# 5 Sentencias if



Programar a menudo implica examinar un conjunto de condiciones y decidir qué acción tomar en función de esas condiciones. La sentencia if de Python te permite examinar el estado actual de un programa y responder adecuadamente a ese estado.

En este capítulo, aprenderás a escribir pruebas condicionales, que te permiten comprobar cualquier condición de interés. Aprenderás a escribir sencillas sentencias if, y aprenderás a crear una serie más compleja de sentencias if para identificar cuándo se dan las condiciones exactas que deseas. A continuación, aplicarás este concepto a las listas, de modo que podrás escribir un bucle for que trate la mayoría de los elementos de una lista de una manera, pero que trate determinados elementos con valores específicos de una manera diferente.

## Un ejemplo sencillo

El siguiente ejemplo muestra cómo las pruebas if te permiten responder correctamente a situaciones especiales. Imagina que tienes una lista de coches y quieres imprimir el nombre de cada coche. Los nombres de los coches son nombres propios, por lo que los nombres de la mayoría de los coches deberían imprimirse en mayúsculas. Sin embargo, el valor 'bmw' debe imprimirse en mayúsculas. El siguiente código recorre una lista de nombres de coches y busca el valor 'bmw'. Siempre que el valor sea 'bmw', se imprime en mayúsculas en lugar de en mayúsculas de título:

**cars.py**

cars = ['audi', 'bmw', 'subaru', 'toyota']  
  
for car in cars:  
❶ if car == 'bmw':  
 print(car.upper())  
 else:  
 print(car.title())

El bucle de este ejemplo comprueba primero si el valor actual de car es 'bmw' ❶. Si lo es, el valor se imprime en mayúsculas. Si el valor de car es cualquier otra cosa que 'bmw', se imprime en mayúsculas:

Audi  
BMW  
Subaru  
Toyota

Este ejemplo combina varios de los conceptos que aprenderás en este capítulo. Empecemos por ver los tipos de pruebas que puedes utilizar para examinar las condiciones de tu programa.

## Pruebas condicionales

En el corazón de cada sentencia if hay una expresión que puede evaluarse como True o False y que se denomina *conditional test*. Python utiliza los valores True y False para decidir si debe ejecutarse el código de una sentencia if. Si una prueba condicional se evalúa como True, Python ejecuta el código que sigue a la sentencia if. Si el resultado de la prueba es False, Python ignora el código que sigue a la sentencia if.

### Comprobación de igualdad

La mayoría de las pruebas condicionales comparan el valor actual de una variable con un valor específico de interés. La prueba condicional más sencilla comprueba si el valor de una variable es igual al valor de interés:

>>> car = 'bmw'  
>>> car == 'bmw'  
True

La primera línea establece el valor de car en 'bmw' utilizando un único signo igual, como ya has visto muchas veces. La siguiente línea comprueba si el valor de car es 'bmw' utilizando un signo igual doble (==). Este *equality operator* devuelve True si los valores a izquierda y derecha del operador coinciden, y False si no coinciden. Los valores de este ejemplo coinciden, por lo que Python devuelve True.

Cuando el valor de car es distinto de 'bmw', esta prueba devuelve False:

>>> car = 'audi'  
>>> car == 'bmw'  
False

Un solo signo igual es realmente una sentencia; podrías leer la primera línea de código aquí como "Establece el valor de car igual a 'audi'." En cambio, un signo igual doble formula una pregunta: "¿Es el valor de car igual a 'bmw'?" La mayoría de los lenguajes de programación utilizan los signos igual de esta forma.

### Ignorar mayúsculas y minúsculas al comprobar la igualdad

En Python, la comprobación de la igualdad distingue entre mayúsculas y minúsculas. Por ejemplo, dos valores con mayúsculas diferentes no se consideran iguales:

>>> car = 'Audi'  
>>> car == 'audi'  
False

Si las mayúsculas y minúsculas importan, este comportamiento es ventajoso. Pero si las mayúsculas y minúsculas no importan y sólo quieres comprobar el valor de una variable, puedes convertir el valor de la variable a minúsculas antes de hacer la comparación:

>>> car = 'Audi'  
>>> car.lower() == 'audi'  
True

Esta prueba devolverá True independientemente de cómo esté formateado el valor 'Audi', porque ahora la prueba no distingue entre mayúsculas y minúsculas. El método lower() no cambia el valor almacenado originalmente en car, por lo que puedes hacer este tipo de comparación sin afectar a la variable original:

>>> car = 'Audi'  
>>> car.lower() == 'audi'  
True  
>>> car  
'Audi'

Primero asignamos la cadena en mayúsculas 'Audi' a la variable car. A continuación, convertimos el valor de car a minúsculas y comparamos el valor en minúsculas con la cadena 'audi'. Las dos cadenas coinciden, por lo que Python devuelve True. Podemos ver que el valor almacenado en car no se ha visto afectado por el método lower().

Los sitios web aplican ciertas reglas a los datos que introducen los usuarios de forma similar a ésta. Por ejemplo, un sitio puede utilizar una prueba condicional como ésta para asegurarse de que cada usuario tiene un nombre de usuario realmente único, y no sólo una variación de las mayúsculas del nombre de usuario de otra persona. Cuando alguien introduce un nuevo nombre de usuario, éste se convierte a minúsculas y se compara con las versiones en minúsculas de todos los nombres de usuario existentes. Durante esta comprobación, un nombre de usuario como 'John' será rechazado si ya se utiliza alguna variación de 'john'.

### Comprobación de la desigualdad

Cuando quieras determinar si dos valores no son iguales, puedes utilizar la sentencia *inequality operator* (!=). Utilicemos otra sentencia if para examinar cómo utilizar el operador de desigualdad. Vamos a almacenar un ingrediente de pizza solicitado en una variable y luego imprimiremos un mensaje si la persona no pidió anchoas:

**toppings.py**

requested\_topping = 'mushrooms'  
  
if requested\_topping != 'anchovies':  
 print("Hold the anchovies!")

Este código compara el valor de requested\_topping con el valor 'anchovies'. Si estos dos valores no coinciden, Python devuelve True y ejecuta el código que sigue a la sentencia if. Si los dos valores coinciden, Python devuelve False y no ejecuta el código que sigue a la sentencia if.

Como el valor de requested\_topping no es 'anchovies', se ejecuta la función print():

Hold the anchovies!

La mayoría de las expresiones condicionales que escribas comprobarán la igualdad, pero a veces te resultará más eficaz comprobar la desigualdad.

### Comparaciones numéricas

Comprobar valores numéricos es bastante sencillo. Por ejemplo, el siguiente código comprueba si una persona tiene 18 años:

>>> age = 18  
>>> age == 18  
True

También puedes comprobar si dos números no son iguales. Por ejemplo, el código siguiente imprime un mensaje si la respuesta dada no es correcta:

**magic\_number.py**

answer = 17  
if answer != 42:  
 print("That is not the correct answer. Please try again!")

La prueba condicional pasa, porque el valor de answer (17) no es igual a 42. Como la prueba pasa, se ejecuta el bloque de código sangrado:

That is not the correct answer. Please try again!

También puedes incluir varias comparaciones matemáticas en tus sentencias condicionales, como menor que, menor o igual que, mayor que y mayor o igual que:

>>> age = 19  
>>> age < 21  
True  
>>> age <= 21  
True  
>>> age > 21  
False  
>>> age >= 21  
False

Cada comparación matemática puede utilizarse como parte de una sentencia if, que puede ayudarte a detectar las condiciones exactas de interés.

### Comprobar varias condiciones

Puede que quieras comprobar varias condiciones al mismo tiempo. Por ejemplo, a veces puedes necesitar que dos condiciones sean True para realizar una acción. Otras veces, puedes estar satisfecho con que sólo una condición sea True. Las palabras clave and y or pueden ayudarte en estas situaciones.

#### Utilizar y para comprobar varias condiciones

Para comprobar si dos condiciones son ambas True simultáneamente, utiliza la palabra clave and para combinar las dos pruebas condicionales; si cada prueba pasa, la expresión global se evalúa como True. Si alguna de las pruebas falla o si ambas fallan, la expresión se evalúa como False.

Por ejemplo, puedes comprobar si dos personas tienen más de 21 años utilizando la siguiente prueba:

>>> age\_0 = 22  
>>> age\_1 = 18  
❶ >>> age\_0 >= 21 and age\_1 >= 21  
False  
❷ >>> age\_1 = 22  
>>> age\_0 >= 21 and age\_1 >= 21  
True

En primer lugar, definimos dos edades, age\_0 y age\_1. Luego comprobamos si ambas edades son mayores de 21 años ❶. La prueba de la izquierda pasa, pero la de la derecha falla, por lo que la expresión condicional global se evalúa como False. A continuación, cambiamos age\_1 por 22 ❷. El valor de age\_1 es ahora mayor que 21, por lo que las dos pruebas individuales pasan, haciendo que la expresión condicional global se evalúe como True.

Para mejorar la legibilidad, puedes utilizar paréntesis alrededor de las pruebas individuales, pero no es necesario. Si utilizas paréntesis, tu prueba tendría este aspecto:

(age\_0 >= 21) and (age\_1 >= 21)

#### Utilizar o para comprobar varias condiciones

La palabra clave or también te permite comprobar varias condiciones, pero se supera cuando se supera una o las dos pruebas individuales. Una expresión or sólo falla cuando fallan las dos pruebas individuales.

Consideremos de nuevo dos edades, pero esta vez buscaremos que sólo una persona tenga más de 21 años:

>>> age\_0 = 22  
>>> age\_1 = 18  
❶ >>> age\_0 >= 21 or age\_1 >= 21  
True  
❷ >>> age\_0 = 18  
>>> age\_0 >= 21 or age\_1 >= 21  
False

Volvemos a empezar con dos variables de edad. Como la prueba para age\_0 ❶ pasa, la expresión global se evalúa como True. A continuación, bajamos age\_0 a 18. En la prueba final ❷, ahora ambas pruebas fallan y la expresión global se evalúa como False.

### Comprobar si un valor está en una lista

A veces es importante comprobar si una lista contiene un determinado valor antes de realizar una acción. Por ejemplo, puede que quieras comprobar si un nuevo nombre de usuario ya existe en una lista de nombres de usuario actuales antes de completar el registro de alguien en un sitio web. En un proyecto cartográfico, puede que quieras comprobar si una ubicación enviada ya existe en una lista de ubicaciones conocidas.

Para averiguar si un valor concreto ya está en una lista, utiliza la palabra clave in. Consideremos un código que podrías escribir para una pizzería. Haremos una lista de los ingredientes que un cliente ha pedido para una pizza y luego comprobaremos si determinados ingredientes están en la lista.

>>> requested\_toppings = ['mushrooms', 'onions', 'pineapple']  
>>> 'mushrooms' in requested\_toppings  
True  
>>> 'pepperoni' in requested\_toppings  
False

La palabra clave in indica a Python que compruebe la existencia de 'mushrooms' y 'pepperoni' en la lista requested\_toppings. Esta técnica es bastante potente porque puedes crear una lista de valores esenciales y luego comprobar fácilmente si el valor que estás comprobando coincide con uno de los valores de la lista.

### Comprobar si un valor no está en una lista

Otras veces, es importante saber si un valor no aparece en una lista. Puedes utilizar la palabra clave not en esta situación. Por ejemplo, considera una lista de usuarios que tienen prohibido comentar en un foro. Puedes comprobar si un usuario ha sido vetado antes de permitir que esa persona envíe un comentario:

**banned\_users.py**

banned\_users = ['andrew', 'carolina', 'david']  
user = 'marie'  
  
if user not in banned\_users:  
 print(f"{user.title()}, you can post a response if you wish.")

En este caso, la declaración if es bastante clara. Si el valor de user no está en la lista banned\_users, Python devuelve True y ejecuta la línea sangrada.

El usuario 'marie' no está en la lista banned\_users, por lo que ve un mensaje invitándole a enviar una respuesta:

Marie, you can post a response if you wish.

### Expresiones booleanas

A medida que aprendas más sobre programación, oirás en algún momento el término *Boolean expression*. Una expresión booleana no es más que otro nombre para una prueba condicional. Un *Boolean value* es True o False, igual que el valor de una expresión condicional después de haber sido evaluada.

Los valores booleanos se utilizan a menudo para controlar determinadas condiciones, como si un juego se está ejecutando o si un usuario puede editar cierto contenido en un sitio web:

game\_active = True  
can\_edit = False

Los valores booleanos proporcionan una forma eficaz de realizar un seguimiento del estado de un programa o de una condición concreta que sea importante en tu programa.

## Pruébalo tú mismo

5-1. Pruebas condicionales: Escribe una serie de pruebas condicionales. Imprime una declaración que describa cada prueba y tu predicción para los resultados de cada una de ellas. Tu código debería tener este aspecto:

car = 'subaru'  
print("Is car == 'subaru'? I predict True.")  
print(car == 'subaru')  
  
print("\nIs car == 'audi'? I predict False.")  
print(car == 'audi')

* Fíjate bien en tus resultados y asegúrate de que entiendes por qué cada línea se evalúa como True o False.
* Crea al menos 10 pruebas. Haz que al menos 5 pruebas evalúen a True y otras 5 pruebas evalúen a False.

5-2. Más pruebas condicionales: No tienes por qué limitar a 10 el número de pruebas que crees. Si quieres probar más comparaciones, escribe más pruebas y añádelas a *conditional\_tests.py*. Ten al menos un resultado de True y otro de False para cada una de las siguientes:

* Pruebas de igualdad y desigualdad con cadenas
* Pruebas con el método lower()
* Pruebas numéricas de igualdad y desigualdad, mayor que y menor que, mayor o igual que y menor o igual que
* Pruebas con las palabras clave and y or
* Comprueba si un elemento está en una lista
* Comprueba si un elemento no está en una lista

## Sentencias if

Cuando entiendas las pruebas condicionales, podrás empezar a escribir sentencias if. Existen varios tipos de sentencias if, y tu elección dependerá del número de condiciones que necesites probar. Has visto varios ejemplos de sentencias if en la discusión sobre las pruebas condicionales, pero ahora vamos a profundizar en el tema.

### Sentencias if simples

El tipo más sencillo de sentencia if tiene una prueba y una acción:

if conditional\_test:  
 do something

Puedes poner cualquier prueba condicional en la primera línea y casi cualquier acción en el bloque sangrado que sigue a la prueba. Si el resultado de la prueba condicional es True, Python ejecuta el código que sigue a la sentencia if. Si el resultado de la prueba es False, Python ignora el código que sigue a la sentencia if.

Supongamos que tenemos una variable que representa la edad de una persona, y queremos saber si esa persona tiene edad suficiente para votar. El código siguiente comprueba si la persona puede votar:

**voting.py**

age = 19  
if age >= 18:  
 print("You are old enough to vote!")

Python comprueba si el valor de age es mayor o igual que 18 años. Si lo es, Python ejecuta la llamada sangrada a print():

You are old enough to vote!

La sangría juega el mismo papel en las sentencias if que en los bucles for. Todas las líneas sangradas después de una sentencia if se ejecutarán si la prueba pasa, y todo el bloque de líneas sangradas se ignorará si la prueba no pasa.

Puedes tener tantas líneas de código como quieras en el bloque que sigue a la sentencia if. Añadamos otra línea de salida si la persona tiene edad suficiente para votar, preguntándole si ya se ha registrado para votar:

age = 19  
if age >= 18:  
 print("You are old enough to vote!")  
 print("Have you registered to vote yet?")

La prueba condicional pasa, y ambas llamadas a print() están sangradas, por lo que se imprimen ambas líneas:

You are old enough to vote!  
Have you registered to vote yet?

Si el valor de age es inferior a 18, este programa no produciría ninguna salida.

### Sentencias if-else

A menudo, querrás realizar una acción cuando se supere una prueba condicional y una acción diferente en los demás casos. La sintaxis if-else de Python lo hace posible. Un bloque if-else es similar a una simple sentencia if, pero la sentencia else te permite definir una acción o conjunto de acciones que se ejecutan cuando falla la prueba condicional.

Mostraremos el mismo mensaje que teníamos anteriormente si la persona tiene edad suficiente para votar, pero esta vez añadiremos un mensaje para quien no tenga edad suficiente para votar:

age = 17  
❶ if age >= 18:  
 print("You are old enough to vote!")  
 print("Have you registered to vote yet?")  
❷ else:  
 print("Sorry, you are too young to vote.")  
 print("Please register to vote as soon as you turn 18!")

Si la prueba condicional ❶ pasa, se ejecuta el primer bloque de llamadas sangradas a print(). Si la prueba evalúa False, se ejecuta el bloque else ❷. Como age es menor que 18 esta vez, la prueba condicional falla y se ejecuta el código del bloque else:

Sorry, you are too young to vote.  
Please register to vote as soon as you turn 18!

Este código funciona porque sólo tiene dos situaciones posibles que evaluar: una persona tiene edad suficiente para votar o no tiene edad suficiente para votar. La estructura if-else  funciona bien en situaciones en las que quieres que Python ejecute siempre una de las dos acciones posibles. En una cadena if-else sencilla como ésta, siempre se ejecutará una de las dos acciones.

### La cadena if-elif-else

A menudo, necesitarás probar más de dos situaciones posibles, y para evaluarlas puedes utilizar la sintaxis if-elif-else de Python. Python sólo ejecuta un bloque en una cadena if-elif-else. Ejecuta cada prueba condicional en orden, hasta que una pasa. Cuando una prueba pasa, se ejecuta el código que sigue a esa prueba y Python se salta el resto de las pruebas.

Muchas situaciones del mundo real implican más de dos condiciones posibles. Por ejemplo, piensa en un parque de atracciones que cobra tarifas diferentes según la edad:

* Los menores de 4 años entran gratis.
* La entrada para las personas de entre 4 y 18 años cuesta 25 $.
* La entrada para mayores de 18 años cuesta 40 $.

¿Cómo podemos utilizar una sentencia if para determinar la tarifa de admisión de una persona? El siguiente código comprueba el grupo de edad de una persona y luego imprime un mensaje de precio de admisión:

**amusement\_park.py**

age = 12  
❶ if age < 4:  
 print("Your admission cost is $0.")  
❷ elif age < 18:  
 print("Your admission cost is $25.")  
❸ else:  
 print("Your admission cost is $40.")

La prueba if ❶ comprueba si una persona tiene menos de 4 años. Cuando se supera la prueba, se imprime un mensaje apropiado y Python se salta el resto de las pruebas. La línea elif ❷ es en realidad otra prueba if, que se ejecuta sólo si falla la prueba anterior. En este punto de la cadena, sabemos que la persona tiene al menos 4 años porque la primera prueba falló. Si la persona es menor de 18 años, se imprime un mensaje apropiado y Python se salta el bloque else. Si fallan las pruebas if y elif, Python ejecuta el código del bloque else ❸.

En este ejemplo, la prueba if ❶ evalúa a False, por lo que su bloque de código no se ejecuta. Sin embargo, la prueba elif se evalúa como True (12 es menor que 18), por lo que su código se ejecuta. La salida es una frase, que informa al usuario del coste de la entrada:

Your admission cost is $25.

Cualquier edad superior a 17 años provocaría el fallo de las dos primeras pruebas. En estas situaciones, se ejecutaría el bloque else y el precio de la entrada sería de 40$.

En lugar de imprimir el precio de la entrada dentro del bloque if-elif-else, sería más conciso establecer sólo el precio dentro de la cadena if-elif-else  y luego tener una única llamada a print() que se ejecute después de que se haya evaluado la cadena:

age = 12  
  
if age < 4:  
 price = 0  
elif age < 18:  
 price = 25  
else:  
 price = 40  
  
print(f"Your admission cost is ${price}.")

Las líneas sangradas fijan el valor de price en función de la edad de la persona, como en el ejemplo anterior. Después de que la cadena if-elif-else establezca el precio, una llamada independiente sin sangría a print() utiliza este valor para mostrar un mensaje informando del precio de admisión de la persona.

Este código produce la misma salida que el ejemplo anterior, pero el propósito de la cadena if-elif-else es más limitado. En lugar de determinar un precio y mostrar un mensaje, simplemente determina el precio de la entrada. Además de ser más eficaz, este código revisado es más fácil de modificar que el planteamiento original. Para cambiar el texto del mensaje de salida, sólo tendrías que cambiar una llamada a print() en lugar de tres llamadas separadas a print().

### Uso de varios bloques elif

Puedes utilizar tantos bloques elif en tu código como quieras. Por ejemplo, si el parque de atracciones fuera a implementar un descuento para mayores, podrías añadir una prueba condicional más al código para determinar si alguien tiene derecho al descuento para mayores. Digamos que cualquier persona de 65 años o más paga la mitad de la entrada normal, es decir, 20 $:

age = 12  
  
if age < 4:  
 price = 0  
elif age < 18:  
 price = 25  
elif age < 65:  
 price = 40  
else:  
 price = 20  
  
print(f"Your admission cost is ${price}.")

La mayor parte de este código no se modifica. El segundo bloque elif comprueba ahora si una persona tiene menos de 65 años antes de asignarle la tarifa de admisión completa de 40 $. Observa que el valor asignado en el bloque else debe cambiarse a 20 $, porque las únicas edades que llegan a este bloque son las de personas de 65 años o más.

### Omitir el bloque else

Python no requiere un bloque else al final de una cadena if-elif. A veces, un bloque else es útil. Otras veces, es más claro utilizar una sentencia elif adicional que capture la condición específica de interés:

age = 12  
  
if age < 4:  
 price = 0  
elif age < 18:  
 price = 25  
elif age < 65:  
 price = 40  
elif age >= 65:  
 price = 20  
  
print(f"Your admission cost is ${price}.")

El bloque final elif asigna un precio de 20 $ cuando la persona tiene 65 años o más, lo que es un poco más claro que el bloque general else. Con este cambio, cada bloque de código debe superar una prueba específica para poder ejecutarse.

El bloque else es una sentencia catchall. Iguala cualquier condición que no haya sido igualada por una prueba específica de if o elif, y eso a veces puede incluir datos no válidos o incluso maliciosos. Si tienes una condición final específica que estás comprobando, considera la posibilidad de utilizar un bloque elif final y omitir el bloque else. Como resultado, estarás más seguro de que tu código se ejecutará sólo en las condiciones correctas.

### Probar múltiples condiciones

La cadena if-elif-else es potente, pero sólo es apropiado utilizarla cuando sólo necesitas que pase una prueba. En cuanto Python encuentra una prueba que pasa, se salta el resto de las pruebas. Este comportamiento es beneficioso, porque es eficiente y te permite comprobar una condición concreta.

Sin embargo, a veces es importante comprobar todas las condiciones de interés. En este caso, debes utilizar una serie de sentencias if sencillas, sin elif ni bloques else. Esta técnica tiene sentido cuando más de una condición podría ser True, y quieres actuar sobre cada condición que sea True.

Reconsideremos el ejemplo de la pizzería. Si alguien solicita una pizza con dos ingredientes, tendrás que asegurarte de incluir ambos ingredientes en su pizza:

**toppings.py**

requested\_toppings = ['mushrooms', 'extra cheese']  
  
if 'mushrooms' in requested\_toppings:  
 print("Adding mushrooms.")  
❶ if 'pepperoni' in requested\_toppings:  
 print("Adding pepperoni.")  
if 'extra cheese' in requested\_toppings:  
 print("Adding extra cheese.")  
  
print("\nFinished making your pizza!")

Comenzamos con una lista que contiene los ingredientes solicitados. La primera sentencia if comprueba si la persona ha pedido champiñones en su pizza. Si es así, se imprime un mensaje confirmando ese ingrediente. La prueba de pepperoni ❶ es otra sentencia simple if, no una sentencia elif o else, por lo que esta prueba se ejecuta independientemente de si la prueba anterior se ha superado o no. La última sentencia if comprueba si se ha pedido queso extra, independientemente de los resultados de las dos primeras pruebas. Estas tres pruebas independientes se ejecutan cada vez que se ejecuta este programa.

Como se evalúan todas las condiciones de este ejemplo, tanto los champiñones como el queso extra se añaden a la pizza:

Adding mushrooms.  
Adding extra cheese.  
  
Finished making your pizza!

Este código no funcionaría correctamente si utilizáramos un bloque if-elif-else, porque el código dejaría de ejecutarse después de pasar una sola prueba. Esto es lo que ocurriría:

requested\_toppings = ['mushrooms', 'extra cheese']  
  
if 'mushrooms' in requested\_toppings:  
 print("Adding mushrooms.")  
elif 'pepperoni' in requested\_toppings:  
 print("Adding pepperoni.")  
elif 'extra cheese' in requested\_toppings:  
 print("Adding extra cheese.")  
  
print("\nFinished making your pizza!")

La prueba para 'mushrooms' es la primera que se supera, por lo que se añaden champiñones a la pizza. Sin embargo, los valores 'extra cheese' y 'pepperoni' nunca se comprueban, porque Python no ejecuta ninguna prueba más allá de la primera prueba que pasa en una cadena if-elif-else. Se añadirá el primer ingrediente del cliente, pero se omitirán todos los demás:

Adding mushrooms.  
  
Finished making your pizza!

En resumen, si sólo quieres que se ejecute un bloque de código, utiliza una cadena if-elif-else. Si necesitas que se ejecute más de un bloque de código, utiliza una serie de sentencias if independientes.

## Pruébalo tú mismo

5-3. Colores alienígenas nº 1: Imagina que acaban de derribar a un alienígena en un juego. Crea una variable llamada alien\_color y asígnale un valor de 'green', 'yellow', o 'red'.

* Escribe una sentencia if para comprobar si el color del alienígena es verde. Si lo es, imprime un mensaje indicando que el jugador acaba de ganar 5 puntos.
* Escribe una versión de este programa que pase la prueba if y otra que falle. (La versión que falle no tendrá salida).

5-4. Colores del alienígena nº 2: Elige un color para un alienígena como hiciste en el Ejercicio 5-3, y escribe una cadena if-else.

* Si el color del alienígena es verde, imprime una declaración de que el jugador acaba de ganar 5 puntos por disparar al alienígena.
* Si el color del alienígena no es verde, imprime una declaración que diga que el jugador acaba de ganar 10 puntos.
* Escribe una versión de este programa que ejecute el bloque if y otra que ejecute el bloque else.

5-5. Colores de los Alienígenas nº 3: Convierte tu cadena if-else del Ejercicio 5-4 en una cadena if-elif-else.

* Si el alienígena es verde, imprime un mensaje indicando que el jugador ha ganado 5 puntos.
* Si el alienígena es amarillo, imprime un mensaje indicando que el jugador ha ganado 10 puntos.
* Si el alienígena es rojo, imprime un mensaje indicando que el jugador ha ganado 15 puntos.
* Escribe tres versiones de este programa, asegurándote de que cada mensaje se imprime para el alienígena del color apropiado.

5-6. Etapas de la vida: Escribe una cadena if-elif-else que determine la etapa de la vida de una persona. Establece un valor para la variable age, y luego:

* Si la persona tiene menos de 2 años, imprime un mensaje que indique que la persona es un bebé.
* Si la persona tiene al menos 2 años pero menos de 4, imprime un mensaje indicando que la persona es un niño pequeño.
* Si la persona tiene al menos 4 años pero menos de 13, imprime un mensaje que indique que la persona es un niño.
* Si la persona tiene al menos 13 años pero menos de 20, imprime un mensaje que indique que la persona es un adolescente.
* Si la persona tiene al menos 20 años pero menos de 65, imprime un mensaje que indique que la persona es un adulto.
* Si la persona tiene 65 años o más, imprime un mensaje que indique que la persona es un anciano.

5-7. Fruta favorita: Haz una lista de tus frutas favoritas y, a continuación, escribe una serie de afirmaciones independientes if que comprueben la presencia de determinadas frutas en tu lista.

* Haz una lista de tus tres frutas favoritas y llámala favorite\_fruits.
* Escribe cinco afirmaciones if. Cada una de ellas debe comprobar si un determinado tipo de fruta está en tu lista. Si la fruta está en tu lista, el bloque if debe imprimir una sentencia como la siguiente *You really like bananas!*

## Uso de sentencias if con listas

Puedes hacer cosas interesantes combinando listas y sentencias if. Puedes vigilar valores especiales que deban tratarse de forma diferente a otros valores de la lista. Puedes gestionar eficazmente condiciones cambiantes, como la disponibilidad de determinados artículos en un restaurante a lo largo de un turno. También puedes empezar a probar que tu código funciona como esperas en todas las situaciones posibles.

### Comprobación de elementos especiales

Este capítulo comenzó con un ejemplo sencillo que mostraba cómo tratar un valor especial como 'bmw', que debía imprimirse en un formato diferente al de otros valores de la lista. Ahora que tienes unos conocimientos básicos sobre las pruebas condicionales y las sentencias if, vamos a ver más detenidamente cómo puedes comprobar si hay valores especiales en una lista y manejarlos adecuadamente.

Sigamos con el ejemplo de la pizzería. La pizzería muestra un mensaje cada vez que se añade un ingrediente a la pizza, mientras se está haciendo. El código para esta acción puede escribirse de forma muy eficiente haciendo una lista de los ingredientes que el cliente ha solicitado y utilizando un bucle para anunciar cada ingrediente a medida que se añade a la pizza:

**toppings.py**

requested\_toppings = ['mushrooms', 'green peppers', 'extra cheese']  
  
for requested\_topping in requested\_toppings:  
 print(f"Adding {requested\_topping}.")  
  
print("\nFinished making your pizza!")

La salida es sencilla porque este código es un simple bucle for:

Adding mushrooms.  
Adding green peppers.  
Adding extra cheese.  
  
Finished making your pizza!

¿Pero qué pasa si la pizzería se queda sin pimientos verdes? Una sentencia if dentro del bucle for puede manejar esta situación adecuadamente:

requested\_toppings = ['mushrooms', 'green peppers', 'extra cheese']  
  
for requested\_topping in requested\_toppings:  
 if requested\_topping == 'green peppers':  
 print("Sorry, we are out of green peppers right now.")  
 else:  
 print(f"Adding {requested\_topping}.")  
  
print("\nFinished making your pizza!")

Esta vez, comprobamos cada elemento solicitado antes de añadirlo a la pizza. La sentencia if comprueba si la persona ha solicitado pimientos verdes. Si es así, mostramos un mensaje informándole de por qué no puede tener pimientos verdes. El bloque else garantiza que todos los demás ingredientes se añadan a la pizza.

La salida muestra que cada aderezo solicitado se gestiona adecuadamente.

Adding mushrooms.  
Sorry, we are out of green peppers right now.  
Adding extra cheese.  
  
Finished making your pizza!

### Comprobar que una lista no está vacía

Hemos hecho una suposición sencilla sobre todas las listas con las que hemos trabajado hasta ahora: hemos supuesto que cada lista tiene al menos un elemento. Pronto dejaremos que los usuarios proporcionen la información que se almacena en una lista, por lo que no podremos suponer que una lista tiene algún elemento cada vez que se ejecute un bucle. En esta situación, es útil comprobar si una lista está vacía antes de ejecutar un bucle for.

Como ejemplo, comprobemos si la lista de ingredientes solicitados está vacía antes de construir la pizza. Si la lista está vacía, preguntaremos al usuario y nos aseguraremos de que quiere una pizza normal. Si la lista no está vacía, construiremos la pizza igual que en los ejemplos anteriores:

requested\_toppings = []  
  
if requested\_toppings:  
 for requested\_topping in requested\_toppings:  
 print(f"Adding {requested\_topping}.")  
 print("\nFinished making your pizza!")  
else:  
 print("Are you sure you want a plain pizza?")

Esta vez empezamos con una lista vacía de ingredientes solicitados. En lugar de saltar directamente a un bucle for, primero hacemos una comprobación rápida. Cuando se utiliza el nombre de una lista en una sentencia if, Python devuelve True si la lista contiene al menos un elemento; una lista vacía se evalúa como False. Si requested\_toppings supera la prueba condicional, ejecutamos el mismo bucle for que utilizamos en el ejemplo anterior . Si la prueba condicional falla, imprimimos un mensaje preguntando al cliente si realmente quiere una pizza normal sin ingredientes.

La lista está vacía en este caso, por lo que la salida pregunta si el usuario realmente quiere una pizza normal:

Are you sure you want a plain pizza?

Si la lista no está vacía, la salida mostrará cada uno de los ingredientes solicitados añadidos a la pizza.

### Utilizar varias listas

La gente pide casi cualquier cosa, sobre todo cuando se trata de ingredientes de pizza. ¿Y si un cliente quiere patatas fritas en su pizza? Puedes utilizar listas y afirmaciones de if para asegurarte de que la información tiene sentido antes de actuar en consecuencia.

Tengamos cuidado con las peticiones inusuales de ingredientes antes de construir una pizza. El siguiente ejemplo define dos listas. La primera es una lista de ingredientes disponibles en la pizzería, y la segunda es la lista de ingredientes que ha solicitado el usuario. Esta vez, cada elemento de requested\_toppings se comprueba con la lista de ingredientes disponibles antes de añadirlo a la pizza:

available\_toppings = ['mushrooms', 'olives', 'green peppers',  
 'pepperoni', 'pineapple', 'extra cheese']  
  
❶ requested\_toppings = ['mushrooms', 'french fries', 'extra cheese']  
  
for requested\_topping in requested\_toppings:  
❷ if requested\_topping in available\_toppings:  
 print(f"Adding {requested\_topping}.")  
❸ else:  
 print(f"Sorry, we don't have {requested\_topping}.")  
  
print("\nFinished making your pizza!")

En primer lugar, definimos una lista de ingredientes disponibles en esta pizzería. Ten en cuenta que puede ser una tupla si la pizzería tiene una selección estable de ingredientes. A continuación, hacemos una lista de los ingredientes que ha solicitado un cliente. En este ejemplo hay una petición inusual de un ingrediente: 'french fries' ❶. A continuación, hacemos un bucle con la lista de ingredientes solicitados. Dentro del bucle, comprobamos si cada cobertura solicitada está realmente en la lista de coberturas disponibles ❷. Si lo está, añadimos ese ingrediente a la pizza. Si el ingrediente solicitado no está en la lista de ingredientes disponibles, se ejecutará el bloque else ❸. El bloque else imprime un mensaje indicando al usuario qué ingredientes no están disponibles.

Esta sintaxis de código produce una salida limpia e informativa:

Adding mushrooms.  
Sorry, we don't have french fries.  
Adding extra cheese.  
  
Finished making your pizza!

En sólo unas pocas líneas de código, ¡hemos gestionado una situación del mundo real con bastante eficacia!

## Pruébalo tú mismo

5-8. Hola Admin: Haz una lista de cinco o más nombres de usuario, incluyendo el nombre 'admin'. Imagina que estás escribiendo un código que imprimirá un saludo a cada usuario después de que se conecte a un sitio web. Recorre la lista e imprime un saludo a cada usuario.

* Si el nombre de usuario es 'admin', imprime un saludo especial, como por ejemplo *Hello admin, would you like to see a status report?*
* Si no, imprime un saludo genérico, como *Hello Jaden, thank you for logging in again.*

5-9. No hay usuarios: Añade una prueba if a *hello\_admin.py* para asegurarte de que la lista de usuarios no está vacía.

* Si la lista está vacía, imprime el mensaje *We need to find some users!*
* Elimina todos los nombres de usuario de la lista y asegúrate de que se imprime el mensaje correcto.

5-10. Comprobación de los nombres de usuario: Haz lo siguiente para crear un programa que simule cómo los sitios web se aseguran de que todo el mundo tiene un nombre de usuario único.

* Haz una lista de cinco o más nombres de usuario llamada current\_users.
* Haz otra lista de cinco nombres de usuario llamada new\_users. Asegúrate de que uno o dos de los nuevos nombres de usuario están también en la lista current\_users.
* Recorre la lista new\_users para ver si ya se ha utilizado cada nombre de usuario nuevo. Si es así, imprime un mensaje indicando que la persona deberá introducir un nuevo nombre de usuario. Si un nombre de usuario no se ha utilizado, imprime un mensaje diciendo que el nombre de usuario está disponible.
* Asegúrate de que la comparación no distingue entre mayúsculas y minúsculas. Si se ha utilizado 'John', no debe aceptarse 'JOHN'. (Para ello, tendrás que hacer una copia de current\_users que contenga las versiones en minúsculas de todos los usuarios existentes).

5-11. Números ordinales: Los números ordinales indican su posición en una lista, como *1st* o *2nd*. La mayoría de los números ordinales terminan en *th*, excepto 1, 2 y 3.

* Almacena los números del 1 al 9 en una lista.
* Recorre la lista en bucle.
* Utiliza una cadena if-elif-else dentro del bucle para imprimir el final ordinal adecuado para cada número. Tu salida debe ser "1st 2nd 3rd 4th 5th 6th 7th 8th 9th", y cada resultado debe estar en una línea separada.

## Cómo dar estilo a tus sentencias if

En todos los ejemplos de este capítulo has visto buenos hábitos de estilo. La única recomendación que ofrece PEP 8 para dar estilo a las pruebas condicionales es utilizar un solo espacio alrededor de los operadores de comparación, como ==, >= y <=. Por ejemplo

if age < 4:

es mejor que:

if age<4:

Este espaciado no afecta a la forma en que Python interpreta tu código; sólo hace que tu código sea más fácil de leer para ti y para los demás.

## Pruébalo tú mismo

5-12. Estilizar **las sentencias** if**:** Revisa los programas que has escrito en este capítulo y asegúrate de que has dado el estilo adecuado a tus pruebas condicionales.

5-13. Tus ideas: Llegados a este punto, eres un programador más capaz de lo que eras cuando empezaste este libro. Ahora que tienes una mejor idea de cómo se modelan las situaciones del mundo real en los programas, podrías estar pensando en algunos problemas que podrías resolver con tus propios programas. Anota cualquier idea nueva que tengas sobre problemas que podrías querer resolver a medida que tus habilidades de programación sigan mejorando. Piensa en juegos que te gustaría escribir, conjuntos de datos que te gustaría explorar y aplicaciones web que te gustaría crear.

## Resumen

En este capítulo aprendiste a escribir pruebas condicionales, que siempre se evalúan como True o False. Aprendiste a escribir sentencias simples if, cadenas if-else y cadenas if-elif-else. Empezaste a utilizar estas estructuras para identificar determinadas condiciones que necesitas probar y para saber cuándo se han cumplido esas condiciones en tus programas. Aprendiste a tratar determinados elementos de una lista de forma diferente al resto de elementos, sin dejar de utilizar la eficacia de un bucle for. También revisaste las recomendaciones de estilo de Python para asegurarte de que tus programas, cada vez más complejos, siguen siendo relativamente fáciles de leer y comprender.

En el Capítulo 6 aprenderás sobre los diccionarios de Python. Un diccionario es similar a una lista, pero te permite conectar piezas de información. Aprenderás a construir diccionarios, a recorrerlos en bucle y a utilizarlos en combinación con listas y sentencias if. Aprender sobre diccionarios te permitirá modelizar una variedad aún mayor de situaciones del mundo real.