2.7. Funciones definidas en módulos

Python también proporciona funciones trigonométricas, logaritmos, etc., pero no están directamente disponibles cuando iniciamos una sesión. Antes de utilizarlas hemos de indicar a Python que vamos a hacerlo. Para ello, *importamos* cada función de un módulo.

2.7.1. El módulo *math*

Empezaremos por importar la función seno (*sin*, del inglés «sinus») del módulo matemático (*math*):

```
>>> from math import sin ↲
```

Ahora podemos utilizar la función en nuestros cálculos:

```
>>> sin(0) ↲
0.0
>>> sin(1) ↲
0.841470984808
```

Observa que el argumento de la función seno debe expresarse en radianes.

Inicialmente Python no «sabe» calcular la función seno. Cuando importamos una función, Python «aprende» su definición y nos permite utilizarla. Las definiciones de funciones residen en *módulos*. Las funciones trigonométricas residen en el módulo matemático. Por ejemplo, la función coseno, en este momento, es desconocida para Python.

```
>>> cos(0) ↲
Traceback (innermost last):
  File "<stdin>", line 1, in ?
NameError: cos
```

Antes de usarla, es necesario importarla del módulo matemático:

```
>>> from math import cos ↲
>>> cos(0) ↲
1.0
```

En una misma sentencia podemos importar más de una función. Basta con separar sus nombres con comas:

```
>>> from math import sin, cos ↲
```

Puede resultar tedioso importar un gran número de funciones y variables de un módulo. Python ofrece un atajo: si utilizamos un asterisco, se importan *todos* los elementos de un módulo. Para importar todas las funciones del módulo *math* escribimos:

```
>>> from math import * ↲
```

Así de fácil. De todos modos, no resulta muy aconsejable por dos razones:

- Al importar elemento a elemento, el programa gana en legibilidad, pues sabemos de dónde proviene cada identificador.
- Si hemos definido una variable con un nombre determinado y dicho nombre coincide con el de una función definida en un módulo, nuestra variable será sustituida por la función. Si no sabes todos los elementos que define un módulo, es posible que esta coincidencia de nombre tenga lugar, te pase inadvertida inicialmente y te lleves una sorpresa cuando intentes usar la variable.

He aquí un ejemplo del segundo de los problemas indicados:

```
>>> pow = 1 ↲
>>> from math import * ↲
>>> pow += 1 ↲
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: unsupported operand type(s) for +=: 'builtin_function_or_method'
and 'int'
```

Python se queja de que intentamos sumar un entero y una función. Efectivamente, hay una función *pow* en el módulo *math*. Al importar todo el contenido de *math*, nuestra variable ha sido «machacada» por la función.

Te presentamos algunas de las funciones que encontrarás en el módulo matemático:

Evitando las coincidencias

Python ofrece un modo de evitar el problema de las coincidencias: importar sólo el módulo.

```
>>> import math ↲
```

De esta forma, todas las funciones del módulo *math* están disponibles, pero usando el nombre del módulo y un punto como prefijo:

```
>>> import math ↲
>>> print math.sin(0) ↲
0.0
```

sin(*x*)	Seno de *x*, que debe estar expresado en radianes.
cos(*x*)	Coseno de *x*, que debe estar expresado en radianes.
tan(*x*)	Tangente de *x*, que debe estar expresado en radianes.
exp(*x*)	El número *e* elevado a *x*.
ceil(*x*)	Redondeo hacia arriba de *x* (en inglés, «ceiling» significa techo).
floor(*x*)	Redondeo hacia abajo de *x* (en inglés, «floor» significa suelo).
log(*x*)	Logaritmo natural (en base *e*) de *x*.
*log*10(*x*)	Logaritmo decimal (en base 10) de *x*.
sqrt(*x*)	Raíz cuadrada de *x* (del inglés «square root»).

En el módulo matemático se definen, además, algunas constantes de interés:

```
>>> from math import pi, e ↲
>>> pi ↲
3.1415926535897931
>>> e ↲
2.7182818284590451
```

EJERCICIOS

▶ 30 ¿Qué resultados se obtendrán al evaluar las siguientes expresiones Python? Calcula primero a mano el valor resultante de cada expresión y comprueba, con la ayuda del ordenador, si tu resultado es correcto.

a) *int*(*exp*(2 * *log*(3)))

b) *round*(4*sin*(3 * *pi* / 2))

c) *abs*(*log*10(.01) * *sqrt*(25))

d) *round*(3.21123 * *log*10(1000), 3)

2.7.2. Otros módulos de interés

Existe un gran número de módulos, cada uno de ellos especializado en un campo de aplicación determinado. Precisamente, una de las razones por las que Python es un lenguaje potente y extremadamente útil es por la gran colección de módulos con que se distribuye. Hay módulos para el diseño de aplicaciones para web, diseño de interfaces de usuario, compresión de datos, criptografía, multimedia, etc. Y constantemente aparecen nuevos módulos: cualquier programador de Python puede crear sus propios módulos, añadiendo

Precisión de los flotantes

Hemos dicho que los argumentos de las funciones trigonométricas deben expresarse en radianes. Como sabrás, $\text{sen}(\pi) = 0$. Veamos qué opina Python:

```
>>> from math import sin, pi ↲
>>> sin(pi) ↲
1.2246063538223773e-16
```

El resultado que proporciona Python no es cero, sino un número muy próximo a cero: 0.00000000000000012246063538223773. ¿Se ha equivocado Python? No exactamente. Ya dijimos antes que los números flotantes tienen una precisión limitada. El número π está definido en el módulo matemático como 3.1415926535897931, cuando en realidad posee un número infinito de decimales. Así pues, no hemos pedido exactamente el cálculo del seno de π, sino el de un número próximo, pero no exactamente igual. Por otra parte, el módulo matemático hace cálculos mediante algoritmos que pueden introducir errores en el resultado.

así funciones que simplifican la programación en un ámbito cualquiera y poniéndolas a disposición de otros programadores. Nos limitaremos a presentarte ahora unas pocas funciones de un par de módulos interesantes.

Vamos con otro módulo importante: *sys* (sistema), el módulo de «sistema» (*sys* es una abreviatura del inglés «system»). Este módulo contiene funciones que acceden al sistema operativo y constantes dependientes del computador. Una función importante es *exit*, que aborta inmediatamente la ejecución del intérprete (en inglés significa «salir»). La variable *maxint*, también definida en *sys*, contiene el número entero más grande con el que se puede trabajar, y la variable *version*, indica con qué versión de Python estamos trabajando:

```
>>> from sys import maxint, version ↲
>>> maxint ↲
2147483647
>>> version ↲
'2.3 (#1, Aug  2 2003, 09:00:57) \n[GCC 3.3]'
```

¡Ojo! Con esto no queremos decirte que la función *version* o el valor predefinido *maxint* sean importantes y que debas aprender de memoria su nombre y cometido, sino que los módulos de Python contienen centenares de funciones útiles para diferentes cometidos. Un buen programador Python sabe manejarse con los módulos. Existe un manual de referencia que describe todos los módulos estándar de Python. Lo encontrarás con la documentación Python bajo el nombre «Library reference» (en inglés significa «referencia de biblioteca») y podrás consultarla con un navegador web.[8].

[8] En una instalación Linux lo encontrarás en `file:/usr/doc/python/html/index.html` (aunque diferentes distribuciones lo pueden albergar en otro lugar). Si estás trabajando en un ordenador con acceso a Internet, prueba con `http://www.python.org/python/doc/2.3/lib/lib.html`.