Python

Guía para ser un Pythonista

Juan José Lozano Gómez – j2logo.com

4

Tipos de datos básicos

Índice de contenidos

Introducción

En cualquier lenguaje de programación de alto nivel se manejan tipos de datos. Un tipo de datos define un conjunto de valores que tienen una serie de características y propiedades determinadas.

Pensemos por un momento cuando éramos jóvenes y estábamos en el colegio en clase de matemáticas. Seguro que diste alguna clase en la que te enseñaban los distintos conjuntos de números. Los naturales (1, 2, 3, 4, ...), los enteros (..., -2, -1, 0, 1, 2, ...), los reales (... -1.1, -0.3, 2.1, ...), etc. Pues bien, en programación (y por supuesto en Python), cada uno de esos conjuntos sería lo que llamamos tipo de datos.

En definitiva, un tipo de dato establece qué valores puede referenciar una variable y qué operaciones se pueden realizar sobre la misma.

En este tema repasaremos los principales tipos de datos básicos de Python y sus características.

Tipos de datos básicos de Python

Python, como otros lenguajes de programación, pone a nuestra disposición diferentes tipos de datos. En este tema repasaremos los tipos de datos básicos, aunque te introduciré otros tipos de datos que veremos en detalle en temas posteriores.

Los tipos de datos básicos de Python son los ***booleanos***, los ***numéricos*** (enteros, punto flotante y complejos) y las ***cadenas de caracteres***.

Además de los anteriores, Python también define otros tipos de datos, entre los que se encuentran:

- **Secuencias**: Los tipos *list*, *tuple* y *range*
- **Mapas**: El tipo *dict*
- **Conjuntos**: El tipo *set*
- **Iteradores**
- **Clases**
- **Instancias**
- **Excepciones**

A su vez, los tipos anteriores se pueden agrupar de diferente manera. Por ejemplo: el tipo cadena de caracteres es también una secuencia inmutable; las listas, tuplas o diccionarios, entre otros, son

contenedores y colecciones, etc. Pero esto no lo veremos aquí.

En fin, no te agobies con tanto tipo ni tanto concepto nuevo. Tómatelo con calma que estás aprendiendo. Comencemos por lo fácil, revisando los tipos de datos básicos de Python.

Tipos numéricos

Python define tres tipos de datos numéricos: **enteros**, **números de punto flotante** (simularía el conjunto de los números reales, pero ya veremos que no es así del todo) y los **números complejos**.

Números enteros

El tipo de los números enteros es `int`. Este tipo de dato comprende el conjunto de todos los números enteros, pero como dicho conjunto es infinito, en Python **el conjunto está limitado realmente por la capacidad de la memoria disponible**. No hay un límite de representación impuesto por el lenguaje.

Pero tranquilidad, que para el 99% de los programas que desarrolles tendrás suficiente con el subconjunto que puedes representar.

Un objeto de tipo `int` se crea a partir de un literal que represente un número entero o bien como resultado de una expresión o una llamada a una función.

Ejemplos:

```
>>> a = -1    # a es de tipo int y referencia al -1
>>> b = a + 2 # b es de tipo int y referencia al 1
>>> print(b)
1
```

Los números enteros también se pueden representar en formato binario, octal o hexadecimal.

Los números octales se crean anteponiendo `0o` a una secuencia de dígitos octales (del 0 al 7).

Para crear un número entero en hexadecimal, hay que anteponer `0x` a una secuencia de dígitos en hexadecimal (del 0 al 9 y de la A la F).

En cuanto a los números en binario, se antepone `0b` a una secuencia de dígitos en binario (0 y 1).

```
>>> diez = 10
>>> diez_binario = 0b1010
>>> diez_octal = 0o12
>>> diez_hex = 0xa
>>> print(diez)
10
>>> print(diez_binario)
10
>>> print(diez_octal)
10
>>> print(diez_hex)
10
```

Números de punto flotante

¿Recuerdas que te dije que los números de punto flotante representaban, más o menos, al conjunto de los números reales?

Voy a tratar de explicarte esto de forma sencilla para que entiendas qué son los números de punto o coma flotante.

Vamos a hacer un experimento que te va a dejar a cuadros. Abre un terminal y ejecuta el comando `python3` para lanzar el intérprete de Python. A

continuación, introduce la expresión `1.1 + 2.2` y mira cuál es el resultado.

```
>>> 1.1 + 2.2
3.3000000000000003
```

¿Por qué el resultado es 3,3000000000000003 en lugar de, simplemente, 3,3? La explicación se debe a cómo se representan internamente este tipo de valores numéricos.

Representación de los números de punto flotante

Tenemos que repasar un poco de teoría que voy a tratar de simplificar porque la explicación completa da para un artículo entero.

Al igual que ocurre con los números enteros, **los números reales son infinitos** y, por tanto, **es imposible representar todo el conjunto de números reales con un ordenador**.

Para representar el mayor número posible de los números reales con las limitaciones de memoria (tamaños de palabra de 32 y 64 bits), se adaptó la

notación científica de representación de números reales al sistema binario (que es el sistema que se utiliza en programación para representar los datos e instrucciones).

En notación científica, los números se representan así:

Número	Notación científica
101,1	$1{,}011 * 10^{2}$
0,032	$3{,}2 * 10^{-2}$

Si te fijas, un número en notación científica está formado por dos partes:

- Una mantisa (también llamada coeficiente o significando) que contiene los dígitos del número. Mantisas negativas representan números negativos.
- Un exponente que indica dónde se coloca el punto decimal (o binario) en relación con el inicio de la mantisa. Exponentes negativos representan números menores que uno.

Por tanto, debido a las limitaciones en el tamaño de palabra de los ordenadores, los números de punto flotante tienen un determinado número de bits fijo para

los valores de la mantisa y para los valores del exponente.

Volviendo al ejemplo del inicio de esta sección, el caso es que la suma de la representación en punto flotante en binario del número 1,1 y de la representación en punto flotante en binario del número 2,2, da como resultado 3,3000000000000003

Pero hay más casos, como por ejemplo la representación del número 1/3. En algún momento, el ordenador tiene que truncar el número periódico resultante.

La explicación final es que los números de punto flotante se representan en el hardware del ordenador como fracciones de base 2 (binarias). Y el problema está en que la mayoría de las fracciones decimales no se pueden representar de forma exacta como fracciones binarias porque tienen infinitos números decimales. Una consecuencia es que, en general, **los números binarios de punto flotante realmente almacenados en la máquina son una aproximación de los números reales que representan**.

Números de Punto flotante en Python

Pues una vez vista esta simplificada introducción a los números de punto flotante, te diré que este tipo de datos en Python se conoce como `float`.

Puedes usar el tipo `float` sin problemas para representar cualquier número real (siempre teniendo en cuenta que es una aproximación lo más precisa posible). Por tanto, para longitudes, pesos, frecuencias, etc. en los que prácticamente es lo mismo 3,3 que 3,3000000000000003, el tipo `float` es el más apropiado.

Al igual que los números enteros, un `float` se crea a partir de un literal, o bien como resultado de una expresión o una función.

```
>>> un_real = 1.1  # El literal debe incluir el carácter .
>>> otro_real = 1/2  # El resultado de 1/2 es un float
>>> not_cient = 1.23E3  # float en notación científica
```

Y para terminar esta sección, te adelanto que, si por cualquier motivo necesitas una precisión exacta a la hora de trabajar con los números reales, Python tiene otros tipos de datos, como `Decimal` o `Fractional`.

El tipo `Decimal` es ideal a la hora de trabajar, por ejemplo, con dinero o tipos de interés. Este tipo de dato trunca la parte decimal del número para ser más preciso. Hablaremos de los tipos de datos `Decimal` y `Fractional` en un tema posterior.

Números complejos

El último tipo de dato numérico básico que tiene Python es el de los números complejos, `complex`.

Como sabrás, los números complejos están formados por una parte *real* y otra *imaginaria*. En Python, cada una de ellas se representa como un `float`.

Para crear un número complejo, se sigue la siguiente estructura `<parte_real> + <parte_imaginaria>j`.

Es posible acceder a la parte real e imaginaria de un número complejo a través de los atributos `real` e `imag`:

```
>>> complejo = 1+2j
>>> complejo.real
1.0
>>> complejo.imag
2.0
```

Aritmética de los tipos numéricos

Con todos los tipos numéricos se pueden aplicar las operaciones de la aritmética: *suma*, *resta*, *producto*, *división*, ...

En Python está permitido realizar una operación aritmética con números de distinto tipo. En este caso, el tipo numérico «más pequeño» se convierte al del tipo «más grande», de manera que el tipo del resultado siempre es el del tipo mayor. Entendemos que el tipo `int` es menor que el tipo `float` que a su vez es menor que el tipo `complex`. A esta conversión de tipos se le conoce como conversión implícita.

Por tanto, es posible, por ejemplo, sumar un `int` y un `float`:

```
>>> 1 + 2.0
3.0
>>> 2+3j + 5.7
(7.7+3j)
```

Tipo booleano

En Python la clase que representa los valores booleanos es `bool`. Esta clase solo se puede instanciar con dos valores/objetos: `True` para representar verdadero y `False` para representar falso.

Una particularidad del lenguaje es que cualquier objeto puede ser usado en un contexto donde se requiera comprobar si algo es verdadero o falso. Por tanto, cualquier objeto se puede usar, por ejemplo, en la condición de un `if` o un `while` (son estructuras de control que veremos en temas posteriores) o como operando de una operación booleana.

Por defecto, cualquier objeto es considerado como verdadero con dos excepciones:

- Que implemente el método `__bool__()` y este devuelva `False`.
- Que implemente el método `__len__()` y este devuelva `0`.

Además, los siguientes objetos/instancias también son consideradas falsas:

- `None`
- `False`

- El valor cero de cualquier tipo numérico: 0, 0.0, 0j, ...
- Secuencias y colecciones vacías (veremos estos tipos en otros temas): `''`, `()`, `[]`, `{}`, `set()`, `range(0)`

Tipo cadena de caracteres

Una vez que hemos acabado con los números, es el turno de las letras.

Otro tipo básico de Python, e imprescindible, son las secuencias o cadenas de caracteres. Este tipo es conocido como `string`, aunque su clase verdadera es `str`.

Formalmente, **un string es una secuencia inmutable de caracteres en formato Unicode**.

Para crear un string, simplemente tienes que encerrar entre comillas simples `''` o dobles `""` una secuencia de caracteres.

Se puede usar indistintamente comillas simples o dobles, con una particularidad. Si en la cadena de caracteres se necesita usar una comilla simple, tienes

dos opciones: usar comillas dobles para encerrar el string, o bien, usar comillas simples, pero anteponer el carácter `\` a la comilla simple del interior de la cadena. El caso contrario es similar.

Veamos todo esto con un ejemplo:

```
>>> hola = 'Hola "Pythonista"'
>>> hola_2 = 'Hola \'Pythonista\''
>>> hola_3 = "Hola 'Pythonista'"
>>> print(hola)
Hola "Pythonista"
>>> print(hola_2)
Hola 'Pythonista'
>>> print(hola_3)
Hola 'Pythonista'
```

A diferencia de otros lenguajes, en Python no existe el tipo «carácter». No obstante, se puede simular con un string de un solo carácter.

En el curso hay todo un tema dedicado al tipo `str`. Revísalo si quieres conocer más sobre él.

Otros tipos

Hasta aquí hemos repasado los tipos de datos básicos de Python, sin embargo, el lenguaje ofrece muchos tipos más. Te hago aquí un avance de los más importantes (algunos los he mencionado anteriormente), aunque los veremos en detalle en otros temas.

Además de los tipos básicos, otros tipos fundamentales de Python son las secuencias (`list` y `tuple`), los conjuntos (`set`) y los mapas (`dict`).

Todos ellos son tipos compuestos y se utilizan para agrupar juntos varios valores.

- Las listas son secuencias mutables de valores.
- Las tuplas son secuencias inmutables de valores.
- Los conjuntos se utilizan para representar conjuntos únicos de elementos, es decir, en un conjunto no pueden existir dos objetos iguales.
- Los diccionarios son tipos especiales de contenedores en los que se puede acceder a sus elementos a partir de una clave única.

```
>>> lista = [1, 2, 3, 8, 9]
>>> tupla = (1, 4, 8, 0, 5)
>>> conjunto = set([1, 3, 1, 4])
>>> diccionario = {'a': 1, 'b': 3, 'z': 8}
>>> print(lista)
[1, 2, 3, 8, 9]
>>> print(tupla)
(1, 4, 8, 0, 5)
>>> print(conjunto)
{1, 3, 4}
>>> print(diccionario)
{'a': 1, 'b': 3, 'z': 8}
```

Conocer el tipo de una variable

Ahora te voy a presentar dos funciones para que puedas jugar con todo lo que hemos visto en este tema. Son `type()` e `isinstance()`:

`type()` recibe como parámetro un objeto y devuelve el tipo del mismo.

`isinstance()` recibe dos parámetros: un objeto y un tipo. Devuelve `True` si el objeto es del tipo que se pasa como parámetro y `False` en caso contrario.

```
>>> type(3)
<class 'int'>
>>> type(2.78)
<class 'float'>
>>> type('Hola')
<class 'str'>
>>> isinstance(3, float)
False
>>> isinstance(3, int)
True
>>> isinstance(3, bool)
False
>>> isinstance(False, bool)
True
```

Conversión de tipos

Lo último que veremos en este tema sobre tipos de datos es la conversión de tipos.

¿Esto qué significa?

Imagina que tienes una variable `edad` de tipo `string` cuyo valor es `'25'`. Se podría decir que `edad`, aunque realmente es una cadena de caracteres, contiene un número. Sin embargo, si intentas sumar `10` a `edad`, el

intérprete te dará un error porque `edad` es de tipo `str` y `10` de un tipo numérico.

```
>>> edad = '25'
>>> edad = edad + 10
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Python es un lenguaje tipado dinámicamente. Esto significa que no hace falta especificar el tipo de una variable (nombre) y que este se conoce en tiempo de ejecución.
Sin embargo, es un lenguaje fuertemente tipado. Esto implica que no se pueden llevar a cabo operaciones con objetos de distinto tipo, a no ser que se realice una conversión de alguno de ellos.

Volviendo al ejemplo, ¿cómo puedo tratar la variable `edad` como un número? Convirtiéndola a un tipo numérico, por ejemplo, al tipo `int`.

Python ofrece las siguientes funciones para llevar a cabo este tipo de conversiones:

- `str()`: Devuelve la representación en cadena de caracteres del objeto que se pasa como parámetro.
- `int()`: Devuelve un `int` a partir de un número o secuencia de caracteres.
- `float()`: Devuelve un `float` a partir de un número o secuencia de caracteres.
- `complex()`: Devuelve un `complex` a partir de un número o secuencia de caracteres.

Si a las funciones anteriores se les pasa como parámetro un valor inválido, el intérprete mostrará un error.

```
>>> edad = int(edad) + 10  # Convierte edad a int
>>> edad  #  edad es un int
35
>>> edad = str(edad)  # Convierte edad a str
>>> edad  # edad es un str (se muestran las '')
'35'
>>> float('18.66')  # Convierte un str a float
18.66
>>> float('hola')  # Error
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: could not convert string to float: 'hola'
```

A este tipo de conversiones se les conoce como conversiones explícitas. Anteriormente hemos visto conversiones implícitas cuando se llevan a cabo operaciones entre objetos de distinto tipo numérico.