Estruturas condicionais de repetição e lógicas em Python

Você sabia que seu material didático é interativo e multimídia? Isso significa que você pode interagir com o conteúdo de diversas formas, a qualquer hora e lugar. Na versão impressa, porém, alguns conteúdos interativos ficam desabilitados. Por essa razão, fique atento: sempre que possível, opte pela versão digital. Bons estudos!

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. Em Python estruturas são bem definidas, como podemos observar a seguir.

Se desejar, baixe a versão em texto do objeto.

Agora que você já está familiarizado com algumas estruturas em Python, vamos aprofunda nesse tema.

Estruturas condicionais em Python: *if, elif, else*

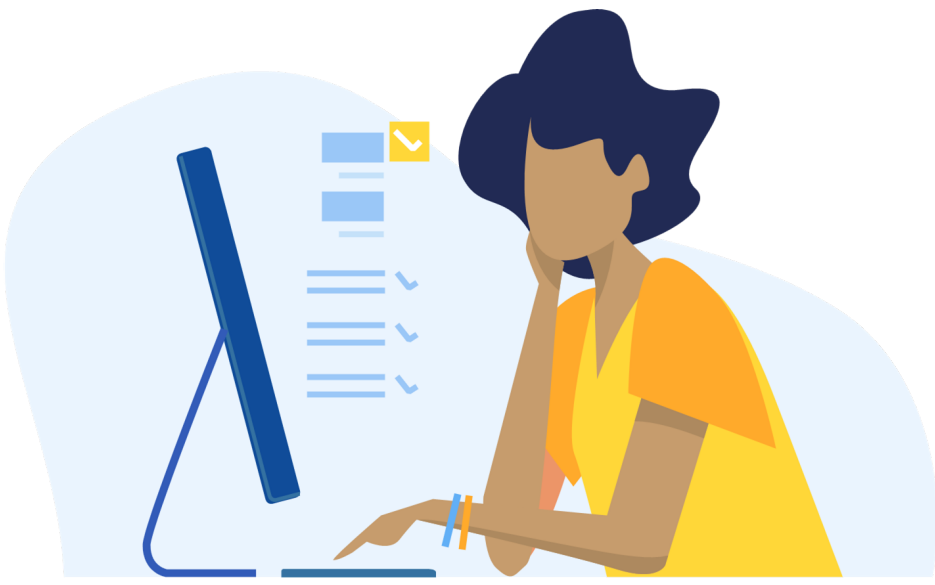
Uma estrutura condicional verifica a condição dos argumentos passados e executa um comando caso a condição seja verdadeira. Para a construção de estruturas condicionais em Python são utilizados os **comandos**:

Por padrão, o bloco de instrução que estiver abaixo da instrução *if* será executado quando a expressão contida na estrutura *if* for verdadeira.

Instrução complementar da estrutura *if*, deve ser executada quando a expressão definida for igual a falso.

Trata-se de uma abreviação do *else if* usado para fazer as condições intermediárias.

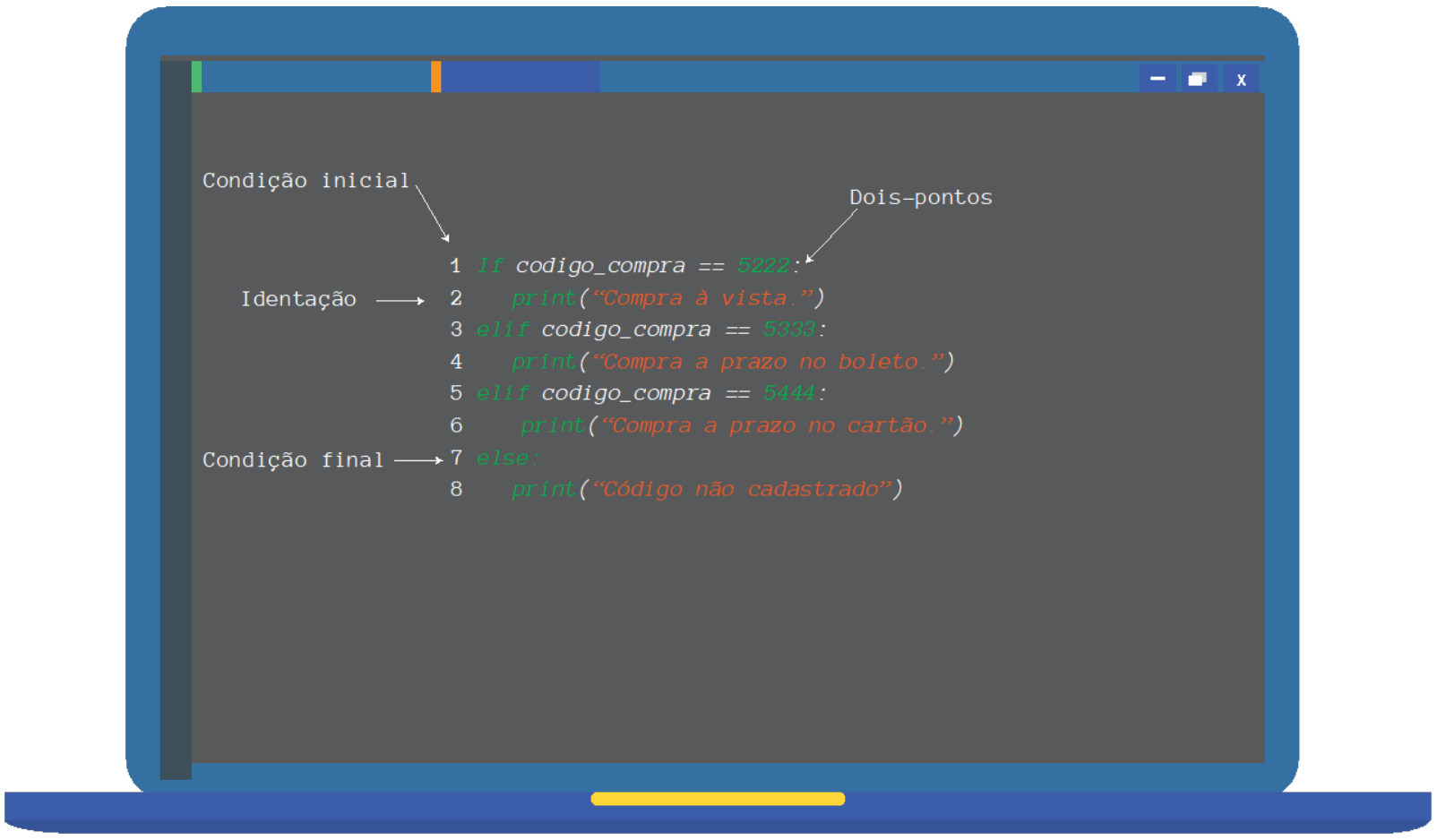
Além dos comandos, o bloco condicional é marcado pelos dois-pontos ao final da condição e pela identação do bloco de comandos a ser executado caso a condição seja verdadeira.



Fonte: Shutterstock.

Exemplos

Sintaxe estrutura condicional:



Fonte: elaborada pela autora.

No exemplo observamos que o interpretador Python identifica a relação de subordinação entre estruturas condicionais e blocos de comando pelo recuo que há na digitação das linhas do programa.

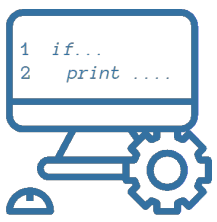
Fonte: Shutterstock.

» Na linha 1 temos a condição inicial.

» Na linha 2 um exemplo de linhas subordinadas estão digitadas com alguns espaços em branco à esquerda. Isso recebe o nome de identação.

» Na linha 3, temos o primeiro *elif*, que está no mesmo nível de identação do *if*, pois um não está subordinado a outro. O mesmo acontece para o *elif* da linha 5 e o else da linha 7.

Saiba mais



Em Python, a **identação** é obrigatória sempre que houver um ou mais comandos subordinados a outro.

"Generalizando, em Python, todo conjunto de comandos subordinados deve estar identado em relação ao seu comando proprietário. Isso vale para if-else, while, for, try, def e qualquer outro em que exista a relação de subordinação" (BANIN, 2018, p. 51).

Praticando

Imagine que você deseja acessar pela primeira vez algum sistema protegido por senha. Nesse exercício vamos implementar um simples verificador de senha do usuário e para isso utilizaremos as estruturas condicionais para permitir o login de um usuário. Vamos aplicar apenas as estruturas *if-else*.

Observe a sintaxe para esse comando:

```
1  login = input("Digite o seu login:")
2  senha = input("Digite sua senha: ")
3  if login == "userpy" and senha == "teste123":
4      print("Bem-vindo userpy01")
5  else:
6      print("Login falhou, verifique sua senha e tente novamente")
```

Fonte: elaborada pela autora.

Nas linhas 1 e 2, input é a função que requisita os dados de login e senha do usuário. A estrutura condicional *if-else*, linhas 3 e 4, determinam se a condição será satisfeita e qual linha será executada. Na linha 3, é testado se o login e a senha do usuário são iguais a "userpy" and senha == "teste123“.

Agora que você já entendeu como é comportamento da sintaxe utilize o emulador para testar o código.

E se tivermos vários usuários python (userpy) para realizar a verificação da senha, como seria? Podemos utilizar a estrutura *elif*, para checar múltiplas condições e executar determinadas linhas de código.

Observe o código para verificação de senha de 4 usuários cadastrados no sistema:

```
1  login = input("Digite o seu login:")
2  senha = input("Digite sua senha: ")
3  if login == "userpy01" and senha == "teste01":
4      print("Bem-vindo userpy01")
5  elif login == "userpy02" and senha == "teste02":
6      print("Bem-vindo userpy02")
7  elif login == "userpy03" and senha == "teste03":
8      print("Bem-vindo userpy03")
9  elif login == "userpy04" and senha == "teste04":
10     print("Bem-vindo userpy04")
11 else:
12     print("Login falhou!")
13
```

Fonte: elaborada pela autora.

As linhas 5, 7 e 9 utilizamos a estrutura *elif* para verificar outras senhas de usuário. Na linha 11, temos a estrutura *else*, que só será acionada se a entrada das linhas 1 e 2 for diferente dos logins e senhas permitidos.

Para testar esse código utilize o emulador.

Estruturas de repetição

O comando *for* em Python

A instrução *for* em Python faz uma iteração sobre os itens de qualquer sequência – por exemplo, iterar sobre os caracteres de uma palavra, já que esta é um tipo de sequência.

O bloco de comandos a ser executado deve ser identado em razão da relação e da subordinação.

A sintaxe do comando envolve:

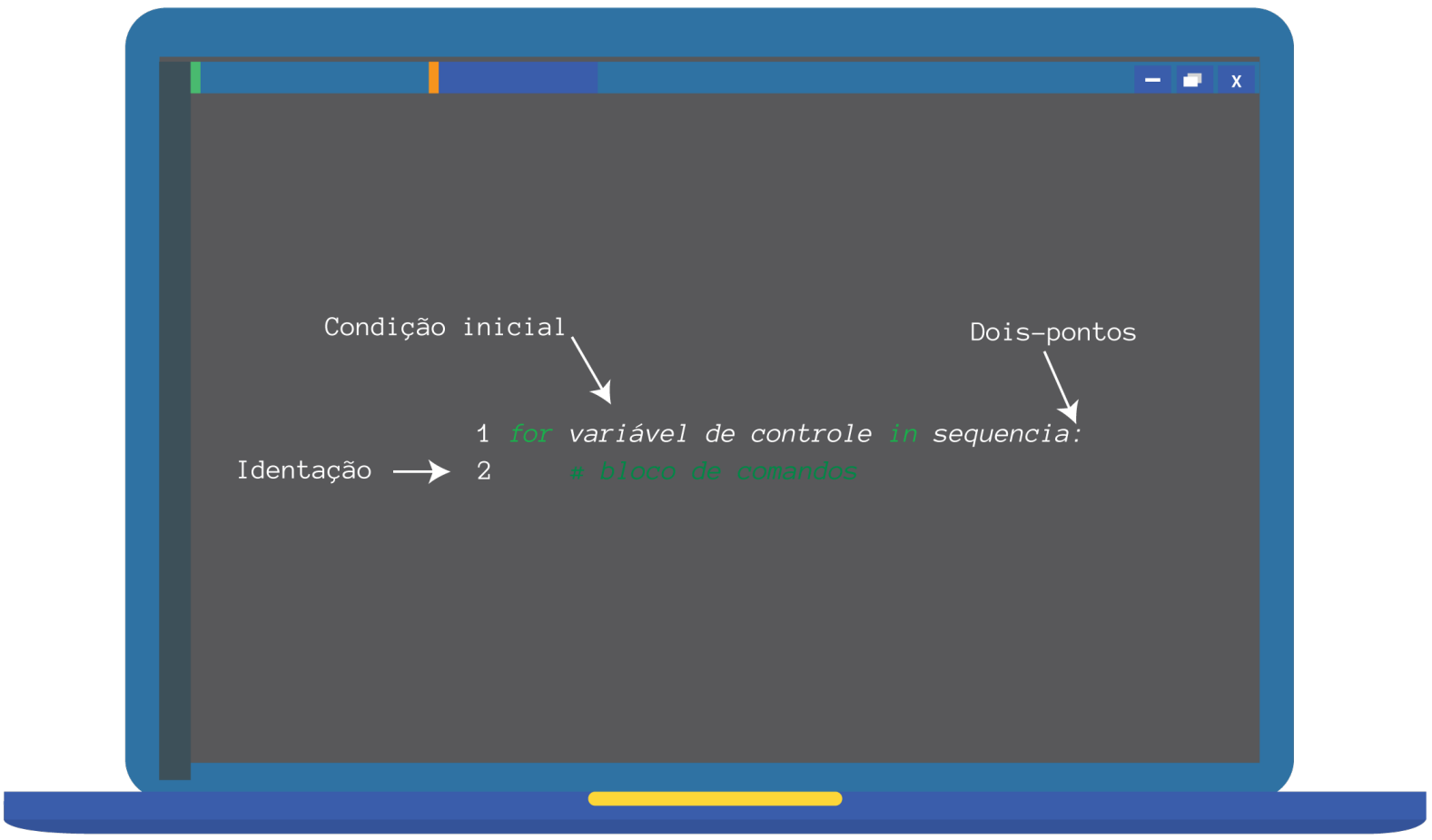
A palavra reservada “**for**” seguido de uma variável de controle;

A palavra reservada “**in**”;

E uma sequência acompanhada por dois-pontos.

Exemplos

Sintaxe do comando *for* :



Fonte: elaborada pela autora.

O comando *while* em Python

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.

Pesquise mais

A construção de uma sequência numérica pode ser feita com o comando *range()*. Ao consultar a documentação oficial, encontramos *range()* na lista de funções *built-in* em Python (2020a) . Veja o exemplo seguinte:

```
>>> list(range(10)) #Gera uma sequência de 0 à 9.
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1)11) #Gera uma sequência de 1 à
11.
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0,30,5)) #Gera uma sequência de 0 à
30 com step = 5.
[0, 5, 10, 15, 20, 25]
>>> list(range(0,-10,-1)) #Gera uma sequência
numérica negativa.
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> list(range(0)) )) #Gera uma lista vazia.
[]
```

Fonte: códigos extraídos da documentação Python (2020b).

Banin (2018), traz uma elucidação importante sobre o tipo range, apresentando-o como uma expressão, e não como uma função. Leia, para aprender mais sobre esse importante recurso, a seção 4.4 da obra: BANIN, S. L. **Python 3 - conceitos e aplicações:** uma abordagem didática. São Paulo: Érica, 2018.

Em Python, quando a mesma instrução precisa ser executada várias vezes seguidas podemos utilizar loop. Esse comando permite executar um bloco de código repetidas vezes, até que uma dada condição seja atendida.

Fonte: Shutterstock.

Praticando

Agora iremos colocar em prática a codificação do *loop* por meio dos comandos *for* e *while*.

O loop *for* é muito utilizado em conjunto com o *range*, o loop facilita a iteração dos valores sem a necessidade de escrever código para alterar esse valor, dificultando a ocorrência de um loop infinito. Observe a sintaxe do exemplo que a contagem inicia em 1 e para em 10.

```
1  contagem = 0
2  for contagem in range(1,10):
3          print(contagem)
```

Fonte: elaborada pela autora.

Para testar o loop *for* utilize o emulador:

A estrutura de repetição ***while*** executa um conjunto de instruções enquanto uma condição for verdadeira. O loop é interrompido quando a condição passa a ser falsa.

Observe o exemplo a seguir, o loop ***while*** continuará executando as duas linhas de código enquanto a contagem for menor que 10, no momento em que atingir 10, a execução é finalizada.

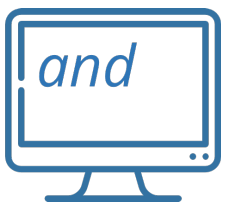
```
contagem = 0
while(contagem < 10):
        print(contagem)
        contagem = contagem + 1
```

Fonte: elaborada pela autora.

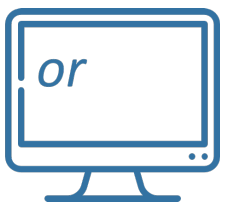
Para testar o loop ***while*** utilize o emulador:

Estruturas lógicas em python: *and*, *or*, *not*

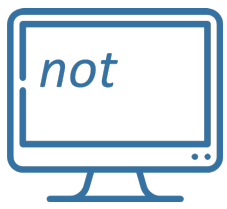
Em Python utilizamos operadores booleanos para construir estruturas de decisões mais complexas. Nesses operadores o Verdadeiro é chamado de True (igual a 1) e o Falso é chamado False (igual a 0).



Faz a expressão lógica **E**. Dessa forma esse operador retorna um valor verdadeiro somente se as duas expressões forem verdadeiras.

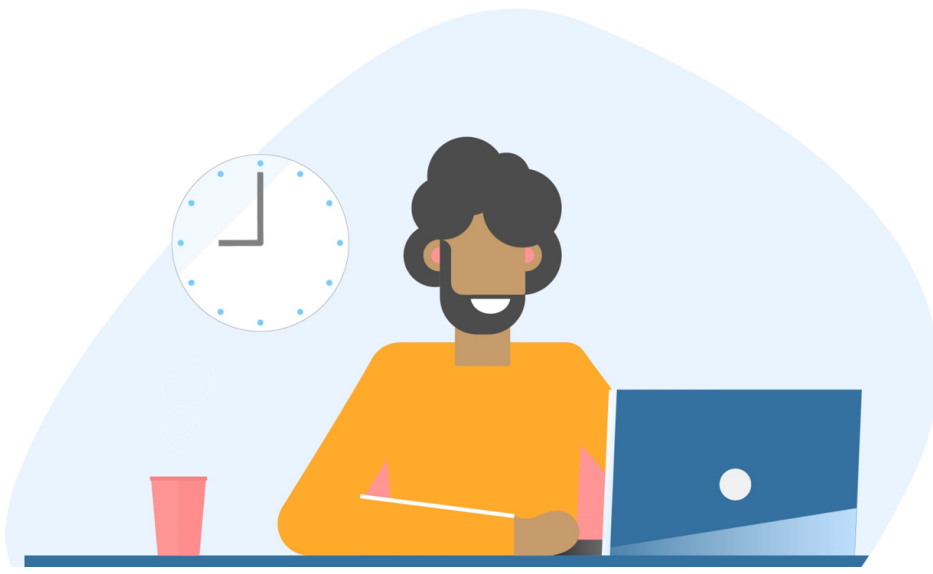


Faz a expressão lógica **OU**. Dessa forma esse operador retorna um valor falso somente se as duas expressões forem falsas.



Faz a expressão lógica **NOT**. Esse operador muda o valor de seu argumento, ou seja, se o argumento for verdadeiro, a operação o transformará em falso e vice-versa.

Os tipos de estruturas aqui apresentados, condicionais e de repetição, representam uma parte fundamental da linguagem de programação em Python, sendo assim, é muito importante conhecer a sintaxe e o funcionamento dessas estruturas.



Fonte: Shutterstock.