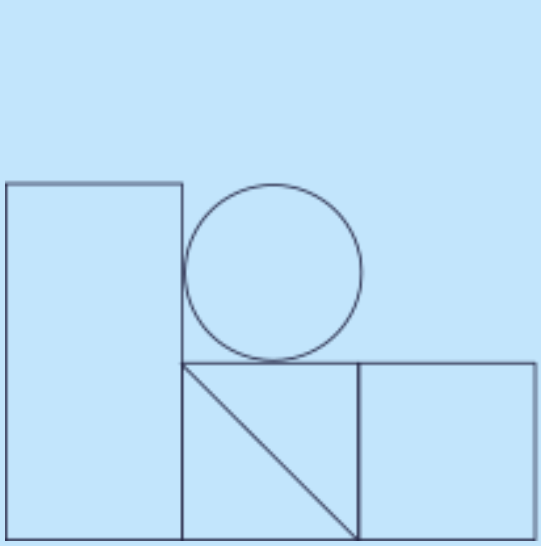
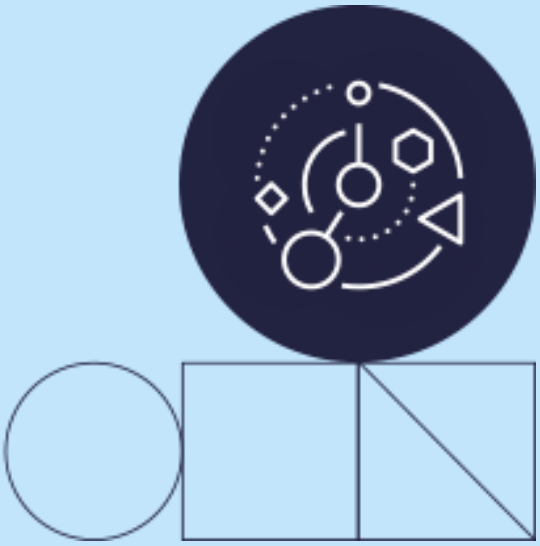
IBM SkillsBuild

IBM

Noções básicas e sintaxe de Python

Operadores de Python





Índice

Introdução

Agora que vimos os primeiros tipos básicos de Python, vamos fazer um interlúdio para introduzir os operadores Python básicos que nos permitirão operar tanto com números como com o resto dos tipos que vamos aprender sobre.

Como veremos em unidades posteriores, um programa Python é composto de módulos, que são compostos de declarações que, por sua vez, são compostas de expressões.

Uma expressão é uma combinação de números (ou outros objetos Python) e operadores que, quando executados, produzem um valor. Nesta seção vamos aprender sobre os operadores que permitem combinar objetos dentro das expressões.

Muitos destes operadores não serão utilizados neste curso, mas manterão a tabela para referência futura.

Operadores

Aritmética	Comparação	Lógica	Atribuição	Especiais
Adicione +	O mesmo que ==	AND	Igual =	IS
Subtrair -	Diferente de !=	OR	Aumento +=	IS NOT
Multiplicação *	Maior do que >	NOT	Diminuição -=	IN
Divisão /	Menos de <		*=	NOT IN
Módulo %	Maior ou igual a >=		/=	
Exponente **	Menor ou igual a <=		%=	
Divisão inteira //			**=	
			//=	

Muitos destes operadores não serão utilizados neste curso, mas manterão a tabela para referência futura.

Ordenando operadores por precedência (do mais baixo para o mais alto)

- **yield x**: Protocolo do gerador “send”
- **lambda args**: **expresión**: Função anônima
- **x if y else z:** Seleção ternária (retorna x se y é verdadeiro)
- **x or y**: OR lógico (y é avaliado somente se x é falso)
- **x and y**: AND lógico (y é avaliado somente se x é verdadeiro)
- **not x**: Negação Lógica
- **x in y, x not in y**: Operadores associados
- **x is y, y is not y**: Operadores de identidade
- **x < y, x<= y, x > y, x >= y**: Comparação de magnitudes. Set, subset e superset
- **x == y, x != y**: Operadores de equidade
- **x | y**: Bit a bit (bitwise) OR. União de sets (conjuntos)
- **x ^ y**: Bitwise XOR. Diferença simétrica de sets
- **x & y**: Bit a bit AND. Intersecção de sets (conjuntos)
- **x << y, x >> y**: Deslocamento de x em y bits à esquerda e à direita
- **x + y**: Adição. Concatenação
- **x – y**: Subtração. Diferença entre sets (conjuntos)
- **x * y**: Multiplicação. Repetição
- **x % y**: O resto da divisão
- **x / y**: Divisão real (verdadeira)
- **x // y**: Divisão truncada
- **-x, +x**: Negação. Identidade
- **~x**: Negação Bit a bit
- **x ** y**: Poder (exponenciação)
- **x[i]**: Indexação (seqüências, mapeamentos, outros)
- **x[i:j:k]**: Cortado (slicing)
- **x(...)**: Chamada (função, método, classe, etc.)
- **x.attr**: Referência a um atributo
- **(...)**: Tuple, expressão, expressão geradora
- **[...]**: Lista, lista para entendimento
- **{...}**: Diccionario, Set, comprensión de diccionarios y sets

Combinando operadores e precedência

Quando vários operadores são combinados na mesma expressão, é necessário levar em conta quais operadores precedem (são avaliados antes) outros operadores. La **Tabela 2** mostra todos os operadores Python em ordem de precedência. Ou seja, os operadores que estão mais altos na tabela têm menor precedência sobre os que estão mais baixos na tabela. Para os operadores da mesma fila, a avaliação é feita da esquerda para a direita, exceto pela exponenciação, que é agrupada da direita para a esquerda, e as comparações, que são acorrentadas da esquerda para a direita.

Assim, em uma operação onde as multiplicações e adições são encadeadas, por exemplo, A * B + C * D, as multiplicações são realizadas primeiro e depois a adição.

Uma maneira de contornar a precedência é usar parênteses para isolar expressões. Em Python, as expressões entre parênteses são sempre avaliadas antes das demais. Portanto, na expressão acima, se quiséssemos avaliar a soma primeiro, faríamos o seguinte: A * (B + C) * D.

```
>>> 2 * 3 + 2 * 4              # Multiplicación precede a suma
14

>>> 2 * (3 + 2) * 4           # Expresiones entre paréntesis se evalúan primero
40
```