# 6 Diccionarios



En este capítulo aprenderás a utilizar los diccionarios de Python, que te permiten conectar piezas de información relacionada. Aprenderás cómo acceder a la información una vez que está en un diccionario y cómo modificarla. Dado que los diccionarios pueden almacenar una cantidad casi ilimitada de información, te mostraré cómo recorrer en bucle los datos de un diccionario. Además, aprenderás a anidar diccionarios dentro de listas, listas dentro de diccionarios e incluso diccionarios dentro de otros diccionarios.

Comprender los diccionarios te permitirá modelar con mayor precisión diversos objetos del mundo real. Podrás crear un diccionario que represente a una persona y almacenar toda la información que quieras sobre ella. Podrás almacenar su nombre, edad, ubicación, profesión y cualquier otro aspecto de una persona que puedas describir. Podrás almacenar dos tipos cualesquiera de información que puedan emparejarse, como una lista de palabras y sus significados, una lista de nombres de personas y sus números favoritos, una lista de montañas y sus elevaciones, etc.

## Un diccionario sencillo

Considera un juego con extraterrestres que pueden tener diferentes colores y valores de puntos. Este sencillo diccionario almacena información sobre un alienígena concreto:

**alien.py**

alien\_0 = {'color': 'green', 'points': 5}  
  
print(alien\_0['color'])  
print(alien\_0['points'])

El diccionario alien\_0 almacena el color y el valor en puntos del alienígena. Las dos últimas líneas acceden a esa información y la muestran, como se muestra aquí:

green  
5

Como ocurre con la mayoría de los nuevos conceptos de programación, utilizar diccionarios requiere práctica. Cuando hayas trabajado un poco con los diccionarios, verás la eficacia con la que pueden modelar situaciones del mundo real.

## Trabajar con diccionarios

Un *dictionary* en Python es una colección de *key-value pairs*. Cada *key* está conectado a un valor, y puedes utilizar una clave para acceder al valor asociado a esa clave. El valor de una clave puede ser un número, una cadena, una lista o incluso otro diccionario. De hecho, puedes utilizar cualquier objeto que puedas crear en Python como valor en un diccionario.

En Python, un diccionario se encierra entre llaves ({}) con una serie de pares clave-valor dentro de las llaves, como se muestra en el ejemplo anterior:

alien\_0 = {'color': 'green', 'points': 5}

Un *key-value pair* es un conjunto de valores asociados entre sí. Cuando proporcionas una clave, Python devuelve el valor asociado a esa clave. Cada clave está unida a su valor por dos puntos, y los pares clave-valor individuales están separados por comas. Puedes almacenar tantos pares clave-valor como quieras en un diccionario.

El diccionario más sencillo tiene exactamente un par clave-valor, como se muestra en esta versión modificada del diccionario alien\_0:

alien\_0 = {'color': 'green'}

Este diccionario almacena una información sobre alien\_0: el color del extraterrestre. La cadena 'color' es una clave de este diccionario, y su valor asociado es 'green'.

### Acceder a los valores de un diccionario

Para obtener el valor asociado a una clave, indica el nombre del diccionario y, a continuación, coloca la clave dentro de un conjunto de corchetes, como se muestra aquí:

**alien.py**

alien\_0 = {'color': 'green'}  
print(alien\_0['color'])

Esto devuelve el valor asociado a la clave 'color' del diccionario alien\_0:

green

Puedes tener un número ilimitado de pares clave-valor en un diccionario. Por ejemplo, aquí tienes el diccionario original alien\_0 con dos pares clave-valor:

alien\_0 = {'color': 'green', 'points': 5}

Ahora puedes acceder al color o al valor en puntos de alien\_0. Si un jugador derriba a este alienígena, puedes consultar cuántos puntos debe ganar utilizando un código como éste:

alien\_0 = {'color': 'green', 'points': 5}  
  
new\_points = alien\_0['points']  
print(f"You just earned {new\_points} points!")

Una vez definido el diccionario, extraemos del diccionario el valor asociado a la clave 'points'. Este valor se asigna a la variable new\_points. La última línea imprime una declaración sobre cuántos puntos acaba de ganar el jugador:

You just earned 5 points!

Si ejecutas este código cada vez que se derriba a un alienígena, se recuperará el valor en puntos del alienígena.

### Añadir nuevos pares clave-valor

Los diccionarios son estructuras dinámicas, y puedes añadir nuevos pares clave-valor a un diccionario en cualquier momento. Para añadir un nuevo par clave-valor, debes indicar el nombre del diccionario seguido de la nueva clave entre corchetes, junto con el nuevo valor.

Vamos a añadir dos nuevos datos al diccionario alien\_0: las coordenadas *x*- y *y*- del alienígena, que nos ayudarán a mostrar al alienígena en una posición determinada de la pantalla. Vamos a colocar al robot en el borde izquierdo de la pantalla, 25 píxeles por debajo de la parte superior. Como las coordenadas de pantalla suelen empezar en la esquina superior izquierda de la pantalla, situaremos al robot en el borde izquierdo de la pantalla poniendo la -coordenada *x* a 0 y a 25 píxeles de la parte superior poniendo su -coordenada *y* a 25 positivo, como se muestra aquí:

**alien.py**

alien\_0 = {'color': 'green', 'points': 5}  
print(alien\_0)  
  
alien\_0['x\_position'] = 0  
alien\_0['y\_position'] = 25  
print(alien\_0)

Empezamos definiendo el mismo diccionario con el que hemos estado trabajando. A continuación, imprimimos este diccionario, mostrando una instantánea de su información. A continuación, añadimos un nuevo par clave-valor al diccionario: la clave 'x\_position' y el valor 0. Hacemos lo mismo con la clave 'y\_position'. Cuando imprimimos el diccionario modificado, vemos los dos pares clave-valor adicionales:

{'color': 'green', 'points': 5}  
{'color': 'green', 'points': 5, 'x\_position': 0, 'y\_position': 25}

La versión final del diccionario contiene cuatro pares clave-valor. Los dos originales especifican el color y el valor del punto, y dos más especifican la posición del alienígena.

Los diccionarios conservan el orden en que fueron definidos. Cuando imprimas un diccionario o hagas un bucle a través de sus elementos, verás los elementos en el mismo orden en que se añadieron al diccionario.

### Empezar con un diccionario vacío

A veces es conveniente, o incluso necesario, empezar con un diccionario vacío y luego añadirle cada nuevo elemento. Para empezar a rellenar un diccionario vacío, define un diccionario con un conjunto de llaves vacío y luego añade cada par clave-valor en su propia línea. Por ejemplo, así es como se construye el diccionario alien\_0 utilizando este método:

**alien.py**

alien\_0 = {}  
  
alien\_0['color'] = 'green'  
alien\_0['points'] = 5  
  
print(alien\_0)

Primero definimos un diccionario alien\_0 vacío, y luego le añadimos los valores de color y punto. El resultado es el diccionario que hemos utilizado en los ejemplos anteriores:

{'color': 'green', 'points': 5}

Normalmente, utilizarás diccionarios vacíos cuando almacenes datos suministrados por el usuario en un diccionario o cuando escribas código que genere un gran número de pares clave-valor automáticamente.

### Modificar valores de un diccionario

Para modificar un valor de un diccionario, indica el nombre del diccionario con la clave entre corchetes y, a continuación, el nuevo valor que quieres asociar a esa clave. Por ejemplo, considera un alienígena que cambia de verde a amarillo a medida que avanza el juego:

**alien.py**

alien\_0 = {'color': 'green'}  
print(f"The alien is {alien\_0['color']}.")  
  
alien\_0['color'] = 'yellow'  
print(f"The alien is now {alien\_0['color']}.")

Primero definimos un diccionario para alien\_0 que contenga sólo el color del alienígena; luego cambiamos el valor asociado a la clave 'color' por 'yellow'. La salida muestra que, efectivamente, el alienígena ha cambiado de verde a amarillo:

The alien is green.  
The alien is now yellow.

Para un ejemplo más interesante, rastreemos la posición de un alienígena que puede moverse a distintas velocidades. Almacenaremos un valor que represente la velocidad actual del robot y lo utilizaremos para determinar la distancia a la derecha a la que debe desplazarse:

alien\_0 = {'x\_position': 0, 'y\_position': 25, 'speed': 'medium'}  
print(f"Original position: {alien\_0['x\_position']}")  
  
# Move the alien to the right.  
# Determine how far to move the alien based on its current speed.  
❶ if alien\_0['speed'] == 'slow':  
 x\_increment = 1  
elif alien\_0['speed'] == 'medium':  
 x\_increment = 2  
else:  
 # This must be a fast alien.  
 x\_increment = 3  
  
# The new position is the old position plus the increment.  
❷ alien\_0['x\_position'] = alien\_0['x\_position'] + x\_increment  
  
print(f"New position: {alien\_0['x\_position']}")

Comenzamos definiendo un alienígena con una posición inicial de *x* y una posición de *y*, y una velocidad de 'medium'. Hemos omitido los valores de color y punto para simplificar, pero este ejemplo funcionaría igual si incluyeras también esos pares clave-valor. También imprimimos el valor original de x\_position para ver cuánto se desplaza el extraterrestre hacia la derecha.

Una cadena if-elif-else determina a qué distancia debe moverse el alienígena hacia la derecha, y asigna este valor a la variable x\_increment ❶. Si la velocidad del alienígena es 'slow', se desplaza una unidad a la derecha; si la velocidad es 'medium', se desplaza dos unidades a la derecha; y si es 'fast', se desplaza tres unidades a la derecha. Una vez calculado el incremento, se suma al valor de x\_position ❷, y el resultado se almacena en el diccionario x\_position.

Como se trata de un alienígena de velocidad media, su posición se desplaza dos unidades a la derecha:

Original x-position: 0  
New x-position: 2

Esta técnica es bastante chula: cambiando un valor del diccionario del alienígena, puedes cambiar el comportamiento general del alienígena. Por ejemplo, para convertir a este alienígena de velocidad media en un alienígena rápido, añadirías esta línea:

alien\_0['speed'] = 'fast'

El bloque if-elif-else asignaría entonces un valor mayor a x\_increment la próxima vez que se ejecute el código.

### Eliminar pares clave-valor

Cuando ya no necesites una información almacenada en un diccionario, puedes utilizar la sentencia del para eliminar por completo un par clave-valor. Todo lo que necesita del es el nombre del diccionario y la clave que quieres eliminar.

Por ejemplo, vamos a eliminar la clave 'points' del diccionario alien\_0, junto con su valor:

**alien.py**

alien\_0 = {'color': 'green', 'points': 5}  
print(alien\_0)  
  
❶ del alien\_0['points']  
print(alien\_0)

La sentencia del ❶ le dice a Python que elimine la clave 'points' del diccionario alien\_0 y que elimine también el valor asociado a esa clave. La salida muestra que la clave 'points' y su valor 5 se eliminan del diccionario, pero el resto del diccionario no se ve afectado:

{'color': 'green', 'points': 5}  
{'color': 'green'}

## Nota

Ten en cuenta que el par clave-valor borrado se elimina permanentemente.

### Un diccionario de objetos similares

El ejemplo anterior consistía en almacenar distintos tipos de información sobre un objeto, un alienígena en un juego. También puedes utilizar un diccionario para almacenar un tipo de información sobre muchos objetos. Por ejemplo, supongamos que quieres hacer una encuesta a varias personas y preguntarles cuál es su lenguaje de programación favorito. Un diccionario es útil para almacenar los resultados de una encuesta sencilla, como ésta:

**favorite\_languages.py**

favorite\_languages = {  
 'jen': 'python',  
 'sarah': 'c',  
 'edward': 'rust',  
 'phil': 'python',  
 }

Como puedes ver, hemos dividido un diccionario más grande en varias líneas. Cada clave es el nombre de una persona que respondió a la encuesta, y cada valor es su elección de lenguaje. Cuando sepas que vas a necesitar más de una línea para definir un diccionario, pulsa INTRO después de la llave de apertura. A continuación, aplica una sangría de un nivel (cuatro espacios) a la línea siguiente y escribe el primer par clave-valor, seguido de una coma. A partir de este momento, cuando pulses INTRO, tu editor de texto debería sangrar automáticamente todos los pares clave-valor siguientes para que coincidan con el primer par clave-valor.

Cuando hayas terminado de definir el diccionario, añade un corchete de cierre en una nueva línea después del último par clave-valor, y aplica una sangría de un nivel para que se alinee con las claves del diccionario. Es una buena práctica incluir también una coma después del último par clave-valor, para que estés preparado para añadir un nuevo par clave-valor en la línea siguiente.

## Nota

La mayoría de los editores tienen alguna funcionalidad que te ayuda a dar formato a las listas y diccionarios extensos de forma similar a este ejemplo. También existen otras formas aceptables de dar formato a los diccionarios largos, por lo que es posible que veas un formato ligeramente distinto en tu editor o en otras fuentes.

Para utilizar este diccionario, dado el nombre de una persona que haya participado en la encuesta, puedes buscar fácilmente su lengua favorita:

**favorite\_languages.py**

favorite\_languages = {  
 'jen': 'python',  
 'sarah': 'c',  
 'edward': 'rust',  
 'phil': 'python',  
 }  
  
❶ language = favorite\_languages['sarah'].title()  
print(f"Sarah's favorite language is {language}.")

Para ver qué lengua eligió Sarah, pedimos el valor en:

favorite\_languages['sarah']

Utilizamos esta sintaxis para sacar la lengua favorita de Sarah del diccionario ❶ y asignarla a la variable language. Al crear una nueva variable, la llamada a print() es mucho más limpia. El resultado muestra el idioma favorito de Sara:

Sarah's favorite language is C.

Podrías utilizar esta misma sintaxis con cualquier individuo representado en el diccionario.

### Utilizar get() para acceder a los valores

Utilizar claves entre corchetes para recuperar el valor que te interesa de un diccionario puede causar un problema potencial: si la clave que pides no existe, obtendrás un error.

Veamos qué ocurre cuando pides el valor en puntos de un alienígena que no tiene un valor en puntos establecido:

**alien\_no\_points.py**

alien\_0 = {'color': 'green', 'speed': 'slow'}  
print(alien\_0['points'])

Esto da lugar a un rastreo, que muestra un KeyError:

Traceback (most recent call last):  
 File "alien\_no\_points.py", line 2, in <module>  
 print(alien\_0['points'])  
 ~~~~~~~^^^^^^^^^^  
KeyError: 'points'

En el Capítulo 10 aprenderás más sobre cómo tratar errores como éste en general. En el caso concreto de los diccionarios, puedes utilizar el método get() para establecer un valor por defecto que se devolverá si la clave solicitada no existe.

El método get() requiere una clave como primer argumento. Como segundo argumento opcional, puedes pasar el valor que se devolverá si la clave no existe:

alien\_0 = {'color': 'green', 'speed': 'slow'}  
  
point\_value = alien\_0.get('points', 'No point value assigned.')  
print(point\_value)

Si la clave 'points' existe en el diccionario, obtendrás el valor correspondiente. Si no existe, obtendrás el valor por defecto. En este caso, points no existe, y obtenemos un mensaje limpio en lugar de un error:

No point value assigned.

Si existe la posibilidad de que la clave que pides no exista, considera la posibilidad de utilizar el método get() en lugar de la notación de corchetes.

## Nota

Si omites el segundo argumento en la llamada a get() y la clave no existe, Python devolverá el valor None. El valor especial None significa "no existe ningún valor". No es un error: es un valor especial para indicar la ausencia de un valor. Verás más usos de None en el Capítulo 8.

## Pruébalo tú mismo

6-1. Persona: Utiliza un diccionario para almacenar información sobre una persona que conozcas. Almacena su nombre, apellidos, edad y la ciudad en la que vive. Deberías tener claves como first\_name, last\_name, age, y city. Imprime cada dato almacenado en tu diccionario.

6-2. Números favoritos: Utiliza un diccionario para almacenar los números favoritos de la gente. Piensa en cinco nombres y utilízalos como claves en tu diccionario. Piensa en un número favorito para cada persona, y almacena cada uno como un valor en tu diccionario. Imprime el nombre de cada persona y su número favorito. Para divertirte aún más, sondea a unos cuantos amigos y obtén datos reales para tu programa.

6-3. Glosario: Un diccionario Python puede utilizarse para modelar un diccionario real. Sin embargo, para evitar confusiones, vamos a llamarlo glosario.

* Piensa en cinco palabras de programación que hayas aprendido en los capítulos anteriores. Utiliza estas palabras como claves de tu glosario, y almacena sus significados como valores.
* Imprime cada palabra y su significado como salida con un formato ordenado. Puedes imprimir la palabra seguida de dos puntos y luego su significado, o imprimir la palabra en una línea y luego imprimir su significado sangrado en una segunda línea. Utiliza el carácter de nueva línea (\n) para insertar una línea en blanco entre cada par palabra-significado.

## Recorrer un diccionario

Un diccionario Python puede contener unos pocos pares clave-valor o millones de pares. Como un diccionario puede contener grandes cantidades de datos, Python te permite recorrerlo en bucle. Los diccionarios se pueden utilizar para almacenar información de diversas formas, por lo que existen varias maneras de recorrerlos. Puedes recorrer todos los pares clave-valor de un diccionario, sus claves o sus valores.

### Recorrer todos los pares clave-valor

Antes de explorar los distintos enfoques del bucle, consideremos un nuevo diccionario diseñado para almacenar información sobre un usuario en un sitio web. El siguiente diccionario almacenaría el nombre de usuario, el nombre y los apellidos de una persona:

**user.py**

user\_0 = {  
 'username': 'efermi',  
 'first': 'enrico',  
 'last': 'fermi',  
 }

Puedes acceder a cualquier dato de user\_0 basándote en lo que ya has aprendido en este capítulo. Pero, ¿y si quisieras ver todo lo almacenado en el diccionario de este usuario? Para ello, podrías recorrer el diccionario utilizando un bucle for:

user\_0 = {  
 'username': 'efermi',  
 'first': 'enrico',  
 'last': 'fermi',  
 }  
  
for key, value in user\_0.items():  
 print(f"\nKey: {key}")  
 print(f"Value: {value}")

Para escribir un bucle for para un diccionario, crea nombres para las dos variables que contendrán la clave y el valor de cada par clave-valor. Puedes elegir los nombres que quieras para estas dos variables. Este código funcionaría igual de bien si hubieras utilizado abreviaturas para los nombres de las variables, como ésta:

for k, v in user\_0.items()

La segunda mitad de la sentencia for incluye el nombre del diccionario seguido del método items(), que devuelve una secuencia de pares clave-valor. A continuación, el bucle for asigna cada uno de estos pares a las dos variables proporcionadas. En el ejemplo anterior, utilizamos las variables para imprimir cada key, seguido del value asociado. El "\n" de la primera llamada a print() garantiza que se inserte una línea en blanco antes de cada par clave-valor en la salida:

Key: username  
Value: efermi  
  
Key: first  
Value: enrico  
  
Key: last  
Value: fermi

El bucle a través de todos los pares clave-valor funciona especialmente bien para diccionarios como el ejemplo *favorite\_languages.py* de la página 96, que almacena el mismo tipo de información para muchas claves diferentes. Si recorres el diccionario favorite\_languages, obtendrás el nombre de cada persona del diccionario y su lenguaje de programación favorito. Como las claves siempre se refieren al nombre de una persona y el valor siempre es un lenguaje, utilizaremos las variables name y language en el bucle en lugar de key y value. Así será más fácil seguir lo que ocurre dentro del bucle:

**favorite\_languages.py**

favorite\_languages = {  
 'jen': 'python',  
 'sarah': 'c',  
 'edward': 'rust',  
 'phil': 'python',  
 }  
  
for name, language in favorite\_languages.items():  
 print(f"{name.title()}'s favorite language is {language.title()}.")

Este código le dice a Python que recorra cada par clave-valor del diccionario. A medida que recorre cada par, la clave se asigna a la variable name, y el valor se asigna a la variable language. Estos nombres descriptivos hacen que sea mucho más fácil ver lo que hace la llamada a print().

Ahora, en unas pocas líneas de código, podemos mostrar toda la información del sondeo:

Jen's favorite language is Python.  
Sarah's favorite language is C.  
Edward's favorite language is Rust.  
Phil's favorite language is Python.

Este tipo de bucle funcionaría igual de bien si nuestro diccionario almacenara los resultados de encuestar a mil o incluso a un millón de personas.

### Recorrer todas las claves de un diccionario

El método keys() es útil cuando no necesitas trabajar con todos los valores de un diccionario. Hagamos un bucle en el diccionario favorite\_languages e imprimamos los nombres de todas las personas que han participado en la encuesta:

favorite\_languages = {  
 'jen': 'python',  
 'sarah': 'c',  
 'edward': 'rust',  
 'phil': 'python',  
 }  
  
for name in favorite\_languages.keys():  
 print(name.title())

Este bucle for le dice a Python que extraiga todas las claves del diccionario favorite\_languages y las asigne de una en una a la variable name. La salida muestra los nombres de todos los participantes en la encuesta:

Jen  
Sarah  
Edward  
Phil

El bucle a través de las claves es en realidad el comportamiento por defecto cuando se hace un bucle a través de un diccionario, por lo que este código tendría exactamente el mismo resultado si escribieras:

for name in favorite\_languages:

en lugar de:

for name in favorite\_languages.keys():

Puedes elegir utilizar el método keys() explícitamente si facilita la lectura de tu código, u omitirlo si lo deseas.

Puedes acceder al valor asociado a cualquier clave que te interese dentro del bucle, utilizando la clave actual. Imprimamos un mensaje a un par de amigos sobre las lenguas que eligieron. Haremos un bucle a través de los nombres en el diccionario como hicimos anteriormente, pero cuando el nombre coincida con uno de nuestros amigos, mostraremos un mensaje sobre su lengua favorita:

favorite\_languages = {  
 --snip--  
 }  
  
friends = ['phil', 'sarah']  
for name in favorite\_languages.keys():  
 print(f"Hi {name.title()}.")  
  
❶ if name in friends:  
❷ language = favorite\_languages[name].title()  
 print(f"\t{name.title()}, I see you love {language}!")

En primer lugar, hacemos una lista de amigos a los que queremos imprimir un mensaje. Dentro del bucle, imprimimos el nombre de cada persona. Luego comprobamos si el name con el que estamos trabajando está en la lista friends ❶. Si lo está, determinamos el idioma favorito de la persona utilizando el nombre del diccionario y el valor actual de name como clave ❷. A continuación, imprimimos un saludo especial, incluyendo una referencia a su lengua de elección.

Se imprime el nombre de todos, pero nuestros amigos reciben un mensaje especial:

Hi Jen.  
Hi Sarah.  
 Sarah, I see you love C!  
Hi Edward.  
Hi Phil.  
 Phil, I see you love Python!

También puedes utilizar el método keys() para averiguar si se ha encuestado a una persona concreta. Esta vez, vamos a averiguar si Erin participó en la encuesta:

favorite\_languages = {  
 --snip--  
 }  
  
if 'erin' not in favorite\_languages.keys():  
 print("Erin, please take our poll!")

El método keys() no es sólo para hacer bucles: en realidad devuelve una secuencia de todas las claves, y la sentencia if simplemente comprueba si 'erin' está en esta secuencia. Como no lo está, se imprime un mensaje invitándola a hacer la encuesta:

Erin, please take our poll!

### Recorrer las claves de un diccionario en un orden determinado

Recorrer un diccionario en bucle devuelve los elementos en el mismo orden en que se insertaron. A veces, sin embargo, querrás recorrer un diccionario en un orden diferente.

Una forma de hacerlo es ordenar las claves a medida que se devuelven en el bucle for. Puedes utilizar la función sorted() para obtener una copia de las claves en orden:

favorite\_languages = {  
 'jen': 'python',  
 'sarah': 'c',  
 'edward': 'rust',  
 'phil': 'python',  
 }  
  
for name in sorted(favorite\_languages.keys()):  
 print(f"{name.title()}, thank you for taking the poll.")

Esta sentencia for es como otras sentencias for, salvo que hemos envuelto la función sorted() alrededor del método dictionary.keys(). Esto le dice a Python que obtenga todas las claves del diccionario y las ordene antes de iniciar el bucle. La salida muestra a todos los que han participado en la encuesta, con los nombres ordenados:

Edward, thank you for taking the poll.  
Jen, thank you for taking the poll.  
Phil, thank you for taking the poll.  
Sarah, thank you for taking the poll.

### Recorrer en bucle todos los valores de un diccionario

Si te interesan principalmente los valores que contiene un diccionario, puedes utilizar el método values() para devolver una secuencia de valores sin ninguna clave. Por ejemplo, digamos que simplemente queremos una lista de todos los idiomas elegidos en nuestra encuesta sobre lenguajes de programación, sin el nombre de la persona que eligió cada idioma:

favorite\_languages = {  
 'jen': 'python',  
 'sarah': 'c',  
 'edward': 'rust',  
 'phil': 'python',  
 }  
  
print("The following languages have been mentioned:")  
for language in favorite\_languages.values():  
 print(language.title())

La sentencia for extrae aquí cada valor del diccionario y lo asigna a la variable language. Cuando se imprimen estos valores, obtenemos una lista de todos los lenguajes elegidos:

The following languages have been mentioned:  
Python  
C  
Rust  
Python

Este método extrae todos los valores del diccionario sin comprobar si hay repeticiones. Esto podría funcionar bien con un número pequeño de valores, pero en una encuesta con un gran número de encuestados, daría como resultado una lista muy repetitiva. Para ver cada lengua elegida sin repeticiones, podemos utilizar un conjunto. Un *set* es una colección en la que cada elemento debe ser único:

favorite\_languages = {  
 --snip--  
 }  
  
print("The following languages have been mentioned:")  
for language in set(favorite\_languages.values()):  
 print(language.title())

Cuando envuelves set() alrededor de una colección de valores que contiene elementos duplicados, Python identifica los elementos únicos de la colección y construye un conjunto a partir de esos elementos. Aquí utilizamos set() para extraer los lenguajes únicos de favorite\_languages.values().

El resultado es una lista no repetitiva de los idiomas que han mencionado las personas que han hecho la encuesta:

The following languages have been mentioned:  
Python  
C  
Rust

A medida que sigas aprendiendo sobre Python, a menudo encontrarás una función integrada en el lenguaje que te ayudará a hacer exactamente lo que quieres con tus datos.

## Nota

Puedes construir un conjunto directamente utilizando llaves y separando los elementos con comas:

>>> languages = {'python', 'rust', 'python', 'c'}  
>>> languages  
{'rust', 'python', 'c'}

Es fácil confundir los conjuntos con los diccionarios porque ambos están envueltos en llaves. Cuando veas llaves pero no pares clave-valor, probablemente estés ante un conjunto. A diferencia de las listas y los diccionarios, los conjuntos no conservan los elementos en ningún orden específico.

## Pruébalo tú mismo

6-4. Glosario 2: Ahora que sabes cómo hacer un bucle a través de un diccionario, limpia el código del Ejercicio 6-3(página 99) sustituyendo tu serie de llamadas a print() por un bucle que recorra las claves y valores del diccionario. Cuando estés seguro de que tu bucle funciona, añade cinco términos Python más a tu glosario. Cuando vuelvas a ejecutar tu programa, estas nuevas palabras y significados deberían incluirse automáticamente en la salida.

6-5. Ríos: Haz un diccionario que contenga tres ríos principales y el país por el que pasa cada río. Un par clave-valor podría ser 'nile': 'egypt'.

* Utiliza un bucle para imprimir una frase sobre cada río, como *The Nile runs through Egypt*.
* Utiliza un bucle para imprimir el nombre de cada río incluido en el diccionario.
* Utiliza un bucle para imprimir el nombre de cada país incluido en el diccionario.

6-6. Sondeo: Utiliza el código de *favorite\_languages.py* (página 96).

* Haz una lista de las personas que deberían participar en la encuesta de lenguas favoritas. Incluye algunos nombres que ya estén en el diccionario y otros que no.
* Recorre la lista de personas que deberían participar en la encuesta. Si ya han participado en la encuesta, imprime un mensaje dándoles las gracias por responder. Si aún no han respondido, imprime un mensaje invitándoles a hacerlo.

## Anidar

A veces querrás almacenar varios diccionarios en una lista, o una lista de elementos como valor en un diccionario. Esto se llama *nesting*. Puedes anidar diccionarios dentro de una lista, una lista de elementos dentro de un diccionario, o incluso un diccionario dentro de otro diccionario. El anidamiento es una función muy potente, como demostrarán los siguientes ejemplos.

### Una lista de diccionarios

El diccionario alien\_0 contiene información variada sobre un alienígena, pero no tiene espacio para almacenar información sobre un segundo alienígena, y mucho menos sobre una pantalla llena de alienígenas. ¿Cómo puedes gestionar una flota de alienígenas? Una forma es hacer una lista de alienígenas en la que cada alienígena sea un diccionario de información sobre ese alienígena. Por ejemplo, el siguiente código construye una lista de tres alienígenas:

**aliens.py**

alien\_0 = {'color': 'green', 'points': 5}  
alien\_1 = {'color': 'yellow', 'points': 10}  
alien\_2 = {'color': 'red', 'points': 15}  
  
❶ aliens = [alien\_0, alien\_1, alien\_2]  
  
for alien in aliens:  
 print(alien)

Primero creamos tres diccionarios, cada uno de los cuales representa a un alienígena diferente. Almacenamos cada uno de estos diccionarios en una lista llamada aliens ❶. Por último, recorremos la lista e imprimimos cada alienígena:

{'color': 'green', 'points': 5}  
{'color': 'yellow', 'points': 10}  
{'color': 'red', 'points': 15}

Un ejemplo más realista incluiría más de tres alienígenas con un código que generara automáticamente cada alienígena. En el siguiente ejemplo, utilizamos range() para crear una flota de 30 alienígenas:

# Make an empty list for storing aliens.  
aliens = []  
  
# Make 30 green aliens.  
❶ for alien\_number in range(30):  
❷ new\_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}  
❸ aliens.append(new\_alien)  
  
# Show the first 5 aliens.  
❹ for alien in aliens[:5]:  
 print(alien)  
print("...")  
  
# Show how many aliens have been created.  
print(f"Total number of aliens: {len(aliens)}")

Este ejemplo comienza con una lista vacía para contener todos los alienígenas que se crearán. La función range() ❶ devuelve una serie de números, que sólo indican a Python cuántas veces queremos que se repita el bucle. Cada vez que se ejecuta el bucle, creamos un nuevo alien ❷ y luego añadimos cada nuevo alien a la lista aliens ❸. Utilizamos una rebanada para imprimir los cinco primeros alienígenas ❹, y finalmente, imprimimos la longitud de la lista para demostrar que realmente hemos generado la flota completa de 30 alienígenas:

{'color': 'green', 'points': 5, 'speed': 'slow'}  
{'color': 'green', 'points': 5, 'speed': 'slow'}  
{'color': 'green', 'points': 5, 'speed': 'slow'}  
{'color': 'green', 'points': 5, 'speed': 'slow'}  
{'color': 'green', 'points': 5, 'speed': 'slow'}  
...  
  
Total number of aliens: 30

Todos estos alienígenas tienen las mismas características, pero Python considera a cada uno un objeto independiente, lo que nos permite modificar cada alienígena individualmente.

¿Cómo podrías trabajar con un grupo de alienígenas como éste? Imagina que en un aspecto del juego algunos alienígenas cambian de color y se mueven más rápido a medida que avanza la partida. Cuando llegue el momento de cambiar los colores, podemos utilizar un bucle for y una sentencia if para cambiar el color de los alienígenas. Por ejemplo, para cambiar los tres primeros aliens a amarillo, aliens de velocidad media que valen 10 puntos cada uno, podríamos hacer esto:

# Make an empty list for storing aliens.  
aliens = []  
  
# Make 30 green aliens.  
for alien\_number in range (30):  
 new\_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}  
 aliens.append(new\_alien)  
  
for alien in aliens[:3]:  
 if alien['color'] == 'green':  
 alien['color'] = 'yellow'  
 alien['speed'] = 'medium'  
 alien['points'] = 10  
  
# Show the first 5 aliens.  
for alien in aliens[:5]:  
 print(alien)  
print("...")

Como queremos modificar los tres primeros alienígenas, hacemos un bucle a través de una porción que incluya sólo a los tres primeros alienígenas. Ahora todos los alienígenas son verdes, pero no siempre será así, así que escribimos una sentencia if para asegurarnos de que sólo modificamos los alienígenas verdes. Si el alienígena es verde, cambiamos el color a 'yellow', la velocidad a 'medium', y el valor en puntos a 10, como se muestra en la siguiente salida:

{'color': 'yellow', 'points': 10, 'speed': 'medium'}  
{'color': 'yellow', 'points': 10, 'speed': 'medium'}  
{'color': 'yellow', 'points': 10, 'speed': 'medium'}  
{'color': 'green', 'points': 5, 'speed': 'slow'}  
{'color': 'green', 'points': 5, 'speed': 'slow'}  
...

Podrías ampliar este bucle añadiendo un bloque elif que convierta a los alienígenas amarillos en rojos y rápidos, con un valor de 15 puntos cada uno. Sin mostrar de nuevo todo el programa, ese bucle tendría el siguiente aspecto:

for alien in aliens[0:3]:  
 if alien['color'] == 'green':  
 alien['color'] = 'yellow'  
 alien['speed'] = 'medium'  
 alien['points'] = 10  
 elif alien['color'] == 'yellow':  
 alien['color'] = 'red'  
 alien['speed'] = 'fast'  
 alien['points'] = 15

Es habitual almacenar varios diccionarios en una lista cuando cada diccionario contiene muchos tipos de información sobre un objeto. Por ejemplo, podrías crear un diccionario para cada usuario de un sitio web, como hicimos en *user.py*  en la página 99, y almacenar los diccionarios individuales en una lista llamada users. Todos los diccionarios de la lista deben tener una estructura idéntica, para que puedas recorrer la lista y trabajar con cada objeto diccionario de la misma manera.

### Una lista dentro de un diccionario

En lugar de poner un diccionario dentro de una lista, a veces es útil poner una lista dentro de un diccionario. Por ejemplo, considera cómo podrías describir una pizza que alguien está pidiendo. Si sólo utilizaras una lista, lo único que podrías almacenar sería una lista de los ingredientes de la pizza. Con un diccionario, una lista de ingredientes puede ser sólo un aspecto de la pizza que estás describiendo.

En el siguiente ejemplo, se almacenan dos tipos de información para cada pizza: un tipo de masa y una lista de ingredientes. La lista de ingredientes es un valor asociado a la clave 'toppings'. Para utilizar los elementos de la lista, damos el nombre del diccionario y la clave 'toppings', como haríamos con cualquier valor del diccionario. En lugar de devolver un único valor, obtenemos una lista de coberturas:

**pizza.py**

# Store information about a pizza being ordered.  
pizza = {  
 'crust': 'thick',  
 'toppings': ['mushrooms', 'extra cheese'],  
 }  
  
# Summarize the order.  
❶ print(f"You ordered a {pizza['crust']}-crust pizza "  
 "with the following toppings:")  
  
❷ for topping in pizza['toppings']:  
 print(f"\t{topping}")

Empezamos con un diccionario que contiene información sobre una pizza que se ha pedido. Una clave del diccionario es 'crust', y el valor asociado es la cadena 'thick'. La siguiente clave, 'toppings', tiene como valor una lista que almacena todos los ingredientes solicitados. Resumimos el pedido antes de construir la pizza ❶. Cuando necesites dividir una línea larga en una llamada a print(), elige un punto apropiado en el que dividir la línea que se está imprimiendo, y termina la línea con una comilla. Pon sangría en la línea siguiente, añade una comilla de apertura y continúa la cadena. Python combinará automáticamente todas las cadenas que encuentre dentro de los paréntesis. Para imprimir los ingredientes, escribimos un bucle for ❷. Para acceder a la lista de ingredientes, utilizamos la clave 'toppings', y Python obtiene la lista de ingredientes del diccionario.

La siguiente salida resume la pizza que pensamos construir:

You ordered a thick-crust pizza with the following toppings:  
 mushrooms  
 extra cheese

Puedes anidar una lista dentro de un diccionario siempre que quieras asociar más de un valor a una sola clave de un diccionario. En el ejemplo anterior de los lenguajes de programación favoritos, si almacenáramos las respuestas de cada persona en una lista, la gente podría elegir más de un lenguaje favorito. Cuando hagamos un bucle en el diccionario, el valor asociado a cada persona sería una lista de idiomas en lugar de un único idioma. Dentro del bucle for del diccionario, utilizamos otro bucle for para recorrer la lista de lenguas asociadas a cada persona:

**favorite\_languages.py**

favorite\_languages = {  
 'jen': ['python', 'rust'],  
 'sarah': ['c'],  
 'edward': ['rust', 'go'],  
 'phil': ['python', 'haskell'],  
 }  
  
❶ for name, languages in favorite\_languages.items():  
 print(f"\n{name.title()}'s favorite languages are:")  
❷ for language in languages:  
 print(f"\t{language.title()}")

El valor asociado a cada nombre en favorite\_languages es ahora una lista. Observa que algunas personas tienen una lengua favorita y otras tienen varias. Cuando hacemos el bucle a través del diccionario ❶, utilizamos el nombre de variable languages para contener cada valor del diccionario, porque sabemos que cada valor será una lista. Dentro del bucle principal del diccionario, utilizamos otro bucle for ❷ para recorrer la lista de lenguas favoritas de cada persona. Ahora cada persona puede enumerar tantas lenguas favoritas como quiera:

Jen's favorite languages are:  
 Python  
 Rust  
  
Sarah's favorite languages are:  
 C  
  
Edward's favorite languages are:  
 Rust  
 Go  
  
Phil's favorite languages are:  
 Python  
 Haskell

Para refinar aún más este programa, podrías incluir una sentencia if al principio del bucle for del diccionario para ver si cada persona tiene más de una lengua favorita examinando el valor de len(languages). Si una persona tiene más de una favorita, la salida seguiría siendo la misma. Si la persona sólo tiene una lengua favorita, podrías cambiar la redacción para reflejarlo. Por ejemplo, podrías decir: "El lenguaje favorito de Sara es C".

## Nota

No debes anidar listas y diccionarios demasiado profundamente. Si estás anidando elementos mucho más profundamente de lo que se ve en los ejemplos anteriores, o si estás trabajando con el código de otra persona con niveles significativos de anidamiento, lo más probable es que haya una forma más sencilla de resolver el problema.

### Un diccionario dentro de otro diccionario

Puedes anidar un diccionario dentro de otro diccionario, pero tu código puede complicarse rápidamente cuando lo hagas. Por ejemplo, si tienes varios usuarios para un sitio web, cada uno con un nombre de usuario único, puedes utilizar los nombres de usuario como claves de un diccionario. A continuación, puedes almacenar información sobre cada usuario utilizando un diccionario como valor asociado a su nombre de usuario. En el siguiente listado, almacenamos tres datos sobre cada usuario: su nombre, apellidos y ubicación. Accederemos a esta información recorriendo los nombres de usuario y el diccionario de información asociado a cada nombre de usuario:

**many\_users.py**

users = {  
 'aeinstein': {  
 'first': 'albert',  
 'last': 'einstein',  
 'location': 'princeton',  
 },  
  
 'mcurie': {  
 'first': 'marie',  
 'last': 'curie',  
 'location': 'paris',  
 },  
  
 }  
  
❶ for username, user\_info in users.items():  
❷ print(f"\nUsername: {username}")  
❸ full\_name = f"{user\_info['first']} {user\_info['last']}"  
 location = user\_info['location']  
  
❹ print(f"\tFull name: {full\_name.title()}")  
 print(f"\tLocation: {location.title()}")

Primero definimos un diccionario llamado users con dos claves: una para cada uno de los nombres de usuario 'aeinstein' y 'mcurie'. El valor asociado a cada clave es un diccionario que incluye el nombre, apellidos y ubicación de cada usuario. A continuación, recorremos en bucle el diccionario users ❶. Python asigna cada clave a la variable username, y el diccionario asociado a cada nombre de usuario se asigna a la variable user\_info. Una vez dentro del bucle principal del diccionario, imprimimos el nombre de usuario ❷.

A continuación, empezamos a acceder al diccionario interno ❸. La variable user\_info, que contiene el diccionario de información del usuario, tiene tres claves: 'first', 'last', y 'location'. Utilizamos cada clave para generar un nombre completo y una ubicación de cada persona con un formato ordenado, y luego imprimimos un resumen de lo que sabemos de cada usuario ❹:

Username: aeinstein  
 Full name: Albert Einstein  
 Location: Princeton  
  
Username: mcurie  
 Full name: Marie Curie  
 Location: Paris

Observa que la estructura del diccionario de cada usuario es idéntica. Aunque Python no lo requiere, esta estructura facilita el trabajo con diccionarios anidados. Si el diccionario de cada usuario tuviera claves diferentes, el código dentro del bucle for sería más complicado.

## Pruébalo tú mismo

6-7. Personas: Empieza con el programa que escribiste para el Ejercicio 6-1(página 98). Crea dos nuevos diccionarios que representen a personas diferentes, y almacena los tres diccionarios en una lista llamada people. Recorre tu lista de personas. Mientras recorres la lista, imprime todo lo que sepas sobre cada persona.

6-8. Mascotas: Haz varios diccionarios, donde cada diccionario represente una mascota diferente. En cada diccionario, incluye el tipo de animal y el nombre del dueño. Guarda estos diccionarios en una lista llamada pets. A continuación, recorre la lista en bucle y, mientras lo haces, imprime todo lo que sepas sobre cada mascota.

6-9. Lugares favoritos: Crea un diccionario llamado favorite\_places. Piensa en tres nombres para utilizarlos como claves en el diccionario, y almacena de uno a tres lugares favoritos para cada persona. Para hacer este ejercicio un poco más interesante, pide a algunos amigos que nombren algunos de sus lugares favoritos. Recorre el diccionario e imprime el nombre de cada persona y sus lugares favoritos.

6-10. Números Favoritos: Modifica tu programa del Ejercicio 6-2(página 98) para que cada persona pueda tener más de un número favorito. Luego imprime el nombre de cada persona junto con sus números favoritos.

6-11. Ciudades: Crea un diccionario llamado cities. Utiliza los nombres de tres ciudades como claves de tu diccionario. Crea un diccionario de información sobre cada ciudad e incluye el país en el que se encuentra la ciudad, su población aproximada y un dato sobre esa ciudad. Las claves del diccionario de cada ciudad deben ser algo así como country, population, y fact. Imprime el nombre de cada ciudad y toda la información que hayas almacenado sobre ella.

6-12. Ampliaciones: Ahora estamos trabajando con ejemplos lo suficientemente complejos como para que puedan ampliarse de muchas maneras. Utiliza uno de los programas de ejemplo de este capítulo y amplíalo añadiendo nuevas claves y valores, cambiando el contexto del programa o mejorando el formato de la salida.

## Resumen

En este capítulo has aprendido a definir un diccionario y a trabajar con la información almacenada en él. Aprendiste a acceder y modificar elementos individuales de un diccionario, y a recorrer en bucle todos los de la información de un diccionario. Aprendiste a recorrer los pares clave-valor de un diccionario, sus claves y sus valores. También has aprendido a anidar varios diccionarios en una lista, a anidar listas en un diccionario y a anidar un diccionario dentro de otro diccionario.

En el próximo capítulo aprenderás sobre los bucles while y cómo aceptar entradas de personas que utilizan tus programas. Éste será un capítulo emocionante, porque aprenderás a hacer que todos tus programas sean interactivos: podrán responder a las entradas del usuario.