Tipos básicos en Python

Antonio Espín Herranz

Tipos

- Tipos de datos numéricos: **int**, **float** y **complex**.
- Las operaciones se adjuntan al tipo de los operandos. Si uno de los dos es real, el resultado será real.
- Enteros: 22, 102, 123, etc.
- Reales: **30.03**, **2e3** => 2.10^3, 2.10^-3
- Complejos = 2.1 + 7.8j

Constructores de tipos

- Tenemos un constructor para cada tipo:
  - Enteros: **int**()
  - Float: **float**()
  - Complejos: **complex**()

- Se pueden utilizar para hacer conversiones cuando leemos de teclado:

```
X = int(input())
Y = float(input())
Z = complex(input())
```

Tipos

- Los enteros no tienen límite de tamaño.
- El límite de los número viene marcado por la memoria de la máquina.
- Podemos realizar este tipo de operaciones:

```
X = 34
Y = 1000
print(X ** Y)
```

Tipos

- El literal que se asigna a la variable también se puede expresar como un **octal**, anteponiendo un cero:

  #027 octal = 23 en base 10

  entero = **0o**27

- O bien en hexadecimal, anteponiendo un 0x:

  # 0×17 **hexadecimal** = 23 en base 10

  entero = **0×**17

Formatear

- También disponemos de los constructores: bin(), oct() y hex() para cambiar de formato un número a binario, octal y hexadecimal.

```
>>> x = 1234
>>> bin(x) '0b10011010010‘
 >>> oct(x) '0o2322'
>>> hex(x) '0x4d2‘
 >>>
```

Formatear

- Alternativamente, podemos utilizar la función format: para formatear la salida a consola:

  >>> format(x, '**b**') '10011010010‘

  >>> format(x, '**o**') '2322'

  >>> format(x, '**x**') '4d2'

- También con los constructores anteriores: int()

- Se puede indicar la base a la que convertir el número:

  print(int('4d2', **16**))

  print(int('10011010010', **2**))

Tipos

- **Reales** representados por **float**.
- Internamente se almacena en el tipo **double de C**.

  ff = 5.678

  type(ff)

  float

- Tipo **complejo** representado
  - Uso científico y matemático.

```
>>> ff  = 5.777
>>> type(ff)
<type 'float'>
>>> cc  = 2.1  + 7j
>>> type(cc)
<type 'complex'>
>>>
```

Tipos

- **Lógicos**: Tipo **bool** . Valores: **True** / **False**.

- Tipo **None**, para indicar nada …, si algo es None y preguntamos en un if será False.

- **Cadenas** (representadas por el tipo **str**)
  - Las cadenas no son más que texto encerrado entre comillas simples (‘cadena’) o dobles (“cadena”). Dentro de las comillas se pueden añadir caracteres especiales escapándolos con \, como \n, el carácter de nueva línea, o \t, el de tabulación.

  - Los caracteres especiales “\...” se interpretan igual en cadenas con comillas simples y dobles.

Tipos

- Una cadena puede estar precedida por el carácter **u** o el carácter **r**, los cuales indican, respectivamente, que se trata de una cadena que utiliza codificación **Unicode** y una cadena ***raw*** *(del inglés, cruda).*

- Las cadenas raw se distinguen de las normales en que los caracteres escapados mediante la barra invertida (\) no se sustituyen por sus contrapartidas. Esto es especialmente útil, por ejemplo, para las expresiones regulares.

- **En python 3 las cadenas son Unicode y no es necesario poner una u delante de la cadena**.

- Ejemplo:
  - unicode = ”äóè”
  - raw = r”\n”

Tipos

- Las cadenas permiten comillas triples.

- En este caso se mantienen los saltos de línea:

  triple = “““primera linea
  esto se vera en otra linea”””

- Soportan el operador + y *
  - a = “uno”
  - b = “dos”
  - c = a + b # c es “unodos”
  - c = a * 3 # c es “unounouno”

Operadores

- Operadores **aritméticos**:

Operador	Descripción	Ejemplo
+	Suma	`r = 3 + 2  # r es 5`
-	Resta	`r = 4 - 7  # r es -3`
-	Negación	`r = -7  # r es -7`
*	Multiplicación	`r = 2 * 6  # r es 12`
**	Exponente	`r = 2 ** 6  # r es 64`
/	División	`r = 3.5 / 2  # r es 1.75`
//	División entera	`r = 3.5 // 2 # r es 1.0`
%	Módulo	`r = 7 % 2  # r es 1`

Operadores

- A nivel de bits:

Operador	Descripción	Ejemplo
&	and	`r = 3 & 2  # r es 2`
|	or	`r = 3 | 2  # r es 3`
^	xor	`r = 3 ^ 2  # r es 1`
~	not	`r = ~3  # r es -4`
<<	Desplazamiento izq.	`r = 3 << 1 # r es 6`
>>	Desplazamiento der.	`r = 3 >> 1 # r es 1`

Operadores sobre bits

- & and sobre bits.
- | or sobre bits.
- ^ xor sobre bits.
- ~ complemento a 1. Cambia ceros por unos.
- >>: rotación a la derecha. Equivalente a dividir por 2^n. Siendo n el número de bits a rotar.
- <<: rotación a la izquierda. Igual que la anterior pero multiplicando.

Operadores

- Lógicos:

Operador	Descripción	Ejemplo
`and`	¿se cumple a y b?	`r = True and False # r es False`
`or`	¿se cumple a o b?	`r = True or False  # r es True`
`not`	No a	`r = not True  # r es False`

Operadores

- Relacionales:

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	`r = 5 == 3 # r es False`
!=	¿son distintos a y b?	`r = 5 != 3 # r es True`
<	¿es a menor que b?	`r = 5 < 3  # r es False`
>	¿es a mayor que b?	`r = 5 > 3  # r es True`
<=	¿es a menor o igual que b?	`r = 5 <= 5 # r es True`
>=	¿es a mayor o igual que b?	`r = 5 >= 3 # r es True`