Desconocido

## 6 Volverse loco



No hay nada peor que tener que hacer lo mismo una y otra vez. Hay una razón por la que nos dicen que contemos ovejas cuando nos cuesta conciliar el sueño, y no tiene nada que ver con los increíbles poderes inductores del sueño de los mamíferos lanudos. Es porque la repetición interminable es aburrida, y tu mente puede dormirse fácilmente si no te centras en algo interesante.

A los programadores tampoco les gusta repetirse, a menos que también estén intentando dormirse. Por suerte, la mayoría de los lenguajes de programación tienen un bucle for , que repite cosas como sentencias y bloques de código automáticamente.

En este capítulo, veremos los bucles for , así como otro tipo de bucle que ofrece Python: el bucle while .



### Uso de los bucles for

Para imprimir hello cinco veces en Python, *podrías* hacer lo siguiente:

>>> print('hello')  
hello  
>>> print('hello')  
hello  
>>> print('hello')  
hello  
>>> print('hello')  
hello  
>>> print('hello')  
hello

Pero esto es bastante tedioso. En su lugar, puedes utilizar un bucle for para reducir la cantidad de escritura y repetición, de la siguiente manera:

➊ >>> for x in range(0, 5):  
 ➋ print('hello')  
  
 hello  
 hello  
 hello  
 hello  
 hello

La función range ➊ puede crear una lista de números que van desde un número inicial hasta el número situado justo antes del número final. Esto puede parecer un poco confuso, así que combinemos la función range con la función list para ver exactamente cómo funciona. La función range en realidad no crea una lista  de números; devuelve un *iterador* , que es un objeto de Python diseñado para trabajar con bucles. Sin embargo, si combinamos range con list , obtendremos una lista de números:

>>> print(list(range(10, 20)))  
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

En nuestro ejemplo anterior, for x in range(0,5): en realidad le está diciendo a Python que haga lo siguiente:

1. Empezar a contar desde 0 y parar antes de llegar a 5.
2. Por cada número que contemos, almacena el valor en la variable x .

A continuación, Python ejecuta la sentencia print(’hello’) . Observa los cuatro espacios adicionales al principio de la línea ➋ (en comparación con la primera línea ➊ ). IDLE debería haber sangrado esto automáticamente por ti. Cuando pulsamos ENTER después de la segunda línea, Python imprime hello cinco veces.

También podríamos utilizar la variable x en nuestra sentencia print para contar el número de saludos:

>>> for x in range(0, 5):  
 print(f'hello {x}')  
hello 0  
hello 1  
hello 2  
hello 3  
hello 4

Si nos deshacemos del bucle for , nuestro código podría tener este aspecto:

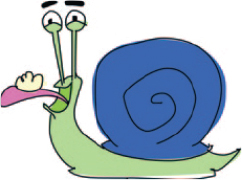
>>> x = 0  
>>> print(f'hello {x}')  
hello 0  
>>> x = 1  
>>> print(f'hello {x}')  
hello 1  
>>> x = 2  
>>> print(f'hello {x}')  
hello 2  
>>> x = 3  
>>> print(f'hello {x}')  
hello 3  
>>> x = 4  
>>> print(f'hello {x}')  
hello 4

¡Usar el bucle for nos ahorró escribir ocho líneas de código adicionales! Es una buena práctica evitar hacer las cosas más de una vez, por lo que el bucle for es una sentencia muy popular entre los programadores.

No hace falta que te limites a utilizar la función range cuando hagas bucles for . También puedes utilizar una lista que ya hayas creado, como la lista de la compra del [Capítulo 3](ch03.xhtml#ch03) , como se indica a continuación:

>>> wizard\_list = ['spider legs', 'toe of frog', 'snail tongue',   
 'bat wing', 'slug butter', 'bear burp']  
>>> for ingredient in wizard\_list:  
 print(ingredient)  
spider legs  
toe of frog  
snail tongue  
bat wing  
slug butter  
bear burp

Este código le dice a Python: "Por cada elemento de wizard\_list , almacena el valor en la variable i , y luego imprime el contenido de esa variable". Si nos deshiciéramos del bucle for , tendríamos que hacer algo como esto:



>>> wizard\_list = ['spider legs', 'toe of frog', 'snail tongue',   
 'bat wing', 'slug butter', 'bear burp']  
>>> print(wizard\_list[0])  
spider legs  
>>> print(wizard\_list[1])  
toe of frog  
>>> print(wizard\_list[2])  
snail tongue  
>>> print(wizard\_list[3])  
bat wing  
>>> print(wizard\_list[4])  
slug butter  
>>> print(wizard\_list[5])  
bear burp

Una vez más, for nos ha ahorrado mucho tecleo.

Vamos a crear otro bucle. Escribe el siguiente código en el intérprete de comandos de Python; debería sangrar automáticamente el código por ti:

>>> hugehairypants = ['huge', 'hairy', 'pants']  
>>> for i in hugehairypants:  
 print(i)  
 print(i)  
  
huge  
huge  
hairy  
hairy  
pants  
pants

En la primera línea, creamos una lista que contiene ‘huge’ , ‘hairy’ , y ‘pants’ . En la línea siguiente, recorremos los elementos de la lista y asignamos cada uno de ellos a la variable i . A continuación, imprimimos el contenido de la variable dos veces en las dos líneas siguientes. Pulsa ENTER en la siguiente línea en blanco para indicar a Python que finalice el bloque. A continuación, ejecuta el código e imprime dos veces cada elemento de la lista.



Recuerda que si introduces un número incorrecto de espacios, aparecerá un mensaje de error. Si introdujeras el código anterior con un espacio de más en la cuarta línea, Python mostraría un error de sangría:

>>> hugehairypants = ['huge', 'hairy', 'pants']  
>>> for i in hugehairypants:  
 print(i)  
 print(i)  
  
IndentationError: unexpected indent

Como aprendiste en el [Capítulo 5](ch05.xhtml#ch05) , Python espera que el número de espacios en un bloque sea coherente. No importa cuántos espacios insertes, siempre que utilices el mismo número para cada nueva línea (esto también hace que el código sea más fácil de leer para los humanos).

Aquí tienes un ejemplo más complicado de un bucle for con dos bloques de código:

>>> hugehairypants = ['huge', 'hairy', 'pants']  
>>> for i in hugehairypants:  
 print(i)  
 for j in hugehairypants:  
 print(j)

¿Dónde están los bloques en este código? El primer bloque es el primer bucle for :

hugehairypants = ['huge', 'hairy', 'pants']  
for i in hugehairypants:  
 print(i) #  
 for j in hugehairypants: # These lines are the FIRST block.  
 print(j) #

El siguiente bloque es la única línea print del segundo bucle for :

hugehairypants = ['huge', 'hairy', 'pants']  
for i in hugehairypants:  
 print(i)  
 for j in hugehairypants:  
 print(j) # This line is also the SECOND block.

¿Puedes averiguar qué va a hacer este código?

Después de crear una lista llamada hugehairypants , podemos deducir de las dos líneas siguientes que Python va a recorrer los elementos de la lista e imprimir cada uno de ellos. Sin embargo, en for j in hugehairypants , volverá a recorrer la lista, esta vez asignando el valor a la variable j , y luego volverá a imprimir cada elemento. Estas dos últimas líneas de código siguen formando parte del primer bucle for , lo que significa que se ejecutarán para cada elemento a medida que el bucle for recorra la lista.

Cuando se ejecute este código, deberíamos ver huge seguido de huge, hairy, pants , y luego hairy seguido de huge, hairy, pants , y así sucesivamente.

Introduce el código en el intérprete de comandos de Python y compruébalo tú mismo:

>>> hugehairypants = ['huge', 'hairy', 'pants']  
 >>> for i in hugehairypants:  
 print(i)  
 for j in hugehairypants:  
 print(j)  
  
→ huge  
 huge  
 hairy  
 pants  
→ hairy  
 huge  
 hairy  
 pants  
→ pants  
 huge  
 hairy  
 pants

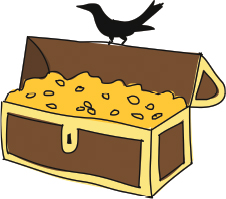
Python entra en el primer bucle e imprime un elemento de la lista. A continuación, entra en el segundo bucle e imprime todos los elementos de la lista. A continuación, imprime el segundo elemento de la lista con print(i) , y luego vuelve a imprimir la lista completa con el comando print(j) en el bucle interior. Por último, vuelve a imprimir el tercer elemento de la lista con print(i) y, a continuación, vuelve a imprimir la lista completa con el bucle interno. En la salida, las líneas marcadas **→** se imprimen con print(i) , y las líneas no marcadas se imprimen con la sentencia print(j) .

Intentemos algo más práctico que imprimir palabras tontas. ¿Recuerdas el cálculo que hicimos en el [Capítulo](ch02.xhtml#ch02) 2 para averiguar cuántas monedas de oro tendrías al final del año si utilizabas la máquina duplicadora de tu abuelo? Era así:

>>> found\_coins + magic\_coins \* 365 - stolen\_coins \* 52

Esto representa 20 monedas encontradas más 10 monedas mágicas multiplicadas por 365 días del año, menos las 3 monedas semanales robadas por el cuervo.

Comprobemos cómo aumentará tu montón de monedas de oro cada semana. Podemos hacerlo con un bucle for , pero antes tenemos que cambiar el valor de nuestra variable magic\_coins para que represente el número total de monedas por semana. Obtenemos 10 monedas mágicas al día, y hay 7 días en una semana, así que magic\_coins debería ser 70:



>>> found\_coins = 20  
>>> magic\_coins = 70  
>>> stolen\_coins = 3

Podemos ver cómo aumenta nuestro tesoro cada semana creando otra variable, llamada coins , y utilizando un bucle for :

>>> found\_coins = 20  
 >>> magic\_coins = 70  
 >>> stolen\_coins = 3  
➊ >>> coins = found\_coins  
 >>> for week in range(1, 53):  
 coins = coins + magic\_coins - stolen\_coins  
 print(f'Week {week} = {coins}')

La variable coins se carga con el valor de la variable found\_coins ➊ ; éste es nuestro número inicial. La siguiente línea configura el bucle for , que ejecutará los comandos del bloque. Cada vez que se ejecuta el bucle, la variable week se carga con el siguiente número del rango de 1 a 52.

La línea que contiene coins = coins + magic\_coins - stolen\_coins es un poco más complicada. Cada semana queremos sumar el número de monedas que hemos creado mágicamente y restar el número de monedas que ha robado el cuervo. Piensa en la variable coins como en un cofre del tesoro. Cada semana, las nuevas monedas se apilan en el cofre. Así que esta línea le está diciendo a Python: "Sustituye el contenido de coins por el número de mis monedas actuales, más las que he creado esta semana". El signo igual ( = ) es un trozo de código mandón que dice: "Resuelve primero lo de la derecha y guárdalo para más tarde, usando el nombre de la izquierda".

La sentencia print imprime en la pantalla el número de la semana y el número total de monedas (hasta el momento). (Considera la posibilidad de releer  "Incrustar valores en cadenas" en [la página 29 .](ch03.xhtml#ch03lev1sec4) ) Si ejecutas este programa, verás algo parecido a [la Figura 6-1 .](ch06.xhtml#ch06fig01)

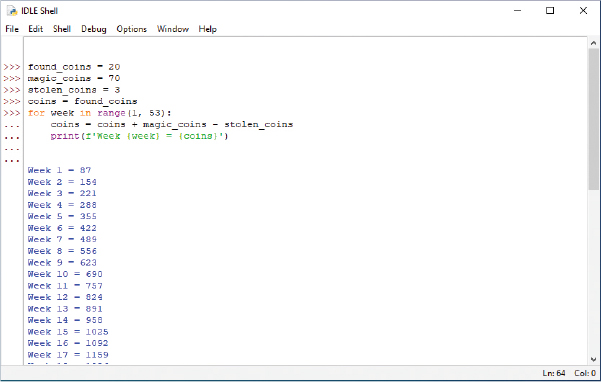


Figura 6-1*: Ejecución del bucle*

### Ya que hablamos de bucles . . .

Un bucle for no es el único tipo de bucle que puedes hacer en Python. También existe el bucle while . Mientras que un bucle for tiene una longitud específica, un bucle while no la tiene. Se utiliza cuando no sabes con antelación cuándo debe detenerse el bucle.

Imagina una escalera con 20 peldaños. La escalera está en el interior, y sabes que puedes subir fácilmente 20 peldaños. Un bucle for es así:

>>> for step in range(0, 20):  
 print(step)

Ahora imagina una escalera que sube por la ladera de una montaña. La montaña es muy alta, y puede que te quedes sin energía antes de llegar a la cima. O puede que el tiempo empeore y te obligue a detenerte. Así es un bucle while :

step = 0  
while step < 10000:  
 print(step)  
 if tired == True:  
 break  
 elif badweather == True:  
 break  
 else:  
 step = step + 1

Si intentas ejecutar este código, obtendrás un error, porque no hemos creado las variables tired y badweather . Aunque aquí no hay suficiente código para hacer un programa que funcione, demuestra un bucle while sencillo.



Empezamos creando la variable step con step = 0 . A continuación, creamos un bucle while que comprueba si el valor de step es menor que 10.000 ( step < 10000 ), que es el número de pasos que hay desde la base de la montaña hasta la cima. Mientras step sea menor que 10.000, Python ejecutará el resto del código.

Con print(step) , imprimimos el valor de step y luego comprobamos si el valor de la variable tired es True con la condición if tired == True . Si lo es, utilizamos break para salir del bucle. La palabra clave break es una forma de saltar fuera (o detener) un bucle inmediatamente, y funciona tanto con los bucles while como con los for .

En este ejemplo, break hace que Python salte del bloque y pase a cualquier comando que apareciera después de la línea step = step + 1 .

La línea elif badweather == True comprueba si badweather está en True ; si es así, break sale del bucle. Si ni tired ni badweather son True (visto en else ), añadimos 1 a la variable step con step = step + 1 , y el bucle continúa.

Los pasos de un bucle while son los siguientes:

1. Comprueba la condición.
2. Ejecuta el código del bloque.
3. Repetir.

Lo más habitual es crear un bucle while con varias condiciones, en lugar de sólo una, como en este caso:

>>> x = 45  
>>> y = 80  
>>> while x < 50 and y < 100:  
 x = x + 1  
 y = y + 1  
 print(x, y)

Aquí, creamos una variable x con el valor 45 y una variable y con el valor 80. El bucle comprueba dos condiciones: si x es menor que 50 y si y es menor que 100. Si ambas condiciones son verdaderas, se ejecutan las líneas siguientes, que suman 1 a ambas variables y luego las imprimen. La salida de este código es la siguiente:

46 81  
47 82  
48 83  
49 84  
50 85

¿Puedes averiguar cómo funciona esto?

Empezamos a contar en 45 para x y en 80 para y , y luego *incrementamos* (añadimos 1 a cada variable) cada vez que se ejecuta el código del bucle. El bucle se ejecutará mientras x sea menor que 50 y y sea menor que 100. Después de ejecutar el bucle cinco veces, el valor de x alcanza 50. Ahora la primera condición ( x < 50 ) ya no es cierta, por lo que Python detiene el bucle.

También podemos utilizar un bucle while para crear un *bucle semi-eterno* que podría durar eternamente, pero que continúa hasta que ocurre algo en el código para salir de él.

Aquí tienes un ejemplo:

while True:  
 lots of code here  
 lots of code here  
 lots of code here  
 if some\_value == True:  
 break

La condición para el bucle while es simplemente True , que siempre es verdadera, por lo que el código del bloque siempre se ejecutará (por tanto, el bucle es eterno). Python saldrá del bucle sólo si la variable some\_value es verdadera.

### Lo que has aprendido

En este capítulo, hemos utilizado dos tipos de bucles para realizar tareas repetitivas: los bucles for y los bucles while . Son similares, pero pueden utilizarse de formas distintas. Le dijimos a Python lo que queríamos repetir escribiendo las tareas dentro de bloques de código, que pusimos dentro de bucles. También utilizamos la palabra clave break para detener el bucle.

### Rompecabezas de programación

Aquí tienes algunos ejemplos de bucles para probar. Las soluciones se encuentran en http://python-for-kids.com [*.*](http://python-for-kids.com)

#### #nº 1: El bucle Hola

¿Qué crees que hará el siguiente código? Adivina qué ocurrirá, y luego ejecuta el código en Python para ver si has acertado.

>>> for x in range(0, 20):  
 print(f'hello {x}')  
 if x < 9:  
 break

#### #2: Números pares

Crea un bucle que imprima números pares hasta llegar a tu edad (si tu edad es un número impar, crea un bucle que imprima números impares hasta llegar a tu edad). Por ejemplo, podría imprimir algo así:

2  
4  
6  
8  
10  
12  
14

#### #3: Mis cinco ingredientes favoritos

Crea una lista que contenga cinco ingredientes diferentes para bocadillos, como la siguiente:

>>> ingredients = ['snails', 'leeches', 'gorilla belly-button lint',   
 'caterpillar eyebrows', 'centipede toes']

Ahora crea un bucle que imprima la lista (incluyendo los números):

1 snails  
2 leeches  
3 gorilla belly-button lint  
4 caterpillar eyebrows  
5 centipede toes

#### #4: Tu peso en la Luna

Si ahora mismo estuvieras en la Luna, tu peso sería el 16,5% del que tienes en la Tierra. Puedes calcularlo multiplicando tu peso en la Tierra por 0,165.

Si aumentaras un kilo cada año durante los próximos 15 años, ¿cuál sería tu peso cuando visitaras la Luna cada año y al final de los 15 años? Escribe un programa utilizando un bucle for que imprima tu peso en la Luna cada año.