# OPERADORES

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

Operadores aritméticos são usados com valores numéricos para realizar operações matemáticas comuns:

| Operador | Nome |
|---|---|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |
| % | Modulus(Resto) |
| ** | Exponencial |
| // | Floor Division(Quociente) |

## Operadores de atribuição Python

Operadores de atribuição são usados para atribuir valores a variáveis:

| Operador | Exemplo | Resultado |
|---|---|---|

```
  =            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
```

## Operadores de comparação Python

Operadores de comparação são usados para comparar dois valores:

| Operador | Nome | Exemplo |
|---|---|---|
| == | igual | x == y |
| != | não igual | x != y |
| > | Maior que | x > y |
| < | Menor que | x < y |
| >= | Maior ou igual a | x >= y |
| <= | Menor ou igual a | x <= y |

## Operadores lógicos Python

Operadores lógicos são usados para combinar declarações condicionais:

| Operador | Descrição | Exemplo |
|---|---|---|
| and | Retorna ‘True’ se ambas declarações forem verdadeira | x = 4; x < 5 and x < 10 |
| or | Retorna ‘True’ se uma declarações for verdadeira | x = 4; x < 5 or x < 4 |
| not | Reverte o resultado, retorna ‘False’ se o resultado for verdadeiro | x = 4; not(x < 5 and x < 10) |

## Operadores de identidade Python

Operadores de identidade são usados para comparar os objetos, não se eles forem iguais, mas se eles forem realmente o mesmo objeto, com o mesmo local 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 |

## Operadores de associação Python

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 bit a bit Python

Operadores bit a bit são usados para comparar números (binários), caso ambos os operandos sejam strings, esses operadores irão trabalhar com os valores ASCII de seus caracteres:

## AND(&)

Compara dois valores utilizando suas representações binárias e retorna um novo valor, para formar esse valor de retorno, cada bit é comparado, retornando 1(True) quando ambos os bits forem iguais a 1(True), caso contrário retorna 0 (False).

### Exemplo:

```
a = 5 #00000101
b = 3 #00000011
#00000101 & 00000011
c = a & b #00000001
```

## OR(|)

Compara dois valores utilizando suas representações binárias e retorna um novo valor, para formar esse valor de retorno, cada bit é comparado, retornando 1(True) se um dos bits comparados forem iguais a 1(True), caso contrário retorna 0 (False).

```
a = 5 #00000101
b = 3 #00000011
#00000101 | 00000011
c = a | b #00000111
```

## XOR(^)

Compara dois valores utilizando suas representações binárias e retorna um novo valor, para formar esse valor de retorno, cada bit é comparado, retornando 1(True) quando os bits comparados forem diferentes, caso contrário retorna 0 (False).

```
a = 5 #00000101
b = 3 #00000011
#00000101 & 00000011
c = a ^ b #00000110
```

## NOT(~)

A inversão bit a bit de uma variável A é definida como – (A + 1). Aplica-se apenas a números integrais.

```
bin(~0b1111) #-0b10000
bin(~0b0000) #-0b1#
bin(~0b1010) #-0b1011
bin(~0b1011) #-0b1100
```

## Deslocamento para esquerda(<<)

Retorna um número com os bits deslocados *n* posições à esquerda. O valor de *n* deve ser positivo. Um deslocamento à esquerda é a mesma coisa que fazer a multiplicação inteira do operando por $2^n$.

### Exemplo:

```
a = 5                    #em binário é 00000101(byte)
bin(a)                   #saída '0b101' (0x=hexa, 0b=binario, 0o=octal)
print("{0:b}".format(a)) #saída '101'
b = a << 3               #desloca 3 casas para esquerda, equivale a multiplicar "a" po
r 2 três vezes
bin(b)                   #saída '0b101000', ou
print("{0:b}".format(b)) #saída '101000'
```

## Deslocamento para direita(>>)

Retorna um número com os bits deslocados *n* posições à direita. O valor de *n* deve ser positivo. Um deslocamento à direita é a mesma coisa que divisão inteira do operando por $2^n$.

```
x = 32                   #em binário é 00100000(byte)
bin(x)                   #saída '0b100000' (0x=hexa, 0b=binario, 0o=octal)
print("{0:b}".format(x)) #saída '100000'
n = 4                    #definindo uma referencia para ser usada
fator = 2 ** n           #32/2**4 = 32/16 = 2
y = x >> n               #desloca 3 casas para direita, equivale a dividir "x" por 2 q
uatro vezes
x // fator               # 100000 -> 010000 -> 001000 -> 000100 -> 000010
print(bin(y))

#Resultado:
100000
0b10
```