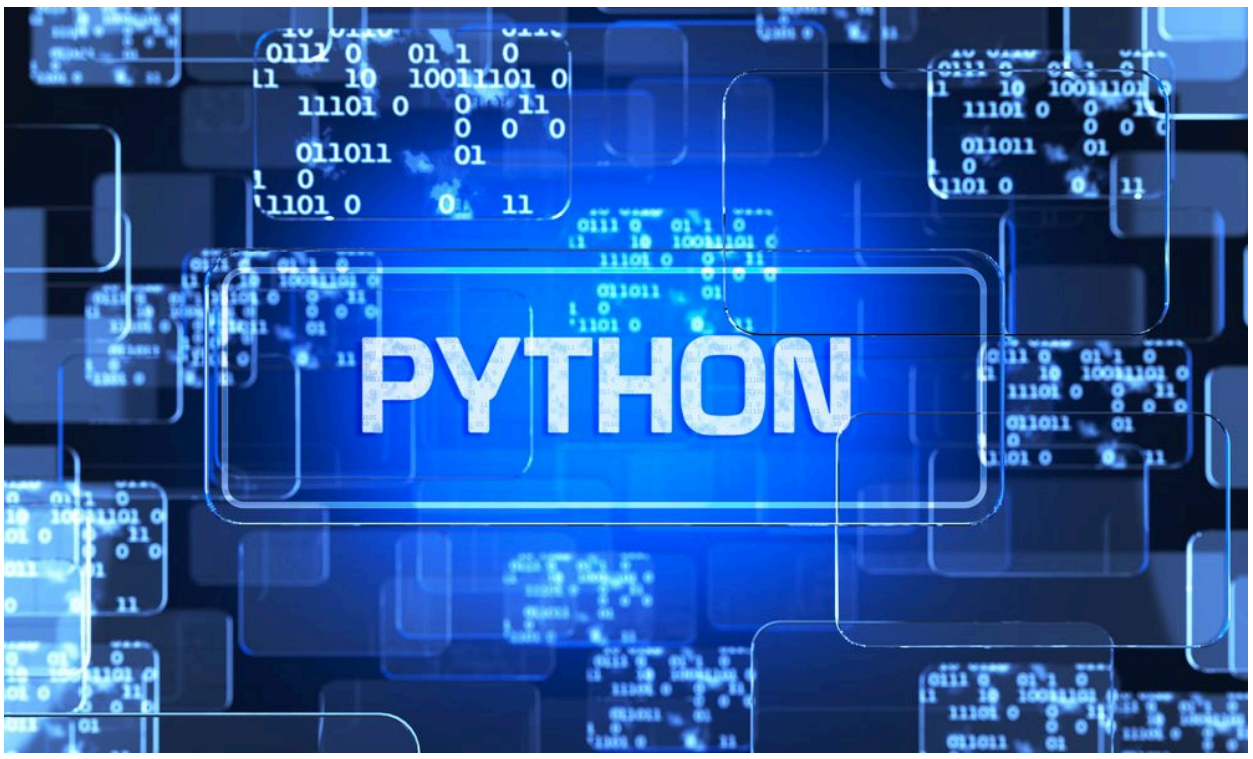
NÃO PODE FALTAR

ESTRUTURAS LÓGICAS, CONDICIONAIS E DE REPETIÇÃO EM PYTHON

Vanessa Cadan Scheffer

AS ESTRUTURAS DE COMANDO EM PYTHON

Vamos conhecer alguns trechos de código e sua lógica de execução.



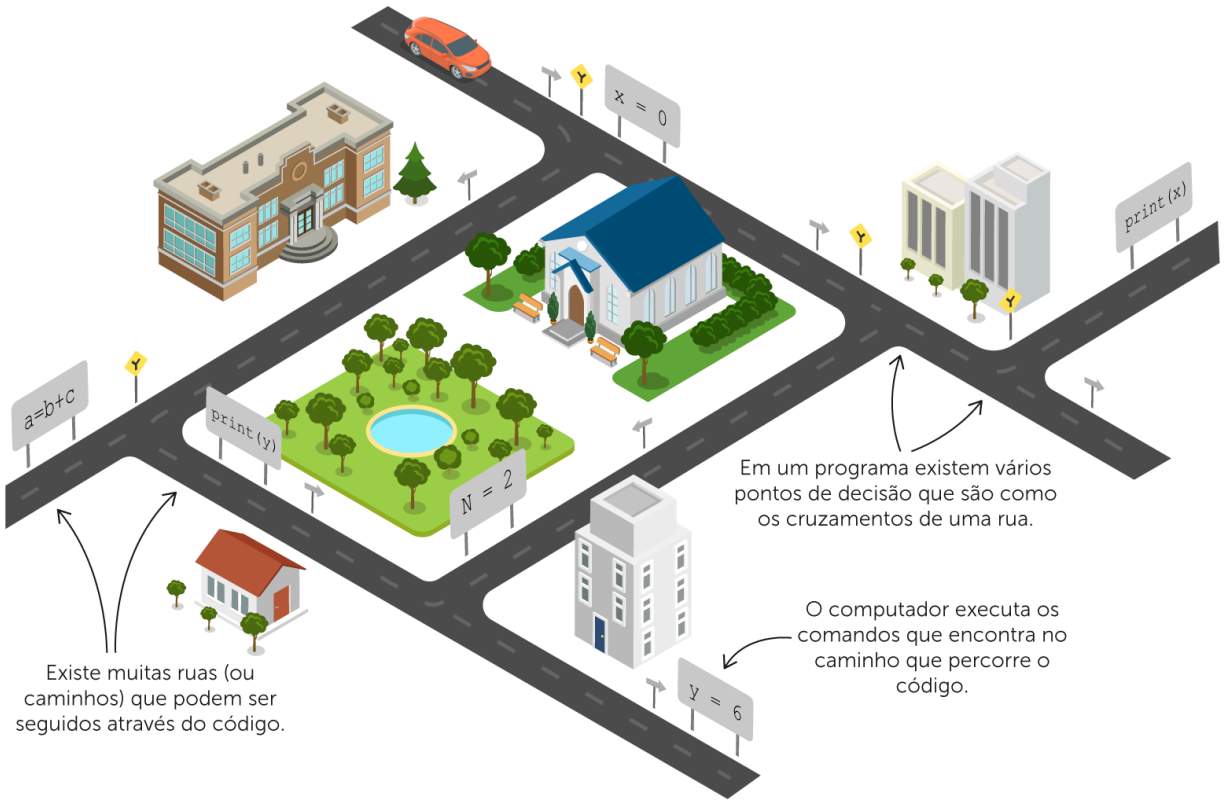
Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Criamos algoritmos para resolver problemas, o que implica tomar decisões. No universo da programação, a técnica que nos permite tomar decisões é chamada de estrutura condicional (MANZANO; OLIVEIRA, 2019). A Figura 1.5 ilustra uma cidade e suas vias. Ao dirigir em uma cidade, você precisa decidir quais caminhos (ruas) seguir com o objetivo de chegar a um destino; por exemplo, em um cruzamento, pode-se virar à direita ou à esquerda e/ou seguir em frente. Um programa funciona de forma similar: deve-se decidir qual caminho será executado em um código. Em geral, em um programa você tem opções de caminhos ou lista de comandos que nada mais são que trechos de códigos que podem ser executados, devendo-se tomar decisões sobre qual trecho de código será executado em um determinado momento.

Figura 1.5 | Fluxo de tomada de decisões de um algoritmo para a escolha de um caminho em uma cidade



Fonte: adaptada de Griffiths e Barry (2009, p. 13).

Para tomarmos decisões, precisamos dos operadores relacionais (Quadro 1.2), que já fazem parte do nosso vocabulário, desde as aulas de matemática do colégio. Vamos usá-los para fazer comparações entre objetos (lembrando que em Python tudo é objeto, então uma variável também é um objeto).

Quadro 1.2 - Operadores relacionais

Operação	Significado
a < b	O valor de a é menor que b?
a <= b	O valor de a é menor OU igual que b?
a > b	O valor de a é maior que b?
a >= b	O valor de a é maior OU igual que b?
a == b	O valor de a é igual ao de b?
a != b	O valor de a é diferente do valor de b?
a is b	O valor de a é idêntico ao valor de b?
a is not b	O valor de a não é idêntico ao valor de b?

Fonte: adaptado de https://docs.python.org/3/library/stdtypes.html.

In [1]:
```
a = 5
b = 10

if a < b:
    print("a é menor do que b")
    r = a + b
    print(r)
```

```
a é menor do que b
15
```

Como você pode observar, a construção do if (se) deve ser feita com dois pontos ao final da condição, e o bloco de ações que devem ser feitas. Caso o trecho seja verdadeiro, deve possuir uma indentação específica de uma tabulação ou 4 espaços em branco. Repare que não precisamos escrever if a < b == True.

O código apresentado na entrada 1(In [1]) refere-se a uma estrutura condicional simples, pois só existem ações caso a condição seja verdadeira. Veja como construir uma estrutura composta:

In [2]:
```
a = 10
b = 5

if a < b:
    print("a é menor do que b")
    r = a + b
    print(r)
else:
    print("a é maior do que b")
    r = a - b
    print(r)
```

```
a é maior do que b
5
```

O código apresentado na entrada 2(In [2]) refere-se a uma estrutura condicional composta, pois existem ações tanto para a condição verdadeira quanto para a falsa. Veja que o else (senão) está no mesmo nível de identação do if, e os blocos de ações com uma tabulação de indentação.

Além das estruturas simples e compostas, existem as estruturas encadeadas. Em diversas linguagens como C, Java, C#, C++, etc, essa estrutura é construída como o comando switch..case. Em Python, não existe o comando switch

(https://docs.python.org/3/tutorial/controlflow.html). Para construir uma estrutura

In [3]:
```
codigo_compra = 5111

if codigo_compra == 5222:
    print("Compra à vista.")
elif codigo_compra == 5333:
    print("Compra à prazo no boleto.")
elif codigo_compra == 5444:
    print("Compra à prazo no cartão.")
else:
    print("Código não cadastrado")
```

```
Código não cadastrado
```

O código apresentado na entrada 3 (In [3]) mostra a construção de uma estrutura condicional encadeada. Veja que começamos pelo teste com if, depois testamos várias alternativas com o elif e, por fim, colocamos uma mensagem, caso nenhuma opção seja verdadeira, no else.

ESTRUTURAS LÓGICAS EM PYTHON: AND, OR, NOT

Além dos operadores relacionais, podemos usar os operadores booleanos para construir estruturas de decisões mais complexas.

- Operador booleano and: Esse operador faz a operação lógica E, ou seja, dada a expressão (a and b), o resultado será True, somente quando os dois argumentos forem verdadeiros.
- Operador booleano or: Esse operador faz a operação lógica OU, ou seja, dada a expressão (a or b), o resultado será True, quando pelo menos um dos argumentos for verdadeiro.
- Operador booleano not: Esse operador faz a operação lógica NOT, ou seja, dada a expressão (not a), ele irá inverter o valor do argumento. Portanto, se o argumento for verdadeiro, a operação o transformará em falso e vice-versa.

Observe o exemplo a seguir, que cria uma estrutura condicional usando os operadores booleanos. Um aluno só pode ser aprovado caso ele tenha menos de 5 faltas e média final superior a 7.

In [4]:
```
qtde_faltas = int(input("Digite a quantidade de faltas: "))
media_final = float(input("Digite a média final: "))

if qtde_faltas <= 5 and media_final >= 7:
    print("Aluno aprovado!")
else:
    print("Aluno reprovado!")
```

```
Digite a quantidade de faltas: 2
Digite a média final: 7
Aluno aprovado!
```

No exemplo da entrada 4 (In [4]), fizemos a conversão do tipo string para numérico, encadeando funções. Como já sabemos, primeiro são resolvidos os parênteses mais internos, ou seja, na linha 1, será feita primeiro a função input(), em seguida, a entrada digitada será convertida para inteira com a função int(). A mesma lógica se

aplica na linha 2, na qual a entrada é convertida para ponto flutuante. Na linha 4 fizemos o teste condicional, primeiro são resolvidos os operadores relacionais, em seguida, os lógicos.

Obs: not tem uma prioridade mais baixa que os operadores relacionais. Portanto, not a == b é interpretado como: not (a == b) e a == not b gera um erro de sintaxe.

Obs2: Assim como as operações matemáticas possuem ordem de precedência, as operações booleanas também têm. Essa prioridade obedece à seguinte ordem: not

In [5]:
```
A = 15
B = 9
C = 9

print(B == C or A < B and A < C)
print((B == C or A < B) and A < C )
```

```
True
False
```

Na entrada 5 (In [5]), temos os seguintes casos:

Linha 5: True or False and False. Nesse caso será feito primeiro o and, que resulta em False, mas em seguida é feito o or, que então resulta em verdadeiro.

Linha 6: (True or False) and False. Nesse caso será feito primeiro o or, que resulta em True, mas em seguida é feito o and, que então resulta em falso, pois ambos teriam que ser verdadeiro para o and ser True.

Utilize o emulador a seguir para testar os códigos. Crie novas variáveis e condições para treinar.

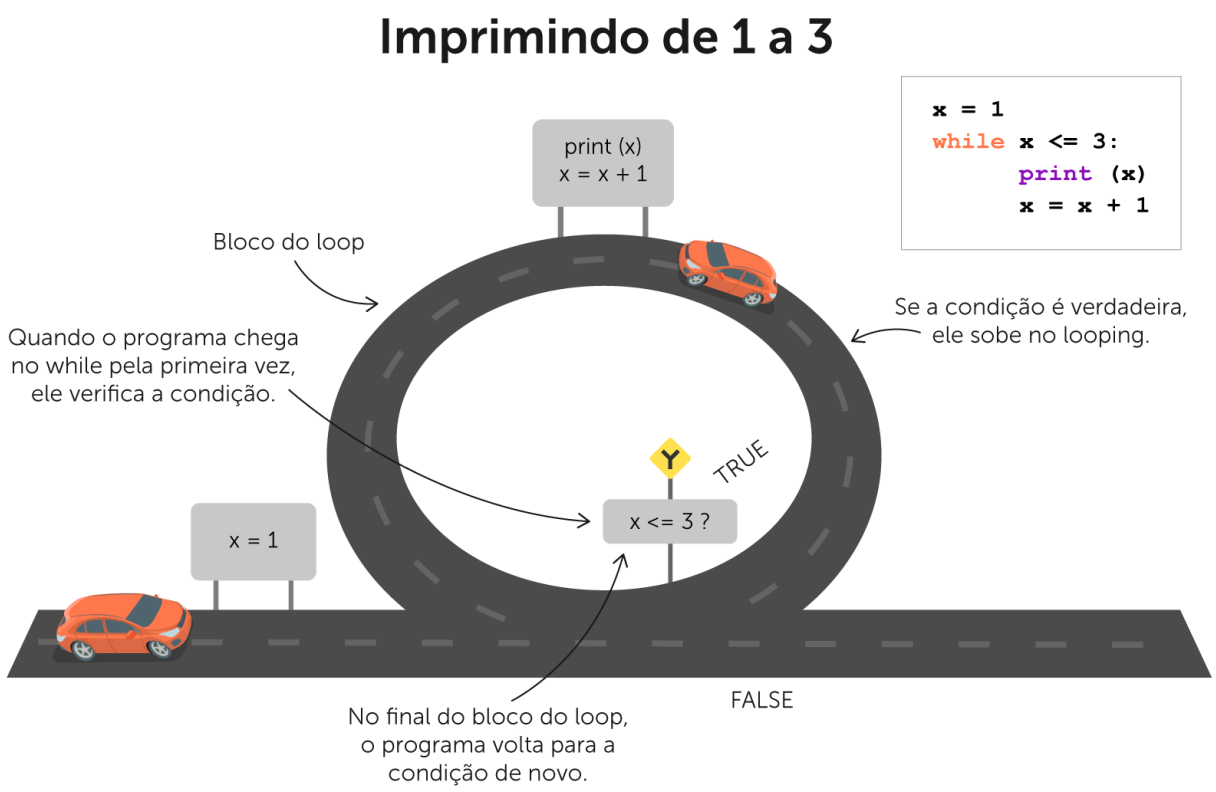
ESTRUTURAS DE REPETIÇÃO EM PYTHON: WHILE E FOR

É comum, durante a implementação de uma solução, criarmos estruturas que precisam executar várias vezes o mesmo trecho de código. Tal estrutura é chamada de estrutura de repetição e, em Python, pode ser feita pelo comando while (enquanto) ou pelo comando for (para). Antes de apresentarmos a implementação, observe a Figura 1.7. Temos a ilustração de uma estrutura de repetição por meio de uma montanha russa. Enquanto uma condição for verdadeira, ou seja, enquanto x for menor ou igual a 3, o condutor continuará no loop; quando for falsa, ou seja, quando x for maior que 3, então ele sairá.

Em uma estrutura de repetição sempre haverá uma estrutura decisão, pois a repetição de um trecho de código sempre está associada a uma condição. Ou seja, um bloco de comandos será executado repetidas vezes, até que uma condição não

seja mais satisfeita.

Figura 1.7 | Fluxograma estrutura de repetição



Fonte: adaptada de Griffiths e Barry (2009, p. 28).

O comando while deve ser utilizado para construir e controlar a estrutura decisão, sempre que o número de repetições não seja conhecido. Por exemplo, quando queremos solicitar e testar se o número digitado pelo usuário é par ou ímpar. Quando ele digitar zero, o programa se encerra. Veja, não sabemos quando o usuário irá digitar, portanto somente o while é capaz de solucionar esse problema. Veja a implementação desse problema a seguir.

In [6]:
```
numero = 1
while numero != 0:
    numero = int(input("Digite um número: "))
    if numero % 2 == 0:
        print("Número par!")
    else:
        print("Número ímpar!")
```

```
Digite um número: 4
Número par!
Digite um número: 7
Número ímpar!
Digite um número: 0
Número par!
```

A seguir, você pode testar o código, verificando o passo a passo de sua execução. Ao clicar em "Next", você irá para a próxima linha a ser executada do "while" para cada um dos casos. Ao clicar em "Prev", você retorna a linha executada anteriormente. Observe que a seta verde na linha de código representa a linha em execução, e a seta vermelha representa a próxima linha a ser executada. Digite um número, submeta, clique em "Next" e observe o fluxo de execução e as saídas do código.

Veja a construção do comando while na entrada 6 (In [6]). Na linha 1, criamos uma variável chamada número com valor 1. Na linha 2 criamos a estrutura de repetição, veja que o comando while possui uma condição: o número tem que ser diferente de zero para o bloco executar. Todo o bloco com a identação de uma tabulação (4 espaços) faz parte da estrutura de repetição. Lembre: todos os blocos de comandos em Python são controlados pela indentação.

Na prática é comum utilizarmos esse tipo de estrutura de repetição, com while, para deixarmos serviços executando em servidores. Por exemplo, o serviço deve ficar executando enquanto a hora do sistema for diferente de 20 horas.

Outro comando muito utilizado para construir as estruturas de repetição é o for. A instrução for em Python difere um pouco do que ocorre em outras linguagens, como C ou Java, com as quais você pode estar acostumado (https://docs.python.org/3/tutorial/controlflow.html). Em vez de sempre dar ao usuário a capacidade de definir a etapa de iteração e a condição de parada (como C), a instrução Python for itera sobre os itens de qualquer sequência, por exemplo, iterar sobre os caracteres de uma palavra, pois uma palavra é um tipo de sequência. Observe um exemplo a seguir.

In [7]:
```
nome = "Guido"
for c in nome:
    print(c)
```

```
G
u
i
d
o
```

Na entrada 7 (In [7]) construímos nossa primeira estrutura de repetição com for. Vamos analisar a sintaxe da estrutura. Usamos o comando "for" seguido da variável de controle "c", na sequência o comando "in", que traduzindo significa "em", por fim, a sequência sobre a qual a estrutura deve iterar. Os dois pontos marcam o início do bloco que deve ser repetido.

Junto com o comando for, podemos usar a função enumerate() para retornar à posição de cada item, dentro da sequência. Considerando o exemplo dado, no qual atribuímos a variável "nome" o valor de "Guido", "G" ocupa a posição 0 na sequência, "u" ocupa a posição 1, "i" a posição 2, e assim por diante. A função enumerate() recebe como parâmetro a sequência e retorna a posição. Para que possamos capturar tanto a posição quanto o valor, vamos precisar usar duas variáveis de controle. Observe o código a seguir, veja que a variável "i" é usada para capturar a posição e a variável "c" cada caracter da palavra.

In [8]:
```
nome = "Guido"
for i, c in enumerate(nome):
    print(f"Posição = {i}, valor = {c}")
```

```
Posição = 0, valor = G
Posição = 1, valor = u
Posição = 2, valor = i
Posição = 3, valor = d
Posição = 4, valor = o
```

CONTROLE DE REPETIÇÃO COM RANGE, BREAK E CONTINUE

Como já dissemos, o comando for em Python requer uma sequência para que ocorra a iteração. Mas e se precisarmos que a iteração ocorra em uma sequência numérica, por exemplo, de 0 a 5? Para criar uma sequência numérica de iteração em Python, podemos usar a função range(). Observe o código a seguir.

In [9]:
```
for x in range(5):
    print(x)
```

```
0
1
2
3
4
```

No comando, "x" é a variável de controle, ou seja, a cada iteração do laço, seu valor é alterado, já a função range() foi utilizada para criar um "iterable" numérico (objeto iterável) para que as repetições acontecesse. A função range() pode ser usada de três formas distintas:

1. Método 1: passando um único argumento que representa a quantidade de vezes que o laço deve repetir;
2. Método 2: passando dois argumentos, um que representa o início das repetições e outro o limite superior (NÃO INCLUÍDO) do valor da variável de controle;
3. Método 3: Passando três argumentos, um que representa o início das repetições; outro, o limite superior (NÃO INCLUÍDO) do valor da variável de controle e um que representa o incremento. Observe as três maneiras a seguir.

A execução dos três métodos pode ser visualizada a seguir. Ao clicar em "Next", você irá para a próxima linha a ser executada do "for" para cada um dos casos. Observe que a seta verde na linha de código representa a linha em execução, e a seta vermelha representa a próxima linha a ser executada.

Além de controlar as iterações com o tamanho da sequência, outra forma de influenciar no fluxo é por meio dos comandos "break" e "continue". O comando break para a execução de uma estrutura de repetição, já com o comando continue, conseguimos "pular" algumas execuções, dependendo de uma condição. Veja o exemplo:

In [10]:
```
# Exemplo de uso do break
disciplina = "Linguagem de programação"
for c in disciplina:
    if c == 'a':
        break
    else:
        print(c)
```

```
L
i
n
g
u
```

No exemplo com uso do break, perceba que foram impressas as letras L i n g u, quando chegou a primeira letra a, a estrutura de repetição foi interrompida. Agora observe o exemplo com continue.

In [11]:
```
# Exemplo de uso do continue
disciplina = "Linguagem de programação"
for c in disciplina:
    if c == 'a':
        continue
    else:
        print(c)
```

```
L
i
n
g
u
g
e
m

d
e

p
r
o
g
r
m
ç
ã
o
```

No exemplo com uso do continue, perceba que foram impressas todas as letras, exceto as vogais "a", pois toda vez que elas eram encontradas, o continue determina que se pule, mas que a repetição prossiga para o próximo valor. Utilize o emulador para executar os códigos e crie novas combinações para explorar.

EXEMPLIFICANDO

Vamos brincar um pouco com o que aprendemos. Vamos criar uma solução que procura pelas vogais "a", "e" em um texto (somente minúsculas). Toda vez que essas vogais são encontradas, devemos informar que encontramos e qual posição do texto ela está. Nosso texto será:

texto = A inserção de comentários no código do programa é uma prática normal. Em função disso, toda linguagem de programação tem alguma maneira de permitir que comentários sejam inseridos nos programas. O objetivo é adicionar descrições em partes do código, seja para documentá-lo ou para adicionar uma descrição do algoritmo implementado (BANIN, 2018, p. 45).

In [12]:
```
texto = """
A inserção de comentários no código do programa é uma prática normal.
Em função disso, toda linguagem de programação tem alguma maneira de permitir que
comentários sejam inseridos nos programas.
O objetivo é adicionar descrições em partes do código, seja para documentá-lo ou
para adicionar uma descrição do algoritmo implementado (BANIN, p. 45, 2018)."
"""
for i, c in enumerate(texto):
    if c == 'a' or c == 'e':
        print(f"Vogal '{c}' encontrada, na posição {i}")
    else:
        continue
```

```
Vogal 'e' encontrada, na posição 6
Vogal 'e' encontrada, na posição 13
Vogal 'e' encontrada, na posição 18
Vogal 'a' encontrada, na posição 45
Vogal 'a' encontrada, na posição 47
Vogal 'a' encontrada, na posição 53
Vogal 'a' encontrada, na posição 61
Vogal 'a' encontrada, na posição 67
Vogal 'a' encontrada, na posição 91
Vogal 'a' encontrada, na posição 98
Vogal 'e' encontrada, na posição 100
Vogal 'e' encontrada, na posição 104
Vogal 'a' encontrada, na posição 111
Vogal 'a' encontrada, na posição 113
Vogal 'e' encontrada, na posição 119
Vogal 'a' encontrada, na posição 122
Vogal 'a' encontrada, na posição 127
Vogal 'a' encontrada, na posição 130
Vogal 'e' encontrada, na posição 132
Vogal 'a' encontrada, na posição 135
Vogal 'e' encontrada, na posição 138
Vogal 'e' encontrada, na posição 141
Vogal 'e' encontrada, na posição 151
Vogal 'e' encontrada, na posição 156
Vogal 'e' encontrada, na posição 166
Vogal 'a' encontrada, na posição 168
Vogal 'e' encontrada, na posição 174
Vogal 'a' encontrada, na posição 190
Vogal 'a' encontrada, na posição 192
Vogal 'e' encontrada, na posição 201
Vogal 'a' encontrada, na posição 209
Vogal 'a' encontrada, na posição 216
Vogal 'e' encontrada, na posição 220
Vogal 'e' encontrada, na posição 227
Vogal 'e' encontrada, na posição 230
Vogal 'a' encontrada, na posição 234
Vogal 'e' encontrada, na posição 237
Vogal 'e' encontrada, na posição 252
Vogal 'a' encontrada, na posição 254
Vogal 'a' encontrada, na posição 257
Vogal 'a' encontrada, na posição 259
Vogal 'e' encontrada, na posição 266
Vogal 'a' encontrada, na posição 278
Vogal 'a' encontrada, na posição 280
Vogal 'a' encontrada, na posição 282
Vogal 'a' encontrada, na posição 289
Vogal 'a' encontrada, na posição 294
Vogal 'e' encontrada, na posição 297
Vogal 'a' encontrada, na posição 309
Vogal 'e' encontrada, na posição 323
Vogal 'e' encontrada, na posição 325
Vogal 'a' encontrada, na posição 328
```

Uma novidade nesse exemplo foi o uso das aspas triplas para que pudéssemos quebrar a linha do texto. As aspas triplas também podem ser usadas para criar comentários de várias linhas em Python. Que tal alterar a solução de busca por vogais e contar quantas vezes cada vogal foi encontrada. Lembre-se de que, quanto mais praticar, mais desenvolverá sua lógica de programação e fluência em uma determinada linguagem!

REFERÊNCIAS E LINKS ÚTEIS

BANIN, S. L. **Python 3 - conceitos e aplicações**: uma abordagem didática. São Paulo: Érica, 2018.

GRIFFITHS, D.; BARRY, P. **Head First Programming**: A learner's guide to programming using the Python language. [**S.l.**]: O'Reilly Media, 2009.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. de. **Algoritmos**: lógica para desenvolvimento de programação de computadores. 29. ed. São Paulo: Érica, 2019.

PYTHON SOFTWARE FOUNDATION. **More Control Flow Tools**. Disponível em: https://docs.python.org/3/tutorial/controlflow.html. Acesso em: 15 abr. 2020.

PYTHON SOFTWARE FOUNDATION. **Numeric Types**: int, float, complex. Disponível em: https://docs.python.org/3/library/stdtypes.html. Acesso em: 15 abr. 2020.

SUBSECRETARIA DE TRIBUTAÇÃO E CONTENCIOSO. **Imposto sobre a renda das pessoas físicas**. Disponível em: http://receita.economia.gov.br/acesso-rapido/tributos/irpf-imposto-de-renda-pessoa-fisica. Acesso em: 18 abr. 2020.