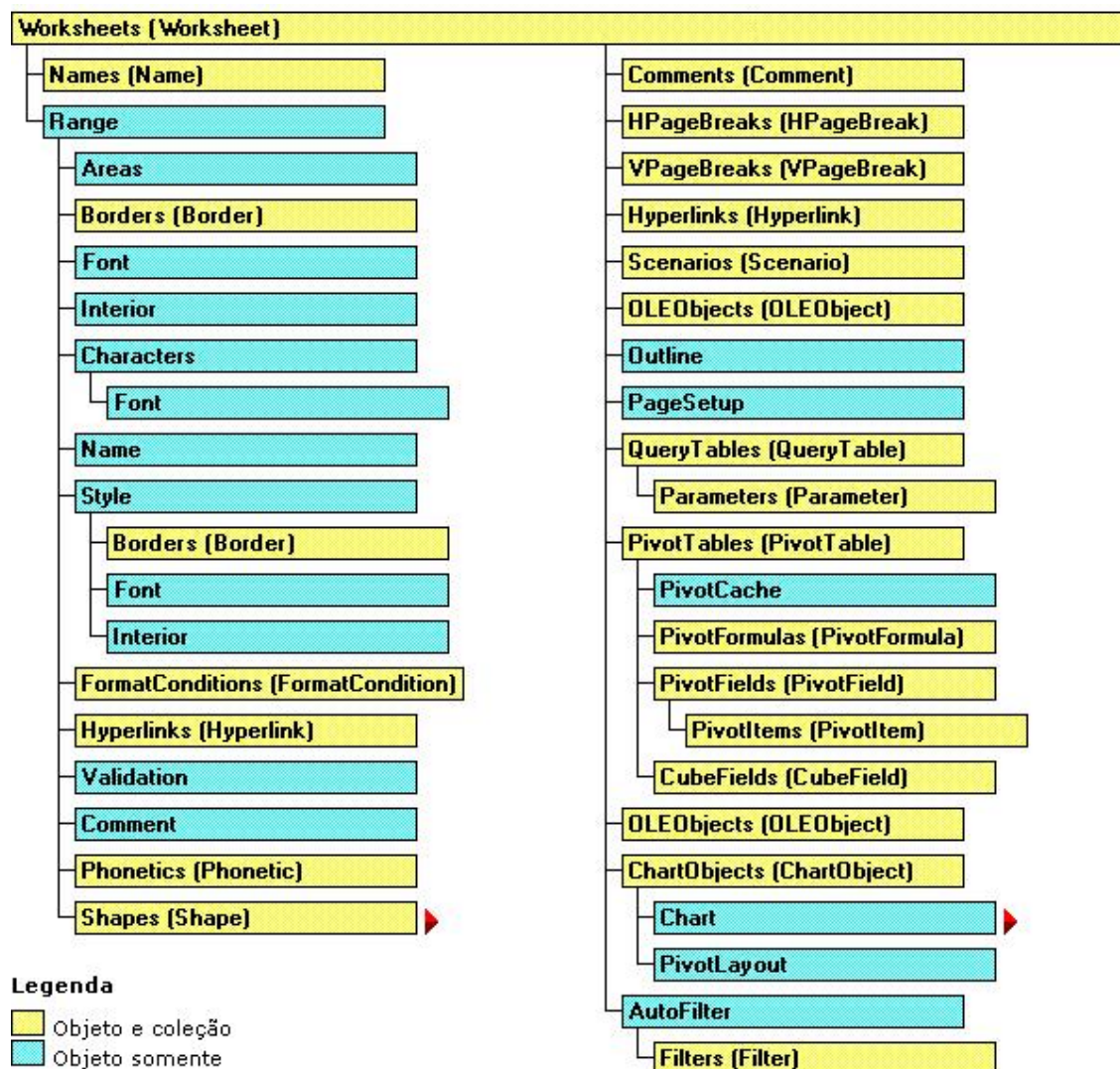
Nas figuras a seguir temos uma visão geral da Biblioteca de Objetos do Microsoft Excel.



Nota: Quando trabalhamos com o VBA, temos que utilizar os nomes dos comandos, objetos, funções, métodos e propriedades em inglês. Não existe tradução. Por exemplo, se ao invés de Left, usarmos Esquerda para o nome da função, será gerado um erro, informando que não existe a função Esquerda.

Nas demais lições desse curso, estudaremos alguns dos principais objetos, da Biblioteca de Objetos do Excel.

Observe que uma biblioteca de objetos forma uma espécie de hierarquia de objetos. Por exemplo, um objeto Worksheet somente existe dentro do contexto de um objeto Application. Já o próprio objeto Worksheet pode possuir outros objetos, cada objeto com os seus métodos e propriedades, conforme indicado na Figura a seguir:



Veja que um único objeto da hierarquia de objetos – Worksheet, é composto de dezenas de outros objetos, métodos, propriedades e coleções. É esse grande número de objetos que possibilita o acesso a todo e qualquer elemento de uma planilha do Excel, através da programação VBA, conforme descrito na introdução deste módulo.

Lição 02: Descrição dos Principais Objetos do Modelo de Objetos do Excel

Nesta lição farei uma breve descrição dos principais objetos do Modelo de Objetos do Excel, objetos estes que serão estudados, em detalhes, no decorrer deste curso. Ter uma visão geral dos principais objetos é importante, principalmente para ter um entendimento preciso de quando utilizar cada um dos objetos, para solucionar problemas práticos.

Ao descrever os principais objetos, apresentarei pequenos exemplos de código, apenas para ilustrar o uso destes objetos. Nas demais lições deste curso você aprenderá, em detalhes, a utilizar os métodos e propriedades destes objetos, através de exemplos práticos.

O Objeto Application:

O objeto Application Representa todo o aplicativo Microsoft Excel. Em outras palavras, um objeto Application representa uma instância do Excel carregada na memória. Por exemplo, ao abrir o Excel, automaticamente é criado na memória do computador, um objeto Application. O objeto Application contém:

- ▶ Definições e opções para o aplicativo como um todo (muitas das opções da caixa de diálogo Opções (menu Ferramentas), por exemplo).
- ▶ Métodos que retornem objetos do nível mais alto, como ActiveCell, ActiveSheet e assim por diante.

Usar o objeto Application:

Use a propriedade Application para retornar o objeto Application. O objeto Application tem, ao todo, 218 propriedades e métodos. Quando estamos trabalhando dentro de uma planilha do Excel, não é preciso a criação explícita de um objeto Application. Por exemplo, se dentro do código, quisermos fazer referência a uma determinada célula, não precisamos criar um objeto Application, depois um Workbook, depois um Worksheet para, finalmente, poder acessar as células de uma planilha (objeto Worksheet). Ao invés disso, podemos fazer referência direta a célula desejada. Com isso o Excel supõe que estamos trabalhando com a instância atual do Excel, dentro da pasta de trabalho atual (arquivo .xls), dentro de uma das suas planilhas, o que é bastante razoável. Apenas teríamos que criar toda essa hierarquia de objetos, se quiséssemos fazer referência a uma célula de uma planilha de uma pasta de trabalho externa.

Muitas das propriedades e métodos que retornam os objetos mais comuns da interface do usuário, como a célula ativa (propriedade ActiveCell), podem ser usados sem o qualificador de objeto Application. Por exemplo, em vez de escrever Application.ActiveCell.Font.Bold = True, você pode escrever ActiveCell.Font.Bold = True.

Nas lições deste módulo, estudaremos, detalhadamente, os métodos e propriedades do objeto Application. O estudo será feito através de exemplos práticos de utilização deste objeto.

O objeto Workbook:

O objeto Workbook representa uma pasta de trabalho do Microsoft Excel (lembre-se que uma pasta de trabalho é um arquivo .xls, dentro do qual pode haver uma ou mais planilhas. Cada planilha é representada por um objeto Worksheet, o qual será detalhado neste curso). O objeto Workbook é um membro da coleção Workbooks. A coleção Workbooks contém todos os objetos Workbook atualmente abertos no Microsoft Excel. A coleção Workbooks é uma coleção do objeto Application.

Usar o objeto Workbook:

As propriedades a seguir, para retornar um objeto Workbook, são descritas nesta seção:

- ▶ Propriedade Workbooks
- ▶ Propriedade ActiveWorkbook
- ▶ Propriedade ThisWorkbook
- ▶ Propriedade Workbooks

Use Workbooks(índice), onde índice é o número de índice ou o nome da pasta de trabalho, para retornar um único objeto Workbook. O exemplo seguinte ativa a pasta de trabalho um.

```
Workbooks(1).Activate
```

O número de índice denota a ordem na qual as pastas de trabalho foram abertas ou criadas. Workbooks(1) é a primeira pasta de trabalho criada e Workbooks(Workbooks.Count) é a última pasta de trabalho criada. A ativação de uma pasta de trabalho não altera seu número de índice. Todas as pastas de trabalho são incluídas na contagem do índice, mesmo que elas estejam ocultas.

A propriedade Name retorna o nome da pasta de trabalho. Você não pode definir o nome usando essa propriedade; se você precisa alterar o nome, use o método SaveAs (equivalente ao comando Arquivo -> Salvar como...), para salvar a pasta de trabalho com um nome diferente. O seguinte exemplo ativa Sheet1 na pasta de trabalho chamada "Cogs.xls" (a pasta de trabalho precisa já estar aberta no Microsoft Excel).

```
Workbooks("cogs.xls").Worksheets("sheet1").Activate
```

Propriedade ActiveWorkbook

A propriedade ActiveWorkbook retorna a pasta de trabalho que está ativa no momento. O exemplo seguinte define o nome do autor da pasta de trabalho ativa.

```
ActiveWorkbook.Author = "Jean Selva"
```

Propriedade ThisWorkbook

A propriedade ThisWorkbook retorna a pasta de trabalho onde há código do Visual Basic sendo executado. Na maioria dos casos, esta é a própria pasta de trabalho ativa. Entretanto, se o código do Visual Basic for parte de um suplemento (Add-in), a propriedade ThisWorkbook

não retornará a pasta de trabalho ativa. Nesse caso, a pasta de trabalho ativa é a pasta de trabalho que está chamando o suplemento, enquanto que a propriedade ThisWorkbook retorna a pasta de trabalho do suplemento.

Se você estiver criando um suplemento a partir de seu código do Visual Basic, você deverá usar a propriedade ThisWorkbook para qualificar qualquer instrução que precise ser executada na pasta de trabalho que você compila em um suplemento.

O objeto Worksheet:

Observe que estamos descendo na hierarquia de objetos do Excel. Primeiro fiz uma apresentação do objeto Application, o qual representa o próprio Excel. Em seguida falei sobre o objeto Workbook, o qual representa uma pasta de trabalho, um arquivo .xls aberto no Excel. Agora vou falar sobre o objeto Worksheet. Para cada planilha de uma pasta de trabalho, existe um objeto Worksheet associado. Observe que o modelo de objetos do Excel, representa a maneira natural como o usuário irá trabalhar com o Excel:

- ▶ Primeiro o usuário abre o Excel – objeto Application.
- ▶ Depois abre um arquivo .xls – Objeto Workbook
- ▶ Em seguida trabalha nas planilhas do arquivo que foi aberto – objeto Worksheet.
- ▶ Dentro de cada planilha, edita, inclui, exclui dados e faz calculos – Objeto Range (será visto logo em seguida).

O objeto Worksheet representa uma planilha O objeto Worksheet é um membro da coleção Worksheets. A coleção Worksheets contém todos os objetos Worksheet em uma pasta de trabalho (objeto Workbook).

Usar o objeto Worksheet

As seguintes propriedades para retornar um objeto Worksheet são descritas nesta seção:

- ▶ Propriedade Worksheets
- ▶ Propriedade ActiveSheet
- ▶ Propriedade Worksheets

Use Worksheets(índice), onde índice é número de índice ou nome da planilha, para retornar um único objeto Worksheet. O exemplo seguinte oculta a planilha um na pasta de trabalho ativa.

```
Worksheets(1).Visible = False
```

O número de índice da planilha denota a posição de uma planilha na barra de guias da pasta de trabalho. Worksheets(1) é a primeira planilha (mais à esquerda) na pasta de trabalho e Worksheets(Worksheets.Count) é a última. Todas as planilhas são incluídas na contagem do índice, mesmo quando estão ocultas.

O nome da planilha é mostrado na guia da planilha. Use a propriedade Name para definir ou retornar o nome da planilha. O exemplo seguinte protege os cenários em Sheet1.

```
Worksheets("sheet1").Protect password:="drowssap", scenarios:=True
```

O objeto Worksheet é também um membro da coleção Sheets. A coleção Sheets contém todas as planilhas da pasta de trabalho (tanto folhas de gráfico quanto planilhas de trabalho).

Propriedade ActiveSheet

Quando uma planilha é a planilha ativa, você pode usar a propriedade ActiveSheet para referir-se a ela. O exemplo seguinte usa o método Activate para ativar Sheet1, define a orientação da página como modo paisagem e, em seguida, imprime a planilha.

```
Worksheets("sheet1").Activate  
ActiveSheet.PageSetup.Orientation = xlLandscape  
ActiveSheet.PrintOut
```

O objeto Range:

O objeto Range Representa uma célula, uma linha, uma coluna, uma seleção de células contendo um ou mais blocos contíguos de células ou um intervalo 3D. Este é, sem dúvidas, o objeto mais utilizado. Você verá dezenas de exemplos práticos neste curso, nos quais será utilizado o objeto Range.

Usar o objeto Range

As seguintes propriedades e métodos para retornar um objeto Range são descritas nesta seção:

- ▶ Propriedade Range
- ▶ Propriedade Cells
- ▶ Range e Cells
- ▶ Propriedade Offset
- ▶ Método Union
- ▶ Propriedade Range

Use Range(argumento), onde argumento nomeia o intervalo, para retornar um objeto Range representando uma única célula ou um intervalo de células. O exemplo seguinte coloca o valor da célula A1 na célula A5.

```
Worksheets("Sheet1").Range("A5").Value = Worksheets("Sheet1").Range("A1").Value
```

O exemplo seguinte preenche o intervalo A1:H8 com números randômicos definindo a fórmula para cada célula do intervalo. Quando usada sem um qualificador de objeto (um objeto à esquerda do ponto), a propriedade Range retorna um intervalo da planilha ativa. Se a planilha ativa não for uma planilha de trabalho, o método falhará. Use o método Activate para ativar uma planilha antes de usar a propriedade Range sem um qualificador de objeto explícito.

```
Worksheets("sheet1").Activate  
Range("A1:H8").Formula = "=rand()"      'Range is on the active sheet
```

O exemplo seguinte limpa o conteúdo do intervalo chamado "Criteria".

```
Worksheets(1).Range("criteria").ClearContents
```

Se você usar um argumento de texto para o endereço do intervalo, você terá que especificar o endereço em notação de estilo A1 (você não poderá usar a notação de estilo L1C1).

Propriedade Cells

Use Cells(linha, coluna) onde linha é o índice da linha e coluna é o índice da coluna, para retornar uma única célula. O exemplo seguinte define o valor da célula A1 como 24.

```
Worksheets(1).Cells(1, 1).Value = 24
```

O exemplo seguinte define a fórmula para a célula A2.

```
ActiveSheet.Cells(2, 1).Formula = "=sum(B1:B5)"
```

Embora você também possa usar Range("A1") para retornar a célula A1, pode haver ocasiões em que a propriedade Cells seja mais conveniente porque você pode usar uma variável para a linha ou coluna. O exemplo seguinte cria cabeçalhos de coluna e linha em Sheet1. Observe que após a planilha ser ativada, a propriedade Cells pode ser usada sem uma declaração explícita de planilha (ela retorna uma célula da planilha ativa).

```
Sub SetUpTable()  
    Worksheets("sheet1").Activate  
    For theYear = 1 To 5  
        Cells(1, theYear + 1).Value = 1990 + theYear  
    Next theYear  
  
    For theQuarter = 1 To 4  
        Cells(theQuarter + 1, 1).Value = "Q" & theQuarter  
    Next theQuarter  
  
End Sub
```

Apesar de você poder usar funções de cadeia de caracteres do Visual Basic para alterar as referências de estilo A1, é muito mais fácil (e é uma prática de programação muito melhor) usar a notação Cells(1, 1).

Use expressão.Cells(linha, coluna), onde expressão é uma expressão que retorne um objeto Range, e linha e coluna são relativas ao canto superior esquerdo do intervalo, para retornar parte de um intervalo. O exemplo seguinte define a fórmula para a célula C5.

```
Worksheets(1).Range("C5:C10").Cells(1, 1).Formula = "=rand()"
```

Range e Cells

Use Range(célula1, célula2), onde célula1 e célula2 são objetos Range que especificam as células inicial e final, para retornar um objeto Range. O exemplo seguinte define o estilo da linha da borda das células 1:J10.

```
With Worksheets(1)
    .Range(.Cells(1, 1), _
        .Cells(10, 10)).Borders.LineStyle = xlThick
End With
```

Observe o ponto na frente de cada ocorrência da propriedade Cells. O ponto será obrigatório se o resultado da instrução With anterior for aplicado à propriedade Cells — nesse caso, para indicar que as células estão na planilha um (sem o ponto, a propriedade Cells retornaria as células da planilha ativa).

Propriedade Offset

Use Offset(linha, coluna), onde linha e coluna são os deslocamentos de linha e coluna, para retornar um intervalo em um deslocamento especificado de um outro intervalo. O exemplo seguinte seleciona a célula três linhas abaixo e uma coluna à esquerda da célula do canto superior esquerdo da seleção atual. Você não pode selecionar uma célula que não esteja na planilha ativa, portanto, você precisa ativar primeiro a planilha.

```
Worksheets("sheet1").Activate
'can't select unless the sheet is active
Selection.Offset(3, 1).Range("A1").Select
Método Union
```

Use Union(intervalo1, intervalo2, ...) para retornar intervalos de várias áreas — isto é, intervalos compostos de dois ou mais blocos contíguos de células. O exemplo seguinte cria um objeto definido como a união de intervalos A1:B2 e C3:D4 e, em seguida, seleciona o intervalo definido.

```
Dim r1 As Range, r2 As Range, myMultiAreaRange As Range
Worksheets("sheet1").Activate
Set r1 = Range("A1:B2")
Set r2 = Range("C3:D4")
Set myMultiAreaRange = Union(r1, r2)
myMultiAreaRange.Select
```

O objetivo desta lição foi apresentar os principais objetos do Modelo de Objetos do Excel:

- ▶ Application
- ▶ Workbook
- ▶ Worksheet
- ▶ Range

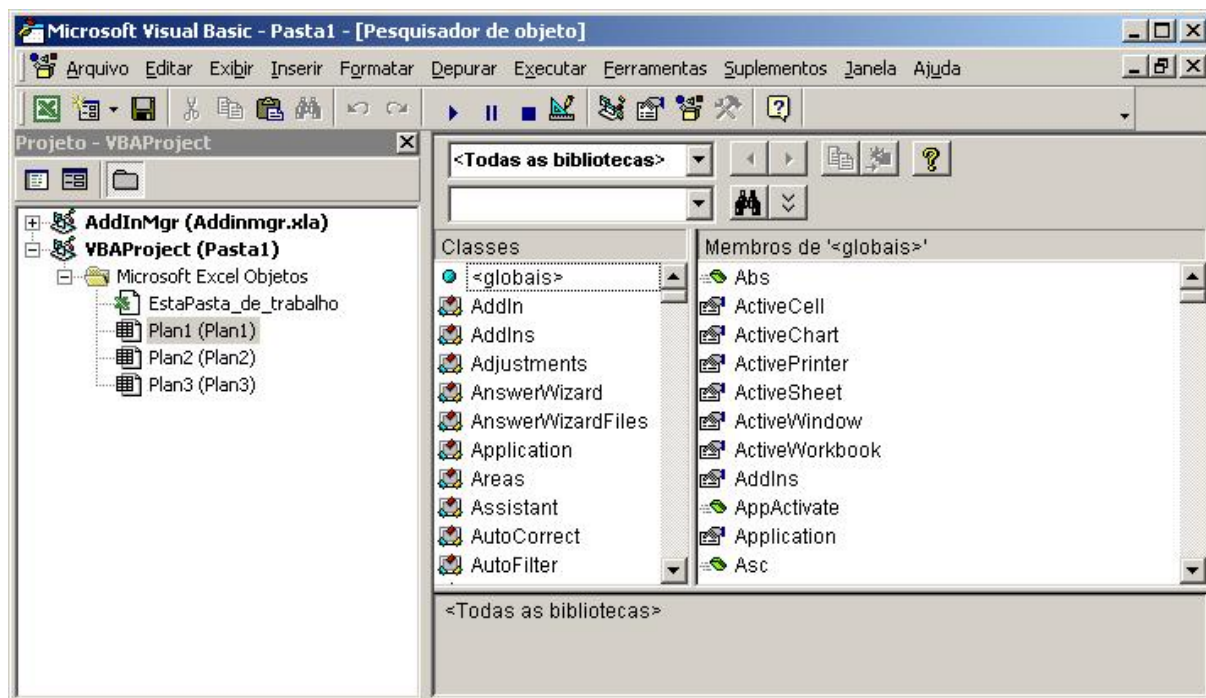
Além da apresentação dos objetos, coloquei pequenos exemplos de código, retirados da Ajuda do Excel. Não se preocupe em entender todos estes métodos e propriedades utilizados nos exemplos, neste momento. O nosso trabalho de agora em diante, será estudar, em detalhes e exemplificar o uso destes objetos. Nas próximas lições você irá estudar cada um destes objetos, tendo acesso a dezenas de exemplos práticos, explicados em detalhes, com o objetivo de ilustrar os diversos métodos, propriedades e coleções de cada objeto. Então, mãos à obra.

Lição 03: O Microsoft Object-Browser: Navegando pela Hierarquia de Objetos

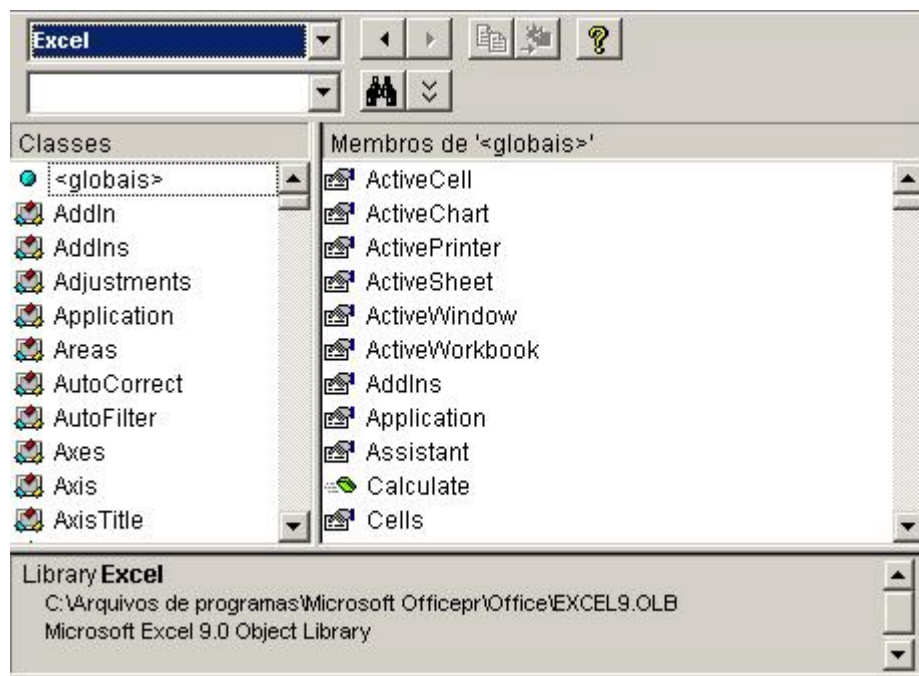
Nesta lição mostrarei como utilizar o recurso Object Browser. Este recurso está disponível no ambiente de programação do VBA. Utilizando o Object Browser é possível selecionar uma biblioteca para que sejam exibidos somente os objetos da biblioteca selecionada. Em seguida você pode clicar em um dos objetos disponíveis. Serão exibidos todos os métodos e propriedades do referido objeto. Em seguida você pode selecionar um método e/ou propriedade e pressionar a tecla F1 para obter ajuda sobre o método e/ou propriedade selecionado. Em resumo, o Object Browser é uma maneira de navegar através das bibliotecas de programação disponíveis, bem como através dos objetos disponíveis em cada biblioteca e dos métodos e propriedades de cada objeto. A seguir mostrarei um exemplo prático de utilização do Object Browser.



Exemplo: Utilizando o Object Browser.

1. Abra o Excel.
2. Pressione Alt+F11 para exibir o editor de Código do VBA.
3. Selecione o comando Exibir -> Pesquisador de Objeto.
4. Será exibido o Pesquisador de Objeto (Object Browser), conforme indicado na Figura a seguir:



5. Observe a lista na parte de cima da janela, onde está selecionada a opção <Todas as bibliotecas>. Você pode abrir esta lista e selecionar apenas uma determinada biblioteca, para exibir somente os objetos da biblioteca selecionada. No exemplo da figura a seguir, selecionei a biblioteca Excel. Observe que são exibidos os diversos objetos da Biblioteca do Excel. Observe que um dos primeiros objetos da lista é o objeto Application, justamente o objeto que iremos estudar nas próximas lições.



6. Clique no objeto Application para selecioná-lo e observe. No painel da direita serão exibidos todos os métodos (indicados por um ) e propriedades (indicadas por um ) do objeto Application.
7. Clique em uma das propriedades para selecioná-la, por exemplo, clique na propriedade ActivePrinter para selecioná-la.
8. Pressione a tecla F1. Em poucos instantes será exibida uma tela de ajuda sobre a propriedade ActivePrinter, do objeto Application, da biblioteca de objetos do Excel.
9. Para fechar o Pesquisador de Objeto clique no botão fechar (x) da janela do Pesquisador de objeto. É o botão x mais de baixo, pois o bem de cima é o botão Fechar do Editor do VBA.

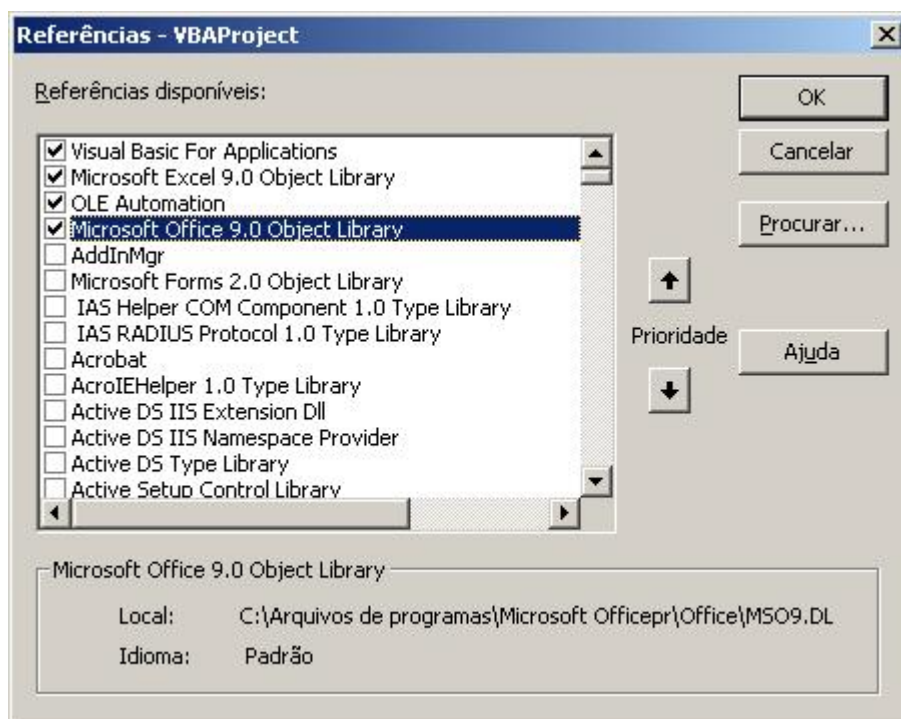
Muito bem, com este pequeno exemplo, já é possível ver que o Pesquisador de Objeto é uma ferramenta bastante útil, principalmente para listar os métodos e propriedades de um objeto e para obter ajuda, rapidamente, sobre um método ou propriedade selecionado.

Fazendo referência a Bibliotecas de Objetos:

A medida que você instala programas no seu computador, novas bibliotecas de programação passam a estar disponíveis para uso na programação VBA no Excel. Por exemplo, ao instalar o Access, novas bibliotecas de programação, de acesso à dados, passam a estar disponíveis para uso no VBA no Excel. Para que os objetos de uma biblioteca possam ser utilizados pelo Excel, deve ser feita uma referência, dentro do módulo de Código, a biblioteca desejada. A referência é feita usando o comando Ferramentas -> Referências..., do ambiente de programação do VBA. Por padrão, já é feita uma referência automática às principais bibliotecas de programação, como por exemplo à biblioteca Microsoft Excel 9.0 Object Library. A seguir mostro um exemplo prático de como fazer referência a uma determinada biblioteca, para poder utilizar no código VBA, os objetos da referida biblioteca. As referências são individuais em cada arquivo .xls, ou seja, em cada arquivo .xls, você deve fazer referência às bibliotecas que serão utilizadas nos módulos VBA do arquivo.

Exemplo: Fazendo referência às bibliotecas de programação

1. Abra o Excel.
2. Pressione Alt+F11 para exibir o editor de Código do VBA.
3. Selecione o comando Ferramentas -> Referências
4. Será exibida a janela Referências – VBAProject, na qual são exibidas as bibliotecas disponíveis. Em primeiro lugar são exibidas as bibliotecas já referenciadas, em ordem alfabética. Em seguida as demais bibliotecas disponíveis, ainda não referenciadas, em ordem alfabética, conforme indicado na Figura a seguir:



5. Para fazer referência a uma nova biblioteca basta localizá-la na lista e marcar a caixa de seleção ao lado do nome da biblioteca. Em seguida clique em OK. Pronto, foi feita a referência a biblioteca selecionada e você poderá utilizar os objetos da biblioteca selecionada, no código VBA.

Lição 04: Objeto Application – O Pai de Todos - Introdução

Nesta lição vamos iniciar o estudo detalhado, dos principais objetos do Modelo de Objetos do Excel. Iniciaremos este estudo pelo objeto Application, o qual, conforme já descrito anteriormente, representa o próprio Excel.

O objeto Application tem, ao todo, 218 propriedades e métodos. Quando estamos trabalhando dentro de uma planilha do Excel, não é preciso a criação explícita de um objeto Application. Por exemplo, se dentro do código, quisermos fazer referência a uma determinada célula, não precisamos criar um objeto Application, depois um objeto Workbook, depois um objeto Worksheet para, finalmente, poder acessar as células de uma planilha usando um objeto Range. Ao invés disso, podemos fazer referência direta a célula desejada. Com isso o Excel supõe que estamos trabalhando com a instância atual do Excel, dentro da pasta de trabalho atual (arquivo .xls), dentro de uma das suas planilhas, o que é bastante razoável. Apenas teríamos que criar toda essa hierarquia de objetos, se quiséssemos fazer referência a uma célula de uma planilha de uma pasta de trabalho externa.

Nessa lição farei uma introdução ao objeto Application. Nas próximas lições estudaremos, em detalhes, os métodos e propriedades do objeto Application.

O objeto Application:

Representa o aplicativo Microsoft Excel. O objeto Application contém:

- ▶ Definições e opções para o aplicativo como um todo (muitas das opções da caixa de diálogo Opções (menu Ferramentas), por exemplo, são configuradas através das propriedades do objeto Application).
- ▶ Métodos que retornem objetos do nível mais alto, como ActiveCell, ActiveSheet e assim por diante.

Usar o objeto Application: Usamos o objeto Application quando temos que configurar alguma opção do Excel, como as opções disponíveis no menu Ferramentas -> Opções, ou quando temos que acessar dados de uma planilha externa. Nesse último caso criamos um objeto Application. Em seguida, criamos um objeto Workbook associado com o arquivo .xls do qual queremos acessar dados. Em seguida usamos a coleção Worksheets e o objeto Range para acessar os dados do referido arquivo. Com isso é possível, a partir de uma planilha do Excel, fazer cálculos que envolvem dados de diversos arquivos .xls diferentes.

Vamos considerar alguns exemplos de código que usa o objeto Application.

Criar um objeto Application e usar o método Open para abrir uma planilha: No exemplo de código a seguir, temos o uso do objeto Application para abrir um arquivo do Excel que está gravado no disco rígido:

```
Set xl = CreateObject("Excel.Sheet")  
xl.Application.Workbooks.Open "C:\ExcelAvançado\ExApp.xls"
```

Nesse exemplo usamos a função CreateObject para criar um objeto do tipo Excel.Sheet, que na prática é uma planilha do Excel:

```
Set xl = CreateObject("Excel.Sheet")
```

Em seguida usamos o método Open, da coleção Workbook do objeto Application, para acessar a planilha C:\ExcelAvançado\ExApp.xls.

Exibindo uma caixa para que o usuário selecione a planilha a ser aberta: Nesse exemplo vamos apresentar um trecho de código, o qual exibe uma janela para que o usuário selecione o arquivo a ser aberto. Uma vez feita a seleção, a planilha é aberta, usando o método Open, usado no exemplo anterior.

```
Set xl = CreateObject("Excel.Sheet")
```

```
ArqParaAbrir = Application.GetOpenFilename("Planilhas do Excel (*.xls), *.xls")
```

```
If ArqParaAbrir <> False Then
```

```
    MsgBox "A seguinte planilha será carregada: " & ArqParaAbrir
```

```
    xl.Application.Workbooks.Open ArqParaAbrir
```

```
End If
```

Vamos comentar, em detalhes, o exemplo anterior.

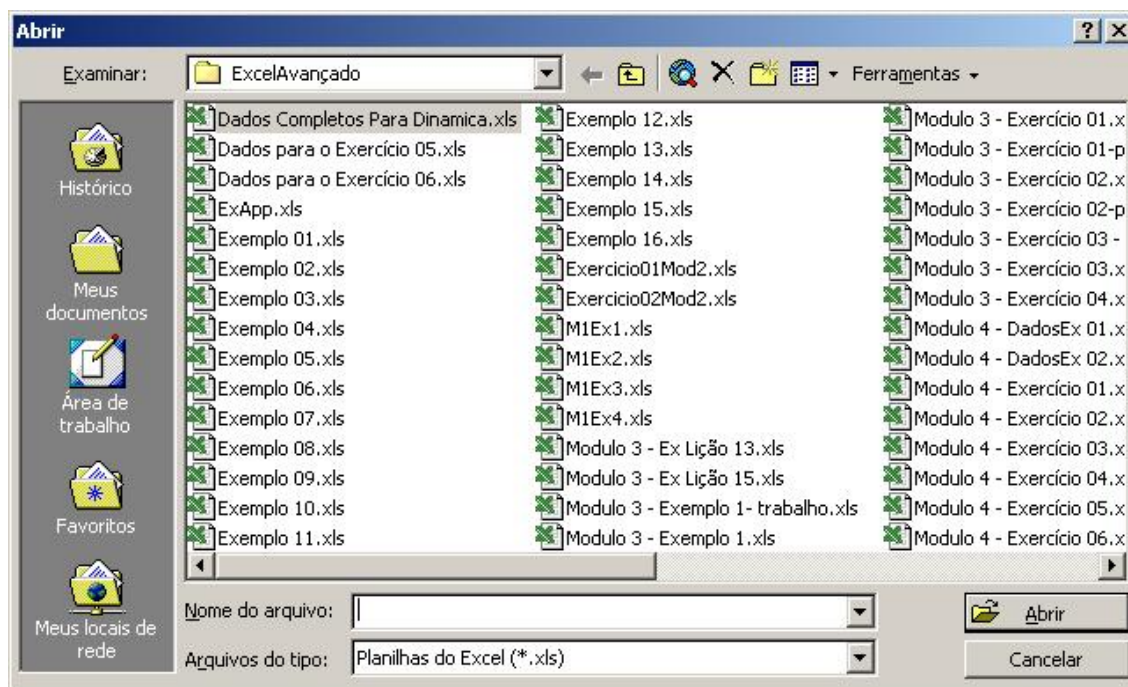
1. Usamos a função CreateObject para criar um objeto do tipo planilha do Excel, objeto esse que é associado com a variável xl:

```
Set xl = CreateObject("Excel.Sheet")
```

2. Agora utilizamos o método GetOpenFilename, do objeto Application. Esse método exibe a caixa de diálogo Abrir (a mesma caixa que é exibida quando você seleciona o comando Arquivo -> Abrir). O nome do arquivo selecionado nessa janela, será atribuído à variável ArqParaAbrir. Será atribuído o caminho completo, por exemplo C:\ExcelAvançado\Teste.xls.

```
ArqParaAbrir = Application.GetOpenFilename("Planilhas do Excel (*.xls), *.xls")
```

Quando essa linha for executada, será exibida a janela a seguir:



Observe que são exibidos apenas os arquivos .xls da pasta de trabalho atual. No exemplo da figura são exibidos os arquivos da pasta C:\ExcelAvançado, porque recém abriu um arquivo dessa pasta. Normalmente são exibidos, por padrão, os arquivos da pasta Meus documentos. Somente são exibidos os arquivos .xls, porque definimos esse parâmetro na chamada do método `GetOpenFilename("Planilhas do Excel (*.xls), *.xls")`

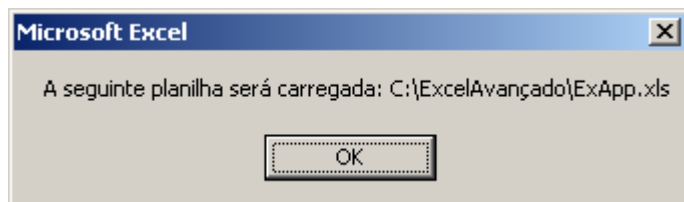
3. Você seleciona um arquivo e clica no botão Abrir. Em seguida o código testa se realmente algum arquivo foi selecionado:

```
If ArqParaAbrir <> False Then
```

4. Caso algum arquivo tenha sido selecionado, o nome do arquivo será exibido:

```
MsgBox "A seguinte planilha será carregada: " & ArqParaAbrir
```

conforme indicado na figura a seguir:



5. e o arquivo selecionado será aberto no Excel.

Existem dezenas de métodos e propriedades do objeto Application. Você encontra exemplos de cada método e propriedade na Ajuda do Excel. Nas próximas lições faremos um estudo detalhado dos métodos e propriedades do objeto Application.

Lição 05: Objeto Application – Como Declarar e Utilizar

Nesta lição mostrarei como declarar uma variável como sendo do tipo Application e como inicializá-la, usando código VBA.

Vou demonstrar a declaração e utilização do objeto Application, através de exemplos práticos.

Exemplo: Considere o trecho de código a seguir:

```
Public Sub TesteApp()  
  
    Dim App As Application  
    Set App = Excel.Application  
  
    'Exibe algumas propriedades do objeto Application.  
  
    MsgBox "Impressora padrão: " & App.ActivePrinter  
    MsgBox "Versão do Excel: " & App.Build  
    MsgBox "Onde está instalado o Excel: " & App.Path  
  
End Sub
```

Comentários do código do exemplo:

1) Neste exemplo, inicialmente declaro uma variável chamada App, como sendo do tipo Application:

```
Dim App As Application
```

2) Inicializo esta variável, apontando-a para a instância atual do Excel. Isso é feito usando o método Application, do objeto Excel:

```
Set App = Excel.Application
```

3) Utilizo a propriedade ActivePrinter, para exibir o nome da impressora configurada como Impressora padrão, ou seja, impressora ativa:

```
MsgBox "Impressora padrão: " & App.ActivePrinter
```

Esta linha de código, produzirá a mensagem indicada no exemplo da Figura a seguir:



4) Utilizo a propriedade Build, para exibir o número de compilação, ou seja, o número oficial de Versão do Excel:

```
MsgBox "Versão do Excel: " & App.Build
```

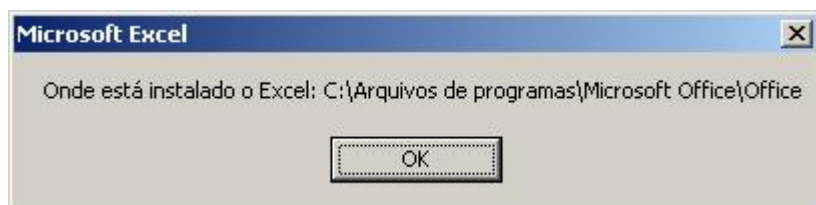
Esta linha de código, produzirá a mensagem indicada no exemplo da Figura a seguir:



5) Utilizo a propriedade Path, para exibir o caminho completo, onde está instalado o executável do Excel:

```
MsgBox "Onde está instalado o Excel: " & App.Path
```

Esta linha de código, produzirá a mensagem indicada no exemplo da Figura a seguir:



Um detalhe importante a observar é que o objeto Excel, aponta para a instância atual do Excel. Em outras palavras, o objeto Excel, representa o objeto Application, associado com a instância do Excel atualmente carregada na memória. Isso pode ser comprovado, pelo fato de o objeto Excel apresentar as mesmas propriedades e os mesmos métodos do objeto Application, ou seja, ele é, basicamente, uma instância do objeto Application.

Você verá, em detalhes, nas próximas lições, mais detalhes sobre o uso dos métodos e propriedades do objeto Application. A seguir mais um pequeno exemplo, no qual utilizo algumas coleções do objeto Application, para exibir os nomes das planilhas da pasta de trabalho atual.

Exemplo: Considere o trecho de código a seguir:

```
Dim wks As Worksheet
Dim msg As String

' Percorro a coleção Workbooks do objeto Application.
' Vou concatenando o nome das planilhas em uma variável do
' tipo String.
' No final uso MsgBox para exibir o conteúdo da variável

msg = "PLANILHAS DA PASTA DE TRABALHO ATUAL:" & Chr(13) & Chr(10)

For Each wks In Excel.Application.ActiveWorkbook.Worksheets

    msg = msg & wks.Name & Chr(13) & Chr(10)
Next
```


`MsgBox mensg`

Comentários do código do exemplo: Este código tem uma série de técnicas interessantes, as quais detalharei nos comentários a seguir

1) Neste exemplo, inicialmente declaro uma variável chamada `wks`, como sendo do tipo `Worksheet` e uma variável `mensg` como sendo do tipo `String`. A variável `Worksheet` será utilizada para fazer referência a cada uma das planilhas, da coleção de planilhas do objeto `Workbook` (lembre que o objeto `Workbook` representa a pasta de trabalho e que dentro de uma pasta de trabalho, pode haver uma ou mais planilhas, as quais são acessadas através da coleção `Worksheets`):

```
Dim wks As Worksheet
Dim mensg As String
```

2) Inicializo a variável `mensg`. Um detalhe interessante a observar é o uso das funções `Chr(13)` e `Chr(10)`.

```
mensg = "PLANILHAS DA PASTA DE TRABALHO ATUAL:" & Chr(13) & Chr(10)
```

O operador `&` é utilizado para concatenar as diferentes partes que irão compor o valor de uma variável do tipo `String`. No exemplo desta linha de código, está sendo concatenado o texto “PLANILHAS DA PASTA DE TRABALHO ATUAL”, com o caractere de nova linha – `Chr(13)` mais o caractere de alimentação de carro `Chr(10)`. O efeito prático da concatenação de `Chr(13) & Chr(10)` é inserir uma quebra de linha na mensagem.

3) A parte final do código é um laço `For Each`, o qual percorre todas as planilhas da coleção de planilhas, da pasta de trabalho atual.

Inicialmente é interessante observar como se fez o acesso à coleção de planilhas da pasta de trabalho atual:

```
For Each wks In Excel.Application.ActiveWorkbook.Worksheets
```

Esta linha de código mostra claramente como é o uso da hierarquia de objetos do Modelo de Objetos do Excel. Lendo esta linha de código de trás para frente, começamos com a coleção `Worksheets`, da pasta de trabalho atual (a qual é acessada usando a propriedade `ActiveWorkbook` do objeto `Application`). Ao utilizar `Excel.Application`, estou fazendo referência a instância do Excel carregada na memória. Em seguida, ao usar a propriedade `ActiveWorkbook`, faço referência a pasta de trabalho atual, ou seja, a pasta de trabalho na qual o código VBA está sendo executado. Por último, uso a propriedade `Worksheets`, para retornar a coleção de planilhas da pasta de trabalho atual. Quero novamente salientar a estrutura hierárquica na utilização dos objetos, do Modelo de objetos do Excel.

Dentro da estrutura `For...Each`, vou anexando o nome de cada planilha da coleção de planilhas, à variável `mensg`. Além do nome da planilha, anexo uma quebra de linha, para exibir o nome de cada planilha em uma linha separada. A linha de código que faz este trabalho está indicada a seguir:

```
msg = msg & wks.Name & Chr(13) & Chr(10)
```

4) O passo final é exibir o valor da variável msg, usando a função MsgBox, conforme código a seguir:

```
MsgBox msg
```

O resultado da execução deste comando, está exemplificado na Figura a seguir, onde temos uma pasta de trabalho com quatro planilhas: Vendas, Produtos, Fornecedores e Clientes.



Nas próximas lições continuaremos o estudo dos métodos e propriedades do objeto Application.

Lição 06: Objeto Application – Propriedades que Retornam Objetos Filho

Nesta lição vamos iniciar um estudo detalhado, dos principais métodos e propriedades do objeto Application. Faremos este estudo utilizando pequenos trechos de código, os quais ilustram a utilização dos métodos e propriedades do objeto Application.

Retornando filhos do objeto Application:

Quando trabalhamos com uma Hierarquia de objetos, que é o caso do Modelo de Objetos do Excel, existe o conceito de objetos Pai e objetos Filho. Em um dos títulos das lições anteriores aditivei o objeto Application como sendo o Pai de Todos, ou seja, não existe um objeto acima do objeto Application, na hierarquia de objetos do Excel. Dentro de um objeto application, pode haver um ou mais objetos do tipo Workbook, os quais são acessados através da coleção Workbooks. Dizemos que cada objeto Workbook (pasta de trabalho) é filho do objeto Application. Em cada objeto Workbook, podem haver uma ou mais planilhas, as quais são acessadas através da coleção Worksheets. Dizemos que cada objeto Worksheet é filho do objeto Workbook, ao qual pertence a planilha.

Muitos dos métodos e propriedades do objeto Application, são utilizados para retornar objetos filhos. Por exemplo, a coleção Workbooks, a propriedade ActiveWorkbook e assim por diante. Nesta lição apresentarei alguns destes métodos e propriedades, os quais retornam objetos filhos do objeto Application.

Retornando elementos ativos: Existe uma série de propriedades que retornam elementos ativos, como por exemplo a célula ativa, a pasta de trabalho ativa ou a planilha atualmente ativa. Estas propriedades estão descritas a seguir:

1) Propriedade ActiveCell: Retorna um objeto Range representando a célula ativa da janela ativa (a janela visível) ou da janela especificada. Se a janela não estiver exibindo uma planilha, essa propriedade falhará. Somente leitura. Quando você não especifica um qualificador de objeto, essa propriedade retorna a célula ativa da janela ativa.

Tenha cuidado de distinguir entre célula ativa e seleção. A célula ativa é uma única célula dentro da seleção atual. A seleção pode conter mais de uma célula, mas somente uma é a célula ativa. De uma maneira simples, a célula ativa é a célula onde está o cursor.

Todos os comandos a seguir retornam a célula ativa, sendo todas equivalentes.

```
ActiveCell  
Application.ActiveCell  
ActiveWindow.ActiveCell  
Application.ActiveWindow.ActiveCell
```

2) Propriedade ActiveChart: Retorna um objeto Chart representando o gráfico ativo (seja um gráfico incorporado ou uma folha de gráfico). Um gráfico incorporado é considerado ativo quando está selecionado ou ativado. Quando nenhum gráfico está ativo, essa propriedade retorna Nothing. É Somente leitura, ou seja, não pode ser utilizada para ativar/selecionar um gráfico

Se você não especificar um qualificador de objeto, essa propriedade retornará o gráfico ativo da pasta de trabalho ativa.

O exemplo a seguir ativa a legenda do gráfico ativo:

```
ActiveChart.HasLegend = True
```

3) Propriedade ActiveSheet: Retorna um objeto representando a planilha ativa (a planilha visível) da pasta de trabalho ativa ou na janela ou pasta de trabalho especificada. Retorna Nothing se não houver planilha ativa. Somente leitura. Se você não especificar um qualificador de objeto, essa propriedade retornará a planilha ativa da pasta de trabalho ativa.

Se uma pasta de trabalho aparece em mais de uma janela, a propriedade ActiveSheet poderá ser diferente em janelas diferentes, dependendo da pasta de trabalho onde o código VBA estiver sendo executado.

Este exemplo exibe o nome da planilha ativa:

```
MsgBox "O nome da Planilha Ativa é: " & ActiveSheet.Name
```

Você poderia usar a sintaxe completa, como no exemplo a seguir:

```
MsgBox "O nome da Planilha Ativa é: " & Excel.Application.ActiveSheet.Name
```

Porém, ao usar a sintaxe abreviada ActiveSheet.Name, o Excel entende que deve fazer referência a planilha atual (ActiveSheet), da pasta de trabalho onde o código está sendo executado, pasta esta que está em execução no contexto do Excel (Excel.Application).

4) Propriedade ActiveWindow: Esta propriedade retorna um objeto Window representando a janela ativa (a janela visível). Somente leitura. Retorna Nothing se nenhuma janela estiver aberta.

Este exemplo exibe o nome (propriedade Caption) da janela ativa:

```
MsgBox "O nome da janela ativa é: " & ActiveWindow.Caption
```

5) Propriedade ActiveWorkbook: Retorna um objeto Workbook representando a pasta de trabalho da janela ativa (a janela visível). Somente leitura. Retorna Nothing se não houver janelas abertas ou se a janela ativa for a janela Informações ou a janela Área de transferência.

Este exemplo exibe o nome da pasta de trabalho ativa.

```
MsgBox "O nome da Pasta de Trabalho ativa é: " & ActiveWorkbook.Name
```

6) Propriedade Rows: Esta propriedade retorna um objeto Range (o qual representa uma faixa de células e será estudado, em detalhes, no Módulo 4), representando todas as linhas da planilha ativa. Se o documento ativo não for uma planilha, a propriedade Rows falhará. É Somente leitura.

O uso dessa propriedade sem um qualificador de objeto equivale a usar `ActiveSheet.Rows`.

Exemplos de uso da propriedade `Rows`:

Este exemplo exclui a linha três na planilha `Plan1`.

```
Worksheets("Sheet1").Rows(3).Delete
```

O exemplo a seguir exclui linhas na região atual da planilha um onde o valor na célula um (célula da primeira coluna da linha) da linha for o mesmo que o valor na célula um da linha anterior.

```
For Each rw In Worksheets(1).Cells(1, 1).CurrentRegion.Rows
    this = rw.Cells(1, 1).Value
    If this = last Then rw.Delete
    last = this
Next
```

Este exemplo exibe o número de linhas na seleção na `Plan1`. Se mais de uma área for selecionada, o exemplo fará um loop por todas elas.

```
Worksheets("Plan1").Activate
areaCount = Selection.Areas.Count

If areaCount <= 1 Then
    MsgBox "A seleção contém: " & _
        Selection.Rows.Count & " linhas."
Else
    i = 1
    For Each a In Selection.Areas
        MsgBox "Area " & i & " da seleção contém: " & a.Rows.Count & " rows."
        i = i + 1
    Next a
End If
```

Comentários sobre este último exemplo:

- ▶ Inicialmente é ativada a planilha chamada `Plan1`. Isso é feito com o seguinte comando:

```
Worksheets("Plan1").Activate
```

- ▶ O próximo passo é armazenar o número de áreas selecionadas na variável `areaCount`. O número de áreas selecionadas é obtido com o uso da propriedade `Count`, da coleção `Areas`, do objeto `Selection` (observe a leitura de trás para frente, conforme o modelo hierárquico do Modelo de Objetos do Excel). Isso é feito com o comando a seguir:

```
areaCount = Selection.Areas.Count
```

- ▶ O restante do código é um teste `if`. Primeiro o código testa se o número de áreas é menor ou igual a 1. Neste caso não teremos nenhuma área selecionada ou teremos uma única área. Se for este o caso, é exibida uma mensagem que informa o número de

linhas da seleção. O número de linhas é obtido com o uso da propriedade Count, da coleção Rows, do objeto Selection, conforme trecho de código a seguir:

```
If areaCount <= 1 Then
    MsgBox "A seleção contém: " & _
        Selection.Rows.Count & " linhas."
```

- A parte Else do teste é executada se existirem duas ou mais áreas selecionadas. Neste caso será exibida uma mensagem, informando o número de linhas de cada seleção. Isso é feito, novamente, com uso da propriedade Count, da coleção Rows, conforme trecho de código a seguir:

```
Else
    i = 1
    For Each a In Selection.Areas
        MsgBox "Area " & i & " da seleção contém: " & a.Rows.Count & " rows."
        i = i + 1
    Next a
End If
```

Nesta lição você aprendeu sobre as principais propriedades do objeto Application que retornam objetos filho. Na próxima lição vou falar sobre as propriedades e métodos do objeto Application, utilizados para habilitar/desabilitar recursos do Excel.

Lição 07: Objeto Application – Exibindo/Ocultando itens do Excel

Nesta lição você aprenderá a utilizar alguns métodos e propriedades os quais são utilizados para alterar a visualização das planilhas do Excel, tais como a propriedade DisplayAlerts, que suprime as mensagens de aviso e a propriedade DisplayFormulaBar, a qual é utilizada para exibir ou ocultar a barra de fórmulas.

1) Propriedade DisplayAlerts: Esta propriedade retorna True se o Microsoft Excel estiver habilitado para exibir certos alertas e mensagens enquanto uma macro estiver sendo executada. É do tipo Boolean (True/False) e é do tipo leitura e gravação, ou seja, podemos ler o valor contido nesta propriedade, como também podemos definir o valor desta propriedade.

O valor padrão é True. Defina essa propriedade como False se você não quiser ser “*incomodado*” por avisos e mensagens de alerta enquanto uma macro estiver sendo executada; sempre que uma mensagem requisitar uma resposta, o Microsoft Excel escolherá a resposta padrão. Na prática, definir esta propriedade como False, evita que sejam exibidas diversas mensagens de confirmação, como por exemplo, quando você está importando dados de um arquivo de texto para dentro de uma planilha do Excel.

Se você definir essa propriedade como False, o Microsoft Excel não a definirá automaticamente de volta para True quando sua macro terminar a execução. Sua macro deverá sempre definir a propriedade de volta para True antes de terminar a execução.

Exemplo da propriedade DisplayAlerts

Este exemplo fecha a pasta de trabalho Book1.xls, usando o método Close do objeto Workbook (objeto este que você estudará em detalhes no próximo módulo) e não pede ao usuário que salve alterações. Quaisquer alterações feitas em Book1.xls não serão salvas. O aviso para salvar a planilha não será exibido porque a propriedade DisplayAlerts foi definida com o valor False.

```
Application.DisplayAlerts = False  
Workbooks("BOOK1.XLS").Close  
Application.DisplayAlerts = True
```

2) Propriedade DisplayFormulaBar: Esta propriedade define se a Barra de fórmulas será ou não exibida. Se o valor da propriedade for True, a Barra de fórmulas será exibida, caso contrário, a Barra de fórmulas não será exibida. Esta propriedade é de leitura e gravação, ou seja, posso ler o valor da propriedade e também posso defini-lo.

No exemplo a seguir, estou definindo o valor da propriedade em False, o que fará com que a Barra de fórmulas deixe de ser exibida:

```
Application.DisplayFormulaBar = False
```

No exemplo a seguir exibo uma mensagem diferente, dependendo do valor da propriedade DisplayFormulaBar:

```
If (Application.DisplayFormulaBar) Then
    MsgBox "A Barra de Fórmulas está sendo exibida"
Else
    MsgBox "A Barra de Fórmulas não está sendo exibida"
End If
```

3) Propriedade DisplayFullScreen: Esta propriedade terá o valor True se o Microsoft Excel está em modo de tela inteira (equivalente ao comando Exibir -> Tela inteira). É do tipo Boolean (True ou False), de leitura e gravação.

O modo de tela inteira maximiza a janela do aplicativo de modo que ela preencha toda a tela, ocultando a barra de título do aplicativo. As barras de ferramentas, a barra de status e a barra de fórmulas mantêm definições de exibição separadas para modo de tela inteira e modo normal.

A linha a seguir faz o Microsoft Excel ser exibido em modo de tela inteira.

```
Application.DisplayFullScreen = True
```

4) Propriedade DisplayGridlines: Esta propriedade tem o valor True se as linhas de grade estiverem sendo exibidas. É do tipo Boolean (True e False), do tipo leitura e gravação.

Esta propriedade só se aplica a planilhas e folhas de macro. Esta propriedade só afeta as linhas de grade exibidas. Use a propriedade PrintGridlines para controlar a impressão das linhas de grade.

O exemplo a seguir alterna a exibição das linhas de grade da janela ativa de Pasta1.xls.

```
Workbooks("BOOK1.XLS").Worksheets("Sheet1").Activate
ActiveWindow.DisplayGridlines = Not(ActiveWindow.DisplayGridlines)
```

Se as linhas de grade estiverem sendo exibidas, serão ocultas, se estiverem ocultas, serão exibidas. O operador Not inverte o valor lógico da propriedade:

```
Not(ActiveWindow.DisplayGridlines)
```

5) Propriedade DisplayWorkbookTabs: Esta propriedade é utilizada para exibir ou ocultar as guias, na parte de baixo da tela, onde são exibidos os nomes das diversas planilhas que compõem a pasta de trabalho. Por padrão, ao criar uma nova pasta de trabalho, estão disponíveis as guias Plan1, Plan2 e Plan3.

Esta propriedade tem o valor True se as guias da pasta de trabalho estiverem sendo exibidas e False, caso contrário. É do tipo Boolean e de leitura e gravação.

Este exemplo ativa as guias da pasta de trabalho.

```
ActiveWindow.DisplayWorkbookTabs = True
```


6) Propriedade DisplayStatusBar: Esta propriedade é utilizada para exibir ou ocultar a Barra de Status, na parte de baixa da janela do Excel. Esta propriedade tem o valor True se a Barra de Status estiver sendo exibida e False, caso contrário. É do tipo Boolean e de leitura e gravação.

O exemplo a seguir salva o estado atual da propriedade DisplayStatusBar e, em seguida, define a propriedade True para que a barra de status fique visível.

```
saveStatusBar = Application.DisplayStatusBar  
Application.DisplayStatusBar = True
```

7) Propriedade DisplayScrollBars: Esta propriedade é utilizada para exibir ou ocultar as Barra de rolagem em todas as pastas de trabalho. É do tipo Boolean e de leitura e gravação.

Este exemplo desativa as barras de rolagem para todas as pastas de trabalho.

```
Application.DisplayScrollBars = False
```

Na próxima lição, você aprenderá sobre métodos e propriedades que habilitam/desabilitam recursos do Excel.

Lição 08: Objeto Application – Habilitando/Desabilitando Recursos do Excel

Nesta lição você aprenderá a utilizar alguns métodos e propriedades, utilizados para habilitar e/ou desabilitar recursos do Excel. Por exemplo, existe uma propriedade que habilita/desabilita eventos, outro que habilita/desabilita sons e assim por diante.

1) Propriedade EnableAnimations: Este é o tipo de propriedade que é importante você conhecer, pois ela é utilizada para desabilitar um dos recursos mais “inúteis” do Excel. Se esta propriedade tiver o valor True, significando que as animações estão habilitadas, as linhas ou colunas da planilha, a medida que forem inseridas irão aparecer vagarosamente e a medida que forem excluídas, irão sumindo vagarosamente, dando uma idéia de animação. Em resumo: quando a animação está ativada, as linhas e colunas inseridas na planilha aparecem lentamente; e as linhas e colunas excluídas da planilha desaparecem lentamente. Realmente não consigo visualizar nenhuma aplicação prática para este recurso do Excel. Para desabilitar o recurso de animação, basta definir a propriedade com o valor False. É do tipo Boolean e de leitura e gravação.

O exemplo a seguir desativa a inserção e exclusão animada:

```
Application.EnableAnimations = False
```

2) Propriedade EnableAutoComplete: Esta propriedade é utilizada para habilitar/desabilitar o recurso de AutoCompletar, o qual está habilitado, por padrão. Com este recurso, quando você digita os primeiros caracteres em uma célula, o Excel busca, dentro da mesma coluna, nos valores anteriores, algum que coincida com o que você está digitando e apresenta este valor como sugestão para a célula. Você pode aceitar a sugestão pressionando a tecla Tab para aceitá-la. Este recurso é útil e poupa digitação, principalmente em colunas que contém um número reduzido de valores. Esta propriedade tem o valor True se o recurso estiver habilitado e False, caso contrário. É do tipo Boolean e de leitura e gravação.

Este exemplo ativa o recurso de AutoCompletar.

```
Application.EnableAutoComplete = True
```

3) Propriedade EnableCancelKey: O valor desta propriedade controla como o Microsoft Excel trata interrupções do usuário causadas pelo uso da combinação de teclas CTRL+BREAK (ou ESC ou COMMAND+PONTO) no procedimento sendo executado. É uma propriedade do tipo Long e do tipo leitura e gravação.

O valor desta propriedade pode assumir o valor de uma das seguintes constantes do VBA (constante XlEnableCancelKey):

Constante	Significado
xlDisabled	Interceptação das teclas de cancelamento completamente desativada. Ou seja, se o usuário pressionar Ctrl+Break, o procedimento em execução não será interrompido. Não é aconselhável desabilitar este recurso, pois ele pode ser útil para interromper a execução de um código com problemas.

xlInterrupt	O procedimento atual é interrompido, e o usuário pode depurar ou finalizar o procedimento.
xlErrorHandler	A interrupção é enviada para o procedimento em execução como um erro, podendo ser interceptado por um tratamento de erros configurado com uma instrução On Error GoTo. O código de erro interceptável é 18.

Importante: Use essa propriedade com muito cuidado. Se você usar xlDisabled, não haverá como interromper um loop infinito ou outro código que não termine a si mesmo, a não ser fechando o Excel na marra (Ctrl+Alt+Del) ou reiniciando o computador. Da mesma forma, se você usar xlErrorHandler mas o seu tratamento de erros sempre retornar usando a instrução Resume, não haverá como suspender um código com problemas.

Devido aos perigos descritos no parágrafo anterior é que a propriedade EnableCancelKey é sempre redefinida como xlInterrupt quando o Microsoft Excel volta para o estado normal sem que haja código em execução. Para interceptar ou desativar o cancelamento em seu procedimento, você precisa alterar explicitamente a propriedade EnableCancelKey toda vez que o procedimento for chamado. Esta é uma medida de proteção.

O exemplo a seguir mostra como você pode usar a propriedade EnableCancelKey para configurar um tratamento personalizado de cancelamento.

```
On Error GoTo handleCancel
Application.EnableCancelKey = xlErrorHandler

MsgBox "A execução pode demorar muito tempo: pressione ESC para cancelar"

For x = 1 To 1000000      ' Executar um laço 1.000.000 vezes
    ' Comandos a serem executados um milhão de vezes
Next x

handleCancel:
If Err = 18 Then
    MsgBox "Você cancelou o procedimento"
End If
```

4) Propriedade EnableEvents: Esta propriedade é utilizada para habilitar ou desabilitar o processamento de eventos para o objeto Application. True se os eventos estão ativados e False se os eventos estiverem desativados. Boolean de leitura e gravação.

Nota: Mais adiante você aprenderá mais sobre eventos.

O código do exemplo a seguir desativa eventos antes de um arquivo ser salvo de forma que o evento BeforeSave não ocorra.

```
Application.EnableEvents = False
ActiveWorkbook.Save
Application.EnableEvents = True
```

Lição 09: Objeto Application – Associando Macros à Teclas Especiais

Nesta lição mostrarei como é possível associar uma função e/ou procedimento a uma combinação especial de teclas, como por exemplo Ctrl+Ç. Ou seja, você pode configurar o Excel, para que execute um procedimento ou função, toda vez que o usuário pressionar uma combinação especial de teclas. Também é possível fazer com que seja executada uma função ou procedimento em um tempo programado.

A associação de uma função e /ou procedimento, com uma combinação de teclas é feito usando o método OnKey, do objeto Application, o qual estudaremos logo a seguir.

Método OnKey:

O método OnKey é utilizado para executar um procedimento especificado quando uma determinada tecla ou combinação de teclas é pressionada.

Sintaxe:

```
expressão.OnKey(Key, Procedure)
```

expressão é obrigatória: Uma expressão que retorne um objeto Application.

Key: String obrigatória. Uma sequência de caracteres indicando a tecla a ser pressionada.

Procedure: Variant opcional. Uma sequência de caracteres indicando o nome do procedimento a ser executado. Se Procedure for "" (texto vazio), nada acontecerá quando Key for pressionada (o que, convenhamos, não tem utilidade prática). Essa forma de OnKey altera o resultado normal dos pressionamentos de teclas no Microsoft Excel. Se Procedure for omitido, Key reverterá seu resultado normal no Microsoft Excel, e quaisquer atribuições de teclas especiais feitas com métodos OnKey anteriores serão limpas.

O argumento Key pode especificar qualquer tecla combinada com ALT, CTRL ou SHIFT, ou qualquer combinação dessas teclas. Cada tecla é representada por um ou mais caracteres, como "a" para o caractere a ou "{ENTER}" para a tecla ENTER.

Considere o exemplo a seguir:

Este exemplo atribui o procedimento "ImportaDados" à sequência de teclas CTRL+SINAL DE ADIÇÃO e atribui "ImprimeEspecial" à sequência de teclas SHIFT+CTRL+SETA À DIREITA.

```
Application.OnKey "^{+}", "ImportaDados"  
Application.OnKey "+^{RIGHT}", "ImprimeEspecial"
```

Para especificar caracteres que não são exibidos quando você pressiona a tecla correspondente (ENTER ou TAB, por exemplo), use os códigos listados na tabela seguinte. Cada código da tabela representa uma tecla no teclado.

Tecla	Código
BACKSPACE	{BACKSPACE} ou {BS}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
CLEAR	{CLEAR}
DELETE ou DEL	{DELETE} ou {DEL}
SETA ABAIXO	{DOWN}
END	{END}
ENTER (teclado numérico)	{ENTER}
ENTER	~ (til)
ESC	{ESCAPE} ou {ESC}
HELP	{HELP}
HOME	{HOME}
INS	{INSERT}
SETA À ESQUERDA	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
RETURN	{RETURN}
SETA À DIREITA	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
SETA ACIMA	{UP}
F1 até F15	{F1} até {F15}

Você também pode usar teclas específicas combinadas com SHIFT e/ou CTRL e/ou ALT. Para especificar uma tecla combinada com uma outra tecla ou teclas, use a tabela a seguir.

Para combinar teclas com	Preceda o código da tecla com
SHIFT	+ (sinal de adição)
CTRL	^ (circunflexo)
ALT	% (sinal de porcentagem)

Para atribuir um procedimento a um dos caracteres especiais (+, ^, % e assim por diante), coloque o caractere entre chaves.

Como usar o método OnKey, na prática: O método OnKey, normalmente, é utilizado em um procedimento associado ao evento Open da pasta de trabalho. Nas lições finais deste capítulo você aprenderá mais sobre Eventos. O código associado ao evento Open será executado, automaticamente, quando o evento Open for disparado. Como o próprio nome sugere, o evento Open é disparado sempre que você abre uma pasta de trabalho (arquivo .xls). Neste caso, o evento Open será disparado, o código associado ao evento será executado. Ao incluir diversos comandos OnKey no evento Open, as associações serão feitas sempre que a pasta de trabalho for aberta. Veja os exemplos nas lições finais deste módulo, para mais detalhes sobre o conceito de eventos e sobre os eventos disponíveis no Excel.

Vamos a mais um exemplo:

Este exemplo desativa a sequência de teclas SHIFT+CTRL+SETA À DIREITA.

```
Application.OnKey "+^{RIGHT}", ""
```

O método OnTime:

O método OnTime é utilizado para programar um procedimento para ser executado em um momento especificado no futuro (seja em uma determinada hora do dia ou após uma quantidade específica de tempo decorrido).

Sintaxe:

```
Application.OnTime(EarliestTime, Procedure, LatestTime, Schedule)
```

expressão: obrigatória. Uma expressão que retorne um objeto Application.

EarliestTime: Variant obrigatória. Especifica quando você deseja que esse procedimento seja executado.

Procedure: String obrigatória. O nome do procedimento a ser executado.

LatestTime: Variant opcional. Especifica até quando o procedimento pode ser executado. Por exemplo, se LatestTime estiver definido como EarliestTime + 30 e o Microsoft Excel não estiver em modo Pronto, Copiar, Recortar ou Localizar em EarliestTime devido a um outro procedimento estar sendo executado, o Microsoft Excel irá esperar 30 segundos para que o primeiro procedimento termine. Se o Microsoft Excel não estiver em modo Pronto dentro de 30 segundos, o procedimento

não será executado. Se esse argumento for omitido, o Microsoft Excel esperará até que o procedimento possa ser executado.

Schedule: Variant opcional. True para programar um novo procedimento OnTime. False para limpar um procedimento definido anteriormente. O valor padrão é True.

Use `Now + TimeValue(time)` para programar algo para ser executado quando uma quantidade de tempo específica (contando a partir de agora) tiver decorrido. Use `TimeValue(time)` para programar algo para ser executado em um momento específico.

Considere os exemplos a seguir:

1) Este exemplo executa my_Procedure daqui a 15 segundos:

```
Application.OnTime Now + TimeValue("00:00:15"), "my_Procedure"
```

2) Este exemplo executa my_Procedure às 17:00:

```
Application.OnTime TimeValue("17:00:00"), "my_Procedure"
```

3) Este exemplo cancela a definição de OnTime do exemplo anterior:

```
Application.OnTime EarliestTime:=TimeValue("17:00:00"), _  
Procedure:="my_Procedure", Schedule:=False
```

Lição 10: Objeto Application – Operações com Arquivos

Nesta lição você aprenderá a utilizar alguns métodos e propriedades do objeto Application, para trabalhar com arquivos do Excel. Veremos que existe desde uma propriedade que define a pasta padrão para os arquivos do Excel (pasta que é selecionada, automaticamente, quando você usa os comandos Arquivo -> Abrir, Arquivo -> Salvar ou Arquivo -> Salvar como...), até métodos para exibir uma caixa de diálogo para que o usuário selecione um arquivo e que retorna o nome deste arquivo, juntamente com o caminho completo.

Método GetOpenFileName:

Ao executar o método GetOpenFileName, será exibida a caixa de diálogo Abrir padrão. O usuário navega através das pastas e seleciona um arquivo. O método GetOpenFileName retorna o nome de arquivo do usuário sem realmente abrir nenhum arquivo. Ou seja, na prática, o método GetOpenFileName é utilizado para obter o nome (caminho completo) de um arquivo e retornar este nome, normalmente, para uma variável do tipo texto.

Sintaxe:

Application.GetOpenFilename(FileFilter, FilterIndex, Title, ButtonText, MultiSelect)

FileFilter: Variant opcional. Uma sequência que especifica critérios de filtragem do arquivo.

Esta sequência consiste em pares de sequências de filtro de arquivo seguidas pela especificação de arquivo curinga do MS-DOS, com cada parte e cada par separados por vírgulas. Cada par separado é listado na caixa de listagem suspensa Arquivos do tipo. Por exemplo, a seguinte sequência especifica dois filtros de arquivo—texto e suplemento: "Arquivos de texto (*.txt),*.txt, Arquivos de suplemento (*.xla),*.xla".

Para usar várias expressões curingas do MS-DOS para um único tipo de filtro, separe as expressões curinga com pontos-e-vírgulas; por exemplo, "Arquivos do Visual Basic (*.bas;*.txt),*.bas;*.txt".

Se omitido, o padrão desse argumento será "Todos os arquivos (*.*),*.*)".

FilterIndex: Variant opcional. Especifica os números de índice dos critérios padrão de filtragem de arquivo, de 1 até o número de filtros especificado em FileFilter. Se esse argumento for omitido ou for superior ao número de filtros presentes, o primeiro filtro de arquivo será usado.

Title: Variant opcional. Especifica o título da caixa de diálogo. Se esse argumento for omitido, o título será "Abrir".

ButtonText: Variant opcional. Somente Macintosh.

MultiSelect: Variant opcional. True para permitir que vários nomes de arquivo sejam selecionados. False para permitir que somente um nome de arquivo seja selecionado. O valor padrão é False

Esse método retorna o nome de arquivo selecionado ou o nome fornecido pelo usuário. O nome retornado pode incluir uma especificação de caminho. Se MultiSelect for True, o valor de retorno será uma matriz dos nomes de arquivo selecionados (mesmo que somente um nome de arquivo seja selecionado). Retorna False se o usuário cancelar a caixa de diálogo.

Alguns exemplos para ilustrar o uso do método GetOpenFileName:

1) Este exemplo exibe a caixa de diálogo Open, com o filtro de arquivo definido para arquivos de texto. Se o usuário escolher um nome de arquivo, o código exibirá esse nome de arquivo em uma caixa de mensagens.

```
fileToOpen = Application.GetOpenFilename("Text Files (*.txt), *.txt")
If fileToOpen <> False Then
    MsgBox "Open " & fileToOpen
End If
```

2) Este exemplo exibe a caixa de diálogo Open, com o filtro de arquivo definido para arquivos de texto (*.txt) e arquivos do Lotus (*.wk?), com um título “Arquivos de Planilhas” e armazena o valor retornado na variável ArquivoACarregar.

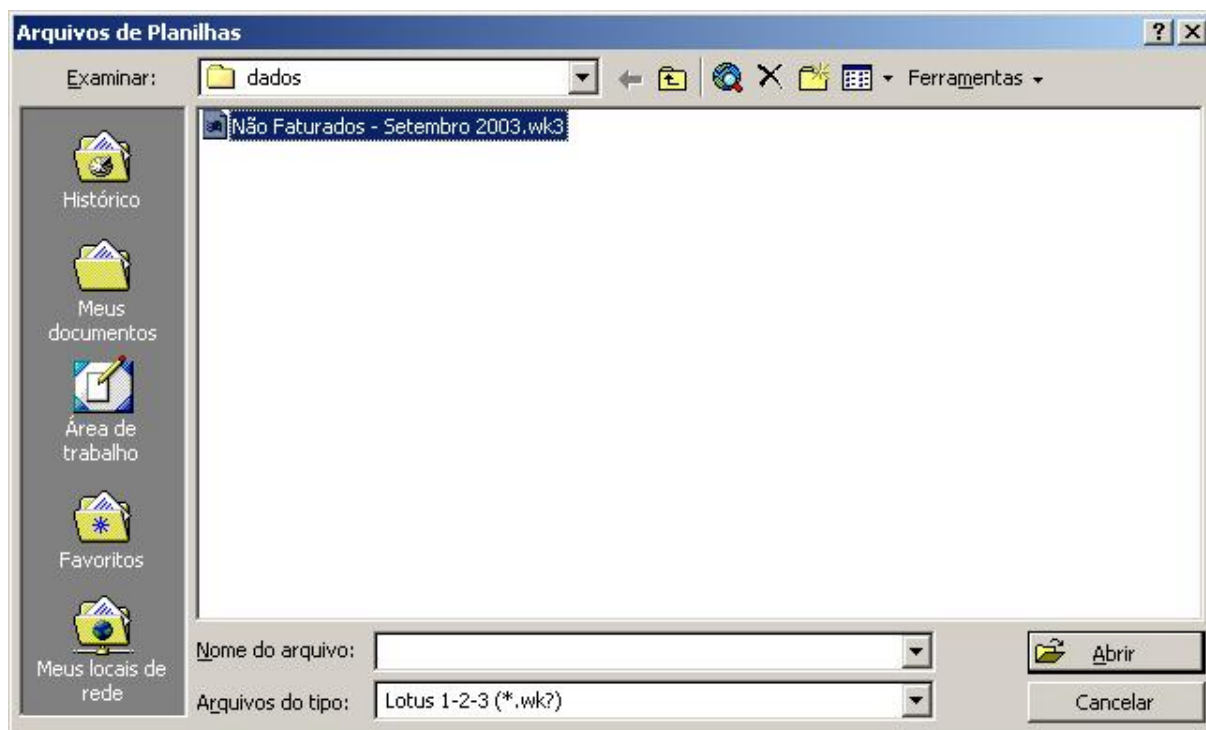
```
ArquivoACarregar = Application.GetOpenFilename("Excel Files (*.xls),*.xls, _
Lotus 1-2-3 (*.wk?),*.wk?",2,"Arquivos de Planilhas")

MsgBox ArquivoACarregar
```

Existem diversos pontos interessantes a serem comentados neste exemplo:

- ▶ Observe que foram definidos dois filtros, um para arquivos do Excel (.xls) e outro para arquivos do Lotus 1-2-3 (*.wk?). Além da descrição Excel Files (*.xls), tem que ser informada, novamente, a extensão *.xls. A definição completa fica assim: Excel Files (*.xls),*.xls. Se houver mais de uma definição elas são separadas por vírgula. Observe que a definição completa vem entre aspas.
- ▶ O valor 2 para o segundo parâmetro indica, dentre os filtros definidos, qual deve ser exibido no campo Arquivos do tipo, da caixa de diálogo Open. Ao definir mais de um filtro (no nosso exemplo definimos dois filtros), a lista dos filtros será exibida na caixa de combinação Arquivos do tipo. A primeira definição tem o índice 1, a segunda o índice 2 e assim por diante.
- ▶ O último parâmetro é o título da Caixa de Diálogo Open.

Na Figura a seguir é exibida a caixa de diálogo Abrir, gerada pelo comando deste exemplo. Observe que na lista Arquivos do tipo é exibido o tipo Lotus 1-2-3 (*.wk?). Este tipo é exibido pela definição do parâmetro FilterIndex (segundo parâmetro), com o valor 2. Observe também o título da janela.



Após ter selecionado um arquivo e clicado em Abrir, o Excel retorna o nome do arquivo selecionado, conforme exemplo da Figura a seguir:



Propriedade DefaultFilePath:

Esta propriedade é utilizada para retornar ou definir o caminho padrão que o Microsoft Excel usa ao abrir arquivos. Define a pasta que será exibida, por padrão, quando for utilizado o comando Arquivo -> Abrir, Arquivo -> Salvar ou Arquivo -> Salvar Como... É do tipo String e é de leitura e gravação.

O exemplo a seguir exibe o atual caminho padrão:

```
MsgBox "Caminha padrão atual: " & Application.DefaultFilePath
```

Propriedade DefaultSaveFormat:

Esta propriedade é utilizada para retornar ou definir o formato padrão para salvar arquivos. Quando você usa o comando Arquivo -> Salvar ou Arquivo -> Salvar como..., já é sugerido um formato padrão de arquivo para salvamento. Este formato é definido por esta propriedade.

Esta propriedade pode assumir uma série de valores, definidos por constantes, conforme indicado na tabela a seguir:

xlAddIn	xlSYLK
xlCSV	xlTemplate
xlCSVMac	xlTextMac
xlCSVMSDOS	xlTextMSDOS
xlCSVWindows	xlTextPrinter
xlCurrentPlatformText	xlTextWindows
xlDBF2	xlUnicodeText
xlDBF3	xlWJ2WD1
xlDBF4	xlWK1
xlDIF	xlWK1ALL
xlExcel2	xlWK1FMT
xlExcel2FarEast	xlWK3
xlExcel3	xlWK4
xlExcel4	xlWK3FM3
xlExcel4Workbook	xlWKS
xlExcel5	xlWorkbookNormal
xlExcel7	xlWorks2FarEast
xlExcel9795	xlWQ1
xlHTML	xlWJ3
xlIntlAddIn	xlWJ3FJ3
xlIntlMacro	

Cada constante já é praticamente autodescritiva do tipo ao qual a constante é associada.

O exemplo define o formato padrão para salvar arquivos, com sendo o formato HTML

```
Application.DefaultSaveFormat = xlHTML
```

Propriedade RecentFiles:

Esta propriedade é utilizada para retornar uma coleção do tipo RecentFiles, a qual representa a lista de arquivos usados recentemente, ou seja, a lista dos últimos arquivos abertos no Excel.

Esta propriedade também pode ser utilizada para definir o número de arquivos que são mantidos na lista de últimos arquivos abertos, conforme exemplo a seguir, onde este número é definido em 6. Ou seja, serão exibidos os nomes das seis últimas pastas de trabalho abertas, quando você usar o comando Arquivo:

```
Application.RecentFiles.Maximum = 6
```

O exemplo de código a seguir, exibe uma Caixa de Mensagem com o nome de cada um dos quatro últimos arquivos abertos no Excel:

```
Application.RecentFiles.Maximum = 6

For Each Pasta In Application.RecentFiles
    MsgBox Pasta.Name
Next
```

Método FindFile:

Este método é utilizado para exibir a caixa de diálogo Abrir. Este método exibe a caixa de diálogo Abrir e permite que o usuário abra um arquivo. Se um novo arquivo for aberto com sucesso, esse método retornará True. Se o usuário cancelar a caixa de diálogo, este método retornará False.

Sintaxe:

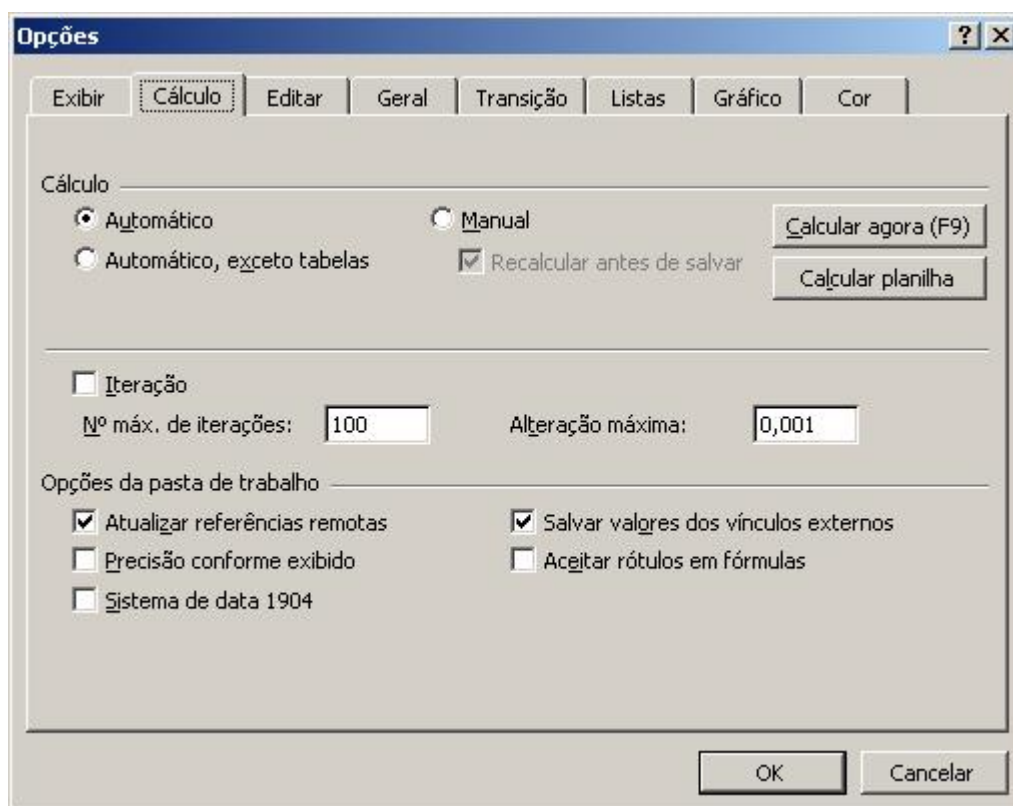
```
Application.FindFile
```

No exemplo a seguir, será aberta a caixa de diálogo Abrir (Arquivo -> Abrir).

```
Application.FindFile
```

Lição 11: Objeto Application – Recálculo da Planilha

Por padrão, o Excel faz o recálculo automático das planilhas. Sempre que o valor de uma célula é alterado, todas as fórmulas que dependem do valor que foi alterado são recalculadas, para exibir os valores atualizados. Você pode controlar a maneira como o Excel faz o recálculo, usando o comando Ferramentas -> Opções -> Guia Cálculo. Por exemplo, para planilhas muito grandes, o recálculo automático pode fazer com que você tenha que esperar vários minutos após alterar um determinado valor. Nestas situações, pode ser vantagem desabilitar o recálculo automático. Aí você altera os valores necessários e depois pressiona a tecla F9 para forçar um recálculo de toda a planilha. Na Figura a seguir, mostra a tela da guia Cálculo, da janela Ferramentas -> Opções:



Observe que o padrão é o Cálculo Automático. Observe também o botão Calcular agora (F9), o qual força um recálculo em toda a planilha. A forma de Cálculo pode ser controlada também através de programação VBA, usando o Método Calculate e a propriedade Calculation.

Método Calculate:

Este método é utilizado para calcular, isto é, forçar um recálculo imediato, em todas as pastas de trabalho abertas (se for usado o método Calculate do objeto Application), uma planilha específica em uma pasta de trabalho (se for usado o método Calculate do objeto Worksheet) ou um intervalo especificado de células em uma planilha (se for utilizado o método calculate do objeto Range), conforme exemplificado na Tabela a seguir:

Para calcular	Exemplo de código
Todas as pastas de trabalho abertas	Application.Calculate (ou apenas Calculate)
Uma planilha específica	Worksheets(1).Calculate
Um intervalo especificado	Worksheets(1).Rows(2).Calculate

A seguir temos mais um exemplo de uso do método Calculate:

Este exemplo calcula as fórmulas das colunas A, B e C no intervalo utilizado em Sheet1.

```
Worksheets("Sheet1").UsedRange.Columns("A:C").Calculate
```

Propriedade Calculation:

Esta propriedade retorna ou define o modo de cálculo. Esta propriedade pode assumir o valor definido por uma das seguintes constantes XlCalculation:

- ▶ XlCalculationAutomatic
- ▶ XlCalculationManual
- ▶ xlCalculationSemiautomatic.

Nota: Para fontes de dados OLAP, esta propriedade só pode retornar ou ser definida como xlNormal.

O código do exemplo faz o Microsoft Excel calcular pastas de trabalho antes de elas serem salvas em disco.

```
Application.Calculation = xlCalculationManual  
Application.CalculateBeforeSave = True
```

O que faz o Excel calcular as planilhas antes de salvá-las é a definição do valor True para a propriedade CalculateBeforeSave.

No exemplo a seguir, defino a fórmula de cálculo para manual, em seguida redefino a forma de cálculo para Automática. Essa técnica é útil quando serão executados comandos que fazem muitas alterações em uma planilha, como por exemplo um laço For que percorre todos os dados de uma coluna, alterando os valores. Neste exemplo, ao alterar os valores, todas as fórmulas que dependem destes valores, serão automaticamente recalculadas, se a fórmula de cálculo estiver em automática. Isso pode fazer com que a execução do código fique extremamente lenta. Para evitar isso, coloca-se a forma de cálculo como Manual, faz-se as alterações necessárias e após as alterações, forçamos um recálculo da planilha e ativamos novamente o método automático de cálculo:

```
Application.Calculation = xlCalculationManual  
  
` Comandos que fazem alterações nos dados  
  
Application.Calculate `Faço o recálculo da planilha  
Application.Calculation = xlCalculationAutomatic
```

Lição 12: Conceitos Avançados na Criação de Funções e Procedimentos

Nas lições do Módulo 1, você aprendeu os conceitos básicos para a criação de funções e procedimentos personalizados. Apresentei inclusive alguns exemplos simples, de criação de funções personalizadas, como por exemplo uma função para cálculo do Imposto de Renda e uma função para cálculo do DV do CPF. Nas demais lições deste módulo, apresentarei uma série de conceitos avançados sobre a criação de Procedimentos e Funções, sobre o uso de Procedimentos e Funções e sobre o tratamento de erro no VBA.

Dica: Não existe limite para o número de linhas de código que pode haver em um procedimento ou função. Porém uma boa prática de programação é não criar procedimentos ou funções muito extensos. Sempre que possível é indicado que você crie procedimentos ou funções para executar tarefas específicas, ou seja, dividir um problema maior em etapas menores e implementar um procedimento ou função para solucionar cada etapa específica. Depois você pode criar um procedimento principal, no qual você chama os diversos procedimentos/funções auxiliares, para solucionar as diversas etapas do problema.

Detalhes sobre a declaração e o escopo de procedimentos no VBA:

Nota: O termo procedimento é um termo genérico, o qual se refere tanto a Sub-procedures quanto à funções. Em outras palavras, um Sub-procedure é um procedimento e uma função também é um procedimento.

Sintaxe para a declaração de Sub-procedimentos:

```
[Private | Public] [Static] Sub NomeDoSub-Procedure (arg1, arg2, ..., argn)  
  
    Comando 1  
    Comando 2  
    Comando 3  
    ...  
    Comando n  
  
End Sub
```

[Private | Public]: Este parâmetro é opcional. Esta primeira parte da declaração, define o escopo onde poderá ser utilizado o Sub-procedimento:

- ▶ **Private:** Ao declarar um sub-procedimento como Private, você indica ao VBA, que este sub-procedimento somente poderá ser chamado por outros sub-procedimentos ou funções, contidas no mesmo módulo onde o sub-procedimento foi criado. Em outras palavras, um sub-procedimento declarado como private, será privativo do módulo onde ele foi criado.
- ▶ **Public:** Ao declarar um sub-procedimento como Public, você indica ao VBA, que este sub-procedimento, poderá ser chamado por qualquer outro sub-procedimento ou função, contido em qualquer outro módulo da pasta de trabalho. Ou seja, o sub-procedimento é de uso público para todos os módulos da pasta de trabalho.

[Static]: Ao usar a opção Static, na declaração de um sub-procedimento, você define que o valor das variáveis do procedimento, serão mantidas na memória, mesmo após ter sido encerrada a execução do sub-procedimento. Conforme vimos nas lições do Módulo 1, por padrão, as variáveis declaradas dentro de um sub-procedimento ou função, tem escopo de procedimento, ou seja, uma vez encerrada a execução do procedimento, o valor das variáveis declaradas no procedimento, é descartado. Com o uso da opção Static, você faz com que o Excel mantenha estes valores, os quais poderão ser usados novamente, se necessário.

NomeDoSub-Procedure: O próximo item é o nome do procedimento. A principal dica é que o nome deve ser descritivo do que faz o sub-procedimento.

Lista de Argumentos: Após o nome, entre parênteses, vem uma lista de argumentos. Os argumentos são utilizados para passar valores para o sub-procedimento. Por exemplo, vamos supor que você está desenvolvendo um sub-procedimento, qual irá atuar sobre uma faixa de células da planilha atual, aumentando em um determinado percentual, o valor das células da faixa que tiverem um valor entre 0 e 100. Neste caso você tem que passar, como parâmetro para o sub-procedimento, a faixa de células sobre a qual deve atuar o sub-procedimento e o valor percentual a ser aplicado. A seguir temos um exemplo da declaração deste sub-procedimento, o qual é do tipo Public, tem o nome de AplicaPercentual e recebe como parâmetros uma faixa de células e um valor percentual.

```
Public Sub AplicaPercentual(Faixa As Range, ValorPercentual As Double)  
    Comando 1  
    Comando 2  
    Comando 3  
    ...  
    Comando n  
End Sub
```

Observe que além do nome de cada argumento, também é declarado o tipo do argumento. Ao declarar o tipo, o VBA não aceita que seja chamado o sub-procedimento, passando tipos diferentes dos declarados. Imagine que você precisa chamar o procedimento AplicaPercentual, dentro de um outro procedimento da pasta de trabalho e que neste procedimento você já declarou e inicializou uma variável do tipo Range, chamada Dados e uma variável do tipo Double chamada Incremento. Neste caso, você pode chamar o procedimento AplicaPercentual, usando o código a seguir:

```
AplicaPercentual(Dados, Incremento)
```

O Excel chama o Sub-procedimento AplicaPercentual e passa o valor da variável Dados para o parâmetro Faixa e o valor da variável Incremento para o parâmetro ValorPercentual. Estes valores são utilizados pelo Sub-procedure AplicaPercentual, para realizar as ações necessárias, de acordo com os comandos que foram definidos no sub-procedimento.

Como o sub-procedimento AplicaPercentual, foi declarado como sendo Public, ele poderá ser chamado por qualquer procedimento ou função, de qualquer módulo, da pasta de trabalho onde ele foi criado. Se ao invés de Public, eu tivesse usado Private, ele somente poderia ser chamado dentro do mesmo módulo de código, onde ele foi criado.

Observações sobre o escopo de um sub-procedimento:

Por padrão, um sub-procedimento é criado com o escopo Public. Você pode até mesmo declarar um sub-procedimento, sem usar a palavra Public, conforme indicado a seguir. Neste exemplo, o sub-procedimento terá o escopo Public, que é o padrão:

Sub AplicaPercentual(Faixa As Range, ValorPercentual As Double)

Comando 1

Comando 2

Comando 3

...

Comando n

End Sub

Dica: Embora a opção Public não seja obrigatório, é recomendada a utilização desta opção, por questões de clareza, ou seja, para deixar documentado no próprio código, que o sub-procedimento é de escopo público. Estas pequenas dicas podem parecer preciosismo, mas posso garantir que elas são de grande valia. Enquanto você está criando as suas rotinas de programação você tem na memória tudo o que foi feito, qual a função de cada procedimento, qual o significado de cada variável e assim por diante. Porém daqui a seis meses ou um ano, quando você precisar fazer alterações no código, você já terá esquecido de quase tudo, desde para que serve cada procedimento, do significado de cada variável e assim por diante. Por isso é importante que, ao criar o seu código, você o crie o mais auto-explicativo possível, inserindo quantos comentários forem necessários, utilizando nomes de variáveis e de procedimentos bastante autodescritivos e deixando bem claro qual o tipo de cada procedimento, variável e função.

Importante: Quando você seleciona o comando Ferramentas -> Macro -> Macros, será exibida uma lista somente com os sub-procedimentos e funções com escopo Public.

Definindo o escopo de um módulo como Privado:

Um módulo de código do VBA é composto de um ou mais procedimentos. Além dos procedimentos, existe uma seção chamada de seção de declaração, onde você pode declarar variáveis como o escopo de módulo, ou seja, variáveis que estarão disponíveis para serem utilizadas em qualquer procedimento dentro do módulo. Além da declaração de variáveis, existe uma série de comandos que podem ser utilizados na seção de declarações e que alteram opções importantes do módulo. Uma das declarações disponíveis está indicada a seguir:

Option Private Module

Esta declaração faz com que todos os procedimentos de um módulo, tenham escopo privado ao módulo, ou seja, somente possam ser acessados dentro do próprio módulo. Isso vale mesmo para os procedimentos que tenham sido declarados com a opção Public, ou seja, a declaração de privacidade do módulo, tem precedência sobre a declaração Public de um ou mais procedimentos do módulo.

Lição 13: Conceitos Avançados na Criação de Funções e Procedimentos

Nesta e nas próximas lições você aprenderá mais detalhes sobre a declaração de argumentos e sobre a passagem de parâmetros para um sub-procedimento e/ou função.

Conforme descrevi na lição anterior, a passagem de parâmetros é o método utilizado, para passar valores para uma função ou procedimento, sendo que estes valores serão utilizados pelos comandos do procedimento ou função. Vamos pegar um exemplo bem simples, que com certeza você já utilizou diversas vezes no Excel: A função Soma é uma função interna do Excel e como tal, pode receber parâmetros. A função Soma, na sua forma mais simples, recebe como parâmetro, uma faixa de células, para a qual será efetuada a soma dos valores, como nos exemplos a seguir:

```
=Soma ( A1 : A50 )  
=Soma ( A1 : A50 ; C3 : C20 )
```

Ao criar nossos procedimentos, também podemos definir quais os argumentos serão passados para a função, se um argumento é obrigatório ou opcional e podemos também criar os chamados parâmetros nomeados. Este será justamente o assunto desta lição, ou seja, como declarar e utilizar argumentos/parâmetros (vamos considerá-los como sinônimos).

Declaração de argumentos:

Ao criar um sub-procedimento ou função, você pode declarar um ou mais parâmetros, bem como o tipo dos parâmetros. A declaração de parâmetros não é obrigatório. Os parâmetros são utilizados quando devem ser passados valores para o procedimento (sub-procedure ou função), valores dos quais depende a execução do procedimento. Ao declarar os argumentos, você define um nome e um tipo para o argumento, conforme indicado nos exemplos a seguir:

1. Declaração de uma função chamada SomaEsperta, a qual faz uma soma em um intervalo de células, somando somente valores que estão dentro de uma determinada faixa e ignorando ou não valores negativos. Esta função tem quatro argumentos, descritos a seguir:

- ▶ **Faixa:** A faixa onde estão os valores a serem somados.
- ▶ **LimInf:** O valor inferior da faixa, dentro da qual serão somados os valores. Por exemplo, se você deseja somar somente os valores que estão entre 200 e 500, o valor do parâmetro LimInf deverá ser 200.
- ▶ **LimSup:** O valor superior da faixa, dentro da qual serão somados os valores. Por exemplo, se você deseja somar somente os valores que estão entre 200 e 500, o valor do parâmetro LimSub deverá ser 500.
- ▶ **SomaNeg:** Este parâmetro será do tipo Boolean, podendo assumir os valores True ou False. Se for True, os valores negativos serão somados, se for False, os valores negativos serão ignorados.

A seguir apresento a declaração da função:

```
Public Function SomaEsperta(Faixa As Range, LimInf As Double, LimSup As Double, _  
SomaNeg As Boolean) As Double
```

Observe que o `As Double`, após o fechar parênteses, indica o tipo de retorno da função. Ou seja, a função `SomaEsperta`, após fazer os cálculos, com base no valor dos parâmetros que foram passados para a função, irá retornar um valor do tipo `Double`. A seguir vamos implementar e utilizar a função `SomaEsperta`, em um exemplo passo-a-passo.

Exemplo: Implementar e utilizar a função `SomaEsperta`, a qual faz uma soma em um intervalo de células, somando somente valores que estão dentro de uma determinada faixa e ignorando ou não valores negativos.

1. Abra o Excel.
2. Abra a planilha `C:\Programação VBA no Excel\ Modulo 2 - SomaEsperta 06.xls`.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione `Alt+F11`.
4. Vamos criar a função `SomaEsperta` de tal maneira que ele possa ser utilizada em qualquer local da planilha. Para isso vamos criá-la como uma função Pública, dentro de um módulo do VBA.
5. Para inserir um módulo, selecione o comando **Inserir -> Módulo**. Será criado o Módulo 1. Agora vamos inserir uma função, dentro desse módulo. Clique em Módulo 1, no painel da esquerda para selecioná-lo.
6. Selecione o comando **Inserir -> Procedimento**.
7. Será aberta a janela Adicionar procedimento. Preencha os dados conforme indicado na Figura a seguir. Observe a opção `Escopo Público`. Esta opção insere a opção `Public` antes do nome da função, o que faz com que a função seja de escopo público, ou seja, possa ser acessada em todos os módulos da pasta de trabalho:



8. Clique em **OK**. Será criada a declaração da função. Agora altere a declaração para incluir os parâmetros, conforme indicado no exemplo a seguir:

```
Public Function SomaEsperta(Faixa As Range, LimInf As Double, _  
    LimSup As Double, SomaNeg As Boolean) As Double  
  
End Function
```

9. O código da função, basicamente irá declarar uma variável soma, e percorrer todas as células do intervalo fornecido testando. Se o valor da célula estiver no intervalo definido pelos parâmetros ValInicial e ValFinal, o código testará o valor do parâmetro SomaNeg. Se o valor do parâmetro SomaNeg for True, o valor será somado à variável de soma; se o valor da variável SomaNeg for False, o valor será testado e, se for negativo, não será somado. No final o valor da variável soma será atribuído ao nome da função, para que este valor seja retornado pela função.

10. Digite o código da função, conforme indicado na listagem a seguir:

```
Public Function SomaEsperta(Faixa As Range, LimInf As Double, LimSup As Double, _  
    SomaNeg As Boolean) As Double  
  
    ' Declaração da variável de Soma  
    Dim ValorSoma As Double  
    ValorSoma = 0  
  
    ' Uso a estrutura For...Each, para percorrer todas as células da faixa fornecida  
    ' no parâmetro Faixa  
  
    For Each Cel In Faixa  
  
        ' Testo para ver se o valor está no intervalo definido pelos parâmetros  
        ' LimInf e LimSup e se devo somar valores Negativos  
  
        If (Cel.Value >= LimInf) And (Cel.Value <= LimSup) And (SomaNeg = True) Then  
            ValorSoma = ValorSoma + Cel.Value  
        End If  
  
        ' Se SomaNeg For False, devo descartar os valores negativos, o que é feito  
        ' pelo teste a seguir:  
  
        If (Cel.Value >= LimInf) And (Cel.Value <= LimSup) And (SomaNeg = False) Then  
            If Cel.Value >= 0 Then  
                ValorSoma = ValorSoma + Cel.Value  
            End If  
        End If  
  
    Next  
  
    ' Atribuo o valor da variável ValorSoma ao nome da função:  
  
    SomaEsperta = ValorSoma  
  
End Function
```

O código desta função é relativamente simples. A seguir mais alguns comentários para esclarecer possíveis pontos de dúvidas. Observe que este talvez não seja o código de melhor desempenho, pois o objetivo é didático, para ilustrar a declaração e uso de argumentos em uma função.

1. Para iniciar a função, declaro uma variável ValorSoma e inicializo esta variável em 0:

```
' Declaração da variável de Soma
Dim ValorSoma As Double
ValorSoma = 0
```

2. O próximo passo é iniciar uma estrutura For...Each, para percorrer todas as células, da coleção de Células, da faixa passada como parâmetro para a função:

```
' Uso a estrutura For...Each, para percorrer todas as células da '
' faixa fornecida
' no parâmetro Faixa

For Each Cel In Faixa
```

3. Dentro da estrutura For..Each, para cada célula faço um teste para ver se a célula está na faixa definida pelos parâmetros LimInf e LimSup e se o valor do parâmetro SomaNeg é True, caso em que os valores negativos também deverão ser somados (desde que a faixa definida pelos parâmetros LimInf e LimSup, também inclua valores negativos).

```
' Testo para ver se o valor está no intervalo definido pelos
' parâmetros LimInf e LimSup e se devo somar valores Negativos

If (Cel.Value >= LimInf) And (Cel.Value <= LimSup) And (SomaNeg = True) Then
    ValorSoma = ValorSoma + Cel.Value
End If
```

Nesta etapa, se o valor da célula estiver dentro da faixa e se o parâmetro SomaNeg tiver sido definido em True, a variável ValorSoma será atualizada para incluir o valor contido na célula.

4. Ainda dentro da estrutura For...Each, uso mais um teste, para ver se a célula está na faixa definida pelos parâmetros LimInf e LimSup e se o valor do parâmetro SomaNeg é False, caso em que os valores negativos deverão ser ignorados (desde que a faixa definida pelos parâmetros LimInf e LimSup, também inclua valores negativos).

```
' Se SomaNeg For False, devo descartar os valores negativos,
' o que é feito pelo teste a seguir:

If (Cel.Value >= LimInf) And (Cel.Value <= LimSup) And (SomaNeg = False) Then
    If Cel.Value >= 0 Then
        ValorSoma = ValorSoma + Cel.Value
    End If
End If
```

5. O passo final é atribuir o valor da variável ValorSoma, ao nome da função, para que este valor seja retornado para a planilha:

```
' Atribuo o valor da variável ValorSoma ao nome da função:
```

```
SomaEsperta = ValorSoma
```

A seguir apresento alguns exemplos de uso da função SomaEsperta, baseados nos valores da Figura da planilha a seguir:

	A	B	C	D	E	F
1	Código	Cliente	Funcionário	Data	Transportadora	Valor
2	10248	VINET	5	04/07/1996	3	R\$50,00
3	10249	TOMSP	6	05/07/1996	1	R\$30,00
4	10250	HANAR	4	08/07/1996	2	R\$60,00
5	10251	VICTE	3	08/07/1996	1	-R\$75,00
6	10252	SUPRD	4	09/07/1996	2	-R\$30,00
7	10253	HANAR	3	10/07/1996	2	R\$75,00
8	10254	CHOPS	5	11/07/1996	2	R\$100,00
9	10255	RICSU	9	12/07/1996	3	R\$95,00
10	10256	WELLI	3	15/07/1996	2	-R\$15,00
11	10257	HILAA	4	16/07/1996	3	R\$30,00
12	10258	ERNSH	1	17/07/1996	1	R\$65,00
13	10259	CENTC	4	18/07/1996	3	R\$48,00
14	10260	OTTIK	4	19/07/1996	1	-R\$20,00
15	10261	QUEDE	4	19/07/1996	2	-R\$75,00
16	10262	RATTC	8	22/07/1996	3	-R\$20,00
17	10263	ERNSH	9	23/07/1996	3	R\$50,00
18	10264	FOLKO	6	24/07/1996	3	R\$55,00
19	10265	BLONP	2	25/07/1996	1	-R\$35,00
20	10266	WARTH	3	26/07/1996	3	R\$40,00
21	10267	FRANK	4	29/07/1996	1	R\$80,00

Exemplo de uso da função	Valor retornado
=SomaEsperta(F2:F21;-50;0;"True")	-120
=SomaEsperta(F2:F21;-50;0;"False")	0
=SomaEsperta(F2:F21;0;50;"True")	248
=SomaEsperta(F2:F21;0;50;"False")	248

Observe alguns detalhes interessantes em relação a esta função. No exemplo: =SomaEsperta(F2:F21;-50;0;"False"), o valor de retorno só pode ser zero, pois a faixa de -50 a 0, só tem valores negativos ou zero e pedi para ignorar os negativos, só restaram os zeros ou nenhum valor, por isso o retorno é zero. Observe os dois últimos exemplos, onde alterei o parâmetro SomaNeg de True para False e o valor de retorno não se alterou, ficando em 248 nos dois casos. Como a faixa de valores está entre 0 e 50, ou seja, sem valores negativos, é indiferente o valor definido para o parâmetro SomaNeg, pois não existem valores negativos na faixa de 0 a 50. Isso comprova que a nossa função está funcionando corretamente, dentro dos objetivos propostos para este exemplo.

Lição 14: Conceitos Avançados na Criação de Funções e Procedimentos

Na última lição falei um pouco mais sobre a declaração de parâmetros, tipo de parâmetros e tipo de retorno de uma função. Você também acompanhou, passo-a-passo, a criação de uma função Personalizada – SomaEsperta, e depois aprendeu a utilizar esta função. Esta é uma técnica importante, pois apesar de o Excel disponibilizar centenas de funções, evidentemente que não existem funções prontas no Excel, capazes de atender todas as necessidades de cálculo do dia-a-dia. Por isso a criação de funções personalizadas, com o uso do VBA, abrem-se novas perspectivas no uso de Excel. Nesta lição continuaremos o estudo sobre funções e sub-procedures no Excel.

Argumentos opcionais:

Ao declarar os argumentos de uma função ou sub-procedure, você pode criá-los como argumentos obrigatórios (que tem que ser passados quando a função ou sub-procedure for chamada, senão irá gerar um erro) ou como parâmetros opcionais, ou seja, podem ser definidos ou não na chamada da função ou sub-procedure. O código VBA deve testar se o valor de um argumento opcional foi ou não passado para a função ou sub-procedure e tomar diferentes caminhos, dependendo de o argumento ter ou não sido passado.

Para definir um argumento como Opcional, é utilizada a palavra chave **Optional**. Considere o exemplo a seguir, onde criamos um procedimento chamado **MudaSelecao**. Este procedimento coloca, em Negrito, as células da seleção atual da planilha e altera o tamanho da fonte, de acordo com o tamanho definido no primeiro parâmetro passado para o procedimento, parâmetro este que é obrigatório. Este procedimento tem um segundo parâmetro, o qual é opcional. Se o segundo parâmetro for definido, a fonte será alterada para o tipo de fonte definido neste segundo parâmetro. Se o segundo parâmetro, que é opcional, não for informado na chamada do procedimento, a fonte não será alterada.

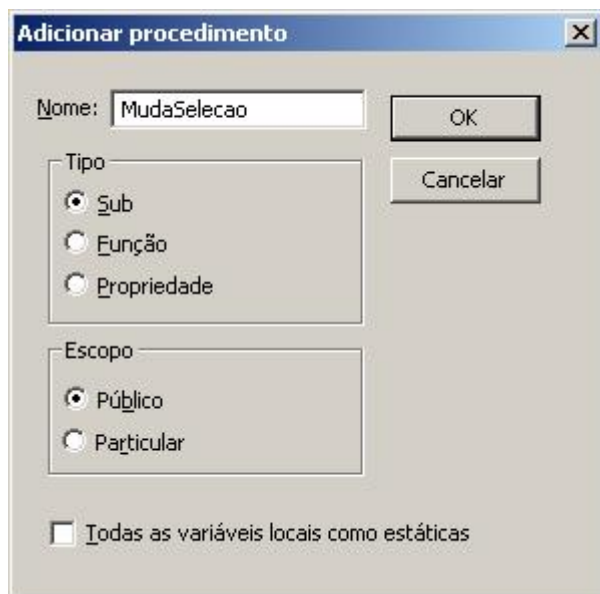
```
Public Sub MudaSelecao(TamFonte As Variant, Optional NomeFonte As String)
End Sub
```

Para ilustrar o uso de argumentos opcionais, vamos implementar e testar a função MudaSelecao.

Exemplo: Implementar e utilizar o procedimento MudaSelecao, a qual faz alterações na seleção atual e altera também o tipo de fonte, dependendo de um argumento opcional ter ou não sido chamado. Na próxima lição vamos criar um procedimento chamado ExMudaSelecao, o qual faz várias chamadas ao procedimento MudaSelecao, com diferentes valores para os seus argumentos. Neste segundo procedimento, faremos diversas chamadas ao procedimento MudaSelecao, testando diferentes valores para os seus argumentos.

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 2 - SomaEsperta 06.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.

4. Vamos criar o procedimento MudaSelecao de tal maneira que ele possa ser chamado em qualquer módulo de código da pasta de trabalho. Para isso vamos cria-lo como um procedimento Público, dentro de um módulo do VBA.
5. Clique em Módulo 1, no painel da esquerda para selecioná-lo. Lembrando que o Módulo 1 foi criado no exemplo da Lição anterior.
6. Selecione o comando **Inserir -> Procedimento**.
7. Será aberta a janela Adicionar procedimento. Preencha os dados conforme indicado na Figura a seguir. Observe a opção Escopo Público. Esta opção insere a opção Public antes do nome do procedimento, o que faz com que o procedimento seja de escopo público, ou seja, possa ser acessada em todos os módulos da pasta de trabalho:



8. Clique em OK. Será criada a declaração do procedimento. Agora altere a declaração para incluir os parâmetros, conforme indicado no exemplo a seguir. Observe o uso da palavra Optional, para definir o segundo argumento como opcional:

```
Public Sub MudaSelecao(TamFonte As Integer, Optional NomeFonte As Variant)  
  
End Sub
```

9. O código do procedimento é bastante simples. Inicialmente cabe comentar o uso do objeto Selection, o qual faz referência a faixa de células atualmente selecionada. Neste caso, o procedimento usa o valor definido no parâmetro TamFonte, para definir o tamanho da fonte para todas as células da faixa atualmente selecionada (objeto Selection). O próximo passo é testar se o parâmetro NomeFonte foi ou não definido, na chamada do procedimento MudaSelecao. Para verificar se o parâmetro foi passado, uso o teste Not IsMissing(NomeDoParâmetro). Este uso merece alguns comentários mais detalhados. A função IsMissing irá retornar verdadeiro se o parâmetro não tiver sido definido. Orá, se o parâmetro não for definido, não queremos que seja feita a alteração. Por isso o uso do Not. Ou seja, quando o parâmetro não for definido, o teste fica Not Verdadeiro, o que significa Falso. Ou seja, quando o parâmetro não for definido, o teste Not IsMissing(NomeDoParâmetro) retorna Falso e os comandos não são executados. Quando o parâmetro for definido, o IsMissing retornar Falso. O Not inverte o Falso para verdadeiro e a formatação é aplicada.

Observe que o efeito prático é que a formatação não é aplicada quando o parâmetro não for definido (Not Verdadeiro = Falso) e a formatação é aplicada quando o parâmetro for definido (Not Falso = Verdadeiro), ou seja, exatamente o comportamento desejado.

10. Digite o código da função, conforme indicado na listagem a seguir:

```
Public Sub MudaSelecao(TamFonte As Integer, Optional NomeFonte As Variant)

    ' Inicialmente altero o tamanho da fonte da seleção atual.

    Selection.Font.Size = TamFonte

    ' Uso a função IsMissing, para testar se o argumento NomeFonte
    ' foi ou não utilizado na chamada do procedimento.

    If Not IsMissing(NomeFonte) Then
        Selection.Font.Name = CStr(NomeFonte)
    End If

    ' Observe que o tipo de fonte somente será definido se o parâmetro
    ' NomeFonte, for passado quando da chamada do procedimento.

End Sub
```

11. O ponto realmente interessante deste procedimento é a lógica definida pelo Not IsMissing, conforme descrito anteriormente. Muito bem, agora temos um procedimento que atua sobre a seleção atual na planilha. Na próxima lição você aprenderá a utilizar este procedimento e a fazer diferentes chamadas, com diferentes valores para os seus parâmetros, bem como a utilizá-lo definindo valor para o parâmetro opcional e sem utilizar o parâmetro opcional.

Importante: Observe que o parâmetro opcional NomeFonte foi definido como sendo do tipo Variant e não diretamente como do tipo String. Isso é necessário porque a função IsMissing, só funciona com dados do tipo Variant (ou seja, sem tipo definido). Em seguida, uso CStr, para converter o valor do parâmetro NomeFonte de variant para String.

Lição 15: Conceitos Avançados na Criação de Funções e Procedimentos

Na lição anterior você aprendeu sobre a utilização de parâmetros Opcionais em procedimentos e funções do VBA. Você viu que para definir um parâmetro como Opcional, utilizamos a palavra chave `Optional`. Você também aprendeu sobre a lógica do uso da função `IsMissing`, para determinar se um parâmetro opcional foi ou não definido.

A questão que fica é como utilizar este procedimento. Existem diferentes maneiras para a execução deste procedimento. Vou optar por criar um segundo procedimento, chamado `ExMudaSelecao`, no qual farei diversas chamadas ao procedimento `MudaSelecao`, usando diferentes valores para os seus argumentos. Em seguida executarei o procedimento `ExMudaSelecao`, diretamente a partir do editor do VBA.

Exemplo: Criar o procedimento `ExMudaSelecao`, para fazer diferentes chamadas ao procedimento `MudaSelecao`, com diferentes valores para os seus argumentos.

1. Abra o Excel.
2. Abra a planilha `C:\Programação VBA no Excel\ Modulo 2 - SomaEsperta 06.xls`.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione `Alt+F11`.
4. Vamos criar o procedimento `ExMudaSelecao` de tal maneira que ele possa ser chamado em qualquer módulo de código da pasta de trabalho. Para isso vamos cria-lo como um procedimento Público, dentro de um módulo do VBA.
5. Clique em Módulo 1, no painel da esquerda para selecioná-lo. Lembrando que o Módulo 1 foi criado no exemplo da Lição 12.
6. Selecione o comando **Inserir -> Procedimento**.
7. Será aberta a janela Adicionar procedimento. Preencha os dados conforme indicado na Figura a seguir. Observe a opção Escopo Público. Esta opção insere a opção `Public` antes do nome do procedimento, o que faz com que o procedimento seja de escopo público, ou seja, possa ser acessada em todos os módulos da pasta de trabalho:



8. Clique em OK. Será criada a declaração do procedimento. Este procedimento não terá nenhum parâmetro. Com isso sua estrutura ficará conforme indicado a seguir:

```
Public Sub ExMudaSelecao()  
  
End Sub
```

9. O código do procedimento é bastante simples. Basicamente farei diferentes chamadas ao procedimento muda seleção. Na primeira chamada defino o valor para os dois parâmetros, definindo um tamanho de fonte 14 e o tipo de fonte como Verdana. Na segunda chamada, altero o tamanho de fonte para 8 e não defino o NomeDaFonte. Nesta segunda chamada estou demonstrando que o parâmetro NomeFonte realmente é opcional. Faço uma terceira chamada, onde defino o tamanho em 12 e o tipo da fonte em Arial. Entre uma chamada e outra, uso o comando Stop, para que possamos alternar para a planilha e mostrar o resultado da execução de cada chamada do procedure MudaSelecao.

10. Digite o código da função, conforme indicado na listagem a seguir:

```
Public Sub ExMudaSelecao()  
  
    MudaSelecao 14, Verdana  
    Stop  
    MudaSelecao 8  
    Stop  
    MudaSelecao 12, Arial  
  
End Sub
```

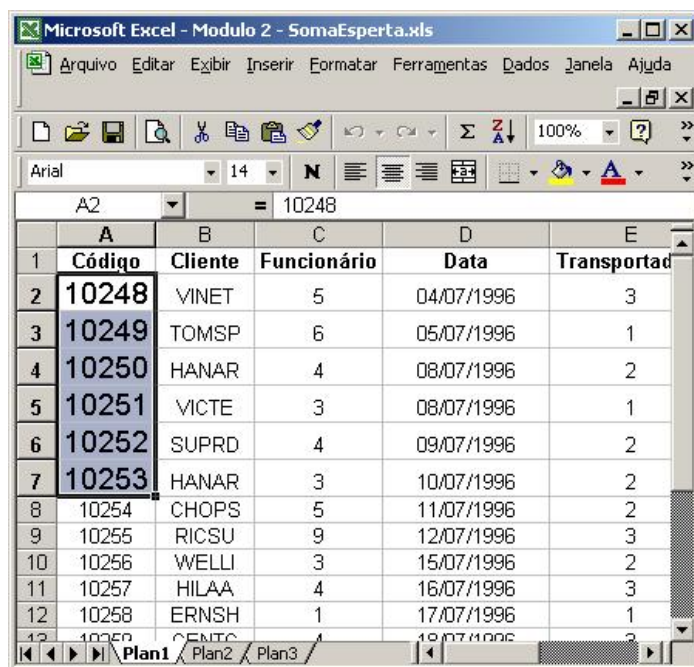
11. Antes de executarmos a função ExMudaSelecao é interessante observar a forma como são passados os argumentos para o procedimento MudaSelecao. Observe que não usamos os parênteses, como no exemplo do uso da função SomaEsperta, criada nas lições anteriores. Usei o nome do procedimento, um espaço e o valor dos parâmetros, separados por vírgula. Na segunda chamada, defino apenas o valor para o primeiro parâmetro. Não defino valor para o parâmetro opcional.

12. Vamos executar o procedimento ExMudaSelecao diretamente a partir do editor do VBA. Para isso certifique-se de que o curso esteja em qualquer linha de código, dentro do procedimento ExMudaSelecao. Pressione a tecla F5 para executar o procedimento.

13. A primeira chamada da função é executada e a execução para no método Stop

```
MudaSelecao 14, Verdana  
Stop
```

14. O resultado desta primeira chamada é indicada na figura a seguir, onde você pode observar que apenas as células previamente selecionadas tiveram a sua formatação alterada para fonte Verdana, tamanho 14.

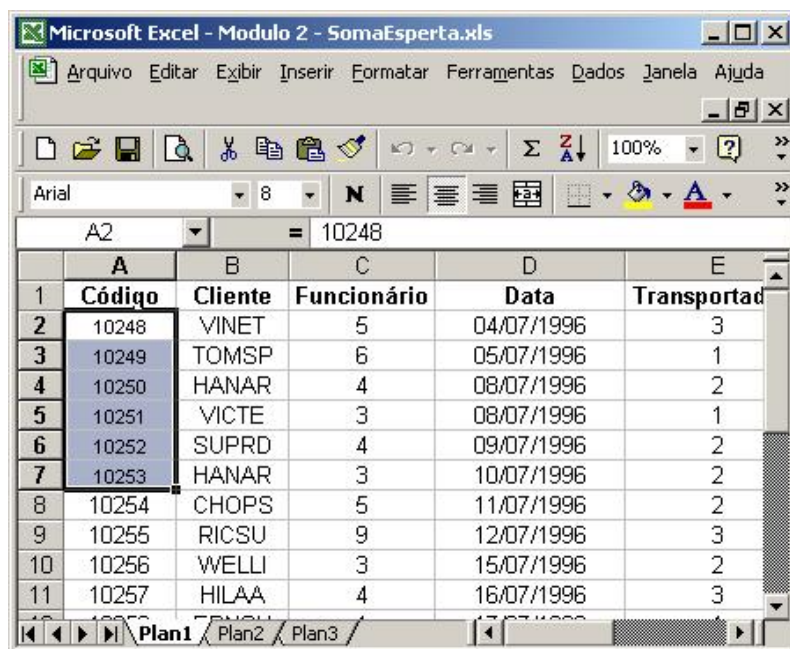


	A	B	C	D	E
1	Código	Cliente	Funcionário	Data	Transportad
2	10248	VINET	5	04/07/1996	3
3	10249	TOMSP	6	05/07/1996	1
4	10250	HANAR	4	08/07/1996	2
5	10251	VICTE	3	08/07/1996	1
6	10252	SUPRD	4	09/07/1996	2
7	10253	HANAR	3	10/07/1996	2
8	10254	CHOPS	5	11/07/1996	2
9	10255	RICSU	9	12/07/1996	3
10	10256	WELLI	3	15/07/1996	2
11	10257	HILAA	4	16/07/1996	3
12	10258	ERNSH	1	17/07/1996	1

15. Para continuar a execução do código pressione a tecla F5.
16. A segunda chamada da função é executada e a execução para no método Stop

MudaSelecao 8
Stop

14. O resultado desta segunda chamada é indicada na figura a seguir, onde você pode observar que apenas as células previamente selecionadas tiveram a sua formatação alterada para fonte tamanho 8.

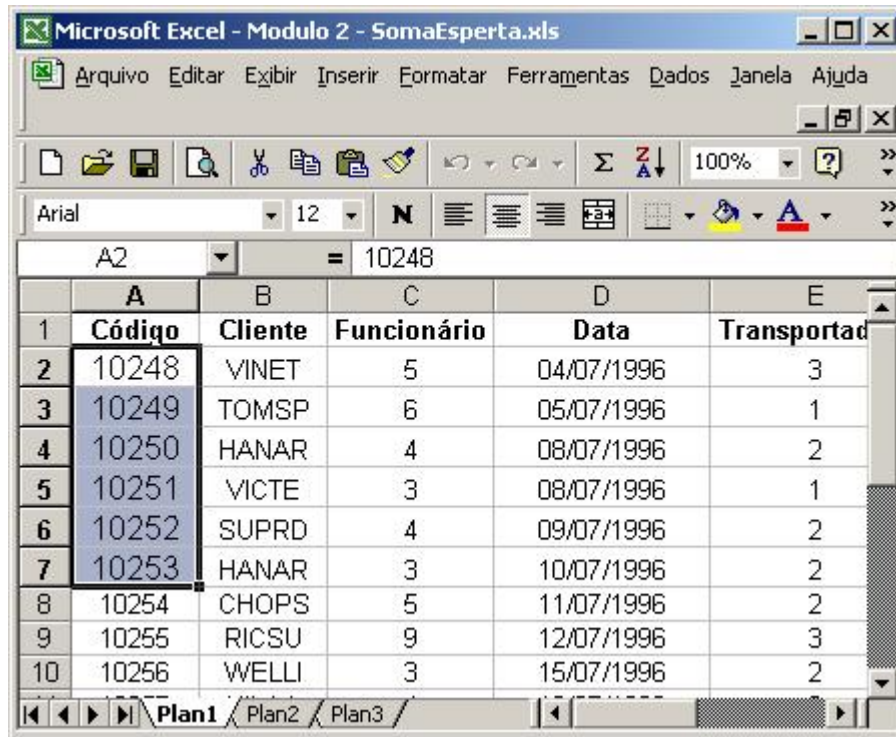


	A	B	C	D	E
1	Código	Cliente	Funcionário	Data	Transportad
2	10248	VINET	5	04/07/1996	3
3	10249	TOMSP	6	05/07/1996	1
4	10250	HANAR	4	08/07/1996	2
5	10251	VICTE	3	08/07/1996	1
6	10252	SUPRD	4	09/07/1996	2
7	10253	HANAR	3	10/07/1996	2
8	10254	CHOPS	5	11/07/1996	2
9	10255	RICSU	9	12/07/1996	3
10	10256	WELLI	3	15/07/1996	2
11	10257	HILAA	4	16/07/1996	3

15. Para continuar a execução do código pressione a tecla F5.
16. A terceira chamada da função é executada e o procedimento é encerrado.

`MudaSelecao 12, Arial`

14. O resultado desta segunda chamada é indicada na figura a seguir, onde você pode observar que apenas as células previamente selecionadas tiveram a sua formatação alterada para fonte tamanho 12 e tipo Arial:



Muito bem, com isso você pode ver, na prática, a declaração e utilização de parâmetros opcionais em um procedimento do VBA. Na próxima lição, mostrarei uma maneira diferente de executar o procedimento `MudaSelecao`, onde os valores para os argumentos serão informados diretamente em células da planilha.

Lição 16: Conceitos Avançados na Criação de Funções e Procedimentos

Nesta lição aprenderemos sobre os chamados argumentos nomeados. Você deve ter observado, dos exemplos anteriores, que ao definir uma lista de argumentos para uma função/procedimento, estes argumentos tem que ser informados, quando da chamada da função/procedimento, na mesma ordem em que foram declarados. Como alternativa tínhamos os argumentos opcionais, normalmente declarados no final da lista, e que, por serem opcionais, poderão ser omitidos na chamada da função/procedimento, conforme exemplo da Lição 14.

Considere o exemplo a seguir:

```
Public Function CalculaImposto(BaseDeCálculo As Double, Percentual As Double)
```

Esta função tem dois parâmetros, obrigatórios, os quais devem ser chamados, na ordem em que foram declarados. A seguir temos um exemplo, onde o valor das variáveis Base e Taxa foram passadas na chamada da função:

```
ParaPagar = CalculaImposto(Base, Taxa)
```

Este tipo de argumento é conhecido como Argumento Posicional, pois é a posição do argumento, na chamada da função/procedimento, que define a qual argumento se refere o valor passado. No exemplo anterior, o valor da variável Base, será atribuído ao argumento BaseDeCálculo devido á sua posição: primeira na chamada, que corresponde ao primeiro argumento declarado na criação da função. O valor da variável Taxa, será atribuído ao argumento Percentual, que corresponde ao segundo argumento declarado na criação da função. Observem que o nome do argumento declarado pode ser diferente do nome da variável passada como parâmetro. Poderíamos nem mesmo utilizar uma variável, mas sim diretamente passar valores como parâmetros, conforme exemplo a seguir:

```
ParaPagar = CalculaImposto(35230, 27.5)
```

O VBA permite a utilização dos chamados Argumentos nomeados, ou seja, é possível atribuir um nome para um ou mais argumentos. A vantagem de utilizar argumentos nomeados é que, quando da chamada da função/procedimento, podemos passar os argumentos em qualquer ordem, desde que informemos o nome do argumento. A sintaxe para utilização de argumentos nomeados é um pouco diferente da sintaxe para argumentos padrão e será discutida nesta lição.

Utilizando Argumentos Nomeados:

O uso de Argumentos nomeados é especialmente útil em funções/procedimentos que tenham um grande número de argumentos. Neste caso fica muito difícil para lembrar a ordem exata dos argumentos, o que acaba fazendo com que sejam gerados muitos erros na hora de utilizar a função/argumento.

Não existe diferença entre declarar os argumentos como fizemos nos exemplos anteriores e para utilizá-los como argumentos nomeados. Em outras palavras, a maneira de declarar argumentos é sempre a mesma, como no exemplo a seguir:

Autor: Júlio Cesar Fabris Battisti

Site: www.juliobattisti.com.br

Confira também o livro: "Windows Server 2003 – Curso Completo, 1568 páginas"

Página 170 de 527


```
Public Sub MudaSelecao(TamFonte As Integer, Optional NomeFonte As Variant)
```

Na hora de utilizar esta função, temos duas opções, conforme descrito a seguir.

Usar argumentos posicionais: Os parâmetros devem ser passados na mesma ordem em que os argumentos foram declarados, conforme o exemplo a seguir:

```
MudaSelecao 12, Arial
```

Se fizéssemos uma chamada invertendo a ordem, seria gerada uma mensagem de erro. O exemplo a seguir, onde invertei os valores dos parâmetros, irá gerar um erro, pois o primeiro parâmetro – TamFonte é do tipo Integer e não pode receber um valor de Texto (Arial no exemplo):

```
MudaSelecao Arial, 12
```

Esta chamada iria gerar a mensagem de erro indicada a seguir:

```
Public Sub ExMudaSelecao()  
  
    MudaSelecao TamFonte:=14, NomeFonte:=Verdana  
    Stop  
    MudaSelecao 8  
    Stop  
    MudaSelecao Arial, 12  
  
End Sub
```



Usar argumentos nomeados: A sintaxe para usar argumentos nomeados é informar o nome do argumento, seguido de um sinal de dois pontos e igual: := e o valor do argumento. Ao usar argumentos nomeados, os argumentos podem ser passados em qualquer ordem e o nome informado deve ser o mesmo nome utilizado na declaração do argumento. No exemplo a seguir, temos dois usos de argumentos nomeados. No primeiro além do nome, os argumentos são passados na mesma ordem em que foram declarados na função. Já no segundo exemplo, os argumentos são passados em uma ordem diferente. Nos dois casos a função será executada, sem problemas:

```
Public Sub MudaSelecao(TamFonte As Integer, Optional NomeFonte As Variant)
```

Utilizando argumentos nomeados e na mesma ordem da declaração:

```
MudaSelecao TamFonte:=12, NomeFonte:=12
```

Utilizando argumentos nomeados e em uma ordem diferente da declaração:

```
MudaSelecao NomeFonte:=12, TamFonte:=12
```


Lição 17: Conceitos Avançados na Criação de Funções e Procedimentos

Você já aprendeu que ao declarar uma função ou procedimento, é possível declarar um ou mais argumentos. Ao utilizar uma função ou procedimento, você deve informar os valores para os argumentos, na mesma ordem em que foram definidos durante a declaração da função ou procedimento. Este processo é conhecido como passagem de parâmetros para a função ou procedimento, ou seja, ao chamar a função ou procedimento, passamos valores que serão utilizados pelo código da função ou procedimento.

Existem duas maneiras diferentes de fazer a passagem dos parâmetros e é importante para o programador, entender exatamente a diferença entre estas duas maneiras:

- ▶ Passagem por Valor - ByVal
- ▶ Passagem por Referência - ByRef

Este conceito é muito importante e vamos entendê-lo através de alguns exemplos simples. Inicialmente vamos ver o que acontece quando utilizamos o tipo de passagem padrão, que é o tipo ByVal, onde apenas o valor do parâmetro é passado para a função/procedimento.

Passagem de parâmetros por Valor – ByVal:

Este é o método padrão, ou seja, ao declarar os argumentos de uma função/procedimento, não é preciso usar a palavra ByVal, pois automaticamente, os argumentos assumem a opção ByVal. As duas declarações a seguir são equivalentes:

```
Public Sub DobraValor(Num As Integer)
```

Ou

```
Public Sub DobraValor(ByVal Num As Integer)
```

Na segunda declaração, explicitamente, estou utilizando a opção ByVal, para indicar que o parâmetro Num será passado por valor. Mas o que significa a passagem de um parâmetro por valor??

Significa que o procedimento receberá apenas o valor do parâmetro e não uma referência ao endereço de memória onde está armazenado o valor do parâmetro. Com isso, quaisquer alterações que sejam feitas no valor do parâmetro, dentro do procedimento, não afetarão o valor original, o qual será o mesmo de antes da chamada da função. Em resumo, apenas o valor é passado para a função/procedimento, este valor é utilizado pelo código da função/procedimento, sem afetar o valor original do parâmetro.

Para exemplificar este conceito, considere o código a seguir, onde tenho uma função principal chamada SubTeste, na qual declaro uma variável MeuValor, do tipo Integer, inicializo esta variável e passo ela, por valor, como parâmetro para o procedimento DobraValor:

```
Public Sub SubTeste()  
  
Dim MeuValor As Integer  
MeuValor = 5  
MsgBox "Valor original de MeuValor: " & MeuValor  
  
' Chamo o procedure DobraValor  
DobraValor (MeuValor)  
  
MsgBox "Valor de MeuValor Após a Execução: " & MeuValor  
  
End Sub  
  
Public Sub DobraValor(ByVal Num As Integer)  
  
    MsgBox "Valor recebido como parâmetro:" & Num  
    Num = Num * 2  
    MsgBox "Valor duplicado:" & Num  
  
End Sub
```

A seguir apresento uma descrição, passo-a-passo, da execução do procedimento SubTeste:

1. Inicialmente é feita a declaração da variável MeuValor como sendo do tipo Integer, esta variável é inicializada com o valor 5 e é exibida uma caixa de mensagens com o valor desta variável. Isso é feito pelas linhas de código a seguir:

```
Dim MeuValor As Integer  
MeuValor = 5  
MsgBox "Valor original de MeuValor: " & MeuValor
```

O resultado destas três linhas de código está indicado a seguir:



2. Em seguida é feita uma chamada ao procedimento DobraValor, passando como parâmetro o valor da variável MeuValor. Observe que o que é passado para o procedimento DobraValor é apenas o valor contido em MeuValor (por isso que é uma passagem por Valor, ou seja do tipo ByVal); não é passada uma referência ao endereço de MeuValor na memória do computador. Com a passagem por valor, o procedimento DobraValor apenas poderá utilizar o valor recebido, em seus cálculos, mas não poderá alterar o valor da variável MeuValor, na memória do computador. Em resumo, o procedimento DobraValor, recebe o valor 5, o qual fica associado ao argumento Num do procedimento. A linha a seguir indica a chamada do procedimento DobraValor:

```
DobraValor (MeuValor)
```

3. Ao fazer uma chamada ao procedimento DobraValor, a execução se desloca para a primeira linha de código deste procedimento. Na chamada, o valor da variável MeuValor foi passado como parâmetro. Este valor será associado ao argumento Num, do procedimento DobraValor, ou seja, dentro do procedimento DobraValor $\text{Num} = 5$. Inicialmente será executado o primeiro comando do procedimento DobraValor:

```
MgBox "Valor recebido como parâmetro:" & Num
```

Este comando produz a mensagem indicada a seguir:



esta mensagem confirma o fato de o valor 5 estar associado com o argumento Num, durante a execução do procedimento DobraValor.

4. Em seguida, o valor de Num é multiplicado por 2 e o resultado armazenado em Num:

```
Num = Num * 2
```

5. O próximo passo é exibir o valor de Num, ainda dentro do procedimento DobraValor:

```
MsgBox "Valor duplicado:" & Num
```

Este comando produz a mensagem indicada a seguir:



este resultado confirma que a variável Num foi duplicada. Mas como estaria o valor original da variável MeuValor? É o que você descobrirá no item a seguir.

6. Após encerrada a execução do procedimento DobraValor, a execução é deslocada de volta para o procedimento SubTeste, para primeira linha após a chamada do procedimento DobraValor()

```
MsgBox "Valor de MeuValor Após a Execução: " & MeuValor
```

Este comando produz a mensagem indicada a seguir:



Observe que o valor da variável `MeuValor` não foi alterado pela execução do procedimento `DobraValor`. Isso confirma a passagem de parâmetro por valor, ou seja, o procedimento `DobraValor` recebeu apenas o valor da variável `MeuValor`. Com isso, os comandos do procedimento `DobraValor` não irão alterar o valor da variável `MeuValor`, pois o procedimento não tem acesso ao endereço de memória da variável `MeuValor`, ao contrário, tem acesso apenas a uma cópia do valor desta variável.

Na prática, ao passar um parâmetro por valor, você passa apenas uma cópia do valor da variável e é nesta cópia que a função/procedimento chamado trabalha, sem afetar o valor original da variável passada como parâmetro.

Passagem de parâmetros por Referência – ByRef:

Para poder atuar/alterar o valor original, a função/procedimento, tem que receber o parâmetro por referência – `ByRef`, ou seja, a função/procedimento tem que receber uma referência ao endereço de memória da variável passada como parâmetro e não uma simples cópia do valor da variável (que é o que acontece na passagem `ByVal`). Ao receber um parâmetro por referência (`ByRef`), as alterações que a função/procedimento fizer, serão feitas diretamente na variável original, pois agora, a função/procedimento tem acesso ao endereço da variável na memória e não mais apenas uma cópia do seu valor. Para que um procedimento possa receber um parâmetro por referência, você deve utilizar a palavra `ByRef`, conforme o exemplo a seguir:

```
Public Sub DobraValor(ByRef Num As Integer)
```

Para exemplificar este conceito, considere o código a seguir, onde tenho uma função principal chamada `SubTeste`, na qual declaro uma variável `MeuValor`, do tipo `Integer`, inicializo esta variável e passo ela, por referência, como parâmetro para o procedimento `DobraValor`:

```
Public Sub SubTeste()  
  
Dim MeuValor As Integer  
MeuValor = 5  
MsgBox "Valor original de MeuValor: " & MeuValor  
  
' Chamo o procedure DobraValor  
Call DobraValor(MeuValor)  
  
MsgBox "Valor de MeuValor Após a Execução: " & MeuValor  
  
End Sub
```

```
Public Sub DobraValor(ByRef Num As Integer)

    MsgBox "Valor recebido como parâmetro:" & Num
    Num = Num * 2
    MsgBox "Valor duplicado:" & Num

End Sub
```

Observe que a única diferença neste código, em relação ao exemplo anterior, é o uso de ByRef, na declaração do argumento Num, do procedimento DobraValor.

A seguir apresento uma descrição, passo-a-passo, da execução do procedimento SubTeste:

1. Inicialmente é feita a declaração da variável MeuValor como sendo do tipo Integer, esta variável é inicializada com o valor 5 e é exibida uma caixa de mensagens com o valor desta variável. Isso é feito pelas linhas de código a seguir:

```
Dim MeuValor As Integer
MeuValor = 5
MsgBox "Valor original de MeuValor: " & MeuValor
```

O resultado destas três linhas de código está indicado a seguir:



2. Em seguida é feita uma chamada ao procedimento DobraValor, passando como parâmetro o valor da variável MeuValor. Observe que o que é passado para o procedimento DobraValor é uma referência (devido ao uso de ByRef na declaração do procedimento DobraValor) a variável MeuValor. Com isso, as alterações feitas pelo procedimento DobraValor, irão afetar o valor original da variável, conforme comprovaremos nos próximos passos.

```
DobraValor (MeuValor)
```

3. Ao fazer uma chamada ao procedimento DobraValor, a execução se desloca para a primeira linha de código deste procedimento. Na chamada, o valor da variável MeuValor foi passado como parâmetro. Este valor será associado ao argumento Num, do procedimento DobraValor, ou seja, dentro do procedimento DobraValor Num = 5. Inicialmente será executado o primeiro comando do procedimento DobraValor:

```
MgBox "Valor recebido como parâmetro:" & Num
```

Este comando produz a mensagem indicada a seguir:



esta mensagem confirma o fato de o valor 5 estar associado com o argumento Num, durante a execução do procedimento DobraValor.

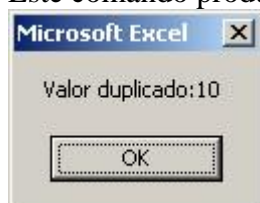
4. Em seguida, o valor de Num é multiplicado por 2 e o resultado armazenado em Num, o que na prática está também afetando o valor da variável MeuValor, conforme você comprovará mais adiante:

```
Num = Num * 2
```

5. O próximo passo é exibir o valor de Num, ainda dentro do procedimento DobraValor:

```
MsgBox "Valor duplicado:" & Num
```

Este comando produz a mensagem indicada a seguir:



este resultado confirma que a variável Num foi duplicada. Mas como estaria o valor original da variável MeuValor? É o que você descobrirá no item a seguir.

6. Após encerrada a execução do procedimento DobraValor, a execução é deslocada de volta para o procedimento SubTeste, para primeira linha após a chamada do procedimento DobraValor()

```
MsgBox "Valor de MeuValor Após a Execução: " & MeuValor
```

Este comando produz a mensagem indicada a seguir:



Observe que o valor da variável MeuValor foi alterado pela execução do procedimento DobraValor. Isso confirma a passagem de parâmetro por referência, ou seja, o procedimento DobraValor recebeu uma referência ao endereço de memória da variável MeuValor. Com isso, os comandos do procedimento DobraValor irão alterar o valor da variável MeuValor, pois o procedimento tem acesso ao endereço de memória da variável MeuValor, ao contrário, do que acontece na passagem por valor, quando o procedimento tem acesso apenas a uma cópia do valor da variável passada como parâmetro.

Lição 18: A função MsgBox em Detalhes

Em diversos exemplos das lições anteriores, utilizamos a função MsgBox para exibir uma janela com uma mensagem. Sempre utilizamos a função MsgBox de uma maneira básica, para exibição de trechos de texto e valor de variáveis, concatenando as partes da mensagem com o operador &. Nesta lição faremos um estudo mais detalhado da função MsgBox.

O funcionamento básico da função MsgBox é exibir uma mensagem em uma caixa de diálogo e aguardar que o usuário clique em um botão. A função retorna um valor do tipo Integer, valor este que indica qual botão o usuário clicou.

Sintaxe:

MsgBox prompt, buttons, title, helpfile, context

Na tabela a seguir descrevo os argumentos para a função MsgBox:

Parâmetro	Descrição
prompt	Obrigatória. É um texto passado entre aspas e que é exibido como mensagem na caixa de diálogo. O comprimento máximo é de aproximadamente 1.024 caracteres, dependendo da largura dos caracteres utilizados. Se o texto consistir em mais de uma linha, você poderá separar as linhas utilizando um caractere de retorno de carro Chr(13), um caractere de alimentação de linha Chr(10) ou uma combinação de caracteres de retorno de carro e alimentação de linha Chr(13) & Chr(10), ao final de cada linha.
buttons	Opcional. É um valor numérico (veja tabela a seguir), que é a soma de valores que especifica o número e o tipo de botões a exibir, o estilo de ícone a utilizar, a identidade do botão padrão e a modalidade da caixa de mensagem. Se omitido, o valor padrão para buttons é 0.
title	Opcional. É um texto que será exibido na barra de título da caixa de diálogo. Se você omitir title, o nome do aplicativo será inserido na barra de título. No caso do Excel, será exibido Microsoft Excel, conforme exemplos da lição anterior.
helpfile	Opcional. Do tipo texto e identifica o arquivo de Ajuda a ser utilizado para fornecer ajuda sensível ao contexto relativa à caixa de diálogo. Se helpfile for fornecido, context também deverá ser fornecido. Você pode informar o caminho de um arquivo no formato de arquivos de Ajuda do Windows, o qual será aberto se o usuário clicar no botão Ajuda, da caixa de mensagem.
context	Opcional. Expressão numérica que é o número de contexto da Ajuda atribuído ao tópico da Ajuda apropriado por seu autor. Se context for fornecido, helpfile também deverá ser fornecido. Este número é como se fosse um link para um ponto específico, dentro de um arquivo de ajuda

O argumento buttons é um valor numérico. Pode ser fornecido um valor numérico ou podem ser utilizadas uma das constantes descritas na próxima tabela. Por exemplo, se este parâmetro tiver o valor 0, o Excel exibe somente o botão OK, se tiver o valor 1, o Excel exibe os botões OK e Cancelar e assim por diante. Na próxima tabela você encontra uma descrição completa de todos os valores possíveis para este parâmetro, as respectivas constantes e os botões que serão exibidos em cada caso.

Constante	Valor	Descrição
VbOKOnly	0	Exibe somente o botão OK.
VbOKCancel	1	Exibe os botões OK e Cancelar.
VbAbortRetryIgnore	2	Exibe os botões Abortar, Repetir e Ignorar.
VbYesNoCancel	3	Exibe os botões Sim, Não e Cancelar.
VbYesNo	4	Exibe os botões Sim e Não.
VbRetryCancel	5	Exibe os botões Repetir e Cancelar.
vbCritical	16	Exibe o ícone Mensagem crítica.
vbQuestion	32	Exibe o ícone Consulta de aviso.
vbExclamation	48	Exibe o ícone Mensagem de aviso.
vbInformation	64	Exibe o ícone Mensagem de informação.
vbDefaultButton1	0	O primeiro botão é o padrão.
vbDefaultButton2	256	O segundo botão é o padrão.
vbDefaultButton3	512	O terceiro botão é o padrão.
vbDefaultButton4	768	O quarto botão é o padrão.
vbApplicationModal	0	Janela restrita do aplicativo; o usuário deve responder à caixa de mensagem antes de continuar o trabalho no aplicativo atual.
vbSystemModal	4096	Janela restrita de sistema; todos os aplicativos são suspensos até que o usuário responda à caixa de mensagem.
vbMsgBoxHelpButton	16384	Adiciona o botão 'Ajuda' à caixa de mensagens
VbMsgBoxSetForeground	65536	Especifica a janela da caixa de mensagens como a janela de primeiro plano
vbMsgBoxRight	524288	O texto é alinhado à direita
vbMsgBoxRtlReading	1048576	Especifica que o texto deve aparecer como leitura da direita para a esquerda em sistemas hebraico e árabe

Importante: O primeiro grupo de valores (0 a 5) descreve o número e o tipo de botões exibidos na caixa de diálogo; o segundo grupo (16, 32, 48, 64) descreve o estilo de ícone; o terceiro grupo (0, 256, 512) determina qual botão é o padrão e o quarto grupo (0, 4.096) determina a modalidade da caixa de mensagem. Quando estiver somando números para criar um valor final para o argumento buttons, utilize somente um número de cada grupo.

Dependendo do botão no qual o usuário clicou, a função MsgBox retorna diferentes valores. Os valores retornados pela função e as constantes associadas, estão descritas na tabela a seguir:

Constante	Valor	Descrição
VbOK	1	OK
vbCancel	2	Cancelar
vbAbort	3	Abortar
vbRetry	4	Repetir
vbIgnore	5	Ignorar
vbYes	6	Sim
vbNo	7	Não

Observações importantes:

Quando os parâmetros `helpfile` e `context` são fornecidos, o usuário pode pressionar F1 (Windows) ou AJUDA (Macintosh) para visualizar o tópico de Ajuda que corresponde ao `context`. Alguns aplicativos host, por exemplo, o Microsoft Excel, também adicionam automaticamente um botão Ajuda à caixa de diálogo.

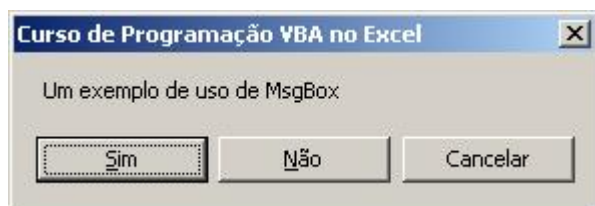
Se a caixa de diálogo exibir um botão Cancelar, pressionar a tecla ESC terá o mesmo efeito que clicar em Cancelar. Se a caixa de diálogo contiver um botão Ajuda, será fornecida a ajuda sensível ao contexto relativa à caixa de diálogo. Entretanto, nenhum valor será retornado até que um dos outros botões seja clicado.

Exemplo de utilização de MsgBox:

No exemplo a seguir, crio uma caixa de mensagem que exibe os botões Sim, Não e Cancelar (valor 3 para o argumento `buttons`), com um título “Curso de Programação VBA no Excel”

MsgBox "Um exemplo de uso de MsgBox", 3, "Curso de Programação VBA no Excel"

Este comando irá produzir a janela indicada a seguir:



O valor retornado pela função depende do botão que for clicado pelo usuário: Clicar em Sim retorna 6, clicar em Não retorna 7 e Cancelar retorna 2, conforme valores da tabela anterior.

Lição 19: A função InputBox em Detalhes

A função InputBox exibe uma janela com uma mensagem e um campo para que o usuário digite um valor ou uma entrada de texto. Também são exibidos botões tais como OK e Cancelar. O valor digitado pelo usuário será o valor de retorno da função. O principal uso da função InputBox é permitir que o usuário digite informações.

Por exemplo, você poderia ter uma planilha que faz uma série de cálculos, os quais dependem da cotação diária do dólar. Neste caso você poderia colocar uma função InputBox, associada ao evento ao Abrir da planilha (um dos assuntos deste módulo será sobre os eventos do Excel), de tal maneira que toda vez que a pasta de trabalho for aberta, será exibida uma caixa de diálogo, solicitando que você digite a cotação atual do dólar em relação ao Real. Com base nesta informação, todas as planilhas são recalculadas e exibem valores atualizados, com base na última cotação fornecida pelo usuário.

Sintaxe: A função InputBox tem a sintaxe genérica indicada a seguir:

InputBox (prompt, title, default, xpos, ypos, helpfile, context)

Na tabela a seguir descrevo os argumentos para a função MsgBox:

Parâmetro	Descrição
Prompt	Obrigatório. É um texto passado entre aspas e que é exibido como mensagem na caixa de diálogo. O comprimento máximo é de aproximadamente 1.024 caracteres, dependendo da largura dos caracteres utilizados. Se o texto consistir em mais de uma linha, você poderá separar as linhas utilizando um caractere de retorno de carro Chr(13), um caractere de alimentação de linha Chr(10) ou uma combinação de caracteres de retorno de carro e alimentação de linha Chr(13) & Chr(10), ao final de cada linha.
Title	Opcional. É um texto que será exibido na barra de título da caixa de diálogo. Se você omitir title, o nome do aplicativo será inserido na barra de título. No caso do Excel, será exibido Microsoft Excel, conforme exemplos da lição anterior.
Default	Opcional. Define um texto padrão que é exibido na caixa de texto da janela da função e que será retornado por padrão, se nenhuma entrada for fornecida. Se você omitir default, a caixa de texto será exibida vazia.
Xpos	Opcional. É um valor numérico que especifica a distância horizontal da borda esquerda da caixa de diálogo em relação à borda esquerda da tela. Se xpos for omitido, a caixa de diálogo será centralizada horizontalmente.
Ypos	Opcional. É um valor numérico que especifica a distância vertical da borda superior da caixa de diálogo em relação ao alto da tela. Se ypos for omitido, a caixa de diálogo será posicionada verticalmente na terça parte inferior da tela.
Helpfile	Opcional. Do tipo texto e identifica o arquivo de Ajuda a ser utilizado para fornecer ajuda sensível ao contexto relativa à caixa de diálogo. Se helpfile for fornecido, context também deverá ser fornecido. Você pode informar o caminho de um arquivo no formato de arquivos de Ajuda do Windows, o qual será aberto se o usuário clicar no botão Ajuda, da caixa de mensagem.

Context	Opcional. Expressão numérica que é o número de contexto da Ajuda atribuído ao tópico da Ajuda apropriado por seu autor. Se context for fornecido, helpfile também deverá ser fornecido. Este número é como se fosse um link para um ponto específico, dentro de um arquivo de ajuda
---------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Quando helpfile e context são fornecidos, o usuário pode pressionar F1 (Windows) ou AJUDA (Macintosh) para visualizar o tópico de Ajuda que corresponde ao context. Alguns aplicativos host, por exemplo, o Microsoft Excel, também adicionam automaticamente um botão Ajuda à caixa de diálogo. Se o usuário clicar em OK ou pressionar , a função InputBox retornará o que estiver na caixa de texto. Se o usuário clicar em Cancelar, a função retornará uma sequência de caracteres de comprimento zero ("").

Exemplo de utilização de InputBox:

Considere o exemplo a seguir, onde utilizo a função InputBox para solicitar que o usuário digite um valor para a cotação do dólar. O valor digitado é atribuído a variável Cotacao e depois é exibido, usando a função MsgBox. Observe também o uso de um laço Do...Loop, para continuar exibindo a função, até que o usuário digite um valor válido, ou seja, um valor numérico. Se o usuário clicar em Cancelar (valor de retorno da função vazia " "), o procedimento é encerrado. Esta verificação evita que o usuário digite um valor de texto e simplesmente seja gerado um erro.

```
Public Sub ObtemCotacao()  
  
    ' Curso de Programação VBA no Excel  
    ' Autor: Júlio Battisti  
    ' Site: www.juliobattisti.com.br  
    ' e-mail: batisti@hotmail.com  
  
    ' Declaração das variáveis  
  
    Dim EntradaDoUsuario As Variant  
    Dim Cotacao As Double  
  
    ' Laço Do que exibe a caixa de mensagem, até que o usuário digite um valor válido  
  
    Do  
  
        ' Exibo a caixa de diálogo usando InputBox  
  
        EntradaDoUsuario = InputBox("Digite a cotação do dólar:", "Cotação do dólar!!!")  
  
        ' Verifico se o usuário clicou em Cancelar. Caso afirmativo,  
        ' encerro o procedimento usando Exit Sub  
  
        If EntradaDoUsuario = "" Then Exit Sub  
  
        ' Verifico se o valor digitado é válido, ou seja, se é um valor numérico  
        ' se for converto o valor de cotacao para Double e  
        ' utilizo o comando Exit do, para sair do laço Do...Loop
```

```
If IsNumeric(EntradaDoUsuario) Then
    Cotacao = CDbI(EntradaDoUsuario)
    MsgBox "Você digitou o seguinte valor: " & Cotacao
    Exit Do
End If

' Se a entrada não for válida, os comandos a seguir serão executados
' Se a entrada for válida, o comando Exit Do anterior, irá encerrar
' o laço Do...Loop e os comandos a seguir não serão executados.

MsgBox "Entrada inválida, clique em OK e digite novamente"
Loop

End Sub
```

Ao ser executado este procedimento, inicialmente será exibida uma caixa de mensagem com um campo para que o usuário digite a cotação do dólar, conforme indicado na figura a seguir:



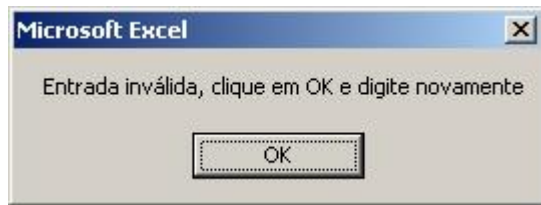
Se o usuário clicar em Cancelar, a função retorna "" e o procedimento será encerrado, pelo comando Exit Sub.

```
If EntradaDoUsuario = "" Then Exit Sub
```

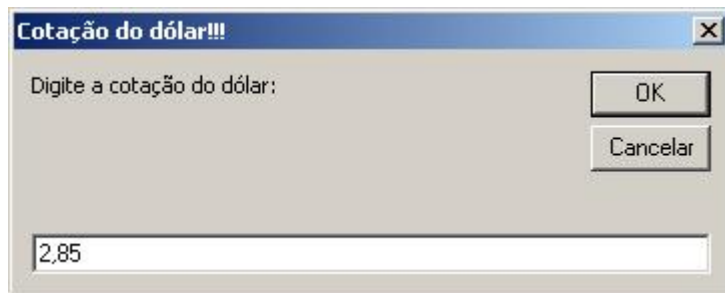
Se o usuário digitar um valor inválido, como por exemplo um valor de texto, conforme indicado na figura a seguir:



, será exibida uma mensagem informando que o valor digitado é inválido, conforme indicado na figura a seguir:



Ao clicar em OK, a caixa para digitação da cotação será exibida novamente. Este processo irá continuar até que o usuário clique em Cancelar ou digite um valor válido. Se o usuário digitar um valor válido, conforme indicado na figura a seguir:



, o valor será processado corretamente, será exibida uma mensagem confirmando o valor digitado (veja figura a seguir) e o procedimento será encerrado.



Muito bem, nesta lição você aprendeu a utilizar a função InputBox e teve um exemplo prático do uso de um laço Do...Loop para fazer uma verificação do valor digitado e continuar a exibir a caixa de mensagem para digitação do valor, até que o usuário clique em Cancelar ou entre com um valor válido. Este é um exemplo que pode ser facilmente adaptado para uso nas aplicações que você irá criar usando o Excel e o VBA.

A partir da próxima lição passaremos a estudar o tratamento de erros no VBA.

Lição 20: O Tratamento de Erros no VBA – Parte 1

Existem diferentes tipos de erros que podem ser gerados em uma rotina VBA.

Erros de sintaxe: O primeiro e mais óbvio é o erro de sintaxe, onde um comando é digitado incorretamente. Por exemplo, ao invés de uma vírgula (,) você digita um ponto-e-vírgula e vice-versa. Este tipo de erro é o mais simples de detectar, pois o próprio ambiente de programação do VBA emite uma mensagem, quando um comando é digitado incorretamente (com erro de sintaxe), conforme exemplo da figura a seguir, onde digitei um comando If...Then incorretamente:



Erros de lógica: Um segundo tipo de erro, este o mais difícil de ser detectado, são os chamados erros lógicos. Este tipo de erro é difícil de ser detectado porque o programa executa normalmente, sem ser emitida nenhuma mensagem de erro, porém os resultados obtidos estão incorretos. Este tipo de erro pode ter inúmeras e diferentes causas. Por exemplo, você pode ter inserido uma fórmula incorretamente, pode ter errado na criação de um teste condicional, trocando um sinal de maior (>) por um de menor (<) ou vice-versa. Para localizar este tipo de erro você tem que utilizar os recursos de execução passo-a-passo, disponíveis no ambiente de edição do VBA, os quais serão descritos nas próximas lições.

Erros de compilação: Alguns erros não tem como ser detectados pelo editor do VBA como sendo erros de sintaxe, pois são comandos que não estão em uma única linha. Por exemplo, você pode ter iniciado um laço Do, sem ter encerrado o laço com um Loop ou pode ter iniciado um laço For...Each, sem ter encerrado o laço com um comando Next. Estes erros somente são detectados quando o código é executado, ou seja, durante a primeira compilação do código VBA. Antes de executar, o VBA compila o código, ao compilar detecta o erro e exibe uma mensagem, como no exemplo a seguir, onde mostro um erro gerado pelo uso de uma estrutura For...Each, sem o fechamento da estrutura com o uso de um Next.



Neste caso, inclusive a mensagem é bem clara (o que é uma exceção em se tratando de mensagens de erros no Excel, as quais normalmente não são muito esclarecedoras).

Erros em tempo de execução: Existem determinados tipos de erros que podem ocorrer dependendo de condições externas. Por exemplo, se você criou uma rotina VBA para importar dados de um arquivo de texto para uma planilha do Excel e este arquivo de texto fica disponível em um drive de rede. Ao executar esta rotina, o computador deve ter acesso ao referido drive de rede. Se o computador tiver acesso ao drive de rede, a rotina será executada normalmente. Se por algum motivo o computador perder o acesso a rede e a rotina for executada enquanto o computador estiver sem acesso à rede, será gerado um erro em tempo de execução. Outro exemplo de erro em tempo de execução é de uma rotina que foi programada para ler dados em um arquivo no disquete e a rotina é executada sem que um disquete esteja inserido no drive. Estes erros são chamados de erros em tempo de execução, porque somente serão detectados durante a execução do código VBA, não sendo possível detectá-los antes. Os erros em tempo de execução é que são candidatos ao tratamento de erros. Por exemplo, no código VBA, você pode prever o fato de o usuário não ter colocado o disquete no drive. Neste caso, você deve usar um tratamento de erro eficiente, o qual exibe uma mensagem informando que o disquete deve ser inserido no drive e dando a opção de o usuário clicar em OK para continuar, ao invés de simplesmente encerrar o procedimento com uma mensagem de erro. Ou seja, o tratamento de erros é utilizado para contornar situações onde o erro foi gerado devido a uma condição externa que pode ser contornada (como por exemplo o fato de o usuário não ter inserido um disquete no drive).

A Instrução On Error:

Para fazer o tratamento de erros no VBA, utilizamos a instrução On Error. Esta instrução tem diferentes formatos e diferentes opções. Nesta lição apresentaremos os aspectos teóricos desta instrução. Na próxima lição você verá exemplos práticos de utilização da instrução On Error.

A instrução On Error ativa uma rotina de tratamento de erro em um local específico do código dentro de um procedimento; pode ser usada também para desativar uma rotina de tratamento de erro.

Sintaxe geral: Este comando apresenta diferentes formas de uso, conforme indicado a seguir:

- ▶ On Error GoTo line
- ▶ On Error Resume Next
- ▶ On Error GoTo 0

A sintaxe da instrução On Error pode ter qualquer uma das seguintes formas:

Instrução	Descrição
On Error GoTo line	Ativa a rotina de tratamento de erro que inicia na linha especificada pelo argumento line, o qual é obrigatório. O argumento line é qualquer rótulo de linha ou número de linha (veja exemplos na próxima lição). Se ocorrer um erro em tempo de execução, o controle desvia para a linha indicada por line, tornando o tratamento de erro ativo. A linha especificada por line deve estar no mesmo procedimento em que se encontra a instrução On Error; caso contrário, ocorrerá um erro em tempo de compilação.

On Error Resume Next	Especifica que, quando ocorrer um erro em tempo de execução, o controle passará para a instrução imediatamente seguinte à instrução onde ocorreu o erro e a execução continua. Esta instrução é utilizada quando você simplesmente quer que sejam ignorados os erros de execução.
On Error GoTo 0	Desativa qualquer manipulador de erro ativo no procedimento atual.

Se você não usar uma instrução On Error, qualquer erro em tempo de execução que ocorrer será fatal, isto é, será exibida uma mensagem de erro e a execução do procedimento será encerrada. Conforme descrito anteriormente, no exemplo do erro devido a um disquete que não está no drive, deixar simplesmente que a execução seja encerrada, com a exibição de uma mensagem de erro padrão, não é uma solução das “mais elegantes”.

Um manipulador de erro "ativado" é o que foi ativado por uma instrução On Error; um manipulador de erro "ativo" é um manipulador ativado que está no processo de tratar um erro. Se ocorrer um erro enquanto o manipulador de erro estiver ativo (entre a ocorrência do erro e uma instrução Resume, Exit Sub, Exit Function ou Exit Property), o manipulador de erro do procedimento atual não poderá tratar o erro. O controle retornará para o procedimento de chamada. Se o procedimento de chamada possuir um tratamento de erro ativado, ele será ativado para tratar o erro. Se o manipulador de erro do procedimento de chamada também estiver ativo, o controle passará de volta pelos procedimentos de chamada anteriores até encontrar um manipulador de erro ativado, mas inativo. Se não for encontrado nenhum manipulador de erro ativado e inativo, o erro será fatal (a execução do código será encerrado) no ponto em que ele realmente ocorreu. Sempre que o manipulador de erro passa o controle de volta para um procedimento de chamada, esse procedimento torna-se o procedimento atual. Uma vez que o erro é tratado por um manipulador de erro em qualquer procedimento, a execução continua no procedimento atual no ponto designado pela instrução Resume.

Observação: Uma rotina de tratamento de erro não é um procedimento Sub ou Function. É uma seção do código marcada por um rótulo ou número de linha.

As rotinas de tratamento de erro dependem do valor na propriedade Number do objeto Err para determinar a causa do erro. A rotina de tratamento de erro deve testar ou salvar valores de propriedade relevantes do objeto Err antes que qualquer outro erro possa ocorrer ou antes que um procedimento que possa causar um erro seja chamado. Os valores de propriedade no objeto Err refletem somente o erro mais recente. A mensagem de erro associada a Err.Number está contida em Err.Description.

Importante: O comando **On Error Resume Next** faz com que a execução continue com a instrução imediatamente após a instrução que causou o erro em tempo de execução ou continue com a instrução imediatamente após a chamada mais recente do procedimento que contém a instrução On Error Resume Next. Essa instrução permite que a execução continue apesar de um erro em tempo de execução. Você pode colocar a rotina de tratamento de erro onde é possível a ocorrência de erro, em vez de transferir o controle para outro local dentro do procedimento. Uma instrução On Error Resume Next torna-se inativa quando outro procedimento for chamado, assim você deve executar uma instrução On Error Resume Next em cada rotina chamada para tratar os erros em linha dentro dessa rotina.

Importante: On Error GoTo 0 desativa o tratamento de erro no procedimento atual. Ele não especifica a linha 0 como início do código de tratamento de erro, mesmo que esse procedimento contenha uma linha com o número 0. Sem a instrução On Error GoTo 0, um tratamento de erro é automaticamente desativado ao sair do procedimento.

Para evitar que o código de tratamento de erro seja executado quando não ocorreram erros, coloque uma instrução Exit Sub, Exit Function ou Exit Property imediatamente antes da rotina de tratamento de erro, como no exemplo a seguir:

```
Sub InitializeMatrix(Var1, Var2, Var3, Var4)
    On Error GoTo TrataErro
    . . .
    Exit Sub
TrataErro:
    . . .
    Resume Next
End Sub
```

Neste exemplo, o tratamento de erro é ativado pelo seguinte comando:

```
On Error GoTo TrataErro
```

Após a ativação do tratamento de erro são executados os comandos que fazem parte do procedimento. Se um destes comandos gerar um erro em tempo de execução, a execução será deslocada para o comando abaixo do rótulo TrataErro: Observe que a criação de um rótulo é seguida do sinal de dois pontos (:). O tratamento de erro diz o seguinte: “Execute a rotina normalmente, se ocorrer um erro de execução, vá para (GoTo) o comando abaixo do rótulo indicado pela instrução On Error, que no nosso exemplo é TrataErro. Os comandos para o tratamento de erro são executados e a execução continua, a partir da primeira linha após a linha que gerou o erro.

Se não ocorrer nenhum erro, as linhas abaixo de On Error... são executadas uma a uma. Observe que o último comando é um Exit Sub. Este comando é utilizado para sair do procedimento. Se não fosse utilizado este comando, a execução continuaria, e os comandos abaixo de TrataErro: seriam executados também, mesmo sem ter sido gerado nenhum erro. Ou seja, uma vez que não foi gerado erro, utilizo Exit Sub para encerrar o procedimento, sem executar os comandos de tratamento de erros.

Na próxima lição mostrarei mais alguns exemplos práticos de tratamento de erros, usando a instrução On Error e suas diferentes formas de utilização.

Lição 21: O Tratamento de Erros no VBA – Parte 2

Nesta lição apresentarei alguns exemplos práticos e técnicas de utilização da instrução OnError, para o tratamento de Erros.

Exemplo 01: Neste exemplo mostro um trecho de código para o tratamento do erro de execução gerado devido a uma divisão por zero, operação que não é permitida na matemática.

```
Sub TrataErroEx01( )

    Dim x

    ' habilito o tratamento de erro e desloco a execução para a linha indicada com o rótulo
    ' ErroDivisao

    On Error GoTo ErroDivisao

    ' Coloco uma divisão por zero para forçar a ocorrência do erro
    X = 100/0

    ' A execução, após a geração do erro, será deslocada para os comandos após
    ' o rótulo ErroDivisão. Após a execução destes comandos, a execução volta para
    ' o primeiro comando após o comando que gerou o erro, ou seja, o comando a seguir.
    ' no qual utilizo Goto 0 para desabilitar o tratamento de erro e depois um Exit Sub
    ' para encerrar o procedimento.

    On Error GoTo 0

    ' Outros comandos. É muito importante salientar o uso do Exit Sub, pois se
    ' não colocarmos este comando, os comandos de tratamento de erro serão executados,
    ' na seqüência, mesmo que nenhum erro seja gerado.

    Exit Sub

ErroDivisao:

    ' Comandos para o tratamento do erro de divisão por zero.
    ' Normalmente você exibirá uma mensagem informando ao usuário que o valor do
    ' denominador não pode ser zero e solicitando que ele digite um novo valor

End Sub
```

Exemplo 02: Neste exemplo mostro como fazer com que a execução do código, após o tratamento do erro, não siga com o comando imediatamente após o comando que executou o erro, ao invés disso, a execução, após o tratamento do erro, é deslocada para um ponto específico do código.

```
Sub TrataErroDesloca()

    Dim X

    ' habilito o tratamento de erro e desloco a execução para a linha indicada com o rótulo
    ' ErroDivisao

    On Error GoTo ErroDivisao

    ' Coloco uma divisão por zero para forçar a ocorrência do erro
    X = 100/0

    ' A execução, após a geração do erro, será deslocada para os comandos após
    ' o rótulo ErroDivisao. Após a execução destes comandos, o padrão seria a
    ' a execução voltar para o primeiro comando após o comando que gerou o
    ' erro, ou seja, o comando a seguir. Porém, neste exemplo, criamos um segundo
    ' rótulo, chamado ErroDivisao2, para o qual será deslocada a execução após
    ' a execução dos comandos de tratamento de erro

    On Error GoTo 0

ErroDivisao2:

    ' Se ocorrer algum erro nos comandos deste rótulo, vá para o rótulo ErroDivisao3
    ' Caso contrário execute os comandos e encerre o procedimento: Exit Sub.

    On Error GoTo ErroDivisao3
    ' Comandos a serem executados

    Exit Sub

ErroDivisao:
    X = 0
    Resume ErroDivisao2

ErroDivisao3:

    ' Comandos para o tratamento de erro
End Sub
```

A seguir você acompanha a execução, passo-a-passo, deste exemplo:

1. O procedimento é iniciado, a variável X é declarada, e um erro é forçado, ao fazer a divisão da variável X por 0:

```
X = 100/0
```

2. Como o tratamento de erros foi habilitado, esse erro deslocará a execução para o rótulo ErroDivisao, que foi o rótulo habilitado na primeira instrução On Error do procedimento.

3. O comando abaixo do Rótulo ErroDivisao é executado, definindo o valor da variável X em zero e o comando Resume desloca a execução para o rótulo ErroDivisao2:

```
X = 0  
Resume ErroDivisao2
```

Neste caso o primeiro deslocamento é feito para o rótulo ErroDivisao, para inicialmente fazer um tratamento das causas mais conhecidas do erro, no nosso caso, foi feita a definição de X=0. Em seguida a execução desloca-se para ErroDivisao2.

4. Os comandos de tratamento de erro do rótulo ErroDivisao2 são executados. Se estes comandos não causarem um novo erro, o procedimento será encerrado. Observe que o efeito prático é que a execução, na ocorrência do primeiro erro (X=100/0), após o tratamento do erro, não retorna para o primeiro comando após o comando que gerou o erro, mas sim para um local alternativo, definido pelo rótulo ErroDivisao2. Ou seja, na prática, estamos executando uma parte do código diferente do padrão que seria voltar a execução para a primeira linha após o comando que gerou o erro.

5. Se for gerado algum erro, pelos comandos do rótulo ErroDivisao2, a execução será deslocada para os comandos do rótulo ErroDivisao3.

Importante: Embora esta possa ser uma técnica útil em algumas situações específicas, sempre é recomendado evitar este “vai para cá”, “volta para lá”, “vai de novo” e assim por diante. Um código com muito Go to e deslocamentos fica difícil de interpretar e de acompanhar a execução. Um código deste tipo é, pejorativamente conhecido, como Código Macarrão.

Na próxima lição mostrarei mais alguns exemplos de código para tratamento de erros no VBA.

Lição 22: O Tratamento de Erros no VBA – Parte 3

Nesta lição apresentarei alguns exemplos práticos e técnicas de utilização da instrução OnError, para o tratamento de Erros.

Exemplo 03: Neste exemplo é ilustrado o uso do comando Resume Next. Esta opção é utilizada quando você precisa que, em caso de erro, a execução continue na linha seguinte a que gerou o erro, porém antes disso você quer executar um ou mais comandos. Ou seja, ocorre o erro, a execução é deslocada para um determinado Label e depois volta para o comando seguinte ao que gerou o erro. No exemplo a seguir, se o comando Resume não fosse utilizado, os erros seguintes não seriam processados e tratados. Considere o exemplo a seguir:

```
Sub ExTrataErro()

    Dim x

    ' habilito o tratamento de erro e desloco a execução para a linha indicada com o rótulo
    ' ErroDivisao

    On Error GoTo ErroDivisao

    x = 100/0

    ' A execução, após a geração do erro, será deslocada para os comandos após
    ' o rótulo ErroDivisão. Após a execução dos comandos abaixo do rótulo Erro Divisão,
    ' é executado o primeiro comando após o comando que gerou o erro, que no caso é
    ' justamente o comando a seguir, o qual desabilita o tratamento de erros.

    On Error GoTo 0

    ' Após a execução do comando On Error GoTo 0, os comandos abaixo do rótulo
    ' ErroDivisao2 serão executados. Neste caso observe que o erro é gerado, a execução
    ' é deslocada para os comandos abaixo do rótulo ErroDivisão. Estes comandos são
    ' executados e, devido ao Resume Next, a execução volta para o primeiro comando
    ' após o comando que gerou o erro, no nosso exemplo é o comando
    ' On Error GoTo 0. Depois a execução segue normalmente, com os comandos na
    ' sequência (abaixo de ErroDivisao2), até atingir o Exit sub.

ErroDivisao2:

    'Comandos a serem executados após o tratamento de erro.
    Exit Sub

ErroDivisao:
    x = 0
    Resume Next

End Sub
```


Com esta estrutura atendemos o objetivo inicial, qual seja, permitir que sejam executados alguns comandos, após os comandos de tratamento do erro. Esta estrutura aplica-se ao caso do erro por não ter um disquete no drive. Neste caso, o comando de tratamento de erro pode ser uma mensagem orientando o usuário a inserir um disquete no drive. O usuário insere um disquete e clica em OK. São executados os comandos após o tratamento de erro, que no exemplo anterior, seriam os comandos abaixo de ErroDivisao2, que no exemplo do disquete, seriam os comandos para ler as informações no disquete.

Exemplo 04: Neste exemplo eu mostro como utilizar o comando On Error Resume Next para fazer com que os erros de execução sejam simplesmente ignorados, ou seja, se for gerado um erro de execução, o erro será ignorado e a execução do programa continua com o comando seguinte ao que gerou o erro.

```
Sub TrataErro4()  
  
    Dim x  
    ' Faz com que o VBA ignore os erros de execução.  
  
    On Error Resume Next  
    x = 1/0  
  
    'desabilita o tratamento de erros  
    On Error GoTo 0  
  
End Sub
```

Exemplo 05: Você pode numerar as linhas de código no VBA. Fazendo isso, você pode usar a constante `erl`, conforme exemplo a seguir:

```
Sub LinhasNumeradas()  
    1 a=5  
    2 On Error GoTo TrataErro  
    3 b=6  
    4 MsgBox 5/0      ' Forço um erro, com uma divisão por zero.  
    5 Exit Sub  
  
    TrataErro:  
        MsgBox "Erro na linha: " & erl      ' Uso erl para exibir o número da linha com erro.  
End Sub
```

Ao executar este exemplo, será gerada a mensagem indicada na figura a seguir:



Lição 23: O Tratamento de Erros no VBA – Parte 4

Nesta lição você aprenderá um dos recursos que eu considero mais importantes e mais úteis para a depuração de erros e, principalmente, para a detecção de erros lógicos: “A execução passo-a-passo e o acompanhamento dos valores de variáveis.”

Nesta e na próxima lição você verá exemplos práticos de utilização dos chamados recursos de depuração do editor do VBA. Com o uso destes recursos você pode fazer com que os comandos de um procedimento/função sejam executados um de cada vez. Após a execução de cada comando você pode verificar o valor das variáveis e parâmetros que estão sendo utilizados, para detectar erros lógicos, os quais estão fazendo com que os resultados obtidos não estejam corretos. A seguir farei um exemplo prático, passo-a-passo, onde você aprenderá como utilizar a execução passo-a-passo. Na próxima lição você aprenderá a utilizar o acompanhamento do valor de variáveis. Para aprender a utilizar estes recursos, vou utilizar a função `Calcula_IRPF`, criada na pasta **Módulo 1 – Exercício 01.xls**, na Lição 21 do Módulo 1, conforme indicado a seguir:

```
Public Function Calcula_IRPF(BaseDeCálculo As Currency) As Currency

' Teste para ver se a Base de Cálculo está na faixa de isenção,
' ou seja, menor do que: R$ 12696,00
' Se estiver nesta faixa, o valor do imposto a pagar será R$ 0,00

' Observe que para fazer com que a função retorne um valor,
' atribuímos o valor a ser retornado para uma variável com o mesmo
' nome da função.

If BaseDeCálculo <= 12696 Then
    Calcula_IRPF = 0
End If

' Teste para ver se a Base de Cálculo está na faixa acima de R$ 12696,00
' até R$ 25380,00. Se estiver nesta faixa, o valor do imposto a pagar
' será de 15% da Base de Cálculo, deduzindo a parcela de R$ 1904,40.

If (BaseDeCálculo > 12696) And (BaseDeCálculo <= 25380) Then
    Calcula_IRPF = (BaseDeCálculo * 0.15) - 1904.4
End If

' Teste para ver se a Base de Cálculo está acima de R$ 25380. Se estiver
' o valor do imposto a pagar será de 27,5% da Base de Cálculo, deduzindo
' a parcela de R$ 5076,90.

If (BaseDeCálculo > 25380) Then
    Calcula_IRPF = (BaseDeCálculo * 0.275) - 5076.9
End If

End Function
```

Dentro do mesmo módulo onde está a função, vamos criar um procedimento chamado Principal, o qual faz uma chamada para a função, obtém o valor retornado pela função e exibe uma mensagem com o valor calculado, conforme você verá no exemplo logo a seguir:

Exemplo: Criar o procedimento Principal, o qual chama a função Calcula_IRPF e executá-lo, passo-a-passo, isto é, comando a comando, durante a execução, siga os passos indicados a seguir:

1. Abra o Excel.
2. Abra a Planilha C:\Programação VBA no Excel\Módulo 1 – Exercício 01.xls.
3. Pressione Alt+F11. Será exibido o editor do VBA. No painel da esquerda clique em Módulo 1, para selecioná-lo.
4. Vá para o final do módulo e, após o último End Sub, digite o código do procedimento Principal, conforme indicado a seguir:

```
Public Sub Principal()  
  
    Dim ValorImposto As Double  
  
    ValorImposto = Calcula_IRPF(36714.35)  
  
    MsgBox "O valor do imposto é: " & ValorImposto  
  
End Sub
```

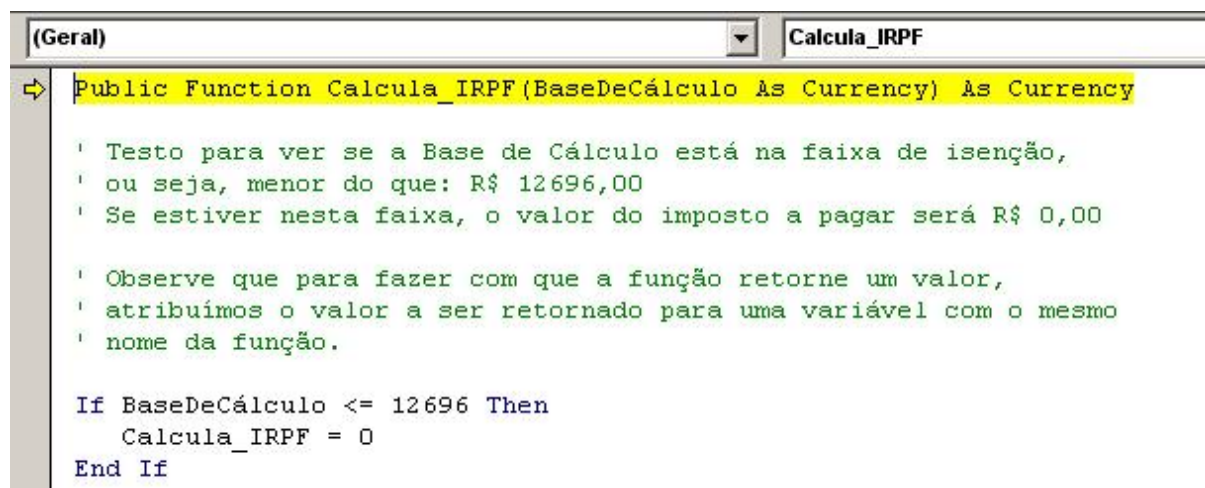
5. Este procedimento basicamente declara uma variável ValorImposto do tipo double e usa uma chamada para a função Calcula_IRPF, passando um valor como parâmetro. O valor retornado pela função Calcula_IRPF é armazenado na variável ValorImposto, valor este que é exibido com um comando MsgBox. O valor é passado com o ponto para separador decimal ao invés da vírgula, pois no código VBA tem que ser o padrão americano, ou seja, ponto ao invés de vírgula, como separador decimal.

6. Agora vamos iniciar a execução passo-a-passo deste procedimento. Clique no nome do procedimento (na palavra Principal), para posicionar o cursor no início deste procedimento.

7. Para iniciar uma execução passo-a-passo, pressione a tecla F8. O primeiro comando será executado e colocado em destaque (em amarelo), conforme indicado na Figura a seguir:

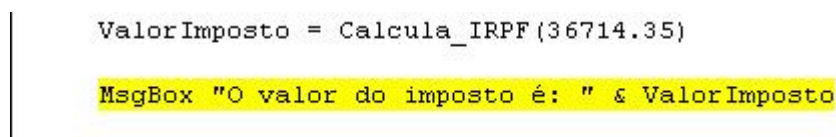
```
Public Sub Principal()  
  
    Dim ValorImposto As Double  
  
    ValorImposto = Calcula_IRPF(36714.35)  
  
    MsgBox "O valor do imposto é: " & ValorImposto  
  
End Sub
```

8. Pressione F8 novamente e observe que o próximo comando (Dim ValorImposto As Double) é colocado em destaque.
9. Pressione F8 novamente. Observe que é feita a chamada à função Calcula_IRPF e a execução se desloca para dentro desta função, conforme indicado na figura a seguir:



```
(Geral) Calcula_IRPF
Public Function Calcula_IRPF(BaseDeCálculo As Currency) As Currency
    ' Testo para ver se a Base de Cálculo está na faixa de isenção,
    ' ou seja, menor do que: R$ 12696,00
    ' Se estiver nesta faixa, o valor do imposto a pagar será R$ 0,00
    ' Observe que para fazer com que a função retorne um valor,
    ' atribuímos o valor a ser retornado para uma variável com o mesmo
    ' nome da função.
    If BaseDeCálculo <= 12696 Then
        Calcula_IRPF = 0
    End If
```

10. Pressione F8 novamente. Observe que as linhas de comentário são ignoradas e a execução passa para o primeiro comando da função, que no nosso exemplo é o teste: If BaseDeCálculo <= 12696. Pressione F8 novamente e observe que o comando dentro do teste é ignorado, isso porque o teste é falso, pois o valor passado como parâmetro para a função Calcula_IRPF foi 36714.35. O comando End If é colocado em destaque.
11. Pressione F8 novamente. Observe que as próximas linhas de comentário são ignoradas e a execução passa para o próximo teste If: If (BaseDeCálculo > 12696) And (BaseDeCálculo <= 25380) Then. Pressione F8 novamente e observe que o comando dentro do teste é ignorado, isso porque o teste é falso, pois o valor passado como parâmetro para a função Calcula_IRPF foi 36714.35. O comando End If é colocado em destaque.
12. Pressione F8 novamente. Observe que as próximas linhas de comentário são ignoradas e a execução passa para o último teste If: If (BaseDeCálculo > 25380) Then. Pressione F8 novamente e observe que o comando dentro do teste é colocado em destaque, pois o teste é verdadeiro. Pressione F8 novamente, o comando é executado e o resultado dos cálculos armazenado em uma variável com o mesmo nome da função, para que esse valor possa retornar ao procedimento Principal.. O comando End If é colocado em destaque.
13. Pressione F8. A função Calcula_IRPF é encerrada e a execução volta para o procedimento Principal, onde é colocado em destaque o primeiro comando abaixo do comando que fez a chamada à função Calcula_IRPF, conforme indicado na figura a seguir:



```
ValorImposto = Calcula_IRPF(36714.35)
MsgBox "O valor do imposto é: " & ValorImposto
```

14. Pressione F8 novamente. O comando MsgBox é executado e os resultados dos cálculos são exibidos, conforme indicado na Figura a seguir:



15. Clique em OK. A mensagem será fechada e o comando End Sub, do procedimento principal será colocado em destaque. Pressione F8 e pronto, será encerrada e execução do procedimento principal, dentro do qual foi feita uma chamada à função Calcula_IRPF.

Observe a importância da execução passo-a-passo, para facilitar o entendimento de como é feita a execução de uma determinada rotina e também para tentar detectar erros de lógica. Na próxima lição mostrarei como exibir o valor de uma ou mais variáveis, a medida que você executa o código passo-a-passo.

Lição 24: O Tratamento de Erros no VBA – Parte 5

Na lição anterior você aprendeu a executar um procedimento passo-a-passo, usando a tecla de função F8. Nesta lição você aprenderá como definir quais variáveis terão seu valor monitorado, a medida que o código é executado passo-a-passo. Este recurso é útil para detectar variáveis que não foram inicializadas corretamente ou para identificar expressões de cálculos com erros lógicos, as quais acabam por gerar resultados incorretos.

Vamos entender como fazer o acompanhamento das variáveis, através de um exemplo prático, no qual utilizarei um procedimento que recebe, como parâmetros, o valor dos lados de um retângulo e calcula e exibe o valor da área, do perímetro e da diagonal do retângulo.

1. Abra o Excel.
2. Abra a Planilha C:\Programação VBA no Excel\Módulo 2 – Exercício 01.xls.
3. Agora vamos iniciar a criação da função Calcula_IRPF.
4. Pressione Alt+F11 para exibir o Editor do VBA.
5. Para criar uma função que possa ser utilizada em qualquer planilha da sua pasta de trabalho (arquivo .XLS), você deve criar a função em um Módulo de Código separado, não pode ser em um dos módulos de código de uma das planilhas ou da pasta de Trabalho. Clique com o botão direito do mouse em VBAProject (Módulo 2 – Exercício 01.xls), no painel da esquerda. No menu que é exibido, clique em Inserir -> Módulo. Será criado um módulo chamado Módulo 1.
6. Dê um clique duplo em Módulo 1, para selecioná-lo. Agora vamos criar o procedimento CalcRetangulo. Digite o código indicado a seguir:

```
Public Sub CalcRetangulo(LadoMenor As Double, LadoMaior As Double)
```

```
    ' Declaração das variáveis utilizadas no procedimento
```

```
    Dim Area, Perimetro, Diagonal As Double
```

```
    ' Cálculo e exibição da área do retângulo
```

```
    Area = LadoMenor * LadoMaior
```

```
    MsgBox "Área do retângulo: " & Area
```

```
    ' Cálculo do perímetro do retângulo
```

```
    Perimetro = (2 * LadoMenor) + (2 * LadoMaior)
```

```
    MsgBox "Perímetro do retângulo: " & Perimetro
```

```
    ' Cálculo e exibição da diagonal do retângulo
```

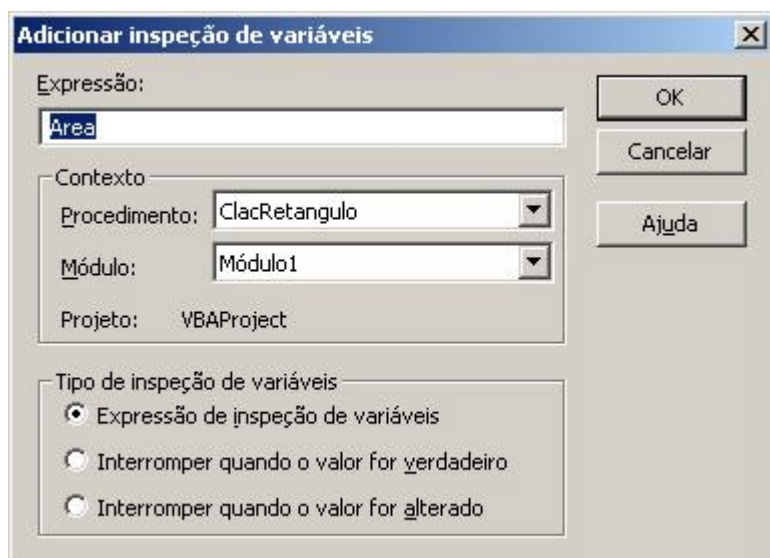
```
    Diagonal = Sqr((LadoMenor ^ 2) + (LadoMaior ^ 2))
```

```
    MsgBox "Diagonal do retângulo: " & Diagonal
```

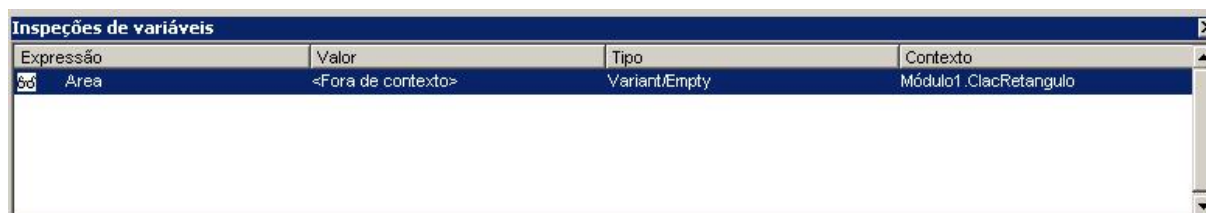
```
End Sub
```

7. Antes de iniciar a execução passo-a-passo, você deve informar quais variáveis serão acompanhadas durante a execução passo-a-passo. Para adicionar uma variável para acompanhamento, siga as orientações dos próximos passos.

8. Dê um clique duplo na variável Area, na linha de declaração da variável (logo após o Dim), para selecioná-la. Execute o comando Depurar - > Adicionar inspeção de variáveis... Será exibida a janela Adicionar inspeção de variáveis, conforme indicado na Figura a seguir:



9. Clique em OK. Pronto, a variável Area foi adicionada à lista de variáveis a ser monitorada durante a execução do código. Uma nova janela é aberta na parte de baixo do editor do VBA: a janela Inspeção de variáveis, na qual já é exibida a variável Area, porém ainda sem valor definido, uma vez que o procedimento ainda não está em execução. Esta janela é indicada na figura a seguir:



10. A janela inspeção de variáveis, por padrão, é exibida na parte de baixo da janela do editor VBA. Você pode deslocar esta janela para qualquer posição na tela. Para isso basta clicar no título da janela (onde está escrito Inspeção de variáveis) e arrastar para a posição desejada. Agora vamos adicionar a inspeção para as demais variáveis do nosso exemplo.

11. Dê um clique duplo na variável Perimetro, na linha de declaração da variável (logo após o Dim), para selecioná-la. Execute o comando Depurar - > Adicionar inspeção de variáveis... Clique em OK. Observe que a variável Perimetro é adicionada na janela Inspeção de variáveis.

12. Dê um clique duplo na variável Diagonal, na linha de declaração da variável (logo após o Dim), para selecioná-la. Execute o comando Depurar - > Adicionar inspeção de variáveis... Clique em OK. Observe que a variável Perimetro é adicionada na janela Inspeção de variáveis.
13. Vamos adicionar, para inspeção, também os parâmetros LadoMaior e LadoMenor. Dê um clique duplo no parâmetro LadoMaior, na linha de declaração da função, para selecioná-lo. Execute o comando Depurar - > Adicionar inspeção de variáveis... Clique em OK. Observe que o parâmetro LadoMaior é adicionada na janela Inspeção de variáveis.
14. Dê um clique duplo no parâmetro LadoMenor, na linha de declaração da função, para selecioná-lo. Execute o comando Depurar - > Adicionar inspeção de variáveis... Clique em OK. Observe que o parâmetro LadoMenor é adicionada na janela Inspeção de variáveis.
15. Mantenha a janela de código aberta. Na próxima lição você irá executar o procedimento CalcRetangulo passo-a-passo e observar como funciona o acompanhamento do valor das variáveis, durante a execução do código.

Lição 25: O Tratamento de Erros no VBA – Parte 6

Nesta lição iremos executar o procedimento CalcRetangulo, passo-a-passo, para acompanharmos como funciona a inspeção de variáveis. Você deve estar com o editor VBA ainda aberto, o qual foi utilizado na última lição.

Exemplo: Executando o procedimento CalcRetangulo passo-a-passo e acompanhando a inspeção de variáveis. Siga os passos indicados a seguir:

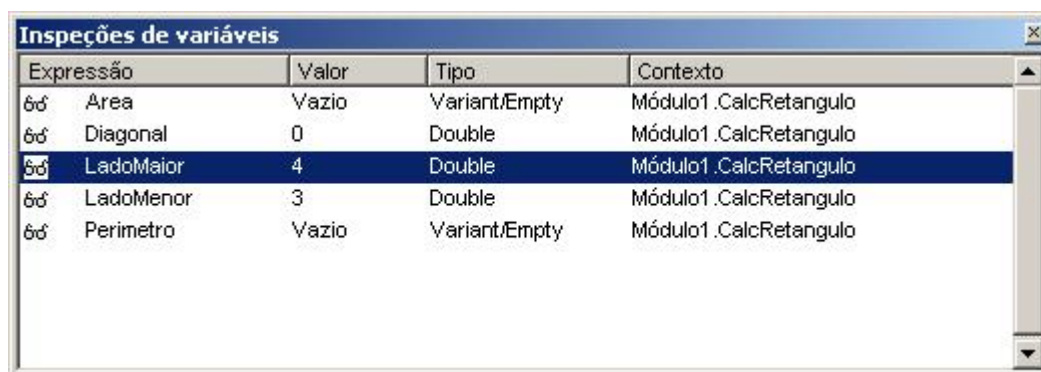
1. Vamos criar um procedimento principal, que faz a chamada ao procedimento CalcRetangulo, definindo valores para os parâmetros LadoMenor e LadoMaior. Após o último End Sub do módulo, digite o código a seguir:

```
Public Sub Principal ( )
```

```
    Call CalcRetangulo(3, 4)
```

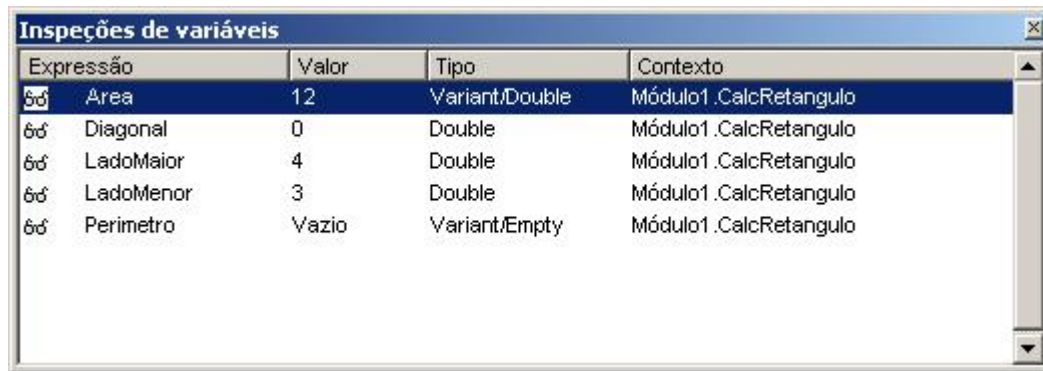
```
End Sub
```

2. Clique no no comando – CalcRetangulo(3,4), para posicionar o cursor neste comando.
3. Pressione F8 para iniciar a execução passo-a-passo.
4. O título do procedimento principal fica em destaque. Pressione F8 novamente. O comando Call CalcRetangulo(3,4) fica em destaque. Pressione F8 novamente, este comando será executado. Observe, na janela Inspeção de variáveis, que os parâmetros LadoMenor e LadoMaior já assumem os valores passados como argumentos para a função, 3 e 4, respectivamente, conforme indicado na figura a seguir:



Expressão	Valor	Tipo	Contexto
Área	Vazio	Variant/Empty	Módulo1.CalcRetangulo
Diagonal	0	Double	Módulo1.CalcRetangulo
LadoMaior	4	Double	Módulo1.CalcRetangulo
LadoMenor	3	Double	Módulo1.CalcRetangulo
Perimetro	Vazio	Variant/Empty	Módulo1.CalcRetangulo

5. Até aqui já é possível ver a utilidade da Inspeção de variáveis. Você pode acompanhar, a cada passo da execução, o valor de uma ou mais variáveis, para tentar detectar, principalmente, os erros de lógica.
6. Pressione F8 novamente. O comando de cálculo da Área será selecionado. Pressione F8 e este comando será executado, o valor da variável Área será calculado e já passa a ser exibido na janela Inspeção de Variáveis, conforme indicado na figura a seguir. Observe que a variável Área apresenta o valor 12 (resultado da multiplicação 4 x 3, ou seja, da multiplicação dos lados passados como parâmetros, para cálculo da área).



Expressão	Valor	Tipo	Contexto
Área	12	Variant/Double	Módulo1.CalcRetangulo
Diagonal	0	Double	Módulo1.CalcRetangulo
LadoMaior	4	Double	Módulo1.CalcRetangulo
LadoMenor	3	Double	Módulo1.CalcRetangulo
Perimetro	Vazio	Variant/Empty	Módulo1.CalcRetangulo

7. Pressione F8. A mensagem que exibe o valor da variável Área será exibida, conforme indicado na figura a seguir. Clique em OK para fechar a mensagem.



8. Pressione F8, o comando para cálculo do perímetro será executado, o valor da variável Perimetro será calculado e já passa a ser exibido na janela Inspeção de Variáveis, conforme indicado na figura a seguir. Observe que a variável Perimetro apresenta o valor 14:

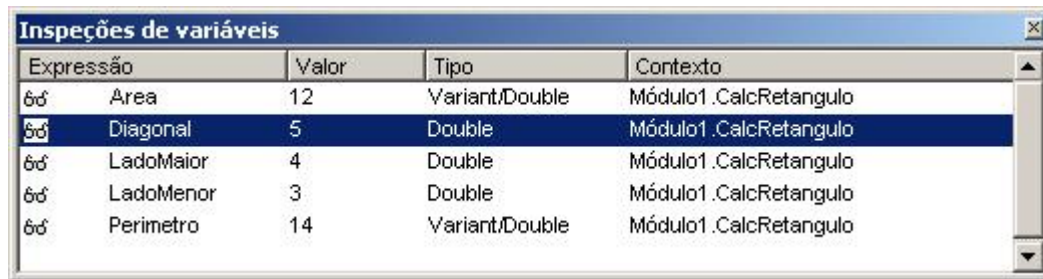


Expressão	Valor	Tipo	Contexto
Área	12	Variant/Double	Módulo1.CalcRetangulo
Diagonal	0	Double	Módulo1.CalcRetangulo
LadoMaior	4	Double	Módulo1.CalcRetangulo
LadoMenor	3	Double	Módulo1.CalcRetangulo
Perimetro	14	Variant/Double	Módulo1.CalcRetangulo

9. Pressione F8. A mensagem que exibe o valor da variável Perimetro será exibida, conforme indicado na figura a seguir. Clique em OK para fechar a mensagem.



10. Pressione F8, o comando para cálculo da diagonal será executado, o valor da variável Diagonal será calculado e já passa a ser exibido na janela Inspeção de Variáveis, conforme indicado na figura a seguir. Observe que a variável Diagonal apresenta o valor 5:

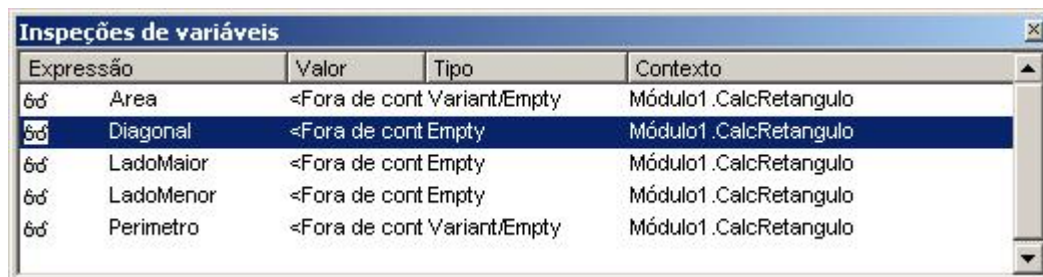


Expressão	Valor	Tipo	Contexto
Área	12	Variant/Double	Módulo1.CalcRetangulo
Diagonal	5	Double	Módulo1.CalcRetangulo
LadoMaior	4	Double	Módulo1.CalcRetangulo
LadoMenor	3	Double	Módulo1.CalcRetangulo
Perimetro	14	Variant/Double	Módulo1.CalcRetangulo

9. Pressione F8. A mensagem que exibe o valor da variável Diagonal será exibida, conforme indicado na figura a seguir. Clique em OK para fechar a mensagem.



10. Pressione F8. A execução é encerrada. Observe na janela Inspeção de variáveis que todas as variáveis estão sem valores definidos (sem contexto), pois uma vez que não existe código em execução, as variáveis simplesmente não tem valor associado a elas, conforme indicado na figura a seguir:



Expressão	Valor	Tipo	Contexto
Área	<Fora de cont Variant/Empty		Módulo1.CalcRetangulo
Diagonal	<Fora de cont Empty		Módulo1.CalcRetangulo
LadoMaior	<Fora de cont Empty		Módulo1.CalcRetangulo
LadoMenor	<Fora de cont Empty		Módulo1.CalcRetangulo
Perimetro	<Fora de cont Variant/Empty		Módulo1.CalcRetangulo

Muito bem, com este exemplo, você pode ter uma idéia de como funciona o recurso de Inspeção de variáveis, recurso este extremamente útil, principalmente na detecção de erros de lógica.

Lição 26: Conclusão do Módulo 2

Neste módulo tratamos de uma série de assuntos de fundamental importância para a programação VBA no Excel. Iniciamos o módulo falando sobre o Modelo de Objetos do Excel. Este é um conceito de fundamental importância e entendê-lo faz toda a diferença para poder programar com eficiência usando VBA no Excel. Em seguida detalhamos uma série de tópicos sobre a criação de funções/procedimentos, declaração de argumentos e passagem de parâmetros. Para encerrar o módulo abordamos sobre o tratamento de erros e os recursos de depuração do Editor do VBA. No próximo capítulo você aprenderá sobre o objeto mais utilizado na programação VBA no Excel: **“O Objeto Range”**.

Módulo 2 – O Modelo de Objetos do Excel

- Lição 01:** O que é um Modelo de Objetos?
- Lição 02:** Descrição dos Principais Objetos do Modelo de Objetos do Excel
- Lição 03:** O Microsoft Object-Browser: Navegando pela Hierarquia de Objetos
- Lição 04:** Objeto Application – O Pai de Todos - Introdução
- Lição 05:** Objeto Application – Como Declarar e Utilizar
- Lição 06:** Objeto Application – Propriedades que Retornam Objetos Filho
- Lição 07:** Objeto Application – Exibindo/Ocultando itens do Excel
- Lição 08:** Objeto Application – Habilitando/Desabilitando Recursos do Excel
- Lição 09:** Objeto Application – Associando Macros à Teclas Especiais
- Lição 10:** Objeto Application – Operações com Arquivos
- Lição 11:** Objeto Application – Recálculo da Planilha
- Lição 12:** Conceitos Avançados na Criação de Funções e Procedimentos
- Lição 13:** Conceitos Avançados na Criação de Funções e Procedimentos
- Lição 14:** Conceitos Avançados na Criação de Funções e Procedimentos
- Lição 15:** Conceitos Avançados na Criação de Funções e Procedimentos
- Lição 16:** Conceitos Avançados na Criação de Funções e Procedimentos
- Lição 17:** Conceitos Avançados na Criação de Funções e Procedimentos
- Lição 18:** A função MsgBox em Detalhes
- Lição 19:** A função InputBox em Detalhes
- Lição 20:** O Tratamento de Erros no VBA – Parte 1
- Lição 21:** O Tratamento de Erros no VBA – Parte 2
- Lição 22:** O Tratamento de Erros no VBA – Parte 3
- Lição 23:** O Tratamento de Erros no VBA – Parte 4
- Lição 24:** O Tratamento de Erros no VBA – Parte 5
- Lição 25:** O Tratamento de Erros no VBA – Parte 6
- Lição 26:** Conclusão do Módulo 2

No próximo módulo faremos um estudo detalhado do objeto Range. Este objeto representa uma ou mais faixas de células em uma planilha do Excel. Já dá para ver que, sem nenhuma dúvida, o objeto Range é o objeto mais utilizado na programação VBA no Excel. Também apresentarei diversos exemplos práticos de utilização do objeto Range.

Bibliografia recomendada:

Confira as dicas de livros de Excel no seguinte endereço:

<http://www.juliobattisti.com.br/indicados/excel.asp>

Módulo 3 – O Objeto Range e Exemplos Práticos

Nas lições deste módulo você aprenderá a trabalhar com o objeto que realmente faz o trabalho no Excel: **Objeto Range**. Falando de uma maneira bem simples, o objeto Range representa uma ou mais faixas de células em uma planilha do Excel. Como a grande maioria do trabalho (para não dizer a totalidade) com o Excel está relacionado a valores em uma faixa de células, fica clara a importância do objeto Range.

Este objeto pode ser utilizado para fazer referência ao intervalo atualmente selecionado, a um intervalo específico de células, tal como: A1:C50 ou B2:G50 ou a vários intervalos não contínuos, como por exemplo: A1:C50 mais H4:X30, mais AB45 e assim por diante. Em resumo, um objeto Range faz referência a um conjunto de células da planilha, conjunto este que é definido na declaração do objeto Range.

Além do estudo teórico, da declaração do objeto Range e do estudo de suas propriedades e métodos, apresentarei diversos exemplos de códigos de funções práticas, as quais utilizam o objeto Range. Sem nenhuma dúvida, você encontrará um grande número de exemplos que poderão ser facilmente adaptados para o seu próprio uso e, em alguns casos, nem sequer será necessário fazer adaptações.

Apenas para ilustrar o uso do objeto Range, vou citar duas dúvidas que recebo, frequentemente, via email e para as quais a solução passa pelo uso do objeto Range, dos eventos do Excel e do VBA, para a criação de funções personalizadas:

- ▶ **Dúvida 01: Como fazer cálculo que envolvem valores de horas no Excel?** Não existe uma ou mais funções prontas, no Excel, que implementam os cálculos necessários com valores de horas. Para tal temos que criar nossas próprias funções personalizadas usando o VBA.
- ▶ **Dúvida 02: Como faço para que o Excel formate, automaticamente, valores que estão dentro de uma determinada faixa. Por exemplo, formatar em negrito e vermelho, todos os valores da faixa de B1 a B100, que estiverem entre 50 e 100?** Para solucionar esta questão, você poderia usar o recurso de Formatação Condicional, descrito no Curso de Excel Básico. Porém com o uso do VBA, você pode automatizar este processo, além de ter um controle bem maior sobre as formatações a serem aplicadas e sobre o número de condições que podem ser levadas em consideração.

Você perceberá que com o uso do objeto Range, muitas tarefas que antes pareciam de difícil resolução, ou até mesmo pareciam impossíveis de resolver com o Excel, parecerão mais simples e perfeitamente possíveis de serem solucionadas.

Eu considero os conhecimentos dos módulos 1 e 2, a base fundamental para a programação VBA. Não tem como programar em VBA, sem os conhecimentos destes módulos básicos. Mas é a partir deste módulo, que você começará a ver a aplicação prática do VBA, para solução de problemas reais, que tantas vezes parecem insolúveis (e realmente o são), só com o uso das funções e comandos do Excel. Em resumo, a partir deste módulo, o amigo leitor irá constatar, na prática, todas as opções que se abrem com o uso da programação VBA no Excel. Então, sejam bem vindos ao estudo do objeto Range.

Lição 01: Apresentação do Objeto Range

Vamos fazer uma rápida recapitulação de como é a hierarquia de objetos do Modelo de Objetos do Excel?

- ▶ **Objeto Application:** O objeto de nível mais alto é o objeto Application, o qual representa a própria aplicação do Excel, ou seja, Application = Excel. Por exemplo, quando você abre o Excel, é criada uma instância do objeto Application, na memória do computador.
- ▶ **Objeto Workbook:** O objeto Workbook representa uma pasta de trabalho do Excel, ou seja, um arquivo .XLS. Por exemplo, você pode abrir o Excel (objeto Application) e abrir um ou mais arquivos .XLS (objetos Workbook). Por isso que o objeto Application, tem uma coleção chamada Workbooks, ou seja, uma coleção que contém uma referência a todas as pastas de trabalho abertas. Em resumo: Pasta de Trabalho = Arquivo .XLS = objeto Workbook.
- ▶ **Objeto Worksheet:** Dentro de uma mesma pasta de trabalho (arquivo .XLS), pode haver várias planilhas (objetos Worksheet). Por padrão, ao criar uma nova pasta de trabalho, são criadas, dentro da nova pasta de trabalho, três planilhas: Plan1, Plan2 e Plan3. O objeto Worksheet é utilizado para fazer referência a uma planilha de uma pasta de trabalho. Por isso que o objeto Workbook (pasta de trabalho), contém uma coleção chamada Worksheets (planilhas), através da qual é possível fazer referência a todas as planilhas da pasta de trabalho atual.
- ▶ **Objeto Range:** Em cada planilha de uma pasta de trabalho, está disponível um grande número de células. Na verdade, no Excel 2000, estão disponíveis: 16.777.216 células (resultado da multiplicação do número de linhas (65.536) pelo número de colunas (256). Para representar um intervalo da planilha, por exemplo de A1 até C100, é utilizado o objeto Range. Em resumo Range = uma faixa de células de uma planilha.

Após termos situado exatamente a posição do objeto Range, na hierarquia de objetos do Excel, podemos iniciar o estudo deste objeto. Nesta lição mostrarei que existem inúmeras maneiras diferentes de criar um objeto Range, usando o VBA.

Declarando um objeto Range:

Existem diferentes opções que retornam um objeto Range. Por exemplo, a propriedade Range, que está disponível nos objetos Application e Worksheet, é uma das maneiras de retornar um objeto Range.

Outra forma de retornar um objeto Range é utilizar a propriedade Cells do Objeto Application. A propriedade Cells retorna um objeto Range representando todas as células da planilha ativa. Se o documento ativo não for uma planilha (por exemplo, se for uma folha de gráfico), essa propriedade falhará. Somente leitura.

Você também pode utilizar a propriedade Cells do objeto Range. Neste caso, a propriedade Cells retorna um objeto Range representando as células do intervalo especificado (em outras palavras, não faz coisa alguma). Somente leitura.

Também está disponível uma propriedade Cells no objeto Worksheet. Neste caso, a propriedade Cells retorna um objeto Range representando todas as células da planilha (não apenas as células atualmente em uso). Somente leitura.

Em resumo, existem diversas maneiras para criar um objeto Range no código VBA. Muito melhor do que as explicações teóricas é vermos diversos exemplos práticos. Nesta lição apresentarei alguns exemplos simples de declaração de um objeto Range. Na próxima lição apresentarei mais exemplos de criação de um objeto Range.

Propriedades Cells, Rows e Columns do objeto Range:

Uma das maneiras mais simples de criar um objeto do tipo Range é, evidentemente, usando o próprio objeto Range e suas propriedades:

- ▶ **Cells:** Esta propriedade retorna uma coleção de células, propriedade esta definida pelos parâmetros passados quando da chamada da propriedade Cells.
- ▶ **Rows:** Esta propriedade retorna uma coleção de linhas do objeto Range. A coleção de linhas também é um objeto do tipo Range.
- ▶ **Columns:** Esta propriedade retorna uma coleção de colunas do objeto Range. A coleção de colunas também é um objeto do tipo Range.

Para entender o funcionamento do objeto Range e das propriedades Cells, Rows e Columns, vamos a alguns exemplos práticos.

Exemplo 01: Considere o código indicado na listagem a seguir:

```
Public Sub CriaRange()  
  
    ' Declaração de um objeto do tipo Range  
    Dim MinhaFaixa As Range  
  
    ' Início o objeto Range, associando a variável a faixa A1:B5  
    Set MinhaFaixa = Range("A1:B5")  
  
    ' Exibo o número de células do objeto MinhaFaixa  
    MsgBox "A faixa do objeto Range contém: " & MinhaFaixa.Count & " Células"  
  
End Sub
```

Inicialmente declaro uma variável MinhaFaixa como sendo do tipo Range. Ou seja, a variável MinhaFaixa é um objeto do tipo Range. Em seguida defino o valor da variável MinhaFaixa, fazendo uma chamada a Range e passando a faixa a ser associada com o objeto, como parâmetro, conforme indicado no trecho de código a seguir:

```
Set MinhaFaixa = Range("A1:B5")
```

Em seguida uso a função MsgBox para exibir o número de células da faixa contida no objeto MinhaFaixa. Para isso uso a propriedade Count, do objeto Range.

Ao executar o código indicado na listagem anterior, você obterá o resultado indicado na Figura a seguir:



Observe que para a criação do objeto Range, bastou uma chamada a Range, passando como parâmetro a faixa associada ao objeto Range.

Exemplo 02: A seguir apresento uma listagem modificada, em relação ao exemplo anterior, onde além do número total de células, uso as propriedades Rows, para exibir o número de linhas da faixa e a propriedade Columns, para exibir o número de colunas da faixa de células associada ao objeto MinhaFaixa:

```
Public Sub CriaRange()  
  
    ' Declaração de um objeto do tipo Range  
    Dim MinhaFaixa As Range  
  
    ' Inicio o objeto Range, associando a variável a faixa A1:B5  
    Set MinhaFaixa = Range("A1:B5")  
  
    ' Exibo o número de células do objeto MinhaFaixa  
    MsgBox "A faixa A1:B5 contém: " & MinhaFaixa.Count & " Células"  
  
    ' Exibo o número de linhas do objeto MinhaFaixa  
    MsgBox "A faixa A1:B5 contém: " & MinhaFaixa.Rows.Count & " Linhas"  
  
    ' Exibo o número de colunas do objeto MinhaFaixa  
    MsgBox "A faixa A1:B5 contém: " & MinhaFaixa.Columns.Count & " Colunas"  
  
End Sub
```

Ao executar este procedimento, você obtém a sequência de telas indicadas a seguir:

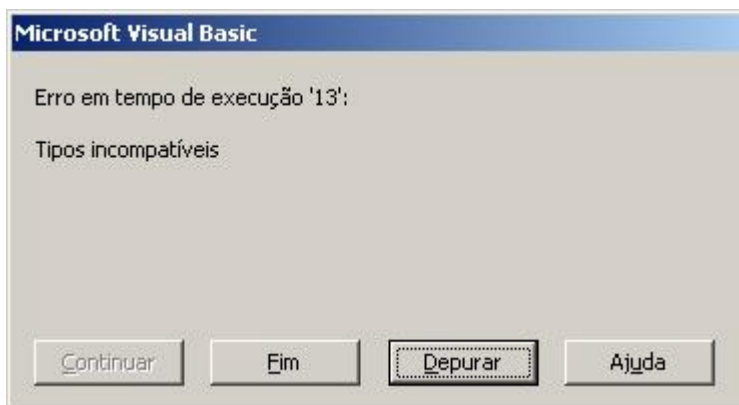




Um erro comum seria utilizar o comando a seguir, para tentar obter o número de linhas do objeto Range:

```
MsgBox "A faixa A1:B5 contém: " & MinhaFaixa.Rows & " Linhas"
```

Rows retorna a coleção de linhas do objeto Range e não o número de linhas do objeto. Se você usasse este comando, obterias uma mensagem de erro, conforme indicado a seguir:



Para obter o número de linhas de um objeto Range, devemos utilizar a propriedade Count, da coleção Rows. Para obter o número de colunas de um objeto Range, devemos utilizar a propriedade Count, da coleção Columns.

Exemplo 03: No exemplo a seguir, mostro como utilizar uma estrutura For...Each, para percorrer todas as células de um objeto Range. A título de exemplo, defino o valor de cada uma das células com o valor numérico 50.

```
Public Sub PercorreFaixa()  
  
    ' Declaração de um objeto do tipo Range  
    Dim MinhaFaixa As Range  
  
    ' Início o objeto Range, associando a variável a faixa A1:B5  
    Set MinhaFaixa = Range("A1:B5")  
  
    ' Uso uma estrutura For...Each, para percorrer todas as células  
    ' da coleção Cells do objeto Range e definir o valor das células  
    ' em 50.  
  
    For Each Celula In MinhaFaixa.Cells  
        Celula.Value = 50  
    Next  
  
End Sub
```

Para definir o valor de cada uma das células, da faixa definida no objeto MinhaFaixa, utilizei a propriedade Value da célula. Após a execução deste procedimento, você obterá os resultados indicados na Figura a seguir:



Nesta lição você já pode ter uma idéia de como declarar e utilizar algumas propriedades básicas de um objeto Range. Na próxima lição mostrarei outras maneiras de declarar e utilizar o objeto Range.

Lição 02: Objeto Range – Outras Maneiras de Criar um Objeto Range

Nesta lição falarei um pouco mais sobre como funciona a propriedade Cells, do objeto Range e como acessar células específicas, dentro de uma faixa, usando os índices da propriedade Cells.

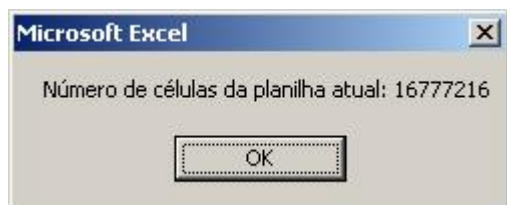
A propriedade Cells do objeto Range:

A propriedade Cells está disponível nos objetos Application, Range e Worksheet, conforme descrito a seguir:

- ▶ **Objeto Application:** A propriedade Cells do objeto Application retorna um objeto Range representando todas as células da planilha ativa. Se o documento ativo não for uma planilha, essa propriedade falhará e irá retornar um erro. É do tipo somente leitura. Considere o trecho de código a seguir:

```
MsgBox "Número de células da planilha atual: " & Excel.Application.Cells.Count
```

Esta linha de código retorna o resultado indicado na figura a seguir, ou seja, o número total de células da planilha atual:



- ▶ **Objeto Range:** A propriedade Cells do objeto Range retorna um objeto Range representando as células do intervalo especificado. É do tipo somente leitura. Considere o trecho de código a seguir:

```
MsgBox "Número de células na faixa A1:C5: " & Range("A1:C5").Cells.Count
```

Esta linha de código retorna o resultado indicado na figura a seguir, ou seja, o número total de células no intervalo definido na chamada do objeto Range, A1:C5, no nosso exemplo:



- ▶ **Objeto Worksheet:** A propriedade Cells do objeto Worksheet retorna um objeto Range representando todas as células da planilha (não apenas as células atualmente em uso). É do tipo somente leitura.

Usando os índices da propriedade Cells:

Você pode usar a propriedade Cells, juntamente com índices passados como parâmetros, para acessar uma célula específica, dentro de uma faixa de células. O uso de índices é útil quando você tem uma faixa regular de células, com um número de linhas e colunas regular, como por exemplo 5 linhas e 10 colunas ou 10 linhas e 15 colunas.

Vamos considerar um objeto Range, associado a Faixa de A1 a C5, indicada na Figura a seguir:



Neste exemplo temos a faixa A1:C5. esta faixa tem cinco linhas (da linha 1 até a linha 5) e três colunas (A, B, C). Podemos imaginar uma faixa de células como se fosse uma matriz. Por exemplo, o valor 125 está na célula da linha 1 com a coluna 1 (isso dentro da faixa). O valor 130 está na célula da linha 2 da coluna 1 (isso dentro da faixa). O valor 95 está na célula da linha 3 da coluna 2 e assim por diante. Nesta matriz, a primeira célula é a célula da linha 1, coluna 1 (L1C1), a segunda célula da primeira coluna, é a célula da linha 2, coluna 1 (L2C1) e assim por diante. Na tabela a seguir temos uma descrição da posição de cada célula, dentro da nomenclatura linha coluna:

L1 C1	L1 C2	L1 C3
L2 C1	L2 C2	L1 C3
L3 C1	L3 C2	L1 C3
L4 C1	L4 C2	L1 C3
L5 C1	L5 C2	L1 C3

Esses índices podem ser utilizados pela propriedade Cells do objeto Range, para acessar uma célula específica, dentro da faixa de células definida pelo objeto Range. Por exemplo, considere o código a seguir:

```
Range.Cells(1,1)
```

Este comando faz referência a célula da primeira linha e primeira coluna da faixa de células (L1C1). Considere outro exemplo

```
Range.Cells(1,3)
```

Este comando faz referência a célula da primeira linha e terceira coluna, da faixa de células (L1C3). No exemplo da figura anterior, seria a célula com o valor 123, ou seja, o valor da célula da primeira linha e terceira coluna da faixa de células definida no objeto Range. A seguir apresento um exemplo prático de uso do objeto Range, com base na planilha indicada na figura do início da lição:

```
Public Sub UsaCells()  
  
    ' Declaração de uma variável do tipo Range  
    Dim MinhaFaixa As Range  
  
    ' Definição da variável MinhaFaixa, associando-a ao intervalo A1:C5  
  
    Set MinhaFaixa = Range("A1:C5")  
  
    ' Agora passo a usar a propriedade Cells, para exibir valores de células dentro da  
    ' faixa A1:C5 Observe a utilização de índices, onde o primeiro índice indica o número  
    ' da linha e o segundo o número da coluna, dentro das linhas e colunas da faixa definida em  
    ' MinhaFaixa  
  
    ' Exibo o valor da célula de segunda linha (2) e terceira coluna (C), ou seja,  
    ' o valor da célula C2 Antes de exibir o valor, uso a propriedade Bold, para colocar  
    ' o valor da respectiva célula em Negrito. Em seguida uso a propriedade ColorIndex,  
    ' para alterar a cor da fonte para vermelho. O índice 3 significa que será usada a terceira  
    ' cor da caixa de cores da fonte, que é justamente o vermelho.  
  
    MinhaFaixa.Cells(2, 3).Font.Bold = True  
    MinhaFaixa.Cells(2, 3).Font.ColorIndex = 3  
  
    MsgBox "Valor da célula da 2ª linha e da 3ª coluna: " & MinhaFaixa.Cells(2, 3)  
  
    ' Exibo o valor da célula de terceira linha (3) e primeira  
    ' coluna (A), ou seja, o valor da célula A1  
    ' Também aplico formatações diferenciadas, para destacar a célula.  
  
    MinhaFaixa.Cells(3, 1).Font.Bold = True  
    MinhaFaixa.Cells(3, 1).Font.ColorIndex = 7  
  
    MsgBox "Valor da célula da 3ª linha e da 1ª coluna: " & MinhaFaixa.Cells(3, 1)  
  
End Sub
```


Neste exemplo, basicamente, declarar uma variável do tipo Range, depois associe esta variável com a faixa de A1:C5:

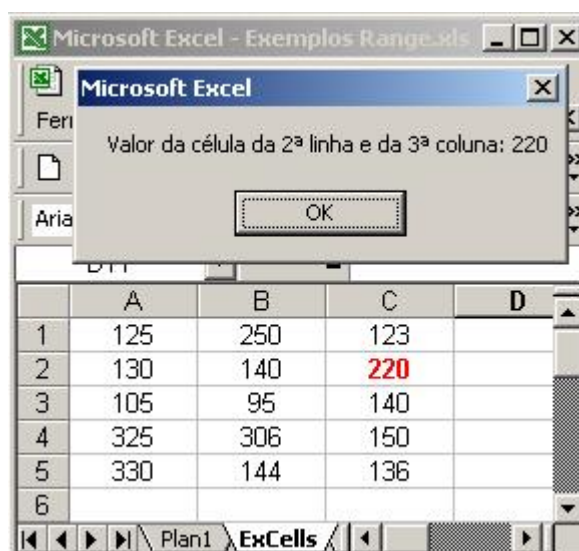
```
Dim MinhaFaixa As Range  
  
Set MinhaFaixa = Range("A1:C5")
```

Em seguida passo a usar a propriedade Cells, do objeto Range, para acessar células específicas, dentro da faixa A1:C5. Inicialmente acesso a célula da 2ª linha e terceira coluna. Uso a propriedade Font, do objeto Cells para definir negrito e a cor da fonte:

```
MinhaFaixa.Cells(2,3).Font.Bold = True  
MinhaFaixa.Cells(2,3).Font.ColorIndex = 3  
  
MsgBox "Valor da célula da 2ª linha e da 3ª coluna: " & MinhaFaixa.Cells(2, 3)
```

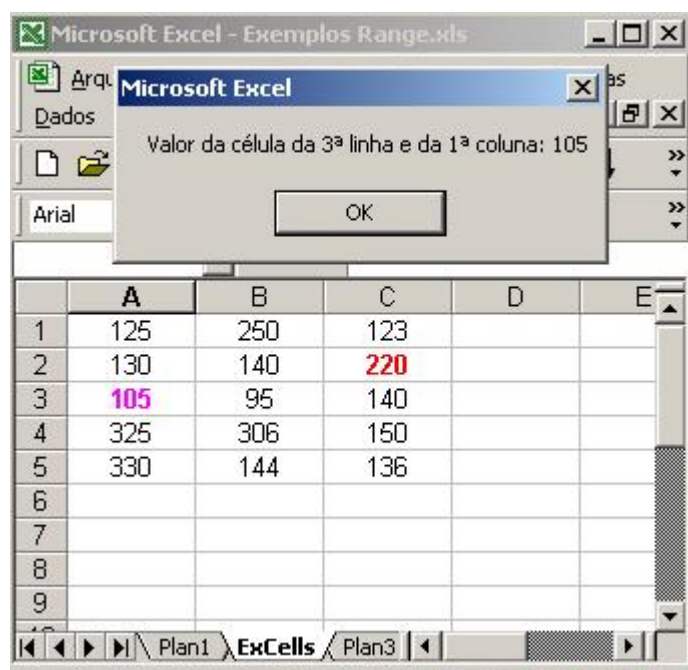
MinhaFaixa.Cells(2,3) é a referência a célula da segunda linha e terceira coluna, dentro da faixa de células do objeto Minha Faixa. Feita a referência, passo a usar a propriedade Font, do objeto Cell, para definir algumas características da fonte da referida célula.

A seguir apresento a sequência de telas que é exibida durante a execução desta rotina, juntamente com as alterações feitas na planilha:



Observe que Cells(2, 3) faz referência a célula da segunda linha e terceira coluna, dentro da faixa de células definida pelo objeto Range. Não é a segunda linha e terceira coluna da planilha como um todo, mas sim a segunda linha dentro da faixa definida pelo objeto Range e a terceira coluna, dentro da faixa definida pelo objeto Range. Observe também que a célula é formatada em negrito e cor vermelha, conforme indicado na Figura a seguir.

Clique em OK. Será acessada a célula da terceira linha e primeira coluna (valor 105), dentro da faixa de células definida pelo objeto Range. Esta célula será formatada em Negrito e cor-de-rosa, conforme indicado na Figura a seguir:



Clique em OK e pronto, a procedimento UsaCells será encerrado. Para fixar o entendimento da coleção Cells, a seguir listo como seria o acesso, a cada uma das células da faixa definida no nosso exemplo:

Uso de Cells	Linha	Coluna	Valor
MinhaFaixa.Cells(1, 1)	1	1	125
MinhaFaixa.Cells(1, 2)	1	2	250
MinhaFaixa.Cells(1, 3)	1	3	123
MinhaFaixa.Cells(2, 1)	2	1	130
MinhaFaixa.Cells(2, 2)	2	2	140
MinhaFaixa.Cells(2, 3)	2	3	220
MinhaFaixa.Cells(3, 1)	3	1	105
MinhaFaixa.Cells(3, 2)	3	2	95
MinhaFaixa.Cells(3, 3)	3	3	140
MinhaFaixa.Cells(4, 1)	4	1	325
MinhaFaixa.Cells(4, 2)	4	2	306
MinhaFaixa.Cells(4, 3)	4	3	150
MinhaFaixa.Cells(5, 1)	5	1	330
MinhaFaixa.Cells(5, 2)	5	2	144
MinhaFaixa.Cells(5, 3)	5	3	136

Lição 03: Objeto Range – Outras Maneiras de Criar um Objeto Range

Nesta lição continuarei a mostrar diferentes maneiras para retornar um objeto do tipo Range, no código VBA.

Ainda usando o objeto Range:

Ainda com o objeto Range, existem diferentes maneiras para inicialiar um objeto do tipo Range, desde uma faixa que se refere a uma única célula, passando por uma faixa contínua de células, até uma faixa formada pela união ou intersecção de diferentes faixas de células na planilha. Vamos considerar alguns exemplos práticos, para clarear estas idéias:

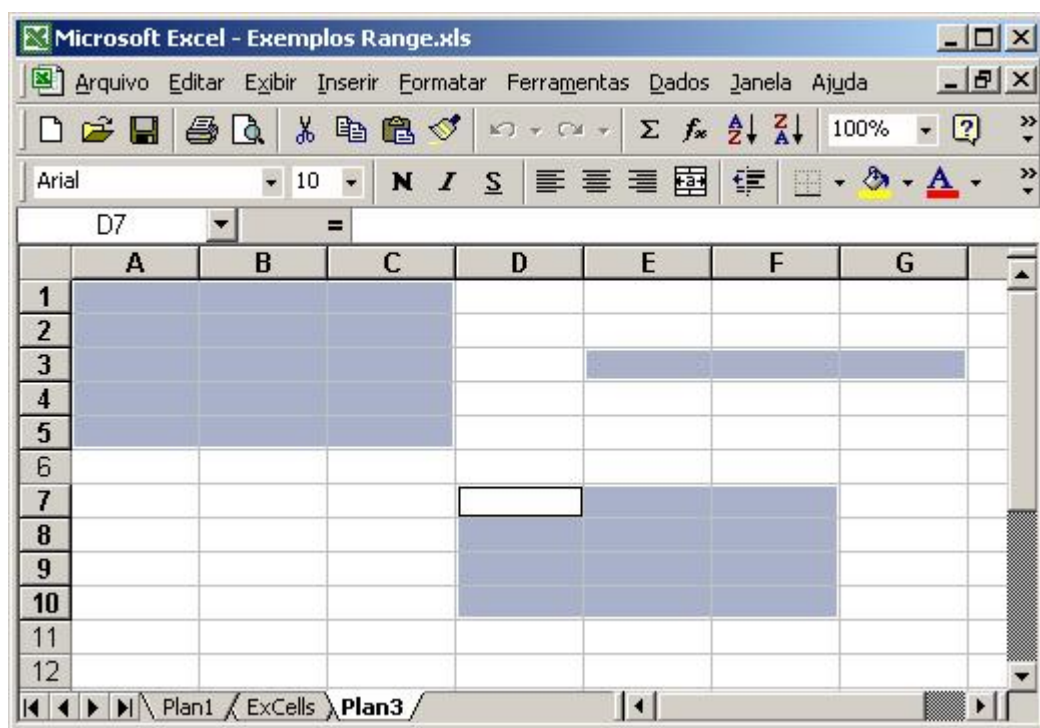
No exemplo a seguir, defino uma faixa que na verdade é uma única célula: C5:

```
Dim MinhaFaixa As Range  
Set MinhaFaixa = Range("C5")
```

No próximo exemplo, defino uma faixa contínua, informando o endereço inicial e o endereço final, separados por dois pontos (:). Este é o uso mais comum. No exemplo, defino a faixa C3:F30:

```
Dim MinhaFaixa As Range  
Set MinhaFaixa = Range("C3:F30")
```

Até agora definimos faixas contínuas. No Excel, é possível definir uma faixa formada por áreas não contínuas de uma planilha, como no exemplo da figura a seguir, onde selecionei as faixas A1:C5 mais E2:G2 e mais D7:F10:



Para criar um objeto Range, formado por faixas não contínuas, basta passar as faixas, como parâmetros para o objeto Range, separando as faixas por vírgula, como no exemplo a seguir:

```
Dim MinhaFaixa As Range
Set MinhaFaixa = Range("A1:C5,E2:G2,D7:F10")
MsgBox "NúmeroDeCélulas: " & MinhaFaixa.Cells.Count
```

A execução deste trecho de código produz o resultado indicado na Figura a seguir:



Vamos conferir se a contagem de células está correta? Para facilitar a conferência, considere a tabela a seguir:

Faixa	Número de Células
A1:C5	15
E2:G2	03
D7:F10	12
Total -> >	30

Pela tabela podemos ver que o Excel sabe contar corretamente, como não poderia deixar de ser.

A declaração usando várias faixas, separadas por vírgula é conhecida como uma declaração de união, ou seja, os vários intervalos são unidas para formar uma única faixa (embora formada por faixas menores, não contínuas).

Ao invés de uma união entre faixas, podemos usar uma intersecção entre faixas. Ao usar uma intersecção, informo dois ou mais intervalos de células, separados por espaço em branco. Farão parte do objeto Range, somente as células que pertencerem a todos os intervalos informados. Se não houver nenhuma célula comum a todos os intervalos, o objeto Range ficará vazio. Observe o exemplo a seguir:

```
Dim MinhaFaixa As Range
Set MinhaFaixa = Range("A1:C5 B4:D10")
MsgBox "NúmeroDeCélulas: " & MinhaFaixa.Cells.Count
```

A questão é, qual a faixa de células definida para o objeto Range? Simples, somente as células comuns as duas faixas passadas como parâmetros, que neste caso serão as células B4, B5, C4 e C5, ou seja, o intervalo B4:C5. Estas são as células que fazem parte tanto do primeiro intervalo – A1:C5, quanto do segundo intervalo: B4:D10. Ao ser executado este trecho de código, será retornada a mensagem indicada a seguir:



Você também pode definir um objeto Range, informando o endereço da célula do canto superior esquerdo e do canto inferior direito. O objeto Range será definido como sendo a faixa que vai da célula informada como canto superior esquerdo, até a célula informada como canto inferior direito. No exemplo a seguir, passo como parâmetros para o objeto range, os endereços das células C5 e F20. Ou seja, na prática, isto definindo a faixa C5:F20:

```
Dim MinhaFaixa As Range  
Set MinhaFaixa = Range("C5","F20")  
MsgBox "NúmeroDeCélulas: " & MinhaFaixa.Cells.Count
```

Este trecho de código, irá retornar o resultado indicado na figura a seguir, ou seja, 64 células. A faixa C5: F20 tem quatro colunas (C, D, E e F) e 16 linhas (5, 6, 7, 8, 9 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 e 20). Quatro colunas vezes dezesseis linhas = 64 células. Ou seja, mais uma vez comprovamos que o nosso amigo Excel sabe contar.



Muito bem, com isso encerramos o nosso estudo sobre as principais maneiras de criar um objeto do tipo Range. A partir da próxima lição faremos um estudo dos principais métodos e propriedades do objeto Range. Na parte final do módulo, apresentarei uma série de exemplos práticos, com o uso do objeto Range, para a solução de problemas práticos no uso do Excel.

Lição 04: Objeto Range – Column, Columns, Row e Rows – Parte 1

Uma faixa (Range) é formada por uma ou mais linhas e uma ou mais colunas. Com isso o objeto Range oferece propriedades e coleções para trabalhar diretamente com uma linha ou coluna do objeto Range. Nesta e na próxima lição faremos o estudo dos seguintes itens:

- ▶ Propriedade Column
- ▶ Coleção Columns
- ▶ Propriedade Row
- ▶ Coleção Rows

Propriedade Column:

Esta propriedade retorna o número da primeira coluna na primeira área do intervalo especificado. É do tipo Long e somente leitura.

A coluna A retorna 1, a coluna B retorna 2, e assim por diante.

Para retornar o número da última coluna do intervalo, use a expressão seguinte.

```
myRange.Columns(myRange.Columns.Count).Column
```

O `myRange.Columns.Count` informa o número de colunas na faixa `myRange`, número este que é passado como parâmetro para a coleção `Columns`, ou seja, é uma maneira de acessar a última coluna da faixa. Usando a propriedade `Column`, retorno o número da última coluna. Vamos a mais um exemplo de uso da propriedade `Column`:

Este exemplo define a largura de coluna como 4 pontos, a cada duas colunas da planilha `Plan1`:

```
For Each col In Worksheets("Plan1").Columns
    If col.Column Mod 2 = 0 Then
        col.ColumnWidth = 4
    End If
Next col
```

Neste casouso a coleção `Worksheets` para fazer referência a coleção de planilhas da pasta de trabalho atual. Passo como parâmetro para esta coleção, o nome da planilha a ser acessado, que no nosso exemplo é `Plan1`. Dentro do laço `For...Each`, uso um teste onde Divido o número da coluna por 2, usando o operador `Mod`. O operador `Mod` (veja lições do Módulo 1) retorna o resto de uma divisão. Se o resultado for zero, defino o tamanho da coluna como sendo quatro pontos. Se eu quisesse definir o tamanho de três em três colunas, faria a divisão por 3, de quatro em quatro colunas, faria a divisão por quatro e assim por diante.

Coleção Columns:

Esta coleção está disponível em diferentes objetos. Está disponível no objeto `Range`, no objeto `Application` e no objeto `Worksheet`, conforme descrito a seguir:

► **Objeto Application:** Retorna um objeto Range representando todas as colunas da planilha ativa. Se o documento ativo não for uma planilha (por exemplo, se for uma folha de gráfico), a propriedade Columns falhará. Somente leitura.

► **Objeto Range:** Retorna um objeto Range representando as colunas no intervalo especificado. Somente leitura.

► **Objeto Worksheet:** Retorna um objeto Range representando todas as colunas da planilha especificada. Somente leitura.

Nota: O uso dessa propriedade sem um qualificador de objeto equivale a usar `ActiveSheet.Columns`.

Quando aplicada a um objeto Range que for uma seleção de várias áreas, essa propriedade retornará colunas apenas da primeira área do intervalo. Por exemplo, se o objeto Range tiver duas áreas — A1:B2 e C3:D4 — `Selection.Columns.Count` retornará 2 (duas colunas da área A1:B2), e não 4. Para usar essa propriedade em um intervalo que possa conter uma seleção de várias áreas, teste `Areas.Count` para determinar se o intervalo contém mais do que uma área. Caso contenha, faça um loop sobre cada área do intervalo, conforme será demonstrado em um dos próximos exemplos e detalhado mais adiante, quando falarmos sobre a propriedade `Areas` do objeto Range.

Considere os exemplos a seguir:

Este exemplo formata a fonte da coluna um (coluna A), na planilha Plan1 com negrito.

```
Worksheets("Plan1").Columns(1).Font.Bold = True
```

O próximo exemplo define como 0 (zero) o valor de todas as células na coluna um no intervalo chamado "MinhaFaixa".

```
Range("MinhaFaixa").Columns(1).Value = 0
```

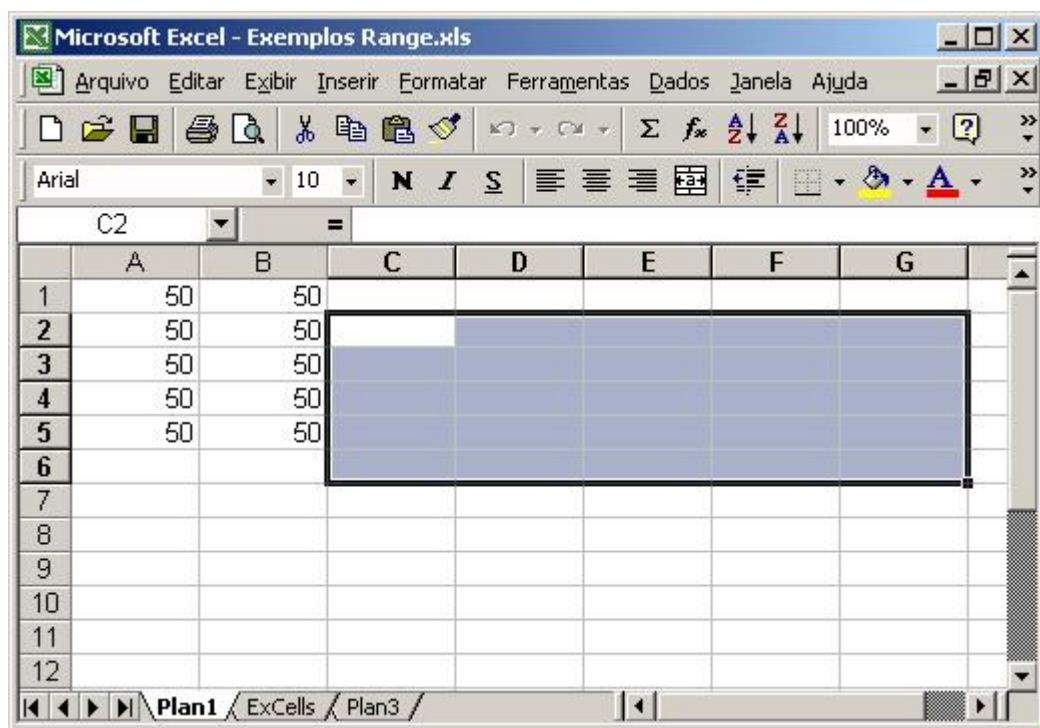
O exemplo que segue exibe o número de colunas na seleção atual na planilha Plan1. Se mais de uma área for selecionada, o exemplo fará um loop por todas elas.

```
Public Sub TestaFaixas()  
  
    ' Ativo a planilha Plan1  
    Worksheets("Plan1").Activate  
  
    ' Uso a propriedade Areas para detectar o número de áreas na seleção atual  
    areaCount = Selection.Areas.Count  
  
    If areaCount <= 1 Then  
        MsgBox "A seleção contém " & Selection.Columns.Count & " colunas."  
    Else
```



```
' Faço um loop por todas as áreas da seleção e vou exibindo o número de colunas  
' de cada área.  
For i = 1 To areaCount  
    MsgBox "Área " & i & " da seleção atual, contém: " & Selection.Areas(i).Columns.Count & " Colunas."  
Next i  
End If  
  
End Sub
```

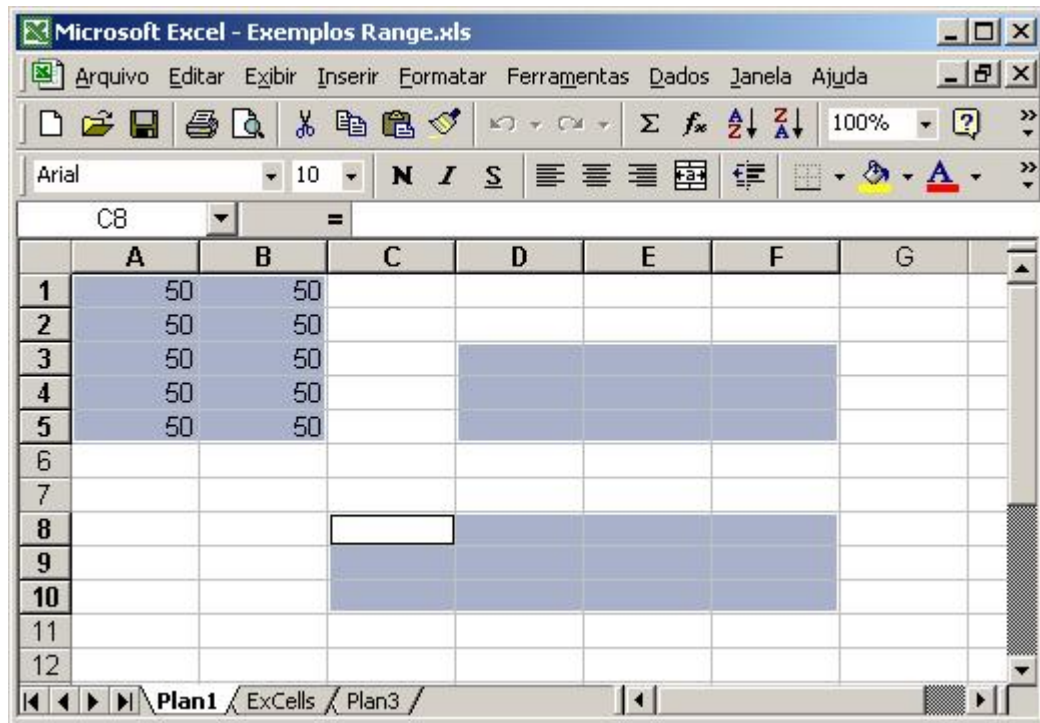
Vamos ver como funciona a execução do procedimento TestaFaixas em diferentes situações. Inicialmente vamos testar este procedimento, quando uma única faixa da planilha Plan1 estiver selecionada, conforme exemplo da figura a seguir, onde está selecionada uma faixa com cinco colunas e 4 linhas: C2:G5:



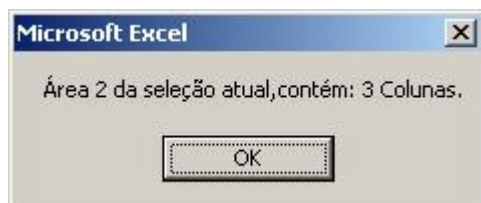
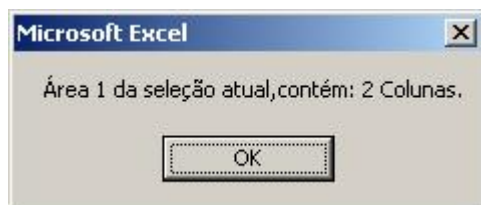
Ao executar o procedimento TestaFaixas, com a seleção da figura anterior, você obtém o resultado indicado a seguir:



Agora vamos acompanhar como seria a execução do procedimento TestaFaixas, quando a seleção atual for formada por duas ou mais faixas de células na planilha, conforme exemplo da figura a seguir, onde a seleção é formada por três faixas de células:



Ao executar o procedimento TestaFaixas, você obterá a sequência de mensagens, indicadas a seguir:



Ou seja, a propriedade Column, retornando o número de colunas de cada área da seleção atual, sendo que para acessar as diferentes áreas, foi utilizada a coleção Areas, do objeto Range. Na próxima lição estudaremos a propriedade Row e a coleção Rows.

Lição 05: Objeto Range – Column, Columns, Row e Rows – Parte 2

Nesta lição apresentaremos a descrição e exemplos de uso da propriedade Row e da coleção Rows, do objeto Range.

A propriedade Row:

Esta propriedade retorna o número da primeira linha, da primeira área do intervalo. É do tipo Long somente leitura. Considere o exemplo a seguir:

```
Dim MinhaFaixa As Range
Set MinhaFaixa = Range("C5","F20")
MsgBox "Número da primeira linha: " & MinhaFaixa.Row
```

Este trecho de código produz o resultado indicado na figura a seguir:



Observe que o número retornado é o número da linha na planilha. No nosso exemplo, a primeira linha da faixa C5:C20 é a linha 5, formada pelas células: C5, D5, E5 e F5. Ou seja, são células da quinta linha da planilha, por isso a propriedade Row retorna o valor 5.

A coleção Rows:

Esta coleção está disponível em diferentes objetos. Está disponível no objeto Range, no objeto Application e no objeto Worksheet, conforme descrito a seguir:

- ▶ **Objeto Application:** A coleção Rows do objeto Application retorna um objeto Range representando todas as linhas da planilha ativa. Se o documento ativo não for uma planilha (como por exemplo uma folha de gráfico), a propriedade Rows falhará. É Somente leitura.
- ▶ **Objeto Range:** A coleção Rows do objeto Range retorna um objeto Range representando as linhas no intervalo especificado. É somente leitura.
- ▶ **Objeto Worksheet:** A coleção Rows do objeto Worksheet retorna um objeto Range representando todas as linhas da planilha especificada. É somente leitura.

Quando aplicado a um objeto Range que é uma seleção múltipla (várias áreas selecionadas, conforme exemplos da lição anterior), essa propriedade retorna linhas apenas da primeira área do intervalo. Por exemplo, se o objeto Range tiver duas áreas — A1:B2 e C3:D4 — Selection.Rows.Count retornará 2, que representa o número de linhas da primeira área (A1:B2) e não 4, que seria o número total de linhas de todas as áreas selecionadas.

Para usar essa propriedade em um intervalo que possa conter uma seleção múltipla, teste `Areas.Count` para determinar se o intervalo é uma seleção múltipla. Se for, faça um loop sobre cada área do intervalo, como mostrado no terceiro exemplo, semelhante ao que fizemos na lição anterior, no exemplo da coleção `Columns`.

Vamos a alguns exemplos de uso da coleção Rows:

Este exemplo exclui a linha três da planilha chamada `Plan1`:

```
Worksheets("Plan1").Rows(3).Delete
```

O exemplo a seguir, seleciona a décima linha da planilha chamada `Plan1`:

```
Worksheets("Plan1").Rows(10).Select
```

O exemplo a seguir exibe o número de linhas na seleção atual, na planilha `Plan1`. Se mais de uma área for selecionada, o exemplo fará um loop por todas elas.

```
Worksheets("Plan1").Activate  
areaCount = Selection.Areas.Count  
  
If areaCount <= 1 Then  
    MsgBox "A seleção contém: " & Selection.Rows.Count & " linhas."  
Else  
    i = 1  
    For Each a In Selection.Areas  
        MsgBox "Area " & i & " da seleção, contém " & a.Rows.Count & " linhas."  
        i = i + 1  
    Next a  
End If
```

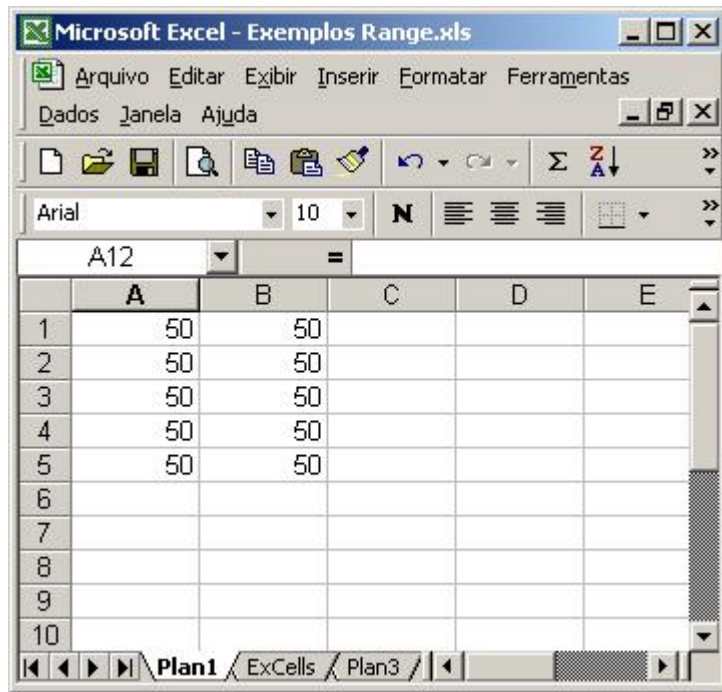
Este exemplo é idêntico ao exemplo da lição anterior, só que ao invés de contarmos o número de colunas de cada área da seleção atual (usando a coleção `Columns`), contamos o número de linhas de cada área da seleção atual. Para isso utilizamos a coleção `Rows`.

Mais algumas observações sobre as coleções Rows e Columns:

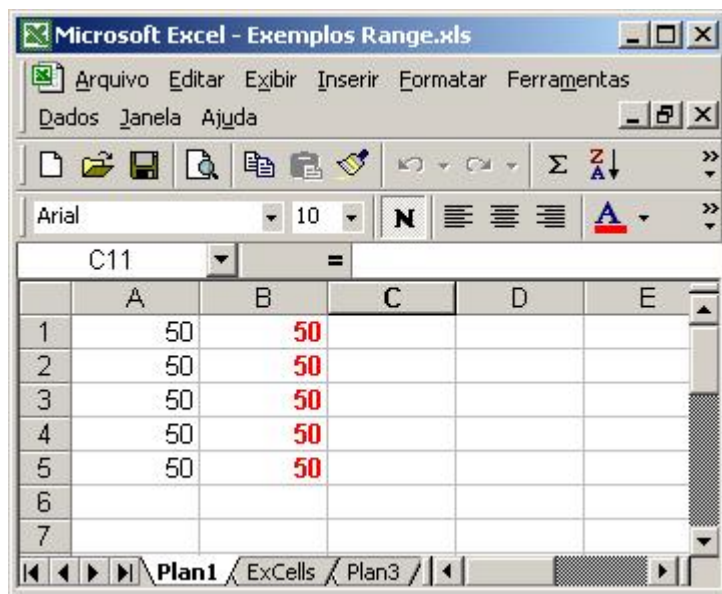
Observação 01: Além do número da coluna, você pode usar a letra indicativa da coluna, entre aspas, como argumento para a chamada da coleção `Columns`. Considere o exemplo a seguir. Esta linha de código irá selecionar a coluna B da planilha `Plan1` e formatar seus valores com Negrito e Cor de Fonte vermelha.

```
Worksheets("Plan1").Columns("B").Select  
  
Selection.Font.Bold = True  
Selection.ColorIndex = 3
```

Na figura a seguir, temos uma planilha, antes da execução deste código:



Após executarmos o código, observe que a coluna B foi selecionada e teve a formatação alterada para Negrito e cor de fonte vermelha:



Observação 02: Você também pode usar uma faixa de letras ou números, para fazer referência a várias colunas ou várias linhas, respectivamente. A primeira linha, a seguir, seleciona as colunas de A até D e a segunda linha de código, seleciona as linhas de 5 a 10:

```
Worksheets("Plan1").Columns("A:D").Select  
Worksheets("Plan1").Rows("5:10").Select
```

Muito bem, com isso encerramos o estudo das propriedades Column e Row e das coleções Columns e Rows.

Lição 06: Objeto Range – Principais Métodos e Propriedades – Parte 1

A partir desta lição estudaremos uma série de propriedades, métodos e coleções do objeto Range. Na parte final do módulo, apresentarei uma série de exemplos práticos, baseados no objeto Range e no uso das propriedades, métodos e coleções que estudaremos nestas e nas próximas lições.

Propriedade Areas:

Conforme já descrito e exemplificado nas lições anteriores, um objeto Range pode ser formado de uma ou mais faixa de células, contínuas ou não. As diversas áreas que formam uma faixa, podem ser acessadas através da propriedade Areas do objeto Range.

Esta propriedade retorna uma coleção Areas, a qual representa todos os intervalos em uma seleção de várias áreas. É do tipo somente leitura.

Para uma única seleção, a propriedade Areas retorna uma coleção que contém um objeto — o próprio objeto Range original. Para uma seleção de várias áreas, a propriedade Areas retorna uma coleção que contém um objeto para cada área selecionada.

A coleção retornada pela propriedade Areas é uma coleção das áreas, ou blocos contíguos de células, dentro de uma seleção ou objeto Range. Não há objeto Area separado; os membros individuais da coleção Areas são objetos Range. A coleção Areas contém um objeto Range para cada intervalo de células contíguo individual dentro da seleção. Se a seleção contiver somente uma área, a coleção Areas conterá um único objeto Range correspondente a essa seleção.

Exemplos de uso da propriedade Areas:

Exemplo 01: O exemplo seguinte limpa a seleção atual se ela contiver mais do que uma área.

```
If Selection.Areas.Count <> 1 Then  
    Selection.ClearUse Areas(índice)
```

,onde índice é um número que indica qual dos itens da coleção de áreas deve ser retornado, para retornar um único objeto Range da coleção. Os números de índice correspondem à ordem na qual as áreas foram selecionadas.

Exemplo 02: O exemplo seguinte limpa a primeira área na seleção atual se a seleção contiver mais do que uma área.

```
If Selection.Areas.Count <> 1 Then  
    Selection.Areas(1).Clear  
End If
```

Dica: Algumas operações não podem ser efetuadas em mais de uma área ao mesmo tempo em uma seleção; você precisa efetuar um loop nas áreas individuais na seleção e efetuar as operações em cada área separadamente.

Exemplo 03: O exemplo seguinte efetua a operação de colocar em negrito todas as células no intervalo selecionado se a seleção contiver somente uma área; se a seleção contiver várias áreas, a formatação em negrito será aplicada em cada área individual da seleção.

```
Set rangeToUse = Selection

If rangeToUse.Areas.Count = 1 Then
    rangeToUse.Font.Bold = True
Else
    For Each singleArea In rangeToUse.Areas
        singleArea.Font.Bold = True
    Next
End If
```

O Método Activate:

Diversos objetos possuem o método Activate. Como o próprio nome sugere, este método é utilizado para ativar, ou melhor, colocar o foco em um determinado objeto. Considere o trecho de código a seguir:

```
Worksheets("Plan2").Activate
```

Esta linha de código torna ativa a planilha Plan2, da pasta de trabalho onde o comando for executado. Na tabela a seguir é apresentado um breve resumo dos objetos nos quais está disponível o método Activate:

Objeto	Descrição
Chart, ChartObject	Torna este gráfico ativo.
Worksheet	Torna esta a planilha ativa. Equivalente a clicar na guia da planilha.
OLEObject	Ativa o objeto.
Pane	Ativa o painel. Se o painel não estiver na janela ativa, a janela à qual esse painel pertence também será ativada. Você não pode ativar um painel congelado.
Range	Ativa uma única célula, a qual precisa estar dentro da seleção atual. Para selecionar um intervalo de células, use o método Select.
Window	Traz a janela para a frente na ordem z. Isso não causa a execução de nenhuma macro Ativar_auto ou Desativar_auto que possa estar anexada à pasta de trabalho (use o método RunAutoMacros para executar essas macros).
Workbook	Ativa a primeira janela associada à pasta de trabalho. Isso não causa a execução de nenhuma macro Ativar_auto ou Desativar_auto que possa estar anexada à pasta de trabalho (use o método RunAutoMacros para executar essas macros).

Vamos considerar alguns exemplos do método Activate:

Exemplo 01: Este exemplo ativa Sheet1.

```
Worksheets("Sheet1").Activate
```

Exemplo 02: Este exemplo seleciona as células A1:C3 de Plan1 e, em seguida, faz de B2 a célula ativa.

```
Worksheets("Plan1").Activate  
Range("A1:C3").Select  
Range("B2").Activate
```

Exemplo 03: Este exemplo ativa a pasta Pasta4.xls. Se a pasta Pasta4.xls tiver várias janelas, o exemplo ativará a primeira janela, Pasta4.xls:1.

```
Workbooks("Pasta4.xls").Activate
```

O Método AutoFill:

Conforme o próprio nome sugere, este método é utilizado para fazer o preenchimento automático da faixa de células do objeto Range. A sintaxe para este método é a seguinte:

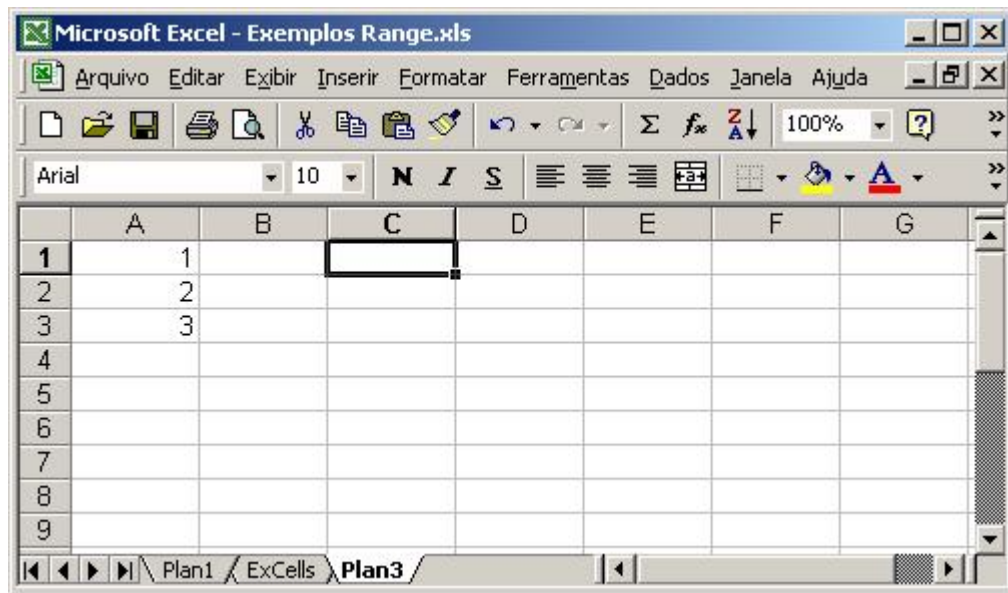
```
ObjetoRange.AutoFill(Destino, Tipo)
```

► **Destino:** É um objeto do tipo Range, para o qual as células serão preenchidas. Ou seja, representa a faixa de células a ser preenchida. O destino deve incluir, também, a faixa de origem, isto é, a faixa onde estão os dados a serem utilizados para o autopreenchimento.

► **Tipo:** Este parâmetro é opcional. Especifica o tipo de preenchimento. Pode ser uma das seguintes constantes XlFillType: xlFillDefault, xlFillSeries, xlFillCopy, xlFillFormats, xlFillValues, xlFillDays, xlFillWeekdays, xlFillMonths, xlFillYears, xlLinearTrend ou xlGrowthTrend. Se esse argumento for xlFillDefault ou for omitido, o Microsoft Excel selecionará o tipo de preenchimento mais apropriado com base no intervalo de origem. Por exemplo, se o intervalo de origem for A1:A3 e tiver os valores 1, 2 e 3, o Excel preenche o intervalo de destino com a sequência dos números: 4, 5, 6 e assim por diante. Outro exemplo, se o intervalo de origem for A1:A3 e tiver os valores 1, 3 e 5, o Excel preenche o intervalo de destino com a sequência de números: 7, 9, 11, 13 e assim por diante.

Vamos ver alguns exemplos práticos, para ilustrar o funcionamento deste método:

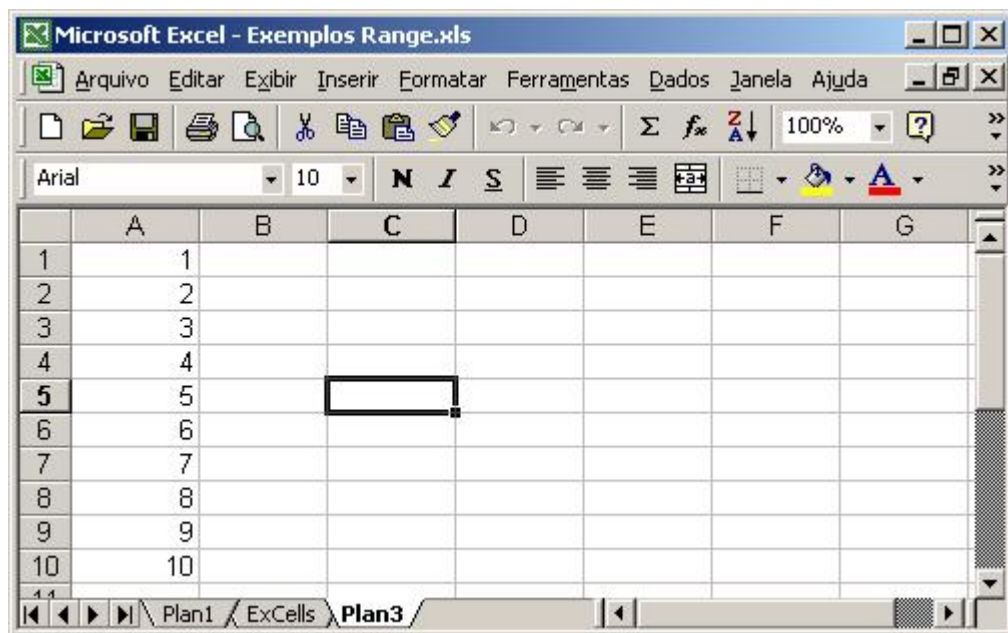
Exemplo 01: Considere a planilha indicada na figura a seguir:



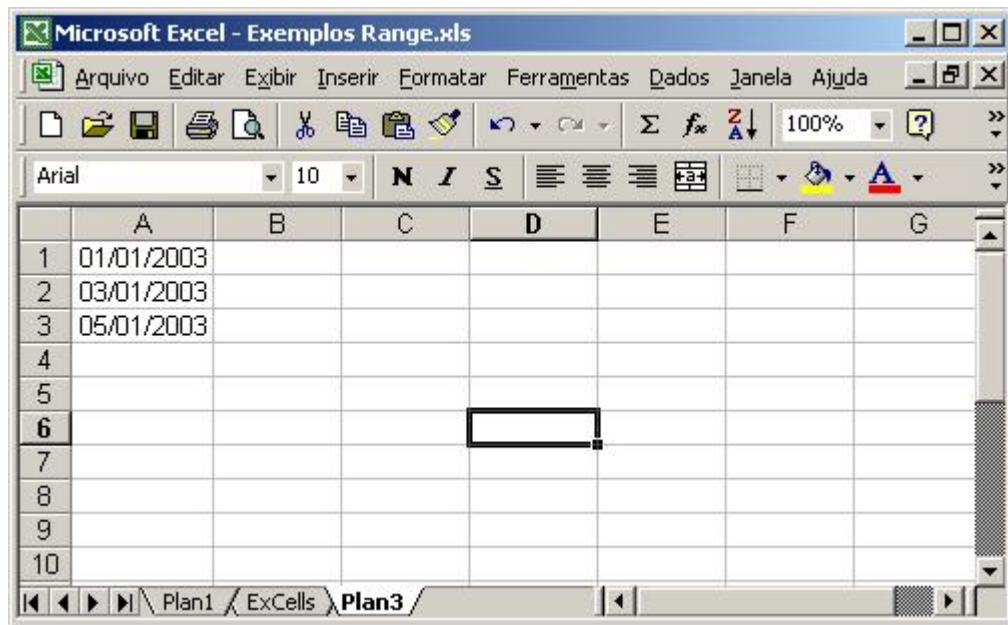
A seguir um exemplo de uso do método AutoFill:

```
Range("A1:A3").AutoFill Range("A1:A10")
```

Este comando irá preencher as células de A1:A10, tomando como referência os dados já existentes em A1:A3, ou seja, números variando de um em um. O resultado da execução deste comando, está indicado na Figura a seguir:



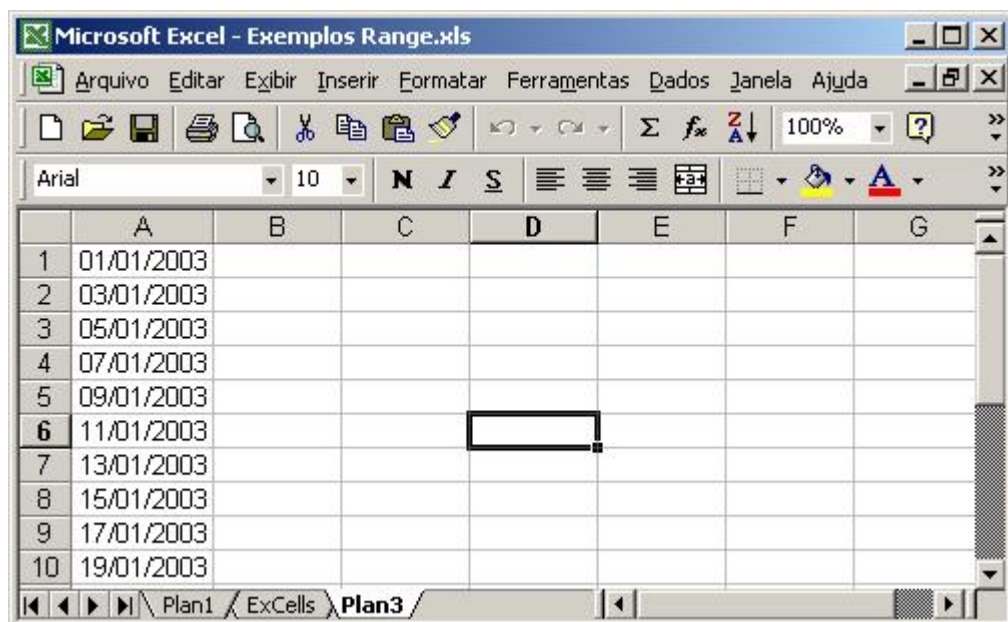
Exemplo 02: Considere a planilha indicada na figura a seguir:



A seguir um exemplo de uso do método AutoFill:

```
Range("A1:A3").AutoFill Range("A1:A10")
```

Este comando irá preencher as células de A1:A10, tomando como referência os dados já existentes em A1:A3, ou seja, datas variando de dois em dois dias. O resultado da execução deste comando, está indicado na Figura a seguir:



Lição 07: Objeto Range – Principais Métodos e Propriedades – Parte 2

Nesta lição continuaremos o estudo dos principais métodos, propriedades e coleções do objeto Range.

O Método AutoFilter:

Este método é utilizado para ter acesso ao recurso de AutoFiltro do Excel (Dados -> Filtrar -> AutoFiltro).

Para habilitar o recurso de AutoFiltro, fazendo com que o cabeçalho de cada coluna se torne uma caixa de listagem (veja todos os detalhes sobre AutoFiltro no Curso de Excel Avançado e VBA), basta utilizar o comando a seguir:

```
Dim Rng as Range  
Set Rng = Range("A1:E50")  
Rng.AutoFilter
```

O método AutoFilter, como o próprio nome sugere, também pode ser utilizado para aplicar um filtro a uma faixa de células, fazendo com que somente sejam exibidos, os dados que atendam a um ou mais critérios de filtragem. A sintaxe geral para o uso do método AutoFilter, está indicada a seguir:

```
ObjetoRange.AutoFilter(Campo,Critério1, Operador, Critério2)
```

O parâmetro Campo é um parâmetro numérico, que indica o número da coluna onde serão buscados os valores definidos como critérios. Por exemplo, se você tiver uma faixa de células de A1:E50, o valor 1 indica a coluna A, o valor 2 a coluna B, o valor 3 a coluna C, o valor 4 a coluna D e o valor 5 a coluna E. Os parâmetros critério1 e critério 2, definem valores que servirão como filtros para a coluna especificada. O parâmetro Operador pode ser um dos operadores lógicos aceitos pelo Excel, tais como o operador “AND”, “OR” e assim por diante.

Nota: Para uma descrição dos operadores lógicos do Excel e um estudo comparativo entre o operador E (AND) e o operador OU (OR), consulte o Curso de Excel Avançado e VBA, disponível para acesso Online no meu site: www.juliobattisti.com.br

Na tabela a seguir, apresento os diferentes valores possíveis para o parâmetro Operador, para o qual pode ser informado o valor numérico ou o valor de uma das constantes pré-definidas:

Valores disponíveis para o parâmetro - Operador	
Constante	Valor numérico correspondente
xlAnd	1
xlOr	2
xlTop10Items	3
xlBottom10Items	4
xlTop10Percent	5
xlBottom10Percent	6

Exemplo: Vamos a um exemplo prático de utilização do método AutoFilter. Para tal, vamos utilizar a planilha indicada na Figura a seguir, onde temos uma lista de pedidos, com campos para o código do cliente, o nome do funcionário, a data e valor do pedido e a cidade e país de destino:

	A	B	C	D	E	F	G
1	Número	Cliente	Funcionário	Data	Valor	Cidade	País
2	10248	VINET	José	04/07/1996	R\$ 32,38	Reims	França
3	10249	TOMSP	José	05/07/1996	R\$ 11,61	Münster	Alemanha
4	10250	HANAR	Maria	08/07/1996	R\$ 65,83	Rio de Janeiro	Brasil
5	10251	VICTE	Maria	08/07/1996	R\$ 41,34	Lyon	França
6	10252	SUPRD	Angela	09/07/1996	R\$ 51,30	Charleroi	Bélgica
7	10253	HANAR	Angela	10/07/1996	R\$ 58,17	Rio de Janeiro	Brasil
8	10254	CHOPS	Angela	11/07/1996	R\$ 22,98	Bern	Suíça
9	10255	RICSU	José	12/07/1996	R\$ 148,33	Genève	Suíça
10	10256	WELLI	José	15/07/1996	R\$ 13,97	Resende	Brasil
11	10257	HILAA	José	16/07/1996	R\$ 81,91	San Cristóbal	Venezuela
12	10258	ERNSH	Maria	17/07/1996	R\$ 140,51	Graz	Áustria
13	10259	CENTC	Maria	18/07/1996	R\$ 3,25	México D.F.	México
14	10260	OTTIK	Maria	19/07/1996	R\$ 55,09	Köln	Alemanha
15	10261	QUEDE	Angela	19/07/1996	R\$ 3,05	Rio de Janeiro	Brasil
16	10262	RATTC	Angela	22/07/1996	R\$ 48,29	Albuquerque	EUA
17	10263	ERNSH	Pedro	23/07/1996	R\$ 146,06	Graz	Áustria
18	10264	FOLKO	Pedro	24/07/1996	R\$ 3,67	Bräcke	Suécia
19	10265	BLONP	Pedro	25/07/1996	R\$ 55,28	Strasbourg	França
20	10266	WARTH	Pedro	26/07/1996	R\$ 25,73	Oulu	Finlândia
21	10267	FRANK	Maria	29/07/1996	R\$ 208,58	München	Alemanha

O trecho de código a seguir, irá habilitar o recurso de AutoFiltro, na planilha Pedidos, dentro da faixa A1:G20:

```
Worksheets("Pedidos").Activate  
Dim Rng as Range  
Set Rng = Range("A1:G21")  
Rng.AutoFilter
```

O resultado da execução deste comando, está indicado na Figura a seguir. Observe as setas no título de cada coluna, indicando que o título foi transformado em uma lista de valores, o que indica que o AutoFiltro está habilitado:

	A	B	C	D	E	F	G
1	Númer ▾	Client ▾	Funcionár ▾	Data ▾	Valor ▾	Cidade ▾	País ▾
2	10248	VINET	José	04/07/1996	R\$ 32,38	Reims	França
3	10249	TOMSP	José	05/07/1996	R\$ 11,61	Münster	Alemanha
4	10250	HANAR	Maria	08/07/1996	R\$ 65,83	Rio de Janeiro	Brasil
5	10251	VICTE	Maria	08/07/1996	R\$ 41,34	Lyon	França
6	10252	SUPRD	Angela	09/07/1996	R\$ 51,30	Charleroi	Bélgica

Vamos aperfeiçoar um pouco o nosso exemplo, para, além de habilitar o AutoFiltro, aplicar um ou mais critérios de filtragem.

A seguir acrescente mais uma linha, na qual estou filtrando somente os pedidos onde a coluna País tem o valor “Brasil”. Observe que informo o número que representa a posição da coluna: Coluna A é o 1, coluna B o 2 e assim por diante. Com isso, o número 7 indica a coluna G, ou seja, a coluna do País:

```
Worksheets("Pedidos").Activate  
Dim Rng as Range  
Set Rng = Range("A1:G21")  
Rng.AutoFilter  
Rng.AutoFilter 7,"Brasil"
```

O resultado da execução deste comando está indicado na Figura a seguir, onde conferimos que estão sendo exibidos apenas os pedidos para o Brasil:



The screenshot shows the Microsoft Excel interface with the file 'Exemplos Range.xls'. The 'Pedidos' worksheet is active, displaying a table with 7 columns: Número, Client, Funcionár, Data, Valor, Cidade, and País. The table is filtered to show only records where the 'País' column is 'Brasil'. The visible rows are 4, 7, 10, and 15. The status bar at the bottom indicates 'Plan1 / ExCells / Plan3 / Pedidos'.

	A	B	C	D	E	F	G
1	Númer	Client	Funcionár	Data	Valor	Cidade	País
4	10250	HANAR	Maria	08/07/1996	R\$ 65,83	Rio de Janeiro	Brasil
7	10253	HANAR	Angela	10/07/1996	R\$ 58,17	Rio de Janeiro	Brasil
10	10256	WELLI	José	15/07/1996	R\$ 13,97	Resende	Brasil
15	10261	QUEDE	Angela	19/07/1996	R\$ 3,05	Rio de Janeiro	Brasil
22							

Vamos alterar um pouco o nosso exemplo, para aplicar um critério mais avançado:

```
Worksheets("Pedidos").Activate  
Dim Rng As Range  
Set Rng = Range("A1:G21")  
  
' Uso a propriedade AutoFilterMode, do objeto Worksheet para  
' verificar se o modo de AutoFiltro está ativo.  
' Se estiver, eu chamo o método AutoFilter para desativá-lo  
  
' Estou desativando o AutoFiltro, para na sequência, ativá-lo  
' novamente. Isso é feito para limpar todos os filtros anteriores  
' e aplicar corretamente os novos critérios de filtragem  
  
If Worksheets("Pedidos").AutoFilterMode Then  
    Rng.AutoFilter  
End If  
  
Rng.AutoFilter  
Rng.AutoFilter 7, "Brasil", xlOr, "França"
```

Neste trecho de código a seguir, inicialmente uso a propriedade `AutoFilterMode`, do objeto `Worksheet`, para detectar se o AutoFiltro está habilitado na planilha. Se o AutoFiltro estiver habilitado, esta propriedade retorna `True`. Se for esse o caso, faço uma chamada ao método `AutoFilter` do objeto `Range`, para desabilitar o AutoFiltro. Você pode estar se perguntando: “Mas porque desabilitar o AutoFiltro”. Muito simples, para fazer com que seja exibida a lista completa novamente. Isso faz com que qualquer filtro que tenha sido aplicado anteriormente seja zerado e a lista completa é exibida. Se eu não fizesse isso, o Excel iria pesquisar somente nos registros retornados pela filtragem anterior e não em toda a lista. No nosso exemplo, se definíssemos novos critérios, sem desabilitar o AutoFiltro e habilitá-lo novamente, a pesquisa seria feita somente nos cinco registros exibidos na figura anterior. Com isso a técnica utilizada é desabilitar o AutoFiltro e habilitá-lo novamente, o que faz com que sejam exibidos todos os registros do objeto `Range`.

Outro ponto importante a observar é o uso do parâmetro `Operador`, o qual foi definido como `xlOr`, ou seja o operador OR. Ao usar um dos operadores, você deve também definir o parâmetro `CrITÉrio2`. No nosso exemplo, defini `crITÉrio1` como “Brasil”, o operador como `xlOr` e `crITÉrio2` como “França”, ou seja, o nosso filtro fica: “Brasil” OU “França”, dizendo ao Excel que quero que sejam exibidos apenas os pedidos do Brasil mais os pedidos da França, o que é comprovado pelo resultado da execução deste código, indicado na Figura a seguir:



	A	B	C	D	E	F	G
1	Númer	Client	Funcionár	Data	Valor	Cidade	País
2	10248	VINET	José	04/07/1996	R\$ 32,38	Reims	França
4	10250	HANAR	Maria	08/07/1996	R\$ 65,83	Rio de Janeiro	Brasil
5	10251	VICTE	Maria	08/07/1996	R\$ 41,34	Lyon	França
7	10253	HANAR	Angela	10/07/1996	R\$ 58,17	Rio de Janeiro	Brasil
10	10256	WELLI	José	15/07/1996	R\$ 13,97	Resende	Brasil
15	10261	QUEDE	Angela	19/07/1996	R\$ 3,05	Rio de Janeiro	Brasil
19	10265	BLONP	Pedro	25/07/1996	R\$ 55,28	Strasbourg	França

Lição 08: Objeto Range – Principais Métodos e Propriedades – Parte 3

Nesta lição continuaremos o estudo dos principais métodos, propriedades e coleções do objeto Range.

O Método Calculate:

Este método se aplica a outros objetos, além do objeto Range. O método Calculate é utilizado para calcular todas as pastas de trabalho (quando for usado com o objeto Workbook) abertas, uma planilha específica em uma pasta de trabalho (quando usado com o método Worksheet) ou um intervalo especificado de células em uma planilha (quando usado com o objeto Range), como se vê na tabela a seguir:

Para calcular	Siga este exemplo
Todas as pastas de trabalho abertas	Application.Calculate (ou apenas Calculate)
Uma planilha específica	Worksheets("Plan1").Calculate
Um intervalo especificado	Worksheets("Plan1").Range("A1:F50").Calculate

Considere o exemplo a seguir:

```
Worksheets("Plan1").UsedRange.Columns("A:E").Calculate
```

Este exemplo calcula as fórmulas das colunas A, B, C, D e E no intervalo utilizado na planilha Plan1.

O Método Clear:

Este método aplica-se a diversos objetos, tais como:

- ▶ ChartArea
- ▶ Legend
- ▶ Range

Quando aplicado a um destes objetos, o método Clear limpa todo o objeto. Quando aplicado a um objeto do tipo Caixa de listagem ou de combinação ActiveX, o método Clear remove todas as entradas da lista.

Sintaxe:

```
expressão.Clear
```

Considere os exemplos a seguir, no uso do método Clear:

Exemplo 01: Este exemplo limpa as fórmulas e a formatação das células A1:F40, na planilha Plan1:

```
Worksheets("Plan1").Range("A1:F40").Clear
```

Exemplo 02: O exemplo a seguir limpa a área do gráfico Graph1 (os dados e a formatação do gráfico).

```
Charts("Graph1").ChartArea.Clear
```

O Método ClearContents:

Este método limpa as fórmulas do intervalo. Limpa os dados de um gráfico mas deixa a formatação.

Sintaxe:

```
expressão.ClearContents
```

expressão: É obrigatória. Uma expressão que retorne um objeto Chart ou Range.

Considere os exemplos a seguir, de uso do método ClearContents:

Exemplo 01: Este exemplo limpa as fórmulas das células A1:F40 na planilha Plan1 mas deixa intacta a formatação.

```
Worksheets("Plan1").Range("A1:F40").ClearContents
```

Exemplo 02: Este exemplo limpa os dados do gráfico Graph1 mas deixa intacta a formatação.

```
Charts("Graph").ChartArea.ClearContents
```

O Método ClearFormats:

Este método é utilizado para limpar a formatação do objeto.

Sintaxe:

```
expressão.ClearFormats
```

expressão: É obrigatória. Uma expressão que retorne um objeto Range ou Graph.

Considere os exemplos a seguir, de uso do método ClearContents:

Exemplo 01: Este exemplo limpa toda a formatação das células A1:F40, na planilha Plan1.

```
Worksheets("Plan1").Range("A1:F40").ClearFormats
```

Exemplo 02: Este exemplo limpa a formatação do primeiro gráfico que foi adicionado à planilha Plan1.

```
Worksheets("Plan1").ChartObjects(1).Chart.ChartArea.ClearFormats
```

Lição 09: Objeto Range – Principais Métodos e Propriedades – Parte 4

Nesta lição continuaremos o estudo dos principais métodos, propriedades e coleções do objeto Range.

O Método Copy:

Este método apresenta diferentes sintaxes, conforme descrevo logo a seguir

Sintaxe 1: Copia o objeto range ou figura selecionada para a Área de transferência.

```
expressão.Copy
```

Sintaxe 2: Copia o Range para o intervalo especificado como destino ou para a Área de transferência:

```
expressão.Copy(Destination)
```

Sintaxe 3: Copia a planilha para um outro lugar na pasta de trabalho.

```
expressão.Copy(Before, After)
```

Nota: Se você não especificar Before ou After, o Microsoft Excel criará uma nova pasta de trabalho contendo a planilha copiada. Vamos a alguns exemplos para entender o funcionamento do método Copy.

Exemplo 01: Este exemplo copia a planilha Plan1, colocando a cópia depois da planilha Plan3:

```
Worksheets("Plan1").Copy after := Worksheets("Plan3")
```

Exemplo 02: Este exemplo copia o intervalo de células usado em Plan1, cria uma nova planilha e, em seguida, cola os valores do intervalo copiado na nova planilha.

```
Worksheets("Plan1").UsedRange.Copy  
Set newSheet = Worksheets.Add  
newSheet.Range("A1").PasteSpecial Paste:=xlValues
```

Nota: Neste exemplo também foi utilizado o método PasteSpecial, o qual será detalhado em uma das próximas lições.

Exemplo 03: Este exemplo copia as fórmulas das células A1:D4 da planilha Plan1 para as células E5:H8 da planilha Plan2.

```
Worksheets("Plan1").Range("A1:D4").Copy destination:=Worksheets("Plan2").Range("E5")
```

Neste exemplo usei o parâmetro destination, para definir para onde será feita a cópia.

O Método Cut:

Este método é utilizado para recortar o conteúdo do objeto Range para a Área de transferência ou o cola em um local especificado.

Sintaxe:

```
expressão.Cut(Destination)
```

Importante: O intervalo recortado precisa ser constituído de células adjacentes. Somente gráficos incorporados podem ser recortados.

Considere o exemplo a seguir, para entender o funcionamento do método Copy.

Este exemplo recorta o intervalo A1:G37 da planilha Plan1 e o coloca na Área de transferência.

```
Worksheets("Plan1").Range("A1:G37").Cut
```

O Método CopyFromRecordset:

Este método copia o conteúdo de um objeto Recordset ADO ou DAO para uma planilha, começando no canto superior esquerdo do intervalo especificado. Se o objeto Recordset tiver campos contendo objetos OLE, este método falhará. Com o uso de um objeto RecordSet (veja os módulos 2 e 3 do curso de Access Avançado e VBA, para todos os detalhes sobre objetos do tipo RecordSet), você pode copiar dados de uma tabela de um banco de dados do Access, ou de uma tabela de um banco de dados SQL Server ou ORACLE, diretamente para uma planilha do Excel. Na prática, com o método CopyFromRecordset, você pode copiar dados a partir de qualquer fonte de dados externa, que esteja em um formato capaz de ser representado como um RecordSet.

Vamos fazer um exemplo prático de uso deste método, para mostrar como ele funciona. Considere o trecho de código a seguir:

```
Dim rs As DAO.Recordset

Set rs = DBEngine.OpenDatabase("C:\dados\pedidos.mdb").OpenRecordset("Pedidos")

Worksheets("PlanPedidos").Range("A1").CopyFromRecordset rs, 10, 6
```

Importante: Para que este código funcione corretamente, você deve fazer referência a biblioteca DAO. Para maiores detalhes sobre como fazer referência a uma biblioteca do VBA, consulte o item **Fazendo referência a Bibliotecas de Objetos**, na Lição 03 do Módulo 2

O primeiro comando declara uma variável do tipo Recordset. O segundo comando faz uma referência a tabela Pedidos do banco de dados C:\dados\pedidos.mdb. A terceira linha é que, efetivamente, utiliza o método CopyFromRecordset, do objeto Range, para copiar as 10

primeiras linhas (segundo parâmetro igual a 10) e copiar os seis primeiros campos de cada registro (terceiro parâmetro igual a seis).

O resultado da execução deste trecho de código está indicado na Figura a seguir:

	A	B	C	D	E	F	G	H
1	10248	VINET	5	04/08/1994	01/09/1994	16/08/1994		
2	10249	TOMSP	6	05/08/1994	16/09/1994	10/08/1994		
3	10250	HANAR	4	08/08/1994	05/09/1994	12/08/1994		
4	10251	VICTE	3	08/08/1994	05/09/1994	15/08/1994		
5	10252	SUPRD	4	09/08/1994	06/09/1994	11/08/1994		
6	10253	HANAR	3	10/08/1994	24/08/1994	16/08/1994		
7	10254	CHOPS	5	11/08/1994	08/09/1994	23/08/1994		
8	10255	RICSU	9	12/08/1994	09/09/1994	15/08/1994		
9	10256	WELLI	3	15/08/1994	12/09/1994	17/08/1994		
10	10257	HILAA	4	16/08/1994	13/09/1994	22/08/1994		
11								

Observe que foram copiados os 10 primeiros registros da tabela pedidos, sendo que foram copiados somente os seis primeiros campos (colunas) de cada registro. O método CopyFromRecordSet é especialmente útil, para situações onde você deseja automatizar um grande volume de importação de dados em um outro formato para dentro de planilhas do Excel.

O Método CreateNames:

Este método é utilizado para criar nomes no intervalo especificado, com base nos rótulos de texto da planilha. A melhor maneira de entender o uso deste método é através de alguns exemplos práticos.

Exemplo 01: Este exemplo cria nomes para as células B1:B3 com base no texto das células A1:A3. Observe que você precisa incluir no intervalo as células que contêm os nomes, muito embora os nomes sejam criados somente para as células B1:B3.

```
Dim rng as Range
Set rng = Worksheets("Plan1").Range("A1:B3")
rng.CreateNames Left:=True
```

A sintaxe do método CreateNames é a seguinte:

```
ObjetoRange.CreateNames(Top, Left, Bottom, Right)
```

Expressão: É obrigatória. Uma expressão que retorne um objeto Range.

Top: É do tipo Variant e é opcional. True para criar nomes usando rótulos da linha superior da faixa. O valor padrão é False.

Left: É do tipo Variant e é opcional. True para criar nomes usando rótulos da coluna esquerda da faixa. O valor padrão é False.

Bottom: É do tipo Variant e é opcional. True para criar nomes usando rótulos da linha inferior da faixa. O valor padrão é False.

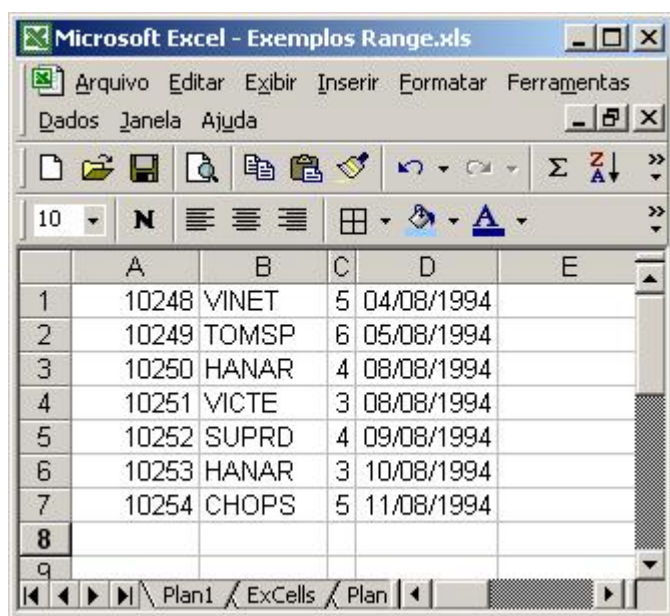
Right: É do tipo Variant e é opcional. True para criar nomes usando rótulos da coluna direita da faixa. O valor padrão é False.

No nosso exemplo usamos o parâmetro Left, pois os rótulos a serem utilizados estavam na coluna A, ou seja, A1:A3, a coluna bem a esquerda (Left), do objeto Range.

Importante: Se você não especificar Top, Left, Bottom ou Right, o Microsoft Excel determinará a localização dos rótulos de texto baseando-se na forma do intervalo especificado.

Propriedade CurrentRegion:

Esta propriedade retorna um objeto Range representando a região atual. A região atual é um intervalo limitado por qualquer combinação de linhas e colunas em branco. É do tipo somente leitura. Considere a figura a seguir:



Neste exemplo a região atual é A1:D7, ou seja, com este método, o Excel tenta detectar a região da planilha que está sendo utilizada, onde estão os dados. Se houver células, linhas ou colunas em branco, dentro da região em uso, o Excel poderá retornar uma faixa incorreta.

Essa propriedade é útil para muitas operações que expandem automaticamente a seleção para incluir toda a região atual, tais como o método AutoFormat. Esta propriedade não pode ser usada em uma planilha protegida.

Vamos apresentar alguns exemplos da propriedade CurrentRegion:

Exemplo 01: Este exemplo ativa a planilha Plan1 e depois seleciona a região atual da planilha.

```
Worksheets("Plan1").Activate  
ActiveCell.CurrentRegion.Select
```

Exemplo 02: Você pode declarar uma variável do tipo Range e depois atribuir o valor retornado por CurrentRegion, para esta variável, conforme exemplo do trecho de código a seguir:

```
Worksheets("Plan1").Activate  
Dim rng as Range  
Set rng = ActiveCell.CurrentRegion
```


Lição 10: Objeto Range – Principais Métodos e Propriedades – Parte 5

Nesta lição continuaremos o estudo dos principais métodos, propriedades e coleções do objeto Range.

O Método Find:

Este método é utilizado para localizar informações específicas em um intervalo/faixa, e retorna um objeto Range representando a primeira célula onde essas informações, caso sejam encontradas células que atendam aos critérios especificados. O método irá retornar Nothing, se nenhuma coincidência for encontrada. Não afeta a seleção da célula ativa.

Antes de vermos a sintaxe completa deste método, vamos ver um exemplo bem simples, o qual facilitará o entendimento da sintaxe do método Find.

Exemplo 01: Para o nosso exemplo, considere os dados indicados na figura a seguir:



	A	B	C	D	E	F	G
1	Número	Cliente	Funcionário	Data	Valor	Cidade	País
2	10248	VINET	José	04/07/1996	R\$ 32,38	Reims	França
3	10249	TOMSP	José	05/07/1996	R\$ 11,61	Münster	Alemanha
4	10250	HANAR	Maria	08/07/1996	R\$ 65,83	Rio de Janeiro	Brasil
5	10251	VICTE	Maria	08/07/1996	R\$ 41,34	Lyon	França
6	10252	SUPRD	Angela	09/07/1996	R\$ 51,30	Charleroi	Bélgica
7	10253	HANAR	Angela	10/07/1996	R\$ 58,17	Rio de Janeiro	Brasil
8	10254	CHOPS	Angela	11/07/1996	R\$ 22,98	Bern	Suíça
9	10255	RICSU	José	12/07/1996	R\$ 148,33	Genève	Suíça
10	10256	WELLI	José	15/07/1996	R\$ 13,97	Resende	Brasil
11							

```
Dim rng As Range
Worksheets("ExFind").Activate

Set rng = Range("A1:G10")

Dim rngResult As Range

Set rngResult = rng.Find("Bélgica", LookIn:=xlValues)
rngResult.Cells(1, 1).Select
rngResult.Cells(1, 1).Font.Bold = True
rngResult.Cells(1, 1).Font.ColorIndex = 3
```

Vamos analisar o código do nosso exemplo:

1. Inicialmente é declarada uma variável do tipo Range (rng), ativada a planilha ExFind e definida a faixa A1:G10 para a variável rng:

```
Dim rng As Range
Worksheets("ExFind").Activate
Set rng = Range("A1:G10")
```

2. Em seguida, declaro uma segunda variável do tipo Range (rngResult), a qual conterá o resultado da execução do método Find:

```
Dim rngResult As Range
```

3. O próximo passo é executar o método Find. No nosso exemplo, executo o método Find, definindo como critério de pesquisa “Bélgica” e defino o parâmetro LookIn com o valor xlValues, que significa buscar nos valores das células. O método Find irá retornar uma faixa de uma única célula, justamente a primeira célula que atende ao critério de pesquisa definido no primeiro parâmetro, que no nosso exemplo é “Bélgica”.

```
Set rngResult = rng.Find("Bélgica", LookIn:=xlValues)
rngResult.Cells(1, 1).Select
rngResult.Cells(1, 1).Font.Bold = True
rngResult.Cells(1, 1).Font.ColorIndex = 3
```

Ao executar este exemplo o método Find localiza a primeira ocorrência de Bélgica em G6 e depois atua sobre esta célula, colocando-a em Negrito e cor de fonte vermelha, conforme indicado na figura a seguir:



	A	B	C	D	E	F	G
1	Número	Cliente	Funcionário	Data	Valor	Cidade	País
2	10248	VINET	José	04/07/1996	R\$ 32,38	Reims	França
3	10249	TOMSP	José	05/07/1996	R\$ 11,61	Münster	Alemanha
4	10250	HANAR	Maria	08/07/1996	R\$ 65,83	Rio de Janeiro	Brasil
5	10251	VICTE	Maria	08/07/1996	R\$ 41,34	Lyon	França
6	10252	SUPRD	Angela	09/07/1996	R\$ 51,30	Charleroi	Bélgica
7	10253	HANAR	Angela	10/07/1996	R\$ 58,17	Rio de Janeiro	Brasil
8	10254	CHOPS	Angela	11/07/1996	R\$ 22,98	Bern	Suíça
9	10255	RICSU	José	12/07/1996	R\$ 148,33	Genève	Suíça
10	10256	WELLI	José	15/07/1996	R\$ 13,97	Resende	Brasil
11							

Muito bem, agora que vimos um exemplo prático, vamos estudar a sintaxe do método Find, em mais detalhes:

Sintaxe do método Find:

`Range.Find(What, After, LookIn, LookAt, SearchOrder, SearchDirection, MatchCase, MatchByte)`

A seguir descrevo os parâmetros que podem ser passados para o método Find:

- ▶ **What:** É do tipo variant e é obrigatório. Os dados pelos quais procurar. Pode ser uma cadeia de caracteres ou qualquer tipo de dados do Microsoft Excel.
- ▶ **After:** É do tipo variant e é opcional. A célula depois da qual você deseja que a busca inicie. Corresponde à posição da célula ativa quando uma pesquisa é feita a partir da interface do usuário. Observe que After precisa ser uma única célula no intervalo. Lembre-se de que a pesquisa começa depois desta célula; a célula especificada não é pesquisada até que o método dê a volta e chegue à ela. Se você não especificar esse argumento, a pesquisa começará após a célula do canto superior esquerdo do intervalo.
- ▶ **LookIn:** É do tipo variant e é opcional. Pode ser uma das seguintes constantes: xlFormulas, xlValues ou xlComments.
- ▶ **LookAt:** É do tipo variant e é opcional. Pode ser uma das seguintes constantes: xlPart ou xlWhole. Esta opção define se o critério deve ser ser uma coincidência inteira (xlWhole) ou o critério pesquisado pode fazer parte do valor de um célula (xlPart). Por exemplo, se você pesquisar por José da Silva e definir este parâmetro como xlWhole, será localizado somente a primeira ocorrência de José da Silva. Se você definir xlPart, será localizada uma ocorrência como Antônio José Paulo da Silva (pois José da Silva faz parte deste nome maior).
- ▶ **SearchOrder:** É do tipo variant e é opcional. Pode ser uma das seguintes constantes: xlByColumns ou xlByRows.
- ▶ **SearchDirection:** É do tipo variant e é opcional. Pode ser uma das seguintes constantes: xlNext ou xlPrevious. A constante padrão é xlNext. O valor xlNext faz com que a pesquisa seja efetuada a partir da posição atual do cursor, em direção ao final da faixa e o valor xlPrevious, faz com que a pesquisa seja efetuada a partir da posição atual do cursor, em direção ao início da faixa.
- ▶ **MatchCase:** É do tipo variant e é opcional. True para fazer a pesquisa coincidir maiúsculas e minúsculas. O valor padrão é False.
- ▶ **MatchByte:** É do tipo variant e é opcional. Usado somente se você tiver selecionado ou instalado suporte de língua de byte duplo. True para que caracteres de dois bytes só coincidam com caracteres de dois bytes. False para que os caracteres de dois bytes coincidam com seus equivalentes de um byte.

Mais algumas observações sobre o método Find:

As definições de LookIn, LookAt, SearchOrder e MatchByte são salvas toda vez que você usa esse método. Se você não especificar valores para esses argumentos na próxima vez que você chamar o método, os valores salvos serão usados. A definição desses argumentos altera as definições da caixa de diálogo Localizar, e a alteração das definições da caixa de diálogo Localizar faz com que os valores salvos, usados quando você omite os argumentos, sejam alterados. Para evitar problemas, defina explicitamente esses argumentos toda vez que você usar esse método.

Nota: Você pode usar os métodos FindNext e FindPrevious para repetir a pesquisa. Estes métodos serão abordados na próxima lição.

Quando a pesquisa chega ao final do intervalo de pesquisa especificado, ela continua do início do intervalo. Para interromper uma pesquisa quando esse retorno ocorre, salve o endereço da primeira célula encontrada e, em seguida, compare o endereço de cada célula sucessiva encontrada com esse endereço salvo.

Para encontrar células coincidentes com padrões de pesquisa mais complicados, use uma instrução For Each...Next com o operador Like. Por exemplo, o código seguinte pesquisa todas as células do intervalo A1:C5 que usam uma fonte cujo nome começa com as letras "Cour". Quando o Microsoft Excel encontra uma coincidência, ele altera a fonte para Times New Roman.

```
Dim c as Range
For Each c In (A1:C5)
    If c.Font.Name Like "Cour*" Then
        c.Font.Name = "Times New Roman"
    End If
Next
```

Lição 11: Objeto Range – Principais Métodos e Propriedades – Parte 6

Nesta lição continuaremos o estudo dos principais métodos, propriedades e coleções do objeto Range.

O Método FindNext:

O método FindNext é utilizado para continuar uma pesquisa que tenha começado com o método Find. Localiza a próxima célula que coincida com as mesmas condições que foram definidas para a última execução do método Find e retorna um objeto Range representando essa célula. Não afeta a seleção da célula ativa.

Sintaxe:

`Range.FindNext(After)`

► **After:** É do tipo variant e é opcional. Se o parâmetro After não for especificado, a busca iniciará novamente no canto superior esquerdo da faixa representada pelo objeto Range. Corresponde à posição da célula ativa quando uma pesquisa é feita a partir da interface do usuário, ou seja, define a célula a partir da qual deve ser iniciada a pesquisa. Observe que After precisa ser uma única célula no intervalo. Lembre-se de que a pesquisa começa depois desta célula; a célula especificada não é pesquisada até que o método dê a volta e chegue à ela. Se esse argumento não for especificado, a pesquisa começará após a célula do canto superior esquerdo do intervalo.

Vamos a um exemplo prático de uso de método FindNex em conjunto com o método Find.

Exemplo: Este exemplo localiza todas as células do intervalo A1:A500 que contêm o valor 2 e define a cor de fundo destas células como Cinza.

```
Public Sub ExFindNext()

Dim wks As Worksheet
Dim rng As Range

Set wks = Worksheets("Plan1")
Set rng = wks.Range("A8:C17")

Set c = rng.Find(2, LookIn:=xlValues)

If Not c Is Nothing Then
    firstAddress = c.Address

    Do
        c.Interior.Pattern = xlPatternGray50
        Set c = rng.FindNext(c)
    Loop While Not c Is Nothing And c.Address <> firstAddress

End Sub
```

End If

End Sub

Vamos ver alguns detalhes sobre o código deste exemplo:

1. Inicialmente faço as declarações e variáveis, criando uma variável do tipo Worksheet, a qual é associada com a planilha Plan1 e uma variável Range, a qual é associada com o intervalo A8:C17

```
Dim wks As Worksheet
Dim rng As Range

Set wks = Worksheets("Plan1")
Set rng = wks.Range("A8:C17")
```

2. O próximo passo é utilizar o método Find, do objeto Range, para localizar a primeira ocorrência do valor 2, na faixa A8:C17. O método Find irá retornar um objeto Range, de uma única célula – a célula onde for encontrada a primeira ocorrência do valor 2. A faixa representada por esta célula é armazenada na variável c.

```
Set c = rng.Find(2, LookIn:=xlValues)
```

3. O terceiro passo do nosso exemplo, é utilizar um laço Do...While. Antes de iniciar o laço, faço um teste para ver se o método Find retornou algum valor, ou seja, se foi encontrada alguma ocorrência do valor 2, dentro da faixa A8:C17. A expressão – c Is Nothing – irá retornar verdadeiro somente quando nenhuma ocorrência for encontrada. Quando for encontrada alguma ocorrência, - c Is Nothing - retornará Falso. Por isso que usamos o operador Not, antes do – c Is Nothing, ou seja, quando for encontrada uma ocorrência, Not c Is Nothing é igual a Not Falso, ou seja, Verdadeir. Em resumo, o teste Not c Is Nothing retorna Verdadeiro sempre que o método Find encontrar uma célula com o valor pesquisado. Quando o teste for verdadeiro, o endereço da célula onde este primeiro valor foi encontrado, é armazenado na variável firstAddress. Este endereço é importante, pois será utilizado pelo método FindNext, para que a busca feita pelo FindNext seja feita a partir da célula seguinte a ocorrência anterior e não a partir do início da faixa novamente. A própria Ajuda do Excel recomenda esta técnica.

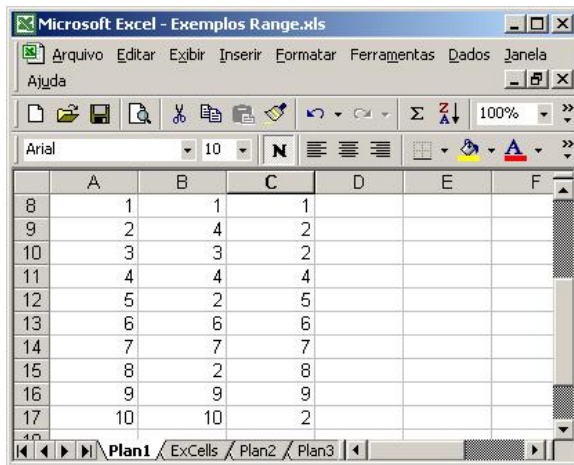
```
If Not c Is Nothing Then
    firstAddress = c.Address
```

4. A seguir utilizo um laço Do...While (com o teste no final). Dentro deste laço, o primeiro passo é definir a cor de fundo da célula representada pela objeto Range armazenado na variável c (c.Interior.Pattern = xlPatternGray50). Para definir a cor de fundo de uma célula utilizamos o objeto Interior (que é uma propriedade de Range) e a sua propriedade Pattern – Interior.Pattern. Em seguida uso o método FindNext, para pesquisar novamente o valor 2 na faixa A8:C17 e passo como parâmetro para o método FindNext, o endereço da célula contida na variável c. Como a variável c contém o endereço da célula retornada pela execução da última pesquisa, a próxima pesquisa se iniciará a partir da célula seguinte a célula representada pela variável c, o que faz sentido, ou seja, a pesquisa se inicia a partir da célula

seguinte a última ocorrência. No final do laço texto para ver se foi encontrado um valor (Not c Is Nothing) e para ver se não chegamos novamente ao início da faixa (c.Address <> firstAddress), pois quando este segundo teste for falso (isto é, quando estivermos de volta ao primeiro endereço), o laço será encerrado. Sem esta segunda parte do teste, o laço continuaria sendo executado, indefinidamente.

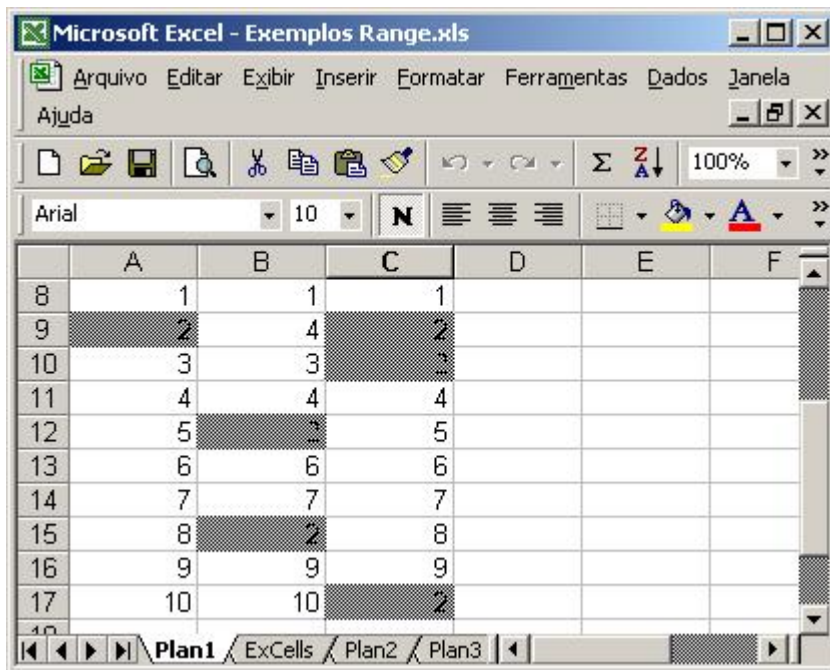
```
Do
    c.Interior.Pattern = xlPatternGray50
    Set c = rng.FindNext(c)
Loop While Not c Is Nothing And c.Address <> firstAddress
End If
```

Na figura a seguir apresento um exemplo da planilha, antes da execução do nosso exemplo:



	A	B	C	D	E	F
8	1	1	1			
9	2	4	2			
10	3	3	2			
11	4	4	4			
12	5	2	5			
13	6	6	6			
14	7	7	7			
15	8	2	8			
16	9	9	9			
17	10	10	2			

Na figura a seguir, é exibido o resultado, após a execução do código do nosso exemplo:



	A	B	C	D	E	F
8	1	1	1			
9	2	4	2			
10	3	3	2			
11	4	4	4			
12	5	2	5			
13	6	6	6			
14	7	7	7			
15	8	2	8			
16	9	9	9			
17	10	10	2			

Lição 12: Objeto Range – Principais Métodos e Propriedades – Parte 7

Nesta lição continuaremos o estudo dos principais métodos, propriedades e coleções do objeto Range.

O Método FindPrevious:

Continua uma pesquisa que tenha começado com o método Find. Localiza a célula anterior que coincida com as mesmas condições e retorna um objeto Range representando essa célula. Não afeta a seleção da célula ativa.

Sintaxe:

ObjetoRange.FindPrevious(After)

► **After:** É do tipo variant e é opcional. A célula antes da qual você deseja pesquisar. Corresponde à posição da célula ativa quando uma pesquisa é feita a partir da interface do usuário. Observe que After precisa ser uma única célula no intervalo. Lembre-se de que a pesquisa começa antes desta célula; a célula especificada não é pesquisada até que o método dê a volta e chegue até ela. Se esse argumento não for especificado, a pesquisa começará antes da célula do canto superior esquerdo do intervalo.

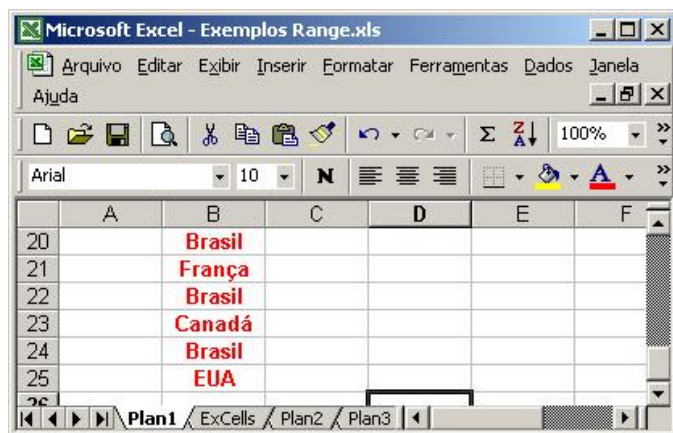
Exemplo: Este exemplo mostra como o método FindPrevious é usado com os métodos Find e FindNext. Antes de executar este exemplo, certifique-se de que a planilha Plan1 contenha pelo menos três ocorrências da palavra “Brasil”, na coluna B.

```
Set fc = Worksheets("Plan1").Columns("B").Find(what:="Brasil")
MsgBox "A primeira ocorrência foi em: " & fc.Address

Set fc = Worksheets("Plan1").Columns("B").FindNext(after:=fc)
MsgBox "A segunda ocorrência foi em: " & fc.Address

Set fc = Worksheets("Plan1").Columns("B").FindPrevious(after:=fc)
MsgBox "A terceira ocorrência foi em: " & fc.Address
```

Considere o exemplo da figura a seguir:



Ao executar o código do exemplo anterior, obteremos a sequência de telas indicadas a seguir:

Inicialmente o método Find encontra a primeira ocorrência de Brasil, que é em B20:



Em seguida, o método FindNext, encontra a segunda ocorrência de Brasil, que é em B22:



Por último, o método FindPrevious encontra a ocorrência anterior a ocorrência retornada pela última pesquisa. Como a última havia sido B22, o método FindPrevious volta no intervalo, em direção ao início e encontra a ocorrência em B20:



Propriedade Fórmula:

Retorna ou define a fórmula do objeto, em notação de estilo A1 e no idioma da macro. É do tipo Variant de leitura e gravação para objetos Range, String de leitura e gravação para todos os demais objetos.

Importante: Esta propriedade não está disponível para fontes de dados OLAP.

Se a célula contiver uma constante, essa propriedade retornará a constante. Se a célula estiver vazia, Formula retornará uma cadeia de caracteres vazia. Se a célula contiver uma fórmula, a propriedade Formula retornará a fórmula como uma cadeia de caracteres no mesmo formato em que ela seria exibida na barra de fórmulas (**incluindo o sinal de igual**).

Quando você define o valor ou fórmula de uma célula com uma data, o Microsoft Excel verifica se a célula já está formatada com um dos formatos numéricos de data ou hora. Se não estiver, o Microsoft Excel alterará o formato numérico para o formato padrão de data abreviada.

Se o intervalo for uni ou bidimensional, você poderá definir a fórmula como uma matriz do Visual Basic com as mesmas dimensões. Da mesma forma, você pode colocar a fórmula em uma matriz do Visual Basic.

A definição da fórmula para um intervalo de várias células preenche todas as células do intervalo com a mesma fórmula.

Exemplo 01: O exemplo a seguir define a fórmula da célula C1, na planilha Plan1:

```
Worksheets("Plan1").Range("C1").Formula = "=$A$1+$B$1"
```

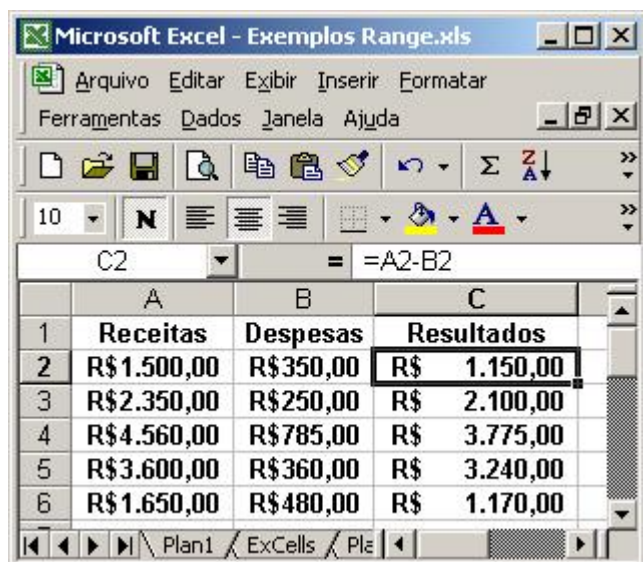
Exemplo 02: Também é possível utilizar endereços relativos e a propriedade Fórmula, para definir as fórmulas para uma faixa de células, de tal maneira que o Excel as adapta, automaticamente. Considere o exemplo da planilha a seguir, onde temos dados nas colunas A – Receitas, na coluna B – Despesas e queremos calcular o Resultado na coluna C, fazendo Receitas – Despesas. Evidentemente que para solucionar este problema não é preciso o uso do VBA. Este exemplo é apenas para ilustrar o uso da propriedade Fórmula, com endereços relativos. Vamos usar a planilha indicada na Figura a seguir:



	A	B	C	D
1	Receitas	Despesas	Resultados	
2	R\$1.500,00	R\$350,00		
3	R\$2.350,00	R\$250,00		
4	R\$4.560,00	R\$785,00		
5	R\$3.600,00	R\$360,00		
6	R\$1.650,00	R\$480,00		

```
Dim rng as Range  
Set rng = Worksheets("Financas").Range("C2:C6")  
  
Rng.Formula = "=A2-B2"
```

Após a execução deste código, as fórmulas da faixa C2:C6 serão definidas e automaticamente adaptadas. Para a célula C2 será definida a fórmula =A2-B2, para a célula C3 será definida a fórmula A3-B3 e assim por diante. Além de definir as fórmulas, estas serão calculadas, conforme indicado na figura a seguir. Observe na Barra de Fórmulas, a fórmula definida para a célula ativa.



The screenshot shows the Microsoft Excel interface with the file 'Exemplos Range.xls'. The menu bar includes 'Arquivo', 'Editar', 'Exibir', 'Inserir', 'Formatar', 'Ferramentas', 'Dados', 'Janela', and 'Ajuda'. The toolbar contains various icons for file operations and formatting. The active cell is C2, and the formula bar displays '=A2-B2'. The worksheet 'Plan1' contains the following data:

	A	B	C
1	Receitas	Despesas	Resultados
2	R\$ 1.500,00	R\$ 350,00	R\$ 1.150,00
3	R\$ 2.350,00	R\$ 250,00	R\$ 2.100,00
4	R\$ 4.560,00	R\$ 785,00	R\$ 3.775,00
5	R\$ 3.600,00	R\$ 360,00	R\$ 3.240,00
6	R\$ 1.650,00	R\$ 480,00	R\$ 1.170,00

Se uma célula contiver um valor constante (texto, número, data, etc.), a propriedade `Formula` irá retornar o valor da célula. Também é possível definir o valor de uma célula, usando a propriedade `Formula`, como no exemplo a seguir, onde defini o valor para a célula A1:

```
Range("A1").Formula = "Controle Financeiro"
```

Lição 13: Objeto Range – Principais Métodos e Propriedades – Parte 8

Nesta lição continuaremos o estudo dos principais métodos, propriedades e coleções do objeto Range.

O Método Sort:

Este método é utilizado para classificar uma faixa associada com o objeto Range. Se não for especificada uma faixa, será classificada a região atual, na qual está a célula ativa. Antes de entendermos a sintaxe do método Sort, vamos apresentar um exemplo prático, o que facilitará o entendimento deste método, dos seus parâmetros e da sua sintaxe.

Exemplo 01: O exemplo a seguir, faz a ordenação da faixa A1:C20, na planilha Plan1, primeiro pela coluna A e depois pela coluna B. Por exemplo, se a coluna A for o campo PaísDeDestino e a coluna B for o campo CidadeDeDestino, a listagem será classificada por País e, dentro do mesmo país, pela Cidade.

```
Worksheets("Plan1").Range("A1:C20").Sort _  
    Key1:=Worksheets("Sheet1").Range("A1"), _  
    Key2:=Worksheets("Sheet1").Range("B1")
```

Neste exemplo foram utilizados os parâmetros nomeados Key1 e Key2.

Nota: Este comando é um único comando, mas foi dividido em linhas (através da inserção do caractere de quebra de linha _), por questões de clareza, para facilitar a leitura e a interpretação do comando.

Exemplo 02: Este exemplo ordena a região atual, a qual contém a célula A1, na planilha Plan1. A ordenação é feita pela Data (coluna A) e utilizando uma linha para o título das colunas, caso esta linha exista. O método Sort determina a região ativa, automaticamente.

```
Worksheets("Plan1").Range("A1").Sort _  
    Key1:=Worksheets("Plan1").Columns("A"), _  
    Header:=xlGuess
```

Sintaxe para o método Sort:

```
Range.Sort(Key1, Order1, Key2, Type, Order2, Key3, Order3, Header,  
OrderCustom, MatchCase, Orientation, SortMethod)
```

- ▶ **Key1:** É um parâmetro opcional e é do tipo Variant. Define o primeiro campo, com base no qual a faixa será classificada. Pode ser um nome associado a coluna ou um objeto do tipo range, como por exemplo “ColunaPaís” ou Celss(1,1).
- ▶ **Order1:** É um parâmetro opcional e do tipo Variant. Define a ordem de classificação para Key1. Pode assumir um dos seguintes valores: xlAscending (ordem crescente) or xlDescending (ordem decrescente). O valor padrão é xlAscending.

- ▶ **Key2:** É um parâmetro opcional e é do tipo Variant. Define o segundo campo, com base no qual a faixa será classificada. Pode ser um nome associado a coluna ou um objeto do tipo range, como por exemplo “ColunaPaís” ou Celss(1,1). Se você omitir este argumento, a faixa será classificado apenas pela campo definido em Key1. Não utilize este argumento quando você estiver classificando uma Tabela Dinâmica.
- ▶ **Type:** É opcional e do tipo Variant. Este parâmetro especifica quais elementos deverão ser classificados. Pode ser uma das seguintes constantes: xlSortLabels or xlSortValues. Somente deve ser utilizado quando você for classificar uma Tabela Dinâmica.
- ▶ **Order2:** É um parâmetro opcional e do tipo Variant. Define a ordem de classificação para Key2. Pode assumir um dos seguintes valores: xlAscending (ordem crescente) or xlDescending (ordem decrescente). O valor padrão é xlAscending.
- ▶ **Key3:** É um parâmetro opcional e é do tipo Variant. Define o terceiro campo, com base no qual a faixa será classificada. Pode ser um nome associado a coluna ou um objeto do tipo range, como por exemplo “ColunaPaís” ou Celss(1,1). Se você omitir este argumento, a faixa será classificado apenas pela campo definido em Key1. Não utilize este argumento quando você estiver classificando uma Tabela Dinâmica.
- ▶ **Order3:** É um parâmetro opcional e do tipo Variant. Define a ordem de classificação para Key3. Pode assumir um dos seguintes valores: xlAscending (ordem crescente) or xlDescending (ordem decrescente). O valor padrão é xlAscending..
- ▶ **Header:** É um parâmetro Opcional e do tipo Variant. Este parâmetro define se a primeira linha da faixa contém ou não os títulos das colunas. Pode assumir um dos seguintes valores: xlGuess, xlNo, or xlYes. Use xlYes se a primeira linha contém os títulos das colunas. Use xlNo se a primeira linha não contém os títulos das colunas. Use xlGuess para deixar que o próprio Excel determina se a primeira linha contém ou não os títulos das colunas. O valor padrão é xlNo. Não use este argumento com Tabelas Dinâmicas.
- ▶ **OrderCustom:** É um parâmetro opcional e do tipo Variant.. Não existe nada que seja “entendível”, a respeito deste parâmetro na documentação oficial do Excel. Nunca utilizei-o e não tem feito falta. Minha recomendação é de que não utilize este parâmetro, até porque não sei para que ele serve.
- ▶ **MatchCase:** É um parâmetro opcional e do tipo Variant. Pode assumir os valores Verdadeiro (diferenciar maiúsculas de minúsculas) ou Falso (não diferenciar). Não utilize este parâmetro com Tabelas Dinâmicas.
- ▶ **Orientation:** É um parâmetro opcional e do tipo Variant. Pode assumir o valor xlSortRows, para que a ordenação seja feita de cima para baixo, por linhas. Pode também assumir o valor xlSortColumns, para que a ordenação seja feita da esquerda para a direita, por colunas.

► **SortMethod:** É um parâmetro opcional e do tipo Variant. Define o tipo de ordenação que será utilizado. Pode assumir um dos seguintes valores: xlPinYin ou xlStroke.

Importante: As definições para os parâmetros Header, Order1, Order2, Order3, OrderCustom e Orientation são salvos cada vez que o método Sort for usado. Na próxima vez que o método Sort for utilizado, se não for definido o valor de um destes argumentos, será utilizado o mesmo valor usado na chamada anterior do método. Para evitar problemas e inconsistências, você deve definir estes parâmetros, explicitamente, cada vez que o método for chamado.

O Método Replace:

Este método é utilizado para encontrar e substituir caracteres em células dentro do intervalo especificado. O uso desse método não altera a seleção nem a célula ativa.

Antes de vermos a sintaxe detalhada deste método, vamos a um exemplo prático, o qual nos ajudará a entender melhor a sintaxe deste método.

Exemplo: Este método substitui todas as ocorrências da função SIN trigonométrico pela função COS. O intervalo de substituição é a coluna A da planilha Plan1:

```
Worksheets("Plan1").Columns("A").Replace _  
    What:="SIN", Replacement:="COS", _  
    SearchOrder:=xlByColumns, MatchCase:=True
```

Sintaxe: A seguir apresento a sintaxe para o método Replace:

```
Range.Replace(What, ReplacementM, LookAt, SearchOrder, MatchCase, MatchByte)
```

- **What:** É do tipo String e é obrigatório. É a cadeia de caracteres que você quer que o Microsoft Excel procure.
- **Replacement:** É do tipo String e é obrigatório. A cadeia de caracteres de substituição.
- **LookAt:** É opcional e do tipo Variant. Pode assumir um dos seguintes valores: xlWhole ou xlPart. xlWhole faz que deva existir uma coincidência exata e xlPart uma coincidência de parte do conteúdo da célula.
- **SearchOrder:** É opcional e do tipo Variant. Pode assumir um dos seguintes valores: xlByRows ou xlByColumns.
- **MatchCase:** É um parâmetro opcional e do tipo Variant. Pode assumir os valores Verdadeiro (diferenciar maiúsculas de minúsculas) ou Falso (não diferenciar). Não utilize este parâmetro com Tabelas Dinâmicas.

Importante: As definições de LookAt, SearchOrder, MatchCase e MatchByte são salvas toda vez que você usa esse método. Se você não especificar valores para esses argumentos na próxima vez que chamar o método, os valores salvos serão usados. A definição desses argumentos altera as definições da caixa de diálogo Localizar, e a alteração das definições na caixa de diálogo Localizar altera os valores salvos que são usados quando você omite os argumentos. Para evitar problemas, defina explicitamente esses argumentos toda vez que você usar esse método.

O método de substituição sempre retorna True.

Lição 14: Objeto Range – Principais Métodos e Propriedades – Parte 9

Nesta lição continuaremos o estudo dos principais métodos, propriedades e coleções do objeto Range.

O Método TextToColumn:

Analisa uma coluna de células contendo texto, dividindo-o em várias colunas. Este método é bastante útil em situações onde você importou dados de um arquivo de texto, onde os dados no arquivo de texto estavam fora do padrão, impedindo que as colunas fossem importadas individualmente. Neste caso, após a importação, você usa o método TextToColumn, para separar o texto que está em uma única coluna, em várias colunas. Antes de aprendermos a sintaxe completa deste método, vamos ver um exemplo prático.

Exemplo: O exemplo a seguir converte o conteúdo da Área de transferência, que contém uma tabela de texto delimitado por espaços, em colunas separadas, na planilha ExText.

Worksheets("ExText").Activate

ActiveSheet.Paste

Selection.TextToColumns DataType:=xlDelimited, ConsecutiveDelimiter:=True, Space:=True

A título de exemplos vamos considerar o texto indicado na figura a seguir, onde temos um arquivo de texto, no formato de Texto Delimitado (para maiores detalhes sobre os formatos de arquivos de texto – delimitado e tamanho fixo, consulte o Módulo 1 do curso de Excel Avançado e VBA).



Autor: Júlio Cesar Fabris Battisti

Site: www.juliobattisti.com.br

Confira também o livro: "Windows Server 2003 – Curso Completo, 1568 páginas"

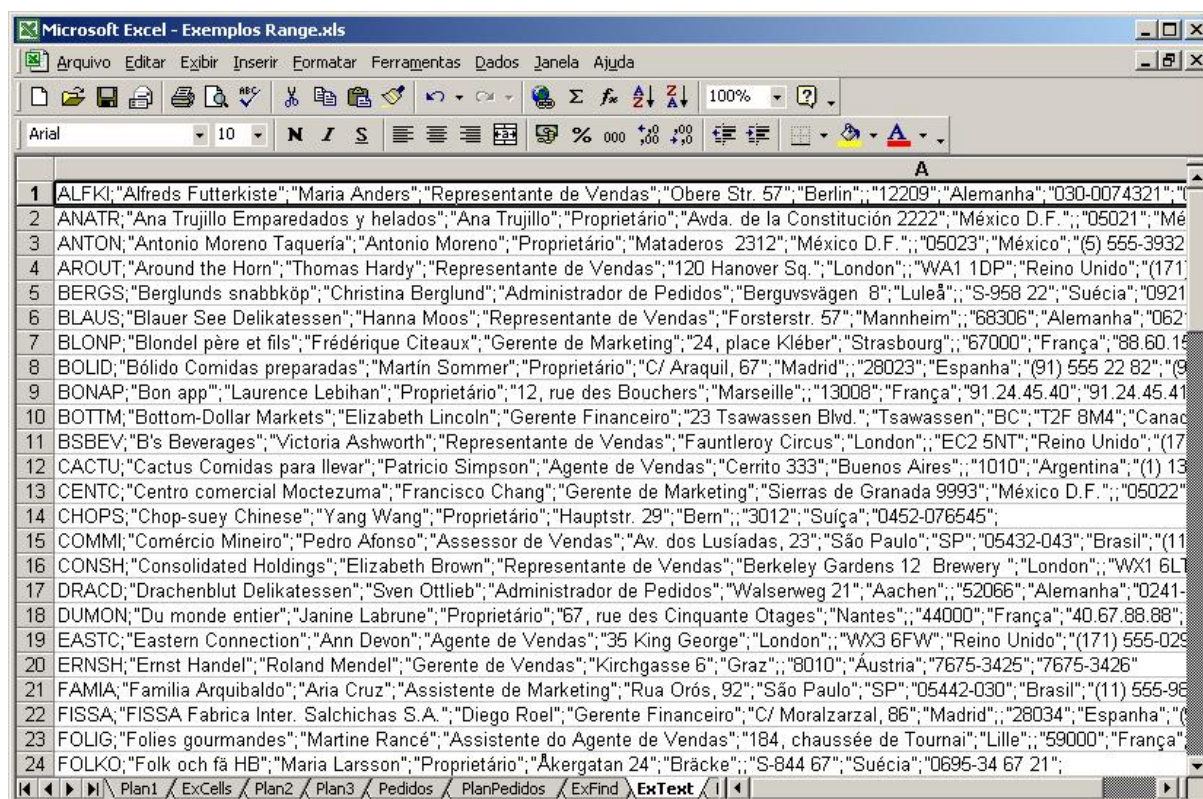
Página 257 de 527

Selecionei todo o texto e utilizei o comando Copiar e Colar, para colocá-lo na Área de Transferência do Windows. Em seguida executei os comandos do nosso exemplo.

O primeiro comando apenas torna a planilha ExText a planilha ativa. O segundo comando cola os dados da área de transferência na planilha ExText.

Worksheets("ExText").Activate
ActiveSheet.Paste

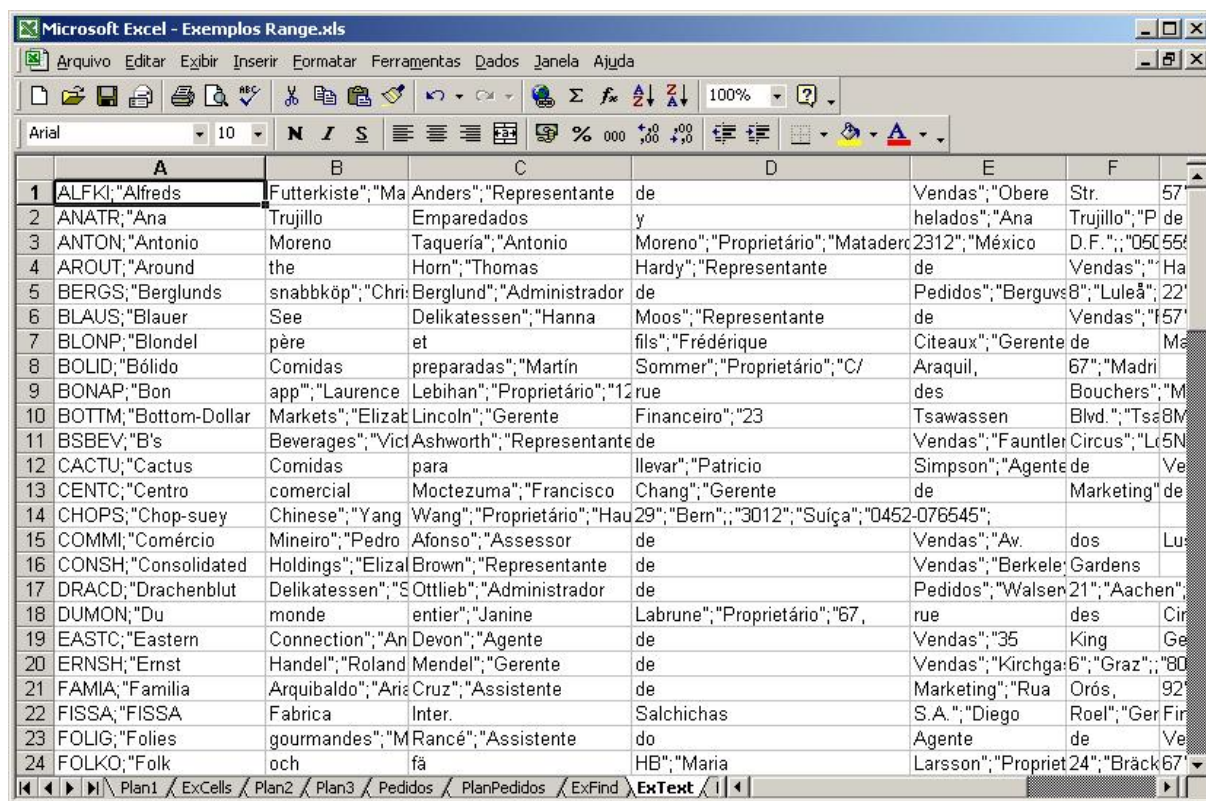
Após a execução destes dois comandos, os dados foram colados, todos na coluna A, sendo cada registro colado em uma célula da coluna A, conforme indicado na Figura a seguir:



O próximo comando, o qual utiliza o método TextToColumn é que efetivamente faz o trabalho, ou seja, separa o texto que está todo junto (cada linha foi colada diretamente em uma célula da coluna A. Agora precisamos separar cada campo em uma coluna):

Selection.TextToColumns DataType:=xlDelimited, ConsecutiveDelimiter:=True, Space:=True

Após a execução deste comando, obtemos o resultado indicado na Figura a seguir:



	A	B	C	D	E	F
1	ALFKI,"Alfreds	Futterkiste", "Ma	Anders", "Representante	de	Vendas", "Obere	Str. 57
2	ANATR,"Ana	Trujillo	Emparedados	y	helados", "Ana	Trujillo", "P de
3	ANTON,"Antonio	Moreno	Taquería", "Antonio	Moreno", "Proprietário", "Matadero	2312", "México	D.F.", "050558
4	AROUT,"Around	the	Horn", "Thomas	Hardy", "Representante	de	Vendas", "Ha
5	BERGS,"Berglunds	snabbköp", "Chri	Berglund", "Administrador	de	Pedidos", "Berguvs	8", "Luleå", "22
6	BLAUS,"Blauer	See	Delikatessen", "Hanna	Moos", "Representante	de	Vendas", "f57
7	BLONP,"Blondel	père	et	fil", "Frédérique	Citeaux", "Gerente de	Ma
8	BOLID,"Bóldo	Comidas	preparadas", "Martín	Sommer", "Proprietário", "C/	Araquil,	67", "Madri
9	BONAP,"Bon	app", "Laurence	Lebihan", "Proprietário", "12	rue	des	Bouchers", "M
10	BOTTM,"Bottom-Dollar	Markets", "Elizab	Lincoln", "Gerente	Financeiro", "23	Tsawassen	Blvd.", "Tse8M
11	BSBEV,"B's	Beverages", "Vict	Ashworth", "Representante	de	Vendas", "Fauntler	Circus", "Lr5N
12	CACTU,"Cactus	Comidas	para	llevar", "Patricio	Simpson", "Agente de	Ve
13	CENTC,"Centro	comercial	Moctezuma", "Francisco	Chang", "Gerente	de	Marketing" de
14	CHOPS,"Chop-suey	Chinese", "Yang	Wang", "Proprietário", "Hau	29", "Bern", "3012", "Suíça", "0452-076545",		
15	COMMI,"Comércio	Mineiro", "Pedro	Afonso", "Assessor	de	Vendas", "Av. dos	Lu
16	CONSH,"Consolidated	Holdings", "Elizab	Brown", "Representante	de	Vendas", "Berkele	Gardens
17	DRACD,"Drachenblut	Delikatessen", "S	Ottlieb", "Administrador	de	Pedidos", "Walsen	21", "Aachen",
18	DUMON,"Du	monde	entier", "Janine	Labrun", "Proprietário", "57,	rue	des
19	EASTC,"Eastern	Connection", "An	Devon", "Agente	de	Vendas", "35	King
20	ERNSH,"Ernst	Handel", "Roland	Mendel", "Gerente	de	Vendas", "Kirchga	6", "Graz", "80
21	FAMIA,"Familia	Arquibaldo", "Aric	Cruz", "Assistente	de	Marketing", "Rua	Orós, 92
22	FISSA,"FISSA	Fabrica	Inter.	Salchichas	S.A.", "Diego	Roel", "Ger Fir
23	FOLIG,"Folies	gourmandes", "M	Rancé", "Assistente	do	Agente	de
24	FOLKO,"Folk	och	fä	HB", "Maria	Larsson", "Propriet	24", "Bräck67

Muito bem, agora vamos entender a sintaxe detalhada deste método.

Sintaxe:

```
Range.TextToColumns(Destination, DataType, TextQualifier,  
ConsecutiveDelimiter, Tab, Semicolon, Comma, Space, Other, OtherChar,  
FieldInfo, DecimalSeparator, ThousandsSeparator)
```

- **Destination:** Variant opcional. Um objeto Range que especifique onde o Microsoft Excel irá colocar os resultados. Se o intervalo for maior que uma única célula, a célula superior esquerda será usada.
- **DataType:** Variant opcional. O formato do texto a ser dividido em colunas. Pode ser uma das seguintes constantes: xlDelimited ou xlFixedWidth. O valor padrão é xlDelimited.
- **TextQualifier:** Variant opcional. O qualificador de texto. Pode ser uma das seguintes constantes: xlTextQualifierDoubleQuote, xlTextQualifierSingleQuote ou xlTextQualifierNone. O valor padrão é xlTextQualifierDoubleQuote.
- **ConsecutiveDelimiter:** Variant opcional. True para que o Microsoft Excel considere delimitadores consecutivos como um só delimitador. O valor padrão é False.
- **Tab:** Variant opcional. True para que DataType seja xlDelimited e para que o caractere tab seja um delimitador. O valor padrão é False.

- ▶ **Semicolon:** Variant opcional. True para que DataType seja xlDelimited e para que o ponto-e-vírgula seja um delimitador. O valor padrão é False.
- ▶ **Comma:** Variant opcional. True para que DataType seja xlDelimited e para que a vírgula seja um delimitador. O valor padrão é False.
- ▶ **Space:** Variant opcional. True para que DataType seja xlDelimited e para que o caractere espaço seja um delimitador. O valor padrão é False.
- ▶ **Other:** Variant opcional. True para que DataType seja xlDelimited e para que o caractere especificado pelo argumento OtherChar seja um delimitador. O valor padrão é False.
- ▶ **OtherChar:** Variant opcional. (necessário quando Other é True). O caractere delimitador quando Other é True. Se mais de um caractere for especificado, somente o primeiro caractere da sequência será usado; os caracteres restantes serão ignorados.
- ▶ **FieldInfo:** Variant opcional. Uma matriz contendo informações de análise para colunas de dados individuais. A interpretação depende do valor de DataType.

Quando os dados estão delimitados, o argumento é uma matriz de matrizes de dois elementos, com cada matriz de dois elementos especificando as opções de conversão para uma determinada coluna. O primeiro elemento é o número da coluna (baseado em 1), e o segundo elemento é uma das constantes xlColumnDataType listadas na tabela a seguir, especificando como a coluna é analisada.

Constante	Descrição
xlGeneralFormat	Geral
xlTextFormat	Texto
xlMDYFormat	Data MDA
xlDMYFormat	Data DMA
xlYMDFormat	Data AMD
xlMYDFormat	Data MAD
xlDYMFormat	Data DAM
xlYDMFormat	Data ADM
xlEMDFormat	Data EMD
xlSkipColumn	Ignora coluna

Nota: Você pode usar xlEMDFormat somente se o suporte ao idioma de Formosa estiver instalado e selecionado. A constante xlEMDFormat especifica que as datas no idioma de Formosa estão sendo usadas.

Os especificadores de coluna podem estar em qualquer ordem. Se um dado especificador de coluna não estiver presente para uma determinada coluna dos dados de entrada, a coluna será analisada com a configuração Geral. Este exemplo faz a terceira coluna ser ignorada, a primeira coluna ser analisada como texto, e as colunas restantes dos dados de origem serem analisadas com a configuração Geral.

```
Array(Array(3, 9), Array(1, 2))
```

Quando os dados de origem têm colunas de largura fixa, o primeiro elemento de cada matriz de dois elementos especifica a posição do caractere inicial na coluna (como um inteiro; 0 (zero) correspondendo ao primeiro caractere). O segundo elemento da matriz de dois elementos especifica a opção de análise para a coluna, como um número de 1 a 9, como listado acima.

O exemplo seguinte analisa duas colunas de um arquivo de largura fixa, com a primeira coluna começando no início da linha e se estendendo por 10 caracteres. A segunda coluna começa na posição 15 e vai até o fim da linha. Para evitar incluir os caracteres entre a posição 10 e a posição 15, o Microsoft Excel adiciona uma entrada de coluna ignorada.

```
Array(Array(0, 1), Array(10, 9), Array(15, 1))
```

Em resumo, o parâmetro FieldInfo é utilizado para detalhar o layout dos dados.

- **DecimalSeparator:** String opcional. O separador decimal que o Microsoft Excel usa quando reconhece números. A configuração padrão é aquela do Windows, definida nas Opções Regionais do Painel de Controle.
- **ThousandsSeparator:** String opcional. O separador de milhares que o Excel usa quando reconhece números. A configuração padrão é aquela do Windows, definida nas Opções Regionais do Painel de Controle.

A tabela a seguir mostra os resultados da importação de texto no Excel para várias configurações de importação. Os resultados numéricos são exibidos na coluna da extrema direita.

Separador decimal do sistema	Separador de milhares do sistema	Valor do separador decimal	Valor do separador de milhares	Texto original	Valor da célula (tipo de dados)
Ponto	Vírgula	Vírgula	Ponto	123.123,45	123,123.45 (numérico)
Ponto	Vírgula	Vírgula	Vírgula	123.123,45	123.123,45 (texto)
Vírgula	Ponto	Vírgula	Ponto	123,123.45	123,123.45 (numérico)
Ponto	Vírgula	Ponto	Vírgula	123 123.45	123 123.45 (texto)
Ponto	Vírgula	Ponto	Espaço	123 123.45	123,123.45 (numérico)

Lição 15: Objeto Range – Principais Métodos e Propriedades – Parte 10

Nesta lição continuaremos o estudo dos principais métodos, propriedades e coleções do objeto Range.

A Propriedade Value:

O significado da propriedade Value depende do objeto ao qual é aplicada, conforme mostrado na tabela a seguir:

Objeto	Valor
Application	Sempre retorna "Microsoft Excel". Somente leitura.
Borders	Sinônimo de Borders.LineStyle.
Name	Uma seqüência de caracteres contendo a fórmula que o nome é definido para consultar. A seqüência está em notação de estilo A1 no idioma da macro, e começa com um sinal de igual. Somente leitura.
Parameter	O valor do parâmetro.
PivotField	O nome do campo especificado no relatório de tabela dinâmica.
PivotItem	O nome do item especificado no campo da tabela dinâmica.
PivotTable	O nome do relatório de tabela dinâmica.
Range	O valor da célula especificada. Se a célula estiver vazia, Value retornará o valor Empty (use a função IsEmpty para testar este caso). Se o objeto Range contiver mais de uma célula, ele retornará uma matriz de valores (use a função IsArray para testar esse caso).
Style	O nome do estilo especificado.
Validation	True se todos os critérios de validação forem atingidos (ou seja, se o intervalo contiver dados válidos).

Vamos a alguns exemplos de uso da propriedade Value.

Exemplo 01: Este exemplo define o valor da célula A1 da planilha Plan1 como 3,14159.

```
Worksheets("Plan").Range("A1").Value = 3.14159
```

Importante: Novamente é importante salientar que, no código VBA, deve ser utilizado o ponto como separador decimal e não a vírgula, independentemente das configurações definidas no ícone Opções Regionais do Painel de controle.

Exemplo 02: Este exemplo faz um loop pelas células se A1:D10 da planilha Plan1. Se uma das células possuir um valor menor do que 0.001, o código substituirá este valor por 0 (zero).


```
For Each c in Worksheets("Plan1").Range("A1:D10")
    If c.Value < 0.001 Then
        c.Value = 0
    End If
Next c
```

O Método RowDifferences:

Este método retorna um objeto Range representando todas as células cujo conteúdo seja diferente do conteúdo da célula de comparação em cada linha.

Sintaxe:

Range.RowDifferences(Comparison)

► **Comparison:** Variant obrigatória. O endereço de uma única célula a ser comparada com o intervalo especificado.

Exemplo: Este exemplo seleciona as células da primeira linha da planilha Plan1, cujo conteúdo seja diferente do conteúdo da célula D1.

```
Worksheets("Plan1").Activate
Set c1 = ActiveSheet.Rows(1).RowDifferences(comparison:=ActiveSheet.Range("D1"))
c1.Select
```

O Método ColumnDifferences:

Este método retorna um objeto Range representando todas as células cujos conteúdos são diferentes da célula de comparação em cada coluna.

Sintaxe:

Range.ColumnDifferences(Comparison)

► **Comparison:** Variant obrigatória. Uma única célula a ser comparada com o intervalo especificado.

Exemplo: Este exemplo seleciona as células da coluna A da planilha Plan1, cujos conteúdos são diferentes da célula A4.

```
Worksheets("Sheet1").Activate
Set r1 = ActiveSheet.Columns("A").ColumnDifferences(Comparison:=ActiveSheet.Range("A4"))
r1.Select
```

O Método PrintPreview:

Este método mostra uma visualização do objeto tal como ele aparece quando impresso.

Sintaxe:

Autor: Júlio Cesar Fabris Battisti

Site: www.juliobattisti.com.br

Confira também o livro: "Windows Server 2003 – Curso Completo, 1568 páginas"

Página 263 de 527

expressão.PrintPreview

O exemplo a seguir, exibe a planilha Plan1 em modo de Visualização de Impressão:

```
Worksheets("Plan1").PrintPreview
```

O Método PrintOut:

Este método é utilizado para imprimir o objeto no qual o método foi chamado (Range, Worksheet, etc.).

Sintaxe:

Range.PrintOut(From, To, Copies, Preview, ActivePrinter, PrintToFile, Collate, PrToFileName)

- ▶ **From:** Variant opcional. O número da página pela qual começar a impressão. Se esse argumento for omitido, a impressão começará pelo início.
- ▶ **To:** Variant opcional. O número da última página a imprimir. Se esse argumento for omitido, a impressão terminará na última página.
- ▶ **Copies:** Variant opcional. O número de cópias a imprimir. Se esse argumento for omitido, será impressa uma cópia.
- ▶ **Preview:** Variant opcional. True para que o Microsoft Excel invoque a visualização de impressão antes de imprimir o objeto. False (ou omitido) para imprimir o objeto imediatamente.
- ▶ **ActivePrinter:** Variant opcional. Define o nome da impressora ativa.
- ▶ **PrintToFile:** Variant opcional. True para imprimir para um arquivo. Se PrToFileName não for especificado, o Microsoft Excel solicitará ao usuário que digite o nome do arquivo de saída.
- ▶ **Collate:** Variant opcional. True para agrupar múltiplas cópias.
- ▶ **PrToFileName:** Variant opcional. Se PrintToFile for definido como True, esse argumento especificará o nome do arquivo para o qual você deseja imprimir.

"Páginas" nas descrições de From e To se referem a páginas impressas — e não a páginas gerais da planilha ou pasta de trabalho.

Este exemplo imprime a planilha ativa.

```
ActiveSheet.PrintOut
```

Lição 16: Exemplos Práticos de Uso dos Objetos do Excel – Parte 1

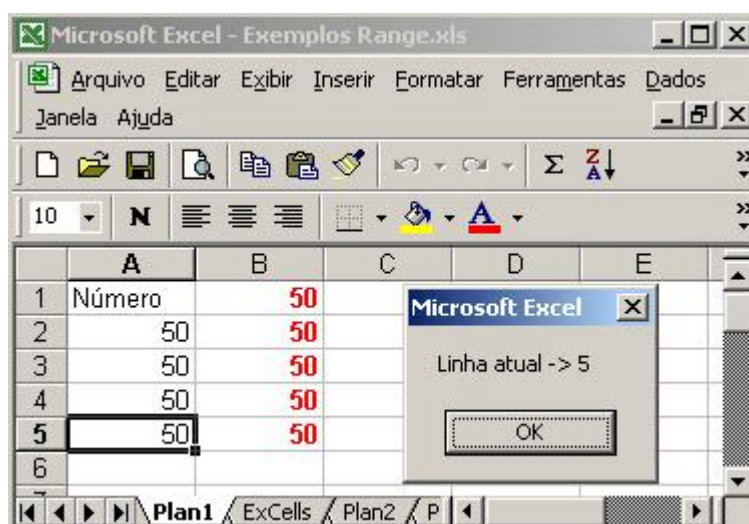
A partir desta lição, até o final do módulo, apresentarei uma série de exemplos práticos, de utilização do objeto Range. Nos exemplos mostro como usar o objeto Range no VBA, para a resolução de uma série de problemas práticos, do dia-a-dia.

Exemplos práticos na utilização do objeto Range – Exemplos com Linhas:

Exemplo 01: O comando a seguir retorna o número da linha onde está o cursor do Excel, ou seja, o número da linha da célula ativa:

```
R = ActiveCell.Row  
MsgBox "Linha atual -> " & R
```

Ao executar este exemplo será retornada a mensagem indicada na Figura a seguir:



Neste exemplo utilizei a propriedade `ActiveCell`, a qual retorna um objeto Range representando a célula ativa da janela ativa (a janela visível) ou da janela especificada. Se a janela não estiver exibindo uma planilha, essa propriedade falhará. Somente leitura.

Nota: Quando você não especifica um qualificador de objeto, essa propriedade retorna a célula ativa da janela ativa.

Importante: Tenha cuidado de distinguir entre célula ativa e seleção. A célula ativa é uma única célula dentro da seleção atual. A seleção pode conter mais de uma célula, mas somente uma é a célula ativa.

Todas as expressões seguintes retornam a célula ativa, sendo todas equivalentes:

```
ActiveCell  
Application.ActiveCell  
ActiveWindow.ActiveCell  
Application.ActiveWindow.ActiveCell
```

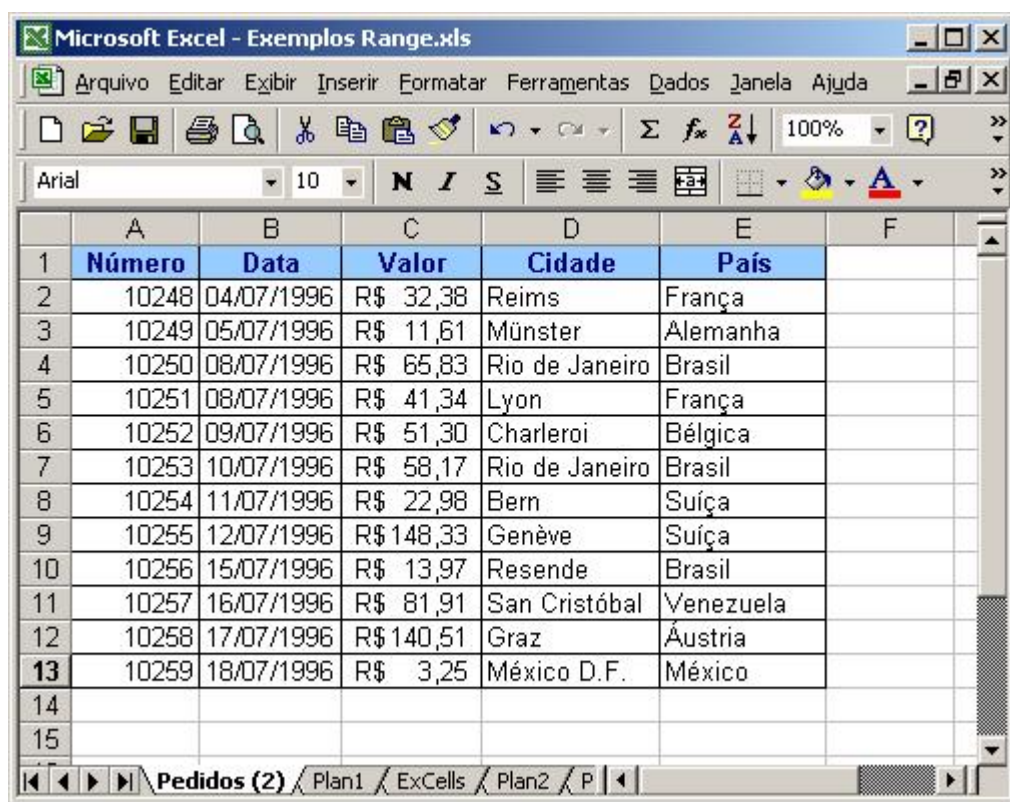
Exemplo 02: Como faço para testar se uma linha está selecionada (a linha toda e não apenas uma faixa de células dentro da linha)?

A seguir mostro um exemplo de como testar para ver se uma linha toda está selecionada:

```
If Selection.Address = Selection.EntireRow.Address Then
    ' Comando 1
    ' Comando 2
    ...
    ' Comando n
End If
```

Neste exemplo utilizei a propriedade Selection, a qual retorna o objeto selecionado, na janela ativa. Também utilizei a propriedade Address, do objeto retornado e comparei este endereço, com o endereço retornado pela propriedade EntireRow, a qual retorna um objeto Range representando a linha (ou linhas) inteira que contém o intervalo especificado. Somente leitura. Se os dois endereços forem iguais, isso significa que a linha inteira está selecionada e o teste é verdadeiro.

Exemplo 03: Neste exemplo vou mostrar como selecionar todas as linhas da planilha, isto é, todas as linhas que tenham dados. Neste exemplo, vou mostrar como fazer com o código VBA, o mesmo que faz o comando Ctrl+Shift+*. Considere a planilha da Figura a seguir:



	A	B	C	D	E	F
1	Número	Data	Valor	Cidade	País	
2	10248	04/07/1996	R\$ 32,38	Reims	França	
3	10249	05/07/1996	R\$ 11,61	Münster	Alemanha	
4	10250	08/07/1996	R\$ 65,83	Rio de Janeiro	Brasil	
5	10251	08/07/1996	R\$ 41,34	Lyon	França	
6	10252	09/07/1996	R\$ 51,30	Charleroi	Bélgica	
7	10253	10/07/1996	R\$ 58,17	Rio de Janeiro	Brasil	
8	10254	11/07/1996	R\$ 22,98	Bern	Suíça	
9	10255	12/07/1996	R\$ 148,33	Genève	Suíça	
10	10256	15/07/1996	R\$ 13,97	Resende	Brasil	
11	10257	16/07/1996	R\$ 81,91	San Cristóbal	Venezuela	
12	10258	17/07/1996	R\$ 140,51	Graz	Áustria	
13	10259	18/07/1996	R\$ 3,25	México D.F.	México	
14						
15						

A faixa de dados desta planilha (para o VBA é a CurrentRegion) é A1:E13. Se você colocar o cursor em qualquer célula desta faixa e pressionar Ctrl+Shift+*, será selecionado somente o intervalo com dados, ou seja A1:E13. Isso pode ser feito com o VBA:

A maneira mais simples: Vamos supor que a faixa de dados inicia na célula A1 (que é o caso do exemplo da figura anterior). O comando a seguir irá selecionar toda a faixa de A1:E13, pois o Excel detecta que esta é a faixa que contém dados:

```
Range("A1").CurrentRegion.Select
```

Você também pode atribuir o resultado da seleção a um objeto Range, conforme exemplificado a seguir:

```
Dim FaixaSel As Range  
Set FaixaSel = Range("A1").CurrentRegion
```

Se a faixa não estiver na planilha atual, você pode fazer referência a faixa onde está a planilha com os dados, conforme exemplo a seguir:

```
Dim FaixaSel As Range  
Set FaixaSel = Workbooks("Pasta1.Xls").Sheets("Plan1").Range("A1").CurrentRegion
```

Ou mais didaticamente, poderíamos separar este exemplo em vários comandos, conforme exemplo a seguir:

```
Dim FaixaSel As Range  
Dim Pasta As Workbook  
dim wks As Worksheet  
Set Pasta = Workbooks("Pasta1.Xls")  
Set wks = Pasta.Sheets("Plan1")  
Set FaixaSel = wks.Range("A1").CurrentRegion
```

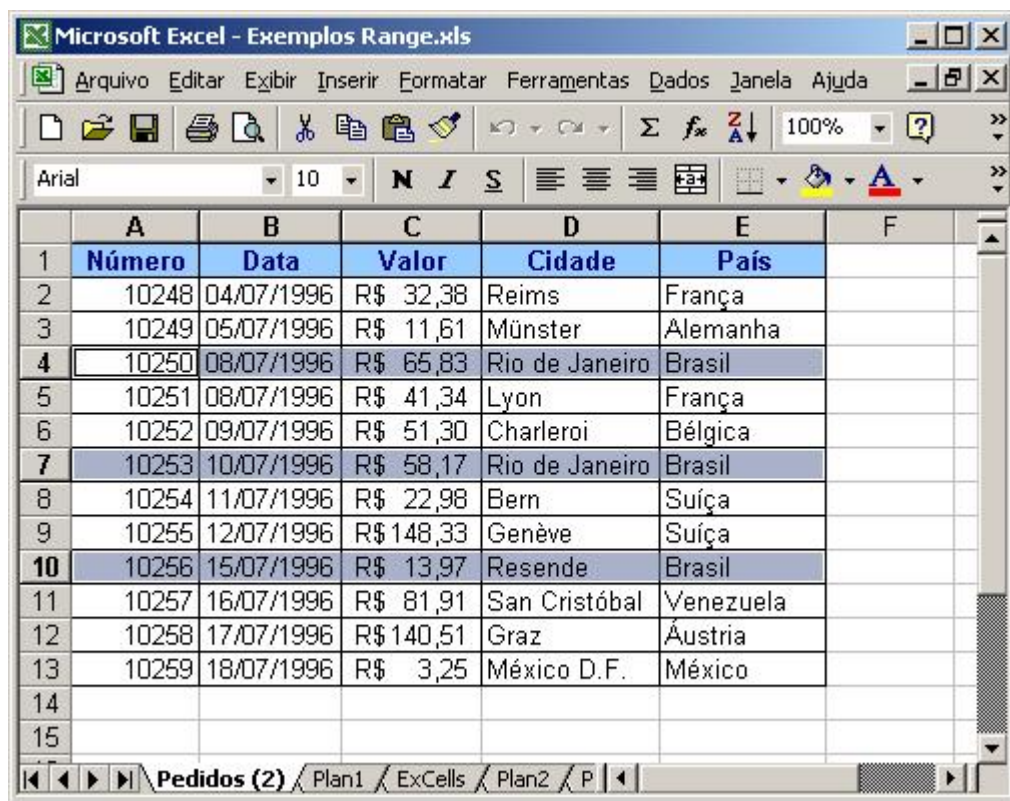
Exemplo 04: Neste exemplo mostrarei como fazer a seleção de uma ou mais linhas em uma planilha, com base em um ou mais valores de pesquisa.

No exemplo a seguir, serão selecionadas as linhas nas quais uma das células da linha contiver um determinado valor. No nosso exemplo, selecionaremos as linhas que contenham “Brasil” em uma de suas células:

```
Range("A1").CurrentRegion.AutoFilter 3, "=Brasil"  
Range("A1").CurrentRegion.Offset(1).Resize(Range("A1").CurrentRegion.Rows.Count-1).SpecialCells(xlVisible).Select  
Range("A1").CurrentRegion.AutoFilter
```

Primeiro vamos entender o funcionamento deste exemplo para depois detalhar o código utilizado. Para este exemplo considere a planilha indicada na Figura anterior, na qual temos dados de A1 até E13.

Após a execução deste exemplo, serão selecionadas as linhas que contiverem Brasil na quinta coluna da faixa, ou seja, na coluna E. O resultado da execução deste código está indicado na figura a seguir, onde foram selecionadas as linhas 4, 7 e 10, as quais tem o valor Brasil na coluna E:



	A	B	C	D	E	F
1	Número	Data	Valor	Cidade	País	
2	10248	04/07/1996	R\$ 32,38	Reims	França	
3	10249	05/07/1996	R\$ 11,61	Münster	Alemanha	
4	10250	08/07/1996	R\$ 65,83	Rio de Janeiro	Brasil	
5	10251	08/07/1996	R\$ 41,34	Lyon	França	
6	10252	09/07/1996	R\$ 51,30	Charleroi	Bélgica	
7	10253	10/07/1996	R\$ 58,17	Rio de Janeiro	Brasil	
8	10254	11/07/1996	R\$ 22,98	Bern	Suíça	
9	10255	12/07/1996	R\$ 148,33	Genève	Suíça	
10	10256	15/07/1996	R\$ 13,97	Resende	Brasil	
11	10257	16/07/1996	R\$ 81,91	San Cristóbal	Venezuela	
12	10258	17/07/1996	R\$ 140,51	Graz	Áustria	
13	10259	18/07/1996	R\$ 3,25	México D.F.	México	
14						
15						

Muito bem, agora é hora de entendermos o código do exemplo. A primeira linha simplesmente define um critério de filtragem – “=Brasil”, para a quinta coluna da faixa. O número 5 passado como primeiro parâmetro para AutoFilter é que garante que o critério seja aplicado a quinta coluna:

```
Range("A1").CurrentRegion.AutoFilter 5, "=Brasil"
```

A segunda linha é que faz todo o trabalho. Ao aplicar o AutoFiltro, ficam visíveis somente as linhas que atendem ao critério. Mas não é esse o efeito desejado. Queremos que todas as linhas sejam exibidas e que somente sejam selecionadas as linhas que atendem ao critério. Neste caso uso a propriedade Offset com o parâmetro 1, para obter um objeto Range com um deslocamento de um em relação a CurrentRegion, ou seja, sem a linha 1 de títulos das colunas. Em seguida chamo o método Resize, para redimensionar este objeto Range, para o número de linhas – 1: `Range("A1").CurrentRegion.Rows.Count-1`. Por último utilizo o método SpecialCells para selecionar somente as células visíveis desta faixa. Observe que o trabalho foi definir uma faixa como sendo toda a faixa original, menos a linha de título e selecionar as células visíveis. O AutoFiltro apeans oculta as linhas que não atendem aos critérios especificados.

```
Range("A1").CurrentRegion.Offset(1).Resize(Range("A1").CurrentRegion.Rows.Count-1).SpecialCells(xlVisible).Select
```

O passo final é chamar AutoFilter novamente, sem nenhum parâmetro, para desativar o Auto Filtro. O resultado final é que as linhas selecionadas no passo dois permanecerão selecionadas e o Auto Filtro desativado. Ou seja, o resultado final é que apenas as linhas que atendem o critério foram selecionadas, conforme demonstrado na figura anterior.

```
Range("A1").CurrentRegion.AutoFilter
```

Lição 17: Exemplos Práticos de Uso dos Objetos do Excel – Parte 2

Nesta lição vamos ver um exemplo de uma macro que solicita ao usuário que seja digitado um valor. Em seguida, a macro irá procurar em todas as planilhas de todas as pastas de trabalho abertas, por células que contém o valor digitado e irá colocar estas células em destaque, alterando a cor de fundo destas células. Inicialmente vamos apresentar o código. Em seguida vamos vê-lo em funcionamento e por último apresentar alguns comentários para esclarecer o funcionamento do código:

```
Sub ProcuraTudo()

    Dim QueProcurar As String
    Dim wb As Workbook
    Dim ws As Worksheet
    Dim r As Range
    Dim rFirst As Range
    Dim bFirstStep As Boolean

    QueProcurar = InputBox("Digite o texto a ser pesquisado:")

    For Each wb In Application.Workbooks
        For Each ws In wb.Sheets
            Set rFirst = ws.Cells.Find(What:=QueProcurar, _
                After:=ws.Cells(1, 1), _
                LookIn:=xlValues, LookAt:= _
                xlPart, SearchOrder:=xlByRows, _
                SearchDirection:=xlNext, MatchCase:=False)

            If Not rFirst Is Nothing Then

                HighLightIt rFirst
                Set r = ws.Cells.FindNext(After:=rFirst)
                While (Not rFirst.Address = r.Address)
                    HighLightIt r
                    Set r = ws.Cells.FindNext(After:=r)
                Wend
            End If
        Next
    Next
End Sub

Sub HighLightIt(r As Range)
    r.Interior.ColorIndex = 6
End Sub
```

A título de exemplo vamos fazer uma pesquisa utilizando Brasil como critério. Em seguida observaremos os resultados obtidos.

Ao executar o procedimento ProcuraTudo, será exibida uma caixa de diálogo, solicitando que você digite o texto a ser pesquisado. Digite Brasil, conforme indicado na Figura a seguir e clique em OK:



A pesquisa é efetuada e todas as células contendo Brasil, serão destacadas com a alteração da cor de fundo, conforme exemplo da Figura a seguir. Este destaque será aplicado a todas as células de todas as planilhas de todas as pastas de trabalho que estiverem abertas:

A screenshot of the Microsoft Excel application window titled "Microsoft Excel - Exemplos Range.xls". The window shows a menu bar (Arquivo, Editar, Exibir, Inserir, Formatar, Ferramentas, Dados, Janela, Ajuda) and a toolbar. Below the toolbar is a status bar showing "Arial", "10", and "N". The main area displays a table with 5 columns: "Número", "Data", "Valor", "Cidade", and "País". The table has 13 rows of data. The cells in the "País" column containing "Brasil" are highlighted in yellow. The table is currently on the "Pedidos (2)" worksheet.

	A	B	C	D	E
1	Número	Data	Valor	Cidade	País
2	10248	04/07/1996	R\$ 32,38	Reims	França
3	10249	05/07/1996	R\$ 11,61	Münster	Alemanha
4	10250	08/07/1996	R\$ 65,83	Rio de Janeiro	Brasil
5	10251	08/07/1996	R\$ 41,34	Lyon	França
6	10252	09/07/1996	R\$ 51,30	Charleroi	Bélgica
7	10253	10/07/1996	R\$ 58,17	Rio de Janeiro	Brasil
8	10254	11/07/1996	R\$ 22,98	Bern	Suíça
9	10255	12/07/1996	R\$ 148,33	Genève	Suíça
10	10256	15/07/1996	R\$ 13,97	Resende	Brasil
11	10257	16/07/1996	R\$ 81,91	San Cristóbal	Venezuela
12	10258	17/07/1996	R\$ 140,51	Graz	Áustria
13	10259	18/07/1996	R\$ 3,25	México D.F.	México

Comentários sobre o código do exemplo: O código é relativamente simples e utiliza os métodos e propriedades vistos nas lições deste módulo. A parte inicial do código é dedicado a declaração de variáveis, com destaque para a variável QueProcurar, a qual irá receber o valor de pesquisa, digitado pelo usuário.

Logo após a declaração das variáveis, utilizo a função InputBox para exibir uma caixa de diálogo para que o usuário digite o valor a ser pesquisado. O valor digitado pelo usuário é atribuído a variável QueProcurar:

```
QueProcurar = InputBox("Digite o texto a ser pesquisado:")
```

Em seguida crio dois laços, um dentro do outro. O laço mais de fora, percorre a coleção de Workbooks, para acessar todas as pastas de trabalho abertas. Para cada pasta de trabalho, o laço interno, percorre todas as suas planilhas – coleção Sheets. Dentro do laço interno, inicialmente, uso o método Find (visto neste módulo), para pesquisar a planilha atual. Se for encontrado um determinado valor (O teste Not rFirst Is Nothing for Verdadeiro), chamo o procedimento HighLightIt para destacar a célula encontrada e uso o método FindNext (visto neste capítulo), para repetir a pesquisa, com os mesmos critérios definidos no método Find, ou seja, localizar a próxima Célula que contenha o valor de pesquisa. Em seguida utilizo um laço While para percorrer todas as células da planilha atual, localizando novas células que atendam o critério de pesquisa (usando o método FindNext) e destacando a célula. Estas funções são executadas pelo trecho de código a seguir, o qual contém toda a lógica deste exemplo:

```
For Each wb In Application.Workbooks
    For Each ws In wb.Sheets
        Set rFirst = ws.Cells.Find(What:=QueProcurar, _
            After:=ws.Cells(1, 1), _
            LookIn:=xlValues, LookAt:= _
            xlPart, SearchOrder:=xlByRows, _
            SearchDirection:=xlNext, MatchCase:=False)

        If Not rFirst Is Nothing Then

            HighLightIt rFirst
            Set r = ws.Cells.FindNext(After:=rFirst)
            While (Not rFirst.Address = r.Address)
                HighLightIt r
                Set r = ws.Cells.FindNext(After:=r)
            Wend
        End If
    Next
Next
End Sub
```

Por último foi criado um procedimento separado, o qual altera a cor da célula passada como parâmetro.

```
Sub HighLightIt(r As Range)
    r.Interior.ColorIndex = 6
End Sub
```

Você pode, facilmente, adaptar este exemplo para realizar outras operações em todas as planilhas de todas as pastas de trabalho abertas. A lógica para percorrer todas as planilhas é a mesma, basta que você altera a ação a ser executada e/ou os critérios de pesquisa.

Lição 18: Exemplos Práticos de Uso dos Objetos do Excel – Parte 3

Nesta lição vamos continuar com mais alguns exemplos de uso do objeto Range e de técnicas relacionadas a operações em linhas de dados na planilha.

Exemplo 01: Neste exemplo mostrarei como fazer a exclusão de linhas, baseado em um ou mais critérios. Esta técnica também é conhecida como Exclusão Condicional.

Importante: Sempre que for feita a exclusão de linhas, usando código VBA, é importante que você faça um laço para percorrer todas as linhas, porém no sentido contrário ao tradicional, ou seja, de baixo para cima. Esta técnica evita que linhas que atendam ao critério de busca sejam descartadas. Por exemplo, vamos supor que você está percorrendo um laço no sentido normal, ou seja, da primeira para a última linha. Agora imagine que a linha 2 atende ao critério selecionado. Neste caso, a linha dois será excluída e a linha 3 passará a ser a linha 2, a linha 4 passará a ser a linha 4 e assim por diante. Porém imagine que a antiga linha 3, agora linha 2 depois da exclusão, também atenda ao critério. Neste caso esta linha (antiga linha 3, atual linha 2, não será excluída), pois a pesquisa continuará na linha 3 (antiga linha 4). Isso faz com que uma ou mais linhas que atendam aos critérios, não sejam excluídas. Fazendo o laço no sentido contrário, este problema é contornado.

Neste primeiro exemplo, vamos percorrer a faixa de E2:E50. Se o valor na célula for zero, a linha respectiva será excluída. Observe que o laço é percorrido de trás para frente, ou seja, da última para a primeira linha, pelos motivos expostos anteriormente.

```
Dim IndexRow As Long
For IndexRow = 50 To 2 Step -1
    If Cells(IndexRow, "E").Value = 0 Then
        Cells(IndexRow, "E").EntireRow.Delete
    End If
Next IndexRow
```

Uso uma variável do tipo Long – IndexRow, para percorrer as linhas no sentido contrário, ou seja, da linha 50 para a linha 2. Passo esse valor como um parâmetro para a propriedade Cells. Outro parâmetro é a letra da coluna – E no nosso exemplo. Faço um teste para verificar se o valor da célula é zero. Se for zero, uso a propriedade EntireRow para retornar um objeto Range que representa a linha da célula e o método Delete, do objeto Range, para excluir essa linha. Simples mas bem funcional.

Outra maneira seria utilizando a propriedade AutoFilter, para fazer uma pesquisa com base no critério desejado. Em seguida temos que usar o método SpecialCells, já descrito na lição anterior, para selecionar e excluir apenas as linhas das células que atendem ao critério especificado, ou seja, as linhas que continuam visíveis após a utilização do método AutoFilter. O código a seguir é um exemplo de uso desta técnica:

```
' Faz com que a execução continue, em caso de erros devido a nenhuma célula atender aos  
' critérios pesquisados
```

```
On Error Resume Next
```

```
Range("E2:E50").AutoFilter Field:=5, Criteria1:="0"
```

```
    If Err = 0 Then
```

```
        Range("E2:E50").SpecialCells(xlCellTypeVisible).EntireRow.Delete
```

```
    End If
```

```
ActiveSheet.AutoFilterMode = False
```

```
ActiveSheet.UsedRange
```

```
' Desabilita o gerenciamento de erro.
```

```
On Error GoTo 0
```

Exemplo 02: A seguir alguns exemplos de código para determinar se uma linha ou coluna está vazia, ou seja, se todas as células da coluna e/ou linha estão em branco.

No exemplo de código a seguir, uma mensagem será exibida se a linha 10 estiver vazia. Para isso utilizo a função CountA, que é a função de planilha Cont.Valores. Esta função determina quantas células, em uma faixa, tem valores. Se o valor for zero é porque todas as células estão vazias.

```
If Application.CountA(Rows(10)) = 0 Then  
    MsgBox "A linha está vazia!!!"  
Else  
    MsgBox "A linha NÃO está vazia!!!"  
End If
```

Uma abordagem um pouco diferente é a utilizada a seguir, onde determinamos se a linha contendo a célula ativa, isto é, a célula onde está o cursor, está vazia.:

```
If Application.CountA(ActiveCell.EntireRow) = 0 Then  
    MsgBox "A linha está vazia!!!"  
Else  
    MsgBox "A linha NÃO está vazia!!!"  
End If
```

Nota: Para determinar se a coluna da célula ativa, está vazia, use EntireColumn, ao invés de Entire Row.

O exemplo a seguir, determina se a coluna C está vazia:

```
If Application.CountA(Columns(3)) = 0 Then  
    MsgBox "A coluna está vazia!!!"  
Else  
    MsgBox "A coluna não está vazia!!!"  
End If
```

Lição 19: Exemplos Práticos de Uso dos Objetos do Excel – Parte 4

Nesta lição apresentarei alguns exemplos de utilização do objeto Range, para a solução de situações práticas envolvendo colunas no Excel.

Exemplo 01: Neste exemplo mostrarei como verificar se a seleção efetuada pelo usuário é composta de uma única coluna ou de uma única linha. Esta técnica é útil quando, como parte da entrada de dados de um programa no Excel, o usuário deve selecionar uma faixa de dados que seja uma única coluna ou uma única linha. Neste caso, se a seleção não for uma única linha ou coluna, você pode emitir uma mensagem, solicitando que o usuário refassa a seleção. Isso evita erros.

Para verificar se a seleção é formada por uma única coluna, podemos usar o trecho de código a seguir:

```
' Verifica o número de colunas. Se for maior do que 1, a macro é encerrada

    If Selection.Columns.Count > 1 Then
        MsgBox "Selecione uma única coluna"
        ' Encerra a macro
        End
    End If

' Código a ser executado se o número de colunas for igual a 1
```

Para verificar se a seleção contém células em mais de uma linha, podemos usar o trecho de código a seguir:

```
Dim Rng As Range
' Obtém o endereço de uma faixa de células, através do use de InputBox

On Error Resume Next

    Set Rng = Application.InputBox( _
        prompt:="Selecione células em uma única linha", _
        Type:=8, default:=Selection.Address)
    On Error GoTo 0

    ' Se nenhuma célula foi selecionada, encerre o procedimento

    If Rng Is Nothing Then Exit Sub

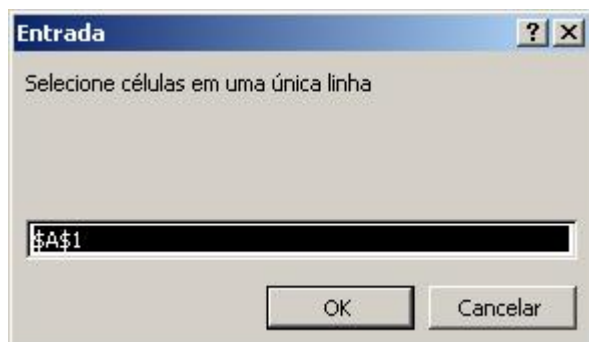
    ' Verifico se foram selecionadas células em mais do que uma linha
    ' Neste caso emito uma mensagem de erro e encerro a Macro.

If Rng.Rows.Count > 1 Then
    MsgBox "Selecione células de uma única linha!!!"

    ' Encerra a Macro
```

```
End  
End If  
  
' Comandos a serem executados quando a seleção for composta de células  
' em uma única linha
```

Ao executar o código deste exemplo, será exibida a janela indicada na Figura a seguir:



Para exibir esta janela, utilizamos o comando a seguir:

```
Set Rng = Application.InputBox(prompt:="Selecione células em uma única linha", _  
                                Type:=8, default:=Selection.Address)
```

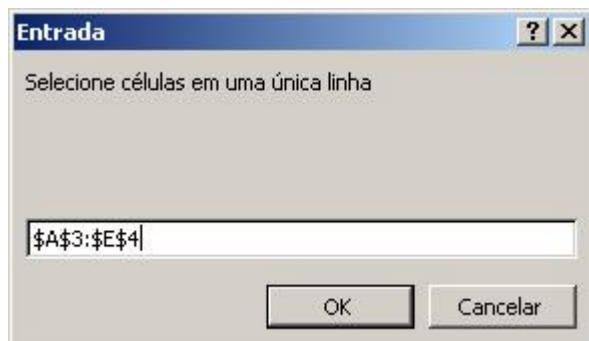
O parâmetro Type é do tipo Variant e é opcional. Ele especifica o tipo de dados retornado. Se esse argumento for omitido, a caixa de diálogo retornará texto. Pode ser um dos valores abaixo ou a soma deles.

Valor	Significado
0	Uma fórmula
1	Um número
2	Texto (uma sequência)
4	Um valor lógico (True ou False)
8	Uma referência a células, como um objeto Range
16	Um valor de erro, como #N/D
64	Uma matriz de valores

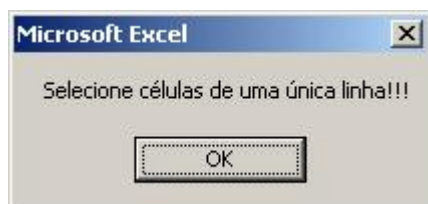
Observe que o valor 8, para o parâmetro Type, indica que a função InputBox irá retornar uma referência de células (o endereço selecionado pelo usuário), como um objeto Range.

O parâmetro default também é opcional. Este parâmetro define o valor que será exibido, por padrão, na caixa de texto do InputBox. Neste exemplo, usei a propriedade Address, do objeto Selection, para exibir o endereço da seleção atual. Como o parâmetro Type foi definido como 8, este InputBox permite que o usuário utilize o mouse para arrastar sobre uma faixa de

células, na planilha. A medida que o usuário arrasta o mouse, o endereço da faixa de células vai sendo preenchido no InputBox, conforme indicado na Figura a seguir:



Neste exemplo, propositadamente, selecionei células em mais de uma linha. Ao clicar em OK, será exibida a mensagem indicada a seguir:



Ao clicar em OK a macro será encerrada. Observe que a rotina verificou que havia células selecionadas em mais de uma linha e, corretamente, emitiu uma mensagem avisando e encerrou a macro. Outro detalhe interessante a reforçar, neste exemplo, é a definição do valor 8, para o parâmetro Type. Este valor transforma o InputBox em uma janela para entrada de uma faixa de endereços, onde o usuário pode utilizar o mouse para arrastar e marcar a faixa de endereços a ser retornada pela função InputBox.

Na próxima lição mostrarei mais alguns exemplos de uso do objeto Range.

Lição 20: Exemplos Práticos de Uso dos Objetos do Excel – Parte 5

Nesta lição apresentarei alguns exemplos de utilização do objeto Range, para a solução de situações práticas envolvendo colunas no Excel.

Exemplo 01: Neste exemplo mostrarei um código que faz a comparação entre os valores de duas colunas, linha a linha. Se o valor da segunda coluna for maior ou igual ao valor da primeira, a célula da segunda coluna será colocada em destaque, através da alteração da sua cor de fundo.

A título de exemplo, utilizaremos a planilha indicada na Figura a seguir:



The screenshot shows a Microsoft Excel window titled "Microsoft Excel - Exemplos Range.xls". The menu bar includes "Arquivo", "Editar", "Exibir", "Inserir", "Formatar", "Ferramentas", "Dados", "Janela", and "Ajuda". The toolbar contains various icons for file operations and editing. The spreadsheet has four columns: A (Mês), B (Vendas - 2003), C (Vendas - 2004), and D. The rows are numbered 1 to 13. The data is as follows:

	A	B	C	D
1	Mês	Vendas - 2003	Vendas - 2004	
2	Janeiro	2500	2800	
3	Fevereiro	3020	2950	
4	Março	3400	3100	
5	Abril	4100	3800	
6	Maio	4500	5200	
7	Junho	5100	5500	
8	Julho	4650	4350	
9	Agosto	5200	5900	
10	Setembro	6200	6500	
11	Outubro	6500	5400	
12	Novembro	5400	5100	
13	Dezembro	6000	6050	

Para a solução do problema proposto será fundamental o uso da propriedade Offset, do objeto Range. Por isso, antes de apresentar uma solução para o problema proposto, vamos entender como funciona a propriedade Offset.

A Propriedade Offset:

Esta propriedade retorna um objeto Range, que representa um intervalo deslocado (um número de linhas e colunas especificadas na chamada da propriedade) do intervalo especificado. Somente leitura.

Sintaxe:

expressão.Offset(RowOffset, ColumnOffset)

- ▶ **Expressão:** Obrigatória. Uma expressão que retorna um objeto Range.
- ▶ **RowOffset:** Variant opcional. O número de linhas (positivo, negativo ou 0 (zero)) com base no qual o intervalo deve ser deslocado. Os valores positivos são deslocados para baixo e os negativos, para cima. O valor padrão é 0.
- ▶ **ColumnOffset:** Variant opcional. O número de colunas (positivo, negativo ou 0 (zero)) com base no qual o intervalo deve ser deslocado. Os valores positivos são deslocados para a direita e os negativos, para a esquerda. O valor padrão é 0.

Exemplos da propriedade Offset (objeto Range):

Este exemplo ativa a célula três colunas à direita (columnOffset:=3) e três linhas abaixo (rowOffset:=3), da célula ativa em Plan1.

```
Worksheets("Plan1").Activate  
ActiveCell.Offset(rowOffset:=3, columnOffset:=3).Activate
```

O exemplo a seguir assume que a planilha Plan1 contém uma tabela que possui uma linha de cabeçalho. O exemplo seleciona a tabela sem selecionar a linha de cabeçalho. A célula ativa deve estar em algum local na tabela antes do exemplo ser executado.

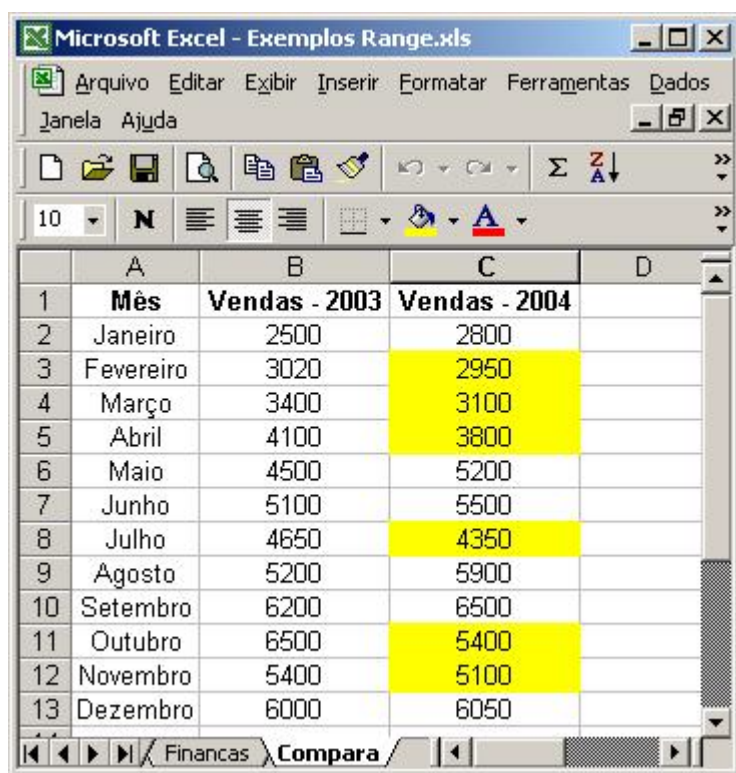
```
Set tbl = ActiveCell.CurrentRegion  
tbl.Offset(1, 0).Resize(tbl.Rows.Count - 1, tbl.Columns.Count).Select
```

Muito bem, agora que já vimos a propriedade Offset, vamos trabalhar na solução do problema proposto.

A solução para o problema proposto: O código a seguir é a solução para o problema proposto:

```
Sub CompareColumns()  
    Dim CurrCell As Range  
  
    ' Faz um loop através das células da faixa a ser comparada  
  
    For Each CurrCell In Range("C2:C13")  
        'Compara os valores; .offset(0,-1) faz referência a coluna B, ou seja  
        ' -1 coluna em relação à Coluna C.  
  
        If CurrCell.Value < CurrCell.Offset(0, -1).Value Then  
            'Altera a cor de fundo para Amarelo  
            CurrCell.Interior.ColorIndex = 6  
        Else  
            'Remove qualquer cor de fundo existente, se o valor não for menor  
            CurrCell.Interior.ColorIndex = xlNone  
        End If  
    Next  
End Sub
```

Na figura a seguir, apresento o resultado da execução deste código:



	A	B	C	D
1	Mês	Vendas - 2003	Vendas - 2004	
2	Janeiro	2500	2800	
3	Fevereiro	3020	2950	
4	Março	3400	3100	
5	Abril	4100	3800	
6	Maio	4500	5200	
7	Junho	5100	5500	
8	Julho	4650	4350	
9	Agosto	5200	5900	
10	Setembro	6200	6500	
11	Outubro	6500	5400	
12	Novembro	5400	5100	
13	Dezembro	6000	6050	

Observe que realmente foi alterada somente a cor de fundo para as células, onde o valor da célula na coluna C é menor do que o valor da mesma linha, na coluna B. Você pode facilmente adaptar este exemplo para usar em suas planilhas. Pode até torná-lo mais genérico, fazendo com que seja exibido um InputBox que solicita o endereço da primeira faixa e o endereço da primeira célula da segunda faixa. O código a seguir faz estas modificações, onde utilizo a função InputBox para solicitar que o usuário entre com os endereços da primeira coluna e com o endereço da primeira célula da segunda coluna:

```
Sub CompareColumns2()
```

```
    Dim cell As Range
```

```
    Dim firstRange As Range, firstCell As Range
```

```
    Dim I As Integer
```

```
    ' Habilita o tratamento de erros. Isso é necessário para evitar um erro de execução
```

```
    ' Caso o usuário clique no botão Cancelar da janela do InputBox.
```

```
    On Error Resume Next
```

```
    ' Uso a função InputBox, com o parâmetro Type:=8. Conforme descrito anteriormente
```

```
    ' o valor 8 faz com que o usuário possa selecionar a faixa com o mouse.
```

```
    ' Também defino o valor padrão como a seleção atual.
```

```
Set firstRange = Application.InputBox(prompt:="Selecione a primeira coluna de células", _
    Type:=8,default:=Selection.Address(False, False))

' Encerra o procedimento se o usuário clicou em Cancelar ou Não.

If firstRange Is Nothing Then Exit Sub

' Emite uma mensagem para que o usuário clique na célula relacionada a
' a primeira célula da coluna selecionada anteriormente. Por exemplo,
' se a primeira faixa selecionada foi B2:B50, o mais provável é que a primeira
' Célula do segundo intervalo seja C2, para que a comparação seja B2 com C2,
' B3 com C3, B4 com C4 e assim por diante.

' Também utilizo o propriedade Address para exibir, no InputBox, o endereço
' da primeira célula (Cells(1), da primeira faixa. O False, False, significa
' que o endereço será retornado como Relativo e não como Absoluto.

Set firstCell = Application.InputBox( prompt:="Selecione a célula relacionada com: " & _
    firstRange.Cells(1).Address(False, False), Type:=8)

' Encerra o procedimento se o usuário clicou em Cancelar ou Não.

If firstCell Is Nothing Then Exit Sub

' Desliga o gerenciamento de erros.
On Error GoTo 0

I = 0

' Percorre todas as células do primeiro intervalo, comparando-as com as células
' Relacionadas no segundo intervalo.

For Each cell In firstRange
    If firstCell.Offset(I, 0).Value > cell.Value Then
        firstCell.Offset(I, 0).Interior.ColorIndex = 6
    Else
        firstCell.Offset(I, 0).Interior.ColorIndex = xlNone
    End If
    ' Incremento o contador I, para ir para a próxima célula.

    I = I + 1
Next

End Sub
```

Lição 21: Exemplos Práticos de Uso dos Objetos do Excel – Parte 6

Nesta lição apresentarei alguns exemplos de utilização do Objeto Range para operações de Editar, Copiar e Colar linhas e colunas. Também apresentarei alguns exemplos do uso do VBA e do objeto Range, para acessar e definir o valor de células em uma planilha do Excel.

Exemplos de Copiar e Colar, usando o objeto Range:

Ao copiar uma faixa de células de uma planilha para outra, não é necessário ativar a planilha de destino. Também é importante especificar a faixa de destino, senão o Excel irá colar a faixa de origem, na mesma faixa de células, na planilha de destino.

Vejamos alguns exemplos de Copiar e Colar, usando o objeto Range:

```
Sub ExemploCopiar()  
  
    Dim srcRng As Range  
    Dim dstRng As Range  
  
    ' Associa as variáveis do tipo Range, com a faixa de origem e destino  
    ' Observe que a faixa de destino é somente uma célula, ou seja, a célula  
    ' que será o canto superior esquerdo da faixa de destino.  
  
    Set srcRng = Workbooks("Pasta1.xls").Sheets("Plan1").Range("A1:C50")  
    Set dstRng = Workbooks("Pasta1.xls").Sheets("Plan2").Range("A1")  
  
    ' Primeira maneira de efetuar a cópia.  
  
    ' Copio a faixa de origem para a memória: Editar -> Copiar ou Ctrl+C  
    srcRng.Copy  
  
    ' Copia a faixa de origem na faixa de destino.  
  
    Workbooks("Pasta1.xls").Sheets("Plan2").Paste dstRng  
  
    ' Segunda maneira de efetuar a cópia.  
  
    ' Ao chamar o método Copy, já especifico a faixa de destino.  
  
    srceRng.Copy destRng  
  
End Sub
```

Vamos a mais um exemplo. O exemplo a seguir mostra como copiar uma faixa de células, em uma planilha, para a mesma faixa, em outra planilha:

```
Sub ExemploCopiar2()  
  
    Dim EndFaixa As String  
    EndFaixa = "C1:E50"  
  
    Worksheets("Plan1").Range(EndFaixa).Copy _  
        Destination:=Worksheets("Plan2").Range(szRange)  
  
End Sub
```

Neste exemplo, faço uma referência a faixa cujo endereço está definido na variável EndFaixa, da Planilha Plan1. Chamo o método Copy e passo, como valor para o parâmetro Destination, a mesma faixa de endereços, só que na planilha Plan2. O efeito prático é que as células C1:E50 da planilha Plan1 são copiadas para a mesma faixa – C1:E50, na planilha Plan2.

Exemplos de leitura e definição de valores, em células de uma planilha:

A seguir mostrarei alguns exemplos de como usar o VBA, para definir valores em células de uma planilha e também como ler valores e detectar o tipo de valor contido em uma célula.

Vamos iniciar com alguns exemplos de como detectar o tipo de dado existente em uma célula. Existem diferentes maneiras para detectar o tipo de dado de uma célula. A seguir um exemplo onde utilizo o objeto ActiveCell, o qual retorna um objeto Range, que faz referência à célula ativa, ou seja, a célula onde está o cursor.

Para acessar o valor da célula ativa e armazená-lo em uma variável, use o código a seguir. Observe que, ao declarar a variável, não defini o seu tipo, pois não sei o tipo de dados existente na célula ativa:

```
Dim ValorDaCelula  
  
' Atribuo o valor da Célula Ativa à variável ValorDaCelula  
  
ValorDaCelula = ActiveCell.Value  
  
' Testo para ver se algum valor foi atribuído. Se a célula estiver vazia, encerro a macro.  
  
If ValorDaCelula = "" Then Exit Sub
```

A seguir mostro outra maneira de testar se a célula ativa está vazia. Neste exemplo, uso a função IsEmpty, a qual foi explicada no Módulo 1:

```
If IsEmpty(ActiveCell) Then  
    MsgBox "A célula está vazia!!!"  
End If
```

No exemplo a seguir, uso a função IsNumeric para testar se o valor da célula atual é um número ou pode ser convertido para número. Pode tanto ser um número que foi digitado diretamente na célula, quanto uma fórmula que retorna um resultado numérico.

```
If IsNumeric(ActiveCell.Value)Then
    MsgBox "O valor da célula é numérico ou pode ser convertido para número"
End If
```

Você também pode utilizar, no código VBA, todas as funções de planilha do Excel. Para isso uso o objeto Application e o nome da função em Inglês. No exemplo a seguir, uso a função IsNumber, como se fosse um método do objeto Application. (na prática todas as funções de planilha podem ser vistas como métodos do objeto Application). Este comando irá retornar Verdadeiro, somente se o valor da célula ativa for um número:

```
Application.IsNumber(ActiveCell.Value)
```

No exemplo a seguir, uso a função IsText. Este comando irá retornar verdadeiro, somente se o valor da célula ativa for do tipo String:

```
Application.Istext(ActiveCell.Value)
```

No exemplo a seguir, utilizo a propriedade HasFormula, para verificar se a célula atual tem uma fórmula. Esta propriedade retorna True se a célula contiver uma fórmula e False, caso contrário. Quando aplicada a uma faixa com mais de uma célula, a propriedade HasFormula retorna True, se todas as células da faixa, contiverem fórmulas; False, se nenhuma das células contiver fórmula e Null, se algumas células contiverem fórmulas e outras não.

```
If ActiveCell.HasFormula Then
    MsgBox "A Célula ativa contém uma fórmula!!!"
End If
```

No exemplo a seguir, utilizo a função IsError, para detectar se a célula atual tem um erro, tal como #DIV/0! ou #REF!

```
If IsError(ActiveCell) Then
    MsgBox "A célula ativa contém um Erro!!!"
End If
```

O exemplo a seguir, retorna o tipo de entrada da célula ativa. Por exemplo, ela retorna String, se a entrada for texto, Double, se a entrada for numérica e Boolean se a entrada for True ou False

```
MsgBox TypeName(ActiveCell.Value)
```


Exemplos de como obter informações a respeito de uma célula:

O exemplo a seguir, exibe o valor da célula ativa:

```
Sub InfoExample1()  
    ' Este exemplo exibe o valor da célula ativa  
    MsgBox ActiveCell.Value  
End Sub
```

O exemplo a seguir, determina se a célula ativa está vazia:

```
Sub InfoExample2()  
    ' Este exemplo verifica se a célula ativa está vazia  
  
    If IsEmpty(ActiveCell) Then  
        MsgBox "A célula está vazia!!!"  
    Else  
        MsgBox "A célula não está vazia"  
    End If  
End Sub
```

O exemplo a seguir determina, quais células, na seleção atual, contém valores numéricos. Uma vez encontrado um valor numérico, este valor é aumentado em 20%:

```
Sub InfoExample3()  
  
    ' São necessários dois testes: IsNumeric e IsEmpty, porque a função  
    ' IsNumeric irá retornar TRUE, se a célula estiver Vazia.  
  
    Dim cell As Range  
  
    For Each cell In Selection  
        If Not IsEmpty(cell) And IsNumeric(cell.Value) Then  
            Cell.Value = Cell.Value * 1.2  
        End If  
    Next  
End Sub
```

Lição 22: Exemplos Práticos de Uso dos Objetos do Excel – Parte 7

Nesta lição apresentarei mais alguns exemplos práticos de utilização do Objeto Range.

Exemplo: Neste exemplo, você aprenderá a utilizar o VBA para separar os dados de uma coluna em uma ou mais colunas. A título de exemplo, vamos supor que você importou dados de um arquivo de texto e os dados da coluna A contém dados do nome e do código do cliente, conforme exemplo da Figura a seguir:

	A	B	C
1	Nome e Código	Código	Nome
2	Josue da Silva (100001)		
3	Maria Aparecida (200002)		
4	Antonio da Costa (300003)		
5	Pedro Silva (400004)		
6	Carlos Santos (500005)		
7	Dorotéia Vernalis (600006)		
8	Canton Reinoldo (700007)		
9			

O objetivo do nosso exemplo é extrair somente o código da coluna A (sem os parênteses) e colocá-lo na coluna B e extrair somente o nome e colocá-lo na coluna C.

A seguir apresento o código para solucionar a questão proposta. Em seguida mostrarei o resultado da execução deste código. No código você encontra comentários detalhados sobre os comandos utilizados

Sub Separa()

```
Dim rngCell As Range
Dim strName As String
Dim AbreParen As Integer
Dim FechaParen As Integer
```

```
' Defino um laço For..Next da célula A2 até a última célula, com dados, na col A.
' Para detectar a última linha, com dados, da coluna A, utilizo: Range("A1").End(xlDown))
' A propriedade End retorna um objeto Range representando a célula no final da região
' que contém o intervalo de origem. Equivalente ao pressionamento de END+SETA
' ACIMA, END+SETA ABAIXO, END+SETA À ESQUERDA ou END+SETA À
' DIREITA. Passei o valor xlDown como parâmetro. Isso é equivalente a pressionar
' End+Seta Para baixo, ou seja, a última célula da coluna A, onde tem dados.
' No nosso exemplo será retornado o endereço da célula A8.
' Com isso a faixa a ser trabalhada será A2:A8.
```

```
For Each rngCell In Range("A2", Range("A1").End(xlDown))

' Armazena o valor da célula na variável strName

    strName = rngCell.Value

' Procura a posição dos parênteses dentro do valor da célula.
' Para isso utilizamos a função InStr, Esta função é utilizada para pesquisar a ocorrência de
' Uma sequência de caracteres, dentro de uma String e retorna o número que indica a posição
' da ocorrência do primeiro caractere que atende a pesquisa. Considere o exemplo a seguir:
' InStr(1,"ABC123456","C"). Esta função inicia a pesquisa na primeira posição – o primeiro parâmetro
' define a posição onde iniciará a pesquisa. A pesquisa é feita na String ABC123456 – segundo
' parâmetro e pesquisa a string "C" – terceiro parâmetro. Este exemplo retorna o valor 3, que é
' a posição do "C", dentro da string ABC123456.

' Retorna a posição do Abre Parênteses (
    AbreParen = InStr(1, strName, "(")

' Retorna a posição do Fecha Parênteses )
    FechaParen = InStr(1, strName, ")")

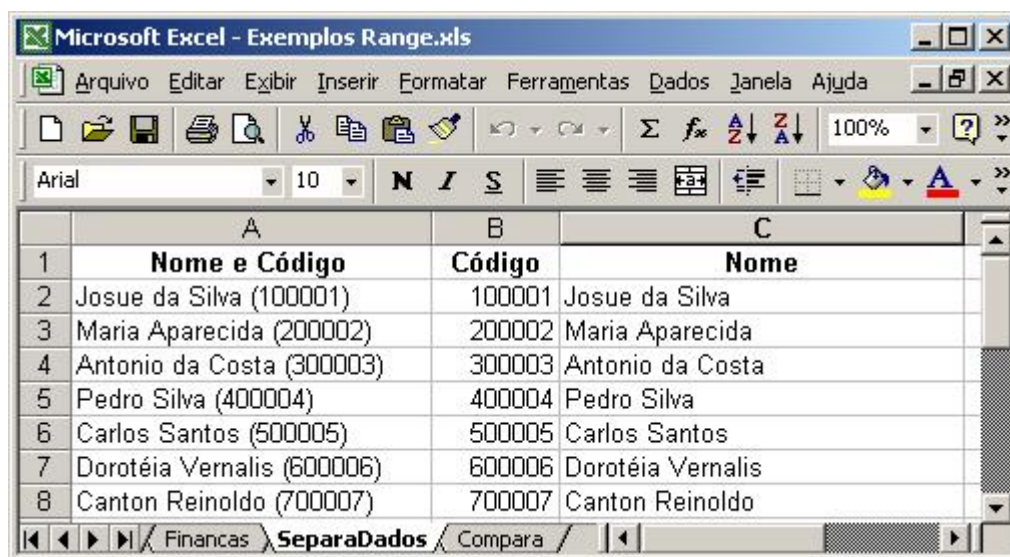
' Com base nas posições do abre e fecha parênteses, extrai o número que está dentro
' do parênteses e salva ele na Célula da Coluna B.

    rngCell.Offset(0, 1).Value = Mid(strName, AbreParen + 1, FechaParen - AbreParen - 1)

' Extrai o nome do cliente e salva na coluna C

    rngCell.Offset(0, 2).Value = Mid(strName, 1, AbreParen - 2)
Next rngCell
End Sub
```

O resultado da execução está indicado na Figura a seguir:



	A	B	C
1	Nome e Código	Código	Nome
2	Josue da Silva (100001)	100001	Josue da Silva
3	Maria Aparecida (200002)	200002	Maria Aparecida
4	Antonio da Costa (300003)	300003	Antonio da Costa
5	Pedro Silva (400004)	400004	Pedro Silva
6	Carlos Santos (500005)	500005	Carlos Santos
7	Dorotéia Vernalis (600006)	600006	Dorotéia Vernalis
8	Canton Reinoldo (700007)	700007	Canton Reinoldo

Este código tem alguns pontos interessantes que eu gostaria de comentar com mais detalhes, apesar de ter colocado bastante comentários nos exemplos. Este exemplo tem técnicas interessantes, as quais podem ser facilmente adaptadas para outras situações, um pouco diferentes, as quais você tenha em seu trabalho diário.

Comentários sobre o Código do Exemplo:

1) Como foi determinada a faixa com valores de dados: O primeiro ponto da lógica do exemplo é entender como foi determinada a faixa de dados, ou seja, até que linha temos dados a serem processados. Uma abordagem mais simples seria colocar a faixa diretamente no código, como no exemplo a seguir:

```
For Each rngCell In Range("A2:A8")
```

porém, sempre que possível, devemos criar nossos procedimentos para que sejam o mais genérico possíveis. Neste exemplo, optei por deixar que o próprio Excel detectasse a faixa de dados a serem processados. Se você não estivesse utilizando o VBA e quisesse localizar a última linha com dados, você colocaria o cursor em uma das células com dados e teclaria End e logo em seguida a Flecha Para baixo. Foi exatamente o que eu fiz, só que usando o código VBA:

```
For Each rngCell In Range("A2", Range("A1").End(xlDown))
```

O método End com o parâmetro xlDown faz exatamente isso, ou seja, é como se o cursor estivesse em A1 e eu pressionasse End e logo em seguida a tecla de Flecha Para Baixo. O método End irá retornar, o endereço da última célula nesta faixa, que no nosso exemplo é A8. Ou seja, o comando Range("A1").End(xlDown)), retorna A8. A vantagem desta segunda abordagem é que ela é genérica, ou seja, se você acrescentar linhas, na próxima vez que o método for executado, ele detectará as novas linhas e retornará o novo endereço da célula da última linha com dados a serem processados. Não será preciso alterar o código, o que seria, caso você tivesse colocado o endereço da faixa a ser processada, diretamente no código VBA.

2) Determinar a posição da abertura e do fechamento dos parênteses: O próximo passo na lógica do nosso exemplo é detectar a posição da abertura e do fechamento dos parênteses. Isso é feito utilizando a função InStr. Esta função recebe como parâmetros, a posição onde iniciar a pesquisa, a string a ser pesquisada e o que deve ser pesquisado. A função retorna um número que indica a posição da primeira ocorrência da string pesquisada. Usamos a função InStr para detectar a ocorrência da abertura dos parênteses e também do fechamento:

```
' Retorna a posição do Abre Parênteses (
  AbreParen = InStr(1, strName, "(")

' Retorna a posição do Fecha Parênteses )
  FechaParen = InStr(1, strName, ")")
```

Os valores de AbreParen e FechaParen serão utilizados, logo em seguida, para extrair somente o código do cliente. Para entender a lógica da nossa função, vamos considerar o exemplo da Célula A2, com a posição de cada caractere, conforme indicado na tabela a seguir:

Texto	J	o	s	u	e		d	a		S	i	l	v	a		(1	0	0	0	0	1)
Posição	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Neste exemplo observe que a abertura do parênteses está na posição 16 (AbreParen=16) e o fechamento do Parênteses está na posição 23 (FechaParen). A seguir mostrarei como estes valores são utilizados pela lógica da função, para extrair o código e o nome, nas colunas B e C.

3) **A lógica da extração do código e do nome:** Inicialmente uso a função Mid para extrair o código do cliente e salvá-lo na coluna B. Para isso uso o método Offset, já descrito anteriormente. O Offset(0,1), significa na mesma linha, porém uma coluna à direita, ou seja, como estamos na coluna A, significa na mesma linha, na coluna B. Observe que a função Mid usa os valores AbreParen e Fecha Paren, conforme indicado a seguir:

```
rngCell.Offset(0, 1).Value = Mid(strName, AbreParen + 1, FechaParen - AbreParen - 1)
```

Por que AbreParen+1? Porque o código começa uma posição após a posição da abertura do parênteses. No nosso exemplo, para o primeiro cliente, seria na posição 16+1, ou seja, na posição 17. OK, exatamente onde começa o código do cliente, conforme tabela anterior. O terceiro parâmetro é FechaParen-AbreParen -1. No nosso exemplo fica 23-16-1 = 6. Este parâmetro indica o número de caracteres a ser retornado. Este número é definido pela diferença entra a posição do Fechaparen e do Abreparen e o -1 é para não incluir o parênteses final, mas somente os números.

Seguinda a nossa lógica, é hora de extrair somente o nome do cliente. Para isso uso novamente o método Offset, já descrito anteriormente. O Offset(0,2), significa na mesma linha, porém duas colunas à direita, ou seja, como estamos na coluna A, significa na mesma linha, na coluna C. Observe que a função Mid usa os valores AbreParen e Fecha Paren, conforme indicado a seguir:

```
rngCell.Offset(0, 2).Value = Mid(strName, 1, AbreParen - 2)
```

Neste caso começo na posição 1, ou seja, no início da célula a retorno um número de caracteres igual ao valor de AbreParen-2. O -2 é para não retornar o próprio parênteses e nem o espaço em branco, entre o final do nome e o abre parênteses.

Muito bem, função testada e entendida, vamos a mais alguns exemplos, na próxima lição.

Lição 23: Exemplos Práticos de Uso dos Objetos do Excel – Parte 8

Nesta lição apresentarei mais alguns exemplos práticos de utilização do Objeto Range. Você aprenderá a usar o código VBA para definir fórmulas em uma ou mais células.

Exemplo: Preenchendo uma faixa de células com fórmulas.

No exemplo a seguir, preencho as células da coluna D, com uma fórmula que soma os valores das colunas A, B e C. Por exemplo, a fórmula na célula D1 será =Sum(A1:C1) ou =A1+B1+C1, a fórmula na célula D2 será =Sum(A2:C2) ou =A2+B2+C2 e assim por diante. Observe que deve ser usado o nome da função, em Inglês, ou seja Sum e não Soma.

```
Sub PreencheFormula()  
  
    Dim myRng As Range  
    Dim lastRw As Long  
  
    ' Uso o método End para obter a última linha da faixa de dados.  
  
    lastRw = Worksheets("Sheet1").Range("C1").End(xlDown).Row  
  
    Worksheets("Sheet1").Range("D1").Formula = "=SUM(A1:C1)"  
    Worksheets("Sheet1").Range("D1").AutoFill Destination:=Worksheets("Sheet1").Range("D1:D" & lastRw&)  
  
End Sub
```

Um detalhe interessante a ser comentado, neste caso, é a possibilidade de montar uma expressão, cujo resultado será utilizado como endereço. Observe os comandos a seguir:

```
Worksheets("Sheet1").Range("D1").Formula = "=SUM(A1:C1)"  
Worksheets("Sheet1").Range("D1").AutoFill Destination:=Worksheets("Sheet1").Range("D1:D" & lastRw&)
```

O primeiro comando, simplesmente, define a fórmula para a célula D1. O segundo comando é que faz o preenchimento da fórmula, para o restante da faixa. Mas o que é o restante da faixa? De D1 até D-ÚltimaLinha. E onde está o valor da última linha? Na variável lastRw, valor este obtido com o método End. Na hora de passar o endereço para o método AutoFill, observe que uso a expressão Range("D1:D" & lastRw&).

Se o valor da última linha for 20, esta fórmula fica assim:

```
Range("D1:D20")
```

Se o valor da última linha for 50, esta fórmula fica assim:

```
Range("D1:D50")
```

ou seja, com esta abordagem obtemos a tão desejada generalidade do procedimento, ou seja, ele se aplica a qualquer número de linhas, sem ter que ser alterado.

Exemplo: Neste exemplo, mostrarei como alterar os valores de células, em uma faixa de células, com base no próprio valor de cada célula.

No exemplo a seguir, uso um laço For...Each, para percorrer todas as células selecionadas em uma faixa. Em seguida faço alterações nos valores das células, dependendo do valor atual da célula.

```
Sub AlteraCondicional()  
  
    Dim cell As Range  
  
    For Each cell In Selection  
        Select Case Cell.Value  
            Case 1 To 17  
                ' Se o valor estiver entre 1 e 17, soma 54 ao valor atual.  
                Cell.Value = .Value + 54  
            Case 18 To 30  
                ' Se o valor estiver entre 18 e 30, soma 24 ao valor atual.  
                Cell.Value = .Value + 24  
        End Select  
  
        ' Neste exemplo, para valores entre 17 e 18 e para valores maiores do que 30,  
        ' não serão feitas alterações  
  
    Next cell  
  
    MsgBox "Alterações Efetuadas com Sucesso!!!"  
End Sub
```

Exemplo: Neste exemplo mostrarei como determinar o número de células atualmente selecionadas.

O procedimento a seguir informa o número de células selecionadas. Ele funciona também no caso de seleções não contínuas, ou seja, quando o usuário utilizou a tecla Ctrl para selecionar faixas de células em áreas não contínuas em uma ou mais planilhas.

```
Sub NumberOfCells()  
  
    Dim number As Long  
    Dim area As Range  
    For Each area In Selection.Areas  
        number = number + area.Cells.Count  
    Next  
    MsgBox number  
End Sub
```


Observe que, basicamente, o procedimento usa um laço For...Each para percorrer a coleção de Areas (diferentes faixas) da seleção atual. Dentro do laço, a variável number vai sendo incrementada com o número de células em cada área.

Exemplo: Como determinar se uma faixa está vazia:

No exemplo a seguir, mostro como usar a função CountA (Cont.Valores em Português) para verificar se o número de valores na faixa A1:E50 é igual a zero. Se for, significa que todas as células da faixa estão vazias.

```
If Application.CountA(Worksheets("Plan1").Range("A1:E50")) = 0 Then
    MsgBox "A faixa está vazia!!!"
Else
    MsgBox "A faixa não está vazia."
End If
```

Outra abordagem pode ser utilizar a função ISEmpty, passando como parâmetro para a função a faixa, conforme o exemplo a seguir:

```
If IsEmpty(Worksheets("Plan1").Range("A1:E50")) Then
    MsgBox "A faixa está vazia!!!"
End If
```

Lição 24: Exemplos Práticos de Uso dos Objetos do Excel – Parte 9

Nesta lição apresentarei mais alguns exemplos práticos de utilização do Objeto Range. Nos exemplos desta lição você aprenderá a determinar os limites de uma faixa, tal como detectar qual a última entrada em uma linha ou coluna. Também apresentarei uma lista de sites, infelizmente todos em Inglês, com excelentes dicas, exemplos e tutoriais sobre a programação VBA no Excel.

Exemplos: A seguir mostro os comandos do VBA equivalentes a pressionar Ctrl-Shift-Down (torna ativa a última planilha da pasta de trabalho, isto é, a planilha mais da direita) e Ctrl-Down (torna ativa a próxima planilha).

' O comando a seguir é equivalente a ctrl-shift-down

```
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

' O comando a seguir é equivalente a ctrl-down

```
ActiveCell.End(xlDown).Select
```

Como determinar a última célula, com dados, em uma coluna:

A seguir mostro a como atribuir um objeto range a uma variável, sendo que o objeto Range é uma referência a última célula de dados em uma coluna. Para colunas com dados contínuas, sem linhas em branco, e a variável topCell representa a primeira célula com dados, você pode chegar a última célula, usando o comando a seguir:

```
Set lastCell = topCell.End(xlDown)
```

Nota: Este comando funciona, mesmo que a variável topCell faça referência a uma célula que não está na planilha ativa.

Para casos mais complicados e, digamos, mais realistas, onde pode existir linhas em branco, você terá que usar uma abordagem um pouco diferente, para desconsiderar as linhas em branco e realmente localizar a última célula, com valores, em uma coluna.

Situação 01: Por exemplo, se você sabe que não existem dados abaixo da linha 1500 e que a planilha ativa é a planilha que contém a célula referenciada pelo objeto Range contido na variável topCell, você pode usar o comando a seguir, para acessar a última célula, com dados, na coluna de topCell:

```
Set lastCell = Cells(1501, topCell.Column).End(xlUp)
```

Situação 02: Se a faixa representada pela variável topCell, não está na planilha ativa, você pode usar o comando a seguir:

```
Set lastCell = topCell.Parent.Cells(1501,topCell.Column).End(xlUp)
```

O objeto Parent faz referência ao objeto pai do objeto Range representado por topCell. O objeto pai de um objeto Range é a planilha na qual o objeto se encontra. Então topCell.Range é uma maneira de fazer referência a planilha onde está topCell.

Situação 03: Vamos complicar um pouco mais? Se você não sabe quantas linhas tem a planilha ou nem uma idéia, como por exemplo, a partir da linha tal não existem dados, você tem que usar uma abordagem diferente das anteriores (mas em compensação mais genérica e que serve para qualquer situação). O exemplo a seguir obtém a última célula, com valores, em uma coluna.

```
Dim maxRow As Long
maxRow = topCell.Parent.UsedRange.Cells(topCell.Parent.UsedRange.Cells.Count).Row + 1
Set lastCell = topCell.Parent.Cells(maxRow, topCell.Column).End(xlUp)
```

No exemplo a seguir, mostro um código para localizar a última célula, não em Branco, em uma coluna. Neste exemplo, assumo que a variável c, contém o número da coluna a ser analisada (1 para a coluna A, 2 para a coluna B e assim por diante) e a planilha ativa é a planilha que contém a coluna na qual queremos localizar a última célula, não em branco.

```
Dim cell As Range
Set cell = Cells(65536, c).End(xlUp)
```

a lógica é extremamente simples: Vai para a última linha e reSSIONa End + Seta para Cima (xlUp).

Se a coluna em questão, não está na planilha ativa, você deve fazer referência à planilha na qual está a coluna a ser analisada, conforme exemplo a seguir:

```
Set cell = Sheets("Plan1").Cells(65536, c).End(xlUp)
```

Você pode tornar o procedimento mais genérico, usando a propriedade Count, para retornar o número da planilha. O exemplo a seguir irá funcionar para versões futuras do Excel, as quais venham a disponibilizar mais do que 65536 linhas por planilha:

```
Dim cell As Range
Set cell = Cells(Cells.Rows.Count, c).End(xlUp)
```

Como determinar a última entrada, com dados, em uma linha:

Existem diferentes maneiras para encontrar a última entrada com dados em uma linha, a exemplo do que ocorre com colunas. O exemplo a seguir é o, digamos assim, mais fácil de entender:

```
Dim lastRowEntryCell As Range
' Associa a variável com a última entrada, com dados, na linha 3
Set lastEntryCell = Cells(3, 256).End(xlLeft)
```

A lógica, novamente, é extremamente simples: Vai para a última coluna da linha – coluna 256 e pressiona End + Tecla para a Esquerda (xlLeft).

Se você tiver várias entradas em uma linha e não tiver células em branco, na faixa utilizada na linha, você pode localizar a última célula da linha, com dados, usando o código a seguir:

```
Dim lastEntryCell As Range
Set lastEntryCell = Range("A1").End(xlRight)
```

Ao utilizar End(xlDown) ou End(xlUp), fará com que seja feita uma rolagem na tela, o que pode causar demora na execução do procedimento, dependendo do tamanho da planilha. Para evitar esta rolagem, você pode usar o código a seguir:

```
' Grava as configurações de rolagem em uma variável
Dim scrollCol As Integer, ScrollRow As Integer

scrollCol = ActiveWindow.ScrollColumn
scrollRow = ActiveWindow.ScrollRow

' Define a variável range como sendo a última célula

' Aqui você insere o comando para localizar a última célula,
' usando um dos exemplos anteriores

' Aplica as configurações anteriores de rolagem
ActiveWindow.ScrollColumn = scrollCol
ActiveWindow.ScrollRow = scrollRow
```

Lista de sites com bons exemplos de programação VBA no Excel:

- ▶ <http://www.mrexcel.com/>
- ▶ <http://archive.baarns.com/pages/faqgen.asp>
- ▶ <http://www.vba-programmer.com/>
- ▶ <http://www.cpearson.com/excel/datetime.htm>
- ▶ <http://www.aldo.floripa.com.br/Dicas%20de%20Excel/dicas.htm>
- ▶ <http://edc.bizhosting.com/english/download.htm>
- ▶ <http://www.erlandsendata.no/english/>
- ▶ <http://www.mvps.org/dmccritchie/excel/excel.htm>
- ▶ <http://www.microsoftexceltraining.com/Excel/default.htm>
- ▶ <http://www.j-walk.com/ss/excel/eee/>
- ▶ <http://www.stfx.ca/people/bliengme/ExcelTips/>
- ▶ <http://www.mindspring.com/~tflynn/excelvba.html#bEnterWorkdays>
- ▶ <http://store.vitalnews.com/etarch.html>
- ▶ <http://www.ozgrid.com/Services/ExternalFree.htm>
- ▶ <http://www.ozgrid.com/download/default.htm>
- ▶ <http://www.meadinkent.co.uk/excel.htm>
- ▶ <http://www.fontstuff.com/vba/>
- ▶ <http://www.exceltip.com/>
- ▶ <http://www.add-ins.com/vbexmpls.htm>
- ▶ <http://www.uncc.edu/sysdev/HowTos/ExcelTipsRev.htm>
- ▶ <http://www.xl-logic.com/pages/vba.html>
- ▶ <http://users.pandora.be/educypedia/computer/msofficeexcel.htm>

Lição 25: Resumo do Módulo

Nas lições deste módulo você aprendeu a trabalhar com o objeto que realmente faz o trabalho no Excel: **Objeto Range**. Falando de uma maneira bem simples, o objeto Range representa uma ou mais faixas de células em uma planilha do Excel. Como a grande maioria do trabalho (para não dizer a totalidade) com o Excel está relacionado a valores em uma faixa de células, fica clara a importância do objeto Range.

Módulo 3 – O Objeto Range – Métodos, Propriedades e Exemplos Práticos

- Lição 01:** Apresentação do Objeto Range
- Lição 02:** Objeto Range – Outras Maneiras de Criar um Objeto Range
- Lição 03:** Objeto Range – Outras Maneiras de Criar um Objeto Range
- Lição 04:** Objeto Range – Column, Columns, Row e Rows – Parte 1
- Lição 05:** Objeto Range – Column, Columns, Row e Rows – Parte 2
- Lição 06:** Objeto Range – Principais Métodos e Propriedades – Parte 1
- Lição 07:** Objeto Range – Principais Métodos e Propriedades – Parte 2
- Lição 08:** Objeto Range – Principais Métodos e Propriedades – Parte 3
- Lição 09:** Objeto Range – Principais Métodos e Propriedades – Parte 4
- Lição 10:** Objeto Range – Principais Métodos e Propriedades – Parte 5
- Lição 11:** Objeto Range – Principais Métodos e Propriedades – Parte 6
- Lição 12:** Objeto Range – Principais Métodos e Propriedades – Parte 7
- Lição 13:** Objeto Range – Principais Métodos e Propriedades – Parte 8
- Lição 14:** Objeto Range – Principais Métodos e Propriedades – Parte 9
- Lição 15:** Objeto Range – Principais Métodos e Propriedades – Parte 10
- Lição 16:** Exemplos Práticos de Uso dos Objetos do Excel – Parte 1
- Lição 17:** Exemplos Práticos de Uso dos Objetos do Excel – Parte 2
- Lição 18:** Exemplos Práticos de Uso dos Objetos do Excel – Parte 3
- Lição 19:** Exemplos Práticos de Uso dos Objetos do Excel – Parte 4
- Lição 20:** Exemplos Práticos de Uso dos Objetos do Excel – Parte 5
- Lição 21:** Exemplos Práticos de Uso dos Objetos do Excel – Parte 6
- Lição 22:** Exemplos Práticos de Uso dos Objetos do Excel – Parte 7
- Lição 23:** Exemplos Práticos de Uso dos Objetos do Excel – Parte 8
- Lição 24:** Exemplos Práticos de Uso dos Objetos do Excel – Parte 9
- Lição 25:** Resumo do Módulo

No próximo módulo faremos um estudo detalhado dos objetos Workbook e Worksheet. O objeto Workbook representa uma pasta de trabalho do Excel (arquivo .xls) e o objeto Worksheet representa as planilhas, dentro de uma pasta de trabalho. Além do estudo dos objetos, veremos vários exemplos práticos, que utilizam estes objetos e os objetos estudados nos módulos 1, 2 e 3.

Bibliografia recomendada:

Confira as dicas de livros de Excel no seguinte endereço:

<http://www.juliobattisti.com.br/indicados/excel.asp>

Módulo 4 – Estudo dos Objetos Workbook e Worksheet

Nas lições deste módulo você aprenderá a trabalhar com mais dois objetos, do Modelo de Objetos do Excel:

- ▶ Objeto Workbook
- ▶ Objeto Worksheet

O objeto Workbook representa uma pasta de trabalho do Microsoft Excel (arquivo .XLS). O objeto Workbook é um membro da coleção Workbooks (coleção Workbooks do objeto Application). A coleção Workbooks contém todos os objetos Workbook (arquivos .XLS) atualmente abertos no Microsoft Excel. Você aprenderá a utilizar os principais métodos e propriedades do objeto Workbook.

O objeto Worksheet representa uma planilha, ou seja, o objeto Worksheet é utilizado para fazer referência a uma planilha do Excel. O objeto Worksheet é um membro da coleção Worksheets (coleção Worksheets do objeto Workbook). A coleção Worksheets contém todos os objetos Worksheet em uma pasta de trabalho. Você aprenderá a utilizar os principais métodos e propriedades do objeto Worksheet.

Além do estudo teórico, da declaração dos objetos Workbook e Worksheet e do estudo de suas propriedades e métodos, apresentarei diversos exemplos de códigos de funções práticas, as quais utilizam os objetos Worksheet, Workbook, Range e demais objetos, já estudados em módulos anteriores. Sem nenhuma dúvida, você encontrará um grande número de exemplos que poderão ser facilmente adaptados para o seu próprio uso e, em alguns casos, nem sequer será necessário fazer adaptações.

Neste módulo você também aprenderá sobre o importante conceito de Eventos. A programação no Windows é baseada no conceito de Eventos. Um Evento é uma ação que o Windows Executa, em resposta a uma ação do Usuário. Por exemplo, sempre que o usuário acessa uma planilha, trazendo-a para primeiro plano, ocorre o Evento Activate da planilha. Você pode criar código VBA, o qual será executado em resposta a um evento. Por exemplo, se você quer que determinados cálculos sejam feitos, sempre que uma planilha for ativada, basta colocar os comandos para realização destes cálculos, no evento Activate da planilha.

Você pode escrever procedimentos de evento no Microsoft Excel em nível de planilha, gráfico, tabela de consulta, pasta de trabalho ou aplicativo. Por exemplo, o evento Activate ocorre em nível de planilha e o evento SheetActivate está disponível em ambos os níveis de aplicativo e de pasta de trabalho. O evento SheetActivate de uma pasta de trabalho ocorre quando alguma planilha da pasta de trabalho é ativada. Em nível de aplicativo, o evento SheetActivate ocorre quando qualquer planilha de uma pasta de trabalho aberta é ativada.

Use a propriedade EnableEvents para ativar ou desativar eventos. Por exemplo, o uso do método Save para salvar uma pasta de trabalho faz com que o evento BeforeSave ocorra. Você pode evitar isso definindo a propriedade EnableEvents como False antes de chamar o método Save. Neste módulo, estudaremos os eventos em detalhes.

Lição 01: Apresentação dos Objetos Workbook e Worksheet

Nesta lição farei uma pequena introdução aos objetos Workbook e Worksheet. Utilizarei vários exemplos de códigos, para mostrar como declarar e criar estes objetos. Você verá que a sintaxe para a utilização destes objetos é bastante simples.

O Objeto Workbook:

O objeto Workbook representa uma pasta de trabalho do Microsoft Excel (arquivo .XLS). O objeto Workbook é um membro da coleção Workbooks. A coleção Workbooks contém todos os objetos Workbook atualmente abertos no Microsoft Excel. Em resumo, o objeto Workbook é utilizado para, através do código VBA, ter acesso a um arquivo .XLS.

Criando um objeto Workbook:

Existem diferentes maneiras para criar um objeto Workbook. Uma delas é usando, diretamente, a coleção Workbooks do objeto Application. Use `Workbooks(índice)`, onde índice é o número de índice ou o nome da pasta de trabalho, para retornar um único objeto Workbook. Por exemplo, suponha que você tenha aberto os arquivos `C:\Planilhas\Vendas.xls`, `C:\Planilhas\Contas.xls` e `C:\Planilhas\Finanças.xls`, nesta ordem. O exemplo seguinte ativa a pasta de trabalho um, ou seja `C:\Planilhas\Vendas.xls`:

```
Workbooks(1).Activate
```

O número de índice denota a ordem na qual as pastas de trabalho foram abertas ou criadas. `Workbooks(1)` é a primeira pasta de trabalho criada e `Workbooks(Workbooks.Count)` é a última criada. A ativação de uma pasta de trabalho não altera seu número de índice. Todas as pastas de trabalho são incluídas na contagem do índice, mesmo que elas estejam ocultas.

A propriedade `Name` retorna o nome da pasta de trabalho. Você não pode definir o nome usando essa propriedade; se você precisa alterar o nome, use o método `SaveAs` para salvar a pasta de trabalho com um nome diferente. O seguinte exemplo ativa a planilha `Plan1` na pasta de trabalho chamada `"Vendas.xls"` (a pasta de trabalho precisa já estar aberta no Microsoft Excel).

```
Workbooks("Vendas.xls").Worksheets("Plan1").Activate
```

Observe que, neste caso, ao invés de usar o índice como argumento para a coleção `Workbooks`, utilizei o nome da pasta de trabalho – `Vendas.xls`

A propriedade `ActiveWorkbook` retorna a pasta de trabalho que está ativa no momento. O exemplo seguinte define o nome do autor da pasta de trabalho ativa.

```
ActiveWorkbook.Author = "Júlio Battisti"
```

A propriedade `ThisWorkbook` retorna a pasta de trabalho onde há código do Visual Basic sendo executado. Na maioria dos casos, esta é a própria pasta de trabalho ativa. Entretanto, se o código do Visual Basic for parte de um suplemento, a propriedade `ThisWorkbook` não

retornará a pasta de trabalho ativa. Nesse caso, a pasta de trabalho ativa é a pasta de trabalho que está chamando o suplemento, enquanto que a propriedade `ThisWorkbook` retorna a pasta de trabalho do suplemento, ou seja, onde está o código que está sendo executado.

O Objeto Worksheet:

Objeto `Worksheet` representa uma planilha do Excel. Lembrando que, pelo modelo Hierárquico do Excel, um arquivo `.XLS` é chamado de Pasta de Trabalho e no VBA é representado pelo objeto `Workbook`. Dentro de uma pasta de trabalho pode haver uma ou mais planilhas, sendo que cada planilha, no VBA, é representada por um objeto `Worksheet`. O objeto `Worksheet` é um membro da coleção `Worksheets` do objeto `Workbook`. A coleção `Worksheets` contém todos os objetos `Worksheet` em uma pasta de trabalho.

Criando um objeto Worksheet:

Para criar um objeto `Worksheet`, você pode utilizar a coleção `Worksheets`. Use `Worksheets(índice)`, onde índice é número de índice ou nome da planilha, para retornar um único objeto `Worksheet`. O exemplo seguinte oculta a planilha um na pasta de trabalho ativa.

```
Worksheets(1).Visible = False
```

O número de índice da planilha denota a posição de uma planilha na barra de guias da pasta de trabalho. `Worksheets(1)` é a primeira planilha (mais à esquerda) na pasta de trabalho e `Worksheets(Worksheets.Count)` é a última. Todas as planilhas são incluídas na contagem do índice, mesmo quando estão ocultas.

Ao invés do índice, você poderia utilizar o nome da planilha. O exemplo seguinte, oculta a planilha `Plan1`:

```
Worksheets("Plan1").Visible = False
```

O nome da planilha é mostrado na guia da planilha. Use a propriedade `Name` para definir ou retornar o nome da planilha. O exemplo seguinte protege os cenários na planilha `Plan1`:

```
Worksheets("Plan1").Protect password:="drowssap", scenarios:=True
```

O objeto `Worksheet` é também um membro da coleção `Sheets`. A coleção `Sheets` contém todas as planilhas da pasta de trabalho (tanto folhas de gráfico quanto planilhas de trabalho).

Quando uma planilha é a planilha ativa, você pode usar a propriedade `ActiveSheet` para referir-se a ela. O exemplo seguinte usa o método `Activate` para ativar a planilha `Plan1`, define a orientação da página como modo paisagem e, em seguida, imprime a planilha.

```
Worksheets("Plan1").Activate  
ActiveSheet.PageSetup.Orientation = xlLandscape  
ActiveSheet.PrintOut
```

A seguir mais alguns exemplos básicos de utilização dos objetos `Workbook` e `Worksheets`.

Exemplos de utilização:

Como ativar uma pasta de trabalho: A Ativação de uma pasta de trabalho pelo uso do método Activate coloca a pasta de trabalho na janela ativa. O procedimento seguinte ativa a pasta de trabalho aberta chamada "Vendas.xls", na pasta C:\Planilhas:

```
Sub AtivaPasta()  
    Workbooks("C:\Planilhas\Vendas.xls").Activate  
End Sub
```

Criar uma nova pasta de trabalho: Para criar uma nova pasta de trabalho você usa o método Add. O procedimento a seguir cria uma nova pasta de trabalho. O Microsoft Excel dá automaticamente à pasta de trabalho o nome PastaN, onde N é o próximo número disponível. A nova pasta de trabalho se torna a pasta de trabalho ativa.

```
Sub AddOne()  
    Workbooks.Add  
End Sub
```

Uma maneira melhor de criar uma nova pasta de trabalho é atribuí-la a uma variável de objeto. No exemplo seguinte, o objeto do tipo Workbook retornado pelo método Add é atribuído a uma variável de objeto, NovaPasta. Em seguida, várias propriedades de NovaPasta são definidas. Você pode facilmente controlar a nova pasta de trabalho usando a variável de objeto.

```
Sub AddNew()  
    Set NovaPasta = Workbooks.Add  
    NovaPasta.Title = "Primeiro Trimestre - 2003"  
    NovaPasta.Subject = "Vendas do Trimestre"  
    NovaPasta.SaveAs filename:="C:\Planilhas\Vendas.xls"  
End Sub
```

Abrir uma pasta de trabalho: Quando você abre uma pasta de trabalho usando o método Open, ela se torna um membro da coleção Workbooks. O procedimento seguinte abre uma pasta de trabalho chamada Vendas.xls localizada na pasta chamada "Planilhas" na unidade C.

```
Sub AbrePasta()  
    Workbooks.Open("C:\Planilhas\Vendas.xls")  
End Sub
```

Referir-se a planilhas pelo número de índice: Um número de índice é um número sequencial atribuído a uma planilha, com base na posição de sua guia de planilha (contando da esquerda para a direita) entre planilhas do mesmo tipo. O procedimento seguinte usa a propriedade Worksheets para ativar a planilha um da pasta de trabalho ativa.

```
Sub AtivaBemDaEsquerda()  
    Worksheets(1).Activate  
End Sub
```

Se você desejar trabalhar com todos os tipos de planilha (planilhas, gráficos, módulos e folhas de caixa de diálogo), use a propriedade Sheets. O procedimento seguinte ativa a planilha

quatro na pasta de trabalho. Observe que Worksheets faz referência apenas a planilhas e não a folhas e caixas de diálogo, já Sheets contém todos os tipos de folhas de planilhas do Excel.

```
Sub AtivaQuarta()  
    Sheets(4).Activate  
End Sub
```

Observação: A ordem dos índices pode ser alterada se você mover, adicionar ou excluir planilhas.

Referir-se a planilhas por nome: Você pode identificar planilhas pelo nome usando as propriedades Worksheets e Charts. As instruções seguintes ativam várias planilhas na pasta de trabalho ativa.

```
Worksheets("Plan1").Activate  
Charts("Gráfico1").Activate  
DialogSheets("Dialogo1").Activate
```

Você pode usar a propriedade Sheets para retornar uma planilha, gráfico, módulo ou folha de caixa de diálogo; a coleção Sheets contém todos estes. O exemplo seguinte ativa a planilha chamada "Plan1" na pasta de trabalho ativa.

```
Sub ActivateChart()  
    Sheets("Plan1").Activate  
End Sub
```

Observação: Os gráficos incorporados em uma planilha são membros da coleção ChartObjects, enquanto que gráficos existentes em suas próprias folhas pertencem à coleção Charts.

Lição 02: Objeto Workbook e Coleção Workbooks – Métodos e Propriedades

A partir desta lição passarei a descrever as principais propriedades e métodos do objeto Workbook. Através de exemplos práticos, você aprenderá a utilizar diversos métodos e propriedades do objeto Workbook.

A coleção Workbooks:

Inicialmente vou falar sobre a coleção Workbooks do objeto Application. Normalmente é esta coleção que utilizamos, para retornar um objeto Application. Esta coleção retorna uma coleção Workbooks representando todas as pastas de trabalho abertas. Esta propriedade é somente leitura.

O uso dessa propriedade sem um qualificador de objeto equivale a usar Application.Workbooks. Ou seja, se você usar, no código, somente Workbooks, será o mesmo que utilizar Application.Workbooks. Por isso que em alguns exemplos da Lição 1, utilizamos apenas Workbooks. Eu, particularmente, por questão de clareza, prefiro utilizar sempre Application.Workbooks.

A coleção retornada pela propriedade Workbooks não inclui os suplementos abertos, que são um tipo especial de pasta de trabalho oculta. Você pode, entretanto, retornar um único suplemento aberto se souber o nome do arquivo. Por exemplo, Workbooks("CalcFinan.xla") retornará o suplemento aberto chamado "CalcFinan.xla" como um objeto Workbook.

Exemplos da propriedade Workbooks:

O exemplo a seguir ativa a pasta de trabalho Vendas.xls.

```
Workbooks("Vendas.xls").Activate
```

O exemplo a seguir abre a pasta de trabalho Vendas.xls. Observe que neste exemplo estou utilizando o recurso de parâmetros nomeados, conforme descrito no Módulo 2:

```
Workbooks.Open filename:="LARGE.XLS"
```

O exemplo a seguir salva as alterações e fecha todas as pastas de trabalho exceto a que está executando o exemplo.

```
For Each Pasta In Workbooks
    If Pasta.Name <> ThisWorkbook.Name Then
        Pasta.Close savechanges:=True
    End If
Next Pasta
```

Este exemplo tem vários pontos interessantes. Utilizamos um laço For...Each, para percorrer todas as pastas de trabalho da coleção Workbooks. Observe que utilizei apenas Workbooks. Também poderia ter utilizado Application.Workbooks.

Em seguida faço um teste, para verificar se o nome da pasta que está sendo analisada, na passagem do laço, é diferente do nome da pasta atual. A referência a pasta atual é feita usando o objeto ThisWorkbook. Se o nome for diferente, ou seja, se não for a pasta atual, esta será fechada e as alterações serão salvas. O que define que as alterações serão salvas é o parametro Savechanges:=True.

O Método Close:

É utilizado para fechar o objeto Workbook, ou seja, para fechar uma pasta de trabalho.

Sintaxe:

`expressão.Close(SaveChanges, FileName, RouteWorkbook)`

► **expressão**: Obrigatória. Uma expressão que retorne um objeto do tipo Workbook

► **SaveChanges**: É do tipo Variant e é opcional. Se não houver alterações na pasta de trabalho, esse argumento será ignorado. Se houver alterações na pasta de trabalho e ela aparecer em outras janelas abertas, esse argumento será ignorado. Se houver alterações na pasta de trabalho, mas ela não aparecer em qualquer outra janela aberta, esse argumento especificará se as alterações devem ser salvas. Os valores possíveis para este argumento, estão indicados na tabela a seguir:

Valor	Ação
True	Salva as alterações na pasta de trabalho. Se ainda não houver um nome de arquivo associado à pasta de trabalho, o valor definido no argumento FileName será usado. Se FileName for omitido, o usuário será solicitado a fornecer um nome de arquivo.
False	Não salva as alterações nesse arquivo.
Omitido	Exibe uma caixa de diálogo perguntando ao usuário se as alterações devem ser salvas.

► **FileName**: É do tipo Variant e é opcional. As alterações são salvas sob este nome de arquivo.

► **RouteWorkbook**: É do tipo Variant e é opcional. Se a pasta de trabalho não precisar ser encaminhada para o próximo destinatário (se não tiver lista de circulação ou se já tiver sido encaminhada), este argumento será ignorado. Caso contrário, o Microsoft Excel encaminhará a pasta de trabalho como mostrado na tabela seguinte.

Valor	Significado
True	Envia a pasta de trabalho para o próximo destinatário.
False	Não envia a pasta de trabalho.

Omitido	Exibe uma caixa de diálogo perguntando ao usuário se a pasta de trabalho deve ser enviada.
---------	--------------------------------------------------------------------------------------------

Exemplo do método Close:

Este exemplo fecha a pasta Plan1.xls e descarta quaisquer alterações que tenham sido feitas nela – parâmetro SaveChanges:=False.

```
Workbooks("BOOK1.XLS").Close SaveChanges:=False
```

O exemplo a seguir fecha todas as pastas de trabalho abertas. Se houver alterações em alguma pasta de trabalho aberta, o Microsoft Excel exibirá os avisos e caixas de diálogo apropriadas para salvamento das alterações. Observe que, neste exemplo, estou usando o método Close da coleção Workbooks e não o método Close de um objeto Workbook específico:

```
Workbooks.Close
```

A Propriedade ActiveSheet:

A propriedade ActiveSheet retorna um objeto representando a planilha ativa (a planilha visível) da pasta de trabalho ativa ou na janela ou pasta de trabalho especificada. Retorna Nothing se não houver planilha ativa. Somente leitura.

Nota: Se você não especificar um qualificador de objeto, essa propriedade retornará a planilha ativa da pasta de trabalho ativa. Se uma pasta de trabalho aparece em mais de uma janela, a propriedade ActiveSheet poderá ser diferente em janelas diferentes.

O exemplo a seguir exibe o nome da planilha ativa:

```
MsgBox "O nome da planilha ativa é: " & ActiveSheet.Name
```

Observe que o uso de ActiveSheet é um atalho, bastante útil, para retornar um objeto que aponta para a planilha ativa.

Propriedade ActiveChart:

Retorna um objeto Chart representando o gráfico ativo (seja um gráfico incorporado ou uma folha de gráfico). Um gráfico incorporado é considerado ativo quando está selecionado ou ativado. Quando nenhum gráfico está ativo, essa propriedade retorna Nothing. Somente leitura.

Nota: Se você não especificar um qualificador de objeto, essa propriedade retornará o gráfico ativo da pasta de trabalho ativa.

Este exemplo ativa a legenda do gráfico ativo.

```
ActiveChart.HasLegend = True
```

Lição 03: Objeto Workbook e Coleção Workbooks – Métodos e Propriedades

Nesta lição apresentarei mais algumas propriedades e métodos do objeto Workbook. Vamos iniciar o módulo com algumas propriedades relacionadas ao nome e ao caminho onde está o arquivo .XLS, associado ao objeto Workbook.

A Propriedade FullName:

Esta propriedade retorna o nome do objeto, incluindo seu caminho no disco, na forma de uma sequência. O valor de retorno é do tipo String e é uma propriedade somente leitura.

Nota: Essa propriedade é equivalente à propriedade Path (que será vista mais adiante), seguida pelo atual separador do sistema de arquivos, e seguida pela propriedade Name.

Exemplo da propriedade FullName:

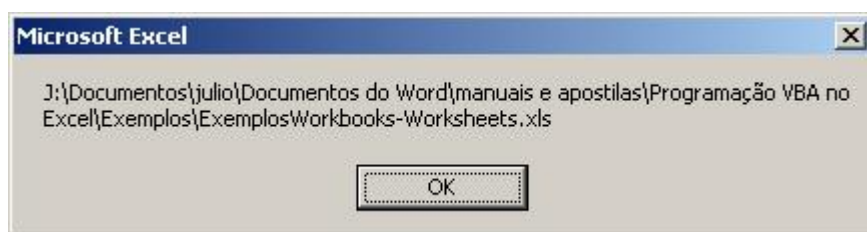
O exemplo a seguir exibe o caminho e o nome de arquivo de cada suplemento disponível. Lembrando que usando a coleção AddIns, temos acesso a todos os suplementos que estão disponíveis para a pasta de trabalho atual.

```
For Each a In AddIns
    MsgBox a.FullName
Next a
```

Este outro exemplo exibe o caminho e o nome de arquivo da pasta de trabalho ativa (desde que a pasta de trabalho tenha sido salva).

```
MsgBox ActiveWorkbook.FullName
```

Na Figura a seguir, temos um exemplo de execução do comando anterior:



A Propriedade Name:

É do tipo leitura e escrita, ou seja, pode ser utilizada para retornar ou definir o nome da pasta de trabalho. O nome de um objeto Range é um objeto Name. Para todos os outros tipos de objeto, o nome é uma cadeia de caracteres.

Considere o exemplo a seguir, aplicado a mesma pasta de trabalho do exemplo anterior:

```
MsgBox ActiveWorkbook.Name
```

O resultado da execução deste comando está indicado na Figura a seguir:



Observe que somente o nome da pasta de trabalho, sem o caminho, é retornado.

A propriedade Path:

A propriedade Path retorna o caminho completo do objeto, excluindo o separador final e o nome do objeto. É uma propriedade do tipo String e é somente leitura.

Nota: Usar esta propriedade sem um qualificador de objeto é equivalente a Application.Path (isto retorna o caminho para o aplicativo Microsoft Excel), isto é, o caminho onde o Excel está instalado.

Considere o exemplo a seguir, aplicado a mesma pasta de trabalho do exemplo anterior:

```
MsgBox ActiveWorkbook.Path
```

O resultado da execução deste comando esta indicado na Figura a seguir:



Observe que foi retornado somente o caminho, sem o nome da pasta de trabalho. Na prática, podemos montar a seguinte equação:

```
FullPath = Path + Name
```

Ou seja, FullPath retorna o caminho completo (Caminho + Nome), Path retorna somente o Caminho e Name somente o nome.

O Método SaveAs:

Este método é o equivalente do comando Arquivo -> Salvar como.... É utilizado para salvar as alterações da planilha (Sintaxe 1) ou pasta de trabalho (Sintaxe 2) em um arquivo diferente do arquivo atual:

Sintaxe 1 – para Planilhas:

expressão.SaveAs(Filename, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AddToMru, TextCodePage, TextVisualLayout)

Sintaxe 2 – para Pasta de Trabalho:

expressão.SaveAs(Filename, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru, TextCodePage, TextVisualLayout)

A seguir coloco a descrição dos parâmetros utilizados pelo Método SaveAs:

- ▶ **expressão:** Obrigatória. Uma expressão que retorne um objeto Chart ou Worksheet (Sintaxe 1) ou um objeto Workbook (Sintaxe 2).
- ▶ **Filename:** Variant opcional. Uma sequência de caracteres que indique o nome do arquivo a ser salvo. Você pode incluir um caminho completo; se não o fizer, o Microsoft Excel salvará o arquivo na pasta atual.
- ▶ **FileFormat:** Variant opcional. O formato do arquivo que deve ser usado ao salvá-lo. Para obter uma lista de escolhas válidas, consulte a propriedade FileFormat, descrita na próxima lição. Para um arquivo existente, o formato padrão é o último formato de arquivo especificado; para um novo arquivo, o padrão é o formato da versão do Excel que está sendo usada.
- ▶ **Password:** Variant opcional. Uma sequência de caracteres distinguindo maiúsculas de minúsculas (de até 15 caracteres) que indique a senha de proteção de abertura a ser dada ao arquivo.
- ▶ **WriteResPassword:** Variant opcional. Uma sequência de caracteres que indique a senha de permissão de gravação deste arquivo. Se um arquivo for salvo com a senha e esta não for fornecida quando o arquivo é aberto, o arquivo será aberto como somente leitura.
- ▶ **ReadOnlyRecommended:** Variant opcional. True para exibir uma mensagem quando o arquivo é aberto, recomendando que o arquivo seja aberto como somente leitura.
- ▶ **CreateBackup:** Variant opcional. True para criar um arquivo de backup, antes de salvar a pasta de trabalho.
- ▶ **AccessMode:** Variant opcional. O modo de acesso da pasta de trabalho. Pode ser uma das seguintes constantes: xlShared (lista compartilhada), xlExclusive (modo exclusivo) ou xlNoChange (não alterar o modo de acesso). Se esse argumento for omitido, o modo de acesso não será alterado. Esse argumento é ignorado quando você salva uma lista compartilhada sem alterar o nome do arquivo. Para alterar o modo de acesso, use o método ExclusiveAccess.
- ▶ **ConflictResolution:** Variant opcional. Especifica o modo no qual os conflitos de alteração serão resolvidos se a pasta de trabalho for uma lista compartilhada. Pode ser uma das seguintes constantes: xlUserResolution (exibir a caixa de diálogo de

solução de conflitos), `xlLocalSessionChanges` (aceitar automaticamente as alterações do usuário local) ou `xlOtherSessionChanges` (aceitar outras alterações em vez das alterações do usuário local). Se esse argumento for omitido, a caixa de diálogo de solução de conflitos é exibida.

► **AddToMru:** Variant opcional. True para adicionar esta pasta de trabalho à lista de arquivos usados recentemente. O valor padrão é False.

► **TextCodePage:** Variant opcional. Não usado no Microsoft Excel em inglês americano.

► **TextVisualLayout:** Variant opcional. Não usado no Microsoft Excel em inglês americano.

Exemplo do método SaveAs: Este exemplo cria uma nova pasta de trabalho, solicita um nome de arquivo ao usuário e, em seguida, salva a pasta de trabalho.

```
Set NovaPasta = Workbooks.Add

Do
    NomeArquivo = Application.GetSaveAsFilename
Loop Until NomeArquivo <> False

NovaPasta.SaveAs Filename:=NomeArquivo
```

Inicialmente crio um objeto do tipo `Workbook`, chamado `NovaPasta`. Para isso uso o método `Add`, da coleção `Workbooks`. Em seguida uso um laço `Do`, para exibir uma caixa de diálogo (método `GetSaveAsFilename`), solicitando que o usuário digite um nome. Enquanto o usuário não digitar um nome de arquivo válido (a verificação se o nome digitado é ou não válido é feita pelo método `GetSaveAsFilename`), a caixa de diálogo continuará sendo exibida. Após o usuário ter digitado um nome válido, o laço `Do...Until` será encerrado e o nome informado pelo usuário será utilizado pelo método `SaveAs`. O nome é passado para o parâmetro `Filename`. É isso.

Lição 04: Objeto Workbook e Coleção Workbooks – Métodos e Propriedades

Vamos continuar o estudo das principais propriedades, métodos e coleções do objeto Workbook.

A Propriedade FileFormat:

Esta propriedade é do tipo somente leitura e retorna o formato e o tipo da pasta de trabalho a qual se refere o objeto Workbook. Esta propriedade pode assumir o valor de uma das constantes indicadas na tabela a seguir:

xlAddIn	xlSYLK
xlCSV	xlTemplate
xlCSVMac	xlTextMac
xlCSVMSDOS	xlTextMSDOS
xlCSVWindows	xlTextPrinter
xlCurrentPlatformText	xlTextWindows
xlDBF2	xlUnicodeText
xlDBF3	xlWJ2WD1
xlDBF4	xlWK1
xlDIF	xlWK1ALL
xlExcel2	xlWK1FMT
xlExcel2FarEast	xlWK3
xlExcel3	xlWK4
xlExcel4	xlWK3FM3
xlExcel4Workbook	xlWKS
xlExcel5	xlWorkbookNormal
xlExcel7	xlWorks2FarEast
xlExcel9795	xlWQ1
xlHTML	xlWJ3
xlIntlAddIn	xlWJ3FJ3
xlIntlMacro	

A maioria dos valores são auto-descritivos. Por exemplo, xlDBF3 significa formato Dbase III, xlHTML, formato HTML e assim por diante.

Exemplo de uso da propriedade FileFormat: O exemplo a seguir, salva a pasta de trabalho ativa, no formato do Excel 2:.

```
ActiveWorkbook.SaveAs fileFormat:=xlExcel2
```

A Propriedade Worksheets:

Esta propriedade Retorna uma coleção de objetos do tipo Sheets, representando todas as planilhas da pasta de trabalho ativa. Somente leitura.

Nota: Os uso dessa propriedade sem um qualificador de objeto retorna todas as planilhas da pasta de trabalho ativa.

Dica: Essa propriedade não retorna folhas de macro; use a propriedade Excel4MacroSheets ou a propriedade Excel4IntlMacroSheets para retornar essas folhas.

Exemplos da propriedade Worksheets:

O exemplo a seguir exibe o valor da célula A1 da planilha Plan1 na pasta de trabalho ativa:

```
MsgBox Worksheets("Plan1").Range("A1").Value
```

Neste exemplo, passo como parâmetro para Worksheets, o nome da planilha a ser acessada. O nome deve ser passado entre aspas.

O exemplo a seguir exibe o nome de todas as planilhas da pasta de trabalho ativa.

```
For Each ws In Worksheets  
    MsgBox ws.Name  
Next ws
```

O exemplo a seguir adiciona uma nova planilha à pasta de trabalho ativa e, em seguida, define o nome da planilha.

```
Set newSheet = Worksheets.Add  
newSheet.Name = "Vendas"
```

No exemplo anterior, se a planilha Vendas já existisse, seria gerado um erro em tempo de execução. Uma técnica recomendada, antes de criar uma nova planilha, é percorrer todas as planilhas existentes, verificando se a planilha já existe. Se já existir você deve tomar uma ação, como por exemplo excluir a planilha já existente ou renomeá-la, dependendo da lógica do programa que você está desenvolvendo. No trecho de código a seguir mostro como fazer esta verificação:

```
Dim NovaPlanilha As Worksheet  
  
' Percorro todas as planilhas da pasta de trabalho atual, para  
' verificar se já existe uma planilha com o nome Vendas.  
' Se já existir, excluo a planilha  
  
Application.DisplayAlerts = False  
  
For Each wks In ActiveWorkbook.Worksheets  
    If wks.Name = "Vendas" Then  
        ActiveWorkbook.Worksheets("Vendas").Delete  
    End If  
Next  
  
Application.DisplayAlerts = False  
  
' Crio a planilha Vendas, usando o método  
  
Set NovaPlanilha = Worksheets.Add  
NovaPlanilha.Name = "Vendas"
```

Observe que antes de iniciar o laço For...Each, desativo os avisos de Alerta - Application.DisplayAlerts = False. Se os avisos não forem desativados, quando o método Delete for chamado, será exibida uma mensagem de aviso e o usuário terá que clicar em OK para seguir com a exclusão.

O Método Add da Coleção Worksheets:

Este método é utilizado para criar uma nova planilha. A nova planilha se torna a planilha ativa. Este método retorna um objeto Worksheet, o qual aponta para a planilha que acabou de ser criada.

Sintaxe:

```
expressão.Add(Before, After, Count, Type)
```

- ▶ **expressão:** Obrigatória. Uma expressão que retorne um objeto Worksheets.
- ▶ **Before:** Variant opcional. Um objeto que especifica a planilha antes da qual a nova planilha será adicionada.
- ▶ **After:** Variant opcional. Um objeto que especifica a planilha após a qual a nova planilha é adicionada.
- ▶ **Count:** Variant opcional. O número de planilhas a serem adicionadas. O valor padrão é um.
- ▶ **Type:** Variant opcional. O tipo da planilha. Pode ser uma das seguintes constantes: xlWorksheet, xlExcel4MacroSheet ou xlExcel4IntlMacroSheet. O valor padrão é xlWorksheet.

Nota: Se Before e After forem omitidos, a nova planilha será inserida antes da planilha ativa.

Exemplos do método Add (coleção Worksheets):

Este exemplo cria uma nova planilha e a insere antes da planilha ativa.

```
ActiveWorkbook.Worksheets.Add
```

Este exemplo adiciona uma nova planilha após a última planilha da pasta de trabalho ativa.

```
Worksheets.Add.Move after:=Worksheets(Worksheets.Count)
```

Worksheets.Count retorna o índice da última planilha. Este número é passado como parâmetro para Worksheets, ou seja, uma referência para a última planilha da pasta de trabalho ativa.

O exemplo a seguir cria uma nova planilha e depois define o nome da planilha como Estoque. Para definir o nome da planilha uso a propriedade Name:

```
Set NovaPlanilha = Worksheets.Add  
NovaPlanilha.Name = "Estoque"
```

Lição 05: Mais Métodos e Propriedades dos Objetos Workbook e Worksheets

Nesta lição vamos estudar mais alguns métodos e propriedades da coleção Worksheets.

O Método Copy:

O Método Copy, da coleção Worksheets, é utilizado para copiar uma planilha para um outro lugar na pasta de trabalho.

Sintaxe:

```
expressão.Copy(Before, After)
```

► **Before:** Variant opcional. A planilha antes da qual a planilha copiada será colocada. Você não pode especificar Before se especificar After.

► **After:** Variant opcional. A planilha após a qual a planilha copiada será colocada. Você não pode especificar After se Before for especificado.

Importante: Se você não especificar Before ou After, o Microsoft Excel criará uma nova pasta de trabalho contendo a planilha copiada.

O exemplo a seguir copia a planilha Plan2, colocando a cópia depois de Sheet3.

```
Worksheets("Sheet1").Copy after := Worksheets("Sheet3")
```

O Método Delete:

Este método é utilizado para excluir uma planilha, da coleção de planilhas do objeto Worksheet.

Sintaxe:

```
expressão.Delete
```

► **expressão:** Obrigatória. Uma expressão que retorne um objeto da lista Relativo a.

O exemplo a seguir exclui a planilha Plan3 da pasta de trabalho ativa sem exibir a caixa de diálogo de confirmação. Para desativar as mensagens de aviso, utilizo o comando Application.DisplayAlerts = True. Se não for utilizado este comando, será exibida uma mensagem de confirmação, para que o usuário clique em OK para confirmar a exclusão da planilha:

```
Application.DisplayAlerts = False  
Worksheets("Plan3").Delete  
Application.DisplayAlerts = True
```


O Método FillAcrossSheets:

Este método copia um intervalo de células, para a mesma área em todas as outras planilhas de uma coleção. Por exemplo, você pode copiar o conteúdo da faixa A1:E50, da planilha Plan1, para todas as demais planilhas da pasta de trabalho. Os dados serão copiados para a faixa A1:E50 (mesma faixa original) em todas as demais planilhas.

Sintaxe:

```
expressão.FillAcrossSheets(Range, Type)
```

- ▶ **expressão:** Obrigatória. Uma expressão que retorne um objeto Sheets ou Worksheets.
- ▶ **Range:** Range obrigatório. O intervalo a preencher em todas as planilhas da coleção. O intervalo precisa ser de uma planilha dentro da coleção.
- ▶ **Type:** Variant opcional. Especifica como copiar o intervalo. Pode ser uma das seguintes constantes: xlFillWithAll, xlFillWithContents ou xlFillWithFormulas. O valor padrão é xlFillWithAll.

O exemplo a seguir preenche o intervalo A1:E50 das planilhas Plan1, Plan4 e Plan6 com o conteúdo do mesmo intervalo em Plan1.

```
x = Array("Plan1", "Plan4", "Plan6")  
Sheets(x).FillAcrossSheets Worksheets("Plan1").Range("A1:E50")
```

Observe que inicialmente crio um Array (um vetor), para conter a lista de planilhas a ser passada como parâmetro para Sheets. Em seguida chamo o método FillAcrossSheets e passo, como parâmetro para este método, a faixa onde estão os dados. Os dados da faixa especificada, serão copiados para a mesma faixa, nas planilhas contidas no Array x.

O Método Move:

Este método é utilizado para mover uma planilha para um outro lugar na pasta de trabalho.

Sintaxe:

```
expressão.Move(Before, After)
```

- ▶ **expressão:** Obrigatória. Uma expressão que retorne um objeto do tipo WorkSheet.
- ▶ **Before:** Variant opcional. A planilha antes da qual a planilha movida será colocada. Você não pode especificar Before se After for especificado.
- ▶ **After:** Variant opcional. A planilha após a qual a planilha movida será colocada. Você não pode especificar After se Before for especificado.

Nota: Se você não especificar Before ou After, o Microsoft Excel criará uma nova pasta de trabalho contendo a planilha movida.

O exemplo a seguir move a planilha Plan1 para depois da planilha Plan5, na pasta de trabalho ativa.

```
Worksheets("Plan1").Move after:=Worksheets("Plan5")
```

O Método PrintOut:

Este método é utilizado para Imprime um objeto, passado como parâmetro.

Sintaxe:

```
expressão.PrintOut(From, To, Copies, Preview, ActivePrinter,  
PrintToFile, Collate, PrToFileName)
```

- ▶ **expressão:** Obrigatória. Uma expressão que retorne um uma referência ao objeto a ser impresso.
- ▶ **From:** Variant opcional. O número da página pela qual começar a impressão. Se esse argumento for omitido, a impressão começará pela primeira página.
- ▶ **To:** Variant opcional. O número da última página a imprimir. Se esse argumento for omitido, todas as páginas serão impressas.
- ▶ **Copies:** Variant opcional. O número de cópias a imprimir. Se esse argumento for omitido, será impressa uma cópia.
- ▶ **Preview:** Variant opcional. True para que o Microsoft Excel invoque a visualização de impressão antes de imprimir o objeto. False (ou omitido) para imprimir o objeto imediatamente.
- ▶ **ActivePrinter:** Variant opcional. Define o nome da impressora ativa.
- ▶ **PrintToFile:** Variant opcional. True para imprimir para um arquivo. Se o parâmetro PrToFileName não for especificado, o Microsoft Excel solicitará ao usuário que digite o nome do arquivo de saída.
- ▶ **Collate:** Variant opcional. True para agrupar múltiplas cópias.
- ▶ **PrToFileName:** Variant opcional. Se PrintToFile for definido como True, esse argumento especificará o nome do arquivo para o qual você deseja imprimir.

O exemplo a seguir, imprime a planilha ativa:

```
ActiveSheet.PrintOut
```

Lição 06: Mais Métodos e Propriedades dos Objetos Workbook e Worksheets

Nesta lição vamos estudar mais alguns métodos e propriedades da coleção Worksheets.

O Método PrintPreview:

Este método é utilizado para mostrar uma visualização do objeto tal como ele aparece quando impresso. É equivalente ao comando Arquivo -> Visualizar impressão.

Sintaxe:

```
expressão.PrintPreview
```

- **expressão:** Obrigatória. Uma expressão que retorne um objeto a ser visualizado no modo de impressão.

O exemplo a seguir exibe a planilha Plan1 no modo de Visualização de Impressão:

```
Worksheets("Plan1").PrintPreview
```

O Método Select:

É utilizado para selecionar o objeto.

Sintaxe:

```
expressão.Select(Replace)
```

- **expressão:** Obrigatória. Uma expressão que retorne um objeto a ser selecionado.
- **Replace:** Variant opcional (usado somente com planilhas). True para substituir a seleção atual pelo objeto especificado. False para estender a seleção atual para incluir qualquer objeto anteriormente selecionado e o objeto especificado.

Nota: Para selecionar uma célula ou um intervalo de células, use o método Select. Para tornar uma única célula a célula ativa, use o método Activate.

O exemplo a seguir seleciona as células A1:E10 da planilha Plan1. Observe que antes de fazer a seleção, a planilha Plan1 é ativada, usando o método Activate, já descrito nas lições anteriores.

```
Worksheets("Plan1").Activate  
Range("A1:E10").Select
```

Propriedade Creator:

Esta propriedade retorna um número inteiro de 32 bits que indica o aplicativo em que esse objeto foi criado. Se o objeto foi criado no Microsoft Excel, essa propriedade retornará a sequência XCEL, que é equivalente ao número hexadecimal 5843454C. Long somente leitura.

Nota: A propriedade Creator foi projetada para ser usada no Microsoft Excel para Macintosh, onde cada aplicativo possui um código de criador de quatro caracteres. Por exemplo, o Microsoft Excel possui o código de criador XCEL.

Este exemplo exibe uma mensagem sobre o criador do objeto associado a variável myObject, a qual está associada com a pasta de trabalho atual – ActiveWorkbook:

```
Set myObject = ActiveWorkbook
If myObject.Creator = &h5843454c Then
    MsgBox "Este é um objeto do Microsoft Excel!!!"
Else
    MsgBox "Este não é um objeto do Microsoft Excel"
End If
```

Propriedade Item:

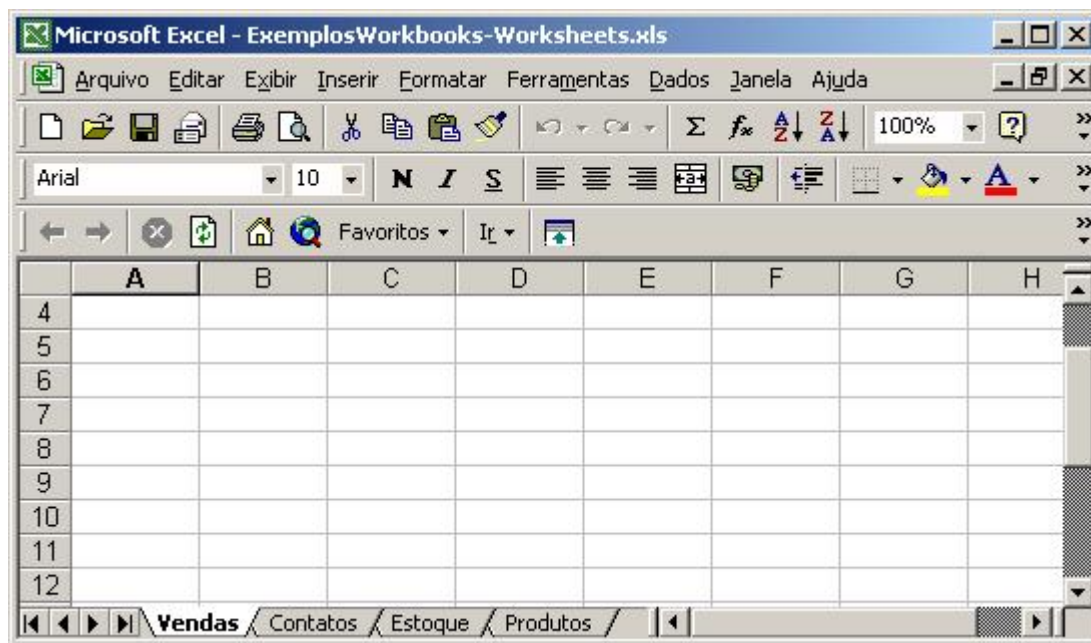
Esta propriedade retorna um único objeto Worksheet de uma coleção Worksheets.

Sintaxe:

`expressão.Item(Index)`

- ▶ **expressão:** Obrigatória. Uma expressão que retorne um objeto Worksheets.
- ▶ **Index:** Variant obrigatória. O nome ou número de índice da planilha. Se for o nome, deve ser informado entre aspas.

Considere a pasta de Trabalho da Figura a seguir, onde temos quatro planilhas: Vendas, Contatos, Estoque e Produtos:



A seguir alguns exemplos de utilização da propriedade Item.

Autor: Júlio Cesar Fabris Battisti

Site: www.juliobattisti.com.br

Confira também o livro: "Windows Server 2003 – Curso Completo, 1568 páginas"

Página 315 de 527

O comando a seguir retorna Vendas:

```
Worksheets.Item(1).Name
```

O comando a seguir também retorna Vendas:

```
Worksheets.Item("Vendas").Name
```

O comando a seguir retorna Estoque. O índice 3 significa a terceira planilha, da esquerda para a direita:

```
Worksheets.Item(3).Name
```

O comando a seguir também retorna Estoque:

```
Worksheets.Item("Estoque").Name
```

A Propriedade Visible:

Esta propriedade se aplica a vários objetos do Excel. Ela é True se o objeto está visível. Para um gráfico ou planilha, essa propriedade pode ser definida como xlVeryHidden. Isso oculta o objeto para que a única maneira de você torná-lo novamente visível seja definindo essa propriedade como True (o usuário não pode tornar o objeto visível). É do tipo Boolean ou Long de leitura e gravação.

Nota: A propriedade Visible para um item de tabela dinâmica é True quando o item está atualmente visível na tabela.

Obs: Se você definir a propriedade Visible de um nome como False, o nome não aparecerá na caixa de diálogo Definir nome.

O exemplo a seguir oculta a planilha Plan1. Para isso, ele atribui o valor False, à propriedade Visible:

```
Worksheets("Plan1").Visible = False
```

O exemplo a seguir torna Plan1 novamente visível:

```
Worksheets("Plan1").Visible = True
```

Este exemplo torna visível todas as planilhas da pasta de trabalho ativa.

```
For Each sh In Sheets  
    sh.Visible = True  
Next sh
```

A partir da próxima lição, você estudará, em detalhes, os principais métodos e propriedades do objeto Worksheet. Muitos dos quais já foram brevemente apresentados em exemplos anteriores e serão detalhados a partir da próxima lição.

Lição 07: O Objeto Worksheet – Métodos e Propriedades – Parte 1

A partir desta lição apresentarei, em mais detalhes, os principais métodos, propriedades e coleções do objeto Worksheet. Neste ponto quero salientar, novamente, a importância de entender como funciona um modelo hierárquico de objetos, como é o Modelo de Objetos do Excel, descrito no módulo 2. Por exemplo, mesmo que você não entenda de programação, o que não é o caso, pois já chegou até aqui neste curso, fica fácil concluir que um objeto Worksheet, que representa uma planilha, deve ter propriedades tais como UsedRange, que representa a faixa de células em uso, uma propriedade Name que contém o nome da planilha e assim por diante. O que faremos nesta e nas próximas lições é estudar as principais propriedades e métodos do objeto Worksheet.

Propriedade AutoFilter:

Esta propriedade retorna um objeto AutoFilter se o filtro (Dados -> Filtrar -> AutoFiltro) estiver ativado. Retorna Nothing se o filtro estiver desativado. Somente leitura.

Nota: Para criar um objeto AutoFilter para uma planilha, você deve ativar o autofiltro para um intervalo na planilha manualmente ou usando o método AutoFilter do objeto Range, conforme descrito e exemplificado no Módulo 3.

Propriedade AutoFilterMode:

Esta propriedade retorna True se as setas suspensas do AutoFiltro estão atualmente exibidas na planilha, ou seja, se o Auto Filtro está ativado. Essa propriedade é independente da propriedade FilterMode. É do tipo Boolean de leitura e gravação.

Importante: Essa propriedade retorna True se as setas suspensas estão atualmente exibidas. Você pode definir essa propriedade como False para remover as setas, mas não pode defini-la como True. Use o método AutoFilter para filtrar uma lista e exibir as setas suspensas. Não me perguntem o porquê de não ser possível definir esta propriedade como True, para exibir as setas de AutoFiltro. Parece um Bug do programa que o pessoal resolveu, avisando na documentação do Excel de que não é possível usar AutoFilterMode para exibir as setas.

O exemplo a seguir exibe o status atual da propriedade AutoFilterMode, na planilha Plan1:

```
If Worksheets("Plan1").AutoFilterMode Then
    isOn = "On"
Else
    isOn = "Off"
End If
MsgBox "AutoFilterMode is " & isOn
```

A Propriedade Cells:

Esta propriedade Retorna um objeto Range representando todas as células da planilha (não apenas as células atualmente em uso). Somente leitura.

Nota: O uso dessa propriedade sem um qualificador de objeto retorna um objeto Range representando todas as células da planilha ativa. Vamos considerar alguns exemplos para entender o uso desta propriedade. Veremos exemplos de uso da propriedade Cells de diversos objetos. Por exemplo, o objeto Range também tem uma propriedade Cells.

Exemplo 01: Este exemplo define o tamanho da fonte para a célula C5, da planilha Plan1 como 14 pontos.

```
Worksheets("Plan1").Cells(5, 3).Font.Size = 14
```

Observe que o primeiro índice – 5 no nosso exemplo, significa o número da linha. O segundo índice – 3 no nosso exemplo, significa o número da coluna. Com isso, Cells(5,3) significa quinta linha – terceira coluna, ou seja C5:

Exemplo 02: Este exemplo limpa a fórmula na célula um (A1) da planilha Plan1:

```
Worksheets("Plan1").Cells(1).ClearContents
```

Exemplo 03: Este exemplo define a fonte e o tamanho da fonte para todas as células da planilha Plan1, como fonte Arial e tamanho de 10 pontos:

```
Worksheets("Plan1").Cells.Font.Name = "Arial"  
Worksheets("Plan1").Cells.Font.Size = 10
```

Exemplo 04: Este exemplo percorre as células A1:J4 da planilha Plan1. Se uma célula contiver um valor menor que 0,001, o exemplo substituirá esse valor por 0 (zero). Observe que o primeiro índice está na variável `rwIndex` e varia de 1 a 4, ou seja, da linha 1 até a linha 4. O segundo índice, passado como parâmetro para Cells, está na variável `colIndex`, a qual varia de 1 a 10, ou seja de A até J. Este é um exemplo de como usar os índices e dois laços `for`, para percorrer um intervalo de células. Os comandos dentro do laço atuam sobre as células do intervalo, com base no resultado de um ou mais testes. Você pode, facilmente, adaptar este exemplo, para resolver uma série de problemas práticos, que envolvem percorrer um intervalo de células, alterando valores com base no resultado de um ou mais testes, em cada célula do intervalo.

```
For rwIndex = 1 to 4  
  For colIndex = 1 to 10  
    If Worksheets("Plan1").Cells(rwIndex, colIndex).Value < .001 Then  
      Worksheets("Plan1").Cells(rwIndex, colIndex).Value = 0  
    End If  
  Next colIndex  
Next rwIndex
```

Exemplo 05: Este exemplo define o estilo da fonte para as células A1:C5 da planilha Plan1 como Itálico.

```
Worksheets("Plan1").Activate  
Range(Cells(1, 1), Cells(5, 3)).Font.Italic = True
```


Cells(1,1) retorna A1 e Cells(5,3) retorna C5. Então, na prática, Range(Cells(1,1),Cells(5,3)) é o mesmo que escrever: Range("A1:C5").

Exemplo 06: Este exemplo varre uma coluna de dados chamada "myRange". Se uma célula tiver o mesmo valor que a célula imediatamente acima, o exemplo exibirá o endereço da célula que contém os dados duplicados.

```
Set r = Range("myRange")
For n = 1 To r.Rows.Count
    If r.Cells(n, 1) = r.Cells(n + 1, 1) Then
        MsgBox "Dados duplicados em " & r.Cells(n + 1, 1).Address
    End If
Next n
```

Este exemplo é realmente interessante. Primeiro mostra como criar um objeto Range, passando como parâmetro para Range o nome de uma faixa de células da planilha. Claro que este nome deve ter sido criado previamente, ou via código ou diretamente na planilha. Depois, uso um laço For, para percorrer todas as linhas da faixa de células. O número de linhas é obtido, usando r.Rows.Count, ou seja, a propriedade Count, do objeto Rows. Em seguida uso Cells, variando apenas o primeiro índice, o índice da linha – n e n+1 -, para definir se o valor de uma célula é igual ao valor da célula na linha de baixo. Se for, retorno o endereço da célula onde houve a segunda ocorrência do valor. Este é outro exemplo que pode ser facilmente adaptado para uso em problemas do dia-a-dia.

A Propriedade CircularReference:

Esta propriedade retorna um objeto Range representando o intervalo contendo a primeira referência circular da planilha, ou retorna Nothing se não houver referência circular na planilha. A referência circular precisa ser removida antes de se proceder o cálculo. Somente leitura. Uma referência circular é quando uma célula contém uma fórmula que faz referência a própria célula. Não são permitidas referências circulares no Excel.

Este exemplo seleciona a primeira célula com uma referência circular, na planilha Plan1:

```
Worksheets("Plan1").CircularReference.Select
```

A Propriedade Columns:

Retorna um objeto Range representando todas as colunas da planilha especificada. Somente leitura. O uso dessa propriedade sem um qualificador de objeto equivale a usar ActiveSheet.Columns. O próximo exemplo formata a fonte da coluna um (coluna A) em Plan1 como negrito:

```
Worksheets("Plan1").Columns(1).Font.Bold = True
```

O próximo exemplo define como 0 (zero) o valor de todas as células na coluna um no intervalo chamado "myRange":

```
Range("myRange").Columns(1).Value = 0
```

Lição 08: O Objeto WorkSheet – Métodos e Propriedades – Parte 2

Nesta lição continuaremos o estudo das principais propriedades e métodos do objeto Worksheet.

A Propriedade DisplayPageBreaks:

True se quebras de página (tanto automáticas quanto manuais) são exibidas na planilha especificada. Boolean de leitura e gravação.

Nota: Você não pode definir essa propriedade se não tiver uma impressora instalada.

O exemplo a seguir faz as quebras de página da planilha Plan1 serem exibidas:

```
Worksheets("Plan1").DisplayPageBreaks = True
```

A Propriedade EnableAutoFilter:

Retorna True se as setas do AutoFiltro estão ativadas quando a proteção está ativada apenas para a interface do usuário. Boolean de leitura e gravação.

Esta propriedade se aplica a cada planilha, não sendo salva com a planilha ou sessão.

O próximo exemplo ativa as setas do AutoFiltro em uma planilha protegida.

```
ActiveSheet.EnableAutoFilter = True  
ActiveSheet.Protect contents:=True, userInterfaceOnly:=True
```

A Propriedade EnableCalculation:

Retorna True se o Microsoft Excel recalcula automaticamente a planilha quando necessário. (False se o Excel não recalcula a planilha). Boolean de leitura e gravação.

Quando o valor dessa propriedade for False, você não poderá solicitar um recálculo. Quando você altera o valor de False para True, o Excel recalcula a planilha.

Este exemplo faz o Microsoft Excel não recalcular automaticamente a planilha um (planilha bem da esquerda):

```
Worksheets(1).EnableCalculation = False
```

A Propriedade EnableSelection:

Esta propriedade é utilizada para retornar ou definir o que pode ser selecionado na planilha. Pode ser uma das seguintes constantes: xlNoRestrictions, xlNoSelection ou xlUnlockedCells. É do tipo Long, de leitura e gravação.

Essa propriedade só funciona quando a planilha está protegida: `xlNoSelection` impede qualquer seleção na planilha, `xlUnlockedCells` permite que apenas as células cuja propriedade `Locked` seja `False` sejam selecionadas e `xlNoRestrictions` permite que qualquer célula seja selecionada.

O próximo exemplo define a planilha um de modo que nada possa ser selecionado nela:

```
Worksheets(1).EnableSelection = xlNoSelection  
Worksheets(1).Protect Contents:=True, UserInterfaceOnly:=True
```

A Propriedade Next:

Esta propriedade retorna um objeto `Chart`, `Range` ou `Worksheet` representando a próxima planilha ou célula. Somente leitura.

Se o objeto for um intervalo, essa propriedade emulará a tecla `TAB`, embora a propriedade retorne a próxima célula sem selecioná-la.

Em uma planilha protegida, essa propriedade retorna a próxima célula desprotegida. Em uma planilha desprotegida, essa propriedade sempre retorna a célula imediatamente à direita da célula especificada.

O próximo exemplo seleciona a próxima célula desprotegida na planilha `Plan1`. Se `Plan1` estiver desprotegida, será selecionada a célula imediatamente à direita da célula ativa.

```
Worksheets("Plan1").Activate  
ActiveCell.Next.Select
```

A Propriedade PageSetup:

Retorna um objeto `PageSetup` (veja descrição a seguir), contendo todas as definições de configuração de página para o objeto especificado. Somente leitura.

Um objeto `PageSetup` representa a descrição das configurações de página (margens, orientação, tamanho, etc). O objeto `PageSetup` contém todos os atributos de configuração de página (margem esquerda, margem inferior, tamanho do papel, etc.) como propriedades.

Utilizamos a propriedade `PageSetup`, do objeto `Worksheet`, para retornar um objeto `PageSetup`. O exemplo seguinte define a orientação como modo paisagem e, em seguida, imprime a planilha.

```
Worksheets("Plan1").PageSetup.Orientation = xlLandscape  
Worksheets("Plan1").PrintOut
```

O exemplo seguinte define todas as margens para a planilha um – planilha bem da esquerda:

```
Worksheets(1).PageSetup.LeftMargin = Application.InchesToPoints(0.5)  
Worksheets(1).PageSetup.RightMargin =  
Application.InchesToPoints(0.75)  
Worksheets(1).PageSetup.TopMargin = Application.InchesToPoints(1.5)
```

```
Worksheets(1).PageSetup.BottomMargin = Application.InchesToPoints(1)
Worksheets(1).PageSetup.HeaderMargin =
Application.InchesToPoints(0.5)
Worksheets(1).PageSetup.FooterMargin =
Application.InchesToPoints(0.5)
```

O próximo exemplo define o texto de cabeçalho central para o gráfico Graph1.

```
Charts("Graph1").PageSetup.CenterHeader = "Vendas Anuais - 2003"
```

A Propriedade Previous:

Esta propriedade retorna um objeto Chart, Range ou Worksheet representando a célula ou planilha anterior. Somente leitura.

Quando o objeto é um intervalo, essa propriedade emula o pressionamento de SHIFT+TAB; entretanto, ao contrário da combinação de teclas, a propriedade retorna a célula anterior sem selecioná-la.

Em uma planilha protegida, essa propriedade retorna a célula desprotegida anterior. Em uma planilha desprotegida, essa propriedade sempre retorna a célula imediatamente à esquerda da célula especificada.

O próximo exemplo seleciona a célula desprotegida anterior na planilha Plan1. Se Plan1 estiver desprotegida, essa célula será a imediatamente à esquerda da célula ativa.

```
Worksheets("Plan1").Activate
ActiveCell.Previous.Select
```

A Propriedade ProtectContents:

Esta propriedade retorna True se o conteúdo da planilha está protegido. Em um gráfico, isso protege o gráfico inteiro. Em uma planilha, isso protege as células individuais. É do tipo Boolean e é somente leitura.

Este exemplo exibe uma caixa de mensagem se o conteúdo da planilha Plan1 estiver protegido:

```
If Worksheets("Plan1").ProtectContents = True Then
    MsgBox "O Conteúdo da Planilha Está Protegido!!!"
End If
```

Lição 09: O Objeto Worksheet – Métodos e Propriedades – Parte 3

Nesta lição será a vez de estudar os principais métodos do objeto Worksheet.

O Método Activate:

Este método, além do objeto Worksheet, está presente em diversos outros objetos, conforme descrito na tabela a seguir. No caso do objeto Worksheet, o método Activate é utilizado para ativar uma determinada planilha, ou seja, é equivalente a clicar na guia com o nome da planilha, na parte de baixo da janela do Excel.

Objetos aos quais se aplica o método Activate:

Objeto	Descrição
Chart, ChartObject	Torna este gráfico ativo.
Worksheet	Torna esta a planilha ativa. Equivalente a clicar na guia da planilha.
OLEObject	Ativa o objeto.
Pane	Ativa o painel. Se o painel não estiver na janela ativa, a janela à qual esse painel pertence também será ativada. Você não pode ativar um painel congelado.
Range	Ativa uma única célula, a qual precisa estar dentro da seleção atual. Para selecionar um intervalo de células, use o método Select.
Window	Traz a janela para a frente na ordem z. Isso não causa a execução de nenhuma macro Ativar_auto ou Desativar_auto que possa estar anexada à pasta de trabalho (use o método RunAutoMacros para executar essas macros).
Workbook	Ativa a primeira janela associada à pasta de trabalho. Isso não causa a execução de nenhuma macro Ativar_auto ou Desativar_auto que possa estar anexada à pasta de trabalho (use o método RunAutoMacros para executar essas macros).

Sintaxe:

`expressão.Activate`

- **expressão:** Obrigatória. Uma expressão que retorne um objeto de um dos tipos indicados na tabela anterior.

O próximo exemplo ativa a planilha Plan1:

`Worksheets("Plan1").Activate`

Este exemplo seleciona as células A1:C3 da planilha Plan1 e, em seguida, faz de B2 a célula ativa.

```
Worksheets("Plan1").Activate  
Range("A1:C3").Select  
Range("B2").Activate
```

O exemplo a seguir ativa a pasta Vendas.xls. Se Vendas.xls tiver várias janelas, o exemplo ativará a primeira janela, representada por Vendas.xls:1.

```
Workbooks("Vendas.xls").Activate
```

O Método Calculate:

O método Calculate também se aplica a outros objetos, além do objeto Worksheet. Este método, dependendo do objeto ao qual está sendo aplicado, calcula todas as pastas de trabalho abertas, uma planilha específica em uma pasta de trabalho ou um intervalo especificado de células em uma planilha, como se vê na tabela seguinte.

Para calcular	Siga este exemplo
Todas as pastas de trabalho abertas	Application.Calculate (ou apenas Calculate)
Uma planilha específica	Worksheets(1).Calculate
Um intervalo especificado	Worksheets(1).Rows(2).Calculate

O exemplo a seguir calcula as fórmulas das colunas A, B e C no intervalo utilizado, na planilha Plan1:

```
Worksheets("Plan1").UsedRange.Columns("A:C").Calculate
```

O Método ChartObjects:

Este método é utilizado para retornar um objeto representando um único gráfico incorporado (um objeto ChartObject, Sintaxe 1 a seguir) ou uma coleção de todos os gráficos incorporados (um objeto ChartObjects, Sintaxe 2 a seguir), na planilha:

Sintaxe 1:

```
expressão.ChartObjects(Index)
```

Sintaxe 2:

```
expressão.ChartObjects
```

- **expressão:** Obrigatória. Uma expressão que retorne um objeto do tipo Worksheet. Se você especificar um objeto Chart, ele terá que ser uma folha de gráfico (ele não pode ser um gráfico incorporado).
- **Index:** Variant opcional. O nome ou número do gráfico. Esse argumento pode ser uma matriz/array, para especificar mais de um gráfico.

Nota: Este método não é equivalente à propriedade Charts. O método retorna gráficos incorporados; a propriedade Charts retorna folhas de gráfico. Use a propriedade Chart para retornar o objeto Chart de um gráfico incorporado.

O próximo exemplo adiciona um título ao primeiro gráfico incorporado na planilha Plan1, isto é, o primeiro gráfico que foi criado na planilha Plan1:

```
Worksheets("Plan1").ChartObjects(1).Chart.HasTitle = True  
Worksheets("Plan1").ChartObjects(1).Chart.ChartTitle.Text = "Vendas de 1998"
```

O exemplo a seguir cria uma nova sequência no primeiro gráfico incorporado na planilha Plan1. A origem de dados para a nova sequência é o intervalo B1:B10 na planilha Plan1:

```
Worksheets("Plan1").ChartObjects(1).Activate  
ActiveChart.SeriesCollection.Add source:=Worksheets("Plan1").Range("B1:B10")
```

Este exemplo limpa a formatação do primeiro gráfico incorporado na planilha Plan1:

```
Worksheets("Plan1").ChartObjects(1).Chart.ChartArea.ClearFormats
```

O Método PasteSpecial:

Este método é utilizado para colar o conteúdo da Área de transferência na planilha, usando um formato especificado. Use este método para colar dados de outros aplicativos ou para colar dados em um formato específico.

Sintaxe:

```
expressão.PasteSpecial(Format, Link, DisplayAsIcon, IconFileName, IconIndex, IconLabel)
```

- ▶ **expressão:** Obrigatória. Uma expressão que retorna um objeto DialogSheet ou Worksheet.
- ▶ **Format:** Variant opcional. Uma sequência de caracteres especificando o formato dos dados da Área de transferência
- ▶ **Link:** Variant opcional. True para estabelecer um vínculo com a origem dos dados colados. Quando os dados de origem são inadequados para vinculação ou quando o aplicativo de origem não suporta vinculação, esse parâmetro é ignorado. O valor padrão é False.
- ▶ **DisplayAsIcon:** Variant opcional. True para exibir como um ícone o que foi colado. Ao dar um clique duplo no ícone, o respectivo conteúdo será carregado no aplicativo de origem e exibido. O valor padrão é False.
- ▶ **IconFileName:** Variant opcional. O nome do arquivo que contém o ícone a ser usado se DisplayAsIcon for True.

► **IconIndex:** Variant opcional. O número de índice do ícone dentro do arquivo de ícones, como por exemplo uma arquivo .DLL que contém ícones.

► **IconLabel:** Variant opcional. O rótulo de texto do ícone.

Importante: Você precisa selecionar o intervalo de destino antes de usar esse método.

Esse método pode modificar a seleção da planilha, dependendo do conteúdo da Área de transferência.

O próximo exemplo cola um objeto documento do Microsoft Word da Área de transferência para a célula D1, da planilha Plan1:

```
Worksheets("Plan1").Range("D1").Select  
ActiveSheet.PasteSpecial format:= "Microsoft Word 8.0 Document Object"
```

O exemplo a seguir cola o mesmo objeto documento do Microsoft Word e o exibe como um ícone.

```
Worksheets("Plan1").Range("F5").Select  
ActiveSheet.PasteSpecial _  
    Format:="Microsoft Word 8.0 Document Object", _  
    DisplayAsIcon:=True
```

Lição 10: O Objeto WorkSheet – Métodos e Propriedades – Parte 4

Nesta lição continuaremos o estudo dos principais Métodos do objeto Worksheet.

O Método Select:

Este método é utilizado para selecionar um objeto Worksheet.

Sintaxe:

```
expressão.Select(Replace)
```

- ▶ **expressão:** Obrigatória. Uma expressão que retorne um objeto do tipo Worksheet
- ▶ **Replace:** Variant opcional (usado somente com planilhas). True para substituir a seleção atual pelo objeto especificado. False para estender a seleção atual para incluir qualquer objeto anteriormente selecionado e o objeto especificado.

Nota: Para selecionar uma célula ou um intervalo de células, use o método Select. Para tornar uma única célula a célula ativa, use o método Activate.

O exemplo a seguir seleciona as células A1:E50 da planilha Plan1:

```
Worksheets("Plan1").Activate  
Range("A1:E50").Select
```

O Método SetBackgroundPicture:

Este método é utilizado para definir um gráfico de segundo plano para uma planilha ou gráfico.

Sintaxe:

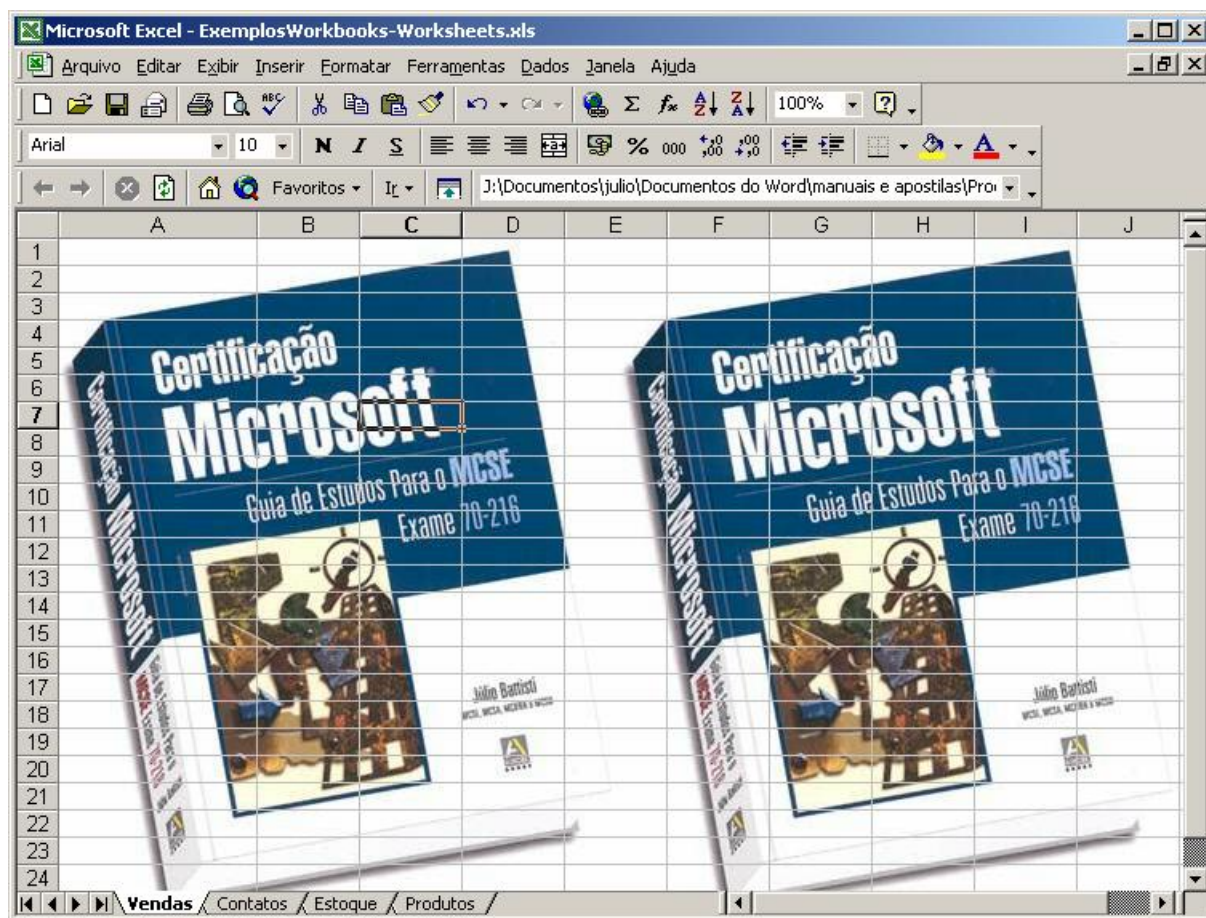
```
expressão.SetBackgroundPicture(FileName)
```

- ▶ **expressão:** Obrigatória. Uma expressão que retorne um objeto Worksheet ou Chart.
- ▶ **FileName** String obrigatória. O nome do arquivo gráfico.

Este exemplo define o gráfico de segundo plano para a planilha Vendas.

```
Worksheets("Vendas").SetBackgroundPicture "c:\dados\NovoLivro.jpg"
```

Na Figura a seguir, é indicado o resultado deste comando, onde o arquivo C:\dados\NovoLivro.jpg foi definido como figura de segundo-plano, da planilha:



O Método ShowAllData:

Este método é utilizado para tornar visíveis todas as linhas da lista atualmente filtradas. Se o AutoFiltro estiver em uso, esse método mudará a opção selecionada nas caixas de combinação para "Tudo".

Sintaxe:

`expressão.ShowAllData`

- **expressão:** Obrigatória. Uma expressão que retorne um objeto Worksheet.

Este exemplo torna visíveis todos os dados da planilha Plan1. O exemplo deve ser executado em uma planilha que contenha uma lista que você tenha filtrado usando o comando AutoFiltro.

`Worksheets("Plan1").ShowAllData`

O Método Unprotect:

Este método remove a proteção de uma planilha ou pasta de trabalho. Este método não tem efeito se a planilha ou pasta de trabalho não estiver protegida.

Sintaxe:

`expressão.Unprotect(Senha)`

- ▶ **expressão:** Obrigatória. Uma expressão que retorne um objeto Chart, Workbook ou Worksheet.
- ▶ **Password:** Variant opcional. Uma sequência que define a senha (que pode usar maiúsculas e minúsculas) a ser usada para desproteger a planilha ou pasta de trabalho. Se a planilha ou pasta de trabalho não estiver protegida com uma senha, esse argumento será ignorado. Se você omitir esse argumento para uma planilha que esteja protegida com uma senha, você será solicitado a digitar a senha. Se você omitir esse argumento para uma pasta de trabalho que esteja protegida com uma senha, o método falhará.

Se você esquecer a senha, você não poderá desproteger a planilha ou pasta de trabalho. É uma boa idéia guardar em lugar seguro uma lista de suas senhas e dos nomes dos documentos correspondentes.

Este exemplo remove a proteção da pasta de trabalho ativa.

`ActiveWorkbook.Unprotect`

O Método Paste:

Cola o conteúdo da Área de transferência na planilha.

Sintaxe:

`expressão.Paste(Destination, Link)`

- ▶ **expressão:** Obrigatória. Uma expressão que retorna um objeto Worksheet.
- ▶ **Destination:** Variant opcional. Um Objeto Range especificando onde o conteúdo da Área de transferência deve ser colado. Se esse argumento for omitido, a seleção atual será usada. Esse argumento só pode ser especificado se o conteúdo da Área de transferência puder ser colado em um intervalo. Se esse argumento for especificado, o argumento Link não poderá ser usado.
- ▶ **Link:** Variant opcional. True para estabelecer um vínculo com a origem dos dados colados. Quando esse argumento é especificado, o argumento Destination não pode ser usado. O valor padrão é False.

Se você não especificar o argumento Destination, terá que selecionar o intervalo de destino antes de usar esse método. Esse método pode modificar a seleção da planilha, dependendo do conteúdo da Área de transferência.

Este exemplo copia dados das células C1:C5 da planilha Plan1 para as células D1:D5 da planilha Plan1:

`Worksheets("Plan1").Range("C1:C5").Copy`
`ActiveSheet.Paste Destination:=Worksheets("Plan1").Range("D1:D5")`

Lição 11: Eventos – Conceitos e Definições

A partir desta lição vamos estudar um conceito muito importante e muito útil, quando tratamos com programação no Windows: O Conceito de **Eventos**.

O que é um Evento?

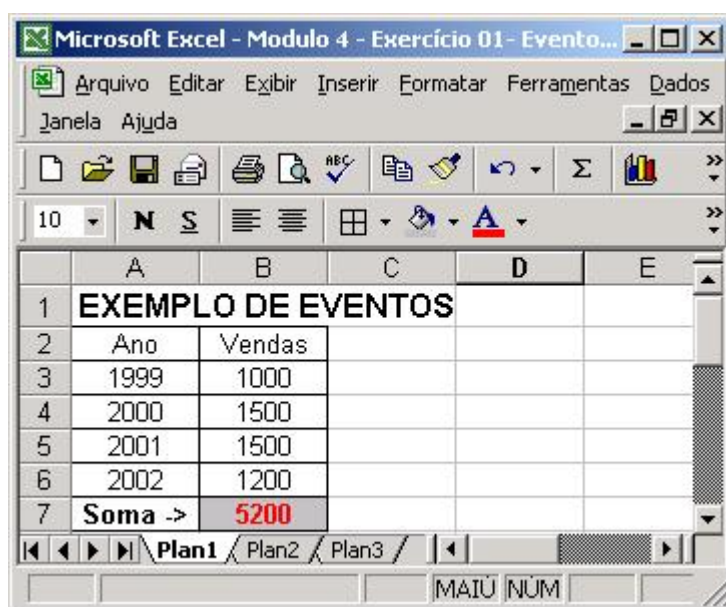
Podemos conceituar um evento como sendo ações realizadas pelo usuário, usando o teclado ou o mouse. Por exemplo, ao abrir uma pasta de trabalho, são disparados diversos eventos do objeto Application e do objeto Workbook. Outro exemplo, quando você clica em um botão de comando, ocorre o evento Ao Clicar do botão.

A próxima pergunta que você pode estar se fazendo é: **Mas na prática, para que servem os eventos??**

Na programação VBA no Excel (e em qualquer ferramenta de programação do Windows), é possível criar um procedimento de código, o qual é executado em resposta a um evento. Por exemplo, você pode criar um procedimento de código, associado ao evento Ao Clicar de um botão de comando. Quando o usuário clica no botão, o evento Ao Clicar é disparado e o procedimento associado ao evento é executado.

Com o uso de eventos e com a possibilidade de executar procedimentos, em resposta aos eventos, as possibilidades da programação VBA são muito enriquecidas, onde você pode criar aplicativos complexos, usando o Excel, onde grande parte da lógica de execução do aplicativo está contida em procedimentos, associados a eventos.

Cada elemento e/ou objeto do Excel, tem um conjunto de eventos associados. Neste módulo iremos estudar os principais eventos dos principais objetos do Excel. Para entender exatamente como funciona um exemplo, vamos iniciar por um exemplo extremamente simples. Para o nosso exemplo, considere a planilha com os dados indicados na Figura a seguir:



The screenshot shows a Microsoft Excel window titled "Microsoft Excel - Módulo 4 - Exercício 01 - Evento...". The spreadsheet has columns A through E and rows 1 through 7. The data is as follows:

	A	B	C	D	E
1	EXEMPLO DE EVENTOS				
2	Ano	Vendas			
3	1999	1000			
4	2000	1500			
5	2001	1500			
6	2002	1200			
7	Soma ->	5200			

The status bar at the bottom shows "Plan1 / Plan2 / Plan3" and "MAIU NUM".

Autor: Júlio Cesar Fabris Battisti

Site: www.juliobattisti.com.br

Confira também o livro: "Windows Server 2003 – Curso Completo, 1568 páginas"

Página 330 de 527

Vamos supor que o objetivo de vendas para os últimos quatro anos seja de 6500. Você quer criar um código VBA que seja executado sempre que a planilha Plan1 for ativada. Se o valor das vendas (valor da célula B7) for menor do que 6500, será exibida a mensagem: META DE VENDAS NÃO ATINGIDA e se o valor das vendas for maior ou igual a 6500, será exibida a mensagem META DE VENDAS ATINGIDA!!! PARABÉNS!!!.

Este exemplo embora extremamente simples, serve para ilustrar como funcionam os exemplos. Vamos criar um código associado ao evento Activate.

O evento Activate ocorre quando um objeto (Planilha, Gráfico, Folha de dados, etc.) torna-se a janela ativa. O evento Deactivate ocorre quando um objeto não é mais a janela ativa, ou seja, quando o objeto perde o foco.

Sintaxe:

```
Private Sub objeto_Activate()  
  
Private Sub objeto_Deactivate()
```

O espaço reservado objeto representa uma expressão de objeto que resulta em um objeto na lista.

Um objeto pode se tornar ativo utilizando-se o método Show no código VBA. O evento Activate pode ocorrer somente quando um objeto está visível. Um UserForm carregado com Load não é visível a menos que você utilize o método Show.

Nota: Para detalhes sobre UserForms consulte o Módulo 5.

Importante: Os eventos Activate e Deactivate ocorrem apenas quando você move o foco dentro de um aplicativo. Mover o foco de ou para um objeto em outro aplicativo não aciona nenhum dos eventos.

Para o nosso exemplo, vamos usar o evento Activate, da planilha Plan1. O código associado a este evento, será executado sempre que a planilha for ativada. Neste código testaremos o valor da célula B7 e, com base neste valor, vamos emitir diferentes mensagens. Vamos à implementação do exemplo proposto.

Exemplo: Para criar um procedimento em resposta ao evento Activate, siga os passos indicados a seguir:

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\Modulo 4 - Exercício 01- Eventos.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Nas opções que são exibidas do lado esquerdo da tela clique em Plan1 (Plan1) para selecionar esta opção.

5. Na lista Geral, selecione Worksheet e na lista ao lado selecione Activate. Pode ocorrer de o Editor do VBA ter criado duas declarações, uma para o evento Activate e uma para o Evento SelectionChange, conforme indicado a seguir:

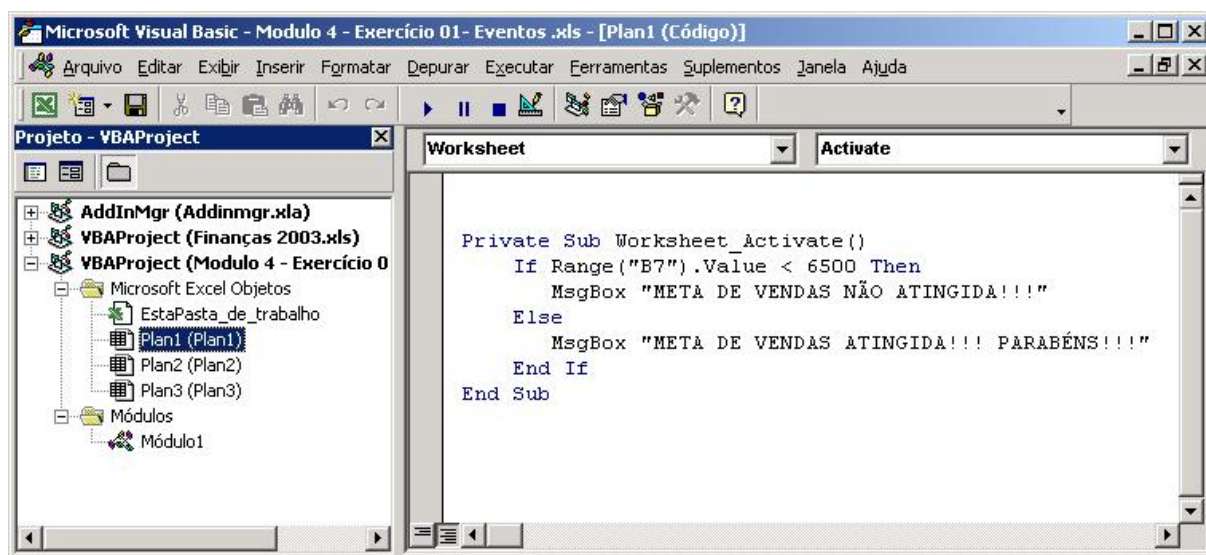
```
Private Sub Worksheet_Activate()  
  
End Sub  
  
Private Sub Worksheet_SelectionChange(ByVal Target As Range)  
  
End Sub
```

6. Exclua a declaração para o evento Worksheet_SelectionChange, mantendo apenas a declaração para o evento Worksheet_Activate

7. Para o procedimento do evento Worksheet_Activate, insira o código indicado a seguir:

```
If Range("B7").Value < 6500 Then  
    MsgBox "META DE VENDAS NÃO ATINGIDA!!!"  
Else  
    MsgBox "META DE VENDAS ATINGIDA!!! PARABÉNS!!!"  
End If
```

8. A sua janela deve estar conforme indicado na Figura a seguir:

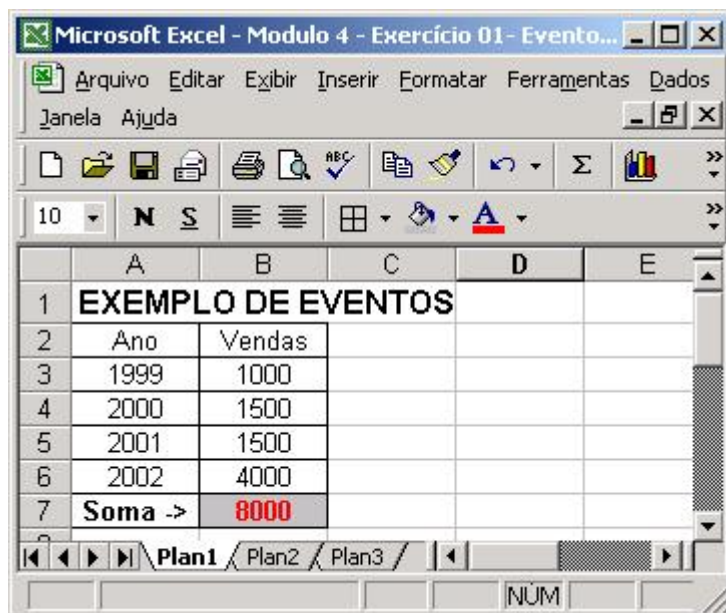


9. Observe o objeto Worksheet selecionado na primeira lista e o evento Activate, selecionado na segunda lista. Ou seja, o Evento Activate do Objeto Worksheet. Clique no botão (💾) para salvar as alterações e feche o Editor do VBA.

10. Agora vamos fazer com que o evento Activate da planilha Plan1 seja disparado. Clique em Plan2 para tirar o foco da planilha Plan1, agora clique de volta em Plan1 para colocar o foco em Plan1. Ao colocar o foco de volta em Plan1, será disparado o evento Activate. O código associado a este evento será executado e será exibida a mensagem indicada na Figura a seguir:



11. Clique em OK para fechar a mensagem. Esta mensagem foi exibida porque o valor da célula B7 é 5200, ou seja, um valor menor do que 6500. Agora vamos fazer uma alteração nas vendas do ano 2002. Vamos alterar as vendas deste ano para 4000, de tal maneira que o total de vendas fique em 8000, conforme indicado na Figura a seguir:



12. Agora vamos fazer com que o evento Activate da planilha Plan1 seja disparado novamente. Clique em Plan2 para tirar o foco da planilha Plan1, agora clique de volta em Plan1 para colocar o foco em Plan1. Ao colocar o foco de volta em Plan1, será disparado o evento Activate. O código associado a este evento será executado e será exibida a mensagem indicada na Figura a seguir:



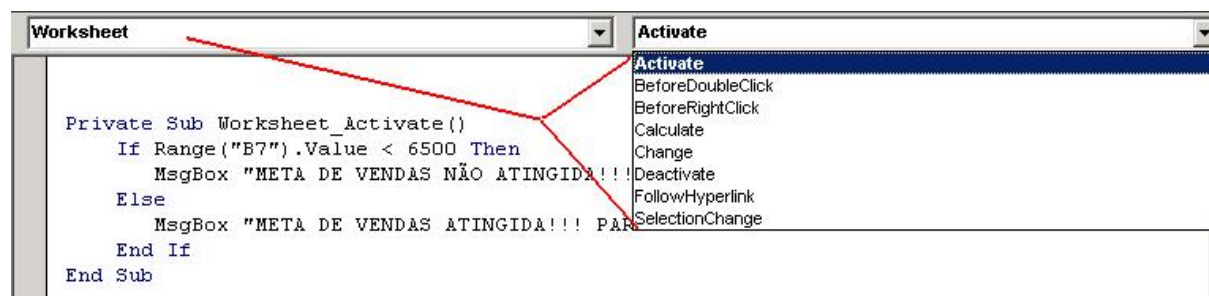
13. Observe que é emitida uma mensagem diferente, uma vez que o valor da célula B7 é maior do que 6500. Isso comprova que o nosso procedimento, associado ao evento Activate, está funcionando corretamente. Este foi um exemplo simples, mas que mostrou como funcionam os eventos no Excel.

14. Salve e feche a planilha Modulo 4 - Exercício 01- Eventos .xls.

Lição 12: Eventos – Eventos do Objeto Worksheet

Na lição anterior fiz uma apresentação do conceito de Eventos e apresentei um exemplo simples de uso do evento Activate, do objeto Worksheet. Nesta lição iniciaremos o estudo dos principais eventos do objeto Worksheet.

No exemplo da lição anterior, quando você selecionou Worksheet, na primeira lista, no ambiente de programação VBA, na segunda lista foram disponibilizados todos os eventos do objeto Worksheet, conforme ilustrado na Figura a seguir:



Vamos entender cada um destes eventos.

Eventos Activate e Deactivate:

O evento Activate ocorre quando um objeto torna-se a janela ativa. O evento Deactivate ocorre quando um objeto não é mais a janela ativa.

Sintaxe:

```
Private Sub object_Activate()  
  
Private Sub object_Deactivate()
```

O espaço reservado object representa uma expressão de objeto que resulta em um objeto na lista. No caso específico de uma planilha, uma expressão que faz referência a um objeto Worksheet.

Um objeto pode se tornar ativo utilizando-se o método Show no código.

Nota: O evento Activate pode ocorrer somente quando um objeto está visível. Um UserForm carregado com Load não é visível a menos que você utilize o método Show.

Importante: Os eventos Activate e Deactivate ocorrem apenas quando você move o foco dentro de um aplicativo. Mover o foco de ou para um objeto em outro aplicativo não aciona nenhum dos eventos.

A declaração para o evento Activate é a seguinte:

```
Private Sub Worksheet_Activate()  
End Sub
```

A declaração para o evento **Deactivate** é a seguinte:

```
Private Sub Worksheet_Deactivate()  
  
End Sub
```

Evento BeforeDoubleClick:

Este evento ocorre quando uma planilha ou gráfico incorporado é clicado duas vezes, antes da ação de duplo clique padrão.

Sintaxe 1:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
```

Sintaxe 2

```
Private Sub object_BeforeDoubleClick(ByVal ElementID As Long, ByVal Arg1 As Long,  
ByVal Arg2 As Long, Cancel As Boolean)
```

Argumentos deste evento:

- ▶ **objeto:** Um objeto do tipo Chart declarado com eventos em um módulo de classe.
- ▶ **Target:** A célula mais próxima do ponteiro do mouse quando o duplo clique ocorre.
- ▶ **Cancel:** False quando o evento ocorre. Quando o procedimento de evento define esse **argumento** como True, a ação de duplo clique padrão não é efetuada quando o procedimento termina.
- ▶ **ElementID:** O objeto clicado duas vezes. O Significado de Arg1 e Arg2 depende do valor de ElementID, conforme mostrado na tabela seguinte.

ElementID	Arg1	Arg2
xlChartArea	Nenhum	Nenhum
xlChartTitle	Nenhum	Nenhum
xlPlotArea	Nenhum	Nenhum
xlLegend	Nenhum	Nenhum
xlFloor	Nenhum	Nenhum
xlWalls	Nenhum	Nenhum
xlCorners	Nenhum	Nenhum
xlDataTable	Nenhum	Nenhum
xlSeries	SeriesIndex	PointIndex

xlDataLabel	SeriesIndex	PointIndex
xlTrendline	SeriesIndex	TrendLineIndex
xlErrorBars	SeriesIndex	Nenhum
xlXErrorBars	SeriesIndex	Nenhum
xlYErrorBars	SeriesIndex	Nenhum
xlLegendEntry	SeriesIndex	Nenhum
xlLegendKey	SeriesIndex	Nenhum
xlAxis	AxisIndex	AxisType
xlMajorGridlines	AxisIndex	AxisType
xlMinorGridlines	AxisIndex	AxisType
xlAxisTitle	AxisIndex	AxisType
xlDisplayUnitLabel	AxisIndex	AxisType
xlUpBars	GroupIndex	None
xlDownBars	GroupIndex	Nenhum
xlSeriesLines	GroupIndex	Nenhum
xlHiLoLines	GroupIndex	Nenhum
xlDropLines	GroupIndex	Nenhum
xlRadarAxisLabels	GroupIndex	Nenhum
xlShape	ShapeIndex	Nenhum
xlPivotChartDropZone	DropZoneType	Nenhum
xlPivotChartFieldButton	DropZoneType	PivotFieldIndex
xlNothing	Nenhum	Nenhum

A tabela seguinte descreve o significado dos argumentos:

Argumento	Descrição
SeriesIndex	Especifica o deslocamento dentro da coleção <u>Series</u> de uma determinada sequência.
PointIndex	Especifica o deslocamento dentro da coleção <u>Points</u> para um ponto específico dentro de uma sequência. O valor – 1 indica que todos os pontos de dados estão selecionados.
TrendlineIndex	Especifica o deslocamento dentro da coleção <u>Trendlines</u> para uma linha de tendência específica dentro de uma sequência.
AxisIndex	Especifica se o eixo é primário ou secundário. Pode ser uma das seguintes constantes XlAxisGroup: xlPrimary ou xlSecondary.
AxisType	Especifica o tipo de eixo. Pode ser uma das seguintes constantes XlAxisType: xlCategory, xlSeriesAxis ou xlValue.
GroupIndex	Especifica o deslocamento dentro da coleção <u>ChartGroups</u> para um grupo gráfico específico.
ShapeIndex	Especifica o deslocamento dentro da coleção <u>Shapes</u> para uma forma específica.

DropZoneType	Especifica o tipo de zona de soltura: campo de coluna, dados, página ou linha. Pode ser uma das seguintes constantes <code>XlPivotFieldOrientation</code> : <code>xlColumnField</code> , <code>xlDataField</code> , <code>xlPageField</code> ou <code>xlRowField</code> . As constantes de campo de linha e coluna especificam os campos de série e categoria, respectivamente.
PivotFieldIndex	Especifica o deslocamento dentro da coleção <code>PivotFields</code> de um campo de coluna específica (série), dados, página ou linha (categoria).

Notas: O método `DoubleClick` não faz esse evento ocorrer. Esse evento não ocorre quando o usuário clica duas vezes na borda de uma célula.

Na próxima lição estudaremos mais alguns eventos do objeto `Worksheet`.

Lição 13: Eventos – Eventos do Objeto Worksheet

Nesta lição finalizaremos o estudo dos principais eventos do objeto Worksheet.

Evento Calculate:

Objeto Chart: Ocorre após o gráfico plotar dados novos ou alterados.

Objeto Worksheet: Ocorre após a planilha ser recalculada automaticamente ou após o usuário pressionar a tecla F9, para forçar um recálculo manual da planilha..

Sintaxe:

```
Private Sub objeto_Calculate()
```

- ▶ **objeto:** Chart ou Worksheet. Para obter informações sobre uso de eventos com o objeto Chart, consulte Uso de eventos com o objeto Chart.

Exemplo do evento Calculate:

Este exemplo ajusta o tamanho das colunas de A a F sempre que a planilha é recalculada. Por exemplo, pode ser que após o recálculo, os valores aumentem, fazendo com que não seja possível exibi-los nas colunas. Nestas situações, o Excel exhibe apenas caracteres #####. Muitos usuários podem pensar que está havendo um erro, quando na verdade é só falta de espaço (largura) na coluna. O método AutoFit equivale a dar um clique duplo na divisória das colunas, ou seja, deixar a coluna do tamanho exato para exibir os dados de suas células.

```
Private Sub Worksheet_Calculate()  
    Columns("A:F").AutoFit  
End Sub
```

Evento Change:

Este evento ocorre quando as células da planilha são alteradas pelo usuário ou por um vínculo externo.

Sintaxe:

```
Private Sub Worksheet_Change(ByVal Target As Range)
```

- ▶ **Target** O intervalo alterado. Pode ser mais de uma célula.

Este evento não ocorre quando as células são alteradas durante um recálculo. Use o evento Calculate, descrito anteriormente, para interceptar um recálculo de planilha.

A exclusão de células não aciona esse evento.

Exemplo do evento Change:

Este exemplo altera para azul a cor das células alteradas.

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Target.Font.ColorIndex = 5
End Sub
```

Observe que o parâmetro target é uma referência a faixa de células, cujos valores forem alterados, alteração esta que disparou o evento Change.

Evento SelectionChange:

Este evento ocorre quando a seleção é alterada em uma planilha.

Sintaxe:

```
Private Sub Worksheet_SelectionChange(ByVal Destino As Excel.Range)
```

Destino O novo intervalo selecionado.

Exemplo do evento SelectionChange:

Este exemplo rola pela janela da pasta de trabalho até a seleção ficar no canto superior esquerdo da janela.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    ActiveWindow.ScrollRow = Target.Row
    ActiveWindow.ScrollColumn = Target.Column
End Sub
```

Evento BeforeRightClick:

Ocorre quando uma planilha ou gráfico incorporado é clicado com o botão direito do mouse, antes da ação padrão de clique com o botão direito.

Sintaxe 1:

```
Private Sub objeto_BeforeRightClick(Cancel As Boolean)
```

Sintaxe 2:

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
```

- ▶ **objeto:** Um objeto do tipo Chart declarado com eventos em um módulo de classe.
- ▶ **Target:** A célula mais próxima ao ponteiro do mouse quando ocorre o clique com o botão direito.

- ▶ **Cancel:** False quando o evento ocorre. Quando o procedimento de evento define esse argumento como True, a ação padrão de clique com o botão direito não ocorre quando o procedimento termina.

Nota: Como outros eventos de planilha, esse evento não ocorre quando você clica com o botão direito enquanto o ponteiro se encontra em uma forma (elemento de desenho, tal como um retângulo, círculo, etc.) ou uma barra de comando (uma barra de ferramentas ou uma barra de menu).

O conceito de eventos é realmente importante. No Módulo 5, quando estudarmos UserForms, veremos o quão importante é o uso de eventos. Praticamente toda a lógica de um aplicativo do Excel, baseado em formulários (UserForms), é baseada na utilização de eventos. Por exemplo, o evento Ao Clicar de um botão de comando, o evento Ao Receber o Foco ou Ao Perder o Foco de uma Caixa de Texto e por aí vai. Estudaremos estes elementos e os respectivos eventos, em detalhes, no Módulo 5.

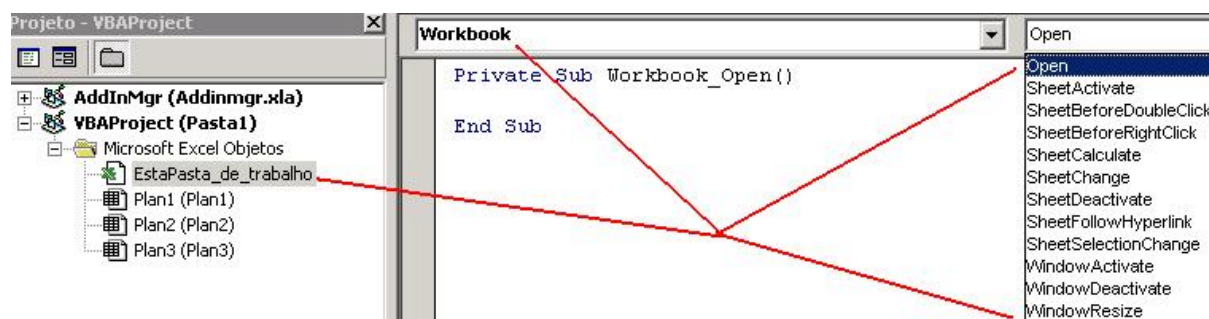
Nas próximas lições deste módulo, você aprenderá sobre os principais eventos dos objetos Workbook e Application. Na sequência apresentarei mais uma série de exemplos práticos, facilmente adaptáveis para seu próprio uso, envolvendo o uso dos diversos objetos já estudados até agora.

Lição 14: Eventos – Eventos do Objeto Workbook

Nesta e na próxima lição estudaremos os eventos do objeto Workbook. Os eventos de pasta de trabalho – objeto Workbook - ocorrem quando a pasta de trabalho é alterada ou quando qualquer planilha da pasta de trabalho é alterada. Os eventos em pastas de trabalho são ativados por padrão.

Para visualizar os procedimentos de evento de uma pasta de trabalho, siga os passos indicados a seguir:

1. Abra o Excel.
2. Abra o arquivo a ser utilizado.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Na árvore de opções, no lado esquerdo do Editor do VBA, clique na opção EstaPasta_de_trabalho.
5. Na lista da esquerda selecione Workbook.
6. Abra a lista da direita e já serão exibidos todos os eventos do objeto Workbook, conforme ilustrado na Figura a seguir:



7. Ao clicar em um dos eventos, será criada a declaração para o procedimento associado ao evento. No exemplo a seguir, foi selecionado o evento Open, para o qual foi criada a seguinte declaração de procedimento:

```
Private Sub Workbook_Open()  
  
End Sub
```

A seguir passo a descrever os principais eventos do objeto Workbook.

O Evento Open:

Este evento ocorre quando a pasta de trabalho é aberta.

Sintaxe:

```
Private Sub Workbook_Open()
```

Exemplo do evento Open:

Este exemplo maximiza o Microsoft Excel sempre que a pasta de trabalho é aberta.

```
Private Sub Workbook_Open()  
    Application.WindowState = xlMaximized  
End Sub
```

Evento BeforeClose:

Este evento ocorre antes da pasta de trabalho ser fechada. Se a pasta de trabalho tiver sido alterada, esse evento ocorrerá antes do usuário ser solicitado a salvar alterações.

Sintaxe:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
```

- ▶ **Cancel:** False quando o evento ocorre. Quando o procedimento de evento define esse argumento como True, a operação de fechamento para a pasta de trabalho é deixada aberta.

Exemplo do evento BeforeClose:

Este exemplo sempre salva a pasta de trabalho, antes de fechá-la, caso esta tenha sido alterada.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)  
    If Me.Saved = False Then Me.Save  
End Sub
```

Me é uma referência a própria pasta de trabalho. Saved é uma propriedade do objeto Workbook, a qual é True se não existirem alterações que ainda não foram salvas. Se Me.Saved for False, significa que ainda existem alterações não salvas. Neste caso, é chamado o método Save para salvar a pasta de trabalho.

O Evento BeforePrint:

Ocorre antes da pasta de trabalho (ou qualquer objeto contido na pasta) ser impressa.

Sintaxe:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
```

- ▶ **Cancel:** False quando o evento ocorre. Quando o procedimento de evento define esse argumento como True, a **pasta** de trabalho não é impressa quando o procedimento termina.

Exemplo do evento BeforePrint:

Este exemplo recalcula todas as planilhas da pasta de trabalho ativa antes de imprimir qualquer coisa.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    For Each wk in Worksheets
        wk.Calculate
    Next
End Sub
```

O procedimento percorre a coleção de planilhas, na coleção Worksheets, chamando o método Calculate (descrito anteriormente).

O Evento BeforeSave:

Este evento ocorre antes da pasta de trabalho ser salva.

Sintaxe:

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUi As Boolean, Cancel As Boolean)
```

- ▶ **SaveAsUi:** True se a caixa de diálogo Salvar como for exibida.
- ▶ **Cancel:** False quando o evento ocorre. Quando o procedimento de evento define esse argumento como True, a pasta de trabalho não é salva quando o procedimento termina.

Exemplo do evento BeforeSave:

Este exemplo pede ao usuário uma resposta sim ou não antes de salvar a pasta de trabalho.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel as Boolean)
    a = MsgBox("Deseja realmente salvar as alterações?", vbYesNo)
    If a = vbNo Then Cancel = True
End Sub
```

Neste caso é emitida uma caixa de mensagem, com os botões Sim e Não. Se o usuário clicar em Não vbNo, então o parâmetro Cancel será definido como True, o que faz com que a pasta de trabalho não seja salva.

Na próxima lição você aprenderá sobre mais alguns eventos do objeto Workbook.

Lição 15: Eventos – Eventos do Objeto Workbook

Nesta lição continuaremos o estudo dos principais eventos do objeto Workbook.

O Evento Deactivate:

Este evento ocorre quando um gráfico, planilha ou pasta de trabalho é desativado.

Sintaxe:

```
Private Sub objeto_Deactivate()
```

- ▶ **objeto:** É um objeto do tipo Chart, Workbook ou Worksheet.

Exemplo do evento Deactivate:

Este exemplo organiza todas as janelas abertas quando a pasta de trabalho é desativada.

```
Private Sub Workbook_Deactivate()  
    Application.Windows.Arrange xlArrangeStyleTiled  
End Sub
```

O Evento SheetActivate:

Este evento ocorre quando alguma das planilhas da pasta de trabalho é ativada.

Sintaxe:

```
Private Sub objeto_SheetActivate(ByVal Sh As Object)
```

- ▶ **objeto:** Um objeto do tipo Application ou Workbook.
- ▶ **Sh:** A planilha ativada. Pode ser um objeto Chart ou Worksheet.

Exemplo do evento SheetActivate:

Este exemplo exibe o nome de cada planilha ativada.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)  
    MsgBox Sh.Name  
End Sub
```

O Evento NewSheet:

Este evento ocorre quando uma nova planilha é criada na pasta de trabalho.

Sintaxe:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
```

- ▶ **Sh:** A nova planilha. Pode ser um objeto Worksheet ou Chart.

Exemplo do evento NewSheet:

Este exemplo move planilhas novas para o final da pasta de trabalho.

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    Sh.Move After:= Sheets(Sheets.Count)
End Sub
```

Sempre que uma nova planilha for criada, o evento NewSheet será disparado e o código associado a este evento será executado. No nosso exemplo, o código usa o método Move, da planilha para movê-la para o final da lista de planilhas. Isso é feito passando o número de planilhas existentes – Sheets.Count, como valor para o parâmetro After do método Move.

O Evento SheetCalculate:

Este evento ocorre após qualquer planilha ser recalculada ou após quaisquer dados alterados serem plotados em um gráfico.

Sintaxe:

```
Private Sub objeto_SheetCalculate(ByVal Sh As Object)
```

- ▶ **objeto:** Application ou Workbook. Para obter mais informações sobre uso de eventos com o objeto Application, consulte as próximas lições.
- ▶ **Sh:** A planilha. Pode ser um objeto Chart ou Worksheet.

Exemplo do evento SheetCalculate:

Este exemplo classifica o intervalo A1:A100 da planilha um quando qualquer uma das planilhas da pasta de trabalho for calculada.

```
Private Sub Workbook_SheetCalculate(ByVal Sh As Object)
    Worksheets(1).Range("A1:A100").Sort Key1:=.Range("A1")
End Sub
```

O Evento SheetChange:

Este evento ocorre quando células de qualquer planilha da pasta de trabalho são alteradas pelo usuário ou por um vínculo externo.

Sintaxe:

```
Private Sub objeto_SheetChange(ByVal Sh As Object, ByVal Source As Range)
```

- ▶ **objeto:** Application ou Workbook. Para obter mais informações sobre uso de eventos com o objeto Application, consulte Uso de eventos com o objeto Application.
- ▶ **Sh:** Um objeto Worksheet representando a planilha.
- ▶ **Source:** O intervalo que foi alterado e que provocou o disparo do evento.

Nota: Esse evento não ocorre em folhas de gráfico.

Exemplo do evento SheetChange:

Este exemplo é executado quando alguma planilha é alterada e coloca em negrito, as células que foram alteradas e que provocaram o evento.

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Source As Range)
    Source.Font.Bold = True
End Sub
```

O Evento WindowActivate:

Este evento ocorre quando qualquer janela de pasta de trabalho é ativada.

Sintaxe:

```
Private Sub objeto_WindowActivate(ByVal Wb As Excel.Workbook, ByVal Wn As Excel.Window)
```

- ▶ **objeto:** Application ou Workbook. Para obter mais informações sobre como usar eventos com o objeto Application, consulte as próximas lições.
- ▶ **Wb:** Usado somente com o objeto Application. A pasta de trabalho exibida na janela ativada.
- ▶ **Wn:** Uma referência à janela ativada.

Exemplo do evento WindowActivate:

Este exemplo maximiza qualquer janela de pasta de trabalho quando esta é ativada.

```
Private Sub Workbook_WindowActivate(ByVal Wn As Excel.Window)
    Wn.WindowState = xlMaximized
End Sub
```

O Evento WindowDeactivate:

Este evento ocorre quando qualquer janela de pasta de trabalho é desativada.

Sintaxe:

Private Sub objeto_WindowDeactivate(ByVal Wb As Excel.Workbook, ByVal Wn As Excel.Window)

- ▶ **objeto:** Application ou Workbook. Para obter mais informações sobre como usar eventos com o objeto Application, consulte as próximas lições.
- ▶ **Wb:** Usado somente com o objeto Application. A pasta de trabalho exibida na janela desativada.
- ▶ **Wn:** A janela desativada.

Exemplo do evento WindowDeactivate:

Este exemplo minimiza qualquer janela de pasta de trabalho quando esta é desativada.

```
Private Sub Workbook_WindowDeactivate(ByVal Wn As Excel.Window)
    Wn.WindowState = xlMinimized
End Sub
```

O Evento WindowResize:

Este evento ocorre quando qualquer janela de pasta de trabalho é redimensionada.

Sintaxe:

Private Sub objeto_WindowResize(ByVal Wb As Excel.Workbook, ByVal Wn As Excel.Window)

- ▶ **objeto:** Application ou Workbook. Para obter mais informações sobre como usar eventos com o objeto Application, consulte as próximas lições.
- ▶ **Wb** Usado somente com o objeto Application. A pasta de trabalho exibida na janela redimensionada.
- ▶ **Wn:** A janela redimensionada.

Exemplo do evento WindowResize:

Este exemplo é executado quando qualquer janela de pasta de trabalho é redimensionada. O código a seguir exibe, na Barra de Status, o título da janela de título, seguida da palavra Redimensionada.

```
Private Sub Workbook_WindowResize(ByVal Wn As Excel.Window)
    Application.StatusBar = Wn.Caption & " Redimensionada"
End Sub
```

Muito bem, nas duas próximas lições, falaremos sobre os eventos do objeto Application.

Lição 16: Eventos – Eventos do Objeto Application

Nesta e na próxima lição estudaremos os eventos do objeto Application.

Os eventos de Application ocorrem quando uma pasta de trabalho é criada ou aberta ou quando qualquer planilha de qualquer pasta de trabalho aberta é alterada. Para escrever procedimentos de evento para o objeto Application, você precisa criar um novo objeto usando a palavra-chave WithEvents em um módulo de código VBA

Antes de você poder usar eventos com o objeto Application, você precisa criar um novo módulo de código VBA e declarar um objeto do tipo Application com eventos. Por exemplo, suponha que um novo módulo código seja criado com o nome de EventClassModule. O novo módulo de classe conterá o seguinte código:

```
Public WithEvents App As Application
```

Após o novo objeto ter sido declarado com eventos, ele aparece na caixa de listagem suspensa Objeto (lista da esquerda) no módulo de classe, e você pode escrever procedimentos de evento para o novo objeto. (Quando você seleciona o novo objeto na caixa Objeto, os eventos válidos para esse objeto são listados na caixa de listagem suspensa Procedimento – Caixa de listagem da direita).

Entretanto, antes dos procedimentos serem executados, você precisa conectar o objeto declarado no módulo de classe com o objeto Application. Você pode fazer isso com o código seguinte a partir de qualquer módulo.

```
Dim X As New EventClassModule  
  
Sub InitializeApp()  
    Set X.App = Application  
End Sub
```

Após executar o procedimento InitializeApp, o objeto App do módulo de classe apontará para o objeto Application do Microsoft Excel, e os procedimentos de evento do módulo de classe serão executados quando os eventos ocorrerem.

Fora estes procedimentos iniciais, para poder utilizar os eventos do objeto Application, muitos dos eventos do objeto Application, são os mesmos do objeto Workbook. Os eventos que são os mesmos, para os dois objetos – Workbook e Application – e que já foram descritos nas lições anteriores, são os seguintes:

- ▶ SheetActivate
- ▶ SheetBeforeDoubleClick
- ▶ SheetCalculate
- ▶ SheetChange
- ▶ WindowActivate
- ▶ WindowDeactivate
- ▶ WindowResize

A seguir passo a descrever alguns eventos que são exclusivos do objeto Application.

O Evento NewWorkbook:

Este evento ocorre quando uma nova pasta de trabalho é criada.

Sintaxe:

```
Private Sub objeto_NewWorkbook(ByVal Wb As Workbook)
```

- ▶ **objeto:** Um objeto do tipo Application declarado com eventos em um módulo de código VBA, conforme descrito no início desta lição.
- ▶ **Wb:** Uma referência a nova pasta de trabalho.

Exemplo do evento NewWorkbook:

Este exemplo organiza todas as janelas ativas quando uma nova pasta de trabalho é criada.

```
Private Sub App_NewWorkbook(ByVal Wb As Workbook)  
    Application.Windows.Arrange xlArrangeStyleTiled  
End Sub
```

O Evento WorkbookActivate:

Este evento ocorre quando alguma pasta de trabalho é ativada.

Sintaxe:

```
Private Sub app_WorkbookActivate(ByVal Wb As Workbook)
```

- ▶ **app:** Um objeto do tipo Application declarado com eventos em um módulo de código VBA, conforme descrito no início desta lição.
- ▶ **Wb:** Uma referência a pasta de trabalho ativada.

Exemplo do evento WorkbookActivate:

Este exemplo organiza as janelas abertas quando uma pasta de trabalho é ativada.

```
Private Sub App_WorkbookActivate(ByVal Wb As Workbook)  
    Application.Windows.Arrange xlArrangeStyleTiled  
End Sub
```

O Evento WorkbookAddinInstall:

Este evento ocorre quando uma pasta de trabalho é instalada como um suplemento.

Sintaxe:

```
Private Sub objeto_WorkbookAddinInstall(ByVal Wb As Workbook)
```

- ▶ **objeto:** Um objeto do tipo Application declarado com eventos em um módulo de código VBA, conforme descrito no início desta lição.
- ▶ **Wb:** Uma referência à pasta de trabalho instalada.

Exemplo do evento WorkbookAddinInstall:

Este exemplo maximiza a janela do Microsoft Excel quando uma pasta de trabalho é instalada como um suplemento.

```
Private Sub App_WorkbookAddinInstall(ByVal Wb As Workbook)  
    Application.WindowState = xlMaximized  
End Sub
```

O Evento WorkbookAddinUninstall:

Este evento ocorre quando alguma pasta de trabalho de suplemento é desinstalada.

Sintaxe:

```
Private Sub objeto_WorkbookAddinUninstall(ByVal Wb As Workbook)
```

- ▶ **objeto:** Um objeto do tipo Application declarado com eventos em um módulo de código VBA, conforme descrito no início desta lição.
- ▶ **Wb:** Uma referência à pasta de trabalho desinstalada.

Exemplo do evento WorkbookAddinUninstall:

Este exemplo minimiza a janela do Microsoft Excel quando uma pasta de trabalho é instalada como um suplemento.

```
Private Sub App_WorkbookAddinUninstall(ByVal Wb As Workbook)  
    Application.WindowState = xlMinimized  
End Sub
```

Na próxima lição veremos os demais eventos do objeto Application.

Lição 17: Eventos – Eventos do Objeto Application

Nesta lição finalizaremos o estudo dos eventos do objeto Application e, com isso, o estudo dos eventos dos principais objetos do modelo de objetos do Excel. Voltaremos a falar sobre eventos no Módulo 5, mais especificamente sobre os eventos de UserForms e dos elementos de um UserForm.

O Evento WorkbookBeforeClose:

Este evento ocorre imediatamente antes de qualquer pasta de trabalho aberta ser fechada.

Sintaxe:

```
Private Sub objeto_WorkbookBeforeClose(ByVal Wb As Workbook, ByVal Cancel As Boolean)
```

- ▶ **objeto:** Um objeto do tipo Application declarado com eventos em um módulo de código VBA, conforme descrito no início da lição anterior.
- ▶ **Wb:** A pasta de trabalho que está sendo fechada.
- ▶ **Cancel:** False quando o evento ocorre. Quando o código dentro do procedimento de evento define esse argumento como True, a pasta de trabalho não é fechada quando o procedimento termina.

Exemplo do evento WorkbookBeforeClose:

Este exemplo pede ao usuário uma resposta sim ou não antes de fechar qualquer pasta de trabalho.

```
Private Sub App_WorkbookBeforeClose(ByVal Wb as Workbook, Cancel as Boolean)
    a = MsgBox("Deseja realmente fechar o arquivo?", vbYesNo)
    If a = vbNo Then Cancel = True
End Sub
```

O Evento WorkbookBeforePrint:

Este evento ocorre antes de qualquer pasta de trabalho aberta ser impressa.

Sintaxe:

```
Private Sub objeto_WorkbookBeforePrint(ByVal Wb As Workbook, ByVal Cancel As Boolean)
```

- ▶ **objeto:** Um objeto do tipo Application declarado com eventos em um módulo de código VBA, conforme descrito no início da lição anterior.
- ▶ **Wb :**A pasta de trabalho.

- ▶ **Cancel:** False quando o evento ocorre. Quando o código dentro do procedimento de evento define esse argumento como True, a pasta de trabalho não é fechada quando o procedimento termina.

Exemplo do evento WorkbookBeforePrint:

Este exemplo recalcula todas as planilhas da pasta de trabalho antes de imprimir alguma coisa.

```
Private Sub App_WorkbookBeforePrint(ByVal Wb As Workbook, Cancel As Boolean)
    For Each wk in Wb.Worksheets
        wk.Calculate
    Next
End Sub
```

O Evento WorkbookBeforeSave:

Este evento ocorre antes de qualquer pasta de trabalho aberta ser salva.

Sintaxe:

```
Private Sub objeto_WorkbookBeforeSave(ByVal Wb As Workbook, ByVal SaveAsUI As Boolean,
ByVal Cancel As Boolean)
```

- ▶ **objeto:** Um objeto do tipo Application declarado com eventos em um módulo de código VBA, conforme descrito no início da lição anterior.
- ▶ **Wb:** A pasta de trabalho.
- ▶ **SaveAsUI:** True se a caixa de diálogo Salvar como for exibida.
- ▶ **Cancel:** False quando o evento ocorre. Quando o código dentro do procedimento de evento define esse argumento como True, a pasta de trabalho não é fechada quando o procedimento termina.

Exemplo do evento WorkbookBeforeSave:

Este exemplo pede ao usuário uma resposta sim ou não antes de salvar qualquer pasta de trabalho.

```
Private Sub App_WorkbookBeforeSave(ByVal Wb As Workbook, _
    ByVal SaveAsUI As Boolean, Cancel As Boolean)
    a = MsgBox("Deseja realmente Salvar?", vbYesNo)
    If a = vbNo Then Cancel = True
End Sub
```

O Evento WorkbookOpen:

Este evento ocorre quando uma pasta de trabalho é aberta.

Sintaxe:

```
Private Sub objeto_WorkbookOpen(ByVal Wb As Workbook)
```

- ▶ **objeto:** Um objeto do tipo Application declarado com eventos em um módulo de código VBA, conforme descrito no início da lição anterior.
- ▶ **Wb:** A pasta de trabalho.

Exemplo do evento WorkbookOpen:

Este exemplo organiza todas as janelas abertas quando uma pasta de trabalho é aberta.

```
Private Sub App_WorkbookOpen(ByVal Wb As Workbook)  
    Application.Windows.Arrange xlArrangeStyleTiled  
End Sub
```

O Evento WorkbookNewSheet:

Este evento ocorre quando uma nova planilha é criada em qualquer pasta de trabalho aberta.

Sintaxe:

```
Private Sub objeto_WorkbookNewSheet(ByVal Wb As Workbook, ByVal Sh As Object)
```

- ▶ **objeto:** Um objeto do tipo Application declarado com eventos em um módulo de código VBA, conforme descrito no início da lição anterior.
- ▶ **Wb:** A pasta de trabalho.
- ▶ **Sh:** A nova planilha.

Exemplo do evento WorkbookNewSheet:

Este exemplo move a planilha nova para o final da pasta de trabalho.

```
Private Sub App_WorkbookNewSheet(ByVal Wb As Workbook, ByVal Sh As Object)  
    Sh.Move After:=Wb.Sheets(Wb.Sheets.Count)  
End Sub
```

Muito bem, com isso encerramos o estudo dos principais eventos, dos principais objetos do modelo de objetos do Excel. Nas demais lições deste módulo, apresentarei mais uma série de exemplos práticos, de utilização dos diversos objetos estudados até agora: Application, Workbook, Worksheets, Range, etc.

Lição 18: Exemplos Práticos

A partir desta lição apresentarei uma série de exemplos práticos, os quais utilizam os eventos dos objetos do Excel e também os objetos já estudados até o momento: Application, Workbook, Worksheet e Range.

Como criar macros que são executadas automaticamente ao abrir uma pasta de trabalho – as macros Auto_Open e Workbook_Open:

Você pode criar uma macro chamada Auto_Open() em um módulo de código VBA. Esta macro será automaticamente executada, sempre que a pasta de trabalho, onde está o módulo, for aberta manualmente no Excel. A macro Auto_Open() no Excel é semelhante a macro Autoexec em um banco de dados do Access. Prova disso é que a execução da macro Auto_Open() também pode ser cancelada, se o usuário ficar pressionando a tecla Shift, enquanto a pasta de trabalho é aberta. No Access também, ou seja, a execução da macro Autoexec será cancelada, se o usuário ficar pressionando a tecla Shift, durante a abertura do banco de dados.

Você também pode criar uma macro chamada Workbook_Open(), no módulo de código do Workbook. Esta macro será automaticamente executada, quando a pasta de trabalho (representada pelo objeto Workbook), for aberta.

Notas: Pode acontecer de as duas macros terem sido criadas – Auto_Open() e Workbook_Open(), neste caso ambas serão executadas. Se o objeto Workbook for aberto usando código VBA, ao invés de manualmente através dos menus do Excel, e a pasta de trabalho contiver as macros Auto_Open() e Workbook_Open(), somente a macro Workbook_Open() será executada, já a macro Auto_Open() não será executada. Porém você pode fazer com que a macro Auto_Open() também seja executada, mesmo quando a pasta de trabalho for aberta via código VBA, para isso você adiciona a seguinte linha de código, ao procedimento que irá abrir a pasta de trabalho:

```
ActiveWorkbook.RunAutoMacros xlAutoOpen
```

Existem outras maneiras de fazer com que a macro Auto_Open() seja executada, conforme o exemplo a seguir, onde mostro um exemplo de código a ser utilizado. Este código fará a abertura de uma pasta de trabalho chamada Vendas.xls e fará com que a macro Auto_Open, além da macro Workbook_Open, seja executada:

```
Dim myWB As Workbook

' o comando a seguir abre a pasta de trabalho Vendas.xls e associa
' esta pasta com o a variável myWB

Set myWB = Workbooks.Open(FileName:="Vendas.xls")

' O comando a seguir fará com que a macro Auto_Open( ) seja executada,
' caso exista uma macro chamada AutoOpen:

myWB.RunAutoMacros xlAutoOpen
```

Nota: Se a pasta de trabalho Vendas.xls não tiver uma macro chamada Auto_Open, o último comando do exemplo anterior, será simplesmente ignorado.

A seguir apresento a estrutura da macro Auto_Open()

```
Sub Auto_Open( )
    'Comando 1
    'Comando 2
    ...
    'Comando n
End Sub
```

Importante: Para que as macros Auto_Open() e/ou Workbook_Open() não sejam executadas durante a abertura de uma pasta de trabalho, basta manter pressionada a tecla Shift, conforme já descrito anteriormente.

Como Executar uma macro toda vez que uma pasta de trabalho é fechada:

Para executar uma macro ou um conjunto de comandos, toda vez que uma pasta de trabalho é fechada, você deve criar um procedimento chamado Auto_Close(). A estrutura deste procedimento está indicada a seguir:

```
Sub Auto_Close( )

    'Comando 1
    'Comando 2
    ...
    'Comando n

End Sub
```

Importante: Se a pasta de trabalho for fechada a partir de código VBA, sendo executado em outra pasta de trabalho, o procedimento Auto_Close() não será executado. Para fazer com que o procedimento Auto_Close() seja executado, mesmo quando a pasta de trabalho for fechada a partir do código VBA, você deve utilizar o comando a seguir, onde está sendo fechada a pasta de trabalho Vendas.xls:

```
Application.Run "'Vendas.xls'!Auto_Close"
```

Caso exista também uma macro chamada Workbook_BeforeClose(), na pasta de trabalho que está sendo fechada, esta macro também será executada quando a pasta de trabalho for fechada. A seguir apresento a estrutura básica da macro Workbook_BeforeClose():

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

    'Comando 1
    'Comando 2
    ...
    'Comando n

End Sub
```

Ordem dos eventos de fechamento de uma pasta de trabalho:

A ordem em que os eventos de fechamento ocorrem é a seguinte:

- ▶ Workbook_BeforeClose
- ▶ Auto_Close
- ▶ Workbook_BeforeSave

Exceção: Quando você usar o comando `ThisWorkbook.Save`, o evento `Workbook_BeforeSave` será chamado, mas será executado antes do evento `Auto_Close`. Esta é uma exceção à ordem normal de execução, descrita na lista anterior.

Interceptando os eventos Close do Excel e do objeto Workbook:

Durante a execução do procedimento `Auto_Close()` você pode verificar a propriedade `ThisWorkbook.Saved`. Se o valor desta propriedade for `True` (Verdadeiro), você pode confirmar o fechamento, pois não existem alterações a serem salvas. Já se o valor desta propriedade for `False` (Falso), você deve exibir uma caixa de diálogo para que o usuário informe se deseja ou não salvar as alterações que ainda não foram salvas.

Se o usuário optar por salvar as alterações, você deve salvar a pasta de trabalho e fechar o Excel. Se o usuário responder que não deseja salvar as alterações, você deve definir a propriedade `ThisWorkbook.Saved` como `True` e fechar o Excel. Se você não definir a propriedade `ThisWorkbook.Saved` como `True`, o Excel exibirá a janela padrão informando que existem alterações a serem salvas. Ao definir `ThisWorkbook.Saved` como `True`, você evita que esta janela seja exibida novamente. Se o usuário optar por Cancelar a caixa de diálogo, você deve cancelar o fechamento da pasta de trabalho e do Excel. Para isso você usa o comando indicado a seguir:

```
Application.ExecuteExcel4Macro "HALT(True)"
```

Lição 19: Exemplos Práticos

Nesta lição continuarei a apresentar uma série de exemplos práticos, os quais utilizam os eventos dos objetos do Excel e também os objetos já estudados até o momento: Application, Workbook, Worksheet e Range.

Como desabilitar a execução de eventos:

Você pode fazer com que os procedimentos de evento parem de ser executados. Para isso você deve definir a propriedade EnableEvents, do objeto Application, com o valor False. Para fazer com que os procedimentos de eventos voltem a ser executados, você deve definir esta propriedade como True. No exemplo a seguir, defino a propriedade EnableEvents como False:

```
Application.EnableEvents = False
```

Para fazer com que os procedimentos de eventos voltem a ser executados, defina a propriedade EnableEvents em True, conforme exemplo a seguir:

```
Application.EnableEvents = True
```

Como executar uma macro a cada “x” minutos:

Uma das dúvidas que mais recebo via email, é como criar uma macro que seja executada, automaticamente, dentro de um intervalo de tempo definido. Por exemplo, uma macro que seja executada de minuto em minuto ou de meia em meia-hora e assim por diante. Para que uma macro possa ser executada periodicamente, o primeiro passo é utilizar o evento OnTime. O exemplo a seguir, mostra como criar uma macro chamada ExecutaACadaMinuto, a qual é configurada para executar de minuto em minuto:

```
Sub ExecutaACadaMinuto()  
    ' Comandos da macro  
    Application.OnTime Now + 1/1440, "Nome da Macro a ser Executado"  
End Sub
```

Nota: O valor 1/1440 é o valor para um minuto, ou seja, 1 dia (1440 minutos) dividido por 1440. O resultado é um.

Você também pode definir o tempo em segundos, conforme descrevo logo a seguir. Considerando que em um dia existem 86.4000 segundos, você pode definir o tempo em segundos, conforme os exemplos a seguir:

Tempo de meio Segundo:

```
Application.OnTime Now + 1/86400 * 0.5, "NomeDoProcedimento"
```

Tempo de dois segundos:

```
Application.OnTime Now + 1/86400 * 2, "NomeDoProcedimento"
```

A seguir mais alguns exemplos de como fazer uma macro ser executada, automaticamente, em diferentes períodos de tempo.

Como fazer uma macro ser executada a cada dois minutos:

A seguir um exemplo de código para fazer com que uma macro seja executada a cada dois minutos.

' No início do procedimento, coloque a seguinte declaração:

```
Dim dNext As Date

Sub HabilitaExecução()
    DoItAgain
End Sub

Sub ExecutaNovamente()
    'Determina o intervalo de execução.
    dNext = Now + TimeValue("00:02:00")

    'Utilizo o evento OnTime
    Application.OnTime dNext, "ExecutaNovamente"

    ' Procedimento a ser chamado
    Executa
End Sub

Sub Executa
    MsgBox "Procedimento que está sendo executado!!!"
End Sub

Sub ParaExecução()
    On Error Resume Next
    'Este procedimento cancela o procedimento de evento
    Application.OnTime dNext, "ExecutaNovamente", schedule:=False
End Sub
```

' Para desabilitar a execução automática, você pode usar o seguinte código, no qual você chama o procedimento ParaExecução(), dentro do evento Auto_Close() da pasta de trabalho:

```
Sub Auto_Close()
    ParaExecução
End Sub
```

Também é possível passar argumentos para o procedimento a ser executado repetidamente, quando OnTime é chamado, conforme exemplos a seguir:

Passando valores numéricos para o procedimento a ser executado:

```
Application.OnTime= "'NomeDoProcedimento 1, 2'"
```

Se o valor dos argumentos for do tipo texto, use a sintaxe a seguir:

```
Application.OnTime = "'NomeDoProcedimento '"Val1"', '"Val2"'"
```

Como cancelar a execução de uma macro agendada pelo evento OnTime:

Para cancelar a execução de uma macro, agendada para execução periódica pelo evento OnTime, você deve gravar em uma variável, o valor da hora quando foi iniciada a execução, isto é, a hora em que o método OnTime foi chamado. É recomendado que este valor seja armazenado em uma variável global ao módulo, para ser utilizada quando você for cancelar a execução da macro.

Cuidado: Se você editar o código VBA, antes de parar a execução a macro, ou utilizar um comando End, para suspender a execução do código, o valor de todas as variáveis será resetado, o que fará com que você perca o valor da hora em que foi chamado o método OnTime. A solução para esta questão é armazenar o valor da hora de chamada em uma célula da planilha.

```
' O exemplo a seguir armazena o valor em uma variável e então  
' chama o método OnTime, configurando um tempo de execução de 1 minuto
```

```
Timetorun=Now + TimeValue("00:01:00")  
Application.OnTime Timetorun, "NomeDaMacro"
```

' A seguir mostro como usar o valor armazenado na variável Timetorun, para
' cancelar a execução periódica da Macro agendada pelo código anterior:

```
Application.OnTime earliesttime:=Timetorun, procedure:="NomeDaMacro", schedule:=False
```

Lição 20: Exemplos Práticos

Nesta lição continuarei a apresentar uma série de exemplos práticos, os quais utilizam os eventos dos objetos do Excel e também os objetos já estudados até o momento: Application, Workbook, Worksheet e Range.

Usando eventos para detectar quando uma célula é alterada:

Para criar código que seja executado, em resposta a alterações nas células de uma planilha, você deve utilizar o evento Worksheet_Change. Este evento é criado no módulo de código da própria planilha e tem a estrutura indicada a seguir:

```
Private Sub Worksheet_Change(ByVal Target As Range)
```

Uma referência a faixa de células onde foi feita a alteração (ou as alterações) é passada no parâmetro Target. Com isso você pode usar este parâmetro para analisar as alterações que foram efetuadas e, com base nestas alterações, executar diferentes ações, dentro do código do evento Worksheet_Change.

Executar uma Macro com base no valor de uma célula:

O evento Worksheet_Change pode ser utilizado para verificar o valor de uma determinada célula e executar diferentes comandos, com base neste valor. No exemplo a seguir, verifico se a célula alterada foi a célula C25 e, se o valor atual da célula é 210. Se estas duas condições forem atendidas, será executado o código dentro dos dois testes If, caso contrário, nada será executado:

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    If Target.Address = "$C$25" Then
        If Target.Value = 210 Then
            ' Comando 1
            ' Comando 2
            ' ...
            ' Comando n
        End If
    End If
End Sub
```

Lembrando sempre que o parâmetro Target é uma referência ao intervalo que foi alterado, normalmente é uma faixa de uma única célula, ou seja, a célula que foi alterada e cuja alteração provocou o evento Worksheet_Change.

Como executar uma macro quando a seleção atual é alterada:

Para detectar quando é feita uma alteração na seleção atual, você pode utilizar o evento de planilha SelectionChange. No exemplo a seguir, é emitida uma mensagem sempre que a seleção atual da planilha for alterada. Este evento deve ser colocado no módulo de código da planilha, conforme descrito nas lições teóricas sobre Eventos:


```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
    MsgBox "Você selecionou: " & Target.Address
End Sub
```

Este evento usa o parâmetro Target, o qual é uma referência a faixa de células da seleção atual. Também utilizo a propriedade Address, do objeto Range para exibir o endereço da seleção atual.

Como eu faço para executar uma Macro quando uma planilha é ativada:

Você pode criar uma macro que seja executada somente quando uma determinada planilha é ativada. Para isso você acessa o módulo de código da planilha e usa o Evento Activate da planilha. O código do exemplo a seguir exibe uma mensagem sempre que a planilha for ativada:

```
Private Sub Worksheet_Activate()
    MsgBox "Bem vindo!!!"
End Sub
```

Por outro lado você pode querer criar uma macro que seja executada sempre que qualquer uma das planilhas, de uma pasta de trabalho, seja ativada. Neste caso você tem que utilizar o evento SheetActivate, do objeto Workbook. O exemplo a seguir emite uma mensagem sempre que qualquer uma das planilhas da pasta de trabalho for ativada. Este código deve ser colocado no módulo de código da pasta de trabalho – ThisWorkbook:

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    MsgBox "Bem vindo!!!"
End Sub
```

A seguir uma abordagem um pouco diferente, onde utilize o evento Auto_Open em conjunto com duas macros criadas separadamente: AtivaMacro e DesativaMacro. O código a ser executado em cada caso está nas respectivas macros:

```
Sub Auto_Open
    ThisWorkbook.Worksheets("Plan1").OnSheetActivate = "AtivaMacro"
    ThisWorkbook.Worksheets("Plan1").OnSheetDeactivate = "DesativaMacro"
End Sub
```

```
Sub SheetActivateMacro
    'Comandos
End Sub
```

```
Sub SheetDeactivateMacro
    'Comandos
End Sub
```

Neste caso, no evento Auto_Open, uso a propriedade OnShetActivate, para associar a macro AtivaMacro com o evento Activate da planilha Plan1. Ou seja, sempre que a planilha Plan1 for ativada, a macro AtivaMacro será executada. Em seguida uso a propriedade

OnSheetDeactivate, para associar a macro DesativaMacro com o evento Deactivate, da planilha Plan1. Dentro do evento Auto_Open() é possível criar associações para os eventos Activate e Deactivate de mais de uma planilha.

Quais os eventos do Excel que são disparados quando uma célula é alterada?

Para entendermos as diferenças entre os eventos Worksheet_Change e Worksheet_SelectionChange, crie os dois procedimentos a seguir, no módulo de código de uma planilha do Excel. Pode ser no módulo de código da planilha Plan1 de uma pasta de trabalho em branco:

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    If target.Address="$A$1" Then
        MsgBox ("A Célula A1 foi alterada para: ") & Target.Value
    End If
End Sub

Private Sub Worksheet_SelectionChange( ByVal Target As Excel.Range)
    If target.Address="$A$1" Then
        MsgBox ("A célula A1 foi selecionada!!!")
    End If
End Sub
```

Digite o valor 25 na célula A1 e clique na célula A2. Será exibida a mensagem indicada na Figura a seguir. Ou seja, o valor da célula foi alterado. Foi disparado o evento Worksheet_Change, no qual verifico se o endereço da célula alterada é A1 e, em caso afirmativo, exibe a mensagem informando que o valor foi alterado e o respectivo valor:



Agora clique de volta na célula A1. Isso provocará o evento Worksheet_SelectionChange e será exibida a mensagem indicada a seguir, a qual é emitida pelo código do evento Worksheet_SelectionChange. Este código verifica se o endereço da seleção atual é A1 = target.Address="\$A\$1" - e, em caso afirmativo, emite a mensagem:



Lição 21: Exemplos Práticos

Nesta lição continuarei a apresentar uma série de exemplos práticos, os quais utilizam os eventos dos objetos do Excel e também os objetos já estudados até o momento: Application, Workbook, Worksheet e Range.

Exemplos de uso do evento Worksheet_Change:

Existem realmente muitos exemplos de usos práticos para o evento Worksheet_Change. Por exemplo, você pode permitir que um usuário digite sempre um valor na célula A1, enquanto o código do evento Worksheet_Change, vai movendo os valores digitados para uma faixa e em outra coluna da planilha. Por exemplo, se o usuário digitar 10 valores em A1, você pode mover estes valores para a faixa de C1 a C10 ou outra faixa qualquer na planilha.

Para criar a estrutura do procedimento associado ao evento Worksheet_Change, siga os passos indicados a seguir:

1. Abra o Excel.
2. Abra a planilha onde está a planilha a ser criado o procedimento para o exemplo Worksheet_Change.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Nas opções que são exibidas do lado esquerdo da tela clique em Plan1 (Plan1) para selecionar esta opção ou no nome da planilha a ser utilizada.
5. Na lista Geral, selecione Worksheet e na lista ao lado selecione Change. Pode ocorrer de o Editor do VBA ter criado duas declarações, uma para o evento Change e uma para o Evento SelectionChange, conforme indicado a seguir:

```
Private Sub Worksheet_Change(ByVal Target As Range)

End Sub

Private Sub Worksheet_SelectionChange(ByVal Target As Range)

End Sub
```

6. Exclua a declaração para o evento Worksheet_SelectionChange, mantendo apenas a declaração para o evento Worksheet_Change.
7. Agora é só inserir o código para o evento Worksheet_Change, o qual será executado, sempre que uma célula da planilha for alterada.

Como fazer a validação dos dados digitados pelo usuário:

Cada vez que o usuário pressiona Enter, é executada a macro configurada pela propriedade OnEntry, do objeto Application. Em outras palavras, você pode associar o nome de uma macro, com a propriedade OnEntry do objeto Application e esta macro será executada, toda vez que for pressionada a tecla Enter, em qualquer planilha de qualquer pasta de trabalho. No exemplo a seguir, estou configurando a macro TrataEntrada, para que seja executada toda vez que for pressionada a tecla Enter:

```
Application.OnEntry = "TrataEntrada"
```

Você também pode configurar uma macro para ser executada, somente em resposta ao uso da tecla Enter, em uma planilha específica e não em todas as planilhas de uma pasta de trabalho. No exemplo a seguir, estou configurando a macro TrataEntrada, para ser executada em resposta ao uso da tecla Enter, na planilha Plan1:

```
Worksheets("Plan1").OnEntry = "TrataEntrada"
```

Toda vez que a tecla Enter for pressionada na planilha Plan1, a macro TrataEntrada será executada.

Para desabilitar a execução de uma macro, associada com o uso da tecla Enter, você atribui um valor vazio à propriedade OnEntry, conforme indicado a seguir:

```
Application.OnEntry = ""
```

Importante: A atribuição de um nome de macro à propriedade OnEntry, permanece válida até que seja atribuído um valor vazio a esta propriedade, ou até que o Excel seja fechado ou quando a macro associada à propriedade OnEntry não existir. Por exemplo, se você fechar o Excel e abri-lo novamente, a atribuição terá que ser refeita. Por isso, se você quiser que a atribuição esteja sempre disponível, você deve colocá-la em um evento do objeto Workbook, como por exemplo o evento Open do objeto Workbook.

Uma das utilizações mais comuns da propriedade OnEntry, é a utilização de uma macro para validação da entrada de dados, conforme o exemplo a seguir. A lógica é bastante simples.

1) Inicialmente utilizo o evento Auto_Open, para definir a macro que será executada, em resposta ao uso da tecla Enter, nas planilhas Plan1 e Plan2. Neste caso será executada uma macro chamada ValidaDados:

```
Sub Auto_Open()  
    Sheets("Plan1").OnEntry = "ValidaDados"  
    Sheets("Plan2").OnEntry = "ValidaDados"  
End Sub
```

2) O próximo passo é implementar a macro ValidaDados propriamente dita:

```
Sub ValidaDados()  
    ' Comandos para validação dos dados de entrada  
    ' Comando 1  
    ' Comando 2  
    ' ...  
    ' Comando n  
End Sub
```

Pronto, agora toda vez que a tecla Enter for utilizada nas planilhas Plan1 e Plan2, será executada a macro ValidaDados.

Nota: Para configurar a propriedade OnEntry, para todas as planilhas de uma pasta de trabalho, utilize o comando a seguir:

```
ThisWorkbook.OnEntry = "ValidaDados"
```

Convertendo todas as entradas para letras MAIÚSCULAS:

A seguir um exemplo de código, associado ao evento Change da planilha, o qual detecta se a célula que está sendo alterada contém um valor de texto e não é uma fórmula. Se for este o caso, o valor digitado pelo usuário é convertido para letras maiúsculas:

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)

    ' desabilita o tratamento de eventos durante a execução da macro, para evitar um
    ' loop de execução, pois a conversão para maiúsculas iria gerar um novo evento,
    ' o qual chamaria novamente o evento, o qual geraria uma nova alteração,
    ' o qual chamaria novamente o evento e assim indefinidamente.

    Application.EnableEvents = False

    ' Verifica se o valor da célula é uma fórmula e se não é um valor numérico

    If (Not ActiveCell.HasFormula) And (Not IsNumeric(ActiveCell.Value)) Then
        ActiveCell.Value = UCase(ActiveCell.Value)
    End If

    ' Habilita novamente a geração de eventos.

    Application.EnableEvents = True

End Sub
```

Lição 22: Exemplos Práticos

Nesta lição continuarei a apresentar uma série de exemplos práticos, os quais utilizam os eventos dos objetos do Excel e também os objetos já estudados até o momento: Application, Workbook, Worksheet e Range.

Conversão para maiúsculas usando OnEntry:

Você pode utilizar uma macro associada com a propriedade OnEntry (lembrando que esta macro será executada em resposta ao evento provocado pelo uso da tecla Enter), macro esta que converte as entradas de uma planilha, automaticamente, para maiúsculas, usando a função UCase. Lembrando da lição anterior, que uma macro pode ser associada com uma das planilhas ou com todas as planilhas de uma pasta de trabalho. Vamos a alguns exemplos práticos:

- ‘ A seguir crio defino o nome de uma macro associada com a propriedade OnEntry
- ‘ Esta associação é específica para a planilha Vendas, ou seja, a macro somente
- ‘ será executada em resposta ao evento gerado na planilha Vendas

```
Sub Set_Upper_Case_Entry_On()  
    Worksheets("Vendas").OnEntry = "TrocaParaMaiusculas"  
End Sub
```

Este procedimento associa a macro TrocaParaMaiusculas com a propriedade OnEntry da planilha vendas, ou seja, na prática, define que a macro TrocaParaMaiusculas será executada, sempre que a tecla Enter for utilizada na planilha Vendas.

A seguir apresento o código da Macro TrocaParaMaiusculas, o qual simplesmente verifica se a entrada da célula ativa é uma fórmula e não é um valor numérico. Se o teste for verdadeiro, o valor é convertido para maiúsculas usando a função UCase

```
Sub TrocaParaMaiusculas()  
  
    If (Not ActiveCell.HasFormula) And (Not IsNumeric(ActiveCell.Value)) Then  
        ActiveCell.Value = UCase(ActiveCell.Value)  
    End If  
  
End Sub
```

Para desabilitar a execução da macro TrocaParaMaiusculas, em resposta ao uso da tecla Enter, basta associar uma entrada vazia, com a propriedade OnEntry da planilha Vendas, conforme exemplo a seguir:

```
Worksheets("Vendas").OnEntry = ""
```

Para configurar uma macro que seja executada em resposta ao uso da tecla Enter, em qualquer planilha, você deve associar a macro com a propriedade OnEntry do objeto Application, conforme exemplo a seguir:

```
Application.OnEntry = "TrocaParaMaiusculas"
```

Para desabilitar OnEntry em todas as planilhas, associe um valor vazio á propriedade OnEntry do objeto Application, conforme indicado no exemplo a seguir:

```
Application.OnEntry = ""
```

Como executar uma macro em resposta a um clique duplo:

Para configurar uma macro que seja executada em resposta a um clique duplo do usuário, utilizamos o evento BeforeDoubleClick, do objeto Worksheet. A seguir descrevo os passos para a criação deste evento.

To do this, you must go into the Visual Basic editor, and double-click the worksheet (under the Microsoft Excel Objects folder on the left) that contains the cell you wish to trigger the macro. Then change the (left dropdown to Worksheet and change the right dropdown to BeforeDoubleClick. Then you can just do something similar to the code below to check the range that's passed to the subroutine (this one is set to run if A1 is double-clicked):

Para criar a estrutura do procedimento associado ao evento Worksheet_BeforeDoubleClick, siga os passos indicados a seguir:

1. Abra o Excel.
2. Abra a planilha onde está a planilha a ser criado o procedimento para o exemplo Worksheet_BeforeDoubleClick.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Nas opções que são exibidas do lado esquerdo da tela clique em Plan1 (Plan1) para selecionar esta opção ou clique no nome da planilha a ser utilizada.
5. Na lista Geral, selecione Worksheet e na lista ao lado selecione BeforeDoubleClick. Pode ocorrer de o Editor do VBA ter criado duas declarações, uma para o evento SelectionChange e uma para o Evento BeforeDoubleClick, conforme indicado a seguir:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
```

```
End Sub
```

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
```

```
End Sub
```

6. Exclua a declaração para o evento Worksheet_SelectionChange, mantendo apenas a declaração para o evento BeforeDoubleClick.
7. Agora é só inserir o código para o evento Worksheet_BeforeDoubleClick, o qual será executado, sempre que o usuário der um clique duplo na planilha. No exemplo a seguir, criei um código que exibe o endereço e o valor da célula onde foi efetuado o clique duplo:

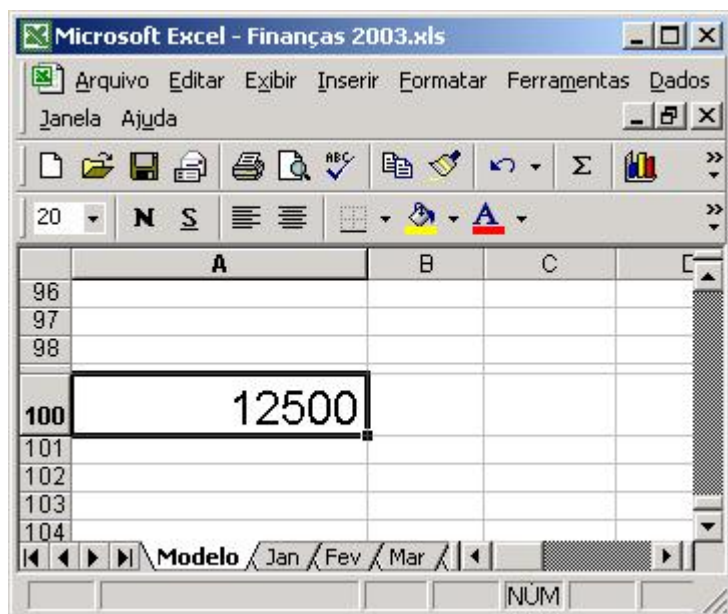

```
Private Sub Worksheet_BeforeDoubleClick( ByVal target As Excel.Range, Cancel As Boolean)
```

```
    ' Exibe o endereço da célula onde foi clicado  
    MsgBox "Endereço da célula clicada: " & target.Address
```

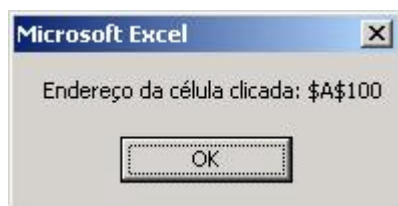
```
    ' Exibe o valor da célula onde foi dado o clique duplo  
    MsgBox "Valor da célula clicada: " & target.Value
```

```
End Sub
```

Para vermos este exemplo em ação, vamos considerar a planilha da figura a seguir, onde temos o valor 12500 na célula A100:



Ao dar um clique duplo na célula A100, será exibida a mensagem a seguir:



Clique em OK. Será exibida a figura a seguir:



Clique em OK para fechá-la.

Lição 23: Exemplos Práticos

Nesta lição continuarei a apresentar uma série de exemplos práticos, os quais utilizam os eventos dos objetos do Excel e também os objetos já estudados até o momento: Application, Workbook, Worksheet e Range.

Como impedir o usuário de fechar um arquivo:

No exemplo a seguir, utilizo o evento BeForeClose para cancelar o fechamento de um arquivo do Excel. Anteriormente, quando falei sobre o evento BeforeClose, do objeto Workbook, apresentei o seguinte texto:

Evento BeforeClose:

Este evento ocorre antes da pasta de trabalho ser fechada. Se a pasta de trabalho tiver sido alterada, esse evento ocorrerá antes do usuário ser solicitado a salvar alterações.

Sintaxe:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
```

- ▶ **Cancel:** False quando o evento ocorre. Quando o procedimento de evento define esse argumento como True, a operação de fechamento para e a pasta de trabalho é deixada aberta.

Observe a descrição do parâmetro Cancel, mais especificamente o trecho a seguir: **... Quando o procedimento de evento define esse argumento como True, a operação de fechamento para e a pasta de trabalho é deixada aberta.**

É exatamente isso que eu utilizo para cancelar o fechamento do arquivo, ou seja, atribuir o valor True, para o parâmetro Cancel, no código do evento BeforeClose, conforme indicado a seguir:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Cancel = True
End Sub
```

Uma vez executado este código, a única maneira de você poder fechar a pasta de trabalho (a não ser usando Ctrl+Alt+Del ou o botão Reset), é desabilitando a geração de eventos, o que é feito com o comando a seguir:

```
Application.EnableEvents = False
```

Ao desabilitar a geração de eventos, o evento BeforeClose não será disparado e o fechamento da pasta de trabalho não será cancelado.

Como impedir o usuário de fechar qualquer arquivo:

A seguir descrevo os passos para criar um procedimento que impedirá o usuário de fechar qualquer um dos arquivos abertos no Excel. Observe que vou utilizar a declaração de Eventos WithEvents, para poder utilizar os eventos do objeto Application, conforme já descrito anteriormente.

Para impedir que o usuário possa fechar qualquer um dos arquivos, siga os passos indicados a seguir:

1. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
2. Selecione o comando Inserir -> Módulo de classe. Um novo módulo, com o nome de Classe1 será inserido.

3. Digite o código indicado a seguir:

```
Public WithEvents xlAppTrap as Excel.Application
```

4. Na caixa de combinação da esquerda, onde aparece Geral, selecione xlAppTrap.

5. Será criada a estrutura de um módulo privado, com as declarações indicadas a seguir:

```
Private Sub xlAppTrap_NewWorkbook(ByVal Wb As Workbook)  
  
End Sub
```

6. Na caixa de combinação da esquerda, selecione o evento WorkbookBeforeClose. A estrutura para o evento WorkbookBeforeClose será criada. Agora você deve estar com a declaração de dois eventos, conforme indicado a seguir:

```
Private Sub xlAppTrap_NewWorkbook(ByVal Wb As Workbook)  
  
End Sub  
  
Private Sub xlAppTrap_WorkbookBeforeClose(ByVal Wb As Workbook, Cancel As Boolean)  
  
End Sub
```

7. Exclua a declaração para o evento NewWorkbook.
8. Para fazer com que o usuário não consiga fechar nenhuma pasta de trabalho aberta no Excel, digite o código a seguir, dentro do evento WorkbookBeforeClose:

```
Cancel=True
```

O passo final é ativar a classe Classe1, fazendo a definição e a inicialização da variável de classe, conforme indicado no próximo passo.

9. Selecione o comando Inserir -> Módulo. Será criado um módulo de código chamado Módulo1. Dê um clique duplo em Módulo1 para selecioná-lo e digite a linha de código a seguir:

```
Public clsAppTrap As New Class1
```

10. No painel da esquerda clique na opção EstaPasta_de_trabalho. Na lista Geral selecione a opção Workbook. Será criada a estrutura do evento Workbook_Open. Digite a seguinte linha de código, dentro do evento Workbook_Open:

```
Set clsAppTrap.AppTrap = Excel.Application
```

11. Quando o código deste exemplo for executado, o usuário será impedido de fechar qualquer pasta de trabalho aberta no Excel, isso porque configuramos o evento WorkbookBeforeClose, do objeto Application, para cancelar o fechamento de qualquer pasta de trabalho – Cancel = True. Para desabilitar a execução deste código você deve utilizar a seguinte linha de código, no evento Auto_Close ou em uma das rotinas executadas pelo VBA:

```
Set clsAppTrap.AppTrap = Nothing
```

Este exemplo é particularmente interessante, pois ele mostra como declarar e utilizar os eventos do objeto Application, usando os passos descritos, teoricamente, em uma das lições anteriores. Neste exemplo, mostrei todos os comandos e declarações necessários para ativar o evento WorkbookBeforeClose, do objeto Application. Para habilitar outros eventos o procedimento é idêntico, bastando substituir o evento WorkbookBeforeClose pelo evento que você deseja ativar.

Stopping Event Looping:

Um cuidado especial que você tem que ter é não fazer com que dentro de um procedimento de evento, o próprio evento seja gerado novamente. Isso criará um loop infinito. Por Exemplo, o evento Worksheet_Change é disparado quando uma alteração é feita na planilha. Se o código dentro do evento fizer outra alteração, esta alteração chamará novamente o evento, o que provocará outra alteração, o que provocará outra chamada do evento e assim indefinidamente, até o computador travar. Para evitar este problema, você pode desabilitar a geração de eventos, no início do procedimento, executar os comandos necessários e depois, no final do procedimento, habilitar novamente a geração de eventos, conforme o exemplo indicado a seguir:

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    Application.EnableEvents = False
    'Código do procedimento.
    Application.EnableEvents = True
End Sub
```

Lição 24: Exemplos Práticos

Nesta lição continuarei a apresentar uma série de exemplos práticos, os quais utilizam os eventos dos objetos do Excel e também os objetos já estudados até o momento: Application, Workbook, Worksheet e Range.

Como detectar quando o usuário altera a célula selecionada:

Para detectar quando o usuário altera a célula atual, isto é, quando o cursor é deslocado de uma célula para outra, você usa o evento SelectionChange, do objeto Worksheet.

No exemplo a seguir, uso o parâmetro Target, do evento SelectionChange, para exibir o endereço da faixa selecionada pelo usuário:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
    MsgBox "Você selecionou a faixa: " & Target.Address(False, False)
End Sub
```

A seguir vamos a um exemplo um pouco mais elaborado. O código do exemplo a seguir, restringe o usuário a selecionar a faixa B5:B10.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
```

```
Dim R As Integer, C As Integer
```

```
    Dim bChangeCell As Boolean
```

```
    ' Desabilita o tratamento de eventos para evitar um loop de eventos
```

```
    Application.EnableEvents = False
```

```
    ' Obtém o número da linha e da coluna da célula ativa e verifica
```

```
    ' se o usuário fez uma seleção fora da faixa permitida: B5:B10
```

```
    ' Se a seleção estiver fora da faixa permitida, a variável bChangeCell
```

```
    ' será definida como True
```

```
    R = ActiveCell.Row
```

```
    C = ActiveCell.Column
```

```
    If R < 5 Then
```

```
        R = 5
```

```
        bChangeCell = True
```

```
    ElseIf R > 10 Then
```

```
        R = 10
```

```
        bChangeCell = True
```

```
    End If
```

```
If C <> 2 Then
C = 2
    bChangeCell = True
End If

' Seleciona as células na faixa permitida e seleciona somente a célula ativa
' se mais que uma célula tiver sido selecionada.

If bChangeCell Then Cells(R, C).Select

If Selection.Cells.Count > 1 Then ActiveCell.Select

' Habilita novamente a geração de eventos
Application.EnableEvents = True

End Sub
```

Como determinar quando uma planilha está selecionada ou quando uma pasta de trabalho está ativa:

O evento `Workbook_SheetActivate` é que detecta quando uma planilha é selecionada. Se você precisa executar código VBA em resposta a seleção de uma planilha, você deve colocar o código VBA a ser executado, no evento `Workbook_SheetActivate`. Considere o exemplo a seguir, o qual exibe o nome da planilha que foi selecionada:

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    MsgBox "Você selecionou a planilha: " & Sh.Name
End Sub
```

Este procedimento exibe o nome da planilha selecionada, quando você se desloca entre as planilhas de uma mesma pasta de trabalho, clicando no nome das planilhas, na parte de baixo da janela. Este evento não é disparado quando você alterna entre diferentes pastas de trabalho do Excel. Por exemplo, você está com a pasta de trabalho `Vendas.xls` aberta. Minimiza ela, vai para a pasta `Contatos.xls` e depois restaura a janela de `Vendas.xls`. Neste situação, o evento `SheetActivate` não é disparado. Nestes casos, ao invés do evento `Workbook_SheetActivate`, você deve utilizar o evento `Activate` do objeto `Workbook`, conforme exemplificado a seguir:

```
Private Sub Workbook_Activate()
    MsgBox "Você está em: " & ActiveWorkbook.Name
End Sub
```

Um pouco mais sobre as macros `Application.Caller` e `OnEntry`:

Quando você utilize `Application.Caller` em uma macro associada com `OnEntry`, `Application.Caller` faz referência a célula que foi modificada.. Considere o exemplo a seguir:

```
Sub TurnOnEntryOn()
    Application.OnEntry = "AppCallerExample"
End Sub
```

O exemplo a seguir ilustra as diferentes propriedades de Application.Caller, propriedades estas que podem ser utilizadas para retornar diferentes informações:

```
Sub AppCallerExample()  
  
    ' Retorna o valor da célula  
    MsgBox Application.Caller.Value  
  
    ' Atribui a faixa da célula modificada a uma var do tipo Range:  
    Dim cell As Range  
    Set cell = Application.Caller  
  
    ' Retorna o endereço da célula que foi modificada  
    MsgBox Application.Caller.Address  
  
    ' Retorna a fórmula contida na célula (se houver uma fórmula),  
    ' incluindo o sinal de igual =  
  
    MsgBox Application.Caller.Formula  
  
End Sub
```

Para desabilitar a propriedade OnEntry, basta associar um valor vazio a esta variável, conforme o exemplo a seguir:

```
Application.OnEntry = ""
```

Muito bem, sobre exemplos de utilização de Eventos no Excel tivemos uma boa dose. No próximo módulo você aprenderá sobre a criação de aplicativos no Excel, baseados nos chamados UserForms.

Lição 25: Resumo do Módulo

Nas lições deste módulo você aprendeu a trabalhar com os objetos Workbook e Worksheet. Apresentei os principais métodos e propriedades destes objetos. Em seguida apresentei o conceito de eventos. Fizemos um estudo dos principais eventos dos objetos Application, Workbook e Worksheet. Para finalizar o módulo, apresentei dezenas de exemplos práticos de utilização de eventos.

Módulo 4 – Estudo dos Objetos Workbook, Worksheet e Eventos

- Lição 01:** Apresentação dos Objetos Workbook e Worksheet
- Lição 02:** Objeto Workbook e Coleção Workbooks – Métodos e Propriedades
- Lição 03:** Objeto Workbook e Coleção Workbooks – Métodos e Propriedades
- Lição 04:** Objeto Workbook e Coleção Workbooks – Métodos e Propriedades
- Lição 05:** Mais Métodos e Propriedades dos Objetos Workbook e Worksheets
- Lição 06:** Mais Métodos e Propriedades dos Objetos Workbook e Worksheets
- Lição 07:** O Objeto WorkSheet – Métodos e Propriedades – Parte 1
- Lição 08:** O Objeto WorkSheet – Métodos e Propriedades – Parte 2
- Lição 09:** O Objeto WorkSheet – Métodos e Propriedades – Parte 3
- Lição 10:** O Objeto WorkSheet – Métodos e Propriedades – Parte 4
- Lição 11:** Eventos – Conceitos e Definições
- Lição 12:** Eventos – Eventos do Objeto Worksheet
- Lição 13:** Eventos – Eventos do Objeto Worksheet
- Lição 14:** Eventos – Eventos do Objeto Workbook
- Lição 15:** Eventos – Eventos do Objeto Workbook
- Lição 16:** Eventos – Eventos do Objeto Application
- Lição 17:** Eventos – Eventos do Objeto Application
- Lição 18:** Exemplos Práticos
- Lição 19:** Exemplos Práticos
- Lição 20:** Exemplos Práticos
- Lição 21:** Exemplos Práticos
- Lição 22:** Exemplos Práticos
- Lição 23:** Exemplos Práticos
- Lição 24:** Exemplos Práticos
- Lição 25:** Resumo do Módulo

No próximo módulo você aprenderá sobre a criação de aplicativos para o Excel, utilizando os chamados UserForms.

Bibliografia recomendada:

Confira as dicas de livros de Excel no seguinte endereço:

<http://www.juliobattisti.com.br/indicados/excel.asp>

Módulo 5 – Criação de Aplicações Usando UserForms

Neste módulo você aprenderá sobre UserForms. Com a utilização de UserForms você pode criar formulários personalizados, semelhantes aos que podem ser criados usando linguagens de programação como o Visual Basic ou Delphi. Mostrarei como criar um UserForm, quais os controles disponíveis e como utilizá-los, como funciona o modelo de eventos do Windows e uma série de exemplos práticos para que você possa começar a criar seus próprios formulários personalizados.

Com o uso de UserForms você pode criar interfaces personalizadas, parecidas com as utilizadas em programas do Windows. Você pode criar um formulários onde são inseridos controles tais como:

- ▶ Caixa de Texto
- ▶ Caixa de Combinação
- ▶ Caixa de listagem
- ▶ Grupo de botões de rádio – Rádio Buttons
- ▶ Grupo de botões de checagem – Check Box
- ▶ Botões de Comando
- ▶ Menus personalizados
- ▶ Controles para imagens

O uso prático de um UserForm é a criação de um formulário que facilita a utilização das planilhas de uma pasta de trabalho do Excel. Por exemplo, você pode criar um formulário onde o usuário seleciona o nome de um cliente em um controle do tipo Caixa de Combinação. Ao selecionar o nome do cliente, no controle Caixa de Combinação, será disparado o Evento Após alterar da Caixa de Combinação. Em resposta a este evento, você pode utilizar código VBA para localizar os dados do cliente selecionado e exibi-los em Caixas de texto no User Form. Este é apenas um pequeno exemplo das possibilidades disponibilizadas pelos UserForms, juntamente com o modelo de Eventos do Windows e a utilização do VBA.

Com o uso destes recursos é possível criar aplicativos bastante sofisticados, usando o Excel. Mesmo com todos estes recursos, sempre é importante lembrar que é um erro querer utilizar o Excel para solucionar todo e qualquer problema. Existe até um ditado bastante popular para descrever esta situação: **“Quando a única ferramenta que você conhece é um martelo, todos os problemas se parecerão com um prego.”**

Ou seja, é um erro querer adaptar todos os problemas para serem resolvidos com o Excel, só porque esta é a ferramenta que você mais domina. Eu já citei no curso de Excel Básico, no curso de Excel Avançado e volto a citar neste curso: Em situações que envolvam um grande volume de registros (milhões de registros), ou um grande número de tabelas ou problemas que envolvem muitos cálculos ou cálculos complexos, a ferramenta mais indicada não é o Excel. Para grandes volumes de dados e para problemas que envolvam um grande número de diferentes tabelas é mais indicado o uso de um banco de dados, como o Microsoft Access ou SQL Server. Em situações que envolvam muitos cálculos e uma lógica complexa, é recomendado o uso de uma linguagem de programação para geração de aplicativos, tais como o Visual Basic, VB.NET, Delphi, C++, C#, Java e assim por diante.

Lição 01: User Form – Introdução e Conceito

Conforme descrito na introdução deste módulo, o uso mais comum e prático para os UserForms é a criação de formulários que atuam como uma interface mais amigável para o usuário utilizar os dados de uma ou mais planilhas de uma pasta de trabalho:

“O uso prático de um UserForm é a criação de um formulário que facilita a utilização das planilhas de uma pasta de trabalho do Excel. Por exemplo, você pode criar um formulário onde o usuário seleciona o nome de um cliente em um controle do tipo Caixa de Combinação. Ao selecionar o nome do cliente, no controle Caixa de Combinação, será disparado o Evento Após alterar da Caixa de Combinação. Em resposta a este evento, você pode utilizar código VBA para localizar os dados do cliente selecionado e exibi-los em Caixas de texto no User Form. Este é apenas um pequeno exemplo das possibilidades disponibilizadas pelos UserForms, juntamente com o modelo de Eventos do Windows e a utilização do VBA.”

Uma situação típica para a utilização de UserForms é quando você precisa criar uma ou mais planilhas, as quais serão utilizadas por funcionários que não foram treinados no Excel e, portanto, não sabem utilizar os recursos de uma planilha do Excel. Nestas situações você cria uma ou mais formulários, usando UserForms, formulários estes que atuam como uma interface entre o usuário final e as planilhas do Excel. Ou seja, o usuário final – que nada conhece do Excel, interage diretamente com os formulários, usando os recursos disponíveis nos formulários, para inserir, pesquisar, classificar e excluir dados.

Na Figura a seguir apresento um exemplo de um formulário para cadastro de Clientes, o qual pode ser criado com a utilização de um UserForm:

Com o uso deste formulário, mesmo um funcionário que não tenha conhecimentos do Excel, será capaz de realizar operações básicas, tais como cadastrar novos clientes, pesquisar o cadastro de um cliente e excluir o cadastro de um cliente.

Você pode inclusive criar formulários mais sofisticados, como o indicado na Figura a seguir, onde foi criado um formulário para cadastro de funcionários. Observe que está disponível um campo para inclusão da foto do funcionário:

Funcionários Nancy Davolio

Informações da Empresa | **Informações Pessoais**

Código do Funcionário: 1

Nome: Nancy

Sobrenome: Davolio

Cargo: Representante de Vendas

Supervisor: Fuller, Andrew

Data de Contratação: 01-05-1992

Ramal: 5467

Registro: 1 de 9

Este formulário também apresenta uma divisão em guias – Informações da Empresa e Informações Pessoais. Esta divisão facilita o agrupamento dos campos de acordo com uma ou mais divisões lógicas. Ao clicar na guia Informações Pessoais, serão exibidos apenas os campos relacionados a este guia, conforme indicado na Figura a seguir:

Funcionários Nancy Davolio

Informações da Empresa | **Informações Pessoais**

Endereço: 507 - 20th Ave. E.
Apt. 24

Cidade: Seattle **Região:** WA

CEP: 98122 **País:** EUA

Telefone Residencial: (206) 555-9857

Forma de Tratamento: Sra.

Data de Nascimento: 08-12-1968

Observações:
Sua educação inclui um bacharelado em Psicologia pela Universidade de Colorado. Nancy é membro do Conselho Internacional de Alimentação.

Registro: 1 de 9

Outro uso bastante comum para os UserForms é para a criação de uma interface personalizada, diferente da interface padrão do Excel. Por exemplo, você pode utilizar o evento Ao Ativar de um UserForm, para substituir a barra de menus padrão do Excel por uma barra de menus Personalizada e para substituir a barra de botões do Excel, por uma barra de botões personalizada.

Observe que, desta maneira, você pode criar um aplicativo bastante parecido com um programa independente do Windows. Claro que com o uso do VBA e UserForms não é possível a criação de programas executáveis, que possam ser instalados e executados sem o uso do Excel. Todo código VBA e todo UserForm no Excel só poderá ser executado dentro do Excel.

Nas próximas lições, você aprenderá a criar um formulário e aprenderá a adicionar controles ao formulário. O próximo passo será aprender a configurar as diversas propriedades de cada controle e a utilizar os eventos associados com o formulário e com os controles do formulário.

De uma maneira resumida, as etapas básicas para a criação e utilização de um formulário, são as seguintes:

- ▶ Inserir um objeto do tipo UserForm na pasta de Trabalho.
- ▶ Criar código VBA para exibir o formulário. Este código normalmente está associado a um Botão de comando na planilha ou a um dos eventos de abertura do objeto Workbook ou do objeto Worksheet. Após ter criado um formulário, você deve utilizar o método Show do UserForm para exibi-lo (conforme você aprenderá nas próximas lições).
- ▶ Adicionar controles ao formulário.
- ▶ Configurar as propriedades de cada controle.
- ▶ Criar o código VBA que fará o tratamento de eventos dos elementos do formulário. Por exemplo, criar código VBA associado com o evento Ao Clicar de um botão de comando, código VBA associado com o evento Após alterar de um controle do tipo Caixa de combinação e assim por diante.

Muito bem, nas próximas lições, através de um exemplo prático, passo-a-passo, você aprenderá a executar todas as etapas necessárias para a criação e configuração de um formulário baseado no objeto UserForm. Mas este já é o assunto para a próxima lição.

Lição 02: User Form – Criando um Novo User Form

Nesta lição você aprenderá a criar um novo UserForm. Você também aprenderá a adicionar um controle do tipo botão de comando. Na próxima lição você aprenderá a configurar os eventos da pasta de trabalho, para que o formulário seja aberto automaticamente, a configurar o evento Ao Clicar do botão de comando, para que ele exiba uma mensagem e a configurar algumas propriedades básicas do formulário e do botão de comando.

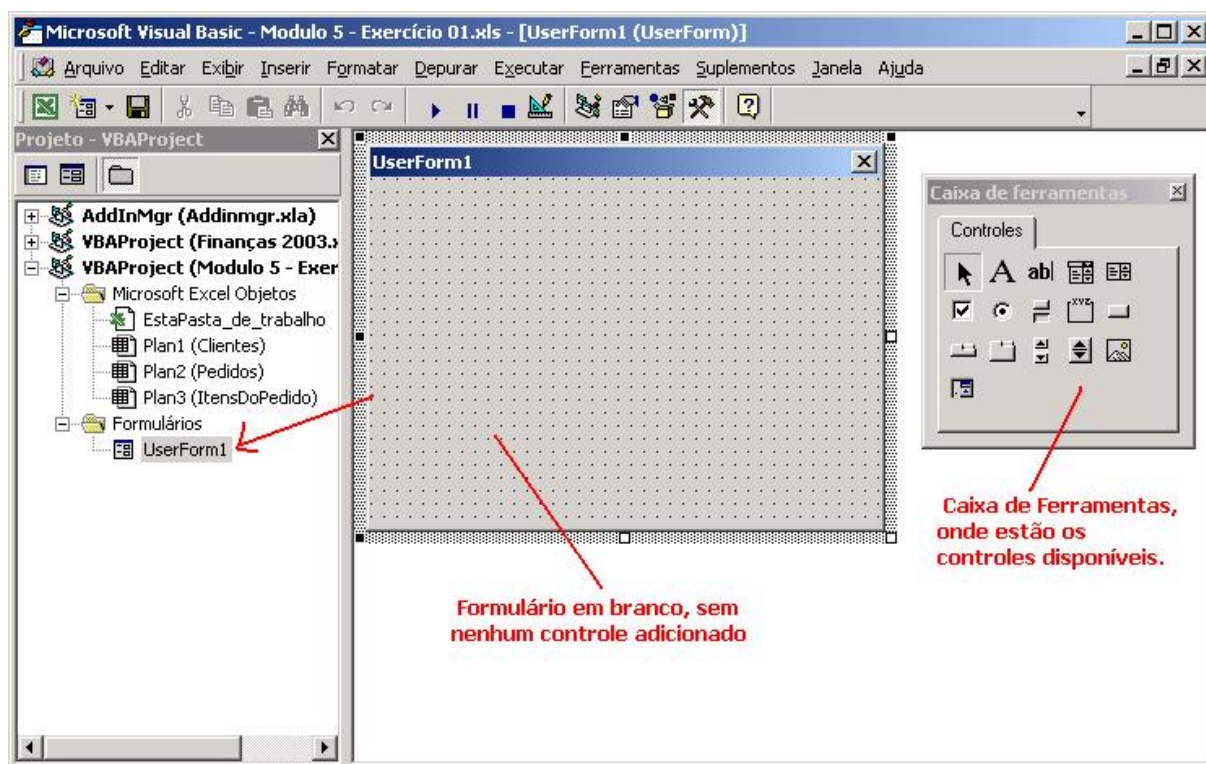
Para o exemplo prático de UserForms, o qual será iniciado nesta lição e continuado nas próximas lições, utilizarei a pasta de trabalho C:\Programação VBA no Excel\ Modulo 5 - Exercício 01.xls. Esta pasta de trabalho tem três planilhas:

- ▶ **Clientes:** Contém um cadastro de clientes da empresa
- ▶ **Pedidos:** Contém um cadastro dos diversos pedidos emitidos para cada cliente.
- ▶ **ItensDoPedido:** Contém uma relação dos itens de cada pedido, sendo que cada item está associado com um pedido através da coluna NúmeroDoPedido.

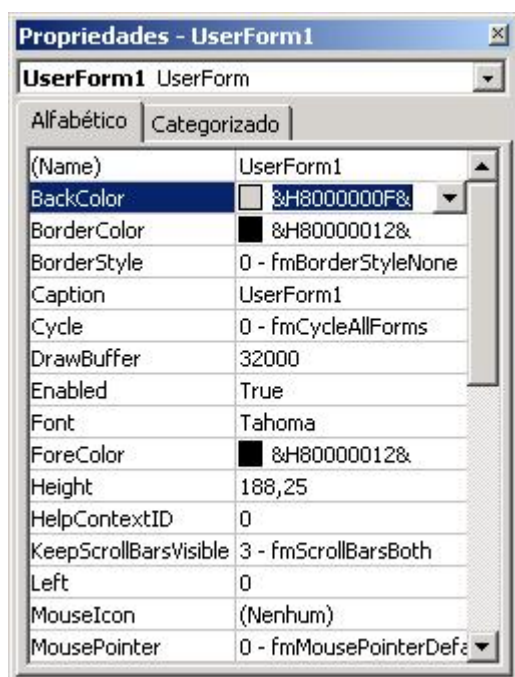
Exemplo: Para criar um UserForm e adicionar um botão de comando, siga os passos indicados a seguir:

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 5 - Exercício 01.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Para criar um novo UserForm, selecione o comando Inserir -> UserForm.
5. Será criado um novo formulário chamado UserForm1. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário. Na Figura da próxima página é exibido o Formulário UserForm1 recém criado, em branco e a Caixa de Ferramentas.
6. Você pode redimensionar o formulário da mesma maneira como redimensiona qualquer janela do Windows. Basta dar um clique no formulário para selecioná-lo, o que será indicado pelos quadriculados que serão exibidos no contorno do formulário. Depois de selecionado, basta apontar o mouse para um dos quadriculados da borda, clicar com o botão esquerdo do mouse e mantê-lo pressionado e arrastar. À medida que você arrasta, o formulário vai sendo redimensionado.
7. O UserForm, bem como cada controle que for adicionado ao UserForm, contém uma série de propriedades que você pode configurar. Por exemplo, o texto que é exibido na barra de títulos do formulário é uma propriedade do UserForm, a qual pode ser alterada.

8. Por padrão a janela de propriedades não é exibida. Para exibi-la clique com o botão direito do mouse em qualquer local do formulário. No menu de opções que é exibido clique em Propriedades. A janela de Propriedades – indicada na segunda figura da próxima página, passará a ser exibida.



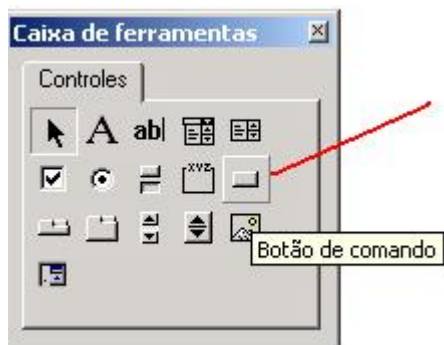
Formulário UserForm1 recém criado e a Caixa de ferramentas.



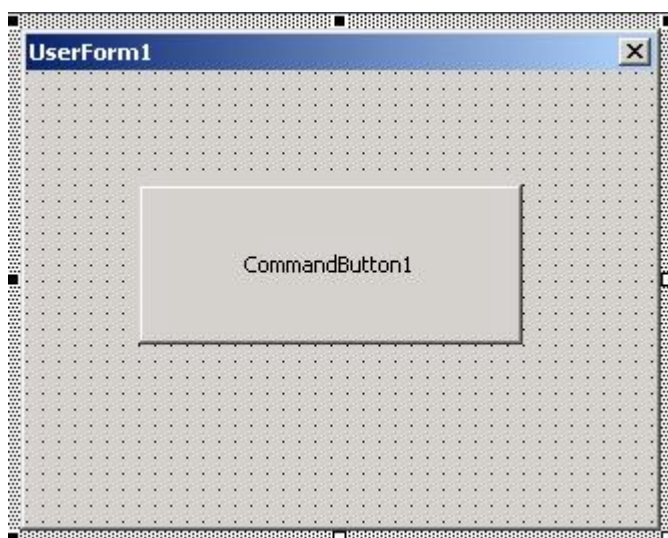
Janela de propriedades do formulário UserForm1.

9. Agora vamos adicionar um botão de comando ao formulário UserForm1 e vamos configurar o evento Ao Clicar deste botão, para exibir a famosa mensagem: “Hello World!!!”.

10. Ao posicionar o mouse em cada controle da Caixa de Ferramentas e manter o ponteiro sobre um controle, por mais de dois segundos, será exibida uma descrição com o nome do controle. Posicione o mouse sobre o controle Botão de Comando, indicado na Figura a seguir:




11. Clique neste controle. Mova o ponteiro do mouse para o formulário, mais ou menos na posição onde você pretende inserir o botão de comando. Observe que o ponteiro do mouse se transforma em uma pequena cruz. Clique com o botão esquerdo do mouse e mantenha este botão pressionado e arraste para formar um retângulo mais ou menos do tamanho que você deseja para o botão de comando que está sendo criado. Libere o botão do mouse e pronto, o botão de comando terá sido adicionado ao formulário, conforme indicado na figura a seguir:



Importante: O procedimento para adicionar os demais controles é exatamente ou mesmo, ou seja, clica no respectivo controle na Caixa de Ferramentas, depois clica no formulário e arrasta um retângulo para definir o tamanho do controle. Ao invés de clicar e arrastar no formulário, você pode simplesmente clicar. Neste caso o controle será inserido de acordo com o tamanho padrão definido no Excel, para cada controle. Após ter adicionado um controle, você pode redimensioná-lo, da mesma maneira que redimensiona o formulário. Ou seja, clica no controle a ser redimensionado para selecioná-lo. Após ter selecionado o controle, aponte o mouse para um dos quadradinhos da borda do controle, mantenha o mouse pressionado e arraste para redimensionar o respectivo controle.

12. Observe que a exemplo do formulário – que foi criado com o nome padrão de UserForm1, o Excel também atribui um nome padrão para cada controle que está sendo adicionado ao formulário. Para o botão de comando do nosso exemplo, foi adicionado o nome padrão CommandButton1. O próximo botão de comando a ser adicionado, receberia o nome de CommandButton2 e assim por diante. É possível alterar o nome padrão do formulário e de qualquer controle inserido no formulário. Esta é uma das configurações que você aprenderá na próxima lição.

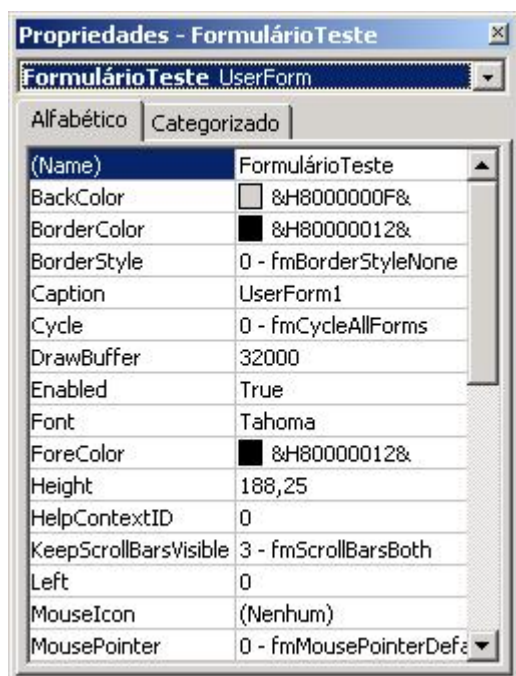
13. Vamos salvar o trabalho feito até agora. Clique no botão Salvar () e mantenha o Editor do VBA aberto, pois continuaremos a alterar o formulário e os seus controles na próxima lição.

Lição 03: User Form – Propriedades e Eventos

Nesta lição mostrarei como alterar o nome do formulário, o nome do botão de comando e como configurar algumas das principais propriedades do formulário e do botão de comando. Também mostrarei como criar um procedimento de evento associado com o botão de comando e como configurar o formulário para que seja exibido, automaticamente, durante a abertura da pasta de trabalho.

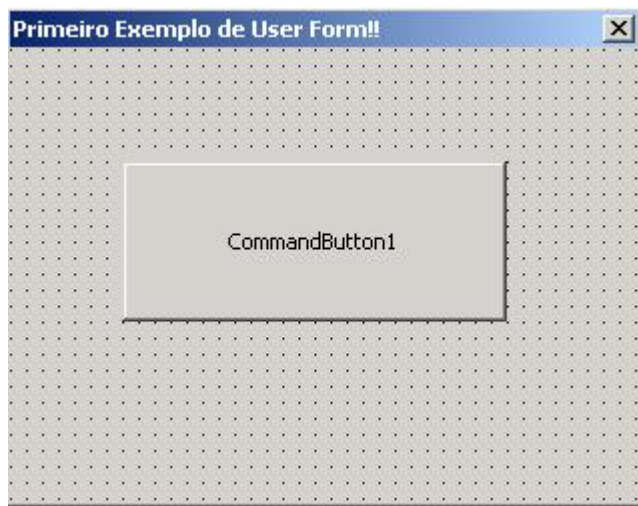
Para alterar as propriedades do formulário e do botão de comando, siga os passos indicados a seguir:

1. Se a janela de Propriedades não estiver sendo exibida, selecione o comando Exibir -> Janela 'Propriedades' ou pressione a tecla F4.
2. Agora vamos alterar o nome do formulário e o texto que é exibido na barra de títulos do formulário.
3. Clique em qualquer local do formulário que não seja no Botão de Comando. Isso selecionará o formulário. Se você clicar no botão de comando, será selecionado o respectivo botão e a janela de propriedades irá exibir as propriedades do botão. Em resumo, a janela de propriedades exibe as propriedades do elemento atualmente selecionado.
4. O nome do formulário é definido pela propriedade Name. Clique no valor UserForm1, ao lado da propriedade Name, exclua UserForm1 e digite FormulárioTeste, conforme indicado na figura a seguir:



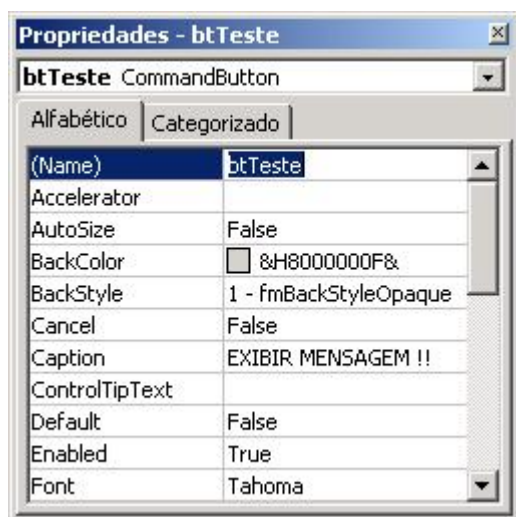
5. Observe, no lado esquerdo do Editor do VBA, onde está a opção Projeto – VBAProject, que o nome do formulário já é exibido como FormulárioTeste.
6. O próximo passo é alterar o texto da Barra de Títulos do formulário. Este texto é definido na propriedade Caption. Clique nesta propriedade, exclua o valor atual UserForm1 e digite: Primeiro Exemplo de User Form!! e pressione a tecla Backspace.

7. Observe que logo após pressionar a tecla TAB, o valor da propriedade Caption é definido e o novo texto já é exibido na barra de títulos do formulário, conforme indicado na figura a seguir:

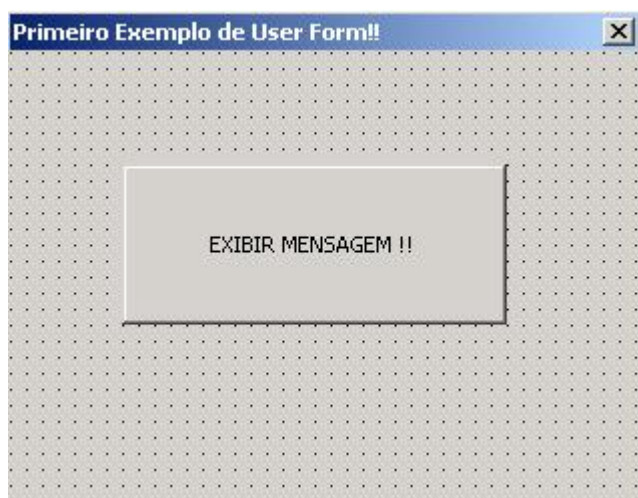


8. Agora vamos alterar as propriedades Name e Caption do controle Botão de Comando. Clique no controle CommandButton1 para selecioná-lo. Isso fará com que as propriedades do botão CommandButton1 estejam acessíveis através da janela Propriedades.

9. Altere a propriedade Name para btTeste e a propriedade Caption para EXIBIR MENSAGEM !! A janela de propriedades deve estar conforme indicado a seguir:



10. Após ter alterado as propriedades Name e Caption do botão de comando, o seu formulário deverá estar conforme indicado na figura a seguir:



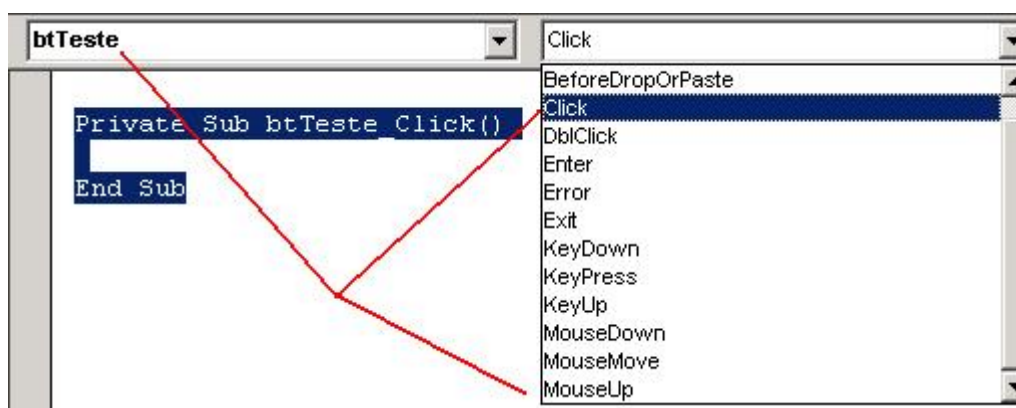
Para criar um procedimento de evento, associado ao evento Ao Clicar do botão de comando, siga os passos indicados a seguir:

1. Clique com o botão direito do mouse no botão de comando e, no menu de opções que é exibido, clique em Exibir código.
2. Será criada a estrutura do procedimento associado ao evento Ao Clicar do botão de comando, conforme indicado a seguir:

```
Private Sub btTeste_Click()
```


```
End Sub
```

Nota: Você pode associar procedimentos de eventos com outros eventos de um botão de comando, além do evento Click. Para isso, basta selecionar o respectivo evento, na lista de eventos do lado direito, conforme indicado na figura a seguir:



3. O código do nosso exemplo será extremamente simples. Será apenas um comando MsgBox, conforme indicado a seguir:

```
Private Sub btTeste_Click()  
    MsgBox "Hello World !!"  
End Sub
```

4. Vamos salvar o trabalho feito até agora. Clique no botão Salvar () . Na árvore de opções do Painel da esquerda, dê um clique duplo em FormulárioTeste, para voltar a exibir o formulário.
5. Muito bem, o nosso primeiro formulário já foi criado, suas propriedades foram configuradas e foi adicionado um botão de comando. As propriedades do botão de comando foram configuradas e foi associado um procedimento de evento ao botão de comando. Agora só nos resta configurar o evento `Workbook_Open()`, para que abra, automaticamente, o formulário FormulárioTeste. É o que faremos a seguir.


Para configurar o evento `Workbook_Open()`, para exibir o formulário FormulárioTeste automaticamente, siga os passos indicados a seguir:

1. Na árvore de opções do Painel da esquerda, dê um clique duplo em EstaPasta_de_trabalho, para selecionar o módulo de código da pasta de trabalho atual.
2. Na lista da esquerda, onde aparece Geral, selecione Workbook. Por padrão, já será criada a estrutura para o evento `Workbook_Open`, conforme indicado a seguir:

```
Private Sub Workbook_Open()  
  
End Sub
```

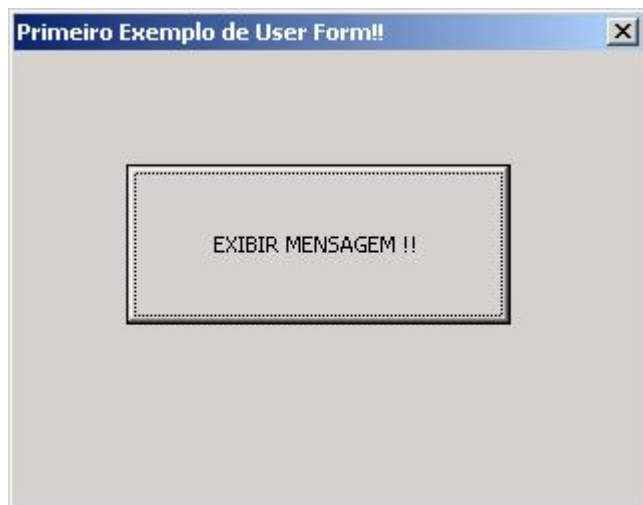
3. Para que o formulário FormulárioTeste seja aberto automaticamente, basta chamar o método `Show` do formulário, dentro do procedimento `Workbook_Open`, conforme indicado a seguir:

```
Private Sub Workbook_Open()  
    FormulárioTeste.Show  
End Sub
```

4. Vamos salvar o trabalho feito até agora. Clique no botão Salvar () .
5. Muito bem, agora é hora de testar o nosso “Obra de arte” (nem tanto).

Para testar a execução do formulário FormulárioTeste, siga os passos indicados a seguir:

1. Feche a pasta de trabalho Modulo 5 - Exercício 01.xls.
2. Abra a pasta de trabalho Modulo 5 - Exercício 01.xls.
3. Se você receber um aviso de segurança, dizendo que existem macros na Pasta de trabalho, clique no botão Ativar macros. A pasta de trabalho será aberta e o formulário FormulárioTeste será exibido, conforme indicado na primeira figura da próxima página.
4. Dê um clique no botão EXIBIR MENSAGEM !!.
5. O evento Ao Clicar do botão será disparado e a Caixa de mensagem indicada na segunda figura, da próxima página, será exibida. Clique em OK para fechar a mensagem.
6. Muito bem, isso comprova que o nosso primeiro formulário está funcionando corretamente. Embora tenha sido um exemplo extremamente simples, serviu para mostrar os princípios básicos de criação de um formulário, a adição de controles, a configuração de propriedades do formulário e dos controles e também mostrou como criar código associado aos eventos de um controle do formulário. A partir da próxima lição, estudaremos os controles em mais detalhes.



O formulário FormulárioTeste exibido automaticamente.



A mensagem exibida pelo evento Ao Clicar do botão de comando.

Lição 04: User Form – Trabalhando com a Caixa de Ferramentas

Nesta e na próxima lição apresentarei alguns detalhes sobre a Caixa de Ferramentas, que é a barra de ferramentas onde estão os botões para acesso a todos os controles disponíveis.

A caixa de ferramentas está indicada na Figura a seguir:



O que é a Caixa de ferramentas?

A Caixa de ferramentas identifica os diferentes controles que você pode adicionar a um formulário. É uma barra de ferramentas como outra qualquer do Excel, onde cada botão representa um controle disponível.

Você pode personalizar a Caixa de ferramentas de várias maneiras, incluindo as seguintes:

- ▶ Adicionando páginas à Caixa de ferramentas.
- ▶ Movendo controles de uma página para outra.
- ▶ Renomeando páginas.
- ▶ Adicionando outros controles à Caixa de ferramentas, inclusive controles ActiveX.
- ▶ Copiando controles personalizados do formulário para a Caixa de ferramentas.

Por exemplo, os botões OK e Cancelar são casos especiais de um CommandButton. Se você adicionar modelos OK e Cancelar à Caixa de ferramentas, você poderá adicioná-los rapidamente a outros formulários.

Como Exibir ou ocultar a Caixa de ferramentas:

No editor do VBA, no menu Exibir, verifique se há uma marca de verificação (um sinalzinho de certo) na frente da opção Caixa de ferramentas. Se a marca de verificação estiver presente, a Caixa de ferramentas estará sendo exibida. Caso contrário, a Caixa de ferramentas estará oculta.

Para exibir a Caixa de ferramentas, certifique-se de que uma marca de verificação esteja aparecendo na frente de Caixa de ferramentas. Se não estiver, selecione o comando Exibir -> Caixa de ferramentas.

Para ocultar a Caixa de ferramentas, certifique-se de que não haja marca de verificação na frente de Caixa de ferramentas. Se houver, selecione o comando Exibir -> Caixa de ferramentas para removê-la.

Como eu faço para alterar o tamanho da Caixa de ferramentas:

1. Posicione o ponteiro do mouse sobre uma borda ou um canto da Caixa de ferramentas. O ponteiro mudará para uma seta de duas pontas.
2. Clique e Arraste o mouse para alterar seu tamanho.

Como eu faço para adicionar um controle à Caixa de ferramentas:

1. Clique com o botão direito do mouse em qualquer ícone de controle da Caixa de ferramentas ou em uma área vazia de qualquer página da Caixa de ferramentas.
2. No menu que é exibido clique em Controles adicionais.
3. A partir da lista Controles adicionais, selecione os novos controles.
4. Clique em OK.
5. Pronto, os novos controles selecionados passarão a estar disponíveis na Caixa de Ferramentas. A quantidade de controles disponíveis varia de computador para computador, dependendo da instalação do Excel e do número de Suplementos (Add-ins) instalados.

Como adicionar um controle a um formulário:

Utilize um dos métodos a seguir para adicionar um controle de uma Caixa de ferramentas a seu formulário.

1. Clique em um controle na Caixa de ferramentas e, em seguida, clique no formulário. O controle aparece em seu tamanho padrão. Você pode, então, arrastar o controle para alterar seu tamanho.

ou

2. Arraste um controle da Caixa de ferramentas para o formulário. O controle aparece em seu tamanho padrão.

ou

3. Clique duas vezes no controle da Caixa de ferramentas e, em seguida, clique no formulário uma vez para cada controle que você deseja criar. Por exemplo, para criar quatro botões de comando, clique duas vezes em CommandButton, na Caixa de ferramentas, e, em seguida, clique quatro vezes no formulário.

Como adicionar um controle personalizado à Caixa de ferramentas:

Você pode criar controles personalizados. Por exemplo, você pode querer criar um botão de comando que sempre vem com o rótulo Pesquisar Dados, ao invés de um rótulo padrão definido pelo Excel. Para criar controles personalizados, siga os passos indicados a seguir:

1. Insira um controle em seu formulário e personalize-o. Por exemplo, para criar um botão OK, insira um CommandButton no formulário, defina sua propriedade Caption como OK e sua propriedade Default como True.
2. Selecione o controle personalizado.

3. Arraste o controle para a Caixa de ferramentas.

Observação: Ao arrastar um controle para a Caixa de ferramentas, você transfere somente valores de propriedades. Qualquer código que você tenha escrito para esse controle não é transferido junto com ele. É preciso escrever um novo código para o ícone ou copiar o código do controle do formulário para o controle da Caixa de ferramentas.

Como excluir um item da Caixa de ferramentas:

1. Na Caixa de ferramentas, clique com o botão direito do mouse no ícone do item que você deseja remover.
3. No menu de opções que é exibido dê um clique em Excluir.

Observação: Se você estiver excluindo controles, poderá utilizar Controles adicionais a partir do menu de atalho e desmarcar as caixas de seleção de todos os controles que deseja excluir.

Como personalizar um ícone da Caixa de ferramentas:

1. Clique com o botão direito do mouse no ícone da Caixa de ferramentas.
2. No menu de opções que é exibido clique em Personalizar.

Proceda a uma das seguintes instruções:

3. Para alterar a Descrição de ferramenta, insira o novo texto desta.
ou
4. Para editar o ícone, escolha Editar figura. Em seguida, escolha a cor que você deseja utilizar e escolha o pixel da imagem no qual você deseja aplicar essa cor.
ou
5. Para atribuir um novo bitmap, escolha Carregar figura. Então, identifique o arquivo contendo o bitmap que você deseja utilizar como ícone. Caso tente carregar uma figura que seja maior que um ícone, ocorrerá um erro.

Como personalizar uma Descrição de ferramenta na Caixa de ferramentas:

1. Selecione o controle na Caixa de ferramentas.
2. Clique com o botão direito do mouse.
3. No menu de opções que é exibido clique em Personalizar. O comando Personalizar incluirá o nome do controle, como "Personalizar rótulo".
4. Insira o novo texto para a Descrição de ferramenta.
5. Clique em OK.

Nota: Este texto é aquela pequena dica que é exibida quando você mantém o botão do mouse apontado para o controle, durante cerca de dois segundos.

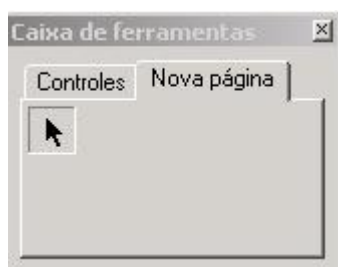
Lição 05 User Form – Trabalhando com a Caixa de Ferramentas

Nesta lição apresentarei mais alguns detalhes de operações com a Caixa de Ferramentas.

Como criar uma nova página da Caixa de ferramentas:

A Caixa de Ferramentas pode ser formada por uma ou mais guias. Cada guia é conhecida como uma página da Caixa de Ferramentas. Por exemplo, além da guia padrão, onde estão disponíveis os controles padrão do Excel, você poderia criar uma nova guia chamada Personalizada, onde você adiciona controles personalizados. A seguir mostro os passos para criar uma nova guia (também chamada de página), na Caixa de Ferramentas:

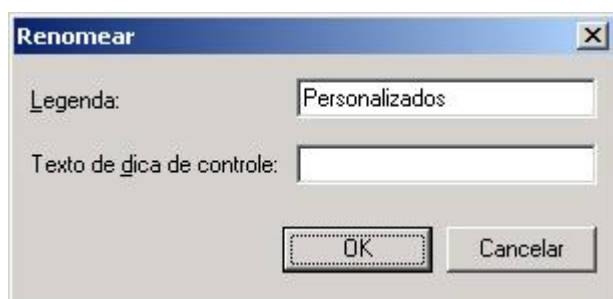
1. Clique com o botão direito do mouse na guia de qualquer página da Caixa de ferramentas. A nova página será inserida após essa página.
2. No menu de opções que é exibido clique em Nova página.
3. Será criada uma nova página, chamada Nova página, conforme indicado na Figura a seguir:



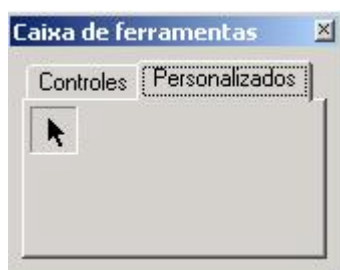
A seguir você aprenderá a renomear, adicionar controles, excluir e a realizar uma série de operações com as páginas da Caixa de Ferramentas.

Como renomear uma página da Caixa de ferramentas:

1. Clique com o botão direito do mouse no nome da Página a ser renomeada. No nosso exemplo, clique com o botão direito do mouse em Nova página.
2. No menu de opções que é exibido clique em Renomear.
3. Será exibida a janela Renomear. No campo Legenda, digite o novo nome da página, como por exemplo Personalizados, conforme indicado na figura a seguir e clique em OK.



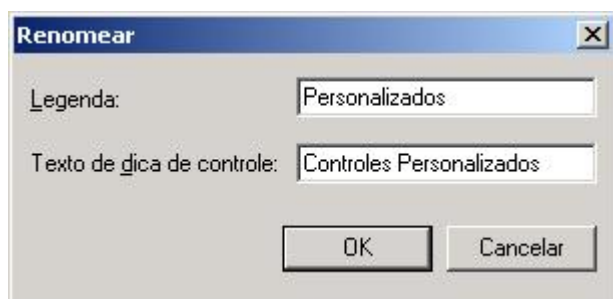
4. A página será renomeada para Personalizadas, conforme indicado na figura a seguir:



Como definir a Descrição de ferramenta para uma página da Caixa de ferramentas:

A descrição é uma dica que é exibida quando você deixa o mouse por aproximadamente dois segundos sobre o nome da página. Por exemplo, vamos adicionar a descrição Controles Personalizados, à página Personalizados, criada anteriormente. Para isso, siga os passos indicados a seguir:

1. Selecione a página da Caixa de ferramentas.
2. Clique com o botão direito do mouse.
3. A partir do menu de atalho, escolha Renomear.
4. Será exibida a janela Renomear. No campo Texto de dica de controle digite Controles Personalizados, conforme indicado a seguir:



5. Clique em OK.
6. Agora aponte o mouse para o nome da página e mantenha o mouse sobre o nome por aproximadamente dois segundos. Será exibido o texto Controles Personalizados, conforme indicado na figura a seguir:



Como alterar a ordem das páginas da Caixa de ferramentas:

1. Clique com o botão direito do mouse na guia de qualquer página da Caixa de ferramentas.

2. A partir do menu de atalho, escolha Mover.
3. Selecione o nome de uma página que você deseja mover.
4. Escolha Mover para cima ou Mover para baixo até a página ficar na posição apropriada na lista de páginas.
5. Clique em OK.

Como excluir uma página da Caixa de ferramentas:

1. Clique com o botão direito do mouse na guia da página da Caixa de ferramentas que você deseja excluir.
2. Escolha Excluir página. Todos os controle da página serão excluídos ao mesmo tempo.

Como importar ou exportar uma página da Caixa de ferramentas:

1. Clique com o botão direito do mouse na guia de qualquer página da Caixa de ferramentas. Se você importar uma página, ela será inserida após essa página.

Proceda a uma das seguintes instruções:

2. Para importar uma página, escolha Importar página. Em seguida, selecione o nome do arquivo da página que você deseja importar.
3. Para exportar uma página, escolha Exportar página. Em seguida, insira um nome para o arquivo que irá armazenar uma cópia da página da Caixa de ferramentas. A exportação de uma página não a remove da Caixa de ferramentas.
4. Clique em OK.

Como mover um item para uma outra página da Caixa de ferramentas:

1. Selecione um controle de qualquer página da Caixa de ferramentas.
2. Arraste o controle para a guia da nova página. Mantenha o ponteiro do mouse sobre a guia até a página aparecer na frente da Caixa de ferramentas.
3. Arraste o controle para a região principal da página.

Observação: Se a página na qual você deseja inserir o controle não estiver visível, você poderá aumentar a largura da Caixa de ferramentas a fim de exibir guias para todas as páginas e, em seguida, arrastar o controle para a página apropriada.

Muito bem, sobre a Caixa de Ferramentas era isso. A partir da próxima lição, vamos estudar os principais controles que podem ser inseridos em um User Form.

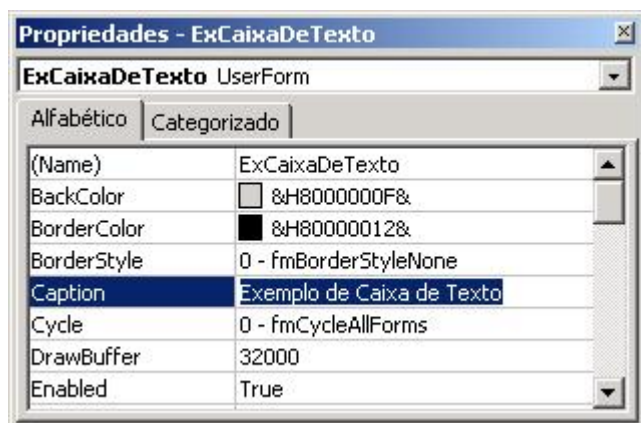
Lição 06: User Form – Trabalhando com Controles – Parte 1

A partir desta lição passaremos a estudar os controles que podem ser adicionados a um formulário. Apresentarei uma descrição da função do controle, as principais propriedades e eventos e um exemplo simples de utilização.

Nesta lição mostrarei como adicionar controles do tipo Caixa de Texto, como configurar as principais propriedades deste tipo de controle. Na próxima lição você aprenderá a alinhar controles e a definir propriedades que afetam as características visuais dos controles. O próximo passo será aprender a criar código VBA para validação dos dados que são inseridos no controle, mas isso já será assunto para a lição 8. Vamos a um exemplo prático.

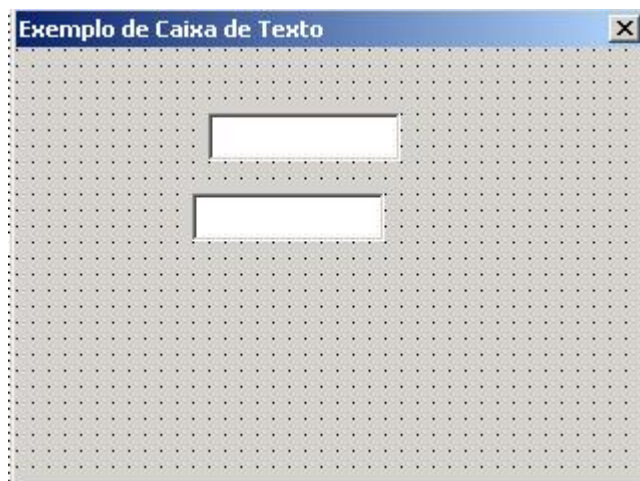
Exemplo: Criar um formulário chamada ExCaixaDeTexto e adicionar controles do tipo Caixa de Texto:

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 5 - Exercício 02.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Para criar um novo UserForm, selecione o comando Inserir -> UserForm.
5. Será criado um novo formulário chamado UserForm1. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário.
6. Clique no formulário para selecioná-lo. Se a janela de Propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la.
7. Na janela de propriedades do formulário, localize a propriedade Name e altere o seu valor para ExCaixaDeTexto e altere o valor da propriedade Caption para Exemplo de Caixa de Texto. A janela de propriedades deverá estar conforme indicado na figura a seguir:

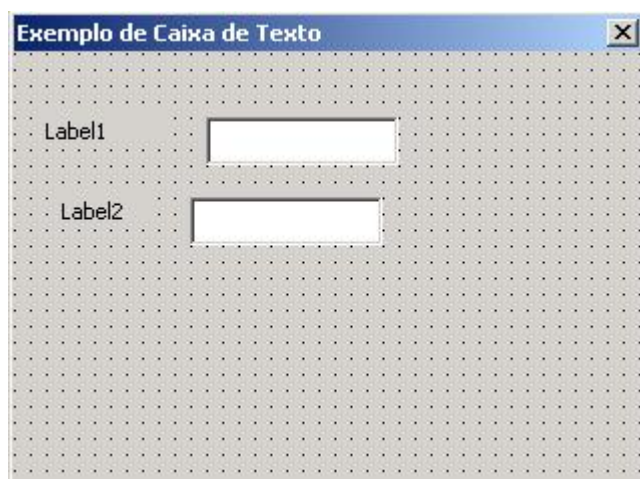


8. Muito bem, agora já criamos o formulário e definimos algumas de suas propriedades. Agora vamos adicionar dois controles do tipo Caixa de Texto e configurar algumas de suas propriedades.
9. Se a Caixa de Ferramentas não estiver sendo exibida, selecione o comando Exibir – Caixa de ferramentas.

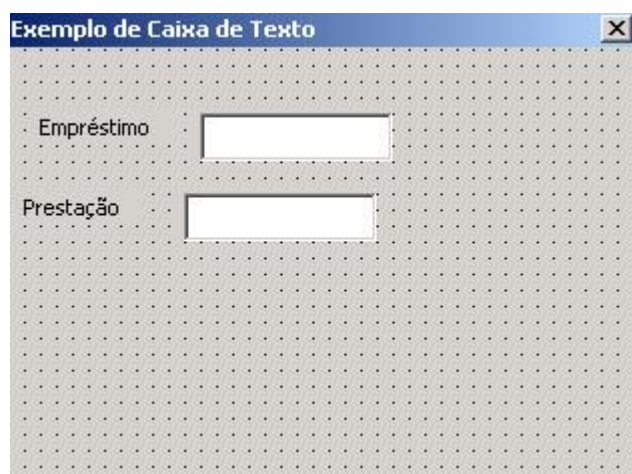
10. Clique no controle Caixa de texto (**ab**) e clique no formulário, mais ou menos na posição onde deve ser inserido o primeiro controle do tipo Caixa de texto.
11. Clique no controle Caixa de texto (**ab**) e clique no formulário, mais ou menos na posição onde deve ser inserido o segundo controle do tipo Caixa de texto.
12. Seu formulário deve estar, mais ou menos conforme indicado na figura a seguir:



13. Além dos controles Caixa de texto é recomendado que você também adicione controles do tipo Rótulo. Os controles do tipo Rótulo normalmente são adicionados à esquerda dos controles Caixa de texto e servem como uma identificação do controle Caixa de texto. Ao lado do primeiro controle Caixa de texto, vamos adicionar um rótulo chamado Empréstimo e à esquerda do segundo controle Caixa de texto, vamos adicionar um rótulo chamado Prestação.
14. Na Caixa de ferramentas, clique no controle Rótulo (**A**) e clique no formulário, à esquerda do primeiro controle do tipo Caixa de texto.
15. Na Caixa de ferramentas, clique no controle Rótulo (**A**) e clique no formulário, à esquerda do segundo controle do tipo Caixa de texto.
16. O seu formulário deve estar, mais ou menos conforme indicado na Figura a seguir. Não se preocupe com o alinhamento dos controles, você irá aprender a alinhá-los na próxima lição.



17. Agora vamos configura algumas propriedades dos controles inseridos no formulário. Vamos começar alterando o texto dos controles do tipo Rótulo. O padrão é Label1, Label2 e assim por diante. Vamos alterar de Label1 para Empréstimo e vamos alterar Label 2 para Prestação.
18. Clique no controle Label1 para selecioná-lo. Se a janela de propriedades não estiver sendo exibida, tecle F4 para exibi-la. Altere a propriedade name para Empréstimo e a propriedade Caption para Empréstimo. A propriedade Name define o nome do controle, nome o qual pode ser utilizado no código VBA para fazer referência ao controle. A propriedade Caption define o texto que é exibido no controle.
19. Clique no controle Label2 para selecioná-lo. Se a janela de propriedades não estiver sendo exibida, tecle F4 para exibi-la. Altere a propriedade name para Empréstimo e a propriedade Caption para Empréstimo.
20. Agora vamos alterar algumas propriedades dos controles do tipo Caixa de Texto. Clique na primeira Caixa de texto para selecioná-la. Altere a propriedade Name para vlEmpréstimo.
21. Clique na segunda Caixa de texto para selecioná-la. Altere a propriedade Name para vlPrestação.
22. Sua janela deve estar conforme indicado na Figura a seguir:



23. Muito bem, já criamos um novo formulário, adicionamos controles do tipo Caixa de texto e do tipo Rótulo e também alteramos algumas propriedades destes controles. Mantenha este formulário aberto, pois na próxima lição você aprenderá a alinhar os controles e a alterar propriedades que alteram as características visuais dos controles.

Lição 07: User Form – Trabalhando com Controles – Parte 2

Nesta lição continuaremos a trabalhar no exemplo criado na lição anterior. Inicialmente aprenderemos a alinhar os controles em um formulário. O primeiro passo para aprender a alinhar e a dimensionar os controles é aprender como fazer para selecionar vários controles ao mesmo tempo.

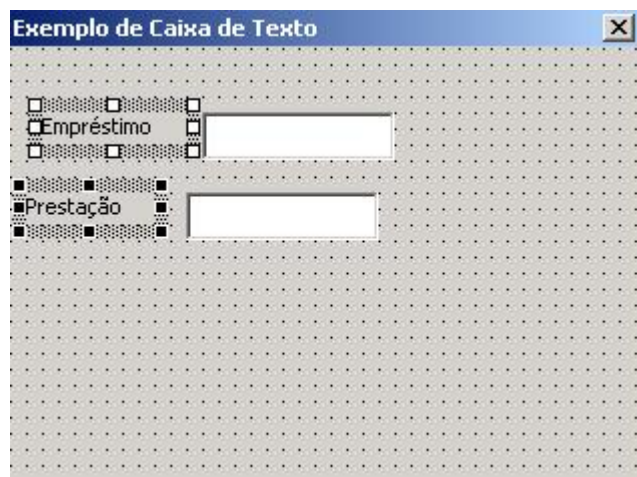
Como selecionar vários controles em um formulário:

1. Pressione a tecla Shift e mantenha-a pressionada.
2. Enquanto a tecla Shift estiver pressionada vá clicando nos controles a serem selecionados. Cada controle que você clicar, mantendo Shift pressionada, será selecionado.

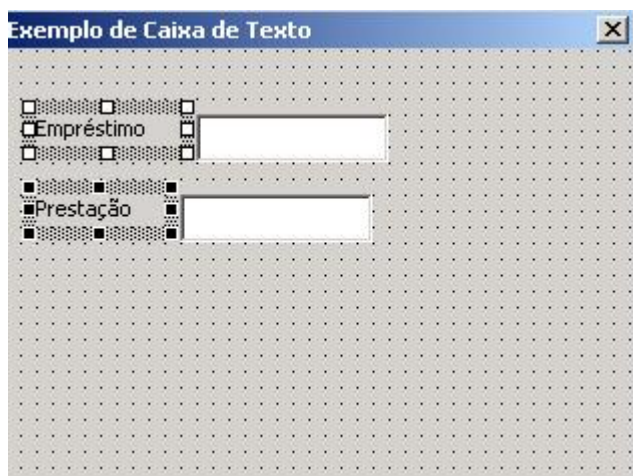
Como alinhar controles no formulário:

Agora vamos aprender a fazer o alinhamento (à direita, à esquerda, superior e inferior) dos controles do formulário.

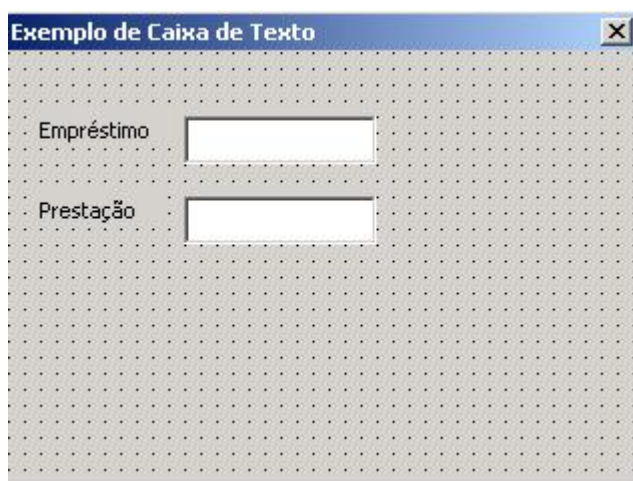
1. Pressione a tecla Shift, mantenha-a pressionada.
2. Clique no rótulo Empréstimo para selecioná-lo.
3. Ainda com a tecla Shift pressionada, clique no rótulo prestação. Observe que o rótulo Prestação é selecionado e o rótulo Empréstimo, selecionado anteriormente, também permanece selecionado, conforme indicado na Figura a seguir:



4. Vamos alinhar os dois controles do tipo rótulo, à esquerda. Para tal selecione o comando: Formatar -> Alinhar -> Esquerdas. Os dois controles do tipo rótulo serão alinhados à esquerda, conforme indicado na primeira figura da próxima página. Através do menu Formatar -> Alinhar, também estão disponíveis as opções para Alinhar à direita, Alinhar Superior, Alinhar Inferior, dentre outras opções. Os comandos de alinhamento somente estarão habilitados quando dois ou mais controles estiverem selecionados. Alinhar significa alinhar dois ou mais controles em relação ao que está mais à esquerda, ou em relação ao que está mais à direita, ou em relação ao que está mais acima ou em relação ao que está mais abaixo.



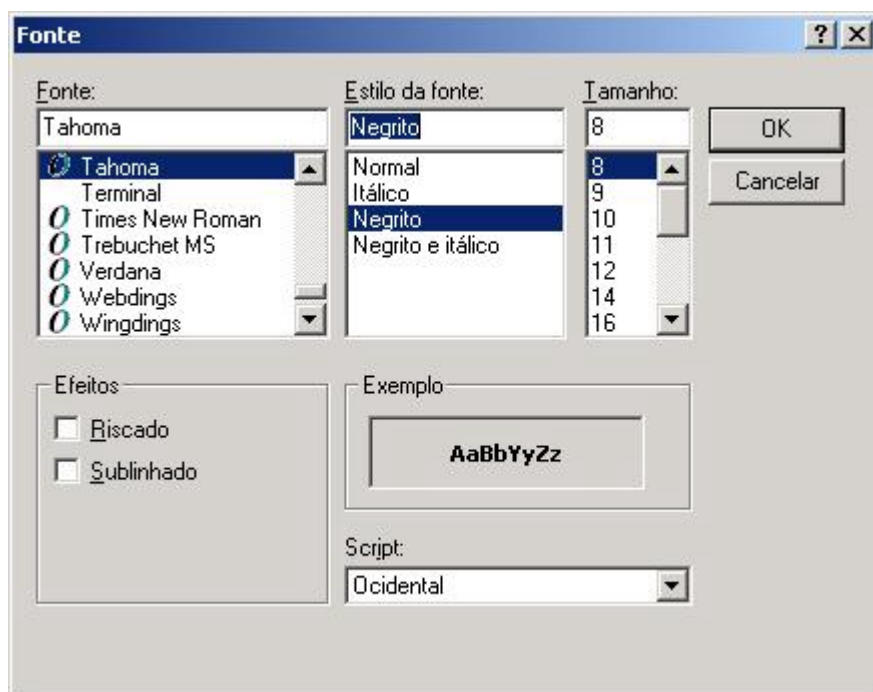
5. Agora vamos alinhar os dois controles do tipo Caixa de texto, à direita.
6. Clique no formulário, fora de qualquer controle, para desfazer a seleção dos dois controles do tipo Rótulo. Pressione a tecla Shift, mantenha-a pressionada.
7. Clique na primeira Caixa de texto para selecioná-la.
8. Ainda com a tecla Shift pressionada, clique na segunda Caixa de texto. Observe que ambas permanecem selecionadas.
9. Vamos alinhar os dois controles Caixa de texto, à direita. Para tal selecione o comando: Formatar -> Alinhar -> Direitas. Os dois controles do tipo Caixa de texto serão alinhados à direita, conforme indicado na figura a seguir:



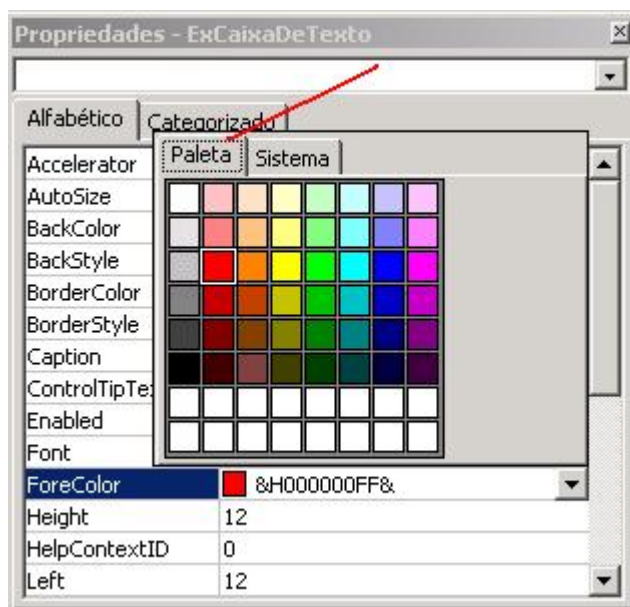
Muito bem, o nosso formulário já está ficando com um aspecto um pouco melhor. Os controles já estão alinhados entre si. O próximo passo será alinhar o texto dos rótulos à esquerda, dentro do rótulo e alterar a fonte e a cor da letra dos rótulos, bem como aplicar um bordo nos controles do tipo rótulo. Faremos isso logo a seguir, aprendendo a configurar mais algumas propriedades dos controles do tipo Rótulo. Você pode configurar as propriedades de vários controles ao mesmo tempo, para isso basta selecionar os controles antes de alterar as propriedades. É isso que faremos logo a seguir.

Alterando propriedades dos controles do tipo Rótulo:

1. Pressione a tecla Shift, mantenha-a pressionada.
2. Clique no rótulo Empréstimo para selecioná-lo.
3. Ainda com a tecla Shift pressionada, clique no rótulo Prestação para selecioná-lo. Observe que ambos permanecem selecionados.
4. Se a janela de propriedades não estiver sendo exibida, tecle F4 para exibi-la. Na janela de Propriedades serão exibidas apenas as propriedades que podem ser configuradas ao mesmo tempo, para mais de um controle. Por exemplo, a propriedade Name não será exibida, quando houver mais de um controle selecionado, pois esta propriedade é diferente para cada controle.
5. Para definir uma borda para os controles do tipo rótulo, você deve alterar a propriedade BorderStyle. Por padrão, esta propriedade tem o valor 0 – fmBorderStyleNone, que significa Sem bordas. Abra a lista de opções da propriedade BorderStyle e selecione a opção 1 – fmBorderStyleSingle, que significa Com bordas.
6. Vamos definir negrito para a fonte dos rótulos e uma cor para a fonte dos rótulos. Para definir negrito, localize a propriedade Fonte, clique no campo desta propriedade. Será exibido um botão com reticências ... Clique neste botão. Será exibida a janela Fonte. Clique na opção Negrito, conforme indicado na Figura a seguir e clique em OK. Nesta janela você também pode definir o tamanho da fonte, o tipo de fonte e efeitos de Riscado e Sublinhado.



7. Agora vamos definir uma cor para a fonte. A cor da fonte é definida na propriedade ForeColor. Esta propriedade é uma lista. Clique na setinha para baixo, para abrir a lista de opções disponíveis nesta propriedade. Na janela que é exibida, clique em Paleta, conforme indicado na Figura a seguir. Depois é só clicar na cor desejada. Para o nosso exemplo, vamos utilizar uma cor de fonte Vermelha. Clique na cor vermelha e pronto, a paleta de cores será fechada e a cor de fonte vermelha já será aplicada a fonte dos controles selecionados.



8. Agora vamos alterar a cor de segundo plano para os controles selecionados. A cor de segundo plano é alterada através da propriedade `BackColor`. Esta propriedade é idêntica a propriedade `ForeColor`. Abra a lista de opções da propriedade `BackColor`, clique em `Paleta` e clique na cor branca. Muito bem, agora os controles do tipo rótulo já deverão estar com bordas, com fonte em negrito e de cor vermelha e com uma cor de fundo branca, conforme indicado na Figura a seguir:



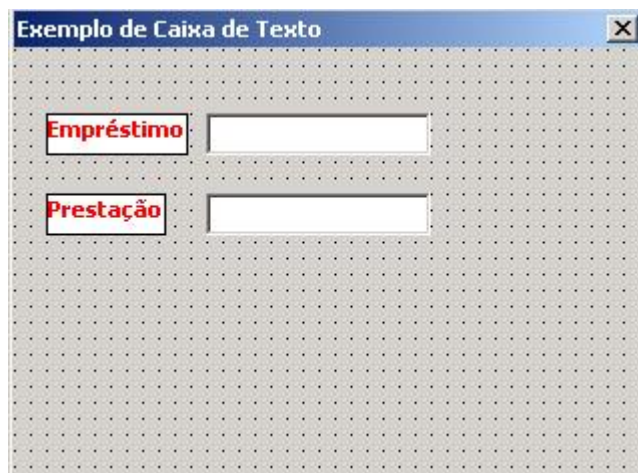
Muito bem, por esta lição já está OK. Na próxima lição você continuará a alterar o visual do nosso formulário e aprenderá a configurar outras propriedades dos controles. Mantenha o formulário aberto, pois iremos utilizá-lo na próxima lição.

Lição 08: User Form – Trabalhando com Controles – Parte 3

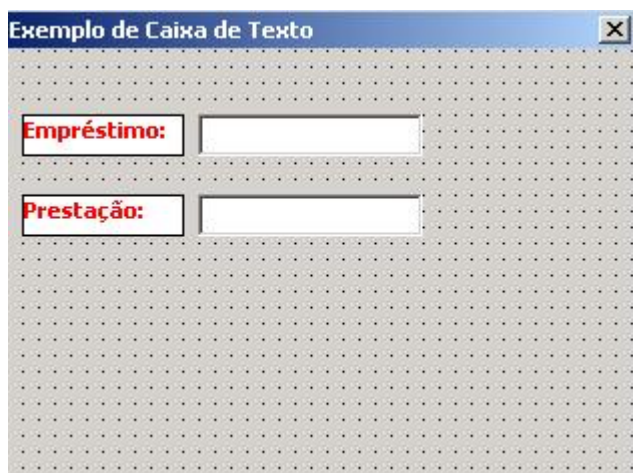
Nesta lição continuaremos a trabalhar no exemplo criado anteriormente. Inicialmente aprenderemos a dimensionar os controles e a alinhar o texto dentro do controle. Em seguida falarei sobre mais algumas propriedades dos controles.

Exemplo: Configurando propriedades dos controles:

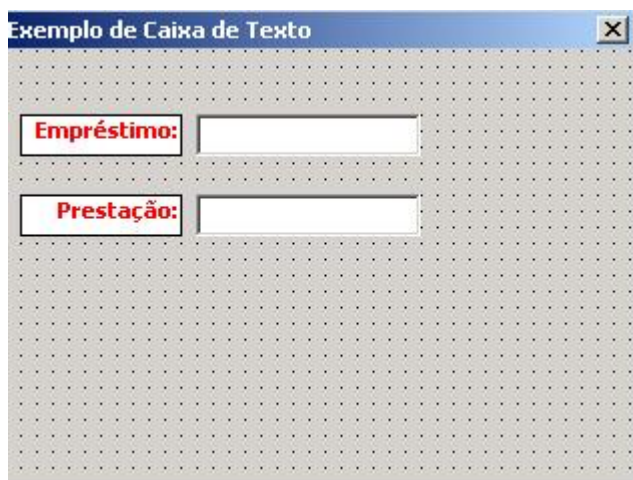
1. Você deve estar com o formulário ExCaixaDeTexto aberto.
2. Selecione os dois controles do tipo rótulo (Empréstimo e Prestação) os dois controles do tipo Caixa de Texto. O nosso objetivo é dimensionar todos os controles com a mesma altura. Deixaremos todos com a altura do controle mais alto.
3. Após ter selecionado os quatro controles, verifique se a janela Propriedades está sendo exibida. Se a janela Propriedades não estiver sendo exibida, tecle F4 para exibi-la. Na janela de propriedades, localize a propriedade Height, que é quem controla a altura dos controles. Digite o valor 16 e pressione a tecla TAB.
4. Pronto, agora todos os controles estão com a mesma altura, conforme indicado na Figura a seguir:



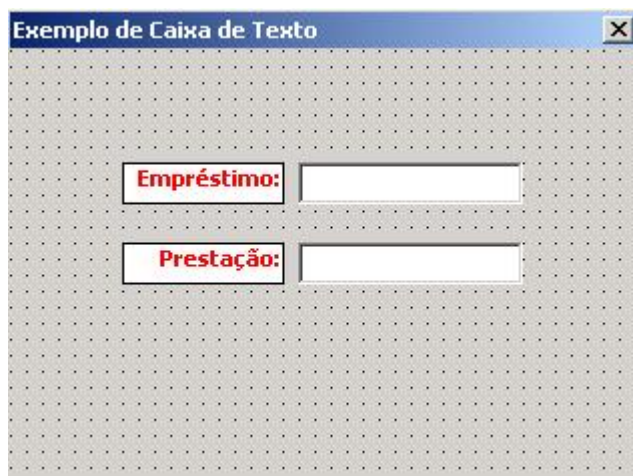
5. Agora vamos definir a largura dos controles do tipo Rótulo. A largura é controlada pela propriedade Width.
6. Selecione os dois controles do tipo Rótulo. Na janela de propriedades localize a propriedade Width, digite 60 e pressione TAB. Agora os dois controles do tipo rótulo estarão com a mesma largura. O próximo passo é alinhar estes dois controles à direita.
7. Selecione os dois controles do tipo rótulo e em seguida selecione o comando Formatar -> Alinhar -> Direitas.
8. Muito bem, agora o nosso formulário já está com um visual bem melhor, conforme indicado na figura a seguir:



9. Agora vamos alinhar o texto dos controles do tipo rótulo, à direita, dentro do rótulo.
10. Selecione os dois controles do tipo rótulo.
11. Na janela de Propriedades (se a janela de Propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la), localize a propriedade TextAlign. Esta propriedade define o alinhamento do texto, dentro do controle.
12. Para fazer o alinhamento à esquerda, selecione a opção 3 – fmTextAlignRight, para esta propriedade. Clique no formulário em qualquer espaço fora dos controles, para desfazer a seleção. Pronto, o texto já está alinhado à direita, conforme indicado na figura a seguir:



13. Agora vamos aprender mais um recurso relacionado ao alinhamento dos controles. Selecione os quatro controles do formulário.
14. Estando os quatro controles selecionados, clique com o mouse no meio de um dos controles (por exemplo, no espaço em branco de um dos controles do tipo Caixa de texto), mantenha o botão do mouse pressionado e arraste os controles para uma posição mais centralizada no formulário. A medida que você arrasta com o mouse, todos os controles selecionados, são movidos para a nova posição.
15. O seu formulário deve estar com um aspecto semelhante ao indicado na Figura a seguir:



16. Muito bem, aos poucos já estamos melhorando a aparência do nosso formulário.

17. Agora veremos um detalhe muito importante. A medida que vamos adicionando controles em um formulário, o Excel vai associando um valor para a propriedade `TabIndex` de cada controle. Esta propriedade define a ordem em que os controles serão percorridos, a medida que o usuário pressiona a tecla `TAB`. O primeiro controle a ser adicionado fica com um valor de `TabIndex = 0`, o segundo com `TabIndex = 1` e assim por diante. Quando o formulário é carregado, o foco estará no controle com `TabIndex = 0`. Quando o usuário pressionar a tecla `TAB`, o foco será deslocado para o controle com `TabIndex = 1`. Pressionando `TAB` novamente, o foco se deslocará para o controle com `TabIndex = 2` e assim por diante. Além da propriedade `TabIndex`, existe também uma propriedade chamada `TabStop`. Esta propriedade define se um controle receberá ou não o foco quando o usuário utiliza a tecla `TAB`. Por padrão, os controles do tipo Rótulo, devem ser configurados com a propriedade `TabStop` com o valor `False`, indicando que os controles do tipo Rótulo não devem receber o foco.

18. No exemplo do nosso formulário, você deve configurar os seguintes valores, para as propriedades `TabIndex` e `TabStop` dos controles do formulário:

Controle	Tipo	TabIndex	TabStop
vEmpréstimo	Caixa de Texto	0	True
vPrestação	Caixa de Texto	1	True
Empréstimo	Rótulo	2	True
Prestação	Rótulo	3	True

19. Clique no controle `vEmpréstimo` para selecioná-lo e configure as propriedades `TabIndex` e `TabStop`, conforme indicado na tabela anterior. Repita esta operação para os demais controles do formulário, configurando os valores das propriedades `TabIndex` e `TabStop`, conforme indicado na tabela anterior.

Muito bem, já aprendemos a configurar uma série de propriedades dos controles Rótulo e Caixa de texto. Na próxima lição, vamos mostrar como usar os eventos dos controles e do formulário, para criar a lógica de processamento do formulário, usando programação VBA.

Lição 09: User Form – Trabalhando com Controles – Parte 4

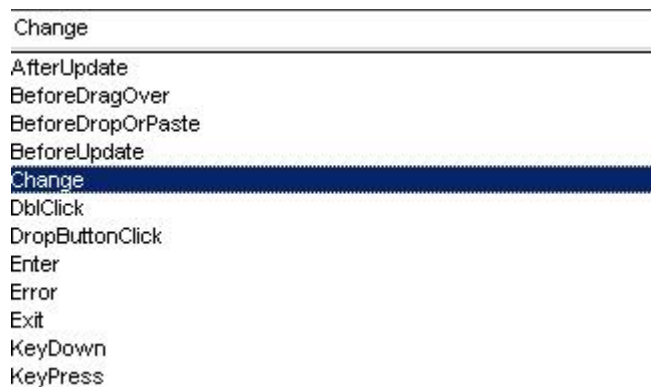
Nesta lição você aprenderá a criar código em resposta a eventos do formulário. Vamos criar dois procedimentos de evento, um associado com cada Caixa de Texto. Os procedimentos irão verificar se os valores digitados na caixa de texto são numéricos e se estão dentro de um intervalo definido. Para o nosso exemplo, iremos definir os limites indicados na tabela a seguir:

Campo	Regra de Validação
vlEmpréstimo	Entre 1000 e 15.000
vlPrestação	Entre 200 e 1000

O nosso objetivo é criar código VBA que garanta que os campos Empréstimo e Prestação não aceitem entradas fora das respectivas faixas, definidas na tabela anterior e que também não aceitem entradas inválidas, tais como uma entrada de texto. Então vamos a criação do código para validação das entradas nos campos Empréstimo e Prestação.

Exemplo: Criar os códigos de validação para os campos vlEmpréstimo e vlPrestação

1. Você deve estar com o formulário ExCaixaDeTexto aberto.
2. Dê um clique duplo no campo vlEmpréstimo, que é o primeiro campo do tipo Caixa de Texto.
3. Será aberto o editor do VBA, com a estrutura para o evento Change, do campo vlEmpréstimo. Este é o evento padrão para um campo do tipo Caixa de Texto. O evento Change ocorre sempre que o valor do campo é alterado.
4. Na lista do lado direito você pode selecionar outros eventos disponíveis para um campo do tipo Caixa de texto, conforme indicado na Figura a seguir:



5. Para o nosso exemplo vamos utilizar o evento AfterUpdate, o qual é disparado após o usuário ter alterado o valor do campo e pressionado TAB para ir para o próximo campo. Na lista de eventos, selecione o evento AfterUpdate. Exclua a estrutura que foi criada para o evento Change. A estrutura básica para este evento AfterUpdate será criada, conforme indicado a seguir:

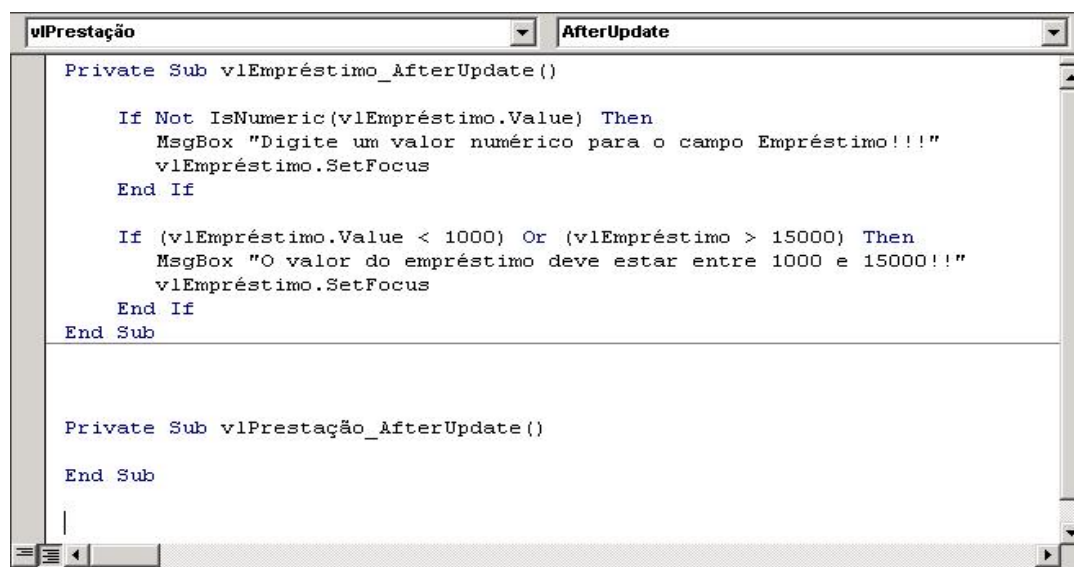
```
Private Sub vlEmpréstimo_AfterUpdate()  
  
End Sub
```

6. Agora vamos inserir o código de validação que, basicamente é uma estrutura IF, a qual testa se o valor é numérico e se está dentro da faixa definida para o campo. Em caso afirmativo, nenhuma ação é executada. Em caso negativo, uma mensagem é exibida e o cursor é posicionado novamente no campo vlEmpréstimo, para que o usuário corrija o valor digitado. A seguir indico o código de validação para o campo vlEmpréstimo;

```
Private Sub vlEmpréstimo_AfterUpdate()  
  
    If Not IsNumeric(vlEmpréstimo.Value) Then  
        MsgBox "Digite um valor numérico para o campo Empréstimo!!!"  
        vlEmpréstimo.Value = ""  
        vlEmpréstimo.SetFocus  
        Exit Sub  
    End If  
  
    If (vlEmpréstimo.Value < 1000) Or (vlEmpréstimo > 15000) Then  
        MsgBox "O valor do empréstimo deve estar entre 1000 e 15000!!"  
        vlEmpréstimo.Value = ""  
        vlEmpréstimo.SetFocus  
        Exit Sub  
    End If  
End Sub
```

7. Um detalhe a comentar é que no código VBA, você faz referência a um controle do formulário, usando o nome do controle (nome definido na propriedade Name do controle). No nosso exemplo, estou fazendo referência ao controle vlEmpréstimo. Cada controle é um objeto, o qual tem diversos métodos e propriedades. No nosso exemplo, utilizo a propriedade Value, para acessar o valor inserido pelo usuário, no controle. Também utilizo a propriedade SetFocus, para colocar o cursor de volta no campo, caso a entrada tenha sido inválida.

8. Agora vamos criar o código de validação para o controle vlPrestação. Na lista da esquerda, onde está sendo exibido vlEmpréstimo, selecione vlPrestação. Automaticamente será criada a estrutura para o evento Change, do controle vlPrestação. Na lista de eventos, no lado direito, selecione o evento AfterUpdate. Com isso também será criada a estrutura para o evento AfterUpdate, do controle vlPrestação. Exclua a estrutura para o evento Change, do controle vlPrestação. A sua janela deverá estar conforme indicado a seguir:



9. Agora vamos inserir o código de validação para o controle vlPrestação, o qual é, basicamente é uma estrutura IF, a qual testa se o valor é numérico e se está dentro da faixa definida para o campo. Em caso afirmativo, nenhuma ação é executada. Em caso negativo, uma mensagem é exibida e o cursor é posicionado novamente no campo vlPrestação, para que o usuário corrija o valor digitado. A seguir indico o código de validação para o campo vlEmpréstimo;

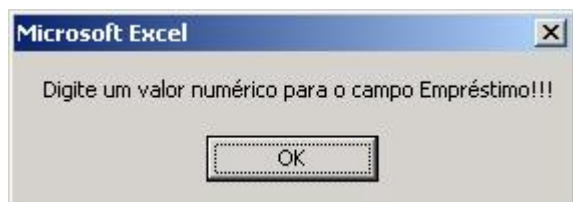
```
Private Sub vlPrestação_AfterUpdate()  
  
    If Not IsNumeric(vlPrestação.Value) Then  
        MsgBox "Digite um valor numérico para o campo Prestação!!!"  
        vlPrestação.Value = ""  
        vlPrestação.SetFocus  
        Exit Sub  
    End If  
  
    If (vlPrestação.Value < 200) Or (vlEmpréstimo > 1000) Then  
        MsgBox "O valor da prestação deve estar entre 200 e 1000!!"  
        vlPrestação.Value = ""  
        vlPrestação.SetFocus  
        Exit Sub  
    End If  
  
End Sub
```

10. Obviamente que no nosso exemplo, não estamos efetuando nenhum cálculo com os valores dos campos vlEmpréstimo e vlPrestação. O objetivo do exemplo é ensinar a trabalhar com controles em um formulário, configurar as propriedades dos controles, formatá-los, alinhá-los e aprender a inserir código VBA em resposta a eventos dos controles.

11. Clique no botão (📁) para salvar o código VBA e feche o editor do VBA. Você estará de volta ao formulário. Agora podemos executar o formulário para testar se o código de validação está funcionando corretamente.

12. Para executar o formulário pressione a tecla F5. O formulário será carregado. No campo empréstimo, vamos digitar um valor de texto, para verificar se o código de validação irá funcionar corretamente. No campo empréstimo, digite a palavra Teste, conforme indicado na Figura a seguir:

13. Pressione a tecla TAB e observe. O código de validação, associado ao evento AfterUpdate do controle é executado e a mensagem de erro a seguir é exibida:



14. Clique em OK. Observe que o valor digitado no campo Empréstimo é apagado. Agora digite um valor fora da faixa permitida para o campo Empréstimo, por exemplo digite 20000 (lembrando que a faixa permitida está entre 1000 e 15000). Digite 20000 no campo Empréstimo e pressione a tecla TAB. Será exibida a mensagem indicada na figura a seguir:



15. Clique em OK e digite um valor válido, por exemplo 5000 e pressione a tecla TAB. Agora sim, o valor é aceito e o cursor se desloca para o campo vlPrestação.

16. Faça testes também com o campo vlPrestação, digitando valores de texto e valores fora e dentro da faixa permitida e observe as mensagens de erro que são exibidas. Isso comprova que o código de validação está funcionando corretamente.

Neste exemplo, utilizei os controles Rótulo e Caixa de Texto. A partir da próxima lição, passaremos ao estudo dos demais controles disponíveis na Caixa de Ferramentas. Vamos iniciar o estudo pelo controle Caixa de combinação. Mas este já é um assunto para a próxima lição.

Lição 10: User Form – Caixa de Combinação

A partir desta lição passaremos a estudar os demais controles disponíveis na Caixa de ferramentas. Para acompanhar os exemplos de cada lição, você pode criar um novo formulário e inserir o controle neste novo formulário. Para facilitar o acompanhamento dos exemplos, a seguir relembro os passos necessários para a criação de um novo formulário.

Passos para a criação de um novo formulário:

1. Abra o Excel.
2. Abra a planilha na qual você deseja criar o formulário.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Para criar um novo formulário, selecione o comando Inserir -> UserForm.
5. Será criado um novo formulário chamado UserForm1 ou UserForm2 e assim por diante. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário.
6. Clique no formulário para selecioná-lo. Se a janela de Propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la.
7. Na janela de propriedades do formulário, localize a propriedade Name e altere o seu valor para o nome que você deseja dar ao formulário e altere o valor da propriedade Caption para o texto que você quer que seja exibido na barra de títulos do formulário.

Pronto, estes são os passos para a criação de um novo formulário. Nos exemplos desta e da próxima lição, vou fazer referência a estes passos, simplesmente usando a expressão: Crie um novo formulário com o nome de ...

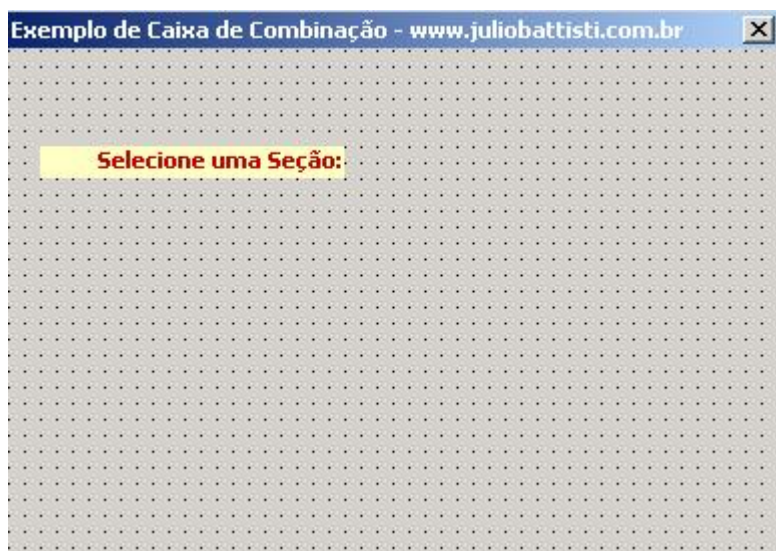
Muito bem, vamos a um exemplo prático de criação e utilização de um controle do tipo Caixa de Combinação. Um controle do tipo Caixa de combinação (conhecido como ComboBox) é utilizado para exibir uma lista de opções. Estes controles são indicados para campos onde existe um conjunto de valores possíveis e limitado. Nestas situações, é muito mais prático para o usuário selecionar um valor em uma pequena lista, do que ter que digitar o valor. Além disso, o fato do usuário selecionar o valor em uma lista, praticamente elimina a possibilidade de erros de digitação. No nosso exemplo, vamos criar um controle do tipo Caixa de Combinação, no qual serão listadas as seções da empresa:

- ▶ Administração
- ▶ Auditoria
- ▶ Contabilidade
- ▶ Finanças
- ▶ Informática
- ▶ Marketing
- ▶ Pesquisa e Desenvolvimento
- ▶ Vendas

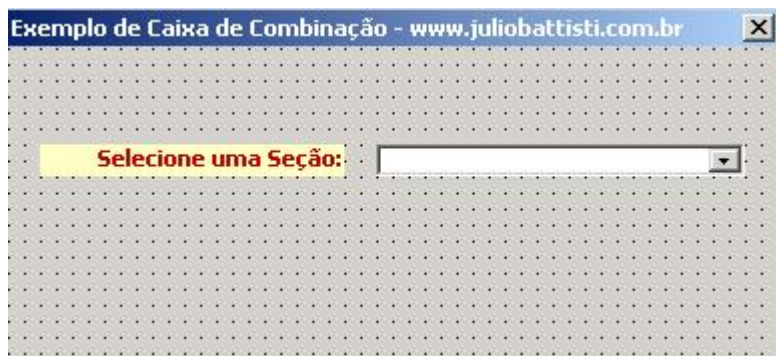
Além do controle do tipo Caixa de combinação, também adicionaremos um controle do tipo rótulo, para identificar o controle Caixa de combinação. Então, mãos a obra.

Exemplo: Criar um formulário chamado ExCaixaCombinação e colocar no formulário um controle do tipo Rótulo e um controle do tipo Caixa de combinação, no qual serão exibidos os nomes de seção, indicados na lista anterior.

1. Crie um novo formulário, chamado ExCaixaCombinação.
2. Adicione um controle do tipo rótulo (conforme explicado em lições anteriores). Altere a propriedade Name do controle para rtListaSeção e a propriedade Caption para Selecione uma Seção: Dimensione o controle de tal maneira que todo o texto da propriedade Caption possa ser exibido. Altere a fonte do rótulo para negrito, a cor da fonte para vermelho e a cor de fundo do rótulo para amarelo (a configuração destas propriedades foi detalhada nas lições anteriores). Seu formulário deverá estar conforme indicado na Figura a seguir:



3. Agora vamos adicionar e configurar um controle do tipo Caixa de Combinação. Na Caixa de ferramentas, clique no botão (ícone de caixa de combinação). Clique no formulário, à direita do rótulo. Será criado um controle do tipo Caixa de combinação, no tamanho padrão. Você pode redimensionar o controle para o tamanho desejado. Seu formulário deve estar semelhante ao indicado na Figura a seguir:



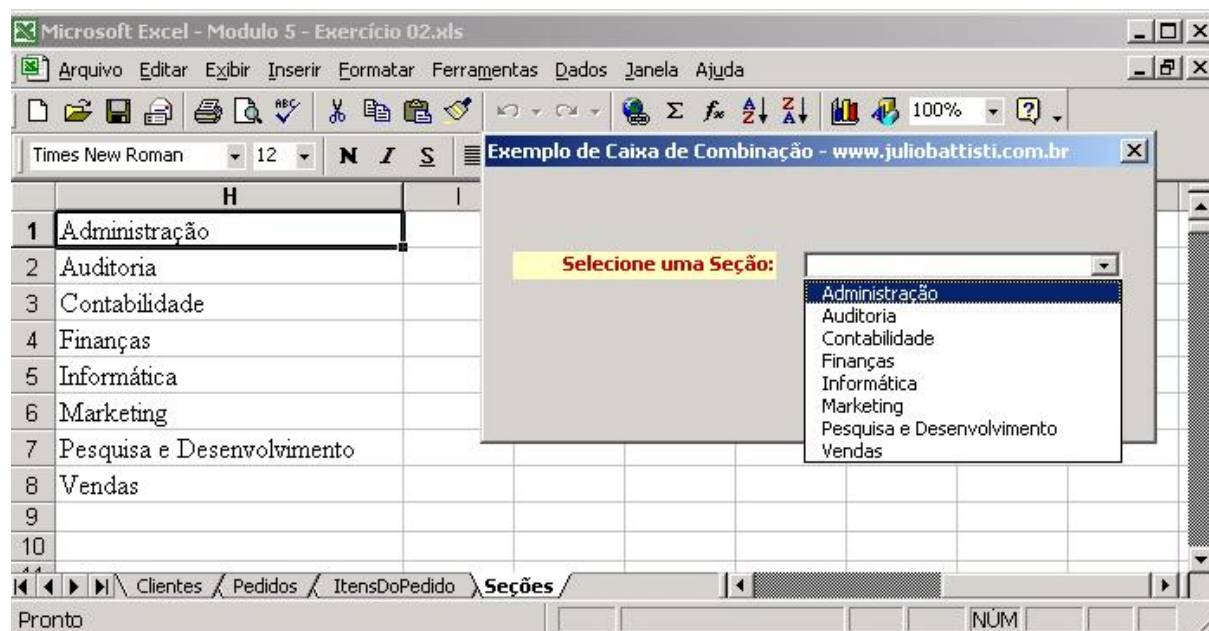
4. Agora vamos definir os itens que serão exibidos no controle do tipo Caixa de combinação.

5. Clique no controle Caixa de combinação para selecioná-lo. Se a janela de propriedades não estiver sendo exibida, pressione a tecla F4 para exibir a janela de propriedades.
6. Altere a propriedade Name para cxListaDeSeções.
7. A lista de valores de um controle do tipo Caixa de Combinação é informado através da propriedade RowSource. Nesta propriedade você deve informar uma faixa de células na planilha, onde estão os valores que devem ser exibidos na Caixa de Combinação. Por exemplo, você pode informar o seguinte valor para a propriedade RowSource:

Seções!H1:H8

neste exemplo, estou informando que serão utilizados os valores da faixa H1 até H8, da planilha Seções, da pasta de trabalho atual. Ou seja, a nossa caixa de combinação terá oito linhas, sendo que o valor da primeira linha é obtido a partir da célula H1 da planilha Seções, o valor da segunda linha é obtido a partir da célula H2 da planilha Seções e assim por diante. Substitua Seções!H1:H8 pelo intervalo onde estão os dados que você deseja exibir no controle Caixa de Combinação.

6. Agora já estamos em condições de testar o nosso controle. Clique em qualquer espaço do formulário para selecioná-lo e pressione a tecla F5 para executá-lo.
7. O formulário será carregado. Abra a lista do controle Caixa de combinação. Observe que são exibidos os valores das células H1 até H8, da planilha Seções, conforme pode ser conferido na figura a seguir:



Na próxima lição iniciaremos o estudo das principais propriedades disponíveis para cada controle.

Lição 11: User Form – Propriedades dos Controles – Parte 1

Nesta lição iremos continuar o estudo do controle Caixa de Combinação. Vamos inicialmente apresentar mais detalhes sobre uma série de propriedades de um controle do tipo Caixa de Combinação.

Principais propriedades do controle Caixa de Combinação:

Importante: Muitas das propriedades que descreverei aqui, se aplicam também para outros tipos de controles. Ao encontrar uma destas propriedades em outros tipos de controles, você poderá voltar a descrição apresentada nestas lições, para revisar a descrição e a utilização da propriedade.

Propriedade AutoSize:

Esta propriedade determina se o controle é automaticamente redimensionado para exibir todo o seu conteúdo. Esta propriedade pode ser configurada na janela de propriedades, definindo o seu valor em True ou False, conforme descreverei mais adiante. Esta propriedade também pode ser alterada via código VBA, usando a sintaxe indicada a seguir:

Sintaxe:

```
objeto.AutoSize = True/False
```

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação.

Esta propriedade pode assumir os valores True ou False, conforme descrito a seguir:

- ▶ **True:** Redimensiona automaticamente o controle para exibir todo o seu conteúdo.
- ▶ **False::** Mantém constante o tamanho do controle. O conteúdo é truncado quando excede a área do controle (padrão).

Para controles com legendas, a propriedade AutoSize especifica se o controle se ajusta automaticamente para exibir a legenda inteira.

Para controles sem legendas, essa propriedade especifica se o controle se ajusta automaticamente para exibir as informações que são armazenadas/digitadas no controle. Em um controle do tipo Caixa de Combinação (ComboBox), por exemplo, definir AutoSize como True define automaticamente a largura da área de exibição para corresponder ao tamanho do texto atual.

Para uma caixa de texto de linha única, definir AutoSize como True define automaticamente a largura da área de exibição como o tamanho do texto na caixa de texto.

Para uma caixa de texto de várias linhas que não contém texto, definir AutoSize como True exibe automaticamente o texto na forma de uma coluna. A largura da coluna de texto é

definida para acomodar a letra mais larga daquele tamanho de fonte. A altura da coluna de texto é definida para exibir o texto inteiro da Caixa de Texto.

Para uma caixa de texto de várias linhas contendo texto, definir `AutoSize` como `True` automaticamente aumenta o tamanho vertical da Caixa de Texto para exibir o texto todo. A largura da Caixa de Texto não é alterada.

Observação: Se você alterar manualmente o tamanho de um controle enquanto `AutoSize` for `True`, a alteração manual substituirá o tamanho definido anteriormente por `AutoSize`.

Propriedade `AutoTab`:

Esta propriedade determina se irá ocorrer a tabulação automática quando um usuário digita o número máximo de caracteres permitido em um `TextBox` ou na parte da caixa de texto de um `ComboBox`. Por exemplo, suponha que você tem um campo do tipo texto, com um tamanho máximo definido em 10 Caracteres. Se `AutoTab` estiver configurada como `True`, quando o usuário digitar o décimo caractere, o cursor irá se deslocar automaticamente para o próximo campo. Este recurso pode facilitar a digitação e melhorar a produtividade de quem utiliza o formulário, porém muitos usuários não gostam de utilizar formulários com este recurso habilitado.

Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.AutoTab = True/False
```

onde objeto é obrigatório e deve ser uma referência ao controle que está sendo configurado. Por exemplo, para definir a propriedade `AutoTab` como `True`, para o controle `vlEmpréstimo`, você usa a sintaxe a seguir:

```
vlEmpréstimo.AutoTab = True
```

Esta propriedade pode assumir os valores `True` ou `False`, conforme descrito a seguir:

- ▶ **True:** A auto-tabulação é habilitada.
- ▶ **False:** Não ocorre tabulação (padrão).

Nota: A propriedade `MaxLength` (que será descrita mais adiante), especifica o número máximo de caracteres permitido em um `TextBox` ou na parte de caixa de texto de um `ComboBox`.

Você pode especificar a propriedade `AutoTab` para um `TextBox` ou `ComboBox` em um formulário para o qual você habitualmente insere um número definido de caracteres. Uma vez que um usuário digita o número máximo de caracteres definido pela propriedade `MaxLengt`,

o foco se moverá automaticamente para o próximo controle na ordem de tabulação, ordem esta definida pela propriedade `TabIndex`, descrita anteriormente.

Por exemplo, se uma `TextBox` exibe números de estoque de inventário que têm sempre cinco caracteres de comprimento, você pode utilizar `MaxLength` para especificar o número máximo de caracteres a ser inserido no `TextBox` e `AutoTab` para tabular automaticamente no próximo controle depois que o usuário digitar cinco caracteres.

Propriedade `BackColor`:

Esta propriedade é utilizada para definir a cor de segundo plano do controle. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.BackColor [= Longo]
```

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Longo:** Opcional. Um valor ou uma constante que determina a cor de fundo de um objeto.

Você pode utilizar qualquer inteiro que represente uma cor válida. Também pode especificar uma cor utilizando a função `RGB` com componentes de cor vermelho, verde e azul. O valor de cada componente de cor é um inteiro que varia de zero a 255. Por exemplo, você pode especificar azul turquesa como o valor inteiro 4966415 ou os componentes vermelho, verde e azul como 15, 200, 75.

Importante: Você verá somente a cor de fundo de um objeto se a propriedade `BackStyle` estiver definida como `fmBackStyleOpaque`. Se a propriedade `BackStyle` estiver definida como `fmBackStyleTransparent`, a cor de fundo será ignorada.

Na janela de propriedades, você pode selecionar a cor de fundo, clicando em uma das cores disponíveis. Para isso, abra a lista de opções desta propriedade e, na janela que é exibida, clique na guia `Paleta`. Agora é só clicar na cor desejada, conforme indicado na figura a seguir:



A cor clicada, será aplicada a propriedade `BackColor`.

Lição 12: User Form – Propriedades dos Controles – Parte 2

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade BackStyle

Esta propriedade é utilizada para retornar ou definir o estilo de segundo plano do controle. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.BackStyle = fmBackStyle
```

A sintaxe da propriedade BackStyle possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmBackStyle:** Opcional. Especifica o padrão de segundo plano do controle. Os valores possíveis para esta propriedade estão descritos a seguir

fmBackStyleTransparent ou 0: O segundo plano é transparente.

fmBackStyleOpaque ou 1: O segundo plano é opaco (padrão).

A propriedade BackStyle determina se o controle é transparente. Se BackStyle for fmBackStyleOpaque, o controle não é transparente e você não poderá ver nada atrás do controle em um formulário. Se BackStyle for fmBackStyleTransparent, você poderá ver através do controle e olhar para qualquer coisa no formulário localizado atrás do controle.

Observação: BackStyle não afeta a transparência de bitmaps. Você deve utilizar um editor de imagens, como o Paintbrush, para tornar um bitmap transparente. Nem todos os controles suportam bitmaps transparentes.

Propriedade BorderColor:

Esta propriedade é utilizada para especifica a cor da borda de um controle. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.BorderColor [= Longo]
```

A sintaxe da propriedade BorderColor possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Longo:** Opcional. Um valor ou constante que determina a cor da borda de um objeto.

Você pode utilizar qualquer inteiro que represente uma cor válida. Você também pode especificar uma cor utilizando a função RGB com componentes de cor vermelho, verde e azul. O valor de cada componente de cor é um inteiro que varia de zero a 255. Por exemplo, você pode especificar o azul turquesa como o valor inteiro 4966415 ou como valores de componentes de cor RGB 15, 200, 75.

Para utilizar a propriedade `BorderColor`, a propriedade `BorderStyle` deve ser definida como um valor diferente de `fmBorderStyleNone`. Mais adiante descreverei a propriedade `BorderStyle`.

`BorderStyle` utiliza `BorderColor` para definir cores da borda. A propriedade `SpecialEffects` utiliza cores do sistema exclusivamente para definir as cores de sua borda. Para sistemas operacionais do Windows, as definições de cores do sistema fazem parte do Painel de controle e são encontradas na guia Aparência da pasta Exibir. No Windows NT 4.0 ou posterior, as definições de cores do sistema são armazenadas na pasta Cor do Painel de controle.

Na janela de propriedades, você pode selecionar a cor de fundo, clicando em uma das cores disponíveis. Para isso, abra a lista de opções desta propriedade e, na janela que é exibida, clique na guia Paleta. Agora é só clicar na cor desejada, conforme indicado na figura a seguir:



A cor clicada, será aplicada a propriedade `BorderColor`.

Propriedade `BorderStyle`:

Esta propriedade especifica o tipo de borda utilizado por um controle ou um formulário. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.BorderStyle [= fmBorderStyle]
```

A sintaxe da propriedade `BorderStyle` possui as partes a seguir:

Autor: Júlio Cesar Fabris Battisti

Site: www.juliobattisti.com.br

Confira também o livro: "Windows Server 2003 – Curso Completo, 1568 páginas"

Página 416 de 527

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmBorderStyle:** Opcional. Especifica o estilo da borda. Pode ser um dos valores descritos a seguir:

fmBorderStyleNone ou 0: O controle não tem linha de borda visível.

FmBorderStyleSingle ou 1: O controle tem uma borda de linha única (padrão).

O valor padrão para os controles ComboBox, Frame, Label, ListBox ou TextBox é 0 (Nenhum). O valor padrão para um controle Image é 1 (Único).

Nota: Para um Frame, a propriedade BorderStyle é ignorada se a propriedade SpecialEffect for Nenhum.

Você pode especificar tanto BorderStyle como SpecialEffect para especificar a borda para um controle, mas não ambas. Se você especificar um valor diferente de zero para uma dessas propriedades, o sistema definirá o valor da outra propriedade como zero. Por exemplo, se você definir BorderStyle como fmBorderStyleSingle, o sistema definirá SpecialEffect como zero (Plano). Se você especificar um valor diferente de zero para SpecialEffect, o sistema definirá BorderStyle como zero.

BorderStyle utiliza BorderColor, descrita anteriormente, para definir as cores de suas bordas.

Lição 13: User Form – Propriedades dos Controles – Parte 3

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade BoundColumn

Esta propriedade identifica a fonte dos dados em um controle do tipo Caixa de combinação (ComboBox) ou Caixa de Listagem (ListBox) de várias colunas. Se o controle tiver duas ou mais colunas, a propriedade BoundColumn indica o valor de qual coluna será associado ao controle, depois que uma das opções da lista tiver sido selecionada.

Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.BoundColumn [= Variante]
```

A sintaxe da propriedade BoundColumn possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Variante:** Opcional. Indica como o valor BoundColumn está selecionado. Variante pode assumir um dos valores indicados a seguir:

0: Atribui o valor da propriedade ListIndex ao controle.

1 ou maior: Atribui o valor da coluna especificada ao controle. As colunas são numeradas a partir de 1 ao utilizar essa propriedade (padrão). Ou seja, a primeira coluna é indicada pelo valor 1, a segunda coluna pelo valor 2 e assim por diante.

Quando o usuário escolhe uma linha em um controle do tipo Caixa de Listagem (ListBox) ou Caixa de Combinação (ComboBox) de várias colunas, a propriedade BoundColumn identifica qual item daquela linha, deve ser armazenado como o valor do controle. Por exemplo, se cada linha contiver 8 itens e BoundColumn for 3, o sistema armazenará as informações da terceira coluna da linha atualmente selecionada como o valor do objeto.

É possível exibir um conjunto de dados para os usuários, mas armazenar diferentes valores associados para o objeto utilizando as propriedades BoundColumn e TextColumn. TextColumn identifica a coluna de dados exibida na parte da caixa de texto de um ComboBox e o valor armazenado na propriedade Text; BoundColumn identifica a coluna dos valores de dados associados armazenados para o controle.

Por exemplo, você poderia configurar uma Caixa de Listagem (ListBox) de várias colunas que contivesse os nomes dos feriados em uma coluna e as datas dos feriados em uma segunda coluna. Para apresentar os nomes de feriados aos usuários, especifique a primeira coluna

como `TextColumn`. Para armazenar as datas dos feriados, especifique a segunda coluna como `BoundColumn`. Para ocultar as datas dos feriados, defina como zero a propriedade `ColumnWidths` da segunda coluna. Outro exemplo, você poderia exibir os nomes dos clientes em uma Caixa de Combinação, em um formulário para entrada de pedidos. O usuário seleciona o nome, mas o valor armazenado no campo é o código do cliente (que é o campo que relaciona as tabelas Clientes e Pedidos). A coluna do código pode ser ocultada, definindo como zero o valor da propriedade `ColumnWidths` para a coluna do código. Com isso você está facilitando a vida de quem vai utilizar o formulário, pois seria praticamente impossível lembrar dos códigos de cada cliente. Lógico que é muito mais fácil selecionar um cliente pelo nome do que pelo código.

Se o controle for ligado a uma fonte de dados, o valor na coluna especificada por `BoundColumn` será armazenado na fonte de dados denominada na propriedade `ControlSource`.

O valor `ListIndex` recupera o número da linha selecionada. Por exemplo, se você quiser saber qual a linha do item selecionado, defina `BoundColumn` como 0 para atribuir o número da linha selecionada como o valor do controle. Certifique-se de recuperar um valor atual ao invés de confiar em um valor anteriormente salvo, se estiver fazendo referência a uma lista cujo conteúdo possa sofrer alterações.

Importante: As propriedades `Column`, `List` e `ListIndex` utilizam numeração com base em zero, isto é, o valor do primeiro item (coluna ou linha) é zero, o valor da segunda coluna é um, e assim por diante. Isto significa que se `BoundColumn` for definida como 3, você poderá acessar o valor armazenado naquela coluna utilizando a expressão `Column(2)`.

Propriedade `ColumnCount`:

Esta propriedade é utilizada para especificar o número de colunas a exibir em uma caixa de listagem ou caixa de combinação. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ColumnCount [= Longo]
```

A sintaxe da propriedade `ColumnCount` possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Longo:** Opcional. Especifica o número de colunas a exibir.

Por exemplo, se você definir a propriedade `ColumnCount` de uma caixa de listagem como 3 em um formulário de funcionários, uma das colunas poderá listar os sobrenomes, outra poderá listar os nomes e a terceira poderá listar os números de identificação dos funcionários.

Dica: Definir `ColumnCount` como 0 não exibirá coluna alguma e defini-la como -1 exibirá todas as colunas disponíveis.

Nota: Para uma fonte de dados desacoplada, há um limite de 10 colunas (0 a 9).

É possível utilizar a propriedade `ColumnWidths` para definir a largura das colunas exibidas no controle.

Propriedade `ColumnHeads`:

Este controle é utilizado para exibir uma única linha de cabeçalhos de coluna para caixas de listagem, caixas de combinação e objetos que aceitam cabeçalhos de coluna. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ColumnHeads [= Booleano]
```

A sintaxe da propriedade `ColumnHeads` possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano:** Opcional. Especifica se os cabeçalhos de coluna são exibidos. Pode assumir os valores indicados a seguir:
 - **True:** Exibe cabeçalhos de coluna.
 - **False:** Não exibe cabeçalhos de coluna (padrão).

Os cabeçalhos nas caixas de combinação aparecem somente quando a lista fica suspensa.

Quando o sistema utiliza a primeira linha de itens de dados como cabeçalhos de coluna, eles não podem ser selecionados.

Lição 14: User Form – Propriedades dos Controles – Parte 4

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade ColumnWidths

Esta propriedade é utilizada para especificar a largura de cada coluna de uma caixa de combinação ou caixa de listagem de várias colunas. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ColumnWidths [= Seqüência]
```

A sintaxe da propriedade ColumnWidths possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Seqüência:** Opcional. Define a largura da coluna em pontos. Uma definição de -1 ou em branco resulta em uma largura calculada automaticamente pelo Excel. Uma largura 0 oculta uma coluna. Para especificar uma unidade de medida diferente, inclua a unidade de medida. Um valor maior que 0 especifica explicitamente a largura da coluna.

Para separar entradas de colunas, utilize pontos-e-vírgulas (;) como separadores de lista. No Windows, utilize o separador de listas especificado na seção Configurações regionais do Painel de controle do Windows para alterar este valor. Por exemplo, para definir o tamanho de três colunas de uma Caixa de Combinação, informe os valores separados por ponto-e-vírgula como no exemplo a seguir: 200;150;300. A largura mínima de coluna calculada é de 72 pontos (1 polegada). Para produzir colunas mais estreitas do que isso, você deve especificar a largura explicitamente.

Qualquer uma ou todas definições da propriedade ColumnWidths podem ficar em branco. Uma definição em branco pode ser criada digitando um separador de listas sem um valor precedente.

Se você especificar -1 na página de propriedades, o valor exibido na página de propriedades ficará em branco.

Para calcular larguras de coluna quando ColumnsWidths estiver em branco ou for -1, a largura do controle será dividida igualmente entre todas as colunas da lista. Se a soma das larguras de coluna especificadas exceder a largura do controle, a lista será alinhada à esquerda dentro do controle e uma ou mais das colunas situadas à extrema direita não serão exibidas. Os usuários podem rolar a lista utilizando a barra de rolagem horizontal para exibir as colunas da extrema direita.

A menos que expressamente especificado, as larguras de coluna são medidas em pontos. Para especificar uma outra unidade de medida, inclua a unidade como parte dos valores. Os exemplos a seguir especificam larguras de coluna em várias unidades de medida e descrevem como as várias definições se ajustariam em uma caixa de listagem de três colunas que possui 4 polegadas de largura.

Definição	Efeito
90;72;90	A primeira coluna tem 90 pontos (1,25 polegada), a segunda coluna tem 72 pontos (1 polegada) e a terceira coluna tem 90 pontos.
6 cm;0;6 cm	A primeira coluna tem 6 centímetros; a segunda coluna está oculta; a terceira coluna tem 6 centímetros. Uma vez que parte da terceira coluna está visível, é exibida uma barra de rolagem horizontal.
1.5 in;0;2.5 in	A primeira coluna tem 1,5 polegadas, a segunda coluna está oculta e a terceira coluna tem 2,5 polegadas.
2 in;;2 in	A primeira coluna tem 2 polegadas, a segunda coluna tem 1 polegada (padrão) e a terceira coluna tem 2 polegadas. Uma vez que somente metade da terceira coluna está visível, é exibida uma barra de rolagem horizontal.
(Em branco)	Todas as três colunas têm a mesma largura (1,33 polegadas).

Propriedade ControlSource:

Esta propriedade identifica a localização de dados utilizada para definir ou armazenar a propriedade Value de um controle. A propriedade ControlSource aceita intervalos de planilha do Microsoft Excel. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ControlSource [= Seqüência]
```

A sintaxe da propriedade ControlSource possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Seqüência:** Opcional. Especifica a célula da planilha vinculada à propriedade Value de um controle.

A propriedade ControlSource identifica uma célula ou campo; ela não contém os dados armazenados na célula ou campo. Se você alterar a propriedade Value do controle, a alteração será automaticamente refletida na célula ou campo vinculado. Da mesma forma, se você alterar o valor da célula ou campo vinculado, a alteração se refletirá automaticamente na propriedade Value do controle.

Você não pode especificar um outro controle para o ControlSource. Fazer isso gera um erro.

O valor padrão para `ControlSource` é uma sequência vazia. Se `ControlSource` contiver um valor diferente de uma sequência vazia, ele identificará uma célula ou campo vinculado. O conteúdo dessa célula ou campo será automaticamente copiado na propriedade `Value` quando o controle for carregado.

Observação: Se a propriedade `Value` for `Null`, nenhum valor aparecerá na localização identificada por `ControlSource`.

Propriedade `ControlTipText`:

Esta propriedade define o texto que aparece quando o usuário mantém, durante um momento, o ponteiro do mouse sobre um controle sem clicar. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ControlTipText [= Seqüência]
```

A sintaxe da propriedade `ControlTipText` possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Seqüência:** Opcional. O texto que aparece quando o usuário mantém o ponteiro do mouse sobre um controle.

A propriedade `ControlTipText` permite que você forneça dicas aos usuários sobre um controle em um formulário que está sendo executado. A propriedade pode ser definida durante a tempo de criação mas aparece somente ao lado do controle durante o tempo de execução.

O valor padrão de `ControlTipText` é uma sequência vazia. Quando o valor de `ControlTipText` é definido como uma sequência vazia, não existe nenhuma dica disponível para esse controle.

Na Figura a seguir você vê um exemplo do texto que é exibido para um controle onde a propriedade `ControlTipText` foi definida como “Selecione uma seção na lista”.



Lição 15: User Form – Propriedades dos Controles – Parte 5

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade DragBehavior

Esta propriedade especifica se o sistema ativa o recurso arrastar-e-soltar para um TextBox ou ComboBox. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.DragBehavior [= fmDragBehavior]
```

A sintaxe da propriedade DragBehavior possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmDragBehavior:** Opcional. Especifica se o recurso arrastar-e-soltar está ativado, conforme indicado na tabela a seguir:

Constante	Valor	Descrição
fmDragBehaviorDisabled	0	Não permite a ação de arrastar-e-soltar (padrão).
fmDragBehaviorEnabled	1	Permite a ação de arrastar-e-soltar.

Se a propriedade DragBehavior estiver ativada, arrastar em uma caixa de texto ou caixa de combinação inicia a operação arrastar-e-soltar no texto selecionado. Se DragBehavior estiver desativada, arrastar em uma caixa de texto ou caixa de combinação seleciona o texto.

Dica: A porção suspensa de um ComboBox não suporta processos de arrastar-e-soltar nem suporta a seleção de itens de lista dentro do texto.

DragBehavior não tem efeito sobre um ComboBox cuja propriedade Style esteja definida como fmStyleDropDownList.

Observação: Você pode combinar os efeitos da propriedade EnterFieldBehavior (descrita mais adiante) e de DragBehavior para criar um grande número de estilos de caixa de texto.

Propriedade DropButtonStyle

Esta propriedade especifica o símbolo exibido no botão suspenso de um controle do tipo Caixa de combinação (ComboBox.), sendo que o símbolo padrão é uma flechinha para baixo. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos

controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.DropButtonStyle [= fmDropButtonStyle]
```

A sintaxe da propriedade DropButtonStyle possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmDropButtonStyle:** Opcional. A aparência de um botão suspenso. A tabela a seguir descreve os valores que podem ser atribuídos à propriedade DropButonStyle:

Constante	Valor	Descrição
fmDropButtonStylePlain	0	Exibe um botão simples, sem símbolo.
fmDropButtonStyleArrow	1	Exibe uma seta para baixo (padrão).
fmDropButtonStyleEllipsis	2	Exibe reticências (...).
fmDropButtonStyleReduce	3	Exibe uma linha horizontal, como um caractere de sublinhado.

Nota: A definição recomendada para apresentar itens em uma lista é fmDropButtonStyleArrow. Se desejar utilizar o botão suspenso de uma outra maneira, como para exibir uma caixa de diálogo, especifique fmDropButtonStyleEllipsis, fmDropButtonStylePlain ou fmDropButtonStyleReduce e intercepte o evento DropButtonClick.

Propriedade Enabled

Esta propriedade especifica se um controle pode receber o foco e responder aos eventos gerados pelo usuário. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Enabled [= Booleano]
```

A sintaxe da propriedade Enabled possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano** Opcional. Se o objeto pode responder a eventos gerados pelo usuário. Esta propriedade pode assumir os valores indicados na tabela a seguir:

Valor	Descrição
True	O controle pode receber o foco e responder aos eventos gerados pelo usuário e é acessível por meio de código (padrão).
False	O usuário não pode interagir com o controle utilizando o mouse, pressionamentos de tecla, teclas aceleradoras ou teclas de ativação. Geralmente, o controle ainda é acessível por meio de código.

Utilize a propriedade `Enabled` para ativar e desativar controles. Um controle desativado aparece esmaecido (com uma tonalidade de cinza mais apagada), enquanto um controle ativado não. Além disso, se um controle exibir um `bitmap`, o `bitmap` ficará esmaecido sempre que o controle estiver com a propriedade `Enabled` com o valor `False`. Se `Enabled` for `False` para um controle do tipo `Image`, o controle não iniciará eventos, mas também não aparecerá esmaecido.

As propriedades `Enabled` e `Locked` (que será descrita mais adiante) trabalham juntas para obter os efeitos a seguir:

- ▶ Se tanto `Enabled` quanto `Locked` forem `True`, o controle poderá receber o foco e aparecerá normalmente (não-esmaecido) no formulário. O usuário pode copiar, mas não editar, dados no controle.
- ▶ Se `Enabled` for `True` e `Locked` for `False`, o controle poderá receber o foco e aparecerá normalmente no formulário. O usuário pode copiar e editar dados no controle.
- ▶ Se `Enabled` for `False` e `Locked` for `True`, o controle não poderá receber o foco e estará esmaecido no formulário. O usuário não pode copiar nem editar dados no controle.
- ▶ Se `Enabled` e `Locked` forem ambos `False`, o controle não poderá receber o foco e estará esmaecido no formulário. O usuário não pode copiar nem editar dados no controle.

Nota: Você pode combinar as definições das propriedades `Enabled` e `TabStop` (descrita mais adiante) para impedir que o usuário selecione um botão de comando com o `TAB` e, ao mesmo tempo, permitir que ele clique no botão. Definir `TabStop` como `False` significa que o botão de comando não aparecerá na ordem de tabulação. Contudo, se `Enabled` for `True`, o usuário poderá ainda clicar no botão de comando, desde que a propriedade `TakeFocusOnClick` esteja definido como `True`.

Quando o usuário tabula para um `MultiPage` ou `TabStrip` ativado, a primeira página ou guia no controle receberá o foco. Se a primeira página ou guia de um `MultiPage` ou `TabStrip` estiver desativada, a primeira página ou guia ativada daquele controle receberá o foco. Se todas as páginas ou guias de um `MultiPage` ou `TabStrip` estiverem desativadas, o controle estará desativado e não poderá receber o foco.

Se um `Frame` estiver desativado, todos os controles nele contidos estarão desativados.

Clicar em um ListBox desativada não inicia o evento Click.

Propriedade EnterFieldBehavior

Esta propriedade Especifica o comportamento da seleção durante a digitação em um controle do tipo Caixa de Texto (TextBox) ou um controle do tipo Caixa de Combinação (ComboBox). Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.EnterFieldBehavior [= fmEnterFieldBehavior]
```

A sintaxe da propriedade EnterFieldBehavior possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmEnterFieldBehavior** Opcional. O comportamento desejado para a seleção. Pode ser um dos valores indicados na tabela a seguir:

Constante	Valor	Descrição
fmEnterFieldBehaviorSelectAll	0	Seleciona todo o conteúdo da região de edição ao inserir o controle (padrão).
fmEnterFieldBehaviorRecallSelection	1	Deixa a seleção inalterada. Visualmente, utiliza a seleção que estava em vigor na última vez que o controle estava ativo.

A propriedade EnterFieldBehavior controla a forma como o texto é selecionado quando o usuário tabula até o controle, e não quando o controle recebe foco como resultado do método SetFocus. Seguindo SetFocus, o conteúdo do controle não é selecionado e o ponto de inserção aparece após o último caractere na região de edição do controle.

Lição 16: User Form – Propriedades dos Controles – Parte 6

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade Font:

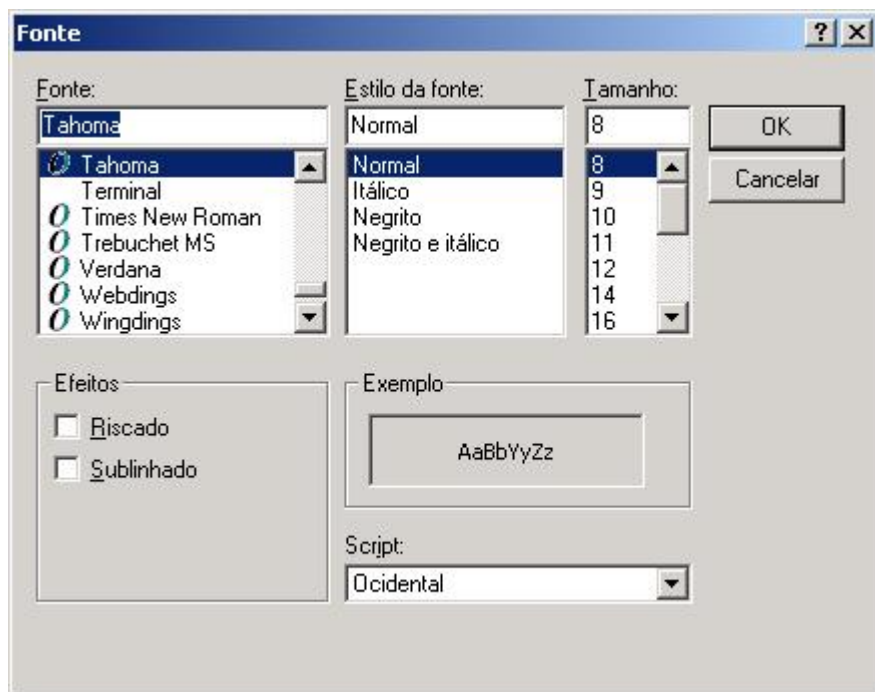
Esta propriedade retorna um objeto do tipo Fonte, o qual define as características do texto utilizado por um controle ou formulário.

Cada controle ou formulário tem seu próprio objeto Font para permitir que você defina as características de seu texto independentemente das características definidas para outros controles e formulários. Utilize as propriedades de fonte para especificar o nome da fonte, definir texto sublinhado ou em negrito ou para ajustar o tamanho do texto.

Observação: As propriedades de fonte de seu formulário ou recipiente determinam os atributos padrão de fonte dos controles que você insere no formulário. Por exemplo, se você quer que todos os novos controles a serem adicionados no formulário, assumam uma fonte de tamanho 12, tipo Verdana e sublinhado, defina estas configurações na propriedade Font do formulário.

A propriedade padrão do objeto Font é a propriedade Name. Se a propriedade Name contiver uma sequência nula, o objeto Font utilizará a fonte padrão do sistema.

Ao clicar na propriedade Font, será habilitado um botão com um sinal de reticências. Ao clicar neste botão, será aberta a janela Font, para que você possa definir as características da fonte, para o controle que está sendo configurado, conforme indicado na Figura a seguir:



Basta selecionar as características desejadas para a fonte e clicar em OK. Pronto, as configurações selecionadas serão aplicadas ao controle que está sendo configurado.

Propriedade ForeColor

Esta propriedade especifica a cor de primeiro plano de um objeto. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ForeColor [= Longo]
```

A sintaxe da propriedade ForeColor possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Longo:** Opcional. Um valor ou constante que determina a cor de primeiro plano de um objeto.

Você pode utilizar qualquer número inteiro que represente uma cor válida. Você também pode especificar uma cor utilizando a função RGB com componentes de cor vermelho, verde e azul. O valor de cada componente de cor é um número inteiro que varia de zero a 255. Por exemplo, você pode especificar azul turquesa com o valor inteiro 4966415 ou com componentes de cor vermelho, verde e azul 15, 200, 75.

Utilize a propriedade ForeColor para controles em formulários para torná-los fáceis de ler ou para transmitir um significado especial. Por exemplo, se uma caixa de texto registra o número de unidades no estoque, você poderá alterar a cor do texto quando o valor cair abaixo do nível de novos pedidos, para chamar a atenção para este controle

Para um controle ScrollBar ou SpinButton (controles estes que serão estudados no módulo 6), ForeColor define a cor das setas. Para um Frame, ForeColor altera a cor de uma legenda. Para um objeto Font, ForeColor determina a cor do texto.

Propriedades Height e Width

Estas propriedades definem a altura (Height) e a largura (Width), em pontos, de um objeto. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Height [= Único]
```

```
objeto.Width [= Único]
```

As sintaxes das propriedades Height e Width possuem as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Único:** Opcional. Uma expressão numérica especificando as dimensões de um objeto.

As propriedades Height e Width são automaticamente atualizadas quando você move ou dimensiona um controle. Se você alterar o tamanho de um controle, a propriedade Height ou Width armazena a nova altura ou largura e a propriedade OldHeight ou OldWidth armazena a altura ou largura anterior. Se você especificar uma definição para a propriedade Left ou Top que seja menor que zero, esse valor será utilizado para calcular a altura ou largura do controle, mas uma parte do controle não ficará visível no formulário.

Se você mover um controle de uma parte do formulário para outra, a definição de Height ou Width só se alterará se você dimensionar o controle à medida que o mover. As definições das propriedades Left e Top do controle serão alteradas para refletir a nova posição do controle em relação às bordas do formulário que o contém.

O valor atribuído a Height ou Width deve ser maior ou igual a zero. Para a maioria dos sistemas, é recomendado um intervalo de valores que vai de 0 a +32.767. Valores superiores também podem funcionar dependendo da configuração do seu sistema.

Propriedade HelpContextID

A propriedade HelpContextID associa um tópico específico em um arquivo de Ajuda do Microsoft Windows a um controle específico. Ou seja, você pode definir um texto de ajuda específico, dentro de um arquivo de ajuda, texto este que será associado ao controle que está sendo configurado.

Sintaxe:

```
objeto.HelpContextID [= Longo]
```

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Longo:** Opcional. Um número inteiro positivo especifica a identificação de contexto de um tópico no arquivo de Ajuda associado ao objeto. Zero indica que nenhum tópico da Ajuda está associado ao objeto (padrão). Deve ser uma identificação de contexto válida no arquivo de Ajuda especificado.

O tópico identificado pela propriedade HelpContextID está disponível aos usuários quando um formulário está em execução. Para exibir o tópico, o usuário deve selecionar o controle ou definir o foco para o controle e, então, pressionar F1.

A propriedade HelpContextID refere-se a um tópico no arquivo de Ajuda personalizado que você criou para descrever o formulário ou o aplicativo. No Visual Basic, o arquivo de Ajuda personalizado é uma propriedade do projeto.

Lição 17: User Form – Propriedades dos Controles – Parte 7

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade HideSelection

Esta propriedade especifica se o texto selecionado permanece realçado quando um controle não tem o foco. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.HideSelection [= Booleano]
```

A sintaxe da propriedade HideSelection possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano:** Opcional. Se o texto selecionado permanece realçado mesmo quando o controle não tem o foco. Pode assumir um dos valores indicados na tabela a seguir:

Valor	Descrição
True	O texto selecionado não é realçado, a menos que o controle tenha o foco (padrão).
False	O texto selecionado sempre aparece realçado.

Você pode utilizar a propriedade HideSelection para manter o texto realçado quando um outro formulário ou caixa de diálogo receber o foco como, por exemplo, em um procedimento de verificação ortográfica.

Propriedades Left e Top

Estas duas propriedades definem a distância entre um controle e a borda esquerda ou superior do formulário onde está o controle. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Left [= Único]
```

```
objeto.Top [= Único]
```

As sintaxes das propriedades Left e Top possuem as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Único:** Opcional. Uma expressão numérica especificando as coordenadas de um objeto.

Definir a propriedade Left ou Top como 0 coloca a borda do controle na borda esquerda ou superior de seu recipiente.

Para a maioria dos sistemas, o intervalo de valores recomendado para Left e Top vai de -32.767 a +32.767. Outros valores também podem funcionar dependendo da configuração do seu sistema. Para um ComboBox, valores de Left e Top aplicam-se à parte de texto do controle, não à parte de lista. Quando você move ou dimensiona um controle, sua nova definição Left é inserida automaticamente na folha de propriedades. Quando você imprime um formulário, a localização horizontal ou vertical do controle é determinada pela sua definição Left ou Top.

Propriedade ListRows

Esta propriedade é utilizada para especificar o número máximo de linhas a exibir na lista de um controle do tipo Caixa de Combinação ou Caixa de Listagem. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ListRows [= Longo]
```

A sintaxe da propriedade ListRows possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Longo:** Opcional. Um número inteiro indicando o número máximo de linhas. O valor padrão é 8.

Se o número de itens na lista exceder o valor da propriedade ListRows, aparecerá uma barra de rolagem vertical na borda direita da parte caixa de listagem da caixa de combinação.

Propriedade ListStyle:

Esta propriedade é utilizada para especificar o aspecto visual da lista em um ListBox ou ComboBox. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ListStyle [= fmListStyle]
```

A sintaxe da propriedade ListStyle possui as partes a seguir:

Autor: Júlio Cesar Fabris Battisti

Site: www.juliobattisti.com.br

Confira também o livro: "Windows Server 2003 – Curso Completo, 1568 páginas"

Página 432 de 527

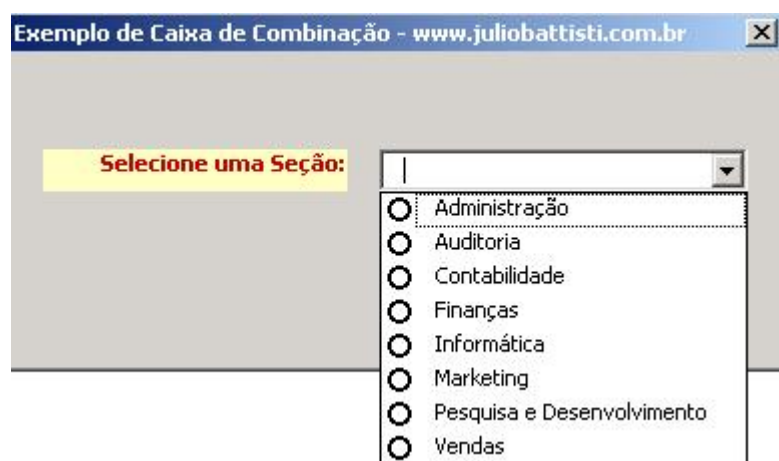
- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmListStyle** Opcional. O estilo visual da lista. Pode assumir os valores indicados na tabela a seguir:

Constante	Valor	Descrição
fmListStylePlain	0	Parece uma caixa de listagem normal, com o segundo plano dos itens realçado.
fmListStyleOption	1	Mostra botões de opção ou caixas de seleção para uma lista de várias seleções (padrão). Quando o usuário seleciona um item do grupo, o botão de opção associado a esse item é selecionado e os botões de opção para os outros itens no grupo têm a seleção anulada.

A propriedade `ListStyle` permite que você mude a apresentação visual de um controle do tipo Caixa de Listagem (`ListBox`) ou Caixa de Combinação (`ComboBox`). Por meio da especificação de uma definição diferente de `fmListStylePlain`, você pode apresentar o conteúdo de ambos os controles como um grupo de itens individuais, com cada item incluindo um indício visual para indicar se está selecionado.

Se o controle suportar uma única seleção (a propriedade `MultiSelect` está definida como `fmMultiSelectSingle`), o usuário pode pressionar um único botão no grupo. Se o controle suportar várias seleções, o usuário pode pressionar dois ou mais botões no grupo.

Na figura a seguir, temos um exemplo da aparência de um controle do tipo Caixa de Combinação, onde a propriedade `ListStyle` foi configurada com o valor `fmListStyleOption`.



Lição 18: User Form – Propriedades dos Controles – Parte 8

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade ListWidth

Esta propriedade é utilizada para especificar a largura da lista em um controle do tipo Caixa de Combinação (ComboBox). Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ListWidth [= Variante]
```

A sintaxe da propriedade ListWidth possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação.
- ▶ **Variante:** Opcional. A largura da lista. Um valor zero faz com que a lista tenha a mesma largura do ComboBox. O valor padrão é tornar a lista tão larga quanto a parte texto do controle.

Se desejar exibir uma lista de várias colunas, digite um valor que tornará a caixa de listagem larga o suficiente para se ajustar a todas as colunas.

Dica: Ao projetar caixas de combinação, certifique-se de deixar espaço suficiente para exibir seus dados e para uma barra de rolagem vertical.

Propriedade Locked

Esta propriedade é utilizada para especificar se um controle pode ter o seu conteúdo editado. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Locked [= Booleano]
```

A sintaxe da propriedade Locked possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano:** Opcional. Se o controle pode ser editado. Pode assumir um dos valores indicados na tabela a seguir:

Valor	Descrição
True	Você não pode editar o valor.
False	Você pode editar o valor (padrão).

Quando um controle está bloqueado e ativado, ele ainda pode iniciar eventos e ainda pode receber o foco, apenas não poderá ter o seu conteúdo alterado.

Propriedade MatchEntry

Esta propriedade é utilizada para retornar ou para definir um valor indicando como um controle do tipo Caixa de Listagem (ListBox) ou Caixa de Combinação (ComboBox) pesquisa sua lista à medida que o usuário digita. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.MatchEntry [= fmMatchEntry]
```

A sintaxe da propriedade MatchEntry possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmMatchEntry:** Opcional. A regra utilizada para comparar entradas na lista. Esta propriedade pode assumir um dos valores indicados na tabela a seguir:

Constante	Valor	Descrição
fmMatchEntryFirstLetter	0	Correspondência básica. O controle pesquisa a próxima entrada que inicia com o caractere digitado. Digitar repetidamente a mesma letra <u>circula</u> por todas as entradas que começam com essa letra.
fmMatchEntryComplete	1	Correspondência estendida. À medida que cada caractere é digitado, o controle pesquisa uma entrada correspondente a todos os caracteres digitados (padrão).
fmMatchEntryNone	2	Sem correspondência.

A propriedade MatchEntry pesquisa entradas da propriedade TextColumn de um ListBox ou ComboBox.

O controle examina a coluna identificada por TextColumn para localizar uma entrada que corresponda à entrada digitada pelo usuário. Ao localizar uma correspondência, a linha que contém a correspondência é selecionada, o conteúdo da coluna é exibido e o conteúdo de sua propriedade BoundColumn torna-se o valor do controle. Se a correspondência for inequívoca, a localização da correspondência inicia o evento Click.

O controle inicia o evento Click assim que o usuário digita uma sequência de caracteres que tenha uma correspondência exata com uma entrada na lista. À medida que o usuário digita, a entrada é comparada com a linha atual na lista e com a próxima linha na lista. Quando a entrada corresponde somente à linha atual, a correspondência é inequívoca.

Importante: No Microsoft Forms, isto se aplica independentemente de a lista ser ou não ordenada. Isto significa que o controle localiza a primeira ocorrência que corresponda à entrada, com base na ordem de itens na lista. Por exemplo, digitar "abc" ou "bc" iniciará o evento Click para a lista a seguir:

abcde
bcdef
abcxyz
bchij

Observe que, em ambos os casos, a entrada correspondente não é exclusiva; contudo, ela é suficientemente diferente da entrada adjacente para que o controle interprete a correspondência como inequívoca e inicie o evento Click.

Propriedade MatchRequired

Especifica se um valor digitado na parte texto de um ComboBox deve corresponder a uma entrada na parte lista existente do controle. O usuário pode digitar valores não-correspondentes, mas não pode deixar o controle enquanto não for digitado um valor correspondente.

Sintaxe:

```
objeto.MatchRequired [= Booleano]
```

A sintaxe da propriedade MatchRequired possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano:** Opcional. Se o texto digitado deve corresponder a um item existente na lista. Pode ser um dos valores indicados na tabela a seguir:

Valor	Descrição
True	O texto digitado deve corresponder a uma entrada da lista existente.
False	O texto digitado pode ser diferente de todas as entradas da lista existentes (padrão).

Se a propriedade MatchRequired for True, o usuário não poderá sair do ComboBox enquanto o texto digitado não corresponder a uma entrada existente na lista. MatchRequired mantém a integridade da lista exigindo que o usuário selecione uma entrada existente.

Lição 19: User Form – Propriedades dos Controles – Parte 9

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade MaxLength

Especifica o número máximo de caracteres que um usuário pode digitar em um TextBox ou ComboBox. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.MaxLength [= Longo]
```

A sintaxe da propriedade MaxLength possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Longo:** Opcional. Um número inteiro indicando o número permitido de caracteres.

Definir a propriedade MaxLength como 0 indica que não há nenhum outro limite além do criado por restrições de memória.

Propriedade MouseIcon

Esta propriedade é utilizada para atribuir um ícone personalizado a um objeto. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.MouseIcon = LoadPicture(nomedocaminho)
```

A sintaxe da propriedade MouseIcon possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **nomedocaminho:** Obrigatória. Uma expressão de sequência que especifica o caminho e o nome de arquivo do arquivo que contém o ícone personalizado.

A propriedade MouseIcon é válida quando a propriedade MousePointer é definida como 99. O ícone de mouse de um objeto é a imagem que aparece quando o usuário move o mouse através desse objeto.

Para atribuir uma imagem ao ponteiro do mouse, você tanto pode atribuir uma figura à propriedade MouseIcon quanto carregar uma figura de arquivo utilizando a função LoadPicture.

Propriedade MousePointer

Esta propriedade é utilizada para especificar o tipo de ponteiro exibido quando o usuário posiciona o mouse sobre um determinado objeto. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.MousePointer [= fmMousePointer]
```

A sintaxe da propriedade MousePointer possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmMousePointer** Opcional. A forma que você deseja para o ponteiro do mouse. Esta constante pode assumir os valores indicados na tabela a seguir:

Constante	Valor	Descrição
fmMousePointerDefault	0	Ponteiro padrão. A imagem é determinada pelo objeto (padrão).
fmMousePointerArrow	1	Seta.
fmMousePointerCross	2	Ponteiro em forma de cruz.
fmMousePointerIBeam	3	Forma de I.
fmMousePointerSizeNESW	6	Seta dupla apontando para o nordeste e sudoeste.
fmMousePointerSizeNS	7	Seta dupla apontando para o norte e sul.
fmMousePointerSizeNWSE	8	Seta dupla apontando para o noroeste e sudeste.
fmMousePointerSizeWE	9	Seta dupla apontando para o oeste e leste.
fmMousePointerUpArrow	10	Seta acima.
fmMousePointerHourglass	11	Ampulheta.
fmMousePointerNoDrop	12	Símbolo de "Não" (círculo com uma linha diagonal) no topo do objeto que está sendo arrastado. Indica um destino de soltura inválido.
fmMousePointerAppStarting	13	Seta com uma ampulheta.
fmMousePointerHelp	14	Seta com um ponto de interrogação.
fmMousePointerSizeAll	15	Dimensionar todos os cursores (setas apontando para

		o norte, sul, leste e oeste).
fmMousePointerCustom	99	Utiliza o ícone especificado pela propriedade MouseIcon.

Utilize a propriedade MousePointer quando desejar indicar mudanças na funcionalidade à medida que o ponteiro do mouse passar sobre controles de um formulário. Por exemplo, a definição de ampulheta (11) é útil para indicar que o usuário deve aguardar o término de um processo ou operação.

Alguns ícones variam dependendo de definições do sistema, como os ícones associados a temas da área de trabalho.

Propriedade Row Source

Esta propriedade é utilizada para especificar a origem que fornece uma lista para um controle do tipo Caixa de Combinação (ComboBox) ou Caixa de Listagem (ListBox.). Já utilizamos esta propriedade em um exemplo deste módulo, no qual criamos uma Caixa de Combinação que exibía uma lista de seções. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.RowSource [= Seqüência]
```

A sintaxe da propriedade RowSource possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Seqüência:** Opcional. A origem da lista para o ComboBox ou ListBox.

Lição 20: User Form – Propriedades dos Controles – Parte 10

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade SelectionMargin

Esta propriedade é utilizada para especificar se o usuário pode selecionar uma linha de texto clicando na região à esquerda do texto. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.SelectionMargin [= Booleano]
```

A sintaxe da propriedade SelectionMargin possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano:** Opcional. Se clicar na margem seleciona uma linha de texto. Esta propriedade pode assumir os valores indicados na Tabela a seguir:

Valor	Descrição
True	Clicar na margem provoca a seleção de texto (padrão).
False	Clicar na margem não provoca a seleção de texto.

Quando a propriedade SelectionMargin é True, a margem da seleção ocupa uma estreita faixa ao longo da borda esquerda da região de edição de um controle. Quando definida como False, toda a região de edição pode armazenar texto.

Se a propriedade SelectionMargin estiver definida como True quando um controle for impresso, a margem da seleção também será impressa.

Propriedade ShowDropButtonWhen:

Esta propriedade é utilizada para especificar quando mostrar o botão suspenso (setinha para baixo) de um controle do tipo Caixa de Combinação (ComboBox) ou Caixa de Listagem (TextBox). Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.ShowDropButtonWhen [= fmShowDropButtonWhen]
```

A sintaxe da propriedade ShowDropButtonWhen possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Listagem
- ▶ **fmShowDropButtonWhen** Opcional. As circunstâncias nas quais o botão suspenso ficará visível.. Esta propriedade pode assumir os valores indicados na tabela a seguir:

Constante	Valor	Descrição
fmShowDropButtonWhenNever	0	Não mostra o botão suspenso sob nenhuma circunstância.
fmShowDropButtonWhenFocus	1	Mostra o botão suspenso quando o controle tem o foco.
fmShowDropButtonWhenAlways	2	Sempre mostra o botão suspenso.

Para um controle do tipo Caixa de Combinação (ComboBox), o valor padrão é fmShowDropButtonWhenAlways; para um controle do tipo Caixa de Texto (TextBox), o valor padrão é fmShowDropButtonWhenNever.

Propriedade SpecialEffect

Esta propriedade é utilizada para especificar o aspecto visual de um objeto. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

Para Caixa de seleção, Botão de opção, Botão de ativação, use a sintaxe a seguir:

```
objeto.SpecialEffect [= fmButtonEffect]
```

Para outros controles, use a sintaxe a seguir:

```
objeto.SpecialEffect [= fmSpecialEffect]
```

A sintaxe da propriedade SpecialEffect possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmButtonEffect:** Opcional. O aspecto visual desejado para um CheckBox, OptionButton ou ToggleButton (estes controles serão estudados no próximo módulo).
- ▶ **fmSpecialEffect:** Opcional. O aspecto visual desejado de um objeto que não um CheckBox, OptionButton, ou ToggleButton. Esta constante pode assumir um dos valores indicados na tabela a seguir:

Constante	Valor	Descrição
fmSpecialEffectFlat	0	O objeto parece plano, sendo diferenciado do formulário circundante por uma borda, uma alteração de cor ou as duas coisas. O padrão para Image e Label, válido para todos os controles.
fmSpecialEffectRaised	1	O objeto tem um realce no topo e à esquerda e uma sombra na base e à direita. Não é válido para caixas de seleção nem para botões de opção.
fmSpecialEffectSunken	2	O objeto tem uma sombra no topo e à esquerda e um realce na base e à direita. O controle e sua borda parecem esculpidos na forma que os contém. O padrão para CheckBox e OptionButton, válido para todos os controles (padrão).
fmSpecialEffectEtched	3	A borda parece esculpida em torno da borda do controle. Não é válido para caixas de seleção nem para botões de opção.
fmSpecialEffectBump	6	O objeto tem uma aresta na base e à direita e parece plano no topo e à esquerda. Não é válido para caixas de seleção nem para botões de opção.

Para um Frame, o valor padrão é Baixo Relevo.

Observe que somente Plano e Baixo Relevo (0 e 2) são valores aceitáveis para CheckBox, OptionButton e ToggleButton. Todos os valores listados são aceitáveis para os outros controles.

Você pode utilizar tanto a propriedade SpecialEffect quanto a BorderStyle para especificar a definição de bordas para um controle, mas não ambas. Se você especificar um valor diferente de zero para uma destas propriedades, o sistema definirá o valor da outra propriedade como zero. Por exemplo, se você definir SpecialEffect como fmSpecialEffectRaised, o sistema definirá BorderStyle como zero (fmBorderStyleNone).

Para um Frame, BorderStyle é ignorado se SpecialEffect for fmSpecialEffectFlat.

SpecialEffect utiliza as cores do sistema para definir suas bordas.

Observação: Ainda que a propriedade SpecialEffect exista em um controle ToggleButton, ela está desativada. Você não pode definir nem retornar um valor para esta propriedade no ToggleButton.

Lição 21: User Form – Propriedades dos Controles – Parte 11

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade Style:

Para um controle do tipo Caixa de Combinação (ComboBox), esta propriedade é utilizada para especificar como o usuário pode escolher ou definir o valor do controle. Para um controle do tipo MultiPage e TabStrip, esta propriedade identifica o estilo das tabulações no controle. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

Para Caixa de combinação:

```
objeto.Style [= fmStyle]
```

Para MultiPágina e Faixa de tabulação

```
objeto.Style [= fmTabStyle]
```

A sintaxe da propriedade Style possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmStyle:** Opcional. Especifica como um usuário define o valor de um controle do tipo Caixa de Combinação (ComboBox). Esta constante pode assumir um dos valores indicados na tabela a seguir:

Constante	Valor	Descrição
fmStyleDropDownCombo	0	O ComboBox se comporta como uma caixa de combinação suspensa. O usuário pode digitar um valor na região de edição ou selecionar um valor da lista suspensa (padrão).
fmStyleDropDownList	2	A ComboBox se comporta como uma caixa de listagem. O usuário deve escolher um valor da lista.

- ▶ **fmTabStyle:** Opcional. Especifica o estilo da tabulação em uma MultiPage ou TabStrip. Esta constante pode assumir um dos valores indicados na tabela a seguir:

Constante	Valor	Descrição
fmTabStyleTabs	0	Exibe tabulações na barra de tabulação (padrão).
fmTabStyleButtons	1	Exibe botões na barra de tabulação.
fmTabStyleNone	2	Não exibe a barra de tabulação.

Propriedade TabIndex:

Esta propriedade é utilizada para especificar a posição de um único controle na ordem de tabulação do formulário. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.TabIndex [= Inteiro]
```

A sintaxe da propriedade TabIndex possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Inteiro** Opcional. Um número inteiro de 0 até um número uma unidade menor que o número de controles no formulário que tiverem uma propriedade TabIndex. Atribuir um valor a TabIndex menor que 0 gera um erro. Se você atribuir um valor TabIndex maior que o maior valor de índice, o sistema redefinirá o valor com o máximo valor permitido.

O valor de índice do primeiro objeto na ordem de tabulação é zero. O primeiro controle terá um valor zero para a propriedade TabIndex, o segundo terá o valor 1, o terceiro o valor 2 e assim por diante. É muito comum você excluir um controle e depois adicionar novamente, mudando o tipo ou outras características. Nestes casos, pode acontecer de a ordem de tabulação ficar incorreta. Por exemplo, o foco vem do primeiro controle, passa para o segundo, pula para o quarto e volta para o terceiro. Isso acontece porque você exclui o terceiro controle e o adicionou novamente. Ao adicioná-lo novamente ele, em termos de índice de tabulação, passou a ser o último controle do formulário. Por isso que ao usar Tab, ele será o último controle a receber o foco. Para resolver esta questão, basta configurar a propriedade TabIndex de cada controle, de acordo com a ordem correta em que eles devem receber o foco.

Propriedade TabStop:

Esta propriedade define se um controle pode receber o foco quando o usuário tabula até ele. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.TabStop [= Booleano]
```

A sintaxe da propriedade TabStop possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano:** Opcional. Se o objeto é uma parada de tabulação. Esta propriedade pode assumir os valores indicados na tabela a seguir:

Valor	Descrição
True	Designa o objeto como uma parada de tabulação (padrão).
False	Desvia do objeto quando o usuário está tabulando, embora o objeto ainda retenha seu lugar na ordem real de tabulação, conforme determinado pela propriedade TabIndex.

Propriedade Tag:

Esta propriedade armazena informações adicionais sobre um objeto. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Tag [= Seqüência]
```

A sintaxe da propriedade Tag possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Seqüência:** Opcional. Uma expressão de seqüência identificando o objeto. O padrão é uma seqüência de comprimento zero ("").

Utilize a propriedade Tag para atribuir uma seqüência de identificação a um objeto sem afetar as definições ou atributos de outras propriedades.

Por exemplo, você pode utilizar Tag para verificar a identidade de um formulário ou controle que é passado como uma variável para um procedimento.

Lição 22: User Form – Propriedades dos Controles – Parte 12

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade TakeFocusOnClick:

Esta propriedade especifica se um controle toma o foco quando clicado. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.TakeFocusOnClick [= Booleano]
```

A sintaxe da propriedade TakeFocusOnClick possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano:** Opcional. Especifica se um controle toma o foco quando clicado. Esta propriedade pode assumir um dos valores indicados na tabela a seguir:

Valor	Descrição
True	O botão toma o foco quando clicado (padrão).
False	O botão não toma o foco quando clicado.

A propriedade TakeFocusOnClick define somente o que acontece quando o usuário clica em um controle. Se o usuário tabula para o controle, usando a tecla TAB, o controle toma o foco, independentemente do valor de TakeFocusOnClick.

Utilize esta propriedade para completar ações que afetam um controle sem exigir que esse controle desista do foco. Por exemplo, suponha que o formulário inclui um TextBox e um CommandButton que verifica a ortografia do texto. Seria conveniente selecionar o texto no TextBox, clicar no CommandButton e executar o corretor ortográfico sem retirar o foco do TextBox. Você pode fazê-lo definindo a propriedade TakeFocusOnClick do CommandButton como False.

Propriedade Text:

Esta propriedade é utilizada para retornar ou para definir o texto em um controle do tipo Caixa de Texto (TextBox). Também é utilizada para mudar a linha selecionada em um controle do tipo Caixa de Combinação (ComboBox) ou Caixa de Listagem (ListBox). Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Text [= Seqüência]
```

A sintaxe da propriedade Text possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Seqüência:** Opcional. Uma expressão de seqüência especificando um texto. O valor padrão é uma seqüência de comprimento zero ("").

Para um TextBox, qualquer valor que você atribua à propriedade Text é também atribuído à propriedade Value.

Para uma Caixa de Combinação (ComboBox), você pode utilizar Text para atualizar o valor do controle. Se o valor de Text corresponder a uma entrada de lista existente, o valor da propriedade ListIndex (o índice da linha atual) é definido como a linha que corresponde a Text. Se o valor de Text não corresponder a uma linha, ListIndex será definido como -1.

Para uma Caixa de Listagem (ListBox), o valor de Text deve corresponder a uma entrada de lista existente. Especificar um valor que não corresponda a uma entrada de lista existente gera um erro.

Você não pode utilizar Text para alterar o valor de uma entrada em uma Caixa de Combinação (ComboBox) ou em uma Caixa de Listagem (ListBox); utilize a propriedade Column ou List para essa finalidade.

A propriedade ForeColor determina a cor do texto.

Propriedade TextAlign:

Esta propriedade especifica como o texto é alinhado em um controle. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.TextAlign [= fmTextAlign]
```

A sintaxe da propriedade TextAlign possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **fmTextAlign:** Opcional. Como um texto é alinhado no controle. Esta propriedade pode assumir um dos valores indicados na tabela a seguir:

Constante	Valor	Descrição
fmTextAlignLeft	1	Alinha o primeiro caractere do texto exibido com a borda esquerda da área de exibição ou edição do controle (padrão).
fmTextAlignCenter	2	Centraliza o texto na área de exibição ou edição do controle.
fmTextAlignRight	3	Alinha o último caractere do texto exibido com a borda direita da área de exibição ou edição do controle.

Para um controle Caixa de Combinação (ComboBox), a propriedade TextAlign afeta somente a região de edição; esta propriedade não tem efeito sobre o alinhamento do texto na lista. Para rótulos autônomos, TextAlign determina o alinhamento da legenda do rótulo.

Propriedade TextLength:

Esta propriedade retorna o tamanho, em caracteres, do texto na região de edição de um controle Caixa de Texto (TextBox) ou Caixa de Combinação (ComboBox). Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

`objeto.TextLength`

A sintaxe da propriedade TextLength possui a parte a seguir:

- **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.

A propriedade TextLength é somente leitura. Para um TextBox de várias linhas, TextLength inclui caracteres de AL (alimentação de linha) e RC (retorno do carro).

Lição 23: User Form – Propriedades dos Controles – Parte 13

Nesta lição continuaremos o estudo das principais propriedades que podem ser configuradas para os controles de um UserForm.

Propriedade TextColumn:

Esta propriedade identifica a coluna em um controle Caixa de Combinação (ComboBox) ou Caixa de Listagem (ListBox), para armazenar na propriedade Text quando o usuário selecionar uma linha. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.TextColumn [= Variante]
```

A sintaxe da propriedade TextColumn possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Variante:** Opcional. A coluna a ser exibida.

Os valores para a propriedade TextColumn variam de -1 até o número de colunas na lista. O valor de TextColumn para a primeira coluna é 1, o valor da segunda coluna é 2, e assim por diante. Definir TextColumn como 0 exibe os valores definidos na propriedade ListIndex. Definir TextColumn como -1 exibe a primeira coluna que tem um valor de ColumnWidths maior que 0.

Na caixa de combinação, o sistema exibe a coluna designada pela propriedade TextColumn na parte da caixa de texto do controle.

Quando o usuário seleciona uma linha de uma Caixa de Combinação (ComboBox) ou de uma Caixa de Listagem (ListBox), a coluna referida por TextColumn é armazenada na propriedade Text. Por exemplo, você poderia configurar uma Caixa de Listagem (ListBox) de várias colunas que contenha os nomes de feriados em uma coluna e as datas dos feriados em uma segunda coluna. Para apresentar os nomes de feriados aos usuários, especifique a primeira coluna como TextColumn. Para armazenar as datas dos feriados, especifique a segunda coluna como BoundColumn. Para ocultar as datas dos feriados, defina a propriedade ColumnWidths da segunda coluna como zero.

Quando a propriedade Text de uma Caixa de Combinação (ComboBox) é alterada (como quando um usuário digita uma entrada no controle), o novo texto é comparado com a coluna de dados especificada por TextColumn.

Propriedade TopIndex:

Esta propriedade define e retorna o item que aparece na posição mais alta da lista. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.TopIndex [= Variante]
```

A sintaxe da propriedade TopIndex possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Variante:** Opcional. O número do item de lista que é exibido na posição mais alta. O padrão é 0 ou o primeiro item na lista.

Nota: Esta propriedade retorna o valor -1 se a lista estiver vazia ou não for exibida.

Propriedade TransitionEffect:

Esta propriedade especifica o efeito visual a utilizar ao mudar de uma página do formulário para outra. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.TransitionEffect [= fmTransitionEffect ]
```

A sintaxe da propriedade TransitionEffect possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, ao qual se aplica a propriedade.
- ▶ **fmTransitionEffect** Opcional. O efeito de transição que você deseja entre as páginas. Esta constante pode assumir os valores indicados e descritos na tabela a seguir:

Constante	Valor	Descrição
fmTransitionEffectNone	0	Sem efeito especial (padrão).
fmTransitionEffectCoverUp	1	A nova página cobre a página antiga, movendo-se da base para o topo.
fmTransitionEffectCoverRightUp	2	A nova página cobre a página antiga, movendo-se do canto inferior esquerdo para o canto inferior direito.

fmTransitionEffectCoverRight	3	A nova página cobre a página antiga, movendo-se da extremidade esquerda para a direita.
fmTransitionEffectCoverRightDown	4	A nova página cobre a página antiga, movendo-se do canto superior esquerdo para o canto inferior direito.
fmTransitionEffectCoverDown	5	A nova página cobre a página antiga, movendo-se do topo para a base.
fmTransitionEffectCoverLeftDown	6	A nova página cobre a página antiga, movendo-se do canto superior direito para o canto inferior esquerdo.
fmTransitionEffectCoverLeft	7	A nova página cobre a página antiga, movendo-se da direita para a esquerda.
fmTransitionEffectCoverLeftUp	8	A nova página cobre a página antiga, movendo-se do canto inferior direito para o canto superior esquerdo.
fmTransitionEffectPushUp	9	A nova página empurra a página antiga para fora da área de visualização, movendo-se da base para o topo.
fmTransitionEffectPushRight	10	A nova página empurra a página antiga para fora da área de visualização, movendo-se da esquerda para a direita.
fmTransitionEffectPushDown	11	A nova página empurra a página antiga para fora da área de visualização, movendo-se do topo para a base.
fmTransitionEffectPushLeft	12	A nova página empurra a página antiga para fora da área de visualização, movendo-se da direita para a esquerda.

Nota: Utilize a propriedade `TransitionPeriod` para especificar a duração de um efeito de transição.

Lição 24: User Form – Propriedades dos Controles – Parte 14

Nesta lição finalizarei a descrição das principais propriedades dos controles que podem ser utilizados em um User Form. No módulo 6, você aprenderá a utilizar os demais tipos de controles, os quais ainda não foram abordados.

Propriedade Value:

Esta propriedade especifica o estado ou conteúdo de um dado controle. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Value [= Variante]
```

A sintaxe da propriedade Value possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Variante:** Opcional. O estado ou conteúdo do controle. Esta propriedade pode assumir um dos valores indicados na tabela a seguir:

Controle	Descrição
CheckBox	Um valor inteiro indicando se o item está selecionado.
	Nulo Indica que o item está em um estado nulo, nem selecionado nem desmarcado.
	-1 Verdadeiro. Indica que o item está selecionado.
	0 Falso. Indica que o item está desmarcado.
OptionButton	Igual a CheckBox.
ToggleButton	Igual a CheckBox.
ScrollBar	Um número inteiro entre os valores especificados para as propriedades Max e Min.
SpinButton	Igual a ScrollBar.
ComboBox, ListBox	O valor na BoundColumn das linhas atualmente selecionadas.
CommandButton	Sempre False.
MultiPage	Um número inteiro indicando a página atualmente ativa.
	Zero (0) indica a primeira página. O valor máximo é um número uma unidade menor que o número de páginas.
TextBox	O texto na região de edição.

Para um CommandButton, definir a propriedade Value como True em uma macro ou procedimento inicia o evento Click do botão.

Para uma Caixa de Combinação (ComboBox), alterar o conteúdo de Value não altera o valor de BoundColumn. Para adicionar ou excluir entradas em um ComboBox, você pode utilizar o método AddItem ou RemoveItem.

Value não pode ser utilizada com uma caixa de listagem que permite seleções múltiplas.

Propriedade Visible:

Esta propriedade especifica se um objeto está visível ou oculto. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Visible [= Booleano]
```

A sintaxe da propriedade Visible possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano** Opcional. Se o objeto está visível ou oculto. Pode assumir um dos valores indicados na tabela a seguir:

Valor	Descrição
True	O objeto está visível (padrão).
False	O objeto está oculto.

Utilize a propriedade Visible para controlar o acesso a informações sem exibi-las. Por exemplo, você poderia utilizar o valor de um controle em um formulário oculto como o critério para uma consulta.

Propriedade WordWrap:

Esta propriedade indica se o conteúdo de um controle quebra automaticamente a linha no final. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.WordWrap [= Booleano]
```

A sintaxe da propriedade WordWrap possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Booleano:** Opcional. Se o controle se expande para ajustar o texto. Esta propriedade pode assumir um dos valores indicados a seguir:

Valor	Descrição
True	O texto quebra a linha (padrão).
False	O texto não quebra a linha.

Para controles que suportam a propriedade MultiLine, bem como a propriedade WordWrap, WordWrap é ignorada quando MultiLine é False.

Propriedade Zoom:

Esta propriedade especifica em quanto alterar o tamanho de um objeto exibido. Esta propriedade pode ser configurada na janela de Propriedades, durante a configuração dos controles do formulário e também pode ser configurada via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Zoom [= Inteiro]
```

A sintaxe da propriedade Zoom possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Inteiro:** Opcional. A porcentagem em que é aumentada ou diminuída a imagem exibida.

O valor da propriedade Zoom especifica uma porcentagem da ampliação ou redução da imagem em que a exibição de uma imagem deve ser alterada. Valores de 10 a 400 são válidos. O valor especificado é uma porcentagem do tamanho original do objeto; assim, uma definição de 400 significa que você deseja ampliar a imagem para quatro vezes o seu tamanho original (400 %), ao passo que uma definição de 10% significa que você deseja reduzir a imagem a um décimo de seu tamanho original (ou 10%).

Lição 25: Resumo do Módulo

Nas lições deste módulo você aprendeu a trabalhar com formulários – User Forms. Mostrei como criar um novo formulário, como adicionar e configurar controles e apresentei alguns exemplos simples de criação de formulários. Também apresentei uma descrição detalhada de, praticamente todas as propriedades disponíveis. No próximo módulo você aprenderá a trabalhar com os diversos tipos de controles disponíveis.

Módulo 5 – Criação de Aplicações Usando UserForms

- Lição 01:** User Form – Introdução e Conceito
- Lição 02:** User Form – Criando um Novo User Form
- Lição 03:** User Form – Propriedades e Eventos
- Lição 04:** User Form – Trabalhando com a Caixa de Ferramentas
- Lição 05:** User Form – Trabalhando com a Caixa de Ferramentas
- Lição 06:** User Form – Trabalhando com Controles – Parte 1
- Lição 07:** User Form – Trabalhando com Controles – Parte 2
- Lição 08:** User Form – Trabalhando com Controles – Parte 3
- Lição 09:** User Form – Trabalhando com Controles – Parte 4
- Lição 10:** User Form – Caixa de Combinação
- Lição 11:** User Form – Propriedades dos Controles – Parte 1
- Lição 12:** User Form – Propriedades dos Controles – Parte 2
- Lição 13:** User Form – Propriedades dos Controles – Parte 3
- Lição 14:** User Form – Propriedades dos Controles – Parte 4
- Lição 15:** User Form – Propriedades dos Controles – Parte 5
- Lição 16:** User Form – Propriedades dos Controles – Parte 6
- Lição 17:** User Form – Propriedades dos Controles – Parte 7
- Lição 18:** User Form – Propriedades dos Controles – Parte 8
- Lição 19:** User Form – Propriedades dos Controles – Parte 9
- Lição 20:** User Form – Propriedades dos Controles – Parte 10
- Lição 21:** User Form – Propriedades dos Controles – Parte 11
- Lição 22:** User Form – Propriedades dos Controles – Parte 12
- Lição 23:** User Form – Propriedades dos Controles – Parte 13
- Lição 24:** User Form – Propriedades dos Controles – Parte 14
- Lição 25:** Resumo do Módulo

No próximo módulo você aprenderá, em detalhes, a utilizar os diversos controles que podem ser inseridos em um User Form.

Bibliografia recomendada:

Confira as dicas de livros de Excel no seguinte endereço:

<http://www.juliobattisti.com.br/indicados/excel.asp>

Módulo 6 –Controles e Exemplos Práticos

Este é o último módulo deste curso. Neste módulo você aprenderá mais detalhes sobre os demais controles disponíveis para utilização em um User Form:

- ▶ Caixa de Listagem
- ▶ Caixa de Seleção
- ▶ Botão de opção
- ▶ Botão de ativação
- ▶ Moldura
- ▶ Botão de comando
- ▶ Barra de rolagem
- ▶ Botão de Rotação
- ▶ Imagem

Mostrarei um exemplo prático do uso de cada um destes controles. Através de exemplos passo-a-passo, você entenderá o funcionamento e quando é recomendada a utilização de cada um destes controles.

Combinando o conhecimento dos módulos 1, 2, 3 e 4, onde você aprendeu sobre a linguagem de programação VBA e o Modelo de Objetos do Excel, com o conhecimento sobre a criação de formulários no Excel, você terá uma base bem consolidada sobre a criação de aplicativos no Excel.

Na parte final deste módulo, apresentarei mais alguns exemplos práticos de códigos VBA, exemplos estes que você pode, facilmente, adaptar para usar em suas próprias planilhas.

Uma pergunta que você pode estar se fazendo é se este curso aborda tudo o que existe sobre programação VBA no Excel? Obviamente que não. O assunto programação, quer seja no Excel ou no Access, é extremamente extenso. Mesmo em um curso de 150 lições, com mais de 500 páginas de conteúdo, não é possível abordar tudo o que existe em termos de programação VBA no Excel. Alguns assuntos não foram abordados e serão abordados em um curso que pretendo escrever no decorrer de 2004 e que será denominado – Programação VBA no Excel – Volume 2. Basicamente, neste curso, serão abordados os seguintes tópicos:

- ▶ O objeto Pivot Table
- ▶ O objeto Chart
- ▶ Criação de Menus Personalizados
- ▶ Mais exemplos de User Forms
- ▶ Mais exemplos de código VBA


Um bom estudo a todos e até o próximo curso. Desejo que este curso possa ajudá-lo a progredir um pouco mais na programação VBA com o Excel.

Lição 01: User Forms – O controle Caixa de Listagem

Existe uma diferença importante entre um controle do tipo Caixa de Combinação (ComboBox) e um controle do tipo Caixa de Listagem (ListBox). Como o próprio nome sugere, em um controle do tipo Caixa de Combinação, o usuário tanto pode selecionar um dos valores da lista, quando pode digitar um valor diretamente no controle. Já em uma Caixa de Listagem, o usuário está limitado a selecionar um valor da lista. Caso o controle Caixa de Listagem tenha sido configurado para permitir seleção múltiplas, o usuário poderá selecionar várias entradas, usando as teclas Ctrl e Shift, a exemplo do que é feito para selecionar múltiplos arquivos e/ou pastas no Windows Explorer. Em resumo, em um controle do tipo Caixa de Listagem, somente é possível selecionar um ou mais valores da lista, não sendo possível digitar um novo valor, não existente na lista.

Nesta lição mostrarei como criar uma Caixa de Listagem, a qual exibe dados de duas colunas da planilha. Mostrarei como configurar a coluna que será vinculada ao controle (e o que significa uma coluna ser vinculada) e como acessar o valor retornado pelo controle, depois que um ou mais valores da lista são selecionados. Então, mãos a obra.

Exemplo: Criar um novo formulário, adicionar um controle do tipo Caixa de Listagem e utilizar suas propriedades:

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 6 - Exercício 01.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Para criar um novo formulário, selecione o comando Inserir -> UserForm.
5. Será criado um novo formulário chamado UserForm1 ou UserForm2 e assim por diante. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário.
6. Clique no formulário para selecioná-lo. Se a janela de Propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la.
7. Na janela de propriedades do formulário, localize a propriedade Name e altere o seu valor para o nome ExCaixaDeListagem e altere o valor da propriedade Caption para o texto que você quer que seja exibido na barra de títulos do formulário.
8. Vamos criar uma caixa de listagem que exibe o código e o nome do cliente. Estes valores são obtidos das colunas A e B da planilha Clientes.
9. Adicione um controle do tipo rótulo (para detalhes sobre o controle do tipo Rótulo, consulte o Módulo 5). Altere a propriedade Name do controle para rtListaClientes e a propriedade Caption para Lista de Clientes: Dimensione o controle de tal maneira que todo o texto da propriedade Caption possa ser exibido. Altere a fonte do rótulo para negrito, a cor da fonte para vermelho e a cor de fundo do rótulo para branco (a configuração destas propriedades foi detalhada no Módulo 5). Clique no formulário, fora do controle rótulo, para selecionar o formulário. Aponte o mouse para um dos quadradinhos na lateral do formulário e arraste para aumentar a largura do formulário.
10. Agora vamos adicionar e configurar um controle do tipo Caixa de Listagem. Na Caixa de ferramentas, clique no botão . Clique no formulário, à direita do rótulo. Será criado um controle do tipo Caixa de listagem, no tamanho padrão. Você pode redimensionar o controle para o tamanho desejado. Torno o controle um pouco mais largo, para que haja espaço para

exibir as colunas Código do Cliente e o nome do cliente. Aumente também a altura do controle Caixa de listagem, para que possam ser exibidas várias linhas ao mesmo tempo. Seu formulário deve estar semelhante ao indicado na Figura a seguir:



11. Agora vamos configurar a propriedade RowSource, para definir que a Caixa de Listagem deve exibir os dados das colunas A e B, da planilha Clientes, no intervalo de A1:B97. Clique no controle Caixa de Listagem para selecioná-lo. Se a janela de propriedades não estiver sendo exibida, tecle F4 para exibi-la.

12. Localize a propriedade RowSource e defina o seguinte valor para esta propriedade: **Clientes!A1:B97**

13. Agora vamos definir outras propriedades importantes do controle. Altere a propriedade Name, da Caixa de Listagem, para ListaDeClientes.

14. Altere a propriedade ColumnCount para 2, pois são duas colunas a serem exibidas.

15. Altere a propriedade BoundColumn para 2, para indicar que a segunda coluna, ou seja, o nome do Cliente, será o valor associado ao controle, quando uma entrada da lista for selecionada.

16. Altere a propriedade ColumnWidths para: 50;100, para definir a largura da primeira coluna em 50 pt e o da segunda coluna em 100 pts.

17. Muito bem, por último, vamos usar o evento Change, do controle Lista, para exibir o valor associado ao controle, que no nosso caso será o nome do Cliente selecionado, já que configuramos a propriedade BoundColumn para 2.

18. Dê um clique duplo no controle Caixa de Listagem. Será criado o esqueleto de código para o evento Click. Na lista de eventos, do lado direito, selecione o evento Change. Será criado a declaração do evento Change. Exclua a declaração do evento Click, deixando somente a declaração do evento Change, conforme indicado a seguir:

```
Private Sub ListaDeClientes_Change()  
  
End Sub
```

19. Agora insira o código a seguir, para exibir o valor associado ao controle, quando o usuário seleciona uma entrada da Caixa de Listagem:


```
Private Sub ListaDeClientes_Change()  
    MsgBox "Valor: " & ListaDeClientes.Value  
End Sub
```

20. Clique no botão fechar mais de baixo, para fechar a janela de código. Agora vamos testar o funcionamento do nosso formulário. Clique no formulário, em qualquer espaço fora dos controles, para selecioná-lo.

21. Pressione a tecla F5 para carregar o formulário. Observe que são exibidas informações do código e do nome do cliente, conforme indicado na Figura a seguir:



22. Clique em uma das entradas da lista. Por exemplo, clique no cliente cujo código é ANTON. O evento Change será disparado e será exibida a mensagem indicada na Figura a seguir:



23. Observe que valor retornado pela propriedade Value do controle é o valor da segunda coluna, ou seja, o nome do cliente. Isso ocorre porque configuramos a propriedade BoundColumn com o valor 2, indicando que o valor associado ao controle é o valor da segunda coluna.

24. Feche o formulário.

Neste exemplo foi possível observar a importância do estudo exaustivo que fizemos, no Módulo 5, sobre as propriedades disponíveis para os controles. Conhecendo bem as propriedades é possível, rapidamente, adicionar e configurar corretamente os controles disponíveis. Sempre que você ficar em dúvida em relação ao uso de uma determinada propriedade, volta ao Módulo 5 e consulte os detalhes sobre a respectiva propriedade.

Lição 02: User Forms – O controle Caixa de Seleção

Nesta lição você aprenderá a utilizar os controles do tipo Caixa de Seleção. Estes controles são exibidos no formulário no formato de um pequeno quadrado que pode estar desmarcado-

☐ Todas - ou marcado - ☒ Desenhos .

Associado a cada quadrado você deve colocar um controle do tipo Rótulo, o qual indica a que se refere cada controle. No exemplo do parágrafo anterior, associado ao controle desmarcado, foi adicionado um controle para exibir o rótulo Todas e associado ao controle marcado, foi exibido um rótulo Desenhos. Marcado significa Sim/True e desmarcado significa Não/False. O controle Caixa de Seleção, no VBA, tem o nome de CheckBox.

Utilizamos um controle do tipo CheckBox para apresentar ao usuário uma escolha entre dois valores, como Sim/Não, Verdadeiro/Falso ou Ativado/Desativado. Quando o usuário seleciona um CheckBox, ele exibe uma marca especial (como um X) e sua definição atual é Sim, Verdadeiro ou Ativado; se o usuário não selecionar o CheckBox, ele estará vazio e sua definição será Não, Falso ou Desativado. Dependendo do valor da propriedade TripleState, o CheckBox também pode ter um valor nulo, o qual normalmente é indicado pelo quadradinho com um fundo acinzentado.

Se um CheckBox estiver ligado a uma fonte de dados, a alteração da definição irá alterar o valor daquela origem. Um CheckBox desativado mostra o valor atual, mas fica esmaecido e não permite alterações do valor a partir da interface do usuário.

Você também pode utilizar caixas de verificação dentro de uma caixa de grupos para selecionar um ou mais de um grupo de itens relacionados. Por exemplo, você pode criar um formulário de pedidos que contenha uma lista de itens disponíveis e colocar um CheckBox precedendo cada item. O usuário pode selecionar um determinado item ou itens assinalando o CheckBox correspondente.

A propriedade padrão de um CheckBox é a propriedade Value.

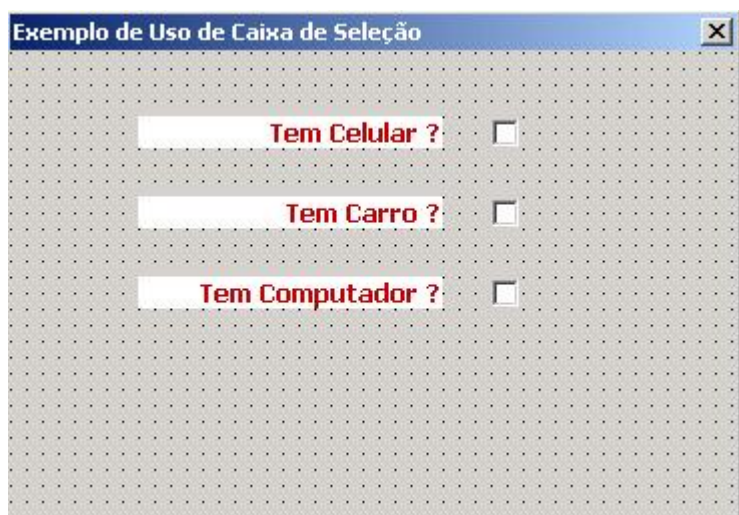
O evento padrão de um CheckBox é o evento Click.

Nesta lição mostrarei como criar um grupo de controles do tipo CheckBox e como acessar os valores retornados pelos controles, caso o usuário tenha ou não selecionado os controles.

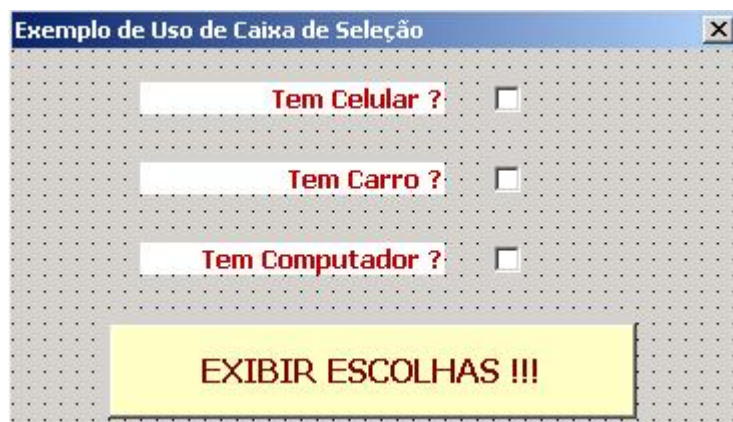
Exemplo: Criar um novo formulário, e adicionar controles do tipo CheckBox e os respectivos rótulos de identificação, bem como um botão de Comando:

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 6 - Exercício 01.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Para criar um novo formulário, selecione o comando Inserir -> UserForm.
5. Será criado um novo formulário chamado UserForm1 ou UserForm2 e assim por diante. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário.

6. Clique no formulário para selecioná-lo. Se a janela de Propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la.
7. Na janela de propriedades do formulário, localize a propriedade Name e altere o seu valor para o nome ExCaixaDeSeleção e altere o valor da propriedade Caption para o texto que você quer que seja exibido na barra de títulos do formulário.
8. Adicione três controles do tipo CheckBox e três rótulos. Altere as propriedades da fonte e da cor de fundo dos controles Rótulo e alinhe os controles conforme exemplo da Figura a seguir:



9. Altere também os nomes dos controles CheckBox, para facilitar o acesso aos controles, via código VBA. Altere a propriedade Name do primeiro CheckBox para TemCelular, a do segundo para TemCarro e a do terceiro para TemComputador.
10. Agora vamos adicionar um botão de Comando e configurar o evento Click do botão de comando, para exibir as informações selecionadas nos controles CheckBox.
11. Adicione um controle Botão de Comando na parte de baixo do formulário, altere a sua propriedade Name para ExibeDados e a sua propriedade Caption para EXIBIR ESCOLHAS !!! Altere a propriedade BackColor do botão de comando, para alterar a cor de fundo e altere a propriedade ForeColor para alterar a cor do rótulo exibido no botão. Use também a propriedade Font do botão de comando, para alterar o tamanho da fonte e para aplicar Negrito, conforme indicado na Figura a seguir:

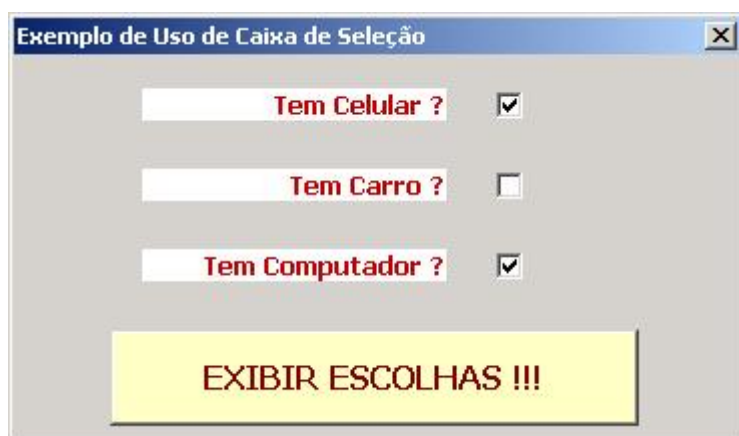


12. Agora vamos definir o código associado ao evento Click, do botão de comando. Dê um clique duplo no Botão de comando. Será criada o esqueleto de código para o evento Click. Complete o código, conforme indicado a seguir:

```
Private Sub ExibeDados_Click()  
  
    ' Verifica se o CheckBox TemCelular foi selecionado  
    If TemCelular.Value Then  
        MsgBox "Você tem Celular!!!"  
    Else  
        MsgBox "Você NÃO tem Celular!!!"  
    End If  
  
    ' Verifica se o CheckBox TemCarro foi selecionado  
    If TemCarro.Value Then  
        MsgBox "Você tem Carro!!!"  
    Else  
        MsgBox "Você NÃO tem Carro!!!"  
    End If  
  
    ' Verifica se o CheckBox TemComputador foi selecionado  
    If TemComputador.Value Then  
        MsgBox "Você tem Computador!!!"  
    Else  
        MsgBox "Você NÃO tem Computador!!!"  
    End If  
  
End Sub
```

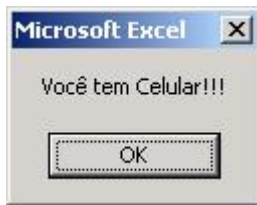
13. Clique no botão fechar mais de baixo, para fechar a janela de código. Agora vamos testar o funcionamento do nosso formulário. Clique no formulário, em qualquer espaço fora dos controles, para selecioná-lo.

14. Pressione a tecla F5 para carregar o formulário. Selecione o primeiro e o terceiro CheckBox, conforme indicado na Figura a seguir:



15. Clique no botão EXIBIR ESCOLHAS !!!

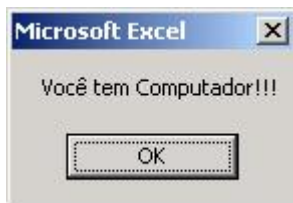
16. Será exibida a mensagem indicada na figura a seguir, pois o primeiro CheckBox foi selecionado:



17. Clique em OK. Será exibida a janela indicada na Figura a seguir, pois o segundo CheckBox não foi selecionado:




18. Clique em OK. Será exibida a janela indicada na Figura a seguir, pois o terceiro CheckBox foi selecionado:



19. Você estará de volta ao formulário. Feche o formulário.

Neste exemplo foi possível observar a criação e utilização de controles do tipo CheckBox. É possível detectar se o controle foi ou não selecionado, usando a propriedade Value. Esta propriedade retorna True, se o controle estiver selecionado e False, caso contrário.

Lição 03: User Forms – O controle Botão de Opção - OptionButton

Nesta lição você aprenderá a utilizar os controles do tipo Botão de opção. É importante entender que existe uma diferença básica entre os controles do tipo CheckBox e OptionButton. Os controles do tipo CheckBox são utilizados quando, de um grupo de opções, uma ou mais opções podem ser selecionadas. Já os controles do tipo OptionButton, são utilizados quando uma única opção pode ser selecionada, em um grupo de opções. Os controles trabalham como se fossem um único grupo, quando ao selecionar um, o que estava selecionado anteriormente é desmarcado. Estes controles são exibidos no formulário no formato de um pequeno círculo que pode estar desmarcado-  Paisagem - ou marcado -



Associado a cada controle você deve colocar um controle do tipo Rótulo, o qual indica a que se refere cada controle. No exemplo do parágrafo anterior, associado ao controle desmarcado, foi adicionado um controle para exibir o rótulo Paisagem e associado ao controle marcado, foi exibido um rótulo Retrato. Marcado significa Sim/True e desmarcado significa Não/False. O controle Botão de Opção, no VBA, tem o nome de ChekBox.

Utilize um OptionButton para mostrar se um único item em um grupo está selecionado. Observe que cada OptionButton de um Frame é mutuamente exclusivo, ou seja, somente um pode estar selecionado ao mesmo tempo.

Se um OptionButton estiver ligado a uma fonte de dados, o OptionButton poderá mostrar o valor dessa fonte de dados como Sim/Não, Verdadeiro/Falso ou Ativado/Desativado. Se o usuário selecionar o OptionButton, a definição atual será Sim, Verdadeiro ou Ativado; se o usuário não selecionar o OptionButton, a definição será Não, Falso ou Desativado. Por exemplo, um OptionButton de um aplicativo para acompanhamento de inventários poderia mostrar se um item está descontinuado. Se o OptionButton estiver ligado a uma fonte de dados, então a alteração das definições irá alterar o valor dessa fonte de dados. Um OptionButton desativado fica esmaecido e não mostra um valor.

Dependendo do valor da propriedade TripleState, um OptionButton pode, também, ter um valor nulo, o qual é indicado por uma aparência acinzentada.

Também é possível utilizar OptionButton dentro de uma caixa de grupo para selecionar um ou mais itens relacionados de um grupo. Por exemplo, você pode criar um formulário de pedidos com uma lista de itens disponíveis, com um OptionButton precedendo cada item. O usuário pode selecionar um item específico assinalando o OptionButton correspondente.

A propriedade padrão de um OptionButton é a propriedade Value. O evento padrão de um OptionButton é o evento Click. Nesta lição mostrarei como criar um grupo de controles do tipo OptionButton e como acessar os valores retornados pelos controles, caso o usuário tenha ou não selecionado os controles.

Uma pergunta que você deve estar se fazendo é: Como o Excel faz para entender que um determinado grupo de comandos OptionButton, funciona como um grupo, ou seja, que somente um deles pode estar selecionado ao mesmo tempo? A resposta a esta questão é bastante simples. Para identificar vários controles do tipo OptionButton, como sendo do

mesmo grupo, basta configurar a propriedade `GroupName` dos controles. Por exemplo, você pode adicionar quatro controles do tipo `OptionButton`, para o usuário marcar o tipo de Cartão de Crédito que ele utiliza e configurar a propriedade `GroupName` destes controles como sendo, por exemplo, `GrupoCartão`. Pronto, os quatro controles, por terem o mesmo valor na propriedade `GroupName`, passarão a atuar como um grupo, sendo que somente um dos controles do grupo, poderá estar selecionada.

Exemplo: Criar um novo formulário, e adicionar controles do tipo `OptionButton` e os respectivos rótulos de identificação, bem como um botão de Comando:

1. Abra o Excel.
2. Abra a planilha `C:\Programação VBA no Excel\ Modulo 6 - Exercício 01.xls`.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione `Alt+F11`.
4. Para criar um novo formulário, selecione o comando `Inserir -> UserForm`.
5. Será criado um novo formulário chamado `UserForm1` ou `UserForm2` e assim por diante. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário.
6. Clique no formulário para selecioná-lo. Se a janela de Propriedades não estiver sendo exibida, pressione a tecla `F4` para exibi-la.
7. Na janela de propriedades do formulário, localize a propriedade `Name` e altere o seu valor para o nome `ExBotãoDeOpção` e altere o valor da propriedade `Caption` para o texto que você quer que seja exibido na barra de títulos do formulário.
8. Adicione quatro controles do tipo `OptionButton` (🔘) e quatro rótulos. Altere as propriedades da fonte e da cor de fundo dos controles Rótulo e alinhe os controles conforme exemplo da Figura a seguir:



9. Altere também os nomes dos controles `OptionButton`, para facilitar o acesso aos controles, via código VBA. Altere a propriedade `Name` do primeiro `CheckBox` para `Visa`, a do segundo para `MasterCard`, a do terceiro para `DinnersClub` e a do quarto para `Outros`.
10. Agora o detalhe mais importante: Configure a propriedade `GroupName`, de todos os controles, como `GrupoCartão`. Ao configurar o mesmo valor para a propriedade `GroupName`, estou indicando ao Excel que os quatro controles fazem parte do mesmo grupo e que, portanto, somente um deles poderá estar selecionado ao mesmo tempo.

11. Agora vamos adicionar um botão de Comando e configurar o evento Click do botão de comando, para exibir informações sobre qual a opção foi selecionada.
12. Adicione um controle Botão de Comando na parte de baixo do formulário, altere a sua propriedade Name para ExibeDados e a sua propriedade Caption para EXIBIR ESCOLHA !!! . Altere a propriedade BackColor do botão de comando, para alterar a cor de fundo e altere a propriedade ForeColor para alterar a cor do rótulo exibido no botão. Use também a propriedade Font do botão de comando, para alterar o tamanho da fonte e para aplicar Negrito, conforme indicado na Figura a seguir:



13. Agora vamos definir o código associado ao evento Click, do botão de comando. Dê um clique duplo no Botão de comando. Será criada o esqueleto de código para o evento Click. Complete o código, conforme indicado a seguir:

```
Private Sub ExibeDados_Click()  
  
    If Visa.Value Then  
        MsgBox "O Seu Cartão é VISA !!!"  
    End If  
  
    If MasterCard.Value Then  
        MsgBox "O Seu Cartão é MASTER CARD !!!"  
    End If  
  
    If DinnersClub.Value Then  
        MsgBox "O Seu Cartão é DINNERS CLUB !!!"  
    End If  
  
    If Outros.Value Then  
        MsgBox "O Seu Cartão é OUTROS !!!"  
    End If  
  
End Sub
```

14. Clique no botão fechar mais de baixo, para fechar a janela de código. Agora vamos testar o funcionamento do nosso formulário. Clique no formulário, em qualquer espaço fora dos controles, para selecioná-lo.
15. Pressione a tecla F5 para carregar o formulário. Selecione a opção Visa, conforme indicado na Figura a seguir:



16. Agora clique em Dinners Club. Observe que a opção Visa é automaticamente desmarcada, conforme indicado na Figura a seguir. Isso acontece porque os quatro controles foram configurados como um grupo, onde somente um pode ser selecionado ao mesmo tempo. Lembre-se de que para criar um grupo de controles, você deve configurar a propriedade `GroupName`, com o mesmo valor, para todos os controles que farão parte do grupo.



17. Clique no botão EXIBIR ESCOLHA !!!
18. Será exibida a mensagem indicada na figura a seguir:



19. Você estará de volta ao formulário. Feche o formulário.

Lição 04: User Forms – O controle Botão de ativação - ToggleButton

Nesta lição você aprenderá a utilizar os controles do tipo Botão de Ativação - ToggleButton. Estes controles são exibidos no formulário no formato de botões que podem estar com a aparência de pressionados, o que indica que estão ativados, ou marcados, ou com o valor True/Yes ou com a aparência de não pressionados, quando assumem o valor False/No. Em um grupo de botões do tipo ToggleButton, mais de um botão pode estar ativado, isto é, selecionado, ou de outra forma, com a aparência de pressionado.

Utilize um ToggleButton para mostrar se um item está selecionado. Se um ToggleButton estiver ligado a uma fonte de dados, o controle ToggleButton mostrará o valor atual dessa fonte de dados como Sim/Não, Verdadeiro/Falso, Ativado/Desativado ou alguma outra escolha de duas definições. Se o usuário selecionar o ToggleButton, a definição atual será Sim, Verdadeiro ou Ativado; se ele não selecionar o ToggleButton, as definições serão Não, Falso ou Desativado. Se o ToggleButton estiver ligado a uma fonte de dados, a alteração da definição irá alterar o valor dessa fonte de dados. Um ToggleButton desativado mostra um valor, mas fica esmaecido e não permite alterações da interface do usuário.

Você pode, também, utilizar um ToggleButton dentro de um Frame para selecionar um ou mais de um grupo de itens relacionados. Por exemplo, é possível criar um formulário de pedidos com uma lista de itens disponíveis, com um ToggleButton precedendo cada item. O usuário pode selecionar um item específico selecionando o ToggleButton apropriado.

A propriedade padrão de um ToggleButton é a propriedade Value.

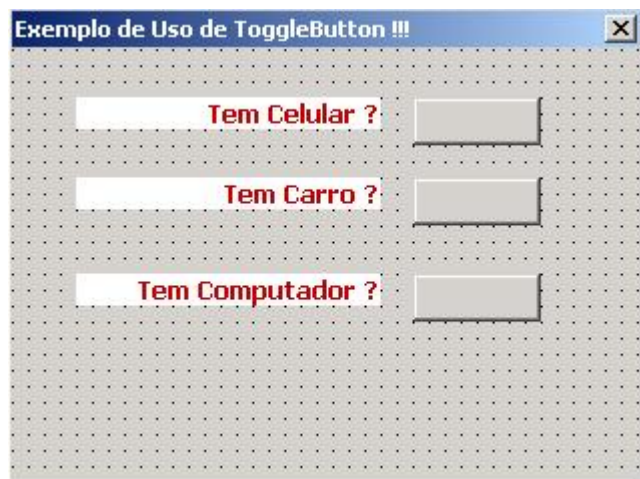
O evento padrão de um ToggleButton é o evento Click.

Nesta lição mostrarei como criar um grupo de controles do tipo ToggleButton e como acessar os valores retornados pelos controles, caso o usuário tenha ou não selecionado os controles.

Exemplo: Criar um novo formulário, e adicionar controles do tipo ToggleButton e os respectivos rótulos de identificação, bem como um botão de Comando:

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 6 - Exercício 01.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Para criar um novo formulário, selecione o comando Inserir -> UserForm.
5. Será criado um novo formulário chamado UserForm1 ou UserForm2 e assim por diante. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário.
6. Clique no formulário para selecioná-lo. Se a janela de Propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la.
7. Na janela de propriedades do formulário, localize a propriedade Name e altere o seu valor para o nome ExToggleButton e altere o valor da propriedade Caption para o texto que você quer que seja exibido na barra de títulos do formulário.

8. Adicione três controles do tipo ToggleButton e três rótulos. Altere as propriedades da fonte e da cor de fundo dos controles Rótulo e alinhe os controles conforme exemplo da Figura a seguir:



9. Altere também os nomes dos controles ToggleButton, para facilitar o acesso aos controles, via código VBA. Altere a propriedade Name do primeiro ToggleButton para TemCelular, a do segundo para TemCarro e a do terceiro para TemComputador.

10. Agora vamos adicionar um botão de Comando e configurar o evento Click do botão de comando, para exibir as informações selecionadas nos controles ToggleButton.

11. Adicione um controle Botão de Comando na parte de baixo do formulário, altere a sua propriedade Name para ExibeDados e a sua propriedade Caption para EXIBIR ESCOLHAS !!! . Altere a propriedade BackColor do botão de comando, para alterar a cor de fundo e altere a propriedade ForeColor para alterar a cor do rótulo exibido no botão. Use também a propriedade Font do botão de comando, para alterar o tamanho da fonte e para aplicar Negrito, conforme indicado na Figura a seguir:

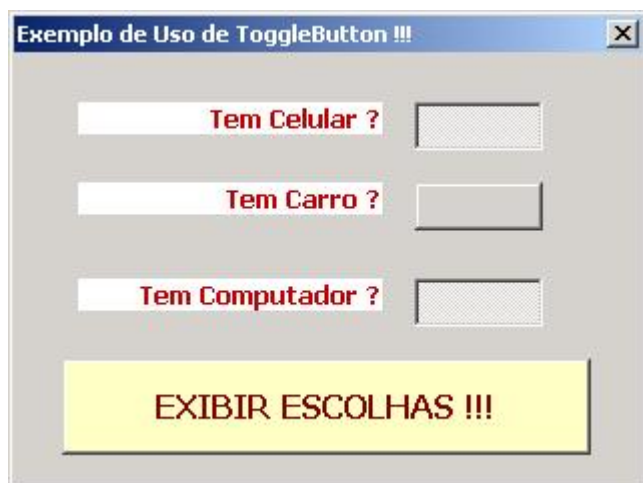


12. Agora vamos definir o código associado ao evento Click, do botão de comando. Dê um clique duplo no Botão de comando. Será criada o esqueleto de código para o evento Click. Complete o código, conforme indicado a seguir:

```
Private Sub ExibeDados_Click()  
  
    ' Verifica se o ToggleButton TemCelular foi pressionado  
    If TemCelular.Value Then  
        MsgBox "Você tem Celular!!!"  
    Else  
        MsgBox "Você NÃO tem Celular!!!"  
    End If  
  
    ' Verifica se o ToggleButton TemCarro foi pressionado  
    If TemCarro.Value Then  
        MsgBox "Você tem Carro!!!"  
    Else  
        MsgBox "Você NÃO tem Carro!!!"  
    End If  
  
    ' Verifica se o ToggleButton TemComputador foi pressionado  
    If TemComputador.Value Then  
        MsgBox "Você tem Computador!!!"  
    Else  
        MsgBox "Você NÃO tem Computador!!!"  
    End If  
  
End Sub
```

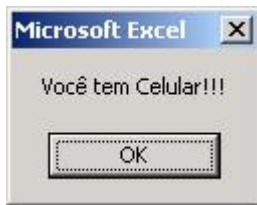
13. Clique no botão fechar mais de baixo, para fechar a janela de código. Agora vamos testar o funcionamento do nosso formulário. Clique no formulário, em qualquer espaço fora dos controles, para selecioná-lo.

14. Pressione a tecla F5 para carregar o formulário. Pressione o primeiro e o terceiro ToggleButton, conforme indicado na Figura a seguir:

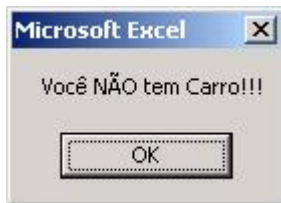


15. Clique no botão EXIBIR ESCOLHAS !!!

16. Será exibida a mensagem indicada na figura a seguir, pois o primeiro ToggleButton foi pressionado:



17. Clique em OK. Será exibida a janela indicada na Figura a seguir, pois o segundo ToggleButton não foi selecionado:



18. Clique em OK. Será exibida a janela indicada na Figura a seguir, pois o terceiro ToggleButton foi selecionado:



19. Você estará de volta ao formulário. Feche o formulário.

Neste exemplo foi possível observar a criação e utilização de controles do tipo ToggleButton. É possível detectar se o controle foi ou não selecionado, usando a propriedade Value. Esta propriedade retorna True, se o controle estiver selecionado e False, caso contrário.

Lição 05: User Forms – Os controles Botão de Comando e Frame

Nesta lição apresentarei mais detalhes sobre os controles Moldura e Botão de Comando. Em alguns exemplos anteriores, já utilizamos o controle botão de comando. Nesta lição você aprenderá mais alguns detalhes sobre este controle.

O controle Frame:

Este controle é utilizado para criar um grupo de controles funcionais e visuais. Um grupo de controles é um conjunto de controles conceitual ou logicamente relacionados. Controles conceitualmente relacionados geralmente são visualizados juntos, mas não necessariamente afetam uns aos outros. Controles que são logicamente relacionados afetam uns aos outros. Por exemplo, configurar um botão em um grupo de botões de opção configura o valor de todos os outros botões no grupo como Falso. Ou seja, utilizar um controle do tipo Frame, é outra maneira de criar um grupo de controles do tipo Botão de Opção.

Todos os botões de opção em um Frame são mutuamente exclusivos, portanto você pode utilizar um Frame para criar um grupo de opções. Você pode, também, utilizar Frame para agrupar controles que têm conteúdos estritamente relacionados. Por exemplo, em um aplicativo que processa pedidos de clientes, convém utilizar Frame para agrupar o nome, endereço e número de conta dos clientes.

Você pode, também, utilizar Frame para criar um grupo de botões alternar, mas os botões não são mutuamente exclusivos.

O evento padrão do controle Frame é o evento Click.

O controle Botão de Comando:

Este comando é utilizado para iniciar, finalizar ou interromper uma ação ou uma série de ações.

A macro ou procedimento de evento atribuído ao evento Click de um comando do tipo Botão de Comando (CommandButton) determina o que o CommandButton faz. Você pode, por exemplo, criar um CommandButton que abre outro formulário. Pode, também, exibir texto, uma figura ou ambos em um CommandButton.

A propriedade padrão de um CommandButton é a propriedade Value.

O evento padrão de um CommandButton é o evento Click.

A seguir descrevo mais algumas propriedades do controle Botão de Comando, propriedades estas que não foram descritas no módulo 5.

Propriedade Accelerator:

Esta propriedade é utilizada para definir ou recuperar a tecla aceleradora para um controle. A tecla acelerador é a tecla que corresponde ao clique no botão de comando. Por exemplo, se

um botão de comando tiver como rótulo Exibir Resultados, e a tecla acelerador for a letra E, isto significa que pressionar Alt+E é equivalente a clicar no botão de comando.

A propriedade Accelerator pode ser definida diretamente na janela de propriedades, durante a criação do formulário, ou via código VBA, usando a sintaxe a seguir:

Sintaxe:

```
objeto.Accelerator [= Seqüência]
```

A sintaxe da propriedade Accelerator possui as partes a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.
- ▶ **Seqüência** Opcional. O caractere a ser utilizado como tecla aceleradora.

Para designar uma tecla aceleradora, digite um único caractere para a propriedade Accelerator. Você pode definir Accelerator na folha de propriedades do controle ou no código. Se o valor dessa propriedade contiver mais de um caractere, o primeiro caractere da seqüência se tornará o valor de Accelerator.

Quando uma tecla aceleradora é utilizada, não há retorno visual de informações (a não ser o foco) para indicar que o controle iniciou o evento Click. Por exemplo, se a tecla aceleradora se aplica a um CommandButton, o usuário não verá o botão pressionado na interface. O botão, no entanto, receberá o foco quando o usuário pressionar a tecla aceleradora.

Se a tecla aceleradora se aplica a um Label, o controle após o Label na ordem de tabulação, e não o próprio Label, recebe o foco.

Propriedade Parent:

Retorna o nome do formulário, objeto ou coleção que contém um determinado controle, objeto ou coleção. Por exemplo, o objeto Parent, de qualquer um dos controles de um formulário é o próprio formulário.

Sintaxe:

```
objeto.Parent
```

A sintaxe da propriedade Parent possui a parte a seguir:

- ▶ **objeto:** Obrigatória. Um objeto válido, como por exemplo o nome de um controle do tipo Caixa de Combinação ou Caixa de Texto.

A propriedade Parent é somente leitura.

Utilize a propriedade Parent para acessar as propriedades, métodos ou controles do pai de um objeto.

Esta propriedade é útil em um aplicativo em que você passa objetos como argumentos. Por exemplo, você poderia passar uma variável de controle a um procedimento geral em um módulo e utilizar Parent para acessar seu formulário pai.

Exemplo de uso da propriedade Parent:

O exemplo a seguir utiliza a propriedade Parent para remeter-se ao controle ou formulário que contenha um determinado controle. Para utilizar este exemplo, copie esse código de exemplo na parte de Declarações de um formulário. Certifique-se de que o formulário contenha:

- ▶ Dois controles Label denominados Label1 e Label2.
- ▶ Um CommandButton denominado CommandButton1.
- ▶ Um ou mais controles adicionais à sua escolha.

```
Dim MyControl As Object
Dim MyParent As Object
Dim ControlsIndex As Integer

Private Sub UserForm_Initialize()
    ControlsIndex = 0
    CommandButton1.Caption = "Obter o Objeto Pai"
    CommandButton1.AutoSize = True
    CommandButton1.WordWrap = True
End Sub

Private Sub CommandButton1_Click()
    ' Acessa os controles do formulário, fazendo uso da coleção
    Controls

    Set MyControl = Controls.Item(ControlsIndex)
    Set MyParent = MyControl.Parent
    Label1.Caption = MyControl.Name
    Label2.Caption = MyParent.Name

    ' Incremente a variável ControlsIndex, para acessar o próximo
    controle

    ControlsIndex = ControlsIndex + 1
    If ControlsIndex >= Controls.Count Then
        ControlsIndex = 0
    End If
End Sub
```

Na próxima lição faremos um estudo e um exemplo prático de uso do controle Barra de Rolagem.

Lição 06: User Forms – O controle Barra de Rolagem

Nesta lição apresentarei mais detalhes sobre o controle Barra de Rolagem (ScrollBar).

O controle Barra de Rolagem (ScrollBar):

Este controle é utilizado para retornar ou define o valor de um outro controle com base na posição da Barra de Rolagem.

Um ScrollBar é um controle autônomo que você pode colocar em um formulário. Visualmente, é parecido com a barra de rolagem que se pode ver em certos objetos, como um ListBox ou a parte suspensa de um ComboBox. Entretanto, diferentemente das barras de rolagem desses exemplos, o ScrollBar autônomo não é parte integrante de nenhum outro controle.

Para utilizar ScrollBar para definir ou ler o valor de um outro controle, você deve escrever código para os eventos e métodos do ScrollBar. Por exemplo, para utilizar o ScrollBar a fim de atualizar o valor de um TextBox, você pode escrever o código que lê a propriedade Value de ScrollBar e, em seguida, define a propriedade Value de TextBox.

A propriedade padrão de um ScrollBar é a propriedade Value.

O evento padrão de um ScrollBar é o evento Change.

Observação: Para criar um ScrollBar horizontal ou vertical, arraste as alças de dimensionamento do ScrollBar horizontal ou verticalmente pelo formulário.

A seguir faremos um exemplo prático, onde um controle do tipo Barra de Rolagem, será utilizado para definir o valor de um controle do tipo Caixa de Texto.

Exemplo: Criar um novo formulário, e adicionar um controle do tipo Caixa de Texto, um controle do tipo Rótulo e um controle do tipo Barra de Rolagem. Utilizar os eventos do controle Barra de Rolagem, para definir o valor do controle Caixa de Texto.

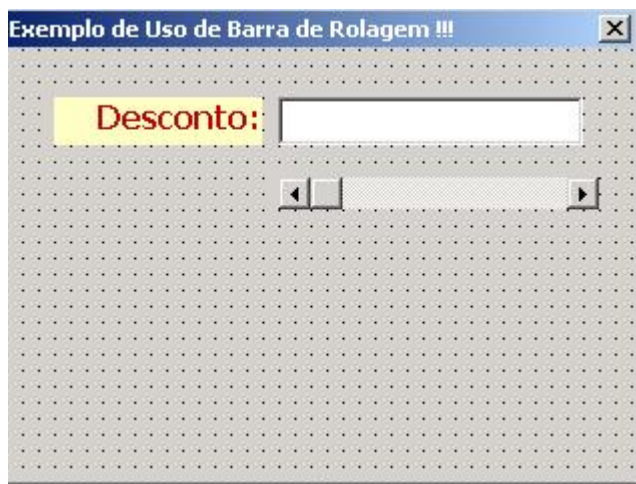
1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 6 - Exercício 01.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Para criar um novo formulário, selecione o comando Inserir -> UserForm.
5. Será criado um novo formulário chamado UserForm1 ou UserForm2 e assim por diante. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário.
6. Clique no formulário para selecioná-lo. Se a janela de Propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la.
7. Na janela de propriedades do formulário, localize a propriedade Name e altere o seu valor para o nome ExBarraDeRolagem e altere o valor da propriedade Caption para o texto que você quer que seja exibido na barra de títulos do formulário.

8. Adicione um controle do tipo Rótulo e um controle do tipo Caixa de Texto. Defina a propriedade Name, do controle Caixa de Texto, como sendo vlDesconto. Formate as propriedades da fonte do controle Rótulo, para que fique conforme indicado na Figura a seguir:



9. Agora vamos adicionar um controle do tipo Barra de Rolagem. Este controle será utilizado para definir o valor que é exibido na Caixa de Texto. Além disso, este controle definirá os limites aceitáveis para a caixa de texto, que estarão entre 20 e 50.

10. Adicione um controle do tipo Barra de Rolagem (☞), abaixo do controle Caixa de Texto. Dimensione o controle de tal maneira que ele fique uma Barra de Rolagem horizontal, conforme indicado na Figura a seguir:



11. Agora vamos passar a configuração das propriedades do controle Barra de Rolagem. Clique no controle para selecioná-lo. Se a janela de propriedades não estiver sendo exibida, pressione a tecla F4 para exibir a janela de propriedades do controle Barra de Rolagem.

12. Para definir os valores máximo e mínimo, retornados pelo controle, defina as propriedades Max e Min, respectivamente. Defina a propriedade Max como 50 e a propriedade Min como 20.

13. Altere a propriedade Name, do controle Barra de Rolagem, para BarraDesconto.

14. O passo final é configurar o evento Change, do controle Barra de Rolagem, para que seja atualizado o valor da Caixa de Texto – controle vlDesconto, sempre que o valor do controle Barra de Rolagem – controle BarraDesconto, for alterado. O controle Barra de Rolagem dispara o evento Change, sempre que o usuário arrasta a barra de posição.

15. Dê um clique duplo no controle Barra de Rolagem. Por padrão, será criada a estrutura de código para o evento Change, do controle. Insira o código de atualização da Caixa de Texto, conforme indicado a seguir:

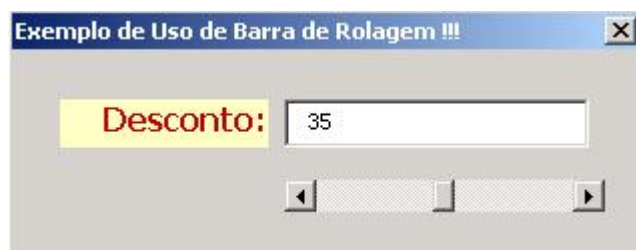
```
Private Sub BarraDesconto_Change()  
    vlDesconto.Value = BarraDesconto.Value  
End Sub
```

16. Este código simplesmente atualiza o valor do controle vlDesconto, sempre que o evento Change, da Barra de Rolagem, for disparado. Agora podemos testar o nosso formulário.

17. Clique no botão fechar (x) de baixo, para fechar a janela de código. Clique no formulário, em qualquer local fora dos controles, para selecioná-lo. Pressione a tecla F5 para carregar o formulário. Observe que o controle Caixa de Texto está vazio. Agora desloque o botão da Barra de Rolagem bem para a direita. Observe que a Caixa de Texto assume o valor 50, que é o valor máximo da Barra de Rolagem (propriedade Max=50), conforme indicado na Figura a seguir:




18. Agora arraste o botão para uma posição intermediária e libere o mouse. Observe que o valor da Caixa de Texto é atualizado, conforme indicado na Figura a seguir. Isso mostra que é possível usar um controle Barra de Rolagem, para definir o valor de uma caixa de texto:



Lição 07: User Forms – O controle Botão de Rotação

Nesta lição apresentarei mais detalhes sobre o controle Botão de Rotação (SpinButton).

Controle SpinButton:

Este controle é utilizado para incrementar e decrementar números () (1). Um clique na setinha para cima, incrementa o número. Um clique na setinha para baixo, decrementa o número.

Clicar em um SpinButton altera somente o valor do próprio SpinButton. Você pode escrever o código que utiliza o SpinButton para atualizar o valor exibido em outro controle. Por exemplo, você pode utilizar um SpinButton para alterar o mês, dia ou ano mostrados em uma data. Pode também usar um SpinButton para se movimentar por um intervalo de valores ou lista de itens ou para alterar o valor exibido em uma caixa de texto.

Para exibir um valor atualizado por um SpinButton, você deve atribuir o valor do SpinButton à parte exibida de um controle, como a propriedade Caption de um Label ou a propriedade Value de um TextBox. Para criar um SpinButton horizontal ou vertical, arraste as alças de dimensionamento do SpinButton horizontal ou verticalmente no formulário.

A propriedade padrão de um SpinButton é a propriedade Value.

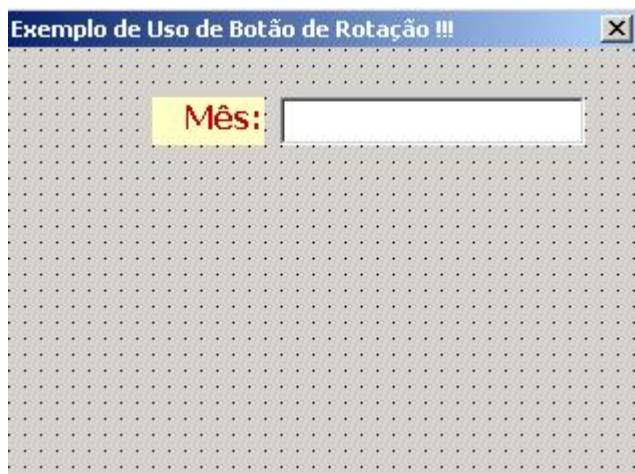
O evento padrão de um SpinButton é o evento Change.

A seguir faremos um exemplo prático, onde um controle do tipo Botão de Rotação, será utilizado para definir o valor de um controle do tipo Caixa de Texto, onde é exibido um número de mês. Limitaremos os valores aos valores válidos para o mês, ou seja, entre 1 e 12.

Exemplo: Criar um novo formulário, e adicionar um controle do tipo Caixa de Texto, um controle do tipo Rótulo e um controle do tipo Barra de Rolagem. Utilizar os eventos do controle Barra de Rolagem, para definir o valor do controle Caixa de Texto.

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 6 - Exercício 01.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Para criar um novo formulário, selecione o comando Inserir -> UserForm.
5. Será criado um novo formulário chamado UserForm1 ou UserForm2 e assim por diante. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário.
6. Clique no formulário para selecioná-lo. Se a janela de Propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la.
7. Na janela de propriedades do formulário, localize a propriedade Name e altere o seu valor para o nome ExBotãoDeRotação e altere o valor da propriedade Caption para o texto que você quer que seja exibido na barra de títulos do formulário.

8. Adicione um controle do tipo Rótulo e um controle do tipo Caixa de Texto. Defina a propriedade Name, do controle Caixa de Texto, como sendo vLMês. Formate as propriedades da fonte do controle Rótulo, para que fique conforme indicado na Figura a seguir:



9. Agora vamos adicionar um controle do tipo Botão de Rotação. Este controle será utilizado para definir o valor que é exibido na Caixa de Texto. Além disso, este controle definirá os limites aceitáveis para a caixa de texto, que estarão entre 1 e 12.

10. Adicione um controle do tipo Botão de Rotação (↕), abaixo do controle Caixa de Texto. Dimensione o controle de tal maneira que ele fique na Vertical, ao lado do controle Caixa de Texto, conforme indicado na Figura a seguir:



11. Agora vamos passar a configuração das propriedades do controle Botão de Rotação. Clique no controle para selecioná-lo. Se a janela de propriedades não estiver sendo exibida, pressione a tecla F4 para exibir a janela de propriedades do controle Barra de Rolagem.

12. Para definir os valores máximo e mínimo, retornados pelo controle, defina as propriedades Max e Min, respectivamente. Defina a propriedade Max como 12 e a propriedade Min como 1.

13. Altere a propriedade Name, do controle Botão de Rotação, para BotãoMês.

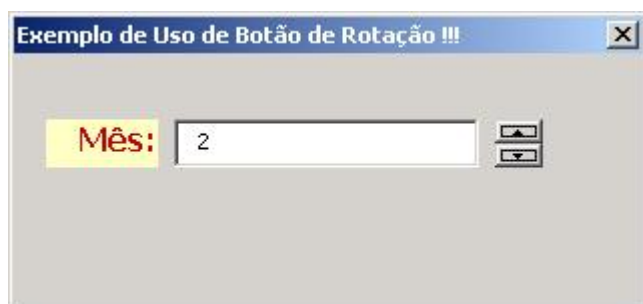
14. O passo final é configurar o evento Change, do controle Botão de Rotação, para que seja atualizado o valor da Caixa de Texto – controle vLMês, sempre que o valor do controle Botão de Rotação – controle BotãoMês, for alterado. O controle Botão de Rotação dispara o evento Change, sempre que o usuário clica na flechinha para cima ou na flechinha para baixo.

15. Dê um clique duplo no controle Botão de Rotação. Por padrão, será criada a estrutura de código para o evento Change, do controle. Insira o código de atualização da Caixa de Texto, conforme indicado a seguir:

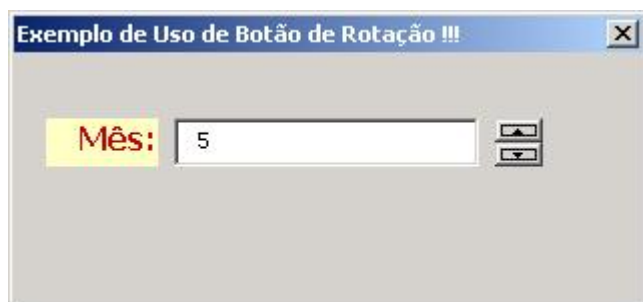

```
Private Sub BarraDesconto_Change()  
    vlMês.Value = BotãoMês.Value  
End Sub
```

16. Este código simplesmente atualiza o valor do controle vlMês, sempre que o evento Change, da Botão de Rotação, for disparado. Agora podemos testar o nosso formulário.

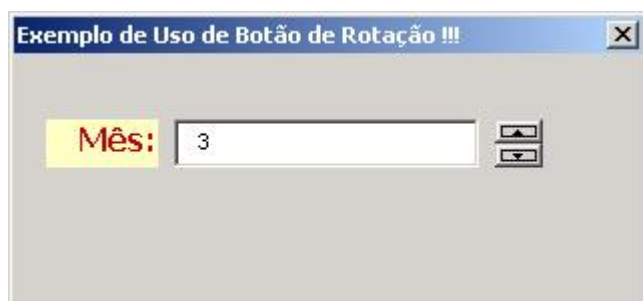
17. Clique no botão fechar (x) de baixo, para fechar a janela de código. Clique no formulário, em qualquer local fora dos controles, para selecioná-lo. Pressione a tecla F5 para carregar o formulário. Observe que o controle Caixa de Texto está vazio. Agora clique na seta para cima, no controle Botão de Rotação. Observe que será exibido o mês 2 (pois ao carregar o formulário, o controle já assume o mês 1. Ao clicar na seta para cima, ele assume o mês 2), conforme indicado na Figura a seguir:



18. Clique mais três vezes na seta para cima. Será exibido o mês 5, conforme indicado na Figura a seguir:



19. Agora clique duas vezes na seta para baixo. Será exibido o mês 3, conforme indicado na Figura a seguir. Isso comprova que o controle Botão de Rotação está funcionando corretamente, em sintonia com a Caixa de Texto:



Lição 08: User Forms – O controle Image

Nesta lição apresentarei mais detalhes sobre o controle Imagem (Image).

O Controle Image:

Este controle é utilizado para exibir uma figura em um formulário.

O controle Image permite que você exiba uma figura como parte dos dados de um formulário. Você poderia, por exemplo, utilizar um Image para exibir as fotografias dos empregados em formulários de cadastro de funcionários ou clientes.

O controle Image permite que você corte, dimensione ou aplique zoom a uma figura, mas não permite editar o seu conteúdo. Por exemplo, não é possível utilizar o Image para alterar as cores da figura, torná-la transparente ou refinar sua imagem. Para isso, você deve utilizar um software de edição de imagens, como o Photoshop ou o Fireworks.

O Image suporta os seguintes formatos de arquivo:

- ▶ *.bmp
- ▶ *.cur
- ▶ *.gif
- ▶ *.ico
- ▶ *.jpg
- ▶ *.wmf

Você pode, também, exibir uma figura em um Label. Entretanto, o Label não permite cortar, dimensionar ou aplicar zoom à figura.

O evento padrão do Image é o evento Click.

A seguir faremos um exemplo onde é adicionado e configurado um controle do tipo Image.

Exemplo: Criar um novo formulário, e adicionar um controle do tipo Rótulo e um controle do tipo Image. Adicionar também um controle do tipo Botão de Comando.

1. Abra o Excel.
2. Abra a planilha C:\Programação VBA no Excel\ Modulo 6 - Exercício 01.xls.
3. Selecione o comando **Ferramentas -> Macro -> Editor Visual Basic...** ou pressione Alt+F11.
4. Para criar um novo formulário, selecione o comando Inserir -> UserForm.
5. Será criado um novo formulário chamado UserForm1 ou UserForm2 e assim por diante. Além do formulário em branco (sem nenhum controle), também é exibida a Caixa de ferramentas, a partir da qual vamos arrastar controles para o formulário.
6. Clique no formulário para selecioná-lo. Se a janela de Propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la.

7. Na janela de propriedades do formulário, localize a propriedade Name e altere o seu valor para o nome ExImage e altere o valor da propriedade Caption para o texto que você quer que seja exibido na barra de títulos do formulário.
8. Adicione um controle do tipo Rótulo, adicione o texto e configure a fonte conforme indicado na figura a seguir. Observe que eu também alterei a cor de fundo do formulário. Para isso é utilizada a propriedade BackColor do formulário.



9. Agora vamos adicionar um controle do tipo Image e configurá-lo para exibir uma imagem. Adicione um controle Image (🖼️) e posicione-o do lado esquerdo do rótulo adicionado no passo 8.
10. Para informar a figura a ser exibida no controle Image, utilizamos a propriedade Picture do controle. Clique no controle Image para selecioná-lo. Se a janela de propriedades não estiver sendo exibida, pressione a tecla F4 para exibi-la. Clique na propriedade Picture. Será exibido um botão com o sinal de reticências ... Clique no botão de reticências. Será aberta uma janela para você selecionar o arquivo de imagem que deve ser exibido no controle Image. Selecione o arquivo a ser exibido e clique em Abrir. A imagem selecionada será exibida no controle Image, conforme exemplo da Figura a seguir:



11. Por último vamos adicionar um controle do tipo botão de comando, com o nome de `btFechar` e com o rótulo (propriedade `Caption`) igual a `Fechar Formulário !!`. Em seguida vamos configurar o exemplo `Click` deste botão, para fechar o formulário.

12. Adicione um botão de comando e configure a sua aparência, conforme indicado na figura a seguir:



13. O passo final é configurar o evento `Click`, do botão de comando, para que este feche o formulário `ExImage`. Dê um clique duplo no botão de comando. Será criada a estrutura do evento `Change`, do botão. Digite o código indicado a seguir:

```
Private Sub btFechar_Click()  
    ExImage.Hide  
End Sub
```

14. Agora vamos testar o formulário. Clique no botão fechar de baixo, para fechar a janela de código. Clique em qualquer espaço em branco, fora dos controles e pressione a tecla `F5` para carregar o formulário. Será exibida a janela indicada na Figura a seguir:



15. Clique no botão `Fechar Formulário`. Pronto, o formulário é fechado. O nosso exemplo está funcionando corretamente.

Lição 09: Exemplos práticos para o seu dia-a-dia

Muito bem, já estudamos uma série de comandos, objetos, eventos e demais itens da programação VBA no Excel. A partir desta lição, até o final deste módulo (e conseqüentemente o final do curso), apresentarei uma série de exemplos de código VBA, os quais abordam soluções para problemas práticos, do dia-a-dia do uso do Excel. Certamente você encontrará soluções para uma série de problemas com os quais já tenha se defrontado. Sempre que for necessário, acrescentarei comentários adicionais, para esclarecer pontos do código dos exemplos, pontos estes que possam gerar dúvidas. Em caso de dúvidas sobre um destes exemplos, entre em contato através do email webmaster@juliobattisti.com.br ou poste a sua dúvida no fórum de Excel Avançado e VBA, disponível no endereço: www.juliobattisti.com.br/forum

Exemplos de operações com arquivos:

Como abrir uma pasta de trabalho (Workbook), usando código VBA:

A seguir um exemplo que utilize o método Open, da coleção Workbooks:

```
Workbooks.Open filename:="c:\Planilhas\abc.xls", UpdateLinks:=False
```

Neste exemplo, utilizei o parâmetro UpdateLinks, definindo o seu valor como False. O valor False, evita que seja exibida uma Caixa de Mensagem, para que o usuário confirme se ele deseja ou não fazer a atualização de links (vínculos com outras fontes de dados), existentes nas planilhas da pasta de trabalho que está sendo aberta.

Para abrir uma pasta de trabalho como sendo Somente Leitura, utilize o commando indicado a seguir:

```
Workbooks.Open filename:="c:\Planilhas\vendas.xls", ReadOnly:=True
```

Você também pode utilizar argumentos Não Nomeados (para detalhes sobre argumentos nomeados e não nomeados, consulte as lições dos módulos 1 e 2). Lembre que ao utilizar argumentos não nomeados, o valor dos argumentos devem ser informados na mesma ordem em que os argumentos foram definidos na declaração do método. A seguir um exemplo de utilização do método Open, onde utilizo argumento não nomeados. São definidos valores para os dois primeiros argumentos definidos no método. Como não foram definidos valores para os demais argumentos, estes assumirão valores Nulos.

```
Workbooks.Open c:\data\myfile.xls", False
```

Dica: Para obter informação sobre os argumentos e a respectiva ordem de um determinado método, simplesmente posicione o cursor sobre o método e pressione a tecla F1. Será exibida uma janela de ajuda, com informações sobre o método onde estava o cursor.

Se ao invés de definir o nome da pasta de trabalho a ser aberta, diretamente no código, você quiser exibir uma janela Abrir, para que o usuário selecione o arquivo a ser aberto e retorne o nome do arquivo selecionado, utilize o código indicado a seguir:

```
fName = Application.GetOpenFileName ("Excel Files (*.xls), *.xls")
If fName <> False Then
    Workbooks.Open fName, False
End if
```

Nota: O método GetOpenFileName foi detalhadamente explicado, durante o estudo do método Application, nas lições do Módulo 3.

Como Detectar se uma Pasta de Trabalho (Workbook) já está aberta:

A seguir apresento uma função a qual é utilizada para detectar se uma pasta de trabalho, passada como parâmetro para a função, já está aberta.

```
Function PastaAberta(NomeDaPasta As String) As Boolean
    Dim wb As Workbook

    ' Verifica o nome de cada pasta de trabalho aberta, pastas
    ' estas que são acessadas através da coleção Workbooks.
    ' Se uma ocorrência for encontrada, significa que
    ' que a pasta passada como parâmetro esta aberta.
    ' Neste caso, a função retorna True
    ' Caso contrário, retorna False.

    For Each wb In Workbooks
        If LCase(wb.Name) = LCase(NomeDaPasta) Then
            PastaAberta = True
            Exit Function
        Else
            PastaAberta = False
        End If
    Next

End Function
```

A seguir uma versão, bem resumida, da função PastaAberta:

```
Function PastaAberta(NomeDaPasta As String) As Boolean
    On Error Resume Next
    PastaAberta = Len(Workbooks(wbName).Name)
End Function
```

Esta versão é bastante interessante. Primeiro habilito o tratamento de erro, fazendo com que quaisquer erros gerados sejam ignorados. Em seguida uso a função Len para detectar o tamanho do nome da pasta. Se a pasta não existir, a função Len irá retornar um erro e a função será encerrada, retornando o valor padrão para uma função do tipo Boolean, que é o valor False. Ou seja, se a pasta não estiver aberta, a função Retorna False. Se a pasta estiver aberta, a função Len não retornará um erro e o valor True será atribuído à função. Ou seja, a função retorna True se a pasta estiver aberta e False caso contrário. Exatamente como deve ser o funcionamento da função. Observe que esta segunda versão, embora seja bem mais enxuta do

que a primeira, é bem menos didática. Ou seja, no mínimo, demanda alguns explicações adicionais, para que o seu funcionamento seja entendido.

Como a função PastaAberta é uma função que retorna True ou False, ela pode ser utilizada diretamente em um teste If, como no exemplo a seguir:

```
If PastaAberta("C:\Dados\Vendas.xls") Then
    ' Comandos a serem executados se a pasta de trabalho estiver aberta
End If
```

A seguir mostro um exemplo de comandos a serem executados quando a pasta não estiver aberta:

```
If Not PastaAberta("C:\Dados\Vendas.xls") Then
    ' Comandos caso a pasta não esteja aberta.
End If
```

Outras variações também poderão ser criadas, para verificar se uma pasta de trabalho está aberta. São muitas as possibilidades, limitadas apenas pela criatividade do programador. A seguir temos uma terceira abordagem, na qual não é utilizado um loop através da coleção Workbooks.

```
Sub test()
    Dim wkName As String

    ' Atribuo um valor vazio a String wkName
    wkName = ""

    ' Atribuo o nome da pasta de trabalho à variável wkName
    ' Uso novamente a abordagem de habilitar o tratamento de erro
    ' e o uso da função Len

    On Error Resume Next
    wkName = Workbooks("BOOK1.XLS").Name

    ' Desabilito o tratamento de erro
    on Error GoTo 0

    ' Se a pasta de trabalho estiver aberta, A função len irá
    retornar
    ' um valor maior do que 0, o que significa True.

    If Len(wkName) = 0 Then
        MsgBox "A Pasta de Trabalho não está aberta !!!"
    End If
Else
    MsgBox "A Pasta de Trabalho está aberta !!!"
End If
End Sub
```


Lição 10: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Como testar se um arquivo existe, antes de abri-lo:

Antes de utilizar o método Open, para abrir uma pasta de trabalho, é recomendado que você teste se o arquivo existe. Se você utilizar o método Open, para tentar abrir um arquivo que não existe, será gerado um erro em tempo de execução.

A função a seguir pode ser utilizada para testar a existência de um arquivo, antes de utilizar o método Open para abri-lo:

```
Sub AbreArquivoSeExiste()  
    Dim FName As String  
    FName = "C:\Dados\Vendas.xls"  
    If Dir(FName) = "" Then  
        MsgBox FName & " Não existe"  
    Else  
        Workbooks.Open FileName:=FName, updateLinks:=False  
    End If  
End Sub
```

Este exemplo utiliza uma função bastante útil, a função DIR, a qual está descrita a seguir.

A Função DIR:

Esta função retorna uma String que representa o nome de um arquivo, diretório ou pasta que corresponde a um padrão ou atributo de arquivo especificado ou o rótulo de volume de uma unidade.

Sintaxe:

Dir[(pathname[, attributes])]

A sintaxe da função Dir possui as partes a seguir:

Parte	Descrição
pathname	Opcional. Expressão de sequência que especifica um nome de arquivo – pode incluir o diretório ou a pasta e a unidade. Retorna uma sequência de comprimento zero ("") se pathname não for encontrado.
attributes	Opcional. Constante ou expressão numérica, cuja soma especifica atributos de arquivo. Se omitido, retorna arquivos que correspondem ao pathname mas não possuem atributos.

O argumento attributes pode assumir os valores indicados na tabela a seguir:

Constante	Valor	Descrição
vbNormal	0	(Padrão) Especifica os arquivos que não possuem atributos.
vbReadOnly	1	Especifica arquivos somente leitura além dos arquivos que não possuem atributos.
vbHidden	2	Especifica arquivos ocultos além dos arquivos que não possuem atributos.
VbSystem	4	Especifica arquivos do sistema além dos arquivos que não possuem atributos. Não disponível no Macintosh.
vbVolume	8	Especifica rótulo de volume; se qualquer outro atributo for especificado, vbVolume será ignorado. Não disponível no Macintosh.
vbDirectory	16	Especifica diretórios ou pastas além dos arquivos que não possuem atributos.
vbAlias	64	O nome do arquivo especificado é um alias. Disponível somente no Macintosh.

Observação: Estas constantes são especificadas pelo Visual Basic for Applications e podem ser usadas em qualquer parte do seu código no lugar dos valores reais.

Importante: No Microsoft Windows, Dir aceita o uso de curingas de múltiplos caracteres (*) e de um único caractere (?) para a especificação de múltiplos arquivos. No Macintosh, esses caracteres são tratados como caracteres de nomes de arquivo válidos e não podem ser usados como curingas para especificar múltiplos arquivos.

Como o Macintosh não aceita os curingas, use o tipo de arquivo para identificar grupos de arquivos. Você pode usar a função MacID para especificar o tipo de arquivo em vez de usar os nomes de arquivo. Por exemplo, a instrução a seguir retorna o nome do primeiro arquivo TEXT da pasta atual:

```
Dir("AlgumCaminho", MacID("TEXT"))
```

Para iterar todos os arquivos em uma pasta, especifique uma sequência de caracteres vazia:

```
Dir( " " )
```

Cuidado: Se você usar a função MacID com Dir no Microsoft Windows, ocorrerá um erro.

Qualquer valor de attribute maior que 256 é considerado um valor MacID.

Você deve especificar pathname na primeira vez que chamar a função Dir ou um erro será gerado. Se especificar também atributos de arquivo, será necessário incluir pathname.

Dir retorna o primeiro nome de arquivo que corresponda a pathname. Para obter qualquer nome de arquivo adicional que corresponda a pathname, chame Dir novamente sem argumentos. Quando não houver mais nenhum nome de arquivo correspondente, Dir retornará

uma sequência de comprimento zero (""). Uma vez que essa sequência seja retornada, você deverá especificar pathname em chamadas subsequentes ou um erro será gerado. Você pode mudar para um novo pathname sem recuperar todos os nomes de arquivo que correspondam ao pathname atual. Entretanto, você não pode chamar a função Dir recorrentemente. Chamar Dir com o atributo vbDirectory não retorna subdiretórios continuamente.

Dica: Em razão de os nomes de arquivo serem recuperados sem nenhuma ordem em particular, pode ser que você deseje armazenar nomes de arquivo retornados em uma matriz e, em seguida, classificá-la.

Exemplos de uso da função Dir:

Este exemplo usa a função Dir para verificar se determinados arquivos e diretórios existem. No Macintosh, "HD:" é o nome da unidade padrão e as partes do nome do caminho são separadas por dois pontos em vez de barras invertidas. Além disso, os caracteres curinga do Microsoft Windows são tratados como caracteres de nome de arquivo válidos no Mac. No entanto, você pode usar a função MacID para especificar os grupos de arquivos.

```
Dim MyFile, MyPath, MyName
```

```
' Retorna "WIN.INI" (no Microsoft Windows) se existir.
```

```
MyFile = Dir("C:\WINDOWS\WIN.INI")
```

```
' Retorna o nome do arquivo com a extensão especificada. Se existir mais de um arquivo *.ini,  
' o primeiro arquivo encontrado será retornado.
```

```
MyFile = Dir("C:\WINDOWS\*.INI")
```

```
' Chamar Dir novamente sem argumentos para retornar o próximo arquivo *.INI no  
' mesmo diretório.
```

```
MyFile = Dir
```

```
' Retornar primeiro arquivo *.TXT com um atributo oculto definido.
```

```
MyFile = Dir("*.TXT", vbHidden)
```

```
' Exibir o nome de todas as pastas do drive C:\
```

```
MyPath = "c:\" ' Definir o caminho.
```

```
MyName = Dir(MyPath, vbDirectory) ' Recuperar a primeira entrada.
```

```
Executar While MyName <> "" ' Iniciar o loop.
```

```
    ' Ignorar o diretório atual e o diretório abrangente.
```

```
    If MyName <> "." And MyName <> ".." Então
```

```
        ' Usar a comparação bit a bit para se certificar de que MyName é um diretório.
```

```
        If (GetAttr(MyPath & MyName) And vbDirectory) = vbDirectory Then
```

```
            Debug.Print MyName ' Exibir a entrada somente se
```

```
            End If ' representar um diretório.
```

```
        End If
```

```
        MyName = Dir ' Obter próxima entrada.
```

```
Loop
```

Lição 11: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Como Salvar e Fechar todas as pastas de trabalho, menos a pasta de trabalho ativa:

O exemplo de código a seguir, percorre a coleção Workbooks e fecha todas as pastas de trabalho, com exceção da pasta de trabalho ativa, ou seja, a pasta de trabalho onde o está o código:

```
For Each wb In Workbooks
    If Not wb Is ThisWorkbook Then
        wb.Close SaveChanges:=True
    End If
Next wb
```

Nota: Para fechar as pastas de trabalho, sem salvá-las antes, atribui o valor False, para o parâmetro SaveChanges, do método close, conforme indicado no exemplo a seguir:

```
For Each wb In Workbooks
    If Not wb Is ThisWorkbook Then
        wb.Close SaveChanges:=False
    End If
Next wb
```

Como retornar o caminho completo e o nome de uma pasta de trabalho:

Para retornar o nome completo e o caminho de uma pasta de trabalho, utilizamos o método FullName, conforme exemplo a seguir:

```
FullPath = ActiveWorkbook.FullName
```

Você também pode utilizar o objeto ThisWorkbook, para fazer referência à pasta de trabalho que contém o código em execução:

```
FullPath = ThisWorkbook.FullName
```

Importante: Se a pasta de trabalho contiver alterações que ainda não foram salvas, o método FullName irá retornar apenas o nome da pasta de trabalho e não o caminho completo.

Como detector a data e a hora da última vez em que a pasta de trabalho foi salva:

O exemplo a seguir, mostra a utilização de uma variável do tipo Variant, a qual recebe o valor da data e hora da última alteração de um determinado arquivo:

```
dModified = FileDateTime("C:\dados\vendas.xls")
```

Para isso usamos a função FileDateTime, a qual é detalhada a seguir.

Função FileDateTime:

Esta função retorna uma Variant (Date) que indica a data e hora em que um arquivo foi criado ou modificado pela última vez.

Sintaxe:

```
FileDateTime(pathname)
```

O argumento pathname obrigatório é uma expressão de sequência de caracteres que especifica um nome de arquivo. O pathname pode incluir o diretório ou pasta e a unidade.

Exemplo da função FileDateTime:

Este exemplo usa a função FileDateTime para determinar a data e a hora em que um arquivo foi criado ou modificado pela última vez. O formato da data e hora exibido é baseado nas definições de localidade do seu sistema.

```
Dim MyStamp
```

```
' Assumir que ARQTESTE foi modificado pela última vez em
```

```
' 12 de fevereiro de 1993, às 4:35:47 PM.
```

```
' Assumir as definições de localidade Inglês/EUA.
```

```
MyStamp = FileDateTime("ARQTESTE")
```

```
' Retorna "2/12/93 4:35:47 PM".
```

Abrir o arquivo modificado mais recentemente, em uma pasta:

A seguir um exemplo de um procedimento, cujo objetivo é identificar e abrir a pasta de trabalho modificada mais recentemente em um determinado diretório. Você poderia fazer modificações para fazer com que o caminho completo da pasta a ser analisada fosse passado como parâmetro para o procedimento, ao invés de colocar este caminho diretamente no código. Eis o exemplo da função AbreOMaisNovo:

```
Sub AbreOMaisNovo( )

    Dim dirName As String
    Dim fName As String
    Dim fileTime As Date
    Dim fileName As String
    Dim latestFile As String

    ' Informa o diretório a ser analisado

    dirName = "C:\Planilhas\"

    ' Pesquisa os arquivos no diretório
    fName = Dir(dirName & ".*")

    ' Define valores que poderão ser alterados mais adiante.
```

```
latestFile = fName
fileTime = FileDateTime(dirName & fName)

' Faz um loop até que todos os arquivos tenham sido processados

While fName <> ""

If FileDateTime(dirName & fName) > fileTime Then

' Se for encontrada um arquivo mais recente, atualiza as variáveis.

latestFile = fName
fileTime = FileDateTime(dirName & fName)
End If

' Localiza o próximo arquivo. Para isso uso a função Dir,
' sem parâmetros, conforme descrito anteriormente.

fName = Dir()
Wend

If latestFile = "" Then
MsgBox "Nenhum arquivo foi encontrado!!!"
Else
Workbooks.Open "C:\Planilhas\" & latestFile
End If

End Sub
```

A lógica do procedimento é bastante simples. Basicamente é um loop através de todos os arquivos do diretório. A cada passagem do laço, comparo a data de atualização do arquivo que está sendo processado, com a data mais atual até o momento. Se a data do arquivo que está sendo processado for mais atual, atualizo as variáveis que contém o nome do arquivo mais atual e a respectiva data e sigo para o próximo arquivo. O nome do próximo arquivo é obtido com o uso da função Dir, sem nenhum parâmetro, conforme descrevi na Lição 11.

O resultado é que, ao final do laço, a variável latestFile irá conter o nome do arquivo mais recente.

Lição 12: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Contar quantas pastas de trabalho (não ocultas), estão abertas atualmente:

A função a seguir, retorna o número de pastas de trabalho abertas e visíveis (não ocultas). A função assume que existe uma única janela para cada pasta de trabalho. Observe o uso da coleção Workbooks, do objeto Application.

```
Function ContarPastasVisíveis( )  
  
    Dim PastasVisíveis As Long  
    Dim bookCounter As Workbook  
  
    For Each bookCounter In Application.Workbooks  
        If Windows(bookCounter.Name).Visible = True Then  
            PastasVisíveis = PastasVisíveis + 1  
        End If  
    Next bookCounter  
  
    ContarPastasVisíveis = PastasVisíveis  
  
End Function
```

Como acessar valores de planilhas de pastas de trabalho fechadas:

Você pode acessar valores de planilhas em pastas de trabalho, sem ter aberto a pasta de trabalho. Para isso você utiliza algumas macros prontas, que estão disponíveis desde o Excel 4.0, conforme exemplos a seguir:

```
' Acessa o valor da célula A1, da planilha Vendas.xls  
x=ExecuteExcel4Macro(" 'C:\[Vendas.xls]Plan1'!R1C1")
```

Neste exemplo, R1C1 significa linha 1 (Row 1 – R1), coluna 1 (Column 1 – C1)

```
' Retorna o número de células, que contém valores, na faixa A1:C3  
x=ExecuteExcel4Macro("COUNT( 'C:\[Vendas.xls]Sheet1'!R1C1:R3C3)")
```

Neste exemplo, R1C1:R3C3 significa da linha 1, coluna 1 até a linha 3, coluna 3, ou seja, A1:C3

```
' Retorna média no intervalo A1:B2  
x=ExecuteExcel4Macro("AVERAGE( 'C:\[Vendas.xls]Sheet1'!R1C1:R2C2)")
```

Observações: É importante salientar que o sinal de igual não é utilizado, no caso de fórmulas, como no segundo e terceiro exemplos. Além disso, é utilizada a notação de endereços RC, ou seja Linha, Coluna

Você também pode acessar valores em planilhas de uma pasta de trabalho fechada, sem ter que utilizar as macros do Excel 4.0. O exemplo a seguir mostra como fazer isso:

```
Sub GetClosedBookValue()  
  
    Dim theValue  
    ' A formula a seguir faz referência a valores de uma pasta  
    ' de trabalho que está fechada.  
  
    Range("A1").Formula='D:\[Vendas.xls]Plan11'!$A$1"  
  
    ' Atualiza os links  
    ActiveWorkbook.UpdateLink ("D:\Vendas.xls")  
  
    ' Extrai o valor atualizado e limpa a fórmula  
  
    theValue = Range("A1").Value  
    Range("A1").ClearContents  
End Sub
```

A seguir mais alguns exemplos de extração de valores em uma pasta de trabalho fechada:

O exemplo a seguir extrai o valor da célula A1 (R1C1), da planilha Plan1:

```
x=ExecuteExcel4Macro(" 'C:\[Vendas.xls]Plan1'!R1C1")
```

O exemplo a seguir, retorna a media da faixa A1:B2:

```
x=ExecuteExcel4Macro("AVERAGE('C:\[Vendas.xls]Plan1'!R1C1:R2C2)")
```

Como definir o diretório padrão para abertura de arquivos:

A pasta padrão que é acessada, quando você usa o método `Application.GetOpenFilename` é também a pasta padrão, que é exibida, quando você utiliza o comando Arquivo -> Abrir. O exemplo de código altera o diretório padrão, o que é confirmado pelo uso do método `GetOpenFilename` e, após a utilização deste método, o diretório padrão é definido de volta para o valor original. Isso é possível, pois armazenamos o caminho original nas variáveis `fNameAndPath`, `currentDir` e `CurrentDrive`:

```
Sub ExTrocaPastaPadrão( )  
    Dim fNameAndPath As String  
    Dim currentDir As String  
    Dim currentDrive As String  
  
    'Armazena os valores do caminho atual  
  
    currentDir = CurDir()  
    currentDrive = Left(currentDir, 1)  
  
    'Define um novo caminho padrão  
  
    ChDrive "C"  
    ChDir "C:\Planilhas"
```

```
fNameAndPath = Application.GetOpenFilename

'Retorna aos valores originais

ChDrive currentDrive
ChDir currentDir

End Sub
```

Trabalhando com o disquete:

O exemplo a seguir alterna o drive corrente para o disquete e faz o tratamento de vários erros que podem ocorrer:

```
Sub Drive_Change( )
    On Error GoTo eTrap
    ChDrive "A"
    Exit Sub

eTrap:
    If Err.Number = 68 Then
        MsgBox "Insira um disco no drive"
        Resume
    ElseIf Err.Number = 70 Then
        MsgBox "Verifique se o disco não está protegido contra
gravação"
        Resume
    ElseIf Err.Number = 71 Then
        MsgBox "O disco não está pronto ou não está formatado!!!"

    Resume
    Else
        MsgBox "O drive não pode ser acessado!!!"
        End
    End If
End Sub
```

Lição 13: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha. Vamos ver mais algumas funções para o trabalho com arquivos, diretamente no VBA. Por exemplo, é possível copiar, excluir e renomear arquivos, diretamente no código VBA.

Copiando, renomeando e movendo arquivos com o VBA:

O exemplo a seguir mostra o uso da função FileCopy, para copiar um arquivo usando o VBA.

```
Dim oldFile As String
Dim newFile As String

' Define os arquivos de origem e destino

oldFile = "c:\temp\book2.xls"
newFile = "c:\active\book2.xls"

' Efetua a cópia do arquivo definido em oldFile, para o destino
' definido em newFile

FileCopy oldFile, newFile
```

Importante: Você deve ter cuidado com o uso da função FileCopy, pois se o arquivo de destino já existir, não será exibida nenhuma mensagem informando que o arquivo será sobrescrito. A seguir mostra uma abordagem para verificar se o arquivo já existe, caso já exista, o comando End será utilizado para encerrar o procedimento e não sobrescrever o arquivo já existente:

```
Dim oldFile As String
Dim newFile As String

' Define os arquivos de origem e destino

oldFile = "c:\temp\book2.xls"
newFile = "c:\active\book2.xls"

If Dir(oldFile) <> "" Then
    MsgBox "O arquivo " & oldFile & " Já existe. " & "Não será copiado!!"
End
Else
    ' Efetua a cópia do arquivo definido em oldFile, para o destino
    ' definido em newFile

    FileCopy oldFile, newFile

End If
```

A Instrução FileCopy:

Esta instrução é utilizada para copiar um arquivo.

Sintaxe:

`FileCopy source, destination`

A sintaxe da instrução FileCopy tem os seguintes argumentos nomeados:

Parte	Descrição
source	Obrigatória. Expressão de sequência de caracteres que especifica o nome do arquivo a ser copiado. A parte source pode incluir diretório ou pasta e unidade de disco.
destination	Obrigatória. Expressão de sequência de caracteres que especifica o nome do arquivo de destino. O destination pode incluir diretório ou pasta e unidade de disco.

Nota: Se você tentar usar a instrução FileCopy em um arquivo atualmente aberto, um erro será gerado.

Movendo um arquivo:

A seguir um exemplo de uso da instrução Name (a qual será detalhada logo em seguir), para mover um arquivo:

```
Dim oldFile As String
Dim newFile As String

' Define origem e destino

oldFile = "c:\temp\book2.xls"
newFile = "c:\active\book2.xls"

' Uso a instrução name para mover o arquivo

Name oldFile As newFile
```

As pastas de origem e destino devem estar no mesmo drive. Se as pastas de origem e destino estiverem em drives diferentes, você deve usar uma instrução FileCopy para copiar o arquivo de origem para o destino e depois uma instrução Kill, para excluir o arquivo original.

Se o arquivo a ser movido já existir no destino, uma mensagem de erro será gerada, ao utilizar a instrução Name. Você pode usar o commando Dir para detector se o arquivo já existe no destino e tomar as ações necessárias, antes de usar o comando Name, como no exemplo a seguir:

```
If Dir(newFile) Then
    ' Comandos a serem executados, caso o arquivo de destino
    ' já exista.
End If
```

Por exemplo, você pode excluir o arquivo de destino antes de mover o arquivo de origem. Para isso você usa o commando Kill, conforme exemplo a seguir:

```
If Dir(newFile) Then  
    Kill newfile  
End If
```

Dica: Para renomear um arquivo, basta especificar a origem e o destino iguais, quando utilizar a instrução Name, conforme exemplo do código a seguir:

```
Dim oldName As String  
Dim newName As String  
  
oldName = "c:\temp\book2.xls"  
newName = "c:\temp\studies.xls"  
  
Name oldFile As newFile
```

Instrução Name:

Esta instrução é utilizada para renomear um arquivo, diretório ou pasta em disco.

Sintaxe:

```
Name oldpathname As newpathname
```

A sintaxe da instrução Name possui as partes a seguir:

Parte	Descrição
oldpathname	Obrigatória. Expressão de sequência de caracteres que especifica o nome e localização de arquivo existente – pode incluir o diretório ou a pasta e a unidade.
newpathname	Obrigatória. Expressão de sequência de caracteres que especifica o novo nome de arquivo e sua localização – pode incluir diretório ou pasta e unidade. O nome de arquivo especificado por newpathname não pode já existir.

A instrução Name renomeia um arquivo e o move para um diretório ou pasta diferente, se necessário. Name pode mover um arquivo através de unidades, mas somente pode renomear um diretório ou pasta existente quando newpathname e oldpathname estiverem localizados na mesma unidade. Name não pode criar um novo arquivo, diretório ou pasta.

O uso de Name em um arquivo aberto gera um erro. Você deve fechar um arquivo que esteja aberto antes de renomeá-lo. Os argumentos de Name não podem conter curingas de múltiplos caracteres (*) nem de caractere único (?).

Lição 14: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Uma rotina para copiar todas os arquivos .xls de uma pasta para outra:

A seguir apresento o código de um procedimento, o qual copia todos os arquivos .xls de uma pasta de trabalho de origem, para uma pasta de trabalho de destino:

```
Sub CopiaTodosXLS()  
  
    Dim fName As String  
    fName = Dir("d:\Dados\*.xls")  
  
    Do  
        FileCopy "d:\Vendas" & fName , "d:\Planilhas\" & fName  
        ' Uso Dir sem parâmetros, para obter o próximo arquivo  
        fName = Dir( )  
    Loop Until fName = ""  
  
End Sub
```

Como excluir um arquivo:

Para excluir arquivos você deve utilizar o comando Kill. Não é possível desfazer a exclusão de um arquivo que foi excluído com o comando Kill. O arquivo a ser excluído não pode estar aberto, se não será gerado um erro em tempo de execução.

A seguir temos um exemplo de uso do comando Kill:

```
If Dir("C:\Dados\Planilhas\Vendas.xls") <> "" Then  
    Kill " C:\Dados\Planilhas\Vendas.xls "  
End If
```

O comando Dir é utilizado para verificar se o arquivo a ser excluído realmente existe. Se o arquivo existir, o teste será verdadeiro e o comando Kill dentro do If, será executado, excluindo efetivamente o arquivo. Se você não verificar se o arquivo existe e tentar executar o comando Kill com um arquivo que não existe, será gerado um erro em tempo de execução.

Outra abordagem possível seria utilizar o tratamento de erros do VBA, conforme exemplo a seguir:

```
' Desabilita a geração de mensagem de erros  
On Error Resume Next  
  
' Exclui o arquivo  
Kill "C:\Fidelio\Quotes\temp.xls"  
  
' habilita novamente o gerenciamento de erro  
On Error GoTo 0
```

Você também pode utilizar caracteres coringa, com o comando Kill, como no exemplo a seguir, onde serão excluídos, todos os arquivos .doc, da pasta C:\Dados:

```
Kill "C:\Dados\*.doc"
```

A Instrução Kill:

A instrução Kill é utilizada para excluir arquivos.

Sintaxe:

```
Kill pathname
```

O argumento pathname é uma expressão de sequência de caracteres que especifica um ou mais nomes de arquivo a serem excluídos. O pathname pode incluir o diretório ou pasta e a unidade.

No Microsoft Windows, Kill aceita o uso de curingas de múltiplos caracteres (*) e de um único caractere (?) para especificar múltiplos arquivos. Entretanto, no Macintosh, esses caracteres são tratados como caracteres de nomes de arquivo válidos e não podem ser usados como curingas para especificar múltiplos arquivos.

Como o Macintosh não aceita os curingas, use o tipo de arquivo para identificar grupos de arquivos a excluir. Você pode usar a função MacID para especificar o tipo de arquivo em vez de repetir o comando com nomes de arquivo separados. Por exemplo, a instrução a seguir exclui todos os arquivos TEXT da pasta atual.

```
Kill MacID("TEXT")
```

Se você usar a função MacID com Kill no Microsoft Windows, ocorrerá um erro.

Se você tentar usar Kill para excluir um arquivo aberto, ocorrerá um erro.

Observação Para excluir pastas, use a instrução RmDir.

Exemplos da instrução Kill:

Este exemplo usa a instrução Kill para excluir um arquivo de um disco.

' Assumir que ARQTESTE é um arquivo que contém alguns dados.

```
Kill "ArqTeste"
```

' Excluir todos os arquivos *.TXT do diretório atual:

```
Kill "*.TXT"
```


Instrução Rmdir:

Esta instrução remove um diretório ou pasta existente.

Sintaxe:

```
Rmdir path
```

O argumento path obrigatório é uma expressão de sequência de caracteres que identifica o diretório ou pasta a ser removida. O path pode incluir a unidade. Se uma unidade não for especificada, Rmdir remove o diretório ou pasta da unidade atual.

Se você tentar usar Rmdir em um diretório ou pasta que contém arquivos, um erro será gerado. Use a instrução Kill para excluir todos os arquivos antes de tentar a remoção de um diretório ou pasta.

Exemplo da instrução Rmdir:

Este exemplo usa a instrução Rmdir para remover um diretório ou pasta existente.

' Assumir que MEUDIR é um diretório ou pasta vazia, sem nenhum arquivo.

```
Rmdir "MEUDIR"
```

Instrução Mkdir:

Esta instrução é utilizada para criar um novo diretório ou pasta.

Sintaxe:

```
Mkdir path
```

O argumento path obrigatório é uma expressão de sequência de caracteres que identifica o diretório ou pasta a ser criado. O path pode incluir a unidade. Se uma unidade não for especificada, Mkdir cria o novo diretório ou pasta na unidade atual.

Exemplo da instrução Mkdir:

Este exemplo usa a instrução Mkdir para criar um diretório ou pasta. Se a unidade de disco não for especificada, o novo diretório ou pasta será criado na unidade atual.

```
' Criar novo diretório ou pasta.  
Mkdir "C:\Dados\Planilhas"
```

Lição 15: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Como obter as propriedades de um arquivo:

Os arquivos do Windows possuem uma série de propriedades, tais como Somente Leitura, Oculto e assim por diante. O VBA disponibiliza uma série de funções, as quais podem ser utilizadas para retornar informações sobre as propriedades de um arquivo.

' A função FileDateTime retorna a data da última alteração no arquivo.

' Se o arquivo estiver aberto, será retornada a hora de abertura.

```
FileDateTime("C:\Dados\Vendas.xls")
```

' A função FileLen retorna o tamanho do arquivo em Bytes.

```
FileLen("C:\Dados\Vendas.xls")
```

O exemplo de código a seguir pode ser utilizado para listar as diferentes propriedades (e respectivos valores) de um arquivo. Algumas das propriedades típicas são o Author, assunto, comentários, data de criação e assim por diante.

```
Sub ListProperties( )
    Dim rw, p
    rw = 1

    On Error Resume Next
    ' Faz um loop através de todas as propriedades e grava
    ' os respectivos valores em células da planilha.

    For Each p In ActiveWorkbook.BuiltinDocumentProperties
        Cells(rw, 1).Value = rw
        Cells(rw, 2).Value = p.Name
        Cells(rw, 3).Value = p.Value
        rw = rw + 1
    Next

    Columns("a:c").AutoFit

End Sub
```

Como criar arquivos no formato .CSV:

Conforme você aprendeu no curso de Excel Avançado, é possível exportar os dados de uma planilha do Excel para o formato de texto conhecido como CSV – Comma Separated Values. Esta exportação pode ser feita usando código VBA, conforme mostrarei no exemplo deste tópico.

Uma das técnicas mais utilizadas é fazer uma cópia da planilha a ser exportada e depois salvar a cópia no formato .CSV:

```
Sub CreateCSVFile()  
  
    ' Desabilita a emissão de mensagens de alerta  
    Application.DisplayAlerts = False  
  
    ' Faz uma cópia da planilha atual  
    ActiveSheet.Copy  
  
    ' Salva a planilha como um arquivo .CSV  
  
    ActiveWorkbook.SaveAs Filename:="C:\Dados\Vendas.CSV", _  
        FileFormat:=xlCSV, CreateBackup:=False  
    ActiveWorkbook.Close False  
End Sub
```

Ao invés de exportar toda a planilha para um arquivo .CSV, você pode querer exportar apenas os dados de uma determinada faixa de células. O exemplo a seguir faz justamente isso, ou seja, exporta apenas uma faixa de células para um arquivo .CSV:

```
Sub Criar_Arquivo_CSV()  
  
    Dim F As Long, fName  
    Dim J As Long, I As Long  
    Dim rng As Range, outputLine As String  
    Dim entrySeparator As String  
    Dim fCol As Long, lCol As Long, fRow As Long, lRow As Long  
    Dim nResponse As Integer  
  
    ' Define o separador a ser utilizado, o qual normalmente é uma vírgula  
    entrySeparator = ","  
  
    ' Uso a função FreeFile para obter o próximo número de arquivo  
    ' disponível para uso  
    F = FreeFile(0)  
  
    ' Defino as variáveis associadas a seleção atual, a qual  
    ' representa a faixa de células a serem exportadas  
    Set rng = Selection  
    fCol = rng.Columns(1).Column  
    lCol = rng.Columns(rng.Columns.Count).Column  
    fRow = rng.Rows(1).Row  
    lRow = rng.Rows(rng.Rows.Count).Row  
  
    ' Obtém um nome de arquivo para ser utilizado.  
  
    fName = InputBox("Informe o nome do arquivo e o caminho completo " & _
```

"A seleção atual será salva como um arquivo .CSV.", _

"Informe o arquivo de saída (ex: C:\Dados\Vendas.csv):")

' Verifica se o usuário clicou em Cancelar

If fName = False Then

 MsgBox "Exportação Cancelada!!!"

End

End If

'Verifica se o arquivo já existe

If Dir(fName) <> "" Then

 nResponse = MsgBox(fName & " Já Existe " & _

 "Clique em OK para substituí-lo", Buttons:=vbOKCancel)

' Verifica se o usuário clicou em Cancelar

 If nResponse <> vbOK Then

 MsgBox "Exportação Cancelada"

 End

End If

End If

' Abre o arquivo para gravar os dados

Open fName For Output As #F

' Percorre as linhas da seleção atual.

For I = fRow To lRow

 outputLine = ""

 For J = fCol To lCol

 If J <> lCol Then

 ' Se não for a última coluna, adiciona o separador

 outputLine = outputLine & Cells(I, J).Value & entrySeparator

 Else

 ' Na última coluna não adiciona o separador

 outputLine = outputLine & Cells(I, J).Value

 End If

 Next J

 ' Salva a linha toda no arquivo

Print #F, outputLine

Next I

'Fecha o arquivo

Close #F

MsgBox "Dados gravados em: " & fName

End Sub

Lição 16: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Como ler um arquivo de texto, linha a linha:

No exemplo a seguir, é mostrado um código que faz a leitura de um arquivo de texto, linha a linha:

Sub LêArquivoTexto()

```
Dim I As Long, myString
' Reseta arquivos que tenham sido abertos anteriormente
Reset

' Abre o arquivo a ser processado
Open "C:\TESTFILE.TXT" For Input As #1
i = 1

' Faz um loop através das linhas dos arquivos

Do While Not EOF(1)
    ' Armazena os dados da linha em uma variável
    Input #1, myString
    ' Salva a saída em uma célula da planilha

    ActiveSheet.Cells(i, 1) = myString
    i = i + 1
Loop

' Fecha o arquivo
Close #1
End Sub
```

Exemplos para trabalho com planilhas – objeto Worksheet:

Como adicionar novas planilhas, via código VBA:

Para adicionar uma nova planilha você utiliza o método Add, da coleção Sheets, conforme exemplos a seguir:

```
Sheets.Add
```

ou, para adicionar a nova planilha, logo após a planilha ativa, use o comando a seguir:

```
Sheets.Add After:=ActiveSheet
```

No exemplo a seguir, utilizo Sheets.Count, para detectar o número de planilhas e adicionar a nova planilha como última planilha da pasta de trabalho atual:

```
Sheets.Add.Move After:=Sheets(Sheets.Count)
```

Como adicionar e renomear uma planilha, ao mesmo tempo:

No exemplo a seguir, uma nova planilha é adicionada, após a planilha chamada Vendas e a nova planilha é nomeada como Totais:

```
Worksheets.Add(After:=Worksheets("Vendas")).Name = "Totais"
```

Como copiar uma planilha como última planilha da pasta de trabalho:

O exemplo a seguir mostra como fazer uma cópia de uma planilha e, ao mesmo tempo, movê-la como última planilha da pasta de trabalho:

```
Sheets("Plan1").Copy After:=Sheets(Sheets.Count)
```

Se você deseja copiar a planilha após uma planilha específica, basta substituir Sheets.Count pelo nome da planilha, após a qual deve ser colocada a cópia que está sendo criada.

```
Sheets("Plan1").Copy After:=Sheets(Plan6)
```

Como copiar uma planilha para uma nova pasta de trabalho:

O exemplo a seguir mostra como copiar uma planilha dentro da mesma pasta de trabalho e depois renomear a planilha recém criada:

```
WorkSheets("Plan1").Copy after:=WorkSheets("Plan9")  
ActiveSheet.Name = "Plan10"
```

O exemplo a seguir mostra como copiar uma planilha para uma nova pasta de trabalho e como ativar esta pasta de trabalho:

```
Dim newBook As Workbook  
Set newBook = Sheets("Plan1").Copy  
newBook.Activate
```

Como excluir planilhas:

A maneira mais indicada é excluir as planilhas, usando o nome da planilha, para evitar enganos, como no exemplo a seguir:

```
' Desativa o tratamento de erros, para evitar que sejam emitidas  
' mensagens, caso a planilha a ser excluída não exista.  
  
On Error Resume Next  
Application.DisplayAlerts = False  
  
Sheets("Vendas").Delete
```

```
Sheets("Produtos").Delete  
Sheets("Clientes").Delete  
  
On Error GoTo 0  
Application.DisplayAlerts = True
```

Se ao invés do nome, você utilizar o índice da planilha, dentro da coleção Sheets, você deve fazer a exclusão a partir da última planilha, em direção a primeira. Se você começar do início para o final, será gerado um erro, pois o Excel fará referência a planilhas que não existem mais. Isso acontece porque o número de planilhas é determinado no início do código, mas a medida que você exclui cada planilha, este número vai sendo alterado.

Um erro muito comum é utilizar o código a seguir, para excluir todas as planilhas vazias. Este código irá gerar um erro em tempo de execução, pelos motivos expostos no parágrafo anterior:

```
Sub Exclui_Planilhas_Vazias_Jeito Errado( )  
    Dim I, J  
    I = Worksheets.Count  
    For J = 1 To I  
        If Application.CountA(Worksheets(J).Cells) = 0 Then  
            Worksheets(J).Delete  
        End If  
    Next J  
End Sub
```

A maneira correta, pelos motivos expostos anteriormente, está indicada no código a seguir:

```
Sub Exclui_Planilhas_Vazias_Jeito_Certo( )  
    Dim I, J  
    I = Worksheets.Count  
    For J = I To 1 Step -1  
        If Application.CountA(Worksheets(J).Cells) = 0 Then  
            Worksheets(J).Delete  
        End If  
    Next J  
End Sub
```

Nota: A função CountA é utilizada para determinar quantas células contém valores, na planilha. Se o número for zero, significa que a planilha está vazia. CountA é a função Cont.Valores, em Português.

Lição 17: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Como excluir planilhas, sem confirmação:

No exemplo a seguir, mostro como excluir uma planilha, evitando que seja exibida uma mensagem de aviso:

```
Application.DisplayAlerts = False
Worksheets("Plan1").Delete
Application.DisplayAlerts = True
```

Neste exemplo, utilizei a propriedade DisplayAlerts, definindo o seu valor em False, antes da exclusão da planilha e voltando a definir o seu valor em True, após a exclusão da planilha.

Como verificar se uma planilha existe em uma pasta de trabalho:

A função ExistePlanilha, indicada a seguir, retorna True se a planilha passada no parâmetro WSName existir, na pasta de trabalho passada no parâmetro WBName. Se a planilha não existir, a função irá retornar False.

```
Function ExistePlanilha (WBName As String, WSName As String) As Boolean

    On Error GoTo EndMacro

    If Workbooks(WBName).Worksheets(WSName).Name <> "" Then
        WorksheetExists = True
        Exit Function
    End If

EndMacro:
    WorksheetExists = False
End Function
```

A seguir um exemplo de utilização da função ExistePlanilha:

```
If ExistePlanilha("Vendas.xls", "Clientes") Then
    ' Código a ser executado se a planilha existe.
Else
    ' Código a ser executado se a planilha não existe.
End If
```

A seguir uma versão simplificada da função ExistePlanilha. Vamos chamala de ExistePlanilha2:

```
Function ExistePlanilha2 (WBName As String, WSName As String) As Boolean
    On Error Resume Next
    bWorksheetExists =(Workbooks(WBName).Worksheets(WSName).Name = WSName)
End Function
```

Como determinar se uma planilha está vazia:

Para determinar se uma planilha está vazia, utilizamos a função CountA (Cont.Valores), como no exemplo a seguir:

```
Application.CountA(Sheets("Plan1").Cells)
```

Se esta expressão retornar zero, significa que a planilha está vazia, ou seja, todas as células em branco.

Você também pode determinar o número de gráficos que foram inseridos em uma planilha, usando o código a seguir:

```
ActiveSheet.ChartObjects.Count
```

A seguir um exemplo de código que usa um teste IF e a função CountA, para determinar se uma planilha está ou não vazia e tomar diferentes ações, com base na planilha estar ou não vazia:

The following determines if a worksheet is empty:

```
If Application.CountA(ActiveSheet.UsedRange) = 0 Then  
    ' Comandos se a planilha estiver vazia  
Else  
    ' Comandos se a planilha NÃO estiver vazia  
End If
```

Como excluir o conteúdo de todas as células de uma planilha, durante a abertura:

A macro a seguir é executada quando uma planilha é aberta manualmente, usando o Excel. A macro irá excluir todos os dados da planilha:

```
Sub Auto_Open  
    Worksheets("sheet1").UsedRange.Clear  
End Sub
```

Diferentes maneiras de percorrer a coleção Worksheets:

Uma coleção Worksheets é formada por objetos do tipo Worksheet. Um objeto Sheet é um membro da coleção Sheets. A diferença é que um objeto Worksheet pode conter apenas planilhas, já um objeto como Sheet pode conter, por exemplo, uma folha de gráficos. A coleção Worksheet é um subconjunto da coleção Sheets. Outros subconjuntos da coleção Sheets são as coleções Charts, Dialogsheets e Modules.

Os exemplos a seguir mostram como usar a estrutura For...Each...Next, para percorrer as diferentes coleções disponíveis.

O código a seguir mostra como percorrer a coleção Worksheets da pasta de trabalho atual:

```
Dim ws As Worksheet
```

```
For Each ws In Worksheets
    ' Comandos a serem executados para cada planilha
    ' Por exemplo, exibo o nome da planilha.
    MsgBox ws.Name
Next
```

No exemplo a seguir, crio um laço para imprimir todas as planilhas da pasta de trabalho Vendas.xls:

```
Dim sh As Sheet
For Each sh In Workbooks("C:\Vendas.xls")
    sh.Printout
Next sh
```

Este exemplo irá imprimir todos os tipos de planilhas existentes em Vendas.xls, inclusive os módulos e as Caixas de Diálogo. Você pode adaptar o exemplo para que sejam impressos somente os objetos do tipo Worksheet e Chart. Para isso utilizamos a função TypeName, para retornar o tipo do objeto Sheet que está sendo analisado a cada passagem do laço.

```
Dim sh As Sheet

For Each sh In Workbooks("C:\Vendas.xls")
    If LCase(TypeName(sh)) = "worksheet" Or LCase(TypeName(sh)) = "chart" Then
        sh.Printout
    End If
Next sh
```

Como ordenar as planilhas pelo nome:

A macro a seguir ordena as planilhas da pasta de trabalho atual, pelo nome:

```
Sub OrdenaPlanilhasPeloNome( )
    Dim numberOfSheets As Integer
    Dim sheetPosition As Integer
    Dim I As Integer
    numberOfSheets = ActiveWorkbook.Worksheets.Count
    sheetPosition = numberOfSheets
    Do
        If sheetPosition = 1 Then Exit Do

        For I = 1 To sheetPosition - 1
            If Sheets(I).Name > Sheets(I + 1).Name Then
                Sheets(I + 1).Move before:=Sheets(I)
            End If
        Next I

        sheetPosition = sheetPosition - 1
    Loop
End Sub
```

Lição 18: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Como criar uma lista com o nome das planilhas da pasta de trabalho atual:

A função CriaListaDePlanilhas, indicada a seguir, cria uma lista com o nome de todas as planilhas de uma pasta de trabalho e grava esta lista em uma planilha em uma nova pasta de trabalho:

```
Sub CriaListaDePlanilhas ( )
    Dim originalSetting As Integer
    Dim wb As Workbook
    Dim oS As Object
    Dim I As Integer

    Set wb = ActiveWorkbook

    ' Cria uma nova pasta de trabalho com uma única planilha
    ' A nova pasta de trabalho torna-se a pasta ativa.

    originalSetting = Application.SheetsInNewWorkbook

    Application.SheetsInNewWorkbook = 1
    Workbooks.Add
    Application.SheetsInNewWorkbook = originalSetting

    ' Faz um loop através das planilhas da
    ' pasta de trabalho original

    For Each oS In wb.Sheets

        I = I + 1
        ' Grava o nome de cada planilha, iniciando na célula A1
        Cells(I, 1).Value = oS.Name

    Next
End Sub
```

Como proteger e desproteger planilhas, usando código VBA:

A seguir alguns exemplos de proteção e “desproteção” de uma planilha. Observe que é definida uma senha de proteção, a qual tem que ser informada quando a planilha for desprotegida:

```
Sub ProtegePlanilha( )
    ActiveSheet.Protect password:="senhaabc"
End Sub

Sub DesprotegePlanilha( )
    ActiveSheet.Unprotect password:="senhaabc"
End Sub
```

Para detectar se uma planilha está ou não protegida, você pode verificar o valor da propriedade ProtectContents da planilha, conforme exemplo a seguir:

```
If Sheets("Plan1").ProtectContents Then
    MsgBox "A Planilha Está Protegida!!"
Else
    MsgBox "A Planilha NÃO Está Protegida!!"
End If
```

Como inserir a data atual em todas as planilhas de uma pasta de trabalho:

O exemplo de código a seguir insere a data atual, na célula A1, de todas as planilhas da pasta de trabalho atual. Este exemplo pode ser facilmente adaptado, sempre que você precisa modificar uma mesma célula, em todas as planilhas de uma pasta de trabalho:

```
Sub DefineDataAtual( )
    Dim mydate As Date
    Dim a As Integer
    mydate = Date

    For a = 1 To ActiveWorkbook.Worksheets.Count
        Worksheets(a).Range("A1").Value = mydate
    Next a
End Sub
```

A seguir uma outra abordagem, porém com o mesmo resultado:

```
Sub DefineDataAtual( )
    Dim ws As Worksheet
    Dim myDate As Date
    myDate = Date
    For Each ws In Worksheets
        ws.Range("A1").Value = myDate
    Next ws
End Sub
```

Importante: Observe que em nenhum dos exemplos, foi necessário ativar cada planilha, antes de definir o valor de uma célula da planilha. Isso porque simplesmente não é necessário ativar uma planilha, para definir o valor de uma ou mais de suas células, via código VBA. A ativação de cada planilha, somente deixaria a execução do código mais lenta.

Como tornar todas as planilhas visíveis:

O exemplo a seguir, pode ser utilizado para tornar todas as planilhas de uma pasta de trabalho, visíveis:

```
Sub MakeAllVisible()
    Dim oSheet As Object
    For Each oSheet In Sheets
        oSheet.Visible = True
    Next
End Sub
```

Lição 19: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Como evitar que os usuários possam adicionar novas planilhas:

O código do exemplo a seguir, pode ser colocado no módulo de código da pasta de trabalho, para evitar que os usuários possam adicionar novas planilhas:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    Application.DisplayAlerts = False
    MsgBox "Não podem ser adicionadas novas planilhas " & _
        "A planilha será excluída!!!"
    Sh.Delete
    Application.DisplayAlerts = True
End Sub
```

Exemplos de funções para trabalho com texto e números:

Como extrair valores de texto separados por uma Barra / :

Por exemplo, vamos imaginar que, em uma coluna da planilha, você tenha importado texto onde os campos estão separados por uma coluna, como no exemplo a seguir:

0012356/José da Silva/São Paulo/1234456567

O objetivo é extrair cada parte do texto ou númeos e colocar cada valor em uma célula de uma coluna separada.

A rotina a seguir pode ser utilizada para executar este trabalho. Você deve criá-la como uma macro e selecionar a faixa de dados, antes de executar a macro. Células em branco serão ignoradas. Foram criadas duas rotinas. Uma chamada ExtrairValores que é quem faz o trabalho de separar os valores e outra chamada ProcessaCélulas, que faz um loop através da seleção atual e chama a rotina ExtrairValores, para separar os valores de cada célula, da seleção atual.

```
Sub ProcessaCélulas( )
    Dim cell As Range
    For Each cell In Selection
        If Not IsEmpty(cell) Then
            ExtrairValores cell
        End If
    Next
End Sub
```

A seguir a rotina ExtrairValores, que é quem realmente faz o trabalho de separação:

```
Sub ExtrairValores (anyCell As Range)
    Dim s As String
    Dim N As Integer, I As Integer

    ' Atribui o valor da célula à variável s
    s = anyCell.Value
    ' Localiza a primeira ocorrência da barra /
    N = InStr(s, "/")

    ' Se a barra for encontrada, processa a string, até que nenhuma barra
    ' mais seja encontrada.

    While N > 0
        ' Indexa o contador utilizado como Offset
        I = I + 1
        ' Grava o valor até antes da primeira barra, na célula
        anyCell.Offset(0, I).Value = Left(s, N - 1)

        ' Remove a primeira parte e a primeira barra.
        s = Mid(s, N + 1)

        ' Pesquisa a próxima ocorrência da barra /
        N = InStr(s, "/")
    Wend

    ' Atualiza novamente a variável de Offset
    I = I + 1

    ' Grava o valor obtido na célula
    anyCell.Offset(0, I).Value = s

End Sub
```

Dica: Se você tiver alguma dificuldade em entender a lógica da função ExtrairValores, basta executá-la, passo-a-passo, usando a tecla F8 e a janela de inspeção de variáveis, conforme descrito a partir da Lição 23 do Módulo 2.

Determinando se um número é Par ou Ímpar:

Para determinar se um número é par ou ímpar, você pode utilizar o trecho de código a seguir, onde o valor do número a ser analisado, está contido na variável x:

```
If x mod 2 = 0 Then
    MsgBox "O Número é Par!!!"
Else
    MsgBox "O Número é Ímpar!!!"
End If
```


Lição 20: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Usando a função Chr, para inserir caracteres especiais:

Você pode utilizar a função Chr(), para inserir caracteres especiais em uma mensagem de um MsgBox, como por exemplo uma quebra de linha. A seguir alguns exemplos que ilustram a utilização da função MsgBox:

' O exemplo a seguir exibe a mensagem entre aspas – Chr(34):

```
MsgBox "Esta não está entre aspas. " & Chr(34) & "Esta está entre aspas" & Chr(34)
```

Este comando produz o resultado indicado a seguir:



' Para inserir uma quebra de linha em um MsgBox, use Chr(13)

```
MsgBox "O nome do arquivo é: " & Chr(13) & ActiveWorkbook.Name
```

Este comando produz o resultado indicado a seguir:



' No exemplo a seguir, uso Chr(13) duas vezes, para
' produzir uma linha em branco e Chr(9), para produzir
' uma tabulação

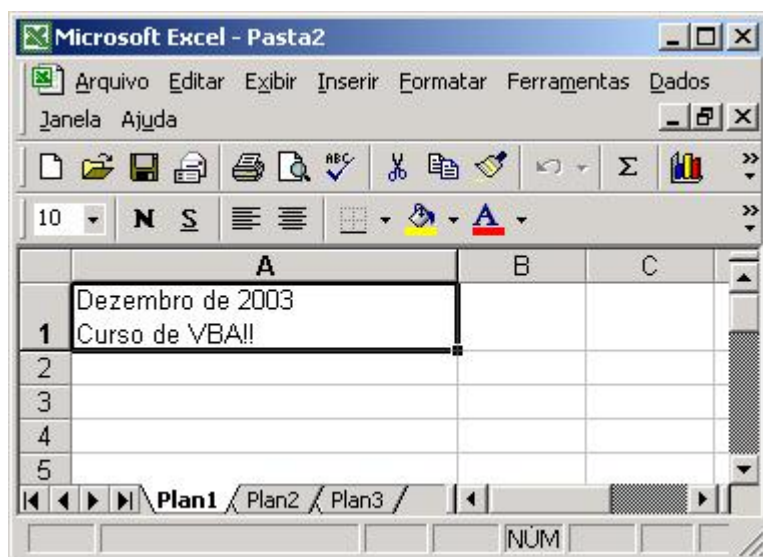
```
MsgBox "O arquivo é: " & Chr(13) & Chr(13) & Chr(9) & ActiveWorkbook.Name
```

Este comando produz o resultado indicado a seguir:



' No exemplo a seguir, utilizo Chr(10), para simular um Alt+Enter

```
ActiveCell.Value = "Dezembro de 2003" & Chr(10) & "Curso de VBA!!"
```

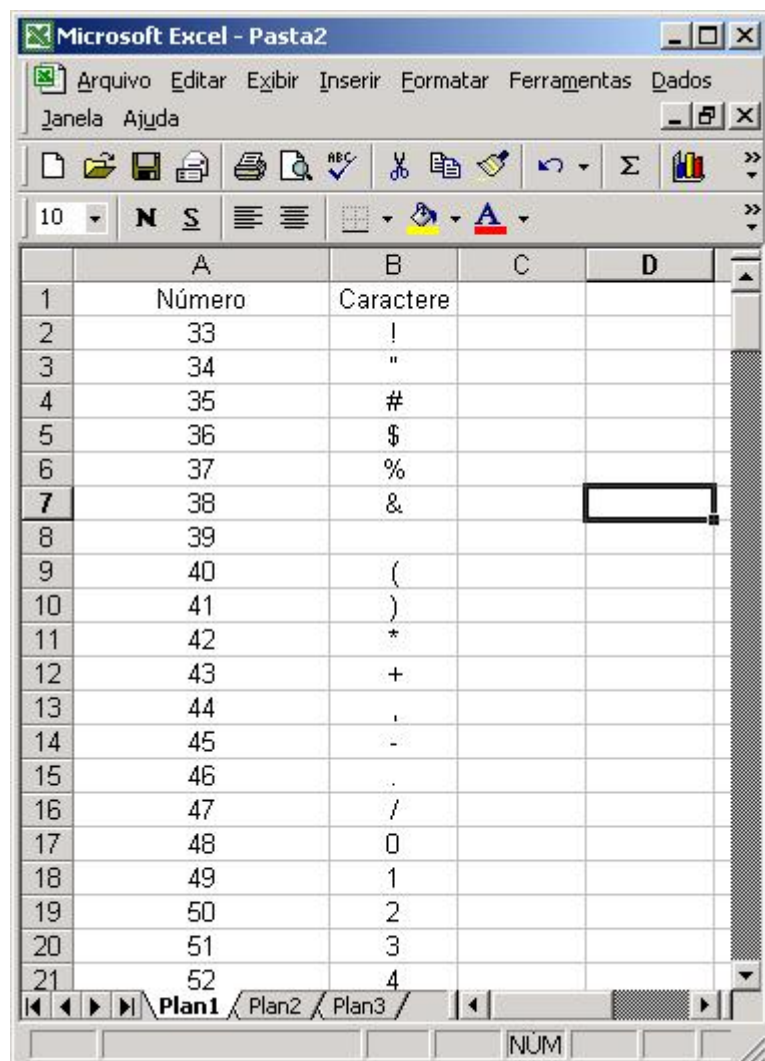


Neste caso tem que ser um Alt+Enter (Chr(10)), pois se for um Enter haverá uma troca de célula, ao invés de uma quebra de linha, dentro da célula.

A função a seguir usa a função Chr para gerar os caracteres de 33 a 255. Os caracteres de 1 a 32 são caracteres especiais, não imprimíveis.

```
Sub MostraTodosOsCaracteres( )  
    Dim I As Integer  
    Worksheets(1).Cells(1, 1).Value = "Número"  
    Worksheets(1).Cells(1, 2).Value = "Caractere"  
  
    For I = 33 To 255  
        Worksheets(1).Cells(I - 30, 1).Value = I  
        Worksheets(1).Cells(I - 30, 2).Value = Chr(I)  
    Next  
    ' Centraliza o conteúdo das colunas  
    Worksheets(1).Columns("a:b").HorizontalAlignment = xlCenter  
End Sub
```

A figura a seguir, mostra parte do resultado gerado por esta função:



Lição 21: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais exemplos práticos, os quais podem ser facilmente adaptados para uso nas planilhas com as quais você trabalha.

Como extrair os números iniciais de uma String:

Por exemplo, vamos supor que você tenha dados que iniciam com números e depois contenham caracteres de texto, como por exemplo:

- ▶ 1234Ab34
- ▶ 45689rfv5
- ▶ 67892ghy5

A função a seguir pode ser utilizada para extrair apenas o número, antes da ocorrência do primeiro caractere de texto:

```
Public Function RetiraNúmero(ByVal anyS As String) As Double
    anyS = UCase(anyS)

    If InStr(anyS, "D") > 0 Then
        anyS = Application.Substitute(anyS, "D", "Z")
    ElseIf InStr(anyS, "E") > 0 Then
        anyS = Application.Substitute(anyS, "E", "Z")
    End If
    ValString = Val(anyS)
End Function
```

Como obter o número de caracteres em uma seleção:

A função a seguir retorna o número de caracteres em uma seleção de células. Os espaços em branco também são contabilizados.

```
Sub ContaCaracteres( )
    Dim cell As Range
    Dim I As Integer
    For Each cell In Selection
        I = I + Len(cell.Value)
    Next
    MsgBox "Existem: " & I & " Caracteres na seleção atual."
End Sub
```

Como testar se uma string faz parte do valor de uma célula ou variável:

Para testar se uma função ou variável contém um determinada String, você pode utilizar a função InStr. Esta função retorna a posição inicial, onde a string pesquisada inicia. A seguir mais alguns detalhes sobre a função InStr.

A Função InStr:

Esta função retorna uma Variant (Long) que especifica a posição da primeira ocorrência de uma sequência de caracteres dentro de outra.

Sintaxe:

```
InStr([start, ]string1, string2[, compare])
```

A sintaxe da função InStr tem os seguintes argumentos:

Parte	Descrição
start	Opcional. Expressão numérica que define a posição inicial de cada pesquisa. Se omitido, a pesquisa iniciará na posição do primeiro caractere. Se start contiver Null, ocorrerá um erro. O argumento start será necessário, se compare for especificado.
string1	Obrigatória. Expressão de sequência sendo pesquisada.
string2	Obrigatória. Expressão de sequência de caracteres procurada.
compare	Opcional. Especifica o tipo de comparação de sequência de caracteres. Se compare for Null, ocorrerá um erro. Se compare for omitido, a configuração Option Compare determinará o tipo de comparação. Especifique um LCID (LocaleID) válido para usar regras específicas da localidade na comparação.

As configurações do argumento compare são as seguintes:

Constante	Valor	Descrição
vbUseCompareOption	-1	Executa uma comparação usando a configuração da instrução Option Compare.
vbBinaryCompare	0	Executa uma comparação binária.
vbTextCompare	1	Executa uma comparação textual.
vbDatabaseCompare	2	Somente Microsoft Access. Efetua uma comparação, com base nas informações existentes no banco de dados.

Os valores retornados pela função InStr são os seguintes:

Se	A InStr retornará
string1 tiver comprimento zero	0
string1 for Null	Null
string2 tiver comprimento zero	start
string2 for Null	Null

string2 não for encontrado	0
string2 for encontrado dentro de string1	A posição em que a correspondência foi encontrada
start > string2	0

Exemplos da função InStr:

Este exemplo usa a função InStr para retornar a posição da primeira ocorrência de uma sequência de caracteres dentro de outra.

```
Dim SearchString, SearchChar, MyPos
SearchString = "XXpXXpXXPXXP" ' Sequência de caracteres a ser pesquisada.
SearchChar = "P" ' Procurar por "P".
```

```
' Uma comparação de texto iniciando na posição 4. Retorna 6.
MyPos = Instr(4, SearchString, SearchChar, 1)
```

```
' Uma comparação binária iniciando na posição 1. Retorna 9.
MyPos = Instr(1, SearchString, SearchChar, 0)
```

```
' A comparação é binária, como padrão (o último argumento é omitido).
MyPos = Instr(SearchString, SearchChar) ' Retorna 9.

MyPos = Instr(1, SearchString, "W") ' Retorna 0.
```

Você também pode fazer uma pesquisa que desconsidera maiúsculas ou minúsculas, definindo o primeiro e o último argumento da função com o valor 1.

Mais alguns exemplos:

```
Dim myString As String
myString = Ucase(Range("A1").Value)
If Instr(myString, "ABC") > 0 Then
    ' Comandos a serem executados, se a String ABC
    ' for encontrada em A1
End If
```

Outra maneira de efetuar comparações que não diferenciam maiúsculas de minúsculas é colocar a declaração a seguir, na seção de declarações do módulo:

```
Option Compare Text
```

Lição 22: Exemplos práticos para o seu dia-a-dia

Nesta e nas demais lições deste módulo, apresentarei alguns exemplos de macros úteis. Os exemplos foram retirados da Internet. Em caso de dúvidas sobre alguns dos comandos e da lógica das Macros, entre em contato pelo email: webmaster@juliobattisti.com.br.

Uma macro para localizar entradas que não estão em uma lista de valores:

A macro do exemplo a seguir, compara duas listas de valores e coloca em destaque os valores da segunda seleção, que estão presentes na segunda seleção e não estão presentes na primeira seleção.

```
Sub BuscaNãoCoincidentes( )
    Dim MasterList As Range
    Dim ItemsToMatch As Range
    Dim cell As Range
    Dim FoundRow As Long
    On Error Resume Next

    ' Pede para que o usuário informe as faixas a serem comparadas.
    Set MasterList = Application.InputBox( prompt:="Selecione a primeira faixa", _
        Type:=8)

    ' Se não for selecionada uma faixa, encerra a macro

    If MasterList Is Nothing Then End

    ' Pede para o usuário informar a segunda faixa

    Set ItemsToMatch = Application.InputBox( prompt:="Selecione a segunda faixa ", _
        Type:=8)

    ' Se não for selecionada uma faixa, encerra a macro
    If ItemsToMatch Is Nothing Then End

    ' Restringe as faixas para a faixa usada na planilha, caso alguma
    ' linha ou coluna inteira, tenha sido selecionada.

    Set MasterList = Intersect(MasterList, ActiveSheet.UsedRange)
    Set ItemsToMatch = Intersect(ItemsToMatch, ActiveSheet.UsedRange)

    ' Para cada uma das células, da segunda faixa, verifica se ela existe
    ' na primeira faixa

    For Each cell In ItemsToMatch
        ' Faz a verificação somente em células não em branco
```



```
If Application.Trim(cell) <> "" Then
    ' Reseta o código de erro para cada passagem do laço
    Err = 0
    FoundRow = Application.Match(cell.Value, MasterList, 0)

    ' Se encontrar uma ocorrência, mantém o código de erro em
    ' zero e altera a cor de fundo da célula.

    If Err = 0 Then
        cell.Interior.ColorIndex = 6
        cell.Interior .Pattern = xlSolid
    End If
End If

Next cell
End Sub
```

Lição 23: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais um exemplo de macro útil para o seu dia-a-dia. Os exemplos foram retirados da Internet. Em caso de dúvidas sobre alguns dos comandos e da lógica das Macros, entre em contato pelo email: webmaster@juliobattisti.com.br.

Excluindo linhas, com base nos valores de uma coluna:

O exemplo de código a seguir, irá excluir todas as linhas que contenham o valor 17 na coluna A:

```
Sub ExcluiLinhas( )
    Dim TestRange As Range
    Dim lastRow As Long, firstRow As Long
    Dim i As Long

    ' Restringe a faixa a ser testada apenas as células da coluna A
    ' da faixa utilizada na planilha ativa.

    Set TestRange = Intersect(Range("A:A"), ActiveSheet.UsedRange)

    ' Obtém o número da última linha na faixa atual.
    lastRow = TestRange.Cells(TestRange.Cells.Count).Row

    ' Obtém o número da primeira linha na faixa atual.
    firstRow = TestRange.Cells(1).Row

    ' Faz um loop começando da última linha em direção à primeira
    ' Esta inversão na ordem – da última para primeira, é necessária
    ' pois linhas poderão ser excluídas.

    For i = lastRow To firstRow Step -1

        If IsNumeric(Cells(i, 1)) Then
            If Cells(i, 1).Value = 17 Then
                Rows(i).Delete
            End If
        End If
    Next

End Sub
```

Se o teste que você deseja executar é baseado em um valor de teste, use o comando a seguir, ao invés do teste numérico do exemplo anterior. Por exemplo, suponhamos que você deseja excluir todas as linhas que tenham o texto SP na coluna A. Neste caso, você utilizaria o teste indicado a seguir:

```
If TypeName(Cells(i, 1).Value) = "String" Then
    ' Converte para maiúsculas e testa
    If Ucase(Cells(i, 1).Value) = "SP" Then
        Rows(i).Delete
    End If
End If
```

Lição 24: Exemplos práticos para o seu dia-a-dia

Nesta lição apresentarei mais um exemplo de macro útil para o seu dia-a-dia. Os exemplos foram retirados da Internet. Em caso de dúvidas sobre alguns dos comandos e da lógica das Macros, entre em contato pelo email: webmaster@juliobattisti.com.br.

Uma macro que gera números aleatórios:

O exemplo de macro a seguir solicita que o usuário informe o número de dígitos dos números a serem gerados, aleatoriamente, em uma faixa de células. A macro irá então gerar os números aleatoriamente ou então modificar as fórmulas, de tal maneira que os resultados sejam aleatórios.

```
Sub Gera_Aleatórios( )
    Dim digits As Variant
    Dim cell As Range

    ' Desabilita a atualização da tela
    Application.ScreenUpdating = False

    ' Solicita que o usuário digite um número

    digits = Application.InputBox( prompt:="Informe o número de dígitos: ", _
        Title:="Número de dígitos!!!", _
        Type:=1, default:=3)

    ' Verifica se o usuário clicou em Cancelar.

    If TypeName(digits) = "Boolean" Then Exit Sub

    ' Converte a String retornada por InputBos em um número
    digits = Val(digits)

    ' Faz um loop através das células da seleção atual

    For Each cell In Intersect(Selection, ActiveSheet.UsedRange)

        ' Verifica se a célula contém uma entrada numérica.

        If Application.IsNumber(cell.Value) Then
            If cell.HasFormula Then
                ' Se for fórmula, modifica para usar Round()
                cell.Formula = "=Round(" & _
                    Mid(cell.Formula, 2) _
                    & ", " & digits & ")"
            Else
                ' Se for número, gera um valor aleatório.
                cell.Value = Application.Round(cell.Value, digits)
            End If
        End If
    Next cell
End Sub
```

```
End If
    End If
Next cell
Application.ScreenUpdating = True
End Sub
```

Lição 25: Resumo do Módulo

Nas lições deste módulo você aprendeu a trabalhar com os diversos controles que podem ser inseridos em um UserForm. Apresentei exemplos práticos, passo-a-passo, de uso destes controles. Na segunda parte do módulo, apresentei uma série de exemplos de código VBA, para solucionar problemas práticos, do dia-a-dia com a programação do VBA. Estes exemplos podem, facilmente, ser adaptados para situações reais, para resolução de problemas que você encontra no uso da programação VBA no Excel.

Módulo 6 – Controles e Exemplos Práticos

- Lição 01:** UserForms – O controle Caixa de Listagem
- Lição 02:** UserForms – O controle Caixa de Seleção
- Lição 03:** UserForms – O controle Botão de Opção - OptionButton
- Lição 04:** UserForms – O controle Botão de ativação - ToggleButton
- Lição 05:** UserForms – Os controles Botão de Comando e Frame
- Lição 06:** UserForms – O controle Barra de Rolagem
- Lição 07:** UserForms – O controle Botão de Rotação
- Lição 08:** UserForms – O controle Image
- Lição 09:** Exemplos práticos para o seu dia-a-dia
- Lição 10:** Exemplos práticos para o seu dia-a-dia
- Lição 11:** Exemplos práticos para o seu dia-a-dia
- Lição 12:** Exemplos práticos para o seu dia-a-dia
- Lição 13:** Exemplos práticos para o seu dia-a-dia
- Lição 14:** Exemplos práticos para o seu dia-a-dia
- Lição 15:** Exemplos práticos para o seu dia-a-dia
- Lição 16:** Exemplos práticos para o seu dia-a-dia
- Lição 17:** Exemplos práticos para o seu dia-a-dia
- Lição 18:** Exemplos práticos para o seu dia-a-dia
- Lição 19:** Exemplos práticos para o seu dia-a-dia
- Lição 20:** Exemplos práticos para o seu dia-a-dia
- Lição 21:** Exemplos práticos para o seu dia-a-dia
- Lição 22:** Exemplos práticos para o seu dia-a-dia
- Lição 23:** Exemplos práticos para o seu dia-a-dia
- Lição 24:** Exemplos práticos para o seu dia-a-dia
- Lição 25:** Resumo do Módulo

Bibliografia recomendada:

Confira as dicas de livros de Excel no seguinte endereço:

<http://www.juliobattisti.com.br/indicados/excel.asp>