# 3 Introducción a las listas



En este capítulo y en el siguiente aprenderás qué son las listas y cómo empezar a trabajar con los elementos de una lista. Las listas te permiten almacenar conjuntos de información en un solo lugar, tanto si tienes unos pocos elementos como millones de ellos. Las listas son una de las funciones más potentes de Python, fácilmente accesibles para los nuevos programadores, y unen muchos conceptos importantes de la programación.

## ¿Qué es una lista?

Un *list* es una colección de elementos en un orden determinado. Puedes hacer una lista que incluya las letras del alfabeto, los dígitos del 0 al 9 o los nombres de todas las personas de tu familia. Puedes poner lo que quieras en una lista, y los elementos de tu lista no tienen por qué estar relacionados de ninguna manera en particular. Como una lista suele contener más de un elemento, es buena idea que el nombre de tu lista sea plural, como letters, digits o names.

En Python, los corchetes ([]) indican una lista, y los elementos individuales de la lista se separan mediante comas. He aquí un ejemplo sencillo de una lista que contiene algunos tipos de bicicletas:

**bicycles.py**

bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles)

Si pides a Python que imprima una lista, Python devuelve su representación de la lista, incluidos los corchetes:

['trek', 'cannondale', 'redline', 'specialized']

Como ésta no es la salida que quieres que vean tus usuarios, vamos a aprender cómo acceder a los elementos individuales de una lista.

### Acceder a los elementos de una lista

Las listas son colecciones ordenadas, por lo que puedes acceder a cualquier elemento de una lista indicando a Python la posición, o *index*, del elemento deseado. Para acceder a un elemento de una lista, escribe el nombre de la lista seguido del índice del elemento entre corchetes.

Por ejemplo, vamos a sacar la primera bicicleta de la lista bicycles:

bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles[0])

Cuando pedimos un solo elemento de una lista, Python devuelve sólo ese elemento sin corchetes:

trek

Este es el resultado que quieres que vean tus usuarios: una salida limpia y bien formateada.

También puedes utilizar los métodos de cadena del Capítulo 2 en cualquier elemento de esta lista. Por ejemplo, puedes formatear el elemento 'trek' para que tenga un aspecto más presentable utilizando el método title():

bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles[0].title())

Este ejemplo produce la misma salida que el ejemplo anterior, excepto que 'Trek' está en mayúsculas.

### Las posiciones de índice empiezan en 0, no en 1

Python considera que el primer elemento de una lista está en la posición 0, no en la 1. Esto es así en la mayoría de los lenguajes de programación, y la razón tiene que ver con cómo se implementan las operaciones de lista en un nivel inferior. Si recibes resultados inesperados, pregúntate si estás cometiendo un error simple pero común de uno en uno.

El segundo elemento de una lista tiene un índice de 1. Utilizando este sistema de conteo, puedes obtener cualquier elemento que desees de una lista restando uno a su posición en la lista. Por ejemplo, para acceder al cuarto elemento de una lista, solicitas el elemento situado en el índice 3.

A continuación se piden las bicicletas en los índices 1 y 3:

bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles[1])  
print(bicycles[3])

Este código devuelve la segunda y la cuarta bicicleta de la lista:

cannondale  
specialized

Python tiene una sintaxis especial para acceder al último elemento de una lista. Si pides el elemento en el índice -1, Python siempre devuelve el último elemento de la lista:

bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles[-1])

Este código devuelve el valor 'specialized'. Esta sintaxis es bastante útil, porque a menudo querrás acceder a los últimos elementos de una lista sin saber exactamente su longitud. Esta convención se extiende también a otros valores de índice negativos. El índice -2 devuelve el segundo elemento desde el final de la lista, el índice -3 devuelve el tercer elemento desde el final, y así sucesivamente.

### Utilizar valores individuales de una lista

Puedes utilizar valores individuales de una lista del mismo modo que utilizarías cualquier otra variable. Por ejemplo, puedes utilizar f-cadenas para crear un mensaje basado en un valor de una lista.

Intentemos sacar la primera bicicleta de la lista y componer un mensaje utilizando ese valor:

bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
message = f"My first bicycle was a {bicycles[0].title()}."  
  
print(message)

Construimos una frase utilizando el valor en bicycles[0] y lo asignamos a la variable message. El resultado es una frase sencilla sobre la primera bicicleta de la lista:

My first bicycle was a Trek.

## Pruébalo tú mismo

Prueba estos breves programas para tener alguna experiencia de primera mano con las listas de Python. Quizá quieras crear una nueva carpeta para los ejercicios de cada capítulo, para mantenerlos organizados.

3-1. Nombres: Almacena los nombres de algunos de tus amigos en una lista llamada names. Imprime el nombre de cada persona accediendo a cada elemento de la lista, de uno en uno.

3-2. Saludos: Empieza con la lista que utilizaste en el Ejercicio 3-1, pero en lugar de imprimir sólo el nombre de cada persona, imprime un mensaje para ellos. El texto de cada mensaje debe ser el mismo, pero cada mensaje debe estar personalizado con el nombre de la persona.

3-3. Tu propia lista: Piensa en tu medio de transporte favorito, como una moto o un coche, y haz una lista que almacene varios ejemplos. Utiliza tu lista para imprimir una serie de afirmaciones sobre estos objetos, como "Me gustaría tener una moto Honda".

## Modificar, añadir y eliminar elementos

La mayoría de las listas que crees serán *dynamic*, lo que significa que construirás una lista y luego añadirás y quitarás elementos de ella a medida que tu programa siga su curso. Por ejemplo, puedes crear un juego en el que un jugador tenga que disparar a unos alienígenas desde el cielo. Podrías almacenar el conjunto inicial de alienígenas en una lista y luego eliminar un alienígena de la lista cada vez que se derribe uno. Cada vez que aparezca un nuevo alienígena en la pantalla, lo añades a la lista. Tu lista de alienígenas aumentará y disminuirá de longitud a lo largo del juego.

### Modificar elementos de una lista

La sintaxis para modificar un elemento es similar a la sintaxis para acceder a un elemento de una lista. Para modificar un elemento, utiliza el nombre de la lista seguido del índice del elemento que quieres cambiar y, a continuación, proporciona el nuevo valor que quieres que tenga ese elemento.

Por ejemplo, supongamos que tenemos una lista de motocicletas y que el primer elemento de la lista es 'honda'. Podemos cambiar el valor de este primer elemento una vez creada la lista:

**motorcycles.py**

motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)  
  
motorcycles[0] = 'ducati'  
print(motorcycles)

Aquí definimos la lista motorcycles, con 'honda' como primer elemento. A continuación, cambiamos el valor del primer elemento a 'ducati'. La salida muestra que se ha cambiado el primer elemento, mientras que el resto de la lista permanece igual:

['honda', 'yamaha', 'suzuki']  
['ducati', 'yamaha', 'suzuki']

Puedes cambiar el valor de cualquier elemento de una lista, no sólo del primero.

### Añadir elementos a una lista

Puedes querer añadir un nuevo elemento a una lista por muchas razones. Por ejemplo, puedes querer que aparezcan nuevos alienígenas en un juego, añadir nuevos datos a una visualización o añadir nuevos usuarios registrados a un sitio web que hayas construido. Python proporciona varias formas de añadir nuevos datos a las listas existentes.

#### Añadir elementos al final de una lista

La forma más sencilla de añadir un nuevo elemento a una lista es *append*. Cuando añades un elemento a una lista, el nuevo elemento se añade al final de la lista. Utilizando la misma lista que teníamos en el ejemplo anterior, añadiremos el nuevo elemento 'ducati' al final de la lista:

motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)  
  
motorcycles.append('ducati')  
print(motorcycles)

Aquí el método append() añade 'ducati' al final de la lista, sin afectar a ninguno de los demás elementos de la lista:

['honda', 'yamaha', 'suzuki']  
['honda', 'yamaha', 'suzuki', 'ducati']

El método append() facilita la construcción dinámica de listas. Por ejemplo, puedes empezar con una lista vacía y luego añadir elementos a la lista mediante una serie de llamadas a append(). Utilizando una lista vacía, añadamos a la lista los elementos 'honda', 'yamaha', y 'suzuki':

motorcycles = []  
  
motorcycles.append('honda')  
motorcycles.append('yamaha')  
motorcycles.append('suzuki')  
  
print(motorcycles)

La lista resultante tiene exactamente el mismo aspecto que las listas de los ejemplos anteriores:

['honda', 'yamaha', 'suzuki']

Construir listas de esta forma es muy habitual, porque a menudo no sabrás los datos que tus usuarios quieren almacenar en un programa hasta después de que éste se esté ejecutando. Para que tus usuarios tengan el control, empieza por definir una lista vacía que contendrá los valores de los usuarios. A continuación, añade cada nuevo valor proporcionado a la lista que acabas de crear.

#### Insertar elementos en una lista

Puedes añadir un nuevo elemento en cualquier posición de tu lista utilizando el método insert(). Para ello, especifica el índice del nuevo elemento y el valor del nuevo elemento:

motorcycles = ['honda', 'yamaha', 'suzuki']  
  
motorcycles.insert(0, 'ducati')  
print(motorcycles)

En este ejemplo, insertamos el valor 'ducati' al principio de la lista. El método insert() abre un espacio en la posición 0 y almacena el valor 'ducati' en ese lugar:

['ducati', 'honda', 'yamaha', 'suzuki']

Esta operación desplaza todos los demás valores de la lista una posición hacia la derecha.

### Eliminar elementos de una lista

A menudo, querrás eliminar un elemento o un conjunto de elementos de una lista. Por ejemplo, cuando un jugador derriba a un alienígena del cielo, lo más probable es que quieras eliminarlo de la lista de alienígenas activos. O cuando un usuario decide cancelar su cuenta en una aplicación web que has creado, querrás eliminar a ese usuario de la lista de usuarios activos. Puedes eliminar un elemento según su posición en la lista o según su valor.

#### Eliminar un elemento utilizando la sentencia del

Si conoces la posición del elemento que quieres eliminar de una lista, puedes utilizar la sentencia del:

motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)  
  
del motorcycles[0]  
print(motorcycles)

Aquí utilizamos la sentencia del para eliminar el primer elemento, 'honda', de la lista de motocicletas:

['honda', 'yamaha', 'suzuki']  
['yamaha', 'suzuki']

Puedes eliminar un elemento de cualquier posición de una lista utilizando la sentencia del si conoces su índice. Por ejemplo, así se elimina el segundo elemento, 'yamaha', de la lista:

motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)  
  
del motorcycles[1]  
print(motorcycles)

El segundo moto se elimina de la lista:

['honda', 'yamaha', 'suzuki']  
['honda', 'suzuki']

En ambos ejemplos, ya no puedes acceder al valor que se ha eliminado de la lista después de utilizar la sentencia del.

#### Eliminar un elemento con el método pop()

A veces querrás utilizar el valor de un elemento después de eliminarlo de una lista. Por ejemplo, podrías querer obtener la posición *x* y *y* de un alienígena que acaba de ser derribado, para poder dibujar una explosión en esa posición. En una aplicación web, puede que quieras eliminar a un usuario de una lista de miembros activos y luego añadirlo a una lista de miembros inactivos.

El método pop() elimina el último elemento de una lista, pero te permite trabajar con ese elemento después de eliminarlo. El término *pop* proviene de pensar en una lista como en una pila de elementos y de quitar un elemento de la parte superior de la pila. En esta analogía, la parte superior de una pila corresponde al final de una lista.

Vamos a sacar una moto de la lista de motos:

❶ motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)  
  
❷ popped\_motorcycle = motorcycles.pop()  
❸ print(motorcycles)  
❹ print(popped\_motorcycle)

Empezamos definiendo e imprimiendo la lista motorcycles ❶. A continuación, extraemos un valor de la lista y lo asignamos a la variable popped\_motorcycle ❷. Imprimimos la lista ❸ para mostrar que se ha eliminado un valor de la lista. Luego imprimimos el valor eliminado ❹ para demostrar que aún tenemos acceso al valor que se eliminó.

La salida muestra que el valor 'suzuki' fue eliminado del final de la lista y ahora está asignado a la variable popped\_motorcycle:

['honda', 'yamaha', 'suzuki']  
['honda', 'yamaha']  
suzuki

¿Qué utilidad puede tener este método pop()? Imagina que las motos de la lista están almacenadas en orden cronológico, según la fecha en que las poseíamos. Si éste es el caso, podemos utilizar el método pop() para imprimir una frase sobre la última moto que compramos:

motorcycles = ['honda', 'yamaha', 'suzuki']  
  
last\_owned = motorcycles.pop()  
print(f"The last motorcycle I owned was a {last\_owned.title()}.")

La salida es una simple frase sobre la última moto que hemos poseído:

The last motorcycle I owned was a Suzuki.

#### Eliminar elementos de cualquier posición de una lista

Puedes utilizar pop() para eliminar un elemento de cualquier posición de una lista incluyendo entre paréntesis el índice del elemento que quieres eliminar:

motorcycles = ['honda', 'yamaha', 'suzuki']  
  
first\_owned = motorcycles.pop(0)  
print(f"The first motorcycle I owned was a {first\_owned.title()}.")

Empezamos eliminando la primera moto de la lista, y luego imprimimos un mensaje sobre esa moto. La salida es una simple frase que describe la primera moto que tuve:

The first motorcycle I owned was a Honda.

Recuerda que cada vez que utilizas pop(), el elemento con el que trabajas deja de almacenarse en la lista.

Si no estás seguro de si utilizar la sentencia del o el método pop(), aquí tienes una forma sencilla de decidirte: cuando quieras eliminar un elemento de una lista y no utilizar ese elemento de ninguna forma, utiliza la sentencia del; si quieres utilizar un elemento mientras lo eliminas, utiliza el método pop().

#### Eliminar un elemento por valor

A veces no conocerás la posición del valor que quieres eliminar de una lista. Si sólo conoces el valor del elemento que quieres eliminar, puedes utilizar el método remove().

Por ejemplo, supongamos que queremos eliminar el valor 'ducati' de la lista de motocicletas:

motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']  
print(motorcycles)  
  
motorcycles.remove('ducati')  
print(motorcycles)

Aquí el método remove() le dice a Python que averigüe dónde aparece 'ducati' en la lista y elimine ese elemento:

['honda', 'yamaha', 'suzuki', 'ducati']  
['honda', 'yamaha', 'suzuki']

También puedes utilizar el método remove() para trabajar con un valor que se elimina de una lista. Vamos a eliminar el valor 'ducati' e imprimir una razón para eliminarlo de la lista:

❶ motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']  
print(motorcycles)  
  
❷ too\_expensive = 'ducati'  
❸ motorcycles.remove(too\_expensive)  
print(motorcycles)  
❹ print(f"\nA {too\_expensive.title()} is too expensive for me.")

Tras definir la lista ❶, asignamos el valor 'ducati' a una variable llamada too\_expensive ❷. A continuación, utilizamos esta variable para decirle a Python qué valor debe eliminar de la lista ❸. El valor 'ducati' se ha eliminado de la lista ❹ pero sigue siendo accesible a través de la variable too\_expensive, lo que nos permite imprimir una declaración sobre por qué hemos eliminado 'ducati' de la lista de motos:

['honda', 'yamaha', 'suzuki', 'ducati']  
['honda', 'yamaha', 'suzuki']  
  
A Ducati is too expensive for me.

## Nota

El método remove() sólo elimina la primera aparición del valor que especifiques. Si existe la posibilidad de que el valor aparezca más de una vez en la lista, tendrás que utilizar un bucle para asegurarte de que se eliminan todas las apariciones del valor. Aprenderás a hacerlo en el Capítulo 7.

## Pruébalo tú mismo

Los siguientes ejercicios son un poco más complejos que los del Capítulo 2, pero te dan la oportunidad de utilizar las listas de todas las formas descritas.

3-4. Lista de invitados: Si pudieras invitar a cenar a cualquier persona, viva o fallecida, ¿a quién invitarías? Haz una lista que incluya al menos a tres personas a las que te gustaría invitar a cenar. Después, utiliza tu lista para imprimir un mensaje dirigido a cada persona, invitándola a cenar.

3-5. Cambiar la lista de invitados: Te acabas de enterar de que uno de tus invitados no puede asistir a la cena, así que tienes que enviar un nuevo juego de invitaciones. Tendrás que pensar en otra persona a la que invitar.

* Empieza con tu programa del Ejercicio 3-4. Añade una llamada a print() al final de tu programa, indicando el nombre del invitado que no puede asistir.
* Modifica tu lista, sustituyendo el nombre del invitado que no puede venir por el nombre de la nueva persona a la que invitas.
* Imprime un segundo juego de mensajes de invitación, uno para cada persona que sigue en tu lista.

3-6. Más invitados: Acabas de encontrar una mesa más grande para cenar, así que ahora hay más espacio disponible. Piensa en tres invitados más a los que invitar a cenar.

* Empieza con tu programa del Ejercicio 3-4 o 3-5. Añade una llamada a print() al final de tu programa, informando a la gente de que has encontrado una mesa más grande.
* Utiliza insert() para añadir un nuevo invitado al principio de tu lista.
* Utiliza insert() para añadir un nuevo invitado a la mitad de tu lista.
* Utiliza append() para añadir un nuevo invitado al final de tu lista.
* Imprime un nuevo conjunto de mensajes de invitación, uno para cada persona de tu lista.

3-7. Reducir la lista de invitados: Te acabas de enterar de que tu nueva mesa no llegará a tiempo para la cena, y ahora sólo tienes espacio para dos invitados.

* Empieza con tu programa del Ejercicio 3-6. Añade una nueva línea que imprima un mensaje diciendo que sólo puedes invitar a dos personas a cenar.
* Utiliza pop() para eliminar invitados de tu lista de uno en uno hasta que sólo queden dos nombres en ella. Cada vez que elimines un nombre de tu lista, imprime un mensaje para esa persona haciéndole saber que lamentas no poder invitarla a cenar.
* Imprime un mensaje a cada una de las dos personas que siguen en tu lista, haciéndoles saber que siguen invitadas.
* Utiliza del para eliminar los dos últimos nombres de tu lista, de modo que tengas una lista vacía. Imprime tu lista para asegurarte de que realmente tienes una lista vacía al final de tu programa.

## Organizar una lista

A menudo, tus listas se crearán en un orden impredecible, porque no siempre puedes controlar el orden en que tus usuarios proporcionan sus datos. Aunque esto es inevitable en la mayoría de las circunstancias, con frecuencia querrás presentar la información en un orden determinado. A veces querrás que conserve el orden original de tu lista, y otras querrás cambiarlo. Python proporciona varias formas diferentes de organizar tus listas, dependiendo de la situación.

### Ordenar una lista permanentemente con el método sort()

El método sort() de Python hace relativamente fácil ordenar una lista. Imagina que tenemos una lista de coches y queremos cambiar el orden de la lista para almacenarlos alfabéticamente. Para simplificar la tarea, supongamos que todos los valores de la lista están en minúsculas:

**cars.py**

cars = ['bmw', 'audi', 'toyota', 'subaru']  
cars.sort()  
print(cars)

El método sort() cambia el orden de la lista permanentemente. Los coches están ahora en orden alfabético, y nunca podremos volver al orden original:

['audi', 'bmw', 'subaru', 'toyota']

También puedes ordenar esta lista en orden alfabético inverso pasando el argumento reverse=True al método sort(). El siguiente ejemplo ordena la lista de coches en orden alfabético inverso:

cars = ['bmw', 'audi', 'toyota', 'subaru']  
cars.sort(reverse=True)  
print(cars)

De nuevo, el orden de la lista se modifica permanentemente:

['toyota', 'subaru', 'bmw', 'audi']

### Ordenar una lista temporalmente con la función sorted()

Para mantener el orden original de una lista pero presentarla ordenada, puedes utilizar la función sorted(). La función sorted() te permite presentar tu lista en un orden determinado, pero no afecta al orden real de la lista.

Probemos esta función con la lista de coches.

cars = ['bmw', 'audi', 'toyota', 'subaru']  
  
❶ print("Here is the original list:")  
print(cars)  
  
❷ print("\nHere is the sorted list:")  
print(sorted(cars))  
  
❸ print("\nHere is the original list again:")  
print(cars)

Primero imprimimos la lista en su orden original ❶ y luego en orden alfabético ❷. Después de mostrar la lista en el nuevo orden, mostramos que la lista sigue guardada en su orden original ❸:

Here is the original list:  
['bmw', 'audi', 'toyota', 'subaru']  
  
Here is the sorted list:  
['audi', 'bmw', 'subaru', 'toyota']  
  
❶ Here is the original list again:  
['bmw', 'audi', 'toyota', 'subaru']

Observa que la lista sigue existiendo en su orden original ❶ después de utilizar la función sorted(). La función sorted() también puede aceptar un argumento reverse=True si quieres mostrar una lista en orden alfabético inverso.

## Nota

Ordenar una lista alfabéticamente es un poco más complicado cuando todos los valores no están en minúsculas. Hay varias formas de interpretar las mayúsculas a la hora de determinar un orden de clasificación, y especificar el orden exacto puede ser más complejo de lo que queremos tratar en este momento. Sin embargo, la mayoría de los enfoques de la ordenación se basarán directamente en lo que has aprendido en esta sección.

### Imprimir una lista en orden inverso

Para invertir el orden original de una lista, puedes utilizar el método reverse(). Si originalmente guardamos la lista de coches en orden cronológico según la fecha en que los tuvimos, podríamos reordenar fácilmente la lista en orden cronológico inverso:

cars = ['bmw', 'audi', 'toyota', 'subaru']  
print(cars)  
  
cars.reverse()  
print(cars)

Observa que reverse() no ordena alfabéticamente hacia atrás; simplemente invierte el orden de la lista:

['bmw', 'audi', 'toyota', 'subaru']  
['subaru', 'toyota', 'audi', 'bmw']

El método reverse() cambia el orden de una lista permanentemente, pero puedes volver al orden original en cualquier momento aplicando reverse() a la misma lista por segunda vez.

### Encontrar la longitud de una lista

Puedes averiguar rápidamente la longitud de una lista utilizando la función len(). La lista de este ejemplo tiene cuatro elementos, por lo que su longitud es 4:

>>> cars = ['bmw', 'audi', 'toyota', 'subaru']  
>>> len(cars)  
4

Te resultará útil len() cuando necesites identificar el número de alienígenas que aún quedan por derribar en un juego, determinar la cantidad de datos que tienes que gestionar en una visualización o averiguar el número de usuarios registrados en un sitio web, entre otras tareas.

## Nota

Python cuenta los elementos de una lista empezando por uno, por lo que no deberías encontrarte con errores de uno en uno al determinar la longitud de una lista.

## Pruébalo tú mismo

3-8. Ver el mundo: Piensa en al menos cinco lugares del mundo que te gustaría visitar.

* Guarda los lugares en una lista. Asegúrate de que la lista no está en orden alfabético.
* Imprime tu lista en su orden original. No te preocupes por imprimir la lista ordenadamente; simplemente imprímela como una lista Python sin procesar.
* Utiliza sorted() para imprimir tu lista en orden alfabético sin modificar la lista real.
* Demuestra que tu lista sigue en su orden original imprimiéndola.
* Utiliza sorted() para imprimir tu lista en orden alfabético inverso sin modificar el orden de la lista original.
* Demuestra que tu lista sigue en el orden original imprimiéndola de nuevo.
* Utiliza reverse() para cambiar el orden de tu lista. Imprime la lista para mostrar que su orden ha cambiado.
* Utiliza reverse() para volver a cambiar el orden de la lista. Imprime la lista para mostrar que ha vuelto a su orden original.
* Utiliza sort() para cambiar tu lista de modo que se almacene en orden alfabético. Imprime la lista para mostrar que se ha cambiado su orden.
* Utiliza sort() para cambiar tu lista de modo que se almacene en orden alfabético inverso. Imprime la lista para mostrar que su orden ha cambiado.

3-9. Invitados a cenar: Trabajando con uno de los programas de los Ejercicios 3-4 a 3-7(páginas 41-42), utiliza len() para imprimir un mensaje indicando el número de personas a las que invitas a cenar.

3-10. Cada Función: Piensa en cosas que podrías almacenar en una lista. Por ejemplo, podrías hacer una lista de montañas, ríos, países, ciudades, idiomas o cualquier otra cosa que te apetezca. Escribe un programa que cree una lista que contenga estos elementos y luego utiliza cada función presentada en este capítulo al menos una vez.

## Evitar errores de índice al trabajar con listas

Hay un tipo de error que es habitual ver cuando trabajas con listas por primera vez. Supongamos que tienes una lista con tres elementos, y preguntas por el cuarto elemento:

**motorcycles.py**

motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles[3])

Este ejemplo da como resultado un *index error*:

Traceback (most recent call last):  
 File "motorcycles.py", line 2, in <module>  
 print(motorcycles[3])  
 ~~~~~~~~~~~^^^  
IndexError: list index out of range

Python intenta darte el elemento en el índice 3. Pero cuando busca en la lista, ningún elemento de motorcycles tiene el índice 3. Debido a la naturaleza de la indexación en las listas, este error es típico. La gente cree que el tercer elemento es el número 3, porque empieza a contar en 1. Pero en Python el tercer elemento es el número 2, porque empieza a indexar en 0.

Un error de índice significa que Python no puede encontrar un elemento en el índice que has solicitado. Si se produce un error de índice en tu programa, intenta ajustar en uno el índice que pides. Luego vuelve a ejecutar el programa para ver si los resultados son correctos.

Ten en cuenta que siempre que quieras acceder al último elemento de una lista, debes utilizar el índice -1. Esto siempre funcionará, aunque tu lista haya cambiado de tamaño desde la última vez que accediste a ella:

motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles[-1])

El índice -1 siempre devuelve el último elemento de una lista, en este caso el valor 'suzuki':

suzuki

La única vez que este método provocará un error es cuando solicites el último elemento de una lista vacía:

motorcycles = []  
print(motorcycles[-1])

No hay ningún elemento en motorcycles, por lo que Python devuelve otro error de índice:

Traceback (most recent call last):  
 File "motorcyles.py", line 3, in <module>  
 print(motorcycles[-1])  
 ~~~~~~~~~~~^^^^  
IndexError: list index out of range

Si se produce un error de índice y no sabes cómo resolverlo, intenta imprimir tu lista o simplemente imprimir la longitud de tu lista. Tu lista puede tener un aspecto muy distinto del que pensabas, sobre todo si ha sido gestionada dinámicamente por tu programa. Ver la lista real, o el número exacto de elementos de tu lista, puede ayudarte a resolver esos errores lógicos.

## Pruébalo tú mismo

3-11. Error intencionado: Si todavía no has recibido un error de índice en uno de tus programas, intenta que se produzca uno. Cambia un índice en uno de tus programas para producir un error de índice. Asegúrate de corregir el error antes de cerrar el programa.

## Resumen

En este capítulo has aprendido qué son las listas y cómo trabajar con los elementos individuales de una lista. Aprendiste a definir una lista y a añadir y eliminar elementos. Aprendiste a ordenar listas de forma permanente y temporal para su visualización. También aprendiste a averiguar la longitud de una lista y a evitar errores de índice cuando trabajas con listas.

En el Capítulo 4 aprenderás a trabajar con los elementos de una lista de forma más eficaz. Recorriendo en bucle cada uno de los elementos de una lista con unas pocas líneas de código podrás trabajar con eficacia, incluso cuando tu lista contenga miles o millones de elementos.