

INTRODUÇÃO AOS CONCEITOS BÁSICOS E INTERMEDIÁRIOS DE CRIAÇÃO DE FUNÇÕES PERSONALIZADAS NO EXCEL



Robert Friedrick Martim

Autor: [Robert F Martim](#)
Publicado: www.juliobattisti.com.br
Contato: rm@earnconsultoria.com.br

Criado em: 1/5/2005
Última edição: 3/6/2005

Conheça outros trabalhos do autor no site

EXCEL – Série Como Fazer

Criando menus, barras de comando e botões personalizados no Excel usando VBA

O autor descreve, de forma detalhada, como criar os diversos tipos de menus e barras de comandos disponíveis no Excel. Crie atalhos no teclado para acessar menu personalizado, FaceIDs personalizadas para os aplicativos, menus de atalho com o botão direito do mouse e muito mais. Uma referência que não pode faltar aos usuários que desejam criar aplicações profissionais com o Excel, definindo seus próprios menus e barras de comandos, personalizadas. **O módulo é acompanhado por 10 pastas de trabalho desenvolvidas com vários exemplos práticos, os quais ajudarão você a entender e a acompanhar os exemplos propostos no curso.**

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelvbamenu/default.asp>

Formulários no Excel Utilizando VBA: Listbox e Combobox

O autor descreve, de forma detalhada, como manipular caixas de manipulação e caixas de listagem. Neste curso o amigo leitor aprenderá, em detalhes, como:

1. Utilizar nomes dinâmicos para preencher caixas de listagem e caixas de combinação.
2. Fazer referência entre os dois controles e outros controles em um formulário, a partir de seleções em uma caixa de combinação ou caixa de listagem.
3. Classificar itens em ordem crescente/decrescente. O leitor aprenderá a lógica por trás da classificação utilizada no Excel. O leitor também aprenderá a criar funções para ordenar listas.
4. Adicionar itens únicos a partir de uma lista onde vários itens se repetem.
5. Referenciar itens que pertencem a uma lista.
6. O leitor aprenderá a lógica por trás de listas de itens únicos e como criar funções para retornar tais listas.
7. Passar itens entre caixas de listagem.
8. Mover itens dentro de uma caixa de listagem.
9. Conectar uma caixa de listagem ao Outlook e filtrar a lista de contatos.
10. Utilizar a lista de contatos filtrada para enviar e-mail utilizando um servidor SMTP virtual.
11. Conectar caixas de combinação a dados de uma tabela ou consulta do Access.
12. Filtrar, ler, escrever e apagar registros no Access, usando programação VBA no Excel.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelvbaforms/default.asp>

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Fórmulas no Excel: Manipulando Datas e Horas

- Dúvidas sobre como fazer cálculos com valores de horas no Excel?
- Dúvidas sobre como fazer cálculos com valores de datas no Excel?
- Procurando soluções práticas, fáceis de entender e adaptar para o seu uso?

Este módulo aborda a manipulação de datas e horas no Excel. Você irá entender exatamente como o Excel armazena valores de datas e horas, aprenderá a criar fórmulas e usar funções para cálculos com datas e horas. Além do arquivo .PDF, **você também receberá 13 planilhas, com os exemplos utilizados no curso.**

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelatashoras/default.asp>

BRINDE: Na compra do **CD-04** você ganha um vídeo curso com 22 minutos de duração, o qual mostra, passo a passo, como criar um controle do tipo calendário(Calendar Control) no VBA.

Fórmulas no Excel: Fazendo Milagres com Fórmulas Matriciais

Veja se você se encaixa em um dos itens a seguir:

- Dificuldades para fazer cálculos complexos no Excel??
- Chegou a um ponto em que acha que não tem solução para alguns cálculos no Excel??
- Procurando soluções práticas, fáceis de entender e adaptar para o seu uso??

O autor descreve, de forma detalhada, como funcionam e como manipular fórmulas matriciais no Excel. Este módulo trata sobre a manipulação e construção de fórmulas matriciais, do ponto de vista prático. Os exemplos visam desenvolver a compreensão do leitor sobre como tais funções funcionam como elas podem ser aplicadas na resolução de problemas práticos. Uma referência que não pode faltar ao leitor que deseja entender como funcionam os cálculos e manipulações complexas de dados no Excel. **O módulo é acompanhado por 9 pastas de trabalho (em um total de 22 planilhas)**, as quais apresentam inúmeros exemplos práticos.

Além disso, o leitor poderá adquirir um curso em vídeo, com mais de 2 horas de duração, o qual fornece uma série de outros exemplos.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelmatric/default.asp>

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Fórmulas no Excel: Fazendo Milagres com Fórmulas Matriciais (Vídeo Curso)

Este novo curso introduz a interatividade do vídeo para ensinar sobre fórmulas e funções matriciais no Excel e VBA. Você aprenderá através das explicações e visualização como as fórmulas são entradas e manipuladas. Aprenda através de aulas em vídeo, interativas, de fácil acompanhamento.

Este vídeo é a companhia perfeita para quem já comprou o curso "Fórmulas e Funções Matriciais no Excel", no formato .PDF. Embora ambos abranjam o mesmo assunto, aspectos diferentes, assim como exemplos diferentes, são tratados e mostrados no vídeo. Ou seja, no vídeo você terá diversos exemplos novos, não presentes no curso em .PDF.

Combinando o material escrito no curso .PDF com o vídeo o aluno terá uma referência completa sobre fórmulas matriciais no Excel... Além disso, o leitor descobrirá que nem sempre o Ajuda do Excel é de "grande ajuda". Principalmente, quando ele informa a maneira incorreta para se manipular matrizes constantes.

IMPORTANTE: Este curso está disponível somente para envio em CD, pelos Correios. Devido ao tamanho dos arquivos de vídeo - quase 400 MB, fica inviável disponibilizá-los via download ou via e-mail.

Para você que gosta de cursos interativos, com vídeo e som, esta sem dúvidas é uma excelente opção. Este é apenas o primeiro de uma série de cursos que serão disponibilizados no formato de vídeo-aulas.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/videocursos/excelmatric/default.asp>

Fórmulas no Excel: Funções de Procura e Referência

Veja se você se encaixa em um dos itens a seguir:

- Dificuldades para usar as funções PROCV e PROCH??
- Dificuldades para utilizar as funções de pesquisa do Excel??
- Procurando soluções práticas, fáceis de entender e adaptar para o seu uso??

Uma referência completa que não pode faltar a todos os usuários que têm a necessidade de trabalhar com fórmulas e funções de pesquisa no Excel.

Neste novo curso o autor descreve, de forma detalhada, como funcionam e como utilizar as funções e fórmulas para pesquisa e validação de dados no Excel. Este módulo trata sobre a manipulação e criação de fórmulas de procura e referência do ponto de vista prático. **CHEGA DE DÚVIDAS SOBRE COMO USAR AS FUNÇÕES PROCV E PROCH, DENTRE OUTRAS.**

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelproc/default.asp>

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Guia avançado do Registro do Excel

O autor descreve de forma detalhada, como manipular o registro do Excel. Este módulo mostra como modificar a Registry do Excel (Windows) de forma a personalizar dezenas de aspectos do Excel, tais como: número de ações que podem ser desfeitas, modelos de gráfico padrão, gerenciamento de abertura de arquivos, segurança de macros, segurança de Internet, etc.

O autor utiliza uma linguagem extremamente didática, de fácil compreensão. O curso é todo baseado em exemplos práticos, detalhadamente explicados.

DEZENAS DE EXEMPLOS PRÁTICOS E ÚTEIS DE CONFIGURAÇÕES DO EXCEL, USANDO A REGISTRY:

- **ENTENDA COMO FUNCIONA A REGISTRY**
- **SAIBA COMO CONFIGURAR O EXCEL USANDO A REGISTRY**
- **APRENDA A MODIFICAR DEZENAS DE OPÇÕES DO EXCEL**
- **CONFIGURAÇÕES NÃO-DOCUMENTADAS NA AJUDA DO EXCEL**
- **APRENDA A ALTERAR O TIPO DE GRÁFICO PADRÃO**
- **ALTERE AS OPÇÕES PADRÃO DE FONTE**
- **APRENDA A REMOVER LINHAS DE GRADE**
- **APRENDA A MODIFICAR AS CONFIGURAÇÕES DE AUTO-RECUPERAÇÃO**
- **DEZENAS DE OUTRAS CONFIGURAÇÕES**

Você não vai acreditar nas configurações que podem ser feitas no Excel usando a Registry do Windows.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelregistry/default.asp>

Autor: Robert F Martim

Criado em: 1/5/2005

Última edição: 3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

WORD – Série Como Fazer

Criando menus, barras de comando e botões personalizados no MS Word usando VBA

Veja se você se encaixa em um dos itens a seguir:

- Dificuldades em entender como funcionam os menus e barras de ferramentas no Word??
- Não sabe como personalizar os menus e barras de ferramentas do Word??
- Não sabe como criar novos menus e novas barras de ferramentas no Word?

Então, sem dúvidas, este curso foi feito sob encomenda para você. Chega de dúvidas, é hora de dominar os MENUS e BARRAS DE FERRAMENTAS no Word.

O autor descreve, de forma detalhada, como manipular barras de comandos, menus e botões de comandos no Word. Este curso trata sobre a manipulação destas ferramentas no Word e como elas podem ser utilizadas para personalizar o desenvolvimento de aplicativos no Word. Este curso segue a linha do curso equivalente no Excel: Criando Menus Personalizados no Excel, porém, a abordagem, conteúdo e exemplos são totalmente diferentes, tornando o curso no Word e no Excel complementares, quando o assunto é desenvolvimento no Office. Se você já possui o curso sobre menus no Excel, este curso de menus no Word é o complemento ideal para uma referência ainda mais completa sobre como manipular estes objetos entre os aplicativos Office.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/wordmenus/default.asp>

ACCESS – Série Como Fazer

Criando menus, barras de comando e botões personalizados no MS Access usando VBA

O autor descreve, de forma detalhada, como criar os diversos tipos de menus e barras de comandos disponíveis no Microsoft Access. Este módulo, visa a criação de atalhos no teclado para acessar menu personalizado, FaceIDs personalizadas para os aplicativos, menus de atalho com o botão direito do mouse e muito mais. Uma referência que não pode faltar às pessoas que estão sérias quando o assunto é menu personalizado no MS Access. **O módulo é acompanhado por 14 bancos de dados e uma pasta de trabalho do Excel como recurso externo.**

- O que é um menu
- O que é uma barra de comando
- Como menus e barras de comando são criadas no Access
- Como remover os menus personalizados
- Removendo menus manualmente
- Removendo menus via código
- Posicionando os objetos
- Executando ações e atalhos de teclado
- Utilizando os FaceIDs do Office no Access
- Faces personalizadas
- Criando “popups” de atalho
- Adicionando um controle a um menu de atalho popup já existente
- Menus “Combobox”
- Automatizando a criação de menus
- Substituindo menu principal do Access pelo seu menu personalizado
- Listando os menus e sub-menus do Access
- Listando somente as barras de comando/ferramentas em um arquivo texto
- Listando as barras de comando/ferramentas e sub-menus no Access

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/accessmenus/default.asp>

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

MATEMÁTICA E ESTATÍSTICA – Série Como Fazer

Introdução a Matemática Financeira

Este curso é um curso teórico sobre Matemática Financeira. O curso apresenta desde uma revisão dos elementos básicos da Matemática, passando pelos elementos básicos da Matemática Financeira, tais como: juros simples, juros compostos, valor presente, valor futuro, fluxo de caixa, capitalização, etc.

O autor utiliza uma linguagem extremamente didática, de fácil compreensão. O curso é todo baseado em exemplos práticos, detalhadamente explicados.

MAIS DE 250 EXERCÍCIOS RESOLVIDOS, OS QUAIS COBREM DIVERSAS SITUAÇÕES PRÁTICAS ENCONTRADAS NO SEU DIA-A-DIA, TAIS COMO:

- **CÁLCULOS DE EMPRÉSTIMOS**
- **CÁLCULOS DE FINANCIAMENTOS**
- **CÁLCULOS DE PRESTAÇÕES DO AUTOMÓVEL**
- **CÁLCULOS DE PRESTAÇÕES DA CASA PRÓPRIA**
- **CÁLCULOS PARA FUNDOS DE APOSENTADORIA**
- **CÁLCULOS PARA FINANCIAMENTO DE CARTÃO DE CRÉDITO**
- **DEZENAS DE OUTROS EXEMPLOS PRÁTICOS**

Para cada capítulo, há uma breve introdução aos conceitos teóricos da Matemática Financeira, e logo a seguir são apresentados exemplos práticos, detalhadamente explicados, resolvidos passo-a-passo. Não são apresentadas longas discussões teóricas, pois este não é o foco do curso. O foco é apresentar o conceito e colocá-lo em prática logo em seguida, para que o leitor possa ver como é o funcionamento dos cálculos.

MESMO QUE VOCÊ NÃO SEJA "MUITO AMIGO DA MATEMÁTICA", COM ESTE CURSO VOCÊ VERÁ COMO É FÁCIL APRENDER MATEMÁTICA FINANCEIRA.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/intmatfin/default.asp>

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

Nota sobre direitos autorais

Este eBook é de autoria de **Robert F Martim**, sendo comercializado 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, desde que sejam para uso pessoal. É vetada a distribuição deste eBook, mediante cópia ou qualquer outro meio de reprodução, para outras pessoas. Se você recebeu este eBook através de e-mail ou via FTP de algum site da Internet, ou através de CD de Revista, saiba que você está com uma cópia pirata, não autorizada. Se for este o seu caso entre em contato com o autor através do e-mail rm@earnconsultoria.com.br ou comunique diretamente ao nosso site através do e-mail webmaster@juliobattisti.com.br.

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

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

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

Pré-requisitos

Para completar este curso é necessário um conhecimento prévio do ambiente de trabalho do VBE.

Este módulo visa a introdução aos conceitos básicos e intermediários da criação de funções no Excel. No decorrer do curso introduzirei conceitos de loops, depuração, descrição de macros, etc. Exemplos práticos são apresentados e exercícios sugeridos no final do curso para preparar o leitor para o desenvolvimento de funções personalizadas.

Se você não possui nenhum conhecimento de do ambiente do Visual Basic Editor (VBE) e de Visual Basic for Application (VBA) é recomendável que você inicie pelo curso básico de Excel que pode ser encontrado em <http://www.juliobattisti.com.br/cursos/excelbasico/default.asp>.

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

Objetivos deste eBook

Este eBook foi dividido em módulos que vão da abordagem dos assuntos básicos e intermediários até o avançado de criação de funções utilizando o VBA.

O trabalho foi desenvolvido a partir da demanda dos usuários do site www.juliobattisti.com.br. O material procura analisar questões pertinentes ao seu dia-a-dia.

Este curso assume que você tem conhecimento e domínio dos assuntos básicos e intermediários no que diz respeito ao ambiente de trabalho VBE e de VBA. Se este não é o seu caso, você pode adquirir o curso de introdução em <http://www.juliobattisti.com.br/cursos/excelbasico/default.asp>.

Os cursos são divididos em partes para que o leitor possa adquirir somente as partes onde há uma deficiência de conhecimento. Desta forma o conteúdo se torna mais relevante e o leitor não precisa obter material sobre o qual ele já tem domínio.

A linguagem utilizada é descontraída e com o mínimo de jargão possível. O objetivo é ter um eBook com conteúdo relevante e de fácil compreensão.

Qualquer dúvida referente a este módulo podem ser enviadas para o autor.

Todo esforço foi feito para assegurar que este eBook está livre de erros. Porém, no improvável caso do leitor encontrar qualquer erro, por favor, envie seus comentários e/ou sugestões diretamente para o autor no endereço rm@earnconsultoria.com.br.

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

ÍNDICE ANALÍTICO

INTRODUÇÃO.....	1
Bem-vindo	1
Antes de continuar.....	1
1. COMPREENDENDO O BÁSICO.....	2
1.1. Se sub-rotinas retornam valores, por que criar uma função para retornar valores?	2
1.2. Quais os tipos de funções que posso escrever?	3
1.2.1. Private Function	3
1.2.2. Public Function	5
1.2.3. Static Function	6
1.3. Onde devo colocar minhas funções?	8
1.4. Quais os tipos de valores retornados por uma função?	9
1.5. Decimais	10
2. COMPREENDENDO OS “LOOPS”	12
2.1. For — Next	12
2.2. For Each ... In ... Next.....	13
2.3. Do Until ... Loop	14
2.4. Do While ... Loop	15
2.5. Aninhando Loops (Loops em multiplas camadas).....	16
2.6. Loops em degrau (Stepped Loops)	18
2.7. Saindo do loop – instrução Exit	20
3. DEBUG DE FUNÇÕES.....	22
3.1. Debug.Print.....	22
3.2. MsgBox.....	23
3.3. Pontos de interrupção	24
4. DEFININDO AS OPÇÕES DAS FUNÇÕES.....	26
4.1. Definindo as informações da função	26
4.2. Adicionando a função a uma categoria	31
5. DEFININDO OS ARGUMENTOS DE SUAS FUNÇÕES.....	34
5.1. Funções sem argumentos	34
5.2. Funções com um ou mais argumentos.....	36
5.3. Funções com argumento opcional.....	38
5.4. Funções número indefinido de argumentos	41

6.	CHECANDO A VALIDADE DOS ARGUMENTOS.....	47
6.1.	Separando valores numéricos	47
6.2.	Separando valores vazios	48
6.3.	Separando valores numéricos e vazios simultaneamente	50
7.	CRIANDO MINHA PRIMEIRA FUNÇÃO.....	52
7.1.	Funções do tipo Boolean	52
7.1.1.	Comparando valores	52
7.1.2.	Definindo se um dado está presente em um conjunto de dados.....	53
7.1.3.	Determinando se um texto está acima do limite de caracteres permitidos.....	53
7.2.	Funções do tipo Double, long, single e Integer	54
7.2.1.	Calculando o movimento percentual	54
7.2.2.	Calcular o número de células com determinado número de cores de fundo.....	55
7.2.3.	Determinando a cor de fundo	55
7.3.	Funções do tipo string	56
7.3.1.	Retornando o nome da cor de fundo da célula selecionada	56
7.3.2.	Determinado o nome do objeto pai.....	57
7.4.	Outros tipos	58
7.4.1.	Adicionando dias úteis a uma data qualquer.....	58
7.4.2.	Determinando o número da semana dentro de um mês	60
8.	DESENVOLVENDO FUNÇÕES MAIS AVANÇADAS.....	61
8.1.	Funções do tipo Boolean	61
8.1.1.	Verificando se uma data é válida.....	61
8.1.2.	Determinando se uma barra de comando está travada	62
8.1.3.	Determinando se o número de um cartão de crédito é válido	62
8.2.	Funções do tipo Double.....	64
8.2.1.	Calculando a área, perímetro e diagonal de um quadrado.	64
8.2.2.	Calculando a área de um quadrado com uma função de seu lado, perímetro ou diagonal.	65
8.3.	Funções do tipo string	66
8.3.1.	Extraindo um elemento dentro de um texto qualquer.....	66
8.3.2.	Extraindo um elemento dentro de um texto qualquer através de uma matriz	67
8.4.	Outros tipos	69
8.4.1.	Adicionando dias úteis a uma data qualquer contabilizando os feriados	69

9.	EXERCÍCIOS PARA DESENVOLVIMENTO DE FUNÇÕES	72
9.1.	Funções do tipo Boolean	72
9.2.	Funções do tipo Double	73
9.3.	Funções do tipo string	73
9.4.	Outros tipos	73
10.	SOBRE O AUTOR	75

Autor: Robert F Martim

Criado em: 1/5/2005

Publicado: www.juliobattisti.com.br

Última edição: 3/6/2005

Contato: rm@earnconsultoria.com.br

Introdução aos conceitos de criação de funções no Excel

por Robert Friedrich Martim

INTRODUÇÃO

BEM-VINDO

Nas séries que serão escritas, cobrirei aspectos distintos do processo de criação de funções personalizadas no Excel. A intenção principal é fornecer ao internauta uma ferramenta que concentre a atenção na solução de um problema específico.

Neste módulo veremos como utilizar o VBA e suas ferramentas para personalizar funções.

ANTES DE CONTINUAR

O trabalho foi desenvolvido com um objetivo prático em mente e assume que você tem domínio dos pontos básicos do ambiente VBE. O objetivo não é martelar o leitor com conceitos que ele já tenha domínio. O objetivo é mostrar como criar funções que possam ser utilizadas no seu dia-a-dia para facilitar o seu trabalho.

Não existe um pré-requisito *per se* para completar este curso; porém, o leitor deve estar ciente de que com o conhecimento básico em dia, o aproveitamento do material será muito melhor. Sem o devido conhecimento de algumas partes básicas, este módulo se tornará mais laborioso e difícil do que realmente é.

Sugestões serão sempre bem-vindas e esperamos que o leitor participe pro-ativamente no desenvolvimento do material aqui apresentado.

Autor: [Robert F Martim](#)

Criado em:

[1/5/2005](#)

Publicado: www.juliobattisti.com.br

Última edição:

[3/6/2005](#)

Contato: rm@earnconsultoria.com.br

1. COMPREENDENDO O BÁSICO

Antes de continuar é necessário que você compreenda algumas diferenças básicas entre funções e sub-rotinas. Embora sub-rotinas possam efetuar cálculos como os efetuados em uma função, funções não podem manipular propriedade de objetos como fazemos em sub-rotinas.

O que você deve sempre ter em mente quando o assunto é funções diz respeito ao escopo das funções. Funções são utilizadas para retornar valores. O tipo de valor retornado pode variar, por exemplo, a função pode retornar um valor variável (variant) ou um valor booleano (boolean). Mas independente do valor retornado, você deve lembrar que sempre será retornado um valor.

O que vejo muitas vezes são pessoas tentando usar uma função para modificar a propriedade de uma célula (e o autor que lhe escreve também já fez isso, então, não se preocupe se você também já tentou e não obteve sucesso). Às vezes, fazemos um cálculo comparativo e se o resultado for 1 a célula deve ser pintada de preto e se for zero a célula deve ser pintada de vermelho. O cálculo pode ser feito pela função, mas a função será incapaz de modificar a cor da célula.

Uma forma de compreender o que você está lendo e lembrar das funções internas do Excel. Você alguma vez usou uma função do Excel para mudar a cor de fundo de uma célula? Com certeza, nunca! Você pode até ter utilizado a formatação condicional, contudo, a formatação condicional utiliza fórmulas com funções, mas não é a função que colore a célula. O que colore a célula é a rotina que avalia o resultado.

1.1. SE SUB-ROTINAS RETORNAM VALORES, POR QUE CRIAR UMA FUNÇÃO PARA RETORNAR VALORES?

Esta é uma pergunta que a experiência lhe dará a melhor resposta. Não obstante, tentarei mostrar os principais motivos por trás da criação de funções ao invés de efetuar os cálculos em uma sub-rotina.

Imagine uma situação onde você deseja saber se algo é verdadeiro ou falso. Se algo é verdadeiro ou falso o resultado é booleano e estamos falando de uma função Boolean. Agora, pense na situação onde você precisa avaliar diversas situações booleanas no mesmo contexto.

Se você efetuar o teste booleano dentro de uma sub-rotina que faz outras coisas, você teria que chamar tal sub-rotina a partir de outra sub-rotina que também utiliza o valor booleano. Se você fizer isso, todo o procedimento da sub-rotina será executado.

Desta forma, a solução seria criar a rotina de comparação dentro da outra sub-rotina, duplicando os esforços. Agora, coloque a rotina de comparação booleana em uma função. A função faz a comparação e retorna apenas o resultado, ela não faz mais nada além disso.

Autor: [Robert F Martim](#)

Criado em:

[1/5/2005](#)

Última edição:

[3/6/2005](#)

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Neste ponto, você deve ter notado que a solução é realmente uma função, pois ela pode ser reutilizada por todas as sub-rotinas e até mesmo por outras funções, sem transformar o seu código em uma verdadeira farofa mista, onde nem você sabe quais são os ingredientes.

Uma outra vantagem de se criar funções é que elas podem ser utilizadas em suas planilhas, também! Assim como as funções SOMA, SOMASE, etc, são utilizadas na planilha para efetuar cálculos, você pode utilizar a sua própria função na planilha para efetuar cálculos.

Esta talvez seja uma das grandes vantagens, pois você tem a flexibilidade converter mega-fórmulas em uma função que efetua todos os cálculos internamente e retorna o valor desejado na célula onde a fórmula anterior se encontrava. Os exemplos clássicos são cálculos de IR e comissão.

Como o cálculo de IR envolve faixas salariais, você teria que inserir várias vezes a função SE para avaliar cada situação e retornar o IR corretamente. Com uma função de cálculo de IR, tudo que você precisa fazer é inserir o salário, número de dependentes e se a pessoa paga ou não pensão. Além das faixas salariais, você terá agravantes como cálculos do INSS, dependentes, etc. Todos estes cálculos são efetuados internamente sem a necessidade de uma mega fórmula na célula.

Se os últimos parágrafos não foram suficientes para convencê-lo que funções devem ser criadas ao invés de fazer tudo em uma sub-rotina, relaxe e continue a ler. Ao começar a desvendar as maravilhas das funções, você verá e apreciará a real beleza das funções personalizadas.

1.2. QUAIS OS TIPOS DE FUNÇÕES QUE POSSO ESCREVER?

Ao criar funções personalizadas, você precisa estar ciente dos três tipos de funções possíveis. Você poderá criar funções Private, Public ou Static; contudo, cada uma tem um escopo diferente e o acesso a tais funções será dependente deste escopo.

1.2.1. Private Function

Private Function indica que a função criada é privada, em outras palavras, ela somente poderá ser acessada a partir do objeto que contém a função. Supondo que você insira a sua função em um módulo chamado “Módulo1” e no mesmo projeto exista um outro módulo chamado “Módulo2”, as funções criadas dentro do Módulo1 e declaradas como **Private** serão de uso exclusivo do Módulo1.

Caso você tente acessar uma função cujo escopo é privado, você obterá o seguinte erro:

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

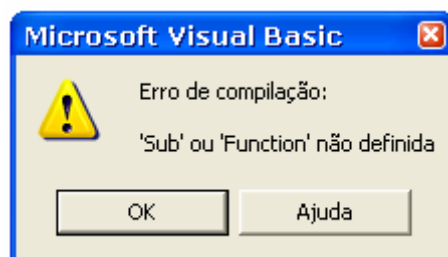


Figura 1-1

Se você alguma vez obteve este erro e ficou sem saber o seu significado, ele indica que a função ou sub-rotina chamada não pode ser acessada. Observe que o erro diz que a sub-rotina ou função não foram definidas. Contudo, a função ou sub-rotina pode ter sido definida, mas o seu escopo não permite acesso por uma rotina ou função externa.

Observe a figura:

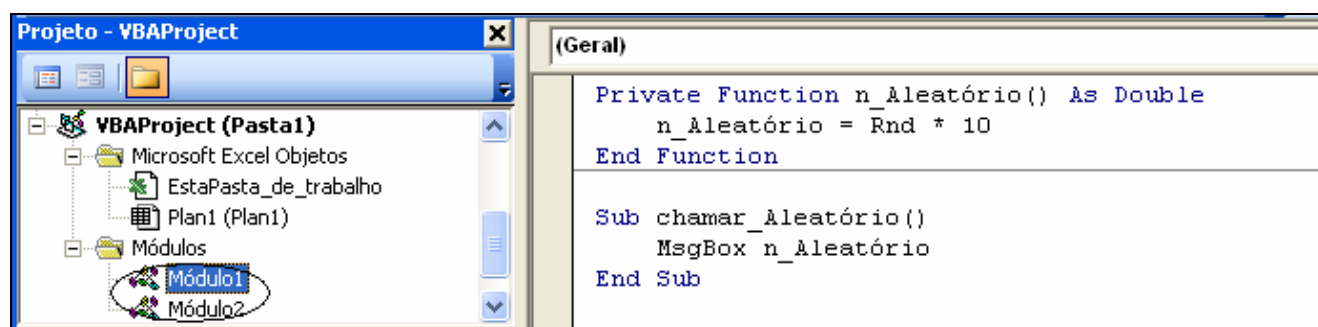


Figura 1-2

O projeto VBA contém dois módulos. O módulo ativo é o Módulo1. Ele possui uma função que retorna um número aleatório multiplicado por 10. No mesmo módulo há uma sub-rotina que chama a função e mostra o resultado em uma caixa de mensagem (MsgBox):

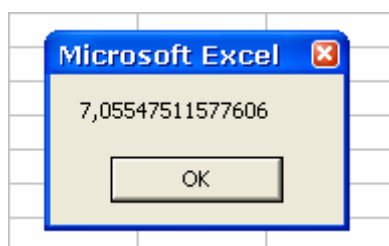


Figura 1-3

No Módulo2 também há um procedimento para chamar a função; contudo, como a rotina está fora do módulo que contém a função e a função tem um escopo privado o acesso é negado e um erro retornado:

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

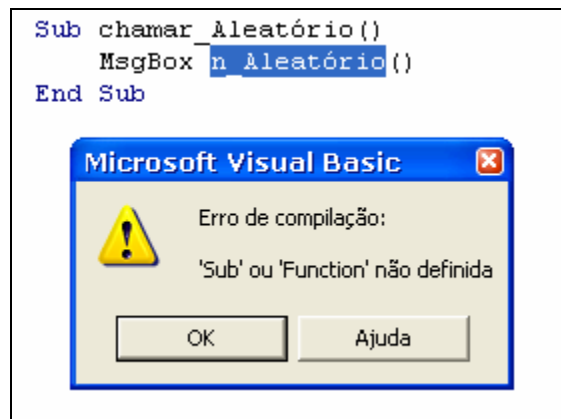


Figura 1-4

Como você pode observar o VBA não somente retorna o erro como mostra o local onde o erro ocorreu. Isso é importante, pois você poderá rapidamente isolar o problema e resolvê-lo. Se você sabe que a função existe em seu projeto, mas obtém a mensagem acima a primeira coisa a fazer é verificar o escopo da função.

Finalmente, uma função Private não será mostrada na caixa de inserção de funções do Excel.

1.2.2.Public Function

Public Function indica que a função criada é pública, em outras palavras, ela poderá ser acessada a partir de qualquer objeto dentro de seu projeto e não somente do objeto contenedor da função. Supondo que você insira a sua função em um módulo chamado “Módulo1” e no mesmo projeto existe um outro módulo chamado “Módulo2”, se a função for declarada como **Public** você poderá chamá-la de qualquer local no seu projeto.

Observe o exemplo:

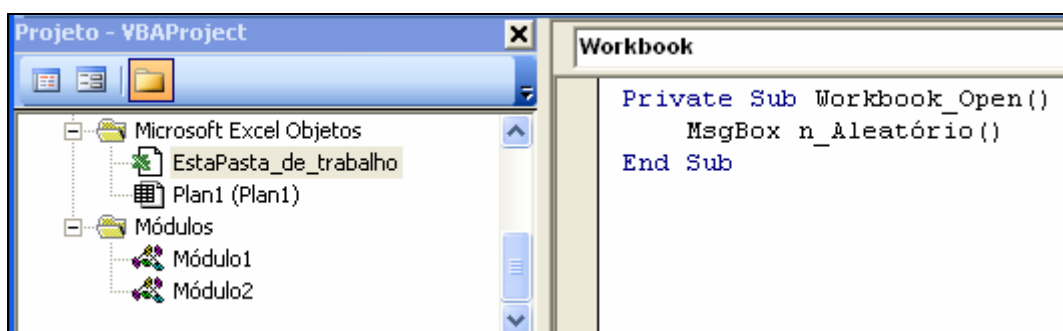


Figura 1-5

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

O Módulo1 continua a ter a função que gera o número aleatório multiplicado por 10. Na figura acima, a função é chamada assim que a pasta de trabalho é aberta e uma caixa de mensagem é mostrada com o número aleatório gerado:

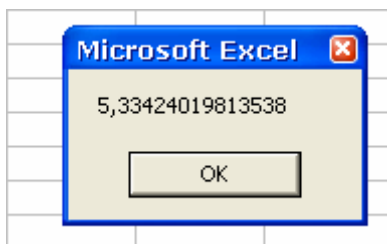


Figura 1-6

1.2.3. Static Function

Static Function indica que a função criada é estática, em outras palavras, as variáveis contidas na função manterão o valor original entre as diversas chamadas da função até que as variáveis sejam reiniciadas.

Não obstante, se você tentar aplicar a função estática a um valor aleatório, obviamente que o valor deixará de ser “estático”. Isso ocorrerá por um simples motivo: a sua função é estática, mas a função **rnd** não é. Como o resultado da função é dependente da geração de números aleatórios através da função **rnd** torne-se claro que a variável estática da função perde valor.

Mas e se você deseja gerar um número aleatório na primeira chamada da rotina e manter este número fixo para as próximas chamadas? E para cada chamada subsequente adicionar um número inteiro aleatório ao valor da primeira chamada e assim sucessivamente?

Aqui, você precisa manter em mente a palavra “Static” (estático). Já sabemos que a função **rnd** gera um novo número cada vez que ela é chamada. O segredo do sucesso é passar o resultado da função **rnd** para uma variável estática a qual será reutilizada toda vez que a função for chamada. Por exemplo:

```
Static Function n_Aleatório() As Double

    If IsEmpty(núm) Then
        núm = Rnd * 10
    Else:
        novoNúm = núm + Int((10 - 0 + 1) * Rnd + 0)
        MsgBox "Valor atual da variável estática: " & núm & vbCrLf _
            & "Vlor do novo número aleatório: " & novoNúm
    End If

    n_Aleatório = núm + novoNúm

End Function

Sub chama_Aleatório()
    MsgBox n_Aleatório()
End Sub
```

Código 1-1

A figura acima mostra como isso pode ser feito. Adicionamos a variável **núm** e checamos se ela está vazia quando a função é ativada. Se ela estiver vazia; então, um número aleatório é gerado para ela.

Nas chamadas subseqüentes, a variável não mais estará vazia e desta vez um novo número aleatório é gerado. Uma caixa de mensagem inserida na função mostra o valor anterior da variável estática **núm** e do novo número aleatório gerado. O resultado da função é o somatório da variável estática e o novo número:

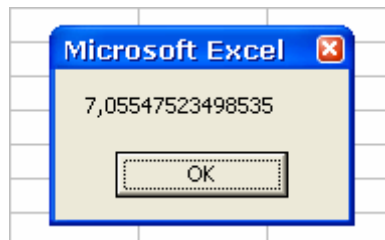


Figura 1-7

A figura acima mostra a primeira chamada da função. Ao acionar a função uma segunda vez:

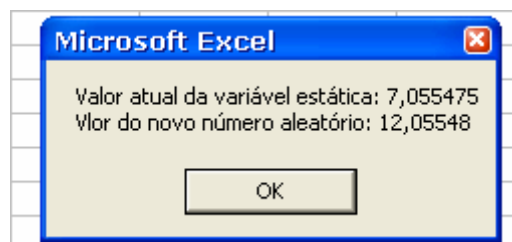


Figura 1-8

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

O valor da primeira chamada não muda entre uma chamada e outra. Geramos um novo número aleatório que é, então, somado ao valor anterior e mostrado na caixa de mensagem da rotina:

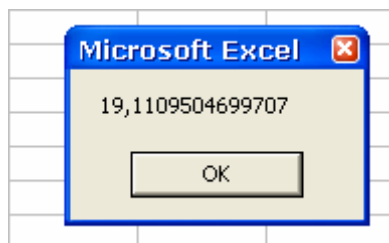


Figura 1-9

Ao chamar a rotina uma terceira vez:

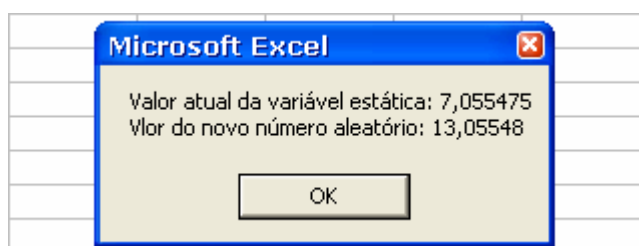


Figura 1-10

O valor da variável estática **núm** permanece exatamente o mesmo. Um novo número aleatório é gerado e somado a ela.

1.3. ONDE DEVO COLOCAR MINHAS FUNÇÕES?

Onde você insere suas função determina o escopo privado ou público de sua função. Nos exemplos anteriores as funções são todas colocadas em um módulo; porém, você não é obrigado a inserir suas funções em módulos. Você pode inserir suas funções em um formulário, na pasta de trabalho, em um módulo, classe ou planilha.

O importante é você lembrar e compreender o que ocorre quando a sua função é inserida em um destes objetos.

Uma função será sempre pública se você a inserir em um módulo. Ao inserir uma função em um módulo você não precisa declarar a função como **Public**, pois todos os outros objetos terão acesso a ela desde que ela não seja explicitamente declarada como **Private**.

A história pode ficar confusa para funções públicas quando elas são adicionadas em objetos como planilhas, formulários e pasta de trabalho.

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

O escopo destes objetos é privado, portanto, qualquer função inserida em um destes objetos é automaticamente classificada como privada. A pergunta que fica no ar é: “- Se as funções são privadas, por que eu iria querer declarar uma função como **Private** ou **Public** dentro destes objetos?”

A verdade é que você não quererá. Como funções privadas têm precedência sobre funções públicas, imagine a situação onde a pasta de trabalho possui uma função (implicitamente privada) com o mesmo nome que uma função em um módulo. O que você acha que acontecerá? Absolutamente nada. Ao chamar a função de dentro do objeto contenedor, a função que a ser executada será a função contida no mesmo objeto e não a função pública no módulo. Não haverá erro de ambigüidade no nome da função.

Os objetos que compõem o seu projeto terão os seguintes escopos:

- Planilha – funções são automaticamente consideradas **Private**. Declarar uma função como **Public** não modificará o escopo de acesso.
- Pasta de trabalho – funções são automaticamente consideradas **Private**. Declarar uma função como **Public** não modificará o escopo de acesso.
- Formulário – funções são automaticamente consideradas **Private**. Declarar uma função como **Public** não modificará o escopo de acesso.
- Módulo – funções são automaticamente consideradas **Public**. Declarar uma função como **Private** modificará o escopo de acesso.
- Classe – funções são automaticamente consideradas **Public**. Declarar uma função como **Private** modificará o escopo de acesso.

1.4. QUAIS OS TIPOS DE VALORES RETORNADOS POR UMA FUNÇÃO?

Funções são utilizadas para retornar valores e mais nada. Portanto, é importante que você conheça e experimente com os diversos tipos disponíveis no VBA.

Uma questão chave sobre o quão eficiente será sua função depende exatamente de conhecimento dos tipos possíveis. Neste módulo, estarei discutindo os tipos básicos e deixarei de fora o tipo **variant**. Este tipo requer um conhecimento mais avançado de funções e o abordarei no módulo de criação de funções avançadas onde mostrei como manipular funções com diversas dimensões. Aqui, apenas apresentarei uma introdução geral ao conceito e alguns exemplos.

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Alguns dos tipos abordados são:

- **Boolean** – o tipo Boolean retorna um valor verdadeiro ou falso. Ao converter valores numéricos para valores booleanos o valor 0 (zero) assume um valor falso e *todos* os outros valores assumem um valor verdadeiro. Se você converter um valor booleano para um outro tipo de dado qualquer, o valor falso é convertido como 0 (zero) e o valor verdadeiro é convertido para -1 (menos 1).
- **Byte** – o tipo Byte retorna um valor compreendido entre 0 e 255 (256 valores)
- **Double** – o tipo Double (Duplo) retorna um número duplo compreendido entre -1,79769313486231E308 a -4,94065645841247E-324 para valores negativos e 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos.
- **Integer** – o tipo Integer (inteiro) retorna um número inteiro e estará compreendido entre os números -32.768 a 32.767.
- **Long** – o tipo Long (longo) retorna um número longo inteiro compreendido entre -2.147.483.648 a 2.147.483.647.
- **Single** – o tipo Single (simples) retorna um número simples compreendido entre -3,402823E38 a -1,401298E-45 para valores negativos e 1,401298E-45 a 3,402823E38 para valores positivos.
- **String** – o tipo String retorna ou manipula texto.

Ao criar uma função você precisa observar o tipo de dado, pois se o valor retornado pela função extrapola o escopo do tipo de dado haverá um estouro na variável.

1.5. DECIMAIS

A questão dos decimais pode gerar confusão para usuários de países que utilizam a vírgula como separador decimal. Em países de língua inglesa, o separador de decimal é geralmente o “ponto”, em outras palavras o número 2450,25 será representado por 2450.25 no VBA.

Se o milhar é separado por ponto no sistema brasileiro, ele será separado por vírgula no sistema americano.

Porém, você não deve utilizar vírgula para separar números no VBA, pois a vírgula é utilizada para separar argumentos. Observe a figura:

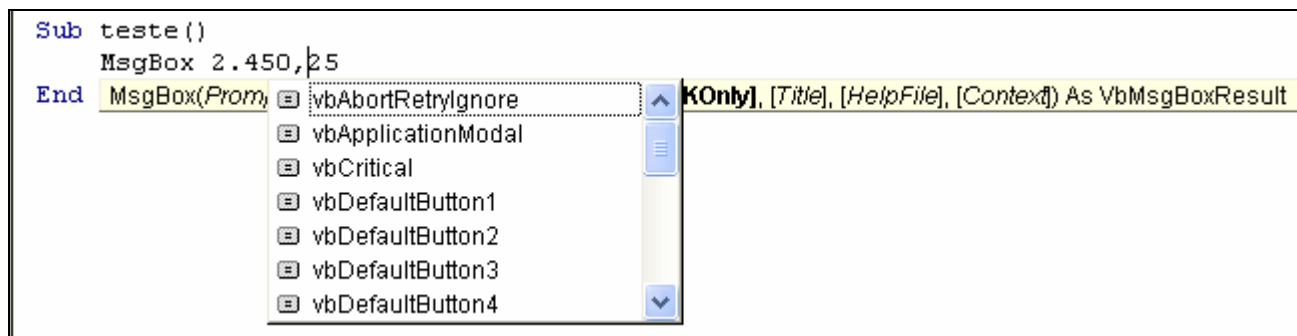


Figura 1-11

Ao inserir a vírgula como separador de decimal, você é requisitado a inserir o próximo argumento da função MsgBox.

Ao criar funções você não precisa separar o milhar, somente é necessário separar o decimal. Mas lembre-se sempre que o decimal é sempre separado com ponto no VBA.

2. COMPREENDENDO OS “LOOPS”

Ao criar funções, muitas vezes precisaremos avaliar vários itens em um conjunto de dados. Para fazer isso utilizamos um “loop” para varrer todos os dados do conjunto. O tipo de loop utilizado determinará o resultado obtido; portanto, é importante que você experimente com os diversos “loops” e se familiarize com eles.

A seguir, apresento os loops que utilizaremos e como eles funcionam.

2.1. FOR — NEXT

O loop For i = NúmeroInicial To NúmeroFinal — Next i é utilizado para varrer um conjunto de dados unidirecional. Os dados sendo varridos podem estar na vertical, horizontal ou diagonal, não importa, mas o loop não varrerá todas as direções. Ele moverá em uma direção apenas.

A variável “i” pode ser modificada para qualquer variável que você desejar (j, k, h, p, etc). A variável é declarada como “i”, pois ela representa um valor inteiro (integer). A letra “i” foi retirada da palavra integer para representar um loop através de uma variável cujo tipo de dado é integer.

	A	B	C
1		55	
2		19	
3		10	
4		44	
5		34	
6		23	
7		98	
8		4	
9		39	

Figura 2-1

A figura acima mostra um conjunto de nove valores contidos na intervalo B1:B9. Você poderia criar o seguinte loop para varrer os dados na coluna B utilizando For – Next:

```
Function minhaSoma()
  For i = 1 To 9
    minhaSoma = minhaSoma + Range("B" & i)
  Next i
End Function
```

Código 2-1

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Esta função não recebe nenhum argumento e assume todos os valores estão no intervalo B1:B9. Cada vez que o loop é executado a soma anterior é adicionada o valor contido na célula "B & i". Se i for igual a 7, o valor retornado é o da célula B & i ou B7.

Não é necessário declarar explicitamente que o próximo i (Next i) seja avaliado. Se você digitar apenas Next, o loop ocorrerá da mesma forma.

2.2. FOR EACH ... IN ... NEXT

O loop For Each ... In... Next possui uma enorme vantagem sobre o For i, pois ele faz um loop em todos os valores contidos no conjunto de dados e, portanto, varrerá em todas as direções ao passo que For i varre somente unidirecionalmente.

A1		fx =minhaSoma(B1:C9)			
	A	B	C	D	E
1	326	55	39		
2		19	59		
3		10	39		
4		44	68		
5		34	33		
6		23	69		
7		98	93		
8		4	85		
9		39	78		

Figura 2-2

Na figura acima, a função minhaSoma() recebe os argumentos do intervalo B1:C9, contudo, ela retorna apenas a soma dos valores em B1:B9. Veja como está o código:

```
Function minhaSoma(valores As Range)
    For i = 1 To valores.Count
        minhaSoma = minhaSoma + valores(i, 1)
    Next i
End Function
```

Código 2-2

Aqui, é ignorada a segunda coluna do conjunto de dados. Como não é possível definir as duas dimensões simultaneamente, você precisa avaliar a coluna 1 e a coluna 2 no seu loop. Uma solução melhor é utilizar For Each ... in ... Next:

	A1		fx	=minhaSoma(B1:C9)		
	A	B	C	D	E	
1	889	55	39			
2		19	59			
3		10	39			
4		44	68			
5		34	33			
6		23	69			
7		98	93			
8		4	85			
9		39	78			

Figura 2-3

Com este loop não precisamos nos preocupar com o número de colunas ou linhas do intervalo, pois o loop será feito para cada valor contido no intervalo selecionado:

```
Function minhaSoma(valores As Range)
    For Each valor In valores
        minhaSoma = minhaSoma + valor
    Next
End Function
```

Código 2-3

O loop ocorre para *cada valor* no conjunto de *valores*. Cada *valor* é somado em nossa função.

2.3. DO UNTIL ... LOOP

Do Until ... Loop executará o loop até que o critério definido na função seja encontrado. Enquanto que o loop pode parecer uma boa idéia, você precisa se atentar para a possibilidade de entrar em um loop infinito.

Suponha que a intenção do loop seja a convergência para um valor qualquer. Se o loop é baseado em valores aleatórios que são comparados com o valor buscado, você pode passar o resto de sua vida buscando por tal número, pois o universo de números aleatórios é infinito. Portanto, a probabilidade de você encontrar o número que deseja a partir de sorteios aleatórios é zero no limite.

O código a seguir gerará um loop eterno:

```
Function minhaSoma(valores As Range)
    Do Until IsEmpty(valores)
        minhaSoma = minhaSoma + valor
    Loop
End Function
```

Código 2-4

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Se você deseja avaliar cada valor contido no conjunto de valores utilizando Do Until ... Loop, uma solução seria:

```
Function minhaSoma(valores As Range)
    lin = 1
    Do Until IsEmpty(valores(lin, 1))
        minhaSoma = minhaSoma + valores(lin, 1) + valores(lin, 2)
        lin = lin + 1
    Loop
End Function
```

Código 2-5

Embora ele resolva o problema para o exemplo, o código é completamente ineficiente se comparado com o do sub-tópico anterior.

2.4. DO WHILE ... LOOP

Do While ... Loop executará o loop até que o critério definido na função seja encontrado. Este loop é bem similar ao Do Until ... Loop. Neste caso, o que você deseja fazer é comparar algum resultado durante o loop onde o loop somente parará quando o resultado for encontrado.

Novamente, você deve ter cuidado com este tipo de loop, pois ele pode entrar em uma execução infinita.

O exemplo abaixo mostra como este loop pode ser utilizado:

```
Function minhaSoma(valores As Range)
    n = valores.Rows.Count
    lin = 1
    Do While lin <= n
        minhaSoma = minhaSoma + valores(lin, 1) + valores(lin, 2)
        lin = lin + 1
    Loop
End Function
```

Código 2-6

Aqui, primeiramente pegamos o número de linhas no conjunto de valores utilizando **valores.Rows.Count**. Em seguida, avaliando o loop até que o valor **lin** seja menor ou igual a **n**. Quando está condição for verdadeira, o loop é encerrado e o resultado da função é retornado.

2.5. ANINHANDO LOOPS (LOOPS EM MULTIPLAS CAMADAS)

Você deve ter notado que de todos os loops apresentados, o que parece ser mais eficiente é o For Each ... In ... Digo parece porque se ele realmente fosse o mais eficiente não haveria necessidade de termos qualquer outro tipo de loop. É importante compreender que cada loop tem um objetivo e cada um será eficiente se o utilizarmos para a sua função principal. Se desejarmos avaliar uma matriz unidirecional, não For i – Next i é muito mais eficiente, pois ele move em apenas uma direção.

Utilizar um loop em camada significa envelopar um ou mais loops em cascata. Em outras palavras, colocar um loop dentro de outro.

Para o primeiro exemplo, utilizarei For i. Utilizando o conjunto de dados anterior observe a figura:

	A	B	C	D
1				
2	889	55	39	
3		19	59	
4		10	39	
5		44	68	
6		34	33	
7		23	69	
8		98	93	
9		4	85	
10		39	78	
11				

Figura 2-4

As setas indicam as direções em que o loop deverá mover. O loop moverá na vertical (linhas) e na horizontal (colunas). Como o Excel é uma grande matriz, se você conseguir visualizar que o loop será de n-linhas por n-colunas, a construção do mesmo será muito simples:

```
Function minhaSoma(valores As Range)
    linhas = valores.Rows.Count
    colunas = valores.Columns.Count

    For i = 1 To linhas
        For j = 1 To colunas
            minhaSoma = minhaSoma + valores(i, j)
        Next j
    Next i
End Function
```

Código 2-7

Neste exemplo, removemos o problema das colunas, pois o número de colunas será contado assim como o número de linhas. Usamos estes valores para criar um loop que avalia uma linha e todas as

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

colunas da linha antes de mover para a próxima linha quando, novamente, serão avaliadas as colunas da linha em questão.

O próximo exemplo de aninhamento de loops é bem similar ao anterior, mas muda a forma como as colunas a serem somadas são inseridas:

```
Function somarColunas(valores As Range, colunas As Byte, colunaInicial As Byte)
    linhas = valores.Rows.Count
    colunas = colunaInicial + colunas - 1

    For i = 1 To linhas
        For j = colunaInicial To colunas
            somarColunas = somarColunas + valores(i, j)
        Next j
    Next i
End Function
```

Código 2-8

Aqui o usuário escolhe em qual coluna se inicia a soma e quantas colunas devem ser somadas:

A1	fx =somarColunas(B2:H18;3;2)							
	A	B	C	D	E	F	G	H
1	2471							
2		98	44	61	89	55	57	13
3		45	8	64	16	70	27	90
4		77	28	86	73	79	32	12
5		76	53	95	94	48	33	80
6		26	15	54	6	70	9	60
7		28	24	19	19	36	50	99
8		20	18	49	33	59	96	26
9		7	5	19	59	6	95	16
10		11	66	64	2	0	37	86
11		56	21	30	95	98	95	82
12		22	34	37	24	85	52	51
13		75	63	78	69	13	85	36
14		12	58	56	0	10	65	77
15		20	32	31	55	95	25	35
16		90	22	92	98	96	51	38
17		43	45	14	96	97	38	37
18		30	81	97	80	13	88	95

Figura 2-5

A loop em si não muda, mas a forma como o alimentamos é diferente da anterior.

O próximo exemplo mostra como criar um loop com Next e Do While sendo que o segundo é aninhado dentro do primeiro:

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

```

Function gerarValor()
    For i = 1 To 10
        Do While n < 10
            n = Rnd * 15
        Loop
        gerarValor = gerarValor + i + n
        n = 0
    Next i
End Function

```

Código 2-9

A função não leva argumento algum. Ela apenas gera um número qualquer que é o somatório de *i* com um valor aleatório maior do que 10.

2.6. LOOPS EM DEGRAU (STEPPED LOOPS)

Existe uma classe de loops que usa um valor de Step (degrau) para pular ou criar um intervalo dentro do loop. Suponha que o seu loop inicia na linha 1, mas você deseja apenas somas os valores contidos nas linhas ímpares. Você pode utilizar um valor Step para fazer isso:

```

Function somarLinhas(valores As Range, nDegrau As Byte)
    ' nDegrau = 1 --> Somar todas a linhas
    ' nDegrau = 2 --> Somar linhas ímpares
    ' nDegrau = 3 --> Somar linhas pares

    If nDegrau = 1 Or nDegrau = 2 Then
        início = 1

        ElseIf nDegrau = 3 Then
            início = 0
            nDegrau = 2
    End If

    On Error Resume Next
    For i = início To valores.Rows.Count Step nDegrau
        For j = 1 To valores.Columns.Count
            somarLinhas = somarLinhas + valores(i, j)
        Next j
    Next i

End Function

```

Código 2-10

Se o usuário deseja somar todas as linhas selecionadas, ele deve definir o número de degraus como sendo 1 (Step 1). Se o desejo for somar somente as linhas ímpares, ele digitará 2 e para a soma das linhas pares ele deve digitar 3.

A figura a seguir mostra como o resultado é colocado na planilha:

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

	B21		fx =somalinhas(A1:G17;B22)					
	A	B	C	D	E	F	G	H
1	98	44	61	89	55	57	13	
2	45	8	64	16	70	27	90	
3	77	28	86	73	79	32	12	
4	76	53	95	94	48	33	80	
5	26	15	54	6	70	9	60	
6	28	24	19	19	36	50	99	
7	20	18	49	33	59	96	26	
8	7	5	19	59	6	95	16	
9	11	66	64	2	0	37	86	
10	56	21	30	95	98	95	82	
11	22	34	37	24	85	52	51	
12	75	63	78	69	13	85	36	
13	12	58	56	0	10	65	77	
14	20	32	31	55	95	25	35	
15	90	22	92	98	96	51	38	
16	43	45	14	96	97	38	37	
17	30	81	97	80	13	88	95	
18								
19								
20								
21	Soma	3165						
22	Tipo	2						

Figura 2-6

O tipo é digitado na célula B22. Esta célula é também utilizada na formatação condicional das células contendo o intervalo de dados. Ao modificar o tipo para 1 ou 3, as células são formatadas para mostrar o que está sendo somado.

Observe que Step refere-se ao “pulo” dentro do loop. Quando Step é igual a 1 o loop ocorre com incrementos de 1. Se você entrasse um valor Step igual a zero, o loop deixaria de ocorrer pois o valor jamais chegar no valor superior do loop.

Em alguns casos você pode desejar um valor Step bem pequeno para fazer uma interação, por exemplo:

```
Function somarSteps()
    For i = 0 To 1 Step 0.01
        somarSteps = somarSteps + 0.01
    Next
End Function
```

Código 2-11

Fará o loop cem vezes e o resultado será 1 dado o somatório de 0,01 a cada loop. Este loop tem o mesmo efeito que o loop a seguir o qual não utiliza o valor Step:

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

```
Function somarCem()  
    For i = 1 To 100  
        somarCem = somarCem + 0.01  
    Next  
End Function
```

Código 2-12

O loop sem Step, como na função acima, será, certamente, mais conhecido do que o anterior. Contudo, ambos fazem exatamente a mesma coisa, mas de um prisma diferente.

2.7. SAINDO DO LOOP – INSTRUÇÃO EXIT

Haverá situações onde você desejará sair do loop se certo critério for encontrado. Suponha que você não queira que um loop ocorra por mais de 100 vezes. Se isso ocorrer, o loop é encerrado e a função retorna um erro.

Neste caso, podemos utilizar a instrução Exit para deixar o loop ao invés de esperar por um loop que pode ser tornar infinito. A instrução Exit pode ser utilizada em vários contextos como em Exit Function ou Exit Sub.

```
Function sairDO()  
    Do While n < 100  
        n = Rnd * 105  
        i = i + 1  
  
        If i >= 5 Then  
            sairDO = i  
            Exit Do  
        End If  
        sairDO = n  
    Loop  
End Function
```

Código 2-13

No exemplo acima, o loop contendo *While* deve ocorrer enquanto *n* for menor do que 100. Contudo, se o loop ocorrer 5 vezes seguidas sem obter um valor superior a 100; então o loop deve ser terminado e o resultado da função é igual a *i*.

O exemplo a seguir mostra como sair do loop para For-Next:

```
Function sairFOR()  
    Application.Volatile  
    For i = 1 To 100  
        sairFOR = Int(Rnd * 300)  
        If sairFOR = 10 Then Exit For  
    Next  
End Function
```

Código 2-14

Se o número sorteado for igual a 10, o loop é parado independentemente do número de loops que tenha ocorrido. Se o número sorteado for 10 logo no primeiro loop, não haverá necessidade de continuar o loop.

3. DEBUG DE FUNÇÕES

Ao criar funções, você notará que a forma de debug é diferente do debug de uma sub-rotina. Isso ocorre porque você não pode rodar a função utilizando F5 ou F8. A função somente rodará quando o cálculo for chamado.

Desta forma precisamos utilizar as ferramentas de debug do VBA para avaliar a função. Um truque que geralmente uso é criar a função como se fosse uma sub-rotina e após terminar os testes eu converto para uma função. Além de dar a flexibilidade do uso de F5 e F8, você também poderá utilizar as ferramentas de debug do VBA.

Para utilizar o método Print do objeto Debug você precisará da janela de verificação imediata aberta. Para isso, vá até Exibir → Janela 'Verificação imediata' ou pressione CTRL+G.

3.1. DEBUG.PRINT

O método Print do objeto Debug serve para “imprimir” o resultado de uma variável na janela de verificação imediata.

A utilização deste método é bastante simples e extremamente útil.

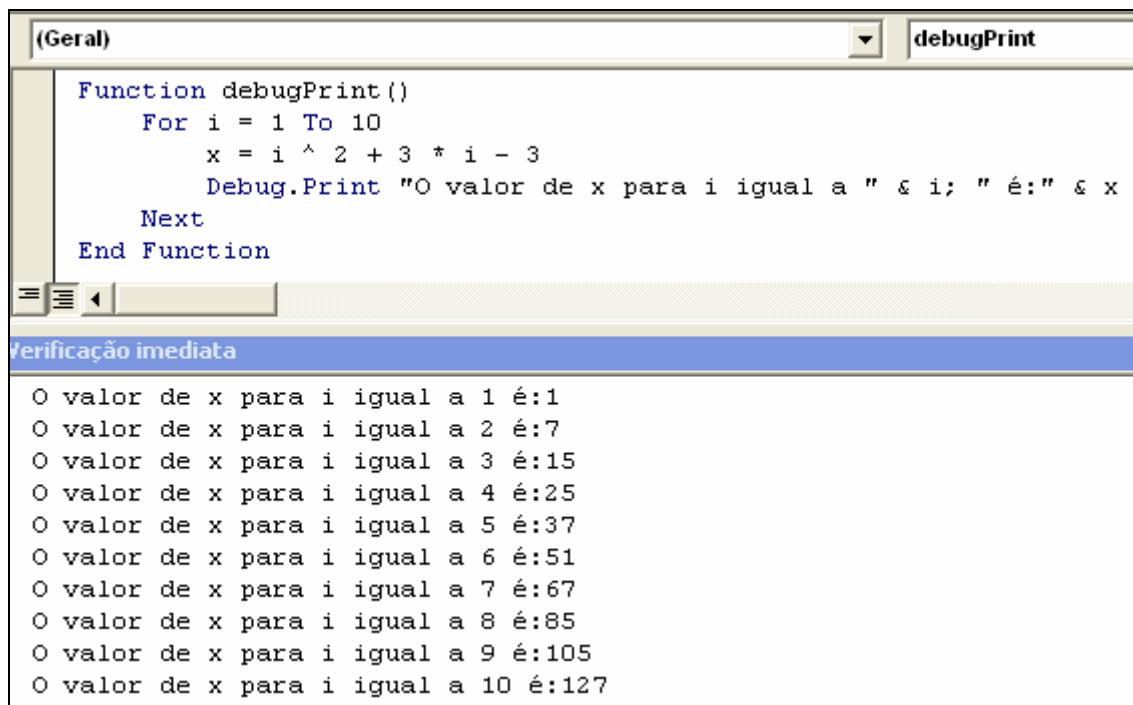


Figura 3-1

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

A figura acima mostra um loop dentro da função para os valores de i de 1 a 10. Para cada loop, o valor de x é calculado e escrito, junto com uma mensagem, na janela de verificação imediata.

O método Print pode ser utilizado em qualquer parte de seu código para avaliar uma ou mais variáveis. Ao criar suas funções o ideal é avaliar cada bloco separadamente e ir corrigindo conforme a necessidade. Se você usa método Print em todos os passos simultaneamente, ele pode acabar ofuscando erros ao invés de apontá-los, pois você terá que lidar com diversos resultados ao mesmo tempo, o que pode ser confuso.

3.2. MsgBox

Utilizar uma caixa de mensagem para fazer depuração de seu código é uma boa idéia se ela avalia apenas um resultado. Se você possui diversas variáveis que precisam ser depuradas, utilize o método Print.

Você verá que isso é verdade quando houver um volume grande de variáveis e para cada resultado você receber uma caixa de mensagem:

```
Function debugMsgBox()  
    For i = 1 To 10  
        x = i ^ 2 + 3 * i - 3  
        MsgBox "O valor de x para i igual a " & i & " é:" & x  
    Next  
End Function
```

Código 3-1

O código acima faz exatamente a mesma coisa que o método Print, porém, a cada loop uma caixa de mensagem é passada com o valor de x :

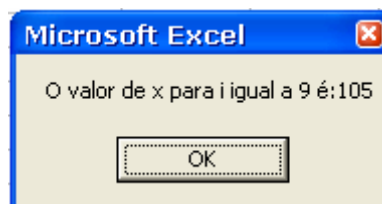


Figura 3-2

Embora depuração com caixa de mensagem seja extremamente útil ela pode ser tornar extremamente irritante lá pela décima mensagem. Geralmente, utilizo caixas de mensagens exatamente para o objetivo delas: passar uma mensagem.

Se existe algo que necessitará de minha atenção, prefiro ter uma caixa de mensagem como lembrete do que necessito fazer do que ter um comentário.

A caixa de mensagem é uma boa forma de evitar o esquecimento de algo que precisará de sua atenção ao final da depuração, além de ser uma ferramenta viável para depuração de uma ou duas variáveis em sua função.

3.3. PONTOS DE INTERRUPÇÃO

Um outro método para depuração de código requer a utilização de pontos de interrupção (line breaks ou breakpoints).

Para inserir um ponto de interrupção em seu código você deve selecionar uma linha que possa ser interrompida. Você não pode colocar um ponto de interrupção em linhas que dimensionam variáveis, por exemplo.

Ao selecionar a linha, para adicionar um ponto de interrupção basta clicar na barra no canto esquerdo e a linha é marcada para interrupção durante a execução do código:

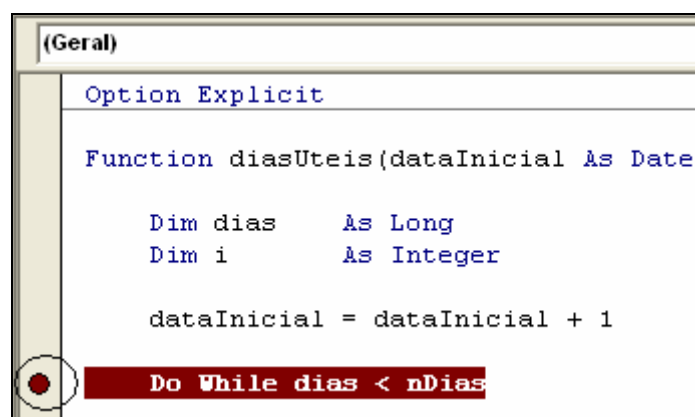


Figura 3-3

Uma outra opção é inserir diretamente do menu Depurar. Clique na linha onde você deseja inserir o ponto de interrupção vá até Depurar → Ativar/Desativar pontos de interrupção.

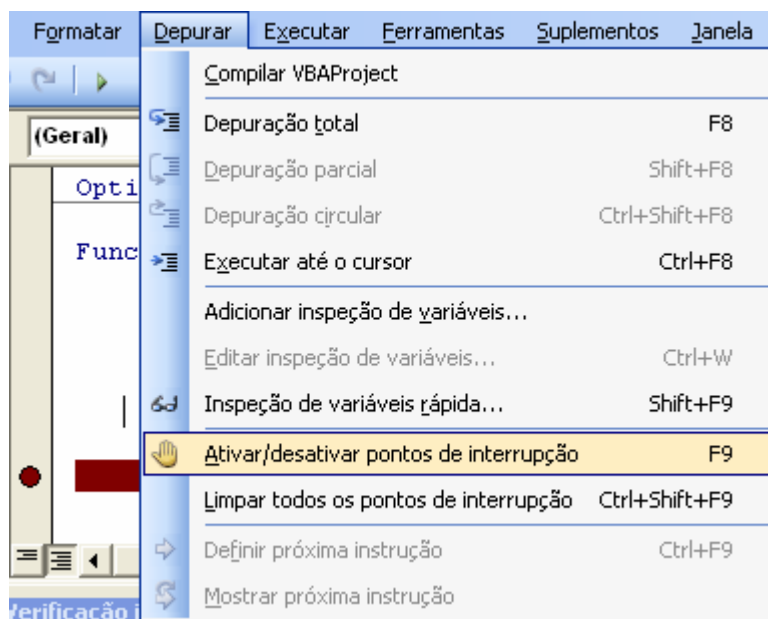


Figura 3-4

Como a figura acima mostra, ainda há a possibilidade de inserir/remover um ponto de interrupção pressionando a tecla F9.

Ao chamar a função, a execução será interrompida na posição onde o ponto de interrupção foi inserido:

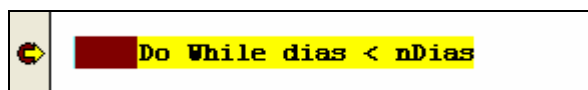


Figura 3-5

Observe que se o ponto de interrupção for colocar em algum ponto dentro de um loop, a execução será interrompida toda vez que o código retornar à linha dentro do loop até que o loop termine.

Não existe uma regra para usar pontos de interrupção, mas eles são extremamente úteis quando uma parte do código está pronta e precisamos depurar apenas pontos específicos. Aqui, fica fácil passar por cima de qualquer coisa que esteja pronta e concentrar no que precisa ser resolvido.

4. DEFININDO AS OPÇÕES DAS FUNÇÕES

Quando criamos funções personalizadas no Excel desejamos realmente personalizar ao máximo o nosso trabalho. Embora existam algumas limitações ao que podemos personalizar, ainda existem várias opções de personalização que podemos utilizar para dar as nossas funções um ar bem pessoal.

Dentre as opções, podemos adicionar comentários (descrição) a nossa função para guiar o usuário, isto é, dar a ele informações sobre o que a função faz e como utilizá-la. Uma outra opção diz respeito à categorização de nossas funções.

Você deve ter notado que ao abrir a caixa de inserção de função você tem várias opções de como mostrar a lista de funções. A primeira lista mostra as funções mais recentes, contudo, você pode selecionar uma das categorias, digamos, Estatística, onde somente as funções estatísticas são mostradas.

Nos dois tópicos a seguir discuto como implementar estas opções.

4.1. DEFININDO AS INFORMAÇÕES DA FUNÇÃO

Quando você cria uma função no Excel, há a possibilidade de adicionar descrições sobre a sua função. Estas descrições podem ser vistas nas funções internas do Excel:

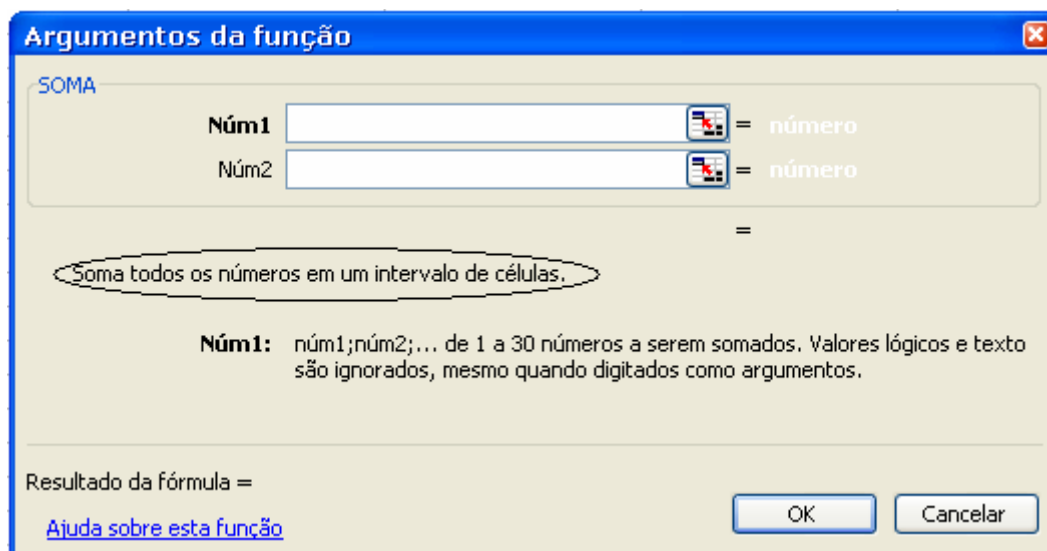


Figura 4-1

Esta descrição pode ser facilmente adicionada a qualquer função personalizada. A informação pertinente aos argumentos, no caso acima **Núm1** e **Núm2**, infelizmente, não pode ser definida no Excel. Certamente que se isso fosse possível as funções teriam um ar muito mais profissional.

Autor: Robert F Martim

Criado em: 1/5/2005

Última edição: 3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Existem várias formas de se inserir tais descrições. A primeira pode ser acessada através de Ferramentas → Macro → Macros (ou simplesmente pressione Alt+F8).

A caixa de diálogo de macros é aberta. Digite o nome da função no campo **Nome da macro** e o botão **Opções** será ativado:

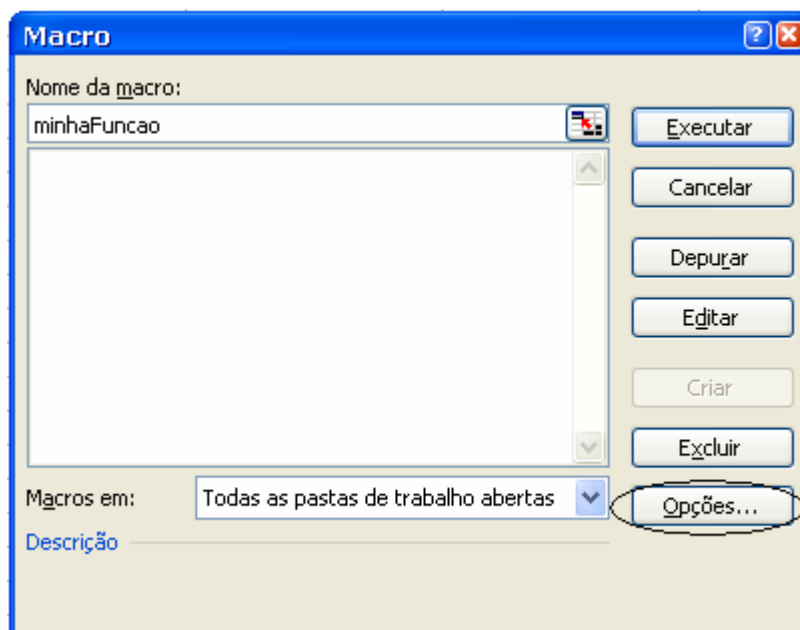


Figura 4-2

Clique no botão de opções para acionar as opções da função onde você poderá digitar os detalhes de sua função:

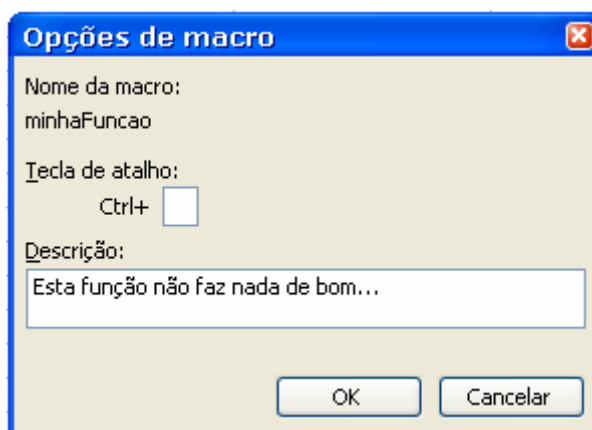


Figura 4-3

Ao acionar a função em sua planilha ela conterá a descrição acima:

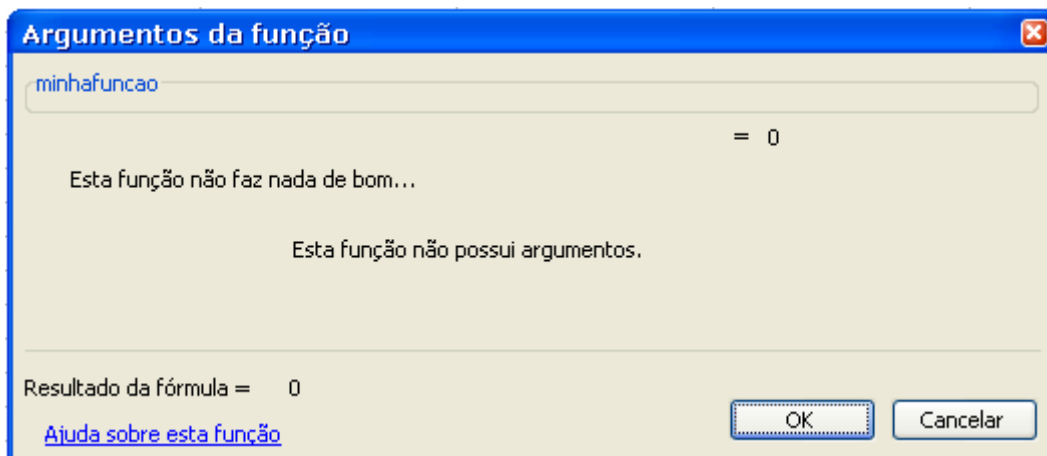


Figura 4-4

Uma outra forma de adicionar a descrição de sua função é através do VBE (Visual Basic Editor). Para acionar o VBE vá até Ferramentas → Macro → Editor do Visual Basic (ou pressione Alt+F11).

Clique em Exibir → Pesquisador de Objeto (ou pressione F2). No pesquisador de objeto, filtre os objetos pelo nome de seu projeto VBA:

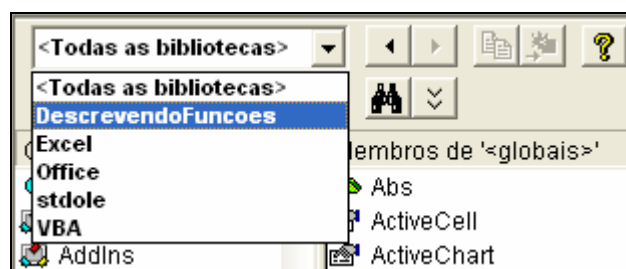


Figura 4-5

As informações de seu projeto serão filtradas e mostradas como segue. Clique no módulo que contém sua função, clique sobre a função e a descrição aparecerá logo abaixo na janela:

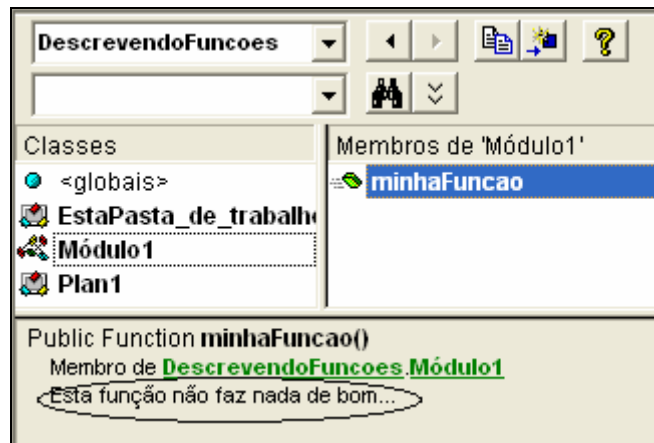


Figura 4-6

Para modificar a descrição da função, clique sobre ela com o botão direito do mouse, selecione a opção **Propriedades** e modifique a descrição conforme necessário:

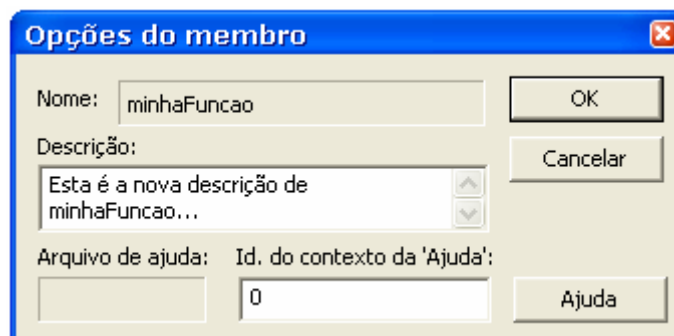


Figura 4-7

E a nova descrição é aceita em nossa função:

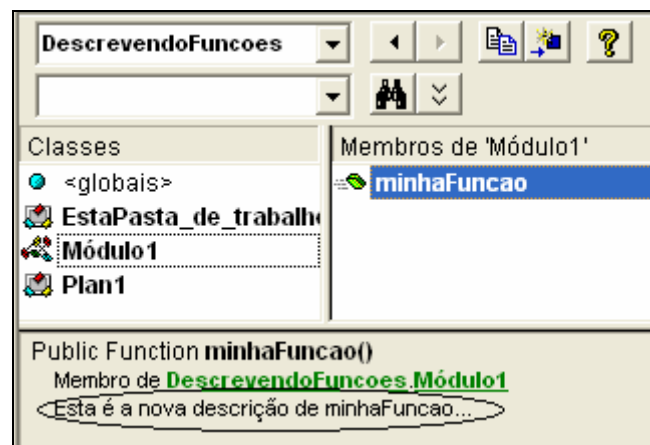


Figura 4-8

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

Você notará, contudo, que este método funciona bem quando ainda não definimos uma descrição para nossa função. Se uma descrição já tiver sido definida, o **Pesquisado de objetos** teimará em não modificar a descrição.

Um terceiro método requer a utilização de uma macro para modificar a descrição em tempo de execução.

```
Sub mudarDescricao()  
    Application.MacroOptions "minhaFuncao", _  
        Description:="Nova descrição via macro..."  
End Sub
```

Código 4-1

A sub-rotina acima utiliza o método MacroOptions para definir a nova descrição de nossa função. Este método é bastante útil se você possui muitas funções, pois é completamente inviável modificar cada opção manualmente. Desta forma, utilizamos um arquivo contendo informações individuais das funções e modificamos todas as funções de uma só vez. Ao rodar o código a descrição da função é modificada:

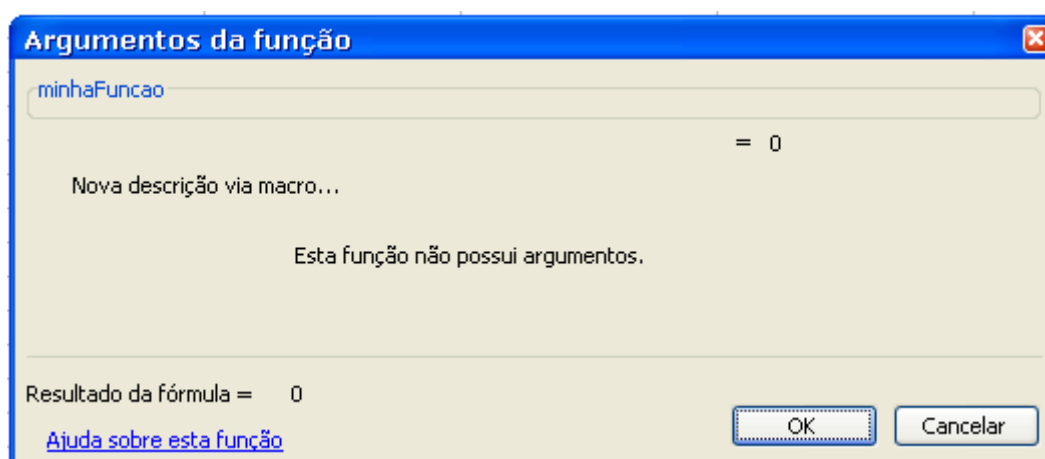


Figura 4-9

4.2. ADICIONANDO A FUNÇÃO A UMA CATEGORIA

O criar funções personalizadas, o Excel automaticamente as coloca sob a categoria de funções definidas pelo usuário.

A figura a seguir mostra uma função personalizada sob tal categoria:

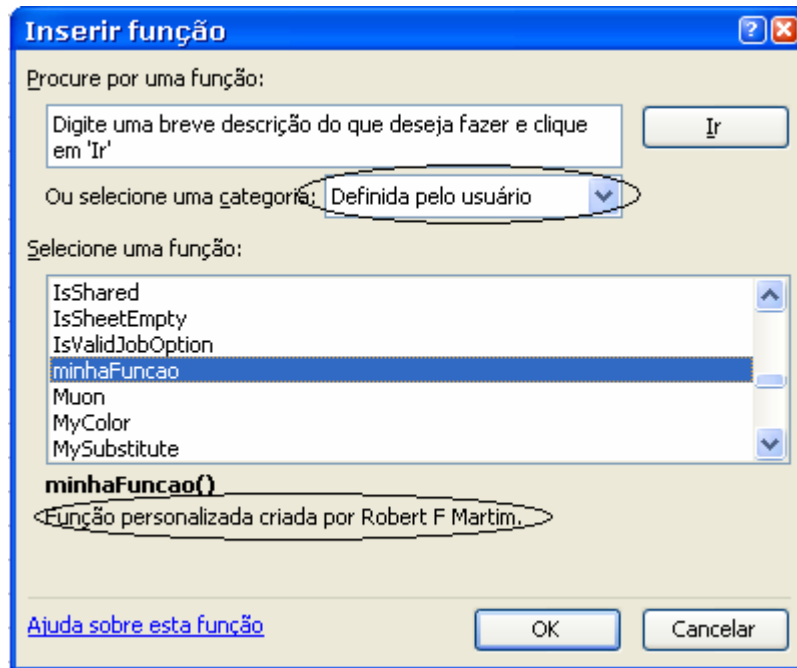


Figura 4-10

Porém, se você está criando uma função financeira, você pode desejar que ela seja categorizada juntamente com as outras funções financeiras do Excel.

Para resolver o problema de categorização nós nos voltamos ao método MacroOptions. Na verdade, este método possui vários argumentos (parâmetros) e um deles já utilizamos:

```
Private Sub Workbook_Open()  
    Application.MacroOptions "minhaFuncao", "Esta função não faz nada...", _  
        Category:=4  
  
    MsgBox "A função minhaFuncao foi inserida na categoria Estatística"  
  
End Sub
```

Código 4-2

Ao inserir o código no evento Open da pasta de trabalho a função definida no método será automaticamente categorizada na abertura da pasta de trabalho:

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

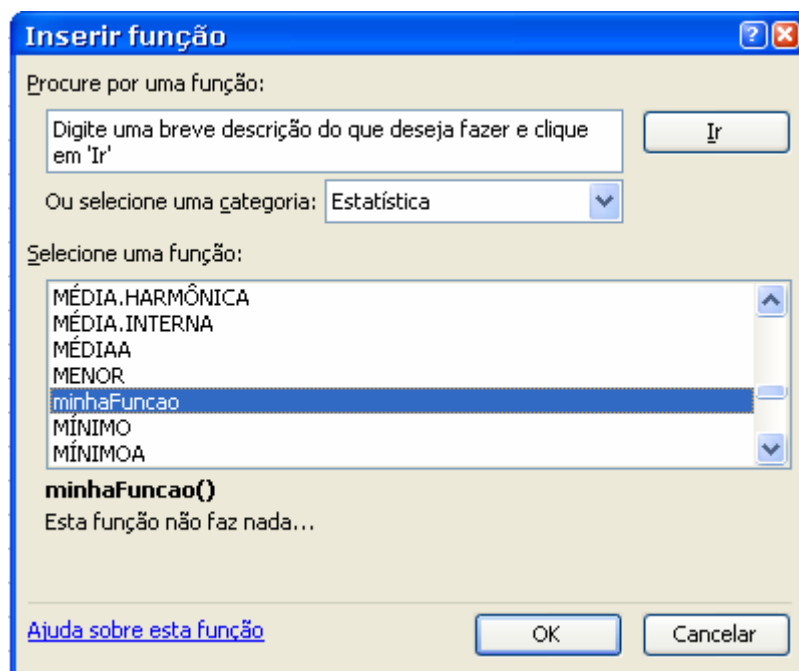


Figura 4-11

Ao abrir a caixa de inserção de função e escolher a categoria Estatística, basta rolar até a letra *m* para encontrar a *minhaFunção* devidamente categorizada.

A tabela a seguir mostra a lista das categorias das funções no Excel:

Valor Inteiro	Categoria
1	Financeira
2	Data & Hora
3	Matemática e Trigonometria
4	Estatística
5	Procura e Referência
6	Banco de Dados
7	Texto
8	Lógica
9	Informação
10	Comandos
11	Personalização
12	Controle de Macro
13	DDE/Externa
14	Definida pelo Usuário
15	Primeira categoria personalizada
16	Segunda categoria personalizada
17	Terceira categoria personalizada

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

18	Quarta categoria personalizada
19	Quinta categoria personalizada
20	Sexta categoria personalizada
21	Sétima categoria personalizada
22	Oitava categoria personalizada
23	Nona categoria personalizada
24	Décima categoria personalizada
25	Décima primeira categoria personalizada
26	Décima segunda categoria personalizada
27	Décima terceira categoria personalizada
28	Décima quarta categoria personalizada
29	Décima quinta categoria personalizada
30	Décima sexta categoria personalizada
31	Décima sétima categoria personalizada
32	Décima oitava categoria personalizada

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

5. DEFININDO OS ARGUMENTOS DE SUAS FUNÇÕES

Antes de iniciarmos a criação de funções personalizadas ainda precisamos conhecer os tipos de argumentos possíveis em uma função.

O que é um argumento? Argumento pode ser uma constante, uma linha de texto, uma ou mais variáveis, etc, que são passadas para a função manipular e resolver conforme a fórmula da função.

A função pode ter a fórmula do desvio padrão, por exemplo, e receber como argumento um “range” contendo diversas observações de uma variável qualquer. O receber tais valores, a função resolverá para o desvio padrão conforme a fórmula inserida na função.

Ao definir suas variáveis você precisa ter em mente o tipo de dado que será recebido. Por exemplo, se a função somente lida com número não fará sentido definir o argumento como String (texto).

Um erro comum, não só de quem cria, mas também de quem usa, em funções é tentar avaliar um argumento onde o tipo não corresponde ao tipo sendo avaliado pela função. Por exemplo, se você tenta efetuar um cálculo com um texto, a menos que o texto possa ser reconhecido e convertido para um valor numérico, a sua função retornará um erro.

Para compreender melhor cada um dos erros retornados, você pode ler o Ajuda do Excel que contém uma lista dos erros e o que eles significam e como resolvê-los. No Ajuda do Excel digite **Corrigir um erro** e uma lista dos erros com explicações sobre cada um e como corrigi-los será apresentada.

É importante que você leia estas explicações e saiba como encontrá-las, pois você certamente precisará ajudar usuários na correção de tais erros e sem compreender cada um ou ao menos saber onde encontrar a resposta, você estará em sérios apuros no suporte ao uso de suas funções (ou até mesmo na solução de erros em funções internas do Excel).

5.1. FUNÇÕES SEM ARGUMENTOS

Uma função sem argumento é aquela que não recebe valor algum de entrada. Ela apenas retorna um valor. A primeira vista, este tipo de função pode parecer estranha, pois, afinal, quem iria querer uma função sem argumento?

A verdade é que existem vários casos onde uma função sem argumento é extremamente útil. Veja por exemplo algumas funções que não recebem argumento no Excel como as funções Aleatório(), Hoje(), Falso(), Verdadeiro(), etc.

Autor: [Robert F Martim](#)

Criado em:

[1/5/2005](#)

Última edição:

[3/6/2005](#)

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Um exemplo de uma função personalizada sem argumento é uma função que retorna a localidade do sistema. Muitos usuários do Excel desconhecem uma funcionalidade onde se você mudar a configuração regional do sistema o Excel assume o sistema decimal do país selecionado. Cada país tem um código e o Excel retornará o código do país de acordo com o código de discagem internacional. Por exemplo, o código de discagem internacional para o Brasil é 55, para os EUA é 1, para a Alemanha é 49, etc.

Se você tem uma função que lhe informa a configuração regional a sua função possui tradução para outros idiomas, você pode disponibilizar a função no idioma selecionado na configuração regional do seu sistema:

```
Function códPaís() As Long
    códPaís = Application.International(xlCountrySetting)
End Function
```

Código 5-1

A função acima faz exatamente isso. O “argumento” da função é a configuração do país (xlCountrySetting). A função busca a configuração atual e retorna o número do país atualmente configurado em seu sistema:

	A1			
	A	B	C	D
1	55			
2				

Figura 5-1

Ao modificar a configuração regional para um outro país, a função automaticamente se reajusta:

	A1			
	A	B	C	D
1	380			
2				

Figura 5-2

No exemplo acima o país é a Ucrânia.

Para ver a função acima em funcionamento, você precisa modificar as configurações regionais de seu sistema. Para tanto vá até Iniciar → Painel de controle → Opções regionais e de idioma. Modifique o idioma padrão de Português (Brasil) para o de um outro país qualquer e a função automaticamente ajusta o valor calculado. Observe que não é necessário fechar e abrir o Excel. Assim que a configuração é modificada a função é reajustada.

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

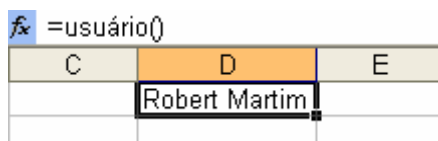
Contato: rm@earnconsultoria.com.br

Um outro exemplo clássico de uma função sem argumento é aquela que retornar o nome do usuário atual:

```
Function usuário()
    usuário = Application.UserName
End Function
```

Código 5-2

Ao inserir a função em sua planilha:



fx =usuário()		
C	D	E
	Robert Martim	

Figura 5-3

5.2. FUNÇÕES COM UM OU MAIS ARGUMENTOS

Uma função com um ou mais argumento é simplesmente isso: uma função com um ou mais argumentos que são definidos quando criamos a função. Diferente dos argumentos indefinidos (veja ParamArray mais abaixo no texto) aqui precisamos definir explicitamente os argumentos da função assim como o tipo de dado de cada argumento.

Supondo que você queira calcular o volume de um retângulo, você precisará da altura, largura e profundidade do retângulo. Os argumentos da função podem ser definidos como:

```
Function volumeRetângulo(altura As Double, largura As Double, _
    profundidade As Double) As Double
End Function
```

Código 5-3

O tipo de dado aceito por cada argumento é duplo (double). Double tem um ótimo escopo, pois engloba os tipos simples (Single), longo (Long), inteiro (Integer) entre outros, facilitando em muito a entrada de dados. Não obstante, se você entrar um valor String, por exemplo, a função retornará um erro, pois ela será incapaz de utilizar um tipo que não seja englobado por Double.

A função acima aceita argumentos duplos e retorna um valor duplo. Contudo a sua função pode receber argumentos de diferentes tipos e retorna um outro tipo qualquer. O tipo Variant é o mais global de todos, pois retornará qualquer tipo.

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

No exemplo abaixo, os argumentos são todos duplos e o resultado é uma matriz contendo duas raízes. A função é utilizada para resolver equações do segundo grau e utiliza a fórmula

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

```
Function segundoGrau(a As Double, b As Double, c As Double) As Variant

    Dim matriz(1)

    matriz(0) = (-b + Sqr(b ^ 2 - 4 * a * c)) / (2 * a)
    matriz(1) = (-b - Sqr(b ^ 2 - 4 * a * c)) / (2 * a)

    segundoGrau = WorksheetFunction.Transpose(matriz())

End Function
```

Código 5-4

Na linha que passa o resultado da matriz para a função utilizo a função de planilha (WorksheetFunction) "Transpose" (Transpor) para transpor o resultado que será em colunas para um resultado em linhas. Para os usuários não conhecem fórmulas matriciais, a função acima retorna uma matriz e precisa ser inserida com CTRL+SHIFT+ENTER:

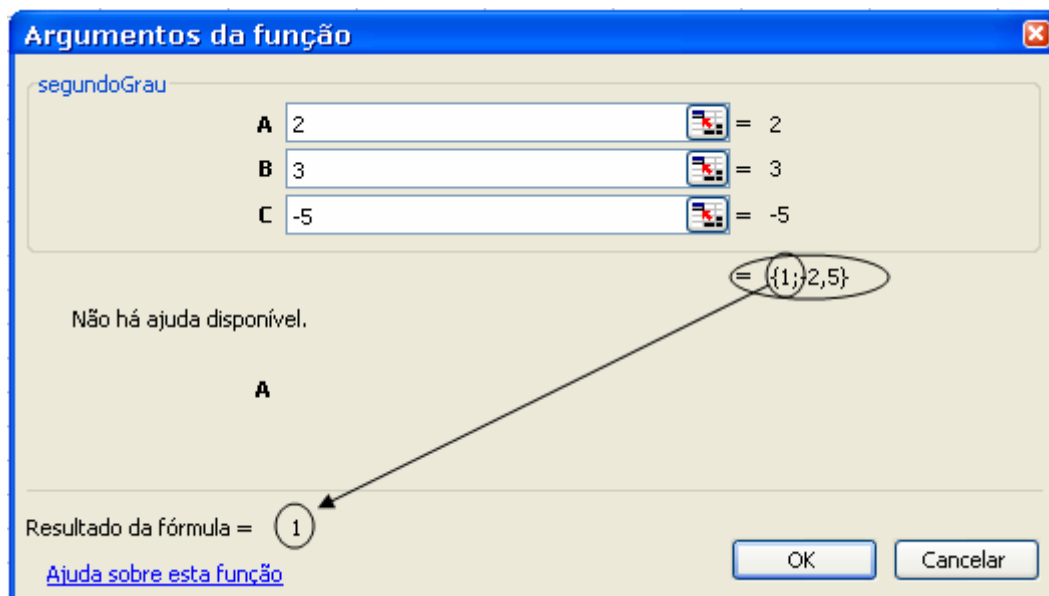


Figura 5-4

Observe que o resultado da função não é completo embora ela mostre a matriz completa {1;-2,5}. Para que ambos os resultados sejam mostrados será necessária a seleção de duas células na vertical (linhas) e pressionar CTRL+SHIFT+ENTER simultaneamente:

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

	A1	fx {=segundoGrau(2;3;-5)}			
	A	B	C	D	E
1	1				
2	-2,5				
3					

Figura 5-5

A função acima retorna as duas raízes da seguinte função $2x^2 + 3x - 5 = 0$. A função, no entanto, não resolve equações com raízes imaginárias.

5.3. FUNÇÕES COM ARGUMENTO OPCIONAL

Argumento opcional é aquele que o usuário não precisa inserir na função para que ela possa efetuar os cálculos. Um exemplo é a função interna VF (valor futuro). Dependendo do tipo de cálculo alguns argumentos são opcionais. Se o usuário não os inserir a função ainda será capaz de calcular o valor procurado pelo usuário.

Argumentos opcionais são substituídos por argumentos padrões dentro da própria função. Supondo que o cálculo requer uma decisão verdadeira ou falsa por parte do usuário. Se o usuário preferir por deixar o argumento em branco a função assumirá que o argumento é verdadeiro (ou falso dependendo de como o argumento opcional tenha sido definido).

Imagine uma função que retorna números aleatórios. Este tipo de função não necessita de argumento, porém, você deseja que os números aleatórios sejam positivos se o usuário digitar 0 (zero) e negativos se o usuário digitar 1 (um).

Não obstante, se o usuário pode optar por deixar o argumento em branco, precisamos decidir qual tipo de número aleatório será retornado.

O exemplo a seguir mostra como declarar um argumento opcional:

```
Function meuAleatório(Optional Tipo As Boolean = True) As Double

    Select Case Tipo
        Case True
            meuAleatório = Rnd
        Case False
            meuAleatório = Rnd * -1
    End Select

End Function
```

Código 5-5

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

O argumento opcional é definido como Boolean (verdadeiro ou falso). Se ele for verdadeiro, o número aleatório será positivo e se for falso o número aleatório será negativo:

A1		fx =meuAleatório(0)	
	A	B	C
1	-0,70903790	=meuAleatório(0)	
2	0,04535276	=meuAleatório()	
3	0,41403270	=meuAleatório(1)	
4			

Figura 5-6

Observe que o valor padrão para o argumento opcional é colocado diretamente na declaração do argumento. Uma outra forma de passar o valor padrão é utilizar uma das funções internas do VBA tais como IsMissing, IsEmpty, etc, para avaliar o argumento antes do cálculo ser iniciado:

No caso específico deste exemplo, se o valor não for digitado ele é considerado como zero. Desta forma quando o argumento não for inserido pelo usuário a função será avaliada para False. A depuração abaixo mostra isso ocorrendo:

```
Function meuAleatórioIsMissing(Optional Tipo As Boolean) As Double
    If IsMissing(Tipo) Then Tipo = True
    Tipo = Falso
    Select Case Tipo
        Case True
            meuAleatórioIsMissing = Rnd
        Case False
            meuAleatórioIsMissing = Rnd * -1
    End Select
End Function
```

Figura 5-7

O tipo continua sendo opcional e desta vez é deixado em branco. Ao iniciar a avaliação da função o argumento é passado como falso e expressão para avaliar o argumento que falta não tem efeito algum. Portanto, se você estiver lidando com argumentos booleanos opcionais é importante definir o caso do opcional em branco logo no argumento, caso contrário o efeito pode não ser o desejado.

O exemplo a seguir utiliza um argumento opcional do tipo String e a função também retorna um texto (String):

```

Function caminho(Optional strCaminho As String) As String

    If strCaminho = "" Then
        caminho = ThisWorkbook.Path
    Else:
        caminho = strCaminho
    End If

End Function

```

Código 5-6

Se nenhum argumento for inserido, isto é, se a String for vazia (""), então, a função retorna o caminho atual da pasta de trabalho:

	A2				
	A	B	C	D	
1	D:\Storage\Robert\Julio Battisti\Parcerias\E				
2	C:\				
3					

Figura 5-8

O caso em A1 é exatamente este. Ao deixar em branco o argumento a função retorna o caminho onde o arquivo está salvo.

O próximo exemplo mostra como utilizar a função IsMissing do VBA para avaliar se o argumento foi digitado ou não:

```

Function InteiroOpcional(Optional inteiro As Integer) As Variant

    If IsMissing(inteiro) Then inteiro = 0

    Select Case inteiro
        Case 0
            InteiroOpcional = "Inteiro não foi digitado."
        Case Else
            InteiroOpcional = inteiro
    End Select

End Function

```

Código 5-7

Se o argumento não for digitado definimos o valor inteiro como sendo zero. Caso o argumento digitado seja zero ele será tratado como se não tivesse sido digitado:

A1		fx =InteiroOpcional()
	A	B
1	Inteiro não foi digitado.	=InteiroOpcional()
2	250	=InteiroOpcional(250)
3	Inteiro não foi digitado.	=InteiroOpcional(0)
4		

Figura 5-9

5.4. FUNÇÕES NÚMERO INDEFINIDO DE ARGUMENTOS

Em algum ponto você certamente deve ter utilizado uma função que não possui um número definido de argumentos. Um exemplo de uma função deste tipo é a função SOMA:

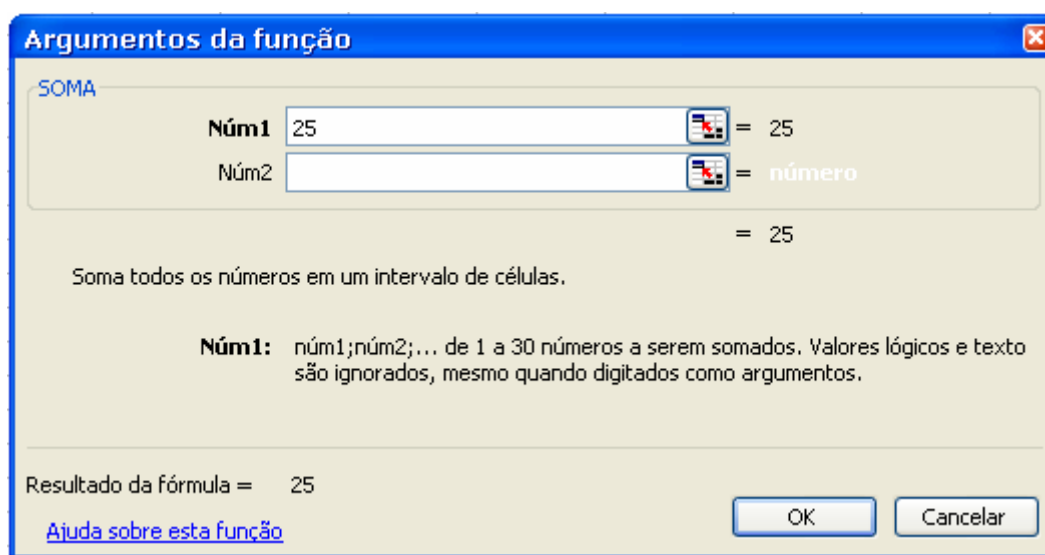


Figura 5-10

Por padrão dois argumentos são disponibilizados para o usuário conforme a figura acima. Ao clicar no segundo argumento a função automaticamente disponibiliza um terceiro argumento para o usuário:

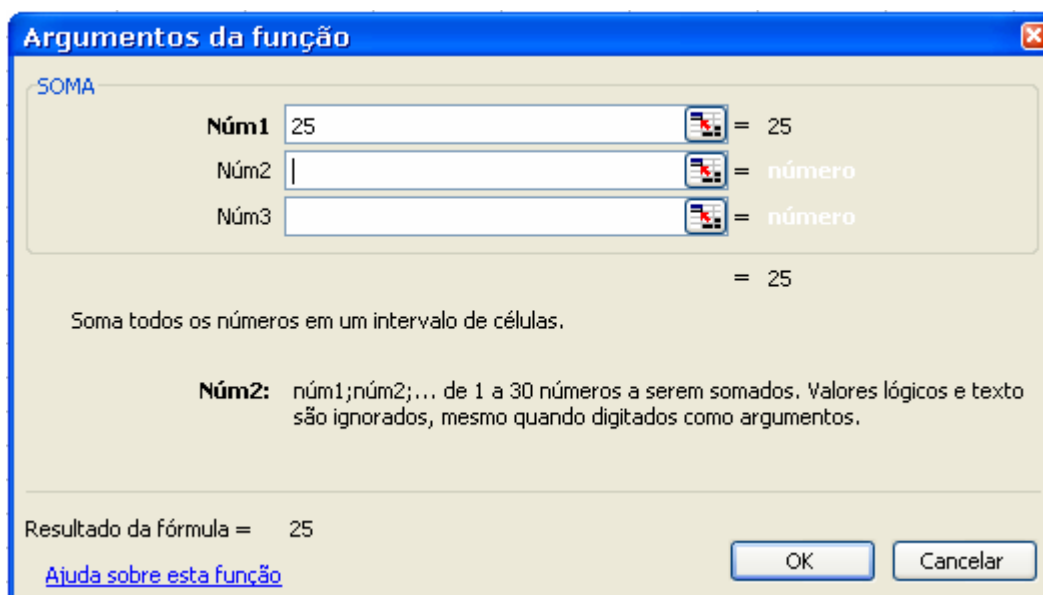


Figura 5-11

O processo se repetirá até que o número máximo de argumentos permitidos na função seja atingido:

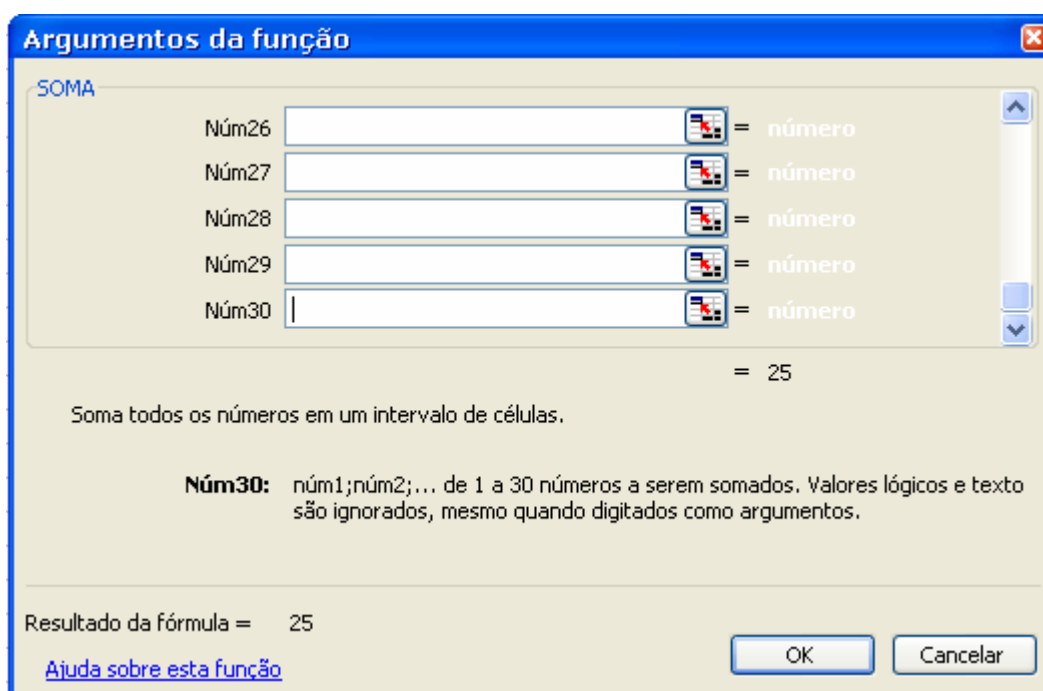


Figura 5-12

É importante observar que cada argumento pode receber mais de um argumento tornando o escopo da combinação do número de variáveis praticamente infinito.

Dependendo do tipo de trabalho que você faça é improvável que este tipo de função venha a ter alguma utilidade, contudo, é importante saber como utilizá-la (e conhecê-la) para uma eventualidade qualquer.

Autor: Robert F Martim

Criado em: 1/5/2005

Última edição: 3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Para criar um função com um número indefinido de argumento você deve declarar o argumento como um **ParamArray**.

ParamArray (Parameter Array) refere-se a uma matriz multi-dimensional. Ao tentar visualizar tal matriz, você deve estar se perguntando como identificar cada parâmetro dentro da matriz. A verdade é que isso não é necessário. A menos que você queira saber qual o valor do parâmetro 20 dentre, digamos, 200 parâmetros, não haverá necessidade de saber onde estão ou quem são os parâmetros

Contudo, se você realmente deseja saber, podemos utiliza a função Ubound para descobrir o limite superior da matriz ou simplesmente pedir para retornar o parâmetro x dentre todos os parâmetros recebidos como argumento da função. Observe o exemplo:

```
Function retornarParam(iParam As Integer, ParamArray parâmetros())
    retornarParam = parâmetros(iParam)
End Function
```

Código 5-8

O primeiro argumento da função refere-se a posição do parâmetro que desejamos retornar. O segundo argumento é a matriz contendo todos os parâmetros. Na função, nós adicionamos os seguintes valores: 25, 30, 45, 60, 59, 33 e 25. O valor que desejamos retornar encontra-se na quinta posição dentro da matriz:

A1		fx =retornarParam(5,25,30,45,60,59,33,25)				
	A	B	C	D	E	F
1	33					
2						

Figura 5-13

Observe que se contarmos cada uma das observações o valor retornado pela função encontra-se na sexta posição. Isso ocorre porque o limite inferior da matriz está na posição zero (0); portanto, a quinta posição é, na verdade, a sexta observação.

Um cenário que pode gerar confusão é demonstrado abaixo.

Supondo que voce tenha um conjunto de dados e ao invés de inserí-los individualmente como argumentos da função, você decide inserí-los em blocos. Ou seja, cada parâmetro passa a ser uma matriz de dados e cada matriz de dados é agrupada para formar a matriz que estará contida na matriz de parâmetros.

Para que o último parágrafo fique mais claro observe a figura:

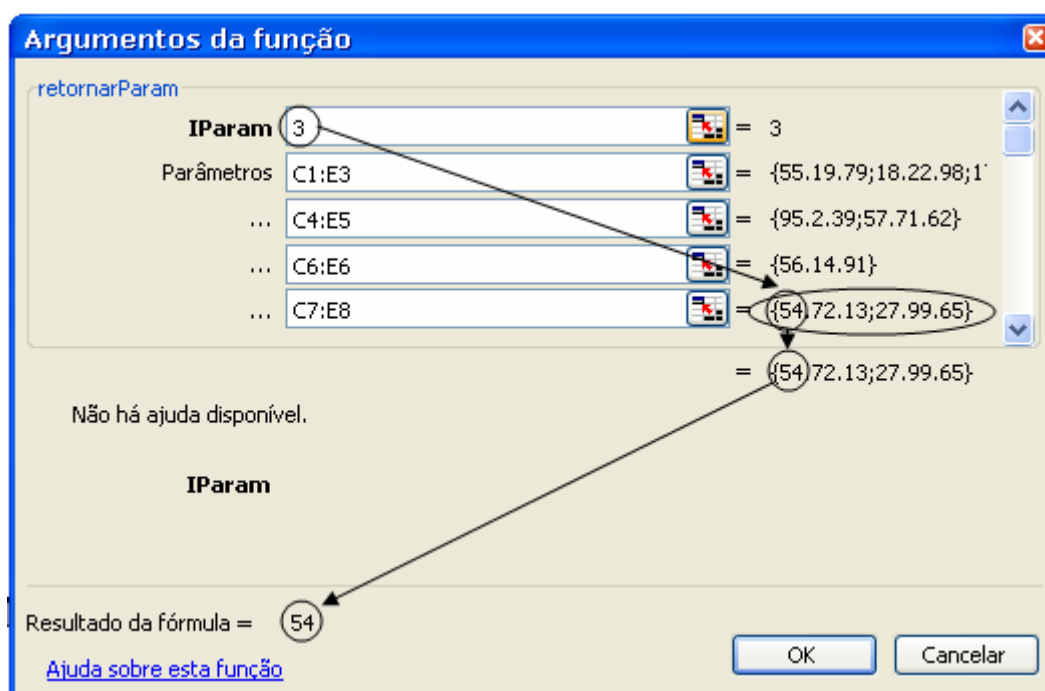


Figura 5-14

Cada parâmetro agora é representado por um conjunto de células que por sua vez formam uma matriz cada um (você reconhecerá matrizes pelas chaves { }). Ao escolhermos o terceiro parâmetro observe que estamos nos referindo ao ParamArray e não ao parâmetro dentro de cada grupo de observações. Portanto, o valor retornado é o primeiro dentre as observações do terceiro parâmetro.

E como resolver isso então? E se eu quero o primeiro item da segunda coluna do terceiro parâmetro, como encontrá-lo dentro desta matriz?

Neste caso, precisamos manter em mente que cada parâmetro é, na verdade, uma matriz. Desta forma precisamos referir a função anterior para levar isso em consideração. Neste caso, precisamos de mais dois argumentos para a função. Um dos argumentos indicará a linha e o outro indicará a coluna de onde o dado deve ser removido.

A função revista fica, portanto:

```
Function retornarParamExato(iParam As Integer, iLin, iCol, _
    ParamArray parâmetros())

    Dim matriz As Variant

    matriz = parâmetros(iParam)

    retornarParamExato = matriz(iLin, iCol)

End Function
```

Código 5-9

O primeiro passo é identificar o parâmetro que contém a matriz. Isso é feito pela linha **matriz = parâmetros(iParam)**

Como **matriz** é um tipo de dado **Variant** ela será capaz de receber todas as observações do parâmetro escolhido. Feito isso, tudo que precisamos é definir dentro da **matriz** em qual linha e em qual coluna a observação deve ser retirada. Isso é feito pela linha: **retornarParamExato = matriz(iLin, iCol)**

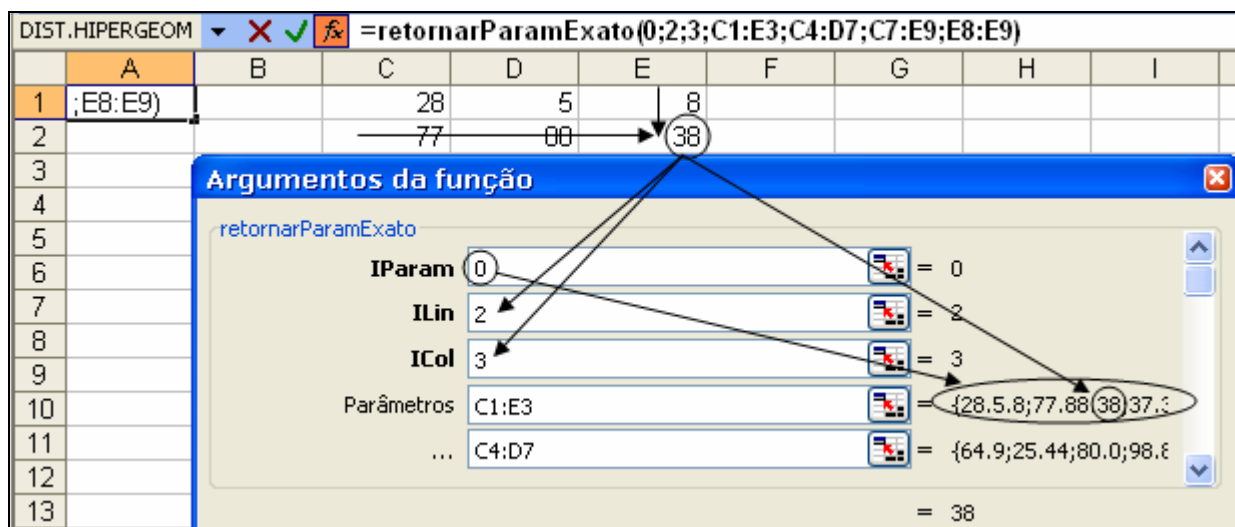


Figura 5-15

Você deve observar o seguinte na figura acima. Se o sistema decimal usado for o americano, isto é, onde o ponto separa o decimal e a vírgula separa o milhar, na matriz que aparecerá nos argumentos da função as linhas da matriz serão separadas por vírgula ao passo que colunas são separadas por ponto-e-vírgula.

No caso brasileiro as linhas são separadas por um ponto ao passo que as colunas são separadas por ponto-e-vírgula.

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Como um exemplo final, mostro como criar uma função simples para somar valores contidos nos parâmetros:

```
Function somarParam(ParamArray valores()) As Double
    Dim matriz As Variant

    For i = 0 To UBound(valores)
        matriz = valores(i)

        For Each valor In matriz
            somarParam = somarParam + valor
        Next
    Next

End Function
```

Código 5-10

A função acima assume que todos os valores são compostos de uma matriz. Se você inserir um valor único como um argumento a função retornará um erro.

6. CHECANDO A VALIDADE DOS ARGUMENTOS

Um outro aspecto que você precisará saber durante a criação de funções é como separar argumentos inválidos dentre vários argumentos selecionados pelo usuário.

Vamos supor que um dos argumentos é do tipo **Range**. Com este tipo de argumento o usuário poderá selecionar uma área da planilha para que os cálculos sejam efetuados. Se a área selecionada conter dados inválidos a função retornará um erro.

Contudo, usuários geralmente não querem ter o trabalho de separar valores válidos dos valores inválidos. Em muitos casos, os usuários nem sabem quais são os valores válidos e quais são inválidos. Eles simplesmente desejam selecionar uma área e deixar a função “se virar” para lhe dar o resultado esperado.

Em situações como estas, você precisará saber como contornar tal situação da melhor forma possível. Um problema clássico é misturar texto com valor numérico e esta é nossa primeira parada.

6.1. SEPARANDO VALORES NUMÉRICOS

Nossa primeira parada é criar uma função para testar se os valores selecionados pelo usuário são realmente numéricos. A função fará um loop por todos os valores selecionados e somente retornará verdadeiro se *todos* os valores forem realmente numéricos.

Para fazer temos duas opções: 1) utilizar a função IsNumeric do VBA ou 2) criar uma função para testar o valor ASCII dos argumentos selecionados. Observe que na segunda opção o ideal é separar a função que checa o valor individual da função que checa todos os valores.

No exemplo, estarei utilizando a função IsNumeric e deixarei a segunda como exercício para o leitor.

A função contém um loop que varre todos os valores selecionados pelo usuário. Se um valor não for numérico, a função retornar FALSO e sai do loop, pois não há mais necessidade de continuar se um valor for falso:

```

Function ÉNumérico(valores As Range) As Boolean
    ÉNumérico = True

    For Each valor In valores
        If Not IsNumeric(valor) Then
            ÉNumérico = False
            Exit For
        End If
    Next

End Function

```

Código 6-1

O função é iniciada assumindo que todos os valores são numéricos, portanto ela é definida como sendo verdadeira logo de início. O loop é iniciado e cada valor dentre os valores selecionados é checado. Se o valor não for numérico (If Not IsNumeric(valor)); então, a função deve retornar falso e sair do loop, pois não há necessidade de continuar tendo em vista que para a função ser verdadeira *todos* os valores devem ser numéricos:

	A12		=ÉNumérico(A1:B10)
	A	B	C
1	1	60	
2	2	61	
3	3	62	
4	4	63	
5	5 A		
6	6	65	
7	7 A		
8	8	67	
9	9	68	
10	10	69	
11	VERDADEIRO	FALSO	
12	FALSO		
13			

Figura 6-1

A figura acima mostra o intervalo A1:A10 sendo avaliado (VERDADEIRO), o intervalo B1:B10 (FALSO) e o intervalo A1:B10 (FALSO).

6.2. SEPARANDO VALORES VAZIOS

Visto o problema de valores numéricos, analisamos agora a situação de valores vazios. Não é toda situação que um valor vazio causa problemas, porém, há situações onde um valor vazio pode causar sérios problemas.

Imagine a situação onde você queira tirar a média de valores no intervalo A1:A10. Se você possui vários argumentos vazios no intervalo, ao utilizar a fórmula =MÉDIA(A1:A10) o Excel ignorará todos os espaços vazios e calculará somente a média dos valores.

	A	B	C
1	30	63	
2	18		
3		34	
4	93	99	
5		22	
6			
7	51,29	35,90	
8			

Figura 6-2

No exemplo acima, a fórmula em A7 utiliza a função MÉDIA para calcular a média dos dados no intervalo A1:B5. A fórmula em B7 utiliza a função personalizada a seguir:

```
Function minhaMédia(valores As Range) As Double

    For Each valor In valores
        minhaMédia = minhaMédia + valor
    Next

    N = valores.Count

    minhaMédia = minhaMédia / N

End Function
```

Código 6-2

Conceitualmente a função está correta, isto é, a média é realmente a soma de todas as observações dividida pelo número (N) de observações. Contudo, a função não leva em conta os argumentos vazios os quais não deveriam ser considerados como observações.

Uma solução é utilizar a função ISEEMPTY para avaliar se um dos argumentos está vazio. Se o argumento não estiver vazio, então, a função adicionar o valor ao cálculo. Caso contrário, o argumento é pulado e o próximo argumento é avaliado:

```

Function minhaMédia(valores As Range) As Double

    For Each valor In valores
        If Not IsEmpty(valor) Then
            minhaMédia = minhaMédia + valor
            N = N + 1
        End If
    Next
    minhaMédia = minhaMédia / N

End Function

```

Código 6-3

Desta vez, ao invés de utilizarmos a propriedade **Count** do objeto **Range** (valores), utilizamos um contador **N** para avaliar o número de observações não vazias no conjunto de dados selecionados. O resultado será igual ao da função MÉDIA:

	B7		f _x =minhaMédia(A1:B5)
	A	B	C
1	30	63	
2	18		
3		34	
4	93	99	
5		22	
6			
7	51,29	51,29	
8			

Figura 6-3

6.3. SEPARANDO VALORES NUMÉRICOS E VAZIOS SIMULTANEAMENTE

Se você já utilizou a função MÉDIA (ou SOMA) em situações onde existem observações em branco ou não numéricas, você deve ter notado que esta função simplesmente ignora estas observações e retorna o resultado como se nada estivesse afetando a função:

B7		=minhaMédia(A1:B5)		
	A	B	C	D
1		30	63	
2		18		
3	A		34	
4		93	99	
5			22	
6				
7	51,29	#VALOR!		
8				

Figura 6-4

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

A função personalizada não é capaz de lidar com ambas as situações simultaneamente. Nos exemplos anteriores vimos como tratar cada exceção separadamente. Neste tópico mostro como tratar ambas as situações simultaneamente.

A solução do problema é, na verdade, bem simples. O que precisamos é adicionar mais uma avaliação condicional lógica à nossa função. Neste caso, utilizaremos a condicional lógica **AND** (E) aninhada dentro da condição **IF** (SE).

A função deverá a seguinte condição: se o valor não for vazio e for numérico, então, calcular a média dos valores. Se você coloca a condição definida claramente, a solução do problema é bem simples:

```
Function minhaMédia2(valores As Range) As Double

    For Each valor In valores
        If Not IsEmpty(valor) And IsNumeric(valor) Then
            minhaMédia2 = minhaMédia2 + valor
            N = N + 1
        End If
    Next
    minhaMédia2 = minhaMédia2 / N

End Function
```

Código 6-4

Se as duas condições forem aceitas, então, o valor é somado ao valor anterior e N é contado. Finalmente, a nossa função revista retornará o valor correto para o intervalo selecionado pelo usuário mesmo quando ele contém espaços vazios ou letras:

	B7		fx =minhaMédia2(A1:B5)		
	A	B	C	D	
1	30	63			
2	18				
3	A	34			
4	93	99			
5		22			
6					
7	51,29	51,29			
8					

Figura 6-5

7. CRIANDO MINHA PRIMEIRA FUNÇÃO

Depois de passarmos pela introdução dos conceitos que você precisará para construir suas funções personalizadas, chegou a hora de colocar a prova aquilo que foi aprendido. Nos exercícios a seguir, mostrarei vários exemplos e a lógica da construção pode trás de cada uma das funções.

Os exemplos são simples para que você possa ir se acostumando com o pensamento lógico por trás de cada desenvolvimento.

7.1. FUNÇÕES DO TIPO BOOLEAN

O objetivo dos exercícios a seguir é construir funções do tipo Boolean que avaliem diferentes cenários. Os cenários são simples para que você possa ir se acostumando com o pensando por trás de cada função desenvolvida. Uma vez que você tenha compreendido cada uma das funções, tente criar cenários novos onde você tenha a necessidade de usar tais funções e crie as suas próprias.

7.1.1.Comparando valores

O objetivo desta função é comparar dois valores selecionados pelo usuário. Se os valores forem iguais a função retorna VERDADEIRO (TRUE) e se forem diferentes a função retorna FALSO (FALSE):

```
Function comparaValores(valor1 As Variant, valor2 As Variant) As Boolean
'   A função recebe como argumento dois valores variáveis
'   A função é definida como "False". Assumimos que os valores são
'   diferentes
    comparaValores = False

'   Se o valor1 for igual ao valor2, a função retornar "True"
    If valor1 = valor2 Then comparaValores = True

End Function
```

7.1.2. Definindo se um dado está presente em um conjunto de dados

O objetivo desta função é buscar um determinado valor em uma lista selecionada pelo usuário. Se o dado procurado for encontrado na lista a função retorna VERDADEIRO (TRUE) e se o dado não estiver na lista a função retorna FALSO (FALSE):

```
Function DadoPresente(Dados As Range, dadoProcurado As Variant) As Boolean
'   A função recebe como argumento dois valores um "Range" e outro "Variant"
'   A função é definida como "False". Assumimos que o dado não se encontra
'   na lista
    DadoPresente = False

'   Para cada dado na lista, comparar o dado com o dadoProcurado
    For Each dado In Dados
        Se forem iguais, então,
            If dado = dadoProcurado Then
                O dadoProcurado encontra-se na lista e a função retorna "True"...
                DadoPresente = True
                e podemos sair do loop
                Exit For
            End If
        Next
    End Function
```

7.1.3. Determinando se um texto está acima do limite de caracteres permitidos

O objetivo desta função é determinar se o texto inserido pelo usuário é mais longo do que o permitido. Se for, a função retorna VERDADEIRO (TRUE) e se não for a função retorna FALSO (FALSE):

```
Function comprimentoTexto(texto As String) As Boolean
'   A função recebe um argumento do tipo String
'   A função é definida como "False". Assumimos que o comprimento do texto
'   é inferior ao determinado
    comprimentoTexto = False

'   Se o comprimento do texto exceder 255 caracteres, o texto excedeu o limite
'   de caracteres permitido

    If Len(texto) > 255 Then comprimentoTexto = True

End Function
```

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

7.2. FUNÇÕES DO TIPO DOUBLE, LONG, SINGLE E INTEGER

O objetivo dos exercícios a seguir é construir funções dos tipos Double, Long, Single e Integer. Não darei um exemplo de cada, pois o conceito para cada uma é exatamente o mesmo, o que muda é a precisão dos dados. Veja a tabela contendo os tipos de dado para verificar a precisão de cada um.

7.2.1. Calculando o movimento percentual

O objetivo desta função é calcular o movimento percentual de dois valores. A função deve resolver para o movimento percentual discreto e contínuo:

```
Function Deltapct(VF As Double, VP As Double, nPer As Integer, _  
    Optional Tipo As Boolean = False) As Double  
  
    ' Tipo define o tipo de cálculo. Tipo "0" o cálculo é discreto  
    ' e Tipo "1" o cálculo é contínuo.  
    Select Case Tipo  
        Case False  
            ' Para cálculo discreto, use a fórmula  
            '  $\% = (VF/VP)^{(1/m)} - 1$   
            ' Onde VF= valor futuro; VP valor presente  
            Deltapct = (VF / VP) ^ (1 / nPer) - 1  
        Case True  
            ' Para cálculo contínuo, use a fórmula  
            '  $\% = \text{LN}(VF/VP) / m$   
            ' Onde VF= valor futuro; VP valor presente  
            Deltapct = Log(VF / VP) / nPer  
    End Select  
  
End Function
```

7.2.2. Calcular o número de células com determinado número de cores de fundo

O objetivo desta função é calcular o número total de células em uma seleção que possuem uma determinada cor de fundo:

```
Function contarCor(células As Range, cor As Integer) As Long
    Dim célula As Range

    ' "Células" é o conjunto de células selecionado pelo usuário
    ' "Cor" é o número do índice da cor

    ' Para cada "célula" no conjunto de "células"
    For Each célula In células
        ' Se a cor de fundo for igual a cor procurada pelo usuário
        If célula.Interior.ColorIndex = cor Then
            ' adicionar 1 a contagem de cores.
            contarCor = contarCor + 1
        End If
    Next
End Function
```

7.2.3. Determinando a cor de fundo

O objetivo desta função é determinar a cor de fundo de uma célula qualquer. No exemplo anterior, sem saber qual o índice da cor de fundo fica impraticável fazer o cálculo. A função a seguir mostra como determinar a cor de fundo de uma célula:

```
Function corFundo(célula As Range) As Integer
    corFundo = célula.Interior.ColorIndex
End Function
```

7.3. FUNÇÕES DO TIPO STRING

O objetivo dos exercícios a seguir é construir funções que retornam textos (Strings). O tipo de texto retornado pode variar. Por exemplo, você pode ter uma função que recebe como argumento um valor numérico e retorna um texto, como é o caso da função “Extenso” que escreve valores numéricos para texto, como R\$10,25 se tornar “dez reais e vinte e cinco centavos”. Ou, por exemplo, uma função que retorne o caminho de um arquivo e assim por diante.

7.3.1. Retornando o nome da cor de fundo da célula selecionada

O objetivo desta função é retornar o nome da cor de fundo de uma célula. Supondo que a célula tenha sido pintada de vermelho, ao entrar a fórmula em uma célula qualquer cujo argumento é a referência da célula pintada, o valor retornado pela função é o nome da cor de fundo:

```
Function nomeCor(número As Range) As String
    cor = número.Interior.ColorIndex

    Select Case cor
        Case 1
            nomeCor = "Preto"
        Case 9
            nomeCor = "Vermelho escuro"
        Case 3
            nomeCor = "Vermelho"
        Case 7
            nomeCor = "Rosa"
        Case 38
            nomeCor = "Rosa claro"
        Case 53
            nomeCor = "Marrom"
        Case 46
            nomeCor = "Laranja"
        Case 45
            nomeCor = "Laranja claro"
        Case 44
            nomeCor = "Ouro"
        Case 40
            nomeCor = "Marrom claro"
        Case Else
            nomeCor = "Esta cor não se encontra na lista"
    End Select
End Function
```

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

O resultado obtido na planilha será:

	B1		f _x =nomeCor(A1)
	A	B	C
1		Preto	
2		Vermelho escuro	
3		Vermelho	
4		Rosa	
5		Rosa claro	
6		Marrom	
7		Laranja	
8		Laranja claro	
9		Ouro	
10		Marrom claro	
11		Esta cor não se encontra na lista	

Figure 7-1

7.3.2. Determinado o nome do objeto pai

O objetivo desta função é introduzir a propriedade "Parent". A função deve retornar o nome do objeto pai dada uma referência qualquer. Por exemplo, se o usuário digita objPai(A1) e a referência retorna o nome do objeto pai (planilha) o nome retornado será o nome da planilha que contém a referência A1. A função a seguir retorna o nome do objeto pai de acordo com a seleção do usuário:

```
Function objPai(célula As Range, Optional Tipo As Boolean = False) As String
' Tipo "false" (0) retorna o nome da planilha
' Tipo "true" (1) retorna o nome da pasta de trabalho
Select Case Tipo
Case False
objPai = célula.Parent.Name
Case True
' Aqui, fazemos um "duplo" Parent, pois
' desejamos saber o nome do pai, do pai da
' célula, isto é, o nome da pasta de trabalho
objPai = célula.Parent.Parent.Name
End Select

End Function
```

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

O resultado:

	A2		fx	=objPai(A2;1)
	A	B	C	
1	Plan2			
2	Tópico 7.3.xls			
3				

Figura 7-1

7.4. OUTROS TIPOS

O objetivo dos exercícios a seguir é construir funções variadas de diferentes tipos para que o leitor tenha contato com um pequeno leque de possibilidades no tocante à criação de funções personalizadas.

7.4.1. Adicionando dias úteis a uma data qualquer

O objetivo desta função é adicionar certo número de dias úteis a uma data qualquer. Supondo que a data atual seja 01/01/2005 e você queira adicionar 5.766 dias úteis a esta data; então, qual a data futura? Observe que estamos falando de *dias úteis* e não apenas da soma de 5.766 dias.

Somar 5.766 dias a data atual é simples. Somar 5.766 dias úteis é outra história. Se você utiliza o Excel há bastante tempo, você saberá que existe uma função que faz exatamente isso, ou seja, é desnecessário criar tal função. O objetivo aqui é simular tal função para que você possa desenvolver o pensamento e a lógica da função.

A ideia de criar tal desafio partiu do tópico http://www.juliobattisti.com.br/forum /forum_posts.asp?TID=6048&PN=2 do fórum Júlio Battisti.

A função DIATRABALHO adicionará um número de dias úteis e também trabalhará com feriados. O nosso primeiro problema é apenas resolver para o número de dias úteis desconsiderando os feriados que possam existir.

Você pode utilizar `Option Explicit` para forçar a declaração das variáveis. Neste exemplo, utilizo `Option Explicit`, pois reutilizarei as variáveis na função mais avançada e é importante manter as variáveis em xeque:

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br


```
Function diasUteis(dataInicial As Date, Optional nDias As Long = 0) As Date
'   Se o usuário não definir o número de dias úteis nDias é igual a zero

    Dim dias          As Long
    Dim Data           As Long
    Dim dataFinal      As Long
    Dim dia            As Byte
    Dim feriado        As Variant
    Dim i              As Integer

    dataInicial = dataInicial + 1

'   Aqui, utilizamos um loop para avaliar o número de dias enquanto
'   ele for menor do que o número de dias definidos pelo usuário
    Do While dias < nDias
'       Dia é definido como o dia da semana (valor numérico do dia)
'       O novo dia é a dataInicial mais i (número de dias)
        dia = Weekday(dataInicial + i, vbMonday)
'       Seleciona os casos de 1 a 5 (segunda a sexta)
        Select Case dia
'           Se for realmente um dia da semana
            Case 1, 2, 3, 4, 5
                adicionar 1 a dias que será comparado com nDias no loop
                dias = dias + 1
'           Se dia for 6 ou 7 (sábado ou domingo)
            Case Else
                dias permanece como está.
                dias = dias
        End Select
'       "i" representa o contador de dias a ser adicionado a dataInicial
        i = i + 1
'       Apenas uma forma de separar i da dataFinal. Você pode escolher
'       uma forma diferente de proceder
        dataFinal = dataFinal + 1
    Loop
'   Define a funcao como sendo o número de dias (dataFinal) menos 1.
'   O valor é menos 1, pois adicionamos 1 a dataInicial logo no início.
    diasUteis = dataInicial + dataFinal - 1
End Function
```

Código 7-1

A lógica por trás da função é explicada na própria sub-rotina acima. Fazendo uma comparação com a função DIATRABALHO, obtemos os seguintes. A função acima não foi exaustivamente testada. O teste foi feito com um total de 1.000 observações aleatórias entre 0 e 15.000 dias. Em todos os casos estudados nenhum retornou uma soma errada para o número de dias úteis:

D2		fx =diasUteis(A2;B2)			
	A	B	C	D	E
1	Data Inicial	Somar dias úteis		Resultado	
2	1/1/2005	150		29/7/2005	=diasUteis(A2;B2)
3				29/7/2005	=DIATRABALHO(A2;B2)
4					

Figura 7-2

7.4.2. Determinando o número da semana dentro de um mês

O objetivo desta função é determinar o número da semana dentro de um mês. Por exemplo, se o mês termina em 31 em uma terça-feira e está é a quinta semana de outubro, então, 1 de novembro é quinta semana do mês anterior e 8 de novembro é a primeira semana de novembro.

Esta função não será repetida aqui. Ela fica como um desafio para o leitor. Uma sugestão encontra-se no arquivo que acompanha este módulo.

8. DESENVOLVENDO FUNÇÕES MAIS AVANÇADAS

Depois passarmos pela construção de algumas funções simples, passamos agora para funções um pouco mais complexas. A complexidade de cada função dependerá de seu conhecimento de VBA, conhecimento matemático, conhecimento de lógica e disposição para resolver problemas. A definição de avançada diz respeito à introdução de conceitos desconhecidos.

As questões abaixo seguem a mesma seqüência que o tópico anterior para facilitar uma comparação.

8.1. FUNÇÕES DO TIPO BOOLEAN

A seguir apresento mais exemplos de funções booleanas para estudo.

8.1.1. Verificando se uma data é válida

O objetivo desta função é determinar se o valor digitado em uma célula é uma data válida ou não. Se for uma data válida a função retorna “true” se não for válida retorna “false”:

```
Function dataVálida(data) As Boolean
    dataVálida = False
    If IsDate(data) Then dataVálida = True
End Function
```

O resultado:

B2		fx =dataVálida(A2)	
	A	B	C
1	1/1/2005	VERDADEIRO	
2	01/01/2005	VERDADEIRO	
3	20050101	FALSO	

Figura 8-1

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

8.1.2. Determinando se uma barra de comando está travada

O objetivo desta função é determinar se uma determinada barra de comando está travada ou não. Observe que embora as barras sejam traduzidas para o português, a barra retém o nome em inglês:

```
Function barraTravada(nomeBarra As String) As Boolean
    Dim cmdbar As CommandBar

    Set cmdbar = Application.CommandBars(nomeBarra)
    barraTravada = False

    If cmdbar.Protection = msoBarNoMove Then barraTravada = True

End Function
```

A figura abaixo mostra a situação onde a barra padrão não está travada (veja o oval) e o menu travado (sem o pontilhado):

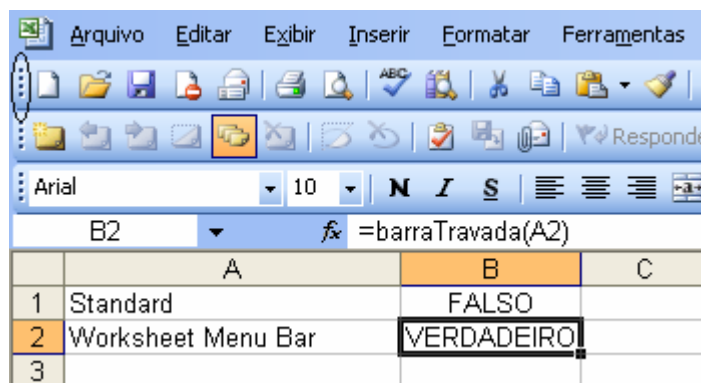


Figura 8-2

8.1.3. Determinando se o número de um cartão de crédito é válido

O objetivo desta função é determinar um número de cartão de crédito é válido ou não¹. Para validar cartões de crédito é utilizada matemática modular (MOD). Os dígitos são lidos de trás para frente e se a leitura for par o número é multiplicado por 2 e se for ímpar é multiplicado por 1. Por exemplo,

5	4	5	4	6	7	9	0	4	2	6	7	3	7	8	0
2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1
10	4	10	4	12	7	18	0	8	2	12	7	6	7	16	0
1+0	4	1+0	4	1+2	7	+18	0	8	2	1+2	7	6	7	1+6	0

Some a última linha e divida por dez. Se não houver resto, o número do cartão é válido. No caso acima, a soma é 69 e o resto é 9; portanto, o número do cartão é inválido.

¹ Para maiores informações veja o link <http://www.beachnet.com/~hstiles/cardtype.html>

O número do cartão acima imita um número MasterCard criado para este exemplo. Observe que para tornar o cartão válido, podemos modificar o número em dois pontos facilmente. Se modificarmos um dos zeros para 1 a soma será 70, tornando o número válido.

A função para resolver a questão fará o seguinte:

1. Extrair os números da direita para a esquerda, dígito por dígito.
2. Se o número cair em uma casa ímpar ele é multiplicado por 1 e se cair em uma casa par ele é multiplicado por 2 (dobrado)
3. Se a multiplicação por 2 resulta em um número de dois algarismos; então, somamos cada algarismo para gerar um novo dígito
4. Somamos cada dígito e no final dividimos a soma por 10. Se não houver resto, o número é válido

```
Function checarCartao(número As String) As Boolean
' Partimos do pressuposto que o cartão é inválido e o checamos
' para validá-lo
checarCartao = False
' Fazemos um loop por cada número do cartão
For i = 1 To Len(número)
' Extraímos cada dígito a partir do último
dígito = Mid(número, Len(número) + 1 - i, 1)
' Dividimos "i" por 2 para saber se há resto. Se não houver,
' multiplicar o dígito por 2
If i Mod 2 = 0 Then
    dígito = dígito * 2
Else
' Se houver, multiplicar dígito por 1
    dígito = dígito * 1
End If
' Se o comprimento do novo dígito por igual a 2
If Len(dígito) = 2 Then
' Somar cada dígito separadamente. Como o número é texto,
' precisamos convertê-lo para valor, caso contrário eles serão
' concatenados ao invés de somados
    dígito = Val(Mid(dígito, 1, 1)) + Val(Mid(dígito, 2, 1))
End If
' Somamos o dígito
soma = soma + dígito
Next
' Dividimos por 10. Se não houver resto, número do cartão é válido
If soma Mod 10 = 0 Then
    checarCartao = True
End If
End Function
```

Números do MasterCard possuem 16 dígitos. Como o Excel somente lida com 15 dígitos significantes, o número é passado como texto. Observe que se você converter o texto para valor

Autor: Robert F Martim

Criado em: 1/5/2005

Última edição: 3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

numérico o cartão apresentado em A2 será inválido, pois o último dígito será tratado como zero e será exatamente igual ao número do cartão em A1.

A validação do cartão:

	B2	fx =checarCartao(A2)	
	A	B	C
1	5454679042673780	FALSO	
2	5454679042673781	VERDADEIRO	
3	5454679142673780	VERDADEIRO	

Figura 8-3

8.2. FUNÇÕES DO TIPO DOUBLE

A seguir apresento algumas funções que retornam valores double, single, long e integer. Embora não dê exemplo de cada um separadamente, os cálculos são os mesmos modificando apenas a precisão de cada um dos tipos listados acima.

8.2.1. Calculando a área, perímetro e diagonal de um quadrado.

Se chamarmos o lado de um quadrado de L , a área do quadrado será L^2 , pois a área é representada pela multiplicação do lado horizontal pelo lado vertical, como ambos os lados são iguais, temos L^2 . Por outro lado, para calcular o perímetro de um quadrado multiplicamos todos os lados. Como cada lado é igual, podemos definir o perímetro como $4L$. Já a diagonal é definida como a raiz da soma de $RAIZ(L^2 + L^2)$.

A função deve calcular a área, o perímetro ou a diagonal de um quadrado de acordo com a seleção do usuário:

```
Function quadrado(Lado As Double, Optional Tipo As Integer = 0) As Double

    Select Case Tipo
        Case 0
            quadrado = Lado ^ 2
        Case 1
            quadrado = 4 * Lado
        Case 2
            quadrado = Lado * Sqr(2)
        Case Else
            quadrado = Lado ^ 2
    End Select

End Function
```

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

A função recebe um argumento opcional que define o tipo que deve ser calculado (área, perímetro ou diagonal). Por padrão, o valor é zero o qual calcula a área. Caso o usuário coloque qualquer valor que não se encontre na lista o valor calculado é para a área do quadrado:

C2		fx =quadrado(B2;A2)		
	A	B	C	D
1	Tipo	Lado	Resultado	
2	0	2	4,00	
3	1	2	8,00	
4	2	2	2,83	
5	3	2	4,00	
6		2	4,00	

Figura 8-4

8.2.2. Calculando a área de um quadrado com uma função de seu lado, perímetro ou diagonal.

Este exemplo é bem similar ao anterior, mas aqui estamos assumindo que você sabe o lado, perímetro e diagonal do quadrado e deseja calcular a área de tal quadrado. A área de um quadrado como função de seu lado é simplesmente L^2 . A área como uma função do perímetro é dada por $P^2/16$ e como uma função de sua diagonal a área é dada por $D^2/2$:

```
Function ÁreaQuadrado(medida As Double, _
    Optional Tipo As Integer = 0) As Double

    Select Case Tipo
        Case 0
            ÁreaQuadrado = medida ^ 2
        Case 1
            ÁreaQuadrado = medida ^ 2 / 16
        Case 2
            ÁreaQuadrado = medida ^ 2 / 2
        Case Else
            ÁreaQuadrado = medida ^ 2
    End Select

End Function
```

D4		fx =ÁreaQuadrado(C4;A4)			
	A	B	C	D	E
1	Tipo	Tipo medida	Medida	Área	
2	0	Lado	10,00	100	
3	1	Perímetro	40,00	100	
4	2	Diagonal	14,14	100	
5	3	Não definido	10,00	100	
6					

Figura 8-5

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

8.3. FUNÇÕES DO TIPO STRING

A seguir apresento algumas funções que retornam texto como resultado de cálculos. As funções a seguir são bastante úteis e certamente poderão ser adaptadas ao seu trabalho.

8.3.1. Extraindo um elemento dentro de um texto qualquer

O objetivo desta função é extrair um elemento de um texto qualquer. Na versão do Excel a partir do Excel XP, existe uma função chamada Split para fazer exatamente isso. O exercício aqui servirá para aqueles com versões anteriores ao Excel XP, pois tal função poderá ser utilizada neste contexto.

Apresentarei três formas diferentes de se fazer isso. Uma simplesmente remove o elemento do texto. A segunda cria uma matriz contendo os elementos e o elemento “n” é removido. Na terceira, a função é uma **Variant** e retorna uma matriz contendo todos os elementos do texto.

Para esta primeira parte mostrarei como extrair o elemento utilizando os dois primeiros métodos.

```
Function extrairTexto(texto As String, separador As String, _
    nTexto As Integer) As String

'   Posição inicial do cursor no texto de onde será extraído
'   o elemento
PosIni = 1
'   Concatemos o texto com o separador. Assim haverá um separador
'   no final do texto. Isso evitará erro no loop quando utilizarmos
'   a função de planilha "Find"
texto = texto & separador

'   Fazer um loop até o elemento que se deseja extrair
For i = 1 To nTexto
'       Se i for igual ao número do elemento a ser extraído
'       If i = nTexto Then
'           Definir a posição final do cursor e...
posFin = WorksheetFunction.Find(separador, texto, PosIni)
'           Saia do "For"
Exit For
'       End If
'       Definir a posição
Pos = WorksheetFunction.Find(separador, texto, PosIni) + 1
'       Redefinir a posição inicial (PosIni)
PosIni = Pos
Next
'   Extraí o texto utilizando a função MID. O extração do texto é
'   iniciado em PosIni e o comprimento do texto é a diferença
'   entre a posição final (PosFin) e posição inicial (PosIni)
extrairTexto = Trim(Mid(texto, PosIni, posFin - PosIni))

End Function
```

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

Aplicando a função na planilha:

B6 fx =extrairTexto(A6;"-";1)			
	A	B	C
1	Robert Friedrich Martim	Robert	
2	Robert Friedrich Martim	Friedrick	
3	Robert Friedrich Martim	Martim	
4			
5	2458-3654-7856	3654	
6	22452365-7715	22452365	
7	21+4536+1245-	1245-	

Figura 8-6

8.3.2.Extraindo um elemento dentro de um texto qualquer através de uma matriz

A função anterior mostra como retornar o valor a partir de um loop e uma final remoção do texto desejado através da diferença da posição final e inicial.

O exemplo a seguir segue basicamente a mesma linha de raciocínio; porém, aqui o resultado é colocado em uma matriz durante o loop e o texto é extraído de acordo com sua posição dentro da matriz.

A repetição da função tem por objetivo introduzir ao leitor o conceito de redimensionamento de uma variável durante a execução do código e manipulação de uma matriz.

Autor: Robert F Martim

Criado em:

1/5/2005

Publicado: www.juliobattisti.com.br

Última edição:

3/6/2005

Contato: rm@earnconsultoria.com.br

```

Function simulaSplit(texto As String, separador As String, _
    nTexto As Integer) As String
'   Dimensiona a matriz que receberá o texto quebrado por
'   elementos
    Dim matriz()

'   Calcula o número de palavras no texto
    n = Len(texto) - Len(WorksheetFunction.Substitute(texto, separador, ""))

'   Redimensiona a matriz para o número de palavras no texto
    ReDim matriz(n)

'   Defini a posição inicial do cursor
    PosIni = 1

'   Concatemos o texto com o separador. Assim haverá um separador
'   no final do texto. Isso evitará erro no loop quando utilizarmos
'   a função de planilha "Find"
    texto = texto & separador

'   Para cada palavra no texto
    For i = 0 To n
'       Encontra a posição final de cada elemento no texto
        posFin = WorksheetFunction.Find(separador, texto, PosIni) + 1
'       Remove o elemento a partir da posição inicial onde o
'       comprimento do texto é a diferença entre a posição final e inicial
        elemento = Trim(Mid(texto, PosIni, posFin - PosIni))
'       Armazena o elemento na primeira posição da matriz
        matriz(i) = Trim(elemento)
'       Defini a nova posição inicial como sendo a posição final do
'       primeiro elemento extraído do texto
        PosIni = posFin
    Next
'   Como o primeiro elemento da matriz é zero, extraímos o elemento
'   (nTexto - 1) de dentro da matriz. Isto é, se o usuário deseja o
'   elemento 4, a posição deste elemento na matriz é 3 (4-1)
    simulaSplit = matriz(nTexto - 1)

End Function

```

Aplicando a função na planilha:

B1	fx =simulaSplit(A1;" ";4)		
	A	B	C
1	Neste segundo, embora bastante similar	bastante	
2			

Figura 8-7

8.4. OUTROS TIPOS

8.4.1. Adicionando dias úteis a uma data qualquer contabilizando os feriados

Neste segundo, embora bastante similar ao exemplo 7.4.1, desta vez o que desejamos fazer é avaliar os feriados também.

Embora possa parecer simples, ao adicionarmos mais este elemento a nossa função precisamos nos atentar para a natureza dos feriados. Se um feriado cai em um dia útil, isto é, entre segunda e sexta-feira inclusive, então, o feriado deve ser tratado como um sábado ou domingo. Contudo, se ele cai em um final de semana, não faz a menor diferença.

A função revista não considera sábado como dia útil; portanto, para aqueles que trabalham sábados e precisam contar este dia como dia útil, a função precisará ser revista.

Um ponto que dever ser observado é que funções personalizadas não possuem o mesmo desempenho que funções internas do Excel ou funções em suplemento, portanto, esta função será ligeiramente mais lenta que a função DIATRABALHO para acréscimo muito elevado de dias úteis. Para acréscimo de 365, por exemplo, o recálculo é bem comparável com uma diferença de milésimos de segundos.

Antes de apresentar a função com os comentários, faço uma digressão abaixo para que o leitor acompanhe o raciocínio por trás da eventual função.

Observando o exemplo anterior, vemos a linha onde a data seguinte é criada quando adicionamos o contador **i** a **dataInicial**. Como sabemos, o feriado nada mais é do que a data atual, isto é, **dataInicial + i**. Contudo, isto somente será verdadeiro se a soma realmente for igual a data do feriado selecionado. Se não for, então, a data não é um feriado.

Se a condição acima for verdadeira, isto é, se o feriado for igual a uma data atual, então, esta data deve ser tratada como um final de semana. Se ela será tratada como sábado ou domingo, não importa.

Feita comparação entre feriado e data atual precisamos saber o dia da semana referente a tal feriado. Se o dia atual é um final de semana, então, o feriado fica como está (é tratado como final de semana mesmo), mas se cair em um dia útil (entre segunda e sexta-feira); então, ele deve ser pulado, pois não é dia útil.

Feitas estas observações, podemos partir para o código que solucionará o problema. Sugiro que o leitor tente resolver o problema antes de olhar o código.

Autor: [Robert F Martim](#)

Criado em:

[1/5/2005](#)

Última edição:

[3/6/2005](#)

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

```

Function diasUteis(dataInicial As Date, Optional nDias As Long = 0, _
    Optional Feriados As Variant = 0) As Date
    Application.Volatile True

    Dim dias          As Long
    Dim Data          As Long
    Dim dataFinal     As Long
    Dim dia           As Byte
    Dim feriado       As Variant
    Dim i             As Integer

    dataInicial = dataInicial + 1
    ' Continua o loop enquanto dias < nDias
    Do While dias < nDias
        ' Adiciona "i" a dataInicial em cada loop e guarda o valor na
        ' variável "Data"
        Data = dataInicial + i
        ' Para cada feriado na lista de feriados
        For Each feriado In Feriados
            ' Se data do feriado for igual a data atual (Data), então
            If CLng(feriado) = Data Then
                ' verificar que dia da semana ele é.
                dia = Weekday(feriado, vbMonday)
                Select Case dia
                    ' Caso o feriado seja um dia da semana (segunda a sexta)
                    Case 1, 2, 3, 4, 5
                        ' Considerar como um dia do final de semana (7)
                        dia = 7
                End Select
                ' Sair do loop dos feriados
                Exit For
            Else:
                ' Se a data do feriado nao for igual a data atual (Data)
                ' avaliar o dia da data atual (Data)
                dia = Weekday(Data, vbMonday)
            End If
        Next
        ' Selecionar o dia da semana
        Select Case dia
            ' Caso seja um final de semana (6 e 7); a variável "dias"
            ' permanece como está.
            Case 6, 7
                dias = dias
        End Select
    Loop
End Function

```

```

Case Else
    Caso seja um dia útil (qualquer outro caso, exceto 6 e 7)
    adicionamos 1 a variável dias
        dias = dias + 1
End Select
Ao contador de dias adicionamos 1
i = i + 1
Ao da data final adicionamos 1 (veja exemplo 7.4.1 para comentário)
dataFinal = dataFinal + 1
Loop
O número de dias úteis é adicionado a dataInicial e subtraímos 1,
como no exemplo anterior
diasUteis = dataInicial + dataFinal - 1

End Function

```

Código 8-1

Colocando a função ao teste:

D2		fx =diasUteis(A2;B2;A4:A20)			
	A	B	C	D	E
1	Data Inicial	Somar dias úteis		Resultado	
2	1/1/2005	250		30/12/2005	=diasUteis(A2;B2;A4:A20)
3				30/12/2005	=DIATRABALHO(A2;B2;A4:A20)
4	1/1/2005	Confraternização Universal			
5	5/2/2005	Carnaval			
6	6/2/2005	Carnaval			
7	7/2/2005	Carnaval			
8	8/2/2005	Carnaval			
9	9/2/2005	Cinzas			
10	25/3/2005	Paixão de Cristo			
11	27/3/2005	Páscoa			
12	21/4/2005	Tiradentes			
13	1/5/2005	Dia do Trabalho			
14	26/5/2005	Corpus Christi			
15	7/9/2005	Independência			
16	12/10/2005	Nossa Senhora de Aparecida			
17	1/11/2005	Finados			
18	15/11/2005	Proclamação da República			
19	25/12/2005	Natal			
20	31/12/2005	Véspera de Ano Novo			

Figura 8-8

Adicionando 250 dias úteis nos leva para 30 de dezembro de 2005. Acrescentando 8960 dias úteis e considerando apenas os feriados do primeiro ano:

B2		fx 8960			
	A	B	C	D	E
1	Data Inicial	Somar dias úteis		Resultado	
2	1/1/2005	8960		20/5/2039	=diasUteis(A2;B2;A4:A20)
3				20/5/2039	=DIATRABALHO(A2;B2;A4:A20)

Figura 8-9

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br

9. EXERCÍCIOS PARA DESENVOLVIMENTO DE FUNÇÕES

A seguir apresento uma série de exercícios para que o leitor possa colocar em prática o que foi aprendido até o momento. Os exercícios visam apenas o desenvolvimento da função. Fica a cargo do leitor utilizar as outras ferramentas aqui introduzidas, tais como descrição, categorização, etc, para melhor o visual geral das função.

Tenha em mente que as respostas dos exercícios são apenas guias para a solução dos problemas apresentados. As respostas não são para ser vistas como a verdade universal da melhor forma de resolver os problemas propostas. Para que o leitor realmente aprenda, ele precisará desafiar as respostas e colocá-las ao teste. O leitor precisa procurar não somente por bugs nas respostas, mas também por outras formas de resolver o problema apresentado.

É neste tópico que você mostrará a sua capacidade de raciocínio lógico além de realmente verificar se o que foi ensinado foi aprendido. Procure não olhar as respostas antes de tentar exaustivamente por uma solução.

9.1. FUNÇÕES DO TIPO BOOLEAN

Os problemas apresentados abaixo devem retornar VERDADEIRO (TRUE) ou FALSO (FALSE). Alguns problemas podem apresentar mais de uma solução e a sua solução pode não ser exatamente igual as soluções apresentas. Compare as suas soluções com as apresentadas e tente aproveitar o que há de melhor nas duas funções.

Você deverá criar uma função para:

- a) Avaliar se um arquivo Excel existe em uma pasta qualquer do seu HD.
- b) Determinar se uma planilha existe na pasta de trabalho atual.
- c) Determinar se uma célula possui uma formatação qualquer (por exemplo, se a cor de fundo é azul, ou se o texto é itálico, etc)²
- d) Determinar se dois textos são exatamente iguais

² A solução para este problema apresenta diferentes cenários

9.2. FUNÇÕES DO TIPO DOUBLE

Para este exercício, você deverá criar uma função para:

- a) Avaliar a seguinte fórmula $\frac{1}{a^n}$
- b) Determinar o resultado da seguinte fórmula: $P\left(1 + \frac{j}{m}\right)^{mt}$
- c) Determinar a seguinte fórmula: $10\log\left(\frac{I}{I_0}\right)$
- d) Encontrar o resultado da seguinte fórmula³: $\frac{\sum_{i=1}^n X_i}{N}$

9.3. FUNÇÕES DO TIPO STRING

Para este exercício, você deverá criar uma função para:

- a) Retornar o endereço da pasta de trabalho atual. Crie uma função com e outra sem argumento.
- b) Retornar o nome da pasta de trabalho dada a referência de uma planilha qualquer
- c) Retornar o nome da pasta de trabalho e planilha de uma célula qualquer
- d) Remover um elemento de um texto qualquer. Por exemplo, se o usuário digitar em A1 o texto “este é o texto que desejo remover” e em B1 digitar =removerTexto(A1;4) o valor retornado pela função deve ser “texto”, pois ele é o quarto elemento do texto em A1.

9.4. OUTROS TIPOS

Esta parte é dedicada a exercícios gerais de criação de funções no Excel. Alguns exemplos podem ser um pouco mais complexos do que os desenvolvidos até o momento. Contudo, eles servirão para te preparar para funções mais complexas do segundo módulo desta série.

Para este exercício, você deverá criar uma função para:

³ Esta é a fórmula da média.

- a) Converter um valor texto (2005-05-30 ou 200505305) em uma data tipo 30/05/2005. Se desejar adicione o tipo de separador, caso ele exista.
- b) Contar um determinado caractere em um texto qualquer. Por exemplo, contar a letra “t” em “texto” onde o resultado será igual a 2.
- c) Converter o caractere par ou ímpar em maiúsculo. O usuário deve escolher se o caractere a ser convertido para maiúsculo é par ou ímpar. Por exemplo, se a fórmula for parMaiúsculo(A1;1) e o texto em A1 for “texto”; então o resultado será TeXtO. Se a fórmula for parMaiúsculo(A1;2) o resultado será tExTo.
- d) Para passar os números de 0 a 10 para extenso.

10. SOBRE O AUTOR

FORMAÇÃO ACADÊMICA:

- Formado e Pós-Graduado em Finanças pela Universidade de Londres, Reino Unido
- Membro da Sociedade Brasileira de Econometria

LINGUAGENS DE PROGRAMAÇÃO E PLATAFORMAS:

- Visual Basic, Calculadores Programáveis Casio e Sharp
- BDs: MS Access and Lotus Approach
- Plataformas: Windows NT, 2000, XP, Linux Red Hat

EXPERIÊNCIA PROFISSIONAL

out02-abr04 **FAIRCOURT CAPITAL LIMITED (REINO UNIDO)**

- ***Diretor TI***

fev96-maio02 **MELVALE GROUP (REINO UNIDO)**

- ***Gerente de Exportação para a África Ocidental***
- ***Gerente de TI***

OUTRAS ESPECIALIZAÇÕES

- Inspeção e regulamentações Nigerianas para importação e exportação (Nigerian-British Chamber of Commerce & Cotecna International)
- Procedimentos de exportação no Reino Unido (The Institute of Export, Reino Unido)
- ICC 500 e Incoterms (The Institute of Export, Reino Unido)

OUTRAS ATIVIDADES

Fornecer suporte *pro bono* em TI à entidade de caridade Nigeriana NIDOE (Nigerians in Diaspora Organisation Europe) desde 2001. Participou ativamente na organização da conferência sobre Boa Governança e Responsabilidade Fiscal promovida pelo ONG em Abuja, Nigéria, em Novembro 2003. Foi um dos principais colaboradores na elaboração do relatório final sobre a conferência entregue a presidência da República Nigeriana em maio de 2004.

Autor do livro Access e VBA na Modelagem Financeira: Uma abordagem prática. Editora Axcel Books, 2005. www.axcel.com.br

Colaborador ativo do fórum Access Avançado do site www.juliobattisti.com.br, onde divide seu conhecimento e experiência com outros membros do espaço.

Colunista dos sites www.linhadecodigo.com.br e www.ativoaccess.com.br

Autor: Robert F Martim

Criado em:

1/5/2005

Última edição:

3/6/2005

Publicado: www.juliobattisti.com.br

Contato: rm@earnconsultoria.com.br