# 7 Entradas del usuario y bucles while



La mayoría de los programas se escriben para resolver un problema del usuario final. Para ello, normalmente necesitas obtener alguna información del usuario. Por ejemplo, supongamos que alguien quiere averiguar si tiene edad suficiente para votar. Si escribes un programa para responder a esta pregunta, necesitarás saber la edad del usuario antes de poder darle una respuesta. El programa tendrá que pedir al usuario que introduzca, o *input*, su edad; una vez que el programa tenga esta información, podrá compararla con la edad para votar para determinar si el usuario tiene la edad suficiente e informar del resultado.

En este capítulo aprenderás a aceptar las entradas del usuario para que tu programa pueda trabajar con ellas. Cuando tu programa necesite un nombre, podrás pedírselo al usuario. Cuando tu programa necesite una lista de nombres, podrás pedir al usuario una serie de nombres. Para ello, utilizarás la función input().

También aprenderás a mantener los programas en ejecución todo el tiempo que los usuarios deseen, para que puedan introducir toda la información que necesiten; después, tu programa podrá trabajar con esa información. Utilizarás el bucle while de Python para mantener los programas en ejecución mientras se cumplan determinadas condiciones.

Con la capacidad de trabajar con la información introducida por el usuario y la posibilidad de controlar cuánto tiempo se ejecutan tus programas, podrás escribir programas totalmente interactivos.

## Cómo funciona la función input()

La función input() pausa tu programa y espera a que el usuario introduzca algún texto. Una vez que Python recibe la entrada del usuario, la asigna a una variable para que te resulte más cómodo trabajar con ella.

Por ejemplo, el siguiente programa pide al usuario que introduzca un texto y, a continuación, muestra ese mensaje al usuario:

**parrot.py**

message = input("Tell me something, and I will repeat it back to you: ")  
print(message)

La función input() toma un argumento: el *prompt* que queremos mostrar al usuario, para que sepa qué tipo de información debe introducir. En este ejemplo, cuando Python ejecuta la primera línea, el usuario ve el prompt Tell me something, and I will repeat it back to you: . El programa espera mientras el usuario introduce su respuesta y continúa después de que el usuario pulse INTRO. La respuesta se asigna a la variable message, y a continuación print(message) muestra la información introducida al usuario:

Tell me something, and I will repeat it back to you: Hello everyone!  
Hello everyone!

## Nota

Algunos editores de texto no ejecutan programas que solicitan la entrada de datos al usuario. Puedes utilizar estos editores para escribir programas que soliciten entradas, pero tendrás que ejecutar estos programas desde un terminal. Consulta "Ejecutar programas Python desdeun terminal" en la página 11.

### Escribir instrucciones claras

Cada vez que utilices la función input(), debes incluir una instrucción clara y fácil de seguir que indique al usuario exactamente qué tipo de información estás buscando. Cualquier instrucción que indique al usuario lo que debe introducir debería funcionar. Por ejemplo:

**greeter.py**

name = input("Please enter your name: ")  
print(f"\nHello, {name}!")

Añade un espacio al final de tus indicaciones (después de los dos puntos en el ejemplo anterior) para separar la indicación de la respuesta del usuario y para dejarle claro dónde introducir su texto. Por ejemplo:

Please enter your name: Eric  
Hello, Eric!

A veces querrás escribir un aviso de más de una línea. Por ejemplo, puede que quieras decirle al usuario por qué le pides que introduzca determinada información. Puedes asignar tu pregunta a una variable y pasar esa variable a la función input(). Esto te permite construir tu pregunta en varias líneas, y luego escribir una sentencia input() limpia.

**greeter.py**

prompt = "If you share your name, we can personalize the messages you see."  
prompt += "\nWhat is your first name? "  
  
name = input(prompt)  
print(f"\nHello, {name}!")

Este ejemplo muestra una forma de construir una cadena de varias líneas. La primera línea asigna la primera parte del mensaje a la variable prompt. En la segunda línea, el operador += toma la cadena que se asignó a prompt y añade la nueva cadena al final.

El mensaje abarca ahora dos líneas, de nuevo con espacio tras el signo de interrogación para mayor claridad:

If you share your name, we can personalize the messages you see.  
What is your first name? Eric  
  
Hello, Eric!

### Uso de int() para aceptar entradas numéricas

Cuando utilizas la función input(), Python interpreta todo lo que introduce el usuario como una cadena. Considera la siguiente sesión del intérprete, que pregunta por la edad del usuario:

>>> age = input("How old are you? ")  
How old are you? 21  
>>> age  
'21'

El usuario introduce el número 21, pero cuando pedimos a Python el valor de age, nos devuelve '21', la representación en cadena del valor numérico introducido. Sabemos que Python interpretó la entrada como una cadena porque ahora el número está entre comillas. Si lo único que quieres es imprimir la entrada, esto funciona bien. Pero si intentas utilizar la entrada como un número, obtendrás un error:

>>> age = input("How old are you? ")  
How old are you? 21  
❶ >>> age >= 18  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
❷ TypeError: '>=' not supported between instances of 'str' and 'int'

Cuando intentas utilizar la entrada para hacer una comparación numérica ❶, Python produce un error porque no puede comparar una cadena con un número entero: la cadena '21' que se asigna a age no puede compararse con el valor numérico 18 ❷.

Podemos resolver este problema utilizando la función int(), que convierte la cadena de entrada en un valor numérico. Esto permite que la comparación se ejecute correctamente:

>>> age = input("How old are you? ")  
How old are you? 21  
❶ >>> age = int(age)  
>>> age >= 18  
True

En este ejemplo, cuando introducimos 21 en el indicador, Python interpreta el número como una cadena, pero el valor se convierte en una representación numérica mediante int() ❶. Ahora Python puede ejecutar la prueba condicional: compara age (que ahora representa el valor numérico 21) y 18 para ver si age es mayor o igual que 18. Esta prueba evalúa a True.

¿Cómo se utiliza la función int() en un programa real? Considera un programa que determine si las personas son lo suficientemente altas para subir a una montaña rusa:

**rollercoaster.py**

height = input("How tall are you, in inches? ")  
height = int(height)  
  
if height >= 48:  
 print("\nYou're tall enough to ride!")  
else:  
 print("\nYou'll be able to ride when you're a little older.")

El programa puede comparar height con 48 porque height = int(height) convierte el valor introducido en una representación numérica antes de realizar la comparación. Si el número introducido es mayor o igual que 48, le decimos al usuario que es lo suficientemente alto:

How tall are you, in inches? 71  
  
You're tall enough to ride!

Cuando utilices entradas numéricas para hacer cálculos y comparaciones, asegúrate de convertir primero el valor de entrada en una representación numérica.

### El operador módulo

Una herramienta útil para trabajar con información numérica es el *modulo operator* (%), que divide un número por otro y devuelve el resto:

>>> 4 % 3  
1  
>>> 5 % 3  
2  
>>> 6 % 3  
0  
>>> 7 % 3  
1

El operador módulo no te dice cuántas veces cabe un número en otro; sólo te dice cuál es el resto.

Cuando un número es divisible por otro, el resto es 0, por lo que el operador módulo siempre devuelve 0. Puedes utilizar este hecho para determinar si un número es par o impar:

**even\_or\_odd.py**

number = input("Enter a number, and I'll tell you if it's even or odd: ")  
number = int(number)  
  
if number % 2 == 0:  
 print(f"\nThe number {number} is even.")  
else:  
 print(f"\nThe number {number} is odd.")

Los números pares siempre son divisibles por dos, por lo que si el módulo de un número y dos es cero (aquí, if number % 2 == 0) el número es par. En caso contrario, es impar.

Enter a number, and I'll tell you if it's even or odd: 42  
  
The number 42 is even.

## Pruébalo tú mismo

7-1. Coche de alquiler: Escribe un programa que pregunte al usuario qué tipo de coche de alquiler le gustaría. Imprime un mensaje sobre ese coche, como "A ver si puedo encontrarte un Subaru".

7-2. Asientos en un restaurante: Escribe un programa que pregunte al usuario cuántas personas hay en su grupo para cenar. Si la respuesta es más de ocho, imprime un mensaje diciendo que tendrán que esperar para una mesa. En caso contrario, informa de que su mesa está lista.

7-3. Múltiplos de diez: Pide al usuario un número y, a continuación, informa de si el número es múltiplo de 10 o no.

## Introducción a los bucles while

El bucle for toma una colección de elementos y ejecuta un bloque de código una vez por cada elemento de la colección. En cambio, el bucle while se ejecuta mientras, o *while*, se cumpla una determinada condición.

### El bucle while en acción

Puedes utilizar un bucle while para contar hacia arriba a través de una serie de números. Por ejemplo, el siguiente bucle while cuenta del 1 al 5:

**counting.py**

current\_number = 1  
while current\_number <= 5:  
 print(current\_number)  
 current\_number += 1

En la primera línea, empezamos a contar desde 1 asignando a current\_number el valor 1. A continuación, se establece el bucle while para que siga ejecutándose mientras el valor de current\_number sea menor o igual que 5. El código dentro del bucle imprime el valor de current\_number y luego añade 1 a ese valor con current\_number += 1. (El operador += es una abreviatura de current\_number = current\_number + 1.)

Python repite el bucle mientras la condición current\_number <= 5 sea cierta. Como 1 es menor que 5, Python imprime 1 y luego añade 1, convirtiendo el número actual en 2. Como 2 es menor que 5, Python imprime 2 y vuelve a añadir 1, convirtiendo el número actual en 3, y así sucesivamente. Una vez que el valor de current\_number es mayor que 5, el bucle deja de ejecutarse y el programa termina:

1  
2  
3  
4  
5

Lo más probable es que los programas que utilizas a diario contengan bucles while. Por ejemplo, un juego necesita un bucle while para seguir ejecutándose todo el tiempo que quieras seguir jugando, y así puede dejar de ejecutarse en cuanto le pidas que salga. Los programas no serían divertidos de usar si dejaran de ejecutarse antes de que se lo pidiéramos o siguieran ejecutándose incluso después de que quisiéramos salir, por eso los bucles while son bastante útiles.

### Dejar que el usuario elija cuándo salir

Podemos hacer que el programa *parrot.py* se ejecute tanto tiempo como quiera el usuario poniendo la mayor parte del programa dentro de un bucle while. Definiremos un *quit value* y luego mantendremos el programa en ejecución mientras el usuario no haya introducido el valor de abandonar:

**parrot.py**

prompt = "\nTell me something, and I will repeat it back to you:"  
prompt += "\nEnter 'quit' to end the program. "  
  
message = ""  
while message != 'quit':  
 message = input(prompt)  
 print(message)

Primero definimos un prompt que indica al usuario sus dos opciones: introducir un mensaje o introducir el valor de abandonar (en este caso, 'quit'). A continuación, creamos una variable message para registrar cualquier valor que introduzca el usuario. Definimos message como una cadena vacía, "", para que Python tenga algo que comprobar la primera vez que llegue a la línea while. La primera vez que se ejecuta el programa y Python llega a la sentencia while, necesita comparar el valor de message con 'quit', pero todavía no se ha introducido ninguna entrada del usuario. Si Python no tiene nada que comparar, no podrá seguir ejecutando el programa. Para resolver este problema de , nos aseguramos de dar a message un valor inicial. Aunque sólo sea una cadena vacía, tendrá sentido para Python y le permitirá realizar la comparación que hace funcionar el bucle while. Este bucle while se ejecuta mientras el valor de message no sea 'quit'.

La primera vez que pasa por el bucle, message es sólo una cadena vacía, por lo que Python entra en el bucle. En message = input(prompt), Python muestra el prompt y espera a que el usuario introduzca sus datos. Lo que introduzca se asigna a message y se imprime; después, Python reevalúa la condición en la sentencia while. Mientras el usuario no introduzca la palabra 'quit', se vuelve a mostrar el prompt y Python espera a que introduzca más datos. Cuando finalmente el usuario introduce 'quit', Python deja de ejecutar el bucle while y el programa termina:

Tell me something, and I will repeat it back to you:  
Enter 'quit' to end the program. Hello everyone!  
Hello everyone!  
  
Tell me something, and I will repeat it back to you:  
Enter 'quit' to end the program. Hello again.  
Hello again.  
  
Tell me something, and I will repeat it back to you:  
Enter 'quit' to end the program. quit  
quit

Este programa funciona bien, excepto porque imprime la palabra 'quit' como si fuera un mensaje real. Una simple comprobación de if lo soluciona:

prompt = "\nTell me something, and I will repeat it back to you:"  
prompt += "\nEnter 'quit' to end the program. "  
  
message = ""  
while message != 'quit':  
 message = input(prompt)  
  
 if message != 'quit':  
 print(message)

Ahora el programa hace una comprobación rápida antes de mostrar el mensaje y sólo imprime el mensaje si no coincide con el valor de abandonar:

Tell me something, and I will repeat it back to you:  
Enter 'quit' to end the program. Hello everyone!  
Hello everyone!  
  
Tell me something, and I will repeat it back to you:  
Enter 'quit' to end the program. Hello again.  
Hello again.  
  
Tell me something, and I will repeat it back to you:  
Enter 'quit' to end the program. quit

### Utilizar un indicador

En el ejemplo anterior, hicimos que el programa realizara determinadas tareas mientras una condición dada fuera cierta. Pero, ¿qué ocurre con programas más complicados en los que muchos sucesos diferentes pueden hacer que el programa deje de ejecutarse?

Por ejemplo, en un juego, varios sucesos diferentes pueden poner fin a la partida. Cuando el jugador se queda sin naves, se le acaba el tiempo o las ciudades que debía proteger son destruidas, el juego debe terminar. Debe terminar si ocurre cualquiera de estos acontecimientos. Si pueden ocurrir muchos eventos posibles para detener el programa, intentar probar todas estas condiciones en una sentencia while se vuelve complicado y difícil.

Para un programa que debe ejecutarse sólo mientras muchas condiciones sean verdaderas, puedes definir una variable que determine si todo el programa está activo o no. Esta variable, llamada *flag*, actúa como una señal para el programa. Podemos escribir nuestros programas de modo que se ejecuten mientras la bandera esté establecida en True y dejen de ejecutarse cuando alguno de varios eventos establezca el valor de la bandera en False. Como resultado, nuestra declaración global while sólo necesita comprobar una condición: si la bandera está actualmente en True. Entonces, todas nuestras demás pruebas (para ver si se ha producido un evento que debería poner la bandera en False) pueden organizarse ordenadamente en el resto del programa.

Añadamos una bandera a *parrot.py* de la sección anterior. Esta bandera, que llamaremos active (aunque puedes llamarla como quieras), controlará si el programa debe seguir ejecutándose o no:

prompt = "\nTell me something, and I will repeat it back to you:"  
prompt += "\nEnter 'quit' to end the program. "  
  
active = True  
❶ while active:  
 message = input(prompt)  
  
 if message == 'quit':  
 active = False  
 else:  
 print(message)

Establecemos la variable active en True para que el programa comience en un estado activo. Hacer esto simplifica la sentencia while porque no se realiza ninguna comparación en la propia sentencia while; de la lógica se encargan otras partes del programa. Mientras la variable active siga siendo True, el bucle seguirá ejecutándose ❶.

En la sentencia if dentro del bucle while, comprobamos el valor de message una vez que el usuario introduce su entrada. Si el usuario introduce 'quit', establecemos active en False, y el bucle while se detiene. Si el usuario introduce algo distinto de 'quit', imprimimos su entrada como un mensaje.

Este programa tiene la misma salida que el ejemplo anterior, en el que colocamos la prueba condicional directamente en la sentencia while. Pero ahora que tiene una bandera para indicar si el programa en general está en estado activo, sería fácil añadir más pruebas (como las sentencias elif ) para los eventos que deberían hacer que active se convirtiera en False. Esto es útil en programas complicados como los juegos, en los que puede haber muchos eventos que deberían hacer que el programa dejara de ejecutarse. Cuando alguno de estos eventos haga que la bandera activa se convierta en False, el bucle principal del juego saldrá, se podrá mostrar un mensaje *Game Over* y se podrá dar al jugador la opción de volver a jugar.

### Usar break para salir de un bucle

Para salir inmediatamente de un bucle while sin ejecutar ningún código restante en el bucle, independientemente de los resultados de cualquier prueba condicional, utiliza la sentencia break. La sentencia break dirige el flujo de tu programa; puedes utilizarla para controlar qué líneas de código se ejecutan y cuáles no, de modo que el programa sólo ejecute el código que tú quieras, cuando tú quieras.

Por ejemplo, considera un programa que pregunta al usuario sobre los lugares que ha visitado. Podemos detener el bucle while en este programa llamando a break en cuanto el usuario introduzca el valor 'quit':

**cities.py**

prompt = "\nPlease enter the name of a city you have visited:"  
prompt += "\n(Enter 'quit' when you are finished.) "  
  
❶ while True:  
 city = input(prompt)  
  
 if city == 'quit':  
 break  
 else:  
 print(f"I'd love to go to {city.title()}!")

Un bucle que comienza con while True ❶ se ejecutará eternamente a menos que llegue a una sentencia break. El bucle de este programa sigue pidiendo al usuario que introduzca los nombres de las ciudades en las que ha estado hasta que introduce 'quit'. Cuando introduzcan 'quit', se ejecutará la sentencia break, haciendo que Python salga del bucle:

Please enter the name of a city you have visited:  
(Enter 'quit' when you are finished.) New York  
I'd love to go to New York!  
  
Please enter the name of a city you have visited:  
(Enter 'quit' when you are finished.) San Francisco  
I'd love to go to San Francisco!  
  
Please enter the name of a city you have visited:  
(Enter 'quit' when you are finished.) quit

## Nota

Puedes utilizar la sentencia break en cualquiera de los bucles de Python. Por ejemplo, podrías utilizar break para salir de un bucle for que esté trabajando a través de una lista o un diccionario.

### Utilizar continue en un bucle

En lugar de salir completamente de un bucle sin ejecutar el resto de su código, puedes utilizar la sentencia continue para volver al principio del bucle, basándote en el resultado de una prueba condicional. Por ejemplo, considera un bucle que cuenta del 1 al 10 pero imprime sólo los números impares de ese rango:

**counting.py**

current\_number = 0  
while current\_number < 10:  
❶ current\_number += 1  
 if current\_number % 2 == 0:  
 continue  
  
 print(current\_number)

Primero, ponemos current\_number a 0. Como es menor que 10, Python entra en el bucle while. Una vez dentro del bucle, incrementamos la cuenta en 1 ❶, por lo que current\_number es 1. A continuación, la sentencia if comprueba el módulo de current\_number y 2. Si el módulo es 0 (lo que significa que current\_number es divisible por 2), la sentencia continue indica a Python que ignore el resto del bucle y vuelva al principio. Si el número actual no es divisible por 2, se ejecuta el resto del bucle y Python imprime el número actual:

1  
3  
5  
7  
9

### Evitar bucles infinitos

Todo bucle while necesita una forma de dejar de ejecutarse para que no siga ejecutándose eternamente. Por ejemplo, este bucle de conteo debería contar de 1 a 5:

**counting.py**

x = 1  
while x <= 5:  
 print(x)  
 x += 1

Sin embargo, si accidentalmente omites la línea x += 1, el bucle se ejecutará eternamente:

# This loop runs forever!  
x = 1  
while x <= 5:  
 print(x)

Ahora el valor de x comenzará en 1 pero nunca cambiará. Como resultado, la prueba condicional x <= 5 siempre se evaluará a True y el bucle while se ejecutará eternamente, imprimiendo una serie de 1s, como esto:

1  
1  
1  
1  
--snip--

Todo programador escribe accidentalmente un bucle infinito while de vez en cuando, especialmente cuando los bucles de un programa tienen sutiles condiciones de salida. Si tu programa se queda atascado en un bucle infinito, pulsa CTRL-C o simplemente cierra la ventana del terminal que muestra la salida de tu programa.

Para evitar escribir bucles infinitos, prueba cada bucle de while y asegúrate de que el bucle se detiene cuando esperas que lo haga. Si quieres que tu programa termine cuando el usuario introduzca un determinado valor de entrada, ejecuta el programa e introduce ese valor. Si el programa no finaliza, examina la forma en que tu programa maneja el valor que debería provocar la salida del bucle. Asegúrate de que al menos una parte del programa puede hacer que la condición del bucle False o hacer que llegue a una sentencia break.

## Nota

VS Code, como muchos editores, muestra la salida en una ventana de terminal incrustada. Para cancelar un bucle infinito, asegúrate de hacer clic en el área de salida del editor antes de pulsar CTRL-C.

## Pruébalo tú mismo

7-4. Pizza Toppings: Escribe un bucle que pida al usuario que introduzca una serie de ingredientes de pizza hasta que introduzca un valor 'quit'. A medida que introduzcan cada ingrediente, imprime un mensaje diciendo que añadirás ese ingrediente a su pizza.

7-5. Entradas de cine: Un cine cobra diferentes precios de entrada según la edad de la persona. Si una persona tiene menos de 3 años, la entrada es gratis; si tiene entre 3 y 12 años, la entrada cuesta 10$; y si tiene más de 12 años, la entrada cuesta 15$. Escribe un bucle en el que preguntes a los usuarios su edad, y luego les digas el precio de su entrada de cine.

7-6. Tres salidas: Escribe diferentes versiones del Ejercicio 7-4 o 7-5 que hagan cada una de las siguientes cosas al menos una vez:

* Utilizar una prueba condicional en la sentencia while para detener el bucle.
* Utilizar una variable active para controlar la duración del bucle.
* Utiliza una sentencia break para salir del bucle cuando el usuario introduzca un valor 'quit'.

7-7. Infinito: Escribe un bucle que no termine nunca y ejecútalo. (Para terminar el bucle, pulsa CTRL-C o cierra la ventana que muestra la salida).

## Utilizar un bucle while con listas y diccionarios

Hasta ahora, sólo hemos trabajado con una información del usuario cada vez. Recibimos la entrada del usuario y luego imprimimos la entrada o una respuesta a ella. La siguiente vez que pasáramos por el bucle while, recibiríamos otro valor de entrada y responderíamos a él. Pero para hacer un seguimiento de muchos usuarios y piezas de información, necesitaremos utilizar listas y diccionarios con nuestros bucles while.

Un bucle for es eficaz para recorrer una lista, pero no debes modificar una lista dentro de un bucle for porque Python tendrá problemas para seguir la pista de los elementos de la lista. Para modificar una lista a medida que la recorres, utiliza un bucle while. Utilizar bucles while con listas y diccionarios te permite recopilar, almacenar y organizar montones de datos de entrada para examinarlos e informar sobre ellos más tarde.

### Mover elementos de una lista a otra

Considera una lista de usuarios recién registrados pero no verificados de un sitio web. Después de verificar a estos usuarios, ¿cómo podemos moverlos a otra lista de usuarios confirmados? Una forma sería utilizar un bucle while para extraer usuarios de la lista de usuarios no confirmados a medida que los verificamos y luego añadirlos a una lista separada de usuarios confirmados. Éste es el aspecto que podría tener ese código:

**confirmed\_users.py**

# Start with users that need to be verified,  
# and an empty list to hold confirmed users.  
❶ unconfirmed\_users = ['alice', 'brian', 'candace']  
confirmed\_users = []  
  
# Verify each user until there are no more unconfirmed users.  
# Move each verified user into the list of confirmed users.  
❷ while unconfirmed\_users:  
❸ current\_user = unconfirmed\_users.pop()  
  
 print(f"Verifying user: {current\_user.title()}")  
❹ confirmed\_users.append(current\_user)  
  
# Display all confirmed users.  
print("\nThe following users have been confirmed:")  
for confirmed\_user in confirmed\_users:  
 print(confirmed\_user.title())

Empezamos con una lista de usuarios sin confirmar ❶ (Alice, Brian y Candace) y una lista vacía para contener los usuarios confirmados. El bucle while se ejecuta mientras la lista unconfirmed\_users no esté vacía ❷. Dentro de este bucle, el método pop() elimina los usuarios no verificados de uno en uno desde el final de unconfirmed\_users ❸. Como Candace es la última de la lista unconfirmed\_users, su nombre será el primero en ser eliminado, asignado a current\_user, y añadido a la lista confirmed\_users ❹. El siguiente es Brian, y después Alice.

Simulamos la confirmación de cada usuario imprimiendo un mensaje de verificación y añadiéndolo después a la lista de usuarios confirmados. A medida que se reduce la lista de usuarios no confirmados, crece la lista de usuarios confirmados. Cuando la lista de usuarios no confirmados esté vacía, el bucle se detiene y se imprime la lista de usuarios confirmados:

Verifying user: Candace  
Verifying user: Brian  
Verifying user: Alice  
  
The following users have been confirmed:  
Candace  
Brian  
Alice

### Eliminar todas las instancias de valores específicos de una lista

En el Capítulo 3, utilizamos remove() para eliminar un valor específico de una lista. La función remove() funcionó porque el valor que nos interesaba sólo aparecía una vez en la lista. Pero, ¿qué ocurre si quieres eliminar todas las instancias de un valor de una lista?

Supongamos que tienes una lista de mascotas con el valor 'cat' repetido varias veces. Para eliminar todas las instancias de ese valor, puedes ejecutar un bucle while hasta que 'cat' deje de estar en la lista, como se muestra aquí:

**pets.py**

pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']  
print(pets)  
  
while 'cat' in pets:  
 pets.remove('cat')  
  
print(pets)

Empezamos con una lista que contiene varias instancias de 'cat'. Tras imprimir la lista, Python entra en el bucle while porque encuentra el valor 'cat' en la lista al menos una vez. Una vez dentro del bucle, Python elimina la primera instancia de 'cat', vuelve a la línea while y vuelve a entrar en el bucle cuando encuentra que 'cat' sigue en la lista. Elimina cada instancia de 'cat' hasta que el valor deje de estar en la lista, momento en el que Python sale del bucle y vuelve a imprimir la lista:

['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']  
['dog', 'dog', 'goldfish', 'rabbit']

### Rellenar un diccionario con datos introducidos por el usuario

Puedes pedir tantas entradas como necesites en cada pasada por un bucle while. Hagamos un programa de encuestas en el que en cada paso por el bucle se pida el nombre y la respuesta del participante. Almacenaremos los datos que recopilemos en un diccionario, porque queremos relacionar cada respuesta con un usuario concreto:

**mountain\_poll.py**

responses = {}  
# Set a flag to indicate that polling is active.  
polling\_active = True  
  
while polling\_active:  
 # Prompt for the person's name and response.  
❶ name = input("\nWhat is your name? ")  
 response = input("Which mountain would you like to climb someday? ")  
  
 # Store the response in the dictionary.  
❷ responses[name] = response  
  
 # Find out if anyone else is going to take the poll.  
❸ repeat = input("Would you like to let another person respond? (yes/ no) ")  
 if repeat == 'no':  
 polling\_active = False  
  
# Polling is complete. Show the results.  
print("\n--- Poll Results ---")  
❹ for name, response in responses.items():  
 print(f"{name} would like to climb {response}.")

El programa define primero un diccionario vacío (responses) y establece una bandera (polling\_active) para indicar que el sondeo está activo. Mientras polling\_active sea True, Python ejecutará el código en el bucle while.

Dentro del bucle, se pide al usuario que introduzca su nombre y una montaña que le gustaría escalar ❶. Esa información se almacena en el diccionario responses ❷, y se pregunta al usuario si desea o no que se siga ejecutando el sondeo ❸. Si introducen yes, el programa entra de nuevo en el bucle while. Si introduce no, la bandera polling\_active se pone a False, el bucle while deja de ejecutarse, y el bloque de código final ❹ muestra los resultados de la encuesta.

Si ejecutas este programa e introduces respuestas de ejemplo, deberías ver una salida como ésta:

What is your name? Eric  
Which mountain would you like to climb someday? Denali  
Would you like to let another person respond? (yes/ no) yes  
  
What is your name? Lynn  
Which mountain would you like to climb someday? Devil's Thumb  
Would you like to let another person respond? (yes/ no) no  
  
--- Poll Results ---  
Eric would like to climb Denali.  
Lynn would like to climb Devil's Thumb.

## Pruébalo tú mismo

7-8. Deli: Haz una lista llamada sandwich\_orders y rellénala con los nombres de varios bocadillos. Luego haz una lista vacía llamada finished\_sandwiches. Recorre la lista de pedidos de bocadillos e imprime un mensaje para cada pedido, como I made your tuna sandwich. A medida que se hace cada bocadillo, muévelo a la lista de bocadillos terminados. Cuando se hayan hecho todos los bocadillos, imprime un mensaje con la lista de cada bocadillo hecho.

7-9. No hay Pastrami: Utilizando la lista sandwich\_orders del Ejercicio 7-8, asegúrate de que el bocadillo 'pastrami' aparece en la lista al menos tres veces. Añade código cerca del principio de tu programa para imprimir un mensaje diciendo que la charcutería se ha quedado sin pastrami, y luego utiliza un bucle while para eliminar todas las apariciones de 'pastrami' de sandwich\_orders. Asegúrate de que ningún bocadillo de pastrami acaba en finished\_sandwiches.

7-10. Vacaciones de ensueño: Escribe un programa que pregunte a los usuarios por sus vacaciones soñadas. Escribe una pregunta similar a *If you could visit one place in the world, where would you go?* Incluye un bloque de código que imprima los resultados de la encuesta.

## Resumen

En este capítulo, has aprendido a utilizar input() para permitir que los usuarios proporcionen su propia información en tus programas. Aprendiste a trabajar tanto con entradas de texto como numéricas y a utilizar los bucles while para que tus programas se ejecuten tanto tiempo como quieran tus usuarios. Viste varias formas de controlar el flujo de un bucle while estableciendo una bandera active, utilizando la sentencia break y utilizando la sentencia continue. Aprendiste a utilizar un bucle while para mover elementos de una lista a otra y a eliminar todas las instancias de un valor de una lista. También aprendiste cómo se pueden utilizar los bucles while con los diccionarios.

En el Capítulo 8 aprenderás sobre las funciones. *Functions* te permiten dividir tus programas en pequeñas partes, cada una de las cuales realiza un trabajo específico. Puedes llamar a una función tantas veces como quieras, y puedes almacenar tus funciones en archivos separados. Utilizando funciones, podrás escribir un código más eficiente, más fácil de solucionar y mantener, y que puede reutilizarse en muchos programas diferentes.