Tipos de datos básicos en Python

Como tantos otros lenguajes de programación, Python ofrece, de entrada, los tipos de datos enteros, reales, booleanos y cadenas de caracteres. También permite el manejo de números complejos sin importar librería alguna. Veamos algunas operaciones básicas con estos tipos de datos.

Enteros

Las operaciones típicas entre enteros son la suma, resta y multiplicación, respetando la prioridad de esta última. Para el caso en que deseemos una prioridad distinta, podemos colocar paréntesis donde sea necesario. También ofrece una operación para calcular potencias, con una prioridad superior aún. He aquí un ejemplo de todas estas cosas juntas:

In [1]:
```
(3+7, 3-7, 3 + 2*7, (3+2)*7, 1 + 2 * 3**2)
```

Out[1]:
```
(10, -4, 17, 35, 19)
```

La división estándar `/` no da lugar a un entero sino a un real. Por eso, existe una división entera `//`, que trunca el resultado al entero inferior, y una operación (`%`) para calcular el resto de esta división:

In [2]:
```
17 / 5, 17 // 5, 17 % 5
```

Out[2]:
```
(3.4, 3, 2)
```

Observa que las tres expresiones anteriores están agrupadas pero esta vez sin paréntesis, aunque el resultado es igualmente una *tupla*, como el caso precedente.

También, observa que la división entera trunca el entero resultante al entero *precedente*, que al ser negativo lo aleja del cero, siendo el resto siempre del signo del divisor:

In [3]:
```
print(37 / 5, 37 // 5, 37 % 5)
print(-37 / 5, -37 // 5, -37 % 5)
print(37 / -5, 37 // -5, 37 % -5)
print(-37 / -5, -37 // -5, -37 % -5)
```

```
7.4 7 2
-7.4 -8 3
-7.4 -8 -3
7.4 7 -2
```

In [4]:
```
# Podríamos haber trabajado igualmente con variables:
dividendo = 17
divisor   = 5
cociente  = dividendo // divisor
resto     = dividendo % divisor

# Y sería bonito comprobar que el dividendo es igual a divisor * cocient

dividendo, divisor * cociente + resto
```

Out[4]: (17, 17)

Como hemos visto, las *variables* en Python se pueden introducir sin declarar el tipo de datos. Pero siempre es posible averiguar el tipo de datos que tiene una variable, o un dato:

In [5]:
```
x = (4 * 9) / 6
y = (4 * 9) // 6
type(x), type(y), type(x+y)
```

Out[5]: (float, int, float)

Aunque el resultado de `(4 * 9) / 6` es el entero `6` matemáticamente, la operación de división `/` genera en Python un resultado de tipo real, y por eso hemos obtenido el tipo `float`, a diferencia del tipo generado por la operación `//`.

Seguidamente, vamos a definir una *función* por primera vez en Python. Deseamos calcular el valor del polinomio $-6 - 4x + 3x^2$, para cualquier valor de la variable x:

In [6]:
```
def poli(x):
    c0 = -6              # término independiente
    c1 = -4 * x          # término de primer grado
    c2 = 3 * x**2        # término de segundo grado
    return c2 + c1 + c0  # valor del polinomio

poli(5)
```

Out[6]: 49

In [7]:
```
# Lo mismo, pero ahora la potencia se va recalculando, seguramente es un

def poli(x):
    s = 0
    pot = 1              # x ** 0
    s = s + (-6 * pot)
    pot = pot * x        # x ** 1
    s = s + (-4 * pot)
    pot = pot * x        # x ** 2
    s = s + (3 * pot)
    pot = pot * x        # x ** 3, no se usa, pero logramos uniformidad p
    return s

poli(5)
```

Out[7]: 49

In [8]:
```
# Lo mismo, pero ahora lo hemos expresado con un bucle:

def poli(x):
    s = 0
    pot = 1             # x ** 0
    for coef in [-6, -4, 3]:
        s = s + (coef * pot)
        pot = pot * x       # x ** i, para i <- [0, 1, 2, 3]
    return s

poli(5)
```

Out[8]: 49

In [9]:
```
# Lo mismo, pero ahora lo hemos expresado con el polinomio como parámetr

def poli(x, coeficientes):
    s = 0
    pot = 1             # x ** 0
    for coef in coeficientes:
        s = s + (coef * pot)
        pot = pot * x       # x ** i, para i <- [0, 1, 2, 3]
    return s

poli(5, [-6, -4, 3])
```

Out[9]: 49

Reales, aritmética de coma flotante

In [10]:
```
# El punto indica la coma decimal:

a = 2.0
b = 3.0

# Asignamos dos (o más) variables al mismo tiempo, con la notación de tu
suma, resta, multi, divi = a + b, a - b, a * b, a / b
suma, resta, multi, divi
```

Out[10]: (5.0, -1.0, 6.0, 0.6666666666666666)

In [11]:
```
# Al multiplicar un entero por un real, el resultado es un real, automát

a * 7, type(a * 7)
```

Out[11]: (14.0, float)

In [12]:
```
# Aunque introduzcamos un entero, el resultado va a ser un real:

def poli(x):
    return -6.0 + -4.0 * x + 3.0 * x**2

poli(5)
```

Out[12]: 49.0

In [13]:
```
# Podemos usar muchas funciones matemáticas:

import math
radio = 3
print(2 * radio * math.pi) # math.pi es un nombre que contiene el número
print(math.sin(math.pi/3)) # math.sin, el seno del ángulo en radianes
print(math.cos(math.pi/3)) # math.cos, el coseno. Obsérverse el error ob
math.sqrt(5) # la raíz cuadrada
```

```
18.84955592153876
0.8660254037844386
0.5000000000000001
```

Out[13]:
```
2.23606797749979
```

Existen distintas operacionse para pasar de real a entero:

In [14]:
```
# 'int' construye un entero truncando la parte decimal:
print(int(3.9), int(-3.9))

# 'round' redondea al entero más próximo:
print(round(3.2), round(3.5), round(-3.5), round(-3.2))

# 'math.floor' redondea al entero inferior:
print(math.floor(3.7), math.floor(-3.2), type(math.floor(3.7)))

# 'math.ceil' redondea al entero siguiente:
print(math.ceil(3.2), math.ceil(-3.2))
```

```
3 -3
3 4 -4 -3
3 -4 <class 'int'>
4 -3
```

Complejos

In [15]:
```
z = complex(2, 4) # Números complejos
z**2
```

Out[15]:
```
(-12+16j)
```

In [16]:
```
# La unidad imaginaria es j
(2+3j)**2 # otra notación
```

Out[16]:
```
(-5+12j)
```

En la librería `cmath` se encuentran las versiones complejas de las funciones de `math` :

In [17]:
```
import cmath
cmath.sqrt(-1)
```

Out[17]:
```
1j
```

Booleanos

In [18]:
```
a = True  # valor lógico de cierto
b = False # valor lógico de falso
```

In [19]:
```
# conjunción lógica (and) y disyunción lógica (or):
a and b, a or b, a or True, b and False
```

Out[19]: (False, True, True, False)

In [20]:
```
a, b = 2, 4
b == a * 2 # la comparación devuelve un valor lógico
```

Out[20]: True

In [21]:
```
# Otros operadores relacionales:
34 != 34, not (34 == 34), a * 3 >= 7, b * 2 < 8, b * 2 <= 8, a >= 2 or b
```

Out[21]: (False, False, False, False, True, True)

In [22]:
```
# Dos maneras de expresar lo mismo en Python:
2 <= a and a < 7, 2 <= a < 7
```

Out[22]: (True, True)

In [23]:
```
def is_in_circle(x, y, r):
    # Averigua si el punto (x, y) dista de (0, 0) a lo más, r
    return x**2 + y**2 <= r*r

is_in_circle(0.5, 0.5, 1), is_in_circle(0.5, 0.9, 1)
```

Out[23]: (True, False)

Lo siguiente se llama **expresión condicional**. No debe confundirse con la instrucción condicional, que se verá más adelante.

In [24]:
```
def is_in_circle(x, y, r):
    # Averigua si el punto (x, y) dista de (0, 0) a lo más, r
    return "Dentro" if x**2 + y**2 <= r*r else "Fuera"

is_in_circle(0.5, 0.5, 1), is_in_circle(0.5, 0.9, 1)
```

Out[24]: ('Dentro', 'Fuera')

Cadenas de caracteres

Las cadenas de caracteres se pueden poner con comillas dobles o simples.

In [25]:
```
a = "Hola"
b = 'Hola'
a, b, a == b
```

Out[25]: ('Hola', 'Hola', True)

In [26]:
```
nombre = "Juan"
saludo = "Hola " + nombre # + es la concatenación de cadenas de caracter
saludo
```

Out[26]: 'Hola Juan'

In [27]:
```
"H" * 10 # replicación
```

Out[27]: 'HHHHHHHHHH'

Las cadenas de caracteres se comparan según el orden alfabético, siendo las mayúsculas y las minúsculas distintas, y concretamente, siendo las mayúsculas 'anteriores' a las minúsculas:

In [28]:
```
"Juan" < "Juap", "j" == "J", "J" < "j"
```

Out[28]: (True, False, True)

In [29]:
```
len("Hola") # la longitud de una longitud de cadena de caracteres
```

Out[29]: 4

Cadenas de caracteres largas

In [30]:
```
quijote = """En un lugar de la Mancha, de cuyo nombre no quiero acordarm
no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero,
adarga antigua, rocín flaco y galgo corredor.
Una olla de algo más vaca que carnero, salpicón las más noches,
duelos y quebrantos los sábados, lantejas los viernes, algún palomino de
consumían las tres partes de su hacienda.
El resto della concluían sayo de velarte, calzas de velludo para las fie
y los días de entresemana se honraba con su vellorí de lo más fino.
Tenía en su casa una ama que pasaba de los cuarenta, y una sobrina que n
y un mozo de campo y plaza, que así ensillaba el rocín como tomaba la po
Frisaba la edad de nuestro hidalgo con los cincuenta años; era de comple
enjuto de rostro, gran madrugador y amigo de la caza.
Quieren decir que tenía el sobrenombre de Quijada, o Quesada, que en est
aunque, por conjeturas verosímiles, se deja entender que se llamaba Quej
Pero esto importa poco a nuestro cuento; basta que en la narración dél n
```

In [31]:
```
print(quijote)
```

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero,
adarga antigua, rocín flaco y galgo corredor.
Una olla de algo más vaca que carnero, salpicón las más noches,
duelos y quebrantos los sábados, lantejas los viernes, algún palomino d
e añadidura los domingos,
consumían las tres partes de su hacienda.
El resto della concluían sayo de velarte, calzas de velludo para las fi
estas, con sus pantuflos de lo mesmo,
y los días de entresemana se honraba con su vellorí de lo más fino.
Tenía en su casa una ama que pasaba de los cuarenta, y una sobrina que
no llegaba a los veinte,
y un mozo de campo y plaza, que así ensillaba el rocín como tomaba la p
odadera.
Frisaba la edad de nuestro hidalgo con los cincuenta años; era de compl
exión recia, seco de carnes,
enjuto de rostro, gran madrugador y amigo de la caza.
Quieren decir que tenía el sobrenombre de Quijada, o Quesada, que en es
to hay alguna diferencia en los autores que deste caso escriben;
aunque, por conjeturas verosímiles, se deja entender que se llamaba Que
jana.
Pero esto importa poco a nuestro cuento; basta que en la narración dél
no se salga un punto de la verdad.
```

Caracteres especiales

In [32]:
```
especial1 = "Las \"comillas\" son caracteres especiales"
especial1
```

Out[32]:
```
'Las "comillas" son caracteres especiales'
```

In [33]:
```
especial2 = "Las \'comillas\' son caracteres especiales"
especial2
```

Out[33]:
```
"Las 'comillas' son caracteres especiales"
```

In [34]:
```
especial3 = "Por tanto la barra invertida también lo es \\"
especial3
```

Out[34]:
```
'Por tanto la barra invertida también lo es \\'
```

In [35]:
```
especial4 = "Hay otros caracteres especiales como el salto de línea.\nQu
print(especial4)
```

```
Hay otros caracteres especiales como el salto de línea.
Que sirve para representar el fin de línea en ficheros.
Lo veremos más adelante
```

Errores comunes

El siguiente ejemplo te invita a reflexionar sobre la precisión de las operaciones y sobre la comparación de números reales.

In [36]:
```
def paradoja(a):
    b = math.sqrt(a)
    return a == b * b # esto no siempre es cierto,
                      # por el error de la representación de los reales.

paradoja(2)           # ... Por ejemplo, aquí
```

Out[36]: False

In [37]:
```
# Otro ejemplo en la misma línea:
# teóricamente, esto también debería ser cierto.

math.cos(math.pi/3) == 0.5
```

Out[37]: False

In [38]:
```
# Y otro más. Lo siguiente, matemáticamente,
# debería dar como resultado un entero:

def fibonacci(n):
    phi = (1 + math.sqrt(5)) / 2
    return (phi ** n - (1 - phi) ** n)/math.sqrt(5)

fibonacci(4)
```

Out[38]: 3.0000000000000004

In [39]:
```
# Una solución sencilla consiste en redondear:

def fibonacci(n):
    phi = (1 + math.sqrt(5)) / 2
    return int(round((phi ** n - (1 - phi) ** n)/math.sqrt(5)))

fibonacci(4)
```

Out[39]: 3

Referencias

Damos una sola referencia, una sola, pero importante y útil:

- https://www.w3schools.com/python/ (https://www.w3schools.com/python/)

Pero nuestra recomendación es mirar, en el menú de la izquierda de esta referencia, y seleccionar del mismo las páginas *Python variables*, *Python numbers*, *Python casting*, *Python strings* y *Python operators*. En ellas se repasa y se amplía suavemente lo que hemos visto en esta pieza.