Enteros y Flotantes

Cada valor utilizado por Python es de un tipo determinado. Los datos de tipo entero son los números sin decimales. Si deseamos trabajar con datos de tipo real, deberemos usar operandos reales. Los operandos reales deben llevar, en principio, una parte decimal, aunque ésta sea nula.

```
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 3 / 2
1.5
>>> a= 3
>>> b = 2
>>> a/b
1.5
>>> print(type(a))
<class 'int'>
>>> c = a/b
>>> print(type(c))
<class 'float'>
>>>
```

```
>>> 3e3 / 10
300.0
>>> a = 3e3
>>> a
3000.0
>>>
>>> 4.0 ** (1.0 / 2) + 1 / 2.0
2.5
>>>
```

Hay diferencias entre enteros y reales en Python, mas allá de que los primeros no tengan decimales y los segundos sí. El número 3 y el número 3.0, por ejemplo, son indistinguibles en matemáticas, pero sí son diferentes en Python. ¿Qué diferencia hay?

- Los enteros suelen ocupar menos memoria.
- Las operaciones entre enteros son, generalmente, más rápidas.

Cuando hablamos de números con decimales, el término “reales” no es adecuado, ya que induce a pensar en los números reales de las matemáticas cuando en una computadora es imposible representar infinitos decimales, y trabajamos con aproximaciones a los números reales. Para facilitar el intercambio de datos, todas las computadoras convencionales utilizan la misma codificación, el Estándar IEEE 754 para coma flotante.

Un número flotante debe especificarse siguiendo ciertas reglas. En principio, consta de dos partes: mantisa y exponente. El exponente se separa de la mantisa con la letra E, por ejemplo, el número flotante 2e3 (o 2E3) tiene mantisa 2 y exponente 3, y representa al número $2 \cdot 10^3$, es decir, 2000.

El exponente puede ser negativo: 3.2e-3 es $3.2 \cdot 10^{-3}$, o sea, 0.0032. Hay que tener en cuenta que si un número flotante no lleva exponente, debe llevar parte fraccionaria.

¡Ah! Un par de reglas más: si la parte entera del número es nula, el flotante puede empezar directamente con un punto, y si la parte fraccionaria es nula, puede acabar con un punto. Veamos un par de ejemplos: el número 0.1 se puede escribir también como .1; por otra parte, el número 2.0 puede escribirse como 2., es decir, en ambos casos el cero es opcional.

Cuando necesitamos representar un número muy grande, podemos usar los valores **+infinito** y **-infinito**, que son el número más grande (o mas chico) que podemos representar en la computadora, ¡aunque no sepamos cuál es ese número!

```
>>> a=float('inf')
>>> b=float("Infinity")
>>> a>b
False
>>> b>a
False
>>> a==b
True
```

Números Complejos

Python permite definir números complejos, los cuales tienen parte real y parte imaginaria, que son números en coma flotante.

Utilizamos “j” o “J” para cargar la parte imaginaria de un número complejo, o para obtener un complejo a partir de un número en coma flotante. Para obtener la parte real o la parte imaginaria de un número complejo z, usamos z.real y z.imag.

```
>>> c1 = 3 + 0j
>>> c2 = 6J
>>> c3 = c1+c2
>>> c3
(3+6j)
>>> a = 10 + c3.real
>>> a
13.0
>>> b = 99 - c3.imag
>>> b
93.0
```

Valores Lógicos

Desde la versión 2.3, Python ofrece un tipo de datos especial que permite expresar sólo dos valores: cierto y falso. El valor cierto se expresa con True y el valor falso con False. Son los valores lógicos o booleanos, en realidad un subtipo del tipo entero.

La denominación booleanos deriva del nombre de un matemático, Boole, que desarrolló un sistema algebraico basado en estos dos valores y tres operaciones: la conjunción, la disyunción y la negación. Python ofrece soporte para estas operaciones con los operadores lógicos.

```
>>> lo1 = True
>>> lo2 = False
>>> lo1 < lo2
False
>>> lo1 > lo2
True
>>> lo1 == True
True
>>> lo1 == False
False
```

En la tabla siguiente se resumen los operadores de Python con su aridad (número de operandos), asociatividad y precedencia. El nivel de precedencia 1 es el de mayor prioridad y el 4, el de menor prioridad.

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4
Igual que	==	Binario	—	5
Distinto de	!=	Binario	—	5
Menor que	<	Binario	—	5
Menor o igual que	<=	Binario	—	5
Mayor que	>	Binario	—	5
Mayor o Igual que	>=	Binario	—	5
Negación	**not**	Unario	—	6
Conjunción	**and**	Binario	Por la izquierda	7
Disyunción	**or**	Binario	Por la izquierda	8

En https://docs.python.org/3/library/stdtypes.html#typesnumeric podemos encontrar la descripción completa de los Tipos de Datos predefinidos en Python 3.