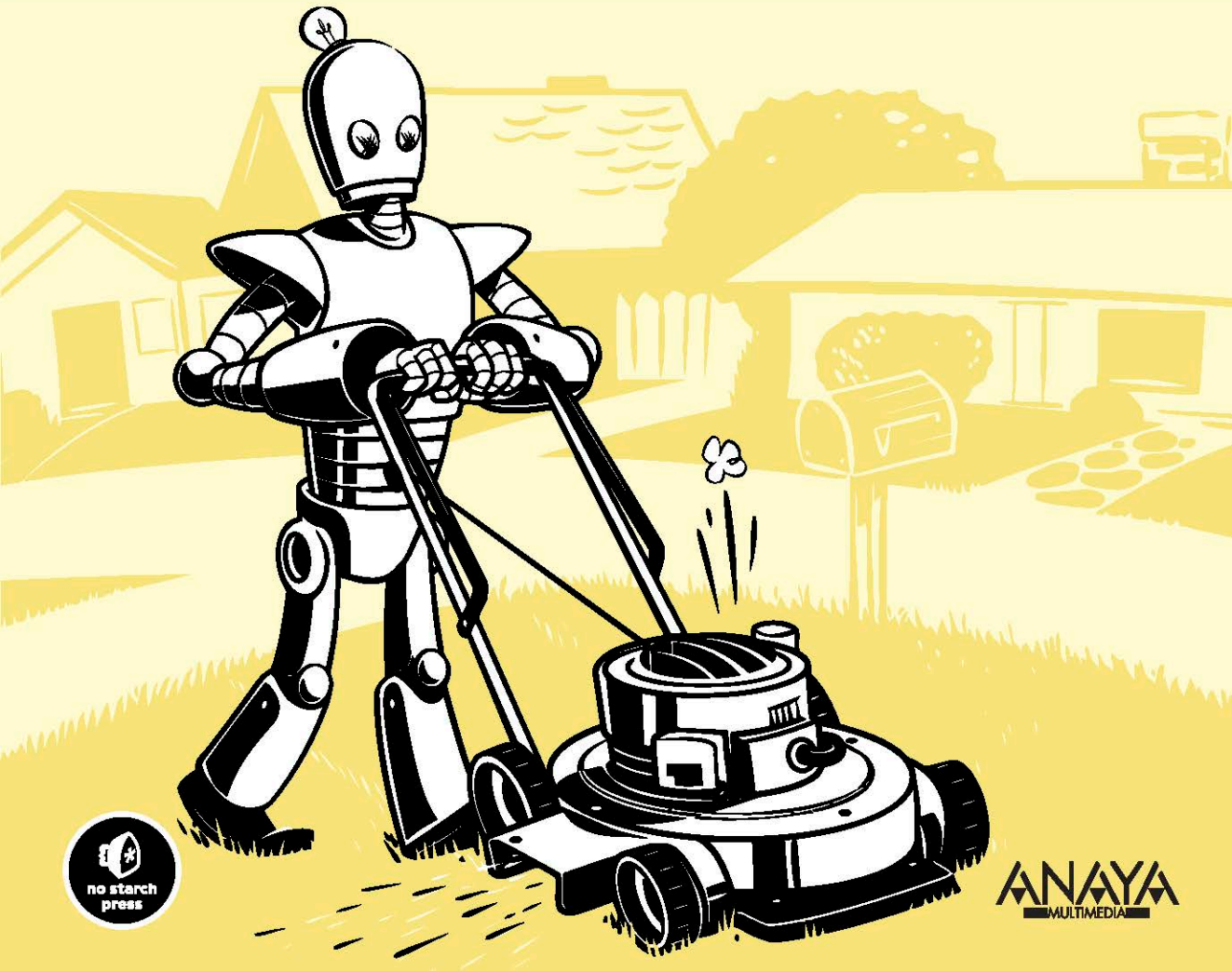
3ª EDICIÓN

AUTOMATIZA LAS TAREAS ABURRIDAS CON PYTHON

PROGRAMACIÓN PRÁCTICA PARA PRINCIPIANTES

AL SWEIGART

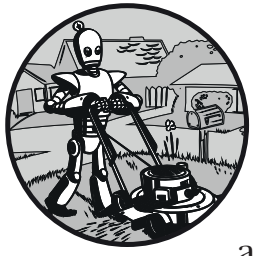


no starch press

ANAYA MULTIMEDIA

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN



"Acabas de hacer en dos horas lo que a nosotros tres nos ha llevado dos días". Mi compañero de habitación en la universidad estaba trabajando en una tienda de electrónica a principios de los años 2000. De vez en cuando, la tienda recibía una hoja de cálculo con miles de precios de productos de otras tiendas. Un equipo de tres empleados tenía que imprimir la hoja en una montaña de papeles y repartírsela entre ellos. Para cada precio de producto, debían buscar el de su tienda y anotar todos los que la competencia vendía más baratos. Por lo general, tardaban un par de días.

"Pues podría escribir un programa para que se ocupe de eso si tenéis el archivo original para imprimir", les dijo mi compañero cuando los vio sentados en el suelo, rodeados de papeles.

Al cabo de dos horas, tenía un programa corto que leía los precios de la competencia de un archivo, buscaba el producto en la base de datos de la tienda y anotaba si la competencia era más barata. Todavía era novato en la programación, así que pasó la mayor parte de ese tiempo buscando documentación en un libro de programación. El programa en sí tardó solo unos segundos en ejecutarse. Mi compañero de habitación y sus compañeros de trabajo se tomaron un buen rato para comer aquel día.

Ese es el poder de la programación informática. Un ordenador es como una navaja suiza con herramientas para un sinfín de tareas. Mucha gente pasa horas haciendo clics y tecleando para completar tareas repetitivas, sin darse cuenta de que la máquina que están usando podría hacer ese trabajo en cuestión de segundos si le diesen las instrucciones adecuadas.

¿Para quién es este libro?

El software es el núcleo de muchas de las herramientas que usamos actualmente: casi todo el mundo usa redes sociales para comunicarse, casi todo el mundo tiene teléfonos interconectados en el bolso o bolsillo y la mayoría de los trabajos de oficina implican interactuar con un ordenador. En consecuencia, la demanda de profesionales que sepan escribir código se ha disparado. Hay innumerables libros, tutoriales y formaciones intensivas para desarrolladores que prometen convertir a ambiciosos principiantes en ingenieros de software con sueldos de seis cifras.

Este libro no es para esa gente. Es para todos los demás.

Por sí solo, el libro no convierte a nadie en desarrollador de software profesional, igual que unas cuantas lecciones de guitarra no convierten a nadie en estrella del rock. Pero, si trabajas en una oficina, eres administrador, profesor o usas un ordenador para trabajo o diversión, aprenderás lo básico de la programación para poder automatizar tareas como estas:

- Mover y renombrar miles de archivos y clasificarlos en carpetas.
- Rellenar formularios en línea sin necesidad de teclear.
- Descargar archivos o copiar texto de un sitio web cada vez que se actualice.
- Hacer que el ordenador envíe notificaciones de texto personalizadas a tu teléfono.
- Actualizar o aplicar formato a hojas de cálculo de Excel.
- Comprobar tu correo electrónico y enviar respuestas prefabricadas.
- Crear bases de datos y consultar información.
- Extraer texto de imágenes y archivos de sonido.

Estas tareas son sencillas, pero consumen mucho tiempo a los humanos y muchas veces son tan triviales o específicas que no existe un software listo para usar que las realice. No obstante, armado con unos pocos conocimientos de programación, puedes conseguir que tu ordenador haga esas tareas por ti.

Convenciones de código empleadas en este libro

Este libro no está diseñado como un manual de referencia; es una guía para principiantes. El estilo del código a veces no se ajusta a las mejores prácticas (por ejemplo, algunos programas utilizan variables globales), pero de ese modo es más fácil aprenderlo. No se cubren conceptos sofisticados de programación, como la programación orientada a objetos, la comprensión de listas y los generadores, por la complejidad

que suponen. Los programadores veteranos podrían ver maneras de cambiar el código del libro para mejorar la eficiencia, pero el objetivo es conseguir programas que funcionen con el menor esfuerzo posible por parte del lector.

¿Qué es la programación?

Las series de la tele y las películas suelen presentar a los programadores tecleando con furia crípticas cadenas de unos y ceros en pantallas brillantes, pero la programación moderna no es tan misteriosa. Programar es escribir instrucciones para el ordenador en un lenguaje que este pueda entender. Tales instrucciones podrían hacer cuentas con números, modificar texto, buscar información en archivos o comunicarse con otros ordenadores por Internet.

Todos los programas usan instrucciones básicas como bloques de construcción. Aquí tenemos algunas de las más habituales, en español:

"Haz esto; luego haz eso".
"Si se da esta condición, realiza esta acción; si no, haz esta otra".
"Realiza esta acción exactamente 27 veces".
"Sigue haciendo eso hasta que esta condición sea verdadera".

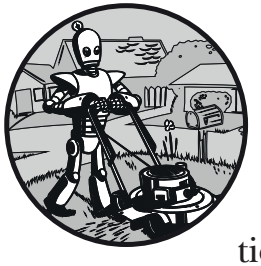
También podemos combinar estos bloques de construcción para implementar decisiones más complejas. Por ejemplo, a continuación tenemos las instrucciones de programación, denominadas código fuente, para un programa sencillo escrito en el lenguaje de programación Python. Empezando desde arriba, el software de Python ejecuta líneas de código (algunas de las cuales solo si se da una condición determinada, si no, Python ejecuta otra línea) hasta llegar al final:

```
❶ password_file = open('ArchivoContraseñaSecreta.txt')
❷ secret_password = password_file.read()
❸ print('Introduce contraseña.')
  typed_password = input()
❹ if typed_password == secret_password:
   ❺ print('Acceso concedido')
   ❻ if typed_password == '12345':
       ❼ print('Esa es la típica contraseña que pone un idiota en la maleta.')
  else:
   ❽ print('Acceso denegado')
```

Es posible que no sepas nada de programación, pero probablemente puedas intuir lo que hace el código anterior leyéndolo sin más. Primero, se abre el archivo `ArchivoContraseñaSecreta.txt` ❶ y, a continuación, se lee la contraseña secreta ❷. Después, se insta al usuario a introducir una contraseña (con el teclado) ❸. Se comparan las dos contraseñas ❹ y, si coinciden, el programa escribe en la pantalla "Acceso concedido" ❺. Luego el programa comprueba si la contraseña es "12345" ❻ y hace notar que esa opción no es la mejor para una contraseña ❼. Si las contraseñas no coinciden, el programa escribe "Acceso denegado" en la pantalla ❽.

3

BUCLES



En el capítulo anterior, aprendiste a hacer que los programas ejecuten algunos bloques de código y se salten otros, pero el control de flujo va más allá. En este capítulo, aprenderás a ejecutar repetidamente bloques de código utilizando bucles. Los dos tipos de bucles de Python, `while` y `for`, desatan todo el poder de la automatización, ya que pueden ejecutar líneas de código millones de veces por segundo. También veremos cómo importar librerías de código, llamadas módulos, para poner más funciones todavía a disposición de tus programas.

Declaraciones de bucle while

Podemos hacer que un bloque de código se ejecute una y otra vez utilizando una declaración `while`. El código de la proposición `while` se ejecutará mientras la condición de la declaración sea `True`. El código de una declaración `while` siempre consta de lo siguiente:

- La palabra clave `while`.
- Una condición (es decir, una expresión que se evalúe como `True` o `False`).
- Dos puntos.
- A partir de la siguiente línea, un bloque de código sangrado (llamado proposición o bloque `while`).

Como verás, una declaración `while` se parece mucho a una declaración `if`. La diferencia está en cómo se comportan: al final de una proposición `if`, la ejecución del programa continúa después de la declaración `if`, mientras que, al final de una proposición `while`, la ejecución vuelve al principio de la declaración `while`. La proposición `while` se denomina a menudo el bucle `while` o, simplemente, el bucle.

Vamos a ver una declaración `if` y un bucle `while` que emplean la misma condición y realizan las mismas acciones basándose en dicha condición. Este es el código con una declaración `if`:

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Y este es el código con una declaración `while`:

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Las dos declaraciones son similares: ambas comprueban el valor de `spam` y, si es menor que `5`, muestran un mensaje. Pero, al ejecutar estos dos fragmentos de código, ocurrirá algo muy distinto con cada uno. Para la declaración `if`, la salida será sencillamente `"Hello, world."`. En cambio, para la declaración `while`, será `"Hello, world."` repetido cinco veces. Echa un vistazo a los diagramas de flujo de estos dos códigos, representados en las figuras 3.1 y 3.2, para ver por qué pasa eso.

El código con la declaración `if` comprueba la condición e imprime `"Hello, world."` solo una vez si esa condición es verdadera. El código con el bucle `while`, por su parte, lo imprime cinco veces. El bucle se detiene después de cinco impresiones porque el entero de `spam` aumenta en uno al final de cada iteración del bucle, lo que significa que este se ejecutará cinco veces antes de que `spam < 5` sea `False`.

En el bucle `while`, la condición siempre se comprueba al inicio de cada iteración (es decir, cada vez que se ejecuta el bucle). Si la condición es `True`, se ejecuta la proposición y se vuelve a comprobar la condición. La primera vez que la condición resulte ser `False`, se omitirá el bucle `while`.

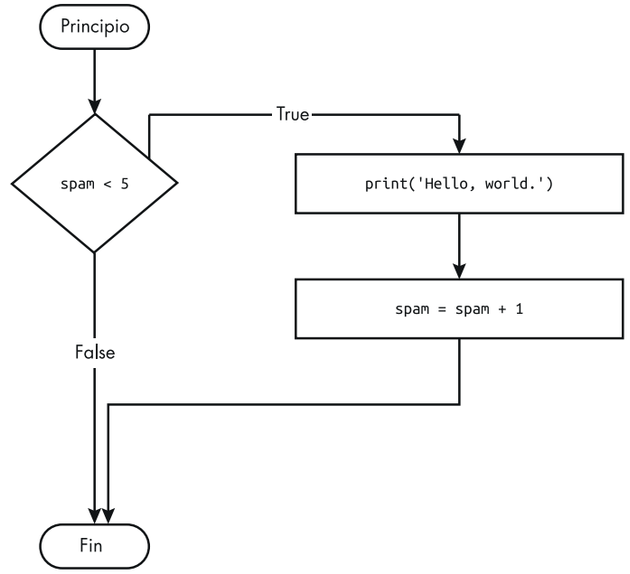


Figura 3.1. Diagrama de flujo para el código de la declaración `if`.

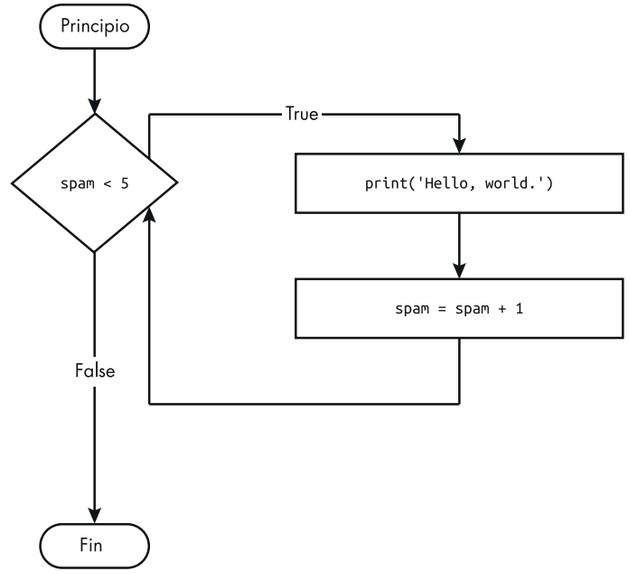
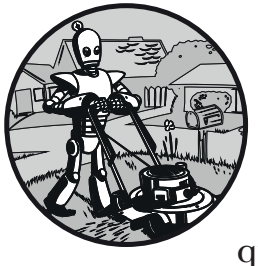


Figura 3.2. Diagrama de flujo para el código de la declaración `while`.

8

EDICIÓN DE CADENAS Y TEXTO



El texto es una de las formas de datos más habituales que manejarán tus programas. Ya sabes cómo concatenar dos valores de cadena con el operador `+`, pero podemos hacer mucho más que eso, como extraer cadenas parciales de valores de cadena, añadir o quitar espacios, poner letras en mayúscula o minúscula y comprobar si una cadena tiene el formato correcto. Podemos incluso escribir código Python para acceder al portapapeles que usamos para copiar y pegar texto.

En este capítulo, aprenderás todo esto y más. Después, trabajaremos en un proyecto de programación para automatizar la aburrida tarea de añadir viñetas a un texto.

Trabajar con cadenas

Echemos un vistazo a algunas de las maneras en las que Python nos permite escribir, imprimir y acceder a cadenas en nuestro código.

Cadenas literales

Aunque los valores de cadena se guardan en la memoria del programa, los valores de cadena que aparecen literalmente en el código se llaman "cadenas literales". Escribir cadenas literales en código Python parece fácil: empiezan y terminan con una comilla

simple y llevan el texto del valor de cadena entre medio. ¿Pero cómo podemos usar comillas dentro de una cadena? Escribir `'That is Alice's cat.'` no funcionará porque Python pensará que la cadena termina después de `Alice` y tratará el resto (`s cat.'`) como código no válido. Por suerte hay varias formas de escribir cadenas literales. El término "cadena" se refiere a un único valor en el contexto de un programa en ejecución y a una cadena literal cuando estamos hablando de escribir código fuente en Python.

Comillas dobles

Las cadenas literales pueden empezar y terminar con comillas dobles, igual que con comillas simples. Una ventaja de usar las dobles es que la cadena puede contener una comilla simple dentro. Escribe lo siguiente en el intérprete de comandos interactivo:

```
>>> spam = "That is Alice's cat."
```

Como la cadena empieza y acaba con comillas dobles, Python sabe que la comilla simple forma parte de la cadena y no la corta ahí. No obstante, si necesitas usar comillas simples y dobles dentro de la cadena, tendrás que recurrir a los caracteres de escape.

Caracteres de escape

Un carácter de escape permite usar caracteres que de otro modo sería imposible poner en una cadena literal. Un carácter de escape consiste en una barra invertida (\) seguida del carácter que se quiere añadir a la cadena. Por ejemplo, `\'` es el carácter de escape para una comilla simple y `\n` es el carácter de escape para el carácter de salto de línea. (Pese a consistir en dos caracteres, a menudo se usa el nombre carácter de escape en singular). Puedes usar esta sintaxis dentro de una cadena que empieza y termina con comillas simples. Para ver cómo funcionan los caracteres de escape, escribe lo siguiente en el intérprete de comandos interactivo:

```
>>> spam = 'Say hi to Bob\'s mother.'
```

Python sabe que, como la comilla de `Bob\'s` lleva una barra invertida, no es una comilla que pretenda terminar el valor de cadena. Los caracteres de escape `\'` y `\"` permiten poner comillas simples y dobles, respectivamente, dentro de una cadena.

La tabla 8.1 recoge los caracteres de escape que puedes usar.

Tabla 8.1. Caracteres de escape.

Carácter de escape	Se imprime como...
`\'`	Comilla simple
`\"`	Comillas dobles
`\t`	Tabulación
`\n`	Salto de línea
`\\`	Barra invertida

Para practicar con esto, escribe lo siguiente en el intérprete de comandos interactivo:

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
Hello there!
How are you?
I'm doing fine.
```

Ten en cuenta que como \ inicia un carácter de escape, si necesitas poner una barra invertida en tu cadena, debes usar el carácter de escape `\\`.

Cadenas crudas

Puedes poner una `r` antes de la comilla inicial de una cadena para marcarla como cadena literal cruda. Una cadena cruda hace que sea más fácil introducir valores de cadena que contengan barras invertidas ignorando todos los caracteres de escape. Para ver un ejemplo, pon esto en el intérprete de comandos interactivo:

```
>>> print(r'The file is in C:\Users\Alice\Desktop')
The file is in C:\Users\Alice\Desktop
```

Como se trata de una cadena cruda, Python considera que la barra invertida es parte de la cadena y no el inicio de un carácter de escape:

```
>>> print('Hello...\n\n...world!')  # Sin cadena cruda.
Hello...
...world!

>>> print(r'Hello...\n\n...world!')  # Con cadena cruda.
Hello...\n\n...world!
```

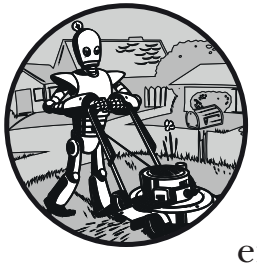
Las cadenas crudas son útiles si los valores de cadena contienen varias barras invertidas, como las utilizadas para las rutas de archivo en Windows, como `r'C:\Users\Al\Desktop'`, o expresiones regulares, que veremos en el siguiente capítulo.

Cadenas multilínea

Aunque puedes usar el carácter de escape `\n` para insertar un salto de línea en una cadena, suele ser más fácil usar cadenas multilínea. Una cadena multilínea de Python empieza y termina con tres comillas simples o dobles. Cualquier comilla, tabulación o salto de línea que quede entre las "comillas triples" se considera parte de la cadena. Las reglas de sangrado de Python para los bloques no se aplican a las líneas de una cadena multilínea.

24

MOTORES DE TEXTO A VOZ Y RECONOCIMIENTO DE VOZ



Este capítulo trata sobre un paquete de conversión de texto a voz, pyttsx3, y un paquete de reconocimiento de voz, Whisper. Los paquetes de texto a voz pueden convertir cadenas de texto en palabras habladas y, después, enviarlas a los altavoces del ordenador o guardarlas en un archivo de sonido. Al añadir esta nueva dimensión a los programas, se libera al usuario de tener que leer el texto en la pantalla. Por ejemplo, una aplicación de recetas de cocina podría leer en voz alta la lista de ingredientes mientras nos movemos por la cocina, y un *script* diario podría recopilar artículos de noticias (o correos electrónicos) y luego preparar un MP3 para reproducirlo durante el trayecto matutino al trabajo.

Por otro lado, las tecnologías de reconocimiento de voz pueden convertir archivos de audio de palabras habladas en valores de cadenas de texto. Podemos utilizar esta capacidad para añadir comandos de voz a nuestros programas o automatizar la transcripción de pódcasts. Y, a diferencia de lo que ocurre con los seres humanos, siempre se puede silenciar a un ordenador que habla demasiado.

Tanto pyttsx3 como Whisper son de uso gratuito y no requieren conexión a Internet. Los motores de texto a voz y de reconocimiento de voz que se presentan en este capítulo no se limitan al inglés, sino que funcionan con la mayoría de los idiomas humanos más hablados.

Motor de texto a voz

Para producir sonido hablado, el paquete de terceros pyttsx3 utiliza el motor de texto a voz integrado en nuestro sistema operativo: Microsoft Speech API (SAPI5) en Windows, NSSpeechSynthesizer en macOS y eSpeak en Linux. En Linux, es posible que tengas que instalar el motor ejecutando `sudo apt install espeak` desde una ventana del terminal. Puedes instalar pyttsx3 ejecutando `pip install pyttsx3` desde un terminal. El apéndice A contiene instrucciones completas para instalar paquetes de terceros.

El nombre del paquete viene de `py` para Python, `tts` para texto a voz, `x` porque es una extensión del paquete `pytts` original y `3` porque es para Python 3.

Generar voz

Producir una voz computarizada es un tema complejo en informática. Por suerte, el motor de texto a voz del sistema operativo hace el trabajo difícil por nosotros y la interacción con este motor es sencilla. Abre el editor de archivos, escribe lo siguiente y guárdalo como `hello_tts.py`:

```
import pyttsx3
engine = pyttsx3.init()
engine.say('Hello. How are you doing?')
engine.runAndWait()  # El ordenador habla.
feeling = input('>')
engine.say('Yes. I am feeling ' + feeling + ' as well.')
engine.runAndWait()  # El ordenador habla otra vez.
```

Después de importar el módulo `pyttsx3`, llamamos a la función `pyttsx3.init()` para inicializar el motor de voz. Esta función devuelve un objeto `Engine`. Podemos pasar una cadena de texto a su método `say()` para indicar al motor qué decir, pero la voz no saldrá hasta que llamemos al método `runAndWait()`. Este método se bloquea (es decir, no vuelve) hasta que el ordenador haya terminado de decir toda la cadena.

El programa no produce salida de texto porque nunca llama a la función `print()`. En su lugar, deberíamos oír al ordenador decir "Hello. How are you doing?". (Asegúrate de que no está silenciado). El usuario puede escribir una respuesta con el teclado, a la que el ordenador replicará oralmente "Yes. I am feeling <tu respuesta> as well".

El módulo `pyttsx3` permite hacer algunos cambios en la voz del ordenador. Podemos pasar las cadenas `'rate'`, `'volume'` y `'voices'` al método `getProperty()` del objeto `Engine` para ver su configuración actual. Introduce lo siguiente en el intérprete de comandos interactivo:


```
>>> import pyttsx3
>>> engine = pyttsx3.init()
>>> engine.getProperty('volume')
1.0
>>> engine.getProperty('rate')
200
>>> engine.getProperty('voices')
[<pyttsx3.voice.Voice object at 0x0000029DA7FB4B10>,
<pyttsx3.voice.Voice object at 0x0000029DAA3DAAD0>]
```

Ten en cuenta que la salida puede ser distinta en tu ordenador. El ajuste del volumen es un flotante, donde `1.0` indica el 100 %. La voz del ordenador habla a una velocidad de 200 palabras por minuto. Continúa el ejemplo con este código:

```
>>> for voice in engine.getProperty('voices'):  # Enumera todas las voces disponibles.
...     print(voice.name, voice.gender, voice.age, voice.languages)
...
Microsoft David Desktop - English (United States) None None []
Microsoft Zira Desktop - English (United States) None None []
```

En mi portátil Windows con idioma inglés (Estados Unidos), `getProperty('voices')` devuelve dos objetos `Voice`. (Fíjate en que la cadena está en plural, `'voices'`, no en singular, `'voice'`). Estos objetos `Voice` tienen atributos `name`, `gender` y `age`, aunque `gender` y `age` están configurados como `None` cuando el sistema operativo no guarda esa información. El atributo `languages` es una lista de los idiomas compatibles con la voz y será una lista vacía si se desconoce esta información.

Sigamos con el ejemplo del intérprete de comandos interactivo llamando al método `setProperty()` para cambiar estos ajustes:

```
>>> engine.setProperty('rate', 300)
>>> engine.setProperty('volume', 0.5)
>>> voices = engine.getProperty('voices')
>>> engine.setProperty('voice', voices[1].id)
>>> engine.say('The quick brown fox jumps over the yellow lazy dog.')
>>> engine.runAndWait()
```

En este ejemplo, hemos cambiado la velocidad a 300 palabras por minuto y hemos bajado el volumen a la mitad pasando `0.5` para `'volume'`. Después, hemos cambiado la voz a la femenina "Zira" que ofrece Windows pasando el atributo `id` del objeto `Voice` en el índice `1` de la lista que devuelve `getProperty('voices')`. Observa también que, para configurar la voz, hemos usado la cadena `'voice'`, en singular, no `'voices'` en plural.

Guardar sonido en archivos WAV

El método `save_to_file()` del módulo `pyttsx3` puede guardar el sonido generado en un archivo WAV (con extensión `.wav`). Escribe lo siguiente en el intérprete de comandos interactivo:

```
>>> import pyttsx3
>>> engine = pyttsx3.init()
>>> engine.save_to_file('Hello. How are you doing?', 'hello.wav')
>>> engine.runAndWait()  # El ordenador crea hello.wav.
```

El primer argumento para `save_to_file()` es una cadena con el discurso que se genera, mientras que el segundo es el nombre del archivo `.wav`. La cadena de texto puede ser una oración corta, como en el ejemplo, o páginas enteras de texto. En mi ordenador, `pyttsx3` fue capaz de convertir una cadena de 1.800 palabras en un archivo de audio de 10 minutos en unos dos segundos. Es importante destacar que solo llamar a `save_to_file()` no es suficiente, también hay que llamar al método `runAndWait()` para que Python cree el archivo `.wav`. El módulo `pyttsx3` puede guardar solo archivos `.wav`, no `.mp3` ni otros formatos de audio.

Reconocimiento de voz

Whisper es un sistema de reconocimiento de voz que puede reconocer varios idiomas. A partir de un archivo de audio o vídeo, Whisper puede devolver el discurso como texto en una cadena de Python. También devuelve los tiempos de inicio y fin de grupos de palabras, que se pueden utilizar para generar archivos de subtítulos.

Instala Whisper ejecutando `pip install openai-whisper` desde el terminal. (Observa que el nombre del paquete de reconocimiento de voz es `openai-whisper`; el paquete `whisper` del sitio web PyPI es otra cosa). Es una descarga de gran tamaño y la instalación puede tardar varios minutos. Además, la primera vez que llamemos a la función `load_model()`, el ordenador descargará el modelo de reconocimiento de voz, que puede tener un tamaño de cientos de megabytes o más.

Supongamos que tenemos un archivo de audio llamado `hello.wav` en el directorio de trabajo actual. (Whisper también puede manejar `.mp3` y varios otros formatos de audio). Podríamos introducir lo siguiente en el intérprete de comandos interactivo:

```
>>> import whisper
>>> model = whisper.load_model('base')
>>> result = model.transcribe('hello.wav')
>>> print(result['text'])
Hello. How are you doing?
```

Después de importar el módulo `whisper`, hay que cargar el modelo de reconocimiento de voz llamando a la función `whisper.load_model()` y pasándole la cadena del modelo de aprendizaje automático entrenado que queremos usar: `'tiny'`, `'base'`, `'small'`, `'medium'` o `'large-v3'`. (Seguirán saliendo nuevos modelos). Los modelos más pequeños pueden transcribir más deprisa, pero los más grandes transcriben con mayor precisión, incluso cuando el audio tiene sonido ambiente de fondo. La primera vez que cargues un modelo, tu ordenador debe estar conectado a Internet para que el módulo `whisper` pueda descargarlo de los servidores de OpenAI. La tabla 24.1 recoge los nombres de los modelos como cadenas que puedes pasar a la función `whisper.load_model()`, junto con su tamaño de archivo, uso de memoria y los resultados de algunas pruebas de tiempo de ejecución en mi ordenador portátil.

Tabla 24.1. Propiedades de los modelos de reconocimiento de voz de Whisper.

Nombre del modelo	Tamaño de archivo del modelo	Memoria necesaria estimada	Tiempo de ejecución para una muestra de audio de 10 palabras, 3 segundos	Tiempo de ejecución para una muestra de audio de 1.800 palabras, 15 minutos
`'tiny'`	74 MB	1 GB	1,9 s	1 m 20 s
`'base'`	142 MB	1 GB	3,0 s	2 m 34 s
`'small'`	472 MB	2 GB	9,6 s	6 m 37 s
`'medium'`	1,5 GB	5 GB	28,6 s	20 m 17 s
`'large-v3'`	3 GB	10 GB	51,9 s	32 m 58 s

Mi opinión personal es que, para el 99 % de los casos, el modelo `'base'` debería ser adecuado y el modelo `'medium'` es suficiente cuando se necesita mayor precisión. Todos los modelos producen errores, por lo que el resultado siempre debe ser revisado por un humano. Puedes probar con modelos más grandes si encuentras muchos errores de transcripción en el texto, o con modelos más pequeños si Whisper tarda demasiado en transcribir el audio. Sin embargo, como se aprecia en la tabla 24.1, hay una diferencia sustancial entre los dos minutos y 34 segundos que tarda el modelo `'base'` en transcribir 15 minutos de audio y los casi 33 que tarda el modelo `'large-v3'`.

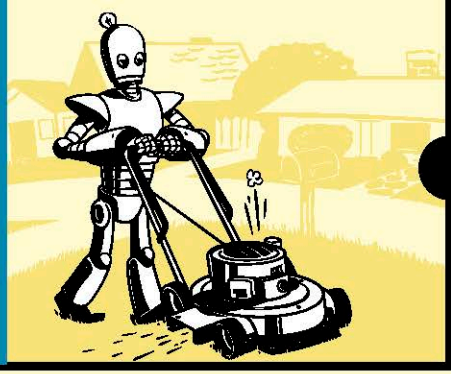
Con el objeto `model.Whisper` que devuelve `whisper.load_model()`, podemos llamar al método `transcribe()` para realizar la transcripción real. Pasa al método una cadena con el nombre del archivo de audio. Este método tardará entre unos segundos y unas horas en ejecutarse, dependiendo del modelo y la duración del archivo de audio. Whisper puede aceptar cualquier archivo de audio o vídeo y lo convierte automáticamente al formato que requiere.

Whisper puede detectar automáticamente el idioma del audio, pero podemos especificar una lengua pasando un argumento de idioma para transcribir, como `model.transcribe('hello.wav', language='English')`. Para encontrar los idiomas que admite Whisper, podemos ejecutar `whisper --help` desde el terminal. Whisper es bastante bueno (aunque no perfecto) a la hora de adivinar dónde debe insertar la puntuación y poner en mayúscula los nombres propios. Sin embargo, siempre debes revisar el resultado para corregir cualquier error.

El diccionario que devuelve `model.transcribe()` tiene varios pares clave-valor, pero la clave `'text'` contiene la cadena de la transcripción.

Por defecto, Whisper utiliza la CPU para transcribir texto, pero, si el ordenador tiene una tarjeta gráfica 3D, se puede acelerar considerablemente las transcripciones configurándolo para utilizar la unidad de procesamiento gráfico (GPU). Encontrarás

APRENDE PYTHON, HAZ COSAS



MÁS DE 750 000 COPIAS VENDIDAS EN TODO EL MUNDO

Si alguna vez has tenido que pasar horas renombrando archivos o actualizando cientos de celdas en una hoja de cálculo, ya sabes lo tedioso que resulta. ¿Y si te dijera que el ordenador puede hacer todo eso por ti?

En esta tercera edición totalmente revisada de *Automatiza las tareas aburridas con Python*, aprenderás a usar Python para escribir programas que hagan en minutos lo que a ti te llevaría horas, y sin necesidad de haber programado nunca. Los primeros capítulos te enseñarán lo básico de Python mediante explicaciones claras y ejemplos entretenidos. Escribirás tu primer programa de Python; trabajarás con cadenas, listas, diccionarios y otras estructuras de datos y, después, usarás expresiones regulares para buscar y manipular patrones de texto.

Cuando ya domines lo básico, emprenderás proyectos que te enseñarán a usar Python para automatizar tareas como:

Buscar por Internet, descargar contenido y rellenar formularios.

- Buscar, extraer y manipular texto y datos en archivos y hojas de cálculo.
- Copiar, mover, renombrar o comprimir archivos guardados en tu ordenador.
- Dividir, fusionar y extraer texto de documentos de Word o en PDF.
- Interactuar con aplicaciones con macros personalizadas para ratón y teclado.
- Gestionar tu buzón de entrada, desapuntarte de listas de distribución y enviar correos o notificaciones de texto.

Novedades en esta edición: Todos los ejemplos de código se han actualizado. También encontrarás cuatro capítulos nuevos sobre la integración de bases de datos, el reconocimiento de voz y la edición de audio y vídeo, además de 16 nuevos proyectos de programación y mayor cobertura de técnicas de desarrollo, como la creación de programas de línea de comandos.

No pierdas tiempo haciendo lo que un mono bien adiestrado podría hacer. Aunque nunca hayas escrito ni una línea de código, puedes delegar ese trabajo monótono en tu ordenador. *Automatiza las tareas aburridas con Python.*

SOBRE EL AUTOR

Al Sweigart es desarrollador de software, miembro de la Python Software Foundation y autor de numerosos libros de programación populares, incluidos *The Big Book of Small Python Projects*, *Beyond the Basic Stuff with Python*, *Coding with Minecraft* y *The Recursive Book of Recursion* (todos de la editorial No Starch Press).

CUBRE PYTHON 3.X

ANAYA MULTIMEDIA
www.anayamultimedia.es
no starch press
ISBN 978-84-415-5278-4
2315285
9 788441 552784