Desconocido

## 7 Recicla tu código con funciones y módulos



Piensa en la cantidad de cosas que tiras cada día: botellas de agua, latas de refresco, bolsas de patatas fritas, envoltorios de plástico de bocadillos, bolsas que contenían palitos de zanahoria o rodajas de manzana, bolsas de la compra, periódicos, revistas, etc. Ahora imagina que toda esa basura la tiras en un montón al final de tu camino de entrada, sin separar el papel, el plástico y las latas.

Probablemente reciclas todo lo posible, lo cual es bueno, porque a nadie le gusta pasar por encima de un montón de basura de camino al colegio. En lugar de estar en una enorme y asquerosa pila, las botellas de vidrio recicladas se funden y se convierten en nuevos tarros y botellas; el papel se pulveriza y se convierte en papel reciclado; y el plástico se convierte en artículos de plástico más pesados. Reutilizamos cosas que de otro modo tiraríamos.

En el mundo de la programación, el reciclaje es igual de importante. Puede que tu programa no desaparezca bajo un montón de basura, pero si nunca reutilizas ningún código, teclearás tanto que tus dedos acabarán desgastándose hasta convertirse en dolorosos muñones. Reciclar también hace que tu código sea más corto y fácil de leer.



Como aprenderás en este capítulo, Python ofrece varias formas de reutilizar código.

### Usar funciones

*Las funciones* son trozos de código que le dicen a Python que haga algo. Son una forma de reutilizar código: puedes usar funciones en tus programas una y otra vez. Python dispone de muchas funciones que puedes utilizar; se denominan funciones *incorporadas* , o *builtins* (para más información sobre builtins, consulta [el Apéndice B](app02.xhtml#app02) ). También hay funciones disponibles en módulos (más información sobre ellos más adelante), e incluso puedes escribir funciones tú mismo.

Empezamos a aprender sobre las funciones en el capítulo anterior, cuando utilizamos range y list para hacer que Python contara:

>>> list(range(0, 5))  
[0, 1, 2, 3, 4]

No es demasiado difícil escribir tú mismo una lista de números consecutivos, pero cuanto mayor sea la lista, más tendrás que escribir. Con las funciones, puedes crear con la misma facilidad una lista de mil números.

Aquí tienes un ejemplo que utiliza las funciones list y range para producir una lista de números:

>>> list(range(0, 1000))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16..., 997, 998, 999]

Cuando escribes programas sencillos, las funciones son útiles. Cuando empiezas a escribir programas largos y complejos -como juegos-, las funciones son *esenciales* (suponiendo que quieras terminar de escribir tu programa este siglo).

Veamos cómo escribir nuestras propias funciones.

### Partes de una función

Una función tiene tres partes: un nombre, parámetros y un cuerpo. Aquí tienes un ejemplo de función simple:

>>> def testfunc(myname):  
 print(f'hello {myname}')

El nombre de esta función es testfunc . Tiene un único parámetro, myname , y su cuerpo es el bloque de código que sigue inmediatamente a la línea que empieza por def (abreviatura de *definir* ). Un *parámetro* es una variable que sólo existe mientras se utiliza una función.

Puedes ejecutar la función llamándola por su nombre, utilizando paréntesis alrededor del valor del parámetro:

>>> testfunc('Mary')  
hello Mary

Las funciones pueden tomar cualquier número de parámetros:

>>> def testfunc(fname, lname):  
 print(f'Hello {fname} {lname}')

Cuando utilices varios parámetros, asegúrate de separar los valores con una coma:

>>> testfunc('Mary', 'Smith')  
Hello Mary Smith

También podemos crear variables primero y luego llamar a la función con ellas:

>>> firstname = 'Joe'  
>>> lastname = 'Robertson'  
>>> testfunc(firstname, lastname)  
Hello Joe Robertson

Una función puede devolver un valor con una sentencia return . Por ejemplo, puedes escribir una función para calcular cuánto dinero estás ahorrando:

>>> def savings(pocket\_money, paper\_route, spending):  
 return pocket\_money + paper\_route - spending

Esta función toma tres parámetros. Suma los dos primeros ( pocket\_money y paper\_route) y resta el último ( spending ). El resultado se devuelve y puede asignarse a una variable (de la misma forma que asignamos otros valores a variables) o imprimirse:

>>> print(savings(10, 10, 5))  
15

Pasamos los parámetros 10 , 10 , y 5 , y la función savings calcula 15 como resultado y luego devuelve el valor.

### Variables y ámbito

Una variable que está dentro del cuerpo de una función no puede volver a utilizarse una vez que la función ha terminado de ejecutarse, porque sólo existe dentro de la función. En el mundo de la programación, el lugar donde se puede utilizar una variable se denomina ámbito *.* Veamos una función sencilla que utiliza un par de variables pero no tiene ningún parámetro:

>>> def variable\_test():  
 first\_variable = 10  
 second\_variable = 20  
 return first\_variable \* second\_variable

En este ejemplo, creamos la función variable\_test , que multiplica first\_variable y second\_variable y devuelve el resultado:

>>> print(variable\_test())  
200

Si llamamos a esta función utilizando print , obtenemos 200. Sin embargo, si intentamos imprimir el contenido de first\_variable (o second\_variable ,  para el caso) fuera del bloque de código de la función, obtendremos un mensaje de error:

>>> print(first\_variable)  
Traceback (most recent call last):  
 File "<pyshell#50>", line 1, in <module>  
 print(first\_variable)  
NameError: name 'first\_variable' is not defined

Si una variable se define fuera de la función, tiene un ámbito diferente. Por ejemplo, definamos una variable antes de crear nuestra función, y luego intentemos utilizarla dentro de la función:

➊ >>> another\_variable = 100  
 >>> def variable\_test2():  
 first\_variable = 10  
 second\_variable = 20  
 ➋ return first\_variable \* second\_variable \* another\_variable

En este código, aunque las variables first\_variable y second\_variable no se pueden utilizar fuera de la función, la variable another\_variable (que se creó fuera de la función ➊ ) se puede utilizar dentro de ella ➋ .

Éste es el resultado de llamar a esta función:

>>> print(variable\_test2())  
20000

Ahora, supón que construyes una nave espacial con algo económico como latas de conserva recicladas. Puedes aplanar dos latas a la semana para crear las paredes curvas de tu nave espacial, pero necesitarás unas 500 latas para terminar el fuselaje. Intentemos escribir una función que imprima el total de latas aplastadas cada semana a lo largo de un año.



Nuestra función calculará cuántas latas hemos aplastado cada semana, con el número de latas como parámetro (así será más fácil cambiar el número de latas más adelante):

>>> def spaceship\_building(cans):  
 total\_cans = 0  
 for week in range(1, 53):  
 total\_cans = total\_cans + cans  
 print(f'Week {week} = {total\_cans} cans')

En la primera línea de la función, creamos la variable total\_cans y establecemos su valor en 0. A continuación, creamos un bucle para las semanas del año y añadimos el número de latas aplastadas cada semana. Este bloque de código constituye el contenido de nuestra función, y las dos últimas líneas constituyen otro bloque del bucle for .

Probemos a introducir esa función en la Shell de Python y a llamarla con distintos valores para el número de latas, empezando por 2:

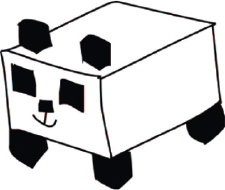
>>> spaceship\_building(2)  
Week 1 = 2 cans  
Week 2 = 4 cans  
Week 3 = 6 cans  
Week 4 = 8 cans  
Week 5 = 10 cans  
Week 6 = 12 cans  
Week 7 = 14 cans  
Week 8 = 16 cans  
...  
Week 50 = 100 cans  
Week 51 = 102 cans  
Week 52 = 104 cans  
  
>>> spaceship\_building(10)  
Week 1 = 10 cans  
Week 2 = 20 cans  
Week 3 = 30 cans  
Week 4 = 40 cans  
Week 5 = 50 cans  
...  
Week 48 = 480 cans  
Week 49 = 490 cans  
Week 50 = 500 cans  
Week 51 = 510 cans  
Week 52 = 520 cans

Esta función puede reutilizarse con distintos valores para el número de latas por semana, lo que es un poco más eficaz que volver a escribir el bucle for cada vez que quieras probarlo con distintos números  . Cuando ejecutamos spaceship\_building(10) , podemos ver que tendremos suficientes latas para construir las paredes de la nave espacial en la semana 50.

Las funciones también pueden agruparse en módulos, lo que hace que Python sea *realmente* útil, en lugar de sólo medianamente útil.

### Utilización de módulos

*Los módulos* se utilizan para agrupar funciones, variables y otras cosas en programas más grandes y potentes. Algunos módulos están integrados en Python, y otros deben descargarse por separado. Hay módulos que te ayudan a escribir juegos (como tkinter , que está incorporado, y PyGame , que no lo está), módulos para manipular imágenes (como Pillow , la Python Imaging Library), y módulos para dibujar gráficos tridimensionales (como Panda3D ).



Los módulos pueden utilizarse para hacer todo tipo de cosas útiles. Por ejemplo, si estuvieras diseñando un juego de simulación y quisieras que el mundo cambiara de acuerdo con el mundo real, podrías calcular la fecha y la hora actuales utilizando el módulo incorporado time :

>>> import time

El comando import indica a Python que queremos utilizar el módulo time .

A continuación, podemos llamar a funciones disponibles en este módulo utilizando el símbolo punto. (En el [Capítulo 4](ch04.xhtml#ch04) utilizamos funciones similares para trabajar con el módulo turtle , como t.forward(50) .) Por ejemplo, así es como podríamos llamar a la función asctime en el módulo time :

>>> print(time.asctime())  
Tue Aug 12 07:05:32 2025

La función asctime es una parte del módulo time que devuelve la fecha y hora actuales en forma de cadena.

Ahora supongamos que quieres pedir a alguien que introduzca un valor, como su fecha de nacimiento o su edad. Puedes hacerlo utilizando una sentencia print para mostrar un mensaje, y el módulo sys (abreviatura de *sistema* ), que contiene utilidades para interactuar con el propio sistema Python. En primer lugar, importamos el módulo sys :



>>> import sys

Dentro del módulo sys hay un objeto especial llamado stdin (para *entrada estándar* ), que proporciona una función útil llamada readline . La función readline se utiliza para leer una línea de texto escrita en el teclado hasta que pulsas ENTER. (Veremos cómo funcionan los objetos en el [Capítulo 8 .](ch08.xhtml#ch08) ) Para probar readline , introduce el siguiente código en la Shell de Python:

>>> import sys  
>>> print(sys.stdin.readline())

Si a continuación escribes unas palabras y pulsas ENTER, esas palabras se imprimirán en el Shell de Python.

Piensa en el código que escribimos en el [Capítulo 5](ch05.xhtml#ch05) , utilizando una sentencia if :

>>> if age >= 10 and age <= 13:  
 print('What is 13 + 49 + 84 + 155 + 97? A headache!')  
 else:  
 print('Huh?')

En lugar de crear la variable age y darle un valor específico antes de la sentencia if , ahora podemos pedir a alguien que introduzca el valor en su lugar. Pero primero, convirtamos el código en una función:

>>> def silly\_age\_joke(age):  
 if age >= 10 and age <= 13:  
 print('What is 13 + 49 + 84 + 155 + 97? A headache!')  
 else:  
 print('Huh?')

Ahora puedes llamar a la función introduciendo su nombre y diciéndole qué número debe utilizar introduciendo el número entre paréntesis. ¿Funciona?

>>> silly\_age\_joke(9)  
Huh?  
>>> silly\_age\_joke(10)  
What is 13 + 49 + 84 + 155 + 97? A headache!

¡Funciona! Ahora hagamos que la función pregunte la edad de una persona. (Puedes añadir o modificar una función tantas veces como quieras).

>>> def silly\_age\_joke():  
 print('How old are you?')  
 ➊ age = int(sys.stdin.readline())  
 ➋ if age >= 10 and age <= 13:  
 print('What is 13 + 49 + 84 + 155 + 97? A headache!')  
 else:  
 print('Huh?')

¿Has reconocido la función int ➊ , que convierte una cadena en un número? Hemos incluido esa función porque sys.stdin. readline() devuelve cualquier cosa que alguien introduzca como cadena, pero nosotros queremos un número para poder compararlo con los números 10 y 13 ➋ . Para probarlo tú mismo, llama a la función sin ningún parámetro y escribe un número cuando aparezca How old are you? :

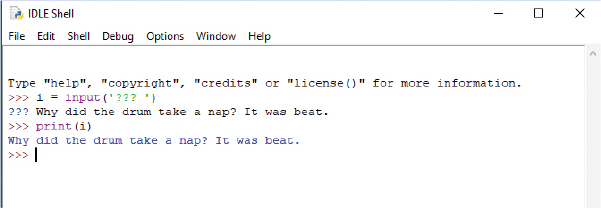
>>> silly\_age\_joke()  
How old are you?  
10  
What is 13 + 49 + 84 + 155 + 97? A headache!  
>>> silly\_age\_joke()  
How old are you?  
15  
Huh?

La primera vez que llamamos a la función, aparece "¿Cuántos años tienes?"; entonces introducimos 10 , e imprime el chiste. La segunda vez, introducimos 15 e imprime "¿Eh?".

### La función de entrada

La función sys.stdin.readline no es la única forma de leer la entrada desde el teclado. Una opción mucho más sencilla es la función incorporada input . La función input toma un parámetro opcional prompt (una cadena con el mensaje que quieres mostrar), y luego devuelve lo que se escriba hasta que se pulse la tecla ENTER. [La Figura 7-1](ch07.xhtml#ch07fig01) muestra lo que ocurre cuando ejecutas este código:

i = input('??? ')  
print(i)



*Figura 7-1: Uso de la función* de *entrada*

Reescribamos la función silly\_age\_joke para utilizar en su lugar input :

>>> def silly\_age\_joke():  
 age = int(input('How old are you?\n'))  
 if age >= 10 and age <= 13:  
 print('What is 13 + 49 + 84 + 155 + 97? A headache!')  
 else:  
 print('Huh?')

Aparte de tener ligeramente menos líneas, otra diferencia entre el código anterior y esta versión es que añadimos un carácter *de nueva línea* (∖ n ) al final de nuestra cadena ( ‘How old are you? ∖ n’ ). Una nueva línea simplemente desplaza el cursor de una línea de la pantalla a la siguiente. Esto ocurre automáticamente con print , pero no con la función input .

En cualquier caso, este código funciona exactamente igual que antes.

### Lo que has aprendido

En este capítulo, has aprendido a crear trozos de código reutilizables en Python con funciones y a utilizar funciones proporcionadas por los módulos de  . Has visto cómo el ámbito de las variables controla si pueden verse dentro o fuera de las funciones. También aprendiste a crear funciones utilizando la palabra clave def y a importar módulos para utilizar su contenido.

### Rompecabezas de programación

Prueba los siguientes ejemplos para experimentar con la creación de tus propias funciones. Las soluciones se encuentran en http://python-for-kids.com [*.*](http://python-for-kids.com)

#### #nº 1: Función básica de peso lunar

En uno de los puzzles de programación [del Capítulo 6](ch06.xhtml#ch06) , creamos un bucle for para determinar tu peso en la luna durante un periodo de 15 años. Ese bucle for podría convertirse fácilmente en una función. Intenta crear una función que tome un peso inicial y aumente su cantidad cada año. Podrías llamar a la nueva función utilizando un código como éste:

>>> moon\_weight(30, 0.25)

#### #2: Función Peso Lunar y Años

Toma la función que acabas de crear y cámbiala para calcular el peso en distintos periodos, como 5 años o 20 años. Asegúrate de cambiar la función para que tome tres argumentos: el peso inicial, el peso ganado cada año y el número de años:

>>> moon\_weight(90, 0.25, 5)

#### #3: Programa Peso Lunar

En lugar de utilizar una función simple y pasar los valores como parámetros, puedes utilizar sys.stdin.readline() o input() para hacer un miniprograma que pida los valores. En este caso, llamas a la función sin ningún parámetro:

>>> moon\_weight()

La función mostrará un mensaje preguntando por el peso inicial, un segundo mensaje preguntando por la cantidad que el peso  aumentará cada año, y un mensaje final preguntando por el número de años. Verías algo como lo siguiente:

Please enter your current Earth weight  
45  
Please enter the amount your weight might increase each year  
0.4  
Please enter the number of years  
12

Recuerda importar el módulo sys antes de crear tu función, si utilizas sys.stdin.readline() :

>>> import sys

#### #4: Programa de peso en Marte

Cambiemos nuestro programa de peso lunar para calcular esta vez el peso en Marte, sólo para toda tu familia. La función debe pedir el peso de cada miembro de la familia, calcular cuánto pesaría en Marte (multiplicando el número por 0,3782), y luego sumar y mostrar el peso total al final. Puedes escribir este código de muchas formas; lo importante es que muestre el peso total al final.