

# FUNDAMENTOS DA CIÊNCIA DOS DADOS E LÓGICA DE PROGRAMAÇÃO



---

# Estrutura de seleção em Python

*Rafael Albuquerque Pinto*

## OBJETIVOS DE APRENDIZAGEM

- > Diferenciar as estruturas de seleção `if` e `if/else` em Python.
  - > Listar os operadores de comparação utilizados em estruturas de seleção.
  - > Aplicar a estrutura de seleção `if/elif/else` em Python.
- 

## Introdução

As estruturas de seleção são essenciais na programação em Python. Elas permitem tomar decisões com base em condições específicas, direcionando o fluxo do programa de acordo com essas condições. No Python, há duas estruturas principais: `if` e `if/else`. A estrutura `if` é usada para executar um bloco de código somente se uma condição for verdadeira. Já a estrutura `if/else` permite executar um bloco de código se a condição for verdadeira e outro bloco se a condição for falsa. Essas estruturas fornecem flexibilidade no controle do programa, permitindo que diferentes caminhos sejam seguidos com base nas condições avaliadas.

Uma forma de estender as estruturas de seleção é utilizando a estrutura `if/elif/else`. Com o uso do `elif` (abreviação de `else if`), é possível criar condições adicionais a serem verificadas. O bloco de código dentro do `elif` será executado somente se a condição correspondente for verdadeira. Caso nenhuma das condições do `if` ou dos `elifs` seja satisfeita, o bloco de código dentro do `else` será executado. Essa estrutura permite lidar com múltiplas condições e direcionar o programa de acordo com cada caso específico.

Neste capítulo, você vai diferenciar as estruturas de seleção `if` e `if/else` em Python. Além disso, vai conhecer os operadores de comparação utilizados nessas estruturas. Por fim, vai ver a aplicação da estrutura `if/elif/else` em Python.

## Diferenças entre as estruturas de seleção `if` e `if/else` em Python

As estruturas de seleção são elementos fundamentais na programação, permitindo que um programa tome decisões com base em condições lógicas (ASCENSIO; CAMPOS, 2012). Conforme Lutz e Ascher (2007), em Python, duas estruturas de seleção comuns são o `if` (seleção simples) e o `if/else` (seleção composta). Essas estruturas fornecem mecanismos poderosos para controlar o fluxo de um programa e executar diferentes blocos de código com base em condições específicas.

A estrutura de seleção simples, representada pela palavra-chave `if`, permite que um bloco de código seja executado apenas se uma condição for avaliada como verdadeira. Por exemplo, se desejamos executar um determinado bloco de código somente quando uma variável atingir um determinado valor, podemos utilizar a estrutura `if`. Caso a condição não seja satisfeita, o bloco de código dentro do `if` é simplesmente ignorado, e a execução do programa continua. Segundo Menezes (2019), a sintaxe básica do `if` em Python é:

```
if condição:  
    #bloco de código a ser executado se a condição for  
    verdadeira
```

Por exemplo, é possível utilizar a estrutura `if` para verificar se uma variável denominada `idade` é maior ou igual a 18. Caso seja, ela deve executar um bloco de código que permite o acesso a uma determinada funcionalidade do programa. Veja o exemplo:

```
idade = int(input("Digite a sua idade: "))  
  
if idade >= 18:  
    print("Acesso permitido.")
```

Nesse exemplo, o bloco de código dentro do `if` será executado apenas se a idade for maior que 18, ou seja, se a condição for verdadeira. Caso contrário, o bloco de código será ignorado.

Por outro lado, a estrutura de seleção composta, representada pelas palavras-chave `if` e `else`, permite que dois blocos de código diferentes sejam executados com base em uma condição. De acordo com Menezes (2019), a sintaxe básica do `if/else` em Python é:

```
if condição:  
    # bloco de código a ser executado se a condição for  
    # verdadeira  
else:  
    # bloco de código a ser executado se a condição for falsa
```

Por exemplo, é possível utilizar o `if/else` para verificar se um número é positivo ou negativo e executar o bloco de código correspondente. Veja o exemplo:

```
numero = int(input("Digite o numero: "))  
  
if numero >= 0:  
    print("O número é positivo.")  
else:  
    print("O número é negativo.")
```

Nesse caso, se o número for maior ou igual a zero, o bloco de código dentro do `if` será executado, e a mensagem “O número é positivo” será exibida. Caso contrário, o bloco de código dentro do `else` será executado, e a mensagem “O número é negativo” será exibida.

Esses são apenas exemplos básicos de como as estruturas de seleção em Python podem ser utilizadas. É importante ressaltar que conhecer e dominar as estruturas de seleção em Python é fundamental para a construção de programas robustos e flexíveis. Ao compreender como utilizar essas estruturas e como combiná-las com outras estruturas de controle, como `loops` e funções, você estará apto a desenvolver soluções mais eficientes e elegantes em Python.



## Fique atento

Um aspecto importante das estruturas de seleção em Python é a utilização de indentação. Diferentemente de outras linguagens de programação que utilizam chaves ({}), ou palavras-chave para delimitar blocos de código, em Python, a indentação é crucial. A indentação é feita com espaços ou tabulações e garante a legibilidade e a correta interpretação do código. Portanto, ao utilizar as estruturas de seleção, lembre-se de aplicar a indentação adequada aos blocos de código (MENEZES, 2019).

Neste capítulo, você vai ver diferentes cenários e exemplos mais complexos, permitindo uma melhor compreensão do funcionamento e da aplicação prática dessas estruturas. Agora que você entendeu a importância e o conceito das estruturas de seleção simples e composta em Python, vai ver, na próxima seção, os operadores de comparação disponíveis na linguagem Python, que são de grande importância para a definição das condições utilizadas no bloco de `if/else`.

## Operadores de comparação em estruturas de seleção em Python

Em Python, os operadores de comparação desempenham um papel fundamental na programação, permitindo comparar valores e expressões para avaliar sua igualdade, diferença, ordem, entre outros aspectos relacionados. Esses operadores retornam valores booleanos (`True` ou `False`) para indicar se a condição especificada é verdadeira ou falsa (SEBESTA, 2018). Eles são amplamente utilizados para tomar decisões e controlar o fluxo de um programa. O Quadro 1 mostra os principais operadores de comparação em Python.

**Quadro 1.** Operadores de comparação em Python

Operador	Descrição	Exemplo
<code>==</code>	Verifica se dois valores são iguais	<code>5 == 5</code> (retorna <code>True</code> )
<code>!=</code>	Verifica se dois valores são diferentes	<code>5 != 3</code> (retorna <code>True</code> )
<code>&gt;</code>	Verifica se o valor à esquerda é maior	<code>7 &gt; 5</code> (retorna <code>True</code> )
<code>&lt;</code>	Verifica se o valor à esquerda é menor	<code>3 &lt; 5</code> (retorna <code>True</code> )

(Continua)

(Continuação)

Operador	Descrição	Exemplo
<code>&gt;=</code>	Verifica se o valor à esquerda é maior ou igual	<code>5 &gt;= 5</code> (retorna True)
<code>&lt;=</code>	Verifica se o valor à esquerda é menor ou igual	<code>3 &lt;= 5</code> (retorna True)

**Fonte:** Adaptado de Lutz e Ascher (2007).

Esses operadores são aplicáveis a diferentes tipos de dados, como números, *strings* e outros objetos comparáveis. Ao utilizar esses operadores, é importante considerar os tipos de dados e as regras de comparação associadas a cada tipo. Essas operações de comparação são essenciais para a construção de lógica condicional e controle de fluxo em programas Python. Elas permitem que você avalie condições e tome decisões com base nos resultados dessas comparações.

## Exemplos de aplicação de operadores de comparação em Python

Vamos explorar alguns exemplos práticos para ilustrar a aplicação dos operadores de comparação em Python. Vamos começar pelo operador de igualdade (`==`). A seguir, veja um exemplo:

```
x = 5
y = 5

if x == y:
    print("x é igual a y")
else:
    print("x é diferente de y")
```

Nesse exemplo, o operador de igualdade (`==`) é usado para verificar se o valor de `x` é igual ao valor de `y`. Como ambos têm o valor 5, a condição é verdadeira, então a mensagem “`x` é igual a `y`” será exibida.

## 6 Estrutura de seleção em Python

A seguir, veja um exemplo com o operador de diferença (!=):

```
x = 5
y = 3

if x != y:
    print("x é diferente de y")
else:
    print("x é igual a y")
```

Nesse exemplo, o operador de diferença (!=) é usado para verificar se o valor de x é diferente do valor de y. Como x é 5 e y é 3, a condição é verdadeira, então a mensagem “x é diferente de y” será exibida.

A seguir, confira um exemplo com o operador maior que (>):

```
x = 7
y = 5

if x > y:
    print("x é maior que y")
else:
    print("x não é maior que y")
```

Nesse exemplo, o operador maior que (>) é usado para verificar se o valor de x é maior do que o valor de y. Como x é 7 e y é 5, a condição é verdadeira, então a mensagem “x é maior que y” será exibida.

Agora veja um exemplo com o operador menor que (<):

```
x = 3
y = 5

if x < y:
    print("x é menor que y")
else:
    print("x não é menor que y")
```

Nesse exemplo, o operador menor que (<) é usado para verificar se o valor de x é menor do que o valor de y. Como x é 3 e y é 5, a condição é verdadeira, então a mensagem “x é menor que y” será exibida.

A seguir, veja um exemplo com o operador maior ou igual que ( $\geq$ ):

```
x = 5
y = 5

if x >= y:
    print("x é maior ou igual a y")
else:
    print("x não é maior ou igual a y")
```

Nesse exemplo, o operador maior ou igual que ( $\geq$ ) é usado para verificar se o valor de  $x$  é maior ou igual ao valor de  $y$ . Como  $x$  é 5 e  $y$  é 5, a condição é verdadeira, então a mensagem “ $x$  é maior ou igual a  $y$ ” será exibida.

Por fim, veja um exemplo com o operador menor ou igual que ( $\leq$ ):

```
x = 3
y = 5

if x <= y:
    print("x é menor ou igual a y")
else:
    print("x não é menor ou igual a y")
```

Nesse exemplo, o operador menor ou igual que ( $\leq$ ) é usado para verificar se o valor de  $x$  é menor ou igual ao valor de  $y$ . Como  $x$  é 3 e  $y$  é 5, a condição é verdadeira, então a mensagem “ $x$  é menor ou igual a  $y$ ” será exibida.

Esses exemplos demonstram como cada operador de comparação em Python é usado para avaliar diferentes condições e tomar decisões com base nos resultados dessas comparações.



### Saiba mais

---

Além dos operadores de comparação já citados/exemplificados anteriormente, existem outros que podem ser úteis para realizar comparações mais complexas em Python. Alguns desses operadores são os seguintes (PYTHON, 2023).

## 8 Estrutura de seleção em Python

- Operador `in`: verifica se um valor está presente em uma sequência, como uma lista, tupla ou *string*. Por exemplo:

```
lista = [1, 2, 3, 4, 5]
if 3 in lista:
    print("O valor 3 está presente na lista.")
```

- Operador `not in`: verifica se um valor não está presente em uma sequência. Por exemplo:

```
lista = [1, 2, 3, 4, 5]
if 6 not in lista:
    print("O valor 6 não está presente na lista.")
```

- Operadores `is` e `is not`: são utilizados para comparar se dois objetos são os mesmos, em vez de comparar apenas seus valores. Esses operadores são úteis ao trabalhar com objetos mutáveis em Python. Por exemplo:

```
list1 = [1, 2, 3]
list2 = [1, 2, 3]
if list1 is list2:
    print("As listas são as mesmas.")
else:
    print("As listas são diferentes.")
```

Esses operadores adicionais podem ser explorados para lidar com situações mais específicas ao realizar comparações em Python. É importante consultar a documentação oficial da linguagem Python, disponível no site oficial do Python, para obter mais informações sobre esses operadores e seus usos específicos.

---

Nesta seção, você conheceu os operadores de comparação em Python e viu como utilizá-los para tomar decisões com base em condições específicas. Na próxima seção, você vai estudar a estrutura de controle `if/elif/else`, essencial quando é necessário verificar mais de uma condição e executar diferentes blocos de código com base nos resultados dessas verificações.

## Estrutura de seleção `if/elif/else` em Python

A estrutura de controle `if/elif/else` é fundamental em Python quando é preciso verificar mais de uma condição e executar diferentes blocos de código com base nos resultados dessas verificações (AMARAL, 2018). Essa estrutura permite controlar o fluxo do programa de acordo com as condições encontradas, garantindo uma lógica condicional flexível e abrangente. A sintaxe básica do `if/elif/else` é (MENEZES, 2019):

```
if condição1:  
    # bloco de código a ser executado se a condição1 for  
    verdadeira  
elif condição2:  
    # bloco de código a ser executado se a condição1 for  
    falsa e a condição2 for verdadeira  
elif condição3:  
    # bloco de código a ser executado se a condição1 e a  
    condição2 forem falsas e a condição3 for verdadeira  
...  
else:  
    # bloco de código a ser executado se todas as condições  
    anteriores forem falsas
```

Nessa estrutura, o bloco de código associado à primeira condição verdadeira será executado. Caso nenhuma condição seja verdadeira, o bloco de código associado ao `else` será executado.

Em outras palavras, a utilização do `if/elif/else` permite lidar com situações em que múltiplas condições precisam ser verificadas e diferentes ações devem ser tomadas com base nos resultados. Cada condição é testada sequencialmente. Assim que uma condição verdadeira é encontrada, o bloco de código correspondente é executado, e as demais condições são ignoradas.

A seguir, você vai ver exemplos práticos para entender melhor como utilizar essa estrutura de controle e para ver como ela pode ser aplicada em situações reais.

## Exemplo 1: verificação de faixa etária

Suponha que você precise classificar as pessoas em três faixas etárias: menor de idade, adulto e idoso. Você pode usar o `if/elif/else` para realizar essa verificação com base na idade informada. Veja o exemplo:

```
idade = int(input("Digite a sua idade: "))

if idade < 18:
    print("Você é menor de idade.")
elif idade >= 18 and idade < 65:
    print("Você é adulto.")
else:
    print("Você é idoso.")
```

Nesse exemplo, a estrutura `if/elif/else` é utilizada para verificar a faixa etária com base na idade informada. Se a idade for menor que 18, o programa imprimirá “Você é menor de idade”. Se a idade estiver entre 18 e 64, o programa imprimirá “Você é adulto”. Se a idade for igual ou superior a 65, o programa imprimirá “Você é idoso”.

## Exemplo 2: classificação de notas

Suponha que você precise classificar as notas dos alunos em conceitos, como Excelente, Bom, Regular e Reprovado. Você pode usar o `if/elif/else` para realizar essa classificação com base na nota informada. Veja o exemplo:

```
nota = float(input("Digite a nota do aluno: "))

if nota >= 9.0:
    print("Conceito A: Excelente.")
elif nota >= 7.0 and nota < 9.0:
    print("Conceito B: Bom.")
elif nota >= 5.0 and nota < 7.0:
    print("Conceito C: Regular.")
else:
    print("Conceito D: Reprovado.")
```

Nesse exemplo, a estrutura `if/elif/else` é utilizada para classificar as notas dos alunos com base em faixas de valores. Se a nota for maior ou igual a 9.0, o programa imprimirá “Conceito A: Excelente”. Se a nota estiver entre 7.0 e 8.9, o programa imprimirá “Conceito B: Bom”. Caso a nota esteja entre 5.0 e 6.9, o programa imprimirá “Conceito C: Regular”. Se a nota for menor que 5.0, o programa imprimirá “Conceito D: Reprovado”.

Esses exemplos demonstram como a estrutura `if/elif/else` pode ser utilizada para realizar verificações de múltiplas condições em Python. Ela oferece flexibilidade e permite que você controle o fluxo do programa de acordo com diferentes cenários. Essa estrutura é amplamente utilizada na programação para tomar decisões complexas com base em múltiplas condições.



## Saiba mais

---

Existem funções e métodos úteis para trabalhar com *strings* e caracteres em Python, permitindo que você faça verificações e operações específicas de acordo com as necessidades do seu código. A seguir, veja alguns exemplos (PYTHON, 2023).

- Função `len()`: retorna o comprimento de um objeto, como uma *string*, lista, tupla, etc. Ela retorna o número de elementos presentes no objeto.
  - Função `any()`: verifica se pelo menos um elemento de um iterável é verdadeiro. Ela retorna `True` se algum elemento for avaliado como verdadeiro e `False` em caso contrário. Pode ser usada para verificar se pelo menos um elemento de uma lista atende a uma determinada condição.
  - Métodos `isupper()` e `islower()`: verificam se todos os caracteres em uma *string* estão em maiúsculas ou minúsculas. O `isupper()` retorna `True` se todos os caracteres forem maiúsculos e `False` em caso contrário. Já o `islower()` retorna `True` se todos os caracteres forem minúsculos e `False` em caso contrário.
  - Método `isdigit()`: verifica se todos os caracteres em uma *string* são dígitos (0 a 9). Ele retorna `True` se todos os caracteres forem dígitos e `False` em caso contrário.
  - Método `isalnum()`: verifica se todos os caracteres em uma *string* são alfanuméricos, ou seja, letras (maiúsculas ou minúsculas) ou dígitos (0 a 9). Ele retorna `True` se todos os caracteres forem alfanuméricos e `False` em caso contrário.
-

Neste capítulo, você viu os operadores de comparação em Python e o uso da estrutura de controle `if/elif/else`. O domínio desses conceitos permite a construção de algoritmos eficientes, a tomada de decisões inteligentes e a resolução de problemas complexos. Recomenda-se a aprendizagem contínua por meio de outras fontes e a prática constante para fortalecer essas habilidades. Dominar os operadores de comparação e a estrutura `if/elif/else` em Python é essencial para profissionais da área de programação, possibilitando o desenvolvimento de programas mais robustos e adaptáveis. Essas competências impulsionam o crescimento profissional e contribuem para o avanço da tecnologia.

## Referências

- AMARAL, F. *Introdução a ciência de dados*. Rio de Janeiro: Alta Books, 2018. E-book.
- ASCENSIO, A. F. G.; CAMPOS, E. A. V. *Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ (padrão ANSI) e Java*. 3. ed. São Paulo: Pearson, 2012.
- LUTZ, M.; ASCHER, D. *Aprendendo Python*. 2. ed. Porto Alegre: Bookman, 2007.
- MENEZES, N. N C. *Introdução à programação com Python: algoritmos e lógica de programação para iniciantes*. 3. ed. Rio de Janeiro: Novatec, 2019.
- PYTHON. Tipos embutidos. *Python Software Foundation*, 2023. Disponível em: <https://docs.python.org/pt-br/3/library/stdtypes.html>. Acesso em: 18 jul. 2023.
- SEBESTA, R. W. *Conceitos de linguagens de programação*. 11. ed. Porto Alegre: Bookman, 2018.

## Leitura recomendada

DOWNEY, A. B. *Pense em Python: pense como um cientista da computação*. Rio de Janeiro: Novatec, 2016.



### Fique atento

Os *links* para sites da web fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declararam não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.



Conteúdo:

**sagah**<sup>+</sup>