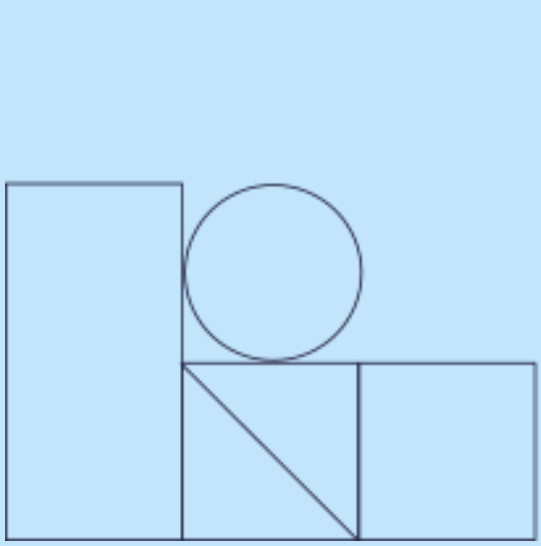
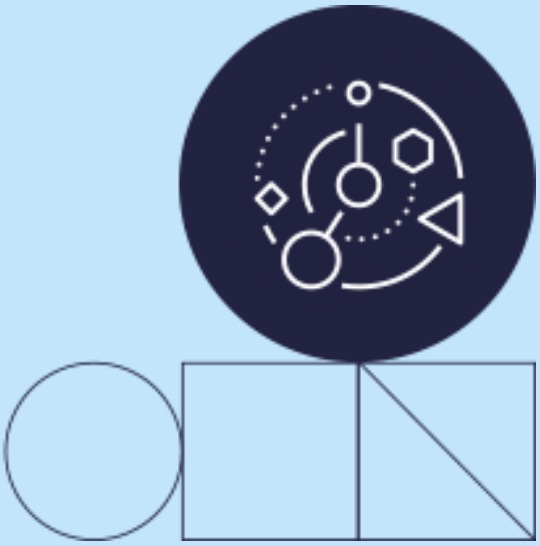
IBM SkillsBuild

IBM

Noções básicas e sintaxe de Python

Tipos de dados em Python





Índice

Introdução 3

Tipos inatos/básicos da Python (Builtins) 4

Objetos básicos que facilitam a programação 4

Os objetos builtin são os blocos de construção básicos para objetos mais complexos 4

Os tipos básicos são muito eficientes 4

Os objetos builtin são uma parte padrão da linguagem 4

Um passeio pelos tipos básicos de Python 5

Números 5

Introdução

Em **Python**, todos os dados com os quais interagimos são objetos, sejam eles dados inatos/básicos (builtins) ou classes criadas por nós mesmos. Por objetos queremos dizer que são simplesmente pedaços de memória onde definimos valores e conjuntos de operações associados a esses valores.

Como veremos em outros vídeos e cursos desta especialização, o fato de que tudo é um objeto tem certas vantagens. E quando queremos dizer tudo, queremos dizer absolutamente tudo, incluindo tipos básicos como números (7, 1,41, etc.), cordas ('hola mundo', 'a') e até mesmo operações (adição, subtração, etc.).

Os tipos básicos ou builtins são de vital importância em **Python**. Portanto, nesta unidade, estudaremos quais são estes tipos e quais as operações que podemos fazer com eles.

Tipos inatos/básicos da Python (Builtins)

Como no resto das linguagens de programação, os elementos básicos com os quais trabalharemos em Python são variáveis. Entendemos variável como um espaço na memória onde um dado será armazenado que pode mudar durante a execução do programa.

Python tem um sistema do tipo básico que é muito rico comparado com outras linguagens como C ou C++. Nesses idiomas, muito do trabalho consiste em criar as estruturas de dados que representam os componentes da aplicação programada. Além disso, essas estruturas requerem a implementação de funções de busca, acesso, gerenciamento de memória, etc. Estas tarefas distraem do objetivo principal do programador, além de serem uma possível fonte de erros.

Tudo isso pode ser minimizado quando a linguagem em que trabalhamos inclui um sistema de tipo poderoso, como no caso do **Python**. Graças a isso, podemos nos concentrar em resolver nosso problema diretamente, sem ter que implementar primeiro um sistema de dados adaptado ao problema. Obviamente, haverá casos em que isso ainda será necessário, mas na maioria dos casos, é preferível tirar proveito dos tipos básicos de **Python**. Aqui estão algumas das razões pelas quais isso acontece:

Objetos básicos que facilitam a programação

Como acabamos de mencionar, para muitas tarefas simples, é suficiente utilizar os tipos de builtins. Estes tipos oferecem ferramentas como coleções (listas, sets), dicionários, etc. que você pode usar imediatamente e que lhe permitem resolver uma ampla gama de problemas.

Os objetos builtin são os blocos de construção básicos para objetos mais complexos

Caso seja necessário programar software mais complexo, pode ser necessário implementar objetos à mão. Como veremos em outras unidades, estes objetos são construídos sobre os tipos básicos de **Python**, tais como listas, tuplas e dicionários.

Os tipos básicos são muito eficientes

Na grande maioria dos casos, os tipos de construção são mais eficientes do que os objetos criados por nós mesmos. Os tipos de builtin **Python** são estruturas de dados que foram otimizadas e implementadas internamente em C para aumentar sua velocidade.

Os objetos builtin são uma parte padrão da linguagem

Isto significa que, como estamos utilizando componentes padrão da linguagem, temos a garantia de estabilidade, interoperabilidade entre sistemas, etc. Isto não pode ser garantido por nenhum componente ou framework de terceiros.

Como regra básica, considere sempre o uso de objetos builtin antes de suas próprias implementações. Na maioria dos casos, isto será suficiente para resolver seu problema, facilitará sua tarefa de programação e tornará seu programa mais fácil de manter, extensível e eficiente.

Um passeio pelos tipos básicos de Python

A tabela a seguir mostra uma lista das principais tipos básicos (builtins) de **Python**, bem como alguns exemplos dos literais que permitem a sua criação.

Tipo	Exemplo de como criá-lo
Números (int, double, etc.)	12, 2.41, 1+3j, 0b101
Strings	'Python', u'caf\xe9'
Listas	[1, 2, 3], [3, ['hola', 2.41]], list(range(3))
Diccionarios	{'pais':'ES', 'city':'Mad'}, dict('p'='ES', c='M')
Tuplas	(1, 2, 3, 4), tuple('Python')
Sets	{'rojo', 'verde', 'azul'}, set('abcdef')
Ficheros	open('mi_fichero.txt')
Otros tipos	Booleanos, None
Tipos de Unidades de programa	funciones, clases, módulos

Como podemos ver no **Tabela 1**, em **Python** até mesmo as funções e módulos são objetos. Isto significa que podemos passá-los como parâmetros para outras funções, armazená-los como atributos de outros objetos, etc.

Um aspecto importante a ter em mente: **Python** é uma linguagem ***tipado dinâmico***. Isto significa que **Python** controla o tipo de objetos para nós, poupando-nos de ter que declarar tipos em nosso código. Mas **Python** é também uma linguagem ***tipado forte***, o que significa que uma vez que declaramos um objeto, só podemos realizar operações sobre esse objeto que sejam válidas para esse tipo.

Nos vídeos deste tópico, vamos passar por estes tipos a fim de aprofundá-los.

Números

Os objetos numéricos builtin em **Python** são típicos de qualquer linguagem de programação: números inteiros e números de *ponto flutuante* (com decimais) assim como outros, tais como *números complexos* com parte imaginária; decimais de precisão fixa; e números racionais com numerador e denominador.

Observe que, na segunda expressão, atribuímos o resultado na variável a. Em **Python**, as variáveis não precisam ser declaradas antecipadamente; elas são criadas quando lhes atribuímos um valor. As variáveis em **Python** podem ser atribuídas a qualquer tipo de objeto e serão substituídas com seu valor sempre que aparecerem em uma expressão.

Além destas expressões, você pode importar módulos externos para realizar mais operações matemáticas:

```
>>> import math
>>> math.pi * 4
12.566370614359172

>>> math.pi ** 2
9.869604401089358

>>> math.sqrt(4)           # Raíz cuadrada
2

>>> math. sqrt(12)
3.4641016151377544
```

Você pode explorar o módulo ***math***, que já está incluída na biblioteca padrão **Python**, para ver mais utilidades e operações disponíveis.

Outro módulo interessante é o ***random*** que, como seu nome sugere, nos permite gerar e selecionar números aleatórios, entre muitas outras opções.

```
>>> import random
>>> random.random ()
0.6567888091274212

>>> lista = [1, 2, 3, 4]

>>> random.choice(lista)
3

>>> random.shuffle(lista)
>>> lista
[2, 3, 1, 4]
```

A variável de lista que acabamos de declarar nada mais é do que uma lista de números. Este é outro dos tipos básicos de **Python** que veremos mais adiante. Por enquanto, basta entender que as listas são coleções de objetos encomendados.