**Operadores no Python**

Veja neste conteúdo os diferentes operadores da linguagem Python e como utilizá-los.

Os operadores são utilizados para construir expressões lógicas, aritméticas, associativas, atribuir valores a variáveis e compará-los. Trata-se de um recurso utilizado nas tarefas mais fundamentais da programação, conforme veremos neste artigo e, portanto, de conhecimento obrigatório para qualquer programador.

Operadores no Python

Os operadores são usados na construção de expressões, as quais contém um número variado de operandos. Por exemplo, na expressão a + b, temos o operador de aritmético + e operandos são as variáveis a e b.

Na linguagem Python temos a seguinte separação entre os diferentes tipos de operadores:

* Operadores aritméticos
* Operadores de atribuição
* Operadores de comparação
* Operadores lógicos
* Operadores de identidade
* Operadores de associação

A seguir veremos cada um deles.

**Operadores aritméticos**

Os operadores aritméticos são utilizados na execução de operações matemáticas, tais como a soma e a subtração, por exemplo. Vejamos na **Tabela 1** a lista deles.

| **Operador** | **Conceito** | **Exemplo** |
| --- | --- | --- |
| + (Adição ou sinal positivo) | - Realiza a soma entre operandos - Adiciona o sinal de positivo ao número | - 10 + 7 - +4 |
| - (Subtração ou sinal negativo) | - Realiza a subtração entre operandos - Adiciona o sinal de negativo ao número | - 10 - 7 - -4 |
| \* (Multiplicação) | Realiza a multiplicação entre operandos | 3 \* 4 |
| / (Divisão) | Realiza a divisão entre operandos | 10 / 5 |
| // (Divisão inteira) | Realiza a divisão entre operandos e a parte decimal do resultado | 10 // 6 |
| % (Módulo) | Retorna o resto da divisão entre operandos | 4 % 2 |
| \*\* (Exponenciação) | Retorna um número elevado a potência de outro | 4 \*\* 2 |

**Tabela 1**. Operadores aritméticos

O **Código 1** mostra um exemplo com os operadores aritméticos usados no Python.

numero\_1 = 5

numero\_2 = 2

soma = numero\_1 + numero\_2

subtracao = numero\_1 - numero\_2

multiplicacao = numero\_1 \* numero\_2

divisao = numero\_1 / numero\_2

divisao\_inteira = numero\_1 // numero\_2

modulo = numero\_1 % numero\_2

exponenciacao = numero\_1 \*\* numero\_2

print(soma) # 7

print(subtracao) # 3

print(multiplicacao) # 10

print(divisao) # 2.5

print(divisao\_inteira) # 2

print(modulo) # 1

print(exponenciacao) # 25

**Código 1**. Operadores aritméticos

Uma característica importante a ser observada quando falamos dos operadores matemáticos é a precedência. Essa característica é relativa à ordem da execução deles e acontece quando mais de um operador está presente numa expressão. Segue a precedência dos operadores no Python.

Nos exemplos abaixo usaremos a função print() para exibir os resultados das expressões.

1. As expressões contidas em parênteses têm a precedência maior na linguagem Python. Isso permite que uma expressão execute antes de outra. Ex.:

print((2 + 5) \* 3) # O resultado será 21

1. Após os parênteses, o próximo operador com maior precedência é o de exponenciação. Ex.:

print( 1 + 5\*\*2 ) # O resultado será 26

1. Multiplicação e divisão têm precedência sobre a adição e subtração: como já é conhecido na matemática, divisão e multiplicação são executadas antes das operações de adição e subtração. Ex.:

print(5 \* 3 + 8) # O resultado será 23

1. Ordem de precedência é avaliada da esquerda para a direita. Portanto, após os operadores anteriores, a sequência da execução será da esquerda para a direita. Ex.:

print(8 + 5 - 10) # O resultado será 3

**Operadores de atribuição**

Os operadores de atribuição atribuem valor a uma variável. Na **Tabela 2** temos uma lista desses operadores.

| **Operador** | **Exemplo** | **Equivalente a** |
| --- | --- | --- |
| = | x = 1 | x = 1 |
| += | x += 1 | x = x + 1 |
| -= | x -= 1 | x = x - 1 |
| \*= | x \*= 1 | x = x \* 1 |
| /= | x /= 1 | x = x / 1 |
| %= | x %= 1 | x = x % 1 |

**Tabela 2**. Operadores de atribuição

No **Código 2** temos um exemplo do uso de operadores de atribuição.

numero\_1 = 5

numero\_1 += 5

print(numero\_1) # O resultado será 10

**Código 2**. Operadores de atribuição

Acima, vemos que a variável numero\_1 recebe na linha 3 o valor dela mesmo somado a 5. Isso fará com que o resultado impresso na linha 5 seja igual a 10.

**Operadores de comparação**

Os operadores de comparação são usados para comparar valores, o que vai retornar True ou False, dependendo da condição. A seguir, na **Tabela 3** temos exemplos de alguns usados no Python.

| **Operador** | **Conceito** | **Exemplo** |
| --- | --- | --- |
| >(Maior que) | Verifica se um valor é maior que outro | x > 5 |
| <(Menor que) | Verifica se um valor é menor que outro | x < 5 |
| == (Igual a) | Verifica se um valor é igual a outro | x == 5 |
| != (Diferente de) | Verifica se um valor é diferente de outro | x != 5 |
| >= (Maior ou igual a) | Verifica se um valor é maior ou igual a outro | x >= 5 |
| <= (Menor ou igual a) | Verifica se um valor é menor ou igual a outro | x <= 5 |

**Tabela 3**. Operadores de comparação

Usando a comparação que é feita com esses operadores, podemos criar condições para os códigos. No exemplo abaixo, vemos como isso acontece na prática:

numero\_1 = 2

numero\_2 = 4

soma = numero\_1 + numero\_2

if soma < 10:

print("soma não é maior que 10")

else:

print("soma é maior ou igual a 10")

**Código 3**. Operador de comparação

Como podemos ver no **Código 3**, duas variáveis são somadas e esse resultado é atribuído à variável soma. Na linha 6, fazemos uma comparação dessa variável verificando se ela é menor do que 10. Vejamos a seguir, uma operação que faz comparação de igualdade entre duas variáveis:

soma\_1 = 7 + 8

soma\_2 = 10 + 5

if soma\_1 == soma\_2:

print("Os resultados são iguais")

else:

print("Os resultados são diferentes")

**Código 4**. Operador de comparação

No **Código 4**, criamos duas variáveis e comparamos seus valores, o que pode resultar em dois resultados distintos. Dessa forma, podemos criar estruturas condicionais com o auxílio desses operadores.

**Operadores lógicos**

Os operadores lógicos são usados para unir duas ou mais expressões condicionais. Isso é feito por meio de conectivos, como podemos ver na **Tabela 4**.

| **Operador** | **Conceito** | **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) |

**Tabela 4**. Operadores lógicos

Vejamos a seguir, um exemplo que utiliza os operadores lógicos:

idade\_lucas = 21

idade\_carolina = 19

# OPERADOR OR

if idade\_lucas >= 18 or idade\_carolina >= 18:

print("Pelo menos um dos dois é maior de idade")

else:

print("Lucas e Carolina não são maiores de idade")

# OPERADOR AND

if idade\_lucas >= 18 and idade\_carolina >= 18:

print("Lucas e Carolina são maiores de idade")

else:

print("Pelo menos um dos dois não é maior de idade")

**Código 5**. Operadores lógicos

No **Código 5** temos um exemplo no qual usamos os operadores or e and. Com o operador or, apenas uma das condições precisa ser verificadas. Já no operador and, todas as condições precisam ser satisfeitas. Com a execução do código teremos a impressão das mensages “Lucas e Carolina são maiores de idade” e "Pelo menos um dos dois é maior de idade".

**Operadores de identidade**

Os operadores de identidade servem para a comparação de objetos. Nessa comparação é verificado se eles ocupam a mesma posição na memória, o que significará que se trata do mesmo objeto, como vemos isso na **Tabela 5**.

| **Operador** | **Conceito** | **Exemplo** |
| --- | --- | --- |
| is | Retorna True se as variáveis comparadas forem o mesmo objeto | nome is ‘Marcos’ |
| is not | Retorna True se as variáveis comparadas não forem o mesmo objeto | x is not ‘Python’ |

**Tabela 5**. Operadores de identidade

Vejamos com mais detalhes o uso dos operadores de identidade:

time\_carlos = 'Botafogo'

time\_luciano = 'Flamengo'

time\_fabricia = 'Botafogo'

if time\_carlos is time\_luciano:

print("time\_carlos - time\_luciano = mesmo objeto")

else:

print("time\_carlos - time\_luciano = objetos diferentes")

if time\_carlos is time\_fabricia:

print("time\_carlos - time\_fabricia = mesmo objeto")

else:

print("time\_carlos - time\_fabricia = objetos diferentes")

**Código 6**. Operadores de identidade

No **Código 6** temos um exemplo do operador is no Python. Com ele, o que está sendo verificado é se as variáveis apontam para o mesmo objeto. Quando declaramos uma string no Python, o seu valor é guardado num local especial da memória. Assim, quando usamos esse valor repetidas vezes como nas linhas 1 e 3, o Python pode utilizar o mesmo objeto em cada uma delas. Por esse motivo is dirá que time\_carlos e time\_fabricia apontam para o mesmo objeto, enquanto que time\_luciano aponta para um objeto diferente.

[**Neste artigo**](https://www.devmedia.com.br/tipos-de-dados-em-python-string/40669), podemos ver strings em mais detalhes.

**Operadores de associação**

Os operadores de associação são utilizados para verificar se uma sequência contém um objeto. Na **Tabela 6** temos os exemplos.

| **Operador** | **Conceito** | **Exemplo** |
| --- | --- | --- |
|  |  |  |
| in | Retorna True caso o valor seja encontrado na sequência | 2 in x |
| not in | Retorna True caso o valor não seja encontrado na sequência | 2 not in x |

**Tabela 6**. Operadores de associação

No **Código 7**, vamos ver um exemplo usando os operadores de associação.

frutas = ["banana","laranja","uva","ameixa"]

fruta\_1 = "ameixa"

fruta\_2 = "melancia"

print(fruta\_1 in frutas) # True

print(fruta\_2 in frutas) # False

**Código 7**. Operadores de associação

O exemplo acima verifica a existência de variáveis dentro de uma lista. Na linha 6 estamos checando se fruta\_1 está contido em frutas. Da mesma forma, fazemos isso na linha 7, só que dessa vez com fruta\_2.

**Conclusão**

Ao usar os operadores no Python, podemos atribuir valores a variáveis, criar condições, fazer comparações etc. Essas características fazem parte dos princípios básicos para montar algoritmos nessa linguagem. É importante dominar esses conceitos sobre os operadores, pois eles são amplamente usados no dia a dia do desenvolvedor.