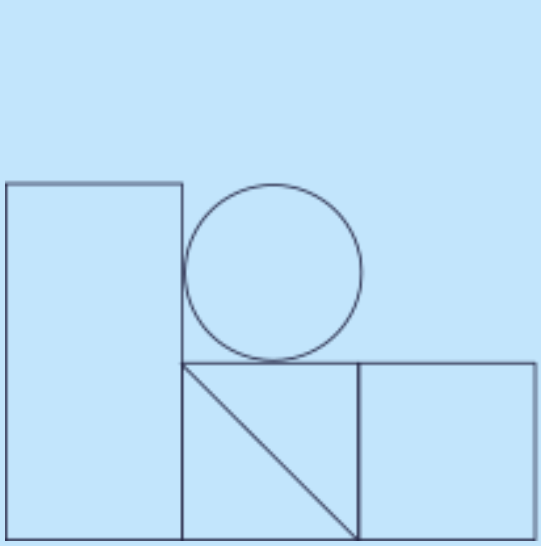
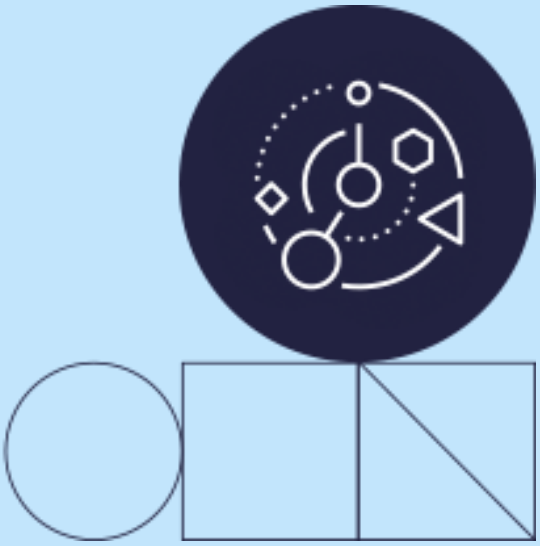
IBM SkillsBuild

IBM

Conceptos básicos y sintaxis de Python

Tipos de datos en Python





Índice

Introducción

En **Python**, todos los datos con los que interactuamos son objetos, tanto si son datos innatos/básicos (builtins) como si son clases creadas por nosotros mismos. Por objetos entendemos que son simplemente trozos de memoria donde se definen valores y conjuntos de operaciones asociadas a esos valores.

Como veremos en otros vídeos y cursos de esta especialización, que todo sea un objeto tiene ciertas ventajas. Y cuando nos referimos a todo, queremos decir absolutamente todo, incluyendo tipos básicos como números (7, 1.41, etc.), cadenas de texto (‘hola mundo’, ‘a’) y hasta operaciones (adición, sustracción, etc.).

Los tipos básicos o builtins son de vital importancia en **Python**. Por ello, en esta unidad, estudiaremos cuales son estos tipos y qué operaciones podemos hacer con ellos.

Los tipos innatos/básicos de Python (Builtins)

Como en el resto de lenguajes de programación, los elementos básicos con los que trabajaremos en Python son las variables. Entendemos variable como un espacio en la memoria donde se guardará un dato que podrá cambiar a lo largo de la ejecución del programa.

Python tiene un sistema de tipos básicos que es muy rico comparado con otros lenguajes como C o C++. En estos lenguajes gran parte del trabajo consiste en crear las estructuras de datos que representan los componentes de la aplicación programada. Además, estas estructuras requieren implementar funciones de búsqueda, acceso, gestión de memoria, etc. Estas tareas, distraen del objetivo principal del programador, además de ser una posible fuente de errores.

Todo esto se puede minimizar cuando el lenguaje en el que trabajamos incluye un sistema de tipos potente, como en el caso de **Python**. Gracias a esto, podemos dedicarnos a resolver nuestro problema directamente, sin tener que implementar antes un sistema de datos adaptado dicho problema. Obviamente, existirán casos en los que esto siga siendo necesario, pero en la mayoría de casos, es preferible aprovechar los tipos básicos de **Python**. A continuación, detallamos algunas razones de porqué ocurre esto:

Los objetos básicos facilitan la tarea de programación

Como acabamos de mencionar, en muchas tareas sencillas, es suficiente con utilizar los tipos builtins. Estos tipos ofrecen herramientas como colecciones (listas, sets), diccionarios, etc. que puedes utilizar inmediatamente y que permiten resolver un gran abanico de problemas.

Los objetos builtins son los ladrillos básicos para objetos más complejos.

En el caso que se requiera programar un software más complejo, puede ser necesario implementar objetos a mano. Como veremos en otras unidades, estos objetos se construyen sobre los tipos básicos de **Python** como listas, tuplas y diccionarios.

Los tipos básicos son muy eficientes.

En la gran mayoría de casos, los tipos builtin son más eficientes que objetos creados por nosotros mismos. Los tipos builtin de **Python** son estructuras de datos que han sido optimizadas y que, interiormente están implementadas en C para aumentar su velocidad.

Los objetos builtin son una parte estándar del lenguaje.

Esto implica que, al estar utilizando componentes estándar del lenguaje, tenemos garantizada la estabilidad, la interoperabilidad entre sistemas, etc. Esto no nos lo puede garantizar ningún componente o framework de terceros.

Como regla básica, considerad siempre utilizar objetos builtin por delante de vuestras propias implementaciones. En una mayoría de casos, esto será suficiente para resolver vuestro problema, facilitará vuestra tarea de programación y hará que vuestro programa sea más mantenible, extensible y eficiente.

Recorrido por los tipos básicos de Python

La próxima tabla nos muestra un listado de los principales tipos básicos (builtins) de **Python** así como algunos ejemplos de los literales que permiten crearlos.

Tipo	Ejemplo de cómo crearlo
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 en la **Tabla 1**, en **Python** hasta las funciones y los módulos son objetos. Esto implica que podemos pasarlos como parámetro a otras funciones, guardarlos como atributos de otros objetos etc.

Un aspecto importante que hay que tener en cuenta: **Python** es un lenguaje de ***tipado dinámico.*** Esto significa que **Python** controla por nosotros el tipo de los objetos, ahorrándonos la tarea de tener que declarar tipos en nuestro código. Pero **Python** es también es un lenguaje de ***tipado fuerte,*** lo que significa que una vez declaramos un objeto, en ese objeto sólo podremos realizar operaciones válidas para ese tipo.

En los vídeos de este tema vamos a hacer un recorrido por estos tipos para ir profundizando en ellos.

Números

Los objetos numéricos builtin de **Python** son los típicos de cualquier lenguaje de programación: números *enteros* y números de *coma flotante* (con decimales) así como otros como *números complejos* con parte imaginaria; decimales de precisión fija; y números racionales con numerador y denominador.

Fijaos que, en la segunda expresión, hemos asignado el resultado en la variable a. En **Python** no es necesario declarar las variables con anterioridad, sino que se crean cuando les asignamos un valor. A las variables en **Python** se les puede asignar cualquier tipo de objeto y será reemplazada con su valor en el momento que aparezca en alguna expresión.

Además de estas expresiones, podemos importar módulos externos para realizar más operaciones matemáticas:

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

>>> math.pi ** 2
9.869604401089358

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

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

Podéis explorar el módulo ***math***, que ya viene incluido en la librería estándar de **Python**, para ver más utilidades y operaciones disponibles.

Otro módulo interesante es el módulo ***random*** que, como su nombre indica, nos permite generar y seleccionar números aleatorios, entre muchas otras opciones.

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

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

>>> random.choice(lista)
3

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

La variable lista que acabamos de declarar no es más que un listado de números. Este es otro de los tipos básicos de **Python** que veremos más adelante. De momento, es suficiente con entender que las listas son colecciones de objetos ordenados.