# 2 Variables y Tipos de Datos Simples



En este capítulo aprenderás sobre los distintos tipos de datos con los que puedes trabajar en tus programas Python. También aprenderás a utilizar variables para representar datos en tus programas.

## Qué ocurre realmente cuando ejecutas hola\_mundo.py

Echemos un vistazo más de cerca a lo que hace Python cuando ejecutas *hello\_world.py*. Resulta que Python hace bastante trabajo, incluso cuando ejecuta un programa sencillo:

**hello\_world.py**

print("Hello Python world!")

Cuando ejecutes este código, deberías ver la siguiente salida:

Hello Python world!

Cuando ejecutas el archivo *hello\_world.py*, la terminación *.py* indica que el archivo es un programa Python. A continuación, tu editor ejecuta el archivo a través del *Python interpreter*, que lee el programa y determina qué significa cada palabra del programa. Por ejemplo, cuando el intérprete ve la palabra print seguida de paréntesis, imprime en la pantalla lo que haya dentro de los paréntesis.

A medida que escribes tus programas, tu editor resalta diferentes partes de tu programa de diferentes maneras. Por ejemplo, reconoce que print() es el nombre de una función y muestra esa palabra en un color. Reconoce que "Hello Python world!" no es código Python y muestra esa frase en un color diferente. Esta función se llama *syntax highlighting* y es bastante útil cuando empiezas a escribir tus propios programas.

## Variables

Intentemos utilizar una variable en *hello\_world.py*. Añade una nueva línea al principio del archivo y modifica la segunda línea:

**hello\_world.py**

message = "Hello Python world!"  
print(message)

Ejecuta este programa para ver qué ocurre. Deberías ver la misma salida que viste anteriormente:

Hello Python world!

Hemos añadido un *variable* llamado message. Cada variable está conectada a un *value*, que es la información asociada a esa variable. En este caso, el valor es el texto "Hello Python world!".

Añadir una variable supone un poco más de trabajo para el intérprete de Python. Cuando procesa la primera línea, asocia la variable message al texto "Hello Python world!". Cuando llega a la segunda línea, imprime en la pantalla el valor asociado a message.

Vamos a ampliar este programa modificando *hello\_world.py* para imprimir un segundo mensaje. Añade una línea en blanco a *hello\_world.py*, y luego añade dos nuevas líneas de código:

message = "Hello Python world!"  
print(message)  
  
message = "Hello Python Crash Course world!"  
print(message)

Ahora, cuando ejecutes *hello\_world.py*, deberías ver dos líneas de salida:

Hello Python world!  
Hello Python Crash Course world!

Puedes cambiar el valor de una variable en tu programa en cualquier momento, y Python siempre mantendrá un registro de su valor actual.

### Nombrar y utilizar variables

Cuando utilices variables en Python, debes respetar algunas reglas y directrices. Incumplir algunas de estas reglas provocará errores; otras directrices simplemente te ayudarán a escribir un código más fácil de leer y comprender. Asegúrate de tener en cuenta las siguientes reglas cuando trabajes con variables:

* Los nombres de las variables sólo pueden contener letras, números y guiones bajos. Pueden empezar por una letra o un guión bajo, pero no por un número. Por ejemplo, puedes llamar a una variable message\_1 pero no 1\_message.
* No se permiten espacios en los nombres de las variables, pero se pueden utilizar guiones bajos para separar palabras en los nombres de las variables. Por ejemplo, greeting\_message funciona, pero greeting message provocará errores.
* Evita utilizar palabras clave de Python y nombres de funciones como nombres de variables. Por ejemplo, no utilices la palabra print como nombre de variable; Python la ha reservado para un fin programático concreto. (Consulta "Palabras clave de Python y funciones incorporadas" en la página 466.)
* Los nombres de las variables deben ser cortos pero descriptivos. Por ejemplo, name es mejor que n, student\_name es mejor que s\_n, y name\_length es mejor que length\_of\_persons\_name.
* Ten cuidado al utilizar la letra minúscula *l* y la mayúscula *O* porque podrían confundirse con los números *1* y *0*.

Puede llevarte algo de práctica aprender a crear buenos nombres de variables, sobre todo a medida que tus programas se vuelvan más interesantes y complicados. A medida que escribas más programas y empieces a leer el código de otras personas, mejorarás en la creación de nombres significativos.

## Nota

Las variables de Python que utilices en este momento deben estar en minúsculas. No tendrás errores si utilizas mayúsculas, pero las mayúsculas en los nombres de variables tienen significados especiales que discutiremos en capítulos posteriores.

### Evitar errores de nombre al utilizar variables

Todo programador comete errores, y la mayoría los comete todos los días. Aunque los buenos programadores pueden cometer errores, también saben responder a esos errores con eficacia. Veamos un error que es probable que cometas al principio y aprendamos a solucionarlo.

Escribiremos un código que genere un error a propósito. Introduce el siguiente código, incluyendo la palabra mal escrita mesage*,* que se muestra en negrita:

message = "Hello Python Crash Course reader!"  
print(mesage)

Cuando se produce un error en tu programa, el intérprete de Python hace todo lo posible para ayudarte a averiguar dónde está el problema. El intérprete proporciona un rastreo cuando un programa no puede ejecutarse correctamente. Un *traceback* es un registro de dónde tuvo problemas el intérprete al intentar ejecutar tu código. He aquí un ejemplo del rastreo que proporciona Python después de que accidentalmente hayas escrito mal el nombre de una variable:

Traceback (most recent call last):  
❶ File "hello\_world.py", line 2, in <module>  
❷ print(mesage)  
 ^^^^^^  
❸ NameError: name 'mesage' is not defined. Did you mean: 'message'?

La salida informa de que se produce un error en la línea 2 del archivo *hello\_world.py* ❶. El intérprete muestra esta línea ❷ para ayudarnos a detectar el error rápidamente y nos dice qué tipo de error ha encontrado ❸. En este caso ha encontrado un *name error* e informa de que la variable que se está imprimiendo, mesage, no está definida. Python no puede identificar el nombre de la variable proporcionada. Un error de nombre suele significar que hemos olvidado definir el valor de una variable antes de utilizarla, o que hemos cometido un error ortográfico al introducir el nombre de la variable. Si Python encuentra un nombre de variable similar al que no reconoce, te preguntará si ese es el nombre que querías utilizar.

En este ejemplo hemos omitido la letra *s* en el nombre de la variable message en la segunda línea. El intérprete de Python no revisa la ortografía de tu código, pero sí se asegura de que los nombres de las variables se escriban de forma coherente. Por ejemplo, observa lo que ocurre cuando escribimos *message* incorrectamente en la línea que define la variable:

mesage = "Hello Python Crash Course reader!"  
print(mesage)

En este caso, ¡el programa se ejecuta correctamente!

Hello Python Crash Course reader!

Los nombres de las variables coinciden, así que Python no ve ningún problema. Los lenguajes de programación son estrictos, pero no tienen en cuenta la buena y la mala ortografía. En consecuencia, no necesitas tener en cuenta las reglas ortográficas y gramaticales del inglés cuando intentes crear nombres de variables y escribir código.

Muchos errores de programación son simples erratas de un solo carácter en una línea de un programa. Si te encuentras pasando mucho tiempo buscando uno de estos errores, debes saber que estás en buena compañía. Muchos programadores experimentados y con talento pasan horas buscando este tipo de errores minúsculos. Intenta reírte de ello y sigue adelante, sabiendo que ocurrirá con frecuencia a lo largo de tu vida de programador.

### Las variables son etiquetas

Las variables se describen a menudo como cajas en las que puedes almacenar valores. Esta idea puede ser útil las primeras veces que utilices una variable, pero no es una forma precisa de describir cómo se representan internamente las variables en Python. Es mucho mejor pensar en las variables como etiquetas que puedes asignar a valores. También puedes decir que una variable hace referencia a un determinado valor.

Es probable que esta distinción no tenga mucha importancia en tus programas iniciales, pero merece la pena aprenderla antes que después. En algún momento, verás un comportamiento inesperado de una variable, y una comprensión precisa de cómo funcionan las variables te ayudará a identificar lo que está ocurriendo en tu código.

## Nota

La mejor forma de entender nuevos conceptos de programación es intentar utilizarlos en tus programas. Si te quedas atascado mientras trabajas en un ejercicio de este libro, intenta hacer otra cosa durante un rato. Si sigues atascado, repasa la parte correspondiente de ese capítulo. Si sigues necesitando ayuda, consulta las sugerencias del Apéndice C.

## Inténtalo tú mismo

Escribe un programa distinto para realizar cada uno de estos ejercicios. Guarda cada programa con un nombre de archivo que siga las convenciones estándar de Python, utilizando letras minúsculas y guiones bajos, como *simple\_message.py* y *simple\_messages.py*.

2-1. Mensaje simple: Asigna un mensaje a una variable y luego imprime ese mensaje.

2-2. Mensajes Simples: Asigna un mensaje a una variable, e imprime ese mensaje. A continuación, cambia el valor de la variable por un nuevo mensaje, e imprime el nuevo mensaje.

## Cadenas

Dado que la mayoría de los programas definen y recogen algún tipo de dato y luego hacen algo útil con él, resulta útil clasificar los distintos tipos de datos. El primer tipo de datos que veremos es la cadena. Las cadenas son bastante sencillas a primera vista, pero puedes utilizarlas de muchas formas distintas.

Una *string* es una serie de caracteres. Cualquier cosa que esté entre comillas se considera una cadena en Python, y puedes utilizar comillas simples o dobles alrededor de tus cadenas, de esta forma:

"This is a string."  
'This is also a string.'

Esta flexibilidad te permite utilizar comillas y apóstrofes dentro de tus cadenas:

'I told my friend, "Python is my favorite language!"'  
"The language 'Python' is named after Monty Python, not the snake."  
"One of Python's strengths is its diverse and supportive community."

Exploremos algunas de las formas en que puedes utilizar cadenas.

### Cambiar mayúsculas y minúsculas en una cadena con métodos

Una de las tareas más sencillas que puedes hacer con las cadenas es cambiar las mayúsculas y minúsculas de las palabras de una cadena. Mira el siguiente código e intenta determinar qué está ocurriendo:

**name.py**

name = "ada lovelace"  
print(name.title())

Guarda este archivo como *name.py* y ejecútalo. Deberías ver esta salida:

Ada Lovelace

En este ejemplo, la variable name hace referencia a la cadena en minúsculas "ada lovelace". El método title() aparece después de la variable en la llamada a print(). Un *method* es una acción que Python puede realizar sobre un dato. El punto (.) después de name en name.title() indica a Python que haga que el método title() actúe sobre la variable name. Cada método va seguido de un conjunto de paréntesis, porque los métodos a menudo necesitan información adicional para realizar su trabajo. Esa información se proporciona dentro de los paréntesis. La función title() no necesita ninguna información adicional, por lo que sus paréntesis están vacíos.

El método title() cambia cada palabra a mayúsculas, donde cada palabra empieza con mayúscula. Esto es útil porque a menudo querrás pensar en un nombre como una información. Por ejemplo, puede que quieras que tu programa reconozca los valores de entrada Ada, ADA, y ada como el mismo nombre, y los muestre todos como Ada.

También existen otros métodos útiles para tratar las mayúsculas y minúsculas. Por ejemplo, puedes cambiar una cadena a todo mayúsculas o todo minúsculas de esta forma:

name = "Ada Lovelace"  
print(name.upper())  
print(name.lower())

Esto mostrará lo siguiente:

ADA LOVELACE  
ada lovelace

El método lower() es especialmente útil para almacenar datos. Normalmente no querrás fiarte de las mayúsculas que te proporcionen tus usuarios, así que convertirás las cadenas a minúsculas antes de almacenarlas. Luego, cuando quieras mostrar la información, utilizarás las mayúsculas y minúsculas que tengan más sentido para cada cadena.

### Uso de variables en cadenas

En algunas situaciones, querrás utilizar el valor de una variable dentro de una cadena. Por ejemplo, podrías querer utilizar dos variables para representar un nombre y un apellido, respectivamente, y luego combinar esos valores para mostrar el nombre completo de alguien:

**full\_name.py**

first\_name = "ada"  
last\_name = "lovelace"  
❶ full\_name = f"{first\_name} {last\_name}"  
print(full\_name)

Para insertar el valor de una variable en una cadena, coloca la letra f inmediatamente antes de la comilla de apertura ❶. Pon llaves alrededor del nombre o nombres de cualquier variable que quieras utilizar dentro de la cadena. Python sustituirá cada variable por su valor cuando se muestre la cadena.

Estas cadenas se llaman *f-strings*. El *f* es para *format*, porque Python formatea la cadena sustituyendo el nombre de cualquier variable entre llaves por su valor. La salida del código anterior es

ada lovelace

Puedes hacer muchas cosas con las cadenas f. Por ejemplo, puedes utilizar cadenas f para componer mensajes completos utilizando la información asociada a una variable, como se muestra aquí:

first\_name = "ada"  
last\_name = "lovelace"  
full\_name = f"{first\_name} {last\_name}"  
❶ print(f"Hello, {full\_name.title()}!")

El nombre completo se utiliza en una frase que saluda al usuario ❶, y el método title() cambia el nombre a mayúsculas y minúsculas. Este código devuelve un saludo sencillo pero bien formateado:

Hello, Ada Lovelace!

También puedes utilizar cadenas f para componer un mensaje, y luego asignar todo el mensaje a una variable:

first\_name = "ada"  
last\_name = "lovelace"  
full\_name = f"{first\_name} {last\_name}"  
❶ message = f"Hello, {full\_name.title()}!"  
❷ print(message)

Este código también muestra el mensaje Hello, Ada Lovelace!, pero al asignar el mensaje a una variable ❶ hacemos que la llamada final a print() sea mucho más sencilla ❷.

### Añadir espacios en blanco a cadenas con tabuladores o nuevas líneas

En programación, *whitespace* se refiere a cualquier carácter no impreso, como espacios, tabuladores y símbolos de fin de línea. Puedes utilizar los espacios en blanco para organizar tu salida de forma que sea más fácil de leer para los usuarios.

Para añadir un tabulador a tu texto, utiliza la combinación de caracteres \t:

>>> print("Python")  
Python  
>>> print("\tPython")  
 Python

Para añadir una nueva línea en una cadena, utiliza la combinación de caracteres \n:

>>> print("Languages:\nPython\nC\nJavaScript")  
Languages:  
Python  
C  
JavaScript

También puedes combinar tabuladores y nuevas líneas en una sola cadena. La cadena "\n\t" le dice a Python que se mueva a una nueva línea y comience la siguiente con un tabulador. El siguiente ejemplo muestra cómo puedes utilizar una cadena de una sola línea para generar cuatro líneas de salida:

>>> print("Languages:\n\tPython\n\tC\n\tJavaScript")  
Languages:  
 Python  
 C  
 JavaScript

Las nuevas líneas y los tabuladores serán muy útiles en los dos próximos capítulos, cuando empieces a producir muchas líneas de salida a partir de unas pocas líneas de código.

### Eliminar espacios en blanco

Los espacios en blanco adicionales pueden confundir en tus programas. Para los programadores, 'python' y 'python ' son prácticamente lo mismo. Pero para un programa, son dos cadenas diferentes. Python detecta el espacio extra en 'python ' y lo considera significativo a menos que le digas lo contrario.

Es importante pensar en los espacios en blanco, porque a menudo querrás comparar dos cadenas para determinar si son iguales. Por ejemplo, un caso importante podría consistir en comprobar los nombres de usuario de las personas cuando se conectan a un sitio web. Los espacios en blanco adicionales también pueden confundir en situaciones mucho más sencillas. Afortunadamente, Python facilita la eliminación de los espacios en blanco sobrantes de los datos que introduce la gente.

Python puede buscar espacios en blanco adicionales en los lados derecho e izquierdo de una cadena. Para asegurarte de que no hay espacios en blanco en el lado derecho de una cadena, utiliza el método rstrip():

❶ >>> favorite\_language = 'python '  
❷ >>> favorite\_language  
'python '  
❸ >>> favorite\_language.rstrip()  
'python'  
❹ >>> favorite\_language  
'python '

El valor asociado a favorite\_language ❶ contiene espacios en blanco adicionales al final de la cadena. Cuando pides a Python este valor en una sesión de terminal, puedes ver el espacio al final del valor ❷. Cuando el método rstrip() actúa sobre la variable favorite\_language ❸, se elimina este espacio extra. Sin embargo, sólo se elimina temporalmente. Si vuelves a preguntar por el valor de favorite\_language, la cadena tendrá el mismo aspecto que cuando se introdujo, incluido el espacio en blanco ❹ de más .

Para eliminar los espacios en blanco de la cadena de forma permanente, tienes que asociar el valor eliminado al nombre de la variable:

>>> favorite\_language = 'python '  
❶ >>> favorite\_language = favorite\_language.rstrip()  
>>> favorite\_language  
'python'

Para eliminar los espacios en blanco de la cadena, elimina los espacios en blanco de la parte derecha de la cadena y luego asocia este nuevo valor a la variable original ❶. Cambiar el valor de una variable se hace a menudo en programación. Así es como se puede actualizar el valor de una variable mientras se ejecuta un programa o en respuesta a una entrada del usuario.

También puedes eliminar los espacios en blanco del lado izquierdo de una cadena utilizando el método lstrip(), o de ambos lados a la vez utilizando strip():

❶ >>> favorite\_language = ' python '  
❷ >>> favorite\_language.rstrip()  
' python'  
❸ >>> favorite\_language.lstrip()  
'python '  
❹ >>> favorite\_language.strip()  
'python'

En este ejemplo, empezamos con un valor que tiene espacios en blanco al principio y al final ❶. A continuación, eliminamos el espacio sobrante del lado derecho ❷, del lado izquierdo ❸, y de ambos lados ❹. Experimentar con estas funciones de eliminación puede ayudarte a familiarizarte con la manipulación de cadenas. En el mundo real, estas funciones se utilizan sobre todo para limpiar las entradas del usuario antes de almacenarlas en un programa.

### Eliminar prefijos

Al trabajar con cadenas, otra tarea común es eliminar un prefijo. Considera una URL con el prefijo común *https://*. Queremos eliminar este prefijo para centrarnos sólo en la parte de la URL que los usuarios deben introducir en la barra de direcciones. He aquí cómo hacerlo:

>>> nostarch\_url = 'https://nostarch.com'  
>>> nostarch\_url.removeprefix('https://')  
'nostarch.com'

Introduce el nombre de la variable seguido de un punto y, a continuación, el método removeprefix(). Dentro del paréntesis, introduce el prefijo que quieras eliminar de la cadena original.

Al igual que los métodos para eliminar espacios en blanco, removeprefix() no modifica la cadena original. Si quieres conservar el nuevo valor con el prefijo eliminado, reasígnalo a la variable original o asígnalo a una nueva variable:

>>> simple\_url = nostarch\_url.removeprefix('https://')

Cuando veas una URL en una barra de direcciones y no aparezca la parte *https://*, es probable que el navegador esté utilizando un método como removeprefix() entre bastidores.

### Evitar errores de sintaxis con cadenas

Un tipo de error que puedes ver con cierta regularidad es un error de sintaxis. Un *syntax error* se produce cuando Python no reconoce una sección de tu programa como código Python válido. Por ejemplo, si utilizas un apóstrofo entre comillas simples, se producirá un error. Esto ocurre porque Python interpreta todo lo que hay entre la primera comilla simple y el apóstrofo como una cadena. Luego intenta interpretar el resto del texto como código Python, lo que provoca errores.

A continuación te explicamos cómo utilizar correctamente las comillas simples y dobles. Guarda este programa como *apostrophe.py* y ejecútalo:

**apostrophe.py**

message = "One of Python's strengths is its diverse community."  
print(message)

El apóstrofo aparece dentro de un conjunto de comillas dobles, por lo que el intérprete de Python no tiene problemas para leer correctamente la cadena:

One of Python's strengths is its diverse community.

Sin embargo, si utilizas comillas simples, Python no puede identificar dónde debe terminar la cadena:

message = 'One of Python's strengths is its diverse community.'  
print(message)

Verás la siguiente salida:

File "apostrophe.py", line 1  
 message = 'One of Python's strengths is its diverse community.'  
 ❶ ^  
SyntaxError: unterminated string literal (detected at line 1)

En la salida puedes ver que el error se produce justo después de la comilla simple final ❶. Este error de sintaxis indica que el intérprete no reconoce algo en el código como código Python válido, y piensa que el problema puede ser una cadena que no está entrecomillada correctamente. Los errores pueden provenir de diversas fuentes, y señalaré algunas comunes a medida que surjan. Es posible que veas errores de sintaxis con frecuencia a medida que aprendas a escribir código Python correctamente. Los errores de sintaxis son también el tipo de error menos específico, por lo que pueden ser difíciles y frustrantes de identificar y corregir. Si te quedas atascado en un error especialmente persistente, consulta las sugerencias del Apéndice C.

## Nota

La función de resaltado de sintaxis de tu editor debería ayudarte a detectar rápidamente algunos errores de sintaxis mientras escribes tus programas. Si ves código Python resaltado como si fuera inglés o inglés resaltado como si fuera código Python, probablemente tengas una comilla mal colocada en algún lugar de tu archivo.

## Pruébalo tú mismo

Guarda cada uno de los siguientes ejercicios como un archivo independiente, con un nombre como *name\_cases.py*. Si te atascas, tómate un descanso o consulta las sugerencias del Apéndice C.

2-3. Mensaje personal: Utiliza una variable para representar el nombre de una persona, e imprime un mensaje para esa persona. Tu mensaje debe ser sencillo, como "Hola Eric, ¿te gustaría aprender algo de Python hoy?".

2-4. Casos de nombre: Utiliza una variable para representar el nombre de una persona, y luego imprime el nombre de esa persona en minúsculas, mayúsculas y mayúsculas de título.

2-5. Cita célebre: Busca una cita de una persona famosa a la que admires. Imprime la cita y el nombre de su autor. Tu resultado debe ser algo parecido a lo siguiente, incluidas las comillas:

Albert Einstein dijo una vez: "Una persona que nunca cometió un error nunca intentó nada nuevo".

2-6. Cita célebre 2: Repite el Ejercicio 2-5, pero esta vez, representa el nombre de la persona célebre utilizando una variable llamada famous\_person. Luego redacta tu mensaje y represéntalo con una nueva variable llamada message. Imprime tu mensaje.

2-7. Despojar nombres: Utiliza una variable para representar el nombre de una persona e incluye algunos caracteres de espacio en blanco al principio y al final del nombre. Asegúrate de utilizar cada combinación de caracteres, "\t" y "\n", al menos una vez.

Imprime el nombre una vez, para que se muestren los espacios en blanco alrededor del nombre. A continuación, imprime el nombre utilizando cada una de las tres funciones de eliminación, lstrip(), rstrip(), y strip().

2-8. Extensiones de archivo: Python tiene un método removesuffix() que funciona exactamente igual que removeprefix(). Asigna el valor 'python\_notes.txt' a una variable llamada filename. Luego utiliza el método removesuffix() para mostrar el nombre del archivo sin la extensión, como hacen algunos exploradores de archivos.

## Números

Los números se utilizan muy a menudo en programación para llevar la puntuación en los juegos, representar datos en visualizaciones, almacenar información en aplicaciones web, etc. Python trata los números de varias formas distintas, dependiendo de cómo se utilicen. Veamos primero cómo gestiona Python los números enteros, porque son los más sencillos de trabajar.

### Números enteros

En Python puedes sumar (+), restar (-), multiplicar (\*) y dividir (/) números enteros.

>>> 2 + 3  
5  
>>> 3 - 2  
1  
>>> 2 \* 3  
6  
>>> 3 / 2  
1.5

En una sesión terminal, Python simplemente devuelve el resultado de la operación. Python utiliza dos símbolos de multiplicación para representar exponentes:

>>> 3 \*\* 2  
9  
>>> 3 \*\* 3  
27  
>>> 10 \*\* 6  
1000000

Python también admite el orden de las operaciones, por lo que puedes utilizar varias operaciones en una misma expresión. También puedes utilizar paréntesis para modificar el orden de las operaciones, de modo que Python pueda evaluar tu expresión en el orden que especifiques. Por ejemplo:

>>> 2 + 3\*4  
14  
>>> (2 + 3) \* 4  
20

El espaciado en estos ejemplos no afecta a la forma en que Python evalúa las expresiones; simplemente te ayuda a detectar más rápidamente las operaciones que tienen prioridad cuando lees el código.

### Flotantes

Python llama a cualquier número con punto decimal *float*. Este término se utiliza en la mayoría de los lenguajes de programación, y se refiere al hecho de que un punto decimal puede aparecer en cualquier posición de un número. Todo lenguaje de programación debe estar cuidadosamente diseñado para gestionar correctamente los números decimales, de modo que los números se comporten adecuadamente, independientemente de dónde aparezca el punto decimal.

En su mayor parte, puedes utilizar números flotantes sin preocuparte de cómo se comportan. Simplemente introduce los números que quieras utilizar, y lo más probable es que Python haga lo que esperas:

>>> 0.1 + 0.1  
0.2  
>>> 0.2 + 0.2  
0.4  
>>> 2 \* 0.1  
0.2  
>>> 2 \* 0.2  
0.4

Sin embargo, ten en cuenta que a veces puedes obtener un número arbitrario de decimales en tu respuesta:

>>> 0.2 + 0.1  
0.30000000000000004  
>>> 3 \* 0.1  
0.30000000000000004

Esto ocurre en todos los lenguajes y no tiene mayor importancia. Python intenta encontrar una forma de representar el resultado con la mayor precisión posible, lo que a veces es difícil dado cómo los ordenadores tienen que representar los números internamente. Ignora los decimales de más por ahora; aprenderás formas de tratar con los decimales de más cuando lo necesites en los proyectos de la Parte II.

### Enteros y flotantes

Cuando divides dos números cualesquiera, aunque sean enteros y den como resultado un número entero, siempre obtendrás un flotante:

>>> 4/2  
2.0

Si mezclas un entero y un flotante en cualquier otra operación, también obtendrás un flotante:

>>> 1 + 2.0  
3.0  
>>> 2 \* 3.0  
6.0  
>>> 3.0 \*\* 2  
9.0

Python utiliza por defecto un flotador en cualquier operación que utilice un flotador, aunque el resultado sea un número entero.

### Guiones bajos en los números

Cuando escribas números largos, puedes agrupar los dígitos utilizando guiones bajos para que los números grandes sean más legibles:

>>> universe\_age = 14\_000\_000\_000

Cuando imprimes un número definido con guiones bajos, Python sólo imprime los dígitos:

>>> print(universe\_age)  
14000000000

Python ignora los guiones bajos cuando almacena este tipo de valores. Aunque no agrupes los dígitos de tres en tres, el valor no se verá afectado. Para Python, 1000 es lo mismo que 1\_000, que es lo mismo que 10\_00. Esta característica funciona tanto para enteros como para flotantes.

### Asignación múltiple

Puedes asignar valores a más de una variable utilizando una sola línea de código. Esto puede ayudar a acortar tus programas y hacerlos más fáciles de leer; utilizarás esta técnica con más frecuencia al inicializar un conjunto de números.

Por ejemplo, así es como puedes inicializar a cero las variables x, y, y z:

>>> x, y, z = 0, 0, 0

Tienes que separar los nombres de las variables con comas, y hacer lo mismo con los valores, y Python asignará cada valor a su respectiva variable. Siempre que el número de valores coincida con el número de variables, Python las emparejará correctamente.

### Constantes

Una *constant* es una variable cuyo valor permanece invariable a lo largo de la vida de un programa. Python no tiene tipos de constantes incorporados, pero los programadores de Python utilizan letras mayúsculas para indicar que una variable debe tratarse como una constante y no modificarse nunca:

MAX\_CONNECTIONS = 5000

Cuando quieras tratar una variable como una constante en tu código, escribe el nombre de la variable en mayúsculas.

## Pruébalo tú mismo

2-9. Número Ocho: Escribe operaciones de suma, resta, multiplicación y división que den como resultado el número 8. Asegúrate de encerrar tus operaciones en llamadas a print() para ver los resultados. Debes crear cuatro líneas con este aspecto:

print(5+3)

Tus resultados deben ser cuatro líneas, con el número 8 apareciendo una vez en cada línea.

2-10. Número favorito: Utiliza una variable para representar tu número favorito. Luego, utilizando esa variable, crea un mensaje que revele tu número favorito. Imprime ese mensaje.

## Comentarios

Los comentarios son una característica extremadamente útil en la mayoría de los lenguajes de programación. Todo lo que has escrito en tus programas hasta ahora es código Python. A medida que tus programas se hacen más largos y complicados, debes añadir notas dentro de tus programas que describan tu enfoque general del problema que estás resolviendo. Un *comment* te permite escribir notas en tu lenguaje hablado, dentro de tus programas.

### ¿Cómo se escriben los comentarios?

En Python, la almohadilla (#) indica un comentario. Cualquier cosa que siga a una almohadilla en tu código es ignorada por el intérprete de Python. Por ejemplo:

**comment.py**

# Say hello to everyone.  
print("Hello Python people!")

Python ignora la primera línea y ejecuta la segunda.

Hello Python people!

### ¿Qué tipo de comentarios debes escribir?

La razón principal para escribir comentarios es explicar qué se supone que debe hacer tu código y cómo lo estás haciendo funcionar. Cuando estás trabajando en un proyecto, entiendes cómo encajan todas las piezas. Pero cuando vuelves al proyecto después de un tiempo de ausencia, es probable que hayas olvidado algunos detalles en . Siempre puedes estudiar tu código durante un tiempo y averiguar cómo se suponía que debían funcionar los segmentos, pero escribir buenos comentarios puede ahorrarte tiempo al resumir claramente tu planteamiento general.

Si quieres convertirte en un programador profesional o colaborar con otros programadores, debes escribir comentarios significativos. Hoy en día, la mayoría del software se escribe en colaboración, ya sea por un grupo de empleados de una empresa o por un grupo de personas que trabajan juntas en un proyecto de código abierto. Los programadores cualificados esperan ver comentarios en el código, así que es mejor que empieces a añadir comentarios descriptivos a tus programas ahora. Escribir comentarios claros y concisos en tu código es uno de los hábitos más beneficiosos que puedes adquirir como nuevo programador.

Cuando decidas si escribir o no un comentario, pregúntate si has tenido que considerar varios enfoques antes de llegar a una forma razonable de hacer que algo funcione; si es así, escribe un comentario sobre tu solución. Es mucho más fácil borrar más tarde los comentarios sobrantes que volver atrás y escribir comentarios para un programa escasamente comentado. A partir de ahora, utilizaré comentarios en los ejemplos de este libro para ayudar a explicar secciones de código.

## Pruébalo tú mismo

2-11. Añadir comentarios**:** Elige dos de los programas que has escrito y añade al menos un comentario a cada uno. Si no tienes nada específico que escribir porque tus programas son demasiado sencillos en este momento, simplemente añade tu nombre y la fecha actual en la parte superior de cada archivo de programa. Después escribe una frase describiendo lo que hace el programa.

## El Zen de Python

Los programadores experimentados de Python te animarán a evitar la complejidad y a buscar la simplicidad siempre que sea posible. La filosofía de la comunidad Python está contenida en "El Zen de Python" de Tim Peters. Puedes acceder a este breve conjunto de principios para escribir buen código Python introduciendo import this en tu intérprete. No reproduciré aquí todo el "Zen de Python", pero compartiré algunas líneas para ayudarte a entender por qué deberían ser importantes para ti como programador principiante de Python.

>>> import this  
The Zen of Python, by Tim Peters  
Beautiful is better than ugly.

Los programadores de Python abrazan la noción de que el código puede ser bello y elegante. En programación, la gente resuelve problemas. Los programadores siempre han respetado las soluciones a los problemas bien diseñadas, eficientes e incluso bellas. A medida que aprendas más sobre Python y lo utilices para escribir más código, alguien podría mirar un día por encima de tu hombro y decir: "¡Vaya, qué código más bonito!".

Simple is better than complex.

Si puedes elegir entre una solución simple y una compleja, y ambas funcionan, utiliza la solución simple. Tu código será más fácil de mantener, y será más fácil para ti y para otros construir sobre ese código más adelante.

Complex is better than complicated.

La vida real es desordenada, y a veces una solución sencilla a un problema es inalcanzable. En ese caso, utiliza la solución más sencilla que funcione.

Readability counts.

Aunque tu código sea complejo, procura que sea legible. Cuando trabajes en un proyecto que implique una codificación compleja, céntrate en escribir comentarios informativos para ese código.

There should be one-- and preferably only one --obvious way to do it.

Si se pide a dos programadores de Python que resuelvan el mismo problema, deberían llegar a soluciones bastante compatibles. Esto no quiere decir que no haya lugar para la creatividad en la programación. Al contrario, ¡hay mucho espacio para la creatividad! Sin embargo, gran parte de la programación consiste en utilizar pequeños enfoques comunes a situaciones sencillas dentro de un proyecto más amplio y creativo. Las tuercas y los tornillos de tus programas deben tener sentido para otros programadores de Python.

Now is better than never.

Podrías pasarte el resto de tu vida aprendiendo todos los entresijos de Python y de la programación en general, pero entonces nunca completarías ningún proyecto. No intentes escribir código perfecto; escribe código que funcione, y luego decide si mejoras tu código para ese proyecto o pasas a algo nuevo.

Cuando pases al siguiente capítulo y empieces a profundizar en temas más complicados, intenta mantener en mente esta filosofía de simplicidad y claridad. Los programadores experimentados respetarán más tu código y estarán encantados de darte su opinión y colaborar contigo en proyectos interesantes.

## Pruébalo tú mismo

2-12. Zen de Python: Entra en import this en una sesión de terminal de Python y hojea los principios adicionales.

## Resumen

En este capítulo has aprendido a trabajar con variables. Aprendiste a utilizar nombres de variables descriptivos y a resolver errores de nombre y de sintaxis cuando surgen. Aprendiste qué son las cadenas y cómo mostrarlas utilizando minúsculas, mayúsculas y mayúsculas de título. Empezaste a utilizar los espacios en blanco para organizar la salida de forma ordenada, y aprendiste a eliminar los elementos innecesarios de una cadena. Empezaste a trabajar con números enteros y flotantes, y aprendiste algunas de las formas de trabajar con datos numéricos. También aprendiste a escribir comentarios explicativos para que tu código sea más fácil de leer para ti y para los demás. Por último, has leído sobre la filosofía de mantener tu código lo más simple posible, siempre que sea posible.

En el Capítulo 3, aprenderás a almacenar colecciones de información en estructuras de datos llamadas *lists*. También aprenderás a trabajar a través de una lista, manipulando cualquier información de esa lista.