Estratégia Concursos

Aula 06

Banco do Brasil (Escriturário - Agente de Tecnologia) Desenvolvimento de Software - 2023 (Pós-Edital)

Autor:
Raphael Henrique Lacerda, Paolla Ramos e Silva

20 de Janeiro de 2023

Índice

1) Python - Prof. Paolla Ramos 3

APRESENTAÇÃO DA AULA

Olá, galera! Vamos iniciar os estudos sobre o **Python!**

Pessoal, eu não vou ser aqueles professores clichês que dizem: **Esse assunto é muito fácil**. **Não é muito fácil!** 😅 Como diria o magnífico professor Herbert Almeida, não se assustem com o tema ou com o tamanho da aula. É necessário SIM ter conhecimento sobre programação básica. É necessário assistir a aula de Noções de programação do brilhante Prof. Raphael Lacerda, e caso não entenda algum comando ou algo do tipo, voltar e rever ou assistir às videoaulas. Sugiro também, sempre que possível, testar os códigos e aprender – não decorar. Por fim, fazer MUITAS questões. Porque Python é LÓGICA. É necessário aprender a lógica porque vai cair uma questão em sua prova em que você terá que resolver usando lógica. Não adianta decorar!

Nesta aula, apresento inúmeras palavras-chave, para que vocês possam entender e internalizar o conteúdo de forma mais simples. Aproveitem o material, além dos esquemas, resumos e mnemônicos. Vamos ao que importa! 😀

PYTHON

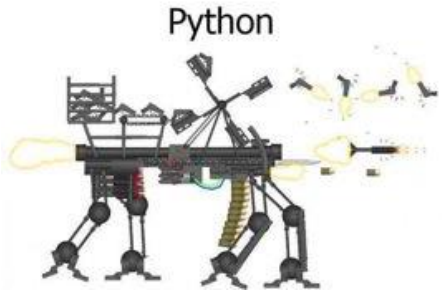
Conceitos Básicos

RELEVÂNCIA EM PROVA: ALTA

Pessoal, vamos falar sobre **Python**! **Python** é uma linguagem de código aberto, de propósito geral. É uma **linguagem de programação** de **alto nível,** ou seja, com sintaxe mais simplificada e próxima da linguagem humana.

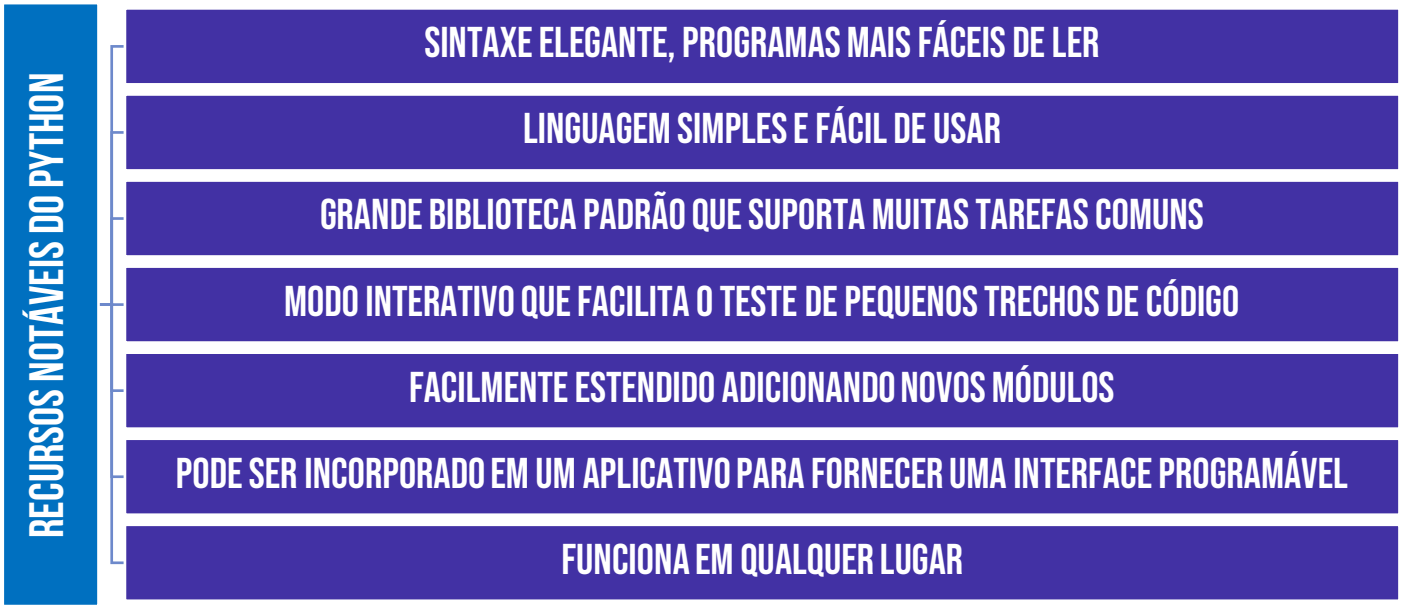
Python é uma linguagem de programação orientada a objetos clara e poderosa, comparável a Perl, Ruby, Scheme ou **Java**.

Alguns dos recursos **notáveis** do Python:



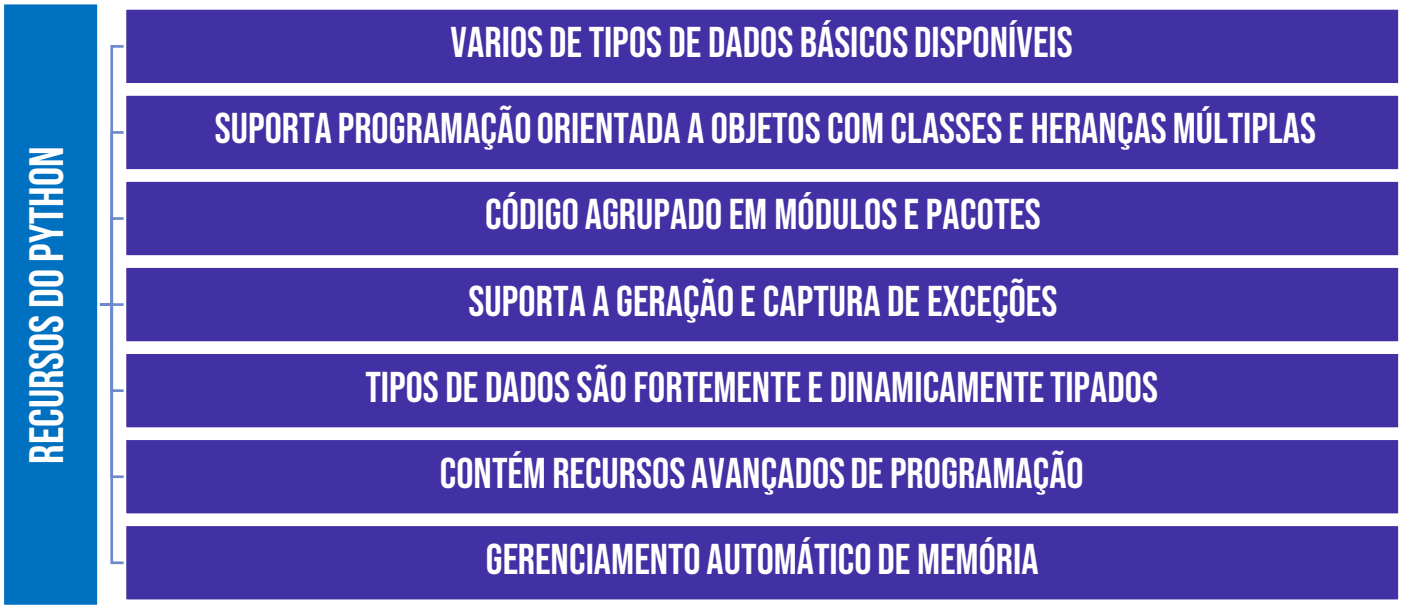
- Usa uma sintaxe elegante, tornando os programas que você escreve mais fáceis de ler.
- É uma linguagem fácil de usar que simplifica o funcionamento do seu programa. Isso torna o Python ideal para desenvolvimento de protótipos e outras tarefas de programação ad-hoc, sem comprometer a capacidade de manutenção.
- Vem com uma grande biblioteca padrão que suporta muitas tarefas comuns de programação, como conectar-se a servidores da Web, pesquisar texto com expressões regulares, ler e modificar arquivos.
- O modo interativo do Python facilita o teste de pequenos trechos de código. Há também um ambiente de desenvolvimento empacotado chamado IDLE.
- É facilmente estendido adicionando novos módulos implementados em uma linguagem compilada como C ou C++.
- Também pode ser incorporado em um aplicativo para fornecer uma interface programável.
- Funciona em qualquer lugar, incluindo Mac OS X, Windows , Linux e Unix , com versões não oficiais também disponíveis para Android e iOS.

- É software livre em dois sentidos. Não custa nada baixar ou usar o Python, ou incluí-lo em seu aplicativo. O Python também pode ser livremente modificado e redistribuído porque, embora a linguagem seja protegida por direitos autorais, ela está disponível sob uma licença de código aberto.



Alguns recursos de linguagem de programação do Python são:

- Uma variedade de tipos de dados básicos está disponível: números (ponto flutuante, inteiros longos complexos e de comprimento ilimitado), strings (ambos ASCII e Unicode), listas e dicionários.
- Python suporta programação orientada a objetos com **classes e heranças múltiplas**.
- O código pode ser agrupado em módulos e pacotes.
- A linguagem suporta a geração e captura de exceções, resultando em um tratamento de erros mais limpo.
- Os tipos de dados são fortemente e dinamicamente tipados. Misturar tipos incompatíveis (por exemplo, tentar adicionar uma string e um número) faz com que uma exceção seja lançada, de modo que os erros são detectados mais cedo.
- Python contém recursos avançados de programação, como geradores e compreensões de lista.
- O gerenciamento automático de memória do Python libera você de ter que alocar e liberar memória manualmente em seu código.



Python é uma linguagem com uma filosofia de simplicidade no seu design, é uma linguagem de código aberto e tem crescido exponencialmente. A própria comunidade oficial da linguagem diz isso: A comunidade do Python é vasta; diversa e visa crescer; Python é aberto.

Além disso, a comunidade cita que ótimos softwares são suportados por ótimas pessoas, e o Python não é exceção. A base de usuários é entusiasmada e dedicada a difundir o uso do Python por toda parte. A comunidade pode ajudar a apoiar o iniciante, o especialista e contribui para a base de conhecimento de **código aberto cada vez maior**.

Python é poderoso... e rápido; roda bem com as outras linguagens; roda em todos os lugares (SO...); é amigável e fácil de aprender; é aberto. Essas são algumas das razões pelas quais as pessoas que usam Python preferem não usar mais nada.

Na documentação é possível encontrar documentação desde a versão 2.7 até a 3.12. E continua evoluindo de forma exponencial.

Python combina um **poder notável com uma sintaxe muito clara**. Possui módulos, classes, exceções, tipos de dados dinâmicos de nível muito alto e tipagem dinâmica. Existem interfaces para muitas chamadas de sistema e bibliotecas, bem como para vários sistemas de janelas. Novos módulos são facilmente embutidos, escritos em C ou C++ (ou outras linguagens, dependendo da implementação escolhida). O **Python também pode ser usado como uma linguagem de extensão** para aplicativos escritos em outras linguagens que precisam de scripts ou interfaces de automação fáceis de usar.

Tipagem dinâmica é uma característica de determinadas linguagens de programação, que **não exigem declarações de tipos de dados**, pois **são capazes de escolher que tipo utilizar dinamicamente para cada variável**, podendo alterá-lo durante a compilação ou a execução do programa.

(CEBRASPE – PC PB– 2022) Na linguagem Python, o tipo de uma variável em tempo de execução é definido pelo interpretador pelo recurso denominado

a) tipagem dinâmica.
b) modo interativo.
c) sintaxe.
d) interpretação bytecode.
e) empacotamento.

Comentários: Tipagem dinâmica é uma característica de determinadas linguagens de programação, que não exigem declarações de tipos de dados, pois são capazes de escolher que tipo utilizar dinamicamente para cada variável, podendo alterá-lo durante a compilação ou a execução do programa. (Gabarito: Letra A).

Cada linguagem de programação reserva um significado especial a um conjunto fixo de palavras. Essas palavras são chamadas de palavras-chave.

Com palavras-chave, o programador pode emitir comandos para o compilador ou interpretador. Eles permitem que você diga ao computador o que fazer. Sem palavras-chave, o computador não poderia fazer sentido a partir do texto aparentemente aleatório em seu arquivo de código.

Observe que, como as palavras-chave são palavras reservadas, você não pode usá-las como nomes de variáveis. As palavras-chave Python mais importantes são as seguintes:

PALAVRA-CHAVE	DESCRIÇÃO
AND	And é um operador lógico. Os operadores lógicos são usados para combinar instruções condicionais. O valor de retorno será True somente se ambas as instruções retornarem True, caso contrário retornará False.
AS	Usada para criar um alias.
ASSERT	Usada ao depurar o código. Permite testar se uma condição em seu código retorna True, caso contrário, o programa gerará um AssertionError. Você pode escrever uma mensagem a ser escrita caso o código retorne false
BREAK	Usada para quebrar um loop for ou um loop.while
CLASS	Usada para criar uma classe. Uma classe é como um construtor de objetos.
CONTINUE	Usada para encerrar a iteração atual em um loop for (ou loop while) e continua na próxima iteração.
DEF	Usada para criar (ou definir) uma função.
DEL	Usada para excluir objetos. Em Python tudo é um objeto, então a palavra-chave del também pode ser usada para deletar variáveis, listas ou partes de uma lista etc.
ELIF	Usada em instruções condicionais (instruções if) e é abreviação de else if.
ELSE	com a ramificação "if", tenta as ramificações "elif" e termina com a ramificação "else" (até ser avaliada como True)

EXCEPT	Usada em blocos try...except. Ele define um bloco de código a ser executado se o bloco try gerar um erro. Você pode definir blocos diferentes para diferentes tipos de erro e blocos para executar se nada der errado
FALSE	A palavra-chave false é um valor booleano e resultado de uma operação de comparação. A palavra-chave false é igual a o (True é igual a 1).
FINALLY	Usada em blocos try...except. Ele define um bloco de código para ser executado quando o bloco try...except...else for final. O bloco finally será executado independentemente de o bloco try gerar um erro ou não. Isso pode ser útil para fechar objetos e limpar recursos.
FOR	Um loop for é usado para iterar sobre uma sequência (que é uma lista, uma tupla, um dicionário, um conjunto ou uma string). Permite executar um conjunto de instruções, uma vez para cada item de uma lista, tupla, conjunto etc. for i in [0,1,2]: print(i)
FROM	Usada para importar apenas uma seção especificada de um módulo.
GLOBAL	Usada para criar variáveis globais em um escopo não global, por exemplo, dentro de uma função.
IF	Usada para criar instruções condicionais (instruções if) e permite que você execute um bloco de código somente se uma condição for True. Use a palavra-chave else para executar o código se a condição for False
IMPORT	Usada para importar módulos.
IN	Retorna Verdadeiro se uma sequência com o valor especificado estiver presente no objeto. Exemplo: x in y
IS	Retorna Verdadeiro se ambas as variáveis forem o mesmo objeto. Exemplo: x is y
LAMBDA	Uma função lambda é uma pequena função anônima. Uma função lambda pode receber qualquer número de argumentos, mas pode ter apenas uma expressão.
NONE	O objeto Python None, denota falta de valor. Este objeto não tem métodos. É usado para definir um valor nulo ou nenhum valor.
NONLOCAL	Usada para trabalhar com variáveis dentro de funções aninhadas, onde a variável não deve pertencer à função interna. Use a palavra-chave nonlocal para declarar que a variável não é local.
NOT	É um operador lógico. O valor de retorno será True se a(s) instrução(ões) não forem True, caso contrário retornará False.
OR	É um operador lógico Os operadores lógicos são usados para combinar instruções condicionais. O valor de retorno será True se uma das instruções retornar True, caso contrário retornará False.
PASS	Usada como um espaço reservado para código futuro. Quando a instrução pass é executada, nada acontece, mas você evita obter um erro quando o código vazio não é permitido. Código vazio não é permitido em loops, definições de função, definições de classe ou em instruções if.
RAISE	Usada para gerar uma exceção. Você pode definir que tipo de erro gerar e o texto a ser impresso para o usuário.
RETURN	Sai de uma função e retornar um valor.

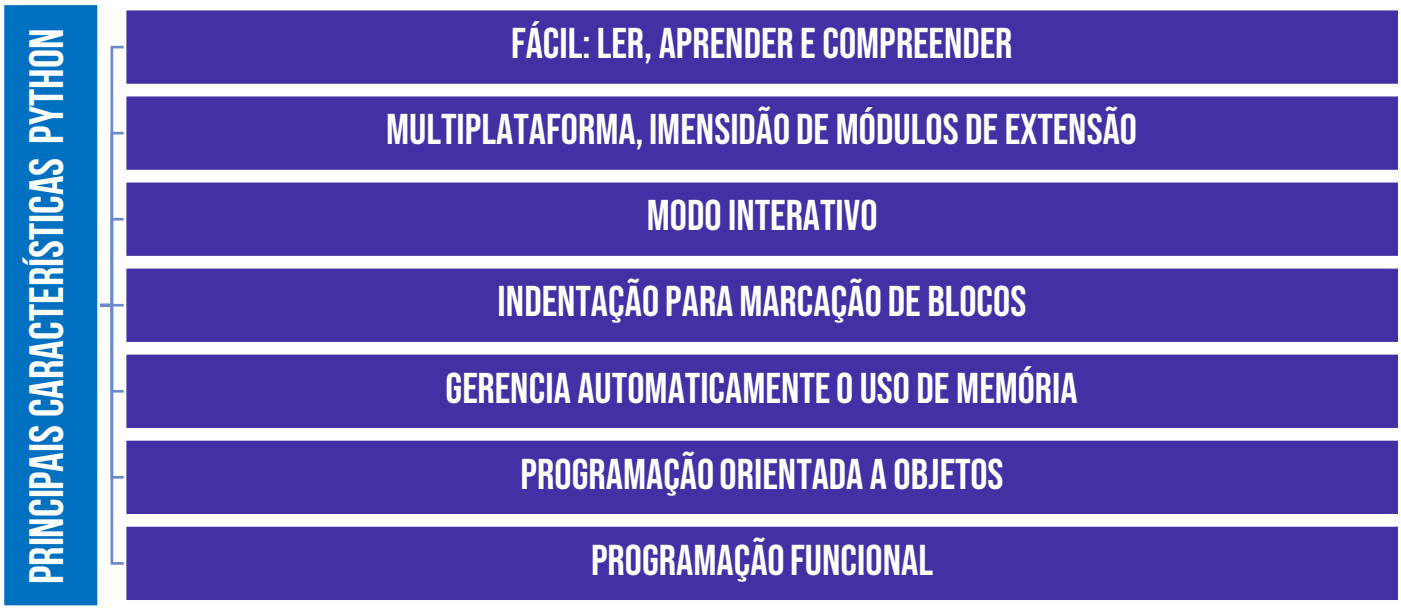
TRUE	Valor booleano e resultado de uma operação de comparação. A palavra-chave True é igual a 1 (False é igual a 0).
TRY	Usada em blocos try...except. Ele define um bloco de teste de código se ele contém algum erro. Você pode definir blocos diferentes para diferentes tipos de erro e blocos para executar se nada der errado.
WHILE	Com o loop while podemos executar um conjunto de instruções desde que uma condição seja verdadeira. O loop while requer que as variáveis relevantes estejam prontas (veja que é necessário declarar e atribuir o valor da variável j no exemplo abaixo). j = 0 while j < 3: print(j) j = j + 1
WITH	Usado para simplificar o tratamento de exceções
YIELD	Para finalizar uma função, retorna um gerador

Pessoal, nessa aula vamos ver muitos exemplos, é importante mencionar que vários exemplos dessa aula foram retirados ou **inspirados** em exemplos do **W3Tutorials** (www.w3schools.com/python). Fizemos isso por dois motivos: (1) os exemplos são excelentes; (2) essa é uma das fontes de inspiração das bancas. Caso você tenha alguma dúvida ou vontade de se aprofundar e testar os programas, você pode usar o link *https://www.w3schools.com/python/trypython.asp?filename=demo_default*
Ou qualquer outro site que disponibilize essa funcionalidade. 😉

Principais Características

Python é conhecida como uma linguagem de aspectos bastante interessantes e de fácil aprendizagem. O objetivo inicial da linguagem era permitir código enxuto e menos verboso, ou seja, com menos caracteres especiais, menos sintaxes complexas e mais estruturas de código simples. Por isso, se destaca

- A facilidade para aprender, ler e compreender;
- Ser multiplataforma;
- Possuir modo interativo;
- Usa indentação para marcação de blocos;
- Quase nenhum uso de palavras-chave associadas com compilação;
- Possuir coletor de lixo para gerenciar automaticamente o uso de memória;
- Programação orientada a objetos;
- Programação funcional; e
- Uma imensidão de módulos de extensão, os quais permitem expandir o poder da linguagem Python.



O Python foi projetado para facilitar a leitura e possui algumas semelhanças com o idioma inglês com influência da matemática. Python usa novas linhas para completar um comando, ao contrário de outras linguagens de programação que geralmente usam ponto e vírgula ou parênteses. A linguagem depende de recuo (indentação), usando espaço em branco, para definir o escopo; como o escopo de loops, funções e classes. Outras linguagens de programação costumam usar colchetes para essa finalidade.

Python é uma **linguagem de programação interpretada**, isso significa que, como desenvolvedor, você escreve arquivos **Python (.py)** em um editor de texto e **depois coloca esses arquivos no interpretador python para serem executados**.

A maneira de executar um arquivo python é assim na linha de comando:

```
C:\Users\Your Name>python helloworld.py
```

Em que "helloworld.py" é o nome do seu arquivo python.

Sintaxe do Python

A sintaxe do Python pode ser executada escrevendo diretamente na linha de comando:

```
>>> print("Hello, World!")
Hello, World!
```

O recuo, ou indentação, refere-se aos espaços no início de uma linha de código. Em outras linguagens de programação, o recuo no código é apenas para legibilidade, **o recuo em Python é muito importante**. Python usa recuo para indicar um bloco de código.

```
if 5 > 2:
  print("Cinco é maior que dois!")
```

O número de espaços fica a seu critério como programador, o uso mais comum é quatro, mas tem que ser pelo menos um.

```
if 5 > 2:
 print("Cinco é maior que dois!")
if 5 > 2:
        print("Cinco é maior que dois!")
```

Você precisa usar o mesmo número de espaços no mesmo bloco de código, caso contrário, o Python fornecerá um erro: **IndentationError: unexpected indent**

Variáveis em Python

Variáveis são contêineres para armazenar valores de dados. Python não tem comando para declarar uma variável. Em Python, as **variáveis** são criadas quando você atribui um valor a ela:

```
x = 5
y = "Hello, World!"
```

Python não tem comando para declarar uma variável. **As variáveis não precisam ser declaradas** com nenhum tipo específico e podem até mudar de tipo depois de terem sido definidas. 😉

AS VARIÁVEIS NÃO PRECISAM SER DECLARADAS

Se você deseja **especificar o tipo** de dados de uma variável, isso pode ser feito com conversão.

```
x = str(3)    # x será uma string com valor '3'
y = int(3)    # y será um inteiro igual a 3
z = float(3)  # será um float de valor igual a 3.0
```

Você pode obter o tipo de dados de uma variável com a função type().

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Resultará em: <class 'int'> e <class 'str'>. Variáveis de string podem ser declaradas usando **aspas simples ou duplas**:

```
x = "Professor"
x = 'Professor'
```

Python permite atribuir valores a várias variáveis **em uma linha**, neste caso, cada variável recebe o respectivo valor entre vírgulas:

```
x, y, z = "Laranja", "Banana", "Cereja"
```

Além disso, é possível atribuir o mesmo valor a várias variáveis em uma linha, vai que você precisa, não é? 🤪

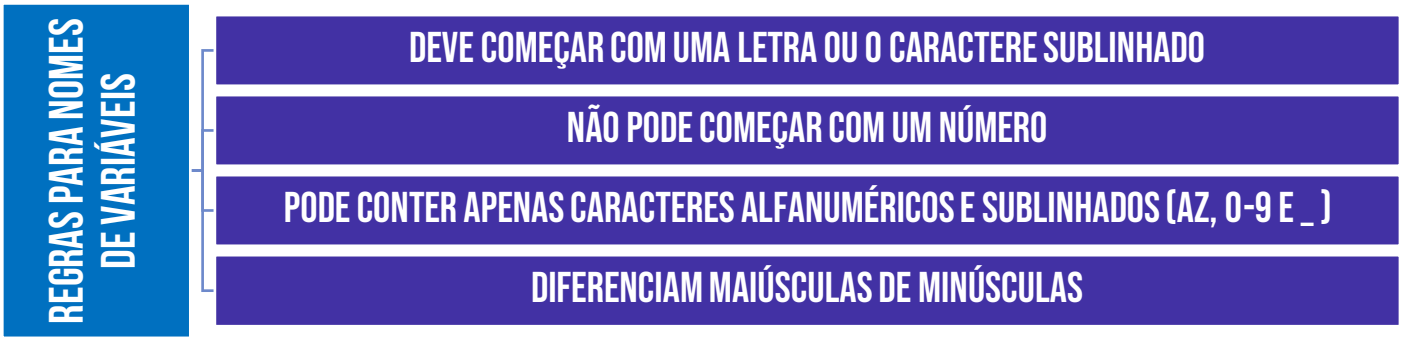
```
x = y = z = "Laranja"
```

Python, assim como Java, é case sensitive, ou seja, os nomes de variáveis diferenciam maiúsculas de minúsculas.

```
a = 4
A = "Professor"
```

Uma variável pode ter um nome curto (como x e y) ou um nome mais descritivo (idade, volume_total, marca_veiculo). Vejamos as regras para nomes de variáveis Python:

- Um nome de variável deve começar com uma letra ou o caractere sublinhado
- Um nome de variável não pode começar com um número
- Um nome de variável pode conter apenas caracteres alfanuméricos e sublinhados (Az, 0-9 e_)
- Os nomes das variáveis diferenciam maiúsculas de minúsculas (idade, idade e IDADE são três variáveis diferentes)



Variáveis criadas fora de uma função (como em todos os exemplos acima) são conhecidas como **variáveis globais**. As **variáveis globais** podem ser usadas por **todos**, tanto dentro das funções quanto fora delas.

```
x = "incrível"

def myfunc():
  print("Python é " + x)

myfunc()
```

Vocês viram, nesse exemplo, que usamos uma concatenação dentro do print. A **concatenação** pode ser definida como a **integração de duas strings em um objeto**. Em Python, você pode executar a concatenação usando o operador **+**. 😉

Se você criar uma variável com o **mesmo nome dentro de uma função**, essa variável será local **e só poderá ser usada dentro da função**. A variável global com o mesmo nome permanecerá como estava, global e com o valor original.

```
x = "incrível"

def myfunc():
  x = "fantástico"
  print("Python é " + x)

myfunc()

print("Python é " + x)
```

A primeira variável x terá o valor "Incrível". Já a segunda – dentro da função myfunc – possui o valor "fantástico". Assim, no print dentro da função myfunc receberemos a mensagem: Python é fantástico. Já fora da função, na última linha do exemplo, o print vai gerar a mensagem: Python é incrível. Ficou claro? 🙃
Normalmente, quando você cria uma variável **dentro de uma função**, essa **variável é local** e só pode ser usada dentro dessa função. Para criar uma **variável global dentro de uma função**, você pode usar a palavra-chave **global**.

Global: Usada para criar **variáveis globais** em um **escopo não global**, por exemplo, dentro de uma função.

```
def myfunc():
  global x
  x = "fantastic"
```

Tipos de dados Python

Na programação, o tipo de dados é um conceito importante. Variáveis podem armazenar dados de diferentes tipos, e diferentes tipos podem fazer coisas diferentes. O Python tem os seguintes tipos de dados **integrados por padrão**, nestas categorias:

TIPO DE DADO	DESCRIÇÃO	EXEMPLO
TEXTO	str	x = "Hello World"
NUMÉRICOS	int, float, complex	Int: x = "Hello World" Float: x = 20.5 Complex: x = 1j
SEQUÊNCIA	list, tupla, range	List: x = ["maçã", "banana", "cereja"] Tupla: x = ("maçã", "banana", "cereja") Range: x = range(6)
MAPEAMENTO	dict	Dict: x = {"nomw=e" : "John", "idade" : 36}
CONJUNTO	set,frozenset	Set: x = {"maçã", "banana", "cereja"} Frozenset: x = frozenset({"maçã", "banana", "cereja"})
BOOLEANO	bool	x = True
BINÁRIO	bytes, bytearray, memoryview	Bytes: x = b"Hello" Bytearray: x = bytearray(5) Memoryview: x = memoryview(bytes(5))
NENHUM TIPO	NoneType	NoneType: x = None

Pode haver momentos em que você deseja especificar um tipo para uma variável. Isso pode ser feito com **casting**. Python é uma linguagem orientada a objetos e, como tal, usa classes para definir tipos de dados, incluindo seus tipos primitivos. A conversão em python é, portanto, feita usando **funções construtoras**:

int() - constrói um número inteiro a partir de um literal inteiro, um literal float (removendo todos os decimais) ou um literal de string (desde que a string represente um número inteiro)
float() - constrói um número float a partir de um literal inteiro, um literal float ou um literal de string (desde que a string represente um float ou um inteiro)
str() - constrói uma string a partir de uma ampla variedade de tipos de dados, incluindo strings, literais inteiros e literais float

Strings podem conter uma ou mais linhas, veja um exemplo de uma string com mais de uma linha usando três aspas – **três aspas duplas (""") ou três aspas simples (''')**:

```
a = """Estratégia Concursos é referência na preparação de alunos para Concursos Públicos. Para ser aprovado em um concurso público, você precisa estudar com estratégia."""
print(a)
```

Strings em Python

Como muitas outras linguagens de programação populares, **strings** em Python são **arrays de bytes** que **representam caracteres unicode**. No entanto, o **Python não possui um tipo de dados de caractere**, um único caractere é simplesmente uma string com um comprimento de 1. Colchetes podem ser usados para acessar elementos da string.

```
a = "Hello, World!"
print(a[1])
```

No exemplo, o resultado será "e" porque o array inicia em o(zero). Como **strings** são **arrays**, podemos fazer um loop pelos caracteres em uma string, com um loop **for**.

```
for x in "banana":
  print(x)
```

Para obter o comprimento de uma string, use a função **len().**

```
a = "Hello, World!"
print(len(a))
```

Para verificar se uma determinada frase ou caractere está presente em uma string, podemos usar a palavra-chave **in**.

```
txt = "Estratégia Concursos é referência na preparação de alunos "
print("Concursos" in txt)
```

É possível procurar uma palavra em uma instrução if. Vejamos um exemplo:

```
txt = "Estratégia Concursos é referência na preparação de alunos "
if "Estratégia" in txt:
  print("Sim!, Estratégia está presente no texto.")
```

Há muitas funções que podem ser realizadas ou utilizadas em strings. É possível retornar um intervalo de caracteres usando a sintaxe de **slice**. Slice consiste em "fatiar", ou seja, cortar uma

parte da string. É necessário especificar **o índice inicial e o índice final**, **separados por dois pontos**, para retornar uma parte da string.

```
b = "Estratégia Concursos!"
print(b[2:5])
```

Esse exemplo retorna "tra". Importante que vocês entendam que o corte final desconsidera o conteúdo do índice final, perceba que (b[5]) = t. Esse valor não é retornado.

Ao omitir o índice inicial, o intervalo começará no primeiro caractere:

```
b = "Estratégia Concursos!"
print(b[:10])
```

Esse exemplo retorna apenas o texto "Estratégia". Por outro lado, ao omitir o índice final , o intervalo irá para o final:

```
b = "Estratégia Concursos!"
print(b[11:])
```

Esse exemplo retorna apenas o texto " Concursos!". Outra forma de cortar a string é utilizando índices negativos. É possível usar índices negativos para iniciar o corte a partir do final da string:

```
b = "Estratégia Concursos!"
print(b[-10])
```

O exemplo acima retorna apenas o texto "C".

Vejamos agora uma lista de métodos integrados disponíveis no Python que você pode usar em strings.

(FGV – MPE GO– 2022) Assinale a lista de números produzida pela execução, na IDLE Shell 3.9.9, do código Python a seguir.

```
for x in range(-1, -10, -1):
 print (x)
```

a) -1 -2 -3 -4 -5 -6 -7 -8 -9
b) -9 -8 -7 -6 -5 -4 -3 -2 -1
c) 0 -1 -2 -3 -4 -5 -6 -7 -8 -9
d) 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
e) -1 -2 -3 -4 -5 -6 -7 -8 -9 -10.

Comentários: Pessoal, as bancas gostam muito de índices negativos, para dificultar um pouco. No caso da questão, é gerada uma sequência de números negativos iniciando em -1 e parando antes do -10 (ou seja, o for para no -9). O For é incrementado de -1 em -1 formando a sequência -1 -2 -3 -4 -5 -6 -7 -8 -9. (Gabarito: Letra A)

MÉTODOS PARA MODIFICAR STRINGS	DESCRIÇÃO
UPPER()	Retorna a string em maiúsculas
LOWER()	Retorna a string em letras minúsculas
STRIP()	Remove qualquer espaço em branco do início ou do fim
REPLACE()	Substitui uma string por outra string. Ex: replace("h", "j")
SPLIT()	Retorna uma lista em que o texto entre o separador especificado se torna os itens da lista.

É possível, também, formatar strings. podemos combinar strings e números usando o método format()! 😀 O método format() pega os argumentos passados, os formata e os coloca na string onde estão os espaços reservados {}:

```
posicao = 01
txt = "O Estratégia Concursos é uma fabrica de {}"
print(txt.format(posicao))
```

O método format() recebe um número ilimitado de argumentos e são colocados nos respectivos espaços reservados:

```
vagas = 431
orgao = "SEFA MG"
cargo = "Auditor Fiscal"
concursos = "Há {} vagas no concurso {} para {}."
print(concursos.format(vagas, orgao, cargo))
```

Para inserir caracteres ilegais em uma string, use um caractere de escape.

Um caractere de escape é uma barra invertida \seguida pelo caractere que você deseja inserir.

Um exemplo de um **caractere ilegal** é uma **aspas dupla** dentro de uma string que é **cercada por aspas duplas: txt = " O Estratégia Concursos é uma "fabrica" de 01s."**. Você receberá um **erro** se usar aspas duplas dentro de uma string cercada por aspas duplas. Para corrigir esse problema, use **o caractere de escape \"**, da seguinte forma: txt = "O Estratégia Concursos é uma \"fabrica\" de 01s."

CARACTER DE ESCAPE	DESCRIÇÃO
\'	Aspas simples
\\	Barra invertida
\N	Nova linha
\R	Volta para margem esquerda e em uma nova linha
\T	Tab
\B	Backspace
\F	Caractere de controle ASCII de quebra de página. Ele força a impressora a ejetar a página atual e a continuar imprimindo na parte superior de outra.
\OOO	Valor octal
\XHH	Valor hexadecimal

Python tem um **conjunto de métodos integrados** que você pode usar em strings.

FUNÇÃO	DESCRIÇÃO
CAPITALIZE()	Converte o primeiro caractere em maiúscula
CASEFOLD()	Converte string em minúsculas
CENTER()	Retorna uma string centralizada
COUNT()	Retorna o número de vezes que um valor especificado ocorre em uma string
ENCODE()	Retorna uma versão codificada da string
ENDSWITH()	Retorna verdadeiro se a string terminar com o valor especificado
EXPANDTABS()	Define o tamanho da guia da string
FIND()	Pesquisa a string por um valor especificado e retorna a posição de onde foi encontrado
FORMAT()	Formata valores especificados em uma string
FORMAT_MAP()	Formata valores especificados em uma string
INDEX()	Pesquisa a string por um valor especificado e retorna a posição de onde foi encontrado
ISALNUM()	Retorna True se todos os caracteres da string forem alfanuméricos
ISALPHA()	Retorna True se todos os caracteres da string estiverem no alfabeto
ISASCII()	Retorna True se todos os caracteres da string forem caracteres ASCII
ISDECIMAL()	Retorna True se todos os caracteres na string forem decimais
ISDIGIT()	Retorna True se todos os caracteres da string forem dígitos
ISIDENTIFIER()	Retorna True se a string for um identificador
ISLOWER()	Retorna True se todos os caracteres da string forem minúsculos
ISNUMERIC()	Retorna True se todos os caracteres da string forem numéricos

ISPRINTABLE()	Retorna True se todos os caracteres da string forem imprimíveis
ISSPACE()	Retorna True se todos os caracteres na string forem espaços em branco
ISTITLE()	Retorna True se a string seguir as regras de um título
ISUPPER()	Retorna True se todos os caracteres da string forem maiúsculos
JOIN()	Converte os elementos de um iterável em uma string
LJUST()	Retorna uma versão justificada à esquerda da string
LOWER()	Converte uma string em letras minúsculas
LSTRIP()	Retorna uma versão de corte à esquerda da string
MAKETRANS()	Retorna uma tabela de tradução para ser usada nas traduções
PARTITION()	Retorna uma tupla onde a string é dividida em três partes
REPLACE()	Retorna uma string onde um valor especificado é substituído por um valor especificado
RFIND()	Pesquisa a string por um valor especificado e retorna a última posição de onde foi encontrado
RINDEX()	Pesquisa a string por um valor especificado e retorna a última posição de onde foi encontrado
RJUST()	Retorna uma versão justificada à direita da string
RPARTITION()	Retorna uma tupla onde a string é dividida em três partes
RSPLIT()	Divide a string no separador especificado e retorna uma lista
RSTRIP()	Retorna uma versão de corte à direita da string
SPLIT()	Divide a string no separador especificado e retorna uma lista
SPLITLINES()	Divide a string nas quebras de linha e retorna uma lista
STARTSWITH()	Retorna verdadeiro se a string começar com o valor especificado
STRIP()	Remove espaços em branco (ou caracteres dentro do parêntese) do início/fim da string
SWAPCASE()	Troca de maiúsculas, minúsculas tornam-se maiúsculas e vice-versa
TITLE()	Converte o primeiro caractere de cada palavra para maiúscula
TRANSLATE()	Retorna uma string traduzida
UPPER()	Converte uma string em maiúscula
ZFILL()	Preenche a string com um número especificado de valores o no início

Booleanos em Python

Booleanos representam um dos dois valores: **True** ou **False**. Na programação, muitas vezes você precisa saber se uma expressão é True ou False. Você pode avaliar qualquer expressão em Python e obter uma das duas respostas. Quando você compara dois valores, a expressão é avaliada e o Python retorna a resposta booleana:

```
print(10 > 9) #Retorna True
```

```
print(10 == 9)#Retorna False
print(10 < 9) #Retorna False
```

Curiosidades: A função bool() permite avaliar qualquer valor e dar a você True ou False em troca. A maioria dos valores são verdadeiros. Quase qualquer valor é avaliado True se tiver algum tipo de conteúdo. Qualquer **string** é True, **exceto strings vazias.** Qualquer número é True, **exceto o (zero)**. Qualquer lista, tupla, conjunto e dicionário são True, **exceto os vazios**.

Por outro lado, há valores avaliados como **False**, exceto valores vazios, como (), [], {}, "", o número o e o valor None. E, claro, o valor False é avaliado como False. Exemplos de valores avaliados como False:

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

Operadores Python

Os operadores são usados para realizar operações em variáveis e valores. Python divide os operadores nos seguintes grupos:

- Operadores aritméticos
- Operadores de atribuição
- Operadores de comparação
- Operadores lógicos
- Operadores de identidade
- Operadores de associação
- Operadores bit a bit

Operadores aritméticos Python

OPERADOR	NOME	EXEMPLO
+	Adição	x + y
-	Subtração	x - y
*	Multiplicação	x * y
/	Divisão	x / y
%	Módulo	x % y

**	Exponenciação	x ** y
//	Floor Division: arredonda o resultado para o número inteiro mais próximo	x // y

Os operadores de atribuição são usados para atribuir valores a variáveis:

FUNÇÃO	DESCRIÇÃO	IGUAL A
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Os operadores de comparação são usados para comparar dois valores:

FUNÇÃO	DESCRIÇÃO	IGUAL A
==	Igual	x == y
!=	Diferente	x != y
>	Maior que	x > y
<	Menor que	x < y
>=	Maior ou igual a	x >= y
<=	Menor ou igual a	x <= y

Os operadores lógicos são usados para combinar instruções condicionais:

OPERADOR	DESCRIÇÃO	EXEMPLO
AND	Retorna True se ambas as declarações forem verdadeiras	x < 5 and x < 10

OR	Retorna True se uma das declarações for verdadeira	x < 5 or x < 4
NOT	Inverte o resultado, retorna False se o resultado for verdadeiro	not(x < 5 and x < 10)

Os operadores de identidade são usados para comparar os objetos, não se forem iguais, mas **se forem realmente o mesmo objeto**, com a **mesma localização de memória**:

OPERADOR	DESCRIÇÃO	EXEMPLO
IS	Retorna True se ambas as variáveis forem o mesmo objeto	x is y
IS NOT	Retorna True se ambas as variáveis não forem o mesmo objeto	x is not y

Os operadores de associação são usados para testar se uma sequência é apresentada em um objeto:

OPERADOR	DESCRIÇÃO	EXEMPLO
IN	Retorna True se uma sequência com o valor especificado estiver presente no objeto	x in y
NOT IN	Retorna True se uma sequência com o valor especificado não estiver presente no objeto	x not in y

Operadores **Bitwise** Python: Operadores bit a bit são usados para comparar números (binários):

OPERADOR	NOME	DESCRIÇÃO
&	AND	Define cada bit como 1 se ambos os bits forem 1
|	OR	Define cada bit como 1 se um dos dois bits for 1
^	XOR	Define cada bit como 1 se apenas um dos dois bits for 1
~	NOT	Inverte todos os bits
<<	Zero fill left shift	Desloque para a esquerda empurrando zeros da direita e deixe os bits mais à esquerda fora
>>	Signed right shift	Desloque para a direita empurrando cópias do bit mais à esquerda da esquerda e deixe os bits mais à direita fora

Pessoal, para tentar explicar melhor, vamos a um exemplo. Suponha: x << y. Isso retorna "x" com os **seus bits movidos** "y" **casas à esquerda**, adicionando zeros aos bits "novos" da direita. Isso é a mesma coisa que **multiplicar x por 2**y**. Exemplo: 14 << 9 # decimal 7168. 1110 << 1001 # binario 1110000000000

Por outro lado, vejamos o >>. x >> y retorna "x" com seus bits movidos "y" casas à direita. Isso é o mesmo que x // 2 ** y. Exemplo: 14 >> 9 # decimal 0 2 >> 1 #decimal 1. 1110 >> 1001 # binario 0 10 >> 1 # binario 1

Coleções em Python

Listas Python

As listas são usadas para armazenar vários itens em uma **única variável**. As listas são um dos 4 tipos de dados internos do Python usados para armazenar **coleções de dados**, os outros 3 são Tupla , Conjunto e Dicionário, todos com qualidades e usos diferentes. As listas são criadas usando colchetes:

```
lista1 = ["maçã", "banana", "cereja"]
print(lista1)
```

Os itens da lista são ordenados, alteráveis e permitem valores duplicados. Os itens da lista são indexados, o primeiro item possui **índice [0]**, o segundo item possui índice [1], etc. Quando dizemos que as listas estão ordenadas, significa que os itens **têm uma ordem definida**, e essa ordem **não será alterada**. Se você adicionar novos itens a uma lista, os novos itens serão colocados no final da lista.

A lista é mutável, o que significa que podemos alterar, adicionar e remover itens em uma lista após ela ter sido criada. Como as listas são indexadas, as listas podem ter itens **com o mesmo valor**:

(CESPE – DPE RO – 2022) Na linguagem Python, são consideradas sequências mutáveis as:
a) strings
b) cadeias
c) tuplas
d) listas
e) ranges.

_____

Comentários: A lista é mutável, o que significa que podemos alterar, adicionar e remover itens em uma lista após ela ter sido criada. Portanto, as listas são consideradas sequências mutáveis. (Gabarito: Letra D).

```
lista2 = ["maçã", "banana", "cereja", "maçã", "cereja"]
print(lista2)
```

Para determinar quantos itens uma lista possui, use a função len():

```
lista3 = ["maçã", "banana", "cereja"]
print(len(lista3))
```

Os itens da lista podem ser de qualquer tipo de dados:

```
lista1 = ["maçã", "banana", "cereja"]
lista2 = [1, 5, 7, 9, 3]
lista3 = [True, False, False]
```

Uma lista pode conter diferentes tipos de dados:

```
lista1 = ["abc", 34, True, 40, "masculino"]
```

Da perspectiva do Python, as listas são definidas como objetos com o tipo de dados 'list'. Também é possível usar o construtor list() ao criar uma nova lista.

Coleções Python (matrizes)
Existem quatro tipos de dados de coleção na linguagem de programação Python:

- **Lista** é uma coleção que é ordenada e mutável. Permite membros duplicados.
- **Tupla** é uma coleção ordenada e imutável. Permite membros duplicados.
- **Set** é uma coleção não ordenada, imutável[1] e não indexada. Nenhum membro duplicado.
- **Dicionário** é uma coleção ordenada[2] e mutável. Nenhum membro duplicado.

Ao escolher um tipo de coleção, é útil entender as propriedades desse tipo. Escolher o tipo certo para um determinado conjunto de dados pode significar retenção de significado e pode significar um aumento na eficiência ou segurança.

Vejamos agora como os itens da lista são indexados e você pode acessá-los consultando o número do índice:

```
lista1 = ["maçã", "banana", "cereja"]
print(lista1[1]) #imprime na tela banana.
```

Por outro lado, a indexação negativa significa começar do fim. -1 refere-se ao último item, -2refere-se ao penúltimo item etc.

```
lista1 = ["maçã", "banana", "cereja"]
print(lista1[-1]) #imprime na tela cereja.
```

[1] Os itens do conjunto são imutáveis, mas você pode remover e/ou adicionar itens sempre que quiser.

[2] A partir da versão 3.7 do Python, os dicionários são ordenados. No Python 3.6 e anteriores, os dicionários não são ordenados.

Você pode especificar um intervalo de índices especificando onde começar e onde terminar o intervalo. Ao especificar um intervalo, **o valor de retorno será uma nova lista** com os itens especificados.

```
lista1 = ["maçã", "banana", "cereja", "laranja", "kiwi", "melão", "manga"]
print(lista1 [2:5]) #imprime na tela ['cereja', 'laranja', 'kiwi']
```

Isso retornará os itens da posição 2 até a posição 5. Lembre-se que o primeiro item é a posição o(zero), e note que **o item na posição 5 NÃO está incluído**

Para determinar se um item especificado está presente em uma lista, use a palavra-chave **in**:

```
lista1 = ["maçã", "banana", "cereja"]
if "maçã" in lista1:
  print("Sim, 'maçã' é uma fruta da lista")
```

Para alterar o valor de um item específico, consulte o número do índice:

```
lista1 = ["maçã", "banana", "cereja"]
lista1 [1] = "abacaxi"
print(lista1) #imprime ['maçã', 'abacaxi', 'cereja']
```

Para alterar o valor dos itens dentro de um intervalo específico, defina uma lista com os novos valores e consulte o intervalo de números de índice onde deseja inserir os novos valores:

```
lista1 = ["maçã", "banana", "cereja", "laranja", "kiwi", "manga"]
lista1[1:3] = ["abacaxi", "melancia"]
print(lista1)  #imprime  ['maçã',  'abacaxi',  'melancia',  'laranja',  'kiwi',
'manga']
```

Python tem um conjunto de métodos embutidos que você pode usar em **listas/matrizes**.

MÉTODO	DESCRIÇÃO
APPEND()	Adiciona um elemento no final da lista
CLEAR()	Remove todos os elementos da lista
COPY()	Retorna uma cópia da lista
COUNT()	Retorna o número de elementos com o valor especificado
EXTEND()	Adiciona os elementos de uma lista (ou qualquer iterável), ao final da lista atual
INDEX()	Retorna o índice do primeiro elemento com o valor especificado
INSERT()	Adiciona um elemento na posição especificada
POP()	Remove o elemento na posição especificada

REMOVE()	Remove o primeiro item com o valor especificado
REVERSE()	Inverte a ordem da lista
SORT()	Inverte a ordem da lista

(FGV – MPE SC – 2022) Analise o código Python a seguir.

```
x1 = {"A", "B", "C"}
x2 = ["AA", "BB", "CC"]
x1.add("B")
x2.append("BB")
x2.append(x1)
print (x2)
```

Dado que os elementos de x1 podem ser exibidos em ordem aleatória, a linha que possivelmente é produzida pelo comando print na execução do código acima é:

a) ['AA', 'BB', 'CC', 'BB', {'C', 'A', 'B'}]
b) ['AA', 'BB', 'CC', 'BB', 'C', 'A', 'B', 'B']
c) ['AA', 'BB', 'CC', 'BB', 'C', 'A', 'B']
d) ['AA', 'BB', 'CC', ['BB'], {'B'}]
e) {'AA', 'BB', 'CC', 'BB', 'C', 'A', 'B'}

Comentários: Pessoal, primeiramente devemos desconsiderar x1 já que o comando da questão diz que os elementos de x1 podem ser exibidos em ordem aleatória. Vejamos então a ordem de x2. Inicialmente temos um ["AA", "BB", "CC"] que consiste no x2 original. Após, teremos BB (devido ao x2.append ("BB")). Portanto ficamos com 'AA', 'BB', 'CC', 'BB'. Dessa forma ficamos entre as opções a,b, c e e. Porém, depois do x2 há um x1 (x2.append(x1)). Dessa forma, podemos ter um{"A", "B", "C"} em uma sequência aleatória – como diz o comando da questão. A única opção que possui um {"A", "B", "C"} em sequência aleatória é a letra A. (Gabarito: Letra A).

Você pode percorrer os itens da lista usando um loop **for**:

```
lista1 = ["maçã", "banana", "laranja"]
for x in lista1:
  print(x)
```

Além disso, é possível percorrer os itens da lista consultando seu número de índice. Use as funções **range()** e **len()** para criar um iterável adequado.

```
lista1 = ["maçã", "banana", "laranja"]
for i in range(len(lista1)):
  print(lista1[i])
```

Outra forma de percorrer a lista é usando um loop while. Use a função len() para determinar o **comprimento da lista**, então comece em o e faça um loop pelos itens da lista consultando seus índices. Lembre-se de aumentar o índice em 1 após cada iteração.

```
lista1= ["maçã", "banana", "laranja"]
i = 0
while i < len(lista1):
  print(lista1[i])
  i = i + 1
```

List Comprehension

O **List Comprehension** oferece uma sintaxe mais curta quando você deseja criar uma nova lista com base nos valores de uma lista existente. Por exemplo, com base em uma lista de frutas, você deseja uma nova lista, contendo apenas as frutas com a letra "a" no nome. Sem compreensão de lista, você terá que escrever uma declaração **for** com um **teste condicional** dentro:

```
frutas = ["maçã", "banana", "laranja", "kiwi", "mango"]
novalista = []

for x in frutas:
  if "a" in x:
    novalista.append(x)

print(novalista)
```

Com **List Comprehension**, você pode fazer tudo isso com apenas **uma linha de código**:

```
frutas = ["maçã", "banana", "laranja"]
novalista = [x for x in frutas if "a" in x]
```

vejamos agora como Classificar lista alfanumérica. Objetos de lista possuem um método sort() que **ordenará a lista de forma alfanumérica**, em ordem crescente, por padrão:

```
frutas.sort()
```

para classificar de forma decrescente, use reverse = True

```
frutas.sort(reverse = True)
```

Agora vejamos como é possível copiar uma lista! Você não pode copiar uma lista simplesmente digitando **list2 = list1**, porque: list2 será apenas uma referência a list1, e as alterações feitas list1 automaticamente também serão feitas em list2. Existem maneiras de fazer uma cópia, uma delas é usar o método interno copy().

```
frutas = ["maçã", "banana", "laranja"]
minhalista = frutas.copy()
print(minhalista)
```

Outra maneira de fazer uma cópia é usar o método interno list().

```
frutas = ["maçã", "banana", "laranja"]
```

```
minhalista = list(frutas)
print(minhalista)
```

Existem várias maneiras de **unir ou concatenar duas ou mais listas** em Python. Uma das maneiras mais fáceis é usando o **operador +**.

```
lista1 = ["a", "b", "c"]
lista2 = [1, 2, 3]

lista3 = lista1 + lista2
print(lista3)
```

Mais uma maneira de juntar duas listas é anexando todos os itens da lista2 na lista1, um por um:

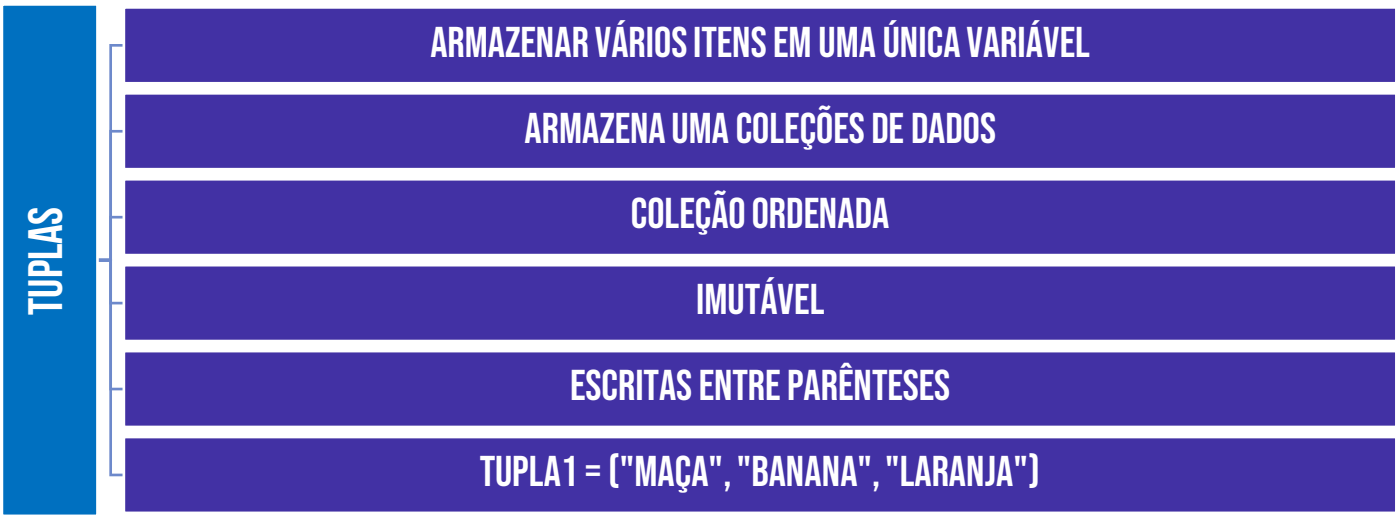
```
lista1 = ["a", "b" , "c"]
lista2 = [1, 2, 3]

for x in lista2:
  lista1.append(x)

print(lista1)
```

Tuplas Python

Tuplas são usadas para **armazenar vários itens em uma única variável**. Tupla é um dos 4 tipos de dados internos do Python usados para **armazenar coleções de dados**, os outros 3 são List , Set e Dictionary , todos com qualidades e usos diferentes. Uma tupla é uma **coleção ordenada e imutável** . Tuplas são **escritas com colchetes**.



Os itens de tupla são **indexados**, o **primeiro item possui índice [0]**, o segundo item possui índice [1], e assim sucessivamente. Quando dizemos que as tuplas estão ordenadas, significa que os itens têm uma ordem definida, e **essa ordem não será alterada**.

As tuplas são **imutáveis**, o que significa que **não podemos alterar, adicionar ou remover itens após a criação da tupla**.

NÃO PODEMOS ALTERAR, ADICIONAR OU REMOVER ITENS APÓS A CRIAÇÃO DA TUPLA.

Como as tuplas são indexadas, elas **podem ter itens com o mesmo valor**, ou seja, tuplas permitem duplicatas.

```
tupla1 = ("maça", "banana", "laranja","maça")
```

Para determinar quantos itens uma tupla possui, use a função **len():**

```
print(len(tupla1))
```

Os itens de tupla podem ser de **qualquer tipo de dados**:

```
tupla1 = ("maça", "banana", "laranja") #tupla de strings
tupla2 = (1, 5, 7, 9, 3) #tupla de inteiros
tupla3 = (True, False, False) #tupla de booleanos
tupla4 = ("abc", 34, True, 40, "masculino") #tupla de diversos tipos
```

Na prática, é muito usado, além de necessário, tuplas de diferentes tipos de dados. A tupla 4 apresenta strings, inteiros e valores booleanos.

Para acessar os itens da tupla, é possível consultar o número do índice, entre colchetes – lembrando que inicia em o (zero):

```
tupla1 = ("maça", "banana", "laranja")
print(tupla1[1]) #imprime banana
```

Da mesma forma que ocorre na lista, é possível acessar tuplas com **índice negativo**. Lembrando que o -1 pega o último item, -2refere-se ao penúltimo item, etc.

```
tupla1 = ("maça", "banana", "laranja")
print(tupla1[-1]) #imprime laranja
```

As tuplas são imutáveis, o que significa que vo**cê não pode alterar, adicionar ou remover itens depois que a tupla for criada**. Mas existem algumas soluções alternativas. Uma delas é converter uma tupla em uma lista, alterar a lista e convertê-la novamente em uma tupla. Para excluir completamente uma tupla, pode-se usar a palavra-chave **del**.

```
tupla1 = ("maça", "banana", "laranja")
del tupla1
```

É possível percorrer os itens da tupla usando um loop for.

```
tupla1 = ("maçã", "banana", "laranja")
for x in tupla1:
  print(x)
```

Pessoal, esse é o exemplo mais simples de todos, e o detalhe é que as questões muitas vezes cobram o loop for, portanto, atenção! Vamos ver mais detalhes quando falarmos sobre o Loop for.

Ademais, é possível percorrer os itens da tupla **consultando seu número de índice**. Use as funções **range()e len()** para criar um iterável adequado.

```
tupla1 = ("maçã", "banana", "laranja")
for i in range(len(tupla1)):
  print(tupla1[i])
```

Além do loop for, é possível percorrer os itens da lista usando um loop **while**. Use a função **len()** para **determinar o comprimento da tupla**, então comece em o e **faça um loop pelos itens da tupla consultando seus índices.** Lembre-se, sempre, de aumentar o índice em 1 após cada iteração.

```
tupla1= ("maçã", "banana", "laranja")
i = 0
while i < len(tupla1):
  print(tupla1[i])
  i = i + 1
```

Além de percorrer as tuplas, é possível concatená-las usando o operador de concatenação **+**:

```
tupla1 = ("a", "b" , "c")
tupla2 = (1, 2, 3)

tupla3 = tupla1 + tupla2
```

```
print(tupla3)
```

Há outra função incrível que pode ser utilizada em **tuplas**! 😄 Se você quiser multiplicar o conteúdo de uma tupla um determinado número de vezes, você pode usar o operador *:

```
frutas = ("maçã", "banana", "laranja")
tupla1 = frutas * 2

print(tupla1)
# imprime ('maçã', 'banana', 'laranja', 'maçã', 'banana', 'laranja')
```

Python possui **dois métodos integrados** que você **pode usar em tuplas**.

MÉTODO	DESCRIÇÃO
COUNT()	Retorna o número de vezes que um valor especificado ocorre em uma tupla
INDEX()	Procura na tupla um valor especificado e retorna a posição de onde foi encontrado

Conjuntos Python

Conjuntos, também conhecidos como **SET**s, são usados para armazenar vários itens em uma única variável. Um **conjunto** é uma **coleção não ordenada , imutável[3] e não indexada.** São escritos com colchetes.

```
conjunto1 = {"maça", "banana", "cereja"}
print(conjunto1)
```

Não ordenado significa que os **itens em um conjunto não têm uma ordem definida**. Os itens do conjunto podem aparecer em uma ordem diferente toda vez que você os usa e **não podem ser referenciados por índice ou chave. Imutável** significa que **não podemos alterar os itens após a criação do conjunto. Os conjuntos não podem ter dois itens com o mesmo valor.**

Diferente das listas, e tuplas, os conjuntos **não podem ter dois itens com o mesmo valor**. Por outro lado, assim como as demais coleções, é possível determinar quantos itens um conjunto possui usando a função **len()**.

Não é possível acessar itens em um conjunto fazendo referência a **um índice ou a uma chave**. Mas você pode **percorrer os itens** do conjunto **usando um loop for** ou perguntar se um valor especificado está presente em um conjunto, usando a palavra-chave **in**.

[3] A partir da versão 3.7 do Python, os dicionários são ordenados. No Python 3.6 e anteriores, os dicionários não são ordenados.

```
thisset = {"maça", "banana", "cereja"}

for x in thisset:
  print(x)
```

Não é possível alterar itens depois que um conjunto é criado, porém é possível **adicionar novos itens**. Para adicionar um item a um conjunto, use o método **add().**

```
thisset = {"maça", "banana", "cereja"}
thisset.add("laranja")
```

Para adicionar **itens de outro conjunto ao conjunto atual**, use o método **update()**.

```
thisset = {"maça", "banana", "cereja"}
tropical = {"pinemaça", "mango", "papaya"}

thisset.update(tropical)
```

O objeto no método **update()** não precisa ser um conjunto, **pode ser qualquer objeto iterável** (tuplas, listas, dicionários etc.).

```
thisset = {"maça", "banana", "cereja"}
mylist = ["kiwi", "laranja"]

thisset.update(mylist)
```

Para remover um item em um conjunto, use o método **remove()**, ou .**discard().** Se o item a ser removido não existir, usando o método remove(), **será gerado um erro**. Por outro lado, se o item a ser removido não existir, usando o método **discard**() **NÃO irá gerar um erro**.

```
thisset = {"maça", "banana", "cereja"}

thisset.remove("banana")
```

Além de **remove()**, e **discard()**, é possível usar o método o **pop()** que irá remover o **último item do conjunto**. Por fim, o método **clear() esvazia o conjunto**, e a palavra-chave **del excluirá o conjunto completamente.**

Vejamos agora as várias maneiras de **unir dois ou mais conjuntos** em Python.

Você pode usar o método **union()** que retorna um **novo conjunto contendo todos os itens** de ambos os conjuntos ou o método **update()** que **insere todos os itens de um conjunto em outro**. Ambos **union()** e **update()** excluirão quaisquer itens duplicados.

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
set1.update(set2)
```

Python é tão incrível que há vários métodos para trabalhar com conjuntos. Vejamos agora os métodos **intersection(), intersection_update(), symmetric_difference(), e symmetric_difference_update().**

O método **intersection()** retornará um **novo conjunto**, que **contém apenas os itens presentes em ambos os conjuntos**. Por outro lado, o método **intersection_update() manterá** apenas os itens presentes em **ambos os conjuntos.**

O método **symmetric_difference()** retornará um novo conjunto, que **contém apenas os elementos que NÃO estão presentes em ambos os conjuntos**. Por fim, o método **symmetric_difference_update()** manterá apenas os elementos que **NÃO estão presentes em ambos os conjuntos.**

Pessoal, para ajuda-los a internalizar o conteúdo, segue uma tabela com os métodos **intersection(), intersection_update(), symmetric_difference(), e symmetric_difference_update().**

MÉTODO	DESCRIÇÃO	EXEMPLO
INTERSECTION()	Gera novo conjunto, que contém apenas os itens presentes em ambos os conjuntos	z = x.intersection(y)
INTERSECTION_UPDATE()	Manterá apenas os itens presentes em ambos os conjuntos.	x.intersection_update(y)
SYMMETRIC_DIFFERENCE()	Novo conjunto, que contém apenas os elementos que NÃO estão presentes em ambos os conjuntos	z = x.symmetric_difference(y)
SYMMETRIC_DIFFERENCE_UPDATE()	Manterá apenas os elementos que NÃO estão presentes em ambos os conjuntos.	x.symmetric_difference_update(y)

Abaixo é apresentada a tabela contendo o conjunto de **métodos integrados** que você pode usar em conjuntos.

MÉTODO	DESCRIÇÃO

ADD()	Adiciona um elemento ao conjunto
CLEAR()	Remove todos os elementos do conjunto
COPY()	Retorna uma cópia do conjunto
DIFFERENCE()	Retorna um conjunto contendo a diferença entre dois ou mais conjuntos
DIFFERENCE_UPDATE()	Remove os itens neste conjunto que também estão incluídos em outro conjunto especificado
DISCARD()	Remove o item especificado
INTERSECTION()	Retorna um conjunto, que é a interseção de dois outros conjuntos
INTERSECTION_UPDATE()	Remove os itens neste conjunto que não estão presentes em outros conjuntos especificados
ISDISJOINT()	Retorna se dois conjuntos têm uma interseção ou não
ISSUBSET()	Retorna se outro conjunto contém este conjunto ou não
ISSUPERSET()	Retorna se este conjunto contém outro conjunto ou não
POP()	Remove um elemento do conjunto
REMOVE()	Remove o elemento especificado
SYMMETRIC_DIFFERENCE()	Retorna um conjunto com as diferenças simétricas de dois conjuntos
SYMMETRIC_DIFFERENCE_UPDATE()	Insere as diferenças simétricas deste conjunto e de outro
UNION()	Retorna um conjunto contendo a união de conjuntos
UPDATE()	Atualize o conjunto com a união deste conjunto e outros

Dicionários Python

Os dicionários são usados para armazenar valores de dados em pares **chave:valor**. Um dicionário é uma **coleção ordenada -** a partir do Python 3.7, **mutável** e que **não permite duplicatas.** Os dicionários são escritos com colchetes e possuem **chaves e valores**:

```
carros = {
  "marca": "Toyota",
  "modelo": "Corolla Cross",
  "ano": 2022
}
```

Para referenciar um item do dicionário usa-se o **nome da chave**. Por exemplo, caso você queira referenciar esse magnífico carro, você pode digitar o seguinte comando: print(carros["modelo"]). Quando dizemos que os dicionários estão **ordenados**, significa que os **itens têm uma ordem definida, e essa ordem não será alterada**. Por outro lado, obviamente, não ordenado significa que os itens **não têm uma ordem definida**, você **não pode fazer referência a um item usando um índice**.

Os dicionários são mutáveis, o que significa que **podemos alterar, adicionar ou remover itens após a criação do dicionário.** Ademais, os dicionários **não podem ter dois itens com a mesma chave.**

MÉTODO	DESCRIÇÃO
CLEAR()	Remove todos os elementos do dicionário
COPY()	Retorna uma cópia do dicionário
FROMKEYS()	Retorna um dicionário com as chaves e o valor especificados
GET()	Retorna o valor da chave especificada
ITEMS()	Retorna uma lista contendo uma tupla para cada par de valores-chave
KEYS()	Retorna uma lista contendo as chaves do dicionário
POP()	Remove o elemento com a chave especificada
POPITEM()	Remove o último par de valores-chave inserido
SETDEFAULT()	Retorna o valor da chave especificada. Se a chave não existir: insira a chave, com o valor especificado
UPDATE()	Atualiza o dicionário com os pares de valores-chave especificados
VALUES()	Retorna uma lista de todos os valores no dicionário

COLEÇÕES

- LISTA É UMA COLEÇÃO ORDENADA E MUTÁVEL. PERMITE MEMBROS DUPLICADOS.
- TUPLA É UMA COLEÇÃO ORDENADA E IMUTÁVEL. PERMITE MEMBROS DUPLICADOS.
- SET É UMA COLEÇÃO NÃO ORDENADA, IMUTÁVEL E NÃO INDEXADA. NENHUM MEMBRO DUPLICADO.
- DICIONÁRIO É UMA COLEÇÃO ORDENADA E MUTÁVEL. NENHUM MEMBRO DUPLICADO.

CRITÉRIO	LIST	TUPLE	SET	DICTIONARY
ORDENAÇÃO	Ordenada	Ordenada	Não ordenada	Não ordenada
MODIFICAÇÃO	Mutável	Imutável	Mutável	Mutável
DUPLICATAS	Permite duplicatas	Permite duplicatas	Não permite duplicatas	Não permite duplicatas
INDEXAÇÃO	Por inteiro	Por inteiro	Não indexada	Por string
DELIMITADOR	Entre colchetes [ ]	Entre parênteses ( )	Entre chaves { }	Entre chaves { }

CATEGORIA	TIPO	PYTHON	EXEMPLO
BOOLEANO	Booleano	bool	`x = True` `x = False`
NUMÉRICO	Inteiro	int	`x = 10` `x = -5`
	Ponto Flutuante	float	`x = 10.7` `x = -2.8`
	Complexo	complex	`x = 345j` `x = 2-9j`
TEXTUAL	Texto	str	`x = 'texto'` `x = "texto"`
COLEÇÃO/SEQUÊNCIA	Lista	list	`x = [4, 8]` `x = list()`
	Tupla	tuple	`x = (5, 10)` `x = tuple()`
	Set	set	`x = {2, 4}` `x = set( )`
	Dicionário	dictionary	`x = {'nome': 'Diego', idade: 31}`

(CESPE – PC PB– 2022) Python é uma linguagem procedural que utiliza quatro tipos de dados predefinidos para lidar com coleções: conjuntos, dicionários, listas e tuplas. A respeito desses tipos de dados, julgue os itens a seguir.

I O conjunto permite o armazenamento de uma tupla, mas não o de uma lista.
II A tupla é idêntica à lista, exceto pela forma mais simples com que sua declaração é realizada.
III A lista é um tipo de dados variável que permite a alteração de seus elementos após a sua criação.
Assinale a opção correta.

a) Apenas o item I está certo.
b) Todos os itens estão certos.
c) Apenas o item II está certo
d) Apenas os itens I e III estão certos.
e) Apenas os itens II e III estão certos.

Comentários: Pessoal, vamos começar pelo erro do item II: Uma tupla é uma estrutura bastante similar a uma lista, com uma diferença especial: os elementos inseridos em uma tupla não podem ser alterados, o que não ocorre em uma lista, já que nesta podem ser alterados livremente. Ademais, vejamos o item I: conjunto permite o armazenamento de uma tupla, mas não o de uma lista. Está correto o item I, porque os elementos de um conjunto não são armazenados em uma ordem específica além disso, conjuntos não contém elementos repetidos. Portanto, conjuntos são diferentes de listas. Por fim, o item III está correto, é possível a alteração dos elementos de uma lista. (Gabarito: Letra D).

Condições do Python

Python suporta as condições lógicas usuais da matemática. Essas condições podem ser usadas de várias maneiras, mais comumente em "**instruções if**" e loops. Uma "**instrução if**" é escrita usando a palavra-chave **if** .

```
a = 33
b = 200
if b > a:
  print("B é maior que A")
```

Vejamos as condições lógicas usuais da matemática que podem ser utilizadas.

Igual a: a == b
Diferentes: a != b
Menor que: a < b
Menor ou igual a: a <= b
Maior que: a > b
Maior ou igual a: a >= b

Pessoal, Python é diferente de outras linguagens como C e Java, que usam colchetes para definir o escopo de um código (função ou classe, por exemplo). Python usa a indentação (espaço em branco no início de uma linha) para definir o **escopo no código**. Veja que no exemplo anterior, o print após a instrução IF(`print("B é maior que A")`) possui uma indentação (espaço em branco). Isso faz com que o que possui recuo está dentro da **instrução if**.

Além da instrução if, há a instrução elif. Elif nada mais é que um else com um if... ou seja, senão se. 😂 Elif é uma maneira de dizer "se as condições anteriores não forem verdadeiras, tente esta condição".

```
a = 33
b = 33
if a > b:
  print("A é maior que B")
if b > a:
  print("B é maior que A")
elif a == b:
  print("A e B são iguais")
```

Neste caso, será impresso na tela que "a e b são iguais".

Pessoal, o if é muito utilizado na prática, além disso **cai muito em provas**! Além dele, temos o else, que é exatamente o contrário... a instrução if testa se uma condição é verdadeira, já o else, captura o resultado do que não "entrar" na instrução if. Um ótimo exemplo para isso é testar se uma pessoa é maior ou menor de idade. Vejamos um exemplo

```
idade = 18
if idade >= 18:
    print('maior de idade')
else:
    print('menor de idade')
```

A palavra-chave **else** captura qualquer coisa que não seja capturada pelas condições anteriores.

```
idade = 18
if idade < 12:
    print('crianca')
elif idade < 18:
    print('adolescente')
elif idade < 60:
    print('adulto')
else:
    print('idoso')
```

Além de todas essas formas de criar uma condição, é possível criar uma condição em uma única linha:

```
if a > b: print("A é maior que B")
```

O mais incrível ainda é criar um **if e else** em apenas uma linha! É possível em Python! Veja o exemplo:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

O exemplo acima, irá imprimir na tela B, já que b = 330. Você também pode ter várias instruções **else** na mesma linha:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

A palavra-chave **and** é um operador lógico e é usada para **combinar instruções condicionais**:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Ambas condições são verdadeiras")
```

Além disso temos a palavra-chave **or** que é outro um operador lógico usado para combinar instruções condicionais. Ele retorna True se **uma das condições for verdadeiras**, caso contrário retorna False

```
a = 200
b = 33
c = 500
if a > b or c > a:
  print("Uma das condições é verdadeira")
```

Por fim, a palavra-chave **not** o inverte o resultado: se o resultado da expressão for True, o operador retorna false. 🙃

OPERADOR	DESCRIÇÃO	EXEMPLO
AND	Retorna True se todas as condições forem verdadeiras, caso contrário retorna False	x > 1 and x < 5
OR	Retorna True se uma das condições for verdadeiras, caso contrário retorna False	x > 1 or x < 5
NOT	Inverte o resultado: se o resultado da expressão for True, o operador retorna false	not(x > 1 and x < 5)

Python While Loops

Com o loop **while** podemos executar um conjunto de instruções desde que uma condição seja verdadeira. O loop **while** requer que as variáveis relevantes estejam prontas, no exemplo abaixo precisamos definir uma variável de indexação, **i** , que definimos como 1.

```
i = 1
while i < 6:
  print(i)
  i += 1 # imprime 1 2 3 4 5 (um abaixo do outro)
```

Além disso, podemos parar uma estrutura de repetição utilizando a instrução break. Com a instrução **break** podemos parar o loop mesmo se a condição while for verdadeira. Por outro lado, a instrução **continue** para a iteração atual e continuar com a próxima.

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

No exemplo acima, será impresso na tela 1 2 3. E irá parar quando entrar no if i == 3 pois foi inserido um comando break dentro dessa condicional.

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

No exemplo acima, serão impressos todos os números da sequência de 1 a 6 exceto o 3. Lembre-se que a instrução **continue** interrompe a iteração atual e **continua com a próxima.** Portanto, quando o valor da variável i for 3, vai pular o print e prosseguir o loop.

(FGV – FunSaúde CE– 2021) Observe o código Python v2.7

```
def F (a, b):
  while a != b:
```

```
    if a > b:
        a = a - b
    elif b > a:
        b -= a
  return a
```

Assinale o valor retornado para F (48,36).

a) 1
b) 12
c) 24
d) 36
e) 48

Comentários: O código consiste em um loop seguido de um condicional. O loop while roda enquanto a variável a tiver um valor diferente de b, assim, resolve-se a condicional. Como 48 é diferente de 36, entramos no condicional, se 48>36 então, a vai receber a – b. a = 46 e b = 36. A vai receber = 12 e retorna esse valor que será impresso. Assim, nosso gabarito é a letra B. (Gabarito: Letra B).

Loop for em Python

Com toda certeza esse é um dos tópicos mais importantes da aula! Afinal, o que é um loop **for**? Um loop **for** é usado para **iterar sobre uma sequência** (que é uma **lista, uma tupla, um dicionário, um conjunto ou uma string**). Traduzindo para o português, for é "para", ou seja, a sintaxe é: para uma lista de valores de 1 até 10 (por exemplo), faça isso. Um bom exemplo: para um vetor de 10 variáveis imprima o valor de cada variável. Isso seria traduzido em algo como:

```
numeros = [1,2,3,4,5,6,7,8,9,10]
for x in numeros:
  print(x)
```

Se você quiser imprimir as letras de seu nome, você pode fazer um loop for para brincar um pouco. Pode fazer isso com qualquer string! Uma string consiste em uma sequencia de caracteres.

```
umaStringQualquer = "Estrategia Concursos"
for x in umaStringQualquer:
  print(x)
```

Usando o loop **for** é possível executar um conjunto de instruções, **uma vez para cada item** de uma lista, tupla, conjunto etc. É possível usar o comando **break** e **continue** no loop for.

Uma das formas mais usadas, e mais cobradas em provas é utilizar o range(). O range() retorna uma **sequência de números**, começando do 0(zero) (por padrão), incrementando por 1 (por padrão) e termina no número especificado. Ela pode ser utilizada em conjunto com a função **for** para iterar sobre **um bloco de instruções por um número específico de vezes**. Vamos ver como funciona:

```
for x in range(6):
  print(x) #Imprimirá 0 1 2 3 4 5
```

Lembrem-se também que essa função pode ser um pouco mais complexa quando possui todos os seus três parâmetros: **start, stop e step**. Vejamos um exemplo:

```
for x in range(2, 20, 3):
  print(x) #Imprimirá 2 5 8 11 14 17
```

Note que o for itera sobre o conjunto de valores que se inicia em 2 (incluso) até 20 (não incluso), pulando de três em três unidades. Agora vejamos a quantidade de questões sobre loop for

(FGV – TCU – 2022) A execução desse código na IDLE Shell produz, na ordem e exclusivamente, os números:

```
def xpto(S):
    for k in range(0,len(S)):
        if k%2 == 0:
            yield(S[k]);

S=[1,2,3,4,5,6]
for x in xpto(S[::-1]):
    print (x)
```

a) 6, 1
b) 5, 3, 1
c) 6, 4, 2
d) 1, 3, 5
e) 2, 4, 6.

Comentários: Pessoal, precisamos entender o que a função faz. xpto consiste em uma função com um for que percorre S. Caso o valor k da lista S seja par (2, 4, 6), então ele cria uma lista (S) com os valores pares (2, 4, 6). Veja que o retorno da função é exatamente a lista gerada pelo yield(S[k]). Agora vamos para o código principal. Após a criação da lista S=[1,2,3,4,5,6], temos for x in xpto(S[::-1]): nesse ponto devemos ficar atentos ao -1. O -1 inverte a lista. Enviando para xpto a lista [6,5,4,3,2,1]. Assim, quando passar pela função xpto, teremos como resultado apenas os valores pares de [6,5,4,3,2,1], ou seja, 6,4,2. Portanto o gabarito é a letra c. (Gabarito: Letra C).

(FGV – MPE GO– 2022) Assinale a lista de números produzida pela execução, na IDLE Shell 3.9.9, do código Python a seguir.

```
for x in range(-1, -10, -1):
 print (x)
```

a) -1 -2 -3 -4 -5 -6 -7 -8 -9
b) -9 -8 -7 -6 -5 -4 -3 -2 -1
c) 0 -1 -2 -3 -4 -5 -6 -7 -8 -9
d) 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
e) -1 -2 -3 -4 -5 -6 -7 -8 -9 -10

Comentários: Pessoal, as bancas gostam muito de índices negativos, para dificultar um pouco. No caso da questão, é gerada uma sequência de números negativos iniciando em -1 e parando antes do -10 (ou seja, o for para no -9). O For é incrementado de -1 em -1 formando a sequência -1 -2 -3 -4 -5 -6 -7 -8 -9. (Gabarito: Letra A).

(FGV – PC AM– 2022) Considere o código Python a seguir.

```
L=[0,1,1,2,3,5,8,13,21]
for k in range(0,len(L),2):
print (L[k])
```

Assinale o resultado exibido pela execução desse código, na IDLE Shell 3.9.9.

a) 1, 2, 5, 13
b) 0, 1, 3, 8, 21, 1, 2, 5, 13, 21
c) 1, 3, 8, 21, 1, 2, 5, 13, 21
d) 0, 1, 3, 8, 21
e) 0, 1, 3, 8

Comentários: A questão consiste em um for que percorre a lista L e avança de 2 em 2 posições (for k in range(0,len(L),2):). Ele percorre sempre de 2 em 2 posições, portanto, a variável k procurará os valores das posições pares: Posições 0,2,4,6,8. Tenham em mente que o valor da posição 0 é 0, da posição 1 é 1, da posição 2 é 1, da posição 3 é 2 e assim por diante, veja: L=[0,1,1,2,3,5,8,13,21]. Se for necessário, desenhe as listas (uma em cima da outra) POSIÇÕES = [0,1,2,3,4,5,6,7,8,9] LISTA = [0,1,1,2,3,5,8,13,21].(Gabarito: Letra D).

(FGV – MPE SC– 2022) Analise o código Python a seguir.

```
s=0
for k in range(16,10, -2):
s -= k
print (s)
```

O valor exibido pela execução desse trecho é:

a) 0
b) -28
c) -30
d) -42
e) -52

Comentários: Pessoal, for k in range(16,10, -2):gera 16,14 e 12. Nessa sequência. s -= k vai somar os valores negativos da seguinte forma: -16 - 14 e -12 que gera como resultado -42.(Gabarito: Letra D).

Funções do Python

Uma função é um bloco de código que só é executado quando é chamado. Você pode passar dados, conhecidos como **parâmetros**, para uma função. Essa função pode **retornar dados como resultado**. Para criar uma função, utiliza-se a palavra-chave **def** :

```
def minhafuncao():
  print("Olá Mundo!")
```

Para chamar uma função, use o nome da função seguido de parênteses:

```
minhafuncao()
```

As informações podem ser passadas para funções como **argumentos**. Os argumentos são especificados **após o nome da função**, dentro dos parênteses. Você pode adicionar quantos argumentos quiser, basta separá-los com uma vírgula. O exemplo a seguir tem uma função com um argumento (nome). Quando a função é chamada, passamos um primeiro nome, que é usado dentro da função para imprimir o nome completo:

```
def minhafuncao(nome):
  print("Prof. " + nome)

minhafuncao("Diego Carvalho")
minhafuncao("Raphael Lacerda")
minhafuncao("Paolla Ramos")
```

Por padrão, uma função deve ser chamada com o número correto de argumentos. Isso significa que, se sua função espera 2 argumentos, você deve chamar a função com 2 argumentos, nem mais, nem menos. Além disso, se você tentar chamar a função com mais ou menos argumentos, receberá um erro. No caso da função do exemplo, se passar um argumento a menos, receberá o erro: **missing 1 required positional argument**, por outro lado, se inserir dois argumentos quando era necessário um, receberá o seguinte erro: **takes 1 positional argument but 2 were given**.

Se você não souber quantos argumentos serão passados para sua função, adicione um *antes do nome do parâmetro na definição da função. Dessa forma, a função receberá uma tupla de argumentos. Vejamos um exemplo.

```
def minhafuncao (*professores):
  print( professores[1] + " é um professor de TI do Estratégia")

minhafuncao("Diego Carvalho", "Raphael Lacerda")
```

No exemplo anterior, será impresso Raphael Lacerda é professor de TI do Estratégia. A tupla professores possui dois valores: ("Diego Carvalho", "Raphael Lacerda"). A posição o refere-se a Diego Carvalho e a posição 1 refere-se a Raphael Lacerda. Vejamos agora como passar argumentos na função usando chave-valor. Desta forma, a ordem dos argumentos não importa.

```
def minhafuncao (prof1, prof2, prof3):
  print( prof1 + " é um professor de TI do Estratégia")

minhafuncao(prof1 = "Diego Carvalho", prof2 = "Raphael Lacerda", prof3 = "Paolla
Ramos")
```

Se você não souber quantos argumentos de palavra-chave serão passados para sua função, adicione dois asteriscos: ** antes do nome do parâmetro na definição da função. Desta forma a função receberá um dicionário de argumentos.

Outra forma de passar argumentos é enviando uam função. Você pode enviar qualquer tipo de dado de argumento para uma função (string, número, lista, dicionário etc.), e ele será tratado como o mesmo tipo de dado dentro dela. Por exemplo, se você enviar uma **lista** como argumento, ela ainda será uma lista quando chegar à função:

```
def minhafuncao(professores):
  for x in professores:
    print(x)

listadeprofessores("Diego Carvalho", "Raphael Lacerda", "Paolla Ramos")
minhafuncao(listadeprofessores)
```

Para permitir que uma função retorne um valor, use a instrução **return**

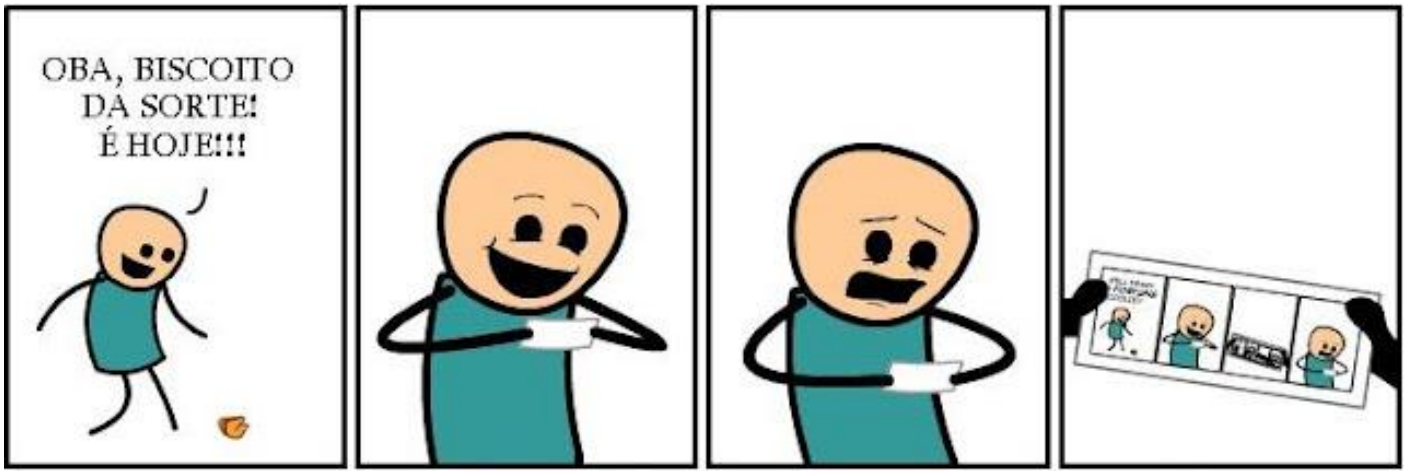
```
def funcao(x):
  return 5 * x

print(funcao(3)) # imprime 15
print(funcao(5)) # imprime 25
print(funcao(9)) # imprime 45
```

Python também aceita **funções recursivas**, o que significa que **uma função definida pode chamar a si mesma.** A recursão é um conceito matemático e de programação comum. Isso tem a vantagem de significar que você pode percorrer os dados para chegar a um resultado.

O desenvolvedor deve ter muito **cuidado** com a recursão, pois pode ser muito fácil escrever uma função que **nunca termina**, ou uma que usa **quantidades excessivas de memória** ou poder do

processador. No entanto, quando **escrita corretamente**, a recursão pode ser uma abordagem de programação **muito eficiente e matematicamente elegante**.



O exemplo mais clássico, e para mim, o melhor, é o fatorial em python! É sério galera, é incrível! Ou no mínimo, é assim que você descobre como Python é incrível! Por exemplo, se você atribuir 5 a uma variável "valor", e 1 a uma variável "fatorial", teremos o seguinte código em Python calculando o fatorial.

```
while (valor > 1):
  fatorial = fatorial * valor
  valor = valor - 1
```

Python Lambda

Uma função lambda é uma pequena função anônima. Ela pode receber qualquer número de argumentos, mas pode ter apenas uma expressão. Vejamos um exemplo de uma função lambda.

```
x = lambda a : a + 10
print(x(5))
```

As funções lambda podem receber qualquer número de argumentos. Por exemplo, vejamos como multiplicar um argumento a por argumento b e retornar o resultado:

```
x = lambda a, b: a * b
print(x(5, 6))
```

O poder do lambda é melhor apresentado quando você os usa **uma função anônima dentro de outra função**. Digamos que você tenha uma definição de função que recebe um argumento e esse argumento será multiplicado por um número desconhecido:

```
def minhafuncao(n):
  return lambda a: a * n
```

Usando essa definição de função, é possível para fazer uma função que **sempre duplique o número que você envia**:

```
def minhafuncao (n):
  return lambda a : a * n

dobravalor = minhafuncao (2)

print(dobravalor(10)) #imprime 20
```

Matrizes ou Arrays Python

Arrays são usados para **armazenar vários valores em uma única variável**. O Python **não possui suporte integrado para Arrays**, mas as Listas do Python podem ser usadas. Para trabalhar com arrays em Python você terá que importar uma biblioteca, como a biblioteca **NumPy** [4].

Um array é uma variável especial, que pode conter mais de um valor por vez.

Se você tiver uma lista de itens (uma lista de nomes de carros, por exemplo), armazenar os carros em variáveis únicas pode ficar conforme o código a seguir.

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```

No entanto, e se você quiser percorrer os carros e encontrar um específico? E se você não tivesse 3 carros, mas 300? A solução é um **array**! Um **Array** pode conter **muitos valores em um único nome** e você pode **acessar os valores referindo-se a um número de índice**. Assim, é possível acessar um valor da seguinte forma:

```
carros = ["Ford", "Volvo", "BMW"]
x = carros[0] # x recebe Ford.
```

Use o método **len()** para retornar o comprimento de uma matriz (o número de elementos em uma matriz). É possível usar o método **append()** para **adicionar um elemento a uma matriz**. Você também pode usar o método **pop()** para remover um elemento da matriz.

```
carros = ["Ford", "Volvo", "BMW"]
x = len(carros) #verifica o tamanho do array
carros.append("Honda") #adiciona Honda ao fim do array
carros.pop(1) #remove o carro da posição 1
```

É possível usar o método **remove()** para remover um elemento da matriz. A diferença entre o **remove()** e o **pop()** é que o primeiro remove **apenas a primeira ocorrência do valor especificado**.

MÉTODO	DESCRIÇÃO

[4] NumPy é uma biblioteca Python, usada para trabalhar com arrays. NumPy consiste em uma abreviação de "Pyton Numérico".

APPEND()	Adiciona um elemento no final da lista
CLEAR()	Remove todos os elementos da lista
COPY()	Retorna uma cópia da lista
COUNT()	Retorna o número de elementos com o valor especificado
EXTEND()	Adicione os elementos de uma lista (ou qualquer iterável) ao final da lista atual
INDEX()	Retorna o índice do primeiro elemento com o valor especificado
INSERT()	Adiciona um elemento na posição especificada
POP()	Remova o elemento na posição especificada
REMOVE()	Remova o primeiro item com o valor especificado
REVERSE()	Inverte a ordem da lista
SORT()	Ordena a lista

(FCC – PGE AM– 2022) Em um programa escrito em Python, uma série de dados foram inseridos no array cargos, por meio da instrução abaixo.

cargos = ["Advogado","Promotor", "Procurador", "Juiz", "Desembargador", "Ministro"];

Para colocar estes dados em ordem alfabética decrescente em um novo array chamado cargos_ordenados utiliza-se a instrução:

a) for cargos_ordenados in cargos reverse True;
b) cargos_ordenados = sorted(cargos,reverse=True);
c) while cargos: cargos_ordenados = cargos- -;
d) cargos_ordenados = descending(cargos);
e) cargos_ordenados = ordered(cargos, descending=True);

_____

Comentários: Pessoal, sort() ordena a lista, no caso, a lista "cargos". A opção reverse=True Inverte a ordem da lista. Portanto nosso gabarito é a Letra B. O resultado da execução da letra B é: ['Promotor', 'Procurador', 'Ministro', 'Juiz', 'Desembargador', 'Advogado']. As demais alternativas geram erros e não existem. (Gabarito: Letra B).

Pessoal, como as bancas estão cobrando Numpy, vamos ver os conceitos básicos sobre essa biblioteca! O que é NumPy? NumPy é uma biblioteca Python usada para trabalhar com arrays. Além disso, possui funções para trabalhar no domínio da álgebra linear, transformada de Fourier e matrizes.

O NumPy foi criado em 2005 por Travis Oliphant. É um projeto de código aberto e você pode usá-lo livremente. NumPy significa **Python Numérico**. Em Python temos listas que servem ao propósito

de arrays, mas são lentas para processar. O NumPy visa fornecer um objeto array **até 50x mais rápido que as listas tradicionais do Python**.

O objeto array no NumPy é chamado ndarray, ele fornece muitas funções de suporte que facilitam ndarraymuito o trabalho.

Os arrays são usados com muita frequência na ciência de dados, onde a velocidade e os recursos são muito importantes.

Classes/objetos Python

Python é uma linguagem de programação **orientada a objetos**. Quase tudo em Python é um objeto, com suas propriedades e métodos. Uma classe é como um **construtor de objetos**, ou um **"projeto" para criar objetos.**

```
class MinhaClasse:
  x = 5
```

Para entender o significado das classes, temos que entender a função embutida **__init__()**. Todas as classes possuem uma função chamada **__init__(),** que sempre é executada quando a classe está sendo iniciada. Use a função **__init__()** para **atribuir valores às propriedades do objeto** ou outras operações que **são necessárias quando o objeto está sendo criado**.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

A função __str__() controla o que deve ser retornado quando o objeto de classe é representado como uma string. Se a função __str__() não for definida, a representação em string do objeto é retornada:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```

Objetos também podem conter métodos. Métodos em objetos são funções que pertencem ao objeto.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

O parâmetro **self** é uma **referência à instância atual da classe** e é usado para **acessar as variáveis que pertencem à classe.** Ele **não precisa ser nomeado self**, você pode chamá-lo como quiser, mas **deve ser o primeiro parâmetro de qualquer função da classe**:

(FGV – TJDFT– 2022) Analise o código Python 3.9 a seguir.
class Teste:
def ----------
self.altura = xaltura
self.largura = xlargura
def dimensoes(self):
print("altura = " + str(self.altura) + "\n" \ + "largura = " + str(self.largura))
x = Teste(12, 20)
x.dimensoes()

Para que a execução desse código exiba
altura = 12
largura = 20
o trecho tracejado (----------) na segunda linha deve ser substituído por:

a) __init__(self, xaltura, xlargura):
b) __init__(xaltura, xlargura):
c) init (xaltura, xlargura):
d) new (self, args[xaltura, xlargura]):
e) new (self, xaltura, xlargura):

_____

Comentários: Pessoal, utiliza-se a função __init__() para atribuir valores às propriedades do objeto. Sendo assim ficamos apenas com as opções a e b. Além de init, é necessário usar o self, que é uma referência à instância atual da classe. Portanto, o correto é a letra A. (Gabarito: Letra A).

Herança Python

A herança nos permite definir uma classe que herda todos os métodos e propriedades de outra classe. A classe pai é a classe que está sendo herdada, também chamada de classe base. A classe filha é a classe que herda de outra classe, também chamada de classe derivada. Qualquer classe pode ser uma classe pai, então a sintaxe é a mesma da criação de qualquer outra classe.

Para criar uma classe que herde a funcionalidade de outra classe, envie a classe pai como parâmetro ao criar a classe filha.

```
class Student(Person):
  pass
```

Use a palavra-chave **pass** quando não desejar adicionar outras propriedades ou métodos à classe. Agora a classe **Student** têm as mesmas propriedades e métodos que a classe Person. Use a classe **Student** para criar um objeto e execute o método **printname**

```
x = Student("Mike", "Olsen")
x.printname()
```

Até agora criamos uma **classe filha** que **herda** as propriedades e métodos de seu **pai**. Queremos adicionar a função **__init__()** à classe filha (em vez da palavra- chave **pass**).

```
class Student(Person):
  def __init__(self, fname, lname):
```

Quando você adiciona a função __init__(), a classe filha **não herdará mais** a função __init__() **do pai**. A função __init__() do filho substitui a herança da função __init__() do pai. Por outro lado, caso deseje manter a herança da função__init__() do pai, adicione uma chamada à função __init__() do pai da seguinte forma:

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

Python também tem uma função **super()** que fará com que a classe filha **herde todos os métodos e propriedades de seu pai**. Ao usar a função **super(),** você **não precisa usar o nome do elemento pai**, ele **herdará automaticamente** os métodos e propriedades de seu pai.

```
class Student(Person):
  def __init__(self, fname, lname):
```

```
    super().__init__(fname, lname)
```

Iteradores Python

Um iterador é um objeto que contém um número contável de valores. Utilizando um iterador, podemos percorrer todos os valores de um objeto iterável (ex: array). Tecnicamente, em Python, um iterador é um objeto que implementa o protocolo do **iterador**, que consiste nos **métodos __iter__() e __next__().**

Listas, tuplas, dicionários e conjuntos são todos objetos iteráveis. Eles são **contêineres iteráveis** dos quais você pode obter um iterador. Todos esses objetos têm ummétodo **iter()** que é usado para obter um iterador. Como sabemos, strings consistem em uma contendo uma sequência de caracteres, assim, **strings** são objetos iteráveis e **podem retornar um iterador**

```
minhatupla = ("maça", "banana", "cereja")
meuiterador = iter(minhatupla)

print(next(meuiterador))
print(next(meuiterador))
print(next(meuiterador))
```

Também podemos usar um loop **for** para **iterar através de um objeto iterável**. O loop **for,** na verdade, cria um objeto **iterador** e **executa o método next()** para cada loop.

```
minhatupla = ("maça", "banana", "cereja")

for x in minhatupla:
  print(x)
```

Para evitar que a iteração **continue para sempre**, podemos usar a instrução **StopIteration**.

No método **__next__()**, podemos **adicionar uma condição de término** para gerar um erro se a iteração for feita um número especificado de vezes:

Escopo do Python

Uma variável só está disponível dentro da região em que foi criada. Isso é chamado de escopo. Uma variável criada dentro de uma função pertence ao **escopo local** dessa função e **só pode ser usada dentro dessa função.**

```
def funcao():
  x = 300
  print(x)

funcao()
```

Conforme explicado no exemplo acima, **a variável x não está disponível fora da função**, mas está disponível para qualquer função dentro da função. Veja um exemplo em que a **variável local** pode ser acessada de uma função dentro da função.

```
def funcao():
  x = 300
  def funcaointerna():
    print(x)
  funcaointerna()

funcao()
```

Vejamos agora como funciona uma variável global! Uma variável criada no corpo principal do código Python é uma **variável global e pertence ao escopo global**. As **variáveis globais** estão disponíveis em **qualquer escopo, global e local**. Uma variável criada fora de uma função é global e pode ser usada por qualquer pessoa.

```
x = 300 #x é uma variável global!

def myfunc():
  print(x)

myfunc()

print(x) #print for a do escopo da funcao!
```

Se você operar com o **mesmo nome de variável dentro e fora de uma função**, o Python as tratará como **duas variáveis separadas**, uma disponível no escopo global (fora da função) e outra disponível no escopo local (dentro da função):

```
x = 300

def myfunc():
  x = 200
  print(x) #imprime 200

myfunc()

print(x) #imprime 300
```

Se você precisar criar uma variável global, mas estiver preso no escopo local, poderá usar a palavra-chave **global**. A palavra-chave **global** torna a variável global. Além disso, pode-se usar global se quiser fazer uma **alteração em uma variável global dentro de uma função**.

```
def myfunc():
  global x
  x = 300
```

Módulos Python

Considere um módulo como sendo o mesmo que uma **biblioteca de código**. Um arquivo contendo um **conjunto de funções que você deseja incluir em seu aplicativo**. Para criar um módulo basta salvar o código desejado em um arquivo com a extensão de arquivo **.py.** Se você criar um código em um arquivo e nomeá-lo como meumodulo.py, você poderá usá-lo invocando "import meumodulo".Além disso, é possível criar um **alias** ao importar um módulo, usando a palavra-chave as, da seguinte forma.

```
import meumodulo as mx
```

O módulo pode conter funções, e variáveis de todos os tipos (arrays, dicionários, objetos etc).

Existem **vários módulos integrados no Python**, que você pode importar sempre que quiser.

```
import platform

x = platform.system()
print(x)
```

Existe uma função interna para listar todos os nomes de funções (ou nomes de variáveis) em um módulo. A função **dir():**

```
import platform

x = dir(platform)
print(x)
```

Datas em Python

Uma data em Python não é um tipo de dado próprio, mas podemos importar um módulo chamado **datetime** para trabalhar com datas como **objetos de data**.

```
import datetime

x = datetime.datetime.now()
```

Quando executamos o código do exemplo acima o resultado será a data do momento atual. A data contém **ano, mês, dia, hora, minuto, segundo e microssegundo**. O módulo **datetime** possui muitos métodos para retornar informações sobre o objeto de data.

Para criar uma data, podemos usar a classe **datetime()** (construtor) do módulo **datetime**. A classe **datetime()** requer **três parâmetros para criar uma data**: ano, mês, dia, conforme pode ser visto no exemplo abaixo. Essa classe também **recebe parâmetros** para hora e fuso horário (hora, minuto, segundo, microssegundo, tzone), mas eles **são opcionais** e têm um valor padrão de o, (None para fuso horário).

```
import datetime

x = datetime.datetime(2022, 1, 20)

print(x) #imprime 2022-01-20 00:00:00
```

O objeto datetime tem um **método para formatar objetos de data** em strings legíveis. O método é chamado strftime()e usa um parâmetro, **format**, para especificar o **formato da string retornada**:

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

DIRETIVA	DESCRIÇÃO	EXEMPLO
%A	Dia da semana, versão curta	Wed
%A	Dia da semana, versão completa	Wednesday
%W	Dia da semana como um número 0-6, o é domingo	3
%D	Dia do mês 01-31	31
%B	Nome do mês, versão curta	Dec

%B	Nome do mês, versão completa	December
%M	Mês como um número 01-12	12
%Y	Ano, versão curta, sem século	18
%Y	Ano, versão completa	2018
%H	Hora 00-23	17
%I	Hora 00-12	05
%P	MANHÃ TARDE	PM
%M	Minuto 00-59	41
%S	Segundo 00-59	08
%F	Microssegundo 000000-999999	548513
%Z	Deslocamento UTC	+0100
%Z	Fuso horário	CST
%J	Número do dia do ano 001-366	365
%U	Número da semana do ano, domingo como o primeiro dia da semana, 00-53	52
%W	Número da semana do ano, segunda-feira como o primeiro dia da semana, 00-53	52
%C	Versão local de data e hora	Mon Dec 31 17:41:00 2018
%C	Século	20
%X	Versão local da data	12/31/18
%X	Versão local do tempo	17:41:00
%%	Um personagem	%
%G	ISO 8601 ano	2018
%U	Dia da semana ISO 8601 (1-7)	1
%V	Número da semana ISO 8601 (01-53)	01

Matemática Python

O Python possui um conjunto de funções matemáticas integradas, incluindo um extenso módulo matemático, que permite realizar tarefas matemáticas em números. As funções min()e max() podem ser usadas para encontrar o valor mais baixo ou mais alto em um iterável.

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x) #imprime 5
print(y) #imprime 25
```

A função **abs()** retorna o valor absoluto (positivo) do número especificado

```
x = abs(-7.25) # x recebe 7.25 positivo
```

DIRETIVA	DESCRIÇÃO
MATH.ACOS()	Retorna o arco cosseno de um número
MATH.ACOSH()	Retorna o cosseno hiperbólico inverso de um número
MATH.ASIN()	Retorna o arco seno de um número
MATH.ASINH()	Retorna o seno hiperbólico inverso de um número
MATH.ATAN()	Retorna o arco tangente de um número em radianos
MATH.ATAN2()	Retorna o arco tangente de y/x em radianos
MATH.ATANH()	Retorna a tangente hiperbólica inversa de um número
MATH.CEIL()	Arredonda um número para o inteiro mais próximo
MATH.COMB()	Retorna o número de maneiras de escolher k itens de n itens sem repetição e ordem
MATH.COPYSIGN()	Retorna um float que consiste no valor do primeiro parâmetro e o sinal do segundo parâmetro
MATH.COS()	Retorna o cosseno de um número
MATH.COSH()	Retorna o cosseno hiperbólico de um número
MATH.DEGREES()	Converte um ângulo de radianos para graus
MATH.DIST()	Retorna a distância euclidiana entre dois pontos (p e q), onde p e q são as coordenadas desse ponto
MATH.ERF()	Retorna a função de erro de um número
MATH.ERFC()	Retorna a função de erro complementar de um número
MATH.EXP()	Retorna E elevado à potência de x
MATH.EXPM1()	Retorna Ex - 1

MATH.FABS()	Retorna o valor absoluto de um número
MATH.FACTORIAL()	Retorna o fatorial de um número
MATH.FLOOR()	Arredonda um número para baixo para o inteiro mais próximo
MATH.FMOD()	Retorna o restante de x/y
MATH.FREXP()	Retorna a mantissa e o expoente, de um número especificado
MATH.FSUM()	Retorna a soma de todos os itens em qualquer iterável (tuplas, arrays, listas, etc.)
MATH.GAMMA()	Retorna a função gama em x
MATH.GCD()	Retorna o máximo divisor comum de dois inteiros
MATH.HYPOT()	Retorna a norma euclidiana
MATH.ISCLOSE()	Verifica se dois valores estão próximos ou não
MATH.ISFINITE()	Verifica se um número é finito ou não
MATH.ISINF()	Verifica se um número é infinito ou não
MATH.ISNAN()	Verifica se um valor é NaN (não é um número) ou não
MATH.ISQRT()	Arredonda um número de raiz quadrada para baixo para o inteiro mais próximo
MATH.LDEXP()	Retorna o inverso de math.frexp() que é x * (2**i) dos números fornecidos x e i
MATH.LGAMMA()	Retorna o valor de log gama de x
MATH.LOG()	Retorna o logaritmo natural de um número ou o logaritmo do número para a base
MATH.LOG10()	Retorna o logaritmo de base 10 de x
MATH.LOG1P()	Retorna o logaritmo natural de 1+x
MATH.LOG2()	Retorna o logaritmo de base 2 de x
MATH.PERM()	Retorna o número de maneiras de escolher k itens de n itens com ordem e sem repetição
MATH.POW()	Retorna o valor de x elevado a y
MATH.PROD()	Retorna o produto de todos os elementos em um iterável
MATH.RADIANS()	Converte um valor de grau em radianos
MATH.REMAINDER()	Retorna o valor mais próximo que pode tornar o numerador completamente divisível pelo denominador
MATH.SIN()	Retorna o seno de um número
MATH.SINH()	Retorna o seno hiperbólico de um número
MATH.SQRT()	Retorna a raiz quadrada de um número
MATH.TAN()	Retorna a tangente de um número
MATH.TANH()	Retorna a tangente hiperbólica de um número
MATH.TRUNC()	Retorna as partes inteiras truncadas de um número

Python RegEx

Um RegEx, ou Expressão Regular, é uma sequência de caracteres que forma um padrão de pesquisa. RegEx pode ser usado para verificar se uma string contém o padrão de pesquisa especificado.

Python tem um pacote embutido chamado re, que pode ser usado para trabalhar com Expressões Regulares. Depois de importar o módulo re, você pode começar a usar expressões regulares. Por exemplo, caso você queira pesquisar uma **string** para ver se ela **começa com "The"** e termina com **"Spain"**

```
import re

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```

FUNÇÃO	DESCRIÇÃO
FINDALL	Retorna uma lista contendo todas as correspondências
SEARCH	Retorna um objeto Match se houver uma correspondência em qualquer lugar da string
SPLIT	Retorna uma lista onde a string foi dividida em cada partida
SUB	Substitui uma ou várias correspondências por uma string

CARACTER	DESCRIÇÃO	EXEMPLO
[]	Um conjunto de caracteres	"[a-m]"
\	Sinaliza uma sequência especial (também pode ser usada para escapar de caracteres especiais)	"\d"
.	Qualquer caractere (exceto caractere de nova linha)	"he..o"
^	Começa com	"^hello"
$	Termina com	"planet$"
*	Zero ou mais ocorrências	"he.*o"
+	Uma ou mais ocorrências	"he.+o"
?	Zero ou uma ocorrência	"he.?o"
{}	Exatamente o número especificado de ocorrências	"he.{2}o"
|	Ou	"falls|stays"
()	Capturar e agrupar	

Uma sequência especial é \seguida por um dos caracteres na lista abaixo e tem um significado especial.

CARACTER	DESCRIÇÃO	EXEMPLO
\A	Retorna uma correspondência se os caracteres especificados estiverem no início da string	"\AThe"
\B	Retorna uma correspondência em que os caracteres especificados estão no início ou no final de uma palavra (o "r" no início está certificando-se de que a string está sendo tratada como uma "string bruta")	r"\bain" r"ain\b"
\B	Retorna uma correspondência onde os caracteres especificados estão presentes, mas NÃO no início (ou no final) de uma palavra (o "r" no início está certificando-se de que a string está sendo tratada como uma "string bruta")	r"\Bain" r"ain\B"
\D	Retorna uma correspondência em que a string contém dígitos (números de 0 a 9)	"\d"
\D	Retorna uma correspondência onde a string NÃO contém dígitos	"\D"
\S	Retorna uma correspondência onde a string contém um caractere de espaço em branco	"\s"
\S	Retorna uma correspondência em que a string NÃO contém um caractere de espaço em branco	"\S"
\W	Retorna uma correspondência em que a string contém qualquer caractere de palavra (caracteres de a a Z, dígitos de 0 a 9 e o caractere sublinhado _)	"\w"
\W	Retorna uma correspondência em que a string NÃO contém nenhum caractere de palavra	"\W"
\Z	Retorna uma correspondência se os caracteres especificados estiverem no final da string	"Spain\Z"

Um conjunto é um conjunto de caracteres dentro de um par de colchetes [] com um significado especial:

CONJUNTO	DESCRIÇÃO
[arn]	Retorna uma correspondência em que um dos caracteres especificados (a, r ou n) está presente
[a-n]	Retorna uma correspondência para qualquer caractere minúsculo, em ordem alfabética entre a e n
[^arn]	Retorna uma correspondência para qualquer caractere, EXCETO a, r e n
[0123]	Retorna uma correspondência onde qualquer um dos dígitos especificados (0, 1, 2 ou 3) está presente
[0-9]	Retorna uma correspondência para qualquer dígito entre 0 e 9

[0-5][0-9]	Retorna uma correspondência para qualquer número de dois dígitos de 00 a 59
[A-ZA-Z]	Retorna uma correspondência para qualquer caractere em ordem alfabética entre a e z, minúsculas OU maiúsculas
[+]	Em conjuntos, +, *, ., |, (), $,{} não tem significado especial, então [+] significa: retornar uma correspondência para qualquer caractere + na string

RESUMO

RECURSOS NOTÁVEIS DO PYTHON

- SINTAXE ELEGANTE, PROGRAMAS MAIS FÁCEIS DE LER
- LINGUAGEM SIMPLES E FÁCIL DE USAR
- GRANDE BIBLIOTECA PADRÃO QUE SUPORTA MUITAS TAREFAS COMUNS
- MODO INTERATIVO QUE FACILITA O TESTE DE PEQUENOS TRECHOS DE CÓDIGO
- FACILMENTE ESTENDIDO ADICIONANDO NOVOS MÓDULOS
- PODE SER INCORPORADO EM UM APLICATIVO PARA FORNECER UMA INTERFACE PROGRAMÁVEL
- FUNCIONA EM QUALQUER LUGAR

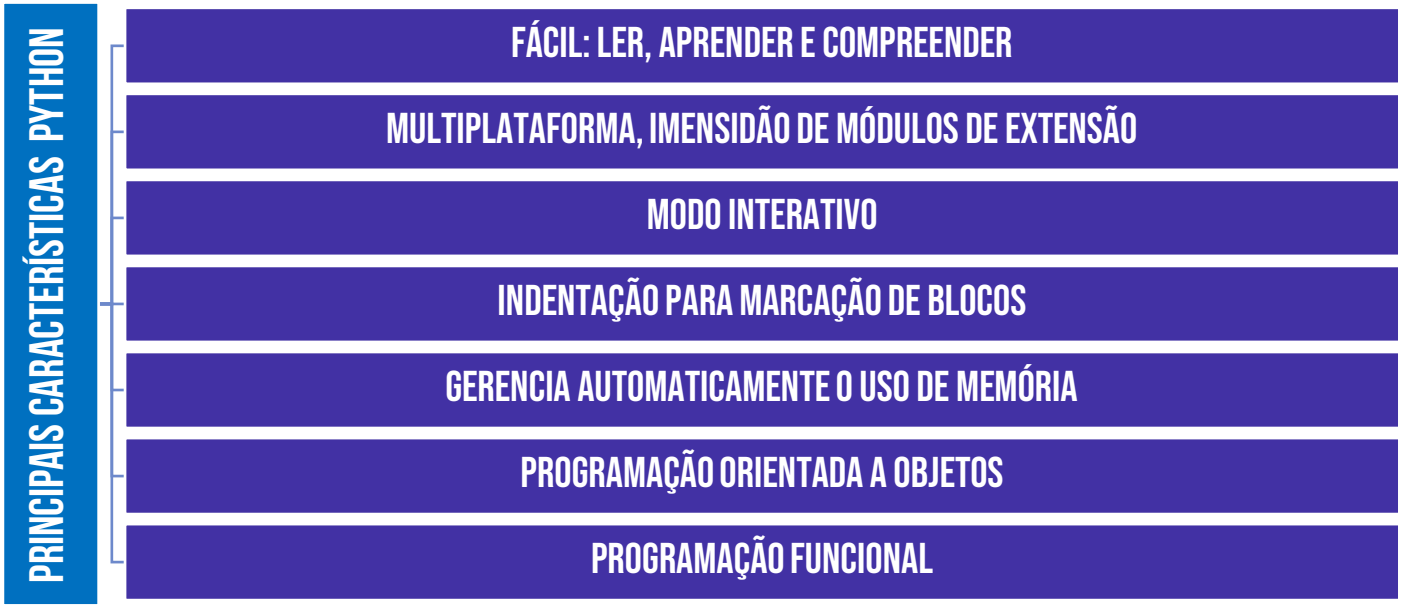
RECURSOS DO PYTHON

- VARIOS DE TIPOS DE DADOS BÁSICOS DISPONÍVEIS
- SUPORTA PROGRAMAÇÃO ORIENTADA A OBJETOS COM CLASSES E HERANÇAS MÚLTIPLAS
- CÓDIGO AGRUPADO EM MÓDULOS E PACOTES
- SUPORTA A GERAÇÃO E CAPTURA DE EXCEÇÕES
- TIPOS DE DADOS SÃO FORTEMENTE E DINAMICAMENTE TIPADOS
- CONTÉM RECURSOS AVANÇADOS DE PROGRAMAÇÃO
- GERENCIAMENTO AUTOMÁTICO DE MEMÓRIA

PALAVRA-CHAVE	DESCRIÇÃO
AND	And é um operador lógico. Os operadores lógicos são usados para combinar instruções condicionais. O valor de retorno será True somente se ambas as instruções retornarem True, caso contrário retornará False.
AS	Usada para criar um alias.
ASSERT	Usada ao depurar o código. Permite testar se uma condição em seu código retorna True, caso contrário, o programa gerará um AssertionError. Você pode escrever uma mensagem a ser escrita caso o código retorne false

BREAK	Usada para quebrar um loop for ou um loop.while
CLASS	Usada para criar uma classe. Uma classe é como um construtor de objetos.
CONTINUE	Usada para encerrar a iteração atual em um loop for (ou loop while) e continua na próxima iteração.
DEF	Usada para criar (ou definir) uma função.
DEL	Usada para excluir objetos. Em Python tudo é um objeto, então a palavra-chave del também pode ser usada para deletar variáveis, listas ou partes de uma lista etc.
ELIF	Usada em instruções condicionais (instruções if) e é abreviação de else if.
ELSE	com a ramificação "if", tenta as ramificações "elif" e termina com a ramificação "else" (até ser avaliada como True)
EXCEPT	Usada em blocos try...except. Ele define um bloco de código a ser executado se o bloco try gerar um erro. Você pode definir blocos diferentes para diferentes tipos de erro e blocos para executar se nada der errado
FALSE	A palavra-chave false é um valor booleano e resultado de uma operação de comparação. A palavra-chave false é igual a o (True é igual a 1).
FINALLY	Usada em blocos try...except. Ele define um bloco de código para ser executado quando o bloco try...except...else for final. O bloco finally será executado independentemente de o bloco try gerar um erro ou não. Isso pode ser útil para fechar objetos e limpar recursos.
FOR	Um loop for é usado para iterar sobre uma sequência (que é uma lista, uma tupla, um dicionário, um conjunto ou uma string). Permite executar um conjunto de instruções, uma vez para cada item de uma lista, tupla, conjunto etc. for i in [0,1,2]: print(i)
FROM	Usada para importar apenas uma seção especificada de um módulo.
GLOBAL	Usada para criar variáveis globais em um escopo não global, por exemplo, dentro de uma função.
IF	Usada para criar instruções condicionais (instruções if) e permite que você execute um bloco de código somente se uma condição for True. Use a palavra-chave else para executar o código se a condição for False
IMPORT	Usada para importar módulos.
IN	Retorna Verdadeiro se uma sequência com o valor especificado estiver presente no objeto. Exemplo: x in y
IS	Retorna Verdadeiro se ambas as variáveis forem o mesmo objeto. Exemplo: x is y
LAMBDA	Uma função lambda é uma pequena função anônima. Uma função lambda pode receber qualquer número de argumentos, mas pode ter apenas uma expressão.
NONE	O objeto Python None, denota falta de valor. Este objeto não tem métodos. É usado para definir um valor nulo ou nenhum valor.
NONLOCAL	Usada para trabalhar com variáveis dentro de funções aninhadas, onde a variável não deve pertencer à função interna. Use a palavra-chave nonlocal para declarar que a variável não é local.

NOT	É um operador lógico. O valor de retorno será True se a(s) instrução(ões) não forem True, caso contrário retornará False.
OR	É um operador lógico Os operadores lógicos são usados para combinar instruções condicionais. O valor de retorno será True se uma das instruções retornar True, caso contrário retornará False.
PASS	Usada como um espaço reservado para código futuro. Quando a instrução pass é executada, nada acontece, mas você evita obter um erro quando o código vazio não é permitido. Código vazio não é permitido em loops, definições de função, definições de classe ou em instruções if.
RAISE	Usada para gerar uma exceção. Você pode definir que tipo de erro gerar e o texto a ser impresso para o usuário.
RETURN	Sai de uma função e retornar um valor.
TRUE	Valor booleano e resultado de uma operação de comparação. A palavra-chave True é igual a 1 (False é igual a 0).
TRY	Usada em blocos try...except. Ele define um bloco de teste de código se ele contém algum erro. Você pode definir blocos diferentes para diferentes tipos de erro e blocos para executar se nada der errado.
WHILE	Com o loop while podemos executar um conjunto de instruções desde que uma condição seja verdadeira. O loop while requer que as variáveis relevantes estejam prontas (veja que é necessário declarar e atribuir o valor da variável j no exemplo abaixo). j = 0 while j < 3: print(j) j = j + 1
WITH	Usado para simplificar o tratamento de exceções
YIELD	Para finalizar uma função, retorna um gerador



REGRAS PARA NOMES DE VARIÁVEIS

- DEVE COMEÇAR COM UMA LETRA OU O CARACTERE SUBLINHADO
- NÃO PODE COMEÇAR COM UM NÚMERO
- PODE CONTER APENAS CARACTERES ALFANUMÉRICOS E SUBLINHADOS (AZ, 0-9 E _)
- DIFERENCIAM MAIÚSCULAS DE MINÚSCULAS

TIPO DE DADO	DESCRIÇÃO	EXEMPLO
TEXTO	str	x = "Hello World"
NUMÉRICOS	int, float, complex	Int: x = "Hello World" Float: x = 20.5 Complex: x = 1j
SEQUÊNCIA	list, tupla, range	List: x = ["maçã", "banana", "cereja"] Tupla: x = ("maçã", "banana", "cereja") Range: x = range(6)
MAPEAMENTO	dict	Dict: x = {"nomw=e" : "John", "idade" : 36}
CONJUNTO	set,frozenset	Set: x = {"maçã", "banana", "cereja"} Frozenset: x = frozenset({"maçã", "banana", "cereja"})
BOOLEANO	bool	x = True
BINÁRIO	bytes, bytearray, memoryview	Bytes: x = b"Hello" Bytearray: x = bytearray(5) Memoryview: x = memoryview(bytes(5))
NENHUM TIPO	NoneType	NoneType: x = None

MÉTODOS PARA MODIFICAR STRINGS	DESCRIÇÃO
UPPER()	Retorna a string em maiúsculas
LOWER()	Retorna a string em letras minúsculas
STRIP()	Remove qualquer espaço em branco do início ou do fim
REPLACE()	Substitui uma string por outra string. Ex: replace("h", "j")
SPLIT()	Retorna uma lista em que o texto entre o separador especificado se torna os itens da lista.

CARACTER DE ESCAPE	DESCRIÇÃO
\'	Aspas simples

\\	Barra invertida
\N	Nova linha
\R	Volta para margem esquerda e em uma nova linha
\T	Tab
\B	Backspace
\F	Caractere de controle ASCII de quebra de página. Ele força a impressora a ejetar a página atual e a continuar imprimindo na parte superior de outra.
\OOO	Valor octal
\XHH	Valor hexadecimal

FUNÇÃO	DESCRIÇÃO
CAPITALIZE()	Converte o primeiro caractere em maiúscula
CASEFOLD()	Converte string em minúsculas
CENTER()	Retorna uma string centralizada
COUNT()	Retorna o número de vezes que um valor especificado ocorre em uma string
ENCODE()	Retorna uma versão codificada da string
ENDSWITH()	Retorna verdadeiro se a string terminar com o valor especificado
EXPANDTABS()	Define o tamanho da guia da string
FIND()	Pesquisa a string por um valor especificado e retorna a posição de onde foi encontrado
FORMAT()	Formata valores especificados em uma string
FORMAT_MAP()	Formata valores especificados em uma string
INDEX()	Pesquisa a string por um valor especificado e retorna a posição de onde foi encontrado
ISALNUM()	Retorna True se todos os caracteres da string forem alfanuméricos
ISALPHA()	Retorna True se todos os caracteres da string estiverem no alfabeto
ISASCII()	Retorna True se todos os caracteres da string forem caracteres ASCII
ISDECIMAL()	Retorna True se todos os caracteres na string forem decimais
ISDIGIT()	Retorna True se todos os caracteres da string forem dígitos
ISIDENTIFIER()	Retorna True se a string for um identificador
ISLOWER()	Retorna True se todos os caracteres da string forem minúsculos
ISNUMERIC()	Retorna True se todos os caracteres da string forem numéricos
ISPRINTABLE()	Retorna True se todos os caracteres da string forem imprimíveis
ISSPACE()	Retorna True se todos os caracteres na string forem espaços em branco
ISTITLE()	Retorna True se a string seguir as regras de um título
ISUPPER()	Retorna True se todos os caracteres da string forem maiúsculos

JOIN()	Converte os elementos de um iterável em uma string
LJUST()	Retorna uma versão justificada à esquerda da string
LOWER()	Converte uma string em letras minúsculas
LSTRIP()	Retorna uma versão de corte à esquerda da string
MAKETRANS()	Retorna uma tabela de tradução para ser usada nas traduções
PARTITION()	Retorna uma tupla onde a string é dividida em três partes
REPLACE()	Retorna uma string onde um valor especificado é substituído por um valor especificado
RFIND()	Pesquisa a string por um valor especificado e retorna a última posição de onde foi encontrado
RINDEX()	Pesquisa a string por um valor especificado e retorna a última posição de onde foi encontrado
RJUST()	Retorna uma versão justificada à direita da string
RPARTITION()	Retorna uma tupla onde a string é dividida em três partes
RSPLIT()	Divide a string no separador especificado e retorna uma lista
RSTRIP()	Retorna uma versão de corte à direita da string
SPLIT()	Divide a string no separador especificado e retorna uma lista
SPLITLINES()	Divide a string nas quebras de linha e retorna uma lista
STARTSWITH()	Retorna verdadeiro se a string começar com o valor especificado
STRIP()	Remove espaços em branco (ou caracteres dentro do parêntese) do início/fim da string
SWAPCASE()	Troca de maiúsculas, minúsculas tornam-se maiúsculas e vice-versa
TITLE()	Converte o primeiro caractere de cada palavra para maiúscula
TRANSLATE()	Retorna uma string traduzida
UPPER()	Converte uma string em maiúscula
ZFILL()	Preenche a string com um número especificado de valores o no início

OPERADOR	NOME	EXEMPLO
+	Adição	x + y
-	Subtração	x - y
*	Multiplicação	x * y
/	Divisão	x / y
%	Módulo	x % y
**	Exponenciação	x ** y
//	Floor Division: arredonda o resultado para o número inteiro mais próximo	x // y

FUNÇÃO	DESCRIÇÃO	IGUAL A

=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

FUNÇÃO	DESCRIÇÃO	IGUAL A
==	Igual	x == y
!=	Diferente	x != y
>	Maior que	x > y
<	Menor que	x < y
>=	Maior ou igual a	x >= y
<=	Menor ou igual a	x <= y

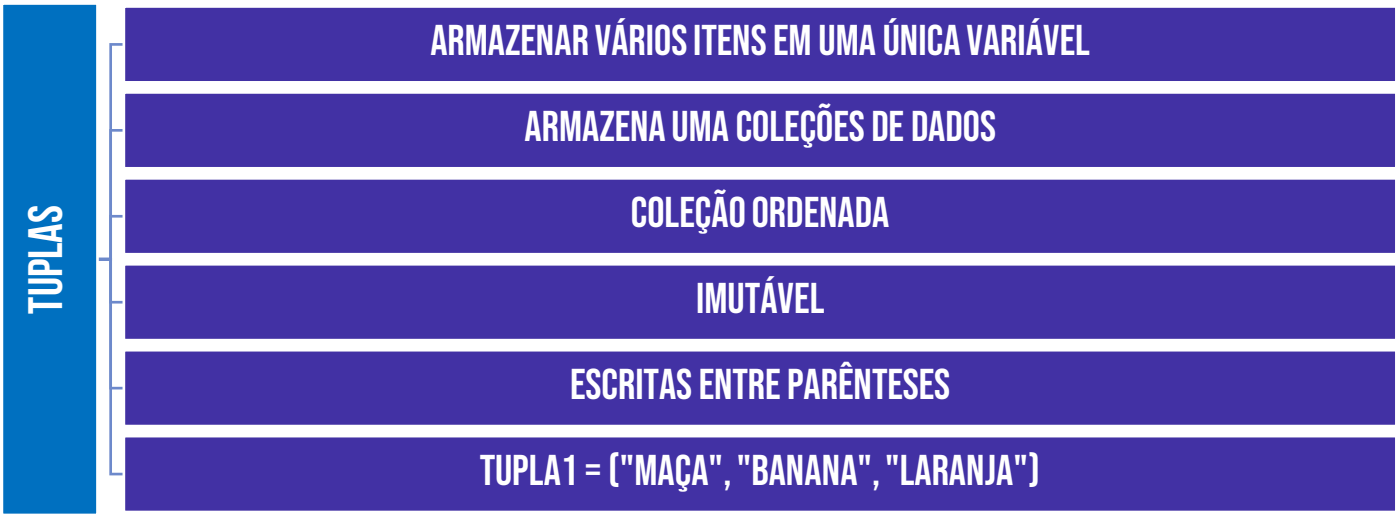
OPERADOR	DESCRIÇÃO	EXEMPLO
AND	Retorna True se ambas as declarações forem verdadeiras	x < 5 and x < 10
OR	Retorna True se uma das declarações for verdadeira	x < 5 or x < 4
NOT	Inverte o resultado, retorna False se o resultado for verdadeiro	not(x < 5 and x < 10)

OPERADOR	DESCRIÇÃO	EXEMPLO
IS	Retorna True se ambas as variáveis forem o mesmo objeto	x is y
IS NOT	Retorna True se ambas as variáveis não forem o mesmo objeto	x is not y
OPERADOR	DESCRIÇÃO	EXEMPLO
IN	Retorna True se uma sequência com o valor especificado estiver presente no objeto	x in y

NOT IN	Retorna True se uma sequência com o valor especificado não estiver presente no objeto	x not in y

OPERADOR	NOME	DESCRIÇÃO
&	AND	Define cada bit como 1 se ambos os bits forem 1
|	OR	Define cada bit como 1 se um dos dois bits for 1
^	XOR	Define cada bit como 1 se apenas um dos dois bits for 1
~	NOT	Inverte todos os bits
<<	Zero fill left shift	Desloque para a esquerda empurrando zeros da direita e deixe os bits mais à esquerda fora
>>	Signed right shift	Desloque para a direita empurrando cópias do bit mais à esquerda da esquerda e deixe os bits mais à direita fora

MÉTODO	DESCRIÇÃO
APPEND()	Adiciona um elemento no final da lista
CLEAR()	Remove todos os elementos da lista
COPY()	Retorna uma cópia da lista
COUNT()	Retorna o número de elementos com o valor especificado
EXTEND()	Adiciona os elementos de uma lista (ou qualquer iterável), ao final da lista atual
INDEX()	Retorna o índice do primeiro elemento com o valor especificado
INSERT()	Adiciona um elemento na posição especificada
POP()	Remove o elemento na posição especificada
REMOVE()	Remove o primeiro item com o valor especificado
REVERSE()	Inverte a ordem da lista
SORT()	Inverte a ordem da lista



NÃO PODEMOS ALTERAR, ADICIONAR OU REMOVER ITENS APÓS A CRIAÇÃO DA TUPLA.

MÉTODO	DESCRIÇÃO
COUNT()	Retorna o número de vezes que um valor especificado ocorre em uma tupla
INDEX()	Procura na tupla um valor especificado e retorna a posição de onde foi encontrado

MÉTODO	DESCRIÇÃO	EXEMPLO
INTERSECTION()	Gera novo conjunto, que contém apenas os itens presentes em ambos os conjuntos	z = x.intersection(y)
INTERSECTION_UPDATE()	Manterá apenas os itens presentes em ambos os conjuntos.	x.intersection_update(y)
SYMMETRIC_DIFFERENCE()	Novo conjunto, que contém apenas os elementos que NÃO estão presentes em ambos os conjuntos	z = x.symmetric_difference(y)
SYMMETRIC_DIFFERENCE_UPDATE()	Manterá apenas os elementos que NÃO estão presentes em ambos os conjuntos.	x.symmetric_difference_update(y)

MÉTODO	DESCRIÇÃO
ADD()	Adiciona um elemento ao conjunto
CLEAR()	Remove todos os elementos do conjunto
COPY()	Retorna uma cópia do conjunto
DIFFERENCE()	Retorna um conjunto contendo a diferença entre dois ou mais conjuntos
DIFFERENCE_UPDATE()	Remove os itens neste conjunto que também estão incluídos em outro conjunto especificado

DISCARD()	Remove o item especificado
INTERSECTION()	Retorna um conjunto, que é a interseção de dois outros conjuntos
INTERSECTION_UPDATE()	Remove os itens neste conjunto que não estão presentes em outros conjuntos especificados
ISDISJOINT()	Retorna se dois conjuntos têm uma interseção ou não
ISSUBSET()	Retorna se outro conjunto contém este conjunto ou não
ISSUPERSET()	Retorna se este conjunto contém outro conjunto ou não
POP()	Remove um elemento do conjunto
REMOVE()	Remove o elemento especificado
SYMMETRIC_DIFFERENCE()	Retorna um conjunto com as diferenças simétricas de dois conjuntos
SYMMETRIC_DIFFERENCE_UPDATE()	Insere as diferenças simétricas deste conjunto e de outro
UNION()	Retorna um conjunto contendo a união de conjuntos
UPDATE()	Atualize o conjunto com a união deste conjunto e outros

MÉTODO	DESCRIÇÃO
CLEAR()	Remove todos os elementos do dicionário
COPY()	Retorna uma cópia do dicionário
FROMKEYS()	Retorna um dicionário com as chaves e o valor especificados
GET()	Retorna o valor da chave especificada
ITEMS()	Retorna uma lista contendo uma tupla para cada par de valores-chave
KEYS()	Retorna uma lista contendo as chaves do dicionário
POP()	Remove o elemento com a chave especificada
POPITEM()	Remove o último par de valores-chave inserido
SETDEFAULT()	Retorna o valor da chave especificada. Se a chave não existir: insira a chave, com o valor especificado
UPDATE()	Atualiza o dicionário com os pares de valores-chave especificados
VALUES()	Retorna uma lista de todos os valores no dicionário

COLEÇÕES

- LISTA É UMA COLEÇÃO ORDENADA E MUTÁVEL. PERMITE MEMBROS DUPLICADOS.
- TUPLA É UMA COLEÇÃO ORDENADA E IMUTÁVEL. PERMITE MEMBROS DUPLICADOS.
- SET É UMA COLEÇÃO NÃO ORDENADA, IMUTÁVEL E NÃO INDEXADA. NENHUM MEMBRO DUPLICADO.
- DICIONÁRIO É UMA COLEÇÃO ORDENADA E MUTÁVEL. NENHUM MEMBRO DUPLICADO.

CRITÉRIO	LIST	TUPLE	SET	DICTIONARY
ORDENAÇÃO	Ordenada	Ordenada	Não ordenada	Não ordenada
MODIFICAÇÃO	Mutável	Imutável	Mutável	Mutável
DUPLICATAS	Permite duplicatas	Permite duplicatas	Não permite duplicatas	Não permite duplicatas
INDEXAÇÃO	Por inteiro	Por inteiro	Não indexada	Por string
DELIMITADOR	Entre colchetes [ ]	Entre parênteses ( )	Entre chaves { }	Entre chaves { }

CATEGORIA	TIPO	PYTHON	EXEMPLO
BOOLEANO	Booleano	bool	`x = True` `x = False`
NUMÉRICO	Inteiro	int	`x = 10` `x = -5`
	Ponto Flutuante	float	`x = 10.7` `x = -2.8`
	Complexo	complex	`x = 345j` `x = 2-9j`
TEXTUAL	Texto	str	`x = 'texto'` `x = "texto"`
COLEÇÃO/SEQUÊNCIA	Lista	list	`x = [4, 8]` `x = list()`
	Tupla	tuple	`x = (5, 10)` `x = tuple()`
	Set	set	`x = {2, 4}` `x = set( )`
	Dicionário	dictionary	`x = {'nome': 'Diego', idade: 31}`

OPERADOR	DESCRIÇÃO	EXEMPLO
AND	Retorna True se todas as condições forem verdadeiras, caso contrário retorna False	x > 1 and x < 5
OR	Retorna True se uma das condições for verdadeiras, caso contrário retorna False	x > 1 or x < 5

NOT	Inverte o resultado: se o resultado da expressão for True, o operador retorna false	not(x > 1 and x < 5)

MÉTODO	DESCRIÇÃO
APPEND()	Adiciona um elemento no final da lista
CLEAR()	Remove todos os elementos da lista
COPY()	Retorna uma cópia da lista
COUNT()	Retorna o número de elementos com o valor especificado
EXTEND()	Adicione os elementos de uma lista (ou qualquer iterável) ao final da lista atual
INDEX()	Retorna o índice do primeiro elemento com o valor especificado
INSERT()	Adiciona um elemento na posição especificada
POP()	Remova o elemento na posição especificada
REMOVE()	Remova o primeiro item com o valor especificado
REVERSE()	Inverte a ordem da lista
SORT()	Ordena a lista

DIRETIVA	DESCRIÇÃO	EXEMPLO
%A	Dia da semana, versão curta	Wed
%A	Dia da semana, versão completa	Wednesday
%W	Dia da semana como um número 0-6, 0 é domingo	3
%D	Dia do mês 01-31	31
%B	Nome do mês, versão curta	Dec
%B	Nome do mês, versão completa	December
%M	Mês como um número 01-12	12
%Y	Ano, versão curta, sem século	18
%Y	Ano, versão completa	2018
%H	Hora 00-23	17
%I	Hora 00-12	05
%P	MANHÃ TARDE	PM
%M	Minuto 00-59	41
%S	Segundo 00-59	08
%F	Microssegundo 000000-999999	548513
%Z	Deslocamento UTC	+0100
%Z	Fuso horário	CST

%J	Número do dia do ano 001-366	365
%U	Número da semana do ano, domingo como o primeiro dia da semana, 00-53	52
%W	Número da semana do ano, segunda-feira como o primeiro dia da semana, 00-53	52
%C	Versão local de data e hora	Mon Dec 31 17:41:00 2018
%C	Século	20
%X	Versão local da data	12/31/18
%X	Versão local do tempo	17:41:00
%%	Um personagem	%
%G	ISO 8601 ano	2018
%U	Dia da semana ISO 8601 (1-7)	1
%V	Número da semana ISO 8601 (01-53)	01

DIRETIVA	DESCRIÇÃO
MATH.ACOS()	Retorna o arco cosseno de um número
MATH.ACOSH()	Retorna o cosseno hiperbólico inverso de um número
MATH.ASIN()	Retorna o arco seno de um número
MATH.ASINH()	Retorna o seno hiperbólico inverso de um número
MATH.ATAN()	Retorna o arco tangente de um número em radianos
MATH.ATAN2()	Retorna o arco tangente de y/x em radianos
MATH.ATANH()	Retorna a tangente hiperbólica inversa de um número
MATH.CEIL()	Arredonda um número para o inteiro mais próximo
MATH.COMB()	Retorna o número de maneiras de escolher k itens de n itens sem repetição e ordem
MATH.COPYSIGN()	Retorna um float que consiste no valor do primeiro parâmetro e o sinal do segundo parâmetro
MATH.COS()	Retorna o cosseno de um número
MATH.COSH()	Retorna o cosseno hiperbólico de um número
MATH.DEGREES()	Converte um ângulo de radianos para graus
MATH.DIST()	Retorna a distância euclidiana entre dois pontos (p e q), onde p e q são as coordenadas desse ponto
MATH.ERF()	Retorna a função de erro de um número
MATH.ERFC()	Retorna a função de erro complementar de um número
MATH.EXP()	Retorna E elevado à potência de x
MATH.EXPM1()	Retorna Ex - 1

MATH.FABS()	Retorna o valor absoluto de um número
MATH.FACTORIAL()	Retorna o fatorial de um número
MATH.FLOOR()	Arredonda um número para baixo para o inteiro mais próximo
MATH.FMOD()	Retorna o restante de x/y
MATH.FREXP()	Retorna a mantissa e o expoente, de um número especificado
MATH.FSUM()	Retorna a soma de todos os itens em qualquer iterável (tuplas, arrays, listas, etc.)
MATH.GAMMA()	Retorna a função gama em x
MATH.GCD()	Retorna o máximo divisor comum de dois inteiros
MATH.HYPOT()	Retorna a norma euclidiana
MATH.ISCLOSE()	Verifica se dois valores estão próximos ou não
MATH.ISFINITE()	Verifica se um número é finito ou não
MATH.ISINF()	Verifica se um número é infinito ou não
MATH.ISNAN()	Verifica se um valor é NaN (não é um número) ou não
MATH.ISQRT()	Arredonda um número de raiz quadrada para baixo para o inteiro mais próximo
MATH.LDEXP()	Retorna o inverso de math.frexp() que é x * (2**i) dos números fornecidos x e i
MATH.LGAMMA()	Retorna o valor de log gama de x
MATH.LOG()	Retorna o logaritmo natural de um número ou o logaritmo do número para a base
MATH.LOG10()	Retorna o logaritmo de base 10 de x
MATH.LOG1P()	Retorna o logaritmo natural de 1+x
MATH.LOG2()	Retorna o logaritmo de base 2 de x
MATH.PERM()	Retorna o número de maneiras de escolher k itens de n itens com ordem e sem repetição
MATH.POW()	Retorna o valor de x elevado a y
MATH.PROD()	Retorna o produto de todos os elementos em um iterável
MATH.RADIANS()	Converte um valor de grau em radianos
MATH.REMAINDER()	Retorna o valor mais próximo que pode tornar o numerador completamente divisível pelo denominador
MATH.SIN()	Retorna o seno de um número
MATH.SINH()	Retorna o seno hiperbólico de um número
MATH.SQRT()	Retorna a raiz quadrada de um número
MATH.TAN()	Retorna a tangente de um número
MATH.TANH()	Retorna a tangente hiperbólica de um número
MATH.TRUNC()	Retorna as partes inteiras truncadas de um número

FUNÇÃO	DESCRIÇÃO

FINDALL	Retorna uma lista contendo todas as correspondências
SEARCH	Retorna um objeto Match se houver uma correspondência em qualquer lugar da string
SPLIT	Retorna uma lista onde a string foi dividida em cada partida
SUB	Substitui uma ou várias correspondências por uma string

CARACTER	DESCRIÇÃO	EXEMPLO
[]	Um conjunto de caracteres	"[a-m]"
\	Sinaliza uma sequência especial (também pode ser usada para escapar de caracteres especiais)	"\d"
.	Qualquer caractere (exceto caractere de nova linha)	"he..o"
^	Começa com	"^hello"
$	Termina com	"planet$"
*	Zero ou mais ocorrências	"he.*o"
+	Uma ou mais ocorrências	"he.+o"
?	Zero ou uma ocorrência	"he.?o"
{}	Exatamente o número especificado de ocorrências	"he.{2}o"
|	Ou	"falls|stays"
()	Capturar e agrupar	

CARACTER	DESCRIÇÃO	EXEMPLO
\A	Retorna uma correspondência se os caracteres especificados estiverem no início da string	"\AThe"
\B	Retorna uma correspondência em que os caracteres especificados estão no início ou no final de uma palavra (o "r" no início está certificando-se de que a string está sendo tratada como uma "string bruta")	r"\bain" r"ain\b"
\B	Retorna uma correspondência onde os caracteres especificados estão presentes, mas NÃO no início (ou no final) de uma palavra (o "r" no início está certificando-se de que a string está sendo tratada como uma "string bruta")	r"\Bain" r"ain\B"
\D	Retorna uma correspondência em que a string contém dígitos (números de 0 a 9)	"\d"
\D	Retorna uma correspondência onde a string NÃO contém dígitos	"\D"
\S	Retorna uma correspondência onde a string contém um caractere de espaço em branco	"\s"
\S	Retorna uma correspondência em que a string NÃO contém um caractere de espaço em branco	"\S"

\W	Retorna uma correspondência em que a string contém qualquer caractere de palavra (caracteres de a a Z, dígitos de 0 a 9 e o caractere sublinhado _)	"\w"
\W	Retorna uma correspondência em que a string NÃO contém nenhum caractere de palavra	"\W"
\Z	Retorna uma correspondência se os caracteres especificados estiverem no final da string	"Spain\Z"

CONJUNTO	DESCRIÇÃO
[ARN]	Retorna uma correspondência em que um dos caracteres especificados (a, r ou n) está presente
[A-N]	Retorna uma correspondência para qualquer caractere minúsculo, em ordem alfabética entre a e n
[^ARN]	Retorna uma correspondência para qualquer caractere, EXCETO a, r e n
[0123]	Retorna uma correspondência onde qualquer um dos dígitos especificados (0, 1, 2 ou 3) está presente
[0-9]	Retorna uma correspondência para qualquer dígito entre 0 e 9
[0-5][0-9]	Retorna uma correspondência para qualquer número de dois dígitos de 00 a 59
[A-ZA-Z]	Retorna uma correspondência para qualquer caractere em ordem alfabética entre a e z, minúsculas OU maiúsculas
[+]	Em conjuntos, +, *, ., |, (), $,{} não tem significado especial, então [+] significa: retornar uma correspondência para qualquer caractere + na string

Automação de Scripts

Automatizar tarefas básicas do dia a dia é fundamental para aumentar a produtividade e valorizar o tempo. Podemos automatizar uma **série de tarefas com a Linguagem Python**. Desde a automação de tarefas na interface gráfica do Windows, MacOS ou Linux, passando pela automação da extração de dados via web scraping, extração de dados de arquivos pdf, testes de software, até automação de planilhas do Excel, há pacotes Python que ajudam nas tarefas de automação para facilitar o seu trabalho no dia a dia. Vejamos algumas ferramentas para Automação.

PyAutoGUI

PyAutoGUI permite que seus scripts Python controlem o mouse e o teclado para automatizar as interações com outros aplicativos. A API foi projetada para ser simples. O PyAutoGUI funciona no Windows, macOS e Linux e é executado no Python 2 e 3.

PyAutoGUI tem vários recursos:

- Movendo o mouse e clicando nas janelas de outros aplicativos.
- Envio de pressionamentos de tecla para aplicativos (por exemplo, para preencher formulários).
- Faça capturas de tela e forneça uma imagem (por exemplo, de um botão ou caixa de seleção) e encontre-a na tela.
- Localize a janela de um aplicativo e mova, redimensione, maximize, minimize ou feche (somente Windows, atualmente).
- Exibir caixas de alerta e mensagem.

pywinauto

pywinauto é um conjunto de módulos Python para automatizar a GUI do Microsoft Windows. Na sua forma mais simples, permite enviar ações de mouse e teclado para diálogos e controles do Windows.

Selenium Python

O S**elenium** é uma poderosa aplicação para automação web e permite uma série de explorações nesse universo.Nesse sentido, se faz importante compreender os componentes do Selenium, suas limitações e vantagens, bem como os principais métodos que serão usados.

Saiba mais com o conteúdo a seguir. Falaremos sobre o **Selenium Python** e mostraremos um tutorial simples de como conseguir a sua primeira automação na web com essa biblioteca.

O **Selenium Python** é uma **biblioteca com diversos métodos que ajudam na automação web**. Em suma, as funções permitem controlar o funcionamento de uma página e a interação com ela de forma automática. O **Selenium webdriver** é um dos elementos do conjunto selenium que agrega todos os métodos importantes que ainda vamos ver mais adiante.

É interessante também definir o que vem a ser um webdriver. Um **webdriver** é justamente **um app especial que controla o navegador**. Existem drivers específicos para cada navegador hoje, com as características necessárias para lidar com cada um.

Dentro do universo do Selenium Python, temos alguns subconceitos relevantes. São eles: **Selenium IDE, Selenium RC (Remote control), Selenium WebDriver e Selenium GRID.**

No Selenium Python temos vários métodos interessantes que podem ser adotados a depender da necessidade. Para clicar em um elemento, o programador pode adotar o click. Um jeito fácil de achar qualquer clicável na página é find_elements_by_xpath(). O método delete_all_cookies apaga os cookies. Há alguns mais simples também, como o maximize_window e o minimize_window, que maximizam e minimizam janelas, respectivamente. Vejamos a seguir uma lista com vários desses métodos e suas finalidades.

MÉTODO	FINALIDADE
BACK	Volta no histórico para a página anterior
CLOSE	Fecha a página atual
EXECUTE_SCRIPT	Executa javascript na janela
FORWARD	Vai adiante no histórico, para a próxima página
GET_WINDOW_SIZE	Pega o tamanho da janela atual
REFRESH	Atualiza a página
GET_SCREENSHOT_AS_FILE	Salvar um print da tela

Splinter

Splinter é uma estrutura Python que fornece uma interface simples e consistente para automação de aplicativos da web.

Características principais:

- Fácil de aprender: a API foi projetada para ser intuitiva e rápida de aprender.
- Mais rápido para codificar: automatize as interações do navegador de forma rápida e confiável sem lutar contra a ferramenta.

- Poderoso: Projetado para casos de uso do mundo real, protege contra peculiaridades de automação comuns.
- Flexível: o acesso a ferramentas de nível inferior nunca é oculto. Quebre o Selênio bruto a qualquer momento.
- Robusto: O suporte está disponível para vários drivers de automação (Selenium, Django, Flask, ZopeTestBrowser).

Splinter é usado para escrever scripts de automação do navegador da web.O projeto tem dois objetivos principais:

- Fornecer uma API comum de alto nível sobre as ferramentas de automação de navegador existentes, como o **Selenium**. A API é uma camada de abstração amigável e projetada para scripts fáceis e eficientes.
- Fornecer funcionalidade integrada para lidar com pontos problemáticos comuns com a automação do navegador.

Scrapy

Uma estrutura de código aberto e colaborativa para extrair os dados que você precisa de sites. De forma **rápida, simples e extensível**. Ele é rápido e poderoso permitindo ao desenvolvedor escrever as regras para extrair os dados e deixar o Scrapy fazer o resto. Ademais, é facilmente extensívelpor design, é possível conectar novas funcionalidades facilmente sem ter que tocar no núcleo. Por ser escrito em Python, ele roda em Linux, Windows, Mac e BSD

Windmill

Windmill é uma estrutura de teste de interface do usuário da Web AJAX de código aberto. O Windmill implementa testes entre navegadores, gravação e reprodução no navegador e funcionalidade para depuração rápida e precisa; e integração do ambiente de teste.

pytest

O framework pytest facilita a escrita de testes pequenos e legíveis e pode ser dimensionada para dar **suporte a testes funcionais complexos** para aplicativos e bibliotecas.

De acordo com o livro **Teste do Python com pytest**, Brian Okken, **pytest** é inegavelmente a melhor escolha para testar projetos Python. É uma estrutura de teste completa, flexível e extensível. O modelo de fixture do **pytest** permite que você compartilhe dados de teste e procedimentos de configuração em várias camadas de testes. A estrutura **pytest** oferece recursos poderosos, como

regravação de assert, parametrização, marcadores, plug-ins, execução de teste paralelo e relatórios claros de falha de teste - sem código clichê.

Escreva testes curtos e de fácil manutenção que expressem com elegância o que você está testando. Acelere os tempos de teste distribuindo testes em vários processadores e executando testes em paralelo. Use as instruções assert integradas do Python em vez de funções auxiliares de assert complicadas para tornar seus testes mais legíveis. Mova o código de configuração para fora dos testes e para os acessórios para separar as falhas de configuração das falhas de teste. Teste condições de erro e casos de canto com testes de exceção esperados e use um teste para executar muitos casos de teste com testes parametrizados. Estenda o **pytest** com plugins, conecte-o a sistemas de integração contínua e use-o em conjunto com testes tox, mock, cobertura e até mesmo unittest existentes.

ReportLab

ReportLab é um framework usado para a criação de documentos PDF baseados em dados complexos e gráficos vetoriais personalizados. É gratuito, de código aberto e escrito em Python.

Report Markup Language é uma linguagem de estilo XML para descrever o layout de documentos. Você define e manipula qualquer aspecto de um documento, incluindo o conteúdo e o estilo, usando tags. Muitas das tags são semelhantes ao HTML.

A saída do documento PDF é gerada a partir do RML usando o módulo python '**rml2pdf**'. O RML é usado em conjunto com Preppyo sistema de modelagem do **ReportLab**. Foi desenvolvido no final de 2000 e está em uso contínuo de produção desde então. É de código aberto (licença BSD).

PDFMiner

PDFMiner é uma ferramenta de extração de texto para documentos PDF.

- Suporta PDF-1.7. (bem, quase)
- Obtém a localização exata do texto, bem como outras informações de layout (fontes, etc.).
- Executa a análise automática de layout.
- Pode converter PDF em outros formatos (HTML/XML).
- Pode extrair um esboço (TOC).
- Pode extrair conteúdo marcado.
- Suporta criptografia básica (RC4 e AES).
- Suporta vários tipos de fonte (Type1, TrueType, Type3 e CID).
- Suporta linguagens CJK e scripts de escrita vertical.

- Possui um analisador de PDF extensível que pode ser usado para outros fins.

Borb

Trabalhar com PDF pode ser complexo e desafiador na melhor das hipóteses. É por isso que o **borb** foi criado com a facilidade de uso em mente. Os usuários não precisam ter amplo conhecimento de PDF. Economize tempo, dinheiro e trabalhe mais feliz com **borb**.

OpenPyXL

openpyxl é uma biblioteca Python para ler/gravar arquivos do Excel 2010 xlsx/xlsm/xltx/xltm. Nasceu da falta de biblioteca existente para ler/escrever nativamente do Python o formato Office Open XML.

PyXLL

Use o Microsoft Excel como um front-end amigável para o seu código Python. Sem VBA, apenas Python! Essa é a proposta do PyXLL. Embora seja um software pago, há uma versão trial que pode ser usada sem custo.

XlsxWriter

XlsxWriter é um módulo Python que pode ser usado para escrever texto, números, fórmulas e hiperlinks para várias planilhas em um arquivo XLSX do Excel 2007+. Ele oferece suporte a recursos como formatação e diversas outras funcionalidades.

Tagui

Tagui é uma API simples e poderosa de RPA para Python que torna a automação robótica de processos divertida! Você pode usá-lo para automatizar rapidamente tarefas repetitivas que consomem tempo em sites, aplicativos de desktop ou linha de comando. Este pacote foi inicialmente desenvolvido na Universidade de Singapura.

Robot Framework

O Robot Framework é um framework de automação genérica e de código aberto. Ele pode ser usado para automação de teste e automação robótica de processos (RPA). É estudado em um dos cursos da Formação Desenvolvedor RPA.

REFERÊNCIAS

https://docs.python.org/pt-br/3/

https://www.w3schools.com/python/

https://www.treinaweb.com.br/blog/principais-estruturas-de-dados-no-python

https://pythonhelp.wordpress.com/2013/06/18/conjuntos-em-python/

https://numpy.org/doc/stable/reference/generated/numpy.std.html

https://www.hashtagtreinamentos.com/automacao-web-no-python

https://splinter.readthedocs.io/en/latest/why.html

https://scrapy.org/

https://docs.pytest.org/en/7.1.x/contents.html

https://pragprog.com/titles/bopytest2/python-testing-with-pytest-second-edition/

https://docs.reportlab.com/rmlfornewbies/

https://borbpdf.com/

https://blog.dsacademy.com.br/15-pacotes-python-para_automacao/

QUESTÕES COMENTADAS

Questões Cespe

1. **(CESPE – DPE RO – 2022)** Na linguagem Python, são consideradas sequências mutáveis as

a) strings
b) cadeias
c) tuplas
d) listas
e) ranges

Comentários:

A lista é mutável, o que significa que podemos alterar, adicionar e remover itens em uma lista após ela ter sido criada. Portanto, as listas são consideradas sequências mutáveis.

Gabarito: Letra D

---

2. **(CEBRASPE – PC PB– 2022)** Na linguagem Python, o tipo de uma variável em tempo de execução é definido pelo interpretador pelo recurso denominado

a) tipagem dinâmica.
b) modo interativo.
c) sintaxe.
d) interpretação bytecode.
e) empacotamento.

Comentários:

Tipagem dinâmica é uma característica de determinadas linguagens de programação, que não exigem declarações de tipos de dados, pois são capazes de escolher que tipo utilizar dinamicamente para cada variável, podendo alterá-lo durante a compilação ou a execução do programa.

Gabarito: Letra A

---

3. **(CESPE – PC PB– 2022)** Python é uma linguagem procedural que utiliza quatro tipos de dados predefinidos para lidar com coleções: conjuntos, dicionários, listas e tuplas. A respeito desses tipos de dados, julgue os itens a seguir.

I O conjunto permite o armazenamento de uma tupla, mas não o de uma lista.
II A tupla é idêntica à lista, exceto pela forma mais simples com que sua declaração é realizada.

III A lista é um tipo de dados variável que permite a alteração de seus elementos após a sua criação.
Assinale a opção correta.

a) Apenas o item I está certo.
b) Todos os itens estão certos.
c) Apenas o item II está certo
d) Apenas os itens I e III estão certos.
e) Apenas os itens II e III estão certos.

Comentários:

Pessoal, vamos começar pelo erro do item II: Uma tupla é uma estrutura bastante similar a uma lista, com uma diferença especial: os elementos inseridos em uma tupla não podem ser alterados, o que não ocorre em uma lista, já que nesta podem ser alterados livremente. Ademais, vejamos o item I: conjunto permite o armazenamento de uma tupla, mas não o de uma lista. Está correto o item I, porque os elementos de um conjunto não são armazenados em uma ordem específica além disso, conjuntos não contém elementos repetidos. Portanto, conjuntos são diferentes de listas. Por fim, o item III está correto, é possível a alteração dos elementos de uma lista.

Gabarito: Letra D

---

4. **(CEBRASPE – (CODEVASF– 2021)** Na linguagem Python, as listas são coleções de qualquer tipo de objetos, com exceção das próprias listas, e seus elementos são alteráveis.

Comentários:

Pessoal, lista é uma coleção ordenada e mutável. Permite membros duplicados. Portanto, o correto seria dizer: Na linguagem Python, as listas são coleções de qualquer tipo de objetos, incluindo as próprias listas, e seus elementos são alteráveis

Gabarito: Errada

---

5. **(CEBRASPE – PF– 2021)** O código Python a seguir apresenta como resultado "True".

```
x = bool(-3)
y = bool("True"*x)
z = bool("False")
print (x and y and z)
```

Comentários:

POLÊMICA! Pessoal, se você rodar esse programa ele vai apresentar como resultado True! Porém a banca teve a coragem de dizer na justificativa de alteração dos gabaritos que "O resultado que deveria ser True, e não "True"". Enfim, vamos para o código: x,y e z são possuem o valor "True".

Quando utilizamos a clase bool para conversões temos uma "regra". Essa regra nos diz que quando um valor a ser convertido não for uma constante definida para ser falsa, None ou False; não for zero de nenhum tipo; ou termos uma sequência de coleções vazias vamos ter como retorno um valor True. Por isso, x, y e z possuem esse valor. E o operador and de três variáveis True gera como resultado True.

Gabarito: Errada

---

6. **(CEBRASPE – SERPRO– 2021)** As tuplas, embora sejam semelhantes às listas, estão limitadas a, no máximo, cinco níveis.

Comentários:

Primeiro lembrete: TUPLAS SÃO IMUTÁVEIS, listas são mutáveis. E diferentemente do que diz a questão não há a limitação de no máximo, cinco níveis nas tuplas.

Gabarito: Errada

---

7. **(CEBRASPE – SERPRO– 2021)** Listas são coleções alteráveis de qualquer tipo de objeto — como, por exemplo, outras listas — capazes de gerar um efeito top-down sem limite de níveis.

Comentários:

Perfeito! Perfeito! As listas são MUTÁVEIS, ou ALTERÁVEIS!

Gabarito: Correta

---

8. **(CEBRASPE – PC DF– 2021)** Com relação a mineração de dados, aprendizado de máquina e aplicações Python, julgue o item a seguir.

   Uma das aplicações de Python é o aprendizado de máquina, que pode ser exemplificado por um programa de computador que aprende com a experiência de detectar imagens de armas e de explosivos em vídeos, tendo seu desempenho medido e melhorado por meio dos erros e de acertos decorrentes da experiência de detecção.

Comentários:

Vejamos o que diz a própria banca: O exemplo apresentado enquadra-se na definição atual de aprendizado de máquina. "O que é aprendizado de máquina? Duas definições de aprendizado de máquina são oferecidas. Arthur Samuel o descreveu como: "o campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados". Essa é uma definição mais antiga e informal. Na definição mais moderna, "um programa de computador aprende com a experiência E com relação a alguma classe de tarefas T e medida de desempenho P, se seu desempenho nas tarefas em T, conforme medido por P, melhora com a experiência E. Por exemplo, jogar damas. E = a experiência de jogar muitos jogos de damas T = a tarefa de jogar damas. P = a probabilidade de o programa vencer o próximo jogo.".

Gabarito: Correta

---

9. **(CEBRASPE – BANESE– 2021)** No que se refere ao pacote NumPy do Python, julgue o item subsequente. O código a seguir retorna o valor do desvio padrão amostral do conjunto de dados {1,2,2,3,5}

```
import numpy
x = numpy.array([1,2,2,3,5])
numpy.std(x,ddof=1)
```

Comentários:

Pessoal, o código apresentado pela questão importa o pacote numpy, cria uma variável x que recebe o array [1, 2, 2, 3, 5]. Por fim, o método std de numpy retorna o desvio padrão amostral do conjunto dado (numpy.std(x,ddof=1)).

Gabarito: Correta

---

10. **(CEBRASPE – SEED PR– 2021)** Em Python, quando mais de um operador aparece em uma expressão, a ordem de avaliação depende das regras de precedência de cada linguagem. Assim, ao programar em Python, além de observar essas regras, é preciso considerar, ainda, a forma como a linguagem representa seus operadores, conforme demonstrado nos comandos a seguir.

```
x=7*3**2%4
print (x)
```

Assinale a opção que corresponde à saída que o compilador Python apresentará para os comandos em questão.

a) 1
b) 3
c) 7
d) 15
e) 15.75

Comentários:

Pessoal, vamos "traduzir" essa expressão. ** Exponenciação; % Módulo. Temos aqui um (7*(3**2))%4) Primeiro: x = 7*9%4 resolvemos o 3 elevado ao quadrado. Depois, x= 63%4 multiplicamos nove por 7. Por fim, só resta o módulo da divisão. x= 3, já que 4*15 é 60 e deixa resto 3.

Gabarito: Letra B

---

11. (CEBRASPE – SEED PR– 2021) Na linguagem de programação Python, existem 3 estruturas para armazenar dados indexados. A estrutura cujos valores são imutáveis depois de sua criação é conhecida como

a) lista
b) operador
c) tupla
d) classe
e) dicionário

Comentários:

Tupla é uma Lista imutável. O que diferencia a Estrutura de Dados Lista da Estrutura de Dados Tupla é que a primeira pode ter elementos adicionados a qualquer momento. Vamos relembrar nossa tabela!

CRITÉRIO	LIST	TUPLE	SET	DICTIONARY
ORDENAÇÃO	Ordenada	Ordenada	Não ordenada	Não ordenada
MODIFICAÇÃO	Mutável	Imutável	Mutável	Mutável
DUPLICATAS	Permite duplicatas	Permite duplicatas	Não permite duplicatas	Não permite duplicatas
INDEXAÇÃO	Por inteiro	Por inteiro	Não indexada	Por string
DELIMITADOR	Entre colchetes [ ]	Entre parênteses ( )	Entre chaves { }	Entre chaves { }

Gabarito: Letra C

---

12. (CEBRASPE – PF– 2018) Considere os seguintes comandos na programação em Python.

```
a = " Hello, World! "
print(a.strip())
```

Esses comandos, quando executados, apresentarão o resultado a seguir.

a[0]=Hello,
a[1]=World!

Comentários:

Pessoal, errada questão! Na verdade, vai apresentar Hello, World!. Strip Remove espaços em branco (ou caracteres dentro do parêntese) do início/fim da string.

Gabarito: Errada

---

13. (CEBRASPE – PF– 2018) Considere o programa a seguir, na linguagem Python.

```
letras == ["P", "F"]
for x in letras
 {
 print(x)
 }
```

A sintaxe do programa está correta e, quando executado, ele apresentará o seguinte resultado.

PF

Comentários:

Pessoal, na verdade há três erros na questão. Vamos imediatamente corrigi-los. Na primeira linha o comando deveria ser letras = ["P", "F"] (com apenas um operador =). Ademais, deveria ser usado o dois pontos no for, e não deve-se usar colchetes para indentação da função for. Portanto, o correto seria:

```
letras = ["P", "F"]
for x in letras:
  print(x)
```

Gabarito: Errada

---

14. (CESPE – 2013 – MPOG – Analista de Sistemas) Em Python, o comando int("1") cria um objeto do tipo int, que recebe 1 como parâmetro no seu construtor.

Comentários:

Essa é uma função de casting! Ela converte variáveis do tipo inteiro, ponto flutuante ou string em um inteiro. No caso, ele está recebendo um número como string (por conta das aspas) e convertendo em um inteiro. Esse número serve de parâmetro para o seu construtor (que é um método que cria um objeto, mas vocês não precisam saber disso).

Gabarito: Correta

---

15. (CESPE – 2013 – MPOG – Analista de Sistemas) Em Python, o comando int("1") cria um objeto do tipo int, que recebe 1 como parâmetro no seu construtor.

Comentários:

Essa é uma função de casting! *O que ela faz?* Bem, ela converte variáveis do tipo inteiro, ponto flutuante ou string em um inteiro. No caso, ele está recebendo um número como string (por conta das aspas) e convertendo em um inteiro. Esse número serve de parâmetro para o seu construtor (que é um método que cria um objeto, mas vocês não precisam saber disso).

Gabarito: Correta

---

16. (CESPE – 2011 – ECT – Analista de Sistemas) A linguagem Python e seu interpretador estão disponíveis para as mais diversas plataformas. Para que seja usado em determinado sistema operacional não suportado, é possível gerar o Python a partir do programa fonte utilizando um compilador C. Nesse caso, o código fonte é traduzido para o formato bytecode, que é multiplataforma e pode ser distribuído de forma independente.

Comentários:

Ele realmente é multiplataforma. Além disso, permite que programas sejam compilados para um formato portável chamado de bytecode. Essa característica faz com que programas escritos nessa linguagem com uma biblioteca padrão sejam executadas da mesma forma em diversos sistemas operacionais que possuam um software interpretador de Python.

Gabarito: Correta

---

17. (CESPE – 2008 – SERPRO – Analista de Sistemas) Python é uma linguagem livre de alto nível, orientada a objetos e de difícil leitura, pois não permite identação de linhas de código.

Comentários:

Ela realmente é de alto nível, orientada a objetos e de fácil leitura. Além disso, permite indentação de linhas de código.

Gabarito: Errado

---

18. (CESPE – 2011 – ECT – Analista de Sistemas) A linguagem Python e seu interpretador estão disponíveis para as mais diversas plataformas. Para que seja usado em determinado sistema operacional não suportado, é possível gerar o Python a partir do programa fonte utilizando um compilador C. Nesse caso, o código fonte é traduzido para o formato bytecode, que é multiplataforma e pode ser distribuído de forma independente.

Comentários:

Ele realmente é multiplataforma. Além disso, permite que programas sejam compilados para um formato portável chamado de bytecode. Essa característica faz com que programas escritos nessa linguagem com uma biblioteca padrão sejam executadas da mesma forma em diversos sistemas operacionais que possuam um software interpretador de Python.

Gabarito: Correta

---

19. **(CESPE – 2008 – SERPRO – Analista de Sistemas)** Python é uma linguagem livre de alto nível, orientada a objetos e de difícil leitura, pois não permite identação de linhas de código.

Comentários:

Ela realmente é de alto nível, orientada a objetos e de fácil leitura. Além disso, permite indentação de linhas de código.

Gabarito: Errado

---

20. **(CESGRANRIO – 2004 – SECAD/TO – Analista de Sistemas)** Um programador de Python recebeu a tarefa de criar uma função chamada calcular que recebe dois parâmetros. Para executar sua atividade, ele deve utilizar a expressão:

a) def calcular (a,b):
b) function calcular (a,b):
c) import calcular (a,b):
d) procedure calcular (a,b):
e) sub calcular (a,b):

Comentários:

Uma função é definida por meio da palavra-chave def.

Gabarito: Letra A

Questões FGV

21. (FGV – TCU – 2022) Considere o código Python a seguir.

```
def xpto(S):
    for k in range(0,len(S)):
        if k%2 == 0:
            yield(S[k]);

S=[1,2,3,4,5,6]
for x in xpto(S[::-1]):
    print (x)
```

A execução desse código na IDLE Shell produz, na ordem e exclusivamente, os números:

a) 6, 1
b) 5, 3, 1
c) 6, 4, 2
d) 1, 3, 5
e) 2, 4, 6

Comentários:

Pessoal, precisamos entender o que a função faz. xpto consiste em uma função com um for que percorre S. Caso o valor k da lista S seja par (2, 4, 6), então ele cria uma lista (S) com os valores pares (2, 4, 6). Veja que o retorno da função é exatamente a lista gerada pelo yield(S[k]). Agora vamos para o código principal. Após a criação da lista S=[1,2,3,4,5,6], temos for x in xpto(S[::-1]): nesse ponto devemos ficar atentos ao -1. O -1 inverte a lista. Enviando para xpto a lista [6,5,4,3,2,1]. Assim, quando passar pela função xpto, teremos como resultado apenas os valores pares de [6,5,4,3,2,1], ou seja, 6,4,2. Portanto o gabarito é a letra c.

Gabarito: Letra C

---

22. (FGV – PC AM– 2022) Considere o código Python a seguir.

```
L=[0,1,1,2,3,5,8,13,21]
for k in range(0,len(L),2):
print (L[k])
```

Assinale o resultado exibido pela execução desse código, na IDLE Shell 3.9.9.
a) 1, 2, 5, 13

b) 0, 1, 3, 8, 21, 1, 2, 5, 13, 21
c) 1, 3, 8, 21, 1, 2, 5, 13, 21
d) 0, 1, 3, 8, 21
e) 0, 1, 3, 8

Comentários:

A questão consiste em um for que percorre a lista L e avança de 2 em 2 posições (for k in range(0,len(L),2):). Ele percorre sempre de 2 em 2 posições, portanto, a variável k procurará os valores das posições pares: Posições 0,2,4,6,8. Tenham em mente que o valor da posição 0 é 0, da posição 1 é 1, da posição 2 é 1, da posição 3 é 2 e assim por diante, veja: L=[0,1,1,2,3,5,8,13,21]. Se for necessário, desenhe as listas (uma em cima da outra) POSIÇÕES = [0,1,2,3,4,5,6,7,8,9] LISTA = [0,1,1,2,3,5,8,13,21].

Gabarito: Letra D

---

23. (FGV – SEFAZ AM– 2022) Analise o código a seguir em linguagem de programação Python:

```
def rotina(array):
    for p in range(0, len(array)):
        element = array[p]

        while p > 0 and array[p - 1] > element:
            array[p] = array[p - 1]
            p -= 1

        array[p] = element

    return array

print ( rotina([9, 5, 31, 42, 20, 56] ) )
```

Ao executar esse script em um terminal, será escrito na saída padrão
a) [9, 5, 31, 42, 20, 56]
b) [8, 4, 30, 41, 19, 55]
c) [56, 20, 42, 31, 5, 9]
d) [56, 42, 31, 20, 9, 5]
e) [5, 9, 20, 31, 42, 56]

Comentários:

Pessoal, o código da questão basicamente ordena um array. Dessa forma, a saída será o array ordenado: [5, 9, 20, 31, 42, 56].

Gabarito: Letra E

---

24. (FGV – MPE GO– 2022) Considere o código Python a seguir.

```
def X(n):
    if (type(N) != int):
        return -1
    elif (N < 1):
        return 0
    elif (N == 1):
        return 1
    else:
        return N * X(N-1)
print (X(4))
print (X(0))
print (X(1))
print (X(1.5))
print (X("A"))
```

Assinale o que acontece quando esse script é executada na IDLE Shell 3.9.9.
a) Erro de compilação, "name 'n ' is not defined"
b) Erro de compilação, "name 'N' is not defined"
c) Executa e produz resultados Corretas com quatro linhas.
d) Executa, mas produz erro de execução na quinta chamada da função X.
e) Executa, mas calcula erradamente o fatorial de 4.

Comentários:

Pessoal, vocês lembram que python é uma linguagem case sensitive? O erro neste caso será NameError: name 'N' is not defined. Porque na definição temos def X(n), depois usa-se o N (maiúsculo).

Gabarito: Letra B

---

25. (FGV – MPE GO– 2022) Assinale a lista de números produzida pela execução, na IDLE Shell 3.9.9, do código Python a seguir.

```
for x in range(-1, -10, -1):
 print (x)
```

a) -1 -2 -3 -4 -5 -6 -7 -8 -9
b) -9 -8 -7 -6 -5 -4 -3 -2 -1
c) 0 -1 -2 -3 -4 -5 -6 -7 -8 -9
d) 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
e) -1 -2 -3 -4 -5 -6 -7 -8 -9 -10

Comentários:

Pessoal, as bancas gostam muito de índices negativos, para dificultar um pouco. No caso da questão, é gerada uma sequência de números negativos iniciando em -1 e parando antes do -10 (ou

seja, o for para no -9). O For é incrementado de -1 em -1 formando a sequência -1 -2 -3 -4 -5 -6 -7 -8 -9.

Gabarito: Letra A

---

26. (FGV – TJDFT– 2022) Analise o código Python 3.9 a seguir.

```
class Teste:
def ----------
self.altura = xaltura
self.largura = xlargura
def dimensoes(self):
 print("altura = " + str(self.altura) + "\n" \ + "largura = " + str(self.largura))
x = Teste(12, 20)
x.dimensoes()
```

Para que a execução desse código exiba
altura = 12
largura = 20
o trecho tracejado na segunda linha deve ser substituído por:

a) __init__(self, xaltura, xlargura):
b) __init__(xaltura, xlargura):
c) init (xaltura, xlargura):
d) new (self, args[xaltura, xlargura]):
e) new (self, xaltura, xlargura):

Comentários:

Pessoal, utiliza-se a função __init__() para atribuir valores às propriedades do objeto. Sendo assim ficamos apenas com as opções a e b. Além de init, é necessário usar o self, que é uma referência à instância atual da classe. Portanto, o gabarito é a letra A.

Gabarito: Letra A

---

27. (FGV – MPE SC– 2022) Analise o código Python a seguir.

```
x1 = {"A", "B", "C"}
x2 = ["AA", "BB", "CC"]
x1.add("B")
x2.append("BB")
x2.append(x1)
print (x2)
```

Dado que os elementos de x1 podem ser exibidos em ordem aleatória, a linha que possivelmente é produzida pelo comando print na execução do código acima é:

a) ['AA', 'BB', 'CC', 'BB', {'C', 'A', 'B'}]
b) ['AA', 'BB', 'CC', 'BB', 'C', 'A', 'B', 'B']
c) ['AA', 'BB', 'CC', 'BB', 'C', 'A', 'B']
d) ['AA', 'BB', 'CC', ['BB'], {'B'}]
e) {'AA', 'BB', 'CC', 'BB', 'C', 'A', 'B'}

Comentários:

Pessoal, primeiramente devemos desconsiderar x1 já que o comando da questão diz que os elementos de x1 podem ser exibidos em ordem aleatória. Vejamos então a ordem de x2. Inicialmente temos um ["AA", "BB", "CC"] que consiste no x2 original. Após, teremos BB (devido ao x2.append ("BB")). Portanto ficamos com 'AA', 'BB', 'CC', 'BB'. Dessa forma ficamos entre as opções a,b, c e e. Porém, depois do x2 há um x1 (x2.append(x1)). Dessa forma, podemos ter um{"A", "B", "C"} em uma sequência aleatória – como diz o comando da questão. A única opção que possui um {"A", "B", "C"} em sequência aleatória é a letra A.

Gabarito: Letra A

---

28. (FGV – MPE SC– 2022) Analise o código Python a seguir.

```
s=0
for k in range(16,10, -2):
s -= k
print (s)
```

O valor exibido pela execução desse trecho é:

a) 0
b) -28
c) -30
d) -42
e) -52

Comentários:

Pessoal, for k in range(16,10, -2):gera 16,14 e 12. Nessa sequência. s -= k vai somar os valores negativos da seguinte forma: -16 - 14 e -12 que gera como resultado -42, nosso gabarito.

Gabarito: Letra D

---

29. (FGV – MPE SC– 2022) Analise o código Python a seguir.

```
x = 0
y = 20
try:
 print (y/x)
except:
 print("Deu erro!")
```

```
else:
 print("Ok")
finally:
 print ("The end")
```

A saída produzida pela execução desse trecho é:
a) a) None
The end

b) Deu erro!
The end

c) Deu erro!
Ok

d) Ok
The end

e) None
Ok
The end

Comentários:

Pessoal, vamos relembrar os conceitos de if, else, finally. finally é usada em blocos try...except. Ele define um bloco de código para ser executado quando o bloco try...except...else for final. O bloco finally será executado independentemente de o bloco try gerar um erro ou não. O foco é esse! Finally será executado independentemente de o bloco try gerar um erro ou não. Portanto, sabemos que vai imprimir o print "The end", já que, vai entrar no finally. Além disso, há um erro, de divisão por zero (ZeroDivisionError: division by zero), dessa forma, também irá imprimir "Deu erro!". O resultado será Deu erro! The end.

Gabarito: Letra B

---

30. (FGV – MPE SC– 2022) Analise o código Python a seguir.

```
def xpto(L):
return (L[::-1])
```

A expressão xpto([1,2,3]) retorna:

a) []
b) [1]
c) [3]
d) [1, 2, 3]

e) [3, 2, 1]

Comentários:

Pessoal, veja como as bancas gostam de inverter listas! Novamente temos L[::-1] que consiste no comando que inverte a lista passada. Dessa forma, sabemos que o resultado será [3, 2, 1].

Gabarito: Letra E

---

31. (FGV – MPE SC– 2022) Analise o código Python a seguir.

```
class xptoClass:
  def __iter__(self):
    self.a = [0]
    return self

  def __next__(self):
    self.a.append( \
        self.a[-1] \
        + self.a[-2] if len(self.a) > 1 else 1)
    return self.a

xpto = xptoClass()
xptoIter = iter(xpto)

for k in range(1,6):
       print(next(xptoIter))
```

No resultado produzido pela execução do código acima, a quinta linha contém exatamente:

a) [0, 1, 1, 2, 2, 3]
b) [0, 1, 1, 2, 3, 5]
c) [0, 1, 2, 3, 4, 5]
d) [0, 1, 3, 5, 7, 9]
e) [1, 2, 3, 4, 5, 6]

Comentários:

Pessoal, essa questão apresenta a sequência de Fibonacci. Na quinta linha temos o resultado [0, 1, 1, 2, 3, 5]. As iterações anteriores são: [0, 1]; [0, 1, 1]; [0, 1, 1, 2]; [0, 1, 1, 2, 3]; [0, 1, 1, 2, 3, 5].

Gabarito: Letra B

---

32. (FGV – IMBEL– 2021) Analise o código Python a seguir.

x = [1,2,3,4,5]
print (x[::-1])
Assinale a opção que indica a saída produzida pela execução desse código.

a) [1,2,3,4,5]
b) 1
c) [5,1]
d) 5
e) [5,4,3,2,1]

Comentários:

Pessoal, novamente cobrança da inversão de lista. x[::-1] retorna o inverso da lista passada. Importante, no caso da questão a lista é denominada x, mas pode ter qualquer nome. As bancas amam! Resposta: e) [5,4,3,2,1].

Gabarito: Letra E

---

33. (FGV – IMBEL – 2021) Analise o código Python a seguir.

x = [1,2,3,4,5]
print (x[-1])
Assinale a opção que indica a saída produzida pela execução desse código.

a) [1,2,3,4,5]
b) 1
c) [5,1]
d) 5
e) [5,4,3,2,1]

Comentários:

Pessoal, (x[-1]) pega o valor da posição 1 do fim da lista para o início. E nesse caso a posição -1 é o valor 5. Portanto, nosso gabarito é a letra D.

Gabarito: Letra D

---

34. (FGV – TCE-AM– 2021) Considere o código Python, versão 2.7.1, na qual o comando print não requer parênteses.

```
def teste(n):
    for k in range(1, n+1):
        yield(k)
for x in teste(10):
    print x
```

A execução desse código:

a) não tem efeito, pois nenhum comando print é acionado;
b) provoca a exibição do número 10 na saída;
c) provoca a exibição dos números de 1 até 10 na saída;
d) provoca um erro de compilação;
e) provoca um erro de execução

Comentários:

Questão um pouco polêmica, porém a FGV claramente diz em seu enunciado: versão 2.7.1, na qual o comando print não requer parênteses. Ok! Já que ela disse, vamos considerar que, neste caso, o comando print não requer parênteses! Porque na Versão 2.7.1, de fato, os códigos Python não precisam dos parênteses para o print. Já, na Versão 3 precisa dos parênteses! Desconsiderando isso, o for da função teste gera uma lista sequencial (1, n+1), e o segundo for apresentar a chamada da função teste com 10 elementos, portanto, a execução desse código provoca a exibição dos números de 1 até 10 na saída.

Gabarito: Letra C

---

35. (FGV – TJ RO– 2021) Analise o código Python 2.7 a seguir.

```
def xpto (n1, n2):
    while n1 != n2:
        if (n1 < n2):
            n2 = n2 - n1
        else:
            n1 = n1 - n2
    return n1
print xpto(50,5)
```

O valor exibido pelo comando print é:

a) 0
b) 1
c) 5
d) 10
e) 50

Comentários:

Pessoal, o while irá executar até que n1 != n2, ou seja, n1 diferente de n2. Dessa forma, ele vai realizar as seguintes operações: 50-5 =45, 45-5=40, 40 -5 = 35 ... até chegar a 5 que é o valor de retorno. Portanto, gabarito letra c.

Gabarito: Letra C

---

36. (FGV – BANESTES– 2021) Considere o código Python 2.7 a seguir:

```
def ABC(L, n):
    while True:
      if len(L) >= n:
       return L
      else:
       L.append(len(L) ** 2)
print ABC([20],10)
```

O resultado da execução desse código é:

a) [1, 4, 9, 16, 25, 36, 49, 64]
b) [1, 4, 9, 16, 25, 36, 49, 64, 81]
c) [20, 1, 4, 9, 16, 25, 36, 49, 64]
d) [20, 1, 4, 9, 16, 25, 36, 49, 64, 81]
e) [20, 4, 9, 16, 25, 36, 49, 64, 81]

Comentários:

Pessoal, é necessário saber que a função principal do código consiste em inserir um elemento no final de uma lista de 10 elementos, perceba: if len(L) >= n aqui, podemos observar que se o tamanho da lista for maior ou igual a n (que vale 10) vai encerrar o Loop e retornar L (a lista). Daí sabemos que a lista terá dez números, então desconsideramos as alternativas com mais ou menos que dez elementos. Nisso, fica apenas a letra D. Mas vamos confirmar nosso gabarito. O primeiro valor é 20 passado no print (ABC([20],10)), os demais serão 1,4,9,16,25,36,49,64,81.

Gabarito: Letra D

---

37. (FGV – FunSaúde CE– 2021) Observe o código Python v2.7.

```
def F (a, b):
```

```
while a != b:
    if a > b:
        a = a - b
    elif b > a:
        b -= a
return a
```

Assinale o valor retornado para F (48,36).

a) 1
b) 12
c) 24
d) 36
e) 48

Comentários:

O código consiste em um loop seguido de um condicional. O loop while roda enquanto a variável a tiver um valor diferente de b, assim, resolve-se a condicional. Como 48 é diferente de 36, entramos no condicional, se 48>36 então, a vai receber a – b. a = 46 e b = 36. A vai receber = 12 e retorna esse valor que será impresso. Assim, nosso gabarito é a letra b.

Gabarito: Letra B

---

38. (FGV – BANESTES– 2021) Considere o código Python a seguir.

```
def F(a, b, c):
 for k in range(a,b):
 print k ** c
```

Dado que uma execução da função F exibiu os números
16, 9, 4, 1, 0, 1,
é Correta afirmar que os valores dos parâmetros a, b, c empregados foram, respectivamente:

a) -4, 1, 2;
b) -4, 2, 2;
c) -4, 0, 4;
d) 4, -1, 1;
e) 4, 2, 2.

Comentários:

Pessoal, nessa questão precisamos saber qual intervalo de valores gera a sequência 16, 9, 4, 1, 0, 1. Sabendo que a função consiste em três valores: a, b e c que em que um valor k é elevado a c. Para gerar 16, 9, 4 precisamos de valores elevado ao quadrado, concordam? 4^2, 3^2 e 2^2 ... Portanto b = 2. Daí eliminamos quase todas as alternativas exceto b e e. Há um erro sutil na letra E. O start (4), primeiro valor, é maior que o stop (2). Por isso, nosso gabarito é a letra B.

Gabarito: Letra B

---

39. (FGV – BANESTES– 2021) Considere o código Python 2.7 a seguir.

```
L=[6,5,4,3,2,1]
for k in range(-3,3):
 print L[k]
```

A execução desse código exibe os números:

a) 1 1 1 6 5 4;
b) 1 2 3 4 5 6;
c) 3 2 1 6 5 4;
d) 6 5 4 3 2 1;
e) 6 5 4 6 5 4.

Comentários:

Pessoal, o for pega os 3 valores – do fim para o início (pega 3,2,1) e depois os três primeiros (pega 6,5,4). Nosso gabarito é 3 2 1 6 5 4.

Gabarito: Letra C

---

40. (FGV – IMBEL– 2021) Analise o script Python 3.8 exibido a seguir.

```
L=["A","E","I","O","U"]
for k in range(-1, -5, -1):
 print (L[k+1])
```

Assinale a saída produzida pela execução desse código.

a) A E I O U
b) A E I U
c) A U O I
d) U A E I
e) U O I E A

Comentários:

Mais uma questão com valores negativos! Vejamos o que ela faz! range(-1, -5, -1). Na verdade, o print pega os índices, nessa sequência: -1 + 1 = 0; -2 + 1 = -1; -3 + 1 = -2; -4 + 1 = -3. Então vai pegar: L[0] = A, L[-1] = U, L[-2] = O , L[-3] = I. Lembrando que os valores negativos iniciam do fim da lista!

Gabarito: Letra C

---

41. (FGV – IMBEL– 2021) Analise o script Python 3.8 exibido a seguir.

```
L=["A","E","I","O","U"]
for k in range(0,len(L)):
 print (L[4-k])
```

Assinale a opção que indica a saída produzida pela execução desse código.

a) A E I O U
b) A E I O
c) E I O U
d) U O I E
e) U O I E A

Comentários:
Pessoal, vamos ver quais valores o print está pegando:
L[4-0] = L[4] = U
L[4-1] = L[3] = O
L[4-2] = L[2] = I
L[4-3] = L[1] = E
L[4-4] = L[0] = A.

Gabarito: Letra E

---

42. (FGV – DPE RJ– 2019) Analise o código Python 2.7 a seguir.

```
frutas = ["banana","laranja","manga","uva"]
for k in range(-1,-4,-2):
    print frutas[k]
```

O conjunto de palavras exibidas pela execução desse código, na ordem, é:

a) banana;
b) laranja, manga;
c) uva, laranja;
d) banana, laranja, manga;

e) banana, laranja, manga;

Comentários:

Inicialmente você deve observar que seu array terá apenas dois elementos, porque a função range está pulando de -2 em -2 elementos – e há 4 elementos no array. O primeiro elemento é o -1, ou seja, uva. O segundo e ultimo será o -3, laranja. Gabarito letra C.

Gabarito: Letra C

---

43. (FGV – MPE AL– 2018) Analise o código Python 2.7 a seguir.

```
L1=[]
L2=[1,2,3,4]
for k in range(3, -4,-1):
 L1.append(L2[k])
for x in L:
 print x
```

Esse programa causa

a) erro de sintaxe.
b) erro de execução.
c)a exibição dos valores 4,3,2,1,4,3,2 nessa ordem.
d) a exibição do valor 4, somente.
e) a exibição dos valores 4,3,2,1 nessa ordem.

Comentários:

Pessoal, ocorrerá um erro de execução, porque a variável L não foi definida (for x in L), veja: NameError: name 'L' is not defined. Se fosse inserido L1 ou L2 no local que foi inserido L, executaria perfeitamente.

Gabarito: Letra B

---

44. (FGV - 2015 – CM/Caruaru - Analista Legislativo - Informática) Analise o código Python a seguir.

```
L1=[10,20,30]
L2=[40,50]
L1.append(L2)
print L1
```

Assinale a opção que descreve corretamente o que acontece quando esse programa é executado no Python 2.7:

a) Produz uma mensagem de erro, porque tenta executar uma operação inválida.
b) Exibe "[10, 20, 30, [40, 50]]".
c) Exibe "[10, 20, 30, 40, 50]".
d) Exibe "[10, 20, 30], [40, 50]".
e) Exibe "[]".

Comentários:

O comando append inclui o valor da variável L2 na posição final do vetor L1. Como o conteúdo da variável L2 é uma lista de tamanho 2 [40,50], ele que será incluído na 4ª posição de L1. *Professor, porque a resposta não é letra C?* Muito bem observado, padawan! O método append inclui a lista L2 como se fosse só um elemento, isto é, a lista final tem 4 elementos ao invés de 5, pois L2 é tratado como se fosse uma coisa só!

Gabarito: Letra B

---

45. (FGV - 2015 – TJ/BA - Analista Judiciário) Analise o trecho de programa Python, na versão 2.7, apresentado a seguir.

```
L=[1,2,3,4,5,6,7,8]
print L[::-1]
```

Ao ser executado, o resultado exibido é:

a) [1, 2, 3, 4, 5, 6, 7, 8]
b) [8]
c) [ ]
d) [8, 7, 6, 5, 4, 3, 2, 1]
e) [1]

Comentários:

O operador de acesso a itens em coleções (listas, sets, tuplas) usam três argumentos:

L[Start:Stop:Step]

Start - primeira posição a ser acessada:

Pode ser: (1) valor positivo - posição inicial de acesso (por exemplo, 0 é a primeira, 1 é a segunda); (2) valor negativo - posição a partir do final do array (por exemplo, -1 é a última posição, -2 é a penúltima); (3) não fornecido - primeira posição do array (posição 0).

Stop - última posição a ser acessada:

Pode ser: (1) valor positivo - posição de acesso (por exemplo, 0 é a primeira, 1 é a segunda); (2) valor negativo - posição a partir do final do array (por exemplo, -1 é a última posição, -2 é a penúltima); (3) não fornecido - última posição do array (posição -1).

Step - valor do incremento no acesso:

Pode ser: (1) número positivo - ordem direta de incremento em incremento; (2) número negativo - ordem inversa de incremento em incremento; (3) não fornecido - ordem de 1 em 1. Quando só step é fornecido e ele é negativo, ele acessa a coleção de forma invertida a partir do último elemento.

Dessa forma, o acesso L[::-1] acessará da primeira até a última posição do array em ordem inversa, ou seja, [8, 7, 6, 5, 4, 3, 2, 1].

Gabarito: Letra D

46.(FGV – 2014 – MPE/AL) Analise o código Python 2.7 a seguir.

```
L = [10, 12, 14, 16]
for k in range(4, -5, -1):
print L[k]
```

Esse programa causa:

a) erro de sintaxe.
b) erro de execução.
c) a exibição de 4 valores, 16,14,12,10, nessa ordem.
d) a exibição de 8 valores, 16,14,12,10,16,14,12,10, nessa ordem.
e) a exibição do valor 16, somente.

Comentários:

A questão é fácil de responder porque ela possui erro de sintaxe! A sintaxe exige que o bloco for tenha um corpo, e o corpo precisa seguir o ":" ou estar dentro de um bloco indentado abaixo do "for". Perceba que não acontece nenhum dos dois casos na questão: não existe nada depois do ":", nem existe comando indentado abaixo do "for". E mesmo que estivesse corretamente indentado, ainda haveria um erro de execução porque o range(4, -5, -1) percorre de 4 a -4 em passos de -1 e não teria como imprimir L[4] porque esse valor sequer existe, L só vai de 0 a 3.

Gabarito: Letra A

Questões FCC

47. (FCC – PGE AM– 2022) Em um programa escrito em Python, uma série de dados foram inseridos no array cargos, por meio da instrução abaixo.

cargos = ["Advogado","Promotor", "Procurador", "Juiz", "Desembargador", "Ministro"];

Para colocar estes dados em ordem alfabética decrescente em um novo array chamado cargos_ordenados utiliza-se a instrução:

a) for cargos_ordenados in cargos reverse True;
b) cargos_ordenados = sorted(cargos,reverse=True);
c) while cargos: cargos_ordenados = cargos- -;
d) cargos_ordenados = descending(cargos);
e) cargos_ordenados = ordered(cargos, descending=True);

Comentários:

Pessoal, sort() ordena a lista, no caso, a lista "cargos". A opção reverse=True Inverte a ordem da lista. Portanto nosso gabarito é a Letra B. O resultado da execução da letra B é: ['Promotor', 'Procurador', 'Ministro', 'Juiz', 'Desembargador', 'Advogado']. As demais alternativas geram erros e não existem.

Gabarito: Letra B

---

48. (FCC – TRF 4 – 2019) Considerando que em um programa Python em condições ideais há um array criado pelo comando nomes = ["Maria", "Pedro", "João"], para exibir os valores contidos nesse array utiliza-se

a) for x in nomes: out.print(x)
b) while x in nomes: print(x)
c) foreach x in nomes: print(x)
d) foreach x in nomes: system.println(x)
e) for x in nomes: print(x)

Comentários:

Pessoal, fizemos vários exemplos e questões usando o for e o print dos arrays. O Correta é for x in nomes: print(x).

Gabarito: Letra E

---

49. (FCC – TRF 4– 2019) Considere o código Python abaixo.

def oper(l, item):

```
pos = 0
x = False
___I___
if l[pos] == item:
x = True
else:
pos = pos+1
return x
v = [1, 2, 32, 8, 17, 19, 42, 13, 0]
print(oper(v, 8))
```

Para que o código exiba na tela o valor True se o item buscado no vetor por meio da função oper for encontrado, a lacuna I deve ser corretamente preenchida por

a) while pos < l and not x:
b) while pos < len(l) && x:
c) while pos < len(l) and not found(x):
d) while (pos < len(l) && not(x)):
e) while pos < len(l) and not x:

Comentários:

A função "oper" irá retornar "True" se o valor for encontrado no vetor ou "False" caso não encontrado. Assim, é necessário que a lacuna I seja preenchida por uma estrutura de repetição while para percorrer o vetor além do " and not x" para verificar se o item buscado no vetor por meio da função "oper" foi encontrado.

Gabarito: Letra E

---

50. (FCC – SANASA– 2019) Uma Analista de TI está desenvolvendo um scanner de rede em Python e, para importar um recurso referente para manipulação de pacotes de rede, utilizou no programa a linha

a) import pyTTS *
b) from Tkinter import *
c) import aiml *
d) from scapy.all import *
e) from numpy import *

Comentários:

Pessoal, a questão solicita um pacote para manipulação de pacotes de rede. Vejamos o que faz cada um dos comandos citados: pyTTS: biblioteca de texto para fala; Tkinter: biblioteca padrão da linguagem Python; aiml: biblioteca para inteligência artificial com aprendizado de máquina; scapy:

biblioteca que suporta envio de pacotes em rede, portanto nosso gabarito. numpy: biblioteca para uso científico, trabalha com vetores multidimensionais.

Gabarito: Letra D

---

51. (FCC – TJ MA– 2019) Considere o programa Python abaixo:

```
numero1 = int(input('Informe o número de Processos: '))
numero2 = int(input('Informe o número de Juízes: '))
..
I
..
 resultado = numero1 / numero2
 print("Há ",resultado, " processos a serem julgados por cada Juiz")
..
II
..
 print("Não é possível divisão por zero")
```

Para tratar a exceção que será lançada se o valor contido na variável numero2 for zero, as lacunas I e II deverão ser corretamente preenchidas por:

a) try: e catch ArithymeticException:
b) throw e catch (ZeroDivisionException $e)
c) try e catch(ArithmeticException ex)
d) throw: e catch(err)
e) try: e except ZeroDivisionError:

Comentários:

Pessoal, de imediato podemos eliminar as opções que não possuem o try. Portanto ficamos apenas com a, c e e. A questão deu a dica: "Não é possível divisão por zero". Daí podemos verificar que o gabarito é a letra E porque há uma divisão no try (resultado = numero1 / numero2).

Gabarito: Letra E

---

52. (FCC – TJ MA– 2019) Considere o programa Python abaixo:

```
..
I..
 for x in pro:
 print(x)
processos = ["0456789908", "0875643087", "0897645109"]
exibir_processos (processos)
```

Para que o programa seja executado corretamente, em condições ideais, a indicação I deve ser substituída por:

a) private exibir_processos(pro):
b) public exibir_processos(pro):
c) function exibir_processos(pro):
d) definition exibir_processos(pro):
e) def exibir_processos(pro):

Comentários:

Pessoal, fizemos várias questões com definição de funções, vamos relembrar? Para criar uma função, utiliza-se a palavra-chave def. Daí já conseguimos saber que o nosso gabarito é a letra E.

Gabarito: Letra E

---

53. (FCC – MPE PE– 2018) Considere o fragmento de código Python abaixo.

```
class Cliente:
I
.....
self.nome = nome
self.renda = renda
p1 = Cliente("Maria", 5678.98)
print(p1.nome)
print(p1.renda)
```

Para que o código seja compilado e executado corretamente, a lacuna I deverá ser preenchida com

a) __init__(self, nome, renda):
b) function __init(self, nome, renda):
c) def __construct(self, nome, renda):
d) def __init__(self, nome, renda):
e) Cliente(self, nome, renda):

Comentários:

Pessoal, todas as classes possuem uma função chamada __init__(), que sempre é executada quando a classe está sendo iniciada. Para que o código seja compilado e executado corretamente devemos utilizar def __init__(self, nome, renda):.

Gabarito: Letra D

---

54. (FCC – TRE SP– 2017) Considere o programa Python, abaixo.

```
import ..I.. as b
```

```
import matplotlib.pyplot as a
x = b.linspace(0, 3, 20)
y = b.linspace(0, 9, 20)
a.plot(x, y)
a.plot(x, y, 'o')
a.show()
```

A lacuna I deve ser preenchida corretamente com

a) numpy
b) matrix
c) mathlab
d) numberplot
e) array

Comentários:

Pessoal, NumPy é uma biblioteca Python usada para trabalhar com arrays. Ela possui funções para trabalhar no domínio da álgebra linear, transformada de Fourier e matrizes. NumPy significa Python Numérico. Para usar a linspace() é necessário importar a biblioteca NumPy.

Gabarito: Letra A

---

55. **(FCC – PRODATER– 2016)** Em Python existe um conjunto de métodos disponíveis para se trabalhar com objetos do tipo lista. Considere o trecho de programa abaixo que faz uso desses métodos.

```
a = [99.15, 323, 323, 2, 12.5]
a.insert(2, -5)
a.append(323)
a.index(323)
a.remove(323)
a.reverse()
```

Ao executar este trecho de programa, o conteúdo final da lista a será:

a) [99.15, −5, 323, 2, 12.5, 323]
b) [−5, 2, 12.5, 99.15, 323, 323]
c) [323, 12.5, 2, 323, −5, 99.15]
d) [99.15, −5, 323, 323, 12.5, 2]
e) [323, 323, 99.15, 12.5, 2, −5]

Comentários:

Pessoal, vejamos o que cada operação gera na lista a:

a.insert(2, -5) # [99.15, 323, -5, 323, 2, 12.5]
a.append(323) # [99.15, 323, -5, 323, 2, 12.5, 323]
a.index(323) # [99.15, 323, -5, 323, 2, 12.5, 323] (não faz nada)
a.remove(323) # [99.15, -5, 323, 2, 12.5, 323] (removeu o primeiro 323)
a.reverse() # [323, 12.5, 2, 323, -5, 99.15] (inverteu a lista)

Gabarito: Letra C

---

56. (FCC – SEMF Teresina– 2016) Considere o programa abaixo, criado na linguagem Python.

```
a = ['ARSETE', 'PRODATER', 'SEMEST', 'STRANS', 'SEMAE']
...
I...
print i, a[i]
```

Considere a saída gerada pelo programa:

```
0 ARSETE
1 PRODATER
2 SEMEST
3 STRANS
4 SEMAE
```

Na lacuna I deve estar o comando:

a) for i in len(a):
b) foreach a as &value(a)):
c) while i<range(len(a)):
d) for i in range(len(a)):
e) for i in range(a):

Comentários:

Pessoal, o Correta a se usar quando é necessário percorrer o array é a letra d: for i in range(len(a)):

Gabarito: Letra D

---

57. (FCC – PGM Teresina– 2016) Considere o código-fonte abaixo, criado na linguagem Python.

```
def dados(n):
resultado = []
a, b = 0, 1
while a < n:
```

```
resultado.append(a)
a, b = b, a+b
return resultado
op = dados(100)
print op
```

O valor final contido na posição de índice 6 de op é

a) 1
b) 8
c) 5
d) 6
e) 13

Comentários:

Pessoal, é apresentado um código que gera a sequência de Fibonacci – muito cobrado em provas principalmente quando se fala em Python! A sequencia gerada é: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. A posição 6 consiste do valor = 8, nosso gabarito.

Gabarito: Letra B

---

58. (FCC – CNMP– 2015) Considere os fragmentos de programas Phyton a seguir:

Fragmento 1:

```
for n in range(2, 10):
for x in range(2, n):
 if n % x == 0:
 print n, '=', x, '*', n/x
 break
else:
 print n, 'é um número primo'
```

Fragmento 2:

```
a = ['Casa', 'Mala', 'Prova']
for x in a:
 print x, len(x)
```

É Correta afirmar que

a) o Fragmento 1 está in Correta, pois laços não podem ter uma cláusula else.
b) no Fragmento 2, a instrução for está incorreta, pois ela não pode iterar sobre a.

c) o Fragmento 1 está in Correta, pois não é possível iterar sobre sequências numéricas utilizando a função range.
d) no Fragmento 1 é verificado se o quociente da divisão de n por x corresponde a 0.
e) os dois fragmentos de código estão Corretas.

Comentários:

Pessoal, ambos fragmentos estão perfeitos.

Gabarito: Letra E

---

59. (FCC – TRT/MG - Técnico Judiciário - Tecnologia Da Informação) Considere o código fonte Python abaixo.

```
def calcular(n):
    resultado = []
    a, b = 0, 1
    while a < n:
              I
            .....
    return resultado

res = calcular(100)
print res
```

Para que seja exibido [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89] a lacuna "I" precisa ser preenchida corretamente com:

a) resultado.add(a)
    a, b = b, a+b
b) resultado.insert(a)
    a, b = b, a+b
c) resultado.append(a)
    a, b = b, a+b
d) resultado.add(a)
    a, b = a, a+b
e) resultado.append(a)
    a, b = a+b, b

Comentários:

Primeiro temos que saber que o Correta a ser utilizado é o append, pois queremos inserir valores a cada iteração sem perder os anteriores. Então, ficamos entre a letra (c) e (e). Em (c), temos **a, b = b, a+b** equivale a **a = b e b = a+b**; e em (e), temos que **a, b = a+b, b** equivale a **a = a+b** e **b = b**. Vamos testar a letra (e)! Antes de entrar na estrutura de repetição while, nós temos que:

a = 0
b = 1

Depois, temos que:

a = a + b
b = b

Note que – se antes b = 1 e depois b = b – então b nunca vai mudar, b sempre será 1. Como antes a = 0 e depois a = a + b, então a sempre aumentará em uma unidade. Logo, resultado em uma sequência [1, 2, 3, 4, 5, 6, 8, 9, ...]. Dessa forma, a resposta é (c)! Notem que ele se trata de uma Sequência de Fibonacci.

```
1ª iteração: resultado = [0]; a = 1; b = 1;
2ª iteração: resultado = [0, 1]; a = 1; b = 2;
3ª iteração: resultado = [0, 1, 1]; a = 2; b = 3;
4ª iteração: resultado = [0, 1, 1, 2]; a = 3; b = 5;
5ª iteração: resultado = [0, 1, 1, 2, 3]; a = 5; b = 8;
6ª iteração: resultado = [0, 1, 1, 2, 3, 5]; a = 8; b = 13;
7ª iteração: resultado = [0, 1, 1, 2, 3, 5, 8]; a = 13; b = 21;
8ª iteração: resultado = [0, 1, 1, 2, 3, 5, 8, 13]; a = 21; b = 34;
9ª iteração: resultado = [0, 1, 1, 2, 3, 5, 8, 13, 21]; a = 34; b = 55;
10ª iteração: resultado = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]; a = 55; b = 89;
11ª iteração: resultado = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]; a = 89; b = 144;
12ª iteração: resultado = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]; a = 144; b = 233;
```

Gabarito: Letra C

60.(FCC – 2012 – MPE/PE – Analista de Sistemas) Em Python, os métodos de lista permitem utilizar listas como pilhas, onde o item adicionado por último é o primeiro a ser recuperado. Para adicionar um item ao topo da pilha, e para recuperar um item do topo da pilha utilizam-se, respectivamente os métodos:

a) append() e pop().
b) insert() e top().
c) addTop() e pop().
d) add() e get().
e) addItem() e top().

Comentários:

append( ) é utilizado para adicionar um elemento no final de uma lista ou no topo de uma pilha; pop( ) é utilizado para recuperar um item de uma lista ou do topo de uma pilha.

Gabarito: Letra A

61.(FCC – 2012 – TRE/CE – Analista de Sistemas) Sobre Python é Correta afirmar:

a) é uma linguagem compilada, ou seja, o código-fonte de um programa é lido pelo compilador, que cria um arquivo binário, executável diretamente pelo hardware.

b) É uma linguagem fortemente tipada, ou seja, é preciso declarar variáveis e seus tipos.

c) Suporta funcionalidades comuns na orientação a objetos: herança, herança múltipla, polimorfismo, reflexão e introspecção.

d) Uma lista em Python é um conjunto de valores acessados por um índice numérico, inteiro, começando em 1. Assim como em outras linguagens, a lista pode armazenar apenas valores de um mesmo tipo.

e) Uma String Python é uma sequência imutável, alocada estaticamente, com restrição de tamanho.

Comentários:

(a) Errado, é uma linguagem interpretada; (b) Errado, ele realmente possui tipagem forte, mas não precisa declarar variáveis e seus tipos; (c) Correta, ele realmente suporta funcionalidades comuns da orientação a objetos como herança simples e múltipla, polimorfismo, reflexão e introspecção; (d) Errado, a lista pode armazenar valores de tipos diferentes e começa em zero; (e) Errado, ela não possui restrição de tamanho.

Gabarito: Letra C

QUESTÕES COMENTADAS – DIVERSAS BANCAS

62.(IESES - 2015 - IFC-SC - Programação Web e Dispositivos Móveis) Sobre listas em Python 3.1.5:

a) list.remove(a) remove o primeiro item da lista cujo valor é a.
b) list.pop(a) adiciona um item de valor a ao início da lista.
c) list.append(a) adiciona um item à lista cujo índice será a.
d) list.index(a) retorna o valor do item cujo índice é a.

Comentários:

(a) Correta. E caso não exista o valor "a", retorna erro; (b) Errado, ele remove o item na posição "a" da lista ou – caso seja chamado sem argumento – remove o último item da lista; (c) Errado, ele adiciona o valor "a" na última posição da lista; (d) Errado, index retorna a posição da primeira ocorrência de "a" na lista.

Gabarito: Letra A

63.(IESES - 2015 - IFC-SC - Programação Web e Dispositivos Móveis) O conjunto Correta de palavras reservadas para a construção de uma estrutura de controle em Python 3.4.3 é:

a) "if", "elsif" e "else".
b) "if", "else if" e "else".
c) Somente "if"; o restante da estrutura de controle ("senão se" e "senão") é realizado simplesmente com indentação.
d) "if", "elif" e "else".

Comentários:

A estrutura correta completa é: if, elif e else.

Gabarito: Letra D

64.(UERJ - UERJ - Analista De Sistemas - Grid) Considere o trecho do programa Python abaixo:

```
def dobra (y):
        x = y + y
        return x

x = 5
dobra(x)
```

```
dobra(x)
print x
```

O valor impresso ao executarmos o programa é:

a) 5
b) 10
c) 15
d) 25

Comentários:

Primeiro, uma observação: as variáveis possuem escopos diferentes. Há duas ocorrências de "x" no código-fonte, mas uma está dentro do escopo local da função dobra( ) e o outro está fora (escopo global). Dito isso, se x = 5, temos que dobra(5). Logo, dentro do escopo dessa função, ele é recebido como y. Então, x = y + y = 5 + 5 = 10. Depois, chama-se a função dobra(x) novamente, mas – dentro desse escopo – x continua igual a 5.

Logo, repetiremos o que acabamos de fazer: x = y + y = 5 + 5 = 10. Por fim, imprime-se x. Ora, o x impresso aqui tem escopo global e, não, local. Logo, ele terá valor = 5.

Gabarito: Letra A

---

65. (UERJ - UERJ - Analista De Sistemas) A linguagem Python possui a seguinte característica:

a) é uma linguagem compilada.
b) exige declaração de código.
c) a tupla é um tipo mutável.
d) é orientada a objetos.

Comentários:

(a) Errado, é uma linguagem interpretada; (b) Errado, não sei o que a questão quis dizer com declaração de código. Se for declaração de tipo, então não exige; (c) Errado, a tupla é equivalente a uma lista, porém imutável; (d) Correta, é realmente orientada a objetos.

Gabarito: Letra D

---

66. (CETAP – 2010 – AL/RR – Analista de Sistemas) Sobre a linguagem de programação PYTHON, marque a alternativa INCORRETA.

a) Python suporta a maioria das técnicas da programação orientada a objetos.

b) Python suporta e faz uso constante de tratamento de exceções como uma forma de testar condições de erro e outros eventos inesperados no programa.

c) As funções são definidas em Phyton utilizando a palavra chave def.

d) A separação de blocos de código em Phyton é feita utilizando a indentação de código.

e) O operador lógico de conjunção ("e", como em a e b) é &&.

Comentários:

(a) Correta, ele realmente suporta a maioria das técnicas da programação orientada a objetos; (b) Correta, ele realmente suporta e faz uso constante de tratamento de exceções como uma forma de testar condições de erro e outros eventos inesperados no programa. Caso haja algum erro, o programa não é interrompido – o erro é tratado sem interromper o programa; (c) Correta, funções são definidas por meio da palavra-chave def; (d) Correta, a separação de blocos se dá por meio da indentação – ignorem o nome errado: é Python e, não, Phyton; (e) Errado, o operador lógico de conjunção é and e, não, &&.

Gabarito: Letra E

---

67. (CESGRANRIO – 2004 – SECAD/TO – Analista de Sistemas) Um programador de Python recebeu a tarefa de criar uma função chamada calcular que recebe dois parâmetros. Para executar sua atividade, ele deve utilizar a expressão:

a) def calcular (a,b):
b) function calcular (a,b):
c) import calcular (a,b):
d) procedure calcular (a,b):
e) sub calcular (a,b):

Comentários:

Uma função é definida por meio da palavra-chave **def**.

Gabarito: Letra A

---

68. (UNIRIO – 2014 – UNIRIO) Sobre o comando range para construção de listas na linguagem Python, é CORRETA afirmar que:

a) range(4,6) gera a lista [4,5].
b) range(5) gera a lista [1,2,3,4,5].
c) range(4,6) gera a lista [4,5,6,7,8,9].

d) range(5,1) gera a lista [5].
e) range(5,1,-2) gera a lista [4,5].

Comentários:

(a) Correta, range(4,6) retorna a lista [4, 5] porque não inclui o valor de índice [6]; (b) Errado, range(5) retorna a lista [0, 1, 2, 3, 4]; (c) Errado, já vimos que range(4,6) retorna a lista [4, 5]; (d) Errado, range(5,1) retorna [ ] porque o step padrão é 1 e o stop é menor que o start; (e) Errado, range(5,1,-2) retorna [5, 3] porque o step é negativo.

Gabarito: Letra A

---

69. (QUADRIX – 2018 – COREN/RS) No que se refere à linguagem de programação Python, assinale a alternativa correta.

a) A Python é uma linguagem de alto nível e robusta. Ela possui seu próprio framework e é incompatível com frameworks de terceiros.

b) A Python utiliza a duck typing (tipagem dinâmica), que nada mais é do que definir um tipo para a variável, com as operações que podem ser aplicadas, antes mesmo de ela ter sido criada, com base em conhecimento prévio do programa. Esta tarefa é executada pelo interpretador.

c) O caractere "/" marca o início de comentário. Qualquer texto depois do "/"será ignorado até o fim da linha.

d) A Python permite que os conteúdos das variáveis sejam sempre alterados, não existindo, dessa forma, tipos imutáveis.

e) Pode ser utilizada como linguagem principal no desenvolvimento de sistemas e também pode ser utilizada como linguagem script em vários softwares.

Comentários:

(a) Errado, ela realmente é uma linguagem de alto nível e robusta, mas seu framework (coleção de pacotes e módulos de extensão) é totalmente compatível com frameworks de terceiros; (b) Errado, ela realmente utiliza Duck Typing, mas não é necessário definir um tipo para variável antes de ela ser criada – isso pode ser inferido pelo interpretador; (c) Errado, comentários são inicializados por meio do caractere "#" e, não, "/"; (d) Errado, tipos podem ser mutáveis ou imutáveis; (e) Correta, além de ser utilizado como linguagem principal no desenvolvimento de sistemas, o Python também é muito utilizado como linguagem script em vários softwares.

Gabarito: Letra E

---

70. **(IFPI – 2012 – IFPI)** Com relação à linguagem de programação Python, é INCORRETA a afirmação:

a) Disponibiliza o conceito de herança múltipla, ou seja, uma classe pode ter mais de uma classe pai.

b) A sintaxe Python permite o uso de somente uma instrução por linha de código. Para que seja possível colocar mais de uma, é necessário separá-las com um ";", entretanto, essa prática é desaconselhada, visto que sua utilização prejudica a legibilidade do código.

c) Os comentários são iniciados com um caractere "#". Para fazer comentários com múltiplas linhas, utilizam-se os caracteres "#", para o início, e "#*" para o final .

d) Diferentemente de outras linguagens de programação que utilizam elementos léxicos para definir os blocos de código, a própria endentação do código é usada, em Python, para essa função.

e) Em Python, os nomes de variáveis, funções, módulos e classes são conhecidos como identificadores. O identificador _quantidade é Correta em Python.

Comentários:

(a) Correta, ele realmente possui herança múltipla, isto é, uma classe com mais de um pai; (b) Correta, Python deve ser bastante legível. É possível inserir ponto-e-vírgula para separar instruções em uma mesma linha, mas prejudica a visibilidade – o ideal mesmo é deixar somente uma instrução por linha; (c) Errado, para comentários de mais de uma linha, utiliza-se três aspas simples ou duplas; (d) Correta, a indentação delimita blocos de código; (e) Correta, esse é um identificador válido.

Gabarito: Letra C

LISTA DE QUESTÕES

Questões Cespe

1. **(CESPE – DPE RO – 2022)** Na linguagem Python, são consideradas sequências mutáveis as

a) strings
b) cadeias
c) tuplas
d) listas
e) ranges

2. **(CEBRASPE – PC PB– 2022)** Na linguagem Python, o tipo de uma variável em tempo de execução é definido pelo interpretador pelo recurso denominado

a) tipagem dinâmica.
b) modo interativo.
c) sintaxe.
d) interpretação bytecode.
e) empacotamento.

3. **(CESPE – PC PB– 2022)** Python é uma linguagem procedural que utiliza quatro tipos de dados predefinidos para lidar com coleções: conjuntos, dicionários, listas e tuplas. A respeito desses tipos de dados, julgue os itens a seguir.

I O conjunto permite o armazenamento de uma tupla, mas não o de uma lista.
II A tupla é idêntica à lista, exceto pela forma mais simples com que sua declaração é realizada.
III A lista é um tipo de dados variável que permite a alteração de seus elementos após a sua criação.
Assinale a opção correta.

a) Apenas o item I está certo.
b) Todos os itens estão certos.
c) Apenas o item II está certo
d) Apenas os itens I e III estão certos.
e) Apenas os itens II e III estão certos.

4. **(CEBRASPE – (CODEVASF– 2021)** Na linguagem Python, as listas são coleções de qualquer tipo de objetos, com exceção das próprias listas, e seus elementos são alteráveis.

5. **(CEBRASPE – PF– 2021)** O código Python a seguir apresenta como resultado "True".

```
x = bool(-3)
y = bool("True"*x)
z = bool("False")
print (x and y and z)
```

6. **(CEBRASPE – SERPRO– 2021)** As tuplas, embora sejam semelhantes às listas, estão limitadas a, no máximo, cinco níveis.

7. **(CEBRASPE – SERPRO– 2021)** Listas são coleções alteráveis de qualquer tipo de objeto — como, por exemplo, outras listas — capazes de gerar um efeito top-down sem limite de níveis.

8. **(CEBRASPE – PC DF– 2021)** Com relação a mineração de dados, aprendizado de máquina e aplicações Python, julgue o item a seguir.

Uma das aplicações de Python é o aprendizado de máquina, que pode ser exemplificado por um programa de computador que aprende com a experiência de detectar imagens de armas e de explosivos em vídeos, tendo seu desempenho medido e melhorado por meio dos erros e de acertos decorrentes da experiência de detecção.

9. **(CEBRASPE – BANESE– 2021)** No que se refere ao pacote NumPy do Python, julgue o item subsequente. O código a seguir retorna o valor do desvio padrão amostral do conjunto de dados {1,2,2,3,5}

```
import numpy
x = numpy.array([1,2,2,3,5])
numpy.std(x,ddof=1)
```

10. **(CEBRASPE – SEED PR– 2021)** Em Python, quando mais de um operador aparece em uma expressão, a ordem de avaliação depende das regras de precedência de cada linguagem. Assim, ao programar em Python, além de observar essas regras, é preciso considerar, ainda, a forma como a linguagem representa seus operadores, conforme demonstrado nos comandos a seguir.

```
x=7*3**2%4
print (x)
```

Assinale a opção que corresponde à saída que o compilador Python apresentará para os comandos em questão.

a) 1
b) 3
c) 7
d) 15
e) 15.75

11. (CEBRASPE – SEED PR– 2021) Na linguagem de programação Python, existem 3 estruturas para armazenar dados indexados. A estrutura cujos valores são imutáveis depois de sua criação é conhecida como

a) lista
b) operador
c) tupla
d) classe
e) dicionário

12. (CEBRASPE – PF– 2018) Considere os seguintes comandos na programação em Python.

```
a = " Hello, World! "
print(a.strip())
```

Esses comandos, quando executados, apresentarão o resultado a seguir.

a[0]=Hello,
a[1]=World!

13. (CEBRASPE – PF– 2018) Considere o programa a seguir, na linguagem Python.

```
letras == ["P", "F"]
for x in letras
 {
 print(x)
 }
```

A sintaxe do programa está correta e, quando executado, ele apresentará o seguinte resultado.

PF

14. (CESPE – 2013 – MPOG – Analista de Sistemas) Em Python, o comando int("1") cria um objeto do tipo int, que recebe 1 como parâmetro no seu construtor.

15. **(CESPE – 2013 – MPOG – Analista de Sistemas)** Em Python, o comando int("1") cria um objeto do tipo int, que recebe 1 como parâmetro no seu construtor.

16. **(CESPE – 2011 – ECT – Analista de Sistemas)** A linguagem Python e seu interpretador estão disponíveis para as mais diversas plataformas. Para que seja usado em determinado sistema operacional não suportado, é possível gerar o Python a partir do programa fonte utilizando um compilador C. Nesse caso, o código fonte é traduzido para o formato bytecode, que é multiplataforma e pode ser distribuído de forma independente.

17. **(CESPE – 2008 – SERPRO – Analista de Sistemas)** Python é uma linguagem livre de alto nível, orientada a objetos e de difícil leitura, pois não permite identação de linhas de código.

18. **(CESPE – 2011 – ECT – Analista de Sistemas)** A linguagem Python e seu interpretador estão disponíveis para as mais diversas plataformas. Para que seja usado em determinado sistema operacional não suportado, é possível gerar o Python a partir do programa fonte utilizando um compilador C. Nesse caso, o código fonte é traduzido para o formato bytecode, que é multiplataforma e pode ser distribuído de forma independente.

19. **(CESPE – 2008 – SERPRO – Analista de Sistemas)** Python é uma linguagem livre de alto nível, orientada a objetos e de difícil leitura, pois não permite identação de linhas de código.

20. **(CESGRANRIO – 2004 – SECAD/TO – Analista de Sistemas)** Um programador de Python recebeu a tarefa de criar uma função chamada calcular que recebe dois parâmetros. Para executar sua atividade, ele deve utilizar a expressão:

a) def calcular (a,b):
b) function calcular (a,b):
c) import calcular (a,b):
d) procedure calcular (a,b):
e) sub calcular (a,b):

Questões FGV

21. (FGV – TCU – 2022) Considere o código Python a seguir.

```
def xpto(S):
    for k in range(0,len(S)):
        if k%2 == 0:
            yield(S[k]);

S=[1,2,3,4,5,6]
for x in xpto(S[::-1]):
    print (x)
```

A execução desse código na IDLE Shell produz, na ordem e exclusivamente, os números:

a) 6, 1
b) 5, 3, 1
c) 6, 4, 2
d) 1, 3, 5
e) 2, 4, 6

22. (FGV – PC AM– 2022) Considere o código Python a seguir.

```
L=[0,1,1,2,3,5,8,13,21]
for k in range(0,len(L),2):
print (L[k])
```

Assinale o resultado exibido pela execução desse código, na IDLE Shell 3.9.9.
a) 1, 2, 5, 13
b) 0, 1, 3, 8, 21, 1, 2, 5, 13, 21
c) 1, 3, 8, 21, 1, 2, 5, 13, 21
d) 0, 1, 3, 8, 21
e) 0, 1, 3, 8

23. (FGV – SEFAZ AM– 2022) Analise o código a seguir em linguagem de programação Python:

```
def rotina(array):
    for p in range(0, len(array)):
        element = array[p]

        while p > 0 and array[p - 1] > element:
            array[p] = array[p - 1]
            p -= 1

        array[p] = element

    return array

print ( rotina([9, 5, 31, 42, 20, 56] ) )
```

Ao executar esse script em um terminal, será escrito na saída padrão
a) [9, 5, 31, 42, 20, 56]
b) [8, 4, 30, 41, 19, 55]
c) [56, 20, 42, 31, 5, 9]
d) [56, 42, 31, 20, 9, 5]
e) [5, 9, 20, 31, 42, 56]

24. (FGV – MPE GO– 2022) Considere o código Python a seguir.

```
def X(n):
    if (type(N) != int):
        return -1
    elif (N < 1):
        return 0
    elif (N == 1):
        return 1
    else:
        return N * X(N-1)
print (X(4))
print (X(0))
print (X(1))
print (X(1.5))
print (X("A"))
```

Assinale o que acontece quando esse script é executada na IDLE Shell 3.9.9.
a) Erro de compilação, "name 'n ' is not defined"
b) Erro de compilação, "name 'N' is not defined"
c) Executa e produz resultados Corretas com quatro linhas.
d) Executa, mas produz erro de execução na quinta chamada da função X.
e) Executa, mas calcula erradamente o fatorial de 4.

25. (FGV – MPE GO– 2022) Assinale a lista de números produzida pela execução, na IDLE Shell 3.9.9, do código Python a seguir.

```
for x in range(-1, -10, -1):
 print (x)
```

a) -1 -2 -3 -4 -5 -6 -7 -8 -9
b) -9 -8 -7 -6 -5 -4 -3 -2 -1
c) 0 -1 -2 -3 -4 -5 -6 -7 -8 -9
d) 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
e) -1 -2 -3 -4 -5 -6 -7 -8 -9 -10

26. (FGV – TJDFT– 2022) Analise o código Python 3.9 a seguir.

```
class Teste:
def ----------
self.altura = xaltura
self.largura = xlargura
def dimensoes(self):
 print("altura = " + str(self.altura) + "\n" \ + "largura = " + str(self.largura))
x = Teste(12, 20)
x.dimensoes()
```

Para que a execução desse código exiba
altura = 12
largura = 20
o trecho tracejado na segunda linha deve ser substituído por:

a) __init__(self, xaltura, xlargura):
b) __init__(xaltura, xlargura):
c) init (xaltura, xlargura):
d) new (self, args[xaltura, xlargura]):
e) new (self, xaltura, xlargura):

27. (FGV – MPE SC– 2022) Analise o código Python a seguir.

```
x1 = {"A", "B", "C"}
x2 = ["AA", "BB", "CC"]
x1.add("B")
x2.append("BB")
x2.append(x1)
print (x2)
```

Dado que os elementos de x1 podem ser exibidos em ordem aleatória, a linha que possivelmente é produzida pelo comando print na execução do código acima é:

a) ['AA', 'BB', 'CC', 'BB', {'C', 'A', 'B'}]
b) ['AA', 'BB', 'CC', 'BB', 'C', 'A', 'B', 'B']
c) ['AA', 'BB', 'CC', 'BB', 'C', 'A', 'B']
d) ['AA', 'BB', 'CC', ['BB'], {'B'}]
e) {'AA', 'BB', 'CC', 'BB', 'C', 'A', 'B'}

28. (FGV – MPE SC– 2022) Analise o código Python a seguir.

```
s=0
for k in range(16,10, -2):
s -= k
print (s)
```

O valor exibido pela execução desse trecho é:

a) 0
b) -28
c) -30
d) -42
e) -52

29. (FGV – MPE SC– 2022) Analise o código Python a seguir.

```
x = 0
y = 20
try:
 print (y/x)
except:
 print("Deu erro!")
else:
 print("Ok")
finally:
 print ("The end")
```

A saída produzida pela execução desse trecho é:
a) a) None
The end

b) Deu erro!
The end

c) Deu erro!
Ok

d) Ok
The end

e) None
Ok
The end

30. (FGV – MPE SC– 2022) Analise o código Python a seguir.

def xpto(L):
return (L[::-1])
A expressão xpto([1,2,3]) retorna:

a) []
b) [1]
c) [3]
d) [1, 2, 3]
e) [3, 2, 1]

31. (FGV – MPE SC– 2022) Analise o código Python a seguir.

```
class xptoClass:
  def __iter__(self):
    self.a = [0]
    return self

  def __next__(self):
    self.a.append( \
        self.a[-1] \
        + self.a[-2] if len(self.a) > 1 else 1)
    return self.a

xpto = xptoClass()
xptoIter = iter(xpto)

for k in range(1,6):
       print(next(xptoIter))
```

No resultado produzido pela execução do código acima, a quinta linha contém exatamente:

a) [0, 1, 1, 2, 2, 3]
b) [0, 1, 1, 2, 3, 5]
c) [0, 1, 2, 3, 4, 5]
d) [0, 1, 3, 5, 7, 9]
e) [1, 2, 3, 4, 5, 6]

32. (FGV – IMBEL– 2021) Analise o código Python a seguir.

```
x = [1,2,3,4,5]
print (x[::-1])
```

Assinale a opção que indica a saída produzida pela execução desse código.

a) [1,2,3,4,5]
b) 1
c) [5,1]
d) 5
e) [5,4,3,2,1]

33. (FGV – IMBEL – 2021) Analise o código Python a seguir.

```
x = [1,2,3,4,5]
print (x[-1])
```

Assinale a opção que indica a saída produzida pela execução desse código.

a) [1,2,3,4,5]
b) 1
c) [5,1]
d) 5
e) [5,4,3,2,1]

34. (FGV – TCE-AM– 2021) Considere o código Python, versão 2.7.1, na qual o comando print não requer parênteses.

```
def teste(n):
    for k in range(1, n+1):
        yield(k)
for x in teste(10):
    print x
```

A execução desse código:

a) não tem efeito, pois nenhum comando print é acionado;
b) provoca a exibição do número 10 na saída;
c) provoca a exibição dos números de 1 até 10 na saída;
d) provoca um erro de compilação;
e) provoca um erro de execução

35. (FGV – TJ RO– 2021) Analise o código Python 2.7 a seguir.

```
def xpto (n1, n2):
    while n1 != n2:
        if (n1 < n2):
            n2 = n2 - n1
        else:
            n1 = n1 - n2
    return n1
print xpto(50,5)
```

O valor exibido pelo comando print é:

a) 0
b) 1
c) 5
d) 10
e) 50

36. (FGV – BANESTES– 2021) Considere o código Python 2.7 a seguir:

```
def ABC(L, n):
   while True:
     if len(L) >= n:
      return L
     else:
      L.append(len(L) ** 2)
print ABC([20],10)
```

O resultado da execução desse código é:

a) [1, 4, 9, 16, 25, 36, 49, 64]
b) [1, 4, 9, 16, 25, 36, 49, 64, 81]
c) [20, 1, 4, 9, 16, 25, 36, 49, 64]
d) [20, 1, 4, 9, 16, 25, 36, 49, 64, 81]
e) [20, 4, 9, 16, 25, 36, 49, 64, 81]

37. (FGV – FunSaúde CE– 2021) Observe o código Python v2.7.

```
def F (a, b):
  while a != b:
    if a > b:
      a = a - b
    elif b > a:
      b -= a
  return a
```

Assinale o valor retornado para F (48,36).

a) 1
b) 12
c) 24
d) 36
e) 48

38. (FGV – BANESTES– 2021) Considere o código Python a seguir.

```
def F(a, b, c):
 for k in range(a,b):
 print k ** c
```

Dado que uma execução da função F exibiu os números
16, 9, 4, 1, 0, 1,

é Correta afirmar que os valores dos parâmetros a, b, c empregados foram, respectivamente:

a) -4, 1, 2;
b) -4, 2, 2;
c) -4, 0, 4;
d) 4, -1, 1;
e) 4, 2, 2.

39. (FGV – BANESTES– 2021) Considere o código Python 2.7 a seguir.

```
L=[6,5,4,3,2,1]
for k in range(-3,3):
 print L[k]
```

A execução desse código exibe os números:

a) 1 1 1 6 5 4;
b) 1 2 3 4 5 6;
c) 3 2 1 6 5 4;
d) 6 5 4 3 2 1;
e) 6 5 4 6 5 4.

40. (FGV – IMBEL– 2021) Analise o script Python 3.8 exibido a seguir.

```
L=["A","E","I","O","U"]
for k in range(-1, -5, -1):
 print (L[k+1])
```

Assinale a saída produzida pela execução desse código.

a) A E I O U
b) A E I U
c) A U O I
d) U A E I
e) U O I E A

41. (FGV – IMBEL– 2021) Analise o script Python 3.8 exibido a seguir.

```
L=["A","E","I","O","U"]
for k in range(0,len(L)):
 print (L[4-k])
```

Assinale a opção que indica a saída produzida pela execução desse código.

a) A E I O U
b) A E I O
c) E I O U
d) U O I E
e) U O I E A

42. (FGV – DPE RJ– 2019) Analise o código Python 2.7 a seguir.

```
frutas = ["banana","laranja","manga","uva"]
for k in range(-1,-4,-2):
    print frutas[k]
```

O conjunto de palavras exibidas pela execução desse código, na ordem, é:

a) banana;
b) laranja, manga;
c) uva, laranja;
d) banana, laranja, manga;
e) banana, laranja, manga;

43. (FGV – MPE AL– 2018) Analise o código Python 2.7 a seguir.

```
L1=[]
L2=[1,2,3,4]
for k in range(3, -4,-1):
 L1.append(L2[k])
for x in L:
 print x
```

Esse programa causa

a) erro de sintaxe.
b) erro de execução.
c)a exibição dos valores 4,3,2,1,4,3,2 nessa ordem.
d) a exibição do valor 4, somente.
e) a exibição dos valores 4,3,2,1 nessa ordem.

44. (FGV - 2015 – CM/Caruaru - Analista Legislativo - Informática) Analise o código Python a seguir.

```
L1=[10,20,30]
L2=[40,50]
L1.append(L2)
print L1
```

Assinale a opção que descreve corretamente o que acontece quando esse programa é executado no Python 2.7:

a) Produz uma mensagem de erro, porque tenta executar uma operação inválida.
b) Exibe "[10, 20, 30, [40, 50]]".
c) Exibe "[10, 20, 30, 40, 50]".
d) Exibe "[10, 20, 30], [40, 50]".
e) Exibe "[]".

45. (FGV - 2015 – TJ/BA - Analista Judiciário) Analise o trecho de programa Python, na versão 2.7, apresentado a seguir.

```
L=[1,2,3,4,5,6,7,8]
print L[::-1]
```

Ao ser executado, o resultado exibido é:

a) [1, 2, 3, 4, 5, 6, 7, 8]
b) [8]
c) [ ]
d) [8, 7, 6, 5, 4, 3, 2, 1]
e) [1]

46. (FGV – 2014 – MPE/AL) Analise o código Python 2.7 a seguir.

```
L = [10, 12, 14, 16]
for k in range(4, -5, -1):
print L[k]
```

Esse programa causa:

a) erro de sintaxe.
b) erro de execução.
c) a exibição de 4 valores, 16,14,12,10, nessa ordem.
d) a exibição de 8 valores, 16,14,12,10,16,14,12,10, nessa ordem.
e) a exibição do valor 16, somente.

Questões FCC

47. (FCC – PGE AM– 2022) Em um programa escrito em Python, uma série de dados foram inseridos no array cargos, por meio da instrução abaixo.

cargos = ["Advogado","Promotor", "Procurador", "Juiz", "Desembargador", "Ministro"];

Para colocar estes dados em ordem alfabética decrescente em um novo array chamado cargos_ordenados utiliza-se a instrução:

a) for cargos_ordenados in cargos reverse True;
b) cargos_ordenados = sorted(cargos,reverse=True);
c) while cargos: cargos_ordenados = cargos- -;
d) cargos_ordenados = descending(cargos);
e) cargos_ordenados = ordered(cargos, descending=True);

48. (FCC – TRF 4 – 2019) Considerando que em um programa Python em condições ideais há um array criado pelo comando nomes = ["Maria", "Pedro", "João"], para exibir os valores contidos nesse array utiliza-se

a) for x in nomes: out.print(x)
b) while x in nomes: print(x)
c) foreach x in nomes: print(x)
d) foreach x in nomes: system.println(x)
e) for x in nomes: print(x)

49. (FCC – TRF 4– 2019) Considere o código Python abaixo.

```
def oper(l, item):
 pos = 0
 x = False
 ___l___
 if l[pos] == item:
 x = True
 else:
 pos = pos+1
 return x
v = [1, 2, 32, 8, 17, 19, 42, 13, 0]
print(oper(v, 8))
```

Para que o código exiba na tela o valor True se o item buscado no vetor por meio da função oper for encontrado, a lacuna I deve ser corretamente preenchida por

a) while pos < l and not x:
b) while pos < len(l) && x:
c) while pos < len(l) and not found(x):
d) while (pos < len(l) && not(x)):
e) while pos < len(l) and not x:

50. (FCC – SANASA– 2019) Uma Analista de TI está desenvolvendo um scanner de rede em Python e, para importar um recurso referente para manipulação de pacotes de rede, utilizou no programa a linha

a) import pyTTS *
b) from Tkinter import *
c) import aiml *
d) from scapy.all import *
e) from numpy import *

51. (FCC – TJ MA– 2019) Considere o programa Python abaixo:

```
numero1 = int(input('Informe o número de Processos: '))
numero2 = int(input('Informe o número de Juízes: '))
..
I
..
 resultado = numero1 / numero2
 print("Há ",resultado, " processos a serem julgados por cada Juiz")
..
II
..
 print("Não é possível divisão por zero")
```

Para tratar a exceção que será lançada se o valor contido na variável numero2 for zero, as lacunas I e II deverão ser corretamente preenchidas por:

a) try: e catch ArithymeticException:
b) throw e catch (ZeroDivisionException $e)
c) try e catch(ArithmeticException ex)
d) throw: e catch(err)
e) try: e except ZeroDivisionError:

52. (FCC – TJ MA– 2019) Considere o programa Python abaixo:

```
..
I..
 for x in pro:
 print(x)
processos = ["0456789908", "0875643087", "0897645109"]
exibir_processos (processos)
```

Para que o programa seja executado corretamente, em condições ideais, a indicação I deve ser substituída por:

a) private exibir_processos(pro):
b) public exibir_processos(pro):
c) function exibir_processos(pro):
d) definition exibir_processos(pro):
e) def exibir_processos(pro):

53. (FCC – MPE PE– 2018) Considere o fragmento de código Python abaixo.

```
class Cliente:
I
.....
self.nome = nome
self.renda = renda
p1 = Cliente("Maria", 5678.98)
print(p1.nome)
print(p1.renda)
```

Para que o código seja compilado e executado corretamente, a lacuna I deverá ser preenchida com

a) __init__(self, nome, renda):
b) function __init(self, nome, renda):
c) def __construct(self, nome, renda):
d) def __init__(self, nome, renda):
e) Cliente(self, nome, renda):

54. (FCC – TRE SP– 2017) Considere o programa Python, abaixo.

```
import ..I.. as b
import matplotlib.pyplot as a
x = b.linspace(0, 3, 20)
```

```
y = b.linspace(0, 9, 20)
a.plot(x, y)
a.plot(x, y, 'o')
a.show()
```

A lacuna I deve ser preenchida corretamente com

a) numpy
b) matrix
c) mathlab
d) numberplot
e) array

55. (FCC – PRODATER– 2016) Em Python existe um conjunto de métodos disponíveis para se trabalhar com objetos do tipo lista. Considere o trecho de programa abaixo que faz uso desses métodos.

```
a = [99.15, 323, 323, 2, 12.5]
a.insert(2, -5)
a.append(323)
a.index(323)
a.remove(323)
a.reverse()
```

Ao executar este trecho de programa, o conteúdo final da lista a será:

a) [99.15, −5, 323, 2, 12.5, 323]
b) [−5, 2, 12.5, 99.15, 323, 323]
c) [323, 12.5, 2, 323, −5, 99.15]
d) [99.15, −5, 323, 323, 12.5, 2]
e) [323, 323, 99.15, 12.5, 2, −5]

56. (FCC – SEMF Teresina– 2016) Considere o programa abaixo, criado na linguagem Python.

```
a = ['ARSETE', 'PRODATER', 'SEMEST', 'STRANS', 'SEMAE']
...
I...
print i, a[i]
```

Considere a saída gerada pelo programa:

o ARSETE

1 PRODATER
2 SEMEST
3 STRANS
4 SEMAE

Na lacuna I deve estar o comando:

a) for i in len(a):
b) foreach a as &value(a)):
c) while i<range(len(a)):
d) for i in range(len(a)):
e) for i in range(a):

57. (FCC – PGM Teresina– 2016) Considere o código-fonte abaixo, criado na linguagem Python.

```
def dados(n):
resultado = []
a, b = 0, 1
while a < n:
resultado.append(a)
a, b = b, a+b
return resultado
op = dados(100)
print op
```

O valor final contido na posição de índice 6 de op é

a) 1
b) 8
c) 5
d) 6
e) 13

58. (FCC – CNMP– 2015) Considere os fragmentos de programas Phyton a seguir:

Fragmento 1:

```
for n in range(2, 10):
for x in range(2, n):
 if n % x == 0:
 print n, '=', x, '*', n/x
```

```
 break
else:
 print n, 'é um número primo'
```

Fragmento 2:

```
a = ['Casa', 'Mala', 'Prova']
for x in a:
 print x, len(x)
```

É Correta afirmar que

a) o Fragmento 1 está in Correta, pois laços não podem ter uma cláusula else.
b) no Fragmento 2, a instrução for está incorreta, pois ela não pode iterar sobre a.
c) o Fragmento 1 está in Correta, pois não é possível iterar sobre sequências numéricas utilizando a função range.
d) no Fragmento 1 é verificado se o quociente da divisão de n por x corresponde a 0.
e) os dois fragmentos de código estão Corretas.

59. **(FCC – TRT/MG - Técnico Judiciário - Tecnologia Da Informação)** Considere o código fonte Python abaixo.

```
def calcular(n):
    resultado = []
    a, b = 0, 1
    while a < n:
            I
          .....
    return resultado

res = calcular(100)
print res
```

Para que seja exibido [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89] a lacuna "I" precisa ser preenchida corretamente com:

a) resultado.add(a)
    a, b = b, a+b
b) resultado.insert(a)
    a, b = b, a+b
c) resultado.append(a)
    a, b = b, a+b
d) resultado.add(a)
    a, b = a, a+b
e) resultado.append(a)

```
a, b = a+b, b
```

60.(FCC – 2012 – MPE/PE – Analista de Sistemas) Em Python, os métodos de lista permitem utilizar listas como pilhas, onde o item adicionado por último é o primeiro a ser recuperado. Para adicionar um item ao topo da pilha, e para recuperar um item do topo da pilha utilizam-se, respectivamente os métodos:

a) append() e pop().
b) insert() e top().
c) addTop() e pop().
d) add() e get().
e) addItem() e top().

61.(FCC – 2012 – TRE/CE – Analista de Sistemas) Sobre Python é Correta afirmar:

a) é uma linguagem compilada, ou seja, o código-fonte de um programa é lido pelo compilador, que cria um arquivo binário, executável diretamente pelo hardware.

b) É uma linguagem fortemente tipada, ou seja, é preciso declarar variáveis e seus tipos.

c) Suporta funcionalidades comuns na orientação a objetos: herança, herança múltipla, polimorfismo, reflexão e introspecção.

d) Uma lista em Python é um conjunto de valores acessados por um índice numérico, inteiro, começando em 1. Assim como em outras linguagens, a lista pode armazenar apenas valores de um mesmo tipo.

e) Uma String Python é uma sequência imutável, alocada estaticamente, com restrição de tamanho.

Questões Comentadas – Diversas Bancas

62. (IESES - 2015 - IFC-SC - Programação Web e Dispositivos Móveis) Sobre listas em Python 3.1.5:

a) list.remove(a) remove o primeiro item da lista cujo valor é a.
b) list.pop(a) adiciona um item de valor a ao início da lista.
c) list.append(a) adiciona um item à lista cujo índice será a.
d) list.index(a) retorna o valor do item cujo índice é a.

63. (IESES - 2015 - IFC-SC - Programação Web e Dispositivos Móveis) O conjunto Correta de palavras reservadas para a construção de uma estrutura de controle em Python 3.4.3 é:

a) "if", "elsif" e "else".
b) "if", "else if" e "else".
c) Somente "if"; o restante da estrutura de controle ("senão se" e "senão") é realizado simplesmente com indentação.
d) "if", "elif" e "else".

64. (UERJ - UERJ - Analista De Sistemas - Grid) Considere o trecho do programa Python abaixo:

```
def dobra (y):
        x = y + y
        return x

x = 5
dobra(x)
dobra(x)
print x
```

O valor impresso ao executarmos o programa é:

a) 5
b) 10
c) 15
d) 25

65. (UERJ - UERJ - Analista De Sistemas) A linguagem Python possui a seguinte característica:

a) é uma linguagem compilada.
b) exige declaração de código.
c) a tupla é um tipo mutável.
d) é orientada a objetos.

66. **(CETAP – 2010 – AL/RR – Analista de Sistemas)** Sobre a linguagem de programação PYTHON, marque a alternativa INCORRETA.

a) Python suporta a maioria das técnicas da programação orientada a objetos.

b) Python suporta e faz uso constante de tratamento de exceções como uma forma de testar condições de erro e outros eventos inesperados no programa.

c) As funções são definidas em Phyton utilizando a palavra chave def.

d) A separação de blocos de código em Phyton é feita utilizando a indentação de código.

e) O operador lógico de conjunção ("e", como em a e b) é &&.

67. **(CESGRANRIO – 2004 – SECAD/TO – Analista de Sistemas)** Um programador de Python recebeu a tarefa de criar uma função chamada calcular que recebe dois parâmetros. Para executar sua atividade, ele deve utilizar a expressão:

a) def calcular (a,b):
b) function calcular (a,b):
c) import calcular (a,b):
d) procedure calcular (a,b):
e) sub calcular (a,b):

68. **(UNIRIO – 2014 – UNIRIO)** Sobre o comando range para construção de listas na linguagem Python, é CORRETA afirmar que:

a) range(4,6) gera a lista [4,5].
b) range(5) gera a lista [1,2,3,4,5].
c) range(4,6) gera a lista [4,5,6,7,8,9].
d) range(5,1) gera a lista [5].
e) range(5,1,-2) gera a lista [4,5].

69. **(QUADRIX – 2018 – COREN/RS)** No que se refere à linguagem de programação Python, assinale a alternativa correta.

a) A Python é uma linguagem de alto nível e robusta. Ela possui seu próprio framework e é incompatível com frameworks de terceiros.

b) A Python utiliza a duck typing (tipagem dinâmica), que nada mais é do que definir um tipo para a variável, com as operações que podem ser aplicadas, antes mesmo de ela ter sido criada, com base em conhecimento prévio do programa. Esta tarefa é executada pelo interpretador.

c) O caractere "/" marca o início de comentário. Qualquer texto depois do "/"será ignorado até o fim da linha.

d) A Python permite que os conteúdos das variáveis sejam sempre alterados, não existindo, dessa forma, tipos imutáveis.

e) Pode ser utilizada como linguagem principal no desenvolvimento de sistemas e também pode ser utilizada como linguagem script em vários softwares.

70. **(IFPI – 2012 – IFPI)** Com relação à linguagem de programação Python, é INCORRETA a afirmação:

a) Disponibiliza o conceito de herança múltipla, ou seja, uma classe pode ter mais de uma classe pai.

b) A sintaxe Python permite o uso de somente uma instrução por linha de código. Para que seja possível colocar mais de uma, é necessário separá-las com um ";", entretanto, essa prática é desaconselhada, visto que sua utilização prejudica a legibilidade do código.

c) Os comentários são iniciados com um caractere "#". Para fazer comentários com múltiplas linhas, utilizam-se os caracteres "#", para o início, e "#*" para o final .

d) Diferentemente de outras linguagens de programação que utilizam elementos léxicos para definir os blocos de código, a própria endentação do código é usada, em Python, para essa função.

e) Em Python, os nomes de variáveis, funções, módulos e classes são conhecidos como identificadores. O identificador _quantidade é Correta em Python.

GABARITO

1. Letra D
2. Letra A
3. Letra D
4. Errado
5. Errado
6. Errado
7. Correta
8. Correta
9. Correta
10. Letra B
11. Letra C
12. Errado
13. Errado
14. Correta
15. Correta
16. Correta
17. Errado
18. Correta
19. Errado
20. Letra A
21. Letra C
22. Letra D
23. Letra E
24. Letra B
25. Letra A
26. Letra A
27. Letra A
28. Letra D
29. Letra B
30. Letra E
31. Letra B
32. Letra E
33. Letra D
34. Letra C
35. Letra C
36. Letra D
37. Letra B
38. Letra B
39. Letra C
40. Letra C
41. Letra E
42. Letra C
43. Letra B
44. Letra B
45. Letra D
46. Letra A
47. Letra B
48. Letra E
49. Letra E
50. Letra D
51. Letra E
52. Letra E
53. Letra D
54. Letra A
55. Letra C
56. Letra D
57. Letra B
58. Letra E
59. Letra C
60. Letra A
61. Letra C
62. Letra A
63. Letra D
64. Letra A
65. Letra D
66. Letra E
67. Letra A
68. Letra A
69. LetraE
70. Letra C

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.