Capítulo 1: Trabajar con números

### **Diferentes tipos de números**

Te habrás dado cuenta de que he utilizado dos tipos de números para demostrar las operaciones matemáticas: números sin punto decimal, que ya conoces como *números enteros*, y números con punto decimal, que los programadores llaman *números de coma flotante*. A los humanos no nos cuesta reconocer y trabajar con números, ya estén escritos como números enteros, decimales con coma flotante, fracciones o números romanos. Pero en algunos de los programas que escribimos en este libro, sólo tendrá sentido realizar una tarea con un tipo concreto de número, por lo que a menudo tendremos que escribir un poco de código para que los programas comprueben si los números que introducimos son del tipo correcto.

Python considera que los números enteros y los de coma flotante son *tipos* diferentes. Si utilizas la función type(), Python te dirá qué tipo de número acabas de introducir. Por ejemplo:

>>> type(3)  
  
  
>>> type(3.5)  
  
  
>>> type(3.0)

Aquí puedes ver que Python clasifica el número 3 como un número entero (tipo 'int'), pero clasifica 3,0 como un número de coma flotante (tipo 'float'). Todos sabemos que 3 y 3,0 son matemáticamente equivalentes, pero en muchas situaciones, Python tratará estos dos números de forma diferente porque son dos tipos distintos.

Algunos de los programas que escribimos en este capítulo sólo funcionarán correctamente con un número entero como entrada. Como acabamos de ver, Python no reconocerá un número como 1,0 o 4,0 como un entero, así que si queremos aceptar números como esos como entrada válida en estos programas, tendremos que convertirlos de números de coma flotante a enteros. Por suerte, hay una función integrada en Python que hace precisamente eso:

>>> int(3.8)  
3  
>>> int(3.0)  
3

La función int() toma el número de coma flotante de entrada, elimina todo lo que venga después del punto decimal y devuelve el número entero resultante. La función float() funciona de forma similar para realizar la conversión inversa:

>>> float(3)  
3.0

float() toma el número entero de entrada y le añade un punto decimal para convertirlo en un número de coma flotante.

#### ***Trabajar con fracciones***

Python también puede manejar fracciones, pero para ello necesitaremos utilizar el módulo fractions de Python. Puedes considerar un *módulo* como un programa escrito por otra persona que puedes utilizar en tus propios programas. Un módulo puede incluir clases, funciones e incluso definiciones de etiquetas. Puede formar parte de la biblioteca estándar de Python o distribuirse desde una ubicación de terceros. En este último caso, tendrías que instalar el módulo antes de poder utilizarlo.

El módulo fractions forma parte de la biblioteca estándar, lo que significa que ya está instalado. Define una clase Fraction, que es la que utilizaremos para introducir fracciones en nuestros programas. Antes de poder utilizarla, tendremos que *importarla*, que es una forma de decirle a Python que queremos utilizar la clase de este módulo. Veamos un ejemplo rápido: crearemos una nueva etiqueta, f, que hace referencia a la fracción 3/4:

➊ >>> from fractions import Fraction  
➋ >>> f = Fraction(3, 4)  
➌ >>> f  
Fraction(3, 4)

Primero importamos la clase Fraction del módulo fractions ➊. A continuación, creamos un objeto de esta clase pasando como parámetros el numerador y el denominador ➋. Esto crea un objeto Fraction para la fracción 3/4. Cuando imprimimos el objeto ➌, Python muestra la fracción de la forma Fraction(numerator, denominator).

Las operaciones matemáticas básicas, incluidas las operaciones de comparación, son todas válidas para las fracciones. También puedes combinar una fracción, un número entero y un número de coma flotante en una misma expresión:

>>> Fraction(3, 4) + 1 + 1.5  
3.25

Cuando tienes un número de coma flotante en una expresión, el resultado de la expresión se devuelve como un número de coma flotante.

En cambio, cuando sólo tienes una fracción y un número entero en la expresión, el resultado es una fracción, aunque el resultado tenga un denominador de 1.

>>> Fraction(3, 4) + 1 + Fraction(1/4)  
Fraction(2, 1)

Ahora ya conoces lo básico para trabajar con fracciones en Python. Pasemos a otro tipo de números.

#### ***Números complejos***

Los números que hemos visto hasta ahora son los llamados *números reales*. Python también admite números *complejos* con la parte imaginaria identificada por la letra *j* o *J* (a diferencia de la letra *i* utilizada en notación matemática). Por ejemplo, el número complejo 2 + *3i* se escribiría en Python como 2 + *3j*:

>>> a = 2 + 3j  
>>> type(a)

Como puedes ver, cuando utilizamos la función type() sobre un número complejo, Python nos dice que se trata de un objeto del tipo complex.

También puedes definir números complejos utilizando la función complex():

>>> a = complex(2, 3)  
>>> a  
(2 + 3j)

Aquí pasamos las partes real e imaginaria del número complejo como dos argumentos a la función complex(), y ésta devuelve un número complejo.

Puedes sumar y restar números complejos del mismo modo que los números reales:

>>> b = 3 + 3j  
>>> a + b  
(5 + 6j)  
>>> a - b  
(-1 + 0j)

La multiplicación y la división de números complejos también se realizan de forma similar:

>>> a \* b  
(-3 + 15j)  
>>> a / b  
(0.8333333333333334 + 0.16666666666666666j)

Las operaciones de módulo (%) y división por el suelo (//) no son válidas para los números complejos.

Las partes real e imaginaria de un número complejo pueden obtenerse utilizando sus atributos real y imag, como se indica a continuación:

>>> z = 2 + 3j  
>>> z.real  
2.0  
>>> z.imag  
3.0

El *conjugado* de un número complejo tiene la misma parte real pero una parte imaginaria de igual magnitud y signo contrario. Se puede obtener utilizando el método conjugate():

>>> z.conjugate()  
(2 - 3j)

Tanto la parte real como la imaginaria son números en coma flotante. Utilizando las partes real e imaginaria, puedes calcular la *magnitud* de un número complejo con la siguiente fórmula, donde *x* e *y* son las partes real e imaginaria del número, respectivamente: image. En Python, sería como sigue:

>>> (z.real \*\* 2 + z.imag \*\* 2) \*\* 0.5  
3.605551275463989

Una forma más sencilla de hallar la magnitud de un número complejo es con la función abs(). La función abs() devuelve el valor absoluto cuando se invoca con un número real como argumento. Por ejemplo, tanto abs(5) como abs(-5) devuelven 5. Sin embargo, para los números complejos, devuelve la magnitud:

>>> abs(z)  
3.605551275463989

El módulo cmath de la biblioteca estándar (cmath para *matemáticas complejas*) proporciona acceso a otras funciones especializadas para trabajar con números complejos.

[anterior](ch01_3.html)[Subtema 4 de 8: (Ver todo)](ch01.html)[siguiente](ch01_5.html)