NÚMEROS

Como en cualquier lenguaje de programación, en Python, la representación y el manejo de números se hacen prácticamente imprescindibles. Para trabajar con ellos, Python cuenta con una serie de tipos y operaciones integradas, de ambos nos ocuparemos en el presente apartado.

Enteros, reales y complejos

Respecto a los tipos de números soportados por Python, contamos con que es posible trabajar con números enteros, reales y complejos. Además, estos pueden ser representados en decimal, binario, octal y hexadecimal, tal y como veremos más adelante.

De forma práctica, la asignación de un número entero a una variable se puede hacer a través de una sentencia como esta:

```
>>> num_entero = 8
```

Por supuesto, en el contexto de los números enteros, la siguiente expresión también es válida:

```
>>> num_negativo = -78
```

Por otro lado, un número real se asignaría de la siguiente forma:

```
>>> num_real = 4.5
```

En lo que respecta a los números complejos, aquellos formados por una parte *real* y otra *imaginaria*, la asignación sería la siguiente:

```
>>> num_complejo = 3.2 + 7j
```

Siendo también válida la siguiente expresión:

```
>>> num_complex = 5J + 3
```

Como el lector habrá deducido, en los números complejos, la parte imaginaria aparece representada por la letra *j*, siendo también posible emplear la misma letra en mayúscula.

Python 2.x distingue entre dos tipos de enteros en función del tamaño del valor que representan. Concretamente, tenemos los tipos *int* y *long*. En Python 3 esta situación ha cambiado y ambos han sido integrados en un único tipo *int*.

Los valores para números reales que podemos utilizar en Python 3 tienen un amplio rango, gracias a que el lenguaje emplea para su representación un bit para el signo (positivo o negativo), 11 para el exponente y 52 para la mantisa. Esto también implica que se utiliza la precisión doble. Recordemos que en algunos lenguajes de programación se emplean dos tipos de datos para los reales, que varían en función de la representación de su precisión. Es el caso de C, que cuenta con el tipo *float* y *double*. En Python no existe esta distinción y podemos considerar que los números reales representados equivalen al tipo *double* de C.

Además de expresar un número real tal y como hemos visto previamente, también es posible hacerlo utilizando notación científica. Simplemente necesitamos añadir la letra *e*, que representa el exponente, seguida del valor para él mismo. Teniendo esto en cuenta, la siguiente expresión sería válida para representar al número $0.5*10^{-7}$:

```
>>> num_real = 0.5e-7
```

Desde un punto de vista escrito los *booleanos* no son propiamente números; sin embargo, estos solo pueden tomar dos valores diferentes: *True* (verdadero) o *False* (falso). Dada esta circunstancia, parece lógico utilizar un tipo entero que necesite menos espacio en memoria que el original, ya que, solo necesitamos dos números: 0 y 1. En realidad, aunque Python cuenta con el tipo integrado *bool*, este no es más que una versión personalizada del tipo *int*.

Si necesitamos trabajar con números reales que tengan una precisión determinada, por ejemplo, dos cifras decimales, podemos utilizar la clase *Decimal*. Esta viene integrada en la librería básica que ofrece el intérprete e incluye una serie de funciones, para, por ejemplo, crear un número real con precisión a través de un variable de tipo *float*.

Operadores

Obviamente, para trabajar con números, no solo necesitamos representarlos a través de diferentes tipos, sino también es importante realizar operaciones con ellos. Python cuenta con diversos operadores para aplicar diferentes operaciones numéricas. Dentro del grupo de las aritméticas, contamos con las básicas suma, división entera y real, multiplicación y resta. En cuanto a las operaciones de bajo nivel y entre bits, existen tanto las operaciones NOT y NOR, como XOR y AND. También contamos con operadores para comprobar la igualdad y desigualdad y para realizar operaciones lógicas como AND y OR.

Como en otros lenguajes de programación, en Python también existe la precedencia de operadores, lo que deberemos tener en cuenta a la hora de escribir expresiones que utilicen varios de ellos. Sin olvidar que los paréntesis pueden ser usados para marcar la preferencia entre unas operaciones y otras dentro de la misma expresión.

La tabla 2-1 resume los principales operadores y operaciones numéricas a las que hacen referencia, siendo *a* y *b* dos variables numéricas.

Expresión con operador	Operación
a + b	Suma
a – b	Resta
a * b	Multiplicación
a % b	Resto
a / b	División real
a // b	División entera
a ** b	Potencia
a | b	OR (bit)
a ^ b	XOR (bit)
a & b	AND (bit)
a == b	Igualdad
a != b	Desigualdad
a or b	OR (lógica)
a and b	AND (lógica)
not a	Negación (lógica)

Tabla 2-1. Principales operaciones y operadores numéricos

Funciones matemáticas

A parte de las operaciones numéricas básicas, anteriormente mencionadas, Python nos permite aplicar otras muchas funciones matemáticas. Entre ellas, tenemos algunas como el valor absoluto, la raíz cuadrada, el cálculo del valor máximo y mínimo de una lista o el redondo para números reales. Incluso es posible trabajar con operaciones trigonométricas como el seno, coseno y tangente. La mayoría de estas operaciones se encuentran disponibles a través de un módulo (ver definición en capítulo 3) llamado *math*. Por ejemplo, el valor absoluto del número -47,67 puede ser calculado de la siguiente forma:

```
>>> abs(-47,67)
```

Para algunas operaciones necesitaremos importar el mencionado módulo *math*, sirva como ejemplo el siguiente código para calcular la raíz cuadrada del número 169:

```
>>> import math
>>> math.sqrt(169)
```

Otras interesantes operaciones que podemos hacer con números es el cambio de base. Por ejemplo, para pasar de decimal a binario o de octal a hexadecimal. Para ello, Python cuenta con las funciones *int()*, *hex()*, *oct()* y *bin()*. La siguiente sentencia muestra cómo obtener en hexadecimal el valor del entero 16:

```
>>> hex(16)
'0x10'
```

Si lo que necesitamos es el valor octal, por ejemplo, del número 8, bastará con lanzar la siguiente sentencia:

```
>>> oct(8)
'0o10'
```

Debemos tener en cuenta que las funciones de cambio de base admiten como argumentos cualquier representación numérica admitida por Python. Esto quiere decir, que la siguiente expresión también sería válida:

```
>>> bin(0xfe)

'0b11111110'
```

CADENAS DE TEXTO

No cabe duda de que, a parte de los números, las cadenas de texto (*strings*) son otro de los tipos de datos más utilizados en programación. El intérprete de Python integra este tipo de datos, además de una extensa serie de funciones para interactuar con diferentes cadenas de texto.

En algunos lenguajes de programación, como en C, las cadenas de texto no son un tipo integrado como tal en el lenguaje. Esto implica un poco de trabajo extra a la hora de realizar operaciones como la concatenación. Sin embargo, esto no ocurre en Python, lo que hace mucho más sencillo definir y operar con este tipo de dato.

Básicamente, una cadena de texto o *string* es un conjunto inmutable y ordenado de caracteres. Para su representación y definición se pueden utilizar tanto comillas dobles ("), como simples ('). Por ejemplo, en Python, la siguiente sentencia crearía una nueva variable de tipo *string*:


```
>>> cadena = "esto es una cadena de texto"
```

Si necesitamos declarar un string que contenga más de una línea, podemos hacerlo utilizando comillas triples en lugar de dobles o simples:

```
>>> cad_multiple = """Esta cadena de texto
... tiene más de una línea. En concreto, cuenta

... con tres líneas diferentes"""
```