Capítulo 6

Funciones

—Y ellos, naturalmente, responden a sus nombres, ¿no? —observó al desgaire el Mosquito.
—Nunca oí decir tal cosa.
—¿Pues de qué les sirve tenerlos —preguntó el Mosquito— si no responden a sus nombres?

LEWIS CARROLL, *Alicia a través del espejo*.

En capítulos anteriores hemos aprendido a utilizar funciones. Algunas de ellas están predefinidas (*abs*, *round*, etc.) mientras que otras deben importarse de módulos antes de poder ser usadas (por ejemplo, *sin* y *cos* se importan del módulo *math*). En este tema aprenderemos a definir nuestras propias funciones. Definiendo nuevas funciones estaremos «enseñando» a Python a hacer cálculos que inicialmente no sabe hacer y, en cierto modo, adaptando el lenguaje de programación al tipo de problemas que deseamos resolver, enriqueciéndolo para que el programador pueda ejecutar acciones complejas de un modo sencillo: llamando a funciones desde su programa.

Ya has usado módulos, es decir, ficheros que contienen funciones y variables de valor predefinido que puedes importar en tus programas. En este capítulo aprenderemos a crear nuestros propios módulos, de manera que reutilizar nuestras funciones en varios programas resultará extremadamente sencillo: bastará con importarlas.

6.1. Uso de funciones

Denominaremos *activar*, *invocar* o *llamar* a una función a la acción de usarla. Las funciones que hemos aprendido a invocar reciben cero, uno o más *argumentos* separados por comas y encerrados entre un par de paréntesis y pueden devolver un valor o no devolver nada.

```
>>> abs(-3) ↲
3
>>> abs(round(2.45, 1)) ↲
2.5
>>> from sys import exit ↲
>>> exit() ↲
```

Podemos llamar a una función desde una expresión. Como el resultado tiene un tipo determinado, hemos de estar atentos a que éste sea compatible con la operación y tipo de los operandos con los que se combina:

```
>>> 1 + (abs(-3) * 2) ↲
7
```

```
>>> 2.5 / abs(round(2.45, 1)) ↲
1.0
>>> 3 + str(3) ↲
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: number coercion failed
```

¿Ves? En el último caso se ha producido un error de tipos porque se ha intentado sumar una cadena, que es el tipo de dato del valor devuelto por *str*, a un entero.

Observa que los argumentos de una función también pueden ser expresiones:

```
>>> abs(round(1.0/9, 4/(1+1))) ↲
0.11
```