

# Lo Basico

## Cómo empezar rápidamente

No te preocupes por lo que hace este código por ahora. Sólo tienes que escribirlo en la ventana de edición. Asegúrese de guardar como "odd.py" antes de continuar.

```
1 from datetime import datetime
2
3 odds = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
4         21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
5         41, 43, 45, 47, 49, 51, 53, 55, 57, 59]
6
7 for i in range(5):
8     right_this_minute = datetime.today().minute
9
10 if right_this_minute in odds:
11     print("This minute seems a little odd.")
12 else:
13     print("Not an odd minute.")
```

Al ejecutarlo saldría esto:

```
Not an odd minute.
[Finished in 0.6s]
```

## Functions + Modules = The Standard Library

## Funciones + Módulos = La Biblioteca Estándar

La biblioteca estándar de Python es muy rica, y proporciona una gran cantidad de código reutilizable. Veamos otro módulo, llamado **os**, que proporciona una forma independiente de plataforma para interactuar con el sistema operativo subyacente (volveremos al módulo `datetime` en un momento). Concentrémonos en sólo una función proporcionada, **getcwd**, que -cuando se invoca- devuelve su directorio de trabajo actual.

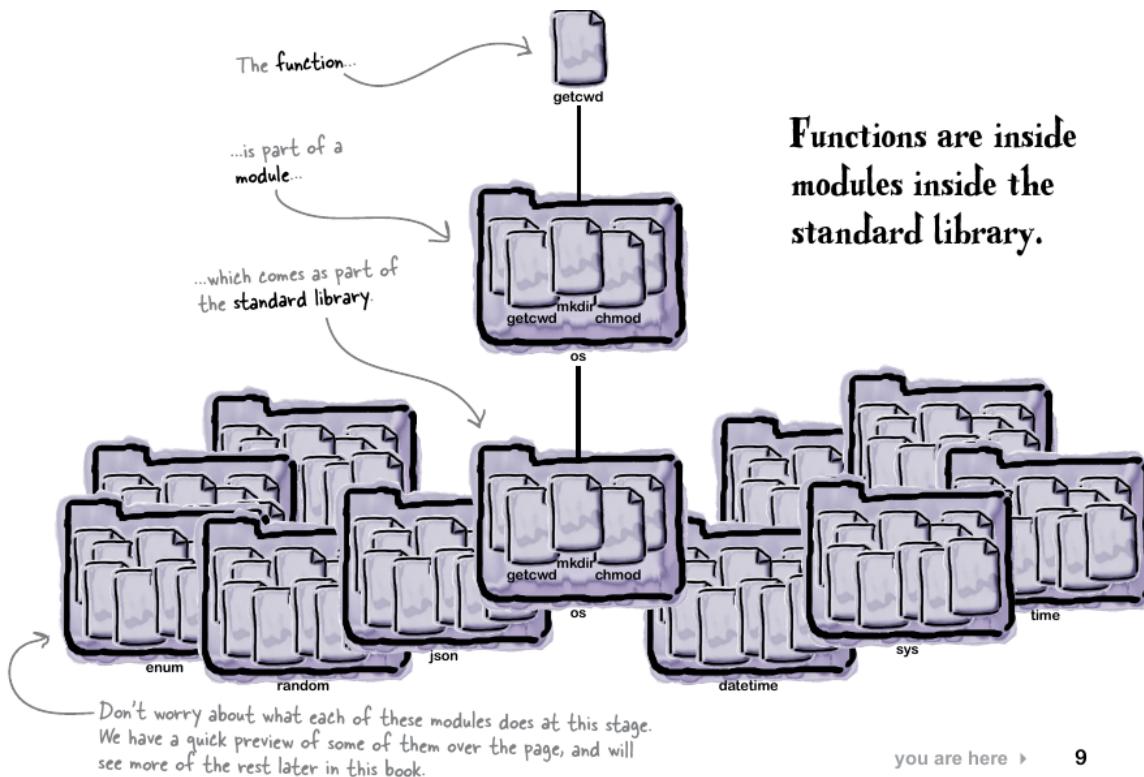
Así es como normalmente se importaría, luego se invocaría, esta función dentro de un programa de Python:

```
from os import getcwd
where_am_I = getcwd()
```

Importar la función de su módulo os y luego invocar según sea necesario.

Una colección de funciones relacionadas compone un módulo, y hay muchos módulos en la biblioteca estándar:

**Las funciones están dentro de los módulos dentro de la biblioteca estándar.**



## Cerrar con la biblioteca estándar - Up Close with the Standard Library

La biblioteca estándar es la joya de la corona de Python, suministrando módulos reutilizables que lo ayudan con todo, desde, por ejemplo, trabajar con datos, manipular archivos ZIP, enviar correos electrónicos, trabajar con HTML. La biblioteca estándar incluso incluye un servidor web, así como la popular tecnología de bases de datos SQLite. En este Up Close, vamos a presentar una visión general de sólo algunos de los módulos más utilizados en la biblioteca estándar. Para seguir adelante, puede ingresar estos ejemplos como se muestra en su mensaje >>> (en IDLE). Si está buscando en la ventana de edición de IDLE, elija Ejecutar ... "Python Shell desde el menú para acceder al mensaje >>>.

Comencemos por aprender un poco sobre el sistema en el que se ejecuta su intérprete. Aunque Python se enorgullece de ser multiplataforma, en que el código escrito en una plataforma puede ser ejecutado (generalmente inalterado) en otro, hay momentos en los que es importante saber que se está ejecutando, digamos, en Mac OS X. El módulo sys existe para ayudarle a aprender más sobre el sistema de su intérprete. A continuación, le indicamos cómo determinar la identidad de su sistema operativo subyacente, primero importando el módulo **sys** y luego accediendo al atributo **platform**:

```
import sys  
sys.platform  
print(sys.platform)
```

su salida es:

```
linux  
[Finished in 0.1s]
```

El módulo sys es un buen ejemplo de un módulo reutilizable que proporciona principalmente acceso a atributos preestablecidos (como la plataforma). Como otro ejemplo, aquí es cómo determinar qué versión de Python se está ejecutando, que pasamos a la función de impresión para mostrar en pantalla:

```
import sys  
print(sys.version)
```

su salida es:

```
3.6.0 (default, Jan 16 2017, 13:35:36)  
[GCC 6.3.1 20170109]  
[Finished in 0.1s]
```

El módulo os es un buen ejemplo de un módulo reutilizable que proporciona funcionalidad, además de proporcionar una forma independiente del sistema para que su código Python interactúe con el sistema operativo subyacente, independientemente del sistema operativo que sea.

Por ejemplo, aquí es cómo calcular el nombre de la carpeta en la que su código está operando usando la función **getcwd**. Como con cualquier módulo, comience por importar el módulo antes de invocar la función:

```
import os  
print(os.getcwd())
```

Su salida es:

```
/home/anton/pruebasparacodigo  
[Finished in 0.1s]
```

Puede acceder a las variables de entorno de su sistema como un todo (utilizando el atributo **environ**) o individualmente (mediante la función **getenv**):

```
import os  
  
print(os.getcwd())  
print("")  
  
print(os.environ)  
  
print("")  
  
print(os.getenv('HOME'))
```

Su salida es:

```
/home/anton/pruebasparacodigo
```

```
environ({'LC_MEASUREMENT': 'es_PE.UTF-8', 'LC_PAPER': 'es_PE.UTF-8', 'LC_MONETARY': 'es_PE.UTF-8', 'XDG_MENU_PREFIX': 'xfce-', '_': '/usr/bin/python', 'LANG': 'es_PE.utf8', 'DISPLAY': ':0.0', 'MOZ_PLUGIN_PATH': '/usr/lib/mozilla/plugins', 'XDG_VTNR': '7', 'PYTHONIOENCODING': 'utf-8', 'SSH_AUTH_SOCK': '/tmp/ssh-k02s8GGxFLQg/agent.587', 'GLADE_CATALOG_PATH': '.', 'LC_NAME': 'es_PE.UTF-8', 'XDG_SESSION_ID': 'c2', 'XDG_GREETER_DATA_DIR': '/var/lib/lightdm-data/anton', 'USER': 'anton', 'GLADE_MODULE_PATH': '.', 'DESKTOP_SESSION': 'xfce', 'QT_QPA_PLATFORMTHEME': 'qt5ct', 'PWD': '/home/anton/pruebasparacodigo', 'HOME': '/home/anton', 'SSH_AGENT_PID': '588', 'XDG_SESSION_TYPE': 'x11', 'XDG_DATA_DIRS': '/usr/local/share:/usr/share', 'XDG_SESSION_DESKTOP': 'xfce', 'LC_ADDRESS': 'es_PE.UTF-8', 'LC_NUMERIC': 'es_PE.UTF-8', 'GLADE_PIXMAP_PATH': '.', 'GTK_MODULES': 'canberra-gtk-module', 'MAIL': '/var/spool/mail/anton', 'SHELL': '/bin/bash', 'XDG_SEAT_PATH': '/org/freedesktop/DisplayManager/Seat0', 'XDG_CURRENT_DESKTOP': 'XFCE', 'SHLVL': '2', 'XDG_SEAT': 'seat0', 'PYTHONPATH': '/home/anton/PP4E/:/home/anton/py3eg', 'LC_TELEPHONE': 'es_PE.UTF-8', 'GDMSESSION': 'xfce', 'LOGNAME': 'anton', 'DBUS_SESSION_BUS_ADDRESS': 'unix:path=/run/user/1000/bus', 'XDG_RUNTIME_DIR': '/run/user/1000', 'XAUTHORITY': '/home/anton/.Xauthority', 'XDG_SESSION_PATH': '/org/freedesktop/DisplayManager/Session0', 'XDG_CONFIG_DIRS': '/etc/xdg', 'PATH': '/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/lib/jvm/default/bin:/usr/bin/site_perl:/usr/bin/vendor_perl:/usr/bin/core_perl', 'LC_IDENTIFICATION': 'es_PE.UTF-8', 'SESSION_MANAGER': 'local/anton-pc:@/tmp/.ICE-unix/581,unix/anton-pc:/tmp/.ICE-unix/581', 'LC_TIME': 'es_PE.UTF-8'})
```

```
/home/anton  
[Finished in 0.2s]
```

El atributo "**environ**" contiene muchos datos.

Puede acceder a un atributo específicamente nombrado (a partir de los datos contenidos en "**environ**") usando "**getenv**".

Trabajar con fechas (y horas) aparece mucho, y la biblioteca estándar proporciona el módulo `datetime` para ayudarlo cuando se trabaja con este tipo de datos. La función `date.today` proporciona la fecha de hoy:

```
import datetime  
print(datetime.date.today())
```

Su salida:

```
2017-04-07  
[Finished in 0.1s]
```

Vemos el dia de hoy en el codigo con el modulo **datetime**

Sin duda, esa es una manera extraña de mostrar la fecha de hoy, ¿no? Puede acceder a los valores de día, mes y año por separado agregando un atributo de acceso a la llamada hasta **date.today**:

```
import datetime  
  
print(datetime.date.today())  
#Los componentes de la fecha de hoy  
print(datetime.date.today().day)  
print(datetime.date.today().month)  
print(datetime.date.today().year)
```

La salida es:

```
2017-04-07  
7  
4  
2017  
[Finished in 0.1s]
```

También puede invocar la función **date.isoformat** y pasar en la fecha de hoy para mostrar una versión mucho más fácil de usar de la fecha de hoy, que se convierte en una cadena por **isoformat**:

```
import datetime  
  
print(datetime.date.today())  
  
print(datetime.date.today().day)  
print(datetime.date.today().month)  
print(datetime.date.today().year)  
print("")  
print(datetime.date.isoformat(datetime.date.today()))
```

La salida es:

2017-04-07  
7  
4  
2017

2017-04-07  
[Finished in 0.1s]

Y luego hay tiempo, que ninguno de nosotros parece tener suficiente. ¿Puede la biblioteca estándar decirnos qué hora es? Sí. Después de importar el módulo de **time**, llame a la función **strftime** y especifique cómo desea que aparezca la hora. En este caso, estamos interesados en los valores de hora actual (% H) y minutos (% M) en formato de 24 horas:

```
import time

print(time.strftime("%H:%M"))
```

La salida:

00:19  
[Finished in 0.1s]

¿Qué tal si trabajas el día de la semana, y si es antes del mediodía? Usar la especificación **% A % p** con strftime hace exactamente eso:

```
import time

print(time.strftime("%H:%M"))
print("")
print(time.strftime("%A %p"))
```

Su salida es:

00:21

Friday AM  
[Finished in 0.1s]

Como ejemplo final del tipo de funcionalidad reutilizable que proporciona la biblioteca estándar, imagine que tiene algún código HTML que le preocupa podría contener algunas etiquetas <script> potencialmente peligrosas. En lugar de analizar el código HTML para detectar y eliminar las etiquetas, ¿por qué no codificar todos aquellos brackets angulares problemáticos utilizando la función de **escape** del **módulo html**? O tal vez usted tiene algún código HTML codificado que le gustaría volver a su forma original? La función **unescape** puede hacer eso. Aquí hay ejemplos de ambos:

```

import html

print(html.escape("This HTML fragment contains a <script>script</script> tag."))
print("")
print(html.unescape("I &hearts; Python's &lt;standard library&gt;"))

```

Su salida es:

This HTML fragment contains a &lt;script&gt;script&lt;/script&gt; tag.

I ♥ Python's <standard library>. [Finished in 0.2s]

## Data Structures Come Built-in

### Las estructuras de datos vienen incorporadas

Además de contar con una biblioteca estándar de primera clase, Python también tiene algunas poderosas estructuras de datos incorporadas. Una de ellas es la **list**, que se puede considerar como una matriz muy potente. Al igual que las matrices en muchos otros idiomas, las listas en Python se adjuntan entre corchetes ([]).

Las siguientes tres líneas de código en nuestro programa (que se muestra a continuación) asignan una lista literal de números impares a una variable llamada probabilidades u odds. En este código, odds es una lista de enteros, pero las listas en Python pueden contener cualquier tipo de datos, e incluso puedes mezclar los tipos de datos en una lista (si eso es lo que estás haciendo). Observe cómo la lista de probabilidades se extiende sobre tres líneas, a pesar de ser una sola declaración. Esto está bien, ya que el intérprete no decidirá que una sola sentencia ha llegado a su fin hasta que encuentre el corchete de cierre () que coincide con el de apertura (). Normalmente, el final de la línea marca el final de una sentencia en Python, pero puede haber excepciones a esta regla general, y las listas multilínea son sólo una de ellas (nos encontraremos con las otras más adelante).

Al igual que los **arrays**, las listas pueden contener datos de cualquier tipo.

The diagram shows a code snippet with annotations explaining list assignment and multiline statements.

```

from datetime import datetime

odds = [ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
        21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
        41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]
        ...

```

**Annotations:**

- An arrow points to the line `odds =` with the text: "This is a new variable, called "odds", which is assigned a list of odd numbers."
- A brace groups the entire list definition from `odds =` to the closing bracket. To its right, text states: "This is the list of odd numbers, enclosed in square brackets. This single statement extends over three lines, which is OK."

Esta es una nueva variable, llamada "**odds**", o impares a la que se le asigna una lista de números impares.

Esta es la lista de números impares, entre corchetes. Esta única sentencia se extiende sobre tres líneas, que está bien.

Hay muchas cosas que se pueden hacer con las listas, pero vamos a aplazar cualquier discusión adicional hasta un capítulo posterior. Todo lo que necesitas saber ahora es que esta lista ya existe, ha sido asignada a la variable **odds** (gracias al uso del operador de asignación, `=`), y contiene los números mostrados.

## **Python variables are dynamically assigned - Las variables de Python se asignan dinámicamente**

Antes de llegar a la siguiente línea de código, quizás se necesiten unas pocas palabras sobre las variables, especialmente si usted es uno de esos programadores que podrían estar acostumbrados a variables predeclarando con información de tipo antes de usarlas (como es el caso en los lenguajes de programación estáticamente mecanografiados).

En Python, las variables aparecen en la existencia la primera vez que las usa, y su tipo no necesita ser predeclarado. Las variables de Python toman su información de tipo del tipo del objeto que están asignados. En nuestro programa, la variable de **odds o impares** se asigna una lista de números, por lo que las probabilidades son una lista en este caso.

Veamos otra declaración de asignación de variables. Como la suerte lo tendría, esto acaba de pasar a ser también la siguiente línea de código en nuestro programa.

Python viene con todos los operadores habituales, incluyendo `<,>`, `<=,> =`, `==,! =`, Así como el operador de asignación.

## **Invoking Methods Obtains Results - Invocación de métodos obtenidos Resultados**

La tercera línea de código en nuestro programa es otra declaración de asignación.

A diferencia del último, éste no asigna una estructura de datos a una variable, sino que asigna el resultado de una llamada de método a otra nueva variable, llamada **right\_this\_minute**. Echa un vistazo a la tercera línea de código:

Here's another variable being created and assigned a value.

```
from datetime import datetime

odds = [ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
         21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
         41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]

right_this_minute = datetime.today().minute

if right_this_minute in odds:
    print("This minute seems a little odd.")
else:
    print("Not an odd minute.")
```

This call generates a value to assign to the variable.

```
from datetime import datetime
```

```
odds = [ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
         21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
         41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]
```

```
right_this_minute = datetime.today().minute
```

```
print(right_this_minute)

if right_this_minute in odds:
    print("Este minuto parece un pequeño impar.")
else:
    print("No es un minuto impar.)")
```

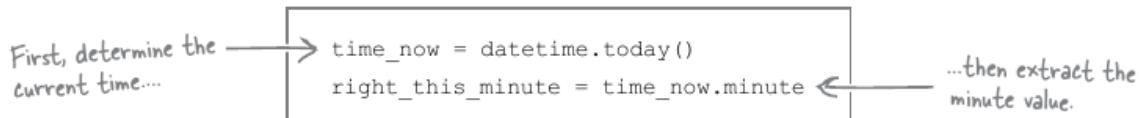
La salida es:

43

Este minuto parece un pequeño impar.  
[Finished in 0.1s]

La tercera línea de código invoca un método llamado **today** que viene con el submódulo **datetime**, que es parte del módulo **datetime** (dijimos que esta estrategia de nomenclatura era un poco confusa). Se puede decir que hoy se está invocando debido a los paréntesis estándar de postfix: () .

Es tentador dividir esta única línea de código en dos líneas para hacerla "más fácil de entender", de la siguiente manera:



```
from datetime import datetime

odds = [ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
         21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
         41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]

#right_this_minute = datetime.today().minute
time_now = datetime.today()
right_this_minute = time_now.minute

print(right_this_minute)

if right_this_minute in odds:
    print("Este minuto parece un pequeño impar.")
else:
    print("No es un minuto impar.")
```

Su salida es la misma:

```
49
Este minuto parece un pequeño impar.
[Finished in 0.1s]
```

Puede hacerlo (si lo desea), pero la mayoría de los programadores de Python prefieren no crear la variable temporal (`time_now` en este ejemplo) a menos que sea necesario en algún momento posterior en el programa.

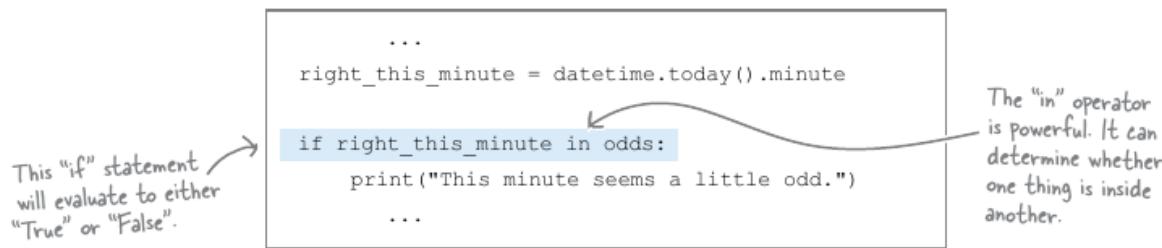
## Deciding When to Run Blocks of Code Decidir cuándo ejecutar bloques de código

En esta etapa tenemos una lista de números llamados impares u `odds`. También tenemos un valor minuto llamado `right_this_minute`. Para averiguar si el valor minuto actual

almacenado en `right_this_minute` es un número odds necesitamos alguna manera de determinar si está en la lista de odss ¿Pero cómo hacemos esto?

Resulta que Python hace este tipo de cosas muy sencillas. Además de incluir todos los operadores de comparación habituales que esperas encontrar en cualquier lenguaje de programación (como, por ejemplo, `>`, `<`, `=`, `<=`, etc.), Python viene con unos pocos operadores "super", Uno de los cuales está en.

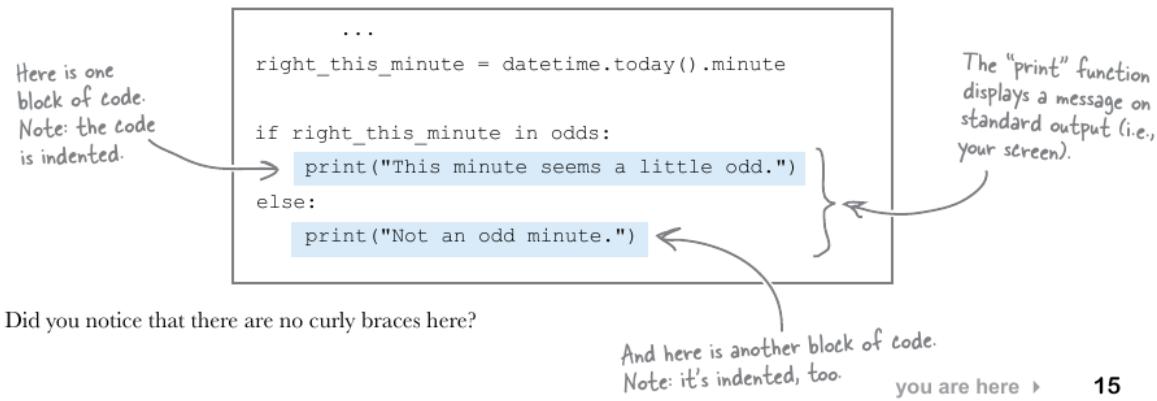
El operador `in` comprueba si una cosa está dentro de otra. Echa un vistazo a la siguiente línea de código en nuestro programa, que utiliza el operador `in` para comprobar si `right_this_minute` está dentro de la lista de impares u odds:



El operador `in` devuelve `True` o `False`. Como era de esperar, si el valor en `right_this_minute` está en `odds`, la sentencia `if` se evalúa como `True` y se ejecuta el bloque de código asociado con la sentencia `if`.

### Los bloques en Python son fáciles de detectar, ya que siempre están sangrados.

En nuestro programa hay dos bloques, que cada uno contiene una sola llamada a la función de `print`. Esta función puede mostrar mensajes en pantalla (y veremos muchos usos de él a lo largo de este libro). Cuando ingresa este código de programa en la ventana de edición, es posible que haya notado que IDLE ayuda a mantenerlo en línea recta automáticamente. Esto es muy útil, pero asegúrese de comprobar que la indentación de IDLE es lo que desea:



you are here ▶ 15

## What Happened to My Curly Braces? ¿Qué pasó con mis llaves cerradas?

Si estás acostumbrado a un lenguaje de programación que usa llaves ({y}) para delimitar bloques de código, encontrar bloques en Python por primera vez puede ser desorientador, ya que Python no utiliza rizadores para este propósito. Python utiliza la indentación para demarcar un bloque de código, que los programadores de Python prefieren llamar suite en lugar de bloquear (sólo para mezclar las cosas un poco).

En lugar de referirse a un "bloque" de código, los programadores de Python usan la palabra "suite". Ambos nombres se usan en la práctica, pero los documentos de Python prefieren "suite".

No es que las llaves no tengan un uso en Python. Lo hacen, pero -como veremos en el capítulo 3- las llaves tienen más que ver con la delimitación de datos de lo que tienen que ver con delimitar suites (es decir, bloques) de código.

Las suites dentro de cualquier programa de Python son fáciles de detectar, ya que siempre están sangradas. Esto ayuda a que su cerebro identifique rápidamente las suites al leer el código. La otra pista visual que debe buscar es el carácter de dos puntos (:), que se utiliza para introducir una suite que está asociada con cualquiera de las sentencias de control de Python (como if, else, for y similares). Verá muchos ejemplos de este uso a medida que avance en este libro.

### Un dos puntos introduce un conjunto de código

El colon o dos puntos(:) es importante, ya que introduce un nuevo conjunto de código que debe estar sangrado a la derecha. Si se olvida de sangrar el código después de dos puntos, el intérprete genera un error. No sólo la declaración if en nuestro ejemplo tiene dos puntos, sino que también tiene uno. Aquí está todo el código de nuevo:

```

from datetime import datetime

odds = [ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19,
         21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
         41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]

right_this_minute = datetime.today().minute

if right_this_minute in odds:
    print("This minute seems a little odd.")
else:
    print("Not an odd minute.")

```

Colons introduce indented suites.

## What “else” Can You Have with “if”?

### ¿Qué "else" puedes tener con "if"?

```

if right_this_minute in odds:
    print("This minute seems a little odd.")
else:
    print("Not an odd minute.")

```

### Ninguno. Python lo deletrea elif.

Si usted tiene un número de condiciones que usted necesita comprobar como parte de una declaración **if**, Python proporciona **elif**, así como otra cosa. Puede tener tantas declaraciones **elif** (cada una con su propia suite) según sea necesario.

He aquí un pequeño ejemplo que asume que una variable llamada **today** o **today** se ha asignado previamente una cadena que representa lo que hoy es:

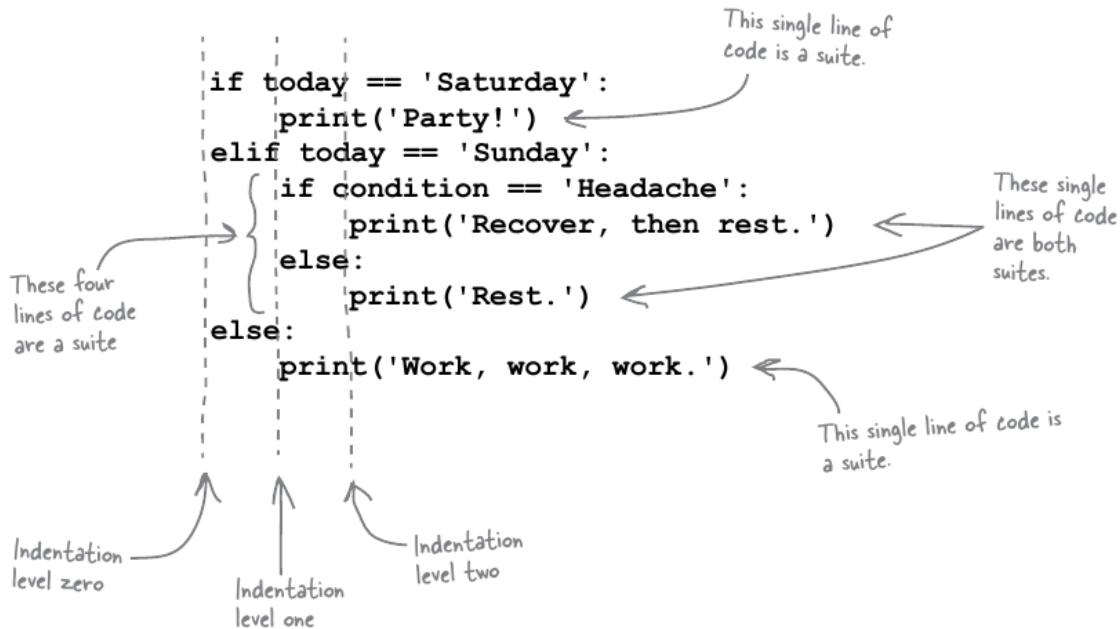
```

if today == 'Saturday':
    print('Party!!')
elif today == 'Sunday':
    print('Recover.')
else:
    print('Work, work, work.')

```

Tres suites individuales: una para el "if", otra para el "elif" y la última para el "else".

## Los suites pueden contener las suites incorporadas



## Iterating Over a Sequence of Objects

### Iterando sobre una secuencia de objetos

Dijimos anteriormente que íbamos a emplear Python para el bucle aquí. El bucle **for** es perfecto para controlar el bucle cuando sabes de antemano cuántas iteraciones necesitas. (Cuando no sabes, te recomendamos el bucle while, pero ahorraremos discutiendo los detalles de este constructo de bucle alternativo hasta que realmente lo necesitamos). En esta etapa, todo lo que necesitamos es **for**, así que vamos a verlo en acción en el prompt >>>.

#### Utilice "for" cuando haga un bucle un número de veces conocido.

Presentamos tres usos típicos de. Veamos cuál es el que mejor se adapta a nuestras necesidades.

**Ejemplo de uso 1.** Este bucle for, a continuación, toma una lista de números e itera una vez para cada número en la lista, mostrando el número actual en la pantalla. A medida que lo hace, el bucle for asigna cada número a su vez a una variable de iteración de bucle, que se da el nombre i en este código.

Dado que este código es más que una sola línea, el intérprete de comandos de shell automáticamente para usted cuando presione Intro después de los dos puntos. Para indicar al shell que ha terminado de introducir el código, presione Intro dos veces al final de la suite del bucle:

```
for i in [1, 2, 3]:  
    print(i)
```

Output:

```
1  
2  
3  
[Finished in 0.1s]
```

Observe la indentación y los dos puntos. Al igual que si las sentencias, el código asociado a una sentencia `for` tiene que ser sangrado.

**Ejemplo de uso 2.** Este bucle `for`, a continuación, itera sobre una cadena, con cada carácter en la cadena que se procesa durante cada iteración. Esto funciona porque una cadena en Python es una secuencia. Una secuencia es una colección ordenada de objetos (y veremos muchos ejemplos de secuencias en este libro), y cada secuencia en Python puede ser iterada por el intérprete.

```
for ch in "Hi!":  
    print(ch)
```

Output:

```
H  
i  
!  
[Finished in 0.6s]
```

En ninguna parte tuviste que decirle al bucle lo grande que es la cuerda. Python es lo suficientemente inteligente como para resolver cuando termina la cadena y arregla terminar (es decir, finalizar) el bucle `for` en su nombre cuando agota todos los objetos de la secuencia.

### Iterando un número específico de veces

Además de utilizar para iterar sobre una secuencia, puede ser más exacto y especificar un número de iteraciones, gracias a la función incorporada llamada `rango` o `range`.

Veamos otro ejemplo de uso que muestra el uso de `range`.

**Ejemplo de uso 3.** En su forma más básica, `range` acepta un solo argumento entero que dicta cuántas veces se ejecuta el bucle `for` (veremos otros usos de `rango` más adelante en este libro). En este bucle, utilizamos `range` para generar una lista de números que se asignan uno a la vez a la variable `num`:

```
for num in range(5):
    print('Head First Rocks!')
```

Output:

```
Head First Rocks!
[Finished in 0.2s]
```

El bucle for no utilizó la variable de iteración de bucle num en ninguna parte de la suite del bucle. Esto no planteó un error, que está bien, ya que depende de usted (el programador) para decidir si o no num debe ser procesado más en la suite. En este caso, no hacer nada con num está bien.

### Modificando el código de los números impares o odds con el bucle for:

```
from datetime import datetime

odds = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
        21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
        41, 43, 45, 47, 49, 51, 53, 55, 57, 59]

for i in range(5):
    right_this_minute = datetime.today().minute

    if right_this_minute in odds:
        print("This minute seems a little odd.")
    else:
        print("Not an odd minute.")
```

```
from datetime import datetime

odds = [ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
        21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
        41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]

for i in range(5):
    right_this_minute = datetime.today().minute
```

```

if right_this_minute in odds:
    print("Este minuto parece un pequeño impar.")
else:
    print("No es un minuto impar.")

```

### Outputs:

No es un minuto impar.  
 [Finished in 0.1s]

### Ejercicio:

Su trabajo consiste en poner todo de nuevo juntos, para que podamos ejecutar la nueva versión de nuestro programa y confirmar que está funcionando como se requiere.

from datetime import datetime

Decide which code magnet goes in each of the dashed-line locations.

.....

.....

```

odds = [ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19,
         21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
         41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]

```

.....

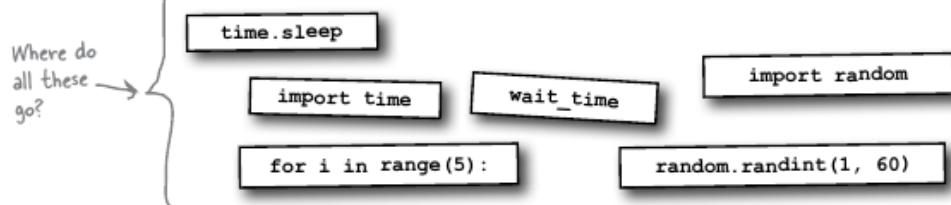
.....

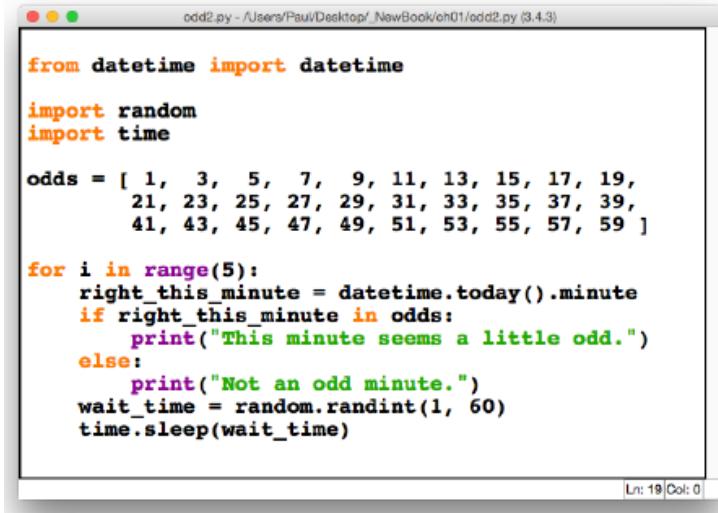
```

right_this_minute = datetime.today().minute
if right_this_minute in odds:
    print("This minute seems a little odd.")
else:
    print("Not an odd minute.")
wait_time = .....(.....)

```

.....





```

odd2.py - /Users/Paul/Desktop/_NewBook/ch01/odd2.py (3.4.3)

from datetime import datetime
import random
import time

odds = [ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
         21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
         41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]

for i in range(5):
    right_this_minute = datetime.today().minute
    if right_this_minute in odds:
        print("This minute seems a little odd.")
    else:
        print("Not an odd minute.")
    wait_time = random.randint(1, 60)
    time.sleep(wait_time)

```

Ln: 19 Col: 0

## Revisión de nuestros experimentos

Antes de seguir adelante y cambiar su programa, analicemos rápidamente el resultado de nuestros experimentos con shell.

Comenzamos escribiendo un bucle for, que itera cinco veces:

```

>>> for num in range(5):
    print('Head First Rocks!')

Head First Rocks!

```

We asked for a range of five numbers,  
so we iterated five times, which  
results in five messages.

Luego utilizamos la función sleep del módulo de tiempo para detener la ejecución de nuestro código durante un número de segundos especificado:

```

>>> import time
>>> time.sleep(5) ← The shell imports the "time" module, letting us
                           invoke the "sleep" function.

```

Y luego experimentamos con la función randint (desde el módulo aleatorio) para generar un entero aleatorio desde un rango proporcionado:

```

>>> import random
>>> random.randint(1,60)
12
>>> random.randint(1,60)
42
>>> random.randint(1,60)
17

```

Note: different integers are generated  
once more, as "randint" returns a different  
random integer each time it's invoked.

Ahora podemos poner todo esto juntos y cambiar nuestro programa.

```
1 import random
2 import time
3
4
5 odds = [ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19,
6         21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
7         41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]
8
9
10 for i in range(5):
11     right_this_minute = datetime.today().minute
12
13     if right_this_minute in odds:
14         print("Este minuto parece un pequeño impar.")
15     else:
16         print("No es un minuto impar.")
17
18     wait_time = random.randint(1,60)
19     time.sleep(wait_time)
```

Ouput:

```
Este minuto parece un pequeño impar.
No es un minuto impar.
No es un minuto impar.
Este minuto parece un pequeño impar.
Este minuto parece un pequeño impar.
```

## Coding a Serious Business Application

### Codificación de una aplicación empresarial seria

Con una punta del sombrero a Head First Java, echemos un vistazo a la versión de Python de la primera aplicación seria de este clásico: la canción de cerveza.

A continuación se muestra una captura de pantalla de la versión de Python del código de la canción de la cerveza. Aparte de una ligera variación en el uso de la función de rango (que discutiremos en un bit), la mayoría de este código debería tener sentido. La ventana de edición IDLE contiene el código, mientras que el final de la salida del programa aparece en una ventana de shell:

```

word = "bottles"
for beer_num in range(10, 0, -1):
    print(beer_num, word, "of beer on the wall.")
    print(beer_num, word, "of beer.")
    print("Take one down.")
    print("Pass it around.")
    if beer_num == 1:
        print("No more bottles of beer on the wall.")
    else:
        new_num = beer_num - 1
        if new_num == 1:
            word = "bottle"
        print(new_num, word, "of beer on the wall.")
print()

```



### Su Output:

10 bottles of beer on the wall.

10 bottles of beer.

Take one down.

pass it around.

9 bottles of beer on the wall.

9 bottles of beer on the wall.

9 bottles of beer.

Take one down.

pass it around.

8 bottles of beer on the wall.

8 bottles of beer on the wall.

8 bottles of beer.

Take one down.

pass it around.

7 bottles of beer on the wall.

7 bottles of beer on the wall.

7 bottles of beer.

Take one down.

pass it around.

6 bottles of beer on the wall.

6 bottles of beer on the wall.

6 bottles of beer.

Take one down.

pass it around.

5 bottles of beer on the wall.

 A screenshot of a Mac OS X desktop showing a terminal window titled 'beersong.py - /Users/Paul/Desktop/\_NewBook/ch01/beersong.py (3.4.3)'. The window displays the Python code for the beer song, which prints lyrics for 10 bottles of beer down to 1 bottle.


 A screenshot of a Python 3.4.3 Shell window titled 'Python 3.4.3 Shell'. It shows the output of running the 'beersong.py' script, displaying the lyrics for 10 bottles of beer down to 1 bottle.

5 bottles of beer on the wall.

5 bottles of beer.

Take one down.

pass it around.

4 bottles of beer on the wall.

4 bottles of beer on the wall.

4 bottles of beer.

Take one down.

pass it around.

3 bottles of beer on the wall.

3 bottles of beer on the wall.

3 bottles of beer.

Take one down.

pass it around.

2 bottles of beer on the wall.

2 bottles of beer on the wall.

2 bottles of beer.

Take one down.

pass it around.

1 bottle of beer on the wall.

1 bottle of beer on the wall.

1 bottle of beer.

Take one down.

pass it around.

No more bottles of beer on the wall.

[Finished in 0.1s]

## Tratar con toda esa cerveza ...

Con el código mostrado arriba escrito en una ventana de edición IDLE y guardado, presionando F5 produce una gran cantidad de salida en el shell. Sólo hemos mostrado un poco de la salida resultante en la ventana de la derecha, ya que la canción de la cerveza comienza con 99 botellas de cerveza en la pared y cuenta hasta que no haya más cerveza. De hecho, el único giro real en este código es cómo maneja esta "cuenta atrás", así que echemos un vistazo a cómo funciona antes de mirar el código del programa en detalle.

## Starting , stopping , and stepping

Dado que el alcance o rango no es el único lugar en el que te encuentras **star**, **stop** y **step**, tomemos un momento para describir lo que cada uno de estos medios, antes de mirar algunos ejemplos representativos (en la página siguiente):

```

>>> help(range)
Help on class range in module builtins:

class range(object)
| range(stop) -> range object ←
| range(start, stop[, step]) -> range object
|
| Return a sequence of numbers from start to stop by step.

```

The "range" function can be invoked in one of two ways.

**El valor START le permite controlar desde WHERE el rango comienza.** Hasta ahora, hemos utilizado la versión de un solo argumento del rango, que -de la documentación- espera un valor para que el stop sea proporcionado. Cuando no se proporciona ningún otro valor, el rango predeterminado utiliza 0 como valor inicial, pero puede establecerlo en un valor de su elección. Cuando lo haga, debe proporcionar un valor para detener. De esta manera, el rango se convierte en una invocación de múltiples argumentos.

**El valor STOP le permite controlar cuándo termina el rango.** Ya hemos visto esto en uso cuando invocamos range (5) en nuestro código. Tenga en cuenta que el rango que se genera nunca contiene el valor stop, por lo que es un caso de parada up-to-but-not-including.

**El valor STEP le permite controlar cómo se genera el rango.** Al especificar valores de inicio y parada, también puede (opcionalmente) especificar un valor para el paso. Por defecto, el valor de paso es 1, y esto le dice al rango que genere cada valor con un paso de 1; Es decir, 0, 1, 2, 3, 4, y así sucesivamente. Puede establecer paso a cualquier valor para ajustar el paso tomado. También puede ajustar el paso a un valor negativo para ajustar la dirección del rango generado.

## Experimenting with Ranges

Ahora que ya sabes un poco sobre el **start**, el **stop** y el **step**, vamos a experimentar en el shell para aprender cómo podemos usar la función de rango para producir muchos rangos diferentes de enteros.

Para ayudar a ver lo que está pasando, utilizamos otra función, lista, para transformar el rango de salida en una lista de lectura humana que podemos ver en la pantalla:

```

>>> range(5) ← This is how we used "range" in our first program.
range(0, 5)

>>> list(range(5)) ← Feeding the output from "range" to "list" produces a list.
[0, 1, 2, 3, 4]

>>> list(range(5, 10)) ← We can adjust the START and STOP values for "range".
[5, 6, 7, 8, 9] ← It is also possible to adjust the STEP value.

>>> list(range(0, 10, 2))
[0, 2, 4, 6, 8]

>>> list(range(10, 0, -2)) ← Things get really interesting when you adjust the
[10, 8, 6, 4, 2] range's direction by negating the STEP value.

>>> list(range(10, 0, 2)) ← Python won't stop you from being silly. If your START
[] value is bigger than your STOP value, and STEP is positive,
you get back nothing (in this case, an empty list).

>>> list(range(99, 0, -1))
[99, 98, 97, 96, 95, 94, 93, 92, ... 5, 4, 3, 2, 1]

```

Después de todas nuestras experimentaciones, llegamos a una invocación de gama (mostrada en último lugar, arriba) que produce una lista de valores de 99 a 1, que es exactamente lo que hace la canción de la cerveza para loop:

```

word = "bottles"
for beer_num in range(99, 0, -1):
    print(beer_num, word, "of beer on")
    print(beer_num, word, "of beer.")
    print("Take one down.")
    print("Pass it around.")

if beer_num == 1:
    print("No more bottles of beer on the wall.") Y si es así, termina la letra de la canción.
else:
    De otra manera..

```

word = "bottles"

```

for beer_num in range(99, 0, -1):
    print(beer_num, word, "of beer on the wall.")
    print(beer_num, word, "of beer.")
    print("Take one down.")
    print("Pass it around.")

if beer_num == 1:
    print("No more bottles of beer on the wall.") Y si es así, termina la letra de la canción.
else:
    De otra manera..

```

Asigne el valor "botellas" (una cadena) a una nueva variable llamada "palabra".

Bucle un número especificado de veces, de 99 a ninguno. Utilice "beer\_num" como variable de iteración de bucle.

Las cuatro llamadas a la función de impresión muestran las letras de la canción de la iteración actual, "99 botellas de cerveza en la pared. 99 botellas de cerveza. Tome una abajo. Pasalo alrededor.", Y así sucesivamente con cada iteración.

Compruebe si estamos en la última cerveza pasada ...

Y si es así, termina la letra de la canción.

De otra manera..

```

new_num = beer_num - 1
Recuerde el número de la siguiente cerveza en otra
variable llamada "new_num".

if new_num == 1:
    Si estamos a punto de beber nuestra última cerveza ...
        word = "bottle"
        Cambie el valor de la variable "word" para que las
        últimas líneas de la letra tengan sentido.

    print(new_num, word, "of beer on the wall.")
    Completa las letras de las canciones de esta
    iteración.

print()                                Al final de esta iteración, imprima una línea en blanco.
                                         Cuando todas las iteraciones estén completas, termine el
                                         programa.

```

## 2 list data

### Trabajar con datos ordenados

**Todos los programas procesan datos y los programas Python no son una excepción.**

De hecho, eche un vistazo: los datos están en todas partes. Una gran parte, si no la mayoría, de la programación se trata de datos: la adquisición de datos, el procesamiento de datos, la comprensión de los datos. Para trabajar con datos de forma eficaz, necesita un lugar para colocar sus datos al procesarlo. Python brilla en este sentido, gracias (en gran parte) a su inclusión de un puñado de estructuras de datos ampliamente aplicables: listas, diccionarios, tuplas y conjuntos. En este capítulo, obtendremos una vista previa de las cuatro, antes de dedicar la mayor parte de este capítulo a profundizar en las listas (y profundizaremos en las otras tres en el próximo capítulo). Estamos cubriendo estas estructuras de datos temprano, ya que la mayoría de lo que probablemente harás con Python girará en torno al trabajo con datos.

#### Numbers, Strings...and Objects

Trabajar con un solo valor de datos en Python funciona igual que lo esperaba. Asigne un valor a una variable, y ya está todo configurado. Con la ayuda de la shell, veamos algunos ejemplos para recordar lo que aprendimos en el capítulo anterior.

#### Numbers

Supongamos que este ejemplo ya ha importado el módulo aleatorio random. A continuación, llamamos a la función random.randint para generar un número aleatorio entre

1 y 60, que se asigna a la variable wait\_time. Como el número generado es un entero, ese es el tipo wait\_time en esta instancia:

```
>>> wait_time = random.randint(1, 60)
>>> wait_time
26
```

Tenga en cuenta que no tenía que decirle al intérprete que wait\_time va a contener un entero. Se asignó un número entero a la variable, y el intérprete se encargó de los detalles (nota: no todos los lenguajes de programación funcionan de esta manera).

## Strings

Si asigna una cadena a una variable, sucede lo mismo: el intérprete se encarga de los detalles. Una vez más, no necesitamos declarar anticipadamente que la variable de word en este ejemplo va a contener una cadena o **string**:

```
>>> word = "bottles"
>>> word
'bottles'
```

## Objects

En Python todo es un objeto. Sabemos que los números, cadenas, funciones, módulos-todo es un objeto. Una consecuencia directa de esto es que todos los objetos pueden asignarse a variables. Esto tiene algunas ramificaciones interesantes, que vamos a empezar a aprender en la siguiente página.

### “Ever ything Is an Object” - "Siempre es un objeto"

Cualquier objeto se puede asignar dinámicamente a cualquier variable en Python. ¿Qué es un objeto en Python? La respuesta: todo es un objeto.

Todos los valores de datos en Python son objetos, aunque -a la vista de las cosas- "Don't panic!" Es una cadena y 42 es un número. Para los programadores de Python, "Do not panic!" Es un objeto de cadena y 42 es un objeto numérico. Al igual que en otros lenguajes de programación, los objetos pueden tener estado (atributos o valores) y comportamiento (métodos).

### Tipo de.

Ciertamente, puedes programar Python de una manera orientada a objetos usando clases, objetos, instancias, etc. (pero sobre todo esto más adelante en este libro), pero no tienes que

hacerlo. Recordemos los programas del último capítulo ... ninguno de ellos necesitaba clases. Esos programas sólo contenían código, y funcionaban bien.

A diferencia de otros lenguajes de programación (especialmente Java), no es necesario comenzar con una clase al crear código en Python: simplemente escriba el código que necesita.

Ahora, habiendo dicho todo eso (y sólo para mantenerte en tus dedos), todo en Python se comporta como si fuera un objeto derivado de alguna clase. De esta manera, se puede pensar en Python como algo más basado en objetos en oposición a puramente orientado a objetos, lo que significa que la programación orientada a objetos es opcional en Python.

## **Conozca las Cuatro Estructuras de Datos Incorporadas**

Python viene con cuatro estructuras de datos integradas que puede utilizar para contener cualquier colección de objetos, y son lista, tupla, diccionario y conjunto.

Tenga en cuenta que por "incorporado" queremos decir que las listas, tuplas, diccionarios y conjuntos están siempre disponibles para su código y no necesitan ser importados antes de su uso: cada una de estas estructuras de datos es parte del lenguaje.

Si piensas que tienes una idea bastante buena de lo que es una lista, piensa de nuevo. La lista de Python es más parecida a lo que se podría pensar como una matriz, a diferencia de una lista vinculada, que es lo que a menudo viene a la mente cuando los programadores oyen la palabra "lista". (Si tienes la suerte de no saber qué Una lista vinculada es, sentarse y estar agradecido).

### **1. List: an ordered mutable collection of objects - Lista: una colección mutable ordenada de objetos**

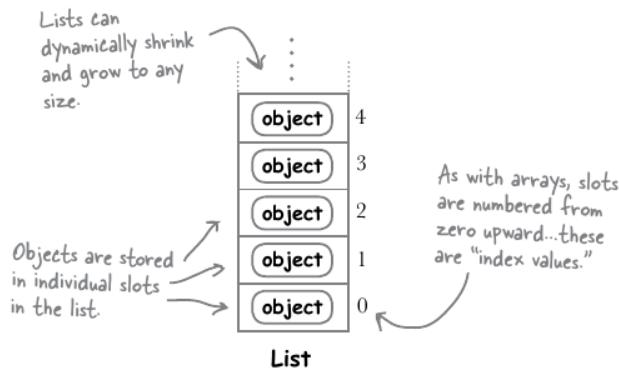
Una lista en Python es muy similar a la noción de una matriz en otros lenguajes de programación, en el que se puede pensar en una lista como una colección indexada de objetos relacionados, con cada ranura o slot en la lista numerada desde cero hacia arriba.

Sin embargo, a diferencia de las matrices de muchos otros lenguajes de programación, las listas son dinámicas en Python, ya que pueden crecer (y reducirse) a la demanda. No hay necesidad de predeclarar el tamaño de una lista antes de usarlo para almacenar cualquier objeto.

Las listas son también heterogéneas, ya que no es necesario predeclarar el tipo de objeto que se está almacenando, puede mezclar objetos de diferentes tipos en la lista si lo desea.

Las listas son **mutables**, ya que puede cambiar una lista en cualquier momento añadiendo, eliminando o cambiando objetos.

Una lista es como una matriz; los objetos que almacena se ordenan secuencialmente en ranuras.



## Ordered Collections Are Mutable/Immutable

### Las colecciones ordenadas son mutables / inmutables

La lista de Python es un ejemplo de una estructura de datos mutable, ya que puede cambiar (o mutar) en tiempo de ejecución. Puede crecer y reducir una lista agregando y eliminando objetos según sea necesario. También es posible cambiar cualquier objeto almacenado en cualquier ranura. Tendremos mucho más que decir sobre las listas en pocas páginas, ya que el resto de este capítulo está dedicado a proporcionar una introducción completa al uso de listas.

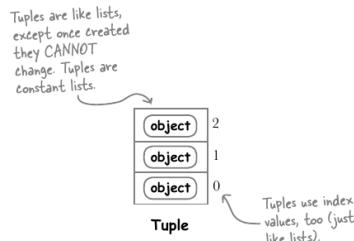
## 2. Tuple: an ordered immutable collection of objects - Tupla: una colección inmutable ordenada de objetos

Una tupla es una lista inmutable. Esto significa que una vez que asigna objetos a una tupla, la tupla no se puede cambiar bajo ninguna circunstancia.

A menudo es útil pensar en una tupla como una lista constante.

La mayoría de los nuevos programadores de Python se rascan la cabeza con desconcierto cuando encuentran tuplas por primera vez, ya que puede ser difícil determinar su propósito. Después de todo, ¿de qué sirve una lista que no puede cambiar? Resulta que hay un montón de casos de uso en los que usted querrá asegurarse de que sus objetos no pueden ser cambiados por su (o cualquier otra persona) del código. Volveremos a las tuplas en el próximo capítulo (así como más

adelante en este libro) cuando hablamos de ellas en un poco más de detalle, así como las usamos.



Las listas y las tuplas son grandes cuando se desea presentar datos de forma ordenada (como una lista de destinos en un itinerario de viaje, donde el orden de destinos es importante). Pero a veces el orden en el que se presentan los datos no es importante. Por ejemplo, es posible que desee almacenar algunos detalles del usuario (como su ID y contraseña), pero es posible que no le importe en qué orden están almacenados (sólo que son). Con datos como este, se necesita una alternativa a la lista / tupla de Python.

## An Unordered Data Structure: Dictionary - Una estructura de datos no ordenada: Diccionario

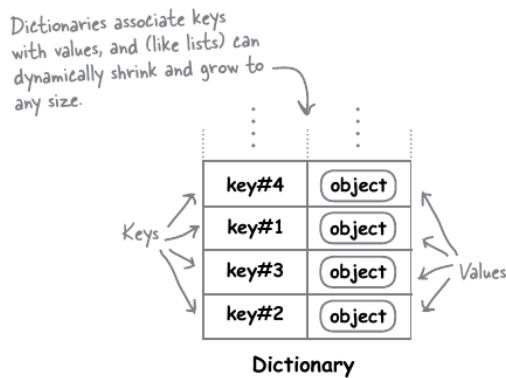
Si mantener tus datos en un orden específico no es importante para ti, pero la estructura es, Python viene con una selección de dos estructuras de datos desordenadas: dictionary y set. Veamos cada uno a su vez, comenzando con el diccionario de Python.

### 3. Dictionary: an unordered set of key/value pairs - Diccionario: un conjunto desordenado de pares clave / valor

Dependiendo de su fondo de programación, es posible que ya sepa lo que es un diccionario, pero es posible que lo conozca por otro nombre, como matriz asociativa, mapa, tabla de símbolos o hash.

Al igual que las otras estructuras de datos en esos otros idiomas, el diccionario de Python le permite almacenar una colección de pares clave / valor. Cada clave única tiene un valor asociado con él en el diccionario, y los diccionarios pueden tener cualquier número de pares. Los valores asociados a una clave pueden ser cualquier objeto.

Los diccionarios son desordenados y mutables. Puede ser útil pensar en el diccionario de Python como una estructura de dos columnas y varias filas. Al igual que las listas, los diccionarios pueden crecer (y reducir) a la demanda.



Algo a tener en cuenta cuando se utiliza un diccionario es que no se puede confiar en el orden interno utilizado por el intérprete. Específicamente, el orden en el que agrega pares clave / valor a un diccionario no es mantenido por el intérprete y no tiene significado (a Python). Esto puede entorpecer a los programadores cuando lo encuentren por primera vez, por lo que lo estamos concienciando ahora para que cuando lo encontremos nuevamente -y en detalle- en el próximo capítulo, obtenga menos sorpresa. Tenga la seguridad: es posible mostrar los datos de su diccionario en un orden específico si es necesario, y también le mostraremos cómo hacerlo en el próximo capítulo.

## A Data Structure That Avoids Duplicates: Set

### Una estructura de datos que evita duplicados: Set

La estructura de datos integrada final es el conjunto, que es genial tener a mano cuando desea eliminar duplicados rápidamente de cualquier otra colección. Y no te preocupes si la mención de conjuntos te ha recordando clase de matemáticas de la escuela secundaria y rompiendo en un sudor frío. La implementación de conjuntos de Python puede utilizarse en muchos lugares.

#### 4. Set: an unordered set of unique objects - Conjunto: un conjunto no ordenado de objetos únicos

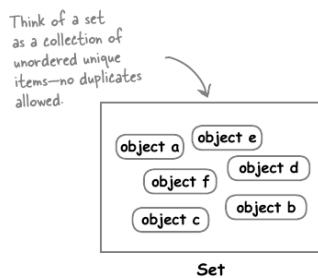
En Python, un conjunto es una práctica estructura de datos para recordar una colección de objetos relacionados mientras se asegura que ninguno de los objetos se dupliquen.

El hecho de que los conjuntos le permiten realizar uniones, intersecciones y diferencias es un bono adicional (especialmente si usted es un tipo de matemáticas que ama la teoría de conjuntos).

Los conjuntos, como listas y diccionarios, pueden crecer (y encogerse) según sea necesario. Al igual que los diccionarios, los conjuntos son desordenados, por lo que no se pueden hacer suposiciones sobre el orden de los objetos en su conjunto. Al

igual que con las tuplas y los diccionarios, podrás ver los conjuntos en acción en el próximo capítulo.

### "Un conjunto no permite objetos duplicados".



## The 80/20 data structure rule of thumb

### La regla general de la estructura de datos 80/20

Las cuatro estructuras de datos incorporadas son útiles, pero no cubren todas las necesidades de datos posibles. Sin embargo, cubren muchos de ellos. Es la historia habitual con tecnologías diseñadas para ser útiles en general: alrededor del 80% de lo que usted necesita hacer está cubierto, mientras que el otro, altamente específico, el 20% requiere que usted haga más trabajo. Más adelante en este libro, aprenderá cómo extender Python para soportar cualquier requisito de datos a medida que tenga. Sin embargo, por ahora, en el resto de este capítulo y el siguiente, vamos a concentrarnos en el 80% de sus necesidades de datos.

El resto de este capítulo está dedicado a explorar cómo trabajar con la primera de nuestras cuatro estructuras de datos integradas: la lista. En el próximo capítulo conoceremos las tres estructuras de datos restantes, diccionario, conjunto y tupla.

## A List Is an Ordered Collection of Objects

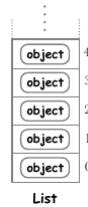
### Una lista es una colección ordenada de objetos

Cuando tienes un montón de objetos relacionados y tienes que ponerlos en alguna parte de tu código, piensa en la lista. Por ejemplo, imagine que tiene un mes de lectura de temperatura diaria; Almacenar estas lecturas en una lista tiene perfecto sentido.

Mientras que las matrices o arrays tienden a ser asuntos homogéneos en otros lenguajes de programación, en que se puede tener una matriz de enteros, o una matriz de cadenas, o una matriz de lecturas de temperatura, la lista de Python es menos restrictiva. Puede tener una lista de objetos y cada objeto puede ser de un tipo diferente. Además de ser heterogéneas, las listas son dinámicas: pueden crecer y contraerse según sea necesario.

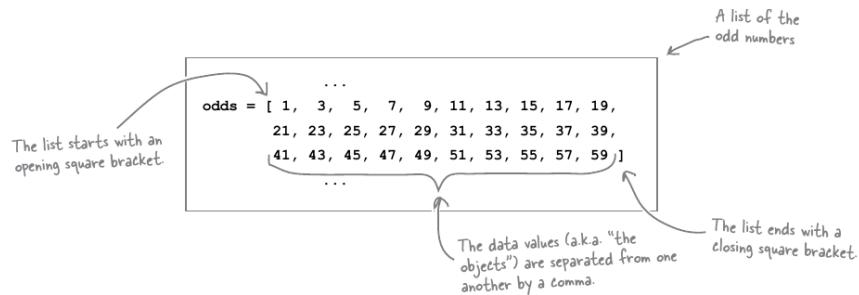
Antes de aprender a trabajar con listas, pasemos un tiempo aprendiendo a detectar listas en código Python.

## How to spot a list in code Cómo detectar una lista en el código



Las listas siempre están entre corchetes, y los objetos contenidos en la lista siempre están separados por una coma.

Recuerde la lista de odds o impares del último capítulo, que contenía los números impares de 0 a 60, como sigue:



Cuando se crea una lista donde los objetos se asignan a una nueva lista directamente en su código (como se muestra arriba), los programadores de Python se refieren a esto como una lista literal, en la que la lista se crea y se rellena de una sola vez.

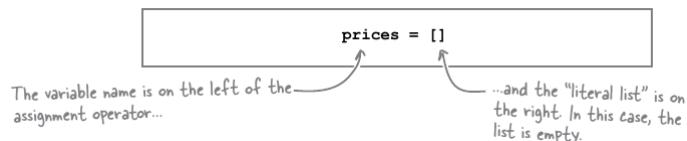
La otra forma de crear y llenar una lista es "hacer crecer" la lista en código, añadiendo objetos a la lista a medida que se ejecuta el código. Veremos un ejemplo de este método más adelante en este capítulo.

Las listas pueden crearse literalmente o "crecer" en el código.

Veamos algunos ejemplos de listas literales.

## Creating Lists Literally - Creación de listas literalmente

Nuestro primer ejemplo crea una lista vacía asignando [ ] a una variable llamada prices:



Aquí hay una lista de temperaturas en grados Fahrenheit, que es una lista de flotadores:

```
temps = [ 32.0, 212.0, 0.0, 81.6, 100.0, 45.3 ]
```

Los objetos (en este caso, algunos flotadores) están separados por comas y rodeados de corchetes, es una lista.

¿Qué tal una lista de las palabras más famosas en la programación de computadoras? Aquí están:

```
words = [ 'hello', 'world' ]
```

A list of  
string objects

Aquí hay una lista de detalles del coche. Observe cómo está bien almacenar datos de tipos mixtos en una lista. Recuerde que una lista es "una colección de objetos relacionados". Las dos cadenas, un flotante y un entero en este ejemplo son todos los objetos Python, por lo que pueden almacenarse en una lista si es necesario:

```
car_details = [ 'Toyota', 'RAV4', 2.2, 60807 ]
```

A list of  
objects  
of  
differing  
type

Nuestros dos ejemplos finales de listas literales explotan el hecho de que -como en el último ejemplo- todo es un objeto en Python. Al igual que las cadenas, los flotantes y los números enteros, las listas también son objetos. A continuación, se muestra un ejemplo de una lista de objetos de lista:

```
everything = [ prices, temps, words, car_details ]
```

And here's an example of a literal list of literal lists:

Lists inside  
of a list

```
odds_and_ends = [ [ 1, 2, 3], ['a', 'b', 'c' ],  
                  [ 'One', 'Two', 'Three' ] ]
```

Don't worry if  
these last two  
examples are  
freaking you  
out. We won't  
be working with  
anything as complex  
as this until a later  
chapter.

No te preocupes si estos dos últimos ejemplos te están volviendo loco. No trabajaremos con nada tan complejo como este hasta un capítulo posterior.

## **Putting Lists to Work - Poner listas a trabajar**

Las listas literales en la última página demuestran cuán rápidamente se pueden crear y llenar listas en el código. Escriba los datos y estará apagado y funcionando.

En una página o dos, cubriremos el mecanismo que le permite crecer (o reducir) una lista mientras se ejecuta el programa. Después de todo, hay muchas situaciones en las que no se sabe de antemano qué datos necesita almacenar ni cuántos objetos va a necesitar. En este caso, su código tiene que crecer (o "generar") la lista según sea necesario. Aprenderá cómo hacerlo en pocas páginas.

Por ahora, imagine que tiene un requisito para determinar si una palabra dada contiene alguna de las vocales (es decir, las letras a, e, i, o, u). ¿Podemos usar la lista de Python para ayudar a codificar una solución a este problema? Vamos a ver si podemos llegar a una solución mediante la experimentación en la shell.

### **Working with lists**

Utilizaremos el shell para definir primero una lista llamada `vocales`, luego veremos si cada letra de una palabra está en la lista de vocales. Vamos a definir una lista de vocales:

```
>>> vowels = ['a', 'e', 'i', 'o', 'u']
```

Una lista de las cinco vocales.

Con las vocales definidas, ahora necesitamos una palabra para comprobar, así que vamos a crear una variable llamada `word` y establecerla en "Milliways":

Here's a word → >>> word = "Milliways"  
to check.

### **¿Hay un objeto dentro de otro? Compruebe con "in"**

Si recuerda los programas del Capítulo 1, recordará que utilizamos el operador `in` de Python para verificar la pertenencia cuando necesitábamos preguntar si un objeto estaba dentro de otro. Podemos aprovechar de nuevo aquí:

```

>>> for letter in word: ← Take each letter in the word...
    if letter in vowels: ← ...and if it is in the "vowels" list...
        print(letter)   ← ...display the letter on screen.

i
i
a ← The output from this code confirms the identity
      of the vowels in the word "Milliways".

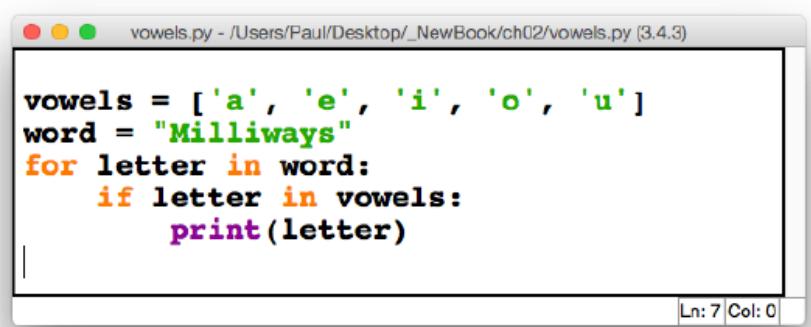
```

Sólo estamos usando las letras aeiou como vocales, aunque la letra y es considerada como una vocal y una consonante.

### Utilicemos este código como base para trabajar con listas.

Cuando haya copiado su código y haya guardado su archivo, su ventana de edición IDLE debería tener este aspecto:

Your list example code  
saved as "vowels.py" inside  
an IDLE edit window.

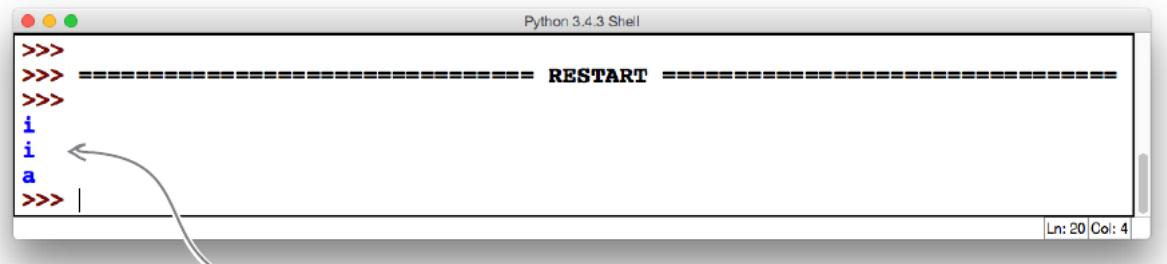


```

vowels = ['a', 'e', 'i', 'o', 'u']
word = "Milliways"
for letter in word:
    if letter in vowels:
        print(letter)

```

No olvide: presione F5 para ejecutar su programa



```

Python 3.4.3 Shell
>>>
>>> ===== RESTART =====
>>>
i
i
a
>>> |

```

### “Growing” a List at Runtime - “Creciendo” una lista en tiempo de ejecución

Nuestro programa actual muestra cada vocal encontrada en la pantalla, incluyendo cualquier duplicado encontrado. Con el fin de enumerar cada vocal única encontrada (y evitar la visualización de duplicados), tenemos que recordar las vocales únicas que

encontramos, antes de mostrarlos en la pantalla. Para ello, necesitamos utilizar una segunda estructura de datos.

No podemos usar la lista de vocales existente porque existe para que podamos determinar rápidamente si la letra que estamos procesando actualmente es una vocal. Necesitamos una segunda lista que empiece vacía, ya que vamos a llenarla en tiempo de ejecución con cualquier vocal que encontremos.

Como hicimos en el último capítulo, vamos a experimentar en el shell antes de realizar cambios en nuestro código de programa. Para crear una lista nueva y vacía, decida un nombre de variable nuevo y, a continuación, asigne una lista vacía. Llámemos a nuestra segunda lista **found**. Aquí asignamos una lista vacía [] a **found**, luego usamos la función integrada de Python **len** para verificar cuántos objetos hay en una colección:

```
>>> found = [] ← An empty list...
>>> len(found) ← ...which the interpreter (thanks
0          to "len") confirms has no objects.
```

La función incorporada "len" informa sobre el tamaño de un objeto.

Las listas vienen con una colección de métodos integrados que puede utilizar para manipular los objetos de la lista. Para invocar un método, utilice la sintaxis de notación de puntos: posfija el nombre de la lista con un punto y la invocación del método. Encontraremos más métodos más adelante en este capítulo. Por ahora, vamos a usar el método **append** para agregar un objeto al final de la lista vacía que acabamos de crear:

```
>>> found.append('a') ← Add to an existing list at runtime
using the "append" method.
>>> len(found) ← The length of the list has now increased.
1
>>> found ← Asking the shell to display the contents of the list
['a']           confirms the object is now part of the list.
```

Las llamadas repetidas al método **append** añaden más objetos al final de la lista:

```

>>> found.append('e')
>>> found.append('i')
>>> found.append('o')
>>> len(found)
4
>>> found
['a', 'e', 'i', 'o']

```

Más adiciones de tiempo de ejecución

Una vez más, utilizamos el shell para confirmar que todo está en orden.

Veamos ahora lo que implica comprobar si una lista contiene un objeto.

## Comprobación de la calidad de miembro con "in"

Ya sabemos cómo hacerlo. Recuerde el ejemplo de "Millyways" de unas pocas páginas, así como el código odds.py del capítulo anterior, que comprobó si un valor minuto calculado estaba en la lista de impares:

The "in" operator checks for membership.

```

...
if right_this_minute in odds:
    print("This minute seems a little odd.")
...

```

## ¿Está el objeto "in" o "not in"?

Además de usar el operador **in** para comprobar si un objeto está contenido dentro de una colección, también es posible comprobar si un objeto no existe dentro de una colección usando la combinación **not in** el operador.

El uso de **not in** permite agregar a una lista existente sólo cuando se sabe que el objeto que se agrega ya no forma parte de la lista:

```

>>> if 'u' not in found:
        found.append('u')

>>> found
['a', 'e', 'i', 'o', 'u']

>>>
>>> if 'u' not in found:
        found.append('u')

>>> found
['a', 'e', 'i', 'o', 'u']

```

Esta primera invocación de las trabajos "append", ya que "u" no existe actualmente dentro de la lista "found" (como se vio en la página anterior, la lista contenía ['a', 'e', 'i', 'o']).

Esta próxima invocación de "append" no se ejecuta, ya que "u" ya existe en "found" por lo que no es necesario volver a agregar.

**Buena atrapada.** Un conjunto podría ser mejor aquí. Pero, vamos a detener el uso de un conjunto hasta el próximo capítulo. Volveremos a este ejemplo cuando lo hagamos. Por ahora, concéntrese en aprender cómo se puede generar una lista en tiempo de ejecución con el método `append`.

## Es hora de actualizar nuestro código

Ahora que sabemos que `not in` y `append`, podemos cambiar nuestro código con cierta confianza. Aquí está el código original de `vowels.py` otra vez:

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = "Milliways"
for letter in word:
    if letter in vowels:
        print(letter)
```

Este código muestra las vocales en "word" tal como se encuentran.

Guardar una copia de este código como `vowels2.py` para que podamos hacer nuestros cambios a esta nueva versión, dejando el código original intacto.

Necesitamos agregar en la creación de una lista encontrada vacía. Entonces necesitamos un código extra para poblar encontrado en tiempo de ejecución. Como ya no mostramos las vocales encontradas cuando las encontramos, se requiere otro bucle `for` para procesar las letras encontradas, y este segundo bucle para ejecutar después del primer bucle (observe como la sangría de ambos bucles está alineada abajo). El nuevo código que necesita está resaltado:

```
This is
"vowels2.py".
Start with
an empty list.

vowels = ['a', 'e', 'i', 'o', 'u']
word = "Milliways"
found = []
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
for vowel in found:
    print(vowel)

Include the code that
decides whether to
update the list of
found vowels.

When this first "for" loop terminates, this
second one gets to run, and it displays the
vowels found in "word".
```

Vamos a hacer un ajuste final a este código para cambiar la línea que establece la palabra a "Milliways" para ser más genérico y más interactivo.

Cambiar la línea de código que lee:

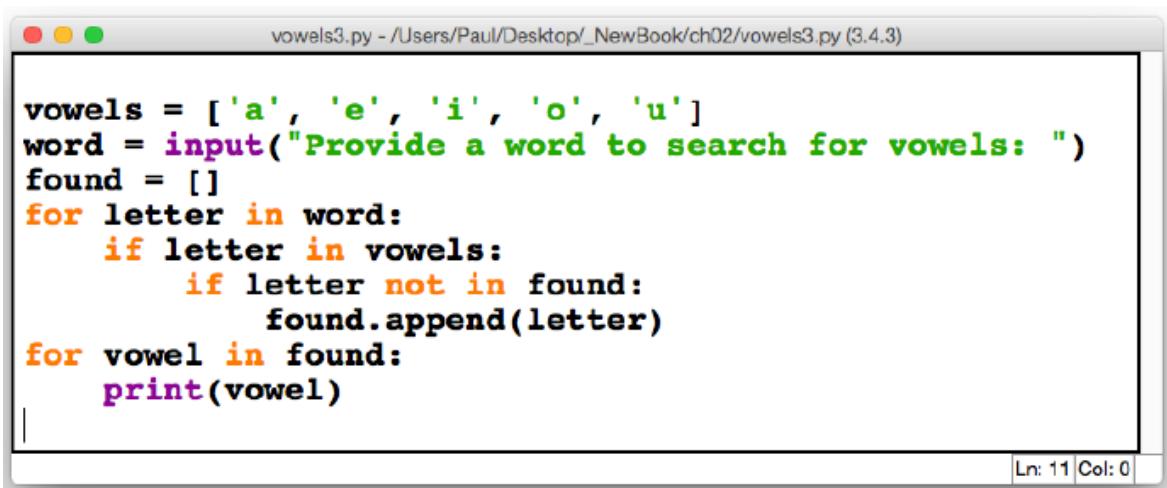
```
word = "Milliways"
```

a:

```
word = input("Provide a word to search for vowels: ")
```

Instruye al intérprete a pedirle a su usuario una palabra para buscar vocales. La función de entrada **input** es otra pieza de calidad incorporada proporcionada por Python.

Haga el cambio como se sugiere a la izquierda, luego guarde su código actualizado como **vocales 3.py**.

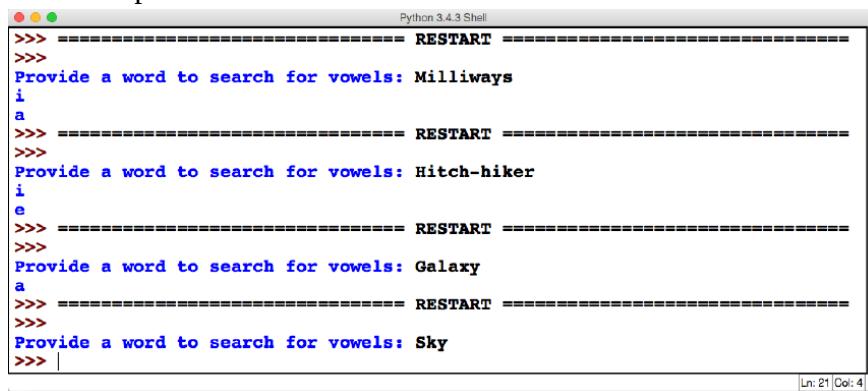


The screenshot shows a Mac OS X-style window titled "vowels3.py - /Users/Paul/Desktop/\_NewBook/ch02/vowels3.py (3.4.3)". The code editor displays the following Python script:

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")
found = []
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
for vowel in found:
    print(vowel)
```

The status bar at the bottom right indicates "Ln: 11 Col: 0".

Y aquí están nuestras pruebas ...



The screenshot shows a terminal window titled "Python 3.4.3 Shell" with the title bar "RESTART". The session logs the following interactions:

```
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: Milliways
i
a
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: Hitch-hiker
i
e
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: Galaxy
a
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: Sky
>>> |
```

The status bar at the bottom right indicates "Ln: 21 Col: 4".

## Manipulando listas

### Eliminación de objetos de una lista

Las listas en Python son como arrays en otros idiomas, y luego algunos.

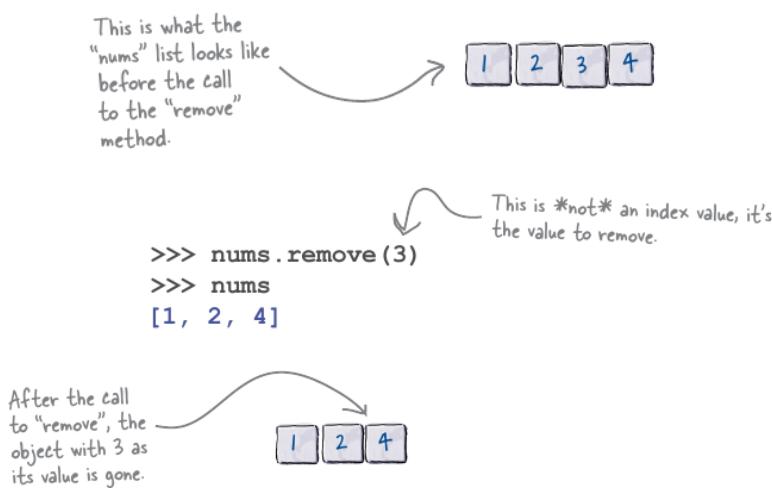
El hecho de que las listas puedan crecer dinámicamente cuando se necesita más espacio (gracias al método **append**) es una enorme ventaja de productividad. Como muchas otras cosas en Python, el intérprete se encarga de los detalles para usted. Si la lista necesita más memoria, el intérprete asigna dinámicamente tanta memoria como sea necesario. Del mismo modo, cuando una lista se reduce, el intérprete recupera dinámicamente la memoria que ya no necesita la lista.

Existen otros métodos para ayudarle a manipular listas. En las siguientes cuatro páginas presentamos cuatro de los métodos más útiles: remove, pop, extend e inserta:

#### 1. Remove: toma el valor de un objeto como su único argumento

El método remove elimina la primera aparición de un valor de datos especificado de una lista. Si el valor de datos se encuentra en la lista, el objeto que lo contiene se elimina de la lista (y la lista se reduce en tamaño por uno). Si el valor de los datos no está en la lista, el intérprete planteará un error (más sobre esto más adelante):

```
>>> nums = [1, 2, 3, 4]
>>> nums
[1, 2, 3, 4]
```



## Popping Objects Off a List - Objetos emergentes de una lista

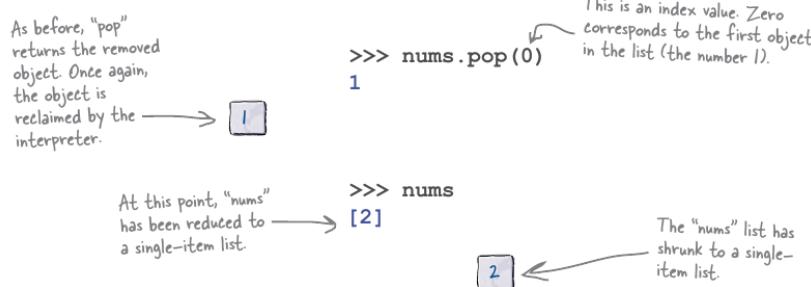
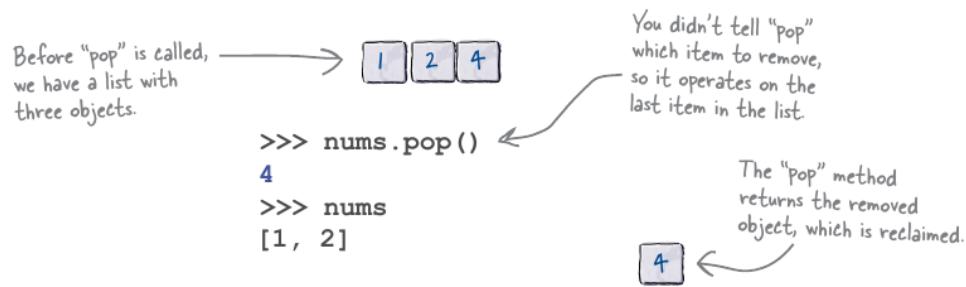
El método `remove` es ideal para cuando se conoce el valor del objeto que desea eliminar. Pero a menudo es el caso que desea eliminar un objeto de una ranura de índice específico.

Para ello, Python proporciona el método `pop`:

### 2. Pop: toma un valor de índice opcional como argumento

El método `pop` elimina y devuelve un objeto de una lista existente basado en el valor de índice del objeto. Si invoca `pop` sin especificar un valor de índice, el último objeto de la lista se elimina y se devuelve. Si especifica un valor de índice, el objeto en esa ubicación se elimina y se devuelve. Si una lista está vacía o invoca `pop` con un valor de índice inexistente, el intérprete genera un error (más sobre esto más adelante).

Los objetos devueltos por `pop` se pueden asignar a una variable si así lo desea, en cuyo caso se mantienen. Sin embargo, si el objeto saltado o popped no está asignado a una variable, su memoria se recupera y el objeto desaparece.



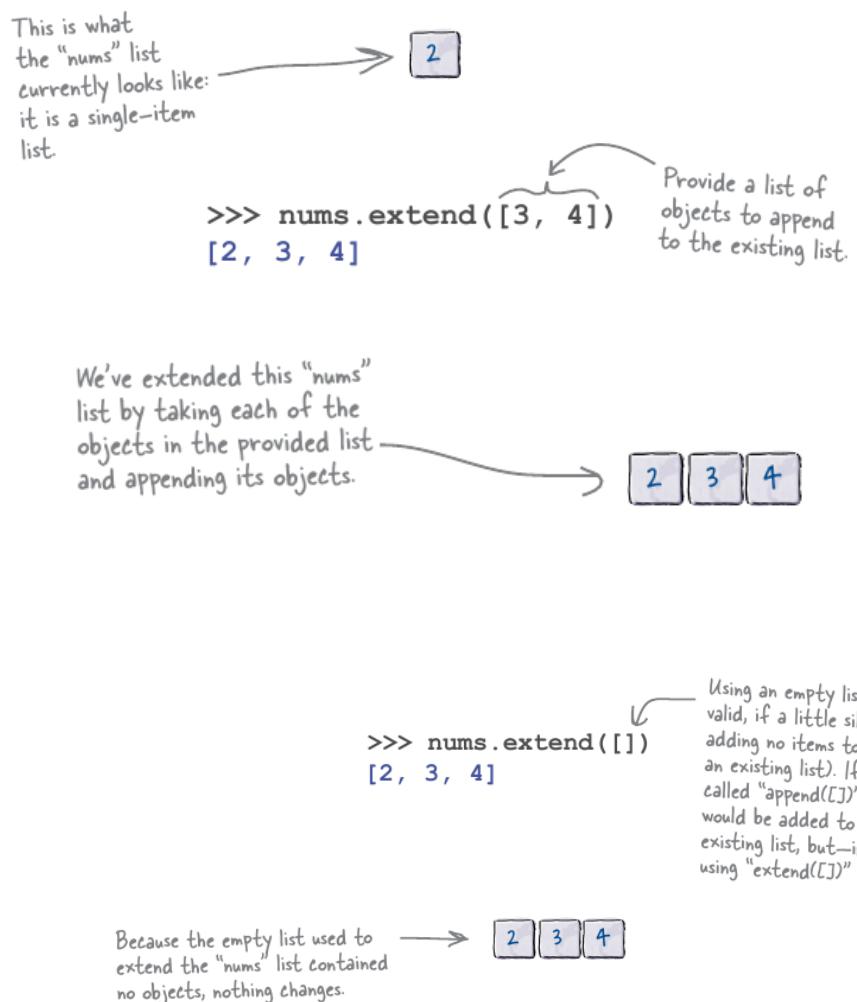
## Extending a List with Objects

### Ampliación de una lista con objetos

Ya sabes que añadir o append puede ser utilizado para agregar un solo objeto a una lista existente. Otros métodos también pueden agregar dinámicamente datos a una lista

#### 3. Extend: toma una lista de objetos como su único argumento

El método extend toma una segunda lista y añade cada uno de sus objetos a una lista existente. Este método es muy útil para combinar dos listas en una:

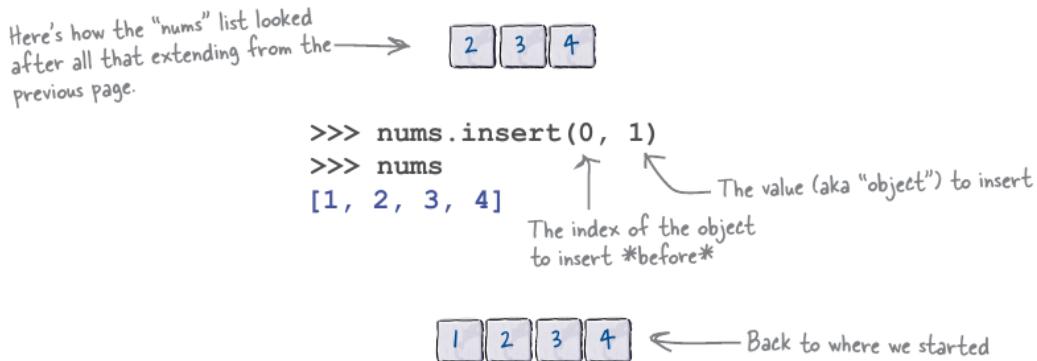


## Inserting an Object into a List - Insertar un objeto en una lista

Los métodos **append** y **extend** obtienen mucho uso, pero están restringidos a añadir objetos al final (el lado derecho) de una lista existente. A veces, deseará agregar al principio (el lado izquierdo) de una lista. Cuando este es el caso, usted querrá utilizar el método de inserción **insert**.

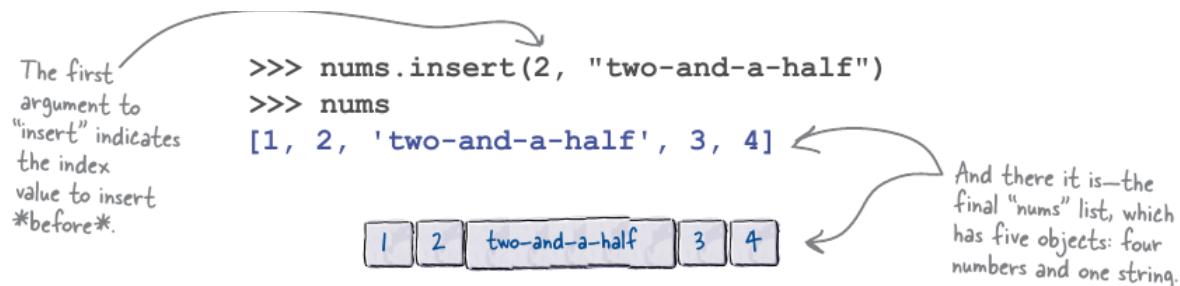
### 4. Insert: toma un valor de índice y un objeto como sus argumentos

El método `insert` inserta un objeto en una lista existente antes de un valor de índice especificado. Esto le permite insertar el objeto al principio de una lista existente o en cualquier lugar dentro de la lista. No es posible insertar al final de la lista, ya que es lo que hace el método `append`:



Después de todo eso de removing, popping, extender e insertar, hemos terminado con la misma lista que comenzamos con unas pocas páginas atrás: [1, 2, 3, 4].

Tenga en cuenta que también es posible usar `insert` para agregar un objeto a cualquier ranura de una lista existente. En el ejemplo anterior, hemos decidido añadir un objeto (el número 1) al principio de la lista, pero podríamos haber usado cualquier número de ranura para insertarlo en la lista. Veamos un ejemplo final, el cual, sólo por diversión, agrega una cadena al medio de la lista `nums`, gracias al uso del valor 2 como el primer argumento para insertar:



## Ahora vamos a ganar algo de experiencia usando estos métodos de lista.

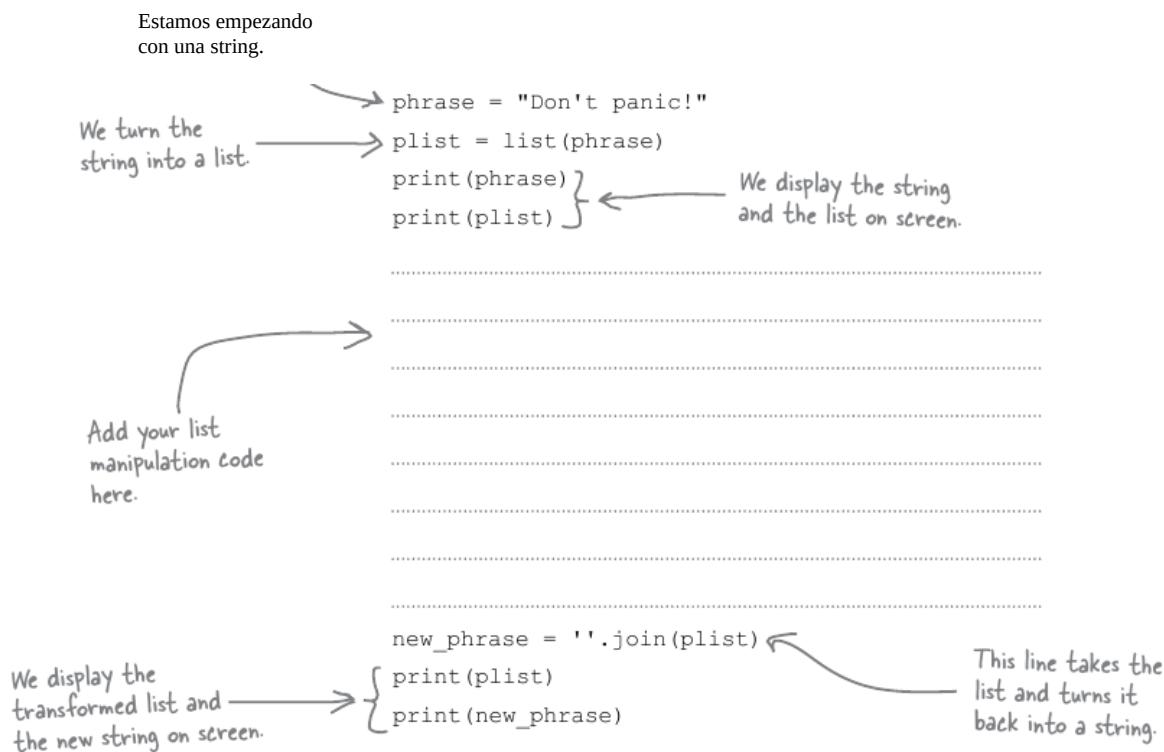
Estudie los mensajes que aparecen en la pantalla. Observe cómo las primeras cuatro líneas de código toman una cadena (en la frase), y la convierten en una lista (en plist), antes de mostrar la frase y plist en la pantalla.

Las otras tres líneas de código tomar plist y transformar de nuevo en una cadena (en new\_phrase) antes de mostrar plist y new\_phrase en la pantalla.

Su desafío es transformar la cadena "¡No se asuste!" En la cadena "on tap" utilizando sólo los métodos de lista mostrados hasta ahora en este libro. (No hay significado oculto en la elección de estas dos strings: es sólo una cuestión de las letras en "on tap" que aparecen en "Don't panic!"). Por el momento, el panic.py muestra "Don't panic! " dos veces.

Su desafío era transformar la cadena "¡No se asuste!" En la cadena "on tap" utilizando sólo los métodos de lista mostrados hasta ahora en este libro. Antes de los cambios, panic.py mostrará "Do not panic!" Dos veces.

Sugerencia: utilice un bucle for cuando realice una operación varias veces.



Su desafío era transformar la cadena "¡No se asuste!" En la cadena "on tap" utilizando sólo los métodos de lista mostrados hasta ahora en este libro. Antes de los cambios, panic.py mostrará "Do not panic!" Dos veces.

La nueva cadena (que muestra "on tap") debe ser almacenada en la variable new\_phrase.

Debes agregar tu código de manipulación de listas aquí. Esto es lo que nos ocurrió, no se preocupe si el suyo es muy diferente al nuestro. Hay más de una forma de realizar las transformaciones necesarias utilizando los métodos de lista.

Deshágase del 'D' al principio de la lista.

Intercambia los dos objetos al final de la lista, haciendo primero popping cada objeto de la lista, luego usando los objetos popped para extender la lista. Esta es una línea de código que tendrá que pensar un poco. Punto clave: los pop ocurren \* primero \* (en el orden mostrado), entonces ocurre la extensión.

```
phrase = "Don't panic!"
plist = list(phrase)
print(phrase)
print(plist)

for i in range(4):
    plist.pop() } This small loop pops the last four objects from "plist". No more "nic!".

> plist.pop(0) Find, then remove, the apostrophe
    plist.remove("'") ← from the list.

    plist.extend([plist.pop(), plist.pop()])
    plist.insert(2, plist.pop(3)) ←

new_phrase = ''.join(plist)
print(plist)
print(new_phrase)

This line of code pops the space from the list, then inserts it back into the list at index location 2. Just like the last line of code, the pop occurs *first*, before the insert happens. And, remember: spaces are characters, too.
```

Esta línea de código despliega el espacio de la lista, luego lo inserta de nuevo en la lista en la ubicación de índice 2. Al igual que la última línea de código, el pop aparece \* primero \*, antes de que se produzca el inserto. Y, recuerde: los espacios también son caracteres.

Busque y elimine el apóstrofo de la lista.

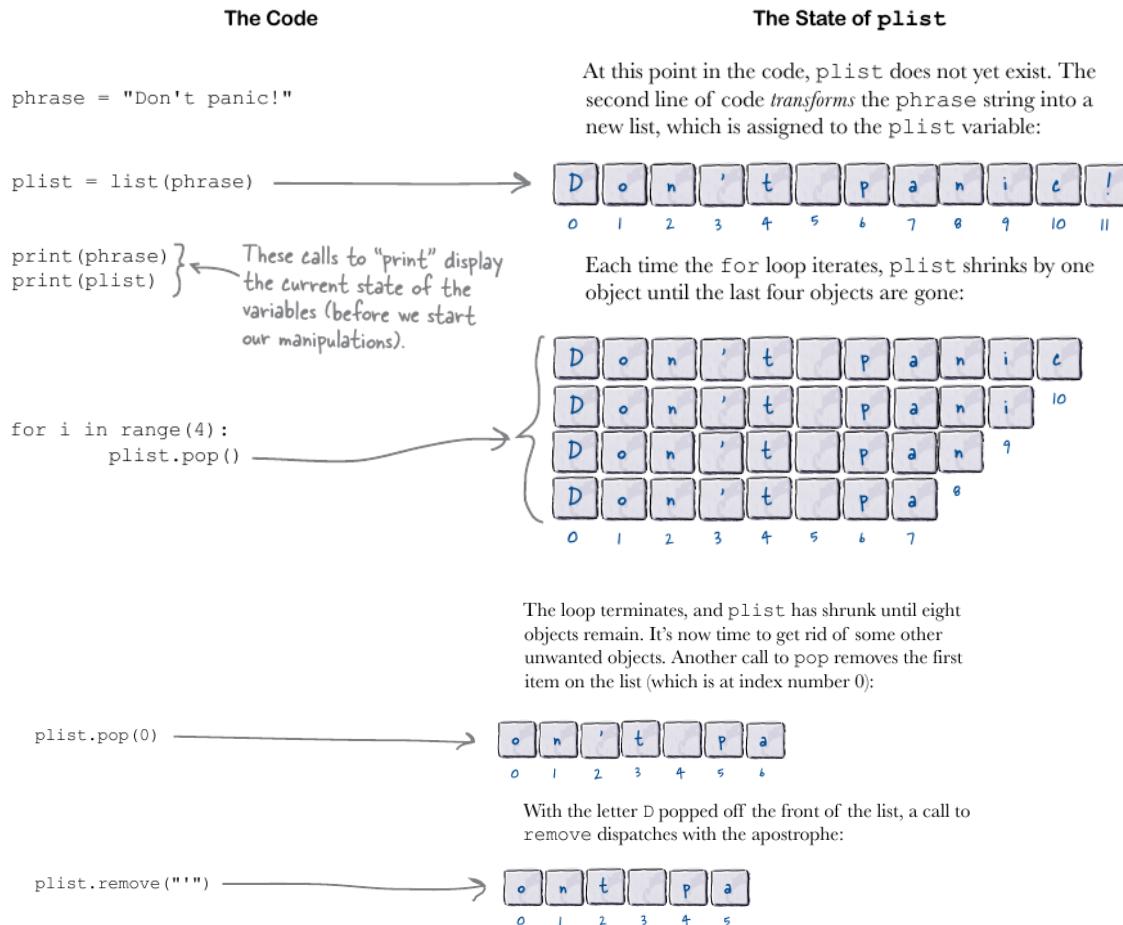
Este pequeño bucle despliega los últimos cuatro objetos de "plist". No más "nic!".

Como hay mucho que hacer en esta solución de ejercicios, las dos páginas siguientes explican este código en detalle.

## ¿Qué pasó con "plist"?

Vamos a hacer una pausa para considerar lo que realmente sucedió a plist como el código en panic.py ejecutado.

A la izquierda de esta página (y la siguiente) está el código de `panic.py`, que, como cualquier otro programa Python, se ejecuta de arriba a abajo. A la derecha de esta página hay una representación visual de `plist` junto con algunas notas sobre lo que está sucediendo. Observe cómo `plist` se contrae dinámicamente y crece a medida que se ejecuta el código:



## Qué pasó con "plist", Continuación

Hemos estado haciendo una pausa por un momento para considerar lo que realmente sucedió a `plist` como el código en `panic.py` ejecutado.

Basado en la ejecución del código de la última página, ahora tenemos una lista de seis ítems con los caracteres `o`, `n`, `t`, espacio, `p`, y `a` disponible para nosotros. Vamos a seguir ejecutando nuestro código:

## The Code

## The State of plist

This is what `plist` looks like as a result of the code on the previous page executing:



`plist.extend([plist.pop(), plist.pop()])`

The next line of code contains **three** method invocations: two calls to `pop` and one to `extend`. The calls to `pop` happen first (from left to right):



The call to `extend` takes the popped objects and adds them to the end of `plist`. It can be useful to think of `extend` as shorthand for multiple calls to the `append` method:



`plist.insert(2, plist.pop(3))`

All that's left to do (to `plist`) is swap the `t` character at location 2 with the space character at index location 3. The next line of code contains **two** method invocations. The first uses `pop` to extract the space character:



Turn "plist" back into a string.

`new_phrase = ''.join(plist)  
print(plist)  
print(new_phrase)`

Then the call to `insert` slots the space character into the correct place (*before* index location 2):



70 Chapter 2

These calls to "print" display the state of the variables (after we've performed our manipulations).

## Lo que parece una copia, pero no es

Cuando se trata de copiar una lista existente a otra, es tentador usar el operador de asignación:

```

>>> first = [1, 2, 3, 4, 5]           Create a new list (and assign
                                         five number objects to it).
>>> first
[1, 2, 3, 4, 5]                         The "first" list's five numbers
>>> second = first                   "Copy" the existing list to a
                                         new one, called "second".
>>> second
[1, 2, 3, 4, 5]                         The "second" list's five numbers

```

Hasta aquí todo bien. Parece que funcionó, ya que los cinco objetos de número desde el principio se han copiado en segundo lugar:



O, ¿verdad? Veamos qué sucede cuando añadimos un nuevo número al segundo, lo que parece una cosa razonable que hacer, pero conduce a un problema:

```

>>> second.append(6)
>>> second
[1, 2, 3, 4, 5, 6]                     This seems OK, but isn't.

```

Una vez más, hasta ahora, tan bueno, pero hay un error aquí. Mira lo que sucede cuando le pedimos a la shell que muestre el contenido de la primera - el nuevo objeto se agrega a la primera también!

```

>>> first
[1, 2, 3, 4, 5, 6]                     Whoops! The new
                                         object is appended to
                                         "first" too.

```



Este es un problema, ya que tanto el primero como el segundo apuntan a los mismos datos. Si cambia una lista, también cambia la otra. Esto no está bien.

## How to Copy a Data Structure

### Cómo copiar una estructura de datos

Si el uso del operador de asignación no es la forma de copiar una lista a otra, ¿cuál es? Lo que pasa es que una **referencia** a la lista se comparte entre la primera y la segunda.



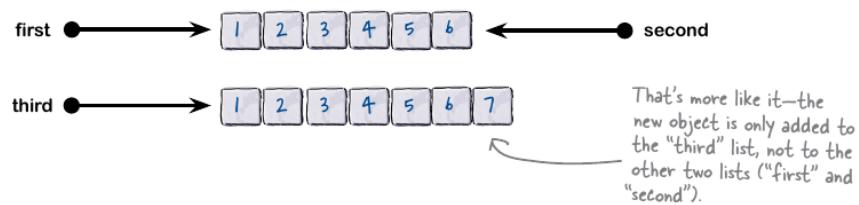
Para resolver este problema, las listas vienen con un método de copia, que hace lo correcto. Echa un vistazo a cómo funciona la copia:

```
>>> third = second.copy()
>>> third
[1, 2, 3, 4, 5, 6]
```



Con el tercero creado (gracias al método de `copy`), vamos a añadir un objeto a él, a continuación, ver lo que sucede:

```
>>> third.append(7)
The "third" list
has grown by
one object.
>>> third
[1, 2, 3, 4, 5, 6, 7]
>>> second
[1, 2, 3, 4, 5, 6] ←
Much better. The existing
list is unchanged.
```



**No utilice el operador de asignación para copiar una lista; Utilice el método de "copy" en su lugar.**

Python soporta la notación de corchete, y luego algunos. Todos los que han utilizado corchetes con una matriz en casi cualquier otro lenguaje de programación sabe que pueden acceder al primer valor en una matriz llamada nombres usando nombres [0]. El valor

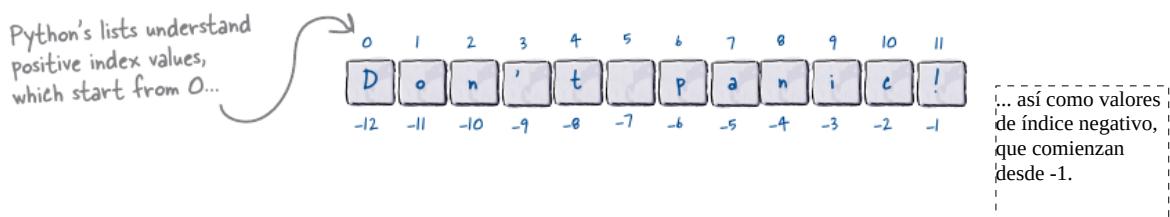
siguiente está en los nombres [1], el siguiente en los nombres [2], y así sucesivamente. Python funciona de esta manera también, cuando se trata de acceder a objetos en cualquier lista. Sin embargo, Python extiende la notación para mejorar este comportamiento estandarizado soportando valores de índice negativos (-1, -2, -3 y así sucesivamente), así como una notación para seleccionar un rango de objetos de una lista.

## Lists Extend the Square Bracket Notation

Extender la lista con la notación de corchetes cuadrados

Toda nuestra charla sobre las listas de Python como arrays en otros lenguajes de programación no era sólo charla ociosa. Al igual que otros lenguajes, Python comienza a contar desde cero cuando se trata de ubicaciones de índice de numeración, y utiliza la conocida notación de corchetes para acceder a los objetos de una lista.

A diferencia de muchos otros lenguajes de programación, Python le permite acceder a la lista relativa a cada extremo: los valores de índice positivo cuentan de izquierda a derecha, mientras que los valores de índice negativo cuentan de derecha a izquierda:



Veamos algunos ejemplos mientras trabajamos en el shell:

```
>>> saying = "Don't panic!"                                Create a list of letters.
>>> letters = list(saying)                                ←
>>> letters
['D', 'o', 'n', "'", 't', ' ', 'p', 'a', 'n', 'i', 'c', '!']
>>> letters[0]                                         ←
'D'
>>> letters[3]                                         {←
'''                                         Using positive index values counts
>>> letters[6]                                         {←
'p'                                         from left to right...
>>> letters[-1]                                         ←
'!'                                         ...whereas negative index values
>>> letters[-3]                                         {←
'i'                                         count right to left.
>>> letters[-6]                                         {←
'p'
```

A medida que las listas crecen y se reducen mientras se ejecuta el código de Python, ser útil para indexar en la lista usando un valor de índice negativo. Por ejemplo, usar -1 como valor de índice siempre está garantizado para devolver el último objeto de la lista sin importar cuán grande sea la lista, así como usar 0 siempre devuelve el primer objeto.

Es fácil obtener los primeros y últimos objetos  
en cualquier lista.

```
>>> first = letters[0]
>>> last = letters[-1]
>>> first
'D'
>>> last
'!'
```

Las extensiones de Python a la notación de corchetes no se detienen con el soporte de valores de índice negativos. Las listas comprenden el **inicio**, la **parada** y el **paso**.

## Lists Understand Start, Stop, and Step

### Listas Entender Inicio, Detener y Paso

Primero nos reunimos con start, stop y step en el capítulo anterior al discutir la versión de tres argumentos de la función range:

```
word = "bottles"
for beer_num in range(99, 0, -1):
    print(beer_num, word, "of beer on")
    print(beer_num, word, "of beer.")
    print("Take one down.")
```

Recuerde lo que significa empezar, detener, y paso cuando se trata de especificar rangos (y vamos a relacionarlos con listas):

- **El valor START le permite controlar WHERE el rango comienza.** Cuando se utiliza con listas, el valor inicial indica el valor del índice inicial.
- **El valor STOP le permite controlar cuándo termina el rango.** Cuando se utiliza con listas, el valor de parada indica el valor del índice para detenerse en, pero no incluir.

- El valor **STEP** le permite controlar **cómo se genera el rango**. Cuando se utiliza con listas, el valor de paso se refiere a la zancada a tomar.

## Puede poner inicio, parada y paso entre corchetes

Cuando se usan con listas, **start**, **stop** y **step** se especifican entre corchetes y se separan entre sí por el carácter de dos puntos (:):

```
letters[start:stop:step]
```

Puede parecer algo contraintuitivo, pero los tres valores son opcionales cuando se usan juntos:

Cuando falta el **start**, tiene un valor predeterminado de 0.

Cuando falta el **stop**, toma el valor máximo permitido para la lista.

Cuando falta **step**, tiene un valor predeterminado de 1.

## List Slices in Action

### Listar rebanadas en acción

Dadas las cartas de lista existentes de unas pocas páginas atrás, puede especificar valores para iniciar, detener y avanzar de varias maneras.

Veamos algunos ejemplos:

```
>>> letters
['D', 'o', 'n', "", "t", ' ', 'p', 'a', 'n', 'i', 'c', '!']

>>> letters[0:10:3]
['D', "", 'p', 'i']

>>> letters[3:10]
["t", ' ', 'p', 'a', 'n', 'i', 'c', '!']

>>> letters[:10]
['D', 'o', 'n', "", "t", ' ', 'p', 'a', 'n', 'i']

>>> letters[::2]
['D', 'n', 't', 'p', 'n', 'c']
```

El uso de la notación de inicio, parada, paso de corte con las listas es muy potente (por no hablar de útil), y se recomienda tomar un tiempo para comprender cómo funcionan estos ejemplos. Asegúrese de seguir a lo largo de su >>> prompt, y no dude en experimentar con esta notación, también.

## Starting and Stopping with Lists

### Inicio y detención con listas

Sigue con los ejemplos de esta página (y la siguiente) en tu solicitud >>> y asegúrate de obtener la misma salida que nosotros.

Empezamos por convertir una cadena en una lista de letras:

```
>>> book = "The Hitchhiker's Guide to the Galaxy"
>>> booklist = list(book)
>>> booklist
['T', 'h', 'e', ' ', 'H', 'i', 't', 'c', 'h', 'i', 'k',
'e', 'r', "'", 's', ' ', 'G', 'u', 'i', 'd', 'e', ' ', 't',
'o', ' ', 't', 'h', 'e', ' ', 'G', 'a', 'l', 'a', 'x', 'y']
```

Turn a string into a list, then display the list.

Note that the original string contained a single quote character. Python is smart enough to spot this, and surrounds the single quote character with double quotes.

Convierta una cadena en una lista y luego muestre la lista.

Tenga en cuenta que la cadena original contenía un carácter de comillas simples. Python es lo suficientemente inteligente para detectar esto, y rodea el carácter de comillas simples con comillas dobles.

La lista recién creada (llamada booklist arriba) se utiliza entonces para seleccionar un rango de letras de dentro de la lista:

```
>>> booklist[0:3] ← Select the first three objects
['T', 'h', 'e']

>>> ''.join(booklist[0:3])
'The' ← Turn the selected range into a string (which
       you learned how to do near the end of the
       "panic.py" code). The second example selects
       the last six objects from the list.

>>> ''.join(booklist[-6:])
'Galaxy'
```

Seleccione los tres primeros objetos (letras) de la lista.

Convierta el rango seleccionado en una secuencia (que aprendió a hacer al final del código "panic.py"). La segunda opción selecciona los últimos seis objetos de la lista.

Asegúrese de tomar tiempo para estudiar esta página (y la siguiente) hasta que esté seguro de que entiende cómo funciona cada ejemplo, y asegúrese de probar cada ejemplo dentro de IDLE.

Con el último ejemplo anterior, tenga en cuenta que el intérprete está dispuesto a utilizar cualquiera de los valores predeterminados para iniciar, detener y paso.

## Stepping with Lists

### Paso a Paso con Listas

Aquí hay dos ejemplos más, que muestran el uso del paso o step con las listas.

El primer ejemplo selecciona todas las letras, comenzando desde el final de la lista (es decir, está seleccionando a la inversa), mientras que el segundo selecciona cada otra letra de la lista. Observe cómo el valor de paso controla este comportamiento:

```
>>> backwards = booklist[::-1]
>>> ''.join(backwards)
"yxalaG eht ot ediuG s'rekihhctiH ehT"
    ^
    Looks like gobbledegook,
    doesn't it? But it is actually
    the original string reversed.

>>> every_other = booklist[::2]
>>> ''.join(every_other)
"TeHthie' ud oteGlx"
    ^
    And this looks like gibberish! But "every_other" is a list made
    up from every second object (letter) starting from the first
    and going to the last. Note: "start" and "stop" are defaulted.
```

Parece un gobbledegook, ¿no? Pero en realidad es la cadena original invertida.

¡Y esto se parece a charabia! Pero "every\_other" es una lista compuesta de cada segundo objeto (letra ") a partir de la primera y va a la última. Nota: "inicio" y "detener" son predeterminados

Dos ejemplos finales confirman que es posible iniciar y detener en cualquier lugar de la lista y seleccionar objetos. Al hacer esto, los datos devueltos se denominan un segmento. Piense en una porción como un fragmento de una lista existente.

Ambos ejemplos seleccionan las letras de la lista de libros booklist que deletrean la palabra 'Hitchhiker'. La primera selección se une para mostrar la palabra 'Hitchhiker', mientras que la segunda muestra 'Hitchhiker' en sentido inverso:

**Una "slice" es un fragmento de una lista.**

```
>>> ''.join(booklist[4:14]) ← Slice out the  
'Hitchhiker' word "Hitchhiker".  
  
>>> ''.join(booklist[13:3:-1])  
'rekihhctiH'  
↑ Slice out the word "Hitchhiker", but  
do it in reverse order (i.e., backward).
```

### Las rebanadas o slices están por todas partes

La notación de corte o slice no sólo funciona con listas. De hecho, encontrarás que puedes dividir cualquier secuencia en Python, accediendo a ella con [start: stop: step].

## Putting Slices to Work on Lists

### Poner los Slices a trabajar en las listas

La notación de corte o slice de Python es una extensión útil para la notación de corchetes, y se usa en muchos lugares a lo largo del lenguaje. Verás muchos usos de las rebanadas o slices mientras sigues trabajando en este libro.

Por ahora, veamos la notación de corchetes de Python (incluyendo el uso de rebanadas) en acción. Vamos a tomar el programa `panic.py` de anterior y refactorizarlo para usar la notación de corchetes y las rodajas o slices para lograr lo que se había logrado anteriormente con los métodos de lista.

Antes de realizar el trabajo real, he aquí un rápido recordatorio de lo que `panic.py` hace.

### Converting “Don’t panic!” to “on tap”

Este código transforma una cadena en otra manipulando una lista existente utilizando los métodos list. Comenzando con la cadena “¡Don’t panic!”, Este código produjo “on tap” después de las manipulaciones:

```
Display the initial state of the string and list.  
Use a collection of list methods to transform and manipulate the list of objects.  
Display the resulting state of the string and list.  
  
phrase = "Don't panic!"  
plist = list(phrase)  
print(phrase)  
print(plist)  
for i in range(4):  
    plist.pop()  
plist.pop(0)  
plist.remove("")  
plist.extend([plist.pop(), plist.pop()])  
plist.insert(2, plist.pop(3))  
new_phrase = ''.join(plist)  
print(plist)  
print(new_phrase)  
  
This is "panic.py".
```

Esta es la salida producida por este programa cuando se ejecuta dentro de IDLE:

```
>>> ===== RESTART =====
>>>
Don't panic!
['D', 'o', 'n', "'", 't', ' ', 'p', 'a', 'n', 'i', 'c', '!']
on tap
>>>
```

The string "Don't panic!" is transformed into "on tap" thanks to the list methods.

## Putting Slices to Work on Lists, Continued

Es hora de que el trabajo real. A continuación, se muestra el código **panic.py**, con el código que debe cambiar resaltado:

```
phrase = "Don't panic!"
plist = list(phrase)
print(phrase)
print(plist)
for i in range(4):
    plist.pop()
    plist.pop(0)
    plist.remove("'")
    plist.extend([plist.pop(), plist.pop()])
    plist.insert(2, plist.pop(3))
    new_phrase = ''.join(plist)
    print(plist)
    print(new_phrase)
```

Para este ejercicio, reemplace el código resaltado arriba con un nuevo código que aproveche la notación de corchetes de Python. Tenga en cuenta que todavía puede utilizar métodos de lista donde tiene sentido. Como antes, estás tratando de transformar "Don't panic!" En "on tap". Agregue su código en el espacio proporcionado y llame a su nuevo programa **panic2.py**:

```
phrase = "Don't panic!"  
plist = list(phrase)  
print(phrase)  
print(plist)
```

```
print(plist)
print(new_phrase)
```

### Solucion:

Para este ejercicio, reemplazar el código resaltado en la página anterior con un nuevo código que aprovecha la notación de corchetes de Python. Tenga en cuenta que todavía puede utilizar métodos de lista donde tiene sentido. Como antes, estás tratando de transformar "Don't panic" En "on tap". Usted debe llamar a su nuevo programa panic2.py:

```
phrase = "Don't panic!"  
plist = list(phrase)  
print(phrase)  
print(plist)
```

```
new_phrase = ''.join(plist[1:3]) ←
```

We started by slicing out the word "on" from "plist"...

```
new_phrase = new_phrase + ''.join([plist[5], plist[4], plist[7], plist[6]])
```

```
print(plist)
print(new_phrase)
```

...then picked out each additional letter that we needed: space, "t", "a", and "p".

Me pregunto cuál de estos dos programas - "panic.py" o "panic2.py" - es mejor?

**Esa es una gran pregunta.**

Algunos programadores verán el código en panic2.py y, al compararlo con el código de panic.py, concluirán que dos líneas de código son siempre mejores que siete, especialmente cuando la salida de ambos programas es la misma. Cuál es una medida fina de "betterness," pero no realmente útil en este caso. Para ver qué queremos decir con esto, echemos un vistazo a la producción producida por ambos programas.

Utilice IDLE para abrir panic.py y panic2.py en ventanas de edición independientes. Seleccione el pánico. Primero, luego presione F5. A continuación, seleccione la ventana panic2.py y, a continuación, presione F5. Compare los resultados de ambos programas en su shell.

```

"panic.py" →
phrase = "Don't panic!"
plist = list(phrase)
print(phrase)
print(plist)

for i in range(4):
    plist.pop()
plist.pop(0)
plist.remove("'")
plist.extend([plist.pop(), plist.pop()])
plist.insert(2, plist.pop(3))

new_phrase = ''.join(plist)
print(plist)
print(new_phrase)

"panic2.py" →
phrase = "Don't panic!"
plist = list(phrase)
print(phrase)
print(plist)

new_phrase = ''.join(plist[1:3])
new_phrase = new_phrase + ''.join([plist[5], plist[4], plist[7], plist[6]])

print(plist)
print(new_phrase)

```

```

Python 3.4.3 Shell
>>> ===== RESTART =====
>>> Don't panic!
['D', 'o', 'n', "'", 't', ' ', 'p', 'a', 'n', 'i', 'c', "'"]
on tap
>>> ===== RESTART =====
>>> Don't panic!
['D', 'o', 'n', "'", 't', ' ', 'p', 'a', 'n', 'i', 'c', "'"]
['D', 'o', 'n', "'", 't', ' ', 'p', 'a', 'n', 'i', 'c', "'"]
on tap
>>>

```

The output produced by running the "panic.py" program → {

The output produced by running the "panic2.py" program → {

Observe cómo son diferentes estas salidas.

## ¿Cuál es mejor? Depende...

Ejecutamos panic.py y panic2.py en IDLE para ayudarnos a determinar cuál de estos dos programas es "mejor".

Eche un vistazo a la segunda línea de resultados de ambos programas:

```

>>>
Don't panic!
['D', 'o', 'n', "'", 't', ' ', 'p', 'a', 'n', 'i', 'c', '!']
['o', 'n', ' ', 't', 'a', 'p']
on tap
>>> ===== RESTART =====
>>>
Don't panic!
['D', 'o', 'n', "'", 't', ' ', 'p', 'a', 'n', 'i', 'c', '!']
['D', 'o', 'n', "'", 't', ' ', 'p', 'a', 'n', 'i', 'c', '!']
on tap
>>>

```

...whereas this output is produced by "panic2.py".

Esta es la salida producida por "panic.py" ...

Aunque ambos programas terminan mostrando la cadena "**on tap**" (habiendo comenzado con la cadena "**Do not panic!**"), **panic2.py** no cambia plist de ninguna manera, mientras que **panic.py** lo hace.

Vale la pena detenerse por un momento para considerar esto.

Recordemos nuestra discusión del principio de este capítulo titulado “¿Qué pasó con ‘plist’?”. Esta discusión detalló los pasos que convirtieron esta lista:



En esta lista mucho más corta:



Todas esas manipulaciones de listas con los métodos `pop`, `remove`, `extend` e `insert` cambiaron la lista, lo cual está bien, ya que eso es lo que los métodos de lista están diseñados para hacer: cambiar la lista. Pero ¿qué pasa con **panic2.py**?

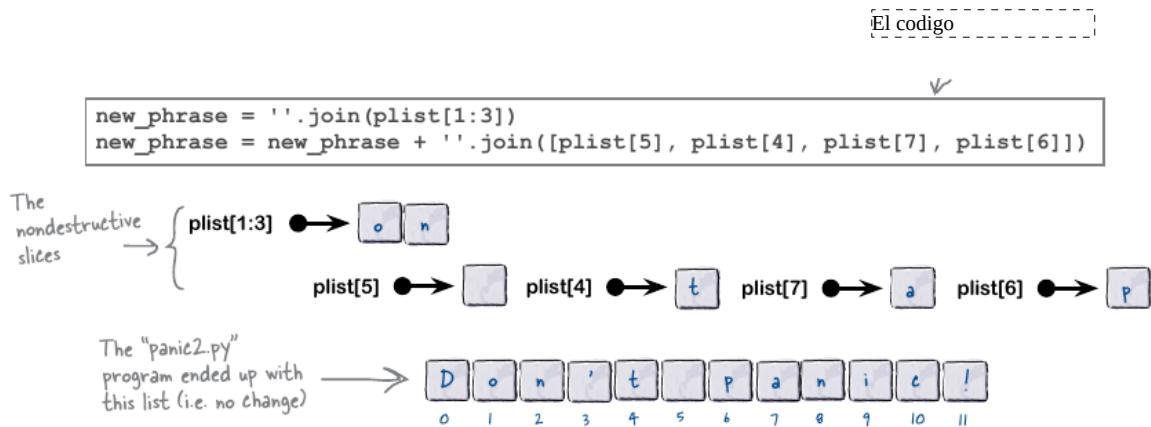
## Slicing a List Is Nondestructive

### Cortar una lista no es destructivo

Los métodos de lista utilizados por el programa `panic.py` para convertir una cadena en otra eran destructivos, ya que el código original alteraba el estado original de la lista. Cortar una lista no es destructiva, ya que extraer objetos de una lista existente no la altera; Los datos originales permanecen intactos.



Las rodajas o slices utilizadas por `panic2.py` se muestran aquí. Tenga en cuenta que cada uno extrae datos de la lista, pero no la cambia. Aquí están las dos líneas de código que hacen todo el trabajo pesado, junto con una representación de los datos de cada fragmento extractos:



### Así que ... ¿cuál es mejor?

El uso de métodos de lista para manipular y transformar una lista existente hace exactamente eso: manipula y transforma la lista. El estado original de la lista ya no está disponible para su programa. Dependiendo de lo que esté haciendo, esto puede (o no) ser un problema. El uso de la notación de corchetes de Python generalmente no altera una lista existente, a menos que decida asignar un nuevo valor a una ubicación de índice existente. El uso de rebanadas también da como resultado ningún cambio en la lista: los datos originales permanecen como estaban.

¿Cuál de estos dos enfoques que usted decide es "mejor" depende de lo que usted está tratando de hacer (y está perfectamente bien que no le guste tampoco). Siempre hay más de

una forma de realizar un cálculo, y las listas de Python son lo suficientemente flexibles como para soportar muchas formas de interactuar con los datos almacenados en ellos.

**Los métodos de lista cambian el estado de una lista, mientras que el uso de corchetes y rebanadas (normalmente) no.**

Estamos casi terminados con nuestra gira inicial de listas. Sólo hay un tema más que presentarle en esta etapa: **iteración de lista**.

## Python's "for" Loop Understands Lists

### Bucle "for" de python para entender listas

El bucle python for conoce todas las listas y, cuando se proporciona con cualquier lista, sabe dónde está el inicio de la lista, cuántos objetos contiene la lista y dónde está el final de la lista. Usted nunca tiene que decirle al bucle for, nada de esto, ya que funciona por sí mismo.

Un ejemplo ilustrativo. Siga adelante abriendo una nueva ventana de edición en IDLE y escribiendo el código que se muestra a continuación. Guarde este nuevo programa como marvin.py, luego presione F5 para tomarlo para un giro:

The screenshot shows the Python IDLE environment. On the left is a code editor window titled "marvin.py". Inside the editor, the following Python code is written:

```
paranoid_android = "Marvin"
letters = list(paranoid_android)
for char in letters:
    print('\t', char)
```

An annotation above the editor says "Execute this small program...". Another annotation below it says "...to produce this output.". To the right of the editor is a terminal window showing the execution of the script:

```
paranoid_android = "Marvin"
letters = list(paranoid_android)
for char in letters:
    print('\t', char)

>>> RESTART: /Users/Paul/Desktop/_NewBook/ch02/marvin.py
    M
    a
    r
    v
    i
    n
```

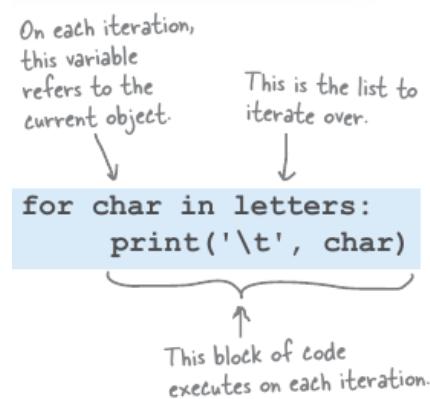
A red curly brace is placed over the characters "M", "a", "r", "v", "i", and "n". A callout bubble points to this brace with the text: "Each character from the 'letters' list is printed on its own line, preceded by a tab character (that's what the '\t' does)."

### Comprender el código de marvin.py

Las primeras dos líneas de marvin.py son familiares: asignar una cadena a una variable (llamada **paranoid\_android**), luego convertir la cadena en una lista de objetos de carácter (asignados a una nueva variable llamada **letters**).

Es la siguiente declaración -el bucle for- en la que queremos que te concentres.

En cada iteración, el bucle for se encarga de tomar cada objeto en la lista de letras y asignar uno a la vez a otra variable, llamada char. Dentro del cuerpo de bucle indentado char toma el valor actual del objeto que está siendo procesado por el bucle for. Tenga en cuenta que el bucle for sabe cuándo comenzar la iteración, cuándo detener la iteración, así como cuántos objetos hay en la lista de letras. Usted no necesita preocuparse de nada de esto: ese es el trabajo del intérprete.



## Python's “for ” Loop Understands Slices

### El bucle "for" de Python entiende las rebanadas

Si utiliza la notación de corchete para seleccionar una rebanada de una lista, el bucle for "hace lo correcto" y sólo itera sobre los objetos cortados. Una actualización de nuestro programa más reciente muestra esto en acción. Guarde una nueva versión de marvin.py como marvin2.py, a continuación, cambie el código para que se vea como se muestra a continuación.

De interés es nuestro uso del operador de **multiplicación de Python (\*)**, que se utiliza para controlar cuántos caracteres de tabulación se imprimen antes de cada objeto en el bucle segundo y tercero. Utilizamos \* aquí para "multiplicar" cuántas veces queremos que aparezca la pestaña:

```

marvin2.py - /Users/Paul/Desktop/_NewBook/ch02/marvin2.py [3.4.3]

paranoid_android = "Marvin, the Paranoid Android"
letters = list(paranoid_android)
for char in letters[:6]:
    print('\t', char)
print()
for char in letters[-7:]:
    print('\t'*2, char)
print()
for char in letters[12:20]:
    print('\t'*3, char)

Ln: 12 Col: 0

```

The first loop iterates over a slice of the first six objects in the list.

The second loop iterates over a slice of the last seven objects in the list. Note how "`*2`" inserts two tab characters before each printed object.

The third (and final) loop iterates over a slice from within the list, selecting the characters that spell the word "Paranoid". Note how "`*3`" inserts three tab characters before each printed object.

```

>>> Marvin
      A
      n
      d
      r
      o
      i
      d
      P
      a
      r
      a
      n
      o
      i
      d
>>> |                                         Ln: 119 Col: 4

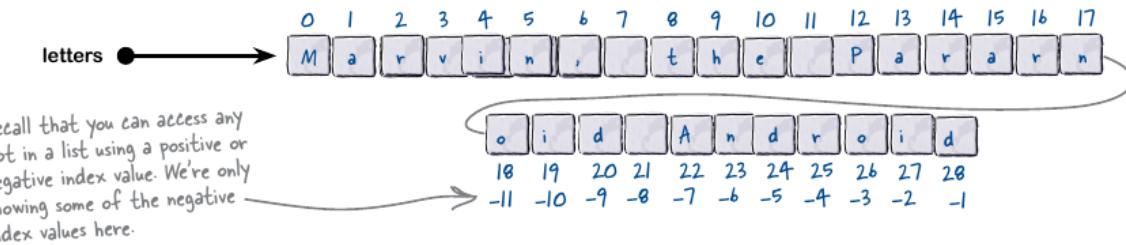
```

## Marvin's Slices in Detail - Marvin's Slices en detalle

Echemos un vistazo a cada uno de los cortes en el último programa en detalle, ya que esta técnica aparece mucho en los programas de Python. A continuación, se presenta una vez más cada línea de código de segmento, junto con una representación gráfica de lo que está pasando.

Antes de mirar las tres rebanadas, tenga en cuenta que el programa comienza asignando una cadena a una variable (llamada `paranoid_android`) y convirtiéndola en una lista (llamada `letters`):

```
paranoid_android = "Marvin, the Paranoid Android"
letters = list(paranoid_android)
```



Recuerde que puede acceder a cualquier ranura de una lista utilizando un valor de índice positivo o negativo. Aquí sólo mostramos algunos de los valores del índice negativo.

Veremos cada una de las rebanadas del programa marvin2.py y veremos lo que producen. Cuando el intérprete ve la especificación de la rebanada, extrae los objetos cortados de las letras y devuelve una copia de los objetos al bucle for. La lista de letters originales no se ve afectada por estos cortes.

La primera sección extrae del comienzo de la lista y termina (pero no incluye) el objeto en la ranura 6:

```
for char in letters[:6]:  
    print('\t', char)
```

letters[:6] → A sequence of boxes containing the characters M, a, r, v, i, n.

La segunda slice extrae del final de la lista de letras, comenzando en la ranura -7 y pasando al final de las letras:

```
for char in letters[-7:]:  
    print('\t'*2, char)
```

letters[-7:] → A sequence of boxes containing the characters A, n, d, r, o, i, d.

Y, por último, la tercera rebanada extractos de la mitad de la lista, comenzando en la ranura 12 e incluyendo todo hasta pero sin incluir la ranura 20:

```
for char in letters[12:20]:  
    print('\t'*3, char)
```

letters[12:20] → A sequence of boxes containing the characters P, a, r, a, n, o, i, d.

Puedo verme poniendo listas a muchos usos en mis programas de Python. ¿Pero hay listas de cosas que no son buenas?

**Las listas se utilizan mucho, pero ...**

No son una panacea de la estructura de datos. Las listas se pueden utilizar en muchos lugares; Si tiene una colección de objetos similares que necesita almacenar en una estructura de datos, las listas son la opción perfecta.

Sin embargo, y quizás algo contraintuitivamente, si los datos con los que está trabajando exhiben alguna estructura, las listas pueden ser una mala elección. Empezaremos a explorar este problema (y lo que puede hacer al respecto) en la siguiente página.

## What's Wrong with Lists? ¿Qué hay de malo en las listas?

Cuando los programadores de Python se encuentran en una situación donde necesitan almacenar una colección de objetos similares, usar una lista es a menudo la opción natural. Después de todo, hemos utilizado nada más que listas en este capítulo hasta ahora.

Recuerde cómo las listas son grandes en el almacenamiento de una colección de letters relacionadas, como con la lista de vowels:

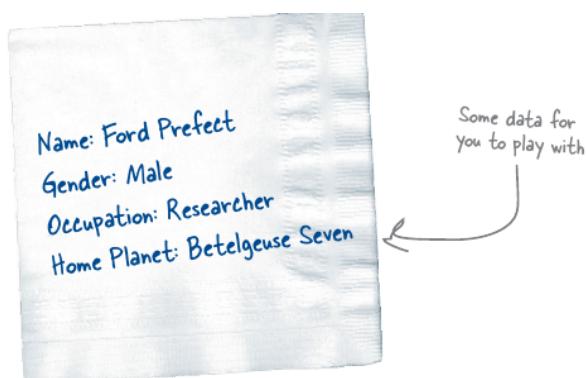
```
vowels = ['a', 'e', 'i', 'o', 'u']
```

Y si los datos son una colección de números, las listas son una gran opción, también:

```
nums = [1, 2, 3, 4, 5]
```

De hecho, las listas son una gran opción cuando usted tiene una colección de cualquier cosa relacionados.

Pero imagine que necesita almacenar datos sobre una persona, y los datos de ejemplo que le han dado se parecen a esto:



Algunos datos para usted jugar

A la vista de las cosas, estos datos sí se ajustan a una estructura, en la que hay etiquetas a la izquierda y valores de datos asociados a la derecha. Entonces, ¿por qué no poner estos datos en una lista? Después de todo, estos datos están relacionados con la persona, ¿verdad?

Para ver por qué no deberíamos, veamos dos maneras de almacenar estos datos usando listas (comenzando en la página siguiente). Vamos a estar totalmente al frente aquí: ambos de nuestros intentos presentan problemas que hacen que usar listas menos que ideal para datos como este. Pero, como el viaje es a menudo la mitad de la diversión de llegar allí, vamos a intentar listas de todos modos.

Nuestro primer intento se concentra en los valores de datos a la derecha de la servilleta, mientras que nuestro segundo intento utiliza las etiquetas a la izquierda, así como los valores de datos asociados. Piense en cómo manejaría este tipo de datos estructurados usando listas, luego vaya a la siguiente página para ver cómo funcionaron nuestros dos intentos.

## When Not to Use Lists

### Cuándo no usar listas

Tenemos nuestros datos de muestra (en la parte de atrás de una servilleta) y hemos decidido almacenar los datos en una lista (ya que es todo lo que sabemos en este momento en nuestros viajes de Python).

Nuestro primer intento toma los valores de datos y los coloca en una lista:

```
>>> person1 = ['Ford Prefect', 'Male',  
'Researcher', 'Betelgeuse Seven']  
>>> person1  
['Ford Prefect', 'Male', 'Researcher',  
'Betelgeuse Seven']
```



¿"Persona [1]" se refiere a género u ocupación? ¡Nunca puedo recordar!

Esto resulta en una lista de objetos de cadena, que funciona. Como se muestra arriba, el shell confirma que los valores de datos están ahora en una lista llamada person1.

Pero tenemos un problema, ya que tenemos que recordar que la primera ubicación del índice (en el valor del índice 0) es el nombre de la persona, el siguiente es el género de la

persona (en el valor del índice 1), y así sucesivamente. Para un pequeño número de elementos de datos, esto no es un gran problema, pero imagínese si estos datos se expandieron para incluir muchos más valores de datos (tal vez para dar soporte a una página de perfil en ese asesino de Facebook que ha estado intentando construir). Con datos como este, el uso de valores de índice para referirse a los datos de la lista personal es quebradizo y lo mejor es evitarlo.

Nuestro segundo intento añade las etiquetas a la lista, de modo que cada valor de datos es precedido por su etiqueta asociada. Conoce a la lista person2:

```
>>> person2 = ['Name', 'Ford Prefect', 'Gender',
   'Male', 'Occupation', 'Researcher', 'Home Planet',
   'Betelgeuse Seven']
>>> person2
['Name', 'Ford Prefect', 'Gender', 'Male',
 'Occupation', 'Researcher', 'Home Planet',
 'Betelgeuse Seven']
```

Esto funciona claramente, pero ahora ya no tenemos un problema; tenemos dos. No sólo tenemos que recordar lo que está en cada ubicación de índice, sino que ahora tenemos que recordar que los valores de índice 0, 2, 4, 6 y así sucesivamente son etiquetas, mientras que los valores de índice 1, 3, 5, 7 y así sucesivamente son valores de datos.

*Seguramente tiene que haber una mejor manera de manejar los datos con una estructura como esta?*

***Si los datos que desea almacenar tienen una estructura identifiable, considere utilizar algo que no sea una lista.***

Existe, e implica renunciar al uso de listas para datos estructurados como este. Necesitamos usar otra cosa, y en Python, que otra cosa se llama un **diccionario**, lo que obtenemos en el próximo capítulo.

## Chapter 2's Code, 1 of 2

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = "Milliways"
for letter in word:
    if letter in vowels:
        print(letter)
```

The first version of the vowels program that displays **\*all\*** the vowels found in the word "Milliways" (including any duplicates).

The "vowels2.py" program added code that used a list to avoid duplicates. This program displays the list of unique vowels found in the word "Milliways".

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = "Milliways"
found = []
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
for vowel in found:
    print(vowel)
```

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")
found = []
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
for vowel in found:
    print(vowel)
```

The third (and final) version of the vowels program for this chapter, "vowels3.py", displays the unique vowels found in a word entered by our user.

It's the best advice in the universe: "Don't panic!" This program, called "panic.py", takes a string containing this advice and, using a bunch of list methods, transforms the string into another string that describes how the Head First editors prefer their beer: "on tap".

```
phrase = "Don't panic!"
plist = list(phrase)
print(phrase)
print(plist)

for i in range(4):
    plist.pop()
plist.pop(0)
plist.remove('"')
plist.extend([plist.pop(), plist.pop()])
plist.insert(2, plist.pop(3))

new_phrase = ''.join(plist)
print(plist)
print(new_phrase)
```

## Chapter 2's Code, 2 of 2

```
phrase = "Don't panic!"
plist = list(phrase)
print(phrase)
print(plist)

new_phrase = ''.join(plist[1:3])
new_phrase = new_phrase + ''.join([plist[5], plist[4], plist[7], plist[6]])

print(plist)
print(new_phrase)
```

When it comes to manipulating lists, using methods isn't the only game in town. The "panic2.py" program achieved the same end using Python's square bracket notation.

```
paranoid_android = "Marvin"
letters = list(paranoid_android)
for char in letters:
    print('\t', char)
```

The shortest program in this chapter, "marvin.py", demonstrated how well lists play with Python's "for" loop. (Just don't tell Marvin...if he hears that his program is the shortest in this chapter, it'll make him even more paranoid than he already is).

The "marvin2.py" program showed off Python's square bracket notation by using three → slices to extract and display fragments from a list of letters.

```
paranoid_android = "Marvin, the Paranoid Android"
letters = list(paranoid_android)
for char in letters[:6]:
    print('\t', char)
print()
for char in letters[-7:]:
    print('\t'*2, char)
print()
for char in letters[12:20]:
    print('\t'*3, char)
```

# 3 structured data

## Trabajo con datos estructurados

Las listas son geniales, pero a veces necesito más estructura en mi vida ...

**Estructura de datos de la lista de Python es grande, pero no es una panacea de datos.?**

Cuando tienes datos verdaderamente estructurados (y usar una lista para almacenarlo puede no ser la mejor opción), Python viene a tu rescate con su diccionario incorporado. Fuera de la caja, el diccionario le permite almacenar y manipular cualquier colección de pares clave / valor. Miramos largo y duro en el diccionario de Python en este capítulo, y-a lo largo del camino conoceremos conjunto s y tupla, también. Junto con la lista (que conocimos en el capítulo anterior), las estructuras de datos de diccionario, conjunto y tupla proporcionan un conjunto de herramientas de datos integradas que ayudan a que Python y los datos sean una combinación potente.

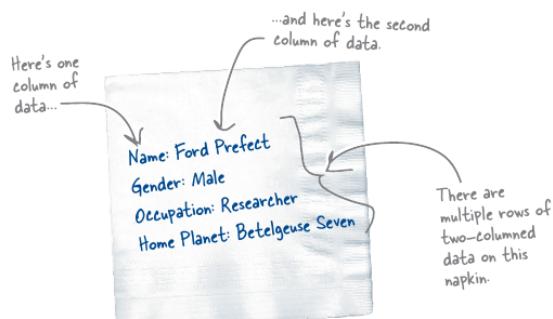
**A Dictionary Stores Key/Value Pairs**

**Un diccionario almacena pares clave / valor**

A diferencia de una lista, que es una colección de objetos relacionados, el diccionario se utiliza para contener una colección de pares clave / valor, donde cada clave única tiene un valor asociado. El diccionario se refiere a menudo como una matriz asociativa por los científicos de la computadora, y otros lenguajes de programación utilizan a menudo otros nombres para el diccionario (tal como mapa, hash, y tabla).

La parte **clave** de un diccionario de Python es típicamente una **cadena**, mientras que la parte de **valor** asociada puede ser cualquier **objeto** Python.

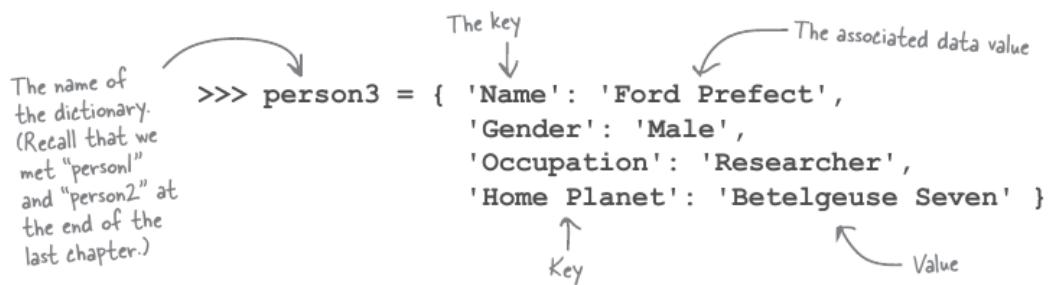
Los datos que se ajustan al modelo de diccionario son fáciles de detectar: hay dos columnas, con potencialmente múltiples filas de datos. Con esto en mente, eche otra mirada a nuestra "servilleta de datos" desde el final del último capítulo:



**En C ++ y Java, un diccionario se conoce como "mapa", mientras que Perl y Ruby usan el nombre "hash".**

Parece que los datos de esta servilleta son un ajuste perfecto para el diccionario de Python.

Volvamos a la shell >>> para ver cómo crear un diccionario usando nuestros datos de servilleta. Es tentador intentar ingresar al diccionario como una sola línea de código, pero no vamos a hacer esto. Como queremos que nuestro código de diccionario sea fácil de leer, introduciremos intencionalmente cada fila de datos (es decir, cada par clave / valor) en su propia línea. Echar un vistazo:



## Haga que los diccionarios sean fáciles de leer

key#4	object
key#1	object
key#3	object
key#2	object

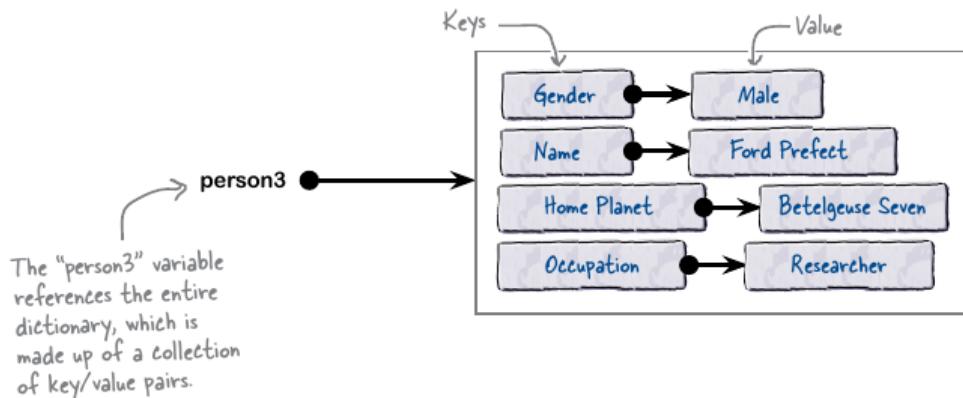
Dictionary

```
>>> person3 = { 'Name': 'Ford Prefect', 'Gender': 'Male', 'Occupation': 'Researcher', 'Home Planet': 'Betelgeuse Seven' }
```

A pesar de que el intérprete no le importa cuál enfoque utiliza, entrar en un diccionario como una larga línea de código es difícil de leer, y debe evitarse siempre que sea posible.

Si desecha su código con diccionarios que son difíciles de leer, otros programadores (lo que incluye a usted en seis meses) se molestan ... así que tome el tiempo para alinear su código de diccionario para que sea fácil de leer.

Esta es una representación visual de cómo el diccionario aparece en la memoria de Python después de ejecutar cualquiera de estas instrucciones de asignación de diccionario:



Esta es una estructura más complicada que la lista de tipo array. Si la idea detrás del diccionario de Python es nueva para usted, a menudo es útil pensar en ella como una tabla de búsqueda. La clave de la izquierda se utiliza para buscar el valor de la derecha (al igual que buscar una palabra en un diccionario de papel).

Pasemos un tiempo conociendo el diccionario de Python con más detalle. Comenzaremos con una explicación detallada de cómo detectar un diccionario de Python en su código, antes de hablar sobre algunas características y usos únicos de esta estructura de datos.

## Cómo detectar un diccionario en el código

Eche un vistazo a cómo definimos el diccionario `person3` en el shell >>>. Para empezar, todo el diccionario está encerrado entre llaves. Cada clave está encerrada entre comillas, ya que son cadenas, como es cada valor, que también son cadenas en este ejemplo. Cada clave está separada de su valor asociado por un carácter de dos puntos (:), y cada par de clave / valor (aka "fila") está separado del siguiente por un coma:

The diagram shows the Python code: `{'Name': 'Ford Prefect', 'Gender': 'Male', 'Occupation': 'Researcher', 'Home Planet': 'Betelgeuse Seven'}`. Callouts explain: "An opening curly brace starts each dictionary.", "Each key is enclosed in quotes.", "A colon associates each key with its value.", "In this dictionary, the values are all string objects, so they are enclosed in quotes.", "Each key/value pair is separated from the next by a comma.", "A closing curly brace ends each dictionary."

Como se dijo anteriormente, los datos de esta servilleta se correlacionan bien con un diccionario de Python. De hecho, cualquier dato que exhibe una estructura similar- múltiples filas de dos columnas- es el ajuste perfecto que es probable encontrar. Lo cual es genial, pero tiene un precio. Volvamos al prompt >>> para saber cuál es el precio:

```
>>> person3  
{'Gender': 'Male', 'Name': 'Ford Prefect', 'Home  
Planet': 'Betelgeuse Seven', 'Occupation': 'Researcher'}
```

Ask the shell to display the contents of the dictionary...

...and there it is. All the key/value pairs are shown.

Pide a la shell que muestre el contenido del diccionario ...

... y ahí está. Se muestran todos los pares clave / valor.

### ¿Qué pasó con la orden de inserción?

Echa un vistazo largo y duro al diccionario que muestra el intérprete. ¿Se dio cuenta de que la orden es diferente de lo que se utilizó en la entrada? Cuando creó el diccionario, insertó las filas en nombre, género, ocupación y orden de origen del planeta, pero el shell los está mostrando en género, nombre, planeta natal y orden de ocupación. El orden ha cambiado.

### ¿Que está pasando aquí? ¿Por qué cambió el orden?

## Insertion Order Is NOT Maintained -No se mantiene la orden de inserción

A diferencia de las listas, que mantienen sus objetos ordenados en el orden en que los insertó, el diccionario de Python no. Esto significa que no puede asumir que las filas de un diccionario están en un orden particular; Para todos los intentos y propósitos, están desordenados.

Echa un vistazo al diccionario person3 y compara el orden de entrada con el mostrado por el intérprete en el prompt >>>:

```
>>> person3 = { 'Name': 'Ford Prefect',  
                 'Gender': 'Male',  
                 'Occupation': 'Researcher',  
                 'Home Planet': 'Betelgeuse Seven' }  
  
>>> person3  
{'Gender': 'Male', 'Name': 'Ford Prefect', 'Home Planet': 'Betelgeuse Seven', 'Occupation': 'Researcher'}
```

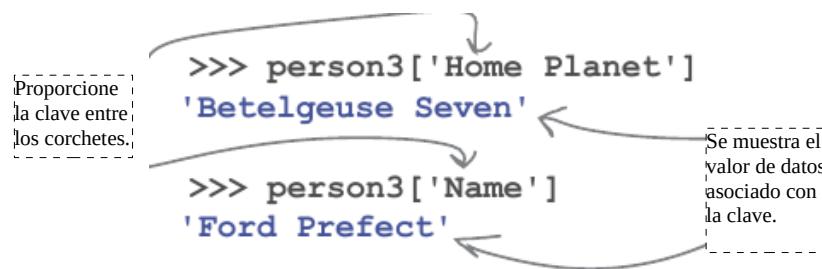
You insert your data into a dictionary in one order...

...but the interpreter uses another ordering.

Si te estás rascando la cabeza y preguntándote por qué quieras confiar tus preciosos datos en una estructura de datos desordenada, no te preocupes, ya que la ordenación rara vez marca la diferencia. Cuando selecciona los datos almacenados en un diccionario, no tiene nada que ver con el orden del diccionario, y todo lo que tiene que ver con la clave que utilizó. Recuerde: **una clave se utiliza para buscar un valor.**

## Los diccionarios entienden los corchetes

Al igual que las listas, los diccionarios entienden la notación de corchetes. Sin embargo, a diferencia de las listas, que utilizan valores de índice numérico para acceder a los datos, los diccionarios usan claves para acceder a sus valores de datos asociados. Veamos esto en acción en el prompt >>> del intérprete:



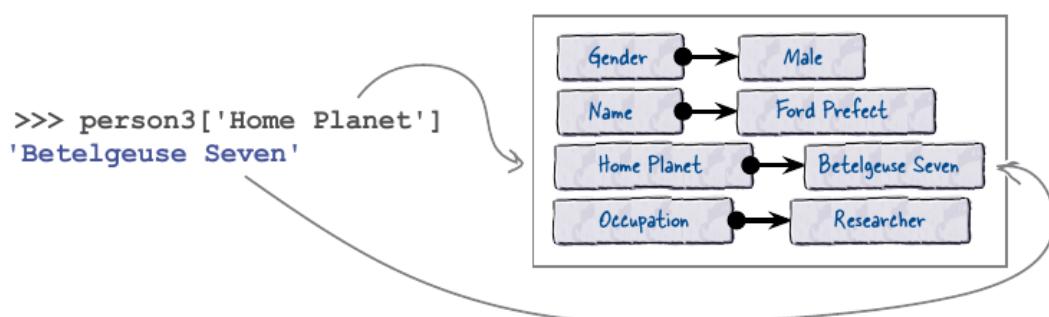
Cuando considere que puede acceder a sus datos de esta manera, se hace evidente que no importa en qué orden el intérprete almacena sus datos.

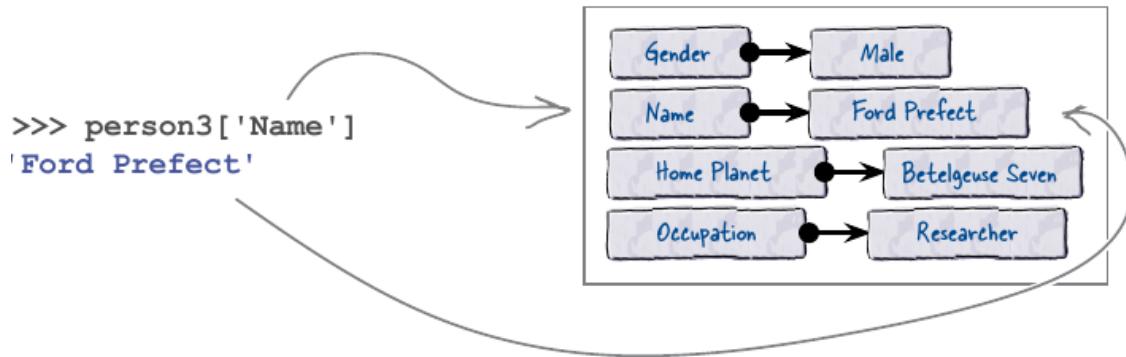
**Utilice las claves para acceder a los datos de un diccionario.**

## Búsqueda de valores con corchetes cuadrados

El uso de corchetes con diccionarios funciona de la misma manera que con las listas. Sin embargo, en lugar de acceder a sus datos en una ranura especificada utilizando un valor de índice, con el diccionario de Python accede a sus datos a través de la clave asociada a ella.

Como vimos al final de la última página, al colocar una clave dentro de los corchetes de un diccionario, el intérprete devuelve el valor asociado a la clave. Consideremos nuevamente esos ejemplos para ayudar a consolidar esta idea en su cerebro:





## Búsqueda de diccionario es rápido!

Esta capacidad de extraer cualquier valor de un diccionario usando su clave asociada es lo que hace que el diccionario de Python sea tan útil, ya que hay muchas ocasiones en que se hace necesario, por ejemplo, buscar los detalles de usuario en un perfil. Haciendo aquí con el diccionario **person3**.

No importa en qué orden se almacena el diccionario. Lo único que importa es que el intérprete pueda acceder rápidamente al valor asociado a una clave (no importa cuán grande sea su diccionario). La buena noticia es que el intérprete hace precisamente eso, gracias al empleo de un algoritmo de hashing altamente optimizado. Al igual que con muchos componentes internos de Python, puedes dejar al intérprete sin problemas de manejar todos los detalles aquí, mientras te llevas aprovechando lo que el diccionario de Python tiene para ofrecer.

El diccionario Python se implementa como una tabla de hash redimensionable, que ha sido muy optimizada para muchos casos especiales. Como resultado, los diccionarios realizan búsquedas muy rápidamente.

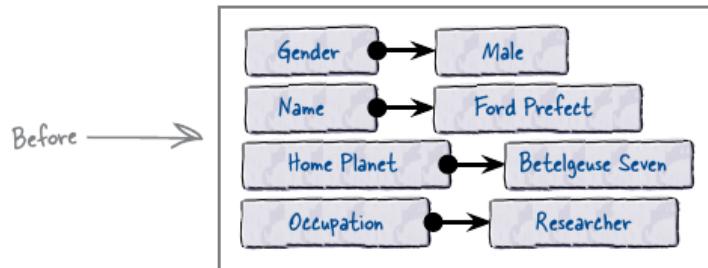
## **Trabajo con diccionarios en tiempo de ejecución**

Saber cómo funciona la notación de corchetes con diccionarios es fundamental para entender cómo crecen los diccionarios en tiempo de ejecución. Si tiene un diccionario existente, puede agregarle un nuevo par clave / valor asignando un objeto a una nueva clave, que se proporciona entre corchetes.

Por ejemplo, aquí mostramos el estado actual del diccionario **person3**, luego añadimos un nuevo par clave / valor que asocia 33 con una clave llamada **Age**. A continuación, mostrar el diccionario **person3** de nuevo para confirmar la nueva fila de datos se agregó correctamente:

Before the new row is added

```
>>> person3
{'Name': 'Ford Prefect', 'Gender': 'Male',
'Home Planet': 'Betelgeuse Seven',
'Occupation': 'Researcher'}
```



Before →

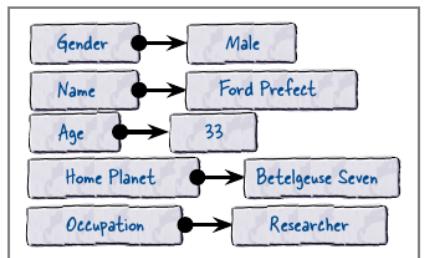
```
>>> person3['Age'] = 33
```

Assign an object (in this case, a number) to a new key to add a row of data to the dictionary.

```
>>> person3
{'Name': 'Ford Prefect', 'Gender': 'Male',
'Age': 33, 'Home Planet': 'Betelgeuse Seven',
'Occupation': 'Researcher'}
```

After the new row is added

Here's the new row of data:  
"33" is associated with "Age".



← After

### *remembering vowels3.py*

### **Recap: Displaying Found Vowels (Lists)**

### **Recapitulación: Visualización de vocales encontradas (listas)**

Como se muestra en la última página, el crecimiento de un diccionario de esta manera se puede utilizar en muchas situaciones diferentes. Una aplicación muy común es realizar un recuento de frecuencia: procesar algunos datos y mantener un recuento de lo que encuentre. Antes de demostrar cómo realizar un recuento de frecuencia usando un diccionario, volvamos a nuestro ejemplo de conteo de vocales del último capítulo.

Recuerde que **vowels3.py** determina una lista única de vocales encontradas en una palabra. Imagine que ahora se le ha pedido que amplíe este programa para producir una salida que detalla cuántas veces cada vocal aparece en la palabra.

Aquí está el código del capítulo 2, que, dada una palabra, muestra una lista única de vocales encontradas:

This is "vowels3.py", →  
which reports on  
the unique vowels  
found in a word.

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")
found = []
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
for vowel in found:
    print(vowel)
```

Ln: 11 Col: 0

Recuerde que hemos ejecutado este código a través de IDLE un número de veces:

```
Python 3.4.3 Shell
>>> ===== RESTART =====
>>> Provide a word to search for vowels: Milliways
i
a
>>> ===== RESTART =====
>>> Provide a word to search for vowels: Hitch-hiker
i
e
>>> ===== RESTART =====
>>> Provide a word to search for vowels: Galaxy
a
>>> ===== RESTART =====
>>> Provide a word to search for vowels: Sky
>>> |
```

Ln: 21 Col: 4

## ¿Cómo puede un diccionario ayudar aquí?

No lo entiendo. El programa "vowels3.py" funciona bien ... así que ¿por qué estás buscando arreglar algo que no esté roto?

### No lo somos.

El programa vowels3.py hace lo que se supone que debe hacer, y usar una lista para esta versión de la funcionalidad del programa tiene sentido.

Sin embargo, imagínese si necesita no sólo enumerar las vocales en cualquier palabra, sino también informar de su frecuencia. ¿Qué pasa si necesita saber cuántas veces cada vocal aparece en una palabra?

Si usted piensa en ello, esto es un poco más difícil de hacer con listas solo. Pero lanzar un diccionario en la mezcla, y las cosas cambian.

Exploremos el uso de un diccionario con el programa de vocales en las próximas páginas para satisfacer este nuevo requisito.

### Cual es la frecuencia, kenneth?

#### Selección de una estructura de datos de cuenta de frecuencia

Queremos ajustar el programa vowels3.py para mantener un conteo de la frecuencia con la que cada vocal está presente en una palabra; Es decir, ¿cuál es la frecuencia de cada vocal? Vamos a esbozar lo que esperamos ver como salida de este programa:

Given the word "hitchhiker", here's the frequency count we expect to see:	
a	0
e	1
i	2
o	0
u	0

Vocales en la columna izquierda

Recuentos de frecuencia en la columna derecha

Esta salida es una combinación perfecta con la forma en que el intérprete considera un diccionario. En lugar de utilizar una lista para almacenar las vocales encontradas (como ocurre en vowels3.py), utilicemos un diccionario. Podemos seguir llamando a la colección **found**, pero tenemos que inicializarla a un diccionario vacío en lugar de una lista vacía.

Como siempre, vamos a experimentar y resolver lo que debemos hacer en el prompt >>>, antes de cometer cualquier cambio en el código vowels3.py. Para crear un diccionario vacío, asigne {} a una variable:

```
>>> found = {}  
>>> found  
{}
```

Curly braces on their own mean the dictionary starts out empty.

Vamos a registrar el hecho de que aún no hemos encontrado vocales creando una fila para cada vocal e inicializando su valor asociado a 0. Cada vocal se usa como clave:

```

>>> found['a'] = 0
>>> found['e'] = 0
>>> found['i'] = 0
>>> found['o'] = 0
>>> found['u'] = 0
>>> found
{'o': 0, 'u': 0, 'a': 0, 'i': 0, 'e': 0}

```

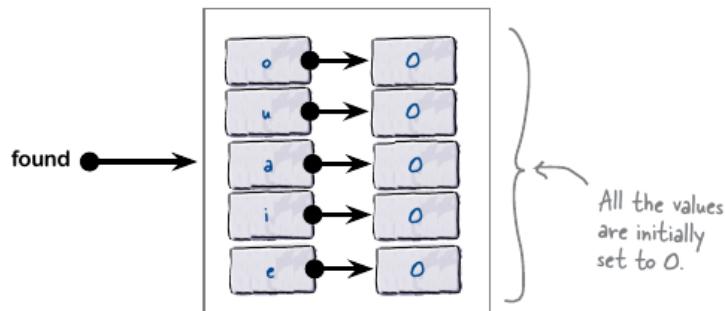
We've initialized all the vowel counts to 0. Note how insertion order is not maintained (but that doesn't matter here).

Hemos inicializado todos los conteos de vocales a 0. Observe cómo no se mantiene el orden de inserción (pero eso no importa aquí).

Todo lo que necesitamos hacer ahora es encontrar una vocal en una palabra dada, luego actualizar estos recuentos de frecuencia según sea necesario.

### **Actualización de un contador de frecuencia**

Antes de llegar al código que actualiza los recuentos de frecuencia, considere como el intérprete ve el diccionario encontrado en la memoria después de ejecutar el código de inicialización del diccionario:



Con los recuentos de frecuencia inicializados a 0, no es difícil incrementar un valor determinado, según sea necesario. Por ejemplo, aquí es cómo incrementar la cuenta de frecuencia de e:

```

>>> found
{'o': 0, 'u': 0, 'a': 0, 'i': 0, 'e': 0}
>>> found['e'] = found['e'] + 1
>>> found
{'o': 0, 'i': 0, 'a': 0, 'u': 0, 'e': 1}

```

Everything is 0.  
Increment e's count.  
The dictionary has been updated. The value associated with "e" has been incremented.

Código como el que se ha resaltado por encima de duda funciona, pero tener que repetir encontrado ['e'] a cada lado del operador de asignación

obtiene muy viejo, muy rápidamente. Así que veamos un atajo para esta operación (en la página siguiente).

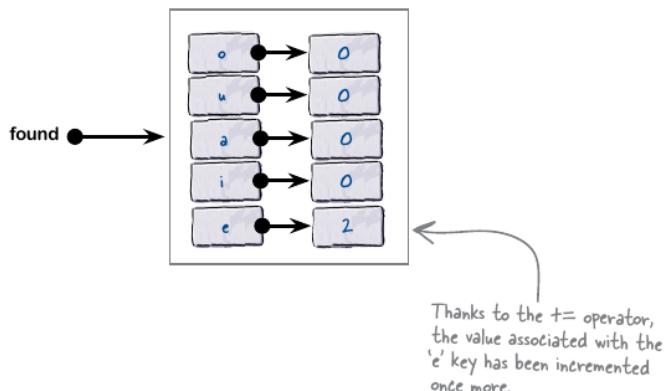
## Más iguales

### Actualización de un contador de frecuencia, v2.0

Tener que poner ['e'] encontrado a cada lado del operador de asignación (=) se convierte rápidamente en fastidioso, por lo que Python apoya al familiar operador +=, que hace lo mismo, pero de una manera más sucinta:

```
>>> found['e'] += 1           Increment e's  
>>> found                   count (once more).  
{'o': 0, 'i': 0, 'a': 0, 'u': 0, 'e': 2}    El diccionario se actualiza de nuevo.
```

En este punto, hemos incrementado el valor asociado con la clave dos veces, así que aquí está cómo el diccionario busca al intérprete ahora:



## Iterando sobre un diccionario

En este punto, le mostramos cómo inicializar un diccionario con datos cero, así como actualizar un diccionario incrementando un valor asociado con una clave. Estamos casi listos para actualizar el programa `vowels3.py` para realizar un recuento de frecuencia basado en vocales encontradas en una palabra. Sin embargo, antes de hacerlo, determinemos qué sucede cuando iteramos sobre un diccionario, ya que una vez que tengamos el diccionario poblado con datos, necesitaremos una forma de mostrar nuestro recuento de frecuencias en la pantalla.

Se le perdonaría pensar que todo lo que necesitamos hacer aquí es usar el diccionario con un bucle `for`, pero al hacerlo se producen resultados inesperados:

```

>>> for kv in found:
        print(kv)

```

We iterate over the dictionary in the usual way, using a "for" loop. Here, we're using "kv" as shorthand for "key/value pair" (but could've used any variable name).

o  
i  
a  
u  
e

The iteration worked, but this isn't what we were expecting. Where have the frequency counts gone? This output is only showing the keys...

Hacemos una iteración sobre el diccionario de la manera habitual, usando un bucle "for". Aquí, estamos usando "kv" como abreviatura de "par clave / valor" (pero podría haber utilizado cualquier nombre de variable).

La iteración funcionó, pero esto no es lo que esperábamos. ¿A dónde han ido los recuentos de frecuencia? Esta salida sólo muestra las keys ...

**Algo realmente no está bien con esta salida. Se muestran las keys o claves, pero no sus valores asociados. ¿Lo que da?**

## Iterando sobre claves y valores

Cuando iteró sobre un diccionario con su bucle for, el intérprete sólo procesó las claves del diccionario.

Para acceder a los valores de datos asociados, debe colocar cada clave entre corchetes y utilizarla junto con el nombre del diccionario para obtener acceso a los valores asociados con la clave.

La versión del bucle que se muestra a continuación hace exactamente eso, proporcionando no sólo las claves, sino también sus valores de datos asociados. Hemos cambiado la suite para acceder a cada valor basado en cada clave proporcionada al bucle for.

A medida que el bucle **for** itera sobre cada par clave / valor en el diccionario, la clave de la línea actual se asigna a k, entonces se encuentra **found[k]** para acceder a su valor asociado. También hemos producido una salida más amigable con el usuario pasando dos cadenas a la llamada a la función de impresión **print**:

```
>>> for k in found:
    print(k, 'was found', found[k], 'time(s).')

o was found 0 time(s).
i was found 0 time(s).
a was found 0 time(s).
u was found 0 time(s).
e was found 2 time(s).
```

Estamos usando "k" para representar la clave, y "found [k]" para acceder al valor.

This is more like it. The keys and the values are being processed by the loop and displayed on screen.

Esto es más como él. Las claves y los valores están siendo procesados por el bucle y visualizados en la pantalla.

Si usted está siguiendo a lo largo de su prompt>>> y su salida se ordena de forma diferente a la nuestra, no se preocupe: el intérprete utiliza un orden interno al azar como usted está usando un diccionario aquí, y no hay garantías sobre el orden cuando se utiliza. Es probable que su ordenación difiera de la nuestra, pero no se alarme. Nuestra principal preocupación es que los datos se almacenan con seguridad en el diccionario, que es.

Obviamente, el ciclo anterior funciona. Sin embargo, hay dos puntos que nos gustaría hacer.

**En primer lugar:** sería bueno si la salida se ordenó a, e, i, o, u, en contraposición a lo aleatorio, ¿no?

**En segundo lugar:** a pesar de que este bucle funciona claramente, codificar una iteración de diccionario de esta manera no es el enfoque preferido -la mayoría de los programadores de Python codifican esto de manera diferente.

Vamos a explorar estos dos puntos en un poco más de detalle (después de una revisión rápida).

### **Especificar el orden de un diccionario en la salida**

Hemos visto que es posible iterar sobre las filas de datos en un diccionario usando este código:

```
>>> for k in sorted(found):
    print(k, 'was found', found[k], 'time(s).')

a was found 0 time(s).
e was found 2 time(s).
i was found 0 time(s).
o was found 0 time(s).
u was found 0 time(s).
```

Al igual que las listas, los diccionarios tienen un montón de métodos integrados, y uno de ellos es el método **items**, que devuelve una lista de los pares clave / valor. El uso de elementos con for es a menudo la técnica preferida para iterar sobre un diccionario, ya que le da acceso a la clave y el valor como variables de bucle, que puede utilizar en su suite. La suite resultante es más fácil en el ojo, lo que hace que sea más fácil de leer.

Aquí están los elementos o items equivalentes del código de bucle anterior. Observe cómo ahora hay dos variables de bucle en esta versión del código ( $k$  y  $v$ ), y que seguimos utilizando la función ordenada para controlar el orden de salida:

```
>>> for k, v in sorted(found.items()):  
    print(k, 'was found', v, 'time(s).')  
  
a was found 0 time(s).  
e was found 2 time(s).  
i was found 0 time(s).  
o was found 0 time(s).  
u was found 0 time(s).  
  
El método  
"items"  
devuelve dos  
variables de  
bucle.
```

Invocamos el método "items" en el diccionario "found".

...but this code is so much easier to read.

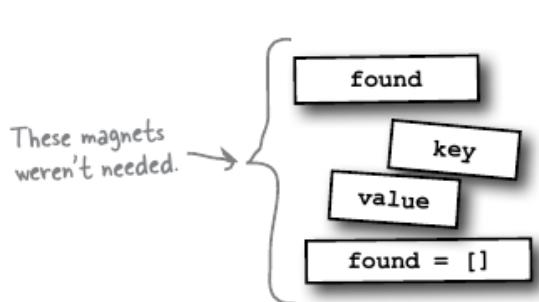
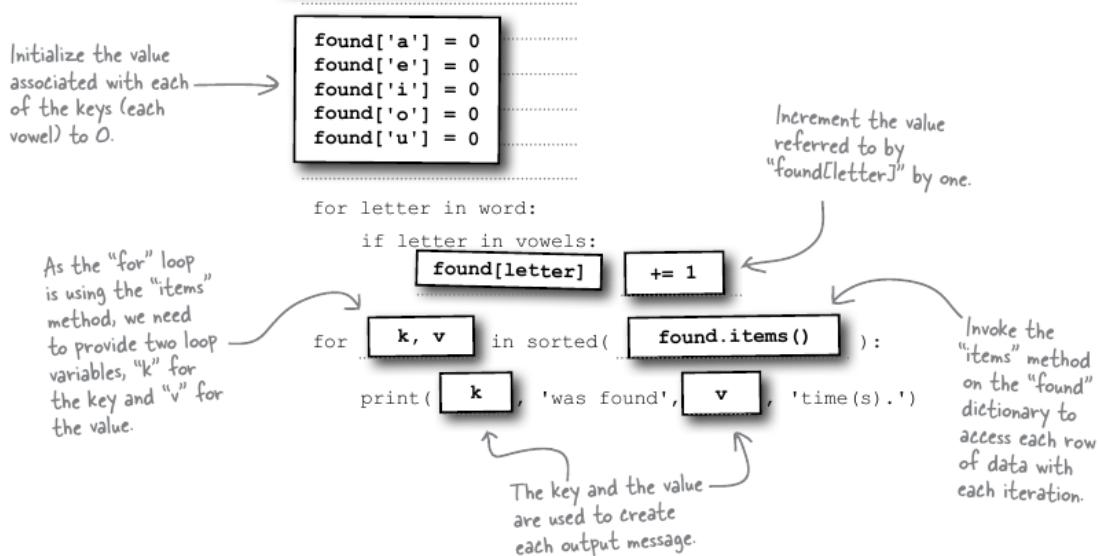
Same output as before...

Habiendo concluido nuestra experimentación en el prompt >>>, ahora es el momento de realizar cambios en el programa `vowels3.py`. A continuación se muestran todos los fragmentos de código que creemos que podrían necesitar. Su trabajo consiste en reorganizar los imanes para producir un programa de trabajo que, cuando se le da una palabra, produce un recuento de frecuencia para cada vocal encontrada.

Una vez que pusiste los imanes donde pensabas que debían ir, debías traer vowels3.py a una ventana de edición de IDLE, renombrarla vowels4.py y luego aplicar los cambios de código a la nueva versión de este programa.

This is the “vowels4.py” program.

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")
found = {}
```



Tomemos `vowels4.py` para una vuelta. Con su código en una ventana de edición IDLE, presione F5 para ver cómo se realiza:

The "vowels4.py" → code

We ran the code three times to see how well it performs.

```

vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")

found = {}

found['a'] = 0
found['e'] = 0
found['i'] = 0
found['o'] = 0
found['u'] = 0

for letter in word:
    if letter in vowels:
        found[letter] += 1

for k, v in sorted(found.items()):
    print(k, 'was found', v, 'time(s.)')

```

Python 3.4.3 Shell

```

>>> ===== RESTART =====
>>>
Provide a word to search for vowels: hitch-hiker
a was found 0 time(s).
e was found 1 time(s).
i was found 2 time(s).
o was found 0 time(s).
u was found 0 time(s).
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: life, the universe, and everything
a was found 1 time(s).
e was found 6 time(s).
i was found 3 time(s).
o was found 0 time(s).
u was found 1 time(s).
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: sky
a was found 0 time(s).
e was found 0 time(s).
i was found 0 time(s).
o was found 0 time(s).
u was found 0 time(s).
>>>

```

These three "runs" produce the output we expect them to.

*Me gusta a donde está yendo esto. ¿Pero realmente necesito que me digan cuando no se encuentra una vocal?*

### **¿Qué tan dinámicos son los diccionarios?**

El programa `vowels4.py` informa sobre todas las vocales encontradas, incluso cuando no se encuentran. Esto no le puede molestar, pero vamos a imaginar que lo hace y desea que este código sólo mostrar resultados cuando los resultados se encuentran realmente. Es decir, no desea ver ninguno de esos "mensajes encontrados 0 tiempo (s)".

¿Cómo puede usted ir sobre la solución de este problema?

*El diccionario de Python es dinámico, ¿verdad? Por lo tanto, todo lo que tenemos que hacer es quitar las cinco líneas que inicializar el conteo de frecuencia de cada vocal? Con esas líneas desaparecidas, sólo se contarán las vocales encontradas, ¿verdad?*

### Parece que podría funcionar.

Actualmente tenemos cinco líneas de código cerca del inicio del programa vowels4.py que hemos incluido con el fin de establecer inicialmente el número de frecuencias de cada vocal a 0. Esto crea un par clave / valor para cada vocal, aunque algunas nunca pueden ser usado. Si quitamos esas cinco líneas, deberíamos terminar solo registrando los conteos de frecuencia para las vocales encontradas, e ignoraremos el resto.

Vamos a probar esta idea.

Tome el código en vowels4.py y guárdelo como vowels5.py. A continuación, quite las cinco líneas de código de inicialización. Su ventana de edición de IDLE debería verse como a la derecha de esta página.

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")

found = {}

for letter in word:
    if letter in vowels:
        found[letter] += 1

for k, v in sorted(found.items()):
    print(k, 'was found', v, 'time(s).')
```

Ya sabes que hacer. Asegúrese de que vowels5.py esté en una ventana de edición IDLE, luego presione F5 para ejecutar su programa. Usted será confrontado por un mensaje de error de tiempo de ejecución:

Esto no puede ser bueno.

```
Python 3.4.3 Shell
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: hitchhiker
Traceback (most recent call last):
  File "/Users/Paul/Desktop/_NewBook/ch03/vowels5.py", line 9, in <module>
    found[letter] += 1
KeyError: 'i'
>>>
```

Está claro que eliminar las cinco líneas de código de inicialización no era el camino a seguir aquí. Pero, ¿por qué ha sucedido esto? El hecho de que el diccionario de Python crezca dinámicamente en tiempo de ejecución debería significar que este código no se puede bloquear, pero lo hace. ¿Por qué recibimos este error?

### Se deben inicializar las claves de diccionario

Eliminar el código de inicialización ha resultado en un error de tiempo de ejecución, específicamente un **KeyError**, que se genera cuando intenta tener acceso a un valor asociado con una clave inexistente. Debido a que no se puede encontrar la clave, no se puede encontrar el valor asociado con él y obtiene un error.

¿Significa esto que tenemos que volver a poner el código de inicialización? Después de todo, es sólo cinco líneas cortas de código, así que ¿cuál es el daño? Ciertamente podemos hacer esto, pero vamos a pensar en hacerlo por un momento.

Imagine que, en lugar de cinco cuentas de frecuencia, tiene un requisito para realizar un seguimiento de mil (o más). De repente, tenemos un montón de código de inicialización. Podríamos "automatizar" la inicialización con un bucle, pero todavía estaríamos creando un diccionario grande con muchas filas, muchas de las cuales pueden terminar sin ser usadas.

Si sólo hubiera una manera de crear un par clave / valor sobre la marcha, tan pronto como nos damos cuenta de que lo necesitamos.

Un enfoque alternativo para tratar este problema es tratar con la excepción de tiempo de ejecución planteada aquí (que es un "KeyError" en este ejemplo). Estamos deteniéndonos hablando de cómo Python maneja las excepciones de ejecución hasta un capítulo posterior, por lo que soporta con nosotros por ahora.

**Me pregunto si el operador "in" trabaja con diccionarios?**

Comprobar con in

## Evitar KeyErrors en tiempo de ejecución

Al igual que con las listas, es posible utilizar el operador **in** para comprobar si existe una clave en un diccionario; El intérprete devuelve **True** o **False** dependiendo de lo que se encuentre.

Utilicemos este hecho para evitar esa excepción de **KeyError**, ya que puede ser molesto cuando el código se detiene como resultado de que se genere este error durante un intento de llenar un diccionario en tiempo de ejecución.

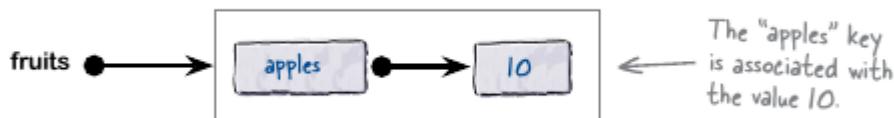
Para demostrar esta técnica, vamos a crear un diccionario llamado frutas, a continuación, utilizar el operador **in** para evitar el aumento de un **KeyError** al acceder a una clave inexistente. Comenzamos creando un diccionario vacío; Entonces asignamos un par clave / valor que asocia el valor 10 con las manzanas clave. Con la fila de datos en el diccionario, podemos usar el operador **in** para confirmar que las manzanas clave ya existen:

```
>>> fruits
{}
>>> fruits['apples'] = 10
>>> fruits
{'apples': 10}
>>> 'apples' in fruits
True
```

This is all as expected. The value is associated with the key, and there's no runtime error when we use the "in" operator to check for the key's existence.

Esto es todo como se esperaba. El valor se asocia con la clave y no hay error de tiempo de ejecución cuando usamos el operador "in" para comprobar la existencia de la clave.

Antes de hacer otra cosa, consideremos cómo el intérprete ve el diccionario de frutas en la memoria después de ejecutar el código anterior:



## Comprobación de la calidad de miembro con "en"

Vamos a agregar en otra fila de datos al diccionario de frutas para los plátanos y ver qué pasa. Sin embargo, en lugar de una asignación directa a los plátanos (como en el caso de las manzanas), vamos a incrementar el valor asociado a los plátanos por 1 si ya existe en el diccionario de frutas o, si no existe, vamos a inicializar bananas a 1. Esta es una actividad muy común, especialmente cuando está realizando conteos de frecuencia usando un diccionario, y la lógica que empleamos debería esperanzadamente ayudarnos a evitar un KeyError.



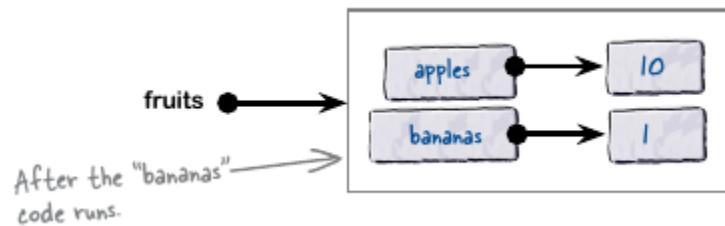
Antes de que se ejecute el código de "bananas"

En el código que sigue, el operador in junto con una sentencia if evita cualquier deslizamiento con plátanos, que -como dicen las palabras- es bastante malo (incluso para nosotros):

```
>>> if 'bananas' in fruits: ← Comprobamos si la clave
    fruits['bananas'] += 1   "plátanos" está en el
else:                                diccionario, y como no lo
    fruits['bananas'] = 1   es, inicializamos su valor a
                           1. Críticamente, evitamos
                           cualquier posibilidad de un
                           "KeyError".
```

We've set the "bananas" value to 1.

El código anterior cambia el estado del diccionario de frutas dentro de la memoria del intérprete, como se muestra aquí:



Como se esperaba, el diccionario de frutas ha crecido por un par clave / valor, y el valor de los bananos se ha inicializado a 1. Esto sucedió porque la condición asociada con la sentencia if evaluada a False (como la clave no se encontró), por lo que el Segunda suite (es decir, la asociada con else) ejecutada en su lugar. Veamos qué sucede cuando este código se ejecuta de nuevo.

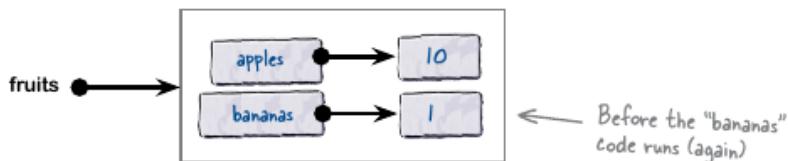
*Si está familiarizado con el operador ternario ?: de otros idiomas, tenga en cuenta que Python admite una construcción similar. Usted puede decir esto:*

`x = 10 if y > 3 else 20`

*Para establecer x a 10 o 20 dependiendo de si o no el valor de y es mayor que 3. Dicho esto, la mayoría de los programadores de Python de su uso, como el equivalente si ... else ... declaraciones se consideran más fáciles de leer .*

## **Garantizar la inicialización antes del uso**

Si ejecutamos el código nuevamente, el valor asociado a los plátanos ahora debe ser incrementado en 1, ya que si la suite se ejecuta esta vez debido al hecho de que la clave bananas ya existe en el diccionario de frutas:



Para ejecutar

este código nuevamente, presione Ctrl-P (en un Mac) o Alt-P (en Linux / Windows) para retroceder a través de sus instrucciones de código previamente ingresadas mientras está en el prompt >>> de IDLE (como usando la flecha hacia arriba para recuperar la entrada No funciona en el prompt >>> de IDLE). Recuerde pulsar Intro dos veces para ejecutar el código una vez más:

```

>>> if 'bananas' in fruits:
    fruits['bananas'] += 1
else:
    fruits['bananas'] = 1

>>> fruits
{'bananas': 2, 'apples': 10}

```

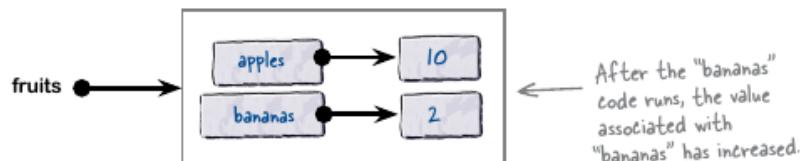
↑  
We've increased the "bananas" value by 1.

This time around, the "bananas" key does exist in the dictionary, so we increment its value by 1. As before, our use of "if" and "in" together stop a "KeyError" exception from crashing this code.

Esta vez, la clave de "plátanos" existe en el diccionario, por lo que incrementar su valor por 1. Como antes, nuestro uso de "if" y "in" juntos detienen una excepción "KeyError" de bloquear este código.

Hemos aumentado el valor de "bananas" en 1.

A medida que el código asociado con la instrucción if se ejecuta ahora, el valor asociado con bananas se incrementa dentro de la memoria del intérprete:



Este mecanismo es tan común que muchos programadores de Python acortan estas cuatro líneas de código invirtiendo la condición. En vez de comprobar con **in**, no utilizan. Esto permite que usted inicialice la llave a un valor del arrancador (generalmente 0) si no se encuentra, después realiza el incremento después de.

*Echemos un vistazo a cómo funciona este mecanismo.*

### **Substituting “not in” for “in” - Sustituyendo “no in” por “in”**

En la parte inferior de la última página, afirmamos que la mayoría de los programadores de Python refactorizar las cuatro líneas originales de código para usar **not in** vez de **in**. Vamos a ver esto en acción mediante este mecanismo para asegurarse de que la clave **peras** se establece en 0 antes de intentar Para incrementar su valor:

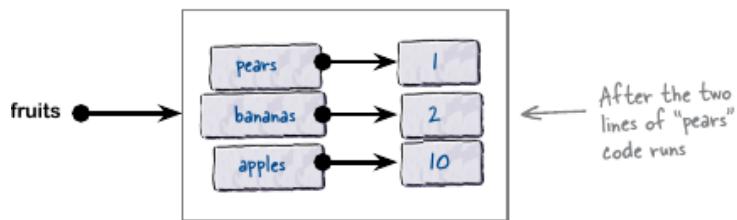
```

>>> if 'pears' not in fruits:
    fruits['pears'] = 0 ← Initialize (if needed).

>>> fruits['pears'] += 1 ← Increment
>>> fruits
{'bananas': 2, 'pears': 1, 'apples': 10}

```

Estas tres líneas de código han crecido el diccionario una vez más. Ahora hay tres pares clave / valor en el diccionario de frutas:

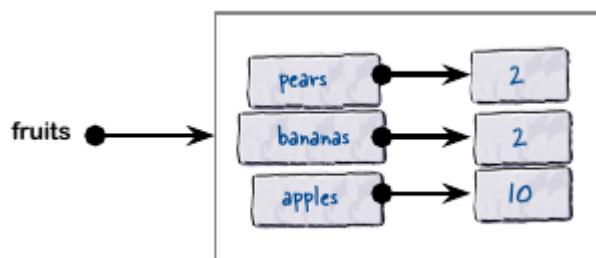


Las tres líneas de código anteriores son tan comunes en Python que el lenguaje proporciona un método de diccionario que hace esto **if/ no in** combinación más conveniente y menos propenso a errores. El método **setdefault** hace lo que hacen las instrucciones two-line **if/ not in**, pero usa sólo una sola línea de código.

Aquí está el equivalente del código de peras de la parte superior de la página reescrito para usar **setdefault**:

```
>>> fruits.setdefault('pears', 0) ← Initialize (if needed).
>>> fruits['pears'] += 1 ← Increment.
>>> fruits
{'bananas': 2, 'pears': 2, 'apples': 10}
```

La única llamada a **setdefault** ha reemplazado la instrucción two-line **if / not in**, y su uso garantiza que una clave siempre se inicializa a un valor inicial antes de que se use. Cualquier posibilidad de una excepción **KeyError** se anula. El estado actual del diccionario de frutas se muestra aquí abajo para confirmar que la invocación de **setdefault** después de que una clave ya existe no tiene ningún efecto (como es el caso de las peras), que es exactamente lo que queremos en este caso.



### Poner el método "setdefault" a trabajar

Recuerde que nuestra versión actual de **vowels5.py** resulta en un error de tiempo de ejecución, específicamente un **KeyError**, que se lanza debido a nuestro código que intenta acceder al valor de una clave inexistente:

This code produces this error.

```

vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")

found = {}

for letter in word:
    if letter in vowels:
        found[letter] += 1

for k, v in sorted(found.items()):
    print(k, 'was found', v, 'time(s).')

```

```

>>> ===== RESTART =====
>>>
Provide a word to search for vowels: hitchhiker
Traceback (most recent call last):
  File "/Users/Paul/Desktop/_NewBook/ch03/vowels5.py", line 9, in <module>
    found[letter] += 1
KeyError: 'i'
>>>

```

De nuestros experimentos con las frutas, sabemos que podemos llamar a `setdefault` tan a menudo como nos gusta sin tener que preocuparse por cualquier error desagradable. Sabemos que el comportamiento de `setdefault` está garantizado para inicializar una clave inexistente a un valor predeterminado suministrado, o para no hacer nada (es decir, para dejar cualquier valor existente asociado con cualquier clave existente). Si invocamos `setdefault` inmediatamente antes de tratar de usar una clave en nuestro código de `vowels5.py`, estamos garantizados para evitar un `KeyError`, ya que la clave existirá o no. De cualquier manera, nuestro programa sigue funcionando y ya no se bloquea (gracias a nuestro uso de `setdefault`).

En su ventana de edición IDLE, cambie el primero de los bucles del programa `vowels5.py` para que se parezca a esto (agregando la llamada a `setdefault`), luego guarde su nueva versión como `vowels6.py`:

**Utilice "setdefault" para evitar la excepción "KeyError".**

```

for letter in word:
    if letter in vowels:
        found.setdefault(letter, 0)
        found[letter] += 1

```

A single line of code can often make all the difference.

Con el programa `vowels6.py` más reciente en su ventana de edición IDLE, presione F5. Ejecute esta versión varias veces para confirmar que la desagradable excepción `KeyError` ya no aparece.

```
Python 3.4.3 Shell
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: hitch-hiker
e was found 1 time(s).
i was found 2 time(s).
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: life, the universe, and everything
a was found 1 time(s).
e was found 6 time(s).
i was found 3 time(s).
u was found 1 time(s).
>>> | Ln: 23 Col: 4
```

Esto se ve bien.  
El "KeyError"  
se ha ido.

El uso del método setdefault ha resuelto el problema de KeyError que tuvimos con nuestro código. El uso de esta técnica le permite crecer dinámicamente un diccionario en tiempo de ejecución, seguro sabiendo que sólo creará un nuevo par clave / valor cuando realmente lo necesite.

Cuando usa setdefault de esta manera, nunca necesita pasar tiempo inicializando todas sus filas de datos de diccionario antes de tiempo.

## *¿No son suficientes los diccionarios (y las listas)?*

Hemos estado hablando de estructuras de datos para las edades ... ¿cuánto más de esto está ahí? Seguramente los diccionarios, junto con las listas, son lo único que necesitaré la mayor parte del tiempo.?

### *Diccionarios (y listas) son grandes.*

Pero no son el único espectáculo en la ciudad.

Por supuesto, usted puede hacer mucho con diccionarios y listas, y muchos programadores de Python rara vez necesitan algo más. Pero, a decir verdad, estos programadores se están perdiendo, ya que las dos estructuras de datos integradas restantes -**conjunto** y **tupla**- son útiles en circunstancias específicas, y su uso puede simplificar grandemente su código, de nuevo en circunstancias específicas.

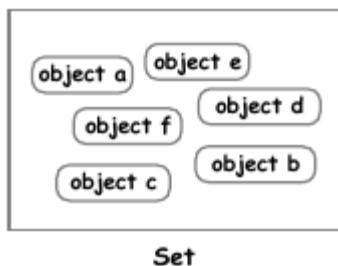
El truco está detectando cuando ocurren las circunstancias específicas. Para ayudar con esto, echemos un vistazo a los ejemplos típicos para el **conjunto** y la **tupla**, comenzando con el **conjunto**.

## **Sets Don't Allow Duplicates - Los conjuntos no permiten duplicados**

La estructura de datos conjuntos de Python es como los conjuntos que aprendió en la escuela: tiene ciertas propiedades matemáticas que siempre tienen, la característica clave es que los valores duplicados están prohibidos.

Imagine que se le proporciona una larga lista de todos los nombres para todos en una gran organización, pero sólo está interesado en la lista (mucho más pequeña) de nombres únicos. Necesita una forma rápida e infalible para eliminar cualquier duplicado de su larga lista de nombres. Los conjuntos son ideales para resolver este tipo de problemas: simplemente convierte la larga lista de nombres en un conjunto (que elimina los duplicados), luego convierte el conjunto en una lista y-ta da! -se tiene una lista de nombres únicos.

La estructura de datos de Python está optimizada para una búsqueda muy rápida, lo que hace que usar un conjunto sea mucho más rápido que su lista equivalente cuando la búsqueda es el requisito principal. Como las listas siempre realizan búsquedas secuenciales lentas, los conjuntos siempre deben ser preferidos para la búsqueda.



## **Spotting sets in your code – detectando conjuntos en su código**

Los juegos son fáciles de detectar en el código: una colección de objetos están separados unos de otros por comas y rodeados de llaves.

Por ejemplo, aquí hay un conjunto de vocales:

For example, here's a set of vowels:

```
>>> vowels = { 'a', 'e', 'e', 'i', 'o', 'u', 'u' }  
>>> vowels  
{'e', 'u', 'a', 'i', 'o'}
```

Check out the ordering.  
It's changed from what was originally inserted, and the duplicates are gone too.

Sets start and end with a curly brace.

Objects are separated from one another by a comma.

El hecho de que un conjunto está encerrado entre llaves puede a menudo resultar en su cerebro confundiendo un conjunto de un diccionario, que también está incluido en llaves. La diferencia clave es el uso del carácter de dos puntos (:) en los diccionarios para separar las claves de los valores. El colon o dos puntos nunca aparece en un conjunto, sólo comas.

Además de prohibir duplicados, tenga en cuenta que, como en un diccionario, el intérprete no mantiene el orden de inserción cuando se utiliza un conjunto. Sin embargo. Como todas las demás estructuras de datos, los conjuntos se pueden ordenar en salida con la función ordenada **sorted**. Y, al igual que listas y diccionarios, los conjuntos también pueden crecer y reducirse según sea necesario.

Al ser un conjunto, esta estructura de datos puede realizar operaciones de tipo conjunto, como diferencia, intersección y unión. Para demostrar los conjuntos en acción, vamos a revisar nuestro programa de conteo de vocales anteriormente en este capítulo una vez más. Hicimos una promesa cuando estábamos desarrollando **vowels3.py** (en el último capítulo) que consideraríamos un conjunto sobre una lista como la estructura de datos primaria para ese programa. Vamos a cumplir esta promesa ahora.

### ***Creating Sets Efficiently***

#### ***Creación de conjuntos de manera eficiente***

Vamos a echar un vistazo a **vowels3.py**, que utiliza una lista para averiguar qué vocales aparecen en cualquier palabra.

Aquí está el código una vez más. Tenga en cuenta cómo tenemos la lógica en este programa para asegurarnos de que solo recordamos cada vocal encontrada una vez. Es decir, estamos muy deliberadamente asegurando que no se añadan nunca vocales duplicadas a la lista encontrada:

This is "vowels3.py", which reports on the unique vowels found in a word. This code uses a list as its primary data structure.

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")
found = []
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
for vowel in found:
    print(vowel)
```

We never allow duplicates in the "found" list.

[Ln: 11 Col: 0]

Antes de continuar, utilice IDLE para guardar este código como `vowels7.py` para que podamos realizar cambios sin tener que preocuparnos por romper nuestra solución basada en listas (que sabemos que funciona). Como se está convirtiendo en nuestra práctica estándar, vamos a experimentar en el prompt `>>>` primero antes de ajustar el código de `vowels7.py`. Editaremos el código en la ventana de edición de IDLE una vez que hayamos calculado el código que necesitamos.

### **Creación de conjuntos de secuencias**

Comenzamos creando un conjunto de vocales usando el código de la mitad de la última página (puedes saltar este paso si ya has escrito ese código en tu prompt `>>>`):

```
>>> vowels = { 'a', 'e', 'e', 'i', 'o', 'u', 'u' } ←  
>>> vowels  
{'e', 'u', 'a', 'i', 'o'}  
  
Below is a useful shorthand that allows you to pass any sequence (such as a string)  
to the set function to quickly generate a set. Here's how to create the set of  
vowels using the set function:  
  
>>> vowels2 = set('aeiouuu') ←  
>>> vowels2  
{'e', 'u', 'a', 'i', 'o'}  
  
These two lines of code  
do the same thing: both  
assign a new set object to  
a variable.  
  
A continuación se muestra una abreviatura útil que le permite pasar cualquier secuencia (como una cadena) a la función set para generar rápidamente un conjunto. A continuación se muestra cómo crear el conjunto de vocales utilizando la función set:  
  
Estas dos líneas de código hacen lo mismo: asignan un nuevo objeto set a una variable.
```

### **Aprovechando los métodos de conjuntos**

Ahora que tenemos nuestras vocales en un conjunto, nuestro siguiente paso es tomar una palabra y determinar si alguna de las letras de la palabra son vocales. Podríamos hacer esto comprobando si cada letra de la palabra está en el conjunto, ya que el operador `in` trabaja con conjuntos de la misma manera que lo hace con diccionarios y listas. Es decir, podríamos utilizar para determinar si un conjunto contiene alguna letra, y luego recorrer las letras de la palabra usando un bucle `for`.

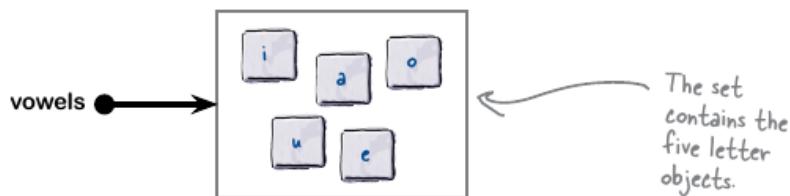
Sin embargo, no sigamos esa estrategia aquí, ya que los métodos `set` pueden hacer mucho de este trabajo de bucle para nosotros.

Hay una manera mucho mejor de realizar este tipo de operación cuando se utilizan conjuntos. Se trata de aprovechar los métodos que vienen con cada conjunto, y que le permiten realizar operaciones como la unión, la diferencia y la intersección. Antes de

cambiar el código en `vowels7.py`, vamos a aprender cómo funcionan estos métodos experimentando en el prompt `>>>` y considerando cómo el intérprete ve los datos establecidos. Asegúrese de seguir a lo largo de su computadora. Comencemos creando un conjunto de vocales, luego asignando un valor a la variable de palabra:

```
>>> vowels = set('aeiou')
>>> word = 'hello'
```

El intérprete crea dos objetos: un conjunto y una cadena. Esto es lo que parece el conjunto de vocales en la memoria del intérprete:



Veamos qué sucede cuando realizamos una unión de las vocales definidas y el conjunto de letras creado a partir del valor en la variable palabra. Vamos a crear un segundo conjunto sobre la marcha, pasando la variable de la palabra a la función de conjunto, que luego se pasa al método de unión proporcionado por las vocales. El resultado de esta llamada es otro conjunto, que asignamos a otra variable (llamada `u` aquí). Esta nueva variable es una combinación de los objetos en ambos conjuntos (una unión):

```
>>> u = vowels.union(set(word))
```

Python convierte el valor de "word" en un conjunto de objetos de letra (eliminando cualquier duplicado a medida que lo hace).

The "union" method combines one set with another, which is then assigned to a new variable called "u" (which is another set).

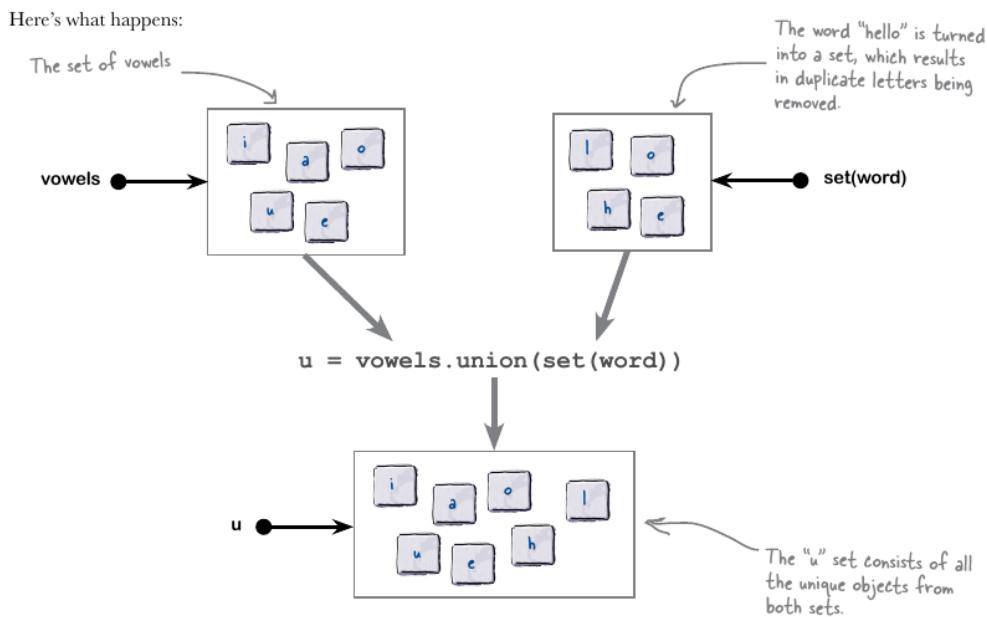
El método "union" combina un conjunto con otro, que luego se asigna a una nueva variable llamada "u" (que es otro conjunto).

**Después de esta llamada al método de unión, ¿qué aspecto tienen las vocales y los conjuntos de `u`?**

## ***union Works by Combining Sets - Trabajos de unión por la combinación de conjuntos***

En la parte inferior de la página anterior se usó el método `union` para crear un nuevo conjunto llamado `u`, que era una combinación de las letras de las vocales definidas junto con el conjunto de letras únicas en `word`. El acto de crear este nuevo conjunto no tiene ningún impacto en las vocales, que permanece como estaba antes de la unión. Sin embargo, el conjunto `u` es nuevo, ya que se crea como resultado de la unión.

Sin embargo, el conjunto `u` es nuevo, ya que se crea como resultado de la unión.  
Esto es lo que sucede:



### ***¿Qué pasó con el código de bucle?***

Esa única línea de código empaqueta mucho. Tenga en cuenta que no ha instruido específicamente al intérprete para realizar un bucle. En cambio, le dijiste al intérprete lo que querías hacer, no cómo lo querías hacer, y el intérprete ha obligado a crear un nuevo conjunto que contenga los objetos que buscas.

Un requisito común (ahora que hemos creado la unión) es convertir el conjunto resultante en una lista ordenada. Hacerlo es trivial, gracias a las funciones `listadas` y `de lista`:

Una lista ordenada de letras únicas

```
>>> u_list = sorted(list(u))
>>> u_list
['a', 'e', 'h', 'i', 'l', 'o', 'u']
```



## ***difference Tells You What's Not Shared***

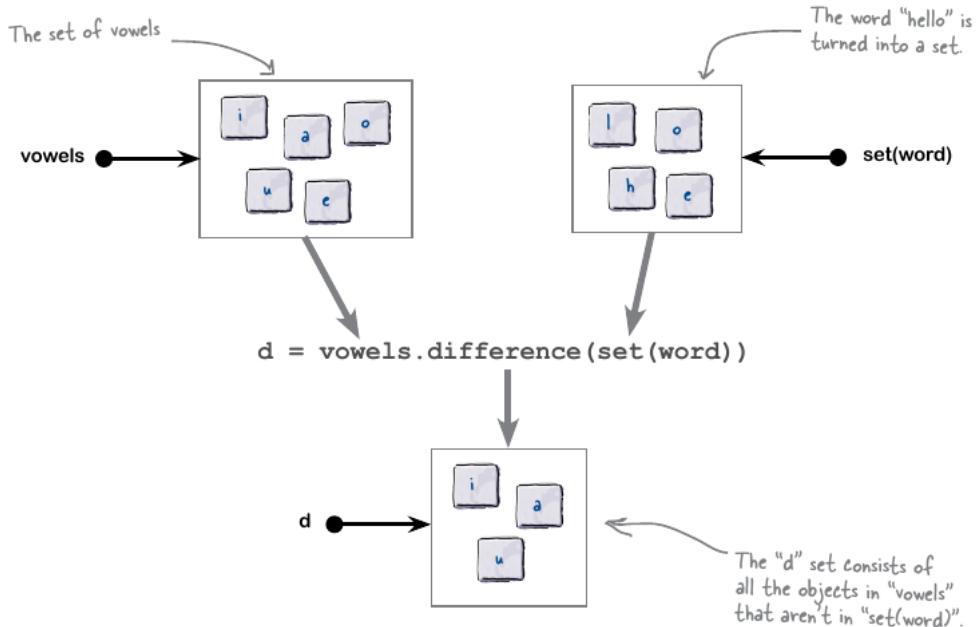
### ***Diferencia te dice lo que no está compartido***

Otro método de conjunto es la diferencia, que, dado dos conjuntos, puede decir lo que está en un conjunto, pero no el otro. Utilicemos la diferencia de la misma forma que hicimos con la unión y veamos con qué terminamos:

```
>>> d = vowels.difference(set(word))
>>> d
{'u', 'i', 'a'}
```

La función diferencia compara los objetos de las vocales con los objetos del conjunto (word), luego devuelve un nuevo conjunto de objetos (llamados de aquí) que están en las vocales definidas pero no en el conjunto (word).

Esto es lo que sucede:



Una vez más, llamamos la atención sobre el hecho de que este resultado se ha logrado sin usar un bucle for. La función de diferencia hace todo el trabajo de gruñido aquí; Todo lo que hicimos fue declarar lo que se requería.

Vuelve a la página siguiente para ver un método de conjunto final: **intersección**.

### **Intersección Informes sobre la comunalidad**

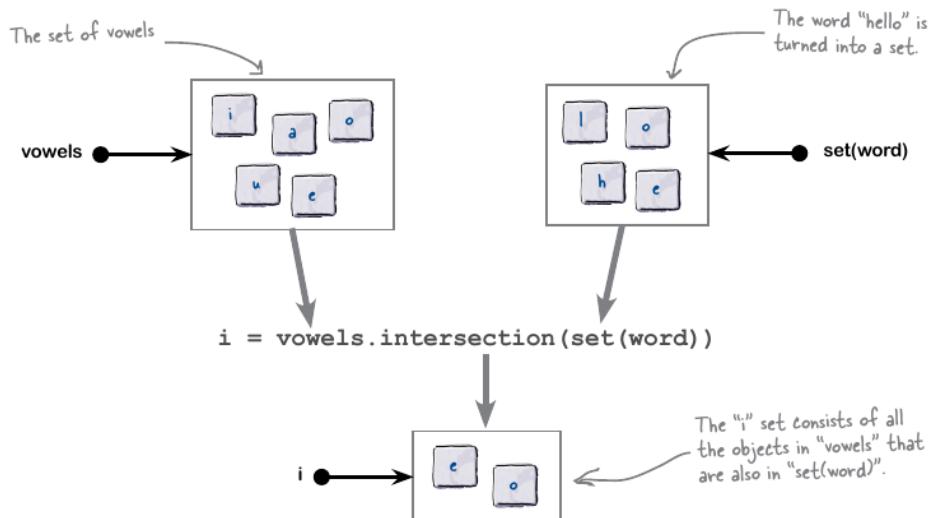
El tercer método de conjunto que vamos a ver es la intersección, que toma los objetos en un conjunto y los compara con los de otro, a continuación, informes sobre los objetos comunes encontrados.

En relación con los requisitos que tenemos con vowels7.py, lo que el método de intersección suena muy prometedor, como queremos saber cuál de las letras en la palabra del usuario son vocales.

Recordemos que tenemos la cadena "hola" en la variable palabra, y nuestras vocales en las vocales establecidas. Aquí está el método de intersección en acción:

```
>>> i = vowels.intersection(set(word))  
>>> i  
{'e', 'o'}
```

El método de intersección confirma que las vocales e y o están en la variable palabra. Esto es lo que sucede:



Hay más métodos establecidos que los tres que hemos examinado en estas últimas páginas, pero de los tres, la intersección es de mayor interés para nosotros aquí. En una sola línea de código, hemos resuelto el problema que planteamos cerca del comienzo del último capítulo: identificar las vocales en cualquier cadena. Y todo sin tener que usar ningún código de bucle. Volvamos al programa vowels7.py y aplicamos lo que sabemos ahora.

Aquí está el código del programa vowels3.py una vez más. Basado en lo que ahora sabes sobre los sets, agarra tu lápiz y saca el código que ya no necesitas. En el espacio proporcionado a la derecha, proporcione el código que añadiría para convertir este

programa utilizando la lista para aprovechar un conjunto. Sugerencia: terminará con mucho menos código.

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")
found = []
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
for vowel in found:
    print(vowel)
```

Cuando haya terminado, asegúrese de cambiar el nombre de su archivo `vowels7.py`.

### Solución:

The diagram illustrates the refactoring of the vowel search code. It starts with the original code on the left, which uses a list of vowels and iterates through each character of the input word to check if it's a vowel. Handwritten notes above this code say "There's lots of code to get rid of." and "Hint: you'll end up with a lot less code." An arrow points from the original code to a simplified version on the right. This simplified version creates a set of vowels and then finds the intersection of that set with the input word, resulting in a single-line solution. Handwritten notes for this part say "Create a set of vowels." and "These five lines of list-processing code are replaced by a single line of set code."

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")
found = []
for letter in word:
    if letter in vowels:
        if letter not in found:
            found.append(letter)
for vowel in found:
    print(vowel)
```

```
vowels = set('aeiou')
found = vowels.intersection(set(word))
for vowel in found:
    print(vowel)
```

*Me siento engañado ... todo ese tiempo perdido aprendiendo sobre listas y diccionarios, y la mejor solución a este problema de las vocales fue usar un conjunto? ¿Seríamente?*

**No fue una pérdida de tiempo.**

Ser capaz de detectar cuando se utiliza una estructura de datos incorporada sobre otra es importante (ya que usted querrá estar seguro de que está escogiendo la correcta). La única manera que puedes hacer esto es conseguir experiencia usando todos ellos. Ninguna de las estructuras de datos incorporadas califica como una tecnología de "talla única", ya que

todas tienen sus fortalezas y debilidades. Una vez que comprenda lo que son, estará mejor preparado para seleccionar la estructura de datos correcta en función de los requisitos de datos específicos de su aplicación.

Vamos a tomar **vowels7.py** para un giro para confirmar que la versión basada en set de nuestro programa se ejecuta como se esperaba:

```
vowels = set('aeiou')
word = input("Provide a word to search for vowels: ")
found = vowels.intersection(set(word))
for vowel in found:
    print(vowel)
```

```
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: hitch-hiker
e
i
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: Galaxy
a
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: life, the universe, and everything
i
a
u
e
>>> ===== RESTART =====
>>>
Provide a word to search for vowels: sky
```

Todo está funcionando como se esperaba.

### **Usando un conjunto fue la elección perfecta aquí ...**

Pero eso no quiere decir que las otras dos estructuras de datos no tengan sus usos. Por ejemplo, si necesita realizar, digamos, un recuento de frecuencia, el diccionario de Python funciona mejor. Sin embargo, si usted está más preocupado con el mantenimiento de orden de inserción, entonces sólo una lista hará ... que es casi cierto. Hay otra estructura de datos incorporada que mantiene el orden de inserción y que aún no hemos discutido: la **tupla**.

Vamos a pasar el resto de este capítulo en compañía de la tupla de Python.

## Making the Case for Tuples - Cómo hacer valer las tuplas

Cuando la mayoría de los programadores nuevos en Python primero se encuentran con la tupla, se preguntan por qué existe tal estructura de datos. Después de todo, una tupla es como una lista que no se puede cambiar una vez que se crea (y se llenan con datos). Las tuplas son inmutables: no pueden cambiar. Entonces, ¿por qué los necesitamos?

Resulta que tener una estructura de datos inmutable a menudo puede ser útil. Imagine que necesita protegerse contra los efectos secundarios asegurando que algunos datos de su programa nunca cambien. O tal vez usted tiene una gran lista constante (que usted sabe que no va a cambiar) y usted está preocupado por el rendimiento. ¿Por qué incurrir en el costo de todo ese código de procesamiento de lista extra (mutable) si nunca va a necesitarlo? El uso de una tupla en estos casos evita la sobrecarga innecesaria y protege contra los efectos secundarios desagradables de los datos (si se produjeran).

### How to spot a tuple in code

#### Cómo detectar una tupla en el código

Como las tuplas están estrechamente relacionadas con las listas, no es de extrañar que se parezcan similares (y se comportan de manera similar.) Las tuplas están rodeadas de paréntesis, mientras que las listas usan corchetes. Una rápida visita al indicador nos permite comparar tuplas con listas. Observe cómo estamos utilizando la función integrada de `type` para confirmar el tipo de cada objeto creado:

```
>>> vowels = [ 'a', 'e', 'i', 'o', 'u' ]
>>> type(vowels)
<class 'list'>
>>> vowels2 = ( 'a', 'e', 'i', 'o', 'u' )
>>> type(vowels2)
<class 'tuple'>
```

There's nothing new here. A list of vowels is created.

The "type" built-in function reports the type of any object.

Esta tupla se parece a una lista, pero no lo es. Las tuplas están rodeadas por paréntesis (no corchetes).

Ahora que existen **vocales** y **vocales2** (y se llenan con datos), podemos pedirle al shell que muestre lo que contienen. Al hacerlo, se confirma que la tupla no es exactamente la misma que la lista:

```
>>> vowels
['a', 'e', 'i', 'o', 'u']
>>> vowels2
('a', 'e', 'i', 'o', 'u') ←
The
parentheses
indicate that
this is a tuple.
```

Pero, ¿qué pasa si intentamos cambiar una tupla?

## Tuples Are Immutable

Como las tuplas son una especie de listas, apoyan la misma nota de corchete comúnmente asociada con las listas. Ya sabemos que podemos usar esta notación para cambiar el contenido de una lista. Esto es lo que haríamos para cambiar la letra minúscula i en la lista de vocales para que sea una I mayúscula:

```
>>> vowels[2] = 'I'  
>>> vowels  
['a', 'e', 'I', 'o', 'u']
```

Assign an uppercase "I"  
to the third element  
of the "vowels" list.

Como se esperaba, el tercer elemento de la lista (en la ubicación del índice 2) ha cambiado, lo cual está bien y se espera, ya que las listas son mutables. Sin embargo, mira lo que sucede si tratamos de hacer lo mismo con la vocal de las vocales2:

```
>>> vowels2[2] = 'I'  
Traceback (most recent call last):  
  File "<pyshell#16>", line 1, in <module>  
    vowels2[2] = 'I'  
TypeError: 'tuple' object does not support item assignment  
>>> vowels2  
('a', 'e', 'i', 'o', 'u')
```

El intérprete se queja en voz alta si intenta  
cambiar una tupla.

No change here, as tuples  
are immutable

Un Tuples son inmutables, por lo que no podemos quejarnos cuando el intérprete proteste en nuestro intentar cambiar los objetos almacenados en la tupla. Después de todo, ese es el punto completo de una tupla: una vez creado y poblado con datos, una tupla no puede cambiar.

No se equivoquen: este comportamiento es útil, especialmente cuando es necesario asegurarse de que algunos datos no pueden cambiar. La única manera de asegurar esto es poner los datos en una tupla, que entonces ordena al intérprete para detener cualquier código de intentar cambiar los datos de la tupla.

A medida que avanzamos en el resto de este libro, usaremos siempre tuplas cuando tenga sentido hacerlo. Con referencia al código de procesamiento de vocales, debería quedar claro ahora que la estructura de datos de las vocales debe estar siempre almacenada en una tupla en oposición a una lista, ya que no tiene sentido usar una estructura de datos mutable en este caso (como los cinco Las vocales nunca necesitan cambiar).

No hay mucho más que las tuplas piensen en ellas como listas inmutables, nada más. Sin embargo, hay un uso que los viajes de muchos programadores, así que vamos a aprender lo que esto es para que pueda evitarlo.

**Si los datos de su estructura nunca cambian, póngalos en una tupla.**

### Una advertencia de tupla

## Cuidado con las Tuplas de Un Solo Objeto

Imaginemos que desea almacenar una sola cadena en una tupla. Es tentador poner la cadena entre paréntesis y luego asignarla a un nombre de variable ... pero al hacerlo no se produce el resultado esperado.

Echa un vistazo a esta interacción con el prompt >>>, lo que demuestra lo que sucede cuando haces esto:

```
>>> t = ('Python')
>>> type(t)
<class 'str'>
>>> t
'Python'
```

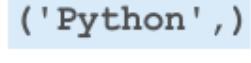


This is not what we expected. We've ended up with a string. What happened to our tuple?

Esto no es lo que esperábamos. Terminamos con una cadena. ¿Qué pasó con nuestra tupla?

Lo que parece una tupla de un solo objeto no lo es; Es una cadena. Esto ha sucedido debido a una peculiaridad sintáctica en el lenguaje Python. La regla es que, para que una tupla sea una tupla, cada tupla debe incluir al menos una coma entre los paréntesis, incluso cuando la tupla contiene un solo objeto. Esta regla significa que para asignar un solo objeto a una tupla (estamos asignando un objeto de cadena en esta instancia), necesitamos incluir la coma de arrastre, así:

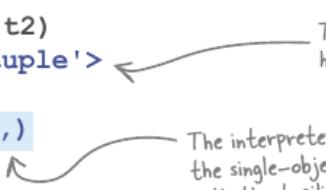
```
>>> t2 = ('Python',)
```



Esta coma trailing hace toda la diferencia, ya que le dice al intérprete que esto es una tupla.

Esto parece un poco raro, pero no dejes que te preocupe. Sólo recuerda esta regla y estarás bien: cada tupla debe incluir al menos una coma entre los paréntesis. Cuando usted ahora pide que el intérprete le diga qué tipo t2 es (así como exhibe su valor), usted aprende que t2 es una tupla, que es qué se espera:

```
>>> type(t2)
<class 'tuple'>
>>> t2
('Python',)
```



That's better: we now have a tuple.  
Eso es mejor: ahora tenemos una tupla.

The interpreter displays the single-object tuple with the trailing comma.  
El intérprete muestra la tupla de objeto único con la coma al final.

Es bastante común que las funciones acepten y devuelvan sus argumentos como una tupla, incluso cuando aceptan o devuelven un solo objeto. Por lo tanto, se encontrará con esta sintaxis a menudo cuando se trabaja con funciones. Tendremos más que decir acerca de la relación entre funciones y tuplas en un poco; De hecho, dedicaremos el siguiente capítulo a las funciones (así no tendrás que esperar mucho tiempo).

Ahora que conoce las cuatro estructuras de datos incorporadas, y antes de llegar al capítulo de funciones, tomemos un pequeño desvío y exprimiremos un ejemplo corto y divertido de una estructura de datos más compleja.

## ***Combining the Built-in Data Structures***

### ***Combinación de las estructuras de datos integradas***

*Toda esta charla de estructuras de datos me hace preguntarme si las cosas pueden llegar a ser más complejas. En concreto, ¿puedo almacenar un diccionario en un diccionario?*

#### ***Esta pregunta se hace mucho.***

Una vez que los programadores se acostumbran a almacenar números, cadenas y booleanos en listas y diccionarios, se gradúan muy rápidamente para preguntarse si el soporte incorporado admite datos más complejos. Es decir, ¿pueden las estructuras de datos incorporadas almacenar estructuras de datos integradas?

La respuesta es sí, y la razón de esto es así es debido al hecho de que todo es un objeto en Python.

Todo lo que hemos almacenado hasta ahora en cada uno de los elementos integrados ha sido un objeto. El hecho de que hayan sido "objetos simples" (como números y cadenas) no importa, ya que los built-ins pueden almacenar cualquier objeto. Todas las incorporadas a pesar de ser "complejas") son objetos, también, por lo que puede mezclar y coincidir en la forma que elija. Simplemente asigne la estructura de datos integrada como lo haría con un objeto simple, y estará de oro.

Veamos un ejemplo que usa un diccionario de diccionarios.

**P:** ¿Lo que está a punto de hacer sólo funciona con diccionarios? ¿Puedo tener una lista de listas, o un conjunto de listas, o una tupla de diccionarios?

**R:** Sí, usted puede. Demostramos cómo funciona un diccionario de diccionarios, pero puede combinar los elementos integrados en la forma que elija.

#### ***Una tabla mutable***

## ***Almacenamiento de una tabla de datos***

Como todo es un objeto, cualquiera de las estructuras de datos incorporadas puede almacenarse en cualquier otra estructura de datos incorporada, permitiendo la construcción de estructuras de datos arbitrariamente complejas ... sujeto a la capacidad de su cerebro para visualizar realmente lo que está pasando. Por ejemplo, aunque un diccionario de listas que contienen tuplas que contienen conjuntos de diccionarios podría sonar como una buena idea, puede que no lo sea, ya que su complejidad está fuera de escala.

Una estructura compleja que surge mucho es un diccionario de diccionarios. Esta estructura se puede utilizar para crear una tabla mutable. Para ilustrar, imaginemos que tenemos esta tabla describiendo una colección heterogénea de personajes:

Name	Gender	Occupation	Home Planet
Ford Prefect	Male	Researcher	Betelgeuse Seven
Arthur Dent	Male	Sandwich-Maker	Earth
Tricia McMillan	Female	Mathematician	Earth
Marvin	Unknown	Paranoid Android	Unknown

Recall how, at the start of this chapter, we created a dictionary called `person3` to store Ford Prefect's data:

```
person3 = { 'Name': 'Ford Prefect',
            'Gender': 'Male',
            'Occupation': 'Researcher',
            'Home Planet': 'Betelgeuse Seven' }
```

Recuerde cómo, al comienzo de este capítulo, creamos un diccionario denominado `person3` para almacenar los datos de Ford Prefect:

En lugar de crear (y luchar con) cuatro variables de diccionario individuales para cada línea de datos en nuestra tabla, creemos una única variable de diccionario, llamada **gente**. A continuación, utilizaremos la gente para almacenar cualquier número de otros diccionarios. Para empezar, primero creamos un diccionario de personas o `people` vacías, luego asignamos los datos de Ford Prefect a una clave:

```
>>> people = {}           Start with a new, empty dictionary.
>>> people['Ford'] = { 'Name': 'Ford Prefect',
                        'Gender': 'Male',
                        'Occupation': 'Researcher',
                        'Home Planet': 'Betelgeuse Seven' }

The key is "Ford",
and the value is
another dictionary.
```

## *A Dictionary Containing a Dictionary*

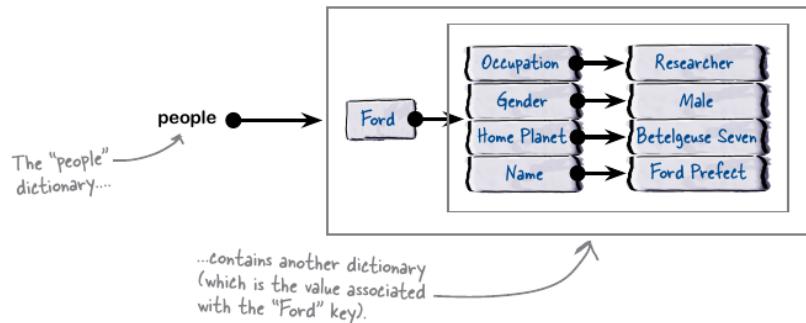
### **Diccionario que contiene un diccionario**

Con el diccionario de people creado y una fila de datos agregados (de Ford), podemos pedirle al intérprete que muestre el diccionario de people en el prompt >>. La salida resultante parece un poco confusa, pero todos nuestros datos están allí:

```
>>> people
{'Ford': {'Occupation': 'Researcher', 'Gender': 'Male',
'Home Planet': 'Betelgeuse Seven', 'Name': 'Ford Prefect'}}
```

Un diccionario incrustado en un diccionario-necesita los rizadores adicionales.

Sólo hay un diccionario incrustado en people (por el momento), por lo que llamar a este un "diccionario de diccionarios" es un poco un tramo, ya que la gente contiene sólo uno ahora. Esto es lo que a people parece al intérprete:



... contiene otro diccionario (que es el valor asociado con la tecla "Ford").

Ahora podemos proceder a agregar los datos de las otras tres filas de nuestra tabla:

```
>>> people['Arthur'] = { 'Name': 'Arthur Dent',
                           'Gender': 'Male',
                           'Occupation': 'Sandwich-Maker',
                           'Home Planet': 'Earth' }
>>> people['Trillian'] = { 'Name': 'Tricia McMillan',
                            'Gender': 'Female',
                            'Occupation': 'Mathematician',
                            'Home Planet': 'Earth' }
>>> people['Robot'] = { 'Name': 'Marvin',
                         'Gender': 'Unknown',
                         'Occupation': 'Paranoid Android',
                         'Home Planet': 'Unknown' }
```

Los datos de Marvin están asociados con la clave "Robot".

Arthur's data

Los datos de Tricia están asociados con el Trillian ".

**Solo son datos**

## **A Dictionary of Dictionaries (a.k.a. a Table)**

### **Un diccionario de diccionarios (también conocido como una tabla)**

Con el diccionario de personas con cuatro diccionarios incrustados, podemos pedirle al intérprete que muestre el diccionario de personas en el prompt >>>.

Si lo hace, se producirá un desorden profano de datos en la pantalla (véase más abajo).

A pesar del desastre, todos nuestros datos están allí. Tenga en cuenta que cada abrazadera de apertura abre un nuevo diccionario, mientras que una abrazadera de cierre termina un diccionario. Adelante y cuente (hay cinco de cada uno):

```
>>> people
{'Ford': {'Occupation': 'Researcher', 'Gender': 'Male',
'Home Planet': 'Betelgeuse Seven', 'Name': 'Ford Prefect'},
'Trillian': {'Occupation': 'Mathematician', 'Gender':
'Female', 'Home Planet': 'Earth', 'Name': 'Tricia
McMillan'}, 'Robot': {'Occupation': 'Paranoid Android',
'Gender': 'Unknown', 'Home Planet': 'Unknown', 'Name':
'Marvin'}, 'Arthur': {'Occupation': 'Sandwich-Maker',
'Gender': 'Male', 'Home Planet': 'Earth', 'Name': 'Arthur
Dent'}}
```

Es un poco difícil de leer, pero todos los datos están ahí.



*El intérprete deposita los datos en la pantalla. ¿Alguna posibilidad de que podamos hacer esto más presentable?*

**Sí, podemos hacer esto más fácil de leer.**

Podríamos pasar al prompt >>> y codificar un bucle rápido que podría iterar sobre cada una de las claves del diccionario de personas. Al hacer esto, un bucle anidado podría procesar cada uno de los diccionarios incrustados, asegurándose de mostrar algo más fácil de leer en la pantalla.

Podríamos ... pero no lo vamos a hacer, ya que alguien más ya ha hecho este trabajo para nosotros.

## **Pretty-Printing Complex Data Structures**

### **Bonita Impresión de Estructuras de Datos Complejos**

La biblioteca estándar incluye un módulo llamado **pprint** que puede tomar cualquier estructura de datos y mostrarla en un formato más fácil de leer. El nombre **pprint** es una abreviatura de "pretty print".

Vamos a utilizar el módulo pprint con nuestro diccionario de personas (de diccionarios). A continuación, una vez más mostrar los datos "en bruto" en el prompt >>>, y luego importar el módulo pprint antes de invocar su función pprint para producir la salida que necesitamos:

```
>>> people
{'Ford': {'Occupation': 'Researcher', 'Gender': 'Male',
'Home Planet': 'Betelgeuse Seven', 'Name': 'Ford Prefect'},
'Trillian': {'Occupation': 'Mathematician', 'Gender':
'Female', 'Home Planet': 'Earth', 'Name': 'Tricia
McMillan'}, 'Robot': {'Occupation': 'Paranoid Android',
'Gender': 'Unknown', 'Home Planet': 'Unknown', 'Name':
'Marvin'}, 'Arthur': {'Occupation': 'Sandwich-Maker',
'Gender': 'Male', 'Home Planet': 'Earth', 'Name': 'Arthur
Dent'}}
>>>
>>> import pprint
>>> pprint pprint(people)
{'Arthur': {'Gender': 'Male',
'Home Planet': 'Earth',
'Name': 'Arthur Dent',
'Occupation': 'Sandwich-Maker'},
'Ford': {'Gender': 'Male',
'Home Planet': 'Betelgeuse Seven',
'Name': 'Ford Prefect',
'Occupation': 'Researcher'},
'Robot': {'Gender': 'Unknown',
'Home Planet': 'Unknown',
'Name': 'Marvin',
'Occupation': 'Paranoid Android'},
'Trillian': {'Gender': 'Female',
'Home Planet': 'Earth',
'Name': 'Tricia McMillan',
'Occupation': 'Mathematician'}}}
```

Nuestro diccionario de diccionarios es difícil de leer.

Importe el módulo "pprint", luego invoque la función "pprint" para realizar el trabajo.

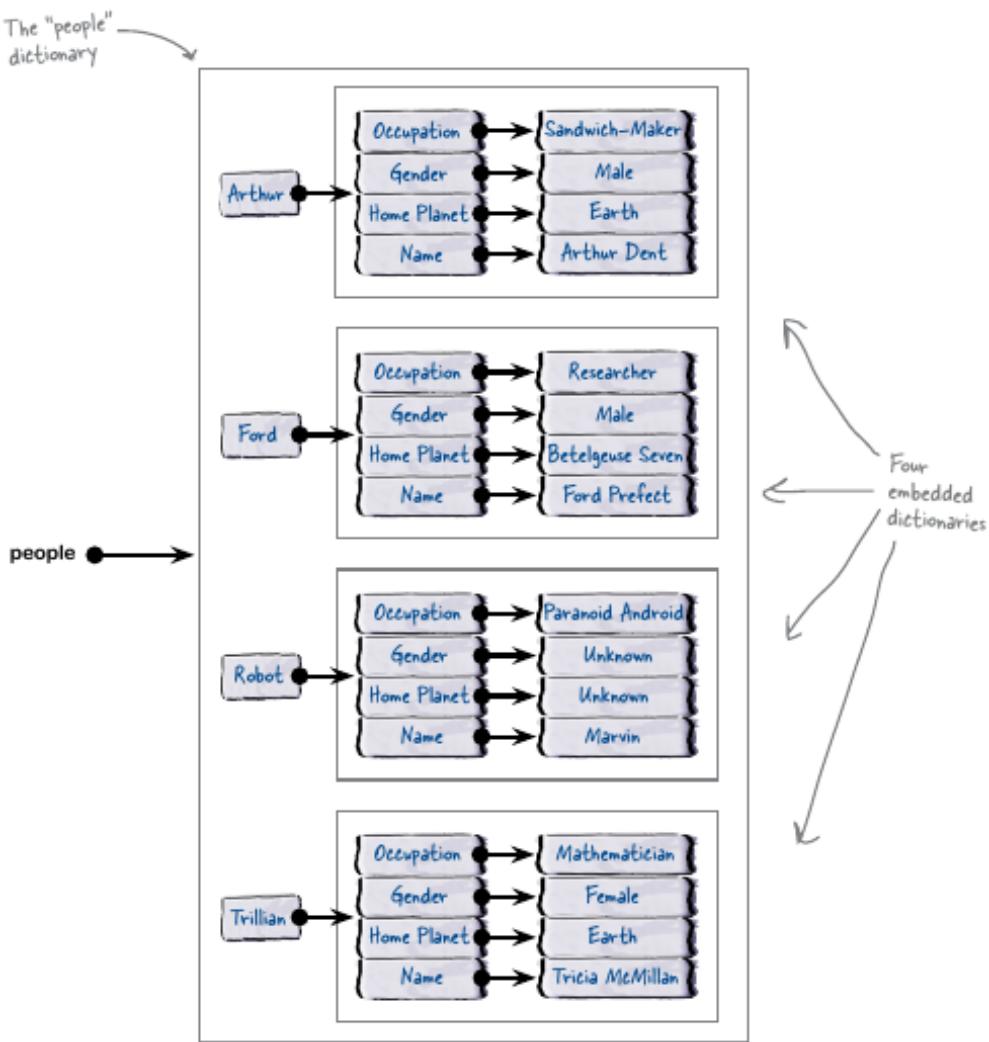
This output is much easier on the eye. Note that we still have five opening and five closing curly braces. It's just that—thanks to "pprint"—they are now so much easier to see (and count).

Esta salida es mucho más fácil en el ojo. Tenga en cuenta que todavía tenemos cinco orificios de apertura y cinco de cierre. Es sólo que gracias a "pprint" - que ahora son mucho más fáciles de ver (y contar).

*como luce*

## **Visualización de estructuras de datos complejas**

Vamos a actualizar nuestro diagrama que representa lo que el intérprete ahora "ve" cuando el diccionario de personas de diccionarios se rellena con datos:



En este punto, una pregunta razonable a hacer es: Ahora que tenemos todos estos datos almacenados en un diccionario de diccionarios, **¿cómo podemos llegar a él? Responda a esta pregunta en la página siguiente.**

## **Accessing a Complex Data Structure's Data**

### **Acceso a datos complejos de estructuras de datos**

Ahora tenemos nuestra tabla de datos almacenada en el diccionario de personas. Recordemos lo que parecía la tabla original de datos:

Name	Gender	Occupation	Home Planet
Ford Prefect	Male	Researcher	Betelgeuse Seven
Arthur Dent	Male	Sandwich-Maker	Earth
Tricia McMillan	Female	Mathematician	Earth
Marvin	Unknown	Paranoid Android	Unknown

Si se nos pidiera que averiguara lo que hace Arthur, comenzaríamos por mirar la columna **Nombre** para el nombre de Arthur y luego miraríamos la fila de datos hasta que llegáramos a la columna **Ocupación**, donde podríamos Para leer "Sandwich-Maker".

Cuando se trata de acceder a datos en una estructura de datos compleja (como nuestro diccionario de diccionarios de diccionarios), podemos seguir un proceso similar, que ahora vamos a demostrar en el prompt >>>.

Comenzamos por encontrar los datos de Arturo en el diccionario de people lo que podemos hacer poniendo la clave de Arturo entre corchetes:

Pregunte por la fila de datos de Arthur.

```
>>> people['Arthur']
{'Occupation': 'Sandwich-Maker', 'Home Planet': 'Earth',
 'Gender': 'Male', 'Name': 'Arthur Dent'}
```

La fila de datos de diccionario asociados con la clave "Arturo"

Después de haber encontrado la fila de datos de Arthur, ahora podemos pedir el valor asociado con la clave de ocupación. Para ello, empleamos un segundo par de corchetes para indexar el diccionario de Arthur y acceder a los datos que buscamos:

Identify the row. →  
 Identify the column. ↘

```
>>> people['Arthur']['Occupation']
'Sandwich-Maker'
```

El uso de corchetes dobles le permite acceder a cualquier valor de datos de una tabla identificando la fila y la columna que le interesa. La fila corresponde a una clave utilizada por el diccionario adjunto (personas, en nuestro ejemplo), mientras que la columna corresponde a cualquiera de Las claves utilizadas por un diccionario incrustado.

## ***Los datos son tan complejos como usted lo hace***

Ya sea que tenga una pequeña cantidad de datos (una lista simple) o algo más complejo (un diccionario de diccionarios), es bueno saber que las cuatro estructuras de datos integradas de Python pueden acomodar sus necesidades de datos. Lo que es especialmente agradable es la naturaleza dinámica de las estructuras de datos que construye; Aparte de las tuplas, cada una de las estructuras de datos puede crecer y contraerse según sea necesario, con el intérprete de Python cuidando cualquier asignación de memoria / detalles de asignación para usted.

Todavía no hemos terminado con los datos, y volveremos a este tema más adelante en este libro. Por ahora, sin embargo, sabes lo suficiente como para seguir adelante con las cosas.

En el próximo capítulo, comenzamos a hablar sobre técnicas para reutilizar eficazmente el código con Python, aprendiendo acerca de las tecnologías de reutilización de código más básicas: funciones.

## ***Chapter 3's Code, 1 of 2***

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")

found = {}

found['a'] = 0
found['e'] = 0
found['i'] = 0
found['o'] = 0
found['u'] = 0

for letter in word:
    if letter in vowels:
        found[letter] += 1

for k, v in sorted(found.items()):
    print(k, 'was found', v, 'time(s).')
```

This is the code for "vowels4.py", which performed a frequency count. This code was (loosely) based on "vowels3.py", which we first saw in Chapter 2.

Este es el código para "vowels4.py", que realizó un recuento de frecuencia. Este código se basaba (vagamente) en "vowels3.py", que vimos por primera vez en el capítulo 2.

In an attempt to remove the dictionary initialization code, we created "vowels5.py", which crashed with a runtime error (due to us failing to initialize the frequency counts).

```
vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")

found = {}

for letter in word:
    if letter in vowels:
        found[letter] += 1

for k, v in sorted(found.items()):
    print(k, 'was found', v, 'time(s).')
```

En un intento de eliminar el código de inicialización del diccionario, creamos "vowels5.py", que se estrelló con un error de tiempo de ejecución (debido a que no podemos inicializar los recuentos de frecuencia).

```

vowels = ['a', 'e', 'i', 'o', 'u']
word = input("Provide a word to search for vowels: ")

found = {}

for letter in word:
    if letter in vowels:
        found.setdefault(letter, 0)
        found[letter] += 1

for k, v in sorted(found.items()):
    print(k, 'was found', v, 'time(s).')

```

"vowels6.py" fixed the runtime error thanks to the use of the "setdefault" method, which comes with every dictionary (and assigns a default value to a key if a value isn't already set).

"Vowels6.py" corrigió el error de tiempo de ejecución gracias al uso del método "setdefault", que viene con cada diccionario (y asigna un valor predeterminado a una clave si un valor no está ya establecido).

## *Chapter 3's Code, 2 of 2*

```

vowels = set('aeiou')
word = input("Provide a word to search for vowels: ")
found = vowels.intersection(set(word))
for vowel in found:
    print(vowel)

```

The final version of the vowels program, "vowels7.py", took advantage of Python's set data structure to considerably shrink the list-based "vowels3.py" code, while still providing the same functionality.

La versión final del programa, "vowels7.py", aprovechó la estructura de datos establecida de Python para reducir considerablemente el código basado en listas, al mismo tiempo que proporciona la misma funcionalidad.

*¿No había ningún programa de muestra que aprovechara las tuplas?*

**No, no había. Pero eso está bien.**

No hemos explotado las tuplas en este capítulo con un programa de ejemplo, ya que las tuplas no entran en su propio hasta que se discutió en relación con las funciones. Como ya hemos dicho, veremos las tuplas de nuevo cuando nos encontremos con las funciones (en el próximo capítulo), así como en otras partes de este libro. Cada vez que los veamos, estaremos seguros de señalar cada uso de tupla. A medida que continúe con sus viajes de Python, verá tuplas aparecer en todo el lugar.

## 4 code reuse



# Functions and Modules



*No importa cuánto código escribo, las cosas se vuelven totalmente inmanejables después de un tiempo ...*

***Reusing code is key to building a maintainable system.***

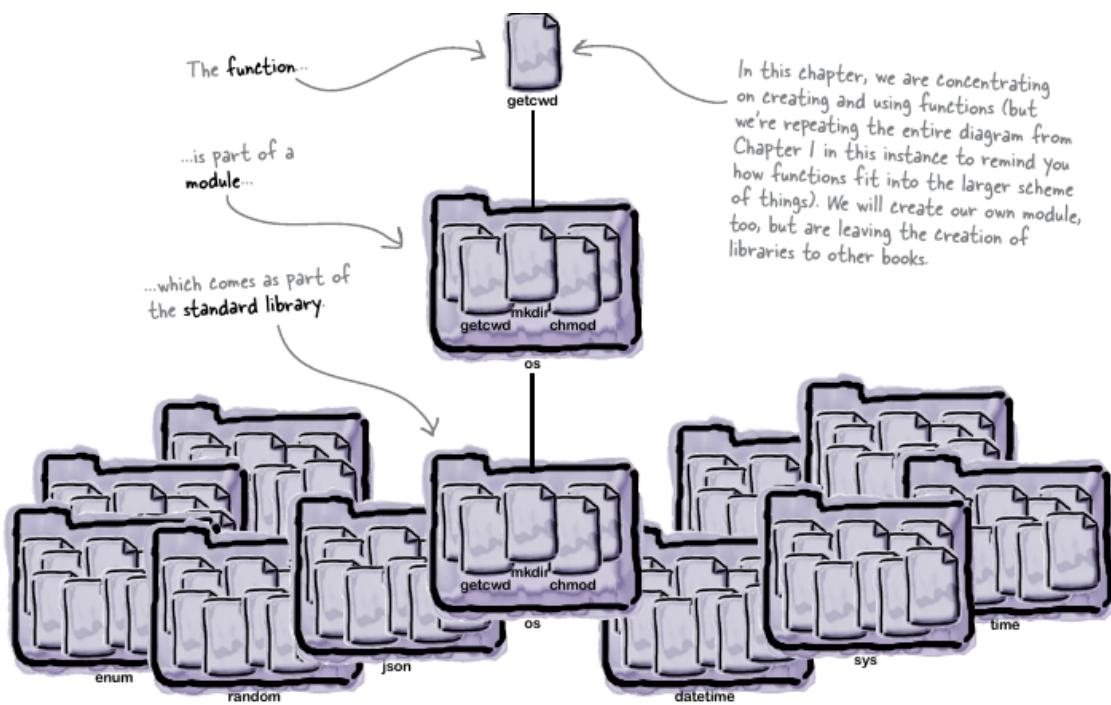
***Reutilizar código es clave para construir un sistema que pueda ser mantenido.***

Y cuando se trata de reutilizar código en Python, todo comienza y termina con la función humilde. Tomar algunas líneas de código, darles un nombre, y usted tiene una función (que puede ser reutilizado). Tome una colección de funciones y los empaque como un archivo, y usted tiene un módulo (que también puede ser reutilizado). Es verdad lo que dicen: es bueno compartir, y al final de este capítulo, estarás bien en tu camino para compartir y reutilizar tu código, gracias a la comprensión de cómo funcionan las funciones y los módulos de Python.

### ***Reusing Code with Functions***

Aunque algunas líneas de código pueden lograr mucho en Python, tarde o temprano encontrarás que la base de código de tu programa está creciendo ... y, cuando lo hace, las cosas se vuelven más difíciles de manejar. Lo que comenzó como 20 líneas de código Python ha aumentado de alguna manera a 500 líneas o más! Cuando esto sucede, es hora de empezar a pensar en las estrategias que puedes utilizar para reducir la complejidad de su código base.

Como muchos otros lenguajes de programación, Python soporta la modularidad, ya que se pueden dividir grandes trozos de código en partes más pequeñas y más manejables. Esto se hace mediante la creación de funciones, que se puede pensar en como trozos de código. Recuerde este diagrama del capítulo 1, que muestra la relación entre funciones, módulos y la biblioteca estándar:



En este capítulo, nos estamos concentrando en crear y usar funciones (pero estamos repitiendo todo el diagrama del Capítulo 1 en este caso para recordarles cómo las funciones encajan en el esquema más amplio de las cosas). Vamos a crear nuestro propio módulo, también, pero están dejando la creación de bibliotecas a otros libros.

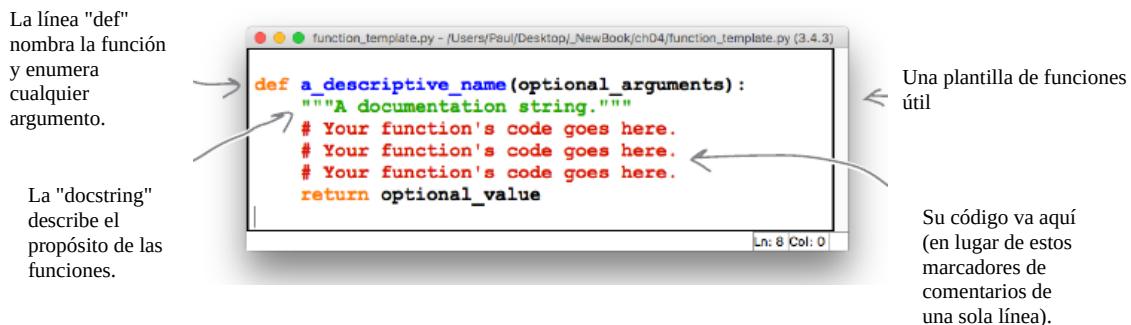
En este capítulo, vamos a concentrarnos en lo que implica crear sus propias funciones, que se muestran en la parte superior del diagrama. Una vez que esté felizmente creando funciones, también le mostraremos cómo crear un módulo.

## ***Introducción a las funciones***

Antes de llegar a convertir algo de nuestro código existente en una función, pasemos un momento mirando la anatomía de cualquier función en Python. Una vez que esta introducción esté completa, veremos parte de nuestro código existente y recorreremos los pasos necesarios para convertirlo en una función que puede reutilizar.

No se preocupe por los detalles todavía. Todo lo que necesitas hacer aquí es tener una idea de cómo funcionan las funciones en Python, como se describe en esta página y en la siguiente. Exploraremos los detalles de todo lo que necesita saber a medida que avanza este capítulo. La ventana IDLE en esta página presenta una plantilla que puede utilizar al crear cualquier función. Como usted lo está viendo, considere lo siguiente:

- Las funciones introducen dos nuevas palabras clave:** def y return. Ambas palabras clave están coloreadas en naranja en IDLE. La palabra clave def define la función (mostrada en azul) y detalla los argumentos que la función pueda tener. El uso de la palabra clave return es opcional y se utiliza para devolver un valor al código que invocó la función.
- Las funciones pueden aceptar datos de argumento** Una función puede aceptar datos de argumento (es decir, entrada a la función). Puede especificar una lista de argumentos entre los paréntesis en la línea def, siguiendo el nombre de la función.
- Las funciones contienen código y (por lo general) la documentación** El código está indentado un nivel debajo de la línea def, e incluye los comentarios donde tiene sentido. Demostramos dos maneras de agregar comentarios al código: usando una cadena de tres comillas (mostrada en verde en la plantilla y conocida como docstring) y usando un comentario de una sola línea, el cual está prefijado con el símbolo # (y se muestra en rojo). A continuación.



*Python utiliza el nombre "function" para describir un fragmento reutilizable de código. Otros lenguajes de programación usan nombres como "procedimiento", "subrutina" y "método". Cuando una función forma parte de una clase Python, se la conoce como "método". Aprenderá todo acerca de las clases y métodos de Python en un capítulo posterior.*

### ¿Qué hay sobre la información de tipo?

Echa un vistazo a nuestra plantilla de funciones. Aparte de algún código para ejecutar, ¿crees que falta algo? ¿Hay algo que esperas que se especifique, pero no lo es? Echa otro vistazo:

function\_template.py - /Users/Paul/Desktop/\_NewBook/ch04/function\_template.py (3.4.3)

```
def a_descriptive_name(optional_arguments):
    """A documentation string."""
    # Your function's code goes here.
    # Your function's code goes here.
    # Your function's code goes here.
    return optional_value
```

Ln: 8 Col: 0

¿Falta algo en esta plantilla de función?

El intérprete de Python no le obliga a especificar el tipo de argumentos de su función o el valor de retorno. Dependiendo de los lenguajes de programación que haya utilizado antes, esto puede asustar. No lo dejes.

Python te permite enviar cualquier objeto como un argumento y devolver cualquier objeto como un valor de retorno. El intérprete no se preocupa ni verifica qué tipo de objetos son (sólo que se proporcionan).

Con Python 3, es posible indicar los tipos esperados de argumentos / valores de retorno, y lo haremos más adelante en este capítulo. Sin embargo, indicar los tipos esperados no cambia "mágicamente" la comprobación de tipo, ya que Python nunca comprueba los tipos de los argumentos o cualquier valor de retorno.

### **Nombrar un trozo de código con "def"**

Una vez que haya identificado un fragmento de su código Python que desea reutilizar, es hora de crear una función. Crea una función utilizando la palabra clave def (que es la abreviatura define). La palabra clave def viene seguida por el nombre de la función, una lista de argumentos opcionalmente vacía (entre paréntesis), dos puntos y, a continuación, una o más líneas de código sangrado.

Recordemos el programa vowels7.py desde el final del último capítulo, el cual, dado una palabra, imprime las vocales contenidas en esa palabra:

The diagram shows handwritten notes on the left pointing to specific parts of a block of code. The notes are:

- Take a set of vowels...
- ...and a word...
- ...then perform an intersection.

The code block contains the following Python code:

```
vowels = set('aeiou')
word = input("Provide a word to search for vowels: ")
found = vowels.intersection(set(word))
for vowel in found:
    print(vowel)
```

A bracket on the right side of the code block is labeled "Display any results."

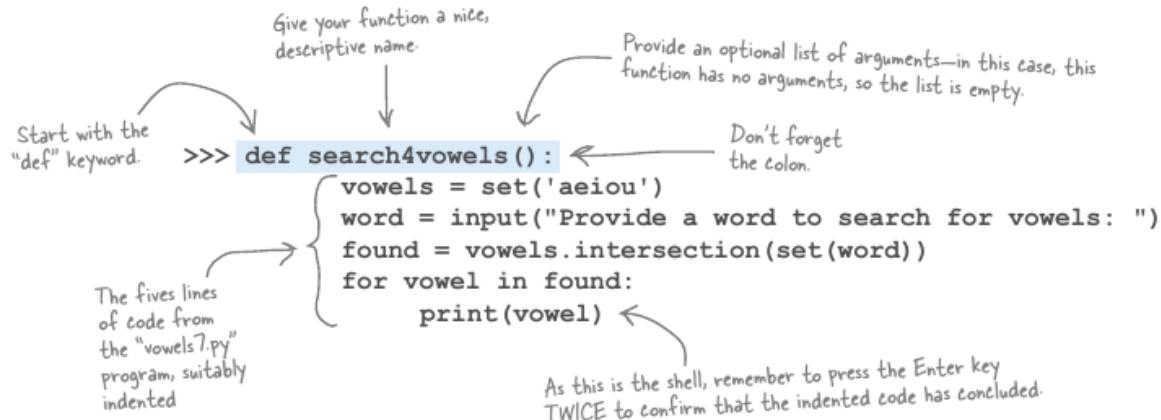
To the right of the code block, there is a note:

Esto es  
"vowels7.py"  
"desde el  
final del  
Capítulo 3.

Imaginemos que planeas usar estas cinco líneas de código muchas veces en un programa mucho más grande. Lo último que querrás hacer es copiar y pegar este código donde sea necesario ... así, para mantener las cosas manejables y para asegurarte de que sólo necesitas mantener una copia de este código, vamos a crear una función.

Vamos a demostrar cómo en el Python Shell (por ahora). Para convertir las cinco líneas de código anteriores en una función, utilice la palabra clave def para indicar que se está iniciando una función; Dar a la función un nombre descriptivo (siempre una buena idea); Proporcione una lista opcionalmente vacía de argumentos entre paréntesis, seguida de dos puntos; Y luego indent las líneas de código en relación con la palabra clave def, de la siguiente manera:

**Tómese el tiempo para elegir un buen nombre descriptivo para su función.**



Ahora que la función existe, invocémosla para ver si está funcionando como esperamos.

### **Llamando funciones**

#### **Invocando su función**

Para invocar funciones en Python, proporcione el nombre de la función junto con valores para cualquier argumento que la función espera. Como la función `search4vowels` (actualmente) no toma argumentos, podemos invocarla con una lista de argumentos vacía, así:

```
>>> search4vowels()
Provide a word to search for vowels: hitch-hiker
e
i
```

La invocación de la función se vuelve a ejecutar:

```
>>> search4vowels()
Provide a word to search for vowels: galaxy
a
```

No hay sorpresas aquí: invocar la función ejecuta su código.

#### **Edite su función en un editor, no en el prompt**

Por el momento, el código para la función `search4vowels` se ha introducido en el prompt `>>>`, y se ve así:

```

>>> def search4vowels():
    vowels = set('aeiou')
    word = input("Provide a word to search for vowels: ")
    found = vowels.intersection(set(word))
    for vowel in found:
        print(vowel)

Our function
as entered
at the shell
prompt.

```

*Asegúrese de haber guardado su código como "vsearch.py" después de copiar el código de la función desde el shell.*

Para poder seguir trabajando con este código, puede volver a llamarlo en el prompt >>> y editararlo, pero esto se vuelve muy difícil de manejar, muy rápidamente. Recuerde que una vez que el código con el que está trabajando en el prompt >>> es más de unas pocas líneas, es mejor copiar el código en una ventana de edición IDLE. Puedes editar lo mucho más fácilmente allí. Así que, hagamos eso antes de continuar.

Cree una ventana de edición IDLE nueva y vacía, luego copie el código de la función desde el indicador >>> (asegurándose de no copiar los caracteres >>>) y péguelo en la ventana de edición. Una vez que esté satisfecho de que el formato y la sangría son correctos, guarde el archivo como vsearch.py antes de continuar.

### **Use IDLE's Editor to Make Changes Utilice el editor de IDLE para realizar cambios**

Esto es lo que parece el archivo vsearch.py en IDLE:

El código de la función está ahora en una ventana de edición IDLE y se ha guardado como "vsearch.py".

```

def search4vowels():
    vowels = set('aeiou')
    word = input("Provide a word to search for vowels: ")
    found = vowels.intersection(set(word))
    for vowel in found:
        print(vowel)

```

Si IDLE muestra un mensaje de error cuando presiona F5, ¡no entre en pánico! Vuelva a editar su ventana y compruebe que su código es el mismo que el nuestro, luego vuelva a intentarlo.

Si presiona F5 mientras está en la ventana de edición, suceden dos cosas: el shell IDLE se lleva al primer plano y el shell se reinicia. Sin embargo, nada aparece en la pantalla. Pruebe

esto ahora para ver lo que queremos decir: presione F5. La razón para no mostrar nada es que todavía tiene que invocar la función. Vamos a invocarlo en un poco, pero por ahora vamos a hacer un cambio a nuestra función antes de seguir adelante. Es un cambio pequeño, pero importante sin embargo. Vamos a agregar algo de documentación a la parte superior de nuestra función. Para agregar un comentario de varias líneas (un docstring) a cualquier código, incluya el texto del comentario entre comillas triples.

Aquí está el archivo vsearch.py una vez más, con un docstring añadido a la parte superior de la función. Adelante y haga este cambio a su código, también:

```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3.4.3)

def search4vowels():
    """Display any vowels found in an asked-for word."""
    vowels = set('aeiou')
    word = input("Provide a word to search for vowels: ")
    found = vowels.intersection(set(word))
    for vowel in found:
        print(vowel)

Lc: 9 Col: 0
```

Una docstring se ha agregado al código de la función, que (brevemente) describe el propósito de esta función.

## ¿Cuál es el trato con todas esas Cadenas?

Eche un vistazo a la función tal como está actualmente. Preste especial atención a las tres cadenas de este código, que están todas de color verde por IDLE:

```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3.4.3)

def search4vowels():
    """Display any vowels found in an asked-for word."""
    vowels = set('aeiou')
    word = input("Provide a word to search for vowels: ")
    found = vowels.intersection(set(word))
    for vowel in found:
        print(vowel)

Lc: 9 Col: 0
```

IDLE sintaxis de resaltado muestra que tenemos de cadena un problema de coherencia con su inútil? citas.  
¿Cuándo usamos qué estilo

## Understanding the string quote characters

En Python, las cadenas pueden encerrarse en un carácter de comillas simples ('), un carácter de comillas dobles ("") o lo que se conoce como comillas triples ("""o """).

Como se mencionó anteriormente, las comillas triples alrededor de las cadenas son conocidas como docstrings, ya que se utilizan principalmente para documentar el propósito de una función (como se muestra arriba). Aunque puede usar """ o "" para rodear sus

docstrings, la mayoría de los programadores de Python prefieren usar """". Las cadenas de documentos tienen una característica interesante, ya que pueden abarcar varias líneas (otros lenguajes de programación usan el nombre "heredoc" para el mismo concepto).

### ***Seguir las mejores prácticas según las PEPs***

Cuando se trata de formatear el código (no sólo las cadenas), la comunidad de programación de Python ha dedicado mucho tiempo a establecer y documentar las mejores prácticas. Esta mejor práctica se conoce como PEP 8. PEP es la abreviatura de "Python Enhancement Protocol".

PEP 8 es la guía de estilo para código Python. Se recomienda leer para todos los programadores de Python, y es el documento que sugiere el consejo "ser consistente" para las citas de cadenas descritas en la última página. Tómese el tiempo para leer PEP 8 al menos una vez. Otro documento, PEP 257, ofrece convenciones sobre cómo dar formato a docstrings, y vale la pena leer, también.

Aquí está la función search4vowels una vez más en su forma PEP 8 y PEP 257. Los cambios no son extensos, pero la estandarización de los caracteres de comillas simples en torno a nuestras cadenas (pero no alrededor de nuestras docstrings) se ve un poco mejor:

Se trata de una docstring compatible con PEP 257.

```
def search4vowels():
    """Display any vowels found in an asked-for word."""
    vowels = set('aeiou')
    word = input('Provide a word to search for vowels: ')
    found = vowels.intersection(set(word))
    for vowel in found:
        print(vowel)
```

Hemos escuchado el consejo de PEP 8 de ser consistente con el carácter de comillas simples que usamos para rodear nuestras cadenas.

Por supuesto, no es necesario escribir código que se ajuste exactamente a PEP 8. Por ejemplo, nuestro nombre de función, search4vowels, no se ajusta a las directrices, lo que sugiere que las palabras en el nombre de una función deben estar separadas por un guión bajo: a El nombre más compatible es search\_for\_vowels. Tenga en cuenta que PEP 8 es un conjunto de directrices, no reglas. Usted no tiene que cumplir, sólo considere, y nos gusta el nombre search4vowels.

Volvamos ahora a mejorar la función **search4vowels** para aceptar argumentos.

### Add an argument

### *Functions Can Accept Arguments*

En lugar de tener la función de pedir al usuario una palabra para buscar, vamos a cambiar la función **search4vowels** para que podamos pasar la palabra como entrada a un argumento.

Agregar un argumento es sencillo: basta con insertar el nombre del argumento entre los paréntesis de la línea **def**. Este nombre de argumento se convierte entonces en una variable en la suite de la función. Esta es una edición fácil.

También vamos a eliminar la línea de código que le pide al usuario que proporcione una palabra para buscar, que es otra edición fácil.

Recordemos el estado actual de nuestro código:

```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3.4.3)

def search4vowels():
    """Display any vowels found in an asked-for word."""
    vowels = set('aeiou')
    word = input('Provide a word to search for vowels: ')
    found = vowels.intersection(set(word))
    for vowel in found:
        print(vowel)

Ln: 9 Col: 0
```

Here's our original function.

Esta línea ya no es necesaria.

La aplicación de las dos ediciones sugeridas (de arriba) a nuestra función da como resultado la ventana de edición IDLE que se ve así (nota: también hemos actualizado nuestra docstring, que siempre es una buena idea):

```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3.4.3)

def search4vowels(word):
    """Display any vowels found in a supplied word."""
    vowels = set('aeiou')
    found = vowels.intersection(set(word))
    for vowel in found:
        print(vowel)

Ln: 8 Col: 0
```

Put the argument's name between the parentheses.

The call to the "input" function is gone (as we don't need that line of code anymore).

Asegúrese de guardar su archivo después de cada cambio de código, antes de presionar F5 para tomar la nueva versión de su función para una vuelta.

Con el código cargado en la ventana de edición de IDLE (y guardado), presione F5, luego invoque la función unas cuantas veces y vea lo que sucede:

The current "search4vowels" code

```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3.4.3)
def search4vowels(word):
    """Display any vowels found in a supplied word."""
    vowels = set('aeiou')
    found = vowels.intersection(set(word))
    for vowel in found:
        print(vowel)

Python 3.4.3 Shell
>>> ===== RESTART =====
>>>
>>> search4vowels()
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    search4vowels()
TypeError: search4vowels() missing 1 required positional argument: 'word'
>>> search4vowels('hitch-hiker')
e
i
>>> search4vowels('hitch-hiker', 'galaxy')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    search4vowels('hitch-hiker', 'galaxy')
TypeError: search4vowels() takes 1 positional argument but 2 were given
>>> |
```

Ln: 12 Col: 4

Aunque hemos invocado la función "search4vowels" tres veces en este Test Drive La única invocación que se ejecutó correctamente fue la que pasó en un único argumento de cadenas. Los otros dos fracasaron. Tómese un momento para leer los mensajes de error producidos por el intérprete para saber por qué falló cada una de las llamadas incorrectas.

**P: ¿Estoy restringido a un único argumento al crear funciones en Python?**

**R:** No, puede tener tantos argumentos como desee, dependiendo del servicio que su función esté proporcionando. Estamos deliberadamente empezando con un ejemplo sencillo, y llegaremos a ejemplos más complicados a medida que avanza este capítulo. Usted puede hacer mucho con los argumentos a las funciones en Python, y planeamos discutir la mayor parte de lo que es posible en las próximas docenas de páginas o menos.

### ***Functions Return a Result***

### ***Funciones Devolver un resultado***

Además de utilizar una función para abstraer algún código y darle un nombre, los programadores normalmente desean que las funciones devuelvan algún valor calculado, con lo que puede trabajar el código que llamó a la función. Para soportar la devolución de un valor (o valores) de una función, Python proporciona la sentencia **return**.

Cuando el intérprete encuentra una sentencia de devolución en la suite de su función, suceden dos cosas: la función termina en la sentencia de devolución y cualquier valor proporcionado a la sentencia de devolución se devuelve a su código de llamada. Este comportamiento imita cómo funciona el retorno en la mayoría de otros lenguajes de programación.

Comencemos con un ejemplo sencillo de devolver un solo valor de nuestra función `search4vowels`. Específicamente, retomamos True o False dependiendo de si la palabra suministrada como argumento contiene alguna vocal.

Esto es un poco una salida de la funcionalidad existente de nuestra función, pero soporta con nosotros, ya que vamos a construir algo más complejo (y útil) en un poco. Comenzando con un simple ejemplo asegura que tenemos los elementos básicos en su lugar antes de seguir adelante.

Eso suena como un plan con el que puedo vivir. La única pregunta que tengo es ¿cómo sé si algo es verdadero o falso?

### **La verdad es...**

Python viene con una función incorporada llamada **bool** que, cuando se proporciona con cualquier valor, le indica si el valor se evalúa como True o False.

No sólo funciona con cualquier valor, sino que funciona con cualquier objeto de Python. El efecto de esto es que la noción de verdad de Python se extiende mucho más allá del 1 para Verdadero y el 0 para Falso que emplean otros lenguajes de programación.

Hagamos una pausa y echemos un breve vistazo a Verdadero y Falso antes de volver a nuestra discusión sobre el retorno.

Cada objeto en Python tiene un valor de verdad asociado con él, en el que el objeto se evalúa a True o False.

Algo es False si se evalúa a 0, el valor None, una cadena vacía o una estructura de datos integrada vacía. Esto significa que todos estos ejemplos son falsos:

```
>>> bool(0)      } ← If an object evaluates to  
False  
>>> bool(0.0)    O, it is always False.  
False  
  
>>> bool('')     } ← An empty string, an empty list, and  
False  
>>> bool([])      an empty dictionary all evaluate to  
False  
  
>>> bool({})      } ← Python's "None" value is  
False  
>>> bool(None)    always False.  
False
```

Cada otro objeto en Python se evalúa como True. Estos son algunos ejemplos de objetos que son Verdaderos:

Podemos pasar cualquier objeto a la función `bool` y determinar si es verdadero o falso.

Criticamente, cualquier estructura de datos no vacía se evalúa como True.

### ***Returning One Value . Devolver un valor***

Echemos otra mirada al código de nuestra función, que actualmente acepta cualquier valor como argumento, busca el valor suministrado para las vocales y luego muestra las vocales encontradas en la pantalla:

```
def search4vowels(word):
    """Display any vowels found in a supplied word."""
    vowels = set('aeiou')
    found = vowels.intersection(set(word))
    for vowel in found:
        print(vowel) } ← We'll change these two lines.
```

El cambio de esta función para devolver True o False, basado en si se han encontrado vocales, es sencillo. Simplemente reemplace las dos últimas líneas de código (el bucle for) con esta línea de código:

The diagram shows the original code for the `search4vowels` function. A callout points to the `return bool(found)` line with the text: "Call the 'bool'" function, and...". Another callout points to the parameter `found` with the text: "...Pass in the name of the data structure that contains the results of the vowels search."

```
def search4vowels(phrase):
    found = []
    for vowel in 'aeiou':
        if vowel in phrase:
            found.append(vowel)
    return bool(found)
```

Si no se encuentra nada, la función devuelve False; De lo contrario, devuelve True. Con este cambio hecho, ahora puede probar esta nueva versión de su función en Python Shell y ver lo que sucede:

The screenshot shows the Python Shell output for three calls to `search4vowels`. The first two calls return `True`, and the third returns `False`. A callout on the left explains that the `return` statement, which uses the `bool` function, returns either `True` or `False`. A callout on the right notes that `'y'` is not considered a vowel.

```
>>> search4vowels('hitch-hiker')
True
>>> search4vowels('galaxy')
True
>>> search4vowels('sky')
False
```

Si continúa viendo el comportamiento de versiones anteriores, asegúrese de haber guardado la nueva versión de su función, así como presionó F5 desde la ventana de edición.

No se sienta tentado a poner paréntesis alrededor del objeto que devuelve el código de llamada. No es necesario. La instrucción `return` no es una llamada de función, por lo que el uso de paréntesis no es un requisito sintáctico. Puede usarlos (si realmente lo desea), pero la mayoría de los programadores de Python no lo hacen.

## **Returning More Than One Value**

### **Devolver más de un valor**

Las funciones están diseñadas para devolver un valor único, pero a veces es necesario devolver más de un valor. La única manera de hacerlo es empaquetar los múltiples valores en una única estructura de datos y luego devolverlos. Por lo tanto, todavía estás devolviendo una cosa, aunque potencialmente contiene muchas piezas individuales de datos.

Esta es nuestra función actual, que devuelve un valor booleano (es decir, una cosa):

```

def search4vowels(word):
    """Return a boolean based on any vowels found."""
    vowels = set('aeiou')
    found = vowels.intersection(set(word))
    return bool(found)

```

Nota: hemos actualizado el comentario.

Es una edición trivial que la función devuelva valores múltiples (en un conjunto) en oposición a un booleano. Todo lo que necesitamos hacer es dar de baja la llamada a `bool`:

```

def search4vowels(word):
    """Return any vowels found in a supplied word."""
    vowels = set('aeiou')
    found = vowels.intersection(set(word))
    return found

```

*We've updated the comment again.*

*Return the results as a data structure (a set).*

Podemos reducir aún más las dos últimas líneas de código en la versión anterior de nuestra función a una línea eliminando el uso innecesario de la variable encontrada `found`. En lugar de asignar los resultados de la intersección a la variable encontrada `found` y devolverla, simplemente devuelva la intersección:

```

def search4vowels(word):
    """Return any vowels found in a supplied word."""
    vowels = set('aeiou')
    return vowels.intersection(set(word))

```

*Return the data without the use of the unnecessary "found" variable.*

Nuestra función ahora devuelve un conjunto de vocales encontradas en una palabra, que es exactamente lo que nos propusimos hacer. Sin embargo, cuando lo probamos, uno de nuestros resultados nos ha rascado la cabeza ...

Tomemos esta última versión de la función `search4vowels` para una vuelta y veamos cómo se comporta. Con el código más reciente cargado en una ventana de edición IDLE, presione F5 para importar la función en el Shell de Python y, a continuación, invoque la función varias veces:

A screenshot of the Python 3.4.3 Shell window titled "RESTART". The code input is:

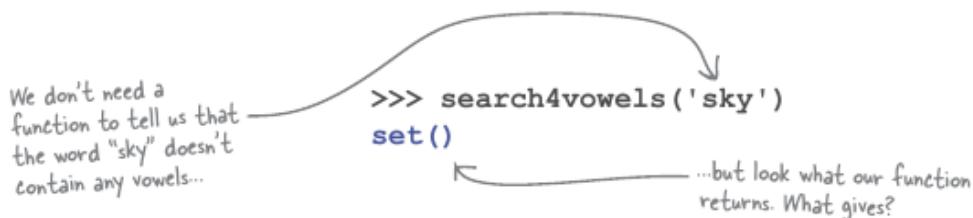
```
>>> search4vowels('hitch-hiker')
{'e', 'i'}
>>> search4vowels('galaxy')
{'a'}
>>> search4vowels('life, the universe and everything')
{'e', 'u', 'a', 'i'}
>>> search4vowels('sky')
set()
>>>
```

The output is displayed in green. A brace on the right side groups the first four results, with a note pointing to it: "Each of these function invocations works as expected, even though the result from the last one looks a little weird." The status bar at the bottom right shows "Ln: 38 Col: 4".

Cada una de estas invocaciones de función funciona como se esperaba, aunque el resultado de la última se ve un poco extraño.

## ¿Cuál es el problema con "set ()"?

Cada ejemplo en el Test Drive anterior funciona bien, ya que la función toma un solo valor de cadena como argumento, luego devuelve el conjunto de vocales encontradas. El resultado, el conjunto, contiene muchos valores. Sin embargo, la última respuesta parece un poco raro, ¿no? Echemos un vistazo más de cerca:



No necesitamos una función para decirnos que la palabra "cielo" no contiene vocales ...  
... pero mira lo que devuelve nuestra función. ¿Lo que da?

Es posible que haya esperado que la función devuelva {} para representar un conjunto vacío, pero eso es un malentendido común, ya que {} representa un diccionario vacío, no un conjunto vacío.

Un conjunto vacío se representa como **set()** por el intérprete.

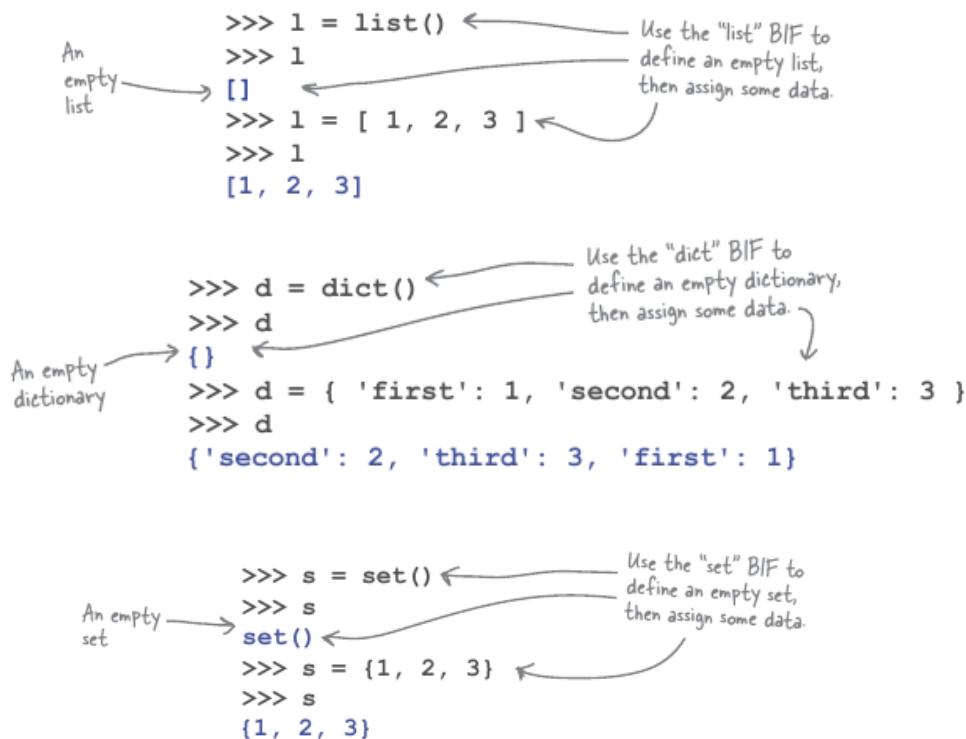
Esto puede parecer un poco raro, pero es sólo la forma en que las cosas funcionan en Python. Tomemos un momento para recordar las cuatro estructuras de datos incorporadas, con el fin de ver cómo cada estructura de datos vacía está representada por el intérprete.

## Recuperación de las estructuras de datos incorporadas

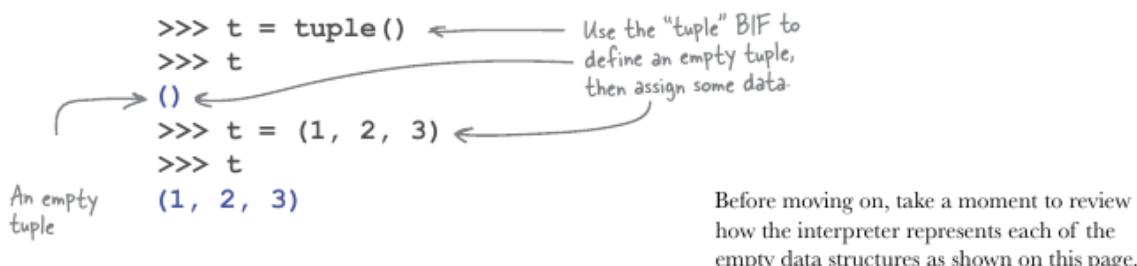
Recordemos las cuatro estructuras de datos incorporadas disponibles. Tomaremos cada estructura de datos a su vez, trabajando a través de lista, diccionario, conjunto y, finalmente, tupla.

Trabajando en el shell, creemos una estructura de datos vacía usando las funciones integradas de estructura de datos (BIFs para abreviar), y luego asignamos una pequeña cantidad de datos a cada uno. A continuación, mostraremos el contenido de cada estructura de datos después de cada asignación:

**BIF es de corta duración para "función incorporada".**



A pesar de que los conjuntos están encerrados en llaves, también lo son los diccionarios. Un diccionario vacío ya está usando las llaves dobles rizadas, por lo que un conjunto vacío tiene que ser representado como "set()".



Antes de seguir adelante, tómese un momento para revisar cómo el intérprete representa cada una de las estructuras de datos vacías, como se muestra en esta página.

### **Anota tus funciones**

## **Usar anotaciones para mejorar tus documentos**

Nuestra revisión de las cuatro estructuras de datos confirma que la función search4vowels devuelve un conjunto. Pero, aparte de llamar a la función y comprobar el tipo de retorno, ¿cómo pueden los usuarios de nuestra función saber esto antes de tiempo? ¿Cómo saben qué esperar?

Una solución es agregar esta información a la docstring. Esto supone que usted indica muy claramente en su docstring cuáles serán los argumentos y el valor devuelto y que esta información es fácil de encontrar. Conseguir que los programadores se pongan de acuerdo sobre un estándar para documentar funciones es problemático (PEP 257 sólo sugiere el formato de docstrings), por lo que ahora Python 3 soporta una anotación llamada anotaciones (también conocidas como sugerencias de tipo). Cuando se utilizan, las anotaciones documentan -en forma estándar- el tipo de retorno, así como los tipos de cualquier argumento. Tenga en cuenta estos puntos:

- **Las anotaciones de funciones son opcionales** Está bien no usarlos. De hecho, una gran cantidad de código Python existente no (ya que sólo se puso a disposición de los programadores en las versiones más recientes de Python 3).
- **Las anotaciones de funciones son informativas.** Proporcionan detalles sobre su función, pero no implican ningún otro comportamiento (como la comprobación de tipo).

Vamos a anotar los argumentos de la función search4vowels. La primera anotación indica que la función espera una cadena como el tipo de la palabra argumento (`: str`), mientras que la segunda anotación indica que la función devuelve un conjunto a su llamador (`-> set`):

Estamos  
afirmando  
que el  
argumento  
de  
"word"  
se espera  
que sea  
una  
cadena.

We are stating that the  
"word" argument is expected  
to be a string.

```
def search4vowels(word:str) -> set:  
    """Return any vowels found in a supplied word."""  
    vowels = set('aeiou')  
    return vowels.intersection(set(word))
```

We are stating that the  
function returns a set to  
its caller.

Estamos indicando que la  
función devuelve un conjunto  
a su llamador.

La sintaxis de anotación es sencilla. Cada argumento de función tiene un colon adjunto a él, junto con el tipo que se espera. En nuestro ejemplo, str especifica que la función espera una cadena. El tipo de retorno se proporciona después de la lista de argumentos, y se indica mediante un símbolo de flecha, que es seguido por el tipo de retorno y luego por los dos puntos. Aquí -> **set**: indica que la función va a devolver un conjunto.

Hasta aquí todo bien.

Hemos anotado nuestra función de manera estándar. Debido a esto, los programadores que utilizan nuestra función ahora saben lo que se espera de ellos, así como qué esperar de la función. Sin embargo, el intérprete no comprobará que la función siempre se llama con una cadena, ni comprueba que la función devuelve siempre un conjunto. Lo que plantea una pregunta bastante obvia ...

### *¿Por qué utilizar las anotaciones de funciones?*

Si el intérprete de Python no va a usar sus anotaciones para comprobar los tipos de argumentos de su función y su tipo de retorno, ¿por qué preocuparse por las anotaciones en absoluto?

El objetivo de las anotaciones no es hacer la vida más fácil para el intérprete; Es para hacer la vida más fácil para el usuario de su función. Las anotaciones son un estándar de documentación, no un mecanismo de aplicación de tipos.

De hecho, el intérprete no le importa qué tipo de argumentos son, ni le importa qué tipo de datos devuelve su función. El intérprete llama a su función con los argumentos que se le proporcionen (sin importar su tipo), ejecuta el código de su función y, a continuación, devuelve al llamador cualquier valor que es dado por la declaración de return. El intérprete no considera el tipo de datos que se pasan de un lado a otro.

Lo que las anotaciones hacen para los programadores que usan su función es deshacerse de la necesidad de leer el código de su función para aprender qué tipos son esperados y devueltos de su función. Esto es lo que tendrán que hacer si no se utilizan anotaciones. Incluso el docstring escrito más maravillosamente todavía tendrá que ser leído si no incluye anotaciones.

Lo que lleva a otra pregunta: ¿cómo podemos ver las anotaciones sin leer el código de la función? Desde el editor de IDLE, presione F5, luego utilice el BIF de ayuda en el prompt >>>.

**Utilice anotaciones para documentar sus funciones y utilice el BIF de "ayuda" para verlas.**

Si aún no lo ha hecho, use el editor de IDLE para anotar su copia de search4vowels, guarde su código y luego presione la tecla F5. El Python Shell se reiniciará y el prompt >>> estará esperando que hagas algo. Pida a la ayuda BIF que muestre la documentación de search4vowels, así:

The screenshot shows a Python 3.4.3 Shell window. The command `>>> help(search4vowels)` is entered, resulting in the following output:

```
>>> ===== RESTART =====
>>>
>>> help(search4vowels)
Help on function search4vowels in module __main__:

search4vowels(word:str) -> set
    Return any vowels found in a supplied word.

>>> |
```

A handwritten note on the right side of the screen says: "Not only does 'help' display the annotations, but it shows the docstring too." An arrow points from this note to the word "Annotations" in the code above, and another arrow points from the note to the word "docstring" in the code above.

## **Funciones: Lo que ya sabemos**

Tomemos un momento para revisar una vez más el código de la función search4vowels. Ahora que acepta un argumento y devuelve un conjunto, es más útil que la primera versión de la función desde el comienzo de este capítulo, ya que ahora podemos usarla en muchos más lugares:

```
def search4vowels(word:str) -> set:
    """Return any vowels found in a supplied word."""
    vowels = set('aeiou')
    return vowels.intersection(set(word))
```

La versión  
más reciente  
de nuestra  
función

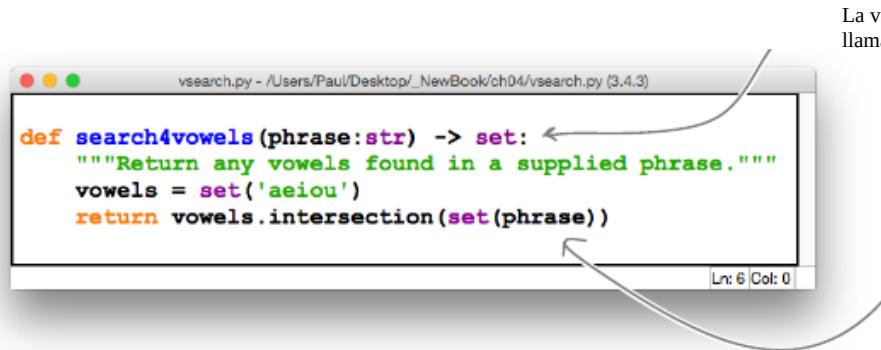
Esta función sería aún más útil si, además de aceptar un argumento para que la palabra a buscar, también aceptara un segundo argumento que detalla qué buscar. Esto nos permitiría buscar cualquier conjunto de letras, no sólo las cinco vocales.

Además, el uso de la palabra nombre como un nombre de argumento es OK, pero no grande, ya que esta función acepta claramente cualquier cadena como un argumento, a diferencia de una sola palabra. Un mejor nombre de variable podría ser frase, ya que se ajusta más a lo que es lo que esperamos recibir de los usuarios de nuestra función.

*Cambiemos ahora nuestra función para reflejar esta última sugerencia.*

## **Haciendo una función genéticamente útil**

Esta es una versión de la función search4vowels (tal como aparece en IDLE) después de haber sido cambiada para reflejar la segunda de las dos sugerencias de la parte inferior de la última página. A saber, hemos cambiado el nombre de la variable de la palabra a la frase más apropiada:



```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py [3.4.3]
def search4vowels(phrase:str) -> set:
    """Return any vowels found in a supplied phrase."""
    vowels = set('aeiou')
    return vowels.intersection(set(phrase))
```

La variable "word" ahora se llama "phrase".

La otra sugerencia de la parte inferior de la última página era permitir a los usuarios especificar el conjunto de letras a buscar, en contraposición a usar siempre las cinco vocales. Para hacer esto podemos agregar un segundo argumento a la función que especifica las letras para buscar la frase para. Este es un cambio fácil de hacer. Sin embargo, una vez que lo hagamos, la función (tal como está) será incorrectamente nombrada, ya que ya no estaremos buscando vocales, estaremos buscando cualquier conjunto de letras. En lugar de cambiar la función actual, creemos una segunda basada en la primera. Esto es lo que nos proponemos hacer:

1. **Proporcione a la nueva función un nombre más genérico** En lugar de seguir ajustando search4vowels, creemos una nueva función llamada search4letters, que es un nombre que refleja mejor el propósito de la nueva función.
2. **Añadir un segundo argumento** Añadir un segundo argumento nos permite especificar el conjunto de letras para buscar la cadena. Llámelo a las letras del segundo argumento. Y no olvidemos anotar las letras, también.
3. **Eliminar la variable de vocales** El uso de las vocales de nombre en la suite de la función ya no tiene sentido, ya que ahora estamos buscando un conjunto de letras especificado por el usuario.
4. **Actualizar la docstring** No hay punto de copiar, a continuación, cambiar, el código si no también ajustar la docstring. Nuestra documentación necesita ser actualizada para reflejar lo que hace la nueva función.

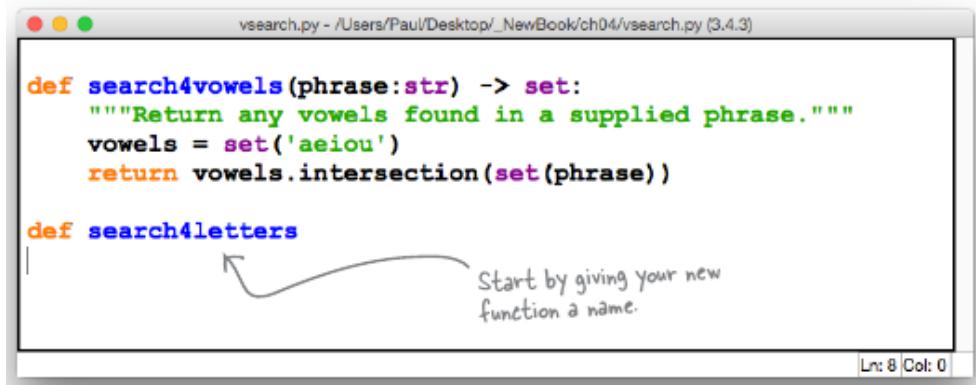
Vamos a trabajar juntos en estas cuatro tareas. A medida que se discute cada tarea, asegúrese de editar su archivo vsearch.py para reflejar los cambios presentados.

### **paso a paso**

#### **Creando otra función, 1 de 3**

Si aún no lo ha hecho, abra el archivo vsearch.py en una ventana de edición IDLE. El primer paso consiste en crear una nueva función, que llamaremos search4letters. Tenga en cuenta que PEP 8 sugiere que todas las funciones de nivel superior están rodeadas por dos líneas en blanco. Todas las descargas de este libro se ajustan a esta guía, pero el código que mostramos en la página impresa no lo hace (ya que aquí el espacio es un premio).

En la parte inferior del archivo, escriba **def** seguido del nombre de su nueva función:



```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3.4.3)

def search4vowels(phrase:str) -> set:
    """Return any vowels found in a supplied phrase."""
    vowels = set('aeiou')
    return vowels.intersection(set(phrase))

def search4letters
```

Start by giving your new function a name.

Para el Paso 2 estamos completando la línea def de la función añadiendo los nombres de los dos argumentos, frase y letras requeridos. Recuerde incluir la lista de argumentos entre paréntesis y no olvide incluir los dos puntos finales (y las anotaciones):

```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3.4.3)

def search4vowels(phrase:str) -> set:
    """Return any vowels found in a supplied phrase."""
    vowels = set('aeiou')
    return vowels.intersection(set(phrase))

def search4letters(phrase:str, letters:str) -> set: ←
    ↗ Specify the list of arguments, and don't
    ↗ forget the colon (and the annotations, too).

```

Ln: 8 Col: 4

¿Notó cómo el editor de IDLE ha anticipado que la siguiente línea de código necesita ser identificada (y colocada automáticamente el cursor)?

Especifique la lista de argumentos y no olvide los dos puntos (y las anotaciones también)

Con los pasos 1 y 2 completados, ahora estamos listos para escribir el código de la función. Este código va a ser similar al de la función search4vowels, excepto que planeamos eliminar nuestra dependencia de la variable **vowels**.

## Creando otra función, 2 de 3

En el Paso 3, que es escribir el código para la función de tal manera que se elimine la necesidad de la variable vocales. Podríamos seguir usando la variable, pero darle un nuevo nombre (ya que las vocales ya no representan lo que hace la variable), pero una variable temporal no es necesaria aquí, por la misma razón de por qué ya no necesitamos la variable encontrada `found` antes. Echa un vistazo a la nueva línea de código en `search4letters`, que hace el mismo trabajo que las dos líneas en `search4vowels`:

```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3.4.3)

def search4vowels(phrase:str) -> set:
    """Return any vowels found in a supplied phrase."""
    vowels = set('aeiou')
    return vowels.intersection(set(phrase))

def search4letters(phrase:str, letters:str) -> set:
    return set(letters).intersection(set(phrase))


```

Two lines of code become one.

Ln: 9 Col: 0

Dos líneas de código se convierten en una.

Si esa sola línea de código en `search4letters` tiene que rascarse la cabeza, no se desespere. Parece más complejo de lo que es. Vamos a pasar por esta línea de código en detalle para

resolver exactamente lo que hace. Se inicia cuando el valor del argumento de letras se convierte en un conjunto:

`set(letters)` ← Create a set object from "letters".

Esta llamada al conjunto BIF crea un objeto set a partir de los caracteres de la variable letters. No es necesario asignar este objeto de conjunto a una variable, ya que estamos más interesados en utilizar el conjunto de letras de inmediato que en almacenar el conjunto en una variable para su uso posterior. Para usar el objeto set recién creado, añada un punto y, a continuación, especifique el método que desea invocar, ya que incluso los objetos que no están asignados a variables tienen métodos. Como sabemos por el uso de conjuntos en el último capítulo, el método de intersección toma el conjunto de caracteres contenidos en su argumento (frase) y los cruza con un objeto set existente (letras):

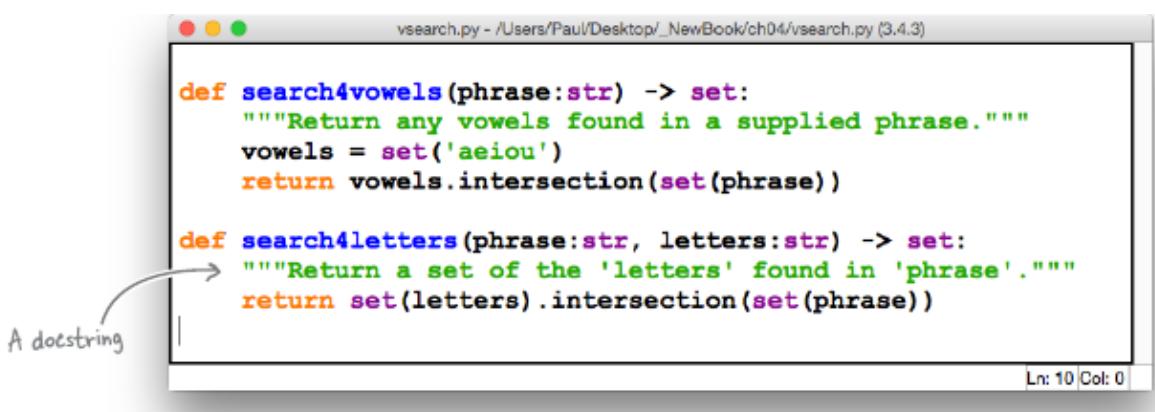
✓ Realizar una intersección de conjunto en el  
objeto conjunto hecho de "letras" con el  
`set(letters).intersection(set(phrase))` objeto de conjunto hecho de "frase".

Y, finalmente, el resultado de la intersección se devuelve al código de llamada, gracias a la instrucción return:

Send the results back to the calling code. ↗  
`return set(letters).intersection(set(phrase))`

## Creando otra función, 3 de 3

Todo lo que queda es el paso 4, donde agregamos una docstring a nuestra función recién creada. Para ello, agregue una cadena de tres comillas justo después de la línea def de su nueva función. Esto es lo que utilizamos (como comentarios son cortos, pero eficaces):



```
vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3,4,3)

def search4vowels(phrase:str) -> set:
    """Return any vowels found in a supplied phrase."""
    vowels = set('aeiou')
    return vowels.intersection(set(phrase))

def search4letters(phrase:str, letters:str) -> set:
    """Return a set of the 'letters' found in 'phrase'."""
    return set(letters).intersection(set(phrase))

A docstring
```

Y con eso, nuestros cuatro pasos están completos y search4letters está listo para ser probado.

*¿Por qué ir a todos los problemas de crear una función de una línea? ¿No es mejor simplemente copiar y pegar esa línea de código cuando lo necesite?*

### **Las funciones también pueden ocultar la complejidad.**

Es correcto observar que acabamos de crear una función de una línea, que puede no sentir como mucho de un "ahorro". Sin embargo, tenga en cuenta que nuestra función contiene una sola línea compleja de código, que estamos ocultando a los usuarios. De esta función, y esto puede ser una práctica muy valiosa (por no mencionar, mucho mejor que todo lo que copiar y pegar). Por ejemplo, la mayoría de los programadores sería capaz de adivinar lo que hace search4letters si se encontrara con una invocación de la misma en un programa. Sin embargo, si se encontraron con esa compleja línea única de código en un programa, que bien puede arañar sus cabezas y se preguntan qué hace. Por lo tanto, aunque search4letters es "corto", sigue siendo una buena idea abstraer este tipo de complejidad dentro de una función.

Guarde el archivo vsearch.py una vez más y, a continuación, presione F5 para probar la función search4letters:

The screenshot shows a Python 3.4.3 Shell window. At the top, it says "Python 3.4.3 Shell" and "RESTART". The shell contains the following text:

```
>>> ===== RESTART =====
>>> help(search4letters)
Help on function search4letters in module __main__:
  search4letters(phrase:str, letters:str) -> set
    Return a set of the 'letters' found in 'phrase'.

  >>> search4letters('hitch-hiker', 'aeiou')
  {'e', 'i'}
  >>> search4letters('galaxy', 'xyz')
  {'x', 'y'}
  >>> search4letters('life, the universe, and everything', 'o')
  set()
  >>>
```

Annotations are present in the code:

- A callout bubble points to the line "Help on function search4letters in module \_\_main\_\_:" with the text "Use the 'help' BIF to learn how to use 'search4letters'."
- A curly brace groups the three examples starting with "search4letters" and points to the first one with the text "All of these examples produce what we expect them to."

La función search4letters es ahora más genérica que search4vowels, ya que toma cualquier conjunto de letras y busca una frase dada para ellos, en lugar de buscar las letras a, e, i, o y u. Esto hace que nuestra nueva función sea mucho más útil que search4vowels. Ahora imaginemos que tenemos una base de código grande y existente que ha utilizado extensivamente search4vowels. Se ha tomado la decisión de retirar search4vowels y reemplazarlo con search4letters, ya que los "powers that be" no ven la necesidad de ambas funciones, ahora que search4letters puede hacer lo que hace search4vowels. Una búsqueda global y reemplazo de su base de código para el nombre "search4vowels" con "search4letters" no funcionará aquí, ya que tendrá que agregar en ese segundo valor de argumento, que siempre va a ser aeiou cuando se simula el comportamiento De search4vowels con search4letters. Así, por ejemplo, esta llamada de un solo argumento:

```
search4vowels("Don't panic!")
```

Ahora necesita ser reemplazado con este argumento de doble argumento (que es una edición mucho más difícil de automatizar):

```
search4letters("Don't panic!", 'aeiou')
```

Sería bueno si pudiéramos de alguna manera especificar un valor predeterminado para el segundo argumento de search4letters, y luego hacer que la función lo use si no se proporciona ningún valor alternativo. Si pudiéramos arreglar el valor predeterminado para aeiou, entonces podríamos aplicar una búsqueda y un reemplazo global (que es una edición fácil).

*¿No sería soñador si Python me permitiera especificar valores predeterminados? Pero sé que es sólo una fantasía ...*

## **Especificar valores predeterminados para argumentos**

Cualquier argumento a una función Python se puede asignar un valor predeterminado, que puede ser utilizado automáticamente si el código que llama a la función no puede proporcionar un valor alternativo. El mecanismo para asignar un valor predeterminado a un argumento es sencillo: incluir el valor predeterminado como una asignación en la línea def de la función.

Aquí está la línea actual de search4letters:

```
def search4letters(phrase:str, letters:str) -> set:
```

Esta versión de la línea de def de nuestra función (arriba) espera exactamente dos argumentos, uno para la frase y otro para las letras. Sin embargo, si asignamos un valor predeterminado a las letras, la línea def de la función cambia para que se vea así:

```
def search4letters(phrase:str, letters:str='aeiou') -> set:
```

Se ha asignado un valor predeterminado al argumento "letters" y se utilizará siempre que el código de llamada no proporcione un valor alternativo.

Podemos seguir utilizando la función search4letters de la misma manera que antes: proporcionando ambos argumentos con valores según sea necesario. Sin embargo, si nos olvidamos de proporcionar el segundo argumento (letras), el intérprete sustituirá en el valor aeiou en nuestro nombre.

Si tuviéramos que hacer este cambio a nuestro código en el archivo vsearch.py (y guardarlo), podríamos entonces invocar nuestras funciones de la siguiente manera:

```
These three function calls all produce →
{ >>> search4letters('life, the universe, and everything')
  {'a', 'e', 'i', 'u'}
>>> search4letters('life, the universe, and everything', 'aeiou')
  {'a', 'e', 'i', 'u'}
>>> search4vowels('life, the universe, and everything') ←
  {'a', 'e', 'i', 'u'}
```

En esta invocación, estamos llamando "search4vowels", no "search4letters".

No sólo estas llamadas de función producen la misma salida, también demuestran que la función search4vowels ya no es necesaria ahora que el argumento letters to search4letters soporta un valor predeterminado (compare las invocaciones primera y última arriba).

Ahora, si se nos pide retirar la función search4vowels y reemplazar todas las invocaciones de la misma dentro de nuestra base de código con search4letters, nuestra explotación del mecanismo de valor por defecto para argumentos de función nos permite hacerlo con una simple búsqueda y sustitución global. Y no tenemos que usar search4letters para buscar

sólo vocales. Ese segundo argumento nos permite especificar cualquier conjunto de caracteres a buscar. Como consecuencia, search4letters es ahora más genérico y más útil.

## Asignación Posicional versus Palabra Clave

Como acabamos de ver, la función search4letters puede ser invocada con uno o dos argumentos, siendo el segundo argumento opcional. Si proporciona sólo un argumento, el argumento de las letras predeterminado es una cadena de vocales. Echa un vistazo a la línea def de la función:

La línea "def" de nuestra función

```
def search4letters(phrase:str, letters:str='aeiou') -> set: ↴
```

Además de soportar argumentos por defecto, el intérprete de Python también le permite invocar una función usando argumentos de **palabras clave**. Para entender qué es un argumento de palabras clave, considere cómo hemos invocado a search4letters hasta ahora, por ejemplo:

```
search4letters('galaxy', 'xyz')  
def search4letters(phrase:str, letters:str='aeiou') -> set:  
    ↴  
    ↴
```

En la invocación anterior, las dos cadenas se asignan a la frase y argumentos de letras basadas en su posición. Es decir, la primera cadena se asigna a la frase, mientras que la segunda se asigna a las letras. Esto se conoce como asignación posicional, ya que se basa en el orden de los argumentos.

En Python, también es posible referirse a los argumentos por su nombre de argumento, y cuando lo hace, el ordenamiento posicional ya no se aplica. Esto se conoce como asignación de palabras clave. Para usar palabras clave, asigne cada cadena en cualquier orden a su nombre de argumento correcto al invocar la función, como se muestra aquí:

El orden de los argumentos no es importante cuando se utilizan argumentos de palabras clave durante la invocación.

```
search4letters(letters='xyz', phrase='galaxy') ↴  
def search4letters(phrase:str, letters:str='aeiou') -> set:  
    ↴  
    ↴
```

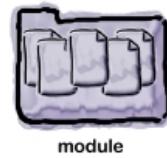
Ambas invocaciones de la función search4letters en esta página producen el mismo resultado: un conjunto que contiene las letras y y z. Aunque puede ser difícil apreciar el beneficio de usar argumentos de palabras clave con nuestra pequeña función de search4letters, la flexibilidad que esta característica le da se vuelve clara cuando invoca una función que acepta muchos argumentos. Veremos un ejemplo de una de estas funciones (proporcionada por la biblioteca estándar) antes del final de este capítulo.

**Estas funciones realmente marcaron para mí. ¿Cómo hago para usarlos y compartirlos?**

***Hay más de una manera de hacerlo.***

Ahora que tiene algún código que vale la pena compartir, es razonable preguntar cómo usar y compartir estas funciones de la mejor manera. Como con la mayoría de las cosas, hay más de una respuesta a esa pregunta. Sin embargo, en las próximas páginas, aprenderá la mejor forma de empaquetar y distribuir sus funciones para asegurar que sea fácil para usted y otros beneficiarse de su trabajo.

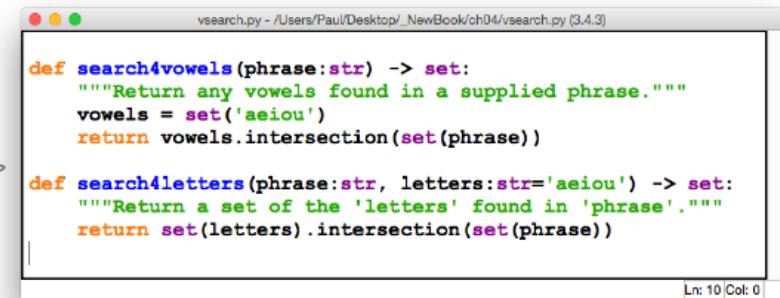
## **Funciones Beget Módulos**



Habiendo tenido problemas para crear una función reutilizable (o dos, como es el caso con las funciones actualmente en nuestro archivo vsearch.py), es razonable preguntar: ¿cuál es la mejor manera de compartir funciones?

Es posible compartir cualquier función copiándola y pegándola a través de su base de código cuando sea necesario, pero como esa es una idea tan inútil y mala, no vamos a considerarla por mucho más tiempo. Tener copias múltiples de la misma función que cubre su base de código es una receta segura para el desastre (si alguna vez decide cambiar el funcionamiento de su función). Es mucho mejor crear un módulo que contenga una copia única canónica de las funciones que deseé compartir. Lo que plantea otra pregunta: ¿cómo se crean los módulos en Python?

La respuesta no podría ser más simple: un módulo es cualquier archivo que contenga funciones. Felizmente, esto significa que vsearch.py ya es un módulo. Aquí está otra vez, en toda su gloria del módulo:



```

vsearch.py - /Users/Paul/Desktop/_NewBook/ch04/vsearch.py (3.4.3)

def search4vowels(phrase:str) -> set:
    """Return any vowels found in a supplied phrase."""
    vowels = set('aeiou')
    return vowels.intersection(set(phrase))

def search4letters(phrase:str, letters:str='aeiou') -> set:
    """Return a set of the 'letters' found in 'phrase'."""
    return set(letters).intersection(set(phrase))

```

Ln: 10 Col: 0

"vsearch.py" contains functions in a file, making it a fully formed module.

"Vsearch.py" contiene funciones en un archivo, lo que lo convierte en un módulo completamente formado.

## ***Creación de módulos no podría ser más fácil, sin embargo ...***

Crear módulos es un pedazo de pastel: simplemente cree un archivo de las funciones que desea compartir.

Una vez que su módulo existe, hacer su contenido disponible para sus programas también es sencillo: todo lo que tiene que hacer es importar el módulo utilizando la instrucción de importación de Python.

Esto en sí mismo no es complejo. Sin embargo, el intérprete hace la suposición de que el módulo en cuestión está en **la ruta de búsqueda**, y asegurar este es el caso puede ser complicado. Vamos a explorar los entresijos de la importación de módulos en las próximas páginas.

### ***¿Dónde está mi módulo?***

### ***¿Cómo se encuentran los módulos?***

Recuerde en el primer capítulo de este libro cómo importamos y luego utilizamos la función **randint** del módulo aleatorio **random**, que viene incluido como parte de la biblioteca estándar de Python. Esto es lo que hicimos en el shell:

Identify the module to import, then... →

```

>>> import random
>>> random.randint(0, 255)
42

```

... invoca una de las funciones del módulo.

Lo que ocurre durante la importación de módulos se describe con gran detalle en la documentación de Python, a la que puede explorar y explorar si los detalles de gran calidad flotan en su barco. Sin embargo, todo lo que realmente necesita saber son las tres ubicaciones principales que el intérprete busca cuando busca un módulo. Estos son:

1. **Su directorio de trabajo actual.** Esta es la carpeta en la que el intérprete piensa que está trabajando.
2. **Ubicaciones de los paquetes de sitios o site-package de su intérprete**  
Estos son los directorios que contienen cualquier módulo de Python de terceros que haya instalado (incluyendo cualquier escrito por usted).
3. **Las ubicaciones de biblioteca estándar** Son los directorios que contienen todos los módulos que forman la biblioteca estándar.

Dependiendo del sistema operativo que esté ejecutando, el nombre dado a una ubicación que contiene archivos puede ser directorio o carpeta. Usaremos "carpeta" en este libro, excepto cuando discutamos el directorio de trabajo actual (que es un término bien establecido).

El orden en que las ubicaciones 2 y 3 son buscadas por el intérprete puede variar dependiendo de muchos factores. Pero no te preocupes: no es importante que sepas cómo funciona este mecanismo de búsqueda. Lo que es importante entender es que el intérprete siempre busca en su directorio de trabajo actual primero, que es lo que puede causar problemas cuando está trabajando con sus propios módulos personalizados.

Para demostrar lo que puede salir mal, vamos a ejecutar un pequeño ejercicio que está diseñado para resaltar el problema. Esto es lo que debe hacer antes de comenzar:

- Cree una carpeta llamada **mymodules**, que utilizaremos para almacenar sus módulos. No importa dónde en su sistema de archivos cree esta carpeta; Sólo asegúrese de que está en algún lugar donde usted tiene acceso de lectura / escritura.
- Mueva su archivo **vsearch.py** a la carpeta **mymodules** recién creada. Este archivo debe ser la única copia del archivo **vsearch.py** de su computadora.

## **Ejecución de Python desde la línea de comandos**

Vamos a ejecutar el intérprete de Python desde la línea de comandos (o terminal) de su sistema operativo para demostrar lo que puede salir mal aquí (aunque el problema que estamos a punto de discutir también se manifiesta en IDLE).

Si está ejecutando cualquier versión de Windows, abra un símbolo del sistema y siga junto con esta sesión. Si no está en Windows, discutiremos su plataforma a mitad de camino en la siguiente página (pero siga leyendo ahora mismo). Puede invocar al intérprete de Python (fuera de IDLE) escribiendo **py -3** en el indicador de Windows C: \>. Observe cómo antes de invocar al intérprete, usamos el comando cd para hacer que la carpeta mymodules sea nuestro directorio de trabajo actual. Además, observe que podemos salir del intérprete en cualquier momento escribiendo **quit()** en el prompt >>>:

Cambie a la carpeta "mymodules".

```
File Edit Window Help Redmond #1
C:\Users\Head First> cd mymodules
C:\Users\Head First\mymodules> py -3
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC
v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import vsearch
>>> vsearch.search4vowels('hitch-hiker')
{'i', 'e'}
>>> vsearch.search4letters('galaxy', 'xyz')
{'y', 'x'}
>>> quit()

C:\Users\Head First\mymodules>
```

Start Python 3.

Import the module.

Use the module's functions.

Exit the Python interpreter and return to your operating system's command prompt.

Yo en linux manjaro:

```
[anton@anton-pc ~]$ cd pruebasparacodigo
[anton@anton-pc pruebasparacodigo]$ ls
Application.py  loggeo      mymodules   prueba.py  __pycache__  Tuto
clock.py        modulosTest prueba2.py  py3gTest   pyside
[anton@anton-pc pruebasparacodigo]$ cd mymodules
[anton@anton-pc mymodules]$ ls
vsearch.py
[anton@anton-pc mymodules]$ python
Python 3.6.0 (default, Jan 16 2017, 13:35:36)
[GCC 6.3.1 20170109] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import vsearch
>>> vsearch.search4vowels('hitch-hiker')
{'i', 'e'}
>>> vsearch.search4letters('galaxy','xyz')
{'y', 'x'}
>>> quit()
[anton@anton-pc mymodules]$
```

se ejecuta igual.

Esto funciona como se espera: importaremos con éxito el módulo vsearch y luego usaremos cada una de sus funciones prefijando el nombre de la función con el nombre de su módulo y un punto. Observe cómo el comportamiento del prompt >>> en la línea de comandos es idéntico al comportamiento dentro de IDLE (la única diferencia es la falta de resaltado de sintaxis). Después de todo, es el mismo intérprete de Python.

Aunque esta interacción con el intérprete tuvo éxito, sólo funcionó porque empezamos en una carpeta que contenía el archivo vsearch.py. Hacer esto hace que esta carpeta sea el directorio de trabajo actual. En base a cómo el intérprete busca módulos, sabemos que el directorio de trabajo actual se busca primero, por lo que no debe sorprendernos que esta interacción funcionó y que el intérprete encontró nuestro módulo.

***Pero, ¿qué sucede si nuestro módulo no está en el directorio de trabajo actual?***

*No importar aquí*

### **Módulos No Encontrados Produce ImportErrors**

Repita el ejercicio desde la última página, después de salir de la carpeta que contiene nuestro módulo. Veamos qué sucede cuando intentamos importar nuestro módulo ahora. Aquí hay otra interacción con el símbolo del sistema de Windows:

The screenshot shows a Windows command prompt window titled 'Redmond #2'. The path 'C:\Users\Head First>' is displayed. The user has typed 'cd \', which changes the directory to the root ('C:\'). A callout points to this action with the text: 'Change to another folder (in this case, we are moving to the top-level folder.)'. Another callout points to the 'cd' command with the text: 'Start Python 3 again.' A third callout points to the resulting error message with the text: 'Try to import the module... ...but this time we get an error!' The error message itself is: 'C:\>py -3 Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32 Type "help", "copyright", "credits" or "license" for more information. >>> import vsearch Traceback (most recent call last): File "<stdin>", line 1, in <module> ImportError: No module named 'vsearch' >>> quit() C:\>'.

en linux:

```
[anton@anton-pc mymodules]$ cd ..
[anton@anton-pc pruebasparacodigo]$ cd ..
[anton@anton-pc ~]$ python
Python 3.6.0 (default, Jan 16 2017, 13:35:36)
[GCC 6.3.1 20170109] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import vsearch
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'vsearch'
>>> quit()
[anton@anton-pc ~]$ █
```

El archivo `vsearch.py` ya no está en el directorio de trabajo actual del intérprete, ya que ahora estamos trabajando en una carpeta que no sea `mymodules`. Esto significa que no se puede encontrar nuestro archivo de módulo, lo que a su vez significa que no podemos importarlo, de ahí el `ImportError` del intérprete.

Si intentamos el mismo ejercicio en una plataforma que no sea Windows, obtenemos los mismos resultados (si estamos en Linux, Unix o Mac OS X). Aquí está la interacción anterior con el intérprete desde dentro de la carpeta `mymodules` en OS X:

Change into the folder and then type "python3" to start the interpreter.

Import the module.

It works: we can use the module's functions.

```

File Edit Window Help Cupertino #1
$ cd mymodules
mymodules$ python3
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import vsearch
>>> vsearch.search4vowels('hitch-hiker')
{'i', 'e'}
>>> vsearch.search4letters('galaxy', 'xyz')
{'x', 'y'}
>>> quit()
mymodules$
```

Exit the Python interpreter and return to your operating system's command prompt.

176 Chapter 4

## **ImportErrors Occur No Matter the Platform**

### **Errores de importación no afectan a la plataforma**

Si crees que correr en una plataforma que no sea Windows solucionará de alguna manera este problema de importación que vimos en esa plataforma, piensa de nuevo: el mismo ImportError se produce en sistemas UNIX, una vez que cambiamos a otra carpeta:

Cambie a otra carpeta (en este caso, nos estamos moviendo a nuestra carpeta de nivel superior).

Start Python 3 again.

Try to import the module...

...but this time we get an error!

Change to another folder (in this case, we are moving to our top-level folder).

```

File Edit Window Help Cupertino #2
mymodules$ cd
$ python3
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import vsearch
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'vsearch'
>>> quit()
$
```

```
[anton@anton-pc pruebasparacodigo]$ ls
Application.py  loggeo      mymodules   prueba.py  __pycache__  Tuto
clock.py        modulosTest  prueba2.py  py3gTest   pyside
[anton@anton-pc pruebasparacodigo]$ python
Python 3.6.0 (default, Jan 16 2017, 13:35:36)
[GCC 6.3.1 20170109] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import vsearch
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'vsearch'
>>> quit()
[anton@anton-pc pruebasparacodigo]$
```

Como en el caso de Windows, el archivo vsearch.py ya no está en el directorio de trabajo actual del intérprete, ya que ahora estamos trabajando en una carpeta que no sea mymodules. Esto significa que no se puede encontrar nuestro archivo de módulo, lo que a su vez significa que no podemos importarlo, de ahí el ImportError del intérprete. Este problema se presenta no importa qué plataforma está ejecutando Python.

**P:** ¿No podemos ser específicos de la ubicación y decir algo como importar C:\mymodules\vssearch en plataformas Windows, o quizás import /mymodules/vsearch en sistemas similares a UNIX?

**A:** No, no puedes. Por supuesto, hacer algo así suena tentador, pero en última instancia, no funcionará, ya que no puede utilizar rutas de esta manera con la declaración de importación de Python. Y, de todos modos, lo último que querrás hacer es poner caminos codificados en cualquiera de tus programas, ya que las rutas a menudo pueden cambiar (por toda una serie de razones). Lo mejor es evitar las rutas de codificación en su código, si es posible.

**P:** Si no puedo usar caminos, ¿cómo puedo organizar que el intérprete encuentre mis módulos?

**R:** Si el intérprete no puede encontrar su módulo en el directorio de trabajo actual, se ve en las ubicaciones de paquetes de sitio o site-packages, así como en la biblioteca estándar (y hay más información sobre los paquetes de sitios en la página siguiente). Si usted puede arreglar para agregar su módulo a una de las localizaciones de los sitios-paquetes, el intérprete puede entonces encontrarla allí (no importa su trayectoria).

## **Instalar en Python**

### ***Getting a Module into Site-packages***

#### ***Cómo obtener un módulo en los paquetes del sitio***

Recordemos lo que teníamos que decir acerca de los paquetes de sitio de unas pocas páginas cuando los presentamos como la segunda de tres ubicaciones buscadas por el mecanismo de importación del intérprete:

- 2. Ubicaciones de los paquetes de sitios o site-package de su intérprete** Estos son los directorios que contienen cualquier módulo de Python de terceros que pueda haber instalado (incluyendo cualquiera escrito por usted).

Dado que la provisión y el soporte de módulos de terceros es fundamental para la estrategia de reutilización de código de Python, no debería sorprendernos que el intérprete tenga la capacidad incorporada de agregar módulos a la configuración de Python.

Tenga en cuenta que el conjunto de módulos incluido en la biblioteca estándar es administrado por los desarrolladores de Python, y esta amplia colección de módulos ha sido diseñada para ser ampliamente utilizada, pero no alterada. Específicamente, no agregue ni elimine sus propios módulos a / de la biblioteca estándar. Sin embargo, la adición o la eliminación de módulos a sus ubicaciones de paquetes de sitio o site-package se anima positivamente, tanto que Python viene con algunas herramientas para hacerlo sencillo.

#### ***Using “setuptools” to install into site-packages***

#### ***Uso de "setuptools" para instalar en paquetes de sitio***

A partir de la versión 3.4 de Python, la biblioteca estándar incluye un módulo denominado setuptools, que se puede utilizar para agregar cualquier módulo a los paquetes de sitio. Aunque los detalles de la distribución de módulos pueden parecer complejos, todo lo que queremos hacer aquí es instalar vsearch en site-packages, algo que setuptools es más que capaz de hacer en tres pasos:

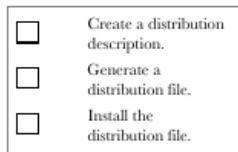
- 1. Crear una descripción de distribución.** Esto identifica el módulo que queremos instalar setuptools.
- 2. Generar un archivo de distribución.** Usando Python en la línea de comandos, crearemos un archivo de distribución compatible para contener nuestro código de módulos.
- 3. Instalar el archivo de distribución.** Una vez más, utilizando Python en la línea de comandos, instale el archivo de distribución (que incluye nuestro módulo) en paquetes de sitio o site-packages.

Python 3.4 (o más reciente) hace que usar setuptools sea fácil. Si no está ejecutando 3.4 (o más reciente), considere actualizar.

El paso 1 nos obliga a crear (como mínimo) dos archivos descriptivos para nuestro módulo: **setup.py** y **README.txt**. Vamos a ver lo que está involucrado.

## ***Creating the Required Setup Files***

### ***Creación de los archivos de instalación necesarios***



*We'll check off each completed step as we work through this material.*

Si seguimos los tres pasos que se muestran en la parte inferior de la última página, terminaremos creando un paquete de distribución para nuestro módulo. Este paquete es un solo archivo comprimido que contiene todo lo necesario para instalar nuestro módulo en paquetes de sitios.

**Para el Paso 1,** cree una descripción de distribución, necesitamos crear dos archivos que colocaremos en la misma carpeta que nuestro archivo **vsearch.py**. Haremos esto sin importar en qué plataforma estamos corriendo. El primer archivo, que debe llamarse **setup.py**, describe nuestro módulo con cierto detalle.

A continuación, encontrará el archivo **setup.py** que creamos para describir el módulo en el archivo **vsearch.py**. Contiene dos líneas de código Python: la primera línea importa la función de configuración o **setup** desde el módulo **setuptools**, mientras que la segunda invoca la función de configuración o **setup**.

La función de configuración **setup** acepta un gran número de argumentos, muchos de los cuales son opcionales. Tenga en cuenta cómo, para fines de legibilidad, nuestra llamada a la configuración **setup** se distribuye en nueve líneas. Estamos aprovechando el soporte de Python para los argumentos de palabra clave para indicar claramente qué valor se está asignando a qué argumento en esta llamada. Se destacan los argumentos más importantes;

El primer nombre de la distribución, mientras que el segundo enumera los archivos .py para incluir al crear el paquete de distribución:

```
from setuptools import setup
setup(
    name='vsearch',
    version='1.0',
    description='The Head First Python Search Tools',
    author='HF Python 2e',
    author_email='hfpy2e@gmail.com',
    url='headfirstlabs.com',
    py_modules=['vsearch'],
```

Importe la función "setup" desde el módulo "setuptools".

Esta es una invocación de la función "setup". Estamos extendiendo sus argumentos sobre muchas líneas.

El argumento "nombre" identifica la distribución. Es una práctica común nombrar la distribución después del módulo.

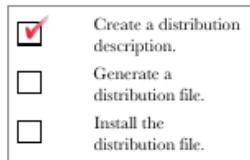
The "name" argument identifies the distribution. It's common practice to name the distribution after the module.

Esta es una lista de archivos ".py" para incluir en el paquete. Para este ejemplo, sólo tenemos uno: "vsearch".

Además de **setup.py**, el mecanismo de **setuptools** requiere la existencia de otro archivo, un archivo "readme", en el cual se puede poner una descripción textual de su paquete. Aunque se requiere este archivo, su contenido es opcional, por lo que (por ahora) puede crear un archivo vacío llamado **README.txt** en la misma carpeta que el archivo **setup.py**. Esto es suficiente para satisfacer el requisito de un segundo archivo en el paso 1.

## ***Creating a Distribution File***

## ***Creación de un archivo de distribución***



En esta etapa, debe tener tres archivos, que hemos incluido en su carpeta de módulos: **vsearch.py**, **setup.py** y **README.txt**.

Ahora estamos listos para crear un paquete de distribución de estos archivos. Éste es el Paso 2 de nuestra lista anterior: Genera un archivo de distribución. Lo haremos en la línea de comandos. Aunque hacerlo es sencillo, este paso requiere que se introduzcan comandos diferentes basados en si está en Windows o en uno de los sistemas operativos tipo UNIX (Linux, Unix o Mac OS X).

## ***Creating a distribution file on Windows***

### ***Creación de un archivo de distribución en Windows***

Si se ejecuta en Windows, abra un símbolo del sistema en la carpeta que contiene los tres archivos y, a continuación, escriba este comando:

Ejecute Python 3 en Windows.

```
C:\Users\Head First\mymodules> py -3 setup.py sdist
```

Ejecutar el código en  
"setup.py" ...

... y pasar "sdist" como  
argumento.

El intérprete de Python comienza a funcionar inmediatamente después de emitir este comando. Un gran número de mensajes aparecen en la pantalla (que mostramos aquí en una forma abreviada):

```
running sdist
running egg_info
creating vsearch.egg-info
...
creating dist
creating 'dist\vsearch-1.0.zip' and adding 'vsearch-1.0' to it
adding 'vsearch-1.0\PKG-INFO'
adding 'vsearch-1.0\README.txt'
...
adding 'vsearch-1.0\vsearch.egg-info\top_level.txt'
removing 'vsearch-1.0' (and everything under it)
```

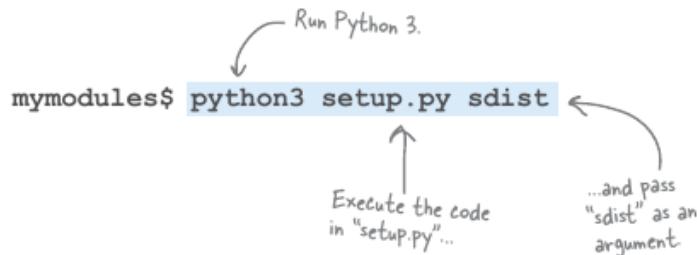
Si ve este mensaje todo  
está bien. Si recibe  
errores, compruebe que  
está ejecutando al  
menos Python 3.4, y  
también asegúrese de  
que su "setup.py" es  
idéntico al nuestro.

Cuando vuelve a aparecer el símbolo del sistema de Windows, los tres archivos se han combinado en un único **archivo de distribución**. Se trata de un archivo instalable que contiene el código fuente de su módulo y, en este caso, se denomina **vsearch-1.0.zip**.

Encontrará su archivo ZIP recientemente creado en una carpeta denominada **dist**, que también ha sido creada por **setuptools** bajo la carpeta en la que está trabajando (que es **mymodules** en nuestro caso).

## **Distribution Files on UNIX-like Oses**

### **Archivos de distribución en sistemas operativos tipo UNIX**



Al igual que en Windows, este comando produce una gran cantidad de mensajes en la pantalla:

```
running sdist
running egg_info
creating vsearch.egg-info
...
running check
creating vsearch-1.0
creating vsearch-1.0/vsearch.egg-info
...
creating dist
Creating tar archive
removing 'vsearch-1.0' (and everything under it) ←
```

Los mensajes  
difieren  
ligeramente de los  
producidos en  
Windows. Si ve  
este mensaje, todo  
está bien. Si no es  
así (como en  
Windows),  
verifique todo.

Cuando vuelve a aparecer la línea de comandos del sistema operativo, los tres archivos se han combinado en un archivo de distribución de origen o **source distribution**(de ahí el argumento sdist anterior). Se trata de un archivo instalable que contiene el código fuente de su módulo y, en este caso, se denomina **vsearch-1.0.tar.gz**.

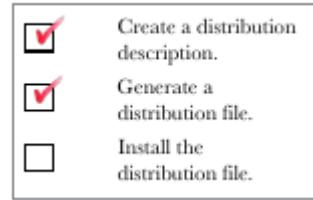
Encontrará su archivo recién creado en una carpeta denominada **dist**, que también ha sido creada por **setuptools** bajo la carpeta en la que está trabajando (que es **mymodules** en nuestro caso).

**Con su archivo de distribución de origen creado (como ZIP o como archivo comprimido de tar), ya está listo para instalar su módulo en paquetes de sitio.**

*listo para instalar*

## ***Installing Packages with “pip”***

### ***Instalación de paquetes con "pip"***



Ahora que su archivo de distribución existe como un ZIP o un archivo tarred (dependiendo de su plataforma), es el momento para el Paso 3: Instalar el archivo de distribución. Como con muchas cosas, Python viene con las herramientas para hacer esto sencillo. En particular, Python 3.4 (y más reciente) incluye una herramienta llamada pip, que es el instalador de paquetes para Python.

#### ***Paso 3 en Windows***

Busque su archivo ZIP recién creado bajo la carpeta **dist** (recuerde que el archivo se llama **vsearch-1.0.zip**). Mientras esté en el Explorador de Windows, mantenga presionada la tecla Mayús y, a continuación, haga clic con el botón derecho del ratón para abrir un menú contextual. Seleccione Abrir ventana de comandos aquí desde este menú. Se abre un nuevo símbolo del sistema de Windows. En este símbolo del sistema, escriba esta línea para completar el paso 3:

Ejecute Python 3 con el módulo pip y, a continuación, pregunte a pip para instalar el archivo ZIP identificado.

```
C:\Users\...\dist> py -3 -m pip install vsearch-1.0.zip
```

Si este comando falla con un error de permisos, es posible que deba reiniciar el símbolo del sistema como administrador de Windows y, a continuación, vuelva a intentarlo.

Cuando el comando anterior tiene éxito, los siguientes mensajes aparecen en la pantalla:

```
Processing c:\users\...\dist\vsearch-1.0.zip
Installing collected packages: vsearch
  Running setup.py install for vsearch
    Successfully installed vsearch-1.0
Success!
```

### **Paso 3 en sistemas operativos tipo UNIX**

En Linux, Unix o Mac OS X, abra un terminal dentro de la carpeta dict recientemente creada y, a continuación, emita este comando en el prompt:

Ejecute Python 3 con el módulo pip y, a continuación, pregunte a pip para instalar el archivo tar comprimido identificado.

```
.../dist$ sudo python3 -m pip install vsearch-1.0.tar.gz
```

Estamos utilizando el comando "sudo" aquí para asegurar que instalamos con los permisos correctos.

Cuando el comando anterior tiene éxito, los siguientes mensajes aparecen en la pantalla:

```
Processing ./vsearch-1.0.tar.gz
Installing collected packages: vsearch
  Running setup.py install for vsearch
    Successfully installed vsearch-1.0
```

Success!

**El módulo vsearch se instala ahora como parte de los site-packages.**

### **Módulos: Lo que ya sabemos**

Ahora que nuestro módulo vsearch ha sido instalado, podemos utilizar import vsearch en cualquiera de nuestros programas, con la seguridad de que el intérprete ahora puede encontrar las funciones del módulo cuando sea necesario.

Si posteriormente decidimos actualizar cualquier código del módulo, podemos repetir estos tres pasos para instalar cualquier actualización en los paquetes de sitio. Si produce una nueva versión de su módulo, asegúrese de asignar un nuevo número de versión dentro del archivo **setup.py**.

Tomemos un momento para resumir lo que ahora sabemos acerca de los módulos:

- Un módulo es una o más funciones guardadas en un archivo.
- Puede compartir un módulo asegurándose de que siempre está disponible con el directorio de trabajo actual del intérprete (que es posible, pero quebradizo) o dentro de los sitios de paquetes del intérprete (de lejos la mejor opción).
- Siguiendo el proceso de tres pasos de setuptools, asegura que su módulo se instale en paquetes de sitio, lo que le permite importar el módulo y utilizar sus funciones, independientemente de cuál sea su directorio de trabajo actual.

## ***Giving your code away (aka. Sharing)***

### ***Dar su código de distancia (aka. Compartir)***

Ahora que tiene un archivo de distribución creado, puede compartir este archivo con otros programadores de Python, permitiéndoles instalar su módulo usando pip, también. Puede compartir su archivo de dos maneras: informalmente o formalmente.

Para compartir su módulo de forma informal, simplemente distribuirlo de la forma que desee y de quien desee (tal vez utilizando correo electrónico, una memoria USB o una descarga de su sitio web personal). Depende de usted, realmente.

Para compartir su módulo de forma formal, puede cargar su archivo de distribución en el repositorio de software basado en la Web administrado de forma centralizada de Python, denominado PyPI, y abreviado para el índice de paquetes de Python. Este sitio existe para permitir que todo tipo de programadores de Python compartan todo tipo de módulos de Python de terceros. Para obtener más información sobre lo que se ofrece, visite el sitio PyPI en: <https://pypi.python.org/pypi>. Para obtener más información sobre el proceso de subir y compartir sus archivos de distribución a través de PyPI, lea la guía en línea mantenida por Python Packaging Authority, que encontrará aquí: <https://www.pypa.io>. (No hay mucho, pero los detalles están más allá del alcance de este libro.)

***Cualquier programador de Python también puede usar pip para instalar su módulo.***

Estamos casi terminados con nuestra introducción a funciones y módulos. Sólo hay un pequeño misterio que necesita nuestra atención (por no más de cinco minutos). Mueva la página cuando esté listo.

## ***Copia o referencia***

### ***El caso de los argumentos de la función de mala conducta***

Tom y Sarah han trabajado a través de este capítulo, y ahora están discutiendo sobre el comportamiento de los argumentos de la función.

Tom está convencido de que cuando los argumentos se pasan a una función, los datos se pasan **por valor**, y está escrito una pequeña función llamada double para ayudar a hacer su caso. La función double de Tom funciona con cualquier tipo de datos que se le proporcionen.

Aquí está el código de Tom:

```
def double(arg):
    print('Before: ', arg)
    arg = arg * 2
    print('After:  ', arg)
```

Sarah, por otra parte, está convencida de que cuando los argumentos se pasan a una función, los datos se pasan **por referencia**. Sarah también ha escrito una pequeña función, llamada change, que funciona con listas y ayuda a demostrar su punto.

```
def change(arg):
    print('Before: ', arg)
    arg.append('More data')
    print('After:  ', arg)
```

Prefiero que nadie discuta sobre este tipo de cosas, ya que hasta ahora, Tom y Sarah han sido los mejores amigos de programación. Para ayudar a resolver esto, vamos a experimentar en el prompt >>> en un intento de ver quién tiene razón: por el valor "Tom, o" por referencia "Sarah. No pueden tener razón, ¿verdad? Ciertamente es un poco un misterio que necesita ser resuelto, lo que lleva a esta pregunta tan frecuente:

### **¿Los argumentos de función soportados por-value o by-reference llaman semántica en Python?**

En caso de que necesite una actualización rápida, tenga en cuenta que el argumento de by-value que pasa se refiere a la práctica de usar el valor de una variable en lugar del argumento de una función. Si el valor cambia en el conjunto de la función, no tiene ningún efecto en el valor de la variable en el código que llamó a la función. Piense en el argumento como una copia del valor de la variable original. By-reference argumento de paso (a veces se refiere como by-address argumento de paso) mantiene un enlace a la variable en el código que llamó a la función. Si se cambia la variable en la suite de la función, también cambia el valor en el código que llamó a la función. Piense en el argumento como un alias a la variable original.

## Demostración de la semántica llamada por valor

Para averiguar qué están discutiendo Tom y Sarah, vamos a poner sus funciones en su propio módulo, que llamaremos `mystery.py`. Aquí está el módulo en una ventana de edición IDLE:

The screenshot shows a Python code editor window titled "mystery.py - /Users/Paul/Desktop/\_NewBook/ch04/mystery.py (3.5.0)". The code contains two functions:

```
def double(arg):
    print('Before: ', arg)
    arg = arg * 2
    print('After: ', arg)

def change(arg):
    print('Before: ', arg)
    arg.append('More data')
    print('After: ', arg)
```

Annotations with arrows explain the behavior of each function:

- An annotation on the left points to both functions with the text: "These two functions are similar. Each takes a single argument, displays it on screen, manipulates its value, and then displays it on screen again."
- An annotation for the `double` function points to the assignment `arg = arg * 2` with the text: "This function doubles the value passed in."
- An annotation for the `change` function points to the line `arg.append('More data')` with the text: "This function appends a string to any passed in list."

Tan pronto como Tom vea este módulo en la pantalla, se sienta, toma el control del teclado, presiona F5 y luego escribe lo siguiente en el prompt `>>>` de IDLE. Una vez hecho esto, Tom se recuesta en su silla, cruza los brazos y dice: "¿Ves? Te dije que es llamada-por-valor". Echa un vistazo a las interacciones de la shell de Tom con su función:

The screenshot shows a Python shell with the following interactions:

```
>>> num = 10
>>> double(num)
Before: 10
After: 20
>>> num
10
>>> saying = 'Hello '
>>> double(saying)
Before: Hello
After: Hello Hello
>>> saying
'Hello '
>>> numbers = [ 42, 256, 16 ]
>>> double(numbers)
Before: [42, 256, 16]
After: [42, 256, 16, 42, 256, 16]
>>> numbers
[42, 256, 16]
```

Annotations with arrows explain the results:

- An annotation on the left points to the first invocation with the text: "Tom invokes the 'double' function three times: once with an integer value, then with a string, and finally with a list."
- An annotation on the right points to the first invocation with the text: "Each invocation confirms that the value passed in as an argument is changed within the function's suite, but that the value at the shell remains unchanged. That is, the function arguments appear to conform to call-by-value semantics."

Tom invoca tres veces la función "doble": una vez con un valor entero, luego con una cadena y finalmente con una lista.

Cada invocación confirma que el valor pasado como argumento se cambia dentro del conjunto de la función, pero que el valor en el shell permanece sin cambios. Es decir: los argumentos de la función parecen estar de acuerdo con la semántica de llamada por valor.

## Demonstración de la semántica Call by-Reference

Sin dejarse intimidar por la aparente caída de golpe de Tom, Sarah se sienta y toma el control del teclado en preparación para interactuar con la shell. Aquí está el código en la ventana de edición de IDLE una vez más, con la función de change de Sarah lista para la acción:

```
mystery.py - /Users/Paul/Desktop/_NewBook/ch04/mystery.py (3.5.0)

def double(arg):
    print('Before: ', arg)
    arg = arg * 2
    print('After: ', arg)

def change(arg):
    print('Before: ', arg)
    arg.append('More data')
    print('After: ', arg)
```

Sarah escribe unas líneas de código en el mensaje, luego se inclina hacia atrás en su silla, cruza los brazos y le dice a Tom: "Bueno, si Python sólo soporta call by value, ¿cómo explicas este comportamiento? "Tom está sin palabras.

Echa un vistazo a la interacción de Sarah con la shell:

Utilizando los  
mismos datos  
de lista que  
Tom, Sarah  
invoca su  
función de  
'cambio'.

```
>>> numbers = [ 42, 256, 16 ]
>>> change(numbers)
Before: [42, 256, 16]
After: [42, 256, 16, 'More data']
>>> numbers
[42, 256, 16, 'More data']
```

Mira lo que ha  
pasado! Esta vez  
el valor del  
argumento se ha  
cambiado tanto en  
la función como  
en la shell. Esto  
parecería sugerir  
que las funciones  
de Python \* also  
\* admiten la  
semántica de  
llamada por  
referencia.

Este es un comportamiento extraño.

La función de Tom muestra claramente la semántica del argumento call-by-value, mientras que la función de Sarah demuestra call-by-reference.

¿Cómo puede ser esto? ¿Qué está pasando aquí? ¿Python soporta ambos?

### ***Resuelto: el caso de los argumentos de la función de mala conducta***

**¿Los argumentos de la función Python soportan semántica de llamada por valor o por referencia?**

Aquí está el pateador: tanto Tom como Sarah tienen razón. Dependiendo de la situación, la semántica de argumentos de función Python admite tanto la llamada por el valor como la llamada por referencia.

Recordemos una vez más que las variables en Python no son variables, ya que estamos acostumbrados a pensar en ellas en otros lenguajes de programación; Las variables son referencias a objetos. Es útil pensar que el valor almacenado en la variable es la dirección de memoria del valor, no su valor real. Es esta dirección de memoria que se pasa a una función, no el valor real. Esto significa que las funciones de Python admiten lo que es más correctamente llamado semántica de llamada de objeto-referencia.

En función del tipo de objeto al que se hace referencia, la semántica de llamadas real que se aplica en cualquier momento puede diferir. Entonces, ¿cómo es posible que en las funciones de Tom y Sarah los argumentos parecieran conformarse a la semántica de los valores de by-value y de referencia? En primer lugar, no lo hicieron, sólo aparecieron. Lo que realmente sucede es que el intérprete mira el tipo del valor referido por la referencia de objeto (la dirección de memoria) y, si la variable se refiere a un valor mutable, se aplica la semántica de llamada por referencia. Si el tipo de datos a los que se hace referencia es inmutable, la semántica de llamada por valor comienza. Considere ahora lo que esto significa para nuestros datos.

Las listas, los diccionarios y los conjuntos (que son mutables) se pasan siempre a una función por referencia; cualquier cambio realizado en la estructura de datos de la variable dentro del conjunto de la función se refleja en el código de llamada. Después de todo, los datos son mutables.

Las cadenas, enteros y tuplas (que son inmutables) siempre se pasan a una función por valor- cualquier cambio en la variable dentro de la función son privados para la función y no se reflejan en el código de llamada. Como los datos son inmutables, no pueden cambiar.

Lo cual tiene sentido hasta que consideres esta línea de código:

`arg = arg * 2`

¿Cómo es que esta línea de código parecía cambiar una lista pasada dentro de la suite de la función, pero cuando la lista se visualizaba en el shell después de la invocación, la lista no había cambiado (lo que llevó a Tom a creer- incorrectamente- A llamar por valor)? En la cara de las cosas, esto parece un error en el intérprete, como acabamos de decir que los cambios a un valor mutable se reflejan de nuevo en el código de llamada, pero no están aquí. Es decir, la función de Tom no cambió la lista de números en el código de llamada, aunque las listas son mutables. Entonces, ¿qué da?

Para entender lo que ha ocurrido aquí, considere que la línea de código anterior es una declaración de asignación. Esto es lo que sucede durante la asignación: el código a la derecha del símbolo `=` se ejecuta primero y luego cualquier valor creado tiene su referencia de objeto asignada a la variable a la izquierda del símbolo `=`. Al ejecutar el código `arg * 2` se crea un nuevo valor, al que se le asigna una nueva referencia de objeto, que se asigna a la variable `arg`, sobrescribiendo la referencia de objeto anterior almacenada en `arg` en la suite de la función. Sin embargo, la referencia de objeto "antiguo" todavía existe en el código de llamada y su valor no ha cambiado, por lo que el shell todavía ve la lista original, no la nueva lista duplicada creada en el código de Tom. Contraste este comportamiento con el código de Sarah, que llama al método `append` en una lista existente. Como no hay asignación aquí, no hay sobrescritura de referencias de objetos, por lo que el código de Sarah cambia la lista en el shell, también, ya que tanto la lista referida en el conjunto de funciones y la lista referida en el código de llamada tienen la misma referencia de objeto .

Con nuestro misterio resuelto, estamos casi listos para el **Capítulo 5**. Sólo hay un problema pendiente.

**¿Qué pasa con pep 8?**

**¿Puedo probar el cumplimiento de PEP 8?**

Tengo una pregunta rápida antes de seguir adelante. Me gusta la idea de escribir PEP 8 código compatible ... ¿hay alguna manera que puedo comprobar automáticamente mi código de cumplimiento?

## Sí. Es posible.

Pero no con Python solo, ya que el intérprete de Python no proporciona ninguna forma de comprobar el código para el cumplimiento de PEP 8. Sin embargo, hay una serie de herramientas de terceros que lo hacen.

Antes de saltar al capítulo 5, vamos a dar un pequeño desvío y mirar una herramienta que puede ayudarle a mantenerse en el lado derecho del cumplimiento de PEP 8.

## ***Preparándose para comprobar el cumplimiento de PEP8***

Vamos a desviarnos por un momento para comprobar nuestro código de conformidad con PEP 8.

La comunidad de programación de Python en general ha dedicado una gran cantidad de tiempo a crear herramientas de desarrollo para hacer las vidas de los programadores de Python un poco mejor. Una de estas herramientas es pytest, que es un framework de pruebas diseñado principalmente para facilitar la prueba de los programas de Python. No importa qué tipo de pruebas que está escribiendo, pytest puede ayudar. Y puede agregar complementos a pytest para ampliar sus capacidades.

Uno de estos plugins es pep8, que utiliza el framework de prueba de pytest para revisar su código por violaciones de las pautas de PEP 8.

## ***Recordando nuestro código***

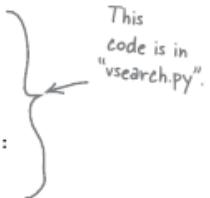
Recordemos nuestro código vsearch.py una vez más, antes de alimentarlo a la combinación pytest / pep8 para descubrir cómo es compatible con PEP 8. Tenga en cuenta que tendremos que instalar ambas herramientas de desarrollo, ya que no vienen instaladas con Python (lo haremos en la página).

Uno más, aquí está el código para el módulo **vsearch.py**, que se va a comprobar para el cumplimiento de las directrices PEP 8:

Obtenga más información sobre pytest desde <http://doc.pytest.org/en/latest/>.

```
def search4vowels(phrase:str) -> set:
    """Return any vowels found in a supplied phrase."""
    vowels = set('aeiou')
    return vowels.intersection(set(phrase))

def search4letters(phrase:str, letters:str='aeiou') -> set:
    """Return a set of the 'letters' found in 'phrase'."""
    return set(letters).intersection(set(phrase))
```



This code is in "vsearch.py".

## **Instalando pytest y el complemento pep8**

Anteriormente en este capítulo, utilizó la herramienta pip para instalar su módulo vsearch.py en el intérprete de Python en su computadora. La herramienta pip también se puede utilizar para instalar código de terceros en su intérprete.

Para ello, debe operar en el símbolo del sistema operativo (y estar conectado a Internet). Utilizará pip en el siguiente capítulo para instalar una biblioteca de terceros. Por ahora, sin embargo, vamos a usar pip para instalar el framework de prueba pytest y el plugin pep8.

### **Prueba más**

#### **Intro Instalar las herramientas de desarrollo de prueba**

En las pantallas de ejemplo siguientes, mostramos los mensajes que aparecen cuando se está ejecutando en la plataforma Windows. En Windows, invoca a Python 3 utilizando el comando py -3. Si está en Linux o Mac OS X, reemplace el comando Windows con sudo python3. Para instalar pytest utilizando pip en Windows, emita este comando desde el símbolo del sistema mientras se ejecuta como administrador (busque cmd.exe, haga clic con el botón derecho en él y elija Ejecutar como administrador en el menú emergente):

```
py -3 -m pip install pytest
Start in Administrator mode...
...then issue the "pip" command to install "pytest"...
...then check whether it installed successfully.

Administrator: Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>py -3 -m pip install pytest
Collecting pytest
  Downloading pytest-2.8.7-py2.py3-none-any.whl (151kB)
    100% :#####: 155kB 1.3MB/s
Collecting colorama (from pytest)
  Downloading colorama-0.3.6-py2.py3-none-any.whl
Collecting py>=1.4.29 (from pytest)
  Downloading py-1.4.31-py2.py3-none-any.whl (81kB)
    100% :#####: 86kB 131kB/s
Installing collected packages: colorama, py, pytest
Successfully installed colorama-0.3.6 py-1.4.31 pytest-2.8.7
C:\Windows\system32>
```

Si examina los mensajes producidos por pip, notará que también se instalaron dos de las dependencias de pytest (colorama y py). Lo mismo sucede cuando usas pip para instalar el complemento pep8: también instala un montón de dependencias. Este es el comando para instalar el plug-in:

```
py -3 -m pip install pytest-pep8
```

Recuerde: si no está ejecutando Windows, reemplace "py -3" por "sudo python3".

While still in Administrator mode, issue this command, which installs the "pep8" plug-in.

This command succeeded too, and also installed the required dependencies.

```
c:\Windows\system32>py -3 -m pip install pytest-pep8
Collecting pytest-pep8
  Downloading pytest-pep8-1.0.6.tar.gz
    Collecting pytest-cache (from pytest-pep8)
      Downloading pytest-cache-1.0.tar.gz
        Requirement already satisfied (use --upgrade to upgrade): pytest>=2.4.2 in c:\program files\python 3.5\lib\site-packages (from pytest-pep8)
      Collecting pep8>=1.3 (from pytest-pep8)
        Downloading pep8-1.7.0-py2.py3-none-any.whl (41kB)
          100% ##### 45kB 174kB/s
      Collecting execnet>=1.1.dev1 (from pytest-cache->pytest-pep8)
        Downloading execnet-1.4.1-py2.py3-none-any.whl (40kB)
          100% ##### 40kB 174kB/s
        Requirement already satisfied (use --upgrade to upgrade): py>=1.4.29 in c:\program files\python 3.5\lib\site-packages (from pytest>=2.4.2->pytest-pep8)
        Requirement already satisfied (use --upgrade to upgrade): colorama in c:\program files\python 3.5\lib\site-packages (from pytest>=2.4.2->pytest-pep8)
      Collecting apipkg>=1.4 (from execnet>=1.1.dev1->pytest-cache->pytest-pep8)
        Downloading apipkg-1.4-py2.py3-none-any.whl
      Installing collected packages: apipkg, execnet, pytest-cache, pep8, pytest-pep8
        Running setup.py install for pytest-cache ... done
        Running setup.py install for pytest-pep8 ... done
        Successfully installed apipkg-1.4 execnet-1.4.1 pep8-1.7.0 pytest-cache-1.0 pytest-pep8-1.0.6
C:\Windows\system32>
```

### En linux manjaro:

para ejecutarlo tenemos que hacerlo como superusuario., entonces instalamos sudo en manjaro asi:

```
[anton@anton-pc ~]$ sudo pacman -S sudo
advertencia: sudo-1.8.19.p2-1 está actualizado -- reinstalándolo
resolviendo dependencias...
buscando conflictos entre paquetes...
```

Paquetes (1) sudo-1.8.19.p2-1

```
Tamaño total de la descarga: 0,93 MiB
Tamaño total de la instalación: 3,96 MiB
Tamaño neto tras actualizar: 0,00 MiB
```

```
:: ¿Continuar con la instalación? [S/n] s
:: Recibiendo los paquetes...
sudo-1.8.19.p2-1-i686 948,9 KiB 463K/s 00:02 [#####] 100%
(1/1) comprobando las claves del depósito [#####] 100%
(1/1) verificando la integridad de los paquetes [#####] 100%
(1/1) cargando los archivos de los paquetes [#####] 100%
(1/1) comprobando conflictos entre archivos [#####] 100%
(1/1) comprobando el espacio disponible en el ... [#####] 100%
:: Procesando los cambios de los paquetes...
(1/1) reinstalando sudo [#####] 100%
:: Ejecutando los «hooks» de posinstalación...
(1/2) Creating temporary files...
```

## (2/2) Arming ConditionNeedsUpdate...

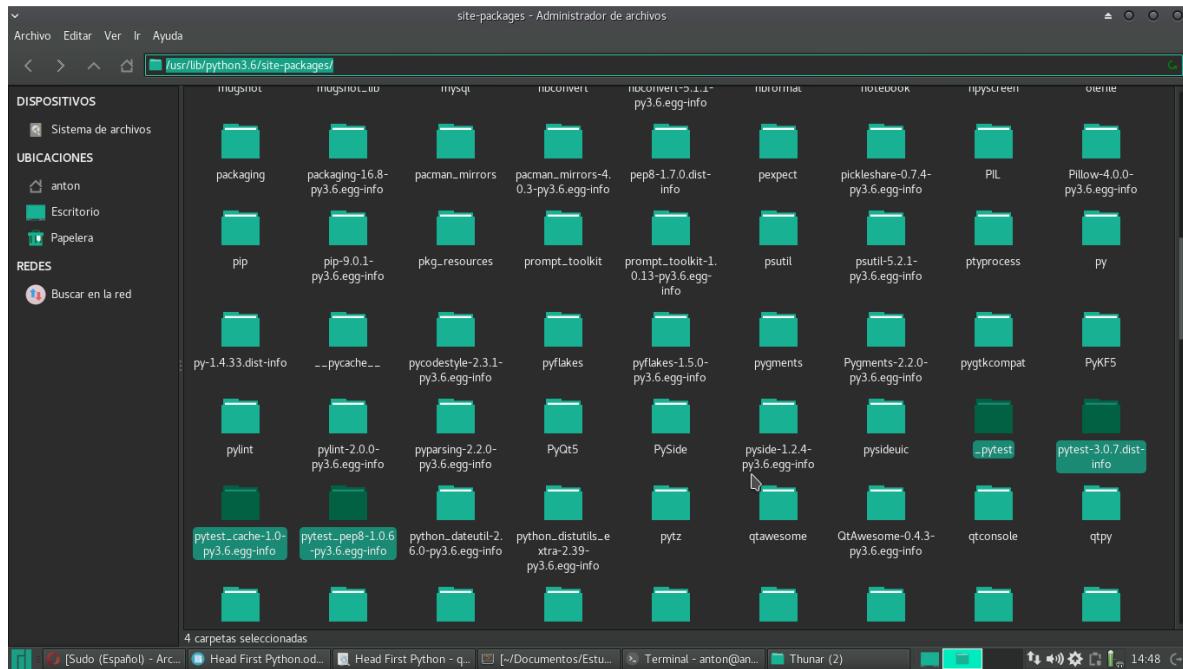
Ya tenemos instalado el sudo en Manjaro ahora podemos instalar el pytest desde pip

```
[anton@anton-pc ~]$ sudo python -m pip install pytest
Collecting pytest
  Downloading pytest-3.0.7-py2.py3-none-any.whl (172kB)
    100% |██████████| 174kB 547kB/s
Collecting py>=1.4.29 (from pytest)
  Downloading py-1.4.33-py2.py3-none-any.whl (83kB)
    100% |██████████| 92kB 2.1MB/s
Requirement already satisfied: setuptools in /usr/lib/python3.6/site-packages (from pytest)
Requirement already satisfied: packaging>=16.8 in /usr/lib/python3.6/site-packages (from setuptools->pytest)
Requirement already satisfied: six>=1.6.0 in /usr/lib/python3.6/site-packages (from setuptools->pytest)
Requirement already satisfied: appdirs>=1.4.0 in /usr/lib/python3.6/site-packages (from setuptools->pytest)
Requirement already satisfied: pyparsing in /usr/lib/python3.6/site-packages (from packaging>=16.8->setuptools->pytest)
Installing collected packages: py, pytest
Successfully installed py-1.4.33 pytest-3.0.7
[anton@anton-pc ~]$
```

Ahora instalamos pytest-pep8

```
[anton@anton-pc ~]$ sudo python -m pip install pytest-pep8
[sudo] password for anton:
Collecting pytest-pep8
  Downloading pytest-pep8-1.0.6.tar.gz
Collecting pytest-cache (from pytest-pep8)
  Downloading pytest-cache-1.0.tar.gz
Requirement already satisfied: pytest>=2.4.2 in /usr/lib/python3.6/site-packages (from pytest-pep8)
Collecting pep8>=1.3 (from pytest-pep8)
  Downloading pep8-1.7.0-py2.py3-none-any.whl (41kB)
    100% |██████████| 51kB 321kB/s
Collecting execnet>=1.1.dev1 (from pytest-cache->pytest-pep8)
  Downloading execnet-1.4.1-py2.py3-none-any.whl (40kB)
    100% |██████████| 40kB 1.1MB/s
Requirement already satisfied: setuptools in /usr/lib/python3.6/site-packages (from pytest>=2.4.2->pytest-pep8)
Requirement already satisfied: py>=1.4.29 in /usr/lib/python3.6/site-packages (from pytest>=2.4.2->pytest-pep8)
Collecting apipkg>=1.4 (from execnet>=1.1.dev1->pytest-cache->pytest-pep8)
  Downloading apipkg-1.4-py2.py3-none-any.whl
Requirement already satisfied: packaging>=16.8 in /usr/lib/python3.6/site-packages (from setuptools->pytest>=2.4.2->pytest-pep8)
Requirement already satisfied: six>=1.6.0 in /usr/lib/python3.6/site-packages (from setuptools->pytest>=2.4.2->pytest-pep8)
Requirement already satisfied: appdirs>=1.4.0 in /usr/lib/python3.6/site-packages (from setuptools->pytest>=2.4.2->pytest-pep8)
Requirement already satisfied: pyparsing in /usr/lib/python3.6/site-packages (from packaging>=16.8->setuptools->pytest>=2.4.2->pytest-pep8)
Installing collected packages: apipkg, execnet, pytest-cache, pep8, pytest-pep8
  Running setup.py install for pytest-cache ... done
  Running setup.py install for pytest-pep8 ... done
Successfully installed apipkg-1.4 execnet-1.4.1 pep8-1.7.0 pytest-cache-1.0 pytest-pep8-1.0.6
[anton@anton-pc ~]$
```

Comprobamos si se instaló correctamente en site-packages de python:  
en la ruta: /usr/lib/python3.6/site-packages/



Y si instalado corectamente.

## ¿Cómo es nuestro código el PEP 8-Compliant?

Con pytest y pep8 instalado, ya está listo para probar su código para el cumplimiento de PEP 8. Independientemente del sistema operativo que esté utilizando, emitirá el mismo comando (ya que sólo las instrucciones de instalación difieren en cada plataforma).

El proceso de instalación de pytest ha instalado un nuevo programa en su computadora llamado **py.test**. Vamos a ejecutar este programa ahora para comprobar nuestro código **vsearch.py** para el cumplimiento de PEP 8. Asegúrese de que se encuentra en la misma carpeta que el que contiene el archivo **vsearch.py** y, a continuación, emita este comando:

```
py.test --pep8 vsearch.py
```

Esta es la salida producida cuando hicimos esto en nuestro equipo con Windows:

UH oh. El resultado rojo no puede ser bueno, ¿no?

A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The command entered is 'E:\\_NewBook\ch04>py.test --pep8 vsearch.py'. The output shows a test session starting with Python 3.5.0, pytest-2.8.7, py-1.4.31, and pluggy-0.3.1. It collects 1 item from 'vsearch.py' and marks it as failing ('F'). A red arrow points from the word 'FAILURES' in the output to the 'F' in 'vsearch.py F'. The output then lists five PEP8-check errors (E231) related to whitespace issues in the 'search4vowels' and 'search4letters' functions. Finally, it states '1 failed in 0.05 seconds'.

```
E:\_NewBook\ch04>py.test --pep8 vsearch.py
=====
platform win32 -- Python 3.5.0, pytest-2.8.7, py-1.4.31, pluggy-0.3.1
rootdir: E:\_NewBook\ch04, inifile:
plugins: pep8-1.0.6
collected 1 items

vsearch.py F
=====
    FAILURES
    PEP8-check
E:\_NewBook\ch04\vsearch.py:2:25: E231 missing whitespace after ':'
def search4vowels(phrase:str) -> set:
^
E:\_NewBook\ch04\vsearch.py:3:56: W291 trailing whitespace
    """Return any vowels found in a supplied phrase."""
^
E:\_NewBook\ch04\vsearch.py:7:1: E302 expected 2 blank lines, found 1
def search4letters(phrase:str, letters:str='aeiou') -> set:
^
E:\_NewBook\ch04\vsearch.py:7:26: E231 missing whitespace after ':'
def search4letters(phrase:str, letters:str='aeiou') -> set:
^
E:\_NewBook\ch04\vsearch.py:7:39: E231 missing whitespace after ':'
def search4letters(phrase:str, letters:str='aeiou') -> set:
^

===== 1 failed in 0.05 seconds =====
E:\_NewBook\ch04>
```

¡Vaya! Parece que tenemos fallas, lo que significa que este código no es tan compatible con las directrices PEP 8 como podría ser.

Tómese un momento para leer los mensajes que se muestran aquí (o en su pantalla, si usted está siguiendo adelante). Todos los "fallos" parecen referirse - de alguna manera - a espacios en blanco (por ejemplo, espacios, pestañas, líneas de nuevo y similares). Echemos un vistazo a cada uno de ellos en un poco más de detalle.

### **Comprendiendo los mensajes de error**

Juntos, pytest y el complemento pep8 han resaltado cinco problemas con nuestro código vsearch.py.

La primera cuestión tiene que ver con el hecho de que no hemos insertado un espacio después del carácter: al anotar los argumentos de nuestra función, y lo hemos hecho en tres lugares. Mire el primer mensaje, anotando el uso de pytest del carácter de intercalación (^) para indicar exactamente dónde está el problema:

The diagram shows the first error message from the terminal output: '...:2:25: E231 missing whitespace after ':''. An annotation with an arrow points to the character '^' in the line 'def search4vowels(phrase:str) -> set:', indicating the position of the missing whitespace. Another annotation with an arrow points to the word 'set:' with the text 'Here's where it's wrong.' A third annotation with an arrow points to the word 'what's' with the text 'Here's what's wrong.'

```
...:2:25: E231 missing whitespace after ':'
def search4vowels(phrase:str) -> set:
^
Here's where
it's wrong.
Here's what's
wrong.
```

Si observa los dos problemas al final de la salida de pytest, verá que hemos repetido este error en tres ubicaciones: una en la línea 2 y dos en la línea 7. Hay una solución fácil: agregue un solo carácter de espacio después del colon.

La próxima edición puede no parecer una gran cosa, pero se plantea como un fracaso porque la línea de código en cuestión (línea 3) rompe una directiva PEP 8 que dice no incluir espacios adicionales al final de las líneas:

Que esta mal

```
...:3:56: W291 trailing whitespace
"""Return any vowels found in a supplied phrase."""
^
Where it's wrong
```

Tratar este problema en la línea 3 es otra solución fácil: eliminar todos los espacios en blanco. El último número (al comienzo de la línea 7) es el siguiente:

```
...7:1: E302 expected 2 blank lines, found 1
def search4letters(phrase:str, letters:str='aeiou') -> set:
^
This issue presents at the start of line 7.
^
Here's what's wrong.
```

Hay una guía PEP 8 que ofrece este consejo para crear funciones en un módulo: Surround de alto nivel de funciones y definiciones de clases con dos líneas en blanco. En nuestro código, las funciones search4vowels y search4letters están en el "nivel superior" del archivo vsearch.py y están separadas entre sí por una sola línea en blanco. Para ser compatible con PEP 8, debe haber dos líneas en blanco aquí.

Una vez más, es una solución fácil: inserte una línea en blanco extra entre las dos funciones. Vamos a aplicar estas correcciones ahora, luego vuelva a probar nuestro código modificado.

### ***Confirmación del cumplimiento de PEP 8***

Con las modificaciones hechas en el código de Python en vsearch.py, el contenido del archivo ahora se parece a esto:

```

def search4vowels(phrase: str) -> set:
    """Return any vowels found in a supplied phrase."""
    vowels = set('aeiou')
    return vowels.intersection(set(phrase))

def search4letters(phrase: str, letters: str='aeiou') -> set:
    """Return a set of the 'letters' found in 'phrase'."""
    return set(letters).intersection(set(phrase))

```

La versión compatible con PEP 8 de "vsearch.py".

Cuando esta versión del código se ejecuta a través del plugin pep8 de pytest, la salida confirma que ya no tenemos problemas con el cumplimiento de PEP 8. Esto es lo que vimos en nuestro ordenador (de nuevo, se ejecuta en Windows):

El verde es bueno, este código no tiene PEP 8 problemas.

```

C:\Windows\system32\cmd.exe
E:\_NewBook\ch04>py.test --pep8 vsearch.py
===== test session starts =====
platform win32 -- Python 3.5.0, pytest-2.8.7, py-1.4.31, pluggy-0.3.1
rootdir: E:\_NewBook\ch04, inifile:
plugins: pep8-1.0.6
collected 1 items

vsearch.py .

===== 1 passed in 0.06 seconds =====
E:\_NewBook\ch04>

```

*En linux fue así:* primero nos ubicamos en la carpeta donde está el archivo a ejecutar:

```

[anton@anton-pc ~]$ cd /home/anton/pruebasparacodigo/mymodules/
[anton@anton-pc mymodules]$ ls
__pycache__ vsearch.py
[anton@anton-pc mymodules]$ py.test --pep8 vsearch.py
===== test session starts =====
platform linux -- Python 3.6.0, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: /home/anton/pruebasparacodigo/mymodules, inifile:
plugins: pep8-1.0.6
collected 1 items

vsearch.py .

===== 1 passed in 0.05 seconds =====
[anton@anton-pc mymodules]$ 

```

Como vemos no presenta errores , porque ya se habia modificado el archivo para su uso en windows.

## La conformidad con PEP 8 es una buena cosa

Si usted está mirando todo esto preguntándose qué es todo el alboroto (especialmente sobre un poco de espacio en blanco), piense cuidadosamente acerca de por qué desea cumplir con PEP 8. La documentación PEP 8 indica que la legibilidad cuenta y Ese código se lee mucho más a menudo de lo que está escrito. Si su código se ajusta a un estilo de codificación estándar, se sigue que la lectura es más fácil, ya que "parece" todo lo que el programador ha visto. La consistencia es una cosa muy buena.

A partir de este punto (y tanto como sea práctico), todo el código de este libro se ajustará a las pautas del PEP 8. Usted debe tratar de asegurar su código también.

Este es el final del desvío pytest. Nos vemos en el capítulo 5.

## Chapter 4's Code

```
def search4vowels(phrase: str) -> set:  
    """Returns the set of vowels found in 'phrase'. """  
    return set('aeiou').intersection(set(phrase))  
  
def search4letters(phrase: str, letters: str='aeiou') -> set:  
    """Returns the set of 'letters' found in 'phrase'. """  
    return set(letters).intersection(set(phrase))
```

This is the code  
from the "vsearch.py"  
module, which contains  
our two functions:  
"search4vowels" and  
"search4letters".

This is the "setup.  
py" file, which  
allowed us to  
turn our module  
into an installable  
distribution.

```
from setuptools import setup  
  
setup(  
    name='vsearch',  
    version='1.0',  
    description='The Head First Python Search Tools',  
    author='HF Python 2e',  
    author_email='hfpy2e@gmail.com',  
    url='headfirstlabs.com',  
    py_modules=['vsearch'],  
)
```

```
def double(arg):
    print('Before: ', arg)
    arg = arg * 2
    print('After:  ', arg)

def change(arg: list):
    print('Before: ', arg)
    arg.append('More data')
    print('After:  ', arg)
```

And this is the "mystery.py" module, which had Tom and Sarah upset at each other. Thankfully, now that the mystery is solved, they are back to being programming buddies once more. ☺



# **5 building a webapp**

## ***La construcción de una aplicación web***

***En esta etapa, usted sabe bastante Python para ser peligroso. ?***

Con los primeros cuatro capítulos de este libro detrás de usted, usted ahora está en una posición para utilizar de manera productiva Python dentro de cualquier número de áreas de aplicación (aunque todavía hay un montón de Python para aprender). En lugar de explorar la larga lista de lo que son estas áreas de aplicación, en este y los subsiguientes capítulos, vamos a estructurar nuestro aprendizaje en torno al desarrollo de una aplicación alojada en la web, que es un área donde Python es especialmente fuerte. En el camino, aprenderás un poco más acerca de Python. Antes de empezar, sin embargo, vamos a tener una recapitulación rápida del Python que ya sabes.

***Vamos a construir algo***

***Vamos a construir una aplicación web.***

En concreto, vamos a tomar nuestra función search4letters y hacerla accesible a través de la Web, permitiendo a cualquier persona con un navegador web acceder al servicio proporcionado por nuestra función.

Podríamos construir cualquier tipo de aplicación, pero la creación de una aplicación web funcional nos permite explorar una serie de características de Python mientras se construye algo que es generalmente útil, además de ser un montón más meatier que los fragmentos de código que has visto hasta ahora en este libro .

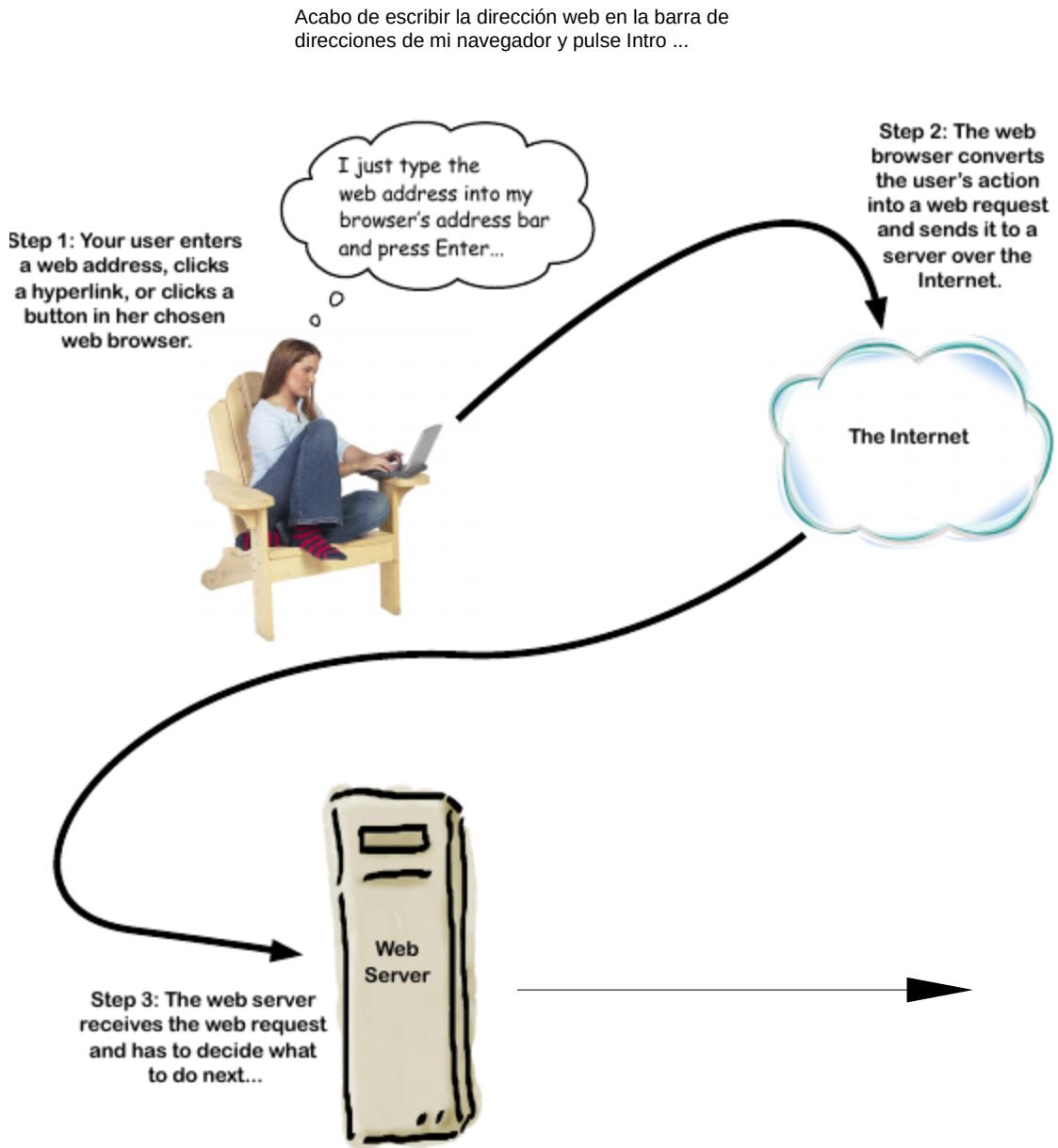
Python es particularmente fuerte en el lado del servidor de la Web, que es donde vamos a construir e implementar nuestra aplicación web en este capítulo.

Pero, antes de empezar, asegúrese de que todos estén en la misma página revisando cómo funciona la Web.

***Cómo funciona la web***

No importa lo que hagas en la Web, se trata de peticiones o request y respuestas o responses. Una solicitud web se envía desde un explorador web a un servidor web como resultado de alguna interacción del usuario. En el servidor web, una respuesta web (o

respuesta) se formula y devuelve al navegador web. Todo el proceso se puede resumir en cinco pasos, de la siguiente manera:



**Paso 1:** su usuario ingresa una dirección web, hace clic en un hipervínculo o hace clic en un botón en el navegador web elegido.

**Paso 2:** El navegador web convierte la acción del usuario en una solicitud web y la envía a un servidor a través de Internet.

**Paso 3:** El servidor web recibe la solicitud web y tiene que decidir qué hacer a continuación

...

### **Decidir qué hacer a continuación**

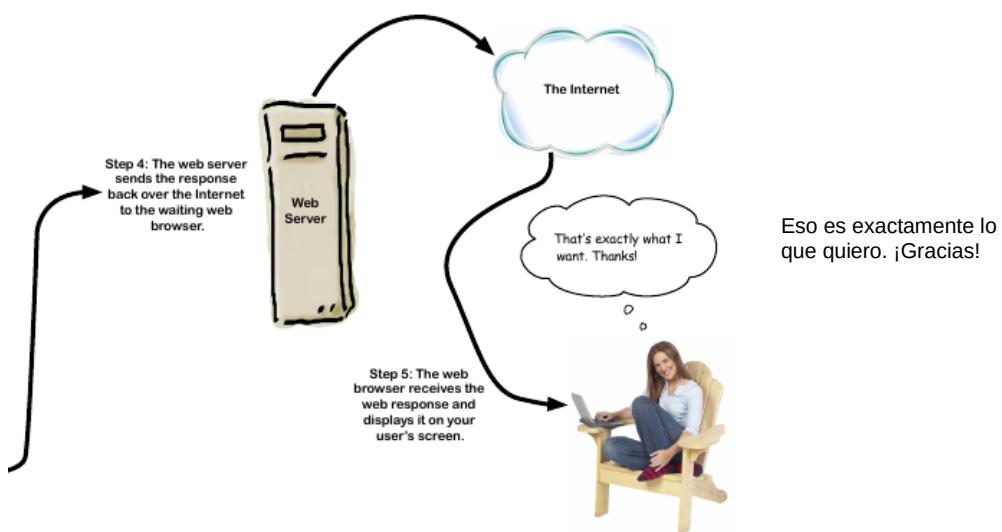
Una de dos cosas suceden en este punto. Si la solicitud web o web request es para **contenido estático**, como un archivo HTML, una imagen o cualquier otra cosa almacenada en el disco duro del servidor web, el servidor web localiza el recurso y se prepara para devolverlo al navegador web como una respuesta web o web response.

Si la solicitud es para **contenido dinámico**, es decir, para el contenido que se debe generar, como resultados de búsqueda o el contenido actual de una cesta de compras en línea, el servidor web ejecuta un código para producir la respuesta web.

### **Los (potencialmente) muchos subpasos del paso 3**

En la práctica, el Paso 3 puede implicar múltiples subpasos, dependiendo de lo que el servidor web deba hacer para producir la respuesta o response. Obviamente, si todo lo que el servidor tiene que hacer es localizar el contenido estático y devolverlo al navegador, los subpasos no son demasiado exigentes, ya que es solo cuestión de leer desde la unidad de disco del servidor web.

Sin embargo, cuando se debe generar contenido dinámico, las subetapas implican que el servidor web ejecute código y luego capturen la salida del programa como una respuesta o response web, antes de enviar la respuesta al explorador web en espera.



**Paso 4:** El servidor web devuelve la respuesta a través de Internet al explorador web en espera.

**Paso 5:** El navegador web recibe la respuesta web y la muestra en la pantalla de su usuario.

### La especificación webapp

## ¿Qué queremos que haga nuestro Webapp?

Tan tentador como siempre es comenzar a codificar, vamos a pensar primero en cómo nuestra aplicación web va a funcionar.

Los usuarios interactúan con nuestra aplicación web utilizando su navegador web favorito. Todo lo que tienen que hacer es ingresar la URL de la aplicación web en la barra de direcciones de su navegador para acceder a sus servicios. Una página web aparece en el navegador solicitando al usuario que proporcione argumentos a la función **search4letters**. Una vez que estos se introducen, el usuario hace clic en un botón para ver sus resultados.

Recuerde la línea def de nuestra versión más reciente de **search4letters**, que muestra la función que espera al menos uno, pero no más de dos argumentos: una **frase** para buscar, junto con las **letras** a buscar. Recuerde, el argumento de **letras** es opcional (predeterminado a aeiou):

La línea "def" para la función "search4letters" que toma uno, pero no, más de dos argumentos

```
 ↴  
def search4letters(phrase:str, letters:str='aeiou') -> set:
```

Vamos a tomar una servilleta de papel y esbozar cómo queremos que nuestra página web aparezca. Esto es lo que nos ocurrió:

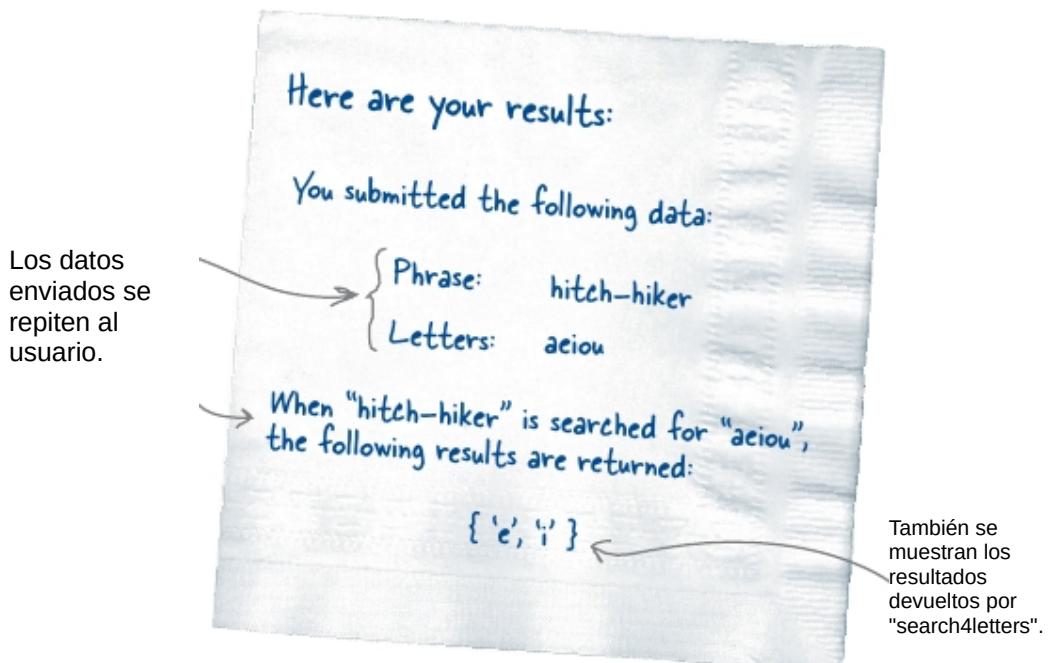


Al hacer clic en este botón se envían los datos del usuario a nuestro servidor web en espera.

## ¿Qué sucede en el servidor web?

Cuando el usuario hace clic en **Do it!**, El navegador envía los datos al servidor web en espera, que extrae los valores de **frase** y **letras**, antes de llamar a la función **search4letters** en nombre del usuario en espera.

Cualquier resultado de la función se devuelve al navegador del usuario como otra página web, que volvemos a esbozar en una servilleta de papel (mostrada a continuación). Por ahora, supongamos que el usuario introdujo "hitch-hiker" como la **frase** y dejó el valor de las **letras** predeterminado a **aeiou**. Esto es lo que la página web de resultados podría verse como:



## ¿Qué necesitamos para empezar?

A parte del conocimiento que ya tienes sobre Python, lo único que necesitas para crear una aplicación web de servidor en el servidor es un marco de aplicación web o **web application framework**, que proporciona un conjunto de tecnologías fundamentales generales en las que puedes construir tu aplicación web.

Aunque es más que posible usar Python para construir todo lo que necesitas desde cero, sería una locura contemplar hacerlo. Otros programadores ya han tomado el tiempo para construir estos marcos web para usted. Python tiene muchas opciones aquí. Sin embargo, no vamos a agonizar sobre qué marco para elegir, y en su lugar sólo va a elegir uno popular llamado **Flask** y seguir adelante.

## Vamos a instalar Flask

Sabemos desde el capítulo 1 que la biblioteca estándar de Python viene con un montón de baterías incluidas. Sin embargo, hay momentos en los que necesitamos utilizar un módulo de terceros específico de la aplicación, que no forma parte de la biblioteca estándar. Los módulos de terceros se importan en su programa Python según sea necesario. Sin embargo, a diferencia de los módulos de biblioteca estándar, los módulos de terceros deben instalarse antes de que se importen y utilicen. **Flask** es uno de estos módulos de terceros.

Como se mencionó en el capítulo anterior, la comunidad Python mantiene un sitio web gestionado de forma centralizada para los módulos de terceros llamados PyPI (abreviatura de Python Package Index), que alberga la última versión de Flask (así como muchos otros proyectos).

Recuerde cómo utilizamos pip para instalar nuestro módulo vsearch en Python anteriormente en este libro. Pip también funciona con PyPI. Si conoce el nombre del módulo que desea, puede utilizar pip para instalar cualquier módulo alojado en PyPI directamente en su entorno Python.

## Instalar Flask desde la línea de comandos con pip

Si se ejecuta en Linux o Mac OS X, escriba el siguiente comando en una ventana de terminal:

Utilice este comando en Mac OS X y Linux.

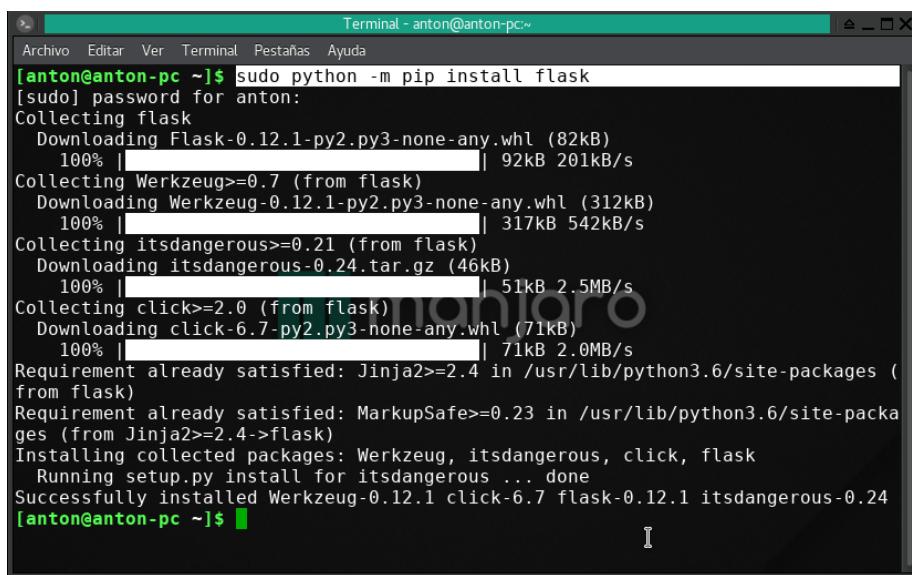
```
$ sudo -H python3 -m pip install flask
```

Nota: el caso es importante aquí. Esa es una minúscula "f" para "flask".

Si se ejecuta en Windows, abra un símbolo del sistema: asegúrese de Ejecutar como administrador (haciendo clic con el botón secundario en la opción y eligiendo en el menú emergente) y, a continuación, emita este comando:

```
C:\> py -3 -m pip install flask
```

Yo lo instalo en linux Manjaro:

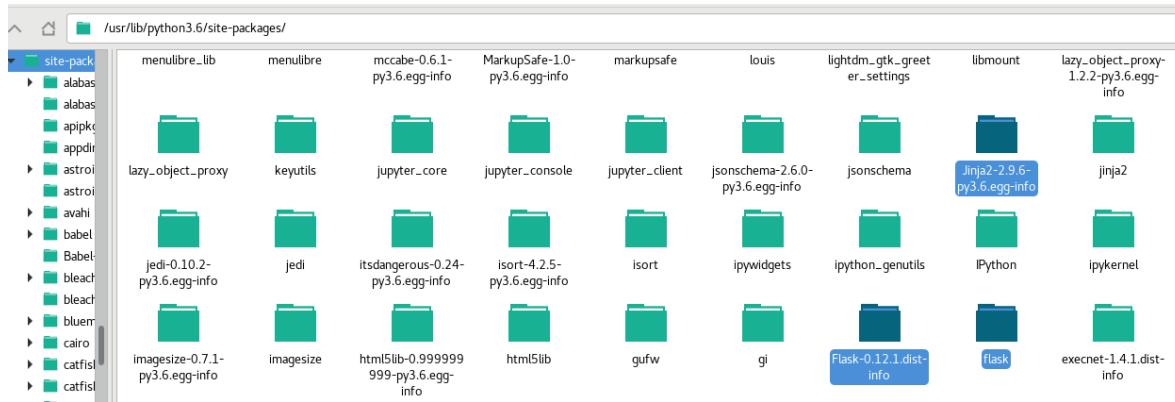


The screenshot shows a terminal window titled "Terminal - anton@anton-pc~". The user runs the command `sudo python -m pip install flask`. The terminal displays the progress of the download and installation of Flask and its dependencies (Werkzeug, Jinja2, MarkupSafe, etc.) from PyPI. The output includes file sizes and download speeds for each package.

```
[anton@anton-pc ~]$ sudo python -m pip install flask
[sudo] password for anton:
Collecting flask
  Downloading Flask-0.12.1-py2.py3-none-any.whl (82kB)
    100% |██████████| 92kB 201kB/s
Collecting Werkzeug>=0.7 (from flask)
  Downloading Werkzeug-0.12.1-py2.py3-none-any.whl (312kB)
    100% |██████████| 317kB 542kB/s
Collecting itsdangerous>=0.21 (from flask)
  Downloading itsdangerous-0.24.tar.gz (46kB)
    100% |██████████| 51kB 2.5MB/s
Collecting click>=2.0 (from flask)
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
    100% |██████████| 71kB 2.0MB/s
Requirement already satisfied: Jinja2>=2.4 in /usr/lib/python3.6/site-packages (from flask)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/lib/python3.6/site-packages (from Jinja2>=2.4->flask)
Installing collected packages: Werkzeug, itsdangerous, click, flask
  Running setup.py install for itsdangerous ... done
Successfully installed Werkzeug-0.12.1 click-6.7 flask-0.12.1 itsdangerous-0.24
[anton@anton-pc ~]$
```

Como observamos hemos escrito lo siguiente en el terminal:

```
[anton@anton-pc ~]$ sudo python -m pip install flask
```



Se comprobo su instalacion de forma correcta en la path=/usr/lib/python3.6/site-packages/

Este comando (independientemente de su sistema operativo) se conecta al sitio web PyPI, descarga e instala el módulo Flask y otros cuatro módulos. Flask depende de: Werkzeug, MarkupSafe, Jinja2 y itsdangerous. No se preocupe (por ahora) sobre lo que hacen estos módulos adicionales; Sólo asegúrese de que instalan correctamente. Si todo va bien, verás un mensaje similar al siguiente en la parte inferior de la salida generada por pip. Tenga en cuenta que la salida se ejecuta a más de una docena de líneas o menos: ...

```
Installing collected packages: Werkzeug, itsdangerous, click, flask
```

```
Running setup.py install for itsdangerous ... done
```

En el momento de la escritura, estos son los números de versión actuales asociados con estos módulos.

## ¿Cómo funciona el Flask?

Flask proporciona una colección de módulos que le ayudan a crear aplicaciones web de servidor. Es técnicamente un marco micro web, ya que proporciona el conjunto mínimo de tecnologías necesarias para esta tarea. Esto significa que Flask no es tan completo como algunos de sus competidores -como Django, la madre de todos los frameworks web de Python- pero es pequeño, ligero y fácil de usar.

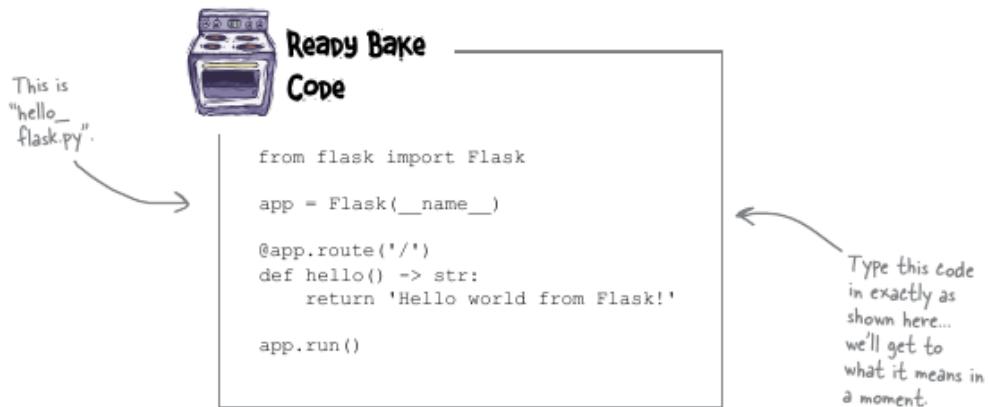
Como nuestros requisitos no son pesados (sólo tenemos dos páginas web), Flask es más que suficiente marco web para nosotros en este momento.

Django es un framework de aplicaciones web enormemente popular dentro de la comunidad de Python. Cuenta con una instalación de administración especialmente sólida y pre-construida que puede hacer que el trabajo con grandes aplicaciones web sea muy manejable. Es un exceso de lo que estamos haciendo aquí, por lo que hemos optado por la mucho más simple, pero más ligero, Flask.

### **Compruebe que el Flask esté instalado y funcionando**

Aquí está el código para el más básico de Flask webapps, que vamos a utilizar para probar que Flask está configurado y listo para funcionar.

Utilice su editor de texto favorito para crear un nuevo archivo y escriba el código que se muestra a continuación en el archivo, guardándolo como **hello\_flask.py** (puede guardar el archivo en su propia carpeta, también, si lo desea, llamamos a nuestra carpeta **webapp**) :



Escriba este código exactamente como se muestra aquí ... vamos a llegar a lo que significa en un momento.

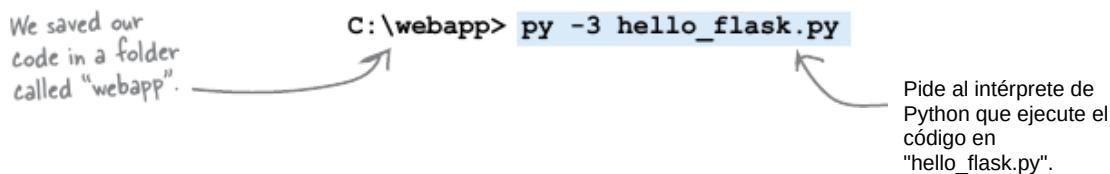
### **Ejecutar Flask desde la línea de comandos de OS**

No se sienta tentado a ejecutar este código de Flask dentro de IDLE, ya que IDLE no fue realmente diseñado para hacer este tipo de cosas bien. IDLE es ideal para experimentar pequeños fragmentos de código, pero cuando se trata de ejecutar aplicaciones, es mucho mejor que ejecute el código directamente a través del intérprete, en la línea de comandos de su sistema operativo. Hagamos eso ahora y veamos qué pasa.

**No utilice IDLE para ejecutar este código.**

## **Funcionamiento de su Webapp Flask por primera vez**

Si está ejecutando Windows, abra un símbolo del sistema en la carpeta que contiene su archivo de programa **hello\_flask.py**. (Sugerencia: si tiene su carpeta abierta en el Explorador de archivos, presione la tecla Mayúsculas junto con el botón derecho del ratón para abrir un menú contextual desde el que puede elegir Abrir ventana de comandos aquí). Con la línea de comandos de Windows lista, escriba este comando para iniciar su aplicación Flask:



Si está en Mac OS X o Linux, escriba el siguiente comando en una ventana de terminal. Asegúrese de emitir este comando en la misma carpeta que contiene su archivo de programa **hello\_flask.py**:

```
$ python3 hello_flask.py
```

No importa qué sistema operativo esté ejecutando, Flask se encargará de este punto, mostrando mensajes de estado en la pantalla siempre que su servidor web incorporado realice alguna operación. Inmediatamente después de la puesta en marcha, el servidor web de Flask confirma que está en funcionamiento y espera para atender las solicitudes web en la dirección web de prueba de Flask (127.0.0.1) y el número de puerto de protocolo (5000):

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Si ve este mensaje, todo está bien.

Lo compruebo:

```
[anton@anton-pc webapp]$ python hello_flask.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

```
[anton@anton-pc webapp]$ python hello_flask.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Significa que funciona.

El servidor web de Flask está listo y esperando. ¿Ahora que? Vamos a interactuar con el servidor web utilizando nuestro navegador web. Abra el navegador que sea su favorito y escriba la URL del mensaje de apertura del servidor web Flask:

**http://127.0.0.1:5000/** ←  
Esta es la dirección donde se ejecuta su aplicación web. Ingrese exactamente como se muestra aquí.



Después de un momento, el mensaje "Hello world from Flask!" De hello\_flask.py debería aparecer en la ventana de su navegador. Además de esto, eche un vistazo a la ventana de terminal donde su aplicación web está en ejecución ... un nuevo mensaje de estado debería haber aparecido también, de la siguiente manera:

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit) Ah ah Algo pasó.  
127.0.0.1 - - [23/Nov/2015 20:15:46] "GET / HTTP/1.1" 200 -
```

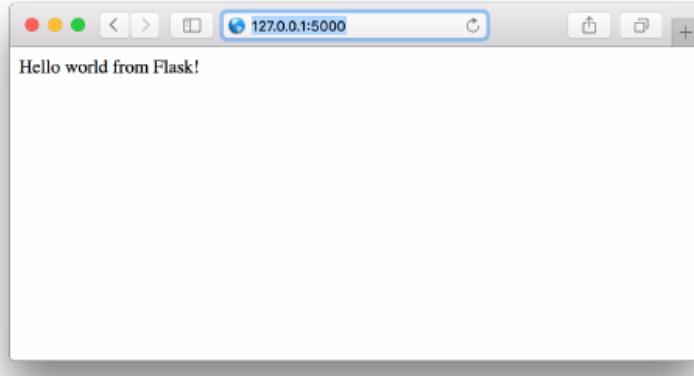
En mi maquina sale asi:

```
[anton@anton-pc webapp]$ python hello_flask.py  
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)  
127.0.0.1 - - [15/Apr/2017 01:46:47] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [15/Apr/2017 01:46:49] "GET /favicon.ico HTTP/1.1" 404 -
```

### ***Esto es lo que pasó (línea por línea)***

Además de Flask actualizando el terminal con una línea de estado, su navegador web muestra ahora la respuesta del servidor web. Así es como se ve nuestro navegador ahora (esto es Safari en Mac OS X):

There's the message returned from the Flask web server.



Al usar nuestro navegador para visitar la URL que aparece en el mensaje de estado de apertura de nuestra webapp, el servidor ha respondido con el mensaje "Hola mundo de Flask!".

Aunque nuestra aplicación web tiene sólo seis líneas de código, hay mucho que hacer aquí, así que vamos a revisar el código para ver cómo sucedió todo esto, tomando cada línea a su vez. Todo lo demás que planeamos hacer se basa en estas seis líneas de código.

La primera línea importa la clase Flask del módulo del flask:

This is the module's name: "flask" with a lowercase "f".

This is the class name: "Flask" with an uppercase "F".

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'
app.run()
```

```
#hello_flask.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello() -> str:

    return 'Hello world from Flask'

app.run()
```

¿Recuerdas cuando hablamos de formas alternativas de importar?

Usted podría haber escrito **import flask** aquí, después referido a la clase del **Flask** como `flask.Flask`, pero usar la versión de la declaración de la importación en esta ocasión se prefiere, pues el uso del **flask.Flask** no es tan fácil de leer.

## **Creación de un objeto Webapp Flask**

La segunda línea de código crea un objeto del tipo Flask, asignándolo a la variable `app`. Esto parece directo, pero para el uso del extraño argumento a Flask, a saber `_name_`:

Crear una instancia de un objeto Flask y asignarla a "app".

```
from flask import Flask  
app = Flask(_name_)  
  
@app.route('/')  
def hello() -> str:  
    return 'Hello world from Flask!'  
  
app.run()
```

¿Cuál es el trato aquí?

El valor `_name_` es mantenido por el intérprete de Python y, cuando se utiliza en cualquier parte dentro del código del programa, se establece en el nombre del módulo activo. Resulta que la clase Flask necesita conocer el valor actual de `_name_` al crear un nuevo objeto Flask, por lo que debe ser pasado como un argumento, por lo que lo usamos aquí (aunque su uso parezca extraño).

Esta sola línea de código, a pesar de ser corta, hace mucho para usted, ya que el marco Flask abstrae muchos detalles del desarrollo web, lo que le permite concentrarse en definir lo que quiere que suceda cuando una solicitud web llega a su servidor web en espera. Hacemos eso empezando en la siguiente línea de código.

Tenga en cuenta que `_name_` es dos caracteres de subrayado seguidos de la palabra "name" seguida de otros dos caracteres de subrayado, que se denominan "doble subrayado" cuando se usa para prefijar y sufijo un nombre en código Python. Verá esta convención de nombres mucho en sus viajes de Python, y en lugar de usar la frase de "frase doble subrayado, nombre, doble subrayado", los programadores inteligentes de Python dicen: "nombre de dunder", que es taquigrafía para el mismo cosa. Como hay un montón de usos de subrayado doble en Python, se les conoce colectivamente como "los adormecedores", y verás muchos ejemplos de otros errores y sus usos a lo largo del resto de este libro.

Además de los dunders, también existe una convención para usar un solo carácter de subrayado para prefijar ciertos nombres de variables. Algunos programadores de Python se refieren a los nombres prefijados de subrayado único por el nombre que induce el gemido "maravilla" (abbreviatura de "un subrayado").

### Cómo decorar una función con una URL

La siguiente línea de código introduce una nueva pieza de la sintaxis de Python: decoradores. Un decorador de funciones, que es lo que tenemos en este código, ajusta el comportamiento de una función existente sin tener que cambiar el código de esa función (es decir, la función que está siendo decorada).

Es posible que desee leer la última frase unas cuantas veces.

En esencia, los decoradores le permiten tomar algún código existente y aumentarlo con un comportamiento adicional según sea necesario. Aunque los decoradores también pueden aplicarse tanto a las clases como a las funciones, se aplican principalmente a las funciones, lo que hace que la mayoría de los programadores de Python se refieran a ellos como decoradores de funciones.

La sintaxis del decorador de Python toma la inspiración de la sintaxis de la anotación de Java, así como el mundo de la programación funcional.

Echemos un vistazo al decorador de funciones en el código de nuestra webapp, que es fácil de detectar, ya que comienza con el símbolo @:

Here's the function  
decorator, which—like all  
decorators—is prefixed  
with the @ symbol.

```
from flask import Flask  
  
app = Flask(__name__)  
  
@app.route('/')  
def hello() -> str:  
    return 'Hello world from Flask!'  
  
app.run()
```

This is the URL.

Aquí está el decorador de funciones, que -como todos los decoradores- está prefijado con el símbolo @.

Un decorador de funciones ajusta el comportamiento de una función existente (sin cambiar el código de la función).

Aunque es posible crear sus propios decoradores de funciones (que vienen en un capítulo posterior), por ahora vamos a concentrarnos en usarlos. Hay un montón de decoradores incorporados a Python, y muchos módulos de terceros (como

Flask) proporcionan decoradores para fines específicos (la ruta o **route** es uno de ellos).

El decorador de **route** o ruta de Flask está disponible para el código de tu webapp a través de la variable app, que se creó en la línea de código anterior.

El decorador de rutas o route le permite asociar una ruta web de URL con una función de Python existente. En este caso, la URL "/" está asociada con la función definida en la línea de código siguiente, que se llama hello. El decorador de route o rutas organiza que el servidor web de Flask llame a la función cuando una solicitud para la URL "/" llegue al servidor. El decorador de route espera entonces cualquier salida producida por la función decorada antes de devolver la salida al servidor, que luego la devuelve al navegador web en espera.

No es importante saber cómo Flask (y el decorador de ruta o route) hace todo lo anterior "magia". Lo importante es que Flask hace todo esto por ti, y todo lo que tienes que hacer es escribir una función que produce la salida que exigir. Flask y el decorador de ruta a continuación, cuidar de los detalles.

## ***Ejecución del (de los) comportamiento(s) de su aplicación web***

Con la línea del decorador de la route escrita, la función adornada por ella comienza en la línea siguiente. En nuestra webapp, esta es la función hello, que solo hace una cosa: devuelve el mensaje "Hello world from Flask!" Cuando se invoca:

Esta es sólo una función regular de Python que, cuando se invoca, devuelve una cadena a su llamador (observe la anotación '`-> str`').

```
from flask import Flask  
  
app = Flask(__name__)  
  
@app.route('/')  
def hello() -> str:  
    return 'Hello world from Flask!'  
  
app.run()
```

La última línea de código toma el objeto Flask asignado a la variable app y le pide a Flask que comience a ejecutar su servidor web. Hace esto invocando ejecutar:

```

from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello world from Flask!'

app.run()

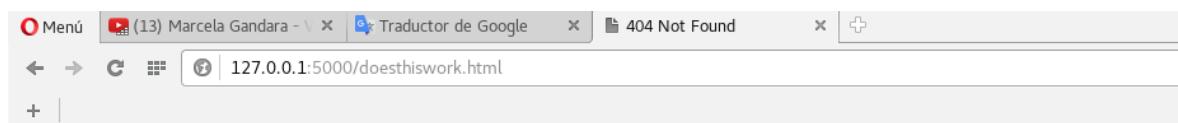
```

Pide que la aplicación web empiece a correr

En este punto, Flask inicia su servidor web incluido y ejecuta su código de webapp dentro de él. Todas las solicitudes recibidas por el servidor web para la URL "/" se responden con el mensaje "Hola mundo de Flask!", Mientras que una solicitud para cualquier otra URL da como resultado un mensaje de error 404 "Recurso no encontrado". Para ver el manejo de errores en acción, escriba esta URL en la barra de direcciones de su navegador:

<http://127.0.0.1:5000/doesthiswork.html>

Su navegador muestra un mensaje "No encontrado" y su aplicación web que se ejecuta dentro de su ventana de terminal actualiza su estado con un mensaje apropiado:



## Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

y en el terminal sale:

```
[anton@anton-pc webapp]$ python hello_flask.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)URL no existe: 404
127.0.0.1 - - [15/Apr/2017 11:02:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [15/Apr/2017 11:02:24] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [15/Apr/2017 11:03:39] "GET /doesthiswork.html HTTP/1.1" 404 -
```

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/Nov/2015 20:15:46] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Nov/2015 21:30:26] "GET /doesthiswork.html HTTP/1.1" 404 -
```

The messages you see may differ slightly.  
Don't let this worry you.

Chapter 5

Los mensajes que vea pueden  
No dejes que esto te preocupe.

iente.

## **Exponer funcionalidad a la Web**

Poniendo a un lado el hecho de que usted acaba de construir una webapp de trabajo en sólo seis líneas de código, considere lo que Flask está haciendo por usted aquí: es proporcionar un mecanismo mediante el cual puede tomar cualquier función existente de Python y mostrar su salida dentro de un navegador web.

Para agregar más funcionalidad a su aplicación web, todo lo que tiene que hacer es decidir la URL con la que desea asociar su funcionalidad y, a continuación, escriba una línea apropiada de decorador **@ app.route** encima de una función que realice el trabajo real. Hagamos esto ahora, usando nuestra funcionalidad **search4letters** del último capítulo.

Modificamos **hello\_flask.py** para incluir una segunda URL: / **search4**. Escriba el código que asocia esta URL con una función llamada **do\_search**, que llama a la función **search4letters** (desde nuestro módulo **vsearch**). Luego ordene que la función **do\_search** devuelva los resultados determinados al buscar la frase: "vida, el universo y todo!" Para esta serie de caracteres: 'eiru ,!'.

A continuación se muestra nuestro código existente, con espacio reservado para el nuevo código que necesita escribir. Su trabajo es proporcionar el código que falta.

Sugerencia: los resultados devueltos de **search4letters** son un conjunto de Python. Asegúrese de emitir los resultados a una cadena llamando al str BIF antes de devolver algo al explorador web en espera, ya que está esperando datos de texto, no un conjunto de Python. (Recuerde: "BIF" es Python para hablar de la función incorporada.)

```
from flask import Flask  
.....  
app = Flask(__name__)  
.....  
@app.route('/')  
def hello() -> str:  
    return 'Hello world from Flask!'  
.....  
app.run()  
.....
```

→ ¿Necesita importar algo?

→ Añadir en un segundo decorador.  
Add code for the "do\_search" function here.

## Solucion:

Debes modificar **hello\_flask.py** para incluir una segunda URL, / search4, escribiendo el código que asocia la URL con una función llamada do\_search, la cual llama a la función **search4letters** (desde nuestro módulo vsearch). Debes arreglar para que la función do\_search devuelva los resultados determinados al buscar la frase: "¡vida, el universo y todo!" Para la cadena de caracteres: 'eiru ,!'.

A continuación se muestra nuestro código existente, con espacio reservado para el nuevo código que necesita escribir. Su trabajo consistía en proporcionar el código que faltaba. ¿Cómo se compara tu código con el nuestro?

```
Debe importar           from flask import Flask
la función             →
"search4letters"       from vsearch import search4letters
desde el
módulo
"vsearch" antes
de llamarla.           app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

Un segundo
decorador
configura la
URL "/"
search4".
→ @app.route('/search4')
def do_search() -> str:
    return str(search4letters('life, the universe, and everything', 'eiru,!')) ←

app.run()
```

The "do\_search" function invokes "search4letters", then returns any results as a string.

La función "do\_search" invoca "search4letters", y luego devuelve cualquier resultado como una cadena.

Para probar esta nueva funcionalidad, necesitará reiniciar su aplicación Web Flask, ya que actualmente está ejecutando la versión anterior del código. Para detener la aplicación web, regrese a la ventana del terminal y, a continuación, presione Ctrl y C juntos. Tu webapp finalizará y volverás al sistema operativo. Pulse la flecha hacia arriba para recuperar el comando anterior (el que ha iniciado hello\_flask.py) y, a continuación, pulse la tecla Intro.

El mensaje de estado inicial del Flask vuelve a aparecer para confirmar que su aplicación actualizada está esperando solicitudes:

```
$ python3 hello_flask.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/Nov/2015 20:15:46] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Nov/2015 21:30:26] "GET /doesthiswork.html HTTP/1.1" 404 -
^C
...then
...restart it.
$ python3 hello_flask.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit) ← We are up and
running again.
```

Probando:

```
#hello_flask.py

from flask import Flask

from vsearch import search4letters

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask'

@app.route('/search4')
def do_search()->str:
    return str(search4letters('life, the universe, and everything','eiru!'))

app.run()
```

```
[anton@anton-pc webapp]$ ls
hello_flask.py __pycache__ vsearch.py
[anton@anton-pc webapp]$ python hello_flask.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Volvemos a ejecutar el servidor como esta en el grafico de mi prompt.

Ahora compruebo si corre la pagina.

A screenshot of a web browser window. The address bar shows the URL "127.0.0.1:5000/search4". The main content area displays the output of the search4 function, which is a set containing the letters 'I', 'r', 'u', and 'e'.

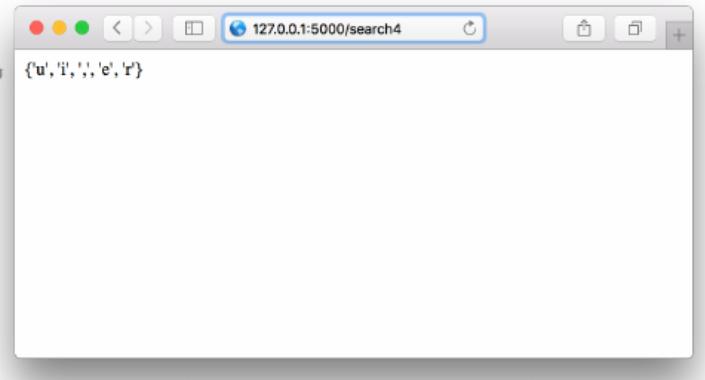
```
{'I', 'r', 'u', 'e'}
```

Y si corre.

Como no ha cambiado el código asociado con la URL predeterminada, la funcionalidad sigue funcionando, mostrando el mensaje "Hola mundo de Flask!".

Sin embargo, si introduce `http://127.0.0.1:5000/search4` en la barra de direcciones de su navegador, verá los resultados de la llamada a `search4letters`:

Hay los resultados de la llamada a "search4letters". Por supuesto, esta salida no es nada para emocionarse, pero sí demuestra que el uso de la URL "/search4" invoca la función y devuelve los resultados.



### Preguntas :

**P: Estoy un poco confundido por el 127.0.0.1 y: 5000 partes de la URL utilizada para acceder a la aplicación web. ¿Cuál es el problema?**

R: Por el momento, está probando su aplicación web en su computadora, ya que está conectada a Internet, tiene su propia dirección IP única. A pesar de este hecho, Flask no utiliza su dirección IP y en su lugar conecta su servidor web de prueba a la dirección de bucle de retorno de Internet: 127.0.0.1, también conocido como localhost. Ambos son la abreviatura de "mi computadora, no importa cuál sea su dirección IP real". Para que su navegador web (también en su computadora) se comunique con su servidor web Flask,

debe especificar la dirección que está ejecutando su aplicación web, a saber: 127.0.0.1. Esta es una dirección IP estándar reservada para este propósito exacto.

La parte: 5000 de la dirección URL identifica el número de puerto de protocolo en el que se ejecuta su servidor web.

Normalmente, los servidores web se ejecutan en el puerto de protocolo 80, que es un estándar de Internet, y como tal, no necesita ser especificado. Puede escribir oreilly.com:80 en la barra de direcciones de su navegador y funcionaría, pero nadie lo hace, ya que oreilly.com solo es suficiente (como se supone: 80).

Cuando se está creando una aplicación web, es muy raro probar en el puerto de protocolo 80 (como se reserva para los servidores de producción), por lo que la mayoría de los frameworks web eligen otro puerto para ejecutarlo. 8080 es una opción popular para esto, pero el Flask utiliza 5000 como su puerto del protocolo de la prueba.

**P: ¿Puedo usar algún puerto de protocolo distinto de 5000 cuando pruebo y ejecuto mi aplicación web Flask?**

R: Sí, `app.run()` le permite especificar un valor para el puerto que se puede establecer en cualquier valor. Pero, a menos que tenga una muy buena razón para cambiar, mantenga el valor predeterminado de 5000 de Flask por ahora.

### ***Recuerde lo que estamos tratando de construir***

Nuestra webapp necesita una página web que acepte entrada, y otra que muestra los resultados de alimentar la entrada a la función `search4letters`. Nuestro código webapp actual está lejos de hacer todo esto, pero lo que tenemos proporciona una base sobre la cual construir lo que se requiere.

A continuación se muestra una copia de nuestro código actual, mientras que a la derecha tenemos copias de las "especificaciones de la servilleta" de este capítulo. Hemos indicado dónde creemos que la funcionalidad de cada servilleta puede proporcionarse en el código:

```

from flask import Flask
from vsearch import search4letters

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4')
def do_search() -> str:
    return str(search4letters( ... ))

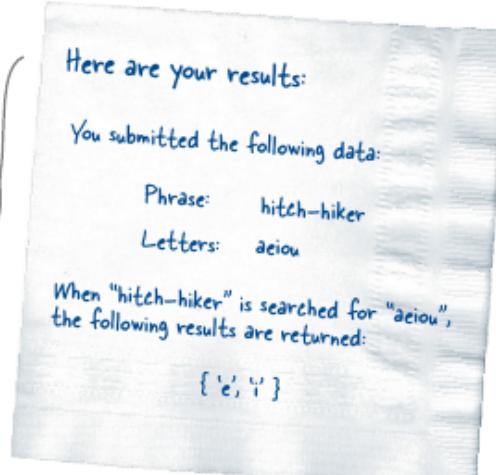
app.run()

```

Note: to make everything fit, we aren't showing the entire line of code here.

## Here's the plan

Let's change the `hello` function to return the HTML form. Then we'll change the `do_search` function to accept the form's input, before calling the `search4letters` function. The results are then returned by `do_search` as another web page.



Nota: para que todo encaje, no mostramos toda la línea de código aquí.

## Aquí está el plan

Cambiemos la función `hello` para devolver el formulario HTML. A continuación, cambiaremos la función `do_search` para aceptar la entrada del formulario, antes de llamar a la función `search4letters`. Los resultados son devueltos por `do_search` como otra página web.

#hello\_flask2.py

```

from flask import Flask
from vsearch import search4letters

app = Flask(__name__)

```

```

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4')
def do_search() -> str:
    return str(search4letters('life, the universe, and everything', 'eiru,!'))

app.run()

```

## ***Creación del formulario HTML***

El formulario HTML necesario no es tan complicado. Aparte del texto descriptivo, el formulario se compone de dos cuadros de entrada y un botón.

### **Pero ... ¿qué pasa si eres nuevo en todas estas cosas HTML?**

No entre en pánico si toda esta charla de formularios HTML, cuadros de entrada y botones que tiene en un tizzy. No tengas miedo, tenemos lo que buscas: la segunda edición de Head First HTML y CSS proporciona la mejor introducción a estas tecnologías si necesitas una cartilla rápida (o una actualización rápida).

Incluso si el pensamiento de dejar a un lado este libro con el fin de hueso en HTML se siente como demasiado trabajo, tenga en cuenta que proporcionamos todo el HTML que necesita para trabajar con los ejemplos en el libro, y lo hacemos sin tener que ser un Experto en HTML. Un poco de exposición a HTML ayuda, pero no es un requisito absoluto (después de todo, este es un libro sobre Python, no HTML).

### **Cree el HTML y envíelo al navegador**

Siempre hay más de una manera de hacer las cosas, y cuando se trata de crear texto HTML desde su webapp Flask, tiene opciones:

Me gusta poner mi HTML dentro de grandes cadenas, que luego incrustar en mi código de Python, devolver las cadenas según sea necesario. De esa manera, todo lo que necesito está ahí en mi código, y tengo el control completo ... que es cómo ruedo. ¿Qué no te gusta, Laura?

Bueno, Bob, poner todo el código HTML en tu código funciona, pero no escala. A medida que su webapp se vuelve más grande, todo lo que HTML incrustado se ensucia un poco ... y

es difícil entregar su HTML a un diseñador web para embellecer. Tampoco es fácil reutilizar trozos de HTML. Por lo tanto, siempre uso un motor de plantilla con mis webapps. Es un trabajo un poco más para empezar, pero con el tiempo me encuentro utilizando plantillas realmente vale la pena ...

Las plantillas correctas de Laura hacen que el HTML sea mucho más fácil de mantener que el enfoque de Bob. Vamos a sumergir en las plantillas en la siguiente página.

## Plantillas de cerca

Los motores de plantilla permiten a los programadores aplicar las nociones orientadas a objetos de herencia y reutilización a la producción de datos textuales, como páginas web.

La apariencia de un sitio web puede definirse en una plantilla HTML de nivel superior o top-level, conocida como **plantilla base**, que es heredada por otras páginas HTML. Si realiza un cambio en la plantilla base, el cambio se refleja en todas las páginas HTML que heredan de ella.

El motor de plantilla enviado con Flask se llama **Jinja2**, y es fácil de usar y potente. No es la intención de este libro enseñarte todo lo que necesitas saber sobre Jinja2, así que lo que aparece en estas dos páginas es, por necesidad, breve y directo. Para más detalles sobre lo que es posible con Jinja2, vea:

<http://jinja.pocoo.org/docs/dev/>

Esta es la plantilla base que usaremos para nuestra aplicación web. En este archivo, llamado base.html, ponemos el marcado HTML que queremos que todas nuestras páginas web compartan. También utilizamos un marcado específico de Jinja2 para indicar el contenido que se proporcionará cuando se rendericen las páginas HTML que heredan de ésta (es decir, preparadas antes de la entrega a un navegador web en espera). Tenga en cuenta que el marcado que aparece entre `{}{y}`, así como el marcado incluido entre `{% %}`, está destinado al motor de plantillas Jinja2: hemos resaltado estos casos para facilitar su detección:

Este es el marcado HTML5 estándar.



```
<!doctype html>
<html>
  <head>
    <title>{{ the_title }}</title>
    <link rel="stylesheet" href="static/hf.css" />
  </head>
  <body>
    {% block body %}
    {% endblock %}
  </body>
</html>
```

This is a Jinja2 directive, which indicates that a value will be provided prior to rendering (think of this as an argument to the template).

This stylesheet defines the look and feel of all the web pages.

These Jinja2 directives indicate that a block of HTML will be substituted here prior to rendering, and is to be provided by any page that inherits from this one.

Esta es la plantilla base.



### Traduciendo lo que esta en la plantilla desde arriba hacia abajo:

Se trata de una directiva Jinja2, que indica que se proporcionará un valor antes de la representación (piense en esto como un argumento para la plantilla).

Esta hoja de estilos define la apariencia de todas las páginas web.

Estas directivas Jinja2 indican que un bloque de HTML se sustituirá aquí a la prestación, y debe ser proporcionado por cualquier página que hereda de ésta.

```
<!--Comentario en HTML -->
<!--web base.html -->

<!doctype html>
<html>
  <head>
    <title>{{ the_title }}</title>
    <link rel="stylesheet" href="static/hf.css" />
  </head>
  <body>
    {% block body %}
    {% endblock %}
  </body>
</html>
```

## Volviendo:

Con la plantilla de base lista, podemos heredar de ella con la directiva de extensiones de Jinja2. Cuando lo hacemos, los archivos HTML que heredan solo necesitan proporcionar el HTML para cualquier bloque con nombre en la base. En nuestro caso, sólo tenemos un bloque con nombre: cuerpo o body.

Aquí está el marcado para la primera de nuestras páginas, que llamamos **entry.html**. Se trata de un marcado para un formulario HTML con el que los usuarios pueden interactuar con el fin de proporcionar el valor de la frase y las letras esperadas por nuestra aplicación web.

Observe cómo no se repite el HTML "calificativo" o "boilerplate" en la plantilla base en este archivo, ya que la directiva extends incluye este marcado para nosotros. Todo lo que necesitamos hacer es proporcionar el HTML que es específico para este archivo, y lo hacemos al proporcionar el marcado dentro del bloque Jinja2 llamado body:

```
{% extends 'base.html' %} → This template inherits  
→ from the base, and  
→ provides a replacement for  
→ the block called "body".  
→  
{% block body %}  
  
<h2>{{ the_title }}</h2>  
  
<form method='POST' action='/search4'>  
<table>  
<p>Use this form to submit a search request:</p>  
<tr><td>Phrase:</td><td><input name='phrase' type='TEXT' width='60'></td></tr>  
<tr><td>Letters:</td><td><input name='letters' type='TEXT' value='aeiou'></td></tr>  
</table>  
<p>When you're ready, click this button:</p>  
<p><input value='Do it!' type='SUBMIT'></p>  
</form>  
  
{% endblock %}
```

Traducción: Esta plantilla hereda de la base, y proporciona un reemplazo para el bloque llamado "cuerpo".

<!--Comentario en HTML -->  
<!--web entry.html que hereda de base.html-->

```
{% extends 'base.html' %}
```

```
{% block body %}
```

```

<h2>{{ the_title }}</h2>

<form method='POST' action='/search4'>
<table>
<p>Use this form to submit a search request:</p>
<tr><td>Phrase:</td><td><input name='phrase' type='TEXT' width='60'></td></tr>
<tr><td>Letters:</td><td><input name='letters' type='TEXT' value='aeiou'></td></tr>
</table>
<p>When you're ready, click this button:</p>
<p><input value='Do it!' type='SUBMIT'></p>
</form>

{% endblock %}

```

Y, finalmente, aquí está el marcado para el archivo **results.html**, que se utiliza para procesar los resultados de nuestra búsqueda. Esta plantilla también hereda de la plantilla base:

```

{% extends 'base.html' %}           As with "entry.html", this
{%
  block body %}                   template also inherits
<h2>{{ the_title }}</h2>         from the base, and also
<p>You submitted the following data:</p> provides a replacement for
<table>                           the block called "body".
<tr><td>Phrase:</td><td>{{ the_phrase }}</td></tr>
<tr><td>Letters:</td><td>{{ the_letters }}</td></tr>
</table>
<p>When "{{the_phrase}}" is search for "{{ the_letters }}", the following
results are returned:</p>
<h3>{{ the_results }}</h3>
{%
  endblock %
}

```

The diagram shows the template code with several annotations:

- An annotation points to the line `As with "entry.html", this template also inherits from the base, and also provides a replacement for the block called "body".` with arrows pointing to the opening brace of the block and the closing brace.
- Another annotation points to the line `When "{{the\_phrase}}" is search for "{{ the\_letters }}", the following results are returned:` with arrows pointing to both instances of `{{the\_phrase}}` and `{{ the\_letters }}`.
- A third annotation points to the line `Note these additional argument values, which you need to provide values for prior to rendering.` with an arrow pointing to the line `{{ the\_results }}`.

traducción: Al igual que con "entry.html", esta plantilla también hereda de la base, y también proporciona un reemplazo para el bloque llamado "cuerpo".

Traducción: Tenga en cuenta estos valores de argumento adicionales, que debe proporcionar valores antes de la representación o rendering.

```

<!--Comentario en HTML -->
<!--web result.html que hereda de base.html-->

{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>

<p>You submitted the following data:</p>

<table>

<tr><td>Phrase:</td><td>{{ the_phrase }}</td></tr>

<tr><td>Letters:</td><td>{{ the_letters }}</td></tr>

</table>

<p>When "{{the_phrase}}" is searched for "{{ the_letters }}", the following
results are returned:</p>

<h3>{{ the_results }}</h3>


```

{% endblock %}

## Es solo html

### **Plantillas relacionadas con la página Web**

Nuestra webapp necesita hacer dos páginas web, y ahora tenemos dos plantillas que pueden ayudar con esto. Ambas plantillas heredan de la plantilla base y por lo tanto heredan la apariencia de la plantilla base. Ahora todo lo que necesitamos hacer es renderizar las páginas o reproducir las paginas.

Antes de ver cómo reproduce o renders Flask (junto con Jinja2), vamos a echar un vistazo a nuestras "especificaciones servilleta" junto con nuestro marcado de plantilla. Tenga en cuenta cómo el código HTML incluido dentro de la directiva Jinja2 **{% block %}** coincide con las especificaciones dibujadas a mano. La omisión principal es el título de cada página, que proporcionamos en lugar de la directiva **{{the\_title}}** durante la renderización. Piense en cada nombre encerrado en llaves dobles como un argumento a la plantilla:

Welcome to search4letters on the Web!

Use this form to submit a search request:

Phrase:

Letters: aeiou

When you're ready, click this button:

**Do it!**

```

    {% extends 'base.html' %}

    {% block body %}

        <h2>{{ the_title }}</h2>

        <form method='POST' action='/search4'>
            <table>
                <p>Use this form to submit a search request:</p>
                <tr><td>Phrase:</td><td><input name='phrase' type='TEXT' width='60'></td></tr>
                <tr><td>Letters:</td><td><input name='letters' type='TEXT' value='aeiou'></td></tr>
            </table>
            <p>When you're ready, click this button:</p>
            <p><input value='Do it!' type='SUBMIT'></p>
        </form>

    {% endblock %}

```

Here are your results:

You submitted the following data:

Phrase: hitch-hiker

Letters: aeiou

When "hitch-hiker" is searched for "aeiou", the following results are returned:

{'e', 'i'}

```

    {% extends 'base.html' %}

    {% block body %}

        <h2>{{ the_title }}</h2>

        <p>You submitted the following data:</p>
        <table>
            <tr><td>Phrase:</td><td>{{ the_phrase }}</td></tr>
            <tr><td>Letters:</td><td>{{ the_letters }}</td></tr>
        </table>
        <p>When "{{the_phrase}}" is search for "{{ the_letters }}", the following results are returned:</p>
        <h3>{{ the_results }}</h3>

    {% endblock %}

```

Don't forget those additional arguments.

## Rendering Templates from Flask

### Plantillas de render de Flask

Flask viene con una función llamada **render\_template**, que, cuando se proporciona con el nombre de una plantilla y los argumentos necesarios, devuelve una cadena de HTML cuando se invoca. Para utilizar **render\_template**, añada su nombre a la lista de importaciones del módulo de flask(en la parte superior de su código), luego invoque la función según sea necesario.

Sin embargo, antes de hacerlo, vamos a renombrar el archivo que contiene el código de nuestra aplicación web (actualmente llamado **hello\_flask.py**) a algo más apropiado. Puede

usar cualquier nombre que desee para su aplicación web, pero estamos renombrando nuestro archivo **vsearch4web.py**. Aquí está el código actualmente en este archivo:

```
from flask import Flask
from vsearch import search4letters

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4')
def do_search() -> str:
    return str(search4letters('life, the universe, and everything', 'eiru,!'))

app.run()
```

This code now resides  
in a file called  
"vsearch4web.py".

traducción: Este código ahora reside en un archivo llamado "vsearch4web.py".

```
#hello_flask2.py
from flask import Flask

from vsearch import search4letters

app = Flask(__name__)

@app.route('/')

def hello() -> str:

    return 'Hello world from Flask!'

@app.route('/search4')

def do_search() -> str:

    return str(search4letters('life, the universe, and everything', 'eiru,!'))

app.run()
```

Para procesar el formulario HTML en la plantilla **entry.html**, necesitamos realizar una serie de cambios en el código anterior:

- 1. Importar la función render\_template**

Añada **render\_template** a la lista de importación de la línea **from flask** en la parte superior del código.

- 2. Crear una nueva URL -en este caso, / entry**

Cada vez que necesite una nueva URL en su webapp de Flask, también debe agregar una nueva línea **@ app.route**. Lo haremos antes de la línea de código **app.run ()**.

- 3. Crear una función que devuelve el HTML correctamente procesado**

Con la línea `@ app.route` escrita, puede asociar código con ella creando una función que realice el trabajo real (y haga que su webapp sea más útil para sus usuarios). La función llama (y devuelve la salida de) la función `render_template`, pasando el nombre del archivo de plantilla (`entry.html` en este caso), así como cualquier valor de argumento que sea requerido por la plantilla (en el caso, necesitamos un valor para `the_title`).

*Hagamos estos cambios en nuestro código existente.*

## ***Displaying the Webapp's HTML Form***

### ***Visualización del formulario HTML del Webapp***

Vamos a agregar el código para habilitar los tres cambios detallados en la parte inferior de la última página. Siga adelante haciendo los mismos cambios en su código:

#### **1. Importar la función `render_template`**

```
from flask import Flask, render_template
```

Agregue "render\_template" a la lista de tecnologías importadas del módulo "flask".

#### **2. Crear una nueva URL: en este caso, / entrada**

```
@app.route('/entry')
```

Debajo de la función "do\_search", pero antes de la línea "app.run()", inserte esta línea para agregar una nueva URL a la aplicación web.

#### **3. Crear una función que devuelve el HTML correctamente procesado**

```
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')
```

Add this function directly underneath the new "@app.route" line.

Provide the name of the template to render.

Provide a value to associate with the "the\_title" argument.

traducciones: de arriba-abajo:

- Proporcione el nombre de la plantilla para procesar
- Agregue esta función directamente debajo de la nueva línea "@ app.route".
- Proporcione un valor para asociar con el argumento "the\_title".

Con estos cambios realizados, el código de nuestra aplicación web, con las adiciones resaltadas, ahora se ve así:

```
from flask import Flask, render_template
from vsearch import search4letters

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4')
def do_search() -> str:
    return str(search4letters('life, the universe, and everything', 'eiru,!'))

@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

app.run()
```

*We're leaving the rest of this code as is for now.*

traducción: Estamos dejando el resto de este código como está por ahora.

```
# vsearch4web.py

from flask import Flask, render_template
from vsearch import search4letters

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4')
def do_search() -> str:
    return str(search4letters('life the universe, and everthing','eiru,!'))

@app.route('/entry')
```

```
def entry_page() ->'html':  
    return render_template('entry.html',  
        the_title='Welcome to search4letters on the web!')  
app.run()
```

## Preparación para ejecutar el código de plantilla

Es tentador abrir un símbolo del sistema y luego ejecutar la última versión de nuestro código. Sin embargo, por varias razones, esto no funcionará de inmediato.

Para empezar, la plantilla de base se refiere a una hoja de estilo llamada **hf.css**, y esto debe existir en una carpeta llamada **estática** (que es relativa a la carpeta que contiene el código). Aquí hay un fragmento de la plantilla base que muestra esto:

Si todavía no lo ha hecho, descargue las plantillas y el CSS desde aquí: <http://python.itcarlow.ie/ed2/>.

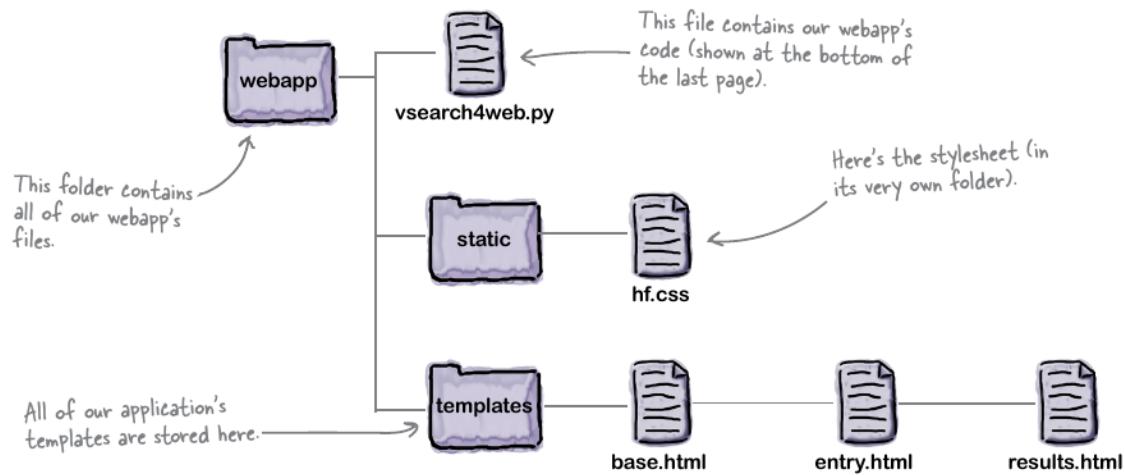
```
...  
<title>{{ the_title }}</title>  
<link rel="stylesheet" href="static/hf.css" />  
</head>  
...
```

El archivo "hf.css" necesita existir (en la carpeta "estática").

Siéntase libre de tomar una copia del archivo CSS del sitio web de soporte de este libro (consulte la URL al lado de esta página). Sólo asegúrese de colocar la hoja de estilos descargada en una carpeta llamada **estática**.

Además de esto, Flask requiere que sus plantillas se almacenen en una carpeta llamada **templates**, que como **static** necesita ser relativa a la carpeta que contiene su código. La descarga de este capítulo también contiene las tres plantillas ... para que pueda evitar escribir todo ese HTML!

Suponiendo que ha puesto el archivo de código de su webapp en una carpeta denominada **webapp**, aquí está la estructura que debería tener en su lugar antes de intentar ejecutar la versión más reciente de **vsearch4web.py**:



traduciendo de arriba-abajo, de izq-derch:

- Esta carpeta contiene todos los archivos de nuestra aplicación web.
- Todas las plantillas de nuestra aplicación se almacenan aquí.

- Este archivo contiene el código de nuestra webapp (mostrado en la parte inferior de la última página).
- Aquí está la hoja de estilo (en su propia carpeta).

## ***Estamos listos para una prueba***

Si tienes todo listo: la hoja de estilo y las plantillas descargadas, y el código actualizado, ya estás listo para tomar tu webapp de Flask para otra vuelta.

Es probable que la versión anterior de su código siga ejecutándose en el símbolo del sistema.

Vuelva a esa ventana ahora y presione Ctrl y C juntos para detener la ejecución de la aplicación web anterior. A continuación, presione la tecla de flecha hacia arriba para recuperar la última línea de comando, editar el nombre del archivo para ejecutar y, a continuación, presione Entrar. Su nueva versión de su código debería ejecutarse, mostrando los mensajes de estado habituales:

```

...
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/Nov/2015 21:51:38] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Nov/2015 21:51:48] "GET /search4 HTTP/1.1" 200 -
^C
$ python3 vsearch4web.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Stop the webapp again... →
Start up your new code (which is in the "vsearch4web.py" file). ←
The new code is up and running, and waiting to service requests. ←

```

Recuerde que esta nueva versión de nuestro código aún admite las URLs de / y / search4, por lo que si utiliza un navegador para solicitarlas, las respuestas serán las mismas que se muestran anteriormente en este capítulo. Sin embargo, si utiliza esta URL:

<http://127.0.0.1:5000/entry>

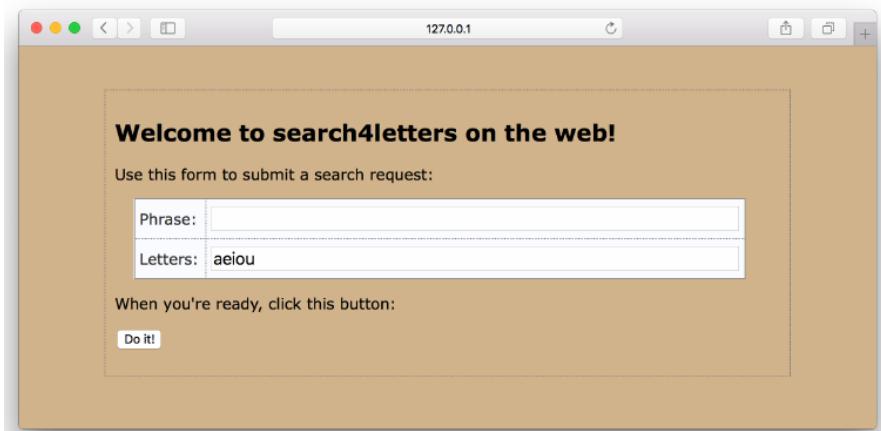
La respuesta que se muestra en el navegador debe ser el formulario HTML representado (mostrado en la parte superior de la página siguiente). La línea de comandos debe mostrar dos líneas de estado adicionales: una para la petición / entry y otra relacionada con la solicitud de su navegador para la hoja de estilo **hf.css**:

```

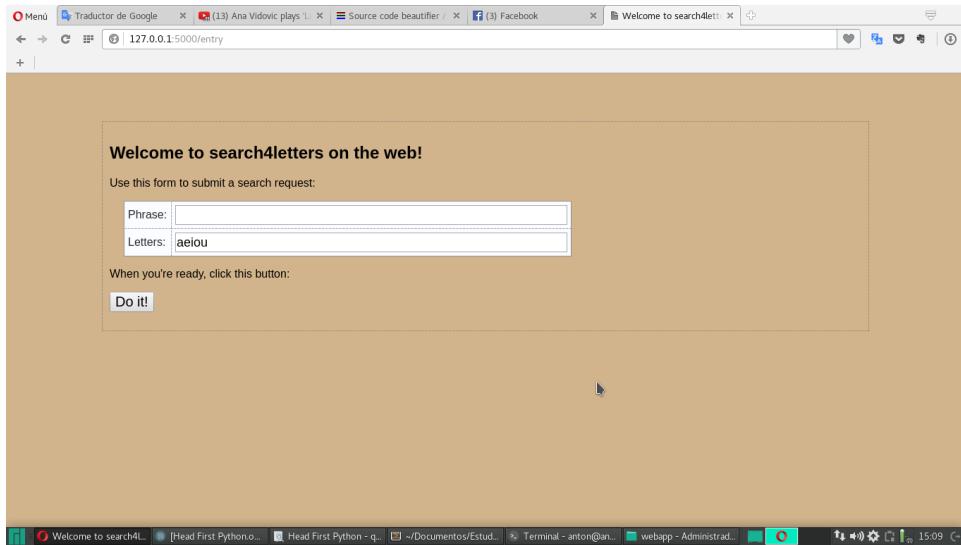
Usted solicita el formulario HTML ....
...
127.0.0.1 - - [23/Nov/2015 21:55:59] "GET /entry HTTP/1.1" 200 -
127.0.0.1 - - [23/Nov/2015 21:55:59] "GET /static/hf.css HTTP/1.1" 304 -
... y su navegador también solicita la hoja de estilo.

```

Esto es lo que aparece en la pantalla cuando escribimos **http://127.0.0.1:5000/entry** en nuestro navegador:

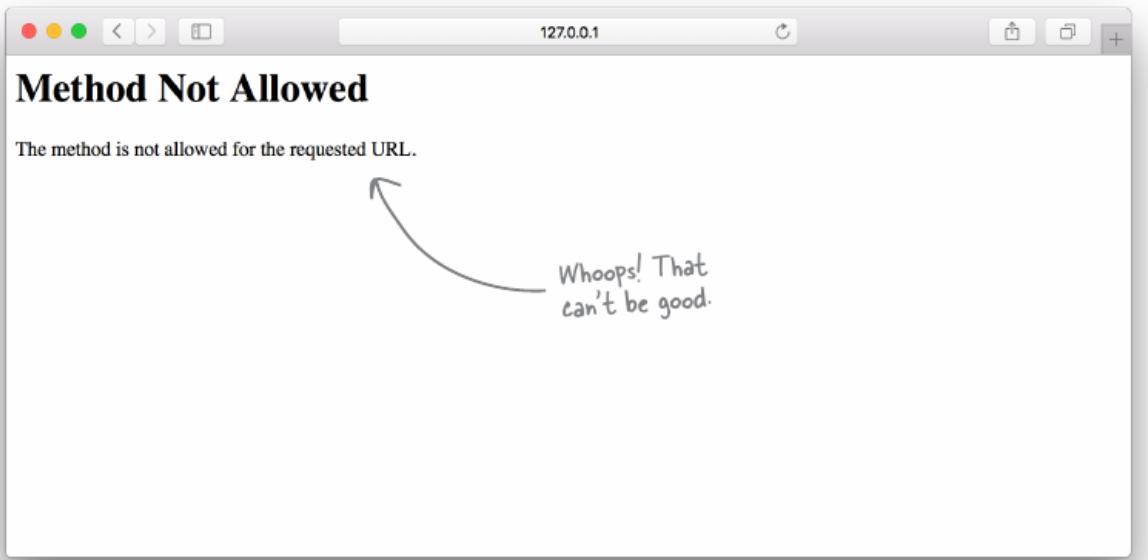


en mi maquina es asi:



No vamos a ganar ningún premio de diseño web para esta página, pero se ve bien, y se parece a lo que teníamos en la parte de atrás de nuestra servilleta. Desafortunadamente, cuando escribe una frase y (opcionalmente) ajusta el valor de Letters para hacerle clic, haga clic en **Do it!** Produce esta página de error:

por ejemplo ingresamos 'antonio' y damos clic en **Do it!**



Esto es un poco desagradable, ¿no? Vamos a ver qué está pasando.

## ¿Qué salió mal?

### Descripción de los códigos de estado HTTP

Cuando algo sale mal con tu webapp, el servidor web responde con un código de estado HTTP (que envía a tu navegador). HTTP es el protocolo de comunicaciones que permite que los navegadores web y los servidores se comuniquen. El significado de los códigos de estado está bien establecido (ver los bits de Geek a la derecha). De hecho, cada solicitud web genera una respuesta de código de estado HTTP.

#### Geek Bits

Esta es una explicación rápida y sucia de los distintos códigos de estado HTTP que se pueden enviar desde un servidor web (por ejemplo, su webapp de Flask) a un cliente web (por ejemplo, su navegador web).

Hay cinco categorías principales de código de estado: 100s, 200s, 300s, 400s y 500s.

Los códigos en el rango de **100 a 199** son **mensajes informativos**: todo está bien y el servidor proporciona detalles relacionados con la solicitud del cliente.

Los códigos en la gama **200-299** son **mensajes de éxito**: el servidor ha recibido, entendido y procesado la solicitud del cliente. Todo es bueno.

Los códigos en el rango **300-399** son **mensajes de redirección**: el servidor está informando al cliente que la solicitud se puede manejar en otro lugar.

Los códigos en el intervalo **400-499** son **mensajes de error del cliente**: el servidor recibió una solicitud del cliente que no entiende y no puede procesar. Normalmente, el cliente tiene la culpa aquí.

Los códigos en el rango **500-599** son **mensajes de error del servidor**: el servidor recibió una solicitud del cliente, pero el servidor falló al intentar procesarla. Normalmente, el servidor está en falta aquí.

Para obtener más detalles, consulte:

[https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes).

Para ver qué código de estado se envió a su navegador desde su aplicación web, revise los mensajes de estado que aparecen en el símbolo del sistema. Esto es lo que vimos:

```
...
127.0.0.1 - - [23/Nov/2015 21:55:59] "GET /entry HTTP/1.1" 200 -
127.0.0.1 - - [23/Nov/2015 21:55:59] "GET /static/hf.css HTTP/1.1" 304 -
127.0.0.1 - - [23/Nov/2015 21:56:54] "POST /search4 HTTP/1.1" 405 -
```

Uh-oh. Something has gone wrong,  
and the server has generated a  
client-error status code.

```
^C[anton@anton-pc webapp]$ python vsearch4web.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [16/Apr/2017 15:04:23] "GET /entry HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2017 15:04:29] "GET /entry HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2017 15:04:41] "GET /entry HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2017 15:11:27] "POST /search4 HTTP/1.1" 405 -
```

El código de estado 405 indica que el cliente (su navegador) envió una solicitud utilizando un método **HTTP** que este servidor no permite. Hay un puñado de **métodos HTTP**, pero para nuestros propósitos, sólo necesita tener en cuenta dos de ellos: **GET** y **POST**.

### ***El método GET***

Los navegadores suelen utilizar este método para solicitar un recurso desde el servidor web, y este método es con mucho el más utilizado. (Nosotros decimos "típicamente" aquí, ya que es posible usar GET para enviar datos desde su navegador al servidor, pero no nos estamos enfocando en esa opción aquí). Todas las URL de nuestra webapp actualmente soportan GET , Que es el método HTTP predeterminado de Flask.

### ***El método POST***

Este método permite que un navegador web envíe datos al servidor a través de HTTP, y está estrechamente asociado con la etiqueta HTML **<form>**. Puede indicarle a su aplicación web Flask que acepte los datos publicados desde un navegador proporcionando un argumento adicional en la línea **@ app.route**.

Vamos a ajustar la línea **@ app.route** emparejada con la URL de nuestra webapp / **search4** para aceptar los datos publicados. Para ello, vuelva a su editor y edite el archivo **vsearch4web.py** una vez más.

### ***Manejo de datos publicados***

Además de aceptar la URL como su primer argumento, el decorador de **@app.route** acepta otros argumentosopcionales.

Uno de ellos es el argumento **methods**, que enumera los métodos HTTP que admite la URL. De forma predeterminada, Flask admite GET para todas las URL. Sin embargo, si al argumento **methods** se le asigna una lista de métodos HTTP para dar soporte, este comportamiento predeterminado se reemplaza. Esto es lo que la línea **@ app.route** parece actualmente:

```
@app.route('/search4')
```

No hemos especificado  
un método HTTP para  
dar soporte aquí, por lo  
que Flask se convierte  
por defecto en GET.

Para tener el POST de búsqueda de la URL `/search4`, agregue el argumento de métodos al decorador y asigne la lista de métodos HTTP que desea que la URL soporte. Esta línea de código, a continuación, indica que la URL `/search4` ahora sólo admite el método **POST** (significa que las solicitudes **GET** ya no son compatibles):

```
@app.route('/search4', methods=['POST'])
```

La URL "/ search4"  
ahora solo admite  
el método POST.

Este pequeño cambio es suficiente para liberar su webapp del mensaje "**Método no permitido**", ya que el POST asociado con el formulario HTML coincide con el POST en la línea `@ app.route`:

Este  
fragmento  
HTML es  
de  
"entry.html"  
...

```
...  
<form method='POST' action='/search4'>  
<table>  
...
```

```
...  
@app.route('/search4', methods=['POST'])  
def do_search() -> str:  
...
```

... y este código de  
Python es del archivo  
"vsearch4web.py".

Tenga en cuenta cómo HTML utiliza "método" (singular),  
mientras que Flask utiliza "métodos" (plural).

## No hay preguntas tontas

P: ¿Qué pasa si necesito que mi URL soporte tanto el método GET como POST? ¿Es eso posible?

**R:** Sí, todo lo que necesitas hacer es agregar el nombre del método HTTP que necesitas para dar soporte a la lista asignada a los argumentos de los métodos. Por ejemplo, si desea añadir soporte GET a la URL / search4, sólo necesita cambiar la línea de código @app.route para que se vea así: @ app.route ('/ search4', methods = ['GET', 'POST']). Para más información sobre esto, vea los documentos de Flask, que están disponibles aquí <http://flask.pocoo.org>.

Nota para saber mas de Get y Post revisar estos dos archivos e imprimirlas:

/home/anton/webapp/material/https/Envio datos formulario html metodos get post diferencias ejemplos.pdf

/home/anton/webapp/material/https/Métodos HTTP.pdf

### Activar la depuración

### **Refinación del ciclo de edición / parada / inicio / prueba**

En este punto, después de haber guardado nuestro código modificado, es un curso de acción razonable detener la aplicación web en el símbolo del sistema y reiniciarla para probar nuestro nuevo código. Este ciclo de edición / detención / inicio / prueba funciona, pero se vuelve tedioso después de un tiempo (especialmente si terminas haciendo una larga serie de pequeños cambios en el código de tu webapp).

Para mejorar la eficiencia de este proceso, Flask te permite ejecutar tu webapp en el modo de depuración, que, entre otras cosas, reinicia automáticamente tu webapp cada vez que Flask note que tu código ha cambiado (normalmente como resultado de hacer y guardar un cambio) . Esto vale la pena hacerlo, así que vamos a cambiar la depuración cambiando la última línea de código en **vsearch4web.py** para que parezca esto:

app.run(debug=True) ← Cambia la depuración

Su código de programa debería verse así:

```
# vsearch4web1.py

from flask import Flask, render_template
from vsearch import search4letters
```

```

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4', methods=['POST'])
def do_search()->str:
    return str(search4letters('life the universe, and everthing','eiru,!'))

@app.route('/entry')
def entry_page() ->'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

app.run()

```

Ahora estamos listos para tomar este código para una prueba. Para ello, detenga la aplicación web en ejecución (por última vez) pulsando Ctrl-C y, a continuación, reiníciela en el símbolo del sistema pulsando la flecha hacia arriba y Intro.

En lugar de mostrar el habitual "Running on http:// 127 ..." mensaje, Flask escupe tres nuevas líneas de estado, que es su manera de decir que el modo de depuración está activo. Esto es lo que vimos en nuestro ordenador:

```

$ python3 vsearch4web.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger pin code: 228-903-465

```

Esta es la forma de Flask de decirle que su webapp se reiniciará automáticamente si cambia el código. También: no se preocupe si su código del depurador del perno es diferente del nuestro (que es ACEPTABLE). No utilizaremos este alfiler.

Ahora que estamos en funcionamiento nuevamente, interactuamos con nuestra aplicación web una vez más y veamos qué ha cambiado.

Vuelva al formulario de entrada escribiendo <http://127.0.0.1:5000/entry> en su navegador:

Welcome to search4letters on the web!

Use this form to submit a search request:

Phrase:

Letters:

When you're ready, click this button:

Todavía está  
bien

El error "Método no permitido" ha desaparecido, pero las cosas todavía no funcionan correctamente. Puede escribir cualquier frase en este formulario, luego haga clic en el botón "¡Do it !" Sin que aparezca el error. Si lo intentas un par de veces, notarás que los resultados devueltos son siempre los mismos (no importa qué frase o letras uses). Vamos a investigar lo que está pasando aquí.

{'u', 'e', 'i', 't', 'r'}

No matter what we type in as the phrase, the results are always the same.

Traducción: No importa lo que escribimos como la frase, los resultados son siempre los mismos.

## Accessing HTML Form Data with Flask

### Acceso a datos de formulario HTML con Flask

Nuestra aplicación web ya no falla con un error "Método no permitido". En su lugar, siempre devuelve el mismo conjunto de caracteres: u, e, coma, iy r. Si echa un vistazo rápido al código que se ejecuta cuando se publica la URL / search4, verá por qué esto es: los valores de la frase y las letras se codifican en la función:

```
...
@app.route('/search4', methods=['POST'])
def do_search() -> str:
    return str(search4letters('life, the universe, and everything', 'eiru,!'))
...
```

No importa lo que escribimos en el formulario HTML, nuestro código siempre va a utilizar estos valores codificados.

Nuestro formulario HTML publica sus datos en el servidor web, pero para poder hacer algo con los datos, necesitamos enmendar el código de nuestra aplicación web para aceptar los datos y realizar alguna operación en él.

Flask viene con un objeto incorporado denominado `request` que proporciona acceso fácil a los datos publicados. El objeto de solicitud o `request` contiene un atributo de diccionario llamado `formulario` que proporciona acceso a los datos de un formulario HTML publicados desde el navegador. Como forma es como cualquier otro diccionario de Python, soporta la misma nota de corchete que vio por primera vez en el Capítulo 3. Para acceder a un fragmento de datos del formulario, coloque el nombre del elemento del formulario entre corchetes:

Los datos de este elemento de formulario están disponibles en el código de nuestra aplicación web como "request.form['phrase']".

The diagram illustrates the relationship between an HTML template and its rendered form in a web browser. On the left, a code editor shows the `entry.html` file containing an HTML form with fields for 'Phrase' and 'Letters'. Arrows point from specific parts of the code to their corresponding elements in the rendered form on the right. The rendered form, displayed in a browser window titled 'Welcome to search4letters on the web!', shows the input fields filled with 'Phrase: aeiou' and 'Letters: aeiou'. A note below the browser states: 'The data from this form element is available in our webapp as "request.form['letters']"'.

```

    (% extends 'base.html' %)

    {% block body %}

    <h2>{{ the_title }}</h2>

    <form method='POST' action='/search4'>
    <table>
    <p>Use this form to submit a search request:</p>
    <tr><td>Phrase:</td><td><input name='phrase' type='TEXT' width='60'></td></tr>
    <tr><td>Letters:</td><td><input name='letters' type='TEXT' value='aeiou'></td></tr>
    </table>
    <p>When you're ready, click this button:</p>
    <p><input value='Do It!' type='SUBMIT'></p>
    </form>

    {% endblock %}

```

The HTML template (in the "entry.html" file)

The rendered form in our web browser

The data from this form element is available in our webapp as "request.form['letters']".

226 Chapter 5

#### traduciendo:

- La plantilla HTML (en el archivo "entry.html")
- El formulario representado en su navegador web
- Los datos de este elemento de aplicación de formulario están disponibles en nuestra web como "request.form['letters']".

## **Using Request Data in Your Webapp**

### **Uso de los datos de solicitud en su Webapp**

Para usar el objeto de petición **request**, importelo en la línea del **flask** en la parte superior del código del programa y, a continuación, acceda a los datos de **request.form** según sea necesario. Para nuestros propósitos, queremos reemplazar el valor de datos codificados en nuestra función **do\_search** con los datos del formulario. Al hacerlo, asegura que cada vez que se use el formulario HTML con diferentes valores de **frase** y **letras**, los resultados devueltos por nuestra aplicación web se ajusten en consecuencia.

Hagamos estos cambios en nuestro código de programa. Comience añadiendo el objeto de solicitud a la lista de importaciones de Flask. Para ello, cambie la primera línea de **vsearch4web.py** para parecerse a esto:

```
from flask import Flask, render_template, request
```

Añada  
"request" o  
"solicitud" a  
la lista de  
importaciones

Sabemos por la información de la última página que podemos acceder a la frase introducida en el formulario HTML dentro de nuestro código como `request.form['frase']`, mientras que

las letras ingresadas están disponibles para nosotros como `request.form['letters']`. Vamos a ajustar la función `do_search` para usar estos valores (y eliminar las cadenas codificadas):

```
Create two new variables... @app.route('/search4', methods=['POST'])  
def do_search() -> str:  
    phrase = request.form['phrase'] ...and assign the HTML form's data  
    letters = request.form['letters'] to the newly created variables...  
    return str(search4letters(phrase, letters))  
... entonces, utilice las variables en la llamada a "search4letters".
```

Cargando desde el nuevo entorno virtual y trabajar:

como vemos nos ubicamos en el entorno asi:

```
(HeadFirst) [anton@anton-pc ~]$ cd /home/anton/.virtualenvs/HeadFirst
```

```
(HeadFirst) [anton@anton-pc First_Codes]$ cd ch05  
(HeadFirst) [anton@anton-pc ch05]$ ls  
__pycache__      static          vsearch4web1.py      vsearch_webapp3.py  
dunder.py        templates       vsearch_requests.py  vsearch_webapp4.py  
hello_flask.py   vsearch.py     vsearch_webapp.py   vsearch_webapp5.py  
hello_flask2.py  vsearch4web.py  vsearch_webapp2.py  webapp
```

Desde sublime Text lo codifico pero lo ejecuto desde el prompt del entorno virtual:

```
^C(HeadFirst) [anton@anton-pc ch05]$ python hello_flask.py  
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)  
127.0.0.1 - - [23/Apr/2017 22:05:17] "GET / HTTP/1.1" 200 -
```

y cargamos la web:



Hello world from Flask!

## **Automatic Reloads - Recarga automática**

Ahora ... antes de hacer cualquier otra cosa (habiendo hecho los cambios en el código del programa anterior) guarde su archivo **vsearch4web.py**, luego vaya a su símbolo del sistema y eche un vistazo a los mensajes de estado producidos por su aplicación web. Esto es lo que vimos (debería ver algo similar):

```
#vsearch4web.py

from flask import Flask, render_template, request

from vsearch import search4letters


app = Flask(__name__)

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')


if __name__ == '__main__':
    app.run(debug=True)
```

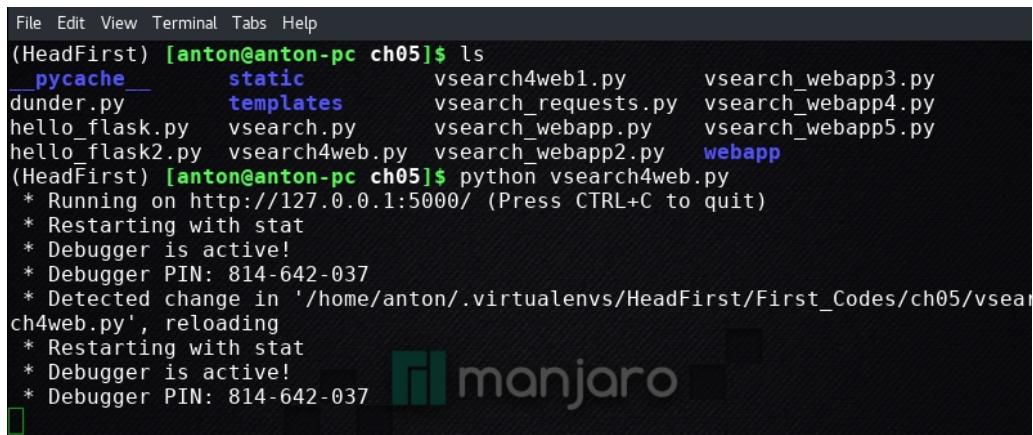
Y al ejecutarlo desde el terminal:

```
$ python3 vsearch4web.py
* Restarting with stat
* Debugger is active!
* Debugger pin code: 228-903-465
127.0.0.1 -- [23/Nov/2015 22:39:11] "GET /entry HTTP/1.1" 200 -
127.0.0.1 -- [23/Nov/2015 22:39:11] "GET /static/hf.css HTTP/1.1" 200 -
127.0.0.1 -- [23/Nov/2015 22:17:58] "POST /search4 HTTP/1.1" 200 -
* Detected change in 'vsearch4web.py', reloading
* Restarting with stat
* Debugger is active!
* Debugger pin code: 228-903-465
```

El depurador de Flask ha detectado los cambios de código y ha reiniciado tu aplicación web para ti. Muy práctico, ¿eh?



En mi maquina sale:



```
File Edit View Terminal Tabs Help
(HeadFirst) [anton@anton-pc ch05]$ ls
__pycache__      static          vsearch4web1.py    vsearch_webapp3.py
dunder.py        templates       vsearch_requests.py vsearch_webapp4.py
hello_flask.py   vsearch.py     vsearch_webapp.py   vsearch_webapp5.py
hello_flask2.py  vsearch4web.py vsearch_webapp2.py  webapp
(HeadFirst) [anton@anton-pc ch05]$ python vsearch4web.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 814-642-037
* Detected change in '/home/anton/.virtualenvs/HeadFirst/First_Codes/ch05/vsearch4web.py', reloading
* Restarting with stat
* Debugger is active!
* Debugger PIN: 814-642-037
```

No entre en pánico si ve algo distinto de lo que se muestra aquí. La recarga automática sólo funciona si los cambios de código que realiza son correctos. Si el código tiene errores, la webapp bombardea a la línea de comandos. Para volver a empezar, corrija los errores de codificación y reinicie su aplicación web manualmente (pulsando la flecha hacia arriba y luego Intro).

Ahora que hemos cambiado nuestra aplicación web para aceptar (y procesar) los datos de nuestro formulario HTML, podemos lanzar diferentes frases y letras en él, y debe hacer lo correcto:



traducción:

- La frase contiene todas menos una de las letters publicadas en el servidor web.
- Sólo aparece la letra 'y' en la frase publicada.
- Recuerde: un conjunto vacío aparece como "set()", por lo que significa que ninguna de las letras 'm', 'n', 'p' o 'q' aparecen en la frase.

comprobamos:

The screenshot shows a web browser window with the URL `127.0.0.1:5000/entry`. The page title is "Welcome to search4lett". It contains a form for submitting a search request. The "Phrase" field contains "this is a test of posting capability" and the "Letters" field contains "aeiou". A button labeled "Do it!" is present.

The screenshot shows a web browser window with the URL `127.0.0.1:5000/search4`. The page title is "Here are your results:". It displays the submitted data and the results of the search. The "Phrase" field contains "this is a test of posting capability" and the "Letters" field contains "aeiou". The results state: "When 'this is a test of posting capability' is searched for 'aeiou', the following results are returned: {'i', 'e', 'o', 'a'}".

La frase contiene todas menos una de las letters publicadas en el servidor web.

The image consists of two vertically stacked screenshots of a web browser window. Both screenshots have a light beige background and show a search interface.

**Screenshot 1 (Top): Welcome to search4letters on the web!**

This screenshot shows the initial search form. The title bar says "Welcome to search4letters on the web!". Below it, instructions say "Use this form to submit a search request:". There are two input fields: "Phrase:" containing "life, the universe, and everything" and "Letters:" containing "xyz". A button labeled "Do it!" is below the letters field.

**Screenshot 2 (Bottom): Here are your results:**

This screenshot shows the results page. The title bar says "Here are your results:". It displays the submitted data: "Phrase: life, the universe, and everything" and "Letters: xyz". Below this, it states: "When 'life, the universe, and everything' is searched for 'xyz', the following results are returned: {y}".

Sólo aparece la letra 'y' en la frase publicada.

Recuerde: un conjunto vacío aparece como "set()", por lo que significa que ninguna de las letras 'm', 'n', 'p' o 'q' aparecen en la frase.

Menu Traductor de Google Source code beautifier Welcome to search4lett... 127.0.0.1:5000/entry

Phrase: hitch-hiker  
Letters: mnopq

Welcome to search4letters on the web!

Use this form to submit a search request:

When you're ready, click this button:

Do it!

Menu Traductor de Google Source code beautifier Here are your results: 127.0.0.1:5000/search4

Phrase: hitch-hiker  
Letters: mnopq

Here are your results:

You submitted the following data:

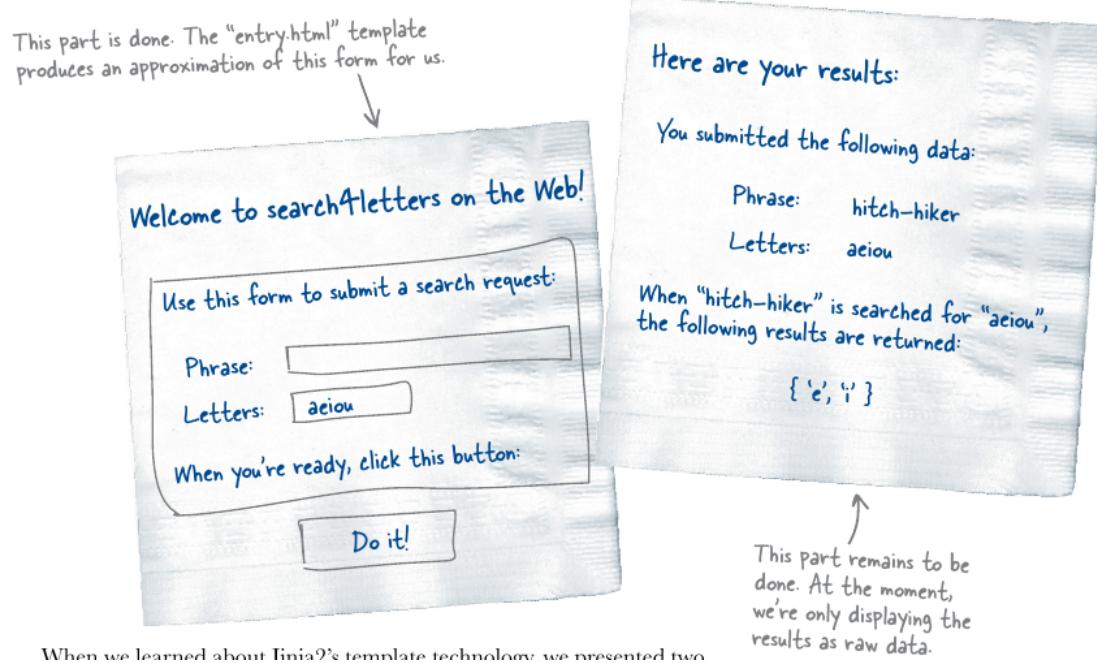
When "hitch-hiker" is searched for "mnopq", the following results are returned:

set()

## Produciendo los resultados como HTML

En este punto, la funcionalidad asociada con nuestra aplicación web está funcionando: cualquier navegador web puede enviar una combinación de **frase / letras**, y nuestra aplicación web invoca **search4letters** en nuestro nombre, devolviendo cualquier resultado. Sin embargo, la producción producida no es realmente una página web HTML, sino sólo los datos en bruto devueltos como texto al explorador en espera (que lo muestra en pantalla).

Recuerde las especificaciones de la parte posterior de la servilleta de principios de este capítulo. Esto es lo que esperábamos producir:



Traducción:

- Esta parte está hecha. La plantilla "entry.html" produce una aproximación de este formulario para nosotros.
- Esta parte todavía queda por hacer. Por el momento, sólo mostramos los resultados como datos sin procesar.

Cuando nos enteramos de la tecnología de plantilla de Jinja2, presentamos dos plantillas HTML. El primero, **entry.html**, se usa para producir el formulario. El segundo, **results.html**, se utiliza para mostrar los resultados. Vamos a usar ahora para tomar nuestra salida de datos sin procesar y convertirlo en HTML.

there are no  
Dumb Questions

P: ¿Es posible usar Jinja2 para planificar datos de texto que no sean HTML?

R: Sí. Jinja2 es un motor de plantilla de texto que se puede poner a muchos usos. Dicho esto, su caso de uso típico es con proyectos de desarrollo web (como se utiliza aquí con Flask), pero no hay nada que le impida usarlo con otros datos de texto si realmente lo desea.

## ***Calculating the Data We Need***

### ***Cálculo de los datos que necesitamos***

Recordemos el contenido de la plantilla results.html presentada anteriormente en este capítulo. El marcado específico de Jinja2 se destaca:

This is results.html".

```
{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>

<p>You submitted the following data:</p>
<table>
<tr><td>Phrase:</td><td>{{ the_phrase }}</td></tr>
<tr><td>Letters:</td><td>{{ the_letters }}</td></tr>
</table>

<p>When "{{the_phrase}}" is search for "{{ the_letters }}", the following
results are returned:</p>
<h3>{{ the_results }}</h3>

{% endblock %}
```

```
<!--Comentario en HTML -->
<!--web result.html que hereda de base.html-->

{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>

<p>You submitted the following data:</p>
<table>
<tr><td>Phrase:</td><td>{{ the_phrase }}</td></tr>
<tr><td>Letters:</td><td>{{ the_letters }}</td></tr>
</table>
```

```

<p>When "{{the_phrase}}" is searched for "{{ the_letters }}", the following
results are returned:</p>

<h3>{{ the_results }}</h3>

{% endblock %}

```

Los nombres resaltados entre paréntesis dobles son variables Jinja2 que toman su valor de las variables correspondientes en tu código Python. Hay cuatro de estas variables: **the\_title**, **the\_phrase**, **the\_letters** y **the\_results**. Eche otra mirada al código de la función **do\_search** (abajo), que vamos a ajustar en un momento para representar la plantilla HTML mostrada arriba. Como se puede ver, esta función ya contiene dos de las cuatro variables que necesitamos para procesar la plantilla (y para mantener las cosas lo más simple posible, hemos utilizado nombres de variables en nuestro código Python que son similares a los utilizados en la plantilla Jinja2):



```

Aquí hay dos de      @app.route('/search4', methods=['POST'])
los cuatro          def do_search() -> str:
valores que          → phrase = request.form['phrase']
necesitamos.          letters = request.form['letters']
                      return str(search4letters(phrase, letters))

```

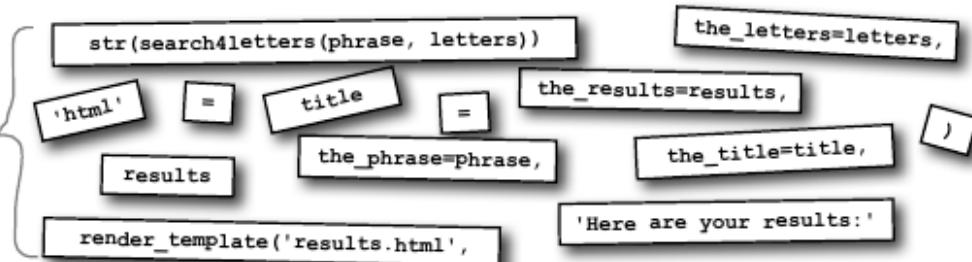
Los dos restantes argumentos de plantilla requeridos (**the\_title** y **the\_results**) todavía necesitan ser creados a partir de variables en esta función y valores asignados.

Podemos asignar la cadena "**Here are your results:**" a **the\_title**, y luego asignar la llamada a **search4letters** a **the\_results**. Las cuatro variables se pueden pasar a la plantilla **results.html** como argumentos antes de la representación.

## Plantilla Imanes

Los autores de Head First se reunieron y, sobre la base de los requisitos para la función actualizada **do\_search** descrita en la parte inferior de la última página, escribieron el código requerido. En el verdadero estilo de Head First, lo hicieron con la ayuda de algunos imanes de codificación ... y una nevera (mejor si no se pregunta). Después de su éxito, las celebraciones resultantes se hicieron tan ruidosas que un editor de ciertas series se topó con la nevera (mientras cantaba la canción de la cerveza) y ahora los imanes están en todo el piso. Su trabajo es colocar los imanes de nuevo en sus ubicaciones correctas en el código.

Here are the magnets you have to work with.



Traducciendo:

- Decida qué código magnético va en cada una de las ubicaciones de línea discontinua.
  - Aquí están los imanes con los que tiene que trabajar.

## **Plantilla Imanes Solución**

Habiendo hecho una nota para mantener un futuro ojo en el consumo de cerveza de un cierto editor de serie, se puso a trabajar restaurando todos los imanes de código para la función actualizada `do_search`. Tu trabajo era pegar los imanes en sus ubicaciones correctas en el código. Esto es lo que nos ocurrió cuando realizamos esta tarea:

```

from flask import Flask, render_template, request
from vsearch import search4letters

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']

    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))

    return render_template('results.html',
                           the_phrase=phrase,
                           the_letters=letters,
                           the_title=title,
                           the_results=results,
                           )

```

*Change the annotation to indicate that this function now returns HTML, not a plain-text string (as in the previous version of this code).*

*Create a Python variable called "title" ...*

*Create another Python variable called "results" ...*

*Render the "results.html" template. Remember: this template expects four argument values.*

*Don't forget the closing parenthesis to end the function call.*

*... and assign a string to "title".*

*... and assign the results of the call to "search4letters" to "results".*

*Each Python variable is assigned to its corresponding Jinja2 argument. In this way, data from our program code is passed into the template.*

```

@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

```

```

app.run(debug=True)

```

traduciendo:

- 1 . Crear una variable de Python llamada "title" ...
- 2 . Crear otra variable de Python llamada "results" ...
- 3 . No olvide el paréntesis de cierre para finalizar la llamada de función.
- 4 . Cambie la anotación para indicar que esta función ahora devuelve HTML, no una cadena de texto sin formato (como en la versión anterior de este código)
- 5 . ... y asignar una cadena a "title".
- 6 . ... y asignar los resultados de la llamada a "search4letters" a "results".
- 7 . Cada variable Python se asigna a su correspondiente argumento Jinja2. De esta manera, los datos de nuestro código de programa se pasan a la plantilla.

Ahora que los imanes están de nuevo en sus ubicaciones correctas, realice estos cambios de código en su copia de **vsearch4web.py**. Asegúrese de guardar su archivo para asegurarse de que Flask recargue automáticamente su aplicación web. Ahora estamos listos para otra prueba.

Vamos a probar la nueva versión de nuestra aplicación web utilizando los mismos ejemplos anteriores de este capítulo. Tenga en cuenta que Flask reinició su aplicación web el momento en que guardó su código.



you are here ➤ 233

## Una pequeña redirección

### Cómo añadir un toque de acabado

Echemos un vistazo al código que compone actualmente `vsearch4web.py`. Esperemos que, por ahora, todo este código debe tener sentido para usted. Un pequeño elemento sintáctico que a menudo confunde a los programadores que se mueven a Python es la inclusión de la coma final en la llamada a `render_template`, ya que la mayoría de los programadores sienten que esto debería ser un error de sintaxis y no debería permitirse. A pesar de que parece un poco extraño (al principio), Python lo permite, pero no lo requiere, para que podamos seguir con seguridad y no preocuparnos por ello:

```

from flask import Flask, render_template, request
from vsearch import search4letters

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results)

@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

app.run(debug=True)

```

This extra comma looks a little strange, but is perfectly fine (though optional) Python syntax.

Traduciendo: Esta coma extra parece un poco extraña, pero está perfectamente bien (aunque opcional) sintaxis de Python.

```

#vsearch4web.py

from flask import Flask, render_template, request
from vsearch import search4letters

app = Flask(__name__)

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results)

```

```

        the_phrase=phrase,
        the_letters=letters,
        the_results=results,)

@app.route('/')
@app.route('/entry')

def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

if __name__ == '__main__':
    app.run(debug=True)

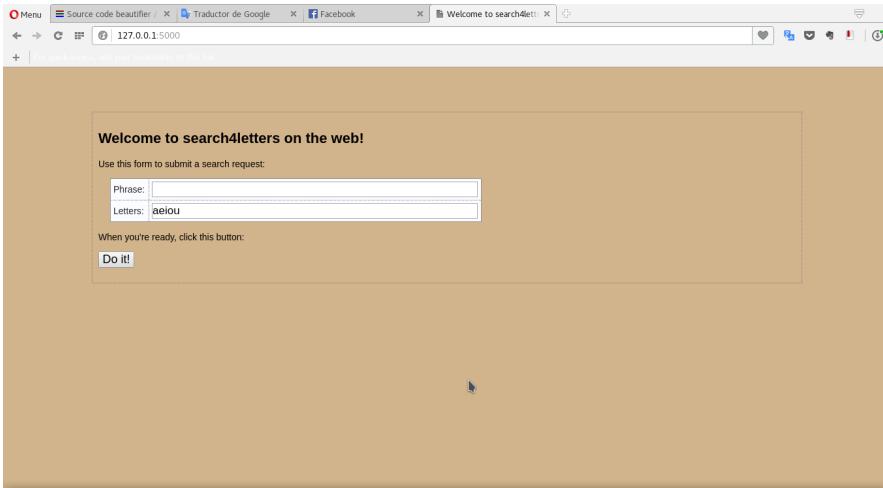
```

para ejecutar desde el terminal:

```

(HeadFirst) [anton@anton-pc First_Codes]$ ls
'Head First Python, 2nd Edition.pdf' __MACOSX ch01 ch02 ch03 ch04 ch05 ch06 ch07 ch08 ch09 ch10 ch11 ch12
(HeadFirst) [anton@anton-pc First_Codes]$ cd ch05
(HeadFirst) [anton@anton-pc ch05]$ ls
__pycache__ hello_flask2.py vsearch.py vsearch_requests.py vsearch_webapp3.py webapp
dunder.py static vsearch4web.py vsearch_webapp.py vsearch_webapp4.py
hello_flask.py templates vsearch4web1.py vsearch_webapp2.py vsearch_webapp5.py
(HeadFirst) [anton@anton-pc ch05]$ python vsearch4web.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 814-642-037
127.0.0.1 - - [29/Apr/2017 13:44:12] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Apr/2017 13:44:12] "GET /static/hf.css HTTP/1.1" 200 -
127.0.0.1 - - [29/Apr/2017 13:44:13] "GET /favicon.ico HTTP/1.1" 404 -

```



Esta versión de nuestra webapp soporta tres URLs: `/`, `/search4` y `/entry`, con algunas que datan de la primera webapp Flask que creamos (justo al inicio de este capítulo). En este momento, el `/` URL muestra el mensaje amistoso, pero algo inútil, "Hola mundo de Flask!".

Podríamos eliminar esta URL y su función hello asociada de nuestro código (ya que ya no necesitamos), pero al hacerlo se produciría un error "No encontrado" 404 en cualquier navegador web que contacte nuestra aplicación web en URL `/`, que es el URL predeterminada para la mayoría de webapps y sitios web. Para evitar este mensaje de error molesto, vamos a pedir a Flask que redirija cualquier solicitud de URL `/` a la URL `/entry`. Hacemos esto ajustando la función hello para devolver un redirecciónamiento HTML a cualquier navegador web que solicite la `/` URL, sustituyendo efectivamente la URL `/entry` por cualquier solicitud hecha por `/`.

## ***Redirect to Avoid Unwanted Errors***

### ***Redirigir para evitar errores no deseados***

Para usar la tecnología de redirección de Flask, añada `redirect` a la línea de importación `from flask` (en la parte superior de su código), luego cambie el código de la función `hello` para que se vea así:

To use Flask's redirection technology, add `redirect` to the `from flask import` line (at the top of your code), then change the `hello` function's code to look like this:

```
from flask import Flask, render_template, request, redirect  
from vsearch import search4letters  
  
app = Flask(__name__)  
  
@app.route('/')  
def hello() -> '302':  
    return redirect('/entry')  
  
...  
  
The rest of  
the code  
remains  
unchanged.
```

Add "redirect" to the list of imports.

Adjust the annotation to more clearly indicate what's being returned by this function. Recall that HTTP status codes in the 300-399 range are redirections, and 302 is what Flask sends back to your browser when "redirect" is invoked.

Call Flask's "redirect" function to instruct the browser to request an alternative URL (in this case, "/entry").

traduciendo:

1. El resto del código permanece sin cambios.
2. Llame a la función de "redirect" de Flask para indicar al navegador que solicite una URL alternativa en este caso, "/entry".
3. Ajuste la anotación para indicar más claramente lo que está siendo devuelto por esta función. Recuerde que los códigos de estado HTT en el rango 300-399 son redirecciones, y 302 es lo que Flask envía de vuelta a su navegador cuando se invoca "redirección".
4. Agregue "redireccionar" a la lista de importaciones.

Esta pequeña edición garantiza que los usuarios de nuestra webapp se muestren en el formulario HTML si solicitan la URL `/entry` o `/`.

Realice este cambio, guarde su código (lo que activa una recarga automática) y, a continuación, intente señalar su navegador a cada una de las URL. El formulario HTML debe aparecer cada vez. Eche un vistazo a los mensajes de estado que se muestra en su aplicación web en el símbolo del sistema. Usted puede ver bien algo como esto:

Has guardado tu código, así que Flask ha vuelto a cargar tu aplicación web.

```
...  
* Detected change in 'vsearch4web.py', reloading  
* Restarting with stat  
* Debugger is active!  
* Debugger pin code: 228-903-465  
127.0.0.1 - - [24/Nov/2015 16:54:13] "GET /entry HTTP/1.1" 200 - ←  
127.0.0.1 - - [24/Nov/2015 16:56:43] "GET / HTTP/1.1" 302 - } ↘  
127.0.0.1 - - [24/Nov/2015 16:56:44] "GET /entry HTTP/1.1" 200 - } ↘
```

Se hace una solicitud para la URL `/entry`, y se sirve inmediatamente. Tenga en cuenta el código de estado 200 (y recuerde anteriormente en este capítulo que los códigos en la gama 200-299 son mensajes de éxito: el servidor ha recibido, entendido y procesado la solicitud del cliente).

Cuando se hace una solicitud para la URL `/`, nuestra webapp responde primero con la redirección 302, y luego el navegador web envía otra solicitud para la URL `/entry`, que es servida con éxito por nuestra aplicación web 200 código de estado).

Como estrategia, nuestro uso de la redirección aquí funciona, pero es un poco derrochador. Una única solicitud para la URL / se convierte en dos solicitudes cada vez (aunque la caché en el lado del cliente puede ayudar, esto todavía no es óptimo). Si sólo Flask podría de alguna manera asociar más de una URL con una función determinada, eliminando efectivamente la necesidad de la redirección por completo. Eso sería bueno, ¿no?

## No más redirección

### Funciones pueden tener varias URL

No es difícil adivinar adónde vamos con esto, ¿verdad?

Resulta que Flask puede asociar más de una URL con una función dada, lo que puede reducir la necesidad de redirecciones como la demostrada en la última página. Cuando una función tiene más de una URL asociada con ella, Flask intenta hacer coincidir cada una de las URL a su vez, y si encuentra una coincidencia, la función se ejecuta.

No es difícil aprovechar esta característica de Flask. Para empezar, elimine el redireccionamiento `redirect` de la línea de importación del flask en la parte superior del código del programa; Ya no lo necesitamos, así que no vamos a importar el código que no tenemos intención de usar. A continuación, utilice su editor, corte la línea de código `@app.route('/')` y péguelo por encima de la línea `@ app.route ('/ entry')` cerca de la parte inferior de su archivo. Finalmente, elimine las dos líneas de código que componen la función hello, ya que nuestra aplicación web ya no las necesita.

Cuando hayas terminado de hacer estos cambios, el código del programa debería tener este aspecto:

```
from flask import Flask, render_template, request
from vsearch import search4letters

app = Flask(__name__)

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

The "hello"
function
has been
removed.

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

app.run(debug=True)
```

We no longer need to import "redirect", so we've removed it from this import line.

The "entry\_page" function now has two URLs associated with it.

Traducimos:

1. Se ha eliminado la función "hello".
2. La función "entry\_page" ahora tiene dos URL asociadas.
3. Ya no es necesario importar "redirect", por lo que hemos eliminado de esta línea de importación.

```

from flask import Flask, render_template, request
from vsearch import search4letters


app = Flask(__name__)

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

if __name__ == '__main__':
    app.run(debug=True)

```

Guardar este código (que activa una recarga) nos permite probar esta nueva funcionalidad. Si visita el URL /, aparece el formulario HTML. Un rápido vistazo a los mensajes de estado de su webapp confirma que procesar / ahora da lugar a una solicitud, en lugar de dos (como era anteriormente):

```

...
* Detected change in 'vsearch4web.py', reloading
* Restarting with stat
* Debugger is active!
* Debugger pin code: 228-903-465
127.0.0.1 - - [24/Nov/2015 16:59:10] "GET / HTTP/1.1" 200 -

```

As always, the new version of our webapp reloads.

One request, one response. That's more like it. ☺

Traduciendo:

1. Como siempre, la nueva versión de nuestra webapp se vuelve a cargar.
2. Una solicitud, una respuesta. Eso es más como él.

## ***Actualización de lo que sabemos***

Acabamos de pasar las últimas 40 páginas creando una pequeña aplicación web que expone la funcionalidad proporcionada por nuestra función search4letters a la World Wide Web (a través de un simple sitio web de dos páginas). Por el momento, la aplicación web se ejecuta localmente en su computadora. En un poco, vamos a discutir la implementación de su aplicación web en la nube, pero por ahora vamos a actualizar lo que sabes:

### ***¿Es todo lo que hay en este capítulo?***

Te perdonarían pensar que este capítulo no introduce mucho nuevo Python. No lo hace. Sin embargo, uno de los puntos de este capítulo fue mostrar cuantas líneas de código Python necesitas para producir algo que es generalmente útil en la Web, gracias en gran parte a nuestro uso de Flask. El uso de una tecnología de plantilla ayuda mucho, ya que le permite mantener su código Python (la lógica de su webapp) separado de sus páginas HTML (la interfaz de usuario de su webapp).

No es un montón de trabajo para ampliar esta aplicación web para hacer más. De hecho, podrías hacer que un whiz-kid de HTML produzca más páginas para ti mientras te concentras en escribir el código de Python que une todo. A medida que su webapp escalas, esta separación de funciones realmente comienza a pagar. Tienes que concentrarte en el código Python (ya que eres el programador en el proyecto), mientras que el HTML whiz-kid se concentra en el marcado (ya que es su bailiwick). Por supuesto, ambos tienen que aprender un poco sobre las plantillas de Jinja2, pero eso no es demasiado difícil, ¿verdad?

**Tengo que amar python en cualquier lugar**

## ***Preparing Your Webapp for the Cloud***

### ***Preparando su Webapp para la nube***

Con su webapp trabajando a la especificación local en su computadora, es hora de pensar en desplegarla para el uso por una audiencia más amplia. Hay un montón de opciones aquí, con muchos diferentes web de alojamiento basado en configuraciones disponibles para

usted como un programador de Python. Un servicio popular está basado en la nube, alojado en AWS, y se llama PythonAnywhere. Nos encanta en Head First Labs.

Como casi todas las demás soluciones de despliegue alojadas en la nube, PythonAnywhere le gusta controlar cómo comienza tu webapp. Para ti, esto significa que PythonAnywhere asume la responsabilidad de llamar a `app.run()` en tu nombre, lo que significa que ya no necesitas llamar a `app.run()` en tu código. De hecho, si intentas ejecutar esa línea de código, PythonAnywhere simplemente se niega a ejecutar tu aplicación web.

Una solución simple a este problema sería quitar esa última línea de código de su archivo antes de desplegar a la nube. Esto ciertamente funciona, pero significa que necesitas volver a poner esa línea de código cuando ejecutes tu aplicación web localmente. Si está escribiendo y probando código nuevo, debe hacerlo localmente (no en PythonAnywhere), ya que utiliza la nube únicamente para la implementación, no para el desarrollo. Además, la eliminación de la línea ofensiva de código efectivamente equivale a tener que mantener dos versiones de la misma aplicación web, una con y una sin esa línea de código. Esto nunca es una buena idea (y es más difícil de manejar a medida que realiza más cambios).

Sería bueno si hubiera una forma de ejecutar código de forma selectiva basándose en si está ejecutando su aplicación web localmente en su computadora o remotamente en PythonAnywhere ...

He mirado un montón de programas de Python en línea, y muchos de ellos contienen una suite cerca de la parte inferior que comienza con: `if __name__ == '__main__':` ¿Algo así ayudaría aquí?

### **Sí, esa es una gran sugerencia.**

Esa línea de código particular se utiliza en muchos programas de Python. Se refiere cariñosamente como "dunder name dunder main". Para entender por qué es tan útil (y por qué podemos aprovecharlo con PythonAnywhere), echemos un vistazo más de cerca a lo que hace y cómo funciona.

## **Dunder Nombre Dunder Principal Subir Cerrar**

Para entender el constructo de programación sugerido al final de la última página, veamos un pequeño programa que lo usa, llamado `dunder.py`. Este programa de tres líneas comienza mostrando un mensaje en la pantalla que imprime el espacio de nombres actualmente activo, almacenado en la variable `__name__`. Una instrucción `if` comprueba si el valor de `__name__` está establecido en `__main__` y, si es, se muestra otro mensaje confirmando el valor de `__name__` (es decir, el código asociado con `if` si se ejecuta):

El código de programa "dunder.py", las tres líneas del mismo.

```
    print('We start off in:', __name__)
    if __name__ == '__main__':
        print('And end up in:', __name__)
```

Muestra el valor de "\_\_name\_\_".

Muestra el valor de "\_\_name\_\_" si está establecido en "\_\_main\_\_".

```
#dunder.py
```

```
print('We start off in:', __name__)
if __name__ == '__main__':
    print('And end up in:', __name__)
```

Utilice su editor (o IDLE) para crear el archivo dunder.py, luego ejecute el programa en el símbolo del sistema para ver qué sucede. Si está en Windows, utilice este comando:

```
C:\> py -3 dunder.py
```

Si está en Linux o Mac OS X, utilice este comando:

```
$ python3 dunder.py
```

Lo comprobamos:

```
(HeadFirst) [anton@anton-pc ch05]$ python dunder.py
We start off in: __main__
And end up in: __main__
(HeadFirst) [anton@anton-pc ch05]$
```

Pero desde sublime tambien ejecuta:

The screenshot shows a Sublime Text window with the following details:

- File Menu:** File, Edit, Selection, Find, View.
- Toolbar:** Goto, Tools, Project, Preferences, Help.
- Project Explorer:** Shows a directory structure:
  - First\_Codes
  - \_\_MACOSX
  - ch01
  - ch02
  - ch03
  - ch04
  - ch05
    - .cache
    - .pycache\_
    - static
    - templates
    - webapp
      - static
      - templates
        - usearch4web.py
- Code Editor:** A tab titled "dunder.py" containing the following code:

```
1 #dunder.py
2
3 print('We start off in:', __name__)
4 if __name__ == '__main__':
5     print('And end up in:', __name__)
6
```
- Output Panel:** Displays the execution results:

```
We start off in: main
And end up in: main
[Finished in 0.1s] ]
```
- Status Bar:** Line 1, Column 1 | Tab Size: 4 | Python (Django)

Independientemente del sistema operativo que esté ejecutando, el programa `dunder.py`, cuando se ejecuta directamente por Python, produce esta salida en la pantalla:

We start off in: main  
And end up in: main

Cuando se ejecuta directamente por Python, ambas llamadas a "print" muestran la salida.

Hasta aquí todo bien.

Ahora, mira lo que sucede cuando importamos el archivo `dunder.py` (que, recuerda, también es un módulo) en el prompt `>>>`. Estamos mostrando la salida en Linux / Mac OS X aquí. Para hacer lo mismo en Windows, reemplace `python3` (abajo) por `py -3`:

```
$ python3
Python 3.5.1 ...
Type "help", "copyright", "credits" or "license" for more information.
>>> import dunder
We start off in: dunder ← Look at this: there's only a single line displayed (as
opposed to two), as "__name__" has been set to
"dunder" (which is the name of the imported module)
```

traducimos:

1 . Mira esto: sólo hay una línea mostrada (en contraposición a dos), ya que "`__name__`" se ha establecido en "dunder" (que es el nombre del módulo importado).

```
(HeadFirst) [anton@anton-pc ch05]$ python
Python 3.6.0 (default, Jan 16 2017, 13:35:36)
[GCC 6.3.1 20170109] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import dunder
We start off in: dunder
>>> █
```

Aquí está el pedacito que usted necesita entender: si su código de programa se ejecuta directamente por Python, una declaración if como la de `dunder.py` devuelve True, ya que el espacio de nombres activo es `__main__`. Si, no obstante, su código de programa es importado como un módulo (como en el ejemplo del ejemplo de Shell de Python anterior), la sentencia if devuelve siempre False, ya que el valor de `__name__` no es `__main__`, sino el nombre del módulo importado (dunder en este caso).

### ***Exploiting Dunder Name Dunder Main***

#### ***Explotación Dunder Nombre Dunder Main***

Ahora que ya sabes lo que hace el nombre de dunder principal dunder, vamos a explotarlo para resolver el problema que tenemos con PythonAnywhere que quieren ejecutar `app.run()` en nuestro nombre.

Resulta que cuando PythonAnywhere ejecuta nuestro código de webapp, lo hace importando el archivo que contiene nuestro código, tratándolo como cualquier otro módulo. Si la importación tiene éxito, PythonAnywhere llama a `app.run()`. Esto explica por qué dejar `app.run()` en la parte inferior de nuestro código es un problema para PythonAnywhere, ya que asume que la llamada `app.run()` no se ha realizado, y falla al iniciar nuestra webapp cuando `app.run()` se ha hecho una llamada.

Para evitar este problema, envuelva la llamada `app.run()` en un nombre dunder dunder main si declaración (que asegura que `app.run()` nunca se ejecuta cuando se importa el código webapp).

Edité `vsearch4web.py` una última vez (en este capítulo, de todos modos) y cambie la última línea de código a esto:

```
if __name__ == '__main__':
    app.run(debug=True)
```

La línea de código "app.run ()" ahora sólo se ejecuta cuando se ejecuta directamente por Python.

Este pequeño cambio le permite continuar ejecutando su aplicación web localmente (donde se ejecutará la línea `app.run()`), así como implementar su aplicación web en PythonAnywhere (donde la línea `app.run()` no se ejecutará). No importa dónde se ejecute su aplicación web, ahora tiene una versión de su código que hace lo correcto.

### ***Deploying to PythonAnywhere (well... almost)***

#### ***Implementación a PythonAnywhere (bueno ... casi)***

Todo lo que queda es que realice esa implementación real en el entorno alojado en cloud de PythonAnywhere.

Tenga en cuenta que, a los efectos de este libro, el despliegue de su aplicación web en la nube no es un requisito absoluto. A pesar de que tenemos la intención de extender vsearch4web.py con funcionalidad adicional en el próximo capítulo, no es necesario implementar en PythonAnywhere para seguir adelante. Usted puede felizmente seguir editando / ejecutar / probar su aplicación web localmente como lo extendemos en el próximo capítulo (y más allá).

Sin embargo, si realmente desea implementar en la nube, consulte el Apéndice B, que proporciona instrucciones paso a paso sobre cómo completar la implementación en PythonAnywhere. No es difícil, y no tomará más de 10 minutos.

Ya sea que se esté desplegando o no en la nube, lo veremos en el capítulo siguiente, donde comenzaremos a ver algunas de las opciones disponibles para guardar los datos desde sus programas Python.

## appendix b: pythonanywhere



# \* Deploying Your Webapp \*

## *Implementación de su Webapp*

Puedo desplegar mi webapp a la nube en unos 10 minutos?!?!? No lo creo

Al final del capítulo 5, afirmamos que el despliegue de su aplicación web a la nube estaba a sólo 10 minutos.

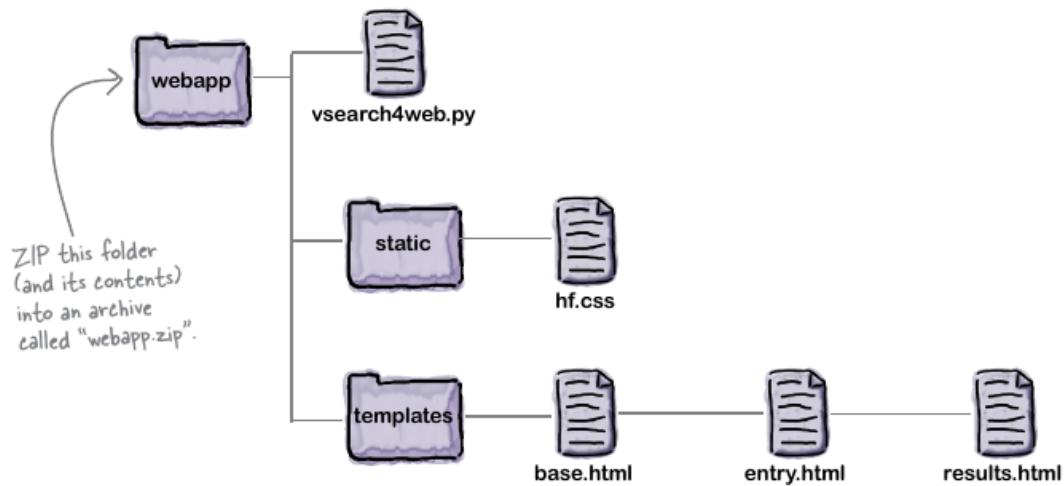
Es hora de cumplir con esa promesa. En este apéndice, vamos a llevarlo a través del proceso de despliegue de su webapp en PythonAnywhere, pasando de cero a desplegado en unos 10 minutos. PythonAnywhere es un favorito entre la comunidad de programación de Python, y no es difícil ver por qué: funciona exactamente como lo esperabas, tiene un gran soporte para Python (y Flask) y, lo mejor de todo, puedes empezar a hospedar Su webapp sin costo alguno. Echemos un vistazo a PythonAnywhere.

### **Step 0: A Little Prep**

### **Paso 0: Preparación**

Por el momento, tiene su código webapp en su computadora en una carpeta llamada webapp, que contiene el archivo `vsearch4web.py` y las carpetas estáticas y de plantillas (como se muestra a continuación). Para preparar todas estas cosas para la implementación,

cree un archivo ZIP de todo lo que contenga en su carpeta webapp y llame al archivo webapp.zip:



traduciendo:

ZIP esta carpeta (y su contenido) en un archivo llamado "webapp.zip".

Además de webapp.zip, también debe cargar e instalar el módulo vsearch del Capítulo 4. Por ahora, todo lo que necesita hacer es localizar el archivo de distribución que creó en ese entonces. En nuestro ordenador, el archivo se llama vsearch-1.0.tar.gz y está almacenado en nuestra carpeta mymodules / vsearch / dist (en Windows, el archivo probablemente se llama vsearch- 1.0.zip).

Recuerde del Capítulo 4 que el módulo "setuptools" de Python crea ZIP en Windows y archivos .tar.gz en todo lo demás.

No necesita hacer nada con ningún archivo de archivo ahora mismo. Sólo tiene que hacer una anotación de donde ambos archivos están en su computadora para que sean fáciles de encontrar cuando los suba a PythonAnywhere. Siéntase libre de agarrar un lápiz y garabatear la ubicación de cada archivo aquí:

**webapp.zip**

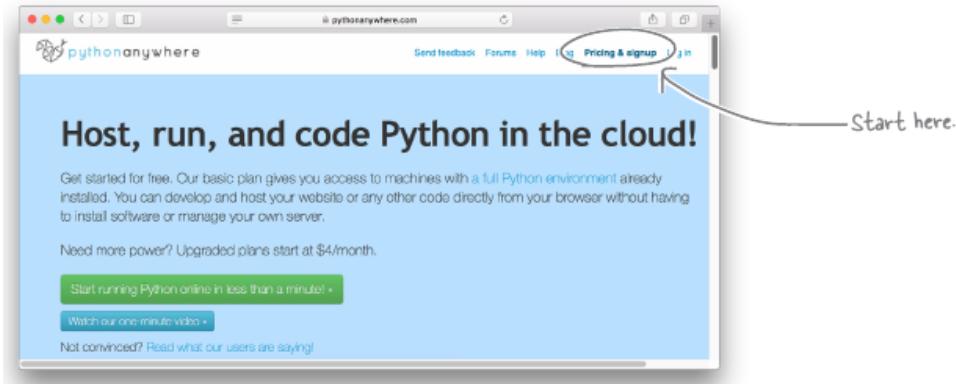
**vsearch-1.0.tar.gz**

Esto es "vsearch.zip" en su lugar si está en Windows.

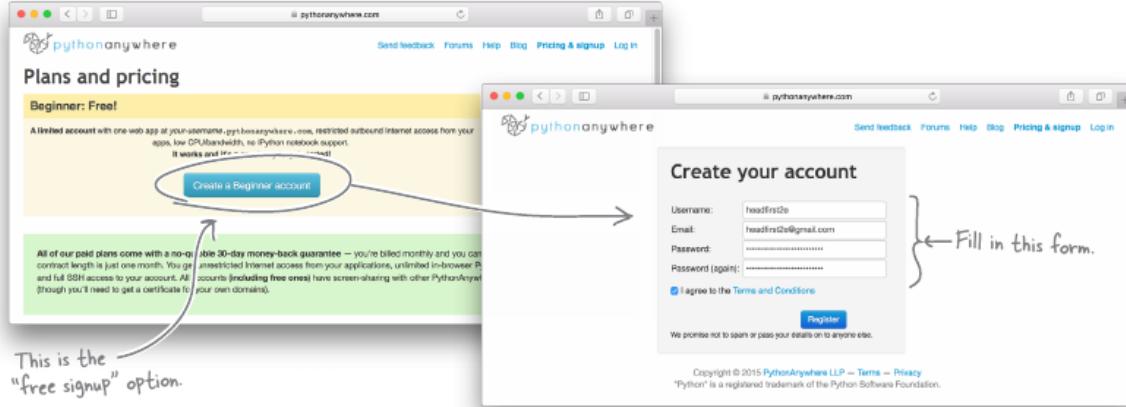
## **Step 1: Sign Up for PythonAnywhere**

### **Paso 1: Regístrate en PythonAnywhere**

Este paso no podría ser más fácil. Navega por encima de [pythonanywhere.com](http://pythonanywhere.com), luego haz clic en el enlace Precios y registro:

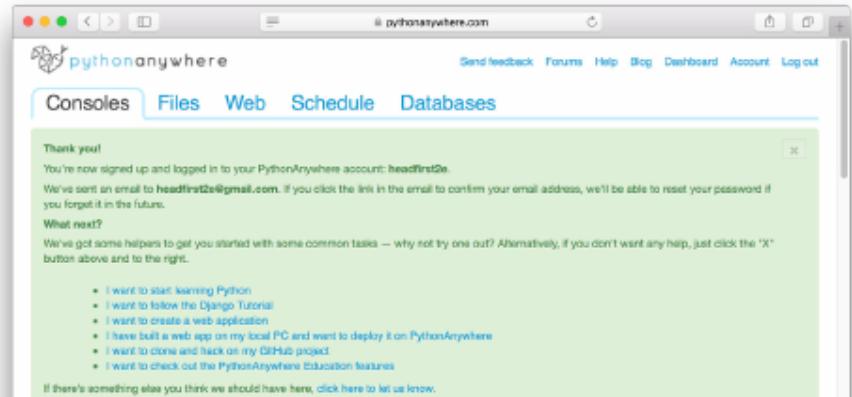


Haga clic en el botón grande y azul para crear una cuenta para principiantes y luego rellene los detalles en el formulario de registro:



Si todo va bien, aparece el panel de control de PythonAnywhere. Nota: ambos están registrados y firmados en este punto:

The PythonAnywhere dashboard. Note the five tabs available to you.



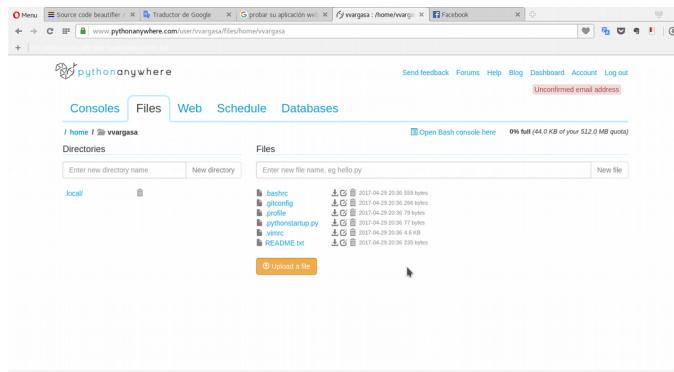
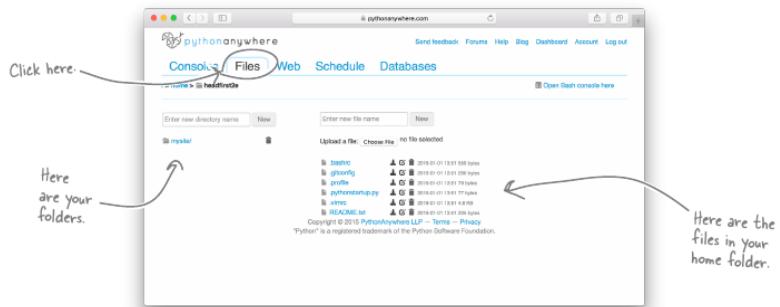
El panel de PythonAnywhere. Tenga en cuenta las cinco pestañas disponibles para usted.

A screenshot of a web browser window showing the PythonAnywhere dashboard. The title bar says "pythonanywhere". The main navigation menu has tabs for "Consoles", "Files", "Web", "Schedule", and "Databases", with "Consoles" being the active tab. A modal dialog box is open, titled "Thank you!", containing text about account creation and a list of five task helpers. At the bottom of the dialog is a link "If there's something else you think we should have here, click here to let us know." The status bar at the bottom right shows "CPU Usage 0% used: 0.00s of 100s" and "Resets in 23 hours, 59 minutes (more info)".

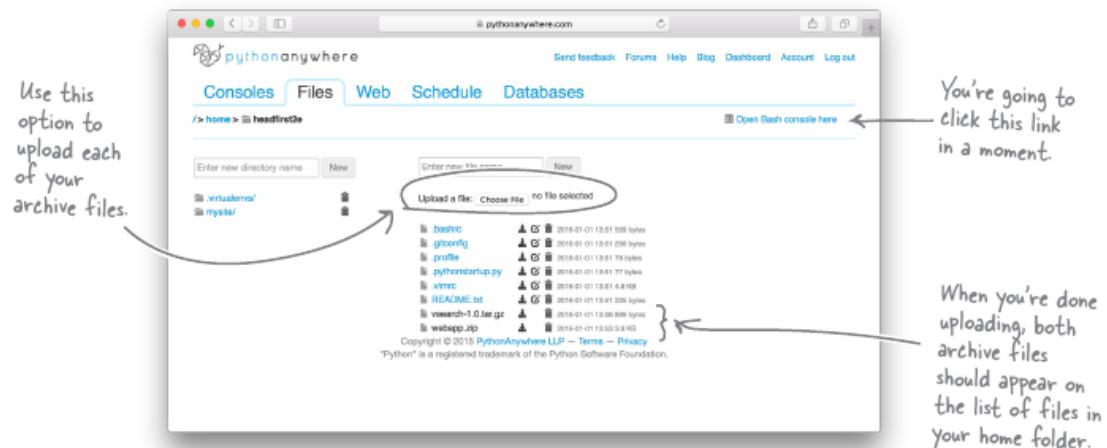
## **Step 2: Upload Your Files to the Cloud**

### **Paso 2: Cargue sus archivos en la nube**

Haga clic en la pestaña Archivos para ver las carpetas y archivos disponibles:



Utilice la opción Cargar un archivo para localizar y cargar los dos archivos del paso 0:



Traduciendo:

1. Utilice esta opción para cargar cada uno de sus archivos de archivo.
2. Vas a hacer clic en este enlace en un momento.
3. Cuando hayas terminado de subir, los dos archivos de archivo deben aparecer en la lista de archivos de tu carpeta de inicio.

Ya está listo para extraer e instalar estos dos archivos de archivo cargados y lo hará durante el Paso 3. Para prepararse, haga clic en el enlace Abrir una consola bash aquí en la parte superior derecha de la página anterior. Esto abre una ventana de terminal en la ventana del navegador (en PythonAnywhere).

### **Step 3: Extract and Install Your Code**

#### **Paso 3: extraiga e instale su código**

Al hacer clic en el vínculo Abrir una consola bash aquí, PythonAnywhere responde reemplazando el panel de Archivos con una consola de Linux basada en navegador (símbolo del sistema). Vas a emitir algunos comandos para extraer e instalar el módulo vsearch así como el código de tu webapp dentro de esta consola. Comience por instalar vsearch en Python como un "módulo privado" (es decir, sólo para su uso) usando este comando (asegúrese de usar vsearch-1.0.zip si está en Windows):

The screenshot shows a PythonAnywhere bash console window. The command `python3 -m pip install vsearch-1.0.tar.gz --user` is being run. The output shows the process of extracting the tarball, building wheels, and installing the module successfully. A note on the right explains that the `--user` option ensures the module is installed for individual use only, as PythonAnywhere does not allow global installations.

```
python3 -m pip install vsearch-1.0.tar.gz --user
Run the command.
Success!
"--user" ensures the "vsearch" module is installed for your use only. PythonAnywhere does not allow you to install a module for everyone's use (just your own).
pythonanywhere.com
Bash console 2149185
13:59 ~ $ ls
README.txt mysite vsearch-1.0.tar.gz webapp.zip
14:00 ~ $ python3 -m pip install vsearch-1.0.tar.gz --user
Processing ./vsearch-1.0.tar.gz
Building wheels for collected packages: vsearch
  Running setup.py bdist_wheel for vsearch
  Stored in directory: /home/headfirst2e/.cache/pip/wheels/eb/fc/ad/734c9fb2d6fcf07f96f045b42e63d536eb2a1f47
14:00 ~ $ Successfully built vsearch
Installing collected packages: vsearch
Successfully installed vsearch
14:00 ~ $
```

traducimos:

1. Ejecute el comando.
2. ¡Éxito!
3. "--user" asegura que el módulo "vsearch" esté instalado para su uso solamente. PythonAnywhere no le permite instalar un módulo para el uso de todos (solo el suyo).

Con el módulo vsearch instalado correctamente, es hora de dirigir su atención al código de su webapp, que debe instalarse en la carpeta `mysite` (que ya existe en su carpeta de inicio de PythonAnywhere). Para ello, debe emitir dos comandos:

The screenshot shows a PythonAnywhere Bash console window titled "Bash console 2149185". The command entered was "unzip webapp.zip" followed by "mv webapp/\* mysite". The output of the command shows the extraction of files from "webapp.zip" into a directory named "mysite". A callout box on the left says "Unpack your webapp's code..." and points to the "unzip" command. Another callout box on the right says "...then move the code into the 'mysite' folder." and points to the "mv" command. A bracket on the left also indicates the extracted files.

```
Bash console 2149185
Building wheels for collected packages: vsearch
  Running setup.py bdist_wheel for vsearch
    Stored in directory: /Home/headfirst2e/.cache/pip/wheels/eb/fc/ad/734c9fb2d6fcf97f96f045b42e63d536eb2a1f47
191735fce
Successfully built vsearch
Installing collected packages: vsearch
Successfully installed vsearch
14:00 ~ $ unzip webapp.zip
Archive: webapp.zip
  creating: webapp/
    inflating: webapp/.DS_Store
  creating: __MACOSX/
  creating: __MACOSX/webapp/
    inflating: __MACOSX/webapp/.DS_Store
  creating: webapp/static/
    inflating: webapp/static/hf.css
  creating: webapp/templates/
    inflating: webapp/templates/base.html
    inflating: webapp/templates/entry.html
    inflating: webapp/templates/results.html
    inflating: webapp/vsearch4web.py
14:08 ~ $ mv webapp/* mysite
14:08 ~ $
```

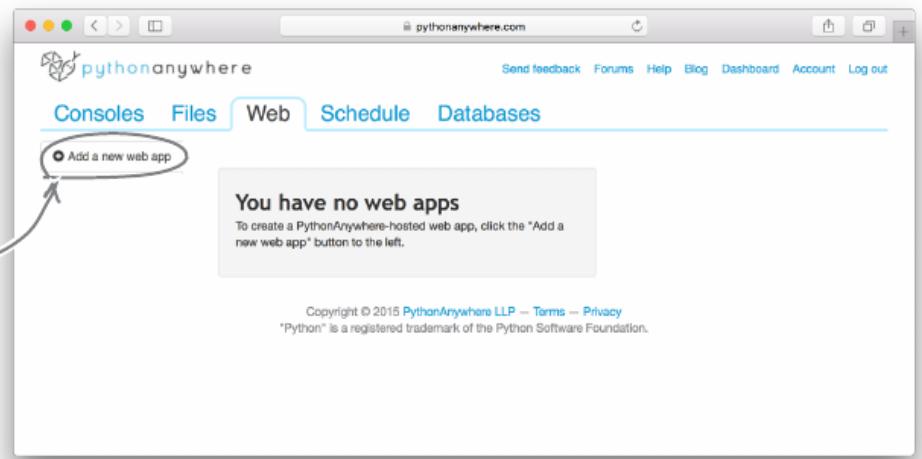
Traducimos:

1. Desempaque el código de su webapp ...
2. Debería ver mensajes similares a estos.
3. ... luego mueva el código a la carpeta "mysite".

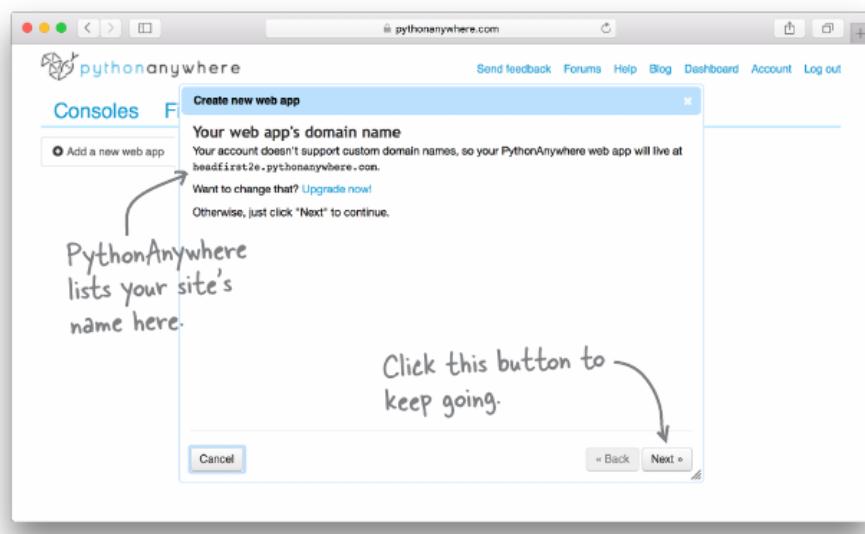
#### **Step 4: Create a Starter Webapp, 1 of 2**

#### **Paso 4: Crear una aplicación web de inicio, 1 de 2**

Con el paso 3 hecho, regrese al panel de PythonAnywhere y seleccione la pestaña Web, donde PythonAnywhere le invita a crear una nueva webapp de inicio. Usted hará esto, después intercambie hacia fuera el código del webapp del arrancador para sus los propios. Tenga en cuenta que cada cuenta principiante obtiene una aplicación web de forma gratuita; Si desea más, tendrá que actualizar a una cuenta de pago. Afortunadamente, por ahora, solo necesitas una, así que vamos a seguir haciendo clic en Añadir una nueva aplicación web:



Como usted está utilizando una cuenta gratuita, su aplicación web se ejecutará en el nombre del sitio que se muestra en la siguiente pantalla. Haga clic en el botón Siguiente para continuar con el nombre del sitio sugerido por PythonAnywhere:

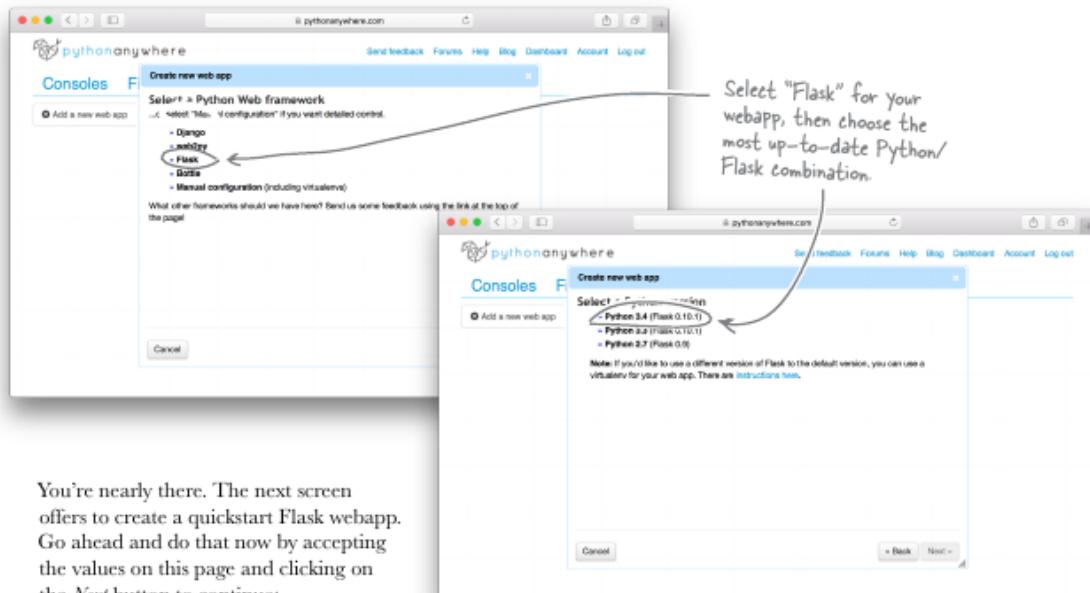


Haga clic en Siguiente para continuar con este paso.

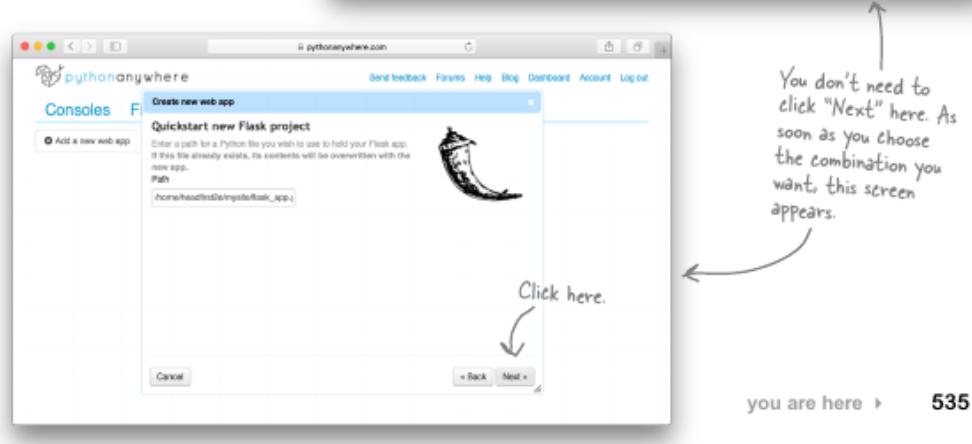
#### **Step 4: Create a Starter Webapp, 2 of 2**

#### **Paso 4: Crear un Starter Webapp, 2 de 2**

PythonAnywhere admite más de un framework web Python, por lo que la siguiente pantalla le ofrece una opción entre los muchos sistemas compatibles. Seleccione Flask y, a continuación, seleccione la versión de Flask y Python que desee implementar. A partir de esta escritura, Python 3.4 y Flask 0.10.1 son las versiones más actualizadas soportadas por PythonAnywhere, así que vaya con esa combinación a menos que se ofrezca una combinación más reciente (en cuyo caso, escoja la más nueva):



You're nearly there. The next screen offers to create a quickstart Flask webapp. Go ahead and do that now by accepting the values on this page and clicking on the *Next* button to continue:



you are here ➤ **535**

## Traduciendo:

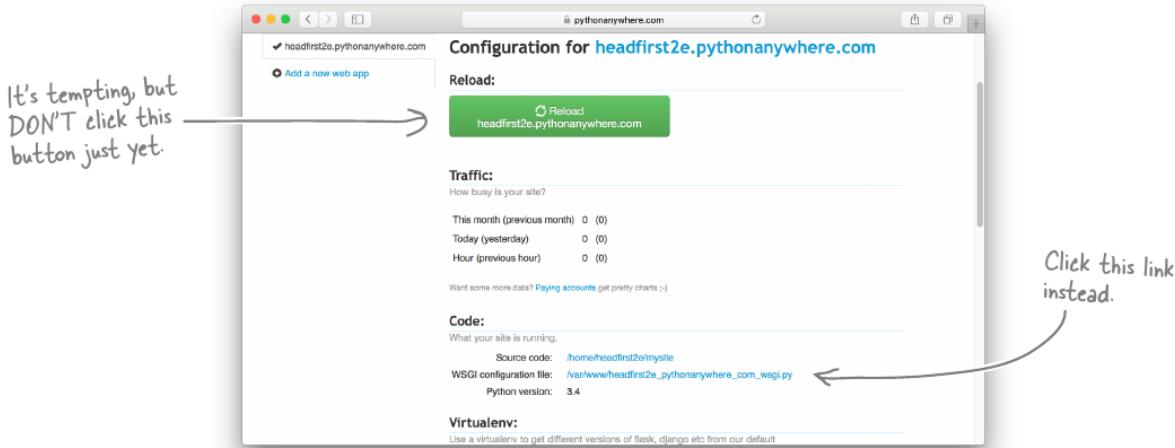
Traduciendo:

1. Ya casi estás allí. La siguiente pantalla ofrece crear una aplicación web Flask de inicio rápido. Vaya y haga eso ahora aceptando los valores en esta página y haciendo clic en el botón Siguiente para continuar:
2. Seleccione "Flask" para su webapp, luego elija la combinación más actualizada de Python / Flask.
3. No es necesario hacer clic en "Siguiente" aquí. Tan pronto como elija la combinación que desea, aparece esta pantalla.

## **Step 5: Configure Your Webapp**

### **Paso 5: Configure su Webapp**

Con el Paso 4 completo, se le presenta el panel de control Web. No se sienta tentado a hacer clic en ese botón grande y verde todavía-todavía no le ha dicho a PythonAnywhere sobre su código, así que mantenga en funcionamiento cualquier cosa por ahora. En su lugar, haga clic en el enlace largo a la derecha de la etiqueta del archivo de configuración WSGI:



traduciendo:

1. Es tentador, pero no haga clic en este botón todavía.
2. Haga clic en este enlace en su lugar.

Al hacer clic en ese vínculo largo, se carga el archivo de configuración de Flask webapp recién creado en el editor de texto basado en Web de PythonAnywhere. Al final del capítulo 5, le hemos dicho que PythonAnywhere importa su código de webapp antes de invocar **app.run()** para usted. Este es el archivo que admite ese comportamiento. Sin embargo, es necesario indicarle que haga referencia a su código, no al código de la aplicación de inicio, por lo que necesita editar la última línea de este archivo (como se muestra a continuación) y, a continuación, haga clic en Guardar:

```

1 # This file contains the WSGI configuration required to serve up your
2 # web application at http://<your-username>.pythonanywhere.com/
3 # It works by setting the variable 'application' to a WSGI handler of some
4 # description.
5 #
6 # The below has been auto-generated for your Flask project
7
8 import sys
9
10 # add your project directory to the sys.path
11 project_home = u'/home/headfirst2e/mysite'
12 if project_home not in sys.path:
13     sys.path = [project_home] + sys.path
14
15 # import flask app but need to call it "application" for WSGI to work
16 from flask_app import app as application
17

```

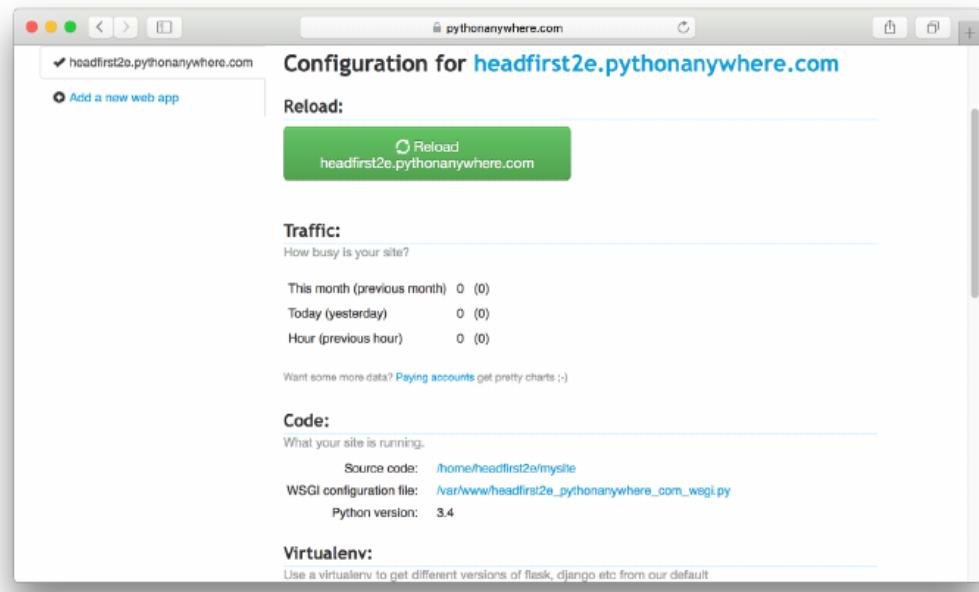
traduciendo:

Cambie la última línea de este archivo para hacer referencia a su módulo "vsearch4web".

## **Step 6: Take Your Cloud-Based Webapp for a Spin!**

### **Paso 6: Lleve a su Webapp basado en la nube para una vuelta!**

Asegúrese de guardar el archivo de configuración cambiado y, a continuación, vuelva a la pestaña Web del tablero de instrumentos. Ahora es el momento de hacer clic en ese botón grande, tentador, verde. ¡Ve a por ello!



Después de un breve momento, tu aplicación web aparece en tu navegador y funciona exactamente igual que cuando lo ejecutaste localmente, pero ahora también cualquiera que tenga una conexión a Internet y un navegador web puede usarlo también:

We're looking good for input and output in the cloud.

Anyone can use this web address to interact with your webapp.

Traduciendo:

1. Estamos buscando buenas entradas y salidas en la nube.
2. Cualquiera puede utilizar esta dirección web para interactuar con su aplicación web.

Y con eso, ya está. La aplicación web que desarrolló en el Capítulo 5 se ha implementado en la nube de PythonAnywhere (en menos de 10 minutos). Hay mucho más que PythonAnywhere que lo que se muestra en este breve apéndice, así que siéntete libre de explorar y experimentar. En algún momento, recuerde devolver el panel de PythonAnywhere y cerrar la sesión. Tenga en cuenta que, a pesar de su cierre de sesión, su webapp sigue funcionando en la nube hasta que le diga lo contrario. Eso es muy bueno, ¿no?

## Chapter 5's Code

Esto es "hello\_flask.py", nuestra primera webapp basada en Flask (una de las tecnologías de framework micro-web de Python).

```
from flask import Flask
from vsearch import search4letters

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'

@app.route('/search4')
def do_search() -> str:
    return str(search4letters('life, the universe, and everything', 'eiru,!'))

app.run()
```

Se trata de "vsearch4web.py". Esta aplicación web expuso la funcionalidad proporcionada por nuestra función "search4letters" a la World Wide Web. Además de Flask, este código explotó el motor de plantillas Jinja2.

```
from flask import Flask, render_template, request
from vsearch import search4letters

app = Flask(__name__)

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to... web!')

if __name__ == '__main__':
    app.run(debug=True)
```

Esto es "dunder.py", que nos ayudó a entender el muy útil "dunder nombre dunder principal" mecanismo.

```
print('We start off in:', __name__)
if __name__ == '__main__':
    print('And end up in:', __name__)
```

## 6 storing and manipulating data



# Where to Put Your Data



**Tarde o temprano, usted necesitará almacenar con seguridad sus datos en alguna parte.**

Y cuando se trata de almacenar datos, Python lo tiene cubierto. En este capítulo, aprenderá a almacenar y recuperar datos de archivos de texto, que -como van a ser los mecanismos de almacenamiento- se sienten un poco simplistas, pero sin embargo se utilizan en muchas áreas problemáticas. Además de almacenar y recuperar sus datos de los archivos, también aprenderá algunos trucos del comercio cuando se trata de manipular datos. Estamos guardando las "cosas serias" (almacenar datos en una base de datos) hasta el próximo capítulo, pero hay mucho que nos mantiene ocupados por ahora al trabajar con archivos.

### **Haciendo algo con los datos de tu Webapp**

Por el momento, su webapp (desarrollada en el Capítulo 5) acepta la entrada de cualquier navegador web (en forma de una frase y algunas letras), realiza una llamada a search4letters y luego devuelve los resultados al navegador web en espera. Una vez hecho esto, su webapp descarta cualquier dato que tenga.

Hay un montón de preguntas que podríamos hacer de los datos de nuestra aplicación web. Por ejemplo: ¿Cuántas solicitudes se han respondido? ¿Cuál es la lista de cartas más común? ¿De qué direcciones IP provienen las peticiones? ¿Qué navegador se utiliza más? y así sucesivamente y así sucesivamente.

Para comenzar a contestar estas (y otras) preguntas, necesitamos guardar los datos de la aplicación web en lugar de simplemente tirarla. La sugerencia anterior tiene mucho sentido: vamos a registrar datos sobre cada solicitud web, entonces-una vez que tengamos el mecanismo de registro en su lugar- responderemos cualquier pregunta que tengamos.

## **Python Supports Open, Process, Close**

### **Soporta Python Open, Process, Close**

No importa el lenguaje de programación, la forma más fácil de almacenar datos es guardarlo en un archivo de texto. Por lo tanto, Python viene con soporte incorporado para abrir, procesar, cerrar. Esta técnica común le permite abrir un archivo, procesar sus datos de alguna manera (leer, escribir y / o agregar datos), y luego cerrar el archivo cuando haya terminado (lo que guarda los cambios).

A continuación se explica cómo utilizar la técnica open, process, close de Python para abrir un archivo, procesarlo añadiendo algunas cadenas cortas a él y, a continuación, cerrar el archivo. Como sólo estamos experimentando por ahora, vamos a ejecutar nuestro código en el shell de Python >>>.

Empezamos llamando open en un archivo llamado **todos.txt**, usando el modo append, ya que nuestro plan es agregar datos a este archivo. Si la llamada para abrir tiene éxito, el intérprete devuelve un objeto (conocido como flujo de archivo) que es un alias para el archivo real. El objeto se asigna a una variable y se le da el nombre todos (aunque podría usar el nombre que desee aquí):

The diagram illustrates the Python code: `>>> todos = open('todos.txt', 'a')`. Handwritten annotations explain the code: 'Open a file...' points to the `open` call, '...which has this filename...' points to 'todos.txt', and '...and open the file in "append-mode"' points to 'a'. A large arrow points from the code up to the annotation 'If all is OK, "open" returns a file stream, which we've assigned to this variable.'.

La variable todos le permite referirse a su archivo en su código (otros lenguajes de programación se refieren a esto como un identificador de archivo). Ahora que el archivo está abierto, vamos a escribir con la impresión print. Observe cómo, a continuación, la impresión toma un argumento adicional (**archivo**), que identifica el flujo de archivo a escribir. Tenemos tres cosas que recordar que hacer (es interminable, en realidad), por lo que llamar a imprimir tres veces:

We print a message... → ...to the file stream.

```
>>> print('Put out the trash.', file=todos)
>>> print('Feed the cat.', file=todos)
>>> print('Prepare tax return.', file=todos)
```

traducimos:

1. Ponga la basura.
2. Alimenta al gato.
3. Preparar la declaración de impuestos

Como no tenemos nada más que añadir a nuestra lista de tareas pendientes, cerremos el archivo llamando al método close, que el intérprete pone a disposición de cada flujo de archivos:

→ We're done, so let's tidy up after ourselves by closing the file stream.

```
>>> todos.close()
```

traducir:

Ya hemos terminado, así que vamos a arreglar después de nosotros mismos, cerrando el flujo de archivos.

Si olvida llamar **close**, podría perder datos. Recordar llamar siempre **close** es importante.

## ***Reading Data from an Existing File***

### ***Lectura de datos de un archivo existente***

"Lectura" es el modo predeterminado de la función "open".

Ahora que ha agregado algunas líneas de datos al archivo todos.txt, veamos el código abierto, de proceso, de cierre necesario para leer los datos guardados del archivo y mostrarlos en la pantalla.

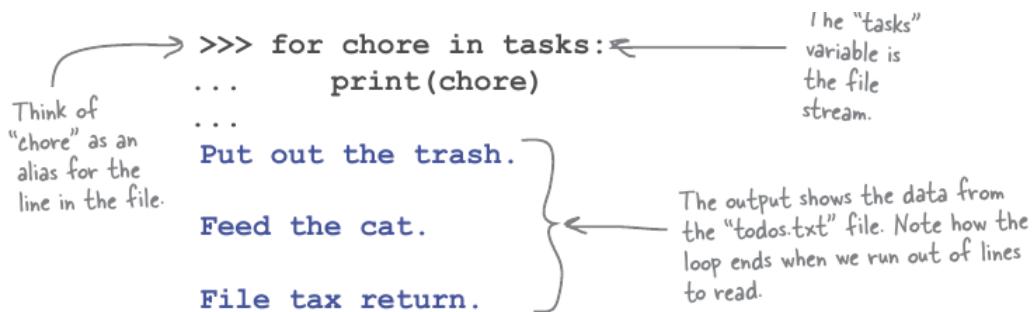
En lugar de abrir el archivo en el modo de anexar o append, esta vez sólo le interesa leer el archivo. Como la lectura es el modo predeterminado de open, no es necesario proporcionar un argumento de modo; El nombre del archivo es todo lo que necesita aquí. No usamos todos como alias para el archivo en este código; En su lugar, nos referiremos al archivo abierto por las **tareas** o **tasks** de nombre (como antes, puede usar el nombre de variable que desee aquí):

Open a file... → ...which has this filename.

```
>>> tasks = open('todos.txt')
```

If all is OK, "open" returns a file stream, which we've assigned to this variable.

Ahora vamos a usar las tareas o tasks con un bucle for para leer cada línea individual del archivo. Cuando hacemos esto, la variable de iteración (tarea) del bucle for se asigna a la línea actual de datos como leída desde el archivo. Cada iteración asigna una línea de datos a la tarea. Cuando se utiliza un flujo de archivos con bucle de Python para, el intérprete es lo suficientemente inteligente como para leer una línea de datos del archivo cada vez que el bucle itera. También es lo suficientemente inteligente como para terminar el bucle cuando no hay más datos para leer:



Traduciendo:

1. Piense en "chore" o "tarea" como un alias para la línea en el archivo.
2. La salida muestra los datos del archivo "todos.txt". Tenga en cuenta cómo termina el bucle cuando nos quedamos sin líneas para leer.
3. La variable "tasks" es el flujo de archivos.

```

>>> os.chdir("/home/anton/.virtualenvs/HeadFirst/")
>>> os.getcwd()
'/home/anton/.virtualenvs/HeadFirst'

>>> todos = open('todos.txt', 'a')
>>> print('Put out the trash.', file=todos)
>>> print('Fedd de cat.', file=todos)
>>> print('Prepare tax return.', file= todos)
>>> todos.close()
>>> tasks = open('todos.txt')
>>> for chore in tasks:
print(chore)

Put out the trash.

Fedd de cat.

Prepare tax return.

>>> tasks.close()

```

Como simplemente está leyendo desde un archivo ya escrito, el cierre de llamada es menos crítico que cuando está escribiendo datos. Pero siempre es una buena idea cerrar un archivo cuando ya no sea necesario, así que llame al método close cuando haya terminado:

```
>>> tasks.close()
```



Ya hemos terminado, así que vamos a arreglar después de nosotros mismos, cerrando el flujo de archivos.

## there are no Dumb Questions

**P:** ¿Cuál es el trato con las nuevas líneas adicionales en la salida? Los datos del archivo tienen tres líneas de largo, pero el bucle for produjo seis líneas de salida en mi pantalla. ¿Lo que da?

**R:** Sí, la salida del bucle for parece extraña, ¿no? Para entender lo que está sucediendo, considere que la función de impresión print agrega una nueva línea a todo lo que se muestra en la pantalla como su comportamiento predeterminado. Cuando se combina esto con el hecho de que cada línea del archivo termina en un carácter de nueva línea (y se lee la línea nueva como parte de la línea), acaba imprimiendo dos líneas nuevas: la del archivo junto con la de impresión . Para indicar a print que no incluya la segunda nueva línea, cambie la **print(chore)** para imprimir **print(chore, end = '')**. Esto tiene el efecto de suprimir el comportamiento de nueva línea de adición de la impresión, por lo que las nuevas líneas extra no aparecen en pantalla.

**P:** ¿Qué otros modos están disponibles para mí cuando estoy trabajando con datos en archivos?

**R:** Hay algunos, que hemos resumido en el siguiente cuadro de bits de Geek. (Esa es una gran pregunta, BTW.)

El primer argumento a abrir open es el nombre del archivo a procesar. El segundo argumento es opcional. Los modos incluyen "lectura", "escritura" y "adición". Aquí están los valores de modo más comunes, donde cada uno (excepto para ' r ') crea un nuevo archivo vacío si el archivo nombrado en el primer argumento ya no existe:

**'r'** Abre un archivo para leer. Este es el modo predeterminado y, como tal, es opcional. Cuando no se proporciona un segundo argumento, se supone '**r**'. También se supone que ya existe el archivo que se está leyendo.

**'w'** Abre un archivo para escribir. Si el archivo ya contiene datos, vacía el archivo de sus datos antes de continuar.

**'a'** Abre un archivo para agregar. Preserva el contenido del archivo, agregando nuevos datos al final del archivo (compare este comportamiento con '**w**').

**'x'** Abre un nuevo archivo para escribir. Error si el archivo ya existe (compare este comportamiento con '**w**' y '**a**').

De forma predeterminada, los archivos se abren en modo de texto, donde se supone que el archivo contiene líneas de datos de texto (por ejemplo, ASCII o UTF-8). Si está trabajando con datos no textuales (por ejemplo, un archivo de imagen o un MP3), puede especificar el modo binario añadiendo "b" a cualquiera de los modos (por ejemplo, 'wb' significa "escribir en un dato binario"). Si incluye "+" como parte del segundo argumento, el archivo se abrirá para leer y escribir (por ejemplo, 'x + b' significa "leer y escribir en un nuevo archivo binario"). Consulte los documentos de Python para obtener más detalles sobre open (incluyendo información sobre sus otros argumentos opcionales).

He mirado un montón de proyectos de Python en GitHub, y la mayoría de ellos utilizan una declaración **"with"** al abrir archivos. Cuál es el problema?

### ***La declaración with es más conveniente.***

Aunque el uso de la función **open** junto con el método de **close**(con un poco de procesamiento en el medio) funciona bien, la mayoría de los programadores de Python evitan abrir, procesar, cerrar a favor de la declaración **with**. Tomemos un tiempo para averiguar por qué.

### ***A Better Open, Process, Close: "with"***

### ***Un mejor Open, Process, Close: "with"***

Antes de describir por qué **with** es tan popular, echemos un vistazo a algún código que utiliza **with**. Aquí está el código que escribimos (hace dos páginas) para leer y mostrar el contenido actual de nuestro archivo **todos.txt**. Tenga en cuenta que hemos ajustado la llamada de función de impresión print para suprimir la extra nueva línea en la salida:

```
tasks = open('todos.txt')
for chore in tasks:
    print(chore, end=' ')
tasks.close()
```

The diagram shows handwritten annotations explaining the code. An arrow points from the first line to the text: "Open the file, assigning the file stream to a variable.". Another arrow points from the closing brace of the for loop to the text: "Perform some processing.". A final arrow points from the close() method call to the text: "Close the file."

traducción:

1. Abra el archivo, asignando el flujo de archivos a una variable.

Vamos a reescribir este código para usar una declaración **with**. Estas tres líneas de código siguientes utilizan para realizar exactamente el mismo procesamiento que las cuatro líneas de código (arriba):

```

    Abre el archivo.           Asigne el flujo de archivos
    with open('todos.txt') as tasks:   a una variable.

    for chore in tasks:
        print(chore, end='')

    Realizar algún proceso (que es
    el mismo coseno de como
    antes).

```

```
>>> with open('todos.txt') as tasks:
    for chore in tasks:
        print(chore, end='')
```

```

Put out the trash.
Fedd de cat.
Prepare tax return.
Put out the trash.
Fedd de cat.
Fedd de cat.
>>>
```

¿Se ha perdido algo? La llamada a `close` no aparece. La sentencia `with` es lo suficientemente inteligente como para recordar llamar `close` en su nombre cada vez que su conjunto de código termina.

Esto es en realidad mucho más útil de lo que inicialmente suena, ya que muchos programadores a menudo se olvidan de llamar `close` cuando se ha terminado de procesar un archivo. Esto no es tan importante cuando todo lo que estás haciendo es leer desde un archivo, pero cuando estás `escribiendo` en un archivo, olvidar llamar a `close` puede causar pérdida de datos o corrupción de datos. Al relevarlo de la necesidad de recordar llamar siempre `close`, la sentencia `with` te permite concentrarse en lo que realmente estás haciendo con los datos del archivo abierto.

**Python soporta "open, process, close". Pero la mayoría de los programadores de Python prefieren usar la sentencia "with".**

### ***El contexto gestionado por sentencia "with"***

La sentencia `with` se ajusta a una convención de codificación integrada en Python llamada el protocolo de gestión de contexto. Estamos aplazando una discusión detallada de este protocolo hasta más adelante en este libro. Por ahora, todo lo que tiene que preocuparse es el hecho de que cuando se utiliza `with` cuando se trabaja con archivos, puede olvidarse de llamar `close`. La sentencia `with` es la administración del contexto dentro del cual se ejecuta su suite, y cuando se usa `with` y `open` juntos, el intérprete limpia después de usted, llamando `close` como y cuando sea necesario.

## Exercise

Vamos a poner lo que ahora sabes acerca de trabajar con archivos para usar. Aquí está el código actual para su aplicación web. Dale otra lectura antes de que te digamos lo que tienes que hacer:

```
from flask import Flask, render_template, request
from vsearch import search4letters

app = Flask(__name__)

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

if __name__ == '__main__':
    app.run(debug=True)
```

This is the "vsearch4web.py" code from Chapter 5.



Su trabajo es escribir una nueva función, llamada **log\_request**, que toma dos argumentos: **req** y **res**. Cuando se invoca, el argumento req se asigna al objeto de solicitud o request de Flask actual, mientras que el argumento res se asigna a los resultados o response de llamar a la función log\_request. La suite de la función log\_request debe agregar el valor de req y res (como una línea) a un archivo llamado vsearch. Iniciar sesión. Te hemos empezado proporcionando la línea def de la función. Usted debe proporcionar el código que falta (sugerencia: use with):

Write this function's suite here.



```
..... def log_request(req: 'flask_request', res: str) -> None:
```



## Exercise Solution

Su trabajo era escribir una nueva función, llamada **log\_request**, que toma dos argumentos: `req` y `res`. Cuando se invoca, el argumento `req` se asigna al objeto de solicitud o `request` de Flask actual, mientras que el argumento `res` se asigna a su respuesta o `response` de llamar a `search4letters`. El conjunto de la función **log\_request** debe agregar el valor de `req` y `res` (como una línea) a un archivo llamado `vsearch.log`. Te pusimos en marcha: debías proporcionar el código que faltaba:

```
def log_request(req: 'flask_request', res: str) -> None:
    with open('vsearch.log', 'a') as log:
        print(req, res, file=log)
```

This annotation may have thrown you a little. Recall that function annotations are meant to be read by other programmers. They are documentation, not executable code: the Python interpreter always ignores them, so you can use any annotation descriptor you like.

Use "with" to open "vsearch.log" in append mode.

Call the "print" BIF to write the values of "req" and "res" to the opened file.

Note the file stream is called "log" in this code.

This annotation uses Python's "None" value to indicate this function has no return value.

traduciendo:

1. Utilice "with" para abrir "vsearch.log" en el modo de agregación.
2. Llame al BIF "print" para escribir los valores de "req" y "res" en el archivo abierto.
3. Tenga en cuenta que el flujo de archivos se denomina "registro" o "log" en este código.
4. Esta anotación utiliza el valor "None" de Python para indicar que esta función no tiene valor devuelto.
5. Esta anotación puede haberte arrojado un poco. Recuerde que las anotaciones de funciones están destinadas a ser leídas por otros programadores. Son documentación, no código ejecutable: el intérprete de Python siempre los ignora, por lo que puede utilizar cualquier descriptor de anotación que desee.

Nos ubicamos en el entorno virtual:

```
[anton@anton-pc ~]$ workon HeadFirst
blog
flaskapp
flasky
miproyecto1
[anton@anton-pc ~]$ workon HeadFirst
(HeadFirst)[anton@anton-pc~]$ cd/home/anton/.virtualenvs/HeadFirst/First_Codes/ch06/
```

```
(HeadFirst) [anton@anton-pc ch06]$ ls
webapp
(HeadFirst) [anton@anton-pc ch06]$ cd webapp
(HeadFirst) [anton@anton-pc webapp]$
```

## Llamamos a **vsearch4web.py**

```
from flask import Flask, render_template, request, escape
from vsearch import search4letters


app = Flask(__name__)

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""
    with open('vsearch.log', 'a') as log:
        print(req.form, req.remote_addr, req.user_agent, res, file=log, sep='|')

@app.route('/search4', methods=['POST'])

def do_search() -> 'html':
    """Extract the posted data; perform the search; return results."""
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    log_request(request, results)
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

@app.route('/')
@app.route('/entry')

def entry_page() -> 'html':
    """Display this webapp's HTML form."""
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')
```

```

@app.route('/viewlog')

def view_the_log() -> 'html':
    """Display the contents of the log file as a HTML table."""

    contents = []

    with open('vsearch.log') as log:
        for line in log:
            contents.append([])

            for item in line.split('|'):
                contents[-1].append(escape(item))

    titles = ('Form Data', 'Remote_addr', 'User_agent', 'Results')

    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,)

if __name__ == '__main__':
    app.run(debug=True)

```

## ***Invoking the logging function***

### ***Invocación de la función de registro***

Ahora que existe la función `log_request`, ¿cuándo la invocamos?

Bueno, para empezar, vamos a agregar el código `log_request` al archivo `vsearch4web.py`. Puede ponerlo en cualquier parte de este archivo, pero lo insertamos directamente encima de la función `do_search` y su decorador `@app.route` asociado. Hemos hecho esto porque vamos a invocarlo desde dentro de la función `do_search`, y ponerlo por encima de la función de llamada parece una buena idea.

Tenemos que estar seguros de llamar a `log_request` antes de que termine la función `do_search`, pero después de que los resultados hayan sido devueltos desde la llamada a `search4letters`. Aquí hay un fragmento del código de `do_search` que muestra la llamada insertada:

```

@app.route('/search4', methods=['POST'])

def do_search() -> 'html':
    """Extract the posted data; perform the search; return results."""

```

```

phrase = request.form['phrase']

letters = request.form['letters']

title = 'Here are your results:'

results = str(search4letters(phrase, letters))

log_request(request, results)

return render_template('results.html',
                      the_title=title,
                      the_phrase=phrase,
                      the_letters=letters,
                      the_results=results,)

...
phrase = request.form['phrase']
letters = request.form['letters']
title = 'Here are your results:'
results = str(search4letters(phrase, letters))
log_request(request, results)
return render_template('results.html',

```

*Call the "log\_request" function here.*

## A Quick Review - Una revisión rápida

Antes de tomar esta última versión de vsearch4web.py para una vuelta, vamos a comprobar que su código es el mismo que el nuestro. Aquí está el archivo completo, con las últimas adiciones destacadas:

```

from flask import Flask, render_template, request
from vsearch import search4letters

app = Flask(__name__)

def log_request(req: 'flask_request', res: str) -> None:
    with open('vsearch.log', 'a') as log:
        print(req, res, file=log)

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    log_request(request, results)
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

if __name__ == '__main__':
    app.run(debug=True)

```

Here are the latest additions, which arrange to log each web request to a file called "vsearch.log".

Es posible que hayas notado que ninguna de las funciones de nuestra webapp contiene comentarios. Esta es una omisión deliberada de nuestra parte (ya que sólo hay mucho espacio en estas páginas, y algo tenía que dar). Tenga en cuenta que cualquier código que descargue del sitio web de soporte de este libro siempre incluye comentarios.

Traduciendo:

Estas son las últimas adiciones que organizan el registro de cada solicitud web en un archivo denominado "vsearch.log".

## Tome su webapp para una vuelta ...

Inicie esta versión de su aplicación web (si es necesario) en un símbolo del sistema. En Windows, utilice este comando:

**C:\webapps> py -3 vsearch4web.py**

While on Linux or Mac OS X, use this command:

**\$ python3 vsearch4web.py**

Con su webapp en funcionamiento, vamos a registrar algunos datos a través del formulario HTML.

**Test Drive -**

Utilice su navegador web para enviar datos a través del formulario HTML de la aplicación web. Si desea seguir con lo que estamos haciendo, envíe tres búsquedas utilizando los siguientes valores de frase y letras:

hitch-hiker with aeiou.

life, the universe, and everything with aeiou.

galaxy with xyz.

Antes de comenzar, tenga en cuenta que el archivo `vsearch.log` todavía no existe.

The first search results for "hitch-hiker" with "aeiou" show results {"'e', 'i'}

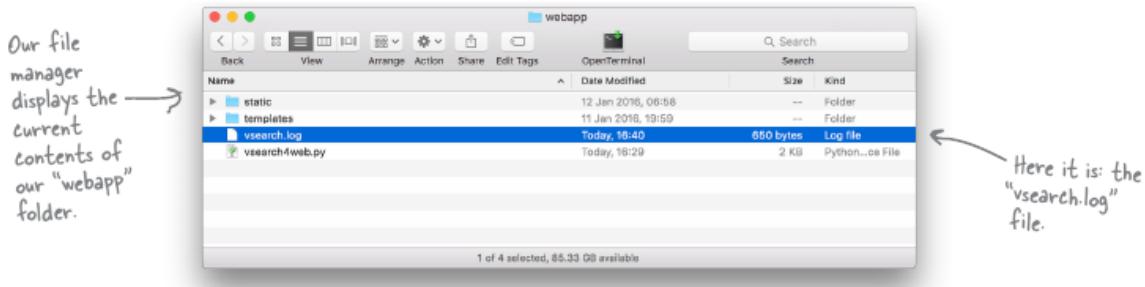
The second search results for "life, the universe, and everything" with "aeiou" show results {"'e', 'u', 'a', 'i'}

The third (and final) search results for "galaxy" with "xyz" show results {"'y', 'x'}

## **Data is logged (behind the scenes)**

### **Los datos se registran (detrás de las escenas)**

Cada vez que se utiliza el formulario HTML para enviar datos a la aplicación web, la función de petición de registro guarda los detalles de la solicitud web y escribe los resultados en el archivo de registro. Inmediatamente después de la primera búsqueda, el archivo `vsearch.log` se crea en la misma carpeta que el código de su aplicación web:



Traduciendo:

1. Nuestro administrador de archivos muestra el contenido actual de nuestra carpeta "webapp".
2. Aquí está: el archivo "vsearch.log".

Es tentador considerar el uso del editor de texto para ver el contenido del archivo `vsearch.log`. Pero, ¿dónde está la diversión en eso? Como se trata de una aplicación web, vamos a proporcionar acceso a los datos registrados a través de la propia aplicación web. De esta forma, nunca tendrá que alejarse de su navegador web al interactuar con los datos de su aplicación web. Vamos a crear una nueva URL, llamada `/viewlog`, que muestra el contenido del registro a petición.

## **View the Log Through Your Webapp**

### **Ver el registro a través de su Webapp**

Vas a añadir soporte para la URL `/viewlog` a tu aplicación web. Cuando su webapp recibe una solicitud para `/viewlog`, debe abrir el archivo `vsearch.log`, leer en todos sus datos y luego enviar los datos al navegador en espera.

La mayoría de lo que necesitas hacer ya lo sabes. Comience creando una nueva línea `@app.route` (estamos añadiendo este código cerca de la parte inferior de `vsearch4web.py`, justo encima de la línea principal `dunder dunder`):

```
@app.route('/viewlog')
```

We have  
a brand  
new URL.

traduciendo: Tenemos una nueva URL.

Una vez decidida la URL, a continuación escribiremos una función para ir con ella. Llamemos a nuestra nueva función `view_the_log`. Esta función no tomará argumentos y devolverá una cadena a su interlocutor; La cadena será la concatenación de todas las líneas de datos del archivo `vsearch.log`. Aquí está la línea def de la función:

```
def view_the_log() -> str:
```

Y tenemos una nueva función, que (de acuerdo con la anotación) devuelve una cadena.

Ahora escribir la suite de la función. Tienes que abrir el archivo para leerlo. Este es el modo predeterminado de la función `open`, por lo que sólo necesita el nombre del archivo como argumento para abrirlo. Vamos a manejar el contexto dentro del cual se ejecuta nuestro código de procesamiento de archivos usando una instrucción `with`:

```
with open('vsearch.log') as log:
```

Abra el archivo de registro para su lectura.

Dentro de la suite de la sentencia `with`, necesitamos leer todas las líneas del archivo. Su primer pensamiento podría ser el bucle a través del archivo, la lectura de cada línea a medida que avanza. Sin embargo, el intérprete proporciona un método de lectura `read` que, cuando se invoca, devuelve todo el contenido del archivo "de una sola vez". Aquí está la única línea de código que hace exactamente eso, creando una nueva cadena llamada `contents`:

```
contents = log.read()
```

Lea todo el archivo "de una vez" y asignarlo a una variable (que hemos denominado "contenido").

Con el archivo leído, la instrucción `with` termina (cerrar el archivo), y ahora está listo para enviar los datos de vuelta al navegador web en espera. Esto es sencillo:

```
return contents
```

Tome la lista de líneas en "contenido" y devuélvalas.

Con todo preparado, ahora tiene todo el código que necesita para responder a la solicitud `/viewlog`; se parece a esto:

Éste es todo el código que necesita para admitir la URL "/viewlog".

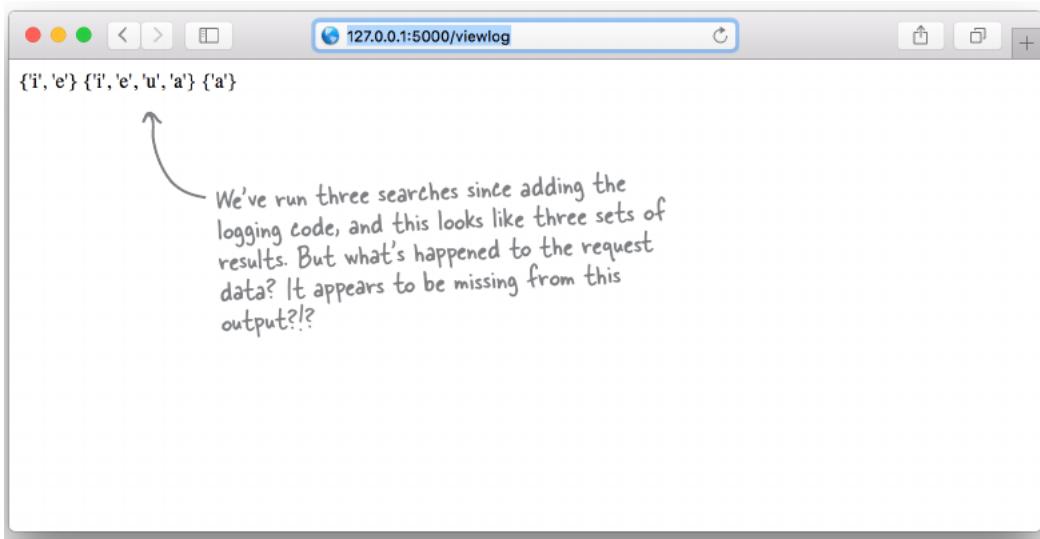
```
{ @app.route('/viewlog')
    def view_the_log() -> str:
        with open('vsearch.log') as log:
            contents = log.read()
        return contents
```



**Test Drive** -

Con el nuevo código agregado y guardado, tu aplicación web debería volver a cargarse automáticamente. Puede introducir algunas búsquedas nuevas, si lo desea, pero las que ha publicado hace unas pocas páginas ya están registradas. Las nuevas búsquedas que realice se anexarán al archivo de registro log. Utilicemos la URL `/viewlog` para echar un vistazo a lo que se ha guardado. Escriba `http://127.0.0.1:5000/viewlog` en la barra de direcciones de su navegador.

Esto es lo que vimos cuando utilizamos Safari en Mac OS X (también comprobamos Firefox y Chrome y conseguimos la misma salida):



traduciendo: Hemos realizado tres búsquedas desde la adición del código de registro, y esto se parece a tres conjuntos de resultados. Pero ¿qué pasó con los datos de la solicitud? Parece que falta en esta salida?!?

### ***Where to start when things go wrong with your output***

#### **Dónde empezar cuando las cosas salen mal con su salida**

Cuando su salida no coincide exactamente con lo que esperaba (lo cual es el caso anterior), lo mejor es empezar por verificar exactamente qué datos le envió la aplicación web. Es importante tener en cuenta que lo que acaba de aparecer en pantalla es una representación o rendering (o interpretación) de los datos de la aplicación web realizados por su navegador web. Todos los principales navegadores le permiten ver los datos sin procesar recibidos sin renderizado aplicado. Esto se conoce como la fuente o **source** de la página, y verla puede ser una útil ayuda de depuración, así como un gran primer paso hacia la comprensión de lo que está pasando aquí.

Si utiliza Firefox o Chrome, haga clic con el botón derecho en la ventana del navegador y seleccione Ver origen de página o **View Page Source** en el menú emergente para ver los datos sin procesar enviados por su aplicación web. Si está ejecutando Safari, primero debe habilitar las opciones de desarrollador: abra las preferencias de Safari y, a continuación,

encienda el menú Mostrar desarrollo en la barra de menús en la parte inferior de la ficha Avanzadas. Una vez hecho esto, puede volver a la ventana del explorador, hacer clic con el botón derecho del ratón y seleccionar Mostrar origen de la página o **Show Page Source** en el menú emergente. Adelante y ver los datos sin procesar ahora, luego compararlo con lo que tenemos (en la página siguiente).

### ***Examine the Raw Data with View Source***

#### ***Examinar los datos sin formato con el código de vista***

Recuerde que la función `log_request` guarda dos datos para cada solicitud web que registra: el objeto `request`, así como los resultados de la llamada a `search4letters`. Pero cuando ve el registro (con `/viewlog`), sólo está viendo los datos de los resultados. ¿La visualización de la fuente (es decir, los datos sin procesar devueltos desde la aplicación web) ofrece alguna pista sobre qué sucedió con el objeto de solicitud?

Esto es lo que vimos cuando usamos Firefox para ver los datos sin procesar. El hecho de que la salida de cada objeto de la petición sea de color rojo es otra pista de que algo está mal con nuestros datos de registro:

```
1 <Request 'http://localhost:5000/search4' [POST]> {'i', 'e'}
2 <Request 'http://localhost:5000/search4' [POST]> {'i', 'e', 'u', 'a'}
3 <Request 'http://localhost:5000/search4' [POST]> {'a'}
```

Traduciendo: Los datos sobre el objeto de solicitud se han guardado en el registro, pero por alguna razón el navegador se niega a mostrarlo en la pantalla.

La explicación de por qué los datos de la solicitud no están rendering o presentando es sutil, y el hecho de que Firefox ha destacado los datos de la solicitud en rojo ayuda a entender lo que está pasando. Parece que no hay nada malo con los datos de la solicitud real. Sin

embargo, parece que los datos entre paréntesis angulares (< y >) están alterando el navegador. Cuando los navegadores ven un soporte de ángulo de apertura, tratan todo entre ese corchete y el corchete de ángulo de cierre correspondiente como una etiqueta HTML. Como <Request> no es una etiqueta HTML válida, los navegadores modernos simplemente la ignoran y se niegan a mostrar cualquier texto entre los corchetes, que es lo que está sucediendo aquí. Esto soluciona el misterio de la desaparición de los datos de solicitud. Pero todavía queremos poder ver estos datos cuando vemos el registro usando /viewlog.

Lo que necesitamos hacer es de alguna manera decirle al navegador que no trate los ángulos que rodean al objeto de solicitud o request como una etiqueta HTML, sino que los trata como texto sin formato. Por suerte, Flask viene con una función que puede ayudar.

## ***It's Time to Escape (Your Data)***

### ***Es hora de escapar (sus datos)***

Cuando se creó HTML por primera vez, sus diseñadores sabían que algunos diseñadores de páginas web querían mostrar corchetes angulares (y los otros caracteres que tienen un significado especial para HTML). En consecuencia, surgieron el concepto conocido como escaping: codificar los caracteres especiales de HTML para que pudieran aparecer en una página web pero no ser interpretados como HTML. Se definieron una serie de traducciones, una para cada carácter especial. Es una idea simple: un carácter especial como < se define como &lt ;, mientras que > se define como &gt ;. Si envía estas traducciones en lugar de los datos sin procesar, su navegador web hace lo correcto: muestra < y > en lugar de ignorarlos y muestra todo el texto entre ellos.

Flask incluye una función llamada `escape` (que en realidad es heredada de Jinja2). Cuando se proporcionan algunos datos sin procesar, escape traduce los datos a su equivalente en HTML-escape. Vamos a experimentar con escape en el Python >>> para obtener una idea de cómo funciona.

Comienza por importar la función de `escape` del módulo del flask, luego llama a escape con una cadena que no contiene ninguno de los caracteres especiales:

```
Import the function. → >>> from flask import escape  
>>> escape('This is a Request') ← Utilice "escape" con una cadena normal.  
Markup('This is a Request') ← Ningún cambio
```

La función de escape devuelve es un objeto de marcado `Markup`, que -para todos los intentos y propósitos- se comporta como una cadena. Cuando se pasa una cadena de escape que contiene cualquiera de los caracteres especiales de HTML, la traducción se hace para usted, como se muestra:

```
>>> escape('This is a <Request>')
Markup('This is a &lt;Request&gt;')
```

Use "escape" con una cadena que contiene algunos caracteres especiales.

Los caracteres especiales se han escapado (es decir, se han traducido).

El objeto `Markup` de Flask es un texto que ha sido marcado como seguro dentro de un contexto HTML / XML. El marcado hereda de la cadena unicode incorporada de Python y se puede usar en cualquier lugar donde se utilice una cadena.

Como en el ejemplo anterior (arriba), también puede tratar este objeto de marcado como si fuera una cadena regular.

Si de alguna manera podemos arreglar para llamar a `escape` en los datos en el archivo de registro o log file, deberíamos ser capaces de resolver el problema que tenemos actualmente con la no visualización de los datos de solicitud. Esto no debería ser difícil, ya que el archivo de registro se lee "de una vez" por la función `view_the_log` antes de ser devuelto como una cadena:

```
@app.route('/viewlog')
def view_the_log() -> str:
    with open('vsearch.log') as log:
        contents = log.read()
    return contents
```

Aquí están nuestros datos de registro (como una cadena).

Para resolver nuestro problema, todo lo que necesitamos hacer es llamar a `escape` en los contenidos o `contents`.

### ***Viewing the Entire Log in Your Webapp***

### ***Visualización del registro completo de su Webapp***

El cambio a su código es trivial, pero hace una gran diferencia. Añadir `escape` a la lista de importación para el módulo de flask (en la parte superior de su programa), a continuación, llamar a `escape` en la cadena devuelta de llamar al método `join`:

```

from flask import Flask, render_template, request, escape
...
@app.route('/viewlog')
def view_the_log() -> str:
    with open('vsearch.log') as log:
        contents = log.read()
    return escape(contents)

```

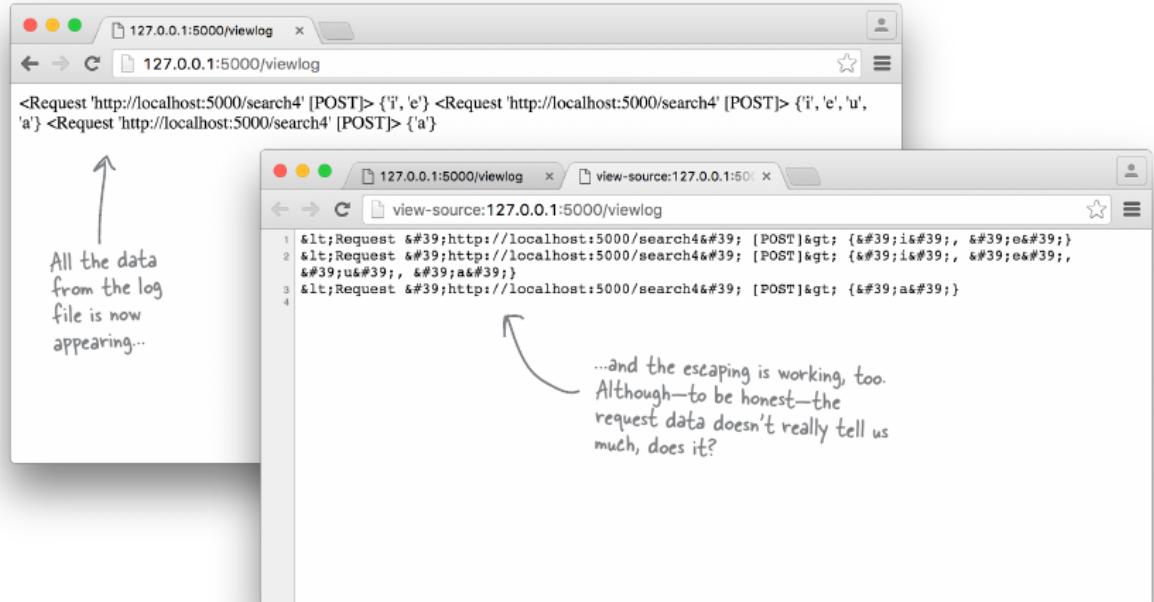
Add to the import list.

Llame "escape" en la cadena devuelta.



## Test Drive -

Modifique su programa para importar y llamar al `escape` como se muestra arriba, luego guarde su código (para que su webapp se vuelva a cargar). A continuación, vuelva a cargar la URL `/viewlog` en su navegador. Todos los datos de registro deben aparecer ahora en la pantalla. Asegúrese de ver el código HTML para confirmar que el `escape` está funcionando. Esto es lo que vimos cuando probamos esta versión de nuestra aplicación web con Chrome:



Traducimos:

1. Todos los datos del archivo de registro están apareciendo ahora ...
2. ... y `escape` está funcionando, también. Aunque—para ser honesto—el. Solicitud de datos realmente no nos decir mucho, ¿no?

## **Learning More About the Request Object**

### **Aprendiendo más información sobre el objeto Request**

Los datos del archivo de registro relacionados con la solicitud web no son realmente tan útiles. Aquí hay un ejemplo de lo que se registra actualmente; Aunque cada resultado registrado es diferente, cada solicitud web registrada aparece como exactamente la misma:

Each logged web request is the same.

```
<Request 'http://localhost:5000/search4' [POST]> {'i', 'e'}  
<Request 'http://localhost:5000/search4' [POST]> {'i', 'e', 'u', 'a'}  
<Request 'http://localhost:5000/search4' [POST]> {'a'}
```



Cada resultado registrado es diferente.

Estamos registrando la solicitud web en el nivel de objeto, pero realmente necesitamos estar buscando dentro de la solicitud y registrar algunos de los datos que contiene. Como se vio anteriormente en este libro, cuando se necesita aprender lo que contiene Python, se lo alimenta al directorio incorporado para ver una lista de sus métodos y atributos.

Hagamos un pequeño ajuste a la función `log_request` para registrar la salida de llamar a `dir` en cada objeto de petición o `request`. No es un cambio enorme ... en lugar de pasar el raw `req` como el primer argumento a imprimir, vamos a pasar en una versión stringificada del resultado de llamar a `dir(req)`. Esta es la nueva versión de `log_request` con el cambio resaltado:

```
def log_request(req:'flask_request', res:str) -> None:  
    with open('vsearch.log', 'a') as log:  
        print(str(dir(req)), res, file=log)
```

Llamamos "dir" en "req", que produce una lista, y luego stringificamos la lista pasando la lista a "str". La cadena resultante se guarda en el archivo de registro junto con el valor resultante de "res".



Vamos a probar este nuevo código de registro para ver qué diferencia hace. Realice los pasos siguientes:

1. Modifique su copia de `log_request` para que coincida con la nuestra.
2. Guarde `vsearch4log.py` para reiniciar su aplicación web.
3. Busque y elimine su archivo `vsearch.log` actual.

4. Utilice su navegador para ingresar tres búsquedas nuevas.
5. Ver el registro recién creado utilizando la URL /viewlog.

Ahora: eche un vistazo a lo que aparece en su navegador. ¿Lo que ve ahora ayuda en absoluto?

Esto es lo que vimos después de haber trabajado a través de los cinco pasos desde la parte inferior de la última página. Estamos usando Safari (aunque cada otro navegador muestra lo mismo):

Todo esto parece un poco desordenado. Pero mira de cerca: aquí están los resultados de una de las búsquedas que realizamos.

```
[__class__, __delattr__, __dict__, __dir__, __doc__, __enter__, __eq__, __exit__, __format__, __ge__, __getattribute__, __gt__, __hash__, __init__, __le__, __lt__, __module__, __ne__, __new__, __reduce__, __reduce_ex__, __repr__, __setattro__, __sizeof__, __str__, __subclasshook__, __weakref__, get_file_stream, get_stream_for_parsing, is_old_module, load_form_data, parse_content_type, parsed_content_type, accept_charset, accept_encodings, accept_languages, accept_mimetypes, access_route, application, args, authorization, base_url, blueprint, cache_control, charset, close, content_encoding, content_length, content_md5, content_type, cookies, data, date, dict_storage_class, disable_data_descriptor, encoding_errors, endpoint, environ, files, form, form_data_parser_class, from_values, full_path, get_data, get_json, headers, host, host_url, if_match, if_modified_since, if_none_match, if_range, if_unmodified_since, input_stream, is_multiprocess, is_multithread, is_run_once, is_secure, is_xhr, json, list_storage_class, make_form_data_parser, max_content_length, max_form_memory_size, max_forwards, method, mimetype, mimetype_params, module, on_json_loading_failed, parameter_storage_class, path, pragma, query_string, range, referrer, remote_addr, remote_user, routing_exception, scheme, script_root, shallow, stream, trusted_hosts, url, url_charset, url_root, url_rule, user_agent, values, view_args, want_form_data_parsed]
```

## *What's all this, then?*

### *¿Qué es todo esto, entonces?*

Usted puede simplemente seleccionar los resultados registrados en la salida anterior. El resto de la salida es el resultado de llamar a dir en el objeto request. Como se puede ver, cada solicitud tiene una gran cantidad de métodos y atributos asociados con él (incluso cuando se ignoran los dunders y maravillas). No tiene sentido registrar todos estos atributos.

Hicimos un vistazo a todos estos atributos, y decidimos que hay tres que creemos que son lo suficientemente importantes como para registrar:

- **req.form:** Los datos publicados en el formulario de laHTML
- **req.remote\_addr:** La dirección IP en la que se está ejecutando el navegador
- **req.user\_agent:** La identidad del navegador que publica los datos.

Vamos a ajustar **log\_request** para registrar estas tres piezas específicas de datos, además de los resultados de la llamada a **search4letters**.

## **Logging Specific Web Request Attributes**

### **Registro de atributos específicos de solicitud Web**

Como ahora tiene cuatro elementos de datos para registrar los detalles del formulario, la dirección IP remota, la identidad del navegador y los resultados de la llamada a search4letters, un primer intento de modificación de log\_request podría resultar en un código similar, donde cada dato se registra con su propia llamada de impresión:

```
def log_request(req:'flask_request', res:str) -> None:
    with open('vsearch.log', 'a') as log:
        print(req.form, file=log)
        print(req.remote_addr, file=log)
        print(req.user_agent, file=log)
        print(res, file=log)
```

Log each data item with its own "print" statement.

traducimos: Registre cada elemento de datos con su propia declaración "print".

Este código funciona, pero tiene un problema en que cada llamada de impresión agrega un carácter de nueva línea de forma predeterminada, lo que significa que hay cuatro líneas que se registran por solicitud web. Esto es lo que los datos se verían si el archivo de registro usó el código anterior:

```
ImmutableMultiDict([('letters', 'aeiou'), ('phrase', 'hitch-hiker')])  
127.0.0.1  
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) ... Safari/601.4.4  
{'i', 'e'}  
ImmutableMultiDict([('letters', 'aeiou'), ('phrase', 'life, the universe, and everything')])  
127.0.0.1  
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) ... Safari/601.4.4  
{'a', 'e', 'i', 'u'}  
ImmutableMultiDict([('letters', 'xyz'), ('phrase', 'galaxy')])  
127.0.0.1  
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) ... Safari/601.4.4  
{'x', 'y'}
```

There's a line of data for each remote IP address.

The data as entered into the HTML form appears on its own line. BTW: the "ImmutableMultiDict" is a Flask-specific version of Python's dictionary (and it works in the same way).

The browser is identified on its own line.

The results of the call to "search4letters" are clearly shown (each on its own line).

Traducimos:

1. Hay una línea de datos para cada dirección IP remota.
2. Los resultados de la llamada a "search4letters" se muestran claramente (cada uno en su propia línea).
3. El navegador se identifica en su propia línea.
4. Los datos introducidos en el formulario HTML aparecen en su propia línea. BTW: el "ImmutableMultiDict" es una versión específica de Flask del diccionario de Python (y funciona de la misma manera).

No hay nada erróneo en esto como estrategia (ya que los datos registrados son fáciles de leer para los humanos). Sin embargo, considere lo que tendría que hacer al leer estos datos en un programa: cada solicitud web registrada requeriría cuatro lecturas del archivo de registro, una por cada línea de datos registrados. Esto es a pesar de que las cuatro líneas de datos se refieren a una única solicitud web. Como estrategia, este enfoque parece un derroche. Sería mucho mejor si el código sólo registró una línea por solicitud web.

### ***Log a Single Line of Delimited Data***

#### ***Registro de una línea única de datos delimitados***

Una mejor estrategia de registro puede ser escribir los cuatro datos como una sola línea, mientras se usa un delimitador seleccionado apropiadamente para separar un elemento de datos del siguiente.

Elegir un delimitador puede ser complicado, ya que no desea elegir un carácter que realmente podría ocurrir en los datos que está registrando. Usar el carácter de espacio como delimitador es inutil (ya que los datos registrados contienen un montón de espacios), e incluso usar dos puntos (:), coma (,) y punto y coma (;) puede ser problemático dados los datos que se están registrando. Hemos consultado con los programadores en Head First Labs, y sugirieron usar una barra vertical (|) como delimitador: es fácil para nosotros los humanos detectar, y es poco probable que sea parte de los datos que registramos. Vamos con esta sugerencia y veamos cómo nos llevamos.

Como se vio anteriormente, podemos ajustar el comportamiento predeterminado de impresión proporcionando argumentos adicionales. Además del argumento de archivo, está el argumento final, que le permite especificar un valor de fin de línea alternativo sobre la línea de inicio predeterminada.

*Piense en un delimitador como una secuencia de uno o más personajes que desempeñan el papel de un límite dentro de una línea de texto. El ejemplo clásico es el carácter de coma (,) utilizado en los archivos CSV.*

Modificamos `log_request` para utilizar una barra vertical como valor de fin de línea, en contraposición a la nueva línea por defecto:

```

def log_request(req: 'flask_request', res: str) -> None
    with open('vsearch.log', 'a') as log:
        print(req.form, file=log, end='|')
        print(req.remote_addr, file=log, end='|')
        print(req.user_agent, file=log, end='|')
        print(res, file=log)

```

Cada una de estas instrucciones de "print" reemplaza la línea nueva por defecto con una barra vertical.

Esto funciona como se esperaba: cada solicitud web ahora da como resultado una sola línea de datos registrados, con una barra vertical que delimita cada elemento de datos registrado. Esto es lo que los datos parecen en nuestro archivo de registro cuando usamos esta versión modificada de `log_request`:

Each web request is written to its own line (which we've word-wrapped in order to fit on this page).

ImmutableMultiDict([('letters', 'aeiou'), ('phrase', 'hitch-hiker')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9|{'e', 'i'}

ImmutableMultiDict([('letters', 'aeiou'), ('phrase', 'life, the universe, and everything')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9|{'e', 'u', 'a', 'i'}

ImmutableMultiDict([('letters', 'xyz'), ('phrase', 'galaxy')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9|{'y', 'x'}

Did you spot the vertical bars used as delimiters? There are three bars, which means we have logged four pieces of data per line.

There were three web requests, so we see three lines of data in the log file.

Traducimos:

1. Cada solicitud web se escribe en su propia línea (que hemos envuelto en palabras para encajar en esta página).
2. ¿Detectó las barras verticales utilizadas como delimitadores? Hay tres barras, lo que significa que hemos registrado cuatro piezas de datos por línea.
3. Había tres peticiones web, por lo que vemos tres líneas de datos en el archivo de registro.

## ***One Final Change to Our Logging Code***

### ***Un cambio final a nuestro código de registro***

Trabajar con código excesivamente detallado es una picadura de muchos programadores de Python. Nuestra versión más reciente de `log_request` funciona bien, pero es más detallado de lo que necesita ser. Específicamente, se siente como una exageración para dar a cada elemento de datos registrados su propia declaración de impresión `print`.

La función de `print` tiene otro argumento opcional, `sep`, que le permite especificar un valor de separación que se utilizará al imprimir varios valores en una sola llamada para imprimir. De forma predeterminada, `sep` se establece en un solo carácter de espacio, pero puede utilizar cualquier valor que desee. En el código que sigue, las cuatro llamadas a imprimir

(desde la última página) han sido reemplazadas por una sola llamada de impresión print, que aprovecha el argumento sep, estableciéndolo en el carácter de barra vertical. Al hacerlo, negamos la necesidad de especificar un valor para end como el valor predeterminado de fin de línea de la impresión, por lo que todas las menciones de final se han eliminado de este código:

Sólo una llamada de "print" en lugar de cuatro

```
def log_request(req: 'flask_request', res: str) -> None:  
    with open('vsearch.log', 'a') as log:  
        print(req.form, req.remote_addr, req.user_agent, res, file=log, sep='|')
```



*¿No tiene PEP 8 algo que decir acerca de esta larga línea de código?*

**Sí, esta línea rompe una pauta de PEP 8.**

Algunos programadores de Python fruncen el ceño en esta última línea de código, ya que el estándar PEP 8 advierte específicamente contra líneas de más de 79 caracteres. Con 80 caracteres, nuestra línea de código está empujando un poco esta guía, pero creemos que es un compromiso razonable dado lo que estamos haciendo aquí.

Recuerde: la estricta adherencia a PEP 8 no es una necesidad absoluta, ya que PEP 8 es una guía de estilo, no un conjunto de reglas irrompible. Creemos que estamos bien.



### Exercise

Veamos qué diferencia hace este nuevo código. Ajuste su función `log_request` para que se vea así:

```
def log_request(req: 'flask_request', res: str) -> None:  
    with open('vsearch.log', 'a') as log:  
        print(req.form, req.remote_addr, req.user_agent, res, file=log, sep='|')
```

A continuación, realice estos cuatro pasos:

1. Guarde `vsearch4log.py` (que reinicia su aplicación web).
2. Busque y elimine su archivo `vsearch.log` actual.
3. Utilice su navegador para ingresar tres búsquedas nuevas.
4. Ver el registro recién creado utilizando la URL `/viewlog`.

Tener otra buena mirada en la pantalla del navegador. ¿Es mejor que antes?



### Test Drive

Después de completar los cuatro pasos detallados en el ejercicio anterior, realizamos nuestras últimas pruebas con Chrome. Esto es lo que vimos en la pantalla:



```

ImmutableMultiDict([('phrase', 'hitch-hiker'), ('letters', 'aeiou')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36{'e', 'i'}
ImmutableMultiDict([('phrase', 'life, the universe, and everything'), ('letters', 'aeiou')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36{'e', 'a', 'u', 'i'} ImmutableMultiDict([('phrase', 'galaxy'), ('letters', 'xyz')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36{'x', 'y'}

```

Traducimos:

Hay ciertamente mucho menos datos aquí que la producción producida por la versión anterior de "log\_request" pero esto es todavía un pedazo de un lío ... y es duro seleccionar los cuatro pedazos de datos registrados (incluso con todas esas barras verticales como delimitadores).

## ***From Raw Data to Readable Output De datos sin procesar a salida legible***

Los datos que se muestran en la ventana del navegador están en su forma sin procesar osea en raw. Recuerde, realizamos el código HTML que se escapa de los datos como leídos en el archivo de registro, pero no hacemos nada más antes de enviar la cadena al navegador web en espera. Los navegadores web modernos recibirán la cadena, eliminarán los caracteres de espacio en blanco no deseados (como espacios adicionales, líneas de nuevo y así sucesivamente) y, a continuación, volcarán o dump los datos a la ventana. Esto es lo que está sucediendo durante nuestro Test Drive. Los datos registrados, todos ellos, son visibles, pero es fácil de leer. Podríamos considerar realizar más manipulaciones de texto en los datos sin procesar (para hacer que la salida sea más fácil de leer), pero un mejor enfoque para producir una salida legible podría ser manipular los datos sin procesar de tal manera que lo conviertan en una tabla:

```

ImmutableMultiDict([('phrase', 'hitch-hiker'), ('letters', 'aeiou')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36{'e', 'i'} ImmutableMultiDict([('phrase', 'life, the universe, and everything'), ('letters', 'aeiou')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36{'e', 'a', 'u', 'i'} ImmutableMultiDict([('phrase', 'galaxy'), ('letters', 'xyz')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36{'x', 'y'}

```

¿Podemos tomar este (ilegible) datos en bruto ..

... y transformarlo en una tabla que se parece a esto?

Form Data	Remote_addr	User_agent	Results
ImmutableMultiDict([('phrase', 'hitch-hiker'), ('letters', 'aeiou')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36	{'e', 'i'}
ImmutableMultiDict([('phrase', 'life, the universe, and everything'), ('letters', 'aeiou')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36	{'e', 'a', 'u', 'i'}
ImmutableMultiDict([('phrase', 'galaxy'), ('letters', 'xyz')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36	{'x', 'y'}

Si nuestra aplicación web podría realizar esta transformación, cualquiera podría ver los datos de registro en su navegador web y probablemente darle sentido.

### ***Does This Remind You of Anything?***

### ***¿Esto le recuerda de cualquier cosa?***

Echa un vistazo a lo que estás tratando de producir. Para ahorrar espacio, solo mostramos la parte superior de la tabla mostrada en la página anterior. ¿Lo que usted está tratando de producir aquí le recuerda algo de antes en este libro?

Form Data	Remote_addr	User_agent	Results
ImmutableMultiDict([('phrase', 'hitch-hiker'), ('letters', 'aeiou')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36	{'e', 'i'}

***Corregirme si estoy equivocado, pero ¿no se parece mucho a mi compleja estructura de datos desde el final del capítulo 3?***

***Sí. Eso se parece a algo que hemos visto antes.***

Al final del capítulo 3, recuerde que tomamos la tabla de datos a continuación y la transformamos en una compleja estructura de datos -un diccionario de diccionarios:

Name	Gender	Occupation	Home Planet
Ford Prefect	Male	Researcher	Betelgeuse Seven
Arthur Dent	Male	Sandwich-Maker	Earth
Tricia McMillan	Female	Mathematician	Earth
Marvin	Unknown	Paranoid Android	Unknown

La forma de esta tabla es similar a lo que esperamos producir arriba, pero ¿es un diccionario de diccionarios la estructura de datos correcta para usar aquí?

### ***Use a Dict of Dicts...or Something Else?***

#### ***Utilizar un Dict de Dicts ... o algo más?***

La tabla de datos del Capítulo 3 se ajusta al diccionario del modelo de diccionarios porque le permitió ingresar rápidamente en la estructura de datos y extraer datos específicos. Por ejemplo, si quería conocer el planeta natal de Ford Prefect, todo lo que tenía que hacer era:

```
people['Ford']['Home Planet']
```

Acceda a los datos de Ford ...

... luego extraiga el valor asociado con la clave "Home Planet".

Cuando se trata de acceder al azar a una estructura de datos, nada supera a un diccionario de diccionarios. Sin embargo, ¿es esto lo que queremos para nuestros datos registrados?

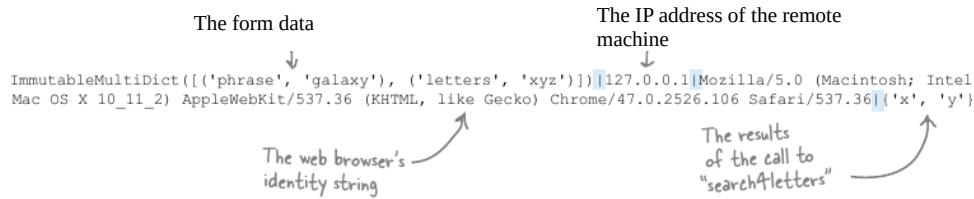
Consideremos lo que tenemos actualmente.

### ***Take a closer look at the logged data***

#### ***Echa un vistazo a los datos registrados***

Recuerde que cada línea registrada contiene cuatro datos, cada uno separado por barras verticales: los datos del formulario HTML, la dirección IP remota, la identidad del navegador web y los resultados de la llamada a `search4letters`.

He aquí una muestra de línea de datos de nuestro archivo `vsearch.log` con cada una de las barras verticales resaltadas:



Cuando los datos registrados se leen del archivo vsearch.log, llega a su código como una lista de cadenas gracias a nuestro uso del método readlines. Debido a que probablemente no necesitará acceder aleatoriamente a elementos de datos individuales de los datos registrados, la conversión de los datos en un diccionario de diccionarios parece un mal movimiento. Sin embargo, es necesario procesar cada línea en orden, así como procesar cada elemento de datos individual dentro de cada línea en orden. Ya tienes una lista de cadenas, por lo que estás a mitad de camino, ya que es fácil procesar una lista con un bucle for. Sin embargo, la línea de datos es actualmente una cadena, y este es el problema. Sería más fácil procesar cada línea si fuera una lista de elementos de datos, a diferencia de una cadena grande. La pregunta es: ¿es posible convertir una cadena en una lista?

### ***What's Joined Together Can Be Split Apart Lo que está unido puede ser dividido aparte***

Ya sabes que puedes tomar una lista de cadenas y convertirlas a una sola cadena usando el "truco de unión o join". Vamos a mostrar esto una vez más en el prompt >>>:

```
Una lista de cadenas individuales
↓
>>> names = ['Terry', 'John', 'Michael', 'Graham', 'Eric']
>>> pythons = '|'.join(names) ← The "join trick" in action.
>>> pythons
'Terry|John|Michael|Graham|Eric' ← A single string with each string from the
                                "names" list concatenated with the next
                                and delimited by a vertical bar
```

Traducimos:

1. Una sola cadena con cada cadena de la lista de "nombres" concatenada con la siguiente y delimitada por una barra vertical
2. El "truco de unión o join" en acción.

Gracias al "truco de unión join", lo que era una lista de cadenas es ahora una sola cadena, con cada elemento de lista separado de la siguiente por una barra vertical (en este caso). Puede invertir este proceso utilizando el método split, que viene incorporado en cada cadena de Python:

```
>>> individuals = pythons.split(' | ') ← Tome la cadena y divídela en una lista usando  
>>> individuals el delimitador dado.  
['Terry', 'John', 'Michael', 'Graham', 'Eric']  
Y ahora volvemos a nuestra lista de  
cadenas.
```

## ***Getting to a list of lists from a list of strings***

### ***Haciendo una lista de listas de una lista de cadenas***

Ahora que tiene el método `split` en su arsenal de codificación, volvamos a los datos almacenados en el archivo de registro o log y consideremos qué debe sucederle. En este momento, cada línea individual del archivo `vsearch.log` es una cadena:

Los datos brutos

```
ImmutableMultiDict([('phrase', 'galaxy'), ('letters', 'xyz')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36|{'x', 'y'}
```

Su código lee actualmente todas las líneas de `vsearch.log` en una lista de cadenas llamadas contenidos. Aquí se muestran las tres últimas líneas de código de la función `view_the_log`, que lee los datos del archivo y produce la cadena grande:

```
... Open the log file...  
with open('vsearch.log') as log: ←  
    contents = log.readlines() ← ... y leer todas las líneas de datos de registro en  
    return escape(''.join(contents)) una lista llamada "contenido".
```

La última línea de la función `view_the_log` toma la lista de cadenas de contenido y las concatena en una cadena grande (gracias a `join`). Esta cadena única se devuelve al navegador web en espera.

Si el `contenido` o `contents` fuera una lista de listas en lugar de una lista de cadenas, abriría la posibilidad de procesar el `contenido` en orden utilizando un bucle `for`. Entonces debería ser posible producir una salida más legible de lo que estamos viendo actualmente en pantalla.

## ***When Should the Conversion Occur?***

### ***¿Cuándo debe ocurrir la conversión?***

Por el momento, la función `view_the_log` lee todos los datos del archivo de registro en una lista de cadenas (llamado `contents`). Pero preferimos tener los datos como una lista de listas. La cosa es, cuando es el "mejor momento" para hacer esta conversión? ¿Deberíamos leer en todos los datos en una lista de cadenas, a continuación, convertirlo a una lista de listas "a

medida que vamos", o debemos construir la lista de listas mientras se lee en cada línea de datos?

*Los datos que necesitamos ya están en "contents", así que convierta eso en una lista de listas.*

*No estoy tan seguro, de esa manera vamos a terminar procesando los datos dos veces: una vez cuando lo leemos, y luego otra vez cuando lo convertimos.*

El hecho de que los datos ya se encuentren en el contents (gracias a nuestro uso del método **readlines**) no debe cegarnos al hecho de que ya hemos pasado por los datos una vez en este punto. Invocar **readlines** puede ser una sola llamada para nosotros, pero el intérprete (mientras ejecuta **readlines**) está haciendo un bucle a través de los datos del archivo. Si volvemos a recorrer los datos (para convertir las cadenas en listas), estamos duplicando la cantidad de bucle que está ocurriendo. Esto no es una gran cosa cuando sólo hay un puñado de entradas de registro ... pero podría ser un problema cuando el registro crece en tamaño. La conclusión es la siguiente: si podemos hacerlo haciendo sólo un bucle una vez, ¡entonces hagámoslo!

## **Processing Data: What We Aready Know**

### **Procesamiento de datos: lo que ya sabemos**

Anteriormente en este capítulo, vio tres líneas de código Python que procesaban las líneas de datos en el archivo todos.txt:

Asigne el flujo de archivos a una variable.

Abre el archivo.

```
with open('todos.txt') as tasks:  
    for chore in tasks:  
        print(chore, end='')
```

Realizar un procesamiento de una línea a la vez.

También has visto el método **split**, que toma una cadena y la convierte en una lista de cadenas basada en algún delimitador (por defecto a un espacio, si no se proporciona ninguno). En nuestros datos, el delimitador es una barra vertical. Supongamos que una línea de datos registrados se almacena en una variable llamada **line**. Puede convertir la única cadena en una lista de cuatro cadenas individuales -utilizando la barra vertical como delimitador- con esta línea de código:

Este es el nombre de la lista recién creada

```
four_strings = line.split('|')
```

Estamos usando una barra vertical como el delimitador

Utilice "split" para dividir la cadena en una lista de subcadenas.

Como nunca puedes estar seguro de si los datos que estás leyendo del archivo de registro están libres de caracteres que tengan un significado especial para HTML, también has aprendido sobre la función de escape. Esta función es proporcionada por Flask y convierte los caracteres especiales HTML de cualquier cadena en sus valores de escape equivalentes:

```
>>> escape('This is a <Request>') ← Utilice "escape" con una cadena que
    Markup('This is a &lt;Request&gt;')
```

Y, comenzando de nuevo en el capítulo 2, aprendiste que puedes crear una nueva lista asignándole una lista vacía ([]). También sabe que puede asignar valores al final de una lista existente llamando al método append y que puede acceder al último elemento de cualquier lista utilizando la anotación [-1]:

```
>>> names = [] ← Create a new, empty list
    called "names".
>>> names.append('Michael') } ← Add some data to the end
>>> names.append('John')   } ← of the existing list.
>>> names[-1]             ← Access the last item in the "names" list.
'John'
```

Armado con este conocimiento, vea si puede completar el ejercicio en la página siguiente.



Aquí está el código actual de `view_the_log`:

```
@app.route('/viewlog')
def view_the_log() -> str:
    with open('vsearch.log') as log:
        contents = log.readlines()
    return escape(''.join(contents))
```

Este código lee los datos del archivo de registro en una lista de cadenas. Su trabajo consiste en convertir este código para leer los datos en una lista de listas.

Asegúrese de que los datos escritos en la lista de listas se escapan correctamente, ya que no desea que ningún carácter especial de HTML pase por alto.

Además, asegúrese de que su nuevo código todavía devuelve una cadena al navegador web en espera.

Te hemos empezado, rellena el código que falta:

The first  
two lines  
remain  
unchanged.

```
→ { @app.route('/viewlog')  
    def view_the_log() -> 'str':
```

The function  
still returns → return str(contents)  
a string.

← Add your new  
code here.

Take your time here. Feel free to experiment at the >>> shell as needed,  
and don't worry if you get stuck—it's OK to flip the page and look at the  
solution.

traducimos:

1. Las dos primeras líneas permanecen sin cambios.
2. La función devuelve una cadena.
3. Tómese su tiempo aquí. Siéntase libre de experimentar en la shell >>> según sea necesario, y no se preocupe si se queda atascado-es bueno voltear la página y mirar la solución.
4. Añada su nuevo código aquí.



Aquí está el código de la función `view_the_log`:

```
@app.route('/viewlog')  
def view_the_log() -> str:  
    with open('vsearch.log') as log:  
        contents = log.readlines()  
    return escape(''.join(contents))
```

Su trabajo era convertir este código para leer los datos en una lista de listas. Usted debía asegurarse de que los datos escritos en la lista de listas se escapó correctamente, ya que no desea que los caracteres especiales de HTML se escapen.

También debes asegurarte de que tu nuevo código devuelva una cadena al navegador web en espera. Habíamos comenzado para usted, y usted debía completar el código que faltaba:

```

@app.route('/viewlog')
def view_the_log() -> 'str':
    Create a new,
    empty list called
    "contents".
    contents = []
    Open the log file and assign it
    to a file stream called "log".
    with open('vsearch.log') as log:
        Loop through
        each line in the
        "log" file stream.
        for line in log:
            contents.append([])
            Append a new, empty
            list to "contents".
            for item in line.split('|'):
                contents[-1].append(escape(item))
                Did you
                remember to
                call "escape"?
                Append the escaped data
                to the end of the list at
                the end of "contents".
    return str(contents)

```

Traduciendo:

1. Cree una lista nueva y vacía llamada "contenido".
2. Bucle a través de cada línea en el flujo de archivos "registro" o "log".
3. Dividir la línea (basada en la barra vertical), luego procesar cada elemento en la "lista de división" resultante.
4. Anexe los datos de escape al final de la lista al final de "contenido".
5. ¿Recuerdas llamar "escape"?
6. Añade una nueva lista vacía a "contenido".
7. Abra el archivo de registro y asignarlo a una secuencia de archivos llamada "log".

No se preocupe si esta línea de código de la anterior reescritura de la función `view_the_log` tiene su cabeza girando:

`contents[-1].append(escape(item))`

Lea este código de adentro hacia afuera, y de derecha a izquierda.

El truco para entender esta línea (inicialmente desalentadora) es leerla desde adentro hacia fuera, y de derecha a izquierda. Comienza con el `elemento` del bucle de inclusión, que se pasa para `escapar`. La cadena resultante se `agrega` a la lista al final (`[-1]`) de los `contenidos`. Recuerde: el `contenido` es una `lista de listas`.



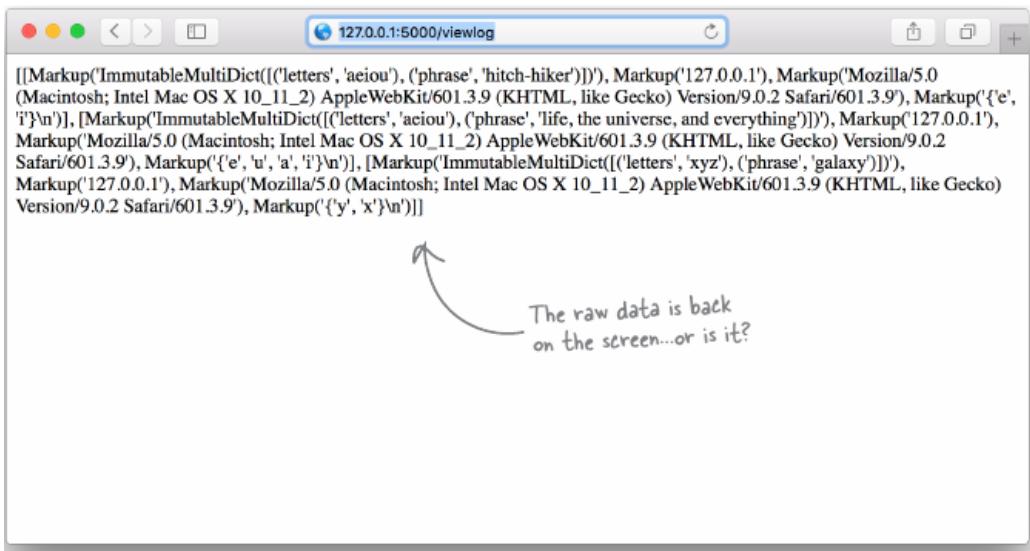
Siga adelante y cambie su función `view_the_log` para que se vea así:

```

@app.route('/viewlog')
def view_the_log() -> 'str':
    contents = []
    with open('vsearch.log') as log:
        for line in log:
            contents.append([])
            for item in line.split('|'):
                contents[-1].append(escape(item))
    return str(contents)

```

Guarde su código (lo que hace que su aplicación web se recargue), luego vuelva a cargar la URL `/viewlog` en su navegador. Esto es lo que vimos en la nuestra:



traduciendo: Los datos en bruto están de vuelta en la pantalla ... o es?

## Echa un vistazo más de cerca a la salida

A primera vista, la salida producida por esta nueva versión de `view_the_log` se ve muy similar a lo que tenías antes. Pero no lo es: esta nueva salida es una lista de listas, no una lista de cadenas. Este es un cambio crucial. Si ahora puede arreglar para procesar el contenido usando una plantilla Jinja2 diseñada apropiadamente, debería ser capaz de acercarse bastante a la salida legible requerida aquí.

## **Generate Readable Output With HTML**

### **Generar salida legible con HTML**

Recuerde que nuestro objetivo es producir una salida que se vea mejor en pantalla que los datos en bruto de la última página. Para ello, HTML incluye un conjunto de etiquetas para definir el contenido de las tablas, incluyendo: `<table>`, `<th>`, `<tr>` y `<td>`. Con esto en mente, echemos un vistazo a la parte superior de la tabla que esperamos producir una vez más. Tiene una fila de datos para cada línea en el registro, dispuestas como cuatro columnas (cada una con un título descriptivo).

Aquí hay una revisión rápida de las etiquetas de la tabla HTML:

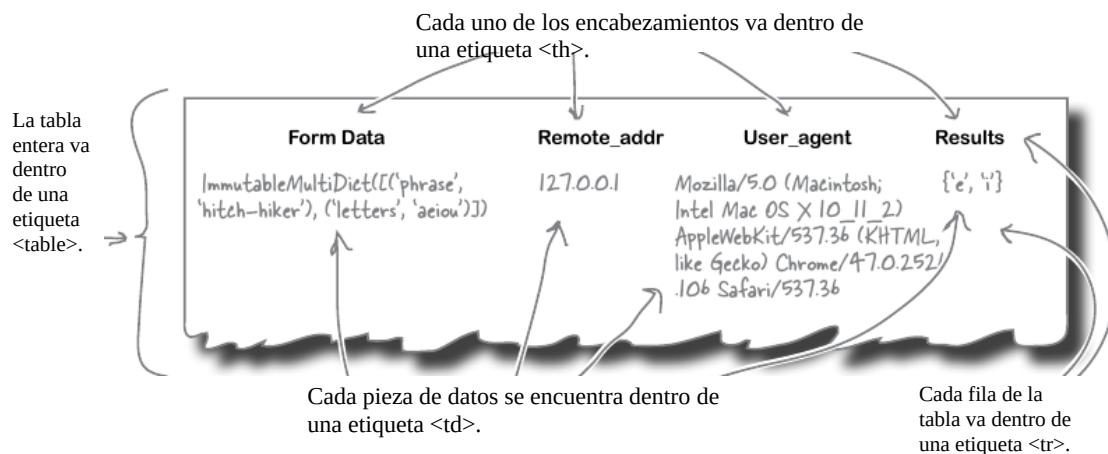
`<table>`: Una tabla.

`<tr>`: Una fila de datos de tabla.

`<th>`: Un encabezado de columna de tabla.

`<td>`: elemento de tabla (celda).

Podría poner toda la tabla dentro de una etiqueta HTML `<table>`, con cada fila de datos con su propia etiqueta `<tr>`. Los títulos descriptivos reciben las etiquetas `<th>`, mientras que cada pieza de datos sin procesar obtiene su propia etiqueta `<td>`:



Siempre que encuentre que necesita generar cualquier HTML (especialmente un `<table>`), recuerde Jinja2. El motor de plantilla Jinja2 está diseñado principalmente para generar HTML y el motor contiene algunas construcciones básicas de programación (basadas en la sintaxis de Python) que puede utilizar para "automatizar" cualquier lógica de pantalla necesaria.

En el último capítulo, has visto cómo las etiquetas `{} y {}` de Jinja2, así como la etiqueta `{% block %}`, te permiten usar variables y bloques de HTML como argumentos para las plantillas. Resulta que las etiquetas `{% y %}` son mucho más generales y pueden contener

cualquier instrucción Jinja2, siendo una de las sentencias soportadas una construcción de bucle for . En la siguiente página encontrarás una nueva plantilla que aprovecha el bucle de Jinja2 para crear la salida legible de la lista de listas contenidas en el contenido o contents.

### ***Embed Display Logic in Your Template***

#### ***Incorporar la lógica de visualización en su plantilla***

*No tiene que crear esta plantilla usted mismo. Descárgalo desde:*

<http://python.itcarlow.ie/ed2/>.

A continuación se muestra una nueva plantilla, llamada viewlog.html, que se puede utilizar para transformar los datos sin procesar del archivo de registro en una tabla HTML. La plantilla espera que la lista de contents de listas sea uno de sus argumentos. Hemos destacado los fragmentos de esta plantilla en los que queremos que te concentres. Tenga en cuenta que Jinja2 para la construcción de bucle es muy similar a Python. Hay dos diferencias principales:

- No hay necesidad de dos puntos (:) al final de la línea for (como la etiqueta %} actúa como un delimitador).
- El conjunto del bucle se termina con {%- endfor %}, ya que Jinja2 no admite sangrado (por lo que se requiere algún otro mecanismo).

Como puede ver, el primer bucle espera encontrar sus datos en una variable llamada the\_row\_titles, mientras que el segundo bucle espera sus datos en algo llamado the\_data. Un bucle tercero (incrustado en el segundo) espera que sus datos sean una lista de elementos:



Para garantizar una apariencia coherente, esta plantilla hereda de la misma plantilla de base utilizada en toda nuestra aplicación web.

La tabla entera va dentro de una etiqueta <table>.

```
{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>

<table>
    <tr>
        {% for row_title in the_row_titles %}
            <th>{{row_title}}</th>
        {% endfor %}
    </tr>
    {% for log_row in the_data %}
        <tr>
            {% for item in log_row %}
                <td>{{item}}</td>
            {% endfor %}
        </tr>
    {% endfor %}
</table>

{% endblock %}
```

Los títulos descriptivos (cada uno dentro de una etiqueta <th>) obtienen su propia fila (la etiqueta <tr>).

Cada elemento individual de datos registrados se incluye dentro de una etiqueta <td> y cada línea del archivo de registro tiene su propia etiqueta <tr>.

Asegúrese de colocar esta nueva plantilla en la carpeta de `templates` de su aplicación web antes de usarla.

## ***Producing Readable Output with Jinja2***

### ***Producción de salida legible con Jinja2***

Como la plantilla `viewlog.html` hereda de `base.html`, debe recordar proporcionar un valor para el argumento `the_title` y proporcionar una lista de títulos de columna (los títulos descriptivos) en `the_row_titles`. Y no olvides asignar contenido al argumento `the_data`.

La función `view_the_log` se ve actualmente así:

```
@app.route('/viewlog')
def view_the_log() -> 'str':
    contents = []
    with open('vsearch.log') as log:
        for line in log:
            contents.append([])
            for item in line.split('|'):
                contents[-1].append(escape(item))
    return str(contents)
```

We currently return a string to the waiting web browser.

traduciendo: Actualmente devolvemos una cadena al navegador web en espera.

Necesita llamar a `render_template` en `viewlog.html` y pasar valores para cada uno de los tres argumentos que espera. Vamos a crear una tupla de títulos descriptivos y asignar a `the_row_titles`, a continuación, asignar el valor de `contents` a los datos. También proporcionaremos un valor adecuado para `the_title` antes de renderizar la plantilla.

*Recuerde: una tupla es una lista de sólo lectura.*

Con todo eso en mente, vamos a modificar `view_the_log` (hemos resaltado los cambios):

```
@app.route('/viewlog')
def view_the_log() -> 'html':
    contents = []
    with open('vsearch.log') as log:
        for line in log:
            contents.append([])
            for item in line.split('|'):
                contents[-1].append(escape(item))

    Create a → titles = ('Form Data', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,) } ← Call "render_template",
                                                providing values for
                                                each of the template's
                                                arguments.
```

Change the annotation to indicate that HTML is being returned (instead of a string).

traduciendo:

1. Cree una tupla de títulos descriptivos.
2. Llame "render\_template", proporcionando valores para cada uno de los argumentos de la plantilla.
3. Cambie la anotación para indicar que HTML está siendo devuelto (en lugar de una cadena).

Siga adelante y realice estos cambios en su función `view_the_log` y guárdelos para que Flask reinicie su aplicación web. Cuando esté listo, vea el registro en su navegador usando la URL <http://127.0.0.1:5000/viewlog>.



Esto es lo que vimos cuando vimos el registro usando nuestra aplicación web actualizada. La página tiene la misma apariencia que todas nuestras otras páginas, por lo que estamos seguros de que nuestra aplicación web utiliza la plantilla correcta.

Estamos muy satisfechos con el resultado (y esperamos que también lo sean), ya que se ve muy similar a lo que esperábamos lograr: salida legible.

Form Data	Remote_addr	User_agent	Results
ImmutableMultiDict([('letters', 'aeiou'), ('phrase', 'hitch-hiker')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9	{'e', 'i'}
ImmutableMultiDict([('letters', 'aeiou'), ('phrase', 'life, the universe, and everything')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9	{'e', 'u', 'a', 'i'}
ImmutableMultiDict([('letters', 'xyz'), ('phrase', 'galaxy')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9	{'y', 'x'}

No sólo es esta salida legible,  
pero también se ve bien.

Si ve el origen de la página anterior, haga clic con el botón derecho del ratón en la página y, a continuación, elija la opción adecuada en el menú emergente; verá que cada elemento de datos del registro recibe su propia etiqueta `<td>`, Cada línea de datos tiene su propia etiqueta `<tr>`, y toda la tabla está dentro de una `<table>` HTML.

### ***The Current State of Our Webapp Code***

### ***El estado actual de nuestro código Webapp***

Hagamos una pausa por un momento y revisemos el código de nuestra webapp. La adición del código de registro (`log_request` y `view_the_log`) se ha añadido a la base de código de nuestra webapp, pero todo sigue en una sola página. Aquí está el código de `vsearch4web.py` que se muestra en una ventana de edición IDLE (que le permite revisar el código en toda su gloria resaltada por sintaxis):

```

vsearch4web.py - /Users/paul/Desktop/_NewBook/ch06/webapp/vsearch4web.py (3.5.1)

from flask import Flask, render_template, request, escape
from vsearch import search4letters

app = Flask(__name__)

def log_request(req:'flask_request', res:str) -> None:
    with open('vsearch.log', 'a') as log:
        print(req.form, req.remote_addr, req.user_agent, res, file=log, sep='|')

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    log_request(request, results)
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

@app.route('/viewlog')
def view_the_log() -> 'html':
    contents = []
    with open('vsearch.log') as log:
        for line in log:
            contents.append([])
            for item in line.split('|'):
                contents[-1].append(escape(item))
    titles = ('Form Data', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,)

if __name__ == '__main__':
    app.run(debug=True)

```

Ln: 2 Col: 0

## *Asking Questions of Your Data*

### *Preguntas sobre sus datos*

La funcionalidad de nuestra webapp está funcionando muy bien, pero ¿estamos más cerca de responder a las preguntas planteadas al principio de este capítulo: ¿Cuántas solicitudes se han respondido? ¿Cuál es la lista de cartas más común? ¿De qué direcciones IP provienen las peticiones? ¿Qué navegador se utiliza más?

Las dos últimas preguntas pueden ser respondidas en cierta medida por la salida que muestra la URL / viewlog. Puede saber de dónde vienen las peticiones (la columna

Remote\_addr), así como ver qué navegador web se está utilizando (la columna User\_agent). Pero, si desea calcular cuál de los principales navegadores es utilizado por los usuarios de su sitio, eso no es tan fácil. Basta con mirar los datos de registro visualizados no es suficiente; Tendrás que realizar cálculos adicionales.

Las dos primeras preguntas tampoco pueden ser contestadas fácilmente. Debe quedar claro que también se deben realizar cálculos adicionales aquí.

*Todo lo que tenemos que hacer es escribir un montón más código para realizar estos cálculos, ¿verdad?*

### Sólo escriba más código cuando tenga que hacerlo.

Si todo lo que teníamos disponible era Python, entonces, sí, tendríamos que escribir mucho más código para responder a estas preguntas (y cualquier otra que pudiera surgir). Después de todo, es divertido escribir código Python, y Python también es genial en la manipulación de datos. Escribir más código para responder a nuestras preguntas parece una obviedad, ¿no?

Bueno ... existen otras tecnologías que facilitan responder al tipo de preguntas que estamos planteando sin que tengamos que escribir mucho más código Python. En concreto, si pudiéramos guardar los datos de registro en una base de datos, podríamos aprovechar la potencia de la tecnología de consulta de la base de datos para responder a casi cualquier pregunta que pudiera surgir.

En el capítulo siguiente, verá qué implica modificar su aplicación web para registrar sus datos en una base de datos en lugar de en un archivo de texto.

## Chapter 6's Code

```
Remember: they both do  
the same thing, but Python  
programmers prefer this code  
over this.  
  
with open('todos.txt') as tasks:  
    for chore in tasks:  
        print(chore, end='')
```

```
tasks = open('todos.txt')  
for chore in tasks:  
    print(chore, end='')  
tasks.close()
```

traducción: Recuerde: ambos hacen lo mismo, pero los programadores de Python prefieren este código sobre esto.

Here's the code we added to the  
webapp to support logging our web  
requests to a text file.

```
...
def log_request(req: 'flask_request', res: str) -> None:
    with open('vsearch.log', 'a') as log:
        print(req.form, req.remote_addr, req.user_agent, res, file=log, sep='|')

...
@app.route('/viewlog')
def view_the_log() -> 'html':
    contents = []
    with open('vsearch.log') as log:
        for line in log:
            contents.append([])
            for item in line.split('|'):
                contents[-1].append(escape(item))
    titles = ('Form Data', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,
                           )
...

```

We aren't showing all the "vsearch4web.py"  
code here, just the new stuff. (You'll find  
the entire program two pages back.)

Traducción :

1. Aquí está el código que agregamos a la aplicación web para admitir el registro de nuestras solicitudes web en un archivo de texto.
2. No mostramos todo el código "vsearch4web.py" aquí, solo las cosas nuevas (Encontrarás todo el programa dos páginas atrás).

## **7 using a database**

# ***Putting Python's DB-API to Use***

### ***Poner la API de DB de Python a usar***

***Almacenamiento de datos en un sistema de base de datos relacional es útil.***

En este capítulo, aprenderá cómo escribir código que interactúa con la popular tecnología de base de datos MySQL, utilizando una API de base de datos genérica llamada DB-API. La DB-API (que viene de serie con cada instalación de Python) le permite escribir código que se transfiere fácilmente de un producto de base de datos a otro ... suponiendo que su base de datos habla SQL. Aunque usaremos MySQL, no hay nada que lo impida de usar su código DB-API con su base de datos relacional favorita, cualquiera que sea. Veamos qué implica el uso de una base de datos relacional con Python. No hay mucho Python nuevo en este capítulo, pero usar Python para hablar con las bases de datos es una gran cosa, así que vale la pena aprender.

#### ***Database-Enabling Your Webapp***

#### ***Database-Habilitación de su Webapp***

El plan para este capítulo es llegar al punto en el que puede modificar su aplicación web para almacenar sus datos de registro en una base de datos, en lugar de un archivo de texto, como en el último capítulo. La esperanza es que, al hacerlo, pueda entonces dar respuestas a las preguntas planteadas en el último capítulo: ¿Cuántas solicitudes se han respondido? ¿Cuál es la lista de cartas más común? ¿De qué direcciones IP provienen las peticiones? ¿Qué navegador se utiliza más?

Para llegar, sin embargo, tenemos que decidir sobre un sistema de base de datos para su uso. Hay un montón de opciones aquí, y sería fácil tomar una docena de páginas o más para presentar un montón de tecnologías alternativas de base de datos mientras se exploran las ventajas y desventajas de cada uno. Pero no vamos a hacer eso. En cambio, vamos a seguir con una opción popular y utilizar MySQL como nuestra tecnología de base de datos.

Habiendo seleccionado MySQL, aquí están las cuatro tareas en las que trabajaremos en las siguientes docenas de páginas:

1. Instalar el servidor MySQL
2. Instalar un controlador de base de datos MySQL para Python
3. Crea tu base de datos y tablas de tu webapp
4. Crear código para trabajar con la base de datos y las tablas de nuestra webapp

Con estas cuatro tareas completas, estaremos en condiciones de modificar la vsearch4web.py código de registro a MySQL en lugar de un archivo de texto. A continuación, utilizar SQL para preguntar y con suerte responder nuestras preguntas.

## *there are no Dumb Questions*

**P:** ¿Tenemos que usar MySQL aquí?

**R:** Si desea seguir los ejemplos en el resto de este capítulo, la respuesta es sí.

**P:** ¿Puedo usar MariaDB en lugar de MySQL?

**R:** Sí. Como MariaDB es un clon de MySQL, no tenemos problemas con el uso de MariaDB como sistema de base de datos en lugar de MySQL "oficial". (De hecho, en Head First Labs, MariaDB es un favorito entre el equipo de DevOps.)

**P:** ¿Qué pasa con PostgreSQL? ¿Puedo usar eso?

**R:** Emm, eh ... sí, con la siguiente advertencia: si ya está utilizando PostgreSQL (o cualquier otro sistema de gestión de base de datos basado en SQL), puede intentar usarlo en lugar de MySQL. Sin embargo, tenga en cuenta que este capítulo no proporciona instrucciones específicas relacionadas con PostgreSQL (o cualquier otra cosa), por lo que puede tener que experimentar por su cuenta cuando algo que le demostrar que trabaja con MySQL no funciona de la misma manera con su Elegida. También hay el SQLite independiente de un solo usuario, que viene con Python y le permite trabajar con SQL sin necesidad de un servidor separado. Dicho esto, que la tecnología de base de datos que utiliza mucho depende de lo que está tratando de hacer.

### ***Task 1: Install the MySQL Server***

#### ***Tarea 1: Instalar el servidor MySQL***

Si ya tiene instalado MySQL en su computadora, no dude en pasar a la Tarea 2.

La forma de instalar MySQL depende del sistema operativo que utilices. Afortunadamente, la gente detrás de MySQL (y su primo cercano, MariaDB) hacer un gran trabajo de hacer el proceso de instalación directa.

Si está ejecutando Linux, no debería tener problemas para encontrar mysql-server (o mariadb-server) en sus repositorios de software. Utilice su utilidad de instalación de software (apt, aptitude, rpm, yum, o lo que sea) para instalar MySQL como lo haría con cualquier otro paquete.

Si está ejecutando Mac OS X, le recomendamos instalar Homebrew (averigüe acerca de Homebrew aquí: <http://brew.sh>) y luego usarlo para instalar MariaDB, ya que en nuestra experiencia esta combinación funciona bien.

Para todos los demás sistemas (incluidas las distintas versiones de Windows), le recomendamos que instale Community Edition del servidor MySQL, disponible en:

<http://dev.mysql.com/downloads/mysql/>

O, si quieres ir con MariaDB, echa un vistazo a:

<https://mariadb.org/download/>

Asegúrese de leer la documentación de instalación asociada a la versión del servidor que descargue e instale.

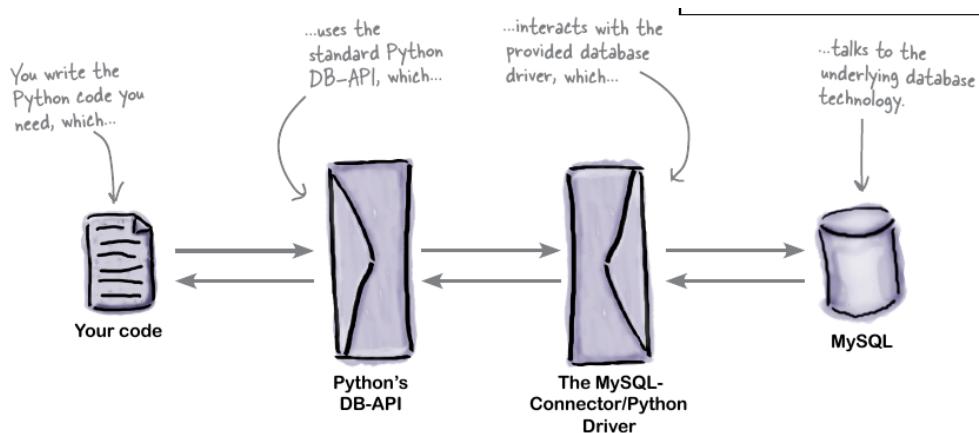
## ***Introducing Python's DB-API***

Con el servidor de bases de datos instalado, vamos a aparcar un poco, mientras que añadimos soporte para trabajar con MySQL en Python.

Fuera de la caja, el intérprete de Python viene con cierto apoyo para trabajar con bases de datos, pero nada específico a MySQL. Lo que se proporciona es una API de base de datos estándar (interfaz de programador de aplicaciones) para trabajar con bases de datos basadas en SQL, conocidas como DB-API. Lo que falta es el controlador para conectar el DB-API hasta la tecnología de base de datos actual que está utilizando.

La convención es que los programadores usan el DB-API al interactuar con cualquier base de datos subyacente usando Python, sin importar lo que sucede con la tecnología de la base de datos. Lo hacen porque el controlador protege a los programadores de tener que entender los detalles de la interacción con la API real de la base de datos, ya que la API de DB proporciona una capa abstracta entre los dos. La idea es que, mediante la programación a la DB-API, puede reemplazar la tecnología de base de datos subyacente según sea necesario sin tener que tirar cualquier código existente.

Más adelante tendremos más información sobre la API de DB en este capítulo. A continuación se muestra una visualización de lo que sucede cuando se utiliza la API de DB de Python:



Traduciendo:

1. Escribe el código de Python que necesitas, que ...
2. ... utiliza el estándar Python DB-API, que ...
3. ... interactúa con el controlador de base de datos proporcionado, que ...
4. ... habla con la tecnología de bases de datos subyacente.

Algunos programadores miran este diagrama y concluyen que el uso de DB-API de Python debe ser enormemente ineficiente. Después de todo, hay dos capas de tecnología entre su código y el sistema de base de datos subyacente. Sin embargo, el uso de la API de DB le permite intercambiar la base de datos subyacente según sea necesario, evitando cualquier "bloqueo" de la base de datos, que se produce al codificar directamente en una base de datos. Cuando consideras que no hay dos dialectos SQL iguales, el uso de DB-API ayuda proporcionando un nivel superior de abstracción.

## ***Task 2: Install a MySQL Database Driver for Python***

### ***Tarea 2: Instalar un controlador de base de datos MySQL para Python***

Cualquier persona es libre de escribir un controlador de base de datos (y muchas personas lo hacen), pero es típico para cada fabricante de bases de datos proporcionar un controlador oficial para cada uno de los lenguajes de programación que admiten. Oracle, el propietario de las tecnologías MySQL, proporciona el controlador MySQL-Connector / Python, y eso es lo que nos proponemos utilizar en este capítulo. Sólo hay un problema: MySQL-Connector / Python no se puede instalar con pip.

¿Significa eso que no tenemos suerte cuando se trata de usar MySQL-Connector / Python con Python? No, lejos de eso. El hecho de que un módulo de terceros no utilice la

maquinaria de pip es rara vez un tapón de espectáculo. Todo lo que necesitamos hacer es instalar el módulo "a mano" -es una pequeña cantidad de trabajo extra (sobre el uso de pip), pero no mucho.

Vamos a instalar el controlador MySQL-Connector / Python a mano (teniendo en cuenta que hay otros controladores disponibles, como [PyMySQL](#), que dicho, preferimos [MySQL-Connector / Python](#), ya que es el controlador oficialmente soportado proporcionado por los creadores de MySQL).

Comience visitando la página de descarga de [MySQL-Connector / Python](#): <https://dev.mysql.com/downloads/connector/python/>. El aterrizaje en esta página web es probable que preseleccione su sistema operativo desde el menú desplegable Seleccionar plataforma. Ignore esto y ajuste el menú desplegable de selección para leer Plataforma Independiente, como se muestra aquí:

The screenshot shows the MySQL Connector/Python download page under the 'Generally Available (GA) Releases' tab. A callout bubble points to the 'Select Platform' dropdown menu, which is currently set to 'Platform Independent'. Another callout bubble points to the '2.1.3' version entry in the first download row, which is circled in red. A third callout bubble points to a note at the bottom of the page: 'We suggest that you use the [MD5 checksums and GnuPG signatures](#) to verify the integrity of the packages you download.'

Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-python-2.1.3.tar.gz)	2.1.3	265.6K	<a href="#">Download</a>
Platform Independent (Architecture Independent), ZIP Archive Python (mysql-connector-python-2.1.3.zip)	2.1.3	347.9K	<a href="#">Download</a>

No se preocupe si su versión es diferente de la nuestra: siempre y cuando sea al menos esta versión, todo está bien.

A continuación, haga clic en cualquiera de los botones de descarga (normalmente, los usuarios de Windows deben descargar el archivo ZIP, mientras que los usuarios de Linux y Mac OS X pueden descargar el archivo GZ). Guarde el archivo descargado en su computadora, luego haga doble clic en el archivo para expandirlo dentro de su ubicación de descarga.

## Instalar MySQL-Connector / Python

Con el controlador descargado y ampliado en su computadora, abra una ventana de terminal en la carpeta recién creada (si está en Windows, abra la ventana de terminal con Ejecutar como Administrador).

En nuestro ordenador, la carpeta creada se llama `mysql-connector-python-2.1.3` y se expandió en nuestra carpeta Descargas. Para instalar el controlador en Windows, emita este comando desde la carpeta `mysql-connector-python-2.1.3`:

```
py -3 setup.py install
```

En Linux o Mac OS X, utilice este comando:

```
sudo -H python3 setup.py install
```

Independientemente del sistema operativo que utilice, la emisión de cualquiera de los comandos anteriores da como resultado una colección de mensajes que aparecen en la pantalla, que deberían ser similares a los siguientes:

```
running install
Not Installing C Extension
running build
running build_py
running install_lib
running install_egg_info
Removing /Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/
mysql_connector_python-2.1.3-py3.5.egg-info
Writing /Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/
mysql_connector_python-2.1.3-py3.5.egg-info
```

Estas rutas pueden ser diferentes en su computadora. No te preocupa si lo son.



Cuando instala un módulo con pip, se ejecuta aunque este mismo proceso, pero oculta estos mensajes de usted. Lo que está viendo aquí son los mensajes de estado que indican que la instalación se realiza sin problemas. Si algo sale mal, el mensaje de error resultante debe proporcionar suficiente información para resolver el problema. Si todo va bien con la instalación, la aparición de estos mensajes es la confirmación de que MySQL-Connector / Python está listo para ser utilizado.

*there are no  
Dumb Questions*

**P: ¿Debo preocuparme por el mensaje "Not Installing C Extension"?**

R: No. Los módulos de terceros a veces incluyen código C incrustado, que puede ayudar a mejorar el procesamiento intensivo desde el punto de vista computacional. Sin embargo, no todos los sistemas operativos vienen con un compilador C preinstalado, por lo que hay que pedir específicamente que se habilite el soporte de extensión C al instalar un módulo (si decide que lo necesita). Cuando no pregunta, la maquinaria de instalación de módulo de

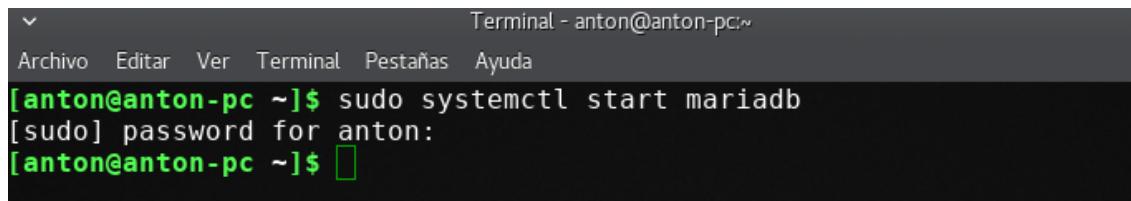
terceros utiliza (potencialmente más lento) código Python en lugar del código C. Esto permite que el módulo funcione en cualquier plataforma, independientemente de la existencia de un compilador C. Cuando un módulo de terceros utiliza el código Python exclusivamente, se le conoce como escrito en "Python puro". En el ejemplo anterior, hemos instalado la versión pura de Python del controlador MySQL-Connector / Python.

### **Task 3: Create Our Webapp's Database and Tables**

### **Tarea 3: Crear la base de datos y tablas de Webapp**

#### **INGRESAMOS A MYSQL:**

*antes de cargar mysql tenemos que cargar primero mariadb server:*



```
Terminal - anton@anton-pc:~  
Archivo Editar Ver Terminal Pestañas Ayuda  
[anton@anton-pc ~]$ sudo systemctl start mariadb  
[sudo] password for anton:  
[anton@anton-pc ~]$ 
```

```
[anton@anton-pc ~]$ sudo systemctl start mariadb  
[sudo] password for anton:  
[anton@anton-pc ~]$ mysql -u root -p  
Enter password:  
Welcome to the MariaDB monitor. Commands end with ; or \g.  
Your MariaDB connection id is 3  
Server version: 10.1.23-MariaDB MariaDB Server
```

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>

luego ya podemos cargar mysql:

```
[anton@anton-pc ~]$ mysql -u root -p  
Enter password:  
Welcome to the MariaDB monitor. Commands end with ; or \g.  
Your MariaDB connection id is 13  
Server version: 10.1.22-MariaDB MariaDB Server  
  
Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
MariaDB [(none)]> SHOW DATABASES;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| test |  
+-----+  
4 rows in set (0.02 sec)  
  
MariaDB [(none)]>
```

Ahora tiene el servidor de base de datos MySQL y el controlador MySQL-Connector / Python instalado en su computadora. Es hora de la Tarea 3, que consiste en crear la base de datos y las tablas requeridas por nuestra aplicación web.

Para ello, vas a interactuar con el servidor MySQL utilizando su herramienta de línea de comandos, que es una pequeña utilidad que se inicia desde la ventana del terminal. Esta herramienta se conoce como la consola de MySQL. Este es el comando para iniciar la consola, iniciando sesión como administrador de base de datos MySQL (que utiliza el ID de usuario raíz):

```
mysql -u root -p
```

Si establece una contraseña de administrador al instalar el servidor MySQL, escriba la contraseña después de pulsar la tecla Intro. Si no tiene contraseña, pulse la tecla Intro dos veces. De cualquier manera, se le llevará al prompt de la consola, que se parece a esto (a la izquierda) cuando se utiliza MySQL, o como este (a la derecha) cuando se utiliza MariaDB:

mysql>

MariaDB [None]>

Los comandos que escriba en el indicador de la consola se entregan al servidor MySQL para su ejecución. Comencemos creando una base de datos para nuestra aplicación web. Recuerde: queremos utilizar la base de datos para almacenar datos de registro, por lo que el nombre de la base de datos debe reflejar este propósito. Llámemos a nuestra base de datos vsearchlogDB. Aquí está el comando de consola que crea nuestra base de datos:

mysql> **create database vsearchlogDB;**

Asegúrese de finalizar cada comando que ingrese en la consola MySQL con un punto y coma.

En mi maquina:

```
MariaDB [(none)]> show databases;
+-----+
| Database      |
+-----+
| DVDCOLLECTION |
| information_schema |
| mysql          |
| performance_schema |
| prueba         |
| sakila         |
| shopping        |
| test           |
+-----+
8 rows in set (0.07 sec)
```

```
MariaDB [(none)]> create database vsearchlogDB;
Query OK, 1 row affected (0.01 sec)
```

```
MariaDB [(none)]> show databases;
+-----+
| Database      |
+-----+
| DVDCOLLECTION |
| information_schema |
| mysql          |
| performance_schema |
| prueba         |
| sakila         |
| shopping        |
| test           |
| vsearchlogDB   |
+-----+
9 rows in set (0.00 sec)
```

```
MariaDB [(none)]>
```

La consola responde con un mensaje de estado (bastante críptico): Consulta OK, 1 fila afectada (0.00 segundos). Esta es la manera de la consola de hacerle saber que todo es de oro.

Vamos a crear un **ID de usuario** y una **contraseña** de base de datos específicamente para nuestra aplicación web para usar cuando se interactúa con MySQL en lugar de usar el ID de usuario raíz todo el tiempo (que se considera como mala práctica). Este siguiente comando crea un nuevo usuario de MySQL denominado **vsearch**, utiliza "**vsearchpasswd**" como contraseña del nuevo usuario y concede al usuario de **vsearch** derechos completos a la base de datos **vsearchlogDB**:

```
mysql> grant all on vsearchlogDB.* to 'vsearch' identified by 'vsearchpasswd';
```

Puede usar una contraseña diferente si lo desea. Sólo recuerde utilizar el suyo en comparación con el nuestro en los ejemplos que siguen.

```
MariaDB [(none)]> grant all on vsearchlogDB.* to 'vsearch' identified by 'vsearchpasswd';
Query OK, 0 rows affected (0.13 sec)
```

```
MariaDB [(none)]>
```

Debería aparecer un mensaje de estado de Consulta OK similar, que confirme la creación de este usuario. Ahora salgamos de la consola usando este comando:

```
MariaDB [(none)]> quit
Bye
```

Aparecerá un mensaje de advertencia amistosa desde la consola antes de volver a su sistema operativo.

### ***Decidir sobre una estructura para sus datos de registro***

Ahora que ha creado una base de datos para utilizarla con su aplicación web, puede crear cualquier número de tablas dentro de esa base de datos (según lo requiera su aplicación). Para nuestros propósitos, una sola tabla será suficiente aquí, ya que todo lo que necesitamos almacenar son los datos relativos a cada solicitud web registrada.

Recuerde cómo almacenamos estos datos en un archivo de texto en el capítulo anterior, con cada línea en el archivo **vsearch.log** que se ajusta a un formato específico:

...as well as the value of "letters".

We log the value of the "phrase" ...

The IP address of the computer that submitted the form data is also logged.

ImmutableMultiDict([('phrase', 'galaxy'), ('letters', 'xyz')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36|{'x', 'y'}

There's a (rather large) string that describes the web browser being used.

Last—but not least—the actual results produced by searching for "letters" in "phrase" are also logged.

Como mínimo, la tabla que cree necesita cinco campos: para la frase, las letras, la dirección IP, la cadena del navegador y los valores de resultado. Pero también incluiremos otros dos campos: un identificador único para cada solicitud registrada, así como una marca de tiempo que registra cuando se registró la solicitud. Como estos dos últimos campos son tan comunes, MySQL proporciona una forma fácil de agregar estos datos a cada solicitud registrada, como se muestra en la parte inferior de esta página.

Puede especificar la estructura de la tabla que desea crear en la consola. Sin embargo, antes de hacerlo, iniciemos como usuario recién creado de vsearch utilizando este comando (y proporcionando la contraseña correcta después de presionar la tecla Intro):

```
mysql -u vsearch -p vsearchlogDB ← Recuerde: establecemos la contraseña de este usuario en "vsearchpasswd".
```

Esta es la instrucción SQL que usamos para crear la tabla requerida (llamada `log`). Tenga en cuenta que el símbolo `->` no forma parte de la instrucción SQL, ya que se agrega automáticamente por la consola para indicar que espera más información de usted (cuando el SQL se ejecuta en varias líneas). La instrucción finaliza (y se ejecuta) cuando escribe el carácter de punto y coma final y, a continuación, presione la tecla Intro:

```
mysql> create table log (
-> id int auto_increment primary key,
-> ts timestamp default current_timestamp,
-> phrase varchar(128) not null,
-> letters varchar(32) not null,
-> ip varchar(16) not null,
-> browser_string varchar(256) not null,
-> results varchar(64) not null );
```

This is the console's continuation symbol.

MySQL proporcionará automáticamente datos para estos campos.

Estos campos contienen los datos para cada solicitud (como se proporciona en los datos del formulario).

```
[anton@anton-pc ~]$ sudo systemctl start mariadb
[sudo] password for anton:
[anton@anton-pc ~]$ mysql -u vsearch -p vsearchlogDB
Enter password:
```

```

ERROR 1045 (28000): Access denied for user 'vsearch'@'localhost' (using password: YES)
[anton@anton-pc ~]$ mysql -u vsearch -p vsearchlogDB
Enter password:
ERROR 1045 (28000): Access denied for user 'vsearch'@'localhost' (using password: YES)
[anton@anton-pc ~]$ mysql -u vsearch -p vsearchlogDB
Enter password:
ERROR 1045 (28000): Access denied for user 'vsearch'@'localhost' (using password: YES)
[anton@anton-pc ~]$ mysql -u vsearch -p vsearchlogDB
Enter password: vsearchpasswd
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 6
Server version: 10.1.23-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [vsearchlogDB]>

MariaDB [vsearchlogDB]> create table log(
    -> id int auto_increment primary key,
    -> ts timestamp default current_timestamp,
    -> phrase varchar(128) not null,
    -> letters varchar(32) not null,
    -> ip varchar(16) not null,
    -> browser_string varchar(256) not null,
    -> results varchar(64) not null);
Query OK, 0 rows affected (0.53 sec)

MariaDB [vsearchlogDB]>

```

## ***Confirm Your Table Is Ready for Data***

### ***Confirme que su tabla es adecuada para los datos***

Con la tabla creada, hemos terminado con la Tarea 3.

Vamos a confirmar en la consola que la tabla de hecho se ha creado con la estructura que necesitamos. Mientras sigue conectado a la consola de MySQL como usuario vsearch, emita el comando describe log en el indicador:

```

mysql> describe log;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id | int(11) | NO | PRI | NULL | auto_increment |
| ts | timestamp | NO | | CURRENT_TIMESTAMP | |
| phrase | varchar(128) | NO | | NULL | |
| letters | varchar(32) | NO | | NULL | |
| ip | varchar(16) | NO | | NULL | |
| browser_string | varchar(256) | NO | | NULL | |
| results | varchar(64) | NO | | NULL | |
+-----+-----+-----+-----+-----+

```

Comprobamos:

```
MariaDB [vsearchlogDB]> describe log;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id | int(11) | NO | PRI | NULL | auto_increment |
| ts | timestamp | NO | | CURRENT_TIMESTAMP | |
| phrase | varchar(128) | NO | | NULL | |
| letters | varchar(32) | NO | | NULL | |
| ip | varchar(16) | NO | | NULL | |
| browser_string | varchar(256) | NO | | NULL | |
| results | varchar(64) | NO | | NULL | |
+-----+-----+-----+-----+-----+
7 rows in set (0.16 sec)
```

```
MariaDB [vsearchlogDB]>
```

Y ahí está: la prueba de que la tabla de registro existe y tiene una estructura que se ajusta a las necesidades de registro de nuestra aplicación web. Escriba quit para salir de la consola (como ya se ha hecho con ella).

```
MariaDB [vsearchlogDB]> quit
Bye
[anton@anton-pc ~]$
```

*Así que ahora estoy listo para agregar datos a la tabla ¿verdad? Mi amigo que es un experto en SQL dice que puedo usar un montón de instrucciones INSERT manual para hacer eso ..*

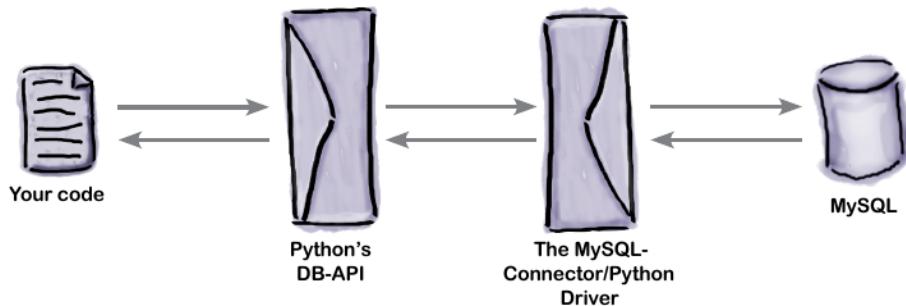
### **Sí, esa es una posibilidad.**

No hay nada que le impida escribir manualmente un montón de sentencias SQL INSERT en la consola para agregar manualmente datos a su tabla recién creada. Pero recuerde: queremos que nuestra aplicación web añada nuestros datos de solicitud web a la tabla de registro automáticamente, y esto también se aplica a las sentencias INSERT.

Para ello, necesitamos escribir algún código Python para interactuar con la tabla de registro. Y para hacer eso, necesitamos aprender más acerca de DB-API de Python.



Recuerde el diagrama de este capítulo que posicionó el DB-API de Python en relación con su código, el controlador de base de datos elegido y el sistema de base de datos subyacente:



La promesa de usar DB-API es que puedes reemplazar la combinación de controlador / base de datos con modificaciones muy pequeñas en tu código de Python, siempre y cuando te limites a usar las facilidades proporcionadas por la DB-API.

Revisemos lo que implica la programación a este importante estándar de Python. Vamos a presentar seis pasos aquí.

### ***DB-API Step 1: Define your connection characteristics***

#### ***DB-API Paso 1: Defina sus características de conexión***

Hay cuatro elementos de información que necesita cuando se conecta a MySQL: (1) la dirección IP / nombre de la computadora que ejecuta el servidor MySQL (conocido como el host), (2) el ID de usuario a utilizar, (3) la contraseña asociada Con el ID de usuario y (4) el nombre de la base de datos con la que el usuario desea interactuar.

El controlador MySQL-Connector / Python le permite poner estas características de conexión en un diccionario Python para facilidad de uso y facilidad de referencia. Hagámoslo ahora escribiendo el código en este Up Close en el prompt >>>. Asegúrese de seguir a lo largo de su computadora. Aquí hay un diccionario (llamado dbconfig) que asocia las cuatro "claves de conexión" requeridas con sus valores correspondientes:

```
>>> dbconfig = { 'host': '127.0.0.1',
                 'user': 'vsearch',
                 'password': 'vsearchpasswd',
                 'database': 'vsearchlogDB', }

1. Our server is running on our local
computer, so we use the localhost IP
address for "host".
2. The "vsearch" user ID
from earlier in this chapter is
assigned to the "user" key.
3. The "password"
key is assigned the
correct password
to use with our
user ID.
4. The database name—"vsearchlogDB" in
this case—is assigned to the "database" key.
```

Traducimos:

1. Nuestro servidor se está ejecutando en nuestro equipo local, por lo que usamos la dirección IP localhost para "host".
2. El identificador de usuario "vsearch" anterior en este capítulo se asigna a la clave "usuario".
3. La clave de "contraseña" se asigna la contraseña correcta para usar con nuestro ID de usuario.

4. El nombre de la base de datos - "vsearchlogDB" en este caso - se asigna a la clave "base de datos"

### ***DB-API Step 2: Import your database driver***

#### ***DB-API Paso 2: Importe el controlador de la base de datos***

Con las características de conexión definidas, es el momento de importar nuestro controlador de base de datos:

```
>>> import mysql.connector
```



Importar el controlador de la base de datos que está utilizando.

Esta importación hace que el controlador específico de MySQL esté disponible para la DB-API.

### ***DB-API Step 3: Establish a connection to the server***

#### ***DB-API Paso 3: Establecer una conexión con el servidor***

Vamos a establecer una conexión con el servidor mediante la función de connect de la DB-API para establecer nuestra conexión. Guardemos una referencia a la conexión en una variable llamada conn. Esta es la llamada a conectar, que establece la conexión con el servidor de base de datos MySQL (y crea conn):

```
>>> conn = mysql.connector.connect(**dbconfig)
```

This call establishes the connection.



Pass in the dictionary of connection characteristics.

Traduciendo:

1. Esta llamada establece la conexión
2. Entre en el diccionario de características de conexión.

Observe el \*\* extraño que precede al único argumento a la función de conexión. (Si eres programador C / C ++, no leas \*\* como "un puntero a un puntero", ya que Python no tiene idea de punteros.) La notación \*\* indica a la función connect que se está suministrando un diccionario de argumentos En una sola variable (en este caso dbconfig, el diccionario que acaba de crear). Al ver el \*\*, la función de conexión amplía el argumento del diccionario único en cuatro argumentos individuales, que luego se utilizan dentro de la función de conexión para establecer la conexión. (Verás más de la notación \*\* en un capítulo posterior, por ahora, solo úsala como está.)

### ***DB-API Step 4: Open a cursor***

#### ***DB-API Paso 4: Abrir un cursor***

Para enviar comandos SQL a su base de datos (a través de la conexión recién abierta), así como recibir resultados de su base de datos, necesita un cursor. Piense en un cursor como el

equivalente de la base de datos del identificador de archivo del último capítulo (que le permite comunicarse con un archivo de disco una vez que se abrió).

Crear un cursor es sencillo: lo hace llamando al método del cursor incluido con cada objeto de conexión. Al igual que con la conexión anterior, guardamos una referencia al cursor creado en una variable (que, en un ajuste salvaje de creatividad imaginativa, hemos llamado cursor):

```
>>> cursor = conn.cursor() ← Create a cursor to send  
                           commands to the server, and to  
                           receive results.
```

Ahora estamos listos para enviar comandos SQL al servidor y, esperamos, obtener algunos resultados.

Pero, antes de hacerlo, tomemos un momento para revisar los pasos completados hasta ahora. Hemos definido las características de la conexión para la base de datos, importado el módulo del conductor, creado un objeto de la conexión, y creado un cursor. No importa qué base de datos utilice, estos pasos son comunes a todas las interacciones con MySQL (sólo el cambio de las características de la conexión). Tenga esto presente mientras interactúa con sus datos a través del cursor.

Con el cursor creado y asignado a una variable, es hora de interactuar con los datos en su base de datos utilizando el lenguaje de consulta SQL.

### ***DB-API Step 5: Do the SQL thing!***

### ***DB-API Paso 5: ¡Haga lo SQL!***

La variable de cursor le permite enviar consultas SQL a MySQL, así como recuperar cualquier resultado producido por el procesamiento de MySQL de la consulta.

Como regla general, los programadores de Python en Head First Labs tienen gusto de codificar el SQL que intentan enviar al servidor de la base de datos en una cadena triple-quoted, después asignan la secuencia a una variable llamada `_SQL`. Se utiliza una cadena de triple comilla porque las consultas SQL a menudo pueden ejecutarse en varias líneas y, al utilizar una cadena de comillas triples, temporalmente se desactiva la regla "final de línea" del intérprete de Python. Usar `_SQL` como nombre de variable es una convención entre los programadores de Head First Labs para definir valores constantes en Python, pero puede usar cualquier nombre de variable (y no tiene que ser todo en mayúsculas ni prefijado dentro de un subrayado).

Comencemos preguntando a MySQL los nombres de las tablas de la base de datos a la que estamos conectados. Para ello, asigne la consulta de tablas de presentación a la variable

\_SQL y, a continuación, llame a la función cursor.execute, pasando a \_SQL como argumento:

Asigne la consulta SQL a una variable.

```
>>> _SQL = """show tables"""
>>> cursor.execute(_SQL)
```

Envíe la consulta en la variable "\_SQL" a MySQL para su ejecución.

Cuando escriba el comando cursor.execute arriba en el prompt >>, la consulta SQL se envía a su servidor MySQL, que procede a ejecutar la consulta (suponiendo que es válida y correcta SQL). Sin embargo, los resultados de la consulta no aparecen inmediatamente; Tienes que pedir para ellos.

Puede solicitar resultados utilizando uno de los tres métodos de cursor:

- **cursor.fetchone** recupera una sola fila de resultados.
- **cursor.fetchmany** recupera el número de filas que especifica.
- **cursor.fetchall** recupera todas las filas que forman los resultados.

Por ahora, vamos a usar el método cursor.fetchall para recuperar todos los resultados de la consulta anterior, asignar los resultados a una variable llamada res, luego mostrar el contenido de res en el prompt >>:

Get all the data returned from MySQL.

```
>>> res = cursor.fetchall()
>>> res
[('log',)]
```

Display the results.

El contenido de res parece un poco raro, ¿no? Probablemente esperabas ver una sola palabra aquí, como sabemos desde antes que nuestra base de datos (vsearchlogDB) contiene una sola tabla llamada log. Sin embargo, lo que devuelve **cursor.fetchall** siempre es una lista de tuplas, incluso cuando sólo hay una sola pieza de datos devueltos (como es el caso anterior). Veamos otro ejemplo que devuelve más datos de MySQL.

Nuestra siguiente consulta, describe el registro log, consulta la información sobre la tabla de registro log almacenada en la base de datos. Como veremos a continuación, la información se muestra dos veces: una vez en su forma cruda (que es un poco complicado) y luego en varias líneas. Recordemos que el resultado devuelto por **cursor.fetchall** es una lista de tuplas.

Aquí está **cursor.fetchall** en acción una vez más:

It looks a little messy, but this is a list of tuples.

```
>>> _SQL = """describe log"""
>>> cursor.execute(_SQL)
>>> res = cursor.fetchall()
>>> res
```

Take the SQL query...  
...then send it to the server...  
...and then access the results.

```
[('id', 'int(11)', 'NO', 'PRI', None, 'auto_increment'), ('ts', 'timestamp', 'NO', '', 'CURRENT_TIMESTAMP', ''), ('phrase', 'varchar(128)', 'NO', '', None, ''), ('letters', 'varchar(32)', 'NO', '', None, ''), ('ip', 'varchar(16)', 'NO', '', None, ''), ('browser_string', 'varchar(256)', 'NO', '', None, ''), ('results', 'varchar(64)', 'NO', '', None, '')]
```

Traducimos:

1. Parece un poco desordenado, pero esta es una lista de tuplas.
2. Tome la consulta SQL ...
3. ... luego enviarlo al servidor ...
4. ... y luego acceder a los resultados.

The diagram illustrates the translation of the following Python code:

```
>>> for row in res:  
     print(row)
```

Annotations explain the code's behavior:

- A bracket on the left groups the list of tuples, with the text "Each tuple from the list of tuples is now on its own line." pointing to it.
- An arrow points from the word "row" in the loop to the text "Take each row in the results...".
- An arrow points from the word "print" to the text "...and display it on its own line."

The resulting output is shown as a list of tuples:

```
('id', 'int(11)', 'NO', 'PRI', None, 'auto_increment')  
(('ts', 'timestamp', 'NO', '', 'CURRENT_TIMESTAMP', '')  
(('phrase', 'varchar(128)', 'NO', '', None, '')  
(('letters', 'varchar(32)', 'NO', '', None, '')  
(('ip', 'varchar(16)', 'NO', '', None, '')  
(('browser_string', 'varchar(256)', 'NO', '', None, '')  
(('results', 'varchar(64)', 'NO', '', None, '')
```

Traducimos:

1. Cada tupla de la lista de tuplas está ahora en su propia línea.
2. Tome cada fila en los resultados ...
3. ... y mostrarlo en su propia línea.

La visualización por fila anterior puede no parecer una gran mejora con respecto a la salida sin procesar, pero compárela con la salida mostrada por la consola de MySQL desde antes (se muestra a continuación). Lo que se muestra arriba es el mismo de los datos que se muestran a continuación, solo que ahora los datos están en una estructura de datos Python llamada res:

Comprobamos:

```
In [1]: dbconfig = {  
...: 'host':'127.0.0.1',  
...: 'user':'vsearch',  
...: 'password':'vsearchpasswd',  
...: 'database':'vsearchlogDB',}
```

```
In [2]: import mysql.connector
```

```
In [3]: conn = mysql.connector.connect(**dbconfig)
```

```
In [4]: cursor = conn.cursor()
```

```
In [5]: _SQL = """show tables"""
```

```
In [6]: cursor.execute(_SQL)
```

```
In [7]: res = cursor.fetchall()
```

```
In [8]: res
```

```
Out[8]: [('log',)]
```

```
In [9]: _SQL = """describe log"""

In [10]: cursor.execute(_SQL)
```

```
In [11]: res = cursor.fetchall()
```

```
In [12]: res
```

```
Out[12]:
```

```
[('id', 'int(11)', 'NO', 'PRI', None, 'auto_increment'),
 ('ts', 'timestamp', 'NO', "", 'CURRENT_TIMESTAMP', ""),
 ('phrase', 'varchar(128)', 'NO', "", None, ""),
 ('letters', 'varchar(32)', 'NO', "", None, ""),
 ('ip', 'varchar(16)', 'NO', "", None, ""),
 ('browser_string', 'varchar(256)', 'NO', "", None, ""),
 ('results', 'varchar(64)', 'NO', "", None, "")]
```

```
In [13]: for row in res:
```

```
...: print(row)
```

```
...:
```

```
('id', 'int(11)', 'NO', 'PRI', None, 'auto_increment')
 ('ts', 'timestamp', 'NO', "", 'CURRENT_TIMESTAMP', "")
 ('phrase', 'varchar(128)', 'NO', "", None, "")
 ('letters', 'varchar(32)', 'NO', "", None, "")
 ('ip', 'varchar(16)', 'NO', "", None, "")
 ('browser_string', 'varchar(256)', 'NO', "", None, "")
 ('results', 'varchar(64)', 'NO', "", None, "")
```

```
In [14]:
```

```
MariaDB [vsearchlogDB]> describe log;
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default      | Extra      |
+-----+-----+-----+-----+-----+
| id         | int(11)    | NO   | PRI | NULL        | auto_increment |
| ts          | timestamp  | NO   |     | CURRENT_TIMESTAMP |
| phrase      | varchar(128) | NO   |     | NULL        |
| letters     | varchar(32)  | NO   |     | NULL        |
| ip          | varchar(16)  | NO   |     | NULL        |
| browser_string | varchar(256) | NO   |     | NULL        |
| results     | varchar(64)  | NO   |     | NULL        |
+-----+-----+-----+-----+-----+
7 rows in set (0.16 sec)
```

```
MariaDB [vsearchlogDB]>
```

Mira de cerca. Son los mismos datos.

Utilicemos una consulta insert para agregar algunos datos de ejemplo a la tabla de registro o log.

Es tentador asignar la consulta mostrada a continuación (que hemos escrito en varias líneas) a la variable `_SQL`, luego llamar a `cursor.execute` para enviar la consulta al servidor:

```
>>> _SQL = """insert into log
    (phrase, letters, ip, browser_string, results)
    values
    ('hitch-hiker', 'aeiou', '127.0.0.1', 'Firefox', "{'e', 'i'}")"""
>>> cursor.execute(_SQL)
```



No nos malinterpreten, lo que se muestra arriba funciona. Sin embargo, la codificación de los valores de datos de esta manera raramente es lo que usted querrá hacer, ya que los valores de datos que almacena en su tabla probablemente cambiarán con cada inserción. Recuerde: planea registrar los detalles de cada solicitud web en la tabla de registro o log, lo que significa que estos valores de datos cambiarán con cada solicitud, por lo que codificar los datos de esta manera sería un desastre.

Para evitar la necesidad de codificar datos (como se muestra arriba), la API de DB de Python le permite posicionar los "marcadores de posición de datos" en la cadena de consulta, que se rellenan con los valores reales cuando se llama a `cursor.execute`. En efecto, esto le permite reutilizar una consulta con muchos valores de datos diferentes, pasando los valores como argumentos a la consulta justo antes de que se ejecute. Los marcadores de posición en su consulta son valores de cadena y se identifican como `%s` en el código siguiente.

Compare estos comandos con los mostrados arriba:

```
>>> _SQL = """insert into log
    (phrase, letters, ip, browser_string, results)
    values
    (%s, %s, %s, %s, %s)"""
>>> cursor.execute(_SQL, ('hitch-hiker', 'xyz', '127.0.0.1', 'Safari', 'set()))
```

Al componer su consulta, utilice marcadores de posición de DB-API en lugar de valores de datos reales.

Hay dos cosas a tener en cuenta más arriba. Primero, en lugar de codificar los valores de datos reales en la consulta SQL, usamos el marcador de posición `%s`, que le dice a DB-API que espera que un valor string sea sustituido en la consulta antes de la ejecución. Como puede ver, hay cinco `%s` marcadores de posición arriba, por lo que la segunda cosa a tener en cuenta es que el cursor. Ejecutar llamada va a esperar cinco parámetros adicionales cuando se llama. El único problema es ese `cursor.execute` no acepta cualquier número de parámetros; Acepta como máximo dos.

¿Cómo puede ser esto?

Mirando la última línea de código mostrada arriba, está claro que `cursor.execute` acepta los cinco valores de datos que se le proporcionan (sin queja), así que ¿qué da?

Tome otra mirada más cercana a esa línea de código. Ver el par de paréntesis alrededor de los valores de datos? El uso de paréntesis convierte los cinco valores de datos en una única tupla (que contiene los valores de datos individuales). En efecto, la línea de código anterior proporciona dos argumentos a `cursor.execute`: la consulta que contiene el marcador de posición, así como una única tupla de valores de datos.

Por lo tanto, cuando el código de esta página se ejecuta, los valores de datos se insertan en la tabla de registro log, ¿verdad? Bueno ... no exactamente.

Cuando utiliza `cursor.execute` para enviar datos a un sistema de base de datos (mediante la consulta `insert`), es posible que los datos no se guarden inmediatamente en la base de datos. Esto se debe a que la escritura en una base de datos es una operación costosa (desde una perspectiva de ciclo de procesamiento), tantos sistemas de base de datos almacenan insertos en caché, y luego los aplican todos de una vez más tarde. Esto a veces puede significar que los datos que crees que están en tu tabla no están todavía, lo que puede provocar problemas.

Por ejemplo, si utiliza `insertar` para enviar datos a una tabla, utilice inmediatamente `select` para volver a leerla, es posible que los datos no estén disponibles, ya que aún están en la caché del sistema de base de datos esperando a ser escritos. Si esto sucede, no tienes suerte, ya que la selección no puede devolver ningún dato. Eventualmente, los datos se escriben, por lo que no se pierde, pero este comportamiento de caché predeterminado puede no ser lo que usted desea.

Si está satisfecho de tomar el resultado de éxito asociado con una escritura de base de datos, puede forzar al sistema de base de datos a que envíe todos los datos potencialmente almacenados en caché a su tabla mediante el método `conn.commit`. Hagámoslo ahora para asegurar que las dos instrucciones de inserción de la página anterior se apliquen a la tabla de registro o log. Con sus datos escritos, ahora puede utilizar una consulta de selección para confirmar que los valores de datos se guardan:

```
"Force" any cached data to be written to the table. → >>> conn.commit()
Here's the "id" value MySQL automatically assigned to this row...
>>> _SQL = """select * from log"""
>>> cursor.execute(_SQL)
>>> for row in cursor.fetchall():
    print(row)
}
Retrieves the just-written data.
(1, datetime.datetime(2016, 3, ..., "'e', 'i'"))
(2, datetime.datetime(2016, 3, ..., 'set()'))
...and here's what it filled in for "ts" (timestamp).
We've abridged the output to make it fit on this page.
```

Traducimos:

1. "Forzar" los datos almacenados en caché que se escriban en la tabla.
2. Aquí está el valor "id" que MySQL asigna automáticamente a esta fila ...
3. ... y aquí es lo que llenó para "ts" (timestamp).
4. Recuperar los datos recién escritos.
5. Hemos resumido la salida para que encaje en esta página.

De lo anterior se puede ver que MySQL ha determinado automáticamente los valores correctos para usar para id y ts cuando los datos se insertan en una fila. Los datos devueltos desde el servidor de bases de datos son (como antes) una lista de tuplas. En lugar de guardar los resultados de cursor.fetchall en una variable que es iterada, hemos utilizado cursor.fetchall directamente en un bucle for de este código. Además, no olvide: una tupla es una lista inmutable y, como tal, soporta la notación de acceso de corchetes. Esto significa que puede indexar en la variable de fila utilizada en el bucle anterior para seleccionar elementos de datos individuales según sea necesario. Por ejemplo, la row[2] selecciona la frase, la row[3] selecciona las letras y la row[-1] selecciona los resultados.

### ***DB-API Step 6: Close your cursor and connection***

### ***DB-API Paso 6: Cierre el cursor y la conexión***

Con sus datos comprometidos o committed a su tabla, ordenar después de sí mismo mediante el cierre del cursor, así como la conexión:

```
>>> cursor.close()
True
>>> conn.close()
```

Siempre es una buena idea ordenar.

Tenga en cuenta que el cursor confirma el cierre satisfactorio devolviendo True, mientras que la conexión simplemente se cierra. Siempre es una buena idea cerrar el cursor y su conexión cuando ya no son necesarios, ya que su sistema de base de datos tiene un conjunto finito de recursos. En Head First Labs, a los programadores les gusta mantener los cursos y las conexiones de la base de datos abiertos durante el tiempo que se requiera, pero ya no.

### ***Task 4: Create Code to Work with Our Webapp's Database and Tables***

### ***Tarea 4: Crear código para trabajar con la base de datos y tablas de Webapp***

Con los seis pasos del DB-API Up Close completado, ahora tiene el código necesario para interactuar con la tabla de registro o log, lo que significa que ha completado la Tarea 4: Crear código para trabajar con la base de datos y las tablas de nuestra aplicación web.

Revisemos el código que puedes usar (en su totalidad):

```

    dbconfig = { 'host': '127.0.0.1',
                 'user': 'vsearch',
                 'password': 'vsearchpasswd',
                 'database': 'vsearchlogDB', }

    import mysql.connector ← Import the database driver.

    Establish a connection and create a cursor. → conn = mysql.connector.connect(**dbconfig)
    cursor = conn.cursor()

    _SQL = """insert into log
              (phrase, letters, ip, browser_string, results)
              values
              (%s, %s, %s, %s, %s)"""
    cursor.execute(_SQL, ('galaxy', 'xyz', '127.0.0.1', 'Opera', "{'x', 'y'}"))

    conn.commit() ← Force the database to write your data.

    _SQL = """select * from log"""
    cursor.execute(_SQL)

    for row in cursor.fetchall():
        print(row)

    cursor.close() } ← Retrieve the (just written) data from the table, displaying the output row by row.

    conn.close() } ← Tidy up when you're done.

```

Annotations:

- Define your connection characteristics.
- Import the database driver.
- Establish a connection and create a cursor.
- Assign a query to a string (note the five placeholder arguments).
- Send the query to the server, remembering to provide values for each of the required arguments (in a tuple).
- Forces the database to write your data.
- Tidy up when you're done.

Traducimos:

1. Defina sus características de conexión.
2. Establecer una conexión y crear un cursor.
3. Forzar la base de datos para escribir sus datos.
4. Arregla cuando termines.
5. Recupere los datos (recién escritos) de la tabla, mostrando la fila de salida por fila.
6. Envíe la consulta al servidor, recordando proporcionar valores para cada uno de los argumentos requeridos (en una tupla).
7. Asigne una consulta a una cadena (observe los cinco argumentos de marcador de posición).
8. Importar el controlador de la base de datos.

Con cada una de las cuatro tareas completadas, ya está listo para ajustar su aplicación web para registrar los datos de solicitud web en su sistema de base de datos MySQL en lugar de un archivo de texto (como actualmente). Comencemos a hacer esto ahora.

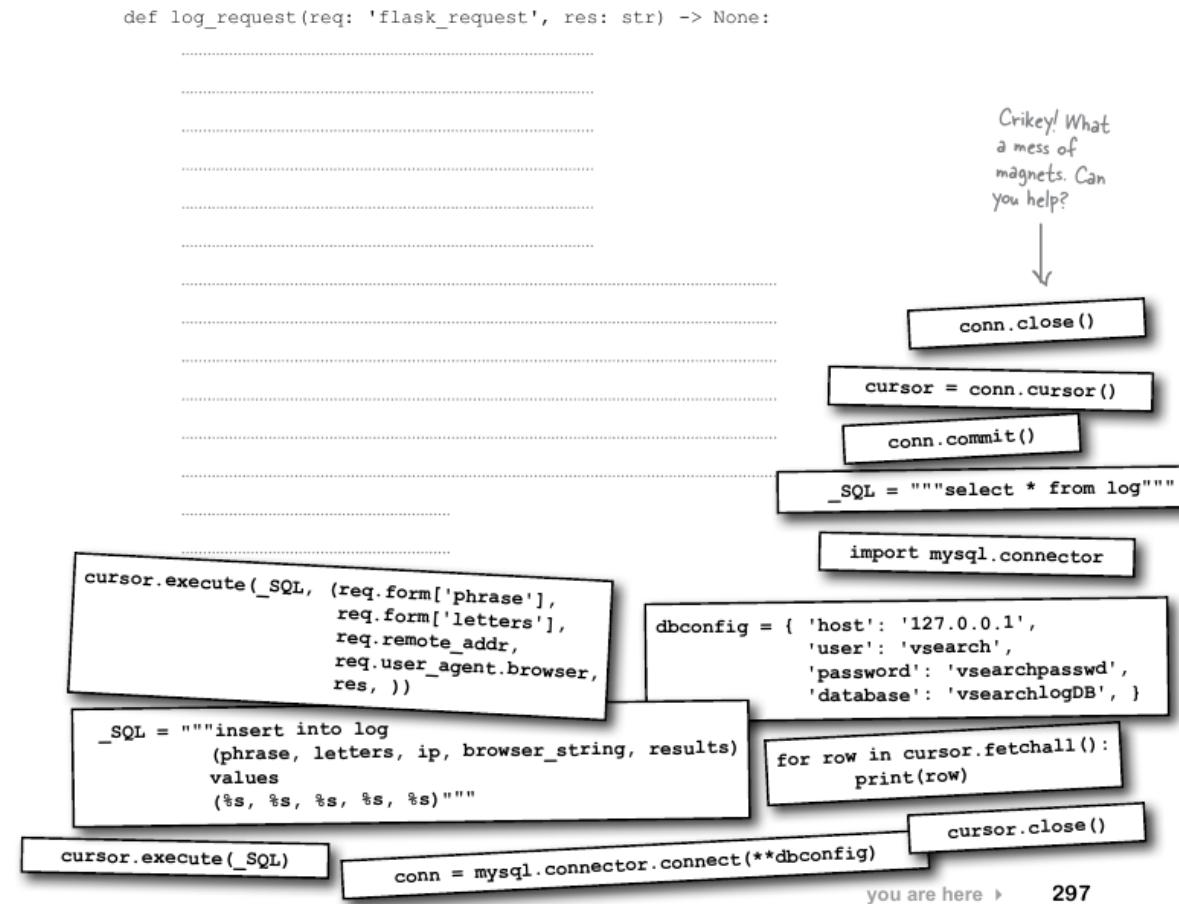
## Database Magnets

Eche un vistazo a la función `log_request` del último capítulo.

Recuerde que esta pequeña función acepta dos argumentos: un objeto de solicitud web y los resultados de la `vsearch`:

```
def log_request(req: 'flask_request', res: str) -> None:
    with open('vsearch.log', 'a') as log:
        print(req.form, req.remote_addr, req.user_agent, res, file=log, sep='|')
```

Su trabajo consiste en reemplazar el conjunto de esta función con el código que se registra en su base de datos (en contraposición al archivo de texto). La línea def se mantendrá sin cambios. Decida sobre los imanes que necesita de los dispersos en la parte inferior de esta página, a continuación, colocarlos para proporcionar el código de la función:

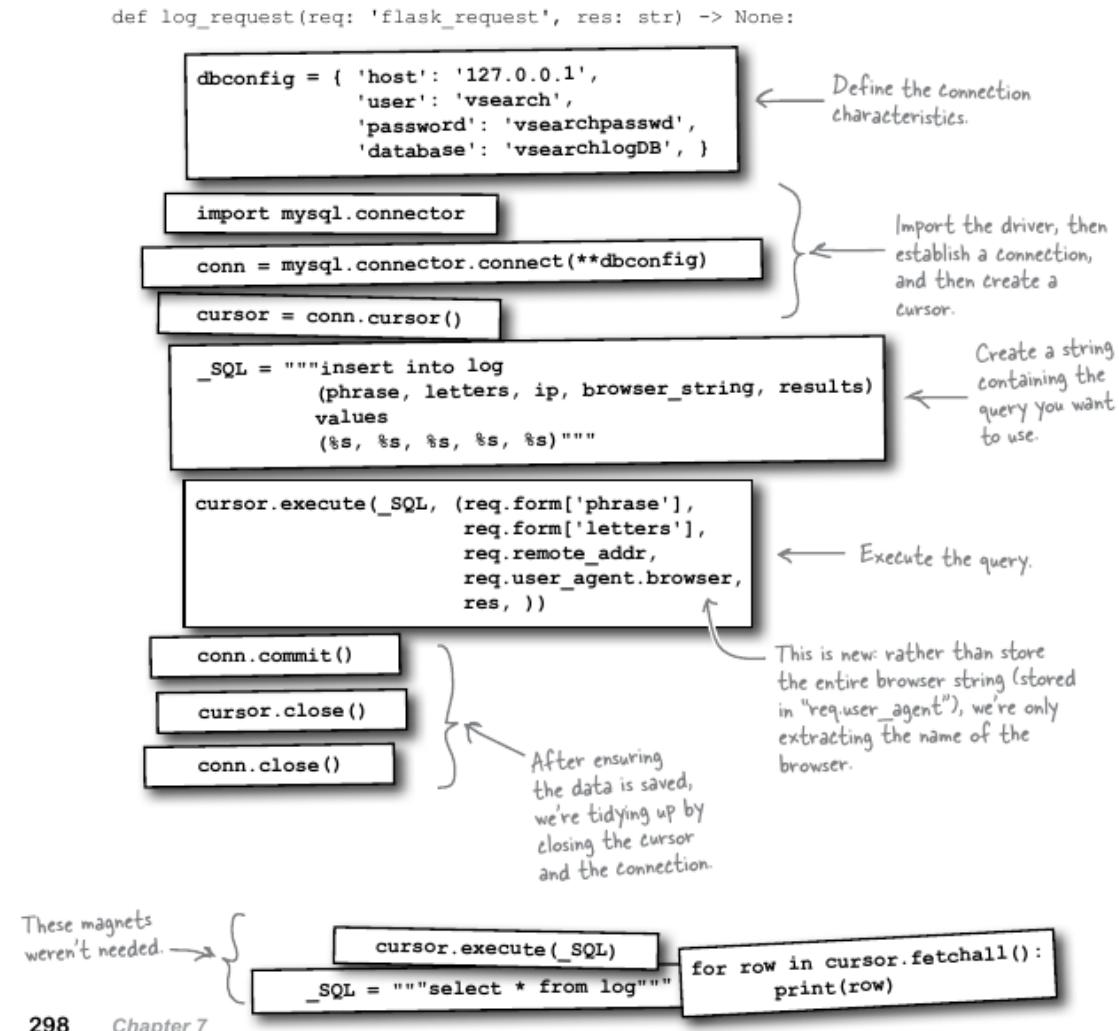


## Database Magnets Solution

Deberías echar un vistazo a la función `log_request` del último capítulo:

```
def log_request(req: 'flask_request', res: str) -> None:
    with open('vsearch.log', 'a') as log:
        print(req.form, req.remote_addr, req.user_agent, res, file=log, sep='|')
```

Tu trabajo consistía en reemplazar la suite de esta función con código que se registra en tu base de datos. La línea def se mantendría sin cambios. Deberías decidir qué imanes necesitas de los dispersos en la parte inferior de la página.



298 Chapter 7

Traducimos:

1. Defina las características de la conexión.
2. Importe el controlador, establezca una conexión y, a continuación, cree un cursor.
3. Cree una cadena que contenga la consulta que desea utilizar.
4. Ejecute la consulta.
5. Esto es nuevo: en lugar de almacenar toda la cadena del navegador (almacenada en "req.user\_agent"), sólo estamos extrayendo el nombre del navegador.
6. Después de asegurar que los datos se guardan, estamos ordenando cerrando el cursor y la conexión.
7. Estos imanes no eran necesarios.

## Test Drive

Cambie el código en su archivo `vsearch4web.py` para reemplazar el código de la función `log_request` original con el de la última página. Cuando haya guardado su código, inicie esta última versión de su aplicación web en un símbolo del sistema. Recuerda que en Windows, necesitas usar este comando:

```
C:\webapps> py -3 vsearch4web.py
```

Mientras esté en Linux o Mac OS X, utilice este comando:

```
$ python3 vsearch4web.py
```

Su webapp debe comenzar a correr en esta dirección web:

```
http://127.0.0.1:5000/
```

Utilice su navegador web favorito para realizar algunas búsquedas para confirmar que su webapp funciona bien. Hay dos puntos que queremos hacer aquí:

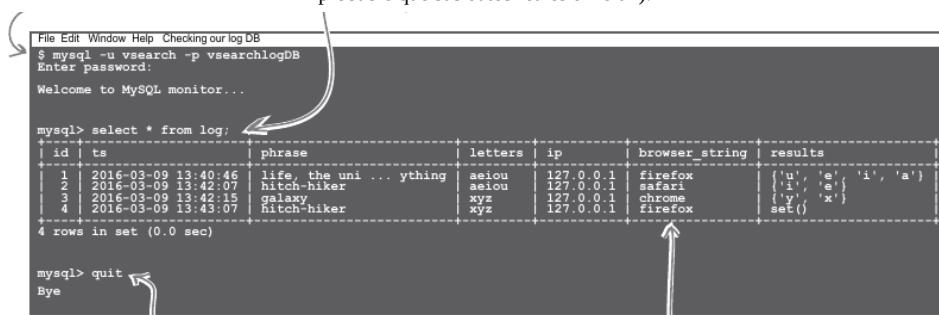
- Su aplicación web funciona exactamente igual que antes: cada búsqueda devuelve una "página de resultados" al usuario.
- Sus usuarios no tienen idea de que los datos de búsqueda se están registrando en una tabla de base de datos en lugar de un archivo de texto.

Lamentablemente, no puede utilizar la URL `/viewlog` para ver estas últimas entradas de registro, ya que la función asociada a esa URL (`view_the_log`) sólo funciona con el archivo de texto `vsearch.log` (no la base de datos). Tendremos más que decir sobre cómo arreglar esto en la página.

Por ahora, vamos a concluir este Test Drive usando la consola MySQL para confirmar que esta nueva versión de `log_request` está registrando datos en la tabla de log. Abra otra ventana de terminal y siga adelante (nota: hemos reformateado y resumido nuestra salida para que encaje en esta página):

Inicie sesión en la consola MySQL.

Esta consulta solicita ver todos los datos de la tabla "log" (es probable que sus datos reales difieran).



id	ts	phrase	letters	ip	browser_string	results
1	2016-03-09 13:40:46	life, the uni ... ything	aeiou	127.0.0.1	firefox	{'u', 'e', 'i', 'a'}
2	2016-03-09 13:42:07	hitch-hiker	aeiou	127.0.0.1	safari	{'i', 'e'}
3	2016-03-09 13:42:15	galaxy	xyz	127.0.0.1	chrome	{'y', 'x'}
4	2016-03-09 13:43:07	hitch-hiker	xyz	127.0.0.1	firefox	set()

No te olvides de salir de la consola cuando hayas terminado.

Recuerde: sólo almacenamos el nombre del navegador.

## ***Storing Data Is Only Half the Battle***

### ***Almacenamiento de datos es sólo la mitad de la batalla***

Después de ejecutar la unidad de prueba en la última página, ha confirmado que el código compatible con Python DB-API en `log_request` almacena de hecho los detalles de cada solicitud web en su tabla de registro o `log`.

Echa un vistazo a la versión más reciente de la función `log_request` una vez más (que incluye una docstring como su primera línea de código):

```
def log_request(req: 'flask_request', res: str) -> None:  
    """Log details of the web request and the results."""  
    dbconfig = { 'host': '127.0.0.1',  
                'user': 'vsearch',  
                'password': 'vsearchpasswd',  
                'database': 'vsearchlogDB', }  
  
    import mysql.connector  
  
    conn = mysql.connector.connect(**dbconfig)  
    cursor = conn.cursor()  
    _SQL = """insert into log  
        (phrase, letters, ip, browser_string, results)  
        values  
        (%s, %s, %s, %s, %s)"""  
    cursor.execute(_SQL, (req.form['phrase'],  
                        req.form['letters'],  
                        req.remote_addr,  
                        req.user_agent.browser,  
                        res, ))  
    conn.commit()  
    cursor.close()  
    conn.close()
```



Los programadores experimentados de Python pueden mirar el código de esta función y soltar un grito de desaprobación. Usted aprenderá por qué en el tiempo de pocas páginas.

## ***This new function is a big change***

### ***Esta nueva función es un gran cambio***

Hay mucho más código en la función `log_request` ahora que cuando se operaba en un archivo de texto simple, pero el código extra es necesario para interactuar con MySQL (que se va a utilizar para responder a las preguntas sobre los datos registrados al final de este Capítulo), por lo que esta nueva, más grande y más compleja versión de `log_request` parece justificada.

Sin embargo, recuerde que su aplicación web tiene otra función, llamada `view_the_log`, que recupera los datos del archivo de registro `vsearch.log` y los muestra en una página web con

formato muy bien. Ahora necesitamos actualizar el código de la función `view_the_log` para recuperar sus datos de la tabla de registro en la base de datos, en contraposición al archivo de texto.

**La pregunta es: ¿cuál es la mejor manera de hacerlo?**

***How Best to Reuse Your Database Code?***

***¿Qué mejor manera de reutilizar su código de base de datos?***

Ahora tiene un código que registra los detalles de cada una de las solicitudes de su aplicación web a MySQL. No debería ser demasiado trabajo hacer algo similar con el fin de recuperar los datos de la tabla de registro para su uso en la función `view_the_log`. La pregunta es: ¿cuál es la mejor manera de hacerlo? Le preguntamos a tres programadores nuestra pregunta ... y obtuvimos tres respuestas diferentes.

- A) Rápidamente cortar y pegar ese código, luego cambiarlo. ¡Hecho!
- B) Yo voto que ponemos que la base de datos de manejo de código en su propia función, a continuación, llamar como sea necesario.
- C) ¿No está claro que es hora de que consideramos el uso de clases y objetos como la forma correcta de manejar este tipo de reutilización?

A su manera, cada una de estas sugerencias es válida, si es un poco sospechoso (especialmente el primero). Lo que puede ser una sorpresa es que, en este caso, un programador de Python sería poco probable que aceptar cualquiera de estas soluciones propuestas por su cuenta.

**reduzca la reutilización reciclan**

***Consider What You're Trying to Reuse***

***Considere lo que está tratando de reutilizar***

Echemos un vistazo a nuestro código de base de datos en la función `log_request`.

Debe quedar claro que hay partes de esta función que podemos reutilizar al escribir código adicional que interactúa con un sistema de base de datos. Por lo tanto, hemos anotado el código de la función para resaltar las partes que creemos que son reutilizables, en contraposición a las partes que son específicos de la idea central de lo que la función `log_request` realmente hace:

```

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""
    dbconfig = { 'host': '127.0.0.1',
                 'user': 'vsearch',
                 'password': 'vsearchpasswd',
                 'database': 'vsearchlogDB', }

    import mysql.connector
    conn = mysql.connector.connect(**dbconfig)
    cursor = conn.cursor()

    _SQL = """insert into log
              (phrase, letters, ip, browser_string, results)
              values
              (%s, %s, %s, %s, %s)"""
    cursor.execute(_SQL, (req.form['phrase'],
                          req.form['letters'],
                          req.remote_addr,
                          req.user_agent.browser,
                          res,))

    conn.commit()
    cursor.close()
    conn.close()

```

These two statements are always going to be the same, so can be reused.

These three statements are also always the same, so can be reused, too.

The database connection characteristics are very specific to what we're doing here, but are likely needed in other places, so should be reusable.

This code is the real "guts" of what's going on inside the function, and can't be reused in any meaningful way (as it's way too specific to the job at hand).

Based on this simple analysis, the `log_request` function has three groups of code statements:

Traducimos:

1. Estas dos declaraciones siempre van a ser las mismas, por lo que pueden ser reutilizadas.
2. Estas tres declaraciones también son siempre las mismas, por lo que pueden ser reutilizadas, también.
3. Las características de conexión a la base de datos son muy específicas de lo que estamos haciendo aquí, pero es probable que se necesiten en otros lugares, por lo que debe ser reutilizable.
4. Este código es la verdadera "tripa" de lo que ocurre dentro de la función, y no puede ser reutilizado de ninguna manera significativa (ya que es demasiado específico para el trabajo a mano).

Basado en este simple análisis, la función `log_request` tiene tres grupos de sentencias de código:

- Declaraciones que pueden ser fácilmente reutilizadas (como la creación de `conn` y `cursor`, así como las llamadas a `commit` y `close`);
- Declaraciones que son específicas para el problema pero que aún deben ser reutilizables (como el uso del diccionario `dbconfig`); y
- Declaraciones que no pueden ser reutilizadas (como la asignación a `_SQL` y la llamada a `cursor.execute`). Cualquier otra interacción con MySQL es muy probable que requiera una consulta SQL diferente, así como diferentes argumentos (si los hubiera).

## ***What About That Import?***

### ***¿Qué pasa con esa importación?***

Toda esta charla de la reutilización es grande ... pero ¿usted olvidó considerar reutilizar esa declaración de la importación "import"?

#### **No, no lo olvidamos.**

La sentencia mysql.connector de importación no se olvidó cuando consideramos reutilizar el código de la función log\_request.

Esta omisión fue deliberada por nuestra parte, ya que quisiéramos llamar esta declaración para un trato especial. El problema no es que no queramos reutilizar esa declaración; Es que no debería aparecer en la suite de la función!

## ***Be careful when positioning your import statements***

### ***Tenga cuidado al posicionar sus declaraciones de importación***

Mencionamos algunas páginas que los programadores experimentados de Python pueden mirar el código de la función log\_request y soltar un grito de desaprobación. Esto se debe a la inclusión de la línea de código mysql.connector de importación en la suite de la función. Y esta desaprobación es a pesar de que nuestro más reciente Test Drive demostró claramente que este código funciona. ¿Entonces, cuál es el problema?

El problema tiene que ver con lo que sucede cuando el intérprete encuentra una sentencia de importación en su código: el módulo importado se lee en su totalidad y luego lo ejecuta el intérprete. Este comportamiento es muy bien cuando la instrucción de importación se produce fuera de una función, ya que el módulo importado (normalmente) sólo se lee una vez y se ejecuta una vez.

Sin embargo, cuando una declaración de importación aparece dentro de una función, se lee y se ejecuta cada vez que se llama a la función. Esto se considera una práctica extremadamente derrochadora (aunque, como hemos visto, el intérprete no le impedirá poner una declaración de importación en una función). Nuestro consejo es simple: piense cuidadosamente sobre dónde coloca sus declaraciones de importación y no coloque ninguna dentro de una función.



## Configurar, hacer, desmantelar - setup, do, teardown

### *Consider What You're Trying to Do*

### *Considere lo que usted está tratando de hacer*

Además de mirar el código en log\_request desde una perspectiva de reutilización, también es posible categorizar el código de la función basada en cuándo se ejecuta.

La "trípia" de la función es la asignación a la variable \_SQL y la llamada a cursor.execute. Esas dos declaraciones representan lo más patentemente lo que la función está destinada a hacer, lo cual -para ser honesto- es el bit más importante. Las sentencias iniciales de la función definen las características de la conexión (en dbconfig), luego crean una conexión y un cursor. Este código de configuración siempre tiene que ejecutarse antes de las entrañas de la función. Las tres últimas declaraciones de la función (el único commit y los dos closes) se ejecutan después de las entrañas de la función. Este es el código de desmontaje, que realiza cualquier arreglo necesario.

Con esta configuración, hacer, patrón de desmontaje en mente, vamos a ver la función una vez más. Tenga en cuenta que hemos reposicionado la sentencia de importación para ejecutar fuera de la suite de la función log\_request (para evitar cualquier juramento de desaprobación):

```
import mysql.connector ← This is a better place for any import statements  
                      (that is, outside the function's suite).  
  
def log_request(req: 'flask_request', res: str) -> None:  
    """Log details of the web request and the results."""  
  
    dbconfig = { 'host': '127.0.0.1',  
                'user': 'vsearch',  
                'password': 'vsearchpasswd',  
                'database': 'vsearchlogDB', } ← This is the setup code,  
                                         which runs before  
                                         the function does its  
                                         thing.  
  
    conn = mysql.connector.connect(**dbconfig)  
    cursor = conn.cursor()  
  
    _SQL = """insert into log  
            (phrase, letters, ip, browser_string, results)  
            values  
            (%s, %s, %s, %s, %s)"""  
    cursor.execute(_SQL, (req.form['phrase'],  
                         req.form['letters'],  
                         req.remote_addr,  
                         req.user_agent.browser,  
                         res, ))  
  
    conn.commit()  
    cursor.close()  
    conn.close() ← This is the teardown code, which  
                  runs after the function has  
                  done its thing.
```

Traducimos:

1. Este código es lo que la función \* en realidad \* hace: registra una solicitud web en la base de datos.
2. Este es un lugar mejor para cualquier declaración de importación (es decir, fuera de la suite de funciones).
3. Este es el código de configuración, que se ejecuta antes de que la función haga su cosa.
4. Este es el código de desmontaje, que se ejecuta después de que la función ha hecho su cosa.

*¿No sería genial si hubiera una manera de reutilizar esta configuración, hacer, patrón de desmontaje?*

## You've Seen This Pattern Before

### Has visto este patrón antes

Considere el patrón que acabamos de identificar: el código de la disposición para conseguir listo, seguido por el código para hacer lo que necesita ser hecho, y después el código del desmontaje para ordenar para arriba. Puede que no sea inmediatamente obvio, pero en el capítulo anterior encontró un código que se ajusta a este patrón. Aquí está de nuevo:

```
Open the file.           ↘           Asigne el flujo de archivos a una
with open('todos.txt') as tasks:
    for chore in tasks:
        print(chore, end=' ')
}                         ↙           variable.
                                ← Realizar algún procesamiento.
```

Recuerde cómo la instrucción `with` gestiona el contexto en el que se ejecuta el código de su suite. Cuando está trabajando con archivos (como en el código anterior), la instrucción `with` organiza para abrir el archivo con nombre y devolver una variable que representa el flujo de archivos. En este ejemplo, esa es la variable `tasks`; Este es el código de configuración. La suite asociada con la instrucción `with` es el código `do`; Aquí que es el bucle `for`, que hace el trabajo real (a.k.a. "el bit importante"). Por último, cuando se utiliza `with` para abrir un archivo, viene con la promesa de que el archivo abierto se cerrará cuando termine el conjunto. Este es el código de desmontaje.

Sería genial si pudiéramos integrar nuestro código de programación de bases de datos en la sentencia `with`. Idealmente, sería genial si pudiéramos escribir código como este, y tener la declaración con el cuidado de toda la configuración de la base de datos y detalles de desmontaje:

```

    dbconfig = { 'host': '127.0.0.1',
                 'user': 'vsearch',
                 'password': 'vsearchpasswd',
                 'database': 'vsearchlogDB', }

    with UseDatabase(dbconfig) as cursor:
        _SQL = """insert into log
                  (phrase, letters, ip, browser_string, results)
                  values
                  (%s, %s, %s, %s, %s)"""
        cursor.execute(_SQL, (req.form['phrase'],
                             req.form['letters'],
                             req.remote_addr,
                             req.user_agent.browser,
                             res, ))

```

We still need to define the connection characteristics.

The "do code" from the last page remains unchanged.

This "with" statement works with databases as opposed to disk files, and returns a cursor for us to work with.

Don't try to run this code, as you've yet to write the "UseDatabase" context manager.

Traducimos:

1. Todavía tenemos que definir las características de la conexión.
2. El "código" de la última página permanece sin cambios.
3. Esta sentencia "with" funciona con la base de datos en lugar de con los archivos de disco, y devuelve un cursor para que trabajemos con with.
4. No intente ejecutar este código, ya que todavía tiene que escribir el gestor de contexto "UseDatabase".

La buena noticia es que Python proporciona el protocolo de gestión de contexto, que permite a los programadores conectarse a la sentencia with, según sea necesario. Lo que nos lleva a las malas noticias ...

## ***The Bad News Isn't Really All That Bad***

### ***Las malas noticias no son realmente tan malas***

En la parte inferior de la última página, declaramos que la buena noticia es que Python proporciona un protocolo de gestión de contexto que permite a los programadores conectarse a la sentencia con cuando sea necesario. Si aprende a hacer esto, puede crear un gestor de contexto llamado `UseDatabase`, que puede utilizarse como parte de una sentencia with para hablar con su base de datos.

La idea es que la configuración y desmontaje "boilerplate" código que acaba de escribir para guardar los datos de registro de su webapp a una base de datos se puede reemplazar por un solo with la declaración que se parece a esto:

```

...
with UseDatabase(dbconfig) as cursor:
...

```

Esta sentencia "with" es similar a la que se utiliza con los archivos y el "open" BIF, excepto que éste funciona con una base de datos en su lugar.

La mala noticia es que la creación de un administrador de contexto es complicado por el hecho de que usted necesita saber cómo crear una clase de Python con el fin de conectar con éxito en el protocolo.

Consideré que hasta este punto en este libro, usted ha logrado escribir un montón de código utilizable sin tener que crear una clase, lo que es bastante bueno, especialmente cuando se considera que algunos lenguajes de programación no le permiten hacer nada sin Primero la creación de una clase (estamos mirando a usted, Java).

Sin embargo, ahora es el momento de morder la bala (aunque, para ser honesto, la creación de una clase en Python no es nada de miedo).

Como la capacidad de crear una clase es generalmente útil, vamos a desviarnos de nuestra discusión actual sobre la adición de código de base de datos a nuestra aplicación web, y dedicar el siguiente capítulo (corto) a las clases. Te mostraremos lo suficiente para que puedas crear el gestor de contexto UseDatabase. Una vez hecho esto, en el capítulo siguiente, volveremos a nuestro código de base de datos (y nuestra aplicación web) y pondremos nuestras capacidades de escritura de clase recién adquiridas para trabajar escribiendo el gestor de contexto UseDatabase.

## Chapter 7's Code

Este es el código de la base de datos que se ejecuta actualmente en su aplicación web (es decir, la función "log\_request").

```
import mysql.connector

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""

    dbconfig = { 'host': '127.0.0.1',
                 'user': 'vsearch',
                 'password': 'vsearchpasswd',
                 'database': 'vsearchlogDB', }

    conn = mysql.connector.connect(**dbconfig)
    cursor = conn.cursor()

    _SQL = """insert into log
              (phrase, letters, ip, browser_string, results)
              values
              (%s, %s, %s, %s, %s)"""
    cursor.execute(_SQL, (req.form['phrase'],
                         req.form['letters'],
                         req.remote_addr,
                         req.user_agent.browser,
                         res, ))
    conn.commit()
    cursor.close()
    conn.close()
```

```

dbconfig = { 'host': '127.0.0.1',
             'user': 'vsearch',
             'password': 'vsearchpasswd',
             'database': 'vsearchlogDB', }

with UseDatabase(dbconfig) as cursor:
    SQL = """insert into log
              (phrase, letters, ip, browser_string, results)
              values
              (%s, %s, %s, %s, %s)"""
    cursor.execute(_SQL, (req.form['phrase'],
                          req.form['letters'],
                          req.remote_addr,
                          req.user_agent.browser,
                          res, ))

```

Este es el código que nos gustaría poder escribir para hacer lo mismo que nuestro código actual (reemplazando la función "log\_request" de la suite). Pero no intente ejecutar este código todavía, ya que no lo hará Trabajo sin el gestor de contexto "UseDatabase".

El código está en:

/home/anton/Documentos/Estudio2017/documentosODT/extrasPython/FirstHead/First\_Codes/ch07/webapp/

```

#vsearch4web.py
from flask import Flask, render_template, request, escape
from vsearch import search4letters

from DBCM import UseDatabase

app = Flask(__name__)

app.config['dbconfig'] = {'host': '127.0.0.1',
                         'user': 'vsearch',
                         'password': 'vsearchpasswd',
                         'database': 'vsearchlogDB', }

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""

    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """insert into log
                  (phrase, letters, ip, browser_string, results)
                  values
                  (%s, %s, %s, %s, %s)"""
        cursor.execute(_SQL, (req.form['phrase'],
                              req.form['letters'],
                              req.remote_addr,
                              req.user_agent.browser,
                              res, ))

```

```
(phrase, letters, ip, browser_string, results)

values

(%s, %s, %s, %s, %s)"""

cursor.execute(_SQL, (req.form['phrase'],

                    req.form['letters'],

                    req.remote_addr,

                    req.user_agent.browser,

                    res, ))
```

```
@app.route('/search4', methods=['POST'])

def do_search() -> 'html':

    """Extract the posted data; perform the search; return results."""

    phrase = request.form['phrase']

    letters = request.form['letters']

    title = 'Here are your results:'

    results = str(search4letters(phrase, letters))

    log_request(request, results)

    return render_template('results.html',

                           the_title=title,

                           the_phrase=phrase,

                           the_letters=letters,

                           the_results=results,)

@app.route('/')

@app.route('/entry')

def entry_page() -> 'html':

    """Display this webapp's HTML form."""

    return render_template('entry.html',

                           the_title='Welcome to search4letters on the web!')
```

```

@app.route('/viewlog')

def view_the_log() -> 'html':

    """Display the contents of the log file as a HTML table."""

    with UseDatabase(app.config['dbconfig']) as cursor:

        _SQL = """select phrase, letters, ip, browser_string, results
                  from log"""

        cursor.execute(_SQL)

        contents = cursor.fetchall()

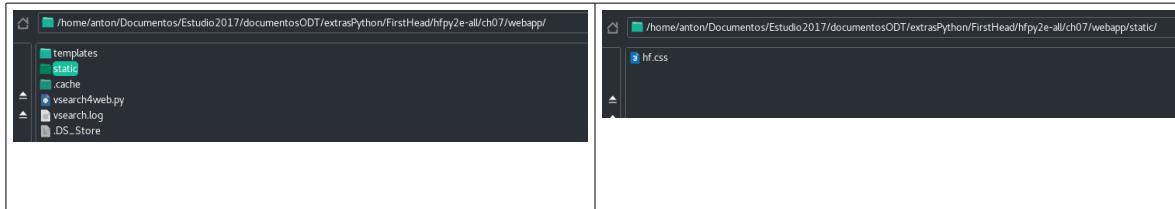
        titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')

    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,)

if __name__ == '__main__':
    app.run(debug=True)

```

## El css:



```

/*hf.css*/
body {
    font-family: Verdana, Geneva, Arial, sans-serif;
    font-size: medium;
    background-color: tan;
    margin-top: 5%;
    margin-bottom: 5%;
}

```

```
margin-left:      10%;  
margin-right:     10%;  
padding:         10px 10px 10px 10px;  
}  
  
a {  
text-decoration: none;  
font-weight:     600;  
}  
  
a:hover {  
text-decoration: underline;  
}  
  
a img {  
border:          0;  
}  
  
h2 {  
font-size:       150%;  
}  
  
table {  
margin-left:     20px;  
margin-right:    20px;  
caption-side:    bottom;  
border-collapse: collapse;  
}  
  
td, th {  
padding:         5px;  
text-align:      left;  
}  
  
.copyright {  
font-size:       75%;  
font-style:      italic;  
}
```

```
.slogan {
    font-size:      75%;
    font-style:     italic;
}

.confirmtry {
    font-weight:    600;
}

/* *** Tables ***/

table {
    font-size:      1em;
    background-color: #fafcff;
    border:          1px solid #909090;
    color:          #2a2a2a;
    padding:        5px 5px 2px;
    border-collapse: collapse;
}

td, th {
    border:          thin dotted gray;
}

/* *** Inputs ***/

input[type=text] {
    font-size:      115%;
    width:          30em;
}

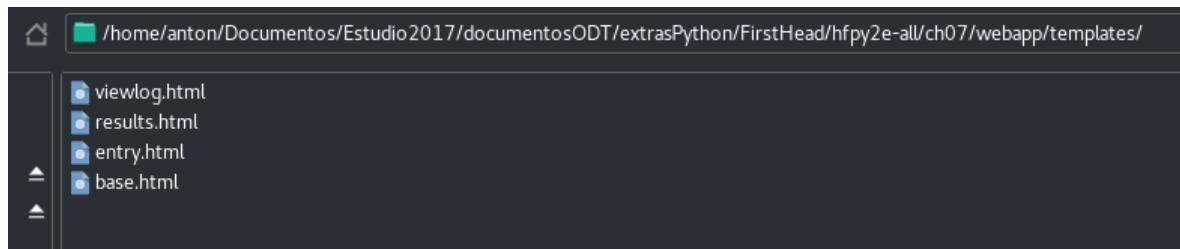
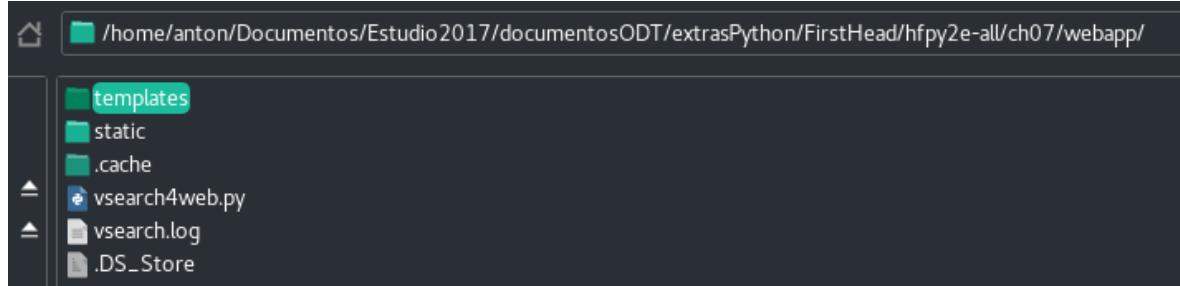
input[type=submit] {
    font-size:      125%;
}
```

```

select {
    font-size: 125%;
}

```

Ahora tenemos las plantillas html:



El base.html:

```

<!--base.html-->
<!doctype html>

<html>
    <head>
        <title>{{ the_title }}</title>
        <link rel="stylesheet" href="static/hf.css" />
    </head>
    <body>
        {% block body %}
        {% endblock %}
    </body>
</html>

```

El entry.html

```
<!--entry.html-->
{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>

<form method='POST' action='/search4'>

<table>

<p>Use this form to submit a search request:</p>

<tr><td>Phrase:</td><td><input name='phrase' type='TEXT' width='60'></td></tr>

<tr><td>Letters:</td><td><input name='letters' type='TEXT' value='aeiou'></td></tr>

</table>

<p>When you're ready, click this button:</p>

<p><input value='Do it!' type='SUBMIT'></p>

</form>

{% endblock %}
```

El results.html:

```
<!--results.html-->
{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>

<p>You submitted the following data:</p>

<table>

<tr><td>Phrase:</td><td>{{ the_phrase }}</td></tr>

<tr><td>Letters:</td><td>{{ the_letters }}</td></tr>

</table>
```

```

<p>When "{{the_phrase}}" is searched for "{{ the_letters }}", the following
results are returned:</p>

<h3>{{ the_results }}</h3>

{% endblock %}

```

*El viewlog.html:*

```

<!--viewlog.html-->
{% extends 'base.html' %}

{% block body %}

<h2>{{ the_title }}</h2>



||
||
||



{% endblock %}

```

## **8** a little bit of class

# **Abstracting Behavior and State**



*Un poco de clase*

**Resumen de comportamiento y estado**

*Bueno ... solo mira aquí: todo mi estado, y todo tu comportamiento ...*

*... y todo está en un lugar. ¡De lujo que!*

**Las clases te permiten agrupar el comportamiento del código y el estado juntos.**

En este capítulo, establecerá su aplicación web a un lado mientras aprende sobre la creación de clases de Python. Estás haciendo esto para llegar al punto en el que puedes crear un gestor de contexto con la ayuda de una clase Python. Como la creación y el uso de clases es algo tan útil para saber de todos modos, le estamos dedicando este capítulo. No cubriremos todo lo relacionado con las clases, pero tocaremos todos los elementos que necesitará comprender para crear con confianza el gestor de contexto que está esperando su aplicación web. Vamos a bucear y ver lo que está involucrado.

**Hooking into the “with” Statement**

**Enganchar en la declaración “with”**

Al decirlo al final del último capítulo, entender cómo conectar su código de configuración y desconexión a Python con una sentencia es sencillo ... suponiendo que sepa cómo crear una clase Python.

A pesar de estar más de la mitad de este libro, has conseguido pasar sin tener que definir una clase. Has escrito código útil y reutilizable usando nada más que la maquinaria de funciones de Python. Hay otras maneras de escribir y organizar su código, y la orientación de objetos es muy popular.

Nunca se te obliga a programar exclusivamente en el paradigma orientado a objetos cuando usas Python, y el lenguaje es flexible cuando se trata de cómo vas escribiendo tu código. Pero, cuando se trata de enganchar a la declaración with, hacerlo a través de una clase es el enfoque recomendado, a pesar de que la biblioteca estándar viene con soporte para hacer

algo similar sin una clase (aunque el enfoque de la biblioteca estándar es menos aplicable, No van a utilizarlo aquí).

Por lo tanto, para conectar con la declaración con, tendrá que crear una clase. Una vez que sepa cómo escribir clases, puede crear una que implemente y se adhiera al protocolo de administración de contexto. Este protocolo es el mecanismo (construido en Python) que se conecta con la sentencia with.

***El protocolo de gestión de contexto le permite escribir una clase que se conecta con la instrucción "with".***

Aprendamos cómo crear y usar clases en Python, antes de volver a nuestra discusión sobre el protocolo de gestión de contexto en el próximo capítulo.

## *there are no Dumb Questions*

**P:** Exactamente qué tipo de lenguaje de programación es Python: orientado a objetos, funcional o procedural?

**R:** Esa es una gran pregunta, que muchos programadores moviéndose a Python finalmente preguntan. La respuesta es que Python apoya los paradigmas de programación tomados de los tres enfoques populares, y Python anima a los programadores a mezclar y combinar según sea necesario. Este concepto puede ser difícil de obtener la cabeza, sobre todo si viene desde la perspectiva en la que todo el código que escribe tiene que ser en una clase que instanciar objetos de (como en otros lenguajes de programación como, por ejemplo, Java).

Nuestro consejo es no dejar que esto te preocupe: crea código en cualquier paradigma que te resulte cómodo, pero no descartes a los demás simplemente porque -como enfoques- te parecen ajenos.

**P:** Entonces ... ¿está mal comenzar siempre creando una clase?

**R:** No, no lo es, si eso es lo que su aplicación necesita. Usted no tiene que poner todo su código en las clases, pero si lo desea, Python no se interpondrá en su camino.

Hasta ahora en este libro, nos hemos pasado sin tener que crear una clase, pero ahora estamos en el punto en el que tiene sentido usar uno para resolver un problema de aplicación específico con el que estamos lidiando: la mejor manera de compartir nuestro Código de procesamiento de bases de datos dentro de nuestra webapp. Estamos mezclando y combinando paradigmas de programación para resolver nuestro problema actual, y eso está bien.

## *An Object-Oriented Primer*

### **Una guía orientada a objetos**

Antes de empezar con las clases, es importante tener en cuenta que no tenemos la intención de cubrir todo lo que hay que saber acerca de las clases en Python en este capítulo. Nuestra intención es simplemente mostrarle lo suficiente para que pueda crear con confianza una clase que implementa el protocolo de gestión de contexto.

Por lo tanto, no vamos a discutir algunos temas que los experimentados profesionales de la programación orientada a objetos (OOP) podría esperar ver aquí, como la herencia y el polimorfismo (aunque Python proporciona soporte para ambos). Eso es porque estamos interesados principalmente en la encapsulación al crear un gestor de contexto.

Si la jerga en ese último párrafo le ha puesto en pánico ciego, no se preocupe: usted puede leer con seguridad sin saber lo que cualquier de ese OOP hablan realmente medios.

En la última página, aprendiste que necesitas crear una clase para engancharte a la sentencia `with`. Antes de llegar a las especificaciones de cómo hacerlo, veamos lo que constituye una clase en Python, escribiendo una clase de ejemplo a medida que avanzamos. Una vez que usted entienda cómo escribir una clase, volveremos al problema de enganchar en la declaración `with` (en el capítulo siguiente).

### **A class bundles behavior and state**

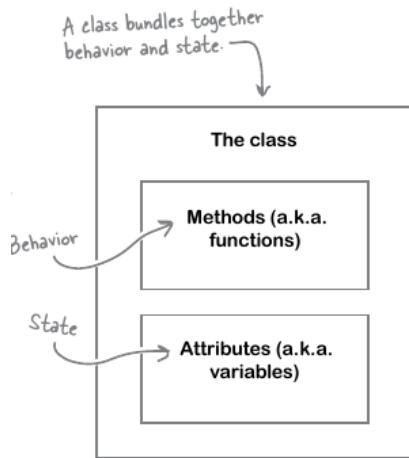
### **Una clase agrupa el comportamiento y el estado**

El uso de una clase permite agrupar el comportamiento y el estado juntos en un objeto.

Cuando escuchas el comportamiento de la palabra, piensa que la función, es decir, un trozo de código que hace algo (o implementa un comportamiento, si lo prefieres).

Cuando escucha la palabra state, piense en variables, es decir, en un lugar para almacenar valores dentro de una clase. Cuando afirmamos que una clase agrupa el comportamiento y el estado juntos, simplemente estamos afirmando que una clase empaqueta funciones y variables.

El resultado de todo lo anterior es esto: si sabes qué es una función y cuáles son las variables, lo más importante es entender qué es una clase (así como cómo crear una).



### ***Classes have methods and attributes***

### ***Las clases tienen métodos y atributos***

En Python, se define un comportamiento de clase mediante la creación de un método.

La palabra método es el nombre OOP dado a una función que se define dentro de una clase. El hecho de que los métodos no se conozcan simplemente como funciones de clase se ha perdido en las nieblas del tiempo, al igual que el hecho de que las variables de clase no se denominen como tales, son conocidas por el atributo name.

### ***Creating Objects from Classes***

### ***Creación de objetos a partir de clases***

Para usar una clase, se crea un objeto a partir de ella (verá un ejemplo de esto a continuación). Esto se conoce como instantiación de objetos. Cuando escuchas la palabra instanciar, piensa invocar; Es decir, invoca una clase para crear un objeto.

Tal vez sorprendentemente, puede crear una clase que no tiene estado o comportamiento, pero sigue siendo una clase en lo que respecta a Python. En efecto, tal clase está vacía. Comencemos nuestros ejemplos de clase con uno vacío y tomemos las cosas de allí. Trabajaremos en el prompt >>> del intérprete, y le animamos a seguir adelante.

Comenzamos por crear una clase vacía llamada `CountFromBy`. Hacemos esto prefijando el nombre de la clase con la palabra clave de la clase, luego proporcionando el conjunto de código que implementa la clase (después de los dos puntos obligatorios):

```
>>> class CountFromBy:
    pass
```

Annotations:

- "Classes start with the "class" keyword." points to the word "class".
- "Here's the class suite." points to the word "pass".
- "The name of the class" points to the identifier "CountFromBy".
- "Don't forget the colon." points to the colon character ":".

Observe cómo el paquete de esta clase contiene el pass de palabras clave de Python, que es la sentencia vacía de Python (en que no hace nada). Puede utilizar el pass en cualquier lugar que el intérprete espera encontrar el código real. En este caso, no estamos listos para llenar los detalles de la clase CountFromBy, por lo que usamos pass para evitar cualquier error de sintaxis que normalmente resultaría cuando tratamos de crear una clase sin ningún código en su suite.

Ahora que la clase existe, vamos a crear dos objetos de ella, una llamada `a` y otra llamada `b`. Tenga en cuenta cómo la creación de un objeto de una clase se parece mucho a la llamada a una función:

```
>>> a = CountFromBy()
>>> b = CountFromBy()
```

Annotations:

- "These look like function calls, don't they?" points to the closing parenthesis of the first line.
- "Create an object by appending parentheses to the class name, then assign the newly created object to a variable." points to the closing parenthesis of the second line.

Traduciendo:

1. Parecen llamadas de funciones, ¿no?
2. Cree un objeto añadiendo paréntesis al nombre de la clase y, a continuación, asigne el objeto recién creado a una variable.

## *there are no* Dumb Questions

**P:** Cuando estoy viendo el código de otra persona, ¿cómo puedo saber si algo como `CountFromBy ()` es un código que crea un objeto o código que llama a una función? Eso parece una llamada de función para mí ...

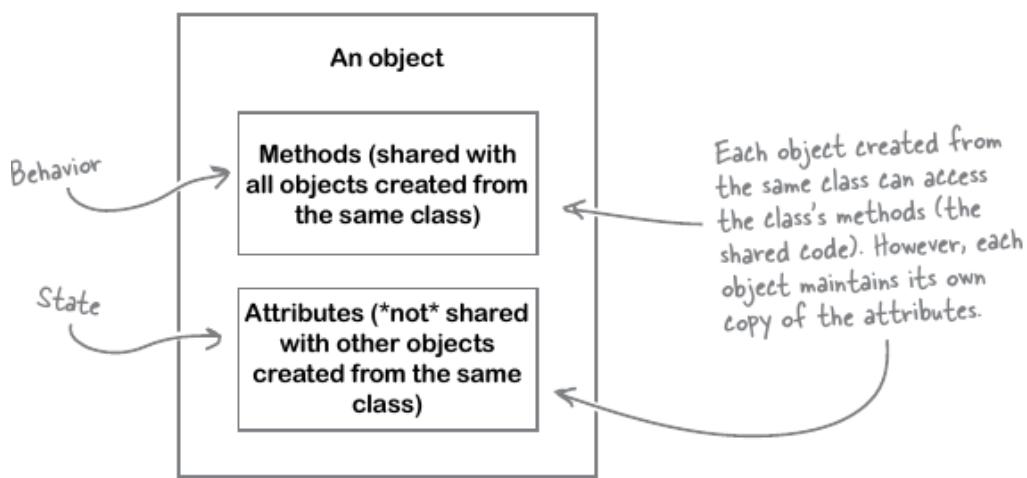
**R:** Es una gran pregunta. En la cara de las cosas, usted no sabe. Sin embargo, hay una convención bien establecida en la comunidad de programación de Python para nombrar funciones usando letras minúsculas (con subrayados para énfasis), mientras que CamelCase (palabras concatenadas, mayúsculas) se utiliza para nombrar clases. Siguiendo esta convención, debe estar claro que `count_from_by ()` es una llamada de función, mientras que `CountFromBy ()` crea un objeto. Todo está bien siempre y cuando todo el mundo siga esta

convención, y se le anima encarecidamente a hacerlo también. Sin embargo, si ignoras esta sugerencia, todas las apuestas están desactivadas y la mayoría de los programadores de Python probablemente evitarán tu código.

### ***Objects Share Behavior but Not State***

#### ***Los objetos comparten comportamiento pero no declaran***

Cuando crea objetos de una clase, cada objeto comparte los comportamientos codificados de la clase (los métodos definidos en la clase), pero mantiene su propia copia de cualquier estado (los atributos):



traduciendo:

1. Comportamiento
2. Estado
3. Cada objeto creado a partir de la misma clase puede acceder a los métodos de la clase (el código compartido).  
Sin embargo, cada objeto mantiene su propia copia de los atributos.

### ***Defining what we want CountFromBy to do***

#### ***Definiendo lo que queremos que CountFromBy haga***

Vamos a definir ahora lo que queremos que la clase CountFromBy realice (ya que una clase vacía rara vez es útil).

Hagamos CountFromBy un contador de incremento. De forma predeterminada, el contador comenzará en 0 y se incrementará (a petición) en 1. También nos permitirá proporcionar un valor de inicio y / o cantidad alternativos para incrementar por. Esto significa que podrá crear, por ejemplo, un objeto CountFromBy que comience en 100 e incremente en 10.

Vamos a ver lo que la clase CountFromBy será capaz de hacer (una vez que hemos escrito su código). Al comprender cómo se utilizará la clase, estarás mejor preparado para entender el código CountFromBy a medida que lo escribamos. Nuestro primer ejemplo usa los valores por defecto de la clase: comienza en 0, e incrementa 1 por solicitud llamando al

método de incremento. El objeto recién creado se asigna a una nueva variable, que hemos llamado c:

```
>>> c = CountFromBy()
The starting value is 0.          Create another new object, and assign it
>>> c                           to an object called "c".
0
>>> c.increase()
>>> c.increase()
>>> c.increase() }             Invoke the "increase" method
                               to increment the value of the
                               counter by one each time.
>>> c
3                                After the three calls to the "increase" method,
                               the value of the object is now three.
```

Traduciendo:

1. El valor inicial es 0.
2. Cree otro objeto nuevo y asignelo a un objeto llamado "c".
3. invoque el método de "aumento" para incrementar el valor del contador por uno cada vez.
4. Después de las tres llamadas al método "increase", el valor del objeto es ahora tres.

## ***Doing More with CountFromBy***

### ***Haciendo más con CountFromBy***

El uso de ejemplo de CountFromBy en la parte inferior de la última página demostró el comportamiento predeterminado: a menos que se especifique, el contador mantenido por un objeto CountFromBy comienza en 0 y se incrementa en 1. También es posible especificar un valor de inicio alternativo, como se demuestra en este Siguiente ejemplo, donde el recuento comienza a partir de 100:

```
The starting value is 100.           When creating this new
>>> d = CountFromBy(100)          object, specify the starting
>>> d                           value.
100
>>> d.increase()
>>> d.increase()
>>> d.increase() }             Invoke the "increase" method
                               to increment the value of
                               the counter by one each time.
>>> d
103                             After the three calls to the "increase" method,
                               the value of the "d" object is now 103.
```

Además de especificar el valor de partida, también es posible especificar la cantidad a incrementar, como se muestra aquí, donde comenzamos a 100 e incrementamos 10:

```
"e" starts at 100, and ends up at 130.   >>> e = CountFromBy(100, 10) ← Specifies both the starting
                                         value as well as the amount
                                         to increment by
                                         ↑
                                         100
                                         ↑
                                         >>> for i in range(3):
                                         >>>     e.increase() }   Invoke the "increase" method three
                                         times within a "for" loop, incrementing
                                         the value of "e" by 10 each time.
                                         ↑
                                         >>> e
                                         130
```

Traducimos:

1. Especifica tanto el valor inicial como la cantidad a incrementar por
2. Invoque el método de "aumento" tres veces dentro de un bucle "for", incrementando el valor de "e" por 10 cada vez.

En este último ejemplo, el contador comienza en 0 (el valor predeterminado), pero incrementa en 15. En lugar de tener que especificar (0, 15) como argumentos para la clase, este ejemplo utiliza un argumento de palabra clave que nos permite especificar la cantidad Para incrementar, dejando el valor inicial en el valor por defecto (0):

The diagram shows a block of Python code with handwritten annotations explaining its behavior. The code defines an object `f` with an initial value of 0, increments it by 15 three times using a `for` loop and `f.increase()`, and finally prints the value 45. Handwritten notes explain: "'f' starts at 0, and ends up at 45." A curly brace groups the line `f.increase()` and the note "As before, call 'increase' three times." Another note specifies "Specifies the amount to increment by".

```
>>> f = CountFromBy(increment=15)
>>> f
0
>>> for j in range(3):
...     f.increase()
...
>>> f
45
```

**It's Worth Repeating Ourselves: Objects Share Behavior but Not State**

**Vale la pena repetirnos: Los objetos comparten el comportamiento pero no lo hacen**

Los ejemplos anteriores crearon cuatro nuevos objetos `CountFromBy`: `c`, `d`, `e`, `f`, cada uno de los cuales tiene acceso al método de aumento, que es un comportamiento que es compartido por todos los objetos creados a partir de la clase `CountFromBy`. Sólo hay una copia del código del método de aumento, que todos estos objetos utilizan. Sin embargo, cada objeto mantiene sus propios valores de atributo. En estos ejemplos, ese es el valor actual del contador, que es diferente para cada uno de los objetos, como se muestra aquí:

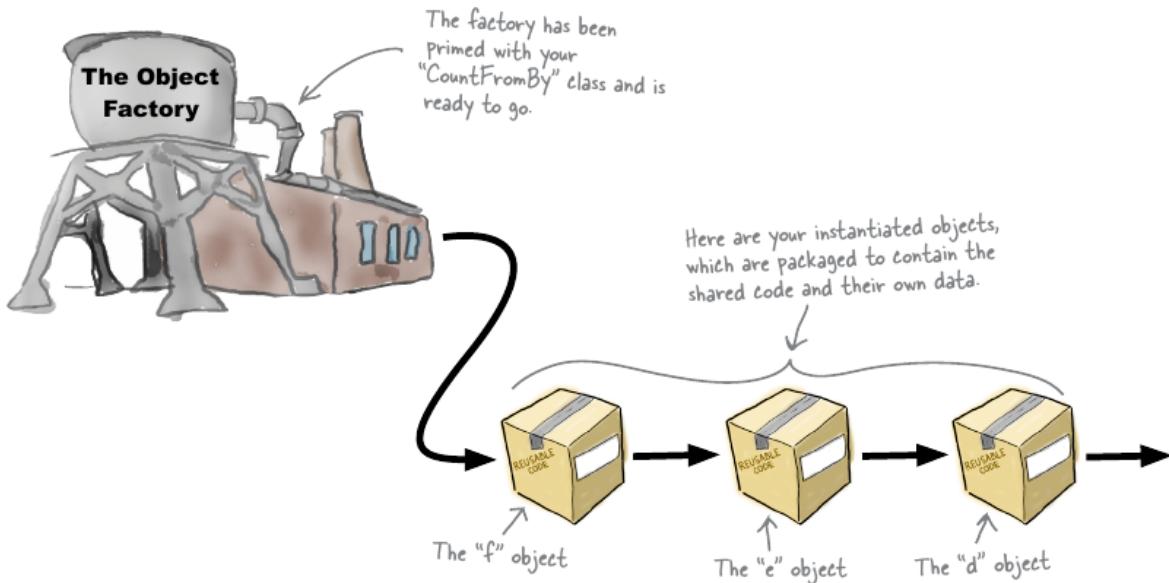
The diagram shows a block of Python code printing values for objects `c`, `d`, `e`, and `f`. A curly brace groups the lines for `d`, `e`, and `f`, with a note stating "Estos cuatro objetos 'CountFromBy' mantienen sus valores de nuestros atributos." The values printed are 3, 103, 130, and 45 respectively.

```
>>> c
3
>>> d
103
>>> e
130
>>> f
45
```

*El comportamiento de clase es compartido por cada uno de sus objetos, mientras que el estado no lo es. Cada objeto mantiene su propio estado.*

Aquí está el punto clave otra vez: el código del método es compartido, pero los datos del atributo no lo son.

Puede ser útil pensar en una clase como una "plantilla de cortador de galletas" que es utilizada por una fábrica para producir objetos que se comportan igual, pero que tienen sus propios datos.



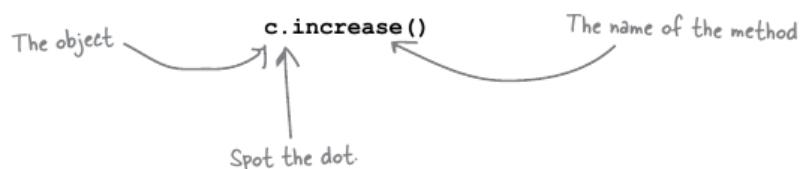
Traducimos:

1. La fábrica ha sido preparada con su clase "CountFromBy" y está lista para funcionar.
2. Aquí están sus objetos instanciados que se empaquetan para contener el código compartido y sus propios datos.

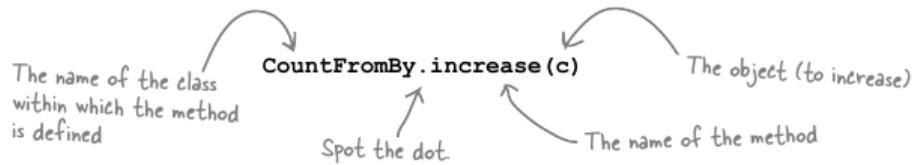
### ***Invoking a Method: Understand the Details***

### ***Invocando un Método: Comprenda los Detalles***

Hemos dicho anteriormente que un método es una función definida dentro de una clase. También vimos ejemplos de un método de CountFromBy que se invoca. El método de aumento se invoca utilizando la notación de punto familiar:



Es instructivo considerar el código que el intérprete ejecuta realmente (detrás de las escenas) cuando encuentra la línea anterior. Esta es la llamada que el intérprete siempre convierte en la línea de código anterior. Tenga en cuenta lo que sucede con c:



*¿El hecho de que esto ocurra significa que puedo escribir "CountFromBy.increase (c)" en mi código y funcionará como si hubiera escrito "c.increase ()"?*

### Sí, lo hace. Pero nadie lo hace.

Y tampoco debes, como el intérprete de Python hace esto para ti de todos modos ... así que ¿por qué escribir más código para hacer algo que se pueda escribir más suavemente?

Sólo por qué el intérprete hace esto se hará más claro a medida que aprenda más sobre cómo funcionan los métodos.

### *Method Invocation: What Actually Happens*

### *Invocación del método: lo que realmente sucede*

A primera vista, el intérprete que se convierte en `c.increase()` En `CountFromBy.increase (c)` puede parecer un poco extraño, pero la comprensión de que esto sucede ayuda a explicar por qué cada método que escribir toma por lo menos un argumento.

Está bien que los métodos tomen más de un argumento, pero el primer argumento siempre tiene que existir para tomar el objeto como un argumento (que, en el ejemplo de la última página, es `c`). De hecho, es una práctica bien establecida en la comunidad de programación de Python dar al primer argumento de cada método un nombre especial: `self`.

Cuando el incremento se invoca como `c.increase()`, se imaginaría que la línea def del método debería tener este aspecto:

```
def increase():
```

Sin embargo, la definición de un método sin el primer argumento obligatorio hará que el intérprete plantea un error cuando se ejecuta el código. Por consiguiente, la línea def del método de aumento realmente necesita escribirse como sigue:

```
def increase(self):
```

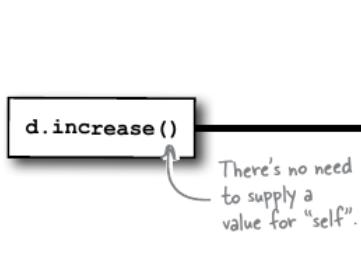
*Al escribir código en una clase, piense en "self" como un alias al objeto actual.*

Se considera como muy mala forma de usar algo que no sea el nombre de sí mismo en su código de clase, a pesar de que el uso de sí mismo toma un poco de acostumbrarse.

(Muchos otros lenguajes de programación tienen una noción similar, aunque favorecen el nombre esto. El `self` de Python es básicamente la misma idea que `this`.)

Cuando invoca un método en un objeto, Python organiza que el primer argumento sea la instancia del objeto que invoca, que siempre se asigna al argumento propio de cada método. Este hecho por sí solo explica por qué el `self` es tan importante y también por qué el `self` necesita ser el primer argumento para cada método de objeto que escriba. Cuando invoca un método, no necesita proporcionar un valor para `self`, como lo hace el intérprete para usted:

**What you write:**



**What Python executes:**

El nombre del objeto

El intérprete asigna el valor de "d" al "self".

Ahora que ha sido introducido a la importancia de uno mismo, echemos un vistazo a escribir el código para el método de aumento o `increase`.

### ***Adding a Method to a Class***

#### ***Agregar un método a una clase***

Creemos un nuevo archivo para guardar nuestro código de clase. Cree `countfromby.py` y, a continuación, agregue el código de la clase anteriormente en este capítulo:

```
class CountFromBy:  
    pass
```

Vamos a agregar el método de incremento a esta clase, y para ello removeremos la instrucción `pass` y la reemplazaremos por la definición del método `increase`. Antes de hacer esto, recuerde cómo se invoca el aumento:

```
c.increase()
```

Sobre la base de esta llamada, usted sería perdonado por asumir el método de aumento no toma argumentos, ya que no hay nada entre los paréntesis, ¿verdad? Sin embargo, esto es sólo la mitad verdad. Como acaba de aprender, el intérprete transforma la línea de código anterior en la siguiente llamada:

```
CountFromBy.increase(c)
```

El código de método que escribimos necesita tener en cuenta esta transformación. Con todo lo anterior en mente, aquí está la línea def para el método de aumento que usaríamos en esta clase:

```
class CountFromBy:  
    def increase(self) -> None:
```

Los métodos son como las funciones,  
por lo que se definen con "def".

El primer argumento para cada método es siempre "self", y  
su valor es suministrado por el intérprete.

Al igual que con las otras  
funciones de este libro,  
proporcionamos una anotación  
para el valor de retorno.

No hay otros argumentos para el método de incremento, por lo que no necesitamos proporcionar nada más que self en la línea def. Sin embargo, es de vital importancia que incluyamos el self aquí, como olvidando resultados en errores de sintaxis.

Con la línea de def escrita, todo lo que necesitamos hacer ahora es añadir algún código para aumentar. Supongamos que la clase mantiene dos atributos: val, que contiene el valor actual del objeto actual, e incr, que contiene la cantidad para incrementar val por cada incremento de tiempo que se invoca. Sabiendo esto, puede ser tentado a agregar esta línea incorrecta de código para aumentar en un intento de realizar el incremento:

```
val += incr
```

Pero aquí está la línea correcta de código para agregar al método de aumento:

```
class CountFromBy:  
    def increase(self) -> None:  
        self.val += self.incr
```

Tomar el valor actual  
del objeto de "val" y  
aumentarlo por el valor  
de "incr".

¿Por qué crees que esta línea de código es correcta, mientras que la anterior era incorrecta?

### ***Are You Serious About "self"?***

### ***¿Es usted serio acerca de "self"?***

Espera un minuto ... Pensé que una de las grandes victorias de Python fue que su código es fácil de leer. Me parece que el uso de "self" nada pero fácil en el ojo, y el hecho de que es parte de las clases (que debe obtener un montón de uso) me tiene pensando: ¿en serio?!?

### ***No te preocunes. Acostumbrarse a self no tomará mucho tiempo.***

Estamos de acuerdo en que el uso de Python de self se ve un poco extraño ... al principio. Sin embargo, con el tiempo, usted se acostumbrará a él, tanto que apenas se dará cuenta de que está allí.

Si te olvidas completamente de ello y no lo añades a tus métodos, sabrás muy pronto que algo anda mal: el intérprete mostrará una gran cantidad de TypeErrors que te informarán de que falta algo y que algo es self.

En cuanto a si el uso self hace que el código de clase de Python sea más difícil de leer ... bueno, no estamos tan seguros. En nuestra mente, cada vez que vemos el self usado como el primer argumento de una función, nuestros cerebros saben automáticamente que estamos mirando un método, no una función. Esto, para nosotros, es una buena cosa.

Piense en ello de esta manera: el uso de self indica que el código que está leyendo es un método, en contraposición a una función (cuando no se utiliza self).

**self == object**

### ***The Importance of “self”***

### ***La Importancia del "self"***

El método de aumento increase, que se muestra a continuación, prefijos cada uno de los atributos de la clase con el self dentro de su suite. Se le pidió que considerara por qué esto podría ser:

```
class CountFromBy:  
    def increase(self) -> None:  
        self.val += self.incr
```

¿Cuál es el trato con el uso de "self" dentro de la suite del método?

Ya sabes que el intérprete asigna al objeto actual el objeto actual cuando se invoca un método y que el intérprete espera que el primer argumento de cada método tenga en cuenta (para que la asignación pueda ocurrir).

Ahora, considere lo que ya sabemos sobre cada objeto creado a partir de una clase: comparte el código de método de la clase (comportamiento a.k.a.) con cualquier otro objeto creado a partir de la misma clase, pero mantiene su propia copia de cualquier dato de atributo (estado a.k.a.). Esto lo hace asociando los valores de los atributos con el objeto, es decir, con el self.

Sabiendo esto, considere esta versión del método de aumento, que, como dijimos hace un par de páginas, es incorrecta:

```
class CountFromBy:  
    def increase(self) -> None:  
        val += incr
```

No hagas esto, no hará lo que creas que debería.



En la cara de las cosas, esa última línea de código parece bastante inocente, ya que todo lo que hace es incrementar el valor actual de val por el valor actual de incr. Pero considere lo que sucede cuando este método de aumento termina: val e incr, que existen dentro de un aumento, ambos salen del alcance y, por consiguiente, son destruidos en el momento en que termina el método.

*Ummm ... permítanme hacer una nota de "salir del alcance" y "destruido." Voy a tener que mirar a los dos más tarde ... o me he perdido algo?*

**¡Vaya! Esa es nuestra mala ...**

Nos deslizamos en esa declaración sobre el alcance sin mucha explicación, ¿no? Para entender lo que tiene que suceder cuando se refieren a los atributos de un método, primero debemos pasar algún tiempo comprendiendo lo que sucede con las variables utilizadas en una función.

## **Coping with Scoping**

### **Cómo lidiar con el escopo**

Para demostrar lo que sucede con las variables utilizadas dentro de una función, vamos a experimentar en el prompt >>>. Pruebe el código a continuación cuando lo lea. Hemos numerado las anotaciones 1 a 8 para guiarlo a medida que lo sigue:

The screenshot shows a Python 3.5.1 Shell window with the following code and annotations:

```
>>> def soundbite(from_outside):
...     insider = 'James'
...     outsider = from_outside
...     print(from_outside, insider, outsider)
...
>>> name = 'Bond'
>>> soundbite(name)
Bond James Bond
>>> name
'Bond'
>>> insider
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    insider
NameError: name 'insider' is not defined
>>> outsider
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    outsider
NameError: name 'outsider' is not defined
>>> from_outside
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    from_outside
NameError: name 'from_outside' is not defined
>>>
>>> |
```

Annotations:

1. The "soundbite" function accepts a single argument.
2. A value is assigned to a variable inside the function.
3. The argument is assigned to another variable inside the function.
4. The function's variables are used to display a message.
5. A value is assigned to a variable called "name".
6. The "soundbite" function is invoked.
7. After the function displays the soundbite, the value of "name" is still accessible.
8. But none of the variables used within the function are accessible, as they only exist within the function's suite.

Traducimos:

1. La función "soundbite" acepta un único argumento.
2. Se asigna un valor a una variable dentro de la función.
3. El argumento se asigna a otra variable dentro de la función.
4. Las variables de la función se utilizan para mostrar un mensaje.
5. Se asigna un valor a una variable llamada "nombre".
6. Se invoca la función "soundbite".
7. Después de que la función muestre el soundbite, el valor de "name" todavía es accesible.
8. Pero ninguna de las variables utilizadas dentro de la función son accesibles, ya que sólo existen dentro de la suite de la función.

Cuando las variables se definen dentro del conjunto de una función, existen mientras la función se ejecuta. Es decir, las variables son "en el ámbito", tanto visibles como utilizables dentro de la suite de la función. Sin embargo, una vez que la función finaliza, las variables definidas dentro de la función se destruyen: están fuera del alcance y todos los recursos que utilizan son recuperados por el intérprete.

Esto es lo que sucede con las tres variables utilizadas dentro de la función soundbite, como se muestra arriba. En el momento en que la función termina, el insider, el outsider y from\_outside. Cualquier intento de hacer referencia a ellos fuera del conjunto de funciones (a.k.a. fuera del ámbito de la función) da como resultado un NameError.

### ***Prefix Your At tribute Name s with “self”***

### ***Prefijo sus nombres de atributo con "self"***

Este comportamiento de función descrito en la última página está bien cuando se trata de una función que se invoca, hace algo de trabajo y, a continuación, devuelve un valor. Por lo general, no le importa lo que suceda con las variables que se usan dentro de una función, ya que normalmente sólo están interesados en el valor de retorno de la función.

Ahora que ya sabes lo que sucede con las variables cuando una función termina, debería estar claro que este código (incorrecto) es probable que cause problemas cuando intentas usar variables para almacenar y recordar valores de atributos con una clase. Como los métodos son funciones por otro nombre, ni val ni incr sobrevivirán a una invocación del método de incremento si así es como se incrementa el código:

```
class CountFromBy:  
    def increase(self) -> None:  
        val += incr
```

No haga esto, ya que estas variables no  
sobrevivirán. Una vez que el método  
termine



Sin embargo, con los métodos, las cosas son diferentes. El método utiliza valores de atributo que pertenecen a un objeto y los atributos del objeto continúan existiendo una vez

finalizado el método. Es decir, los valores de atributo de un objeto no se destruyen cuando termina el método.

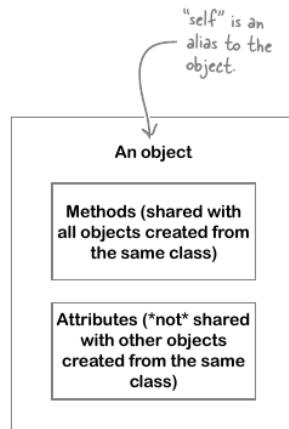
Para que una asignación de atributos sobreviva a la terminación del método, el valor del atributo tiene que ser asignado a algo que no se destruya tan pronto como el método termine. Que algo es el objeto actual que invoca el método, que se almacena en el self, lo que explica por qué cada valor de atributo necesita ser prefijado con self en su código de método, como se muestra aquí:

```
class CountFromBy:  
    def increase(self) -> None:  
        self.val += self.incr
```

Esto es mucho mejor, ya que "val" e "incr" se asocian ahora con el objeto gracias al uso de "self".

La regla es sencilla: si necesita referirse a un atributo de su clase, debe prefijar el nombre del atributo con self. El valor en self como un alias que señala de nuevo al objeto que invoca el método.

En este contexto, cuando ves a self, piensa "en este objeto". Así, self.val puede ser leído como "val de este objeto".



### ***Initialize (At tribute) Values Before Use***

### ***Inicializar (atributo) Valores antes del uso***

Toda la discusión sobre la importancia del yo se desvió de una cuestión importante: ¿cómo se asigna un valor inicial a los atributos? En su estado actual, el código en el método de aumento-el código correcto, que utiliza el self falla si lo ejecuta. Este error se produce porque en Python no se puede utilizar una variable antes de que se le haya asignado un valor, sin importar dónde se use la variable.

Para demostrar la gravedad de este problema, considere esta corta sesión en el prompt >>. Tenga en cuenta cómo la primera instrucción no se ejecuta cuando ninguna de las variables está indefinida:

If you try to execute code that refers to uninitialized variables...

...the interpreter complains.

Assign a value to "val", then try again...

...and the interpreter complains again!

Assign a value to "incr", and try again...

...and it worked this time.

```
>>> val += incr
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    val += incr
NameError: name 'val' is not defined
```

```
>>> val = 0
>>> val += incr
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    val += incr
NameError: name 'incr' is not defined
```

```
>>> incr = 1
>>> val += incr
>>> val
1
>>> incr
1
>>>
```

As "val" is undefined, the interpreter refuses to run the line of code.

As "incr" is undefined, the interpreter continues to refuse to run the line of code.

As both "val" and "incr" have values (i.e., they are initialized), the interpreter is happy to use their values without raising a NameError.

Traducimos:

1. Si intenta ejecutar código que hace referencia a variables no inicializadas ...
2. ... el intérprete se queja.
3. Asigne un valor a "val" y vuelva a intentarlo ...
4. ... y el intérprete se queja de nuevo!
5. Asigne un valor a "incr" e intételo de nuevo ...
6. ... y funcionó esta vez.
7. Como "val" no está definido, el intérprete se niega a ejecutar la línea de código.
8. Como "incr" no está definido, el intérprete continúa negándose a ejecutar la línea de código.
9. Como tanto val como incr se tienen valores (es decir, se inicializan), el intérprete está feliz de usar sus valores sin elevar un NameError.

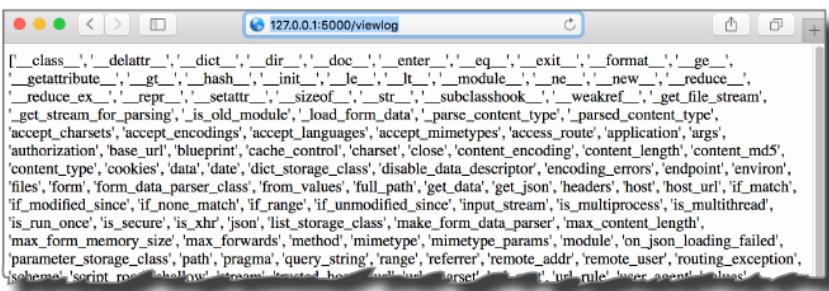
No importa donde uses variables en Python, debes inicializarlas con un valor inicial. La pregunta es: ¿cómo hacemos esto para un nuevo objeto creado a partir de una clase Python?

Si conoce a OOP, la palabra "constructor" puede estar apareciendo en su cerebro ahora. En otros idiomas, un constructor es un método especial que le permite definir lo que sucede cuando se crea un objeto por primera vez, y normalmente implica tanto la instanciaión de objetos como la inicialización de atributos. En Python, la instanciaión de objetos es manejada automáticamente por el intérprete, por lo que no es necesario definir un constructor para hacer esto. Un método mágico llamado `__init__` le permite inicializar atributos según sea necesario. Echemos un vistazo a lo que puede hacer `dunder init`.

## Dunder “init” Initializes Attributes

### Dunder "init" Inicializa los atributos

Vuelve tu mente al último capítulo, cuando usaste la función incorporada de dir para mostrar todos los detalles del objeto Req de Flask. ¿Recuerdas esta salida?



The screenshot shows a browser window with the URL 127.0.0.1:5000/viewlog. The page content is a long list of method names starting with underscores, such as \_\_class\_\_, \_\_delattr\_\_, \_\_dict\_\_, \_\_dir\_\_, \_\_doc\_\_, \_\_enter\_\_, \_\_eq\_\_, \_\_exit\_\_, \_\_format\_\_, \_\_ge\_\_, \_\_getattribute\_\_, \_\_gt\_\_, \_\_hash\_\_, \_\_init\_\_, \_\_le\_\_, \_\_lt\_\_, \_\_module\_\_, \_\_ne\_\_, \_\_new\_\_, \_\_reduce\_\_, \_\_reduce\_ex\_\_, \_\_repr\_\_, \_\_setattr\_\_, \_\_sizeof\_\_, \_\_str\_\_, \_\_subclasshook\_\_, \_\_weakref\_\_, \_\_get\_file\_stream\_\_, \_\_get\_stream\_for\_parsing\_\_, \_\_is\_old\_module\_\_, \_\_load\_form\_data\_\_, \_\_parse\_content\_type\_\_, \_\_parsed\_content\_type\_\_, \_\_accept\_charset\_\_, \_\_accept\_encodings\_\_, \_\_accept\_languages\_\_, \_\_accept\_mimetypes\_\_, \_\_access\_route\_\_, \_\_application\_\_, \_\_args\_\_, \_\_authorization\_\_, \_\_base\_url\_\_, \_\_blueprint\_\_, \_\_cache\_control\_\_, \_\_charset\_\_, \_\_close\_\_, \_\_content\_encoding\_\_, \_\_content\_length\_\_, \_\_content\_md5\_\_, \_\_content\_type\_\_, \_\_cookies\_\_, \_\_data\_\_, \_\_date\_\_, \_\_dict\_storage\_class\_\_, \_\_disable\_data\_descriptor\_\_, \_\_encoding\_errors\_\_, \_\_endpoint\_\_, \_\_environ\_\_, \_\_files\_\_, \_\_form\_\_, \_\_form\_data\_parser\_class\_\_, \_\_from\_values\_\_, \_\_full\_path\_\_, \_\_get\_data\_\_, \_\_get\_json\_\_, \_\_headers\_\_, \_\_host\_\_, \_\_host\_url\_\_, \_\_if\_match\_\_, \_\_if\_modified\_since\_\_, \_\_if\_none\_match\_\_, \_\_if\_range\_\_, \_\_if\_unmodified\_since\_\_, \_\_input\_stream\_\_, \_\_is\_multiprocess\_\_, \_\_is\_multithread\_\_, \_\_is\_run\_once\_\_, \_\_is\_secure\_\_, \_\_is\_xhr\_\_, \_\_json\_\_, \_\_list\_storage\_class\_\_, \_\_make\_form\_data\_parser\_\_, \_\_max\_content\_length\_\_, \_\_max\_form\_memory\_size\_\_, \_\_max\_forwards\_\_, \_\_method\_\_, \_\_mimetype\_\_, \_\_mimetype\_params\_\_, \_\_module\_\_, \_\_on\_json\_loading\_failed\_\_, \_\_parameter\_storage\_class\_\_, \_\_path\_\_, \_\_pragma\_\_, \_\_query\_string\_\_, \_\_range\_\_, \_\_referrer\_\_, \_\_remote\_addr\_\_, \_\_remote\_user\_\_, \_\_routing\_exception\_\_, \_\_schema\_\_, \_\_script\_\_, \_\_shallow\_\_, \_\_stream\_\_, \_\_unquoted\_host\_\_, \_\_urlset\_\_, \_\_urlrule\_\_, \_\_user\_agent\_\_, \_\_value\_\_.

En ese momento, te sugerimos que ignoraras todos esos obstáculos. Sin embargo, ahora es el momento de revelar su propósito: los dunders proporcionan ganchos en el comportamiento estándar de cada clase.

A menos que lo anule, este comportamiento estándar se implementa en una clase llamada objeto. La clase de objeto está integrada en el intérprete, y cada otra clase de Python hereda automáticamente de ella (incluida la tuya). Esta es la OOP para decir que los métodos de dunder proporcionados por objeto están disponibles para su clase para usarlos como es, o para reemplazar cuando sea necesario (proporcionando su propia implementación de ellos).

No es necesario que anule ningún método de objeto si no desea hacerlo. Pero si, por ejemplo, desea especificar qué sucede cuando los objetos creados de su clase se utilizan con el operador de igualdad (==), puede escribir su propio código para el método \_\_eq\_\_. Si desea especificar qué sucede cuando se utilizan objetos con el operador mayor-que (>), puede sobrescribir el método \_\_ge\_\_. Y cuando quiera inicializar los atributos asociados con su objeto, puede utilizar el método \_\_init\_\_.

Como los dunders proporcionados por el objeto son tan útiles, se celebran en la reverencia casi mística por los programadores de Python. Tanto es así, de hecho, que muchos programadores de Python se refieren a estos dunders como los métodos mágicos (ya que dan la apariencia de hacer lo que hacen "como por arte de magia").

Todo esto significa que si usted proporciona un método en su clase con una línea def como la que se muestra a continuación, el intérprete llamará su método \_\_init\_\_ cada vez que cree un nuevo objeto de su clase. Observe la inclusión de self como el primer argumento de dunder init (según la regla para todos los métodos en todas las clases):

*Los métodos estándar de dunder, disponibles para todas las clases, son conocidos como "los métodos mágicos".*

```
def __init__(self):
```



A pesar de la extraña apariencia nombre dunder "init" es un método como cualquier otro. Recuerde: usted debe pasar "self" como su primer argumento.

### ***Initializing Attributes with Dunder "init"***

#### ***Inicializar atributos con Dunder "init"***

Añadamos `__init__` a nuestra clase `CountFromBy` para inicializar los objetos que creamos de nuestra clase.

Por ahora, vamos a agregar un método vacío `__init__` que no hace nada sino pasar (vamos a añadir el comportamiento en un momento):

```
class CountFromBy:  
    def __init__(self) -> None:  
        pass  
    def increase(self) -> None:  
        self.val += self.incr
```

← Por el momento, este dunder "init" no hace nada.  
Sin embargo, el uso de "self" como su primer argumento es una GRAN CLUE que dunder "init" es un método.

Sabemos por el código ya en aumento que podemos acceder a los atributos de nuestra clase prefijando sus nombres con `self`. Esto significa que podemos usar `self.val` y `self.incr` para referirnos a nuestros atributos dentro de `__init__`, también. Sin embargo, queremos usar `__init__` para inicializar los atributos de nuestra clase (`val` e `incr`). La pregunta es: ¿de dónde vienen estos valores de inicialización y cómo sus valores entran en `__init__`?

### ***Pass any amount of argument data to dunder "init"***

#### ***Pasar cualquier cantidad de datos de argumento a dunder "init"***

Como `__init__` es un método, y los métodos son funciones disfrazadas, puedes pasar tantos valores de argumentos como quieras a `__init__` (o cualquier método, para el caso). Todo lo que tienes que hacer es darle a tus argumentos nombres. Vamos a dar el argumento de que usaremos para inicializar `self.val` el nombre `v`, y usar el nombre `i` para `self.incr`.

Añadamos `v` e `i` a la línea `def` de nuestro método `__init__`, luego usamos los valores en la suite de dunder `init` para inicializar nuestros atributos de clase, como sigue:

```

class CountFromBy:
    def __init__(self, v: int, i: int) -> None:
        self.val = v
        self.incr = i
    def increase(self) -> None:
        self.val += self.incr

```

*Add "v" and "i" as arguments to dunder "init".*

*Use the values of "v" and "i" to initialize the class's attributes (which are "self.val" and "self.incr", respectively).*

Traducimos:

1. Utilice los valores de "v" e "i" para inicializar los atributos de la clase (que son "self.val"). Y "self.incr", respectivamente
2. Agregar "v" e "i" como argumentos para dunder "init".

Si de alguna manera podemos organizar que v e i adquieran valores, la última versión de `__init__` inicializará los atributos de nuestra clase. Lo que plantea otra pregunta: ¿cómo podemos obtener valores en v e i? Para ayudar a responder a esta pregunta, necesitamos probar esta versión de nuestra clase y ver qué sucede. Hagámoslo ahora.



## Test Drive

Utilizando la ventana de edición en IDLE, tome un momento para actualizar el código en su archivo `countfromby.py` para que se vea como se muestra a continuación. Cuando haya terminado, presione F5 para comenzar a crear objetos en el prompt >>> de IDLE:

*Press F5 to try out the "CountFromBy" class in IDLE's shell.*

*The latest version of our "CountFromBy" class.*

```

class CountFromBy:
    def __init__(self, v: int, i: int) -> None:
        self.val = v
        self.incr = i
    def increase(self) -> None:
        self.val += self.incr

```

Ln: 2 Col: 0

Al presionar F5 se ejecuta el código en la ventana de edición, que importa la clase `CountFromBy` al intérprete. Mira lo que sucede cuando tratamos de crear un nuevo objeto de nuestra clase `CountFromBy`:

```

Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015, 21:12:44)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch07/countfromby.py ======
>>> g = CountFromBy()
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    g = CountFromBy()
TypeError: __init__() missing 2 required positional arguments: 'v' and 'i'
>>>
>>> |

```

Ln: 13 Col: 4

Crear un nuevo objeto (llamado "g") de la clase ... pero cuando haces esto, obtienes un error!

Esto puede no haber sido lo que esperabas ver. Pero eche un vistazo al mensaje de error (que se clasifica como `TypeError`), prestando especial atención al mensaje en la línea `TypeError`. El intérprete nos está diciendo que el método `__init__` espera recibir dos valores de argumento, `v` y `i`, pero recibe algo más (en este caso, nada). No proporcionamos argumentos a la clase, pero este mensaje de error nos dice que los argumentos proporcionados a la clase (al crear un nuevo objeto) se pasan al método `__init__`.

Teniendo esto en cuenta, vamos a tener otra oportunidad de crear un objeto `CountFromBy`.

Volvamos al prompt `>>>` y creamos otro objeto (llamado `h`) que toma dos valores enteros como argumentos para `v` e `i`:

```

No "TypeError" this time
Python 3.5.1 Shell
>>>
>>> h = CountFromBy(100, 10)
>>> h.val
100
>>> h.incr
10
>>> h.increase()
>>> h.val
110
>>> h
<__main__.CountFromBy object at 0x105a13da0>
>>>
>>> |

```

Ln: 37 Col: 4

Annotations:

- "You can access the value of the "h" object's attributes." points to `h.val`
- "No "TypeError" this time" points to the top of the code block.
- "Invoking the "increase" method does what you expect it to do. It increments "h.val" by the amount in "h.incr"." points to `h.increase()`
- "You were probably expecting to see "110" displayed here, but instead got this (rather cryptic) message instead." points to the final blank line.

Traducción:

Usted probablemente esperaba ver "110" dispuesto aquí, anuncio. Pero en su lugar recibió este mensaje (bastante críptico) en su lugar.

## ***What have we learned from this Test Drive?***

### ***¿Qué hemos aprendido de este Test Drive?***

Estos son los principales puntos de venta de este Test Drive:

- Cuando está creando objetos, los valores de argumento proporcionados a la clase se pasan al método `__init__`, como fue el caso con 100 y 10 anteriores. (Tenga en

cuenta que `v` e `i` dejan de existir tan pronto como termina el dunder `init`, pero no estamos preocupados, ya que sus valores se almacenan de forma segura en los atributos `self.val` y `self.incr` del objeto, respectivamente).

- Podemos acceder a los valores de los atributos combinando el nombre del objeto con el nombre del atributo. Observe cómo usamos `h.val` y `h.incr` para hacer esto. (Para aquellos lectores que vienen a Python desde un lenguaje "OOP" más estricto, tenga en cuenta que lo hicimos sin tener que crear getters o setters).
- Cuando usamos el nombre del objeto por sí solo (como en la última interacción con el shell anterior), el intérprete escupe un mensaje críptico. Justo lo que esto es (y por qué esto sucede) será discutido a continuación.

## ***Understanding CountFromBy's Representation***

### ***Descripción de la representación de CountFromBy***

Cuando tecleamos el nombre del objeto en el shell en un intento de mostrar su valor actual, el intérprete produjo este resultado:

```
<__main__.CountFromBy object at 0x105a13da0> No te preocupes si tienes un valor diferente aquí. Todo  
quedará claro antes del final de esta página.
```

Describimos la salida anterior como "extraña", ya primera vista, ciertamente parecería ser. Para entender lo que significa esta salida, volvamos a la shell de IDLE y creamos otro objeto de `CountFromBy`, que debido a nuestra profunda falta de voluntad de rockear el barco, estamos llamando a `j`.

En la siguiente sesión, observe cómo el mensaje extraño mostrado para `j` está compuesto por valores que se producen cuando llamamos a ciertas funciones integradas (BIF). Siga junto con la sesión primero, luego siga leyendo para una explicación de lo que estos BIFs hacen:

The screenshot shows a Python 3.5.1 Shell window. The session starts with creating a `CountFromBy` object:

```
>>> j = CountFromBy(100, 10)
>>> j
<__main__.CountFromBy object at 0x1035be278>
```

An annotation points to the output with the text: "The output for 'j' is made up of values produced by some of Python's BIFs."

Then, the `type` function is used to inspect the type of `j`:

```
>>> type(j)
<class '__main__.CountFromBy'>
```

An annotation points to the output with the text: "The output for 'j' is made up of values produced by some of Python's BIFs."

Next, the `id` function is used to get the memory address of `j`:

```
>>> id(j)
4351320696
```

An annotation points to the output with the text: "The output for 'j' is made up of values produced by some of Python's BIFs."

Finally, the `hex` function is used to convert the memory address to a hex string:

```
>>> hex(id(j))
'0x1035be278'
```

An annotation points to the output with the text: "The output for 'j' is made up of values produced by some of Python's BIFs."

El tipo BIF muestra información sobre la clase en la que se creó el objeto, informando (arriba) que `j` es un objeto `CountFromBy`.

El identificador BIF muestra información sobre la dirección de memoria de un objeto (que es un identificador único utilizado por el intérprete para realizar un seguimiento de los objetos). Lo que usted ve en su pantalla es probablemente diferente de lo que se ha informado anteriormente.

La dirección de memoria mostrada como parte de la salida de `j` es el valor de `id` convertido en un número hexadecimal (que es lo que hace el hex BIF). Por lo tanto, todo el mensaje que se muestra para `j` es una combinación de salida del tipo, así como `id` (convertido a hexadecimal).

***Una pregunta razonable es: ¿por qué sucede esto?***

En ausencia de que le diga al intérprete cómo desea representar sus objetos, el intérprete tiene que hacer algo, por lo que hace lo que se muestra arriba. Afortunadamente, puede anular este comportamiento predeterminado codificando su propio método mágico `__repr__`.

*Anular dunder "repr" para especificar cómo los objetos son representados por el intérprete.*

## ***Defining CountFromBy's Representation***

### ***Definición de la representación de CountFromBy***

Además de ser un método mágico, la funcionalidad `__repr__` también está disponible como función incorporada llamada `repr`. Aquí es parte de lo que la ayuda BIF muestra cuando le pide que le diga lo que hace: "Devolver la representación de cadena canónica del objeto." En otras palabras, la ayuda BIF le está diciendo que las necesidades de `repr` (y por extensión, `__repr__`) Para devolver una versión stringificada de un objeto.

Lo que esta "versión stringificada de un objeto" parece depender de lo que hace cada objeto individual. Puede controlar lo que sucede con sus objetos escribiendo un método `__repr__` para su clase. Hagamos esto ahora para la clase `CountFromBy`.

Comience agregando una nueva línea `def` a la clase `CountFromBy` para dunder `repr`, que no toma argumentos que no sean el `self` requerido (recuerde: es un método). Como es nuestra práctica, vamos a añadir también una anotación que permite a los lectores de nuestro código saber que este método devuelve una cadena:

Como cualquier otro método que escribas, éste tiene que tener en cuenta que el intérprete siempre proporciona un valor para el primer argumento.

```
def __repr__(self) -> str:
```



Esto permite a los usuarios de este método saber que esta función tiene la intención de devolver una cadena. Recuerde: el uso de anotaciones en su código es opcional, pero útil.

Con la línea def escrita, lo único que queda es escribir el código que devuelve una representación de cadena de un objeto CountFromBy. Para nuestros propósitos, todo lo que queremos hacer aquí es tomar el valor en self.val, que es un entero, y convertirlo en una cadena.

Gracias al str BIF, hacerlo es sencillo:

```
def __repr__(self) -> str:  
    return str(self.val)
```



Tome el valor en "self.val", a su vez en una cadena, y luego devolverlo a la persona que llama de este método.

Cuando agrega esta función corta a su clase, el intérprete la usa siempre que necesite mostrar un objeto CountFromBy en el prompt >>>. El BIF de impresión también utiliza dunder repr para mostrar objetos.

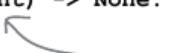
Antes de realizar este cambio y tomar el código actualizado para un giro, vamos a volver brevemente a otro problema que surgió durante la última prueba de unidad.

### ***Providing Sensible Defaults for CountFromBy***

### ***Proporcionar valores predeterminados sensibles para CountFromBy***

Recordemos la versión actual del método \_\_init\_\_ de la clase CountFromBy:

```
...  
def __init__(self, v: int, i: int) -> None:  
    self.val = v  
    self.incr = i  
    ...
```



Esta versión del método dunder "init" espera que se proporcionen dos valores de argumento cada vez que se invoca.

Recuerde que cuando intentamos crear un nuevo objeto de esta clase sin pasar valores para v e i, tenemos un TypeError:

```
>>>  
>>> g = CountFromBy()  
Traceback (most recent call last):  
  File "<pyshell#1>", line 1, in <module>  
    g = CountFromBy()  
TypeError: __init__() missing 2 required positional arguments: 'v' and 'i'  
>>>
```

Yikes! Not good.

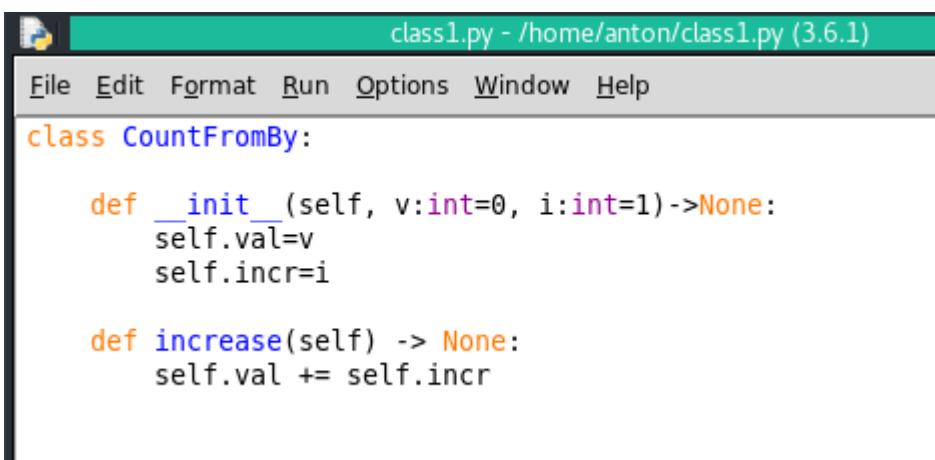
Anteriormente en este capítulo, especificamos que queríamos que la clase CountFromBy soportara el siguiente comportamiento predeterminado: el contador comenzará en 0 y se incrementará (a petición) en 1. Ya sabes cómo proporcionar valores predeterminados a los argumentos de la función y el parámetro Lo mismo vale para los métodos, también-asignar los valores por defecto en la línea def:

```

    ...
def __init__(self, v: int=0, i: int=1) -> None:
    self.val = v
    self.incr = i
    ...

```

Como los métodos son funciones, soportan el uso de valores por defecto para argumentos (aunque estamos anotando un B- aquí para nuestro uso de nombres de variables de un solo carácter: "v" es el valor, mientras que "i" es el valor de incremento).



```

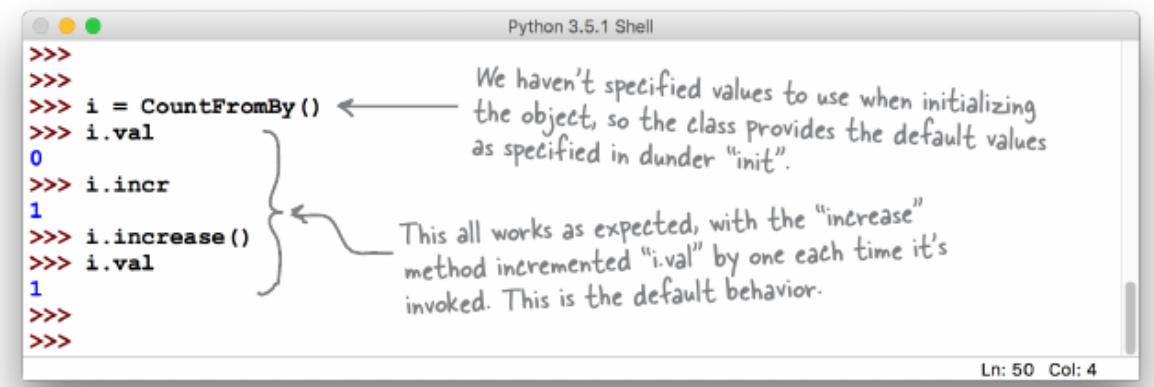
class1.py - /home/anton/class1.py (3.6.1)
File Edit Format Run Options Window Help
class CountFromBy:

    def __init__(self, v:int=0, i:int=1)->None:
        self.val=v
        self.incr=i

    def increase(self) -> None:
        self.val += self.incr

```

Si realiza este pequeño (pero importante) cambio a su código CountFromBy, guarde el archivo (antes de presionar F5 una vez más), verá que ahora se pueden crear objetos con este comportamiento predeterminado:



```

Python 3.5.1 Shell
>>>
>>>
>>> i = CountFromBy() ← We haven't specified values to use when initializing
>>> i.val ← the object, so the class provides the default values
0
>>> i.incr ← as specified in dunder "init".
1
>>> i.increase() ← This all works as expected, with the "increase"
>>> i.val ← method incremented "i.val" by one each time it's
1
>>>
>>>

```

Annotations in the screenshot:

- A callout points from the first two lines of the shell output to the explanatory text: "We haven't specified values to use when initializing the object, so the class provides the default values as specified in dunder "init"."
- A callout points from the last two lines of the shell output to the explanatory text: "This all works as expected, with the "increase" method incremented "i.val" by one each time it's invoked. This is the default behavior."

Traducimos:

1. No hemos especificado valores para usar al inicializar el objeto, de la clase proporciona los valores por defecto como se especifica en dunder "init".

2. Todo esto funciona como se esperaba, con el método de incremento incrementado "i.val" por uno cada vez que se invoca. Este es el comportamiento predeterminado.



## — Test Drive —

AsegúrateEl protocolo de gestión de contexto

Enganchar en Python con la declaraciónse de que el código de su clase (en countfromby.py) es el mismo que el de abajo. Con su código de clase cargado en la ventana de edición de IDLE, presione F5 para tomar su última versión de la clase CountFromBy para una vuelta:

This is the  
"CountFromBy" class  
with the code for  
dunder "repr" added.

```
class CountFromBy:

    def __init__(self, v: int=0, i: int=1) -> None:
        self.val = v
        self.incr = i

    def increase(self) -> None:
        self.val += self.incr

    def __repr__(self) -> str:
        return str(self.val)

|
```

Ln: 13 Col: 0

El objeto "k"  
utiliza los  
valores  
predeterminados  
de la clase, que  
comienzan en 0  
y se incrementan  
en 1.

**Python 3.5.1 Shell**

```
>>> k = CountFromBy()
>>> k
0
>>> k.increase()
>>> k
1
>>> print(k)
1
>>> l = CountFromBy(100)
>>> l
100
>>> l.increase()
>>> print(l)
101
```

When you refer to the object at  
the >>> prompt, or in a call to  
"print", the dunder "repr" code runs.

The "l" object provides  
an alternative starting  
value, then increments  
by 1 each time  
"increase" is called.

El objeto "m"  
proporciona valores  
alternativos para  
ambos valores  
predeterminados.

```
>>> m = CountFromBy(100, 10)
>>> m
100
>>> m.increase()
>>> m
110
>>> n = CountFromBy(i=15)
>>> n
0
>>> n.increase()
>>> n
15
>>>
```

The "n" object uses a  
keyword argument to  
provide an alternative  
value to increment by  
(but starts at 0).

Ln: 33 Col: 4

## **Classes: What We Know**

### **Clases: Lo que sabemos**

Esto es todo bien y dandy ... pero me recuerdan: ¿cuál era el punto de aprender todas estas cosas de clase?

#### **Queríamos crear un gestor de contexto.**

Sabemos que ha sido un tiempo, pero la razón por la que iniciamos este camino fue aprender lo suficiente sobre las clases para que podamos crear código que se conecta al protocolo de administración de contexto de Python. Si podemos enganchar en el protocolo, podemos utilizar el código de la base de datos de nuestra webapp con Python con la sentencia, ya que hacerlo debería facilitar el compartir el código de la base de datos, así como reutilizarlo. Ahora que ya sabe un poco acerca de las clases, está listo para conectarse al protocolo de gestión de contexto (en el próximo capítulo).

#### **Chapter 8's Code**

This is the  
code in the  
"countfromby.  
py" file.

```
class CountFromBy:

    def __init__(self, v: int=0, i: int=1) -> None:
        self.val = v
        self.incr = i

    def increase(self) -> None:
        self.val += self.incr

    def __repr__(self) -> str:
        return str(self.val)
```

## **9** the context management protocol

# **Hooking into Python's with Statement**

### ***El protocolo de gestión de contexto Enganchar en Python con la declaración***

Es hora de tomar lo que acaba de aprender y ponerlo a trabajar.

El Capítulo 7 discutió el uso de una base de datos relacional con Python, mientras que el Capítulo 8 proporcionó una introducción al uso de clases en su código Python. En este capítulo, ambas técnicas se combinan para producir un gestor de contexto que nos permite extender la sentencia `with` para trabajar con sistemas de bases de datos relacionales. En este capítulo, se conectará a la sentencia `with` creando una nueva clase, que se ajuste al protocolo de administración de contexto de Python.

### ***What's the Best Way to Share Our Webapp's Database Code? ¿Cuál es la mejor manera de compartir el código de base de datos de Webapp?***

Durante el Capítulo 7 creó código de base de datos en su función `log_request` que funcionó, pero tuvo que pausar para considerar la mejor manera de compartirla. Recordemos las sugerencias del final del Capítulo 7:

- *Rápidamente cortar y pegar ese código, luego cambiarlo. ¡Hecho!*
- *Yo voto que ponemos que la base de datos de manejo de código en su propia función, a continuación, llamar como sea necesario.*
- *¿No está claro que es hora de que consideramos el uso de clases y objetos como la forma correcta de manejar este tipo de reutilización?*

En ese momento, propusimos que cada una de estas sugerencias era válida, pero creíamos que los programadores de Python tendrían pocas probabilidades de aceptar cualquiera de estas soluciones propuestas por su propia cuenta. Decidimos que una mejor estrategia era enganchar en el protocolo de la gerencia del contexto usar la declaración `with`, pero para hacer eso, usted necesitó aprender un pedacito sobre clases. Ellos fueron el tema del último capítulo. Ahora que sabes cómo crear una clase, es hora de volver a la tarea actual: crear un gestor de contexto para compartir el código de la base de datos de tu aplicación web.

### ***Consider What You're Trying to Do, Revisited***

### ***Considere lo que está tratando de hacer, revisitado***

A continuación se muestra nuestro código de gestión de bases de datos del Capítulo 7. Este código es actualmente parte de nuestra webapp Flask. Recuerde cómo este código se conectó a nuestra base de datos MySQL, guardó los detalles de la solicitud web en la tabla de registro, confirmó los datos no guardados y luego se desconectó de la base de datos:

```
import mysql.connector

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""

    dbconfig = { 'host': '127.0.0.1',
                 'user': 'vsearch',
                 'password': 'vsearchpasswd',
                 'database': 'vsearchlogDB', }

    conn = mysql.connector.connect(**dbconfig)
    cursor = conn.cursor()

    _SQL = """insert into log
              (phrase, letters, ip, browser_string, results)
              values
              (%s, %s, %s, %s, %s)"""
    cursor.execute(_SQL, (req.form['phrase'],
                         req.form['letters'],
                         req.remote_addr,
                         req.user_agent.browser,
                         res,))

    conn.commit()
    cursor.close()
    conn.close()
```

### ***How best to create a context manager?***

### ***¿Cómo crear un gestor de contexto?***

Antes de llegar al punto en el que se puede transformar el código anterior en algo que se puede utilizar como parte de una sentencia `with`, vamos a discutir cómo se logra esto mediante la conformidad con el protocolo de gestión de contexto. Aunque se admite la

creación de gestores de contexto simples en la biblioteca estándar (utilizando el módulo `contextlib`), la creación de una clase que se ajuste al protocolo se considera el enfoque correcto cuando se utiliza `with` para controlar algún objeto externo, como una base de datos `connection` (como es el caso aquí).

Con esto en mente, echemos un vistazo a lo que se entiende por "conformidad con el protocolo de gestión de contexto".

### ***Managing Context with Methods***

#### **Gestión de Contexto con Métodos**

El protocolo de gestión de contexto suena intimidante y aterrador, pero en realidad es bastante simple. Dice que cualquier clase que crees debe definir al menos dos métodos mágicos: `__enter__` y `__exit__`. Este es el protocolo. Cuando usted se adhiere al protocolo, su clase puede enganchar en la declaración `with`.

*Un protocolo es un procedimiento acordado (o un conjunto de reglas) que se debe cumplir.*

#### **Dunder “enter” performs setup**

#### **Dunder "enter" realiza la configuración**

Cuando un objeto se utiliza con una instrucción `with`, el intérprete invoca el método `__enter__` del objeto antes de que empiece la sentencia con la sentencia `with`. Esto proporciona una oportunidad para que realice cualquier código de configuración requerido dentro de dunder enter.

El protocolo establece además que dunder enter puede (pero no tiene que) devolver un valor a la sentencia `with` (verás por qué esto es importante en un poco).

#### **Dunder “exit” does teardown**

#### **Dunder "salida" se desmonta**

Tan pronto como termina la instrucción `with`, el intérprete invoca siempre el método `__exit__` del objeto. Esto ocurre después de que termine la suite `with`, y proporciona una oportunidad para que usted realice cualquier desmontaje requerido.

Como el código en el conjunto de la sentencia `with` puede fallar (y elevar una excepción), la salida dunder debe estar lista para manejar esto si sucede. Volveremos a este problema cuando creemos el código para nuestro método de salida de dunder más adelante en este capítulo.

Si crea una clase que define `__enter__` y `__exit__`, la clase es automáticamente considerada como un gestor de contexto por el intérprete y puede, como consecuencia, conectarse a `with`

(y utilizarse `with`). En otras palabras, una clase de este tipo se ajusta al protocolo de gestión de contexto, e implementa un gestor de contexto.

*Si su clase define dunder "enter" y dunder "exit", es un gestor de contexto.*

### **(As you know) dunder "init" initializes**

### **(Como sabes) dunder "init" se inicializa**

Además de dunder `enter` y dunder `exit`, puede agregar otros métodos a su clase según sea necesario, incluyendo la definición de su propio método `__init__`. Como se sabe del último capítulo, la definición de dunder `init` le permite realizar una inicialización de objetos adicional. Dunder `init` se ejecuta antes de `__enter__` (es decir, antes de ejecutar el código de configuración de su gestor de contexto).

No es un requisito absoluto definir `__init__` para su gestor de contexto (como `__enter__` y `__exit__` son todo lo que realmente necesita), pero a veces puede ser útil hacerlo, ya que le permite separar cualquier actividad de inicialización de cualquier actividad de configuración. Cuando creamos un gestor de contexto para usar con nuestras conexiones de base de datos (más adelante en este capítulo), definimos `__init__` para inicializar nuestras credenciales de conexión a la base de datos. Hacerlo no es absolutamente necesario, pero creemos que ayuda a mantener las cosas bien y ordenadas, y hace que nuestro código de clase del gestor de contexto sea más fácil de leer y entender.

### **You've Already Seen a Context Manager in Action**

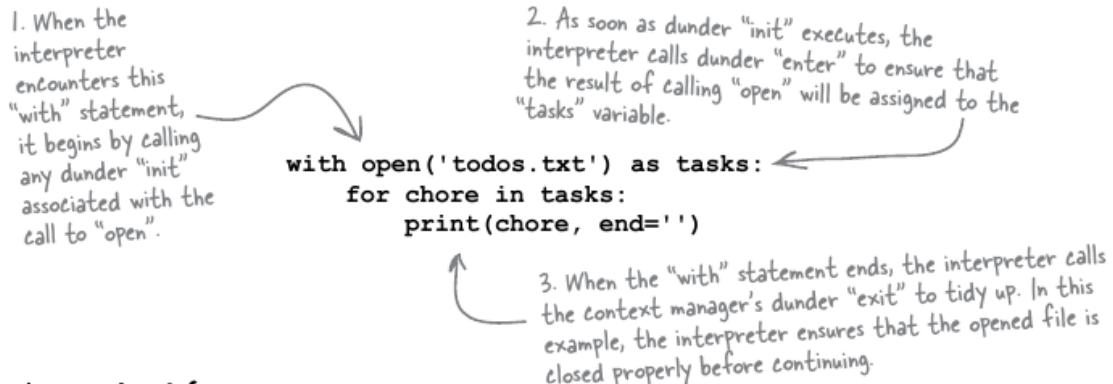
### **Ya has visto un gestor de contexto en acción**

En primer lugar encontró una declaración `with` en el capítulo 6 cuando utilizó uno para garantizar que un archivo abierto anteriormente se cerró automáticamente una vez que su declaración asociada terminó. Recuerde cómo este código abrió el archivo `todos.txt`, luego leyó y mostró cada línea en el archivo uno por uno, antes de cerrar automáticamente el archivo (gracias al hecho de que `open` es un gestor de contexto):

```
with open('todos.txt') as tasks:  
    for chore in tasks:  
        print(chore, end='')
```

↗ Su primera declaración "with" (tomada del capítulo 6).

Echemos un vistazo a esto con la sentencia, destacando donde entran dunder, dunder `exit` y dunder `init`. Hemos numerado cada una de las anotaciones para ayudarle a entender el orden en que los dunders se ejecutan. Tenga en cuenta que no vemos aquí el código de inicialización, configuración o desconexión; Sólo sabemos (y confiamos en) que esos métodos se ejecutan "detrás de las escenas" cuando sea necesario:



Traducimos:

1. Cuando el intérprete encuentra esta sentencia "with", comienza llamando a cualquier dunder "init" asociado con la llamada a "open".
2. Tan pronto como el dunder "init" se ejecuta, el intérprete llama dunder "enter" para asegurar que el resultado de llamar "open" se asignará a la variable "task".
3. Cuando la instrucción "with" finaliza, el intérprete llama al comando dunder del gestor de contexto "exit" para ordenar, en este ejemplo el intérprete asegura que el archivo abierto se cierre correctamente antes de continuar.

## ***What's required from you***

### ***Qué se requiere de usted***

Antes de llegar a crear nuestro propio gestor de contexto (con la ayuda de una nueva clase), revisemos lo que el protocolo de gestión de contexto espera que proporcione para conectarlo a la sentencia with. Debe crear una clase que proporcione:

1. un método `__init__` para realizar la inicialización (si es necesario);
2. un método `__enter__` para hacer cualquier instalación; y
3. un método `__exit__` para hacer cualquier desmontaje (a.k.a. ordenar).

Armado con este conocimiento, vamos a crear ahora una clase de gestor de contexto, escribiendo estos métodos uno por uno, mientras tomamos prestado de nuestro código de base de datos existente según sea necesario.

## ***Cre ate a New Context Manager Class***

### ***Crear una nueva clase de gestor de contexto***

Para empezar, necesitamos darle un nombre a nuestra nueva clase. Además, vamos a poner nuestro nuevo código de clase en su propio archivo, para que podamos reutilizarlo fácilmente (recuerde: cuando pones el código de Python en un archivo separado se convierte en un módulo, que se puede importar a otros programas de Python según sea necesario).

Llamemos a nuestro nuevo archivo `DBcm.py` (abreviado para el gestor de contexto de base de datos), y vamos a llamar a nuestra nueva clase `UseDatabase`. Asegúrese de crear el

archivo `DBcm.py` en la misma carpeta que contiene actualmente su código de webapp, ya que es su aplicación web la que va a importar la clase `UseDatabase` (una vez que lo haya escrito).

Utilice su editor favorito (o IDLE), cree una nueva ventana de edición y guarde el nuevo archivo vacío como `DBcm.py`. Sabemos que para que nuestra clase se ajuste al protocolo de gestión de contexto tiene que:

1. Proporcionar un método `__init__` que realice la inicialización;
2. Proporcionar un método `__enter__` que incluya cualquier código de configuración; y
3. Proporcione un método `__exit__` que incluya cualquier código de desmontaje.

**Recuerde: use CamelCase al nombrar una clase en Python.**

Por ahora, vamos a agregar tres definiciones "vacías" para cada uno de estos métodos requeridos a nuestro código de clase. Un método vacío contiene una instrucción de paso único. Aquí está el código hasta ahora:

This is what our "DBcm.py" file looks like in IDLE. At the moment, it's made up from a single "import" statement, together with a class called "UseDatabase" that contains three "empty" methods.

```
import mysql.connector

class UseDatabase:

    def __init__(self):
        pass

    def __enter__(self):
        pass

    def __exit__(self):
        pass
```

Ln: 14 Col: 0

Traducción:

- Este es el aspecto de nuestro archivo "DBcm.py" en IDLE. En este momento, se compone de una sola instrucción "import", junto con una clase llamada "UseDatabase" que contiene tres métodos "vacíos".

Tenga en cuenta que en la parte superior del archivo `DBCm.py` hemos incluido una instrucción de importación, que incluye la funcionalidad de `MySQL Connector` (de la que depende nuestra nueva clase).

Todo lo que tenemos que hacer ahora es mover los bits pertinentes de la función `log_request` en el método correcto dentro de la clase `UseDatabase`. Bueno ... cuando

decimos nosotros, en realidad te refiero a ti. Es hora de enrollar sus mangas y escribir un código de método.

### **Initialize the Class with the Database Config**

#### **Iniciar la clase con la configuración de base de datos**

Recordemos cómo intentamos utilizar el gestor de contexto UseDatabase. Aquí está el código del último capítulo, reescrito para usar una instrucción with, que utiliza el gestor de contexto UseDatabase que está a punto de escribir:

```
from DBcm import UseDatabase
dbconfig = { 'host': '127.0.0.1',
             'user': 'vsearch',
             'password': 'vsearchpasswd',
             'database': 'vsearchlogDB', }

with UseDatabase(dbconfig) as cursor:
    _SQL = """insert into log
              (phrase, letters, ip, browser_string, results)
              values
              (%s, %s, %s, %s, %s)"""
    cursor.execute(_SQL, (req.form['phrase'],
                          req.form['letters'],
                          req.remote_addr,
                          req.user_agent.browser,
                          res, ))
```

The code is annotated with several comments:

- A callout points to the first line: "Import the context manager from the "DBcm.py" file."
- A callout points to the variable assignment: "Here's the database connection characteristics."
- A callout points to the with statement: "The "UseDatabase" context manager expects to receive a dictionary of database connection characteristics."
- A callout points to the SQL query: "The context manager returns a "cursor"."
- A callout points to the final part of the code: "This code stays the same as before."

Traducimos:

1. Aquí están las características de conexión a la base de datos.
2. Importe el administrador de contexto desde el archivo "Dbcm.py".
3. El gestor de contexto "UseDatabase" espera recibir un diccionario de características de conexión de base de datos.
4. El gestor de contexto devuelve un "cursor".
5. Este código permanece igual que antes.



Comencemos con el método `__init__`, que usaremos para inicializar cualquier atributo en la clase UseDataBase. Basado en el uso mostrado anteriormente, el método dunder init acepta un solo argumento, que es un diccionario de características de conexión llamado config (que deberás agregar a la línea def abajo). Arreglemos para que la configuración se guarde como un atributo

llamado configuración. Agregue el código necesario para guardar el diccionario en el atributo de configuración en el código de dunder init:

```
import mysql.connector  
  
class UseDatabase:  
    def __init__(self, .....)  
  
Save the configuration  
dictionary to an attribute.  
  
Complete the  
"def" line.  
  
Is there  
anything  
missing  
from here?
```

## Traducción:

1. Guarde el diccionario de configuración en un atributo.
  2. Complete la línea "def".
  3. ¿Falta algo de aquí?



Comenzó con el método `__init__`, que debía inicializar cualquier atributo en la clase `UseDataBase`. El método dunder `init` acepta un solo argumento, que es un diccionario de características de conexión llamado `config` (que se necesita para agregar a la línea `def` a continuación). Deberías organizar la configuración para que se guarde en un atributo llamado `configuración`. Deberías agregar el código necesario para guardar el diccionario en el atributo de configuración en el código de dunder `init`:

```
import mysql.connector  
  
class UseDatabase:  
  
    def __init__(self, config: dict) -> None:  
        self.configuration = config
```

The value of the "config" argument is assigned to an attribute called "configuration". Did you remember to prefix the attribute with "self"?

Dunder "init" accepts a single dictionary, which we're calling "config".

The (optional) "None" annotation confirms that this method has no return value (which is nice to know), and the colon terminates the "def" line.

### Traducción:

1. El valor del argumento "config" se asigna a un atributo llamado "configuration". ¿Se acordó de prefijar el atributo con "self"?

2. Dunder "init" acepta un solo diccionario, al que llamamos "config".
3. La anotación (opcional) "Ninguna" confirma que este método no tiene valor de retorno (que es bueno saber), y el colon termina la línea "def".

## **Your context manager begins to take shape**

### **Su gestor de contexto comienza a tomar forma**

Con el método de `init` de dunder escrito, puede pasar a codificar el método dunder `enter` (`__enter__`). Antes de hacerlo, asegúrese de que el código que ha escrito hasta ahora coincide con el nuestro, que se muestra a continuación en IDLE:

```

import mysql.connector

class UseDatabase:

    def __init__(self, config: dict) -> None:
        self.configuration = config

    def __enter__(self):
        pass

    def __exit__(self):
        pass

```

Ln: 14 Col: 0

## **Perform Setup with Dunder "enter"**

### **Realizar la configuración con Dunder "enter"**

El método dunder `enter` proporciona un lugar para ejecutar el código de configuración `setup` que debe ejecutarse antes de que se ejecute `with` en su sentencia. Recuerde el código de la función `log_request` que controla esta configuración o `setup`:

Here's the setup code from the "log\_request" function.

```

...
dbconfig = { 'host': '127.0.0.1',
             'user': 'vsearch',
             'password': 'vsearchpasswd',
             'database': 'vsearchlogDB', }

conn = mysql.connector.connect(**dbconfig)
cursor = conn.cursor()

_SQL = """insert into log
            (phrase, letters, ip, browser_string, results)

```

Este código de configuración usa el diccionario de características de conexión para conectarse a MySQL, luego crea un cursor de base de datos en la conexión (que deberemos enviar comandos a la base de datos desde nuestro código Python). Como este código de

configuración es algo que harás cada vez que escribas código para hablar con tu base de datos, hagamos este trabajo en tu clase de gestor de contexto para que puedas reutilizarlo más fácilmente.



El método dunder enter (`__enter__`) necesita utilizar las características de configuración almacenadas en `self.configuration` para conectarse a la base de datos y crear un cursor. Aparte del argumento `self` obligatorio, dunder enter no toma otros argumentos, pero necesita devolver el cursor. Complete el código para el método siguiente:

```
def __enter__(self) ...  
    .....  
    .....  
    :  
    .....  
  
    return .....  
  
Add the setup code here.  
Don't forget to return the cursor.  
Can you think of an appropriate annotation?
```



El método dunder enter (`__enter__`) utiliza las características de configuración almacenadas en `self.configuration` para conectarse a la base de datos y crear un cursor. Aparte del argumento `self` obligatorio, dunder enter no toma otros argumentos, pero necesita devolver el cursor. Debes completar el código para el siguiente método:

```
def __enter__(self) ...  
    .....  
    .....  
    :  
    .....  
  
    self.conn = mysql.connector.connect(**self.configuration)  
    self.cursor = self.conn.cursor()  
    .....  
  
    return self.cursor  
  
Did you remember to prefix all attributes with "self"?  
Return the cursor.  
Be sure to refer to "self.configuration" here as opposed to "dbconfig".  
This annotation tells users of this class what they can expect to be returned from this method.
```

Traducimos:

1. ¿Te acuerdas de prefijar todos los atributos con "self"?
2. Devuelve el cursor.

3. Asegúrese de hacer referencia a "self.configuration" aquí en lugar de "dbconfig".
4. Esta anotación informa a los usuarios de esta clase de lo que pueden esperar ser devueltos de este método.

### ***Don't forget to prefix all attributes with “self”***

#### ***No se olvide de prefijar todos los atributos con "self"***

Usted puede ser sorprendido que hemos designado conn y cursor como atributos en dunder entrar (prefijando cada uno con el self). Hemos hecho esto con el fin de asegurar tanto conn y cursor sobrevivir cuando el método termina, ya que ambas variables son necesarias en el método `__exit__`. Para asegurarnos de que esto ocurra, agregamos el prefijo propio a las variables conn y cursor; Al hacerlo, los agrega a la lista de atributos de la clase.

Antes de llegar a escribir dunder exit, confirme que su código coincide con el nuestro:

```

DBcm.py - /Users/paul/Desktop/_NewBook/ch09/webapp/DBcm.py (3.5.1)*
import mysql.connector

class UseDatabase:

    def __init__(self, config: dict) -> None:
        self.configuration = config

    def __enter__(self) -> 'cursor':
        self.conn = mysql.connector.connect(**self.configuration)
        self.cursor = self.conn.cursor()
        return self.cursor

    def __exit__(self):
        pass

```

Ln: 16 Col: 0

### ***Perform Teardown with Dunder “exit”***

#### ***Realizar Teardown con Dunder "exit"***

El método de salida dunder proporciona un lugar para ejecutar el código de desmontaje que debe ejecutarse cuando termina la instrucción `with`. Recuerde el código de la función `log_request` que controla el desmontaje:

```

...
cursor.execute(_SQL, (req.form['phrase'],
                      req.form['letters'],
                      req.remote_addr,
                      req.user_agent.browser,
                      res, ))

```

conn.commit()  
cursor.close()  
conn.close()

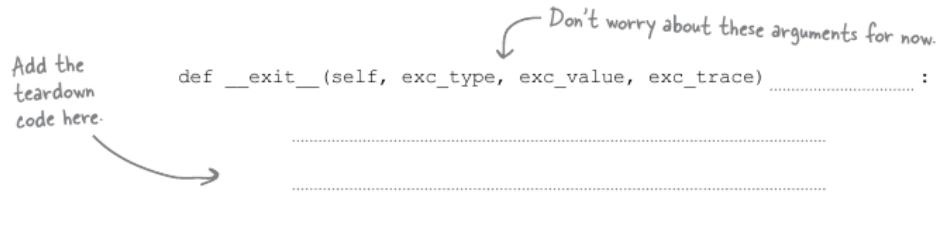
This is the teardown code.

El código de desmontaje transmite cualquier dato a la base de datos, luego cierra el cursor y la conexión. Este desmontaje sucede cada vez que interactúas con la base de datos, así que vamos a añadir este código a tu clase de gestor de contexto moviendo estas tres líneas en salida `exit` de dunder.

Antes de hacer esto, sin embargo, usted necesita saber que hay una complicación con la salida de dunder, que tiene que ver con el manejo de las excepciones que podrían ocurrir en la suite `with`s. Cuando algo sale mal, el intérprete siempre notifica `__exit__` pasando tres argumentos al método: `exec_type`, `exc_value` y `exc_trace`. Su línea `def` necesita tener esto en cuenta, por lo que hemos añadido los tres argumentos al código siguiente. Dicho esto, vamos a ignorar este mecanismo de manejo de excepciones por ahora, pero volveremos a él en un capítulo posterior cuando discutamos lo que puede salir mal y cómo se puede manejar (así que permanezca atento).

### Sharpen your pencil -

El código de desmontaje es donde usted hace su ordenación. Para este gestor de contexto, ordenar implica asegurar que cualquier dato se confía a la base de datos antes de cerrar tanto el cursor como la conexión. Agregue el código que cree que necesita al método siguiente.



The diagram shows a snippet of Python code for a `__exit__` method. A handwritten note on the left says "Add the teardown code here." with an arrow pointing to the start of the method definition. Another handwritten note on the right says "Don't worry about these arguments for now." with an arrow pointing to the parameters `exc_type`, `exc_value`, and `exc_trace`.

```
def __exit__(self, exc_type, exc_value, exc_trace):
```

### Sharpen your pencil - Solution

El código de desmontaje es donde usted hace su ordenación. Para este gestor de contexto, ordenar implica asegurar que cualquier dato se confía a la base de datos antes de cerrar tanto el cursor como la conexión. Deberías agregar el código que crees que necesitas al método siguiente.

```

def __exit__(self, exc_type, exc_value, exc_trace) -> None :
    self.conn.commit()
    self.cursor.close()
    self.conn.close()

```

The previously saved attributes are used to commit unsaved data, as well as close the cursor and connection. As always, remember to prefix your attribute names with "self".

Don't worry about these arguments for now.

This annotation confirms that this method has no return value; such annotations are optional but are good practice..

Traducimos:

1. Los atributos previamente guardados se utilizan para enviar datos no guardados, así como para cerrar el cursor y la conexión. Como siempre, recuerde prefijar sus nombres de atributo con "self".
2. Esta anotación confirma que este método no tiene valor de retorno; Tales anotaciones son opcionales pero son una buena práctica.
3. No te preocupes por estos argumentos por ahora.

## **Your context manager is ready for testing**

### **Su gestor de contexto está listo para la prueba**

Con el código de salida exit dunder escrito, ahora es el momento de probar su gestor de contexto antes de integrarlo en su código webapp. Como ha sido nuestra costumbre, primero probaremos este nuevo código en el intérprete de comandos de Python (el >>>). Antes de hacer esto, realice una última comprobación para asegurarse de que su código es el mismo que el nuestro:

```

DBcm.py - /Users/paul/Desktop/_NewBook/ch09/webapp/DBcm.py (3.5.1)
import mysql.connector

class UseDatabase:

    def __init__(self, config: dict) -> None:
        self.configuration = config

    def __enter__(self) -> 'cursor':
        self.conn = mysql.connector.connect(**self.configuration)
        self.cursor = self.conn.cursor()
        return self.cursor

    def __exit__(self, exc_type, exc_value, exc_trace) -> None:
        self.conn.commit()
        self.cursor.close()
        self.conn.close()

Ln: 18 Col: 0

```

La clase de administrador de contexto "UseDatabase" se completada.

Una clase "real" incluiría documentación, pero la hemos eliminado de este código para ahorrar espacio (en esta página). Las descargas de este libro siempre incluyen comentarios.



Con el código de DBcm.py en una ventana de edición IDLE, presione F5 para probar su gestor de contexto:

Importe la clase del gestor de contexto desde el archivo de módulo "DBcm.py".

Utilice el administrador de contexto para enviar algo de SQL al servidor y recuperar algunos datos.

```
Python 3.5.1 Shell
>>> from DBcm import UseDatabase
>>>
>>> dbconfig = { 'host': '127.0.0.1',
   'user': 'vsearch',
   'password': 'vsearchpasswd',
   'database': 'vsearchlogDB', }

>>> with UseDatabase(dbconfig) as cursor:
    _SQL = """show tables"""
    cursor.execute(_SQL)
    data = cursor.fetchall()

>>> data
[('log',)]
```

The screenshot shows a Python 3.5.1 Shell window. The code imports the UseDatabase class from DBcm. It defines a dbconfig dictionary with host, user, password, and database keys. A context manager is used to create a cursor object, which executes a SQL query to show tables. The result is a list containing a single tuple ('log',). The shell indicates it's on line 45, column 4.

Coloque las características de conexión en un diccionario.

Los datos devueltos pueden parecer un poco extraños ... hasta que recuerdes que la llamada "cursor.fetchall" devuelve una lista de tuplas, con cada tupla correspondiente a una fila de resultados (como devueltos de la base de datos)

***There's not much code here, is there?***

***No hay mucho código aquí, ¿verdad?***

Con suerte, estás viendo el código de arriba y decidiendo que no hay mucho. A medida que ha movido con éxito una parte de su código de gestión de bases de datos en la clase UseDatabase, la inicialización, la instalación y el desmontaje se gestionan ahora "detrás de las escenas" por su gestor de contexto. Todo lo que tiene que hacer es proporcionar las características de conexión y la consulta SQL que desea ejecutar, el gestor de contexto hace todo lo demás. Su código de configuración y desmontaje se reutiliza como parte del administrador de contexto. También es más claro cuál es la "carne" de este código: obtener datos de la base de datos y procesarla. El gestor de contexto oculta los detalles de conexión / desconexión a / de la base de datos (que siempre van a ser los mismos), dejándote así libre para concentrarte en lo que estás tratando de hacer con tus datos.

***Actualizemos tu aplicación web para usar tu administrador de contexto.***

***Reconsidering Your Webapp Code, 1 of 2***

***Reconsiderando su código Webapp, 1 de 2***

Ha pasado bastante tiempo desde que has considerado el código de tu webapp.

La última vez que trabajó en él (en el Capítulo 7), actualizó la función log\_request para guardar la solicitud web de la aplicación web en la base de datos MySQL. La razón por la

que iniciamos el camino hacia el aprendizaje de las clases (en el Capítulo 8) fue determinar la mejor manera de compartir el código de base de datos que agregó a `log_request`. Ahora sabemos que la mejor manera (para esta situación) es utilizar la clase de administrador de contexto `UseDatabase` recién escrita.

Además de modificar `log_request` para utilizar el gestor de contexto, la otra función del código que necesitamos modificar para trabajar con los datos de la base de datos se denomina `view_the_log` (que actualmente funciona con el archivo de texto `vsearch.log`). Antes de llegar a modificar ambas funciones, recordemos el estado actual del código de la aplicación web (en esta página y en la siguiente). Hemos destacado los bits que necesitan ser trabajados:

*El código de tu webapp está en el archivo "vsearch4web.py" de tu carpeta "webapp".*

```
from flask import Flask, render_template, request, escape
from vsearch import search4letters

import mysql.connector ← We need to
app = Flask(__name__) import "DBcm"
here instead.

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""
    dbconfig = {'host': '127.0.0.1',
                'user': 'vsearch',
                'password': 'vsearchpasswd',
                'database': 'vsearchlogDB', }

    conn = mysql.connector.connect(**dbconfig)
    cursor = conn.cursor()
    _SQL = """insert into log
              (phrase, letters, ip, browser_string, results)
              values
              (%s, %s, %s, %s, %s)"""
    cursor.execute(_SQL, (req.form['phrase'],
                          req.form['letters'],
                          req.remote_addr,
                          req.user_agent.browser,
                          res, ))
    conn.commit()
    cursor.close()
    conn.close()

This code has to
be amended to use
the "UseDatabase"
context manager. →
```

Traducimos:

1. Este código tiene que ser modificado para utilizar el gestor de contexto "UseDatabase".
2. Necesitamos importar "DBcm" aquí en su lugar.

## ***Reconsidering Your Webapp Code, 2 of 2***

## ***Reconsiderando su código Webapp, 2 de 2***

```

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    """Extract the posted data; perform the search; return results."""
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    log_request(request, results)
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    """Display this webapp's HTML form."""
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!!')

@app.route('/viewlog')
def view_the_log() -> 'html':
    """Display the contents of the log file as a HTML table."""
    contents = []
    with open('vsearch.log') as log:
        for line in log:
            contents.append([])
            for item in line.split('|'):
                contents[-1].append(escape(item))
    titles = ('Form Data', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,)

if __name__ == '__main__':
    app.run(debug=True)

```

This code needs to be amended to use the data in the database via the "UseDatabase" context manager.

Traduciendo:

Este código debe ser modificado para utilizar los datos en la base de datos a través del gestor de contexto "UseDataBase".

### ***Recalling the “log\_request” Function***

### ***Recordando la función “log\_request”***

Cuando se trata de modificar la función log\_request para utilizar el gestor de contexto UseDatabase, una gran parte del trabajo ya se ha hecho para usted (ya que le mostramos el código que estábamos disparando antes).

Eche un vistazo a log\_request una vez más. Por el momento, el diccionario de características de conexión de base de datos (dbconfig en el código) se define dentro de log\_request. Como querrás usar este diccionario en la otra función que tienes que modificar (view\_the\_log), vamos a moverla de la función log\_request para que puedas compartirla con otras funciones según sea necesario:

```

def log_request(req: 'flask_request', res: str) -> None:

    dbconfig = {'host': '127.0.0.1',
                'user': 'vsearch',
                'password': 'vsearchpasswd',
                'database': 'vsearchlogDB', }

    conn = mysql.connector.connect(**dbconfig)
    cursor = conn.cursor()
    _SQL = """insert into log
              (phrase, letters, ip, browser_string, results)
              values
              (%s, %s, %s, %s, %s)"""
    cursor.execute(_SQL, (req.form['phrase'],
                          req.form['letters'],
                          req.remote_addr,
                          req.user_agent.browser,
                          res, ))
    conn.commit()
    cursor.close()
    conn.close()

```

Let's move this dictionary out of the function so it can be shared with other functions as required.

### Traducimos:

Vamos a mover este diccionario fuera de la función para que pueda ser compartida con otras funciones según sea necesario.

En lugar de mover dbconfig en el espacio global de nuestra webapp, sería útil si de alguna manera pudiéramos añadirlo a la configuración interna de nuestra webapp.

Como suerte, Flask (como muchos otros frameworks web) viene con un mecanismo de configuración incorporado: un diccionario (que Flask llama app.config) le permite ajustar algunos de los ajustes internos de su webapp. Como app.config es un diccionario regular de Python, puede agregar sus propias claves y valores a él según sea necesario, que es lo que usted hará para los datos en dbconfig.

El resto del código log\_request puede ser modificado para usar UseDatabase.

### *Amending the “log\_request” Function*

### *Modificación de la función "log\_request"*

Ahora que hemos aplicado los cambios a nuestra aplicación web, nuestro código se ve así:

```

vsearch4web.py - /Users/paul/Desktop/_NewBook/ch09/webapp/vsearch4web.py (3.5.1)
from flask import Flask, render_template, request, escape
from vsearch import search4letters

from DBcm import UseDatabase

app = Flask(__name__)

app.config['dbconfig'] = {'host': '127.0.0.1',
                         'user': 'vsearch',
                         'password': 'vsearchpasswd',
                         'database': 'vsearchlogDB', }

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """insert into log
                  (phrase, letters, ip, browser_string, results)
                  values
                  (%s, %s, %s, %s, %s)"""
        cursor.execute(_SQL, (req.form['phrase'],
                             req.form['letters'],
                             req.remote_addr,
                             req.user_agent.browser,
                             res, ))

```

Ln: 10 Col: 0

Traduciendo:

1. Hemos cambiado la antigua declaración de "importación" a esta actualización.
2. Hemos ajustado el código para utilizar "UseDatabase", asegurándose de pasar en la configuración de la base de datos de "app.config".
3. Añadimos el diccionario de características de conexión a la configuración del webapp.

Cerca de la parte superior del archivo, hemos reemplazado la instrucción `import mysql.connector` con una instrucción de importación que toma `UseDatabase` de nuestro módulo `DBcm`. El archivo `DBcm.py` incluye la instrucción `import mysql.connector` en su código, de ahí la eliminación de `mysql.connector` de importación de este archivo (ya que no queremos importarlo dos veces).

También hemos trasladado el diccionario de características de conexión a la base de datos en la configuración de nuestra webapp. Y hemos modificado el código `log_request` para usar nuestro gestor de contexto.

Después de todo su trabajo en clases y gestores de contexto, debería ser capaz de leer y entender el código mostrado arriba.

Ahora vamos a modificar la función `view_the_log`. Asegúrese de que su código de webapp sea modificado para ser exactamente como el nuestro antes de girar la página.

## **Recalling the “view\_the\_log” Function**

### **Recuperación de la función "view\_the\_log"**

Echemos un vistazo largo y duro al código en `view_the_log`, ya que ha sido bastante tiempo desde que lo ha considerado en detalle. Para recapitular, la versión actual de esta función extrae los datos registrados del archivo de texto `vsearch.log`, lo convierte en una lista de listas (denominada `contents`) y luego envía los datos a una plantilla denominada `viewlog.html`:

```
@app.route('/viewlog')
def view_the_log() -> 'html':
    contents = []
    with open('vsearch.log') as log:
        for line in log:
            contents.append([])
            for item in line.split('|'):
                contents[-1].append(escape(item))

    titles = ('Form Data', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents)

Grab each line of data from the file, and then transform it into a list of escaped items, which are appended to the "contents" list.
The processed log data is sent to the template for display.
```

Traducimos:

1. Coge cada línea de datos del archivo y luego transforma en una lista de elementos de escape, que se añaden a la lista de "contenido".
2. Los datos de registro procesados se envían a la plantilla para su visualización.

Esto es lo que parece la salida cuando se visualiza la plantilla `viewlog.html` con los datos de la lista de contenido de las listas. Esta funcionalidad está actualmente disponible para su aplicación web a través de la URL `/viewlog`:

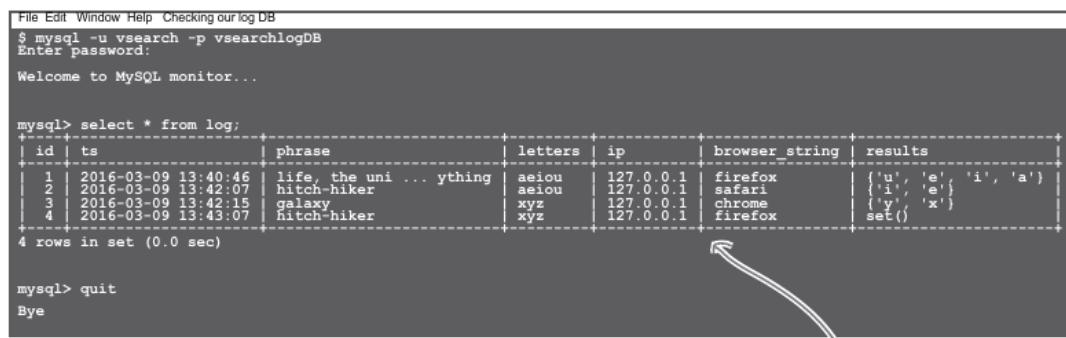
Los datos de "contenido" se muestran en el formulario. Observe cómo los datos del formulario ("frase" y "letras") se presentan en una sola columna.

Form Data	Remote_addr	User_agent	Results
ImmutableMultiDict([('letters', 'aeiou'), ('phrase', 'hitch-hiker')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9	{'e', 'i'}
ImmutableMultiDict([('letters', 'aeiou'), ('phrase', 'life, then you')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9	{'e', 'u'}

## **It's Not Just the Code That Changes**

### **No es sólo el código que cambia**

Antes de entrar y cambiar el código en `view_the_log` para usar el gestor de contexto, hagamos una pausa para considerar los datos almacenados en la tabla de registro log de la base de datos. Cuando probó su código `log_request` inicial en el Capítulo 7, pudo iniciar sesión en la consola MySQL y, a continuación, comprobar que los datos se habían guardado. Recuerde esta sesión de consola de MySQL desde antes:



The screenshot shows a terminal window with the MySQL monitor. The user has run a query to select all columns from the 'log' table. The output is a table with columns: id, ts, phrase, letters, ip, browser, string, and results. There are four rows of data. An arrow points from the text 'Los datos de registro guardados en una tabla de base de datos' to the table output.

id	ts	phrase	letters	ip	browser	string	results
1	2016-03-09 13:40:46	life, the uni ... ything	aeiou	127.0.0.1	firefox		{'u', 'e', 'i', 'a'}
2	2016-03-09 13:42:07	hitch-hiker	aeiou	127.0.0.1	safari		{'i', 'e'}
3	2016-03-09 13:42:15	galaxy	xyz	127.0.0.1	chrome		{'y', 'x'}
4	2016-03-09 13:43:07	hitch-hiker	xyz	127.0.0.1	firefox		set()

mysql> quit  
Bye

Los datos de registro  
guardados en una tabla de  
base de datos

Si considera los datos anteriores en relación con lo que se almacena actualmente en `vsearch.log`, es evidente que ya no se necesita parte del procesamiento `view_the_log` ya que los datos se almacenan en una tabla. A continuación, se muestra un fragmento de los datos de registro en el archivo `vsearch.log`:

```
ImmutableMultiDict([('phrase', 'galaxy'), ('letters', 'xyz')])|127.0.0.1|Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36|('x', 'y')
```

Los datos de registro se guardan  
como una cadena larga en el archivo  
"vsearch.log".

Parte del código actualmente en `view_the_log` sólo existe porque los datos de registro se almacenan actualmente como una colección de cadenas largas (delimitadas por barras verticales) en el archivo `vsearch.log`. Ese formato funcionó, pero necesitamos escribir código extra para tener sentido.

Esto no es el caso con los datos en la tabla de registro, ya que está "estructurado por defecto". Esto debería significar que no necesita realizar ningún procesamiento adicional dentro de `view_the_log`: todo lo que tiene que hacer es extraer los datos de la tabla, Que-felizmente-se le devuelve como una lista de tuplas (gracias al método `fetchall` de DB-API).

Además, los datos de la tabla de registro separan el valor de la frase del valor de las letras. Si realiza un pequeño cambio en el código de representación de la plantilla, la salida producida puede mostrar cinco columnas de datos (a diferencia de las cuatro actuales), lo que hace que el navegador se muestre aún más útil y fácil de leer.

### ***Amending the “view\_the\_log” Function***

#### ***Modificación de la función "view\_the\_log"***

Sobre la base de todo lo comentado en las últimas páginas, tiene dos cosas que hacer para modificar su código actual `view_the_log`:

1. Coge los datos del registro de la tabla de la base de datos (en contraposición al archivo).
2. Ajuste la lista de títulos para admitir cinco columnas (en lugar de cuatro).

Si te estás rascando la cabeza y preguntándote por qué esta pequeña lista de enmiendas no incluye el ajuste de la plantilla `viewlog.html`, no te preguntes nada más: no necesitas hacer ningún cambio en ese archivo, ya que la plantilla actual procesa bastante felizmente Cualquier número de títulos y cualquier cantidad de datos que le envíes.

Aquí está el código actual de la función `view_the_log`, que está a punto de modificar:

```
@app.route('/viewlog')
def view_the_log() -> 'html':
    contents = []
    with open('vsearch.log') as log:
        for line in log:
            contents.append([])
            for item in line.split('|'):
                contents[-1].append(escape(item))

    titles = ('Form Data', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,)
```

As a result of task #1 above, this code needs to be replaced.

As a result of task #2 above, this line needs to be amended.

Traducimos:

1. Como resultado de la tarea # 1 anterior, este código debe ser reemplazado.
2. Como resultado de la tarea # 2 anterior, esta línea necesita ser enmendada.

### ***Here's the SQL quer y you'll need***

#### ***Aquí está la consulta SQL que necesitará***

Antes del próximo ejercicio (donde actualizarás la función `view_the_log`), he aquí una consulta SQL que, cuando se ejecuta, devuelve todos los datos almacenados en la base de datos MySQL de la aplicación web. Los datos se devuelven al código Python de la base de

datos como una lista de tuplas. Necesitará utilizar esta consulta en el ejercicio de la siguiente página:

```
select phrase, letters, ip, browser_string, results  
from log
```



Esta es la función `view_the_log`, que tiene que ser modificada para usar los datos en la tabla de registro. Su trabajo es proporcionar el código que falta. Asegúrese de leer las anotaciones para obtener sugerencias sobre lo que debe hacer:

```
@app.route('/viewlog')  
def view_the_log() -> 'html':  
    with .....:  
        .....  
        _SQL = """select phrase, letters, ip, browser_string, results  
                from log"""  
  
        .....  
        .....  
        titles = (....., ..... , 'Remote_addr', 'User_agent', 'Results')  
  
        .....  
        .....  
        return render_template('viewlog.html',  
                               the_title='View Log',  
                               the_row_titles=titles,  
                               the_data=contents,)
```

Which column titles are missing from here?

Use your context manager here, and don't forget the cursor.

Send the query to the server, then fetch the results.

Traducimos:

1. ¿Qué títulos de columna faltan de aquí?
2. Utilice su gestor de contexto aquí, y no olvide el cursor.
3. Envíe la consulta al servidor y, a continuación, obtenga los resultados.

Sólo voy a tomar nota de lo que está pasando aquí. No sólo es mi nuevo código más corto que lo que tenía antes, es más fácil para mí entender y leer, también.

**Sí, ese fue nuestro objetivo todo el tiempo.**

Al mover los datos de registro en una base de datos MySQL, ha eliminado el requisito de crear y luego procesar un formato de archivo personalizado basado en texto. Además, al volver a utilizar su gestor de contexto, ha simplificado sus interacciones con MySQL cuando trabaja en Python. ¿Qué es no gustar?



## Sharpen your pencil Solution

Esta es la función `view_the_log`, que tiene que ser modificada para usar los datos en la tabla de registro `log`. Su trabajo consistía en proporcionar el código que faltaba.

```
@app.route('/viewlog')
def view_the_log() -> 'html':
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """select phrase, letters, ip, browser_string, results
                  from log"""
        cursor.execute(_SQL)
        contents = cursor.fetchall()
    } ← Send the query to the server, then
         fetch the results. Note the assignment
         of the fetched data to "contents".
    titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
    Add in the
    correct
    column names.
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,)
```

Traducimos:

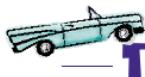
1. Agregue los nombres de columna correctos.
2. Esta es la misma línea de código de la función "log\_request".
3. Envíe la consulta al servidor, obtenga los resultados. Observe la asignación de los datos obtenidos a "contents".

## It's nearly time for one last Test Drive Es casi la hora de un último Test Drive

Antes de tomar esta nueva versión de su webapp para un giro, tómese un momento para confirmar que su función `view_the_log` es la misma que la nuestra:

```
vsearch4web.py - /Users/paul/Desktop/_NewBook/ch09/webapp/vsearch4web.py (3.5.1)

@app.route('/viewlog')
def view_the_log() -> 'html':
    """Display the contents of the log file as a HTML table."""
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """select phrase, letters, ip, browser_string, results
                  from log"""
        cursor.execute(_SQL)
        contents = cursor.fetchall()
    titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,)
```



## Test Drive

Es hora de tomar su webapp lista para una vuelta.

Asegúrese de que el archivo `DBcm.py` esté en la misma carpeta que su archivo `vsearch4web.py` y, a continuación, inicie su aplicación web de la forma habitual en su sistema operativo:

- Utilizar `python3 vsearch4web.py` en Linux / Mac OS X
- Utilice `py -3 vsearch4web.py` en Windows.

Utilice su navegador para ir a la página principal de su webapp (que se ejecuta en `http://127.0.0.1:5000`), a continuación, introduzca un puñado de búsquedas. Una vez que hayas confirmado que la función de búsqueda funciona, utiliza la URL `/viewlog` para ver el contenido de tu registro en la ventana del navegador.

Aunque es muy probable que las búsquedas que ingrese difieran de las nuestras, aquí es lo que vimos en nuestra ventana del navegador, lo que confirma que todo funciona como se esperaba:



Phrase	Letters	Remote_addr	User_agent	Results
life, the universe, and everything	aeiou	127.0.0.1	firefox	{'u', 'e', 'i', 'a'}
hitch-hiker	aeiou	127.0.0.1	safari	{'i', 'e'}
galaxy	xyz	127.0.0.1	chrome	{'y', 'x'}
hitch-hiker	xyz	127.0.0.1	firefox	set()
lightning in a bottle	aeiou	127.0.0.1	firefox	{'l', 'a', 'o', 'e'}
testing the database-enabled webapp	aeiou	127.0.0.1	firefox	{'e', 'a', 'i'}

Esta salida del navegador confirma que los datos registrados se están leyendo desde la base de datos MySQL cuando se accede a la URL `/viewlog`. Esto significa que el código en `view_the_log` está funcionando, lo que, por cierto, confirma que la función `log_request` funciona igual que lo esperado, ya que pone los datos de registro en la base de datos como resultado de cada búsqueda exitosa.

Sólo si se siente la necesidad, tome unos momentos para iniciar sesión en su base de datos MySQL usando la consola de MySQL para confirmar que los datos se almacenan con

seguridad en su servidor de base de datos. (O simplemente confía en nosotros: basado en lo que nuestra aplicación web está mostrando arriba, lo es.)

### ***All That Remains...***

#### ***Todo lo que queda...***

Ahora es el momento de volver a las preguntas planteadas en el capítulo 7:

- ¿Cuántas solicitudes se han respondido?
- ¿Cuál es la lista de cartas más común?
- ¿De qué direcciones IP proceden las peticiones?
- ¿Qué navegador se utiliza más?

Aunque es posible escribir el código de Python para responder a estas preguntas, no vamos a en este caso, a pesar de que acabamos de pasar este y los dos capítulos anteriores mirando cómo Python y bases de datos trabajan juntos. En nuestra opinión, la creación de código Python para responder a este tipo de preguntas es casi siempre un mal movimiento ...

*Así que si no voy a usar Python para responder a estas preguntas, ¿qué debo usar en su lugar? Aprendí un poco acerca de las bases de datos y SQL mientras trabajaba con el Capítulo 7-¿Serían las consultas SQL un buen ajuste aquí?*

#### ***SQL es definitivamente el camino a seguir.***

Estos tipos de "preguntas de datos" se responden mejor por el mecanismo de consulta de su tecnología de base de datos (que, en MySQL, es SQL). Como verá en la siguiente página, es poco probable que produzca código Python tan rápido como escribir las consultas SQL que necesita.

Saber cuándo usar Python y cuándo no es importante, como es saber lo que diferencia a Python de muchas otras tecnologías de programación. Aunque la mayoría de los lenguajes convencionales admiten clases y objetos, pocos proporcionan algo cercano al protocolo de administración de contexto de Python. (En el capítulo siguiente, conocerá otra característica que establece Python aparte de muchos otros idiomas: decoradores de funciones.)

Antes de llegar al siguiente capítulo, echemos un vistazo rápido (una página) a esas consultas SQL ...

### ***Answering the Data Questions***

#### ***Responder a las preguntas de los datos***

Tomemos las preguntas planteadas primero en el capítulo 7 una por una, respondiendo cada una con la ayuda de algunas consultas de base de datos escritas en SQL.

### **¿Cuántas solicitudes se han respondido?**

Si ya eres un tipo de SQL (o dudette), puede estar burlándose de esta pregunta, ya que en realidad no es mucho más simple. Ya sabes que esta más básica de consultas SQL muestra todos los datos en una tabla de base de datos:

```
select * from log;
```

Para transformar esta consulta en una que informe cuántas filas de datos tiene una tabla, pase el \* en el recuento de funciones SQL, de la siguiente manera:

```
select count(*) from log;
```

No estamos mostrando las respuestas aquí. Si desea verlos, tendrá que ejecutar estas consultas en la consola MySQL (consulte el Capítulo 7 para una actualización).

### **What's the most common list of letters?**

### **¿Cuál es la lista de letras más común?**

La consulta SQL que responde a esta pregunta parece un poco de miedo, pero no es realmente. Aquí está:

```
select count(letters) as 'count', letters
from log
group by letters
order by count desc
limit 1;
```

### **¿De qué direcciones IP provienen las peticiones?**

Los dudes / dudettes del SQL hacia fuera allí están pensando probablemente "que es casi demasiado fácil":

```
select distinct ip from log;
```

### **Which browser is being used the most?**

### **¿Qué navegador se utiliza más?**

La consulta SQL que responde a esta pregunta es una ligera variación en la consulta que respondió a la segunda pregunta:

```
select browser_string, count(browser_string) as 'count'
from log
group by browser_string
order by count desc
limit 1;
```

Así que ahí lo tienes: todas tus preguntas urgentes respondieron con unas cuantas consultas SQL sencillas. Siga adelante y probarlos en el prompt de mysql> antes de comenzar en el próximo capítulo.

## ***Chapter 9's Code, 1 of 2***

```
import mysql.connector

class UseDatabase:

    def __init__(self, config: dict) -> None:
        self.configuration = config

    def __enter__(self) -> 'cursor':
        self.conn = mysql.connector.connect(**self.configuration)
        self.cursor = self.conn.cursor()
        return self.cursor

    def __exit__(self, exc_type, exc_value, exc_trace) -> None:
        self.conn.commit()
        self.cursor.close()
        self.conn.close()
```

Este es el  
código del  
gestor de  
contexto en  
"DBcm.py". ➔

Esta es la primera mitad del código de la  
webapp en "vsearch4web.py"

```
from flask import Flask, render_template, request, escape
from vsearch import search4letters

from DBcm import UseDatabase

app = Flask(__name__)

app.config['dbconfig'] = {'host': '127.0.0.1',
                        'user': 'vsearch',
                        'password': 'vsearchpasswd',
                        'database': 'vsearchlogDB', }

def log_request(req: 'flask_request', res: str) -> None:
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """insert into log
                  (phrase, letters, ip, browser_string, results)
                  values
                  (%s, %s, %s, %s, %s)"""
        cursor.execute(_SQL, (req.form['phrase'],
                            req.form['letters'],
                            req.remote_addr,
                            req.user_agent.browser,
                            res, ))
```

## ***Chapter 9's Code, 2 of 2***

Esta es la segunda mitad del código de la webapp en "vsearch4web.py".



```
@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    log_request(request, results)
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')


@app.route('/viewlog')
def view_the_log() -> 'html':
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """select phrase, letters, ip, browser_string, results
                  from log"""
        cursor.execute(_SQL)
        contents = cursor.fetchall()
    titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents)

if __name__ == '__main__':
    app.run(debug=True)
```

## **Reforzmos con gestión de contexto:**

### **Python with Context Managers**

#### **Context Managers and the “with” Statement in Python**

#### **Los administradores de contexto y la declaración "With" en Python**

La declaración "with" en Python es considerada como algo oscuro por algunos. Pero cuando echas un vistazo detrás de las escenas del protocolo de Context Manager subyacente verás que hay poco "magia" involucrado.

Entonces, ¿para qué sirve la declaración? Ayuda a simplificar algunos patrones comunes de administración de recursos al abstraer su funcionalidad y permitir que sean factorizados y reutilizados.

A su vez esto le ayuda a escribir un código más expresivo y hace más fácil evitar fugas de recursos en sus programas.

Una buena manera de ver esta característica utilizada eficazmente es mirando ejemplos en la biblioteca estándar de Python. Un ejemplo bien conocido involucra la función open ():

```
with open('hello.txt', 'w') as f:  
    f.write('hello, world!')
```

Generalmente se recomienda abrir archivos con la instrucción with porque asegura que los descriptores de archivos abiertos se cierran automáticamente después de que la ejecución del programa abandone el contexto de la sentencia with. Internamente, el ejemplo de código anterior se traduce en algo como esto:

```
f = open('hello.txt', 'w')  
try:  
    f.write('hello, world')  
finally:  
    f.close()
```

Ya puedes decir que esto es bastante más detallado. Tenga en cuenta que la sentencia try ... finally es significativa. No sería suficiente con escribir algo como esto:

```
f = open('hello.txt', 'w')  
f.write('hello, world')  
f.close()
```

Esta implementación no garantizará que el archivo esté cerrado si hay una excepción durante la llamada `f.write()` y, por lo tanto, nuestro programa podría perder un descriptor de archivo. Es por eso que la declaración con es muy útil. Hace que la adquisición y la liberación de recursos sea una brisa.

Otro buen ejemplo en el que la instrucción `with` se utiliza eficazmente en la biblioteca estándar de Python es la clase `threading.Lock`:

```
some_lock = threading.Lock()

# Harmful:
some_lock.acquire()
try:
    # Do something...
finally:
    some_lock.release()

# Better:
with some_lock:
    # Do something...
```

En ambos casos, el uso de una instrucción `with` permite extraer la mayor parte de la lógica de manejo de recursos. En lugar de tener que escribir un `Try... finally` declaración cada vez, `with` se ocupa de eso para nosotros.

La instrucción `with` puede hacer que el código que trata con los recursos del sistema sea más legible. También ayuda a evitar errores o fugas, haciendo que sea casi imposible olvidarse de limpiar o liberar un recurso después de haber terminado con él.

### ***Apoyo with en sus propios objetos***

Ahora, no hay nada especial o mágico sobre la función `open()` o la clase `threading.Lock` y el hecho de que se pueden usar con una sentencia `with`. Puede proporcionar la misma funcionalidad en sus propias clases y funciones mediante la implementación de los denominados gestores de contexto.

¿Qué es un gestor de contexto? Es un simple "protocolo" (o interfaz) que su objeto necesita seguir para que pueda usarse con la sentencia `with`. Básicamente, todo lo que necesitas hacer es agregar métodos `__enter__` y `__exit__` a un objeto si quieres que funcione como un gestor de contexto. Python llamará a estos dos métodos en los momentos apropiados en el ciclo de administración de recursos.

Echemos un vistazo a lo que esto se vería en términos prácticos. Así es como una simple implementación del gestor de contexto `open()` podría verse así:

```
class ManagedFile:
    def __init__(self, name):
        self.name = name

    def __enter__(self):
        self.file = open(self.name, 'w')
        return self.file

    def __exit__(self, exc_type, exc_val, exc_tb):
        if self.file:
            self.file.close()
```

Nuestra clase de `ManagedFile` sigue el protocolo del gestor de contexto y ahora soporta la sentencia `with`, al igual que el ejemplo original `open()`:

```
>>> with ManagedFile('hello.txt') as f:
...     f.write('hello, world!')
...     f.write('bye now')
```

Python llama a `__enter__` cuando la ejecución entra en el contexto de la sentencia `with` y es hora de adquirir el recurso. Cuando la ejecución deja el contexto de nuevo, Python llama a `__exit__` para liberar el recurso.

Escribir un gestor de contexto basado en clases no es la única forma de soportar la sentencia `with` en Python. El módulo de utilidad `contextlib` en la biblioteca estándar proporciona algunas más abstracciones construidas sobre el protocolo básico del gestor de contexto. Esto puede hacer su vida un poco más fácil si sus casos de uso coincide con lo que ofrece `contextlib`.

Por ejemplo, puede utilizar el decorador `contextlib.contextmanager` para definir una función de fábrica basada en generador para un recurso que admite automáticamente la sentencia `with`. Esto es lo que reescribir nuestro gestor de contexto `ManagedFile` con esta técnica se ve así:

```
from contextlib import contextmanager

@contextmanager
def managed_file(name):
    try:
        f = open(name, 'w')
        yield f
    finally:
        f.close()

>>> with managed_file('hello.txt') as f:
...     f.write('hello, world!')
...     f.write('bye now')
```

En este caso, `managed_file()` es un generador que adquiere primero el recurso. A continuación, suspende temporalmente su propia ejecución y rinde el recurso para que pueda ser utilizado por el llamador. Cuando el llamador abandona el contexto `with`, el generador continúa ejecutándose para que cualquier paso de limpieza que quede puede ocurrir y el recurso se devuelva al sistema.

Tanto las implementaciones basadas en clases como las basadas en generadores son prácticamente equivalentes. Dependiendo de cuál usted encuentre más legible usted puede ser que prefiera uno sobre el otro.

Un inconveniente de la implementación basada en `@contextmanager` podría ser que requiere la comprensión de conceptos avanzados de Python, como decoradores y generadores.

Una vez más, hacer la elección correcta aquí se reduce a lo que usted y su equipo se sienten cómodos utilizando y encontrar la más legible.

## ***Writing Pretty APIs With Context Managers***

## ***Escribir Pretty APIs con gestores de contexto***

Los gestores de contexto son bastante flexibles y si utiliza la sentencia `with` con creatividad, puede definir API convenientes para sus módulos y clases.

Por ejemplo, ¿qué pasa si el "recurso" que queríamos administrar era los niveles de sangría de texto en algún tipo de programa generador de informes? ¿Qué pasaría si pudiéramos escribir código como este para hacerlo:

```
with Indenter() as indent:
    indent.print('hi!')
    with indent:
        indent.print('hello')
        with indent:
            indent.print('bonjour')
            indent.print('hey')
```

Esto casi se lee como un lenguaje específico de dominio (DSL) para sangrar el texto. Además, observe cómo este código entra y sale del mismo gestor de contexto varias veces para cambiar los niveles de sangría. La ejecución de este fragmento de código debe dar lugar a la siguiente salida e imprimir texto con formato ordenado:

```
hi!
  hello
    bonjour
  hey
```

¿Cómo implementaría un gestor de contexto para soportar esta funcionalidad?

Por cierto, esto podría ser un gran ejercicio para envolver su cabeza alrededor de cómo los gestores de contexto de trabajo. Así que antes de revisar mi implementación a continuación, podría tomar algún tiempo y tratar de implementar esto usted mismo como un ejercicio de aprendizaje.

Listo? A continuación, le indicamos cómo implementar esta funcionalidad mediante un gestor de contexto basado en clases:

```
class Indenter:
    def __init__(self):
        self.level = 0

    def __enter__(self):
        self.level += 1
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.level -= 1

    def print(self, text):
        print('    ' * self.level + text)
```

Otro buen ejercicio sería tratar de refactorizar este código para que se base en el generador.

### Cosas para recordar

- La instrucción with simplifica el tratamiento de excepciones al encapsular los usos estándar de las sentencias try / finally en los denominados gestores de contexto.
- Más comúnmente se utiliza para administrar la adquisición segura y la liberación de recursos del sistema. Los recursos son adquiridos por la sentencia with y liberados automáticamente cuando la ejecución deja el contexto.
- El uso eficaz puede ayudarle a evitar fugas de recursos y hacer que su código sea más fácil de leer.

### Context managers

Los administradores de contexto le permiten asignar y liberar recursos con precisión cuando lo deseé. El ejemplo más utilizado de los gestores de contexto es la sentencia with. Supongamos que tiene dos operaciones relacionadas que desea ejecutar como un par, con un bloque de código entre. Los administradores de contexto le permiten hacer específicamente eso. Por ejemplo:

```
with open('some_file', 'w') as opened_file:  
    opened_file.write('Hola!')
```

El código anterior abre el archivo, escribe algunos datos en él y luego lo cierra. Si se produce un error al escribir los datos en el archivo, intenta cerrarlo. El código anterior es equivalente a:

```
file = open('some_file', 'w')  
try:  
    file.write('Hola!')  
finally:  
    file.close()
```

Al compararlo con el primer ejemplo podemos ver que una gran cantidad de código estandarizado se elimina con sólo usar with. La principal ventaja de usar una sentencia with es que asegura que nuestro archivo se cierra sin prestar atención a cómo se cierra el bloque anidado.

Un caso de uso común de los gestores de contexto es bloquear y desbloquear los recursos y cerrar los archivos abiertos (como ya le mostré).

Veamos cómo podemos implementar nuestro propio gestor de contexto. Esto nos permitiría entender exactamente lo que está sucediendo detrás de las escenas.

### **Implementación del gestor de contexto como una clase:**

Como mínimo, un gestor de contexto tiene definidos los métodos `__enter__` y `__exit__`. Hagamos nuestro propio archivo de apertura de Context Manager y aprendamos lo básico.

```
class File(object):
    def __init__(self, file_name, method):
        self.file_obj = open(file_name, method)

    def __enter__(self):
        return self.file_obj

    def __exit__(self, type, value, traceback):
        self.file_obj.close()
```

Simplemente definiendo los métodos `__enter__` y `__exit__` podemos usarlo en una sentencia `with`. Intentemos:

```
with File('demo.txt', 'w') as opened_file:
    opened_file.write('Hola!')
```

Nuestra función `__exit__` acepta tres argumentos. Son requeridos por cada método `__exit__` que es parte de una clase de gestor de contexto. Hablemos de lo que pasa bajo el capó.

1. La instrucción `with` almacena el método `__exit__` de la clase `File`
2. Llama al método `__enter__` de la clase `File`.
3. El método `__enter__` abre el archivo y lo devuelve.
4. El identificador de archivo abierto se pasa a `open_file`.
5. Escribimos en el archivo utilizando `.write()`
6. Con instrucción llama al método `__exit__` almacenado.
7. El método `__exit__` cierra el archivo.

### **Manejo de excepciones**

No hablamos de los argumentos de tipo, valor y rastreo o traceback del método `__exit__`. Entre el paso 4 y 6, si se produce una excepción, Python pasa el tipo, el valor y el rastreo de la excepción al método `__exit__`. Permite que el método `__exit__` decida cómo cerrar el archivo y si se requieren otros pasos. En nuestro caso no estamos prestando atención a ellos.

¿Qué pasa si nuestro objeto de archivo plantea una excepción? Podemos estar tratando de acceder a un método en el objeto de archivo que no es compatible. Por ejemplo:

```
with File('demo.txt', 'w') as opened_file:  
    opened_file.undefined_function('Hola!')
```

Vamos a enumerar los pasos que toman la sentencia `with` cuando se encuentra un error.

1. Transmite el tipo, valor y rastreo o traceback del error al método `__exit__`.
2. Permite que el método `__exit__` maneje la excepción.
3. Si `__exit__` devuelve True entonces la excepción se manejó correctamente.
4. Si algo más que True es devuelto por el método `__exit__` entonces una excepción es planteada por la instrucción `with`.

En nuestro caso el método `__exit__` devuelve None (cuando no se encuentra ninguna sentencia `return`, entonces el método devuelve None). Por lo tanto, la instrucción `with` genera la excepción.

```
Traceback (most recent call last):  
  File "<stdin>", line 2, in <module>  
  
AttributeError: 'file' object has no attribute 'undefined_function'
```

Intentemos manejar la excepción en el método `__exit__`:

```
class File(object):  
    def __init__(self, file_name, method):  
        self.file_obj = open(file_name, method)  
  
    def __enter__(self):  
        return self.file_obj  
  
    def __exit__(self, type, value, traceback):  
        print("Exception has been handled")  
        self.file_obj.close()  
  
        return True
```

```
with File('demo.txt', 'w') as opened_file:  
    opened_file.undefined_function()
```

# Output: Exception has been handled

Nuestro método `__exit__` devolvió `True`, por lo que no se levantó ninguna excepción con la instrucción `with`.

Esta no es la única manera de implementar gestores de contexto. Hay otra manera y la veremos en esta próxima sección.

## ***Implementing a Context Manager as a Generator***

### ***Implementación de un gestor de contexto como generador***

También podemos implementar gestores de contexto utilizando decoradores y generadores. Python tiene un módulo `contextlib` para este propósito. En lugar de una clase, podemos implementar un Administrador de Contexto usando una función de generador. Veamos un ejemplo básico e inútil:

```
from contextlib import contextmanager

@contextmanager
def open_file(name):
    f = open(name, 'w')
    yield f
    f.close()
```

¡Bueno! Esta forma de implementar los gestores de contexto parece ser más intuitiva y fácil. Sin embargo, este método requiere cierto conocimiento sobre generadores, `yield` y decoradores. En este ejemplo no hemos detectado ninguna excepción que pudiera ocurrir. Funciona en la mayor parte de la misma manera que el método anterior.

Vamos a diseccionar este método un poco.

1. Python encuentra la palabra clave `yield`. Debido a esto crea un generador en lugar de una función normal.
2. Debido a la decoración, `contextmanager` se llama con el nombre de la función (`archivo_abierto`) como argumento.
3. La función `contextmanager` devuelve el generador envuelto por el objeto `GeneratorContextManager`.

4. El GeneratorContextManager se asigna a la función open\_file. Por lo tanto, cuando más tarde llamemos a open\_file, estamos llamando al objeto GeneratorContextManager.

Así que ahora que sabemos todo esto, podemos usar el recién generado Context Manager de esta manera:

```
with open_file('some_file') as f:  
    f.write('holo!')
```

## **10** function decorators

# **Wrapping Functions**

## **Decoradores de funciones** **Funciones de envoltura**

**Cuando se trata de aumentar su código, el protocolo de gestión de contexto del capítulo 9 no es el único juego en la ciudad. ?**

Python también le permite utilizar decoradores de funciones, una técnica mediante la cual puede agregar código a una función existente sin tener que cambiar ningún código de la función existente. Si crees que esto suena como una especie de arte negro, no te desesperes: no es nada de eso. Sin embargo, a medida que van las técnicas de codificación, la creación de un decorador de funciones se considera a menudo en el lado más difícil por muchos programadores de Python, y por lo tanto no se utiliza tan a menudo como debería ser. En este capítulo, nuestro plan es mostrarle que, a pesar de ser una técnica avanzada, crear y usar sus propios decoradores no es tan difícil.

### **Su Webapp está trabajando bien, pero ...**

Has mostrado la última versión de tu webapp a un colega, y están impresionados por lo que has hecho. Sin embargo, plantean una pregunta interesante: ¿es aconsejable dejar que cualquier usuario de la web vea la página de registro?

El punto que están haciendo es que cualquiera que esté al tanto de la URL `/viewlog` puede usarlo para ver los datos registrados, tengan o no su permiso. De hecho, por el momento, cada URL de tu webapp es pública, por lo que cualquier usuario de la web puede acceder a cualquiera de ellas.

Dependiendo de lo que intentes hacer con tu webapp, esto puede o no ser un problema. Sin embargo, es común que los sitios web requieran que los usuarios se autenticen antes de que se les ponga a disposición cierto contenido. Es probablemente una buena idea ser prudente cuando se trata de proporcionar acceso a la URL `/viewlog`. La pregunta es: ¿cómo restringir el acceso a ciertas páginas en su webapp?

## ***Only authenticated users gain access***

### ***Solo los usuarios autenticados acceden***

Normalmente, debe proporcionar un ID y una contraseña cuando acceda a un sitio web que ofrezca contenido restringido. Si su combinación de ID / contraseña coincide, se concede el acceso, ya que ha sido autenticado. Una vez que se autentique, el sistema sabe que le permite acceder al contenido restringido. Mantener este estado (autenticado o no) parece que puede ser tan simple como establecer un cambio a True (acceso permitido, se ha iniciado sesión) o False (acceso prohibido, no se ha iniciado sesión).

*Eso suena sencillo para mí. Un simple formulario HTML puede solicitar las credenciales del usuario y, a continuación, un booleano en el servidor puede establecerse en "Verdadero" o "Falso" según sea necesario, ¿no?*

### ***Es un poco más complicado que eso.***

Hay un giro aquí (debido a la forma en que funciona la Web) lo que hace que esta idea un poco más complicado de lo que al principio aparece. Vamos a explorar qué es esta complicación primero (y ver cómo lidiar con ella) antes de resolver nuestro problema de acceso restringido.

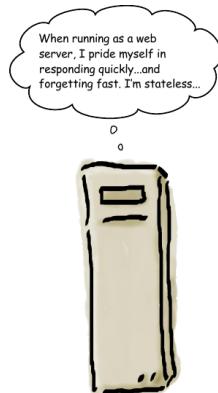
## ***The Web Is Stateless***

### ***La Web es apátrida***

En su forma más básica, un servidor web aparece increíblemente tonto: cada petición que procesa un servidor web es tratada como una petición independiente, no teniendo nada que ver con lo que vino antes, ni lo que viene después.

Esto significa que el envío de tres peticiones rápidas a un servidor web desde su computadora aparece como tres solicitudes individuales independientes. Esto es a pesar de que las tres peticiones se originaron desde el mismo navegador web que se ejecuta en el mismo equipo, que está utilizando la misma dirección IP inalterable (que el servidor web ve como parte de la solicitud).

Como se indica en la parte superior de la página: es como si el servidor web esté siendo tonto. A pesar de que asumimos que las tres peticiones enviadas desde nuestro ordenador están relacionadas, el servidor web no ve las cosas de esta manera: cada solicitud web es independiente de lo que vino antes, así como lo que viene después.



*Cuando se ejecuta como un servidor web, me enorgullezco de responder rápidamente ... y olvidar rápido. Soy apátrida*

### ***HTTP is to blame...***

### ***HTTP es la culpa ...***

La razón por la que los servidores web se comportan de esta manera se debe al protocolo que sostiene la Web y que es utilizado tanto por el servidor web como por su navegador web: HTTP (HyperText Transfer Protocol).

HTTP dicta que los servidores web deben funcionar como se describe anteriormente, y la razón de esto tiene que ver con el rendimiento: si la cantidad de trabajo que un servidor web necesita hacer se minimiza, es posible escalar los servidores web para manejar muchas, muchas solicitudes. Se obtiene un mayor rendimiento a expensas de requerir que el servidor web mantenga información sobre cómo puede estar relacionada una serie de solicitudes. Esta información, conocida como estado en HTTP (y no relacionada con OOP de ninguna manera), no tiene ningún interés para el servidor web, ya que cada solicitud se trata como una entidad independiente. De alguna manera, el servidor web está optimizado para responder rápidamente, pero se olvida rápidamente, y se dice que funciona de manera apátrida. Que es todo bien y bueno hasta el momento en que su webapp necesita recordar algo.

Que es todo bien y bueno hasta el momento en que su webapp necesita recordar algo.

*¿No es eso para lo que las variables son: recordar cosas en el código? Sin duda, esto es una obviedad?*

### ***Si sólo la Web fuera tan simple.***

Cuando su código se ejecuta como parte de un servidor web, su comportamiento puede diferir de cuando lo ejecuta en su equipo. Exploraremos esta cuestión con más detalle.

## **Your Web Server (Not Your Computer) Runs Your Code**

### **Su servidor Web (no su computadora) ejecuta su código**

Cuando Flask ejecuta su webapp en su computadora, mantiene su código en la memoria en todo momento. Con esto en mente, recuerde estas dos líneas de la parte inferior del código de su webapp, que discutimos inicialmente al final del capítulo 5:

```
if __name__ == '__main__':
    app.run(debug=True)
```

← Esta línea de código NO se ejecuta cuando se importa este código.

Esta sentencia if comprueba si el intérprete está ejecutando el código directamente o si el código está siendo importado (por el intérprete o por algo como PythonAnywhere). Cuando Flask se ejecuta en su computadora, el código de su webapp se ejecuta directamente, dando como resultado la ejecución de esta línea app.run. Sin embargo, cuando un servidor web está configurado para ejecutar el código, el código de su aplicación web se importa y la línea app.run no se ejecuta.

¿Por qué? Debido a que el servidor web ejecuta el código de su webapp como le plazca. Esto puede implicar que el servidor web importe el código de su aplicación web, y luego llame a sus funciones según sea necesario, manteniendo el código de su aplicación web en la memoria en todo momento. O el servidor web puede decidir cargar o descargar su código webapp según sea necesario, suponiendo que durante los períodos de inactividad, el servidor web sólo cargará y ejecutará el código que necesita. Es este segundo modo de operación -donde el servidor web carga su código cuando lo necesita- que puede conducir a problemas con el almacenamiento del estado de su aplicación web en variables. Por ejemplo, considere lo que sucedería si agregara esta línea de código a su webapp:

```
logged_in = False
if __name__ == '__main__':
    app.run(debug=True)
```

← La variable "logged\_in" se podría utilizar para indicar si un usuario de su aplicación web ha iniciado sesión o no.

La idea aquí es que otras partes de su aplicación web pueden hacer referencia a la variable logged\_in para determinar si un usuario está autenticado. Además, su código puede cambiar el valor de esta variable según sea necesario (basado, digamos, en un inicio de sesión exitoso). Como la variable logged\_in es de naturaleza global, todo el código de su webapp puede acceder y establecer su valor. Esto parece un enfoque razonable, pero tiene dos problemas.

En primer lugar, su servidor web puede descargar el código de ejecución de su webapp en cualquier momento (y sin previo aviso), por lo que cualquier valor asociado con las variables globales es probable que se pierda, y se va a restablecer a su valor inicial cuando

el código se importa. Si una función previamente cargada establece `logged_in` en `True`, el código reimportado restablece satisfactoriamente `login_in` a `False`, y reina la confusión ...

En segundo lugar, tal y como está, sólo hay una copia de la variable global `logged_in` en tu código de ejecución, lo cual está bien si todo lo que planeas tener es un solo usuario de tu webapp (buena suerte con eso). Si tiene dos o más usuarios que acceden y / o cambien el valor de `logged_in`, no sólo la confusión reinará, sino que la frustración hará que aparezca también una invitada. Como regla general, almacenar el estado de su webapp en una variable global es una mala idea.

***No guarde el estado de su aplicación web en variables globales.***



### ***It's Time for a Bit of a Session***

### ***Es hora de un poco de una sesión***

Como resultado de lo que aprendimos en la última página, necesitamos dos cosas:

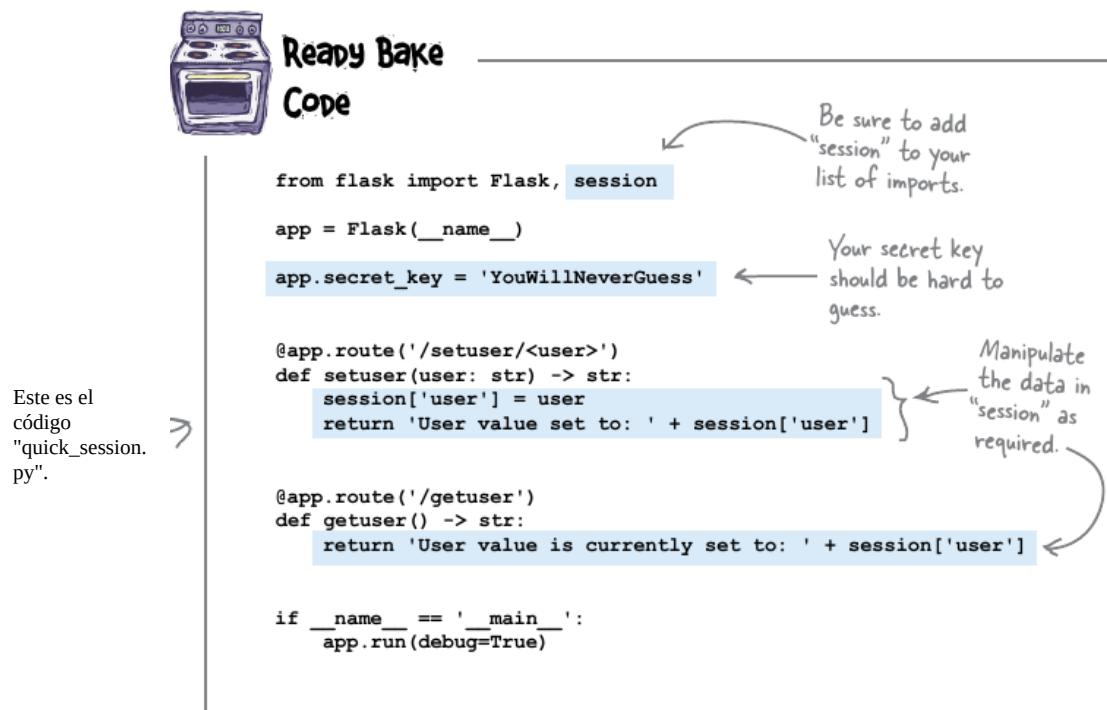
- Una forma de almacenar variables sin recurrir al uso de globales
- Una forma de evitar que los datos de un usuario de la aplicación web interfieran con los de otro usuario

La mayoría de los marcos de desarrollo webapp (incluyendo Flask) proporcionan ambos requisitos utilizando una única tecnología: la sesión.

Piense en una sesión como una capa de estado extendida en la parte superior de la Web sin estado.

Agregando un pequeño fragmento de datos de identificación a su navegador (una cookie), y vinculando esto a una pequeña pieza de datos de identificación en el servidor web (el ID de sesión), Flask utiliza su tecnología de sesión para mantener todo en orden. No sólo puede almacenar estado en su aplicación web que persiste con el tiempo, pero cada usuario de su aplicación web obtiene su propia copia del estado. Confusión y frustración no son más.

Para demostrar cómo funciona el mecanismo de sesión de Flask, echemos un vistazo a una aplicación web muy pequeña que se guarda en un archivo denominado `quick_session.py`. Tómese un momento para leer el código primero, prestando especial atención a las partes resaltadas. Discutiremos lo que está pasando después de haber tenido la oportunidad de leer este código:



Traducimos:

1. Asegúrese de agregar "sesión" a su lista de importaciones.
2. Su clave secreta debe ser difícil de adivinar.
3. Manipule los datos en "sesión" según sea necesario.

## *Flask's Session Technology Adds State*

### *La tecnología de la sesión de Flask agrega el estado*

Para utilizar la tecnología de sesión de Flask, primero debes importar la sesión del módulo del `flask`, que la aplicación web `quick_session.py` que viste hace en su primera línea. Piense en la `sesión` como un diccionario global de Python dentro del cual almacena el estado de su aplicación web (aunque sea un diccionario con algunas superpotencias añadidas):

```

from flask import Flask, session
...

```

Start by importing "session".

Aunque su aplicación web sigue funcionando en la Web sin estado, esta importación única le da a su aplicación web la capacidad de recordar el estado.

Flask garantiza que todos los datos almacenados en la sesión existan durante todo el tiempo que se ejecuta la aplicación web (no importa cuántas veces el servidor web carga y vuelve a cargar el código de la webapp). Además, todos los datos almacenados en la sesión están

marcados por una cookie de navegador única, que garantiza que los datos de sesión se mantengan alejados de los de cualquier otro usuario de su aplicación web.

Cómo es que todo esto no es importante: el hecho de que lo hace es. Para habilitar toda esta bondad extra, necesitas sembrar la tecnología de generación de cookie de Flask con una "clave secreta", que es utilizada por Flask para cifrar tu cookie, protegiéndola de cualquier mirada indiscreta. Así es como `quick_session.py` hace esto:

Cree una nueva webapp Flask de la manera habitual.

```
→ app = Flask(__name__)
    ...
    app.secret_key = 'YouWillNeverGuess'
    ...
```

Seed Flask tecnología de generación de cookie con una clave secreta  
(Nota: cualquier cadena se hará aquí. Pero, al igual que cualquier otra contraseña que utilice, debe ser difícil de adivinar.)

La documentación del flask sugiere elegir una clave secreta que es difícil de adivinar, pero cualquier valor de cadena funciona aquí. Flask utiliza la cadena para cifrar la cookie antes de transmitirla a su navegador.

Una vez que se importe la sesión y se establezca la clave secreta, puede utilizar la sesión en su código como lo haría con cualquier otro diccionario de Python. Dentro de `quick_session.py`, la URL `/setuser` (y su función `setuser` asociada) asigna un valor proporcionado por el usuario a la clave de usuario en la sesión y luego devuelve el valor a su navegador:

El valor de la variable "user" se asigna a la clave "user" en el diccionario "sesión".

```
...  
@app.route('/setuser/<user>')
def setuser(user: str) -> str:
    session['user'] = user
    return 'User value set to: ' + session['user']
...
```

La URL espera recibir un valor para asignar a la variable "user" (verá cómo funciona esto en un poco).

Ahora que hemos establecido algunos datos de sesión, veamos el código que accede a él.

## *Dictionar y Lookup Retrieves State*

### *Búsqueda de diccionario recupera estado*

Ahora que un valor está asociado con la clave de usuario en la sesión, no es difícil acceder a los datos asociados con el usuario cuando lo necesite.

La segunda URL en la aplicación web `quick_session.py/getuser`, está asociada con la función `getuser`. Cuando se invoca, esta función accede al valor asociado con la clave de usuario y la devuelve al navegador web en espera como parte del mensaje con cadenas. La

función `getuser` se muestra a continuación, junto con el nombre de este webapp dunder es igual a la prueba principal dunder (discutido por primera vez cerca del final del capítulo 5):

```
...
@app.route('/getuser')
def getuser() -> str:
    return 'User value is currently set to: ' + session['user']
```

```
{ if __name__ == '__main__':
    app.run(debug=True)}
```

Como es la costumbre con todas las aplicaciones de Flask, controlamos cuando "app.run" se ejecuta utilizando este idioma bien establecido de Python.

No es difícil acceder a los datos en "session". Es una búsqueda de diccionario.

## ***Time for a Test Drive?***

### ***¿Tiempo para un Test Drive?***

Es casi la hora de tomar el `quick_session.py` webapp para una vuelta. Sin embargo, antes de hacerlo, pensemos un poco en lo que queremos probar.

Para empezar, queremos comprobar que la aplicación web almacena y recupera los datos de sesión que se le proporcionan. Además, queremos asegurar que más de un usuario pueda interactuar con la aplicación web sin pisar los dedos de los demás usuarios: los datos de sesión de un usuario no deben afectar los datos de ningún otro.

Para realizar estas pruebas, vamos a simular varios usuarios ejecutando varios navegadores. Aunque todos los navegadores se ejecutan en una computadora, en lo que respecta al servidor web, todos son conexiones independientes e individuales: después de todo, la Web es apátrida. Si repetíramos estas pruebas en tres equipos físicamente diferentes en tres redes diferentes, los resultados serían los mismos, ya que todos los servidores web ven cada solicitud de forma aislada, independientemente de dónde se origine la solicitud. Recuerde que la tecnología de `sesión` en Flask capa una tecnología con estado en la parte superior de la Web sin estado.

Para iniciar esta aplicación web, utilice este comando dentro de un terminal en Linux o Mac OS X:

```
$ python3 quick_session.py
```

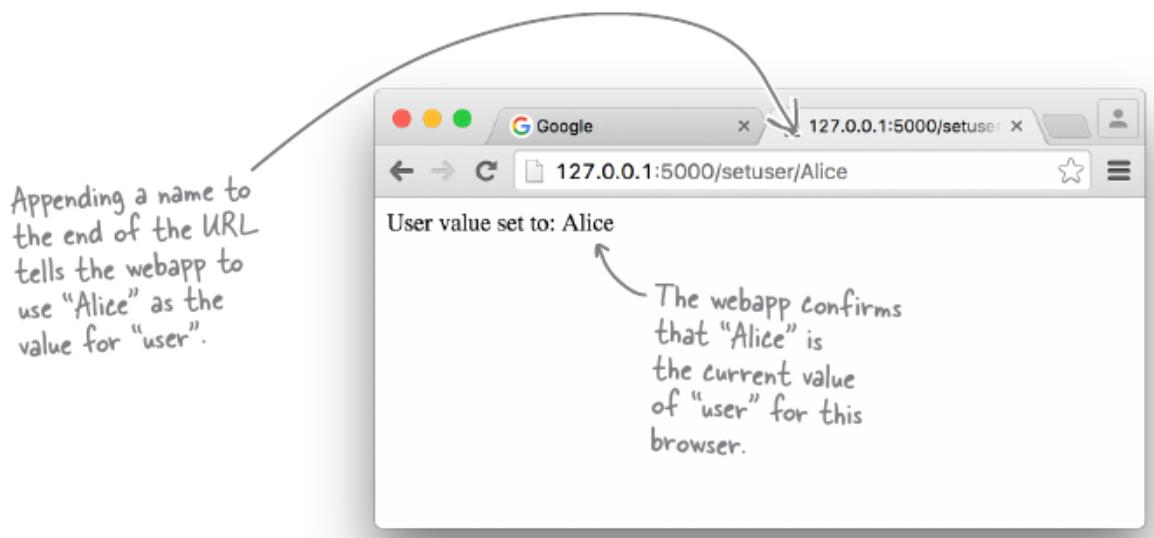
O utilice este comando en un símbolo del sistema en Windows:

```
C:\> py -3 quick_session.py
```

## **Test Drive, 1 of 2**

Con la aplicación web `quick_session.py` en funcionamiento, abramos un navegador Chrome y utilícelo para establecer un valor para la clave de `usuario` en la `sesión`. Hacemos esto

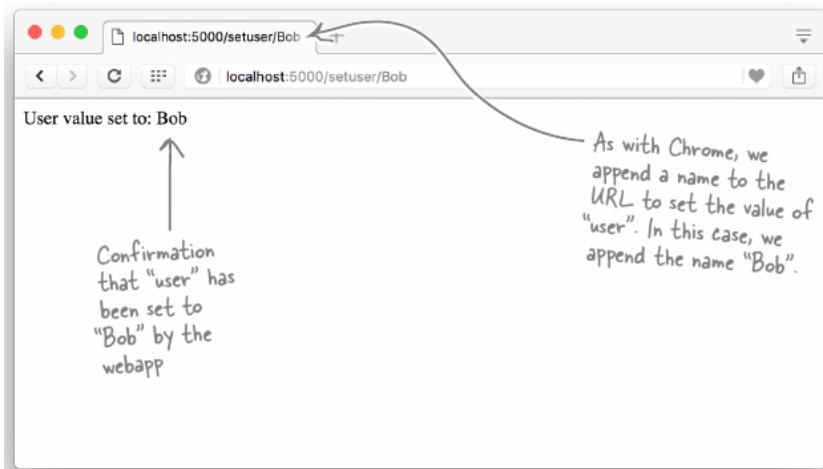
escribiendo `/setuser/Alice` en la barra de ubicación, que indica a la aplicación web que utilice el valor `Alice` para el usuario:



Traducimos:

1. Añadir un nombre al final de la URL le dice a la aplicación web que utilice "Alice" como valor para "user".
2. La webapp confirma que "Alice" es el valor actual de "usuario" para este navegador.

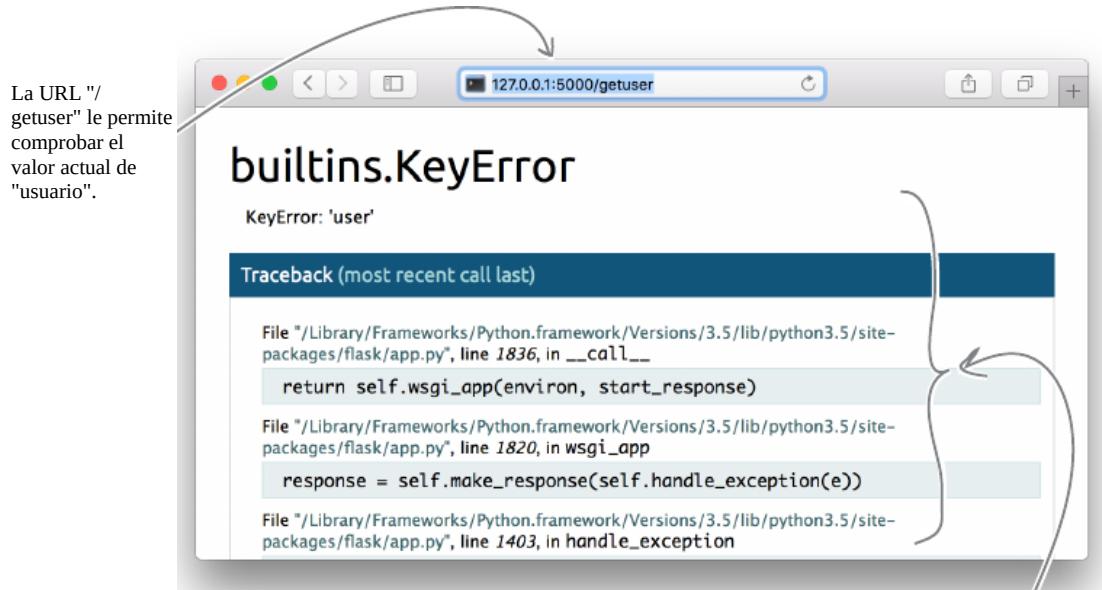
A continuación, vamos a abrir el navegador Opera y usarlo para establecer el valor de usuario a Bob (si no tienes acceso a Opera, usa el navegador que sea útil, siempre y cuando no sea Chrome):



Traduciendo:

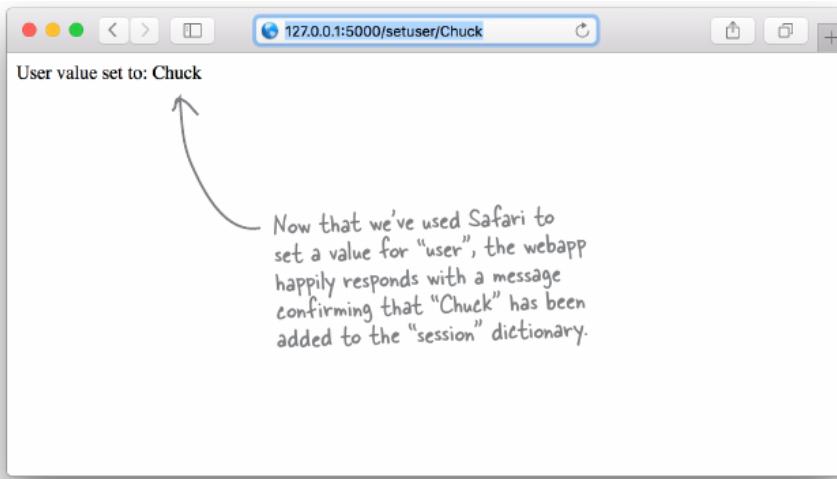
1. Confirmación de que "usuario" ha sido establecido en "Bob" por la aplicación web
2. Al igual que con Chrome, añadimos un nombre a la URL para establecer el valor "usuario". En este caso, añadiremos el nombre "Bob".

Cuando abrimos Safari (o podemos usar Edge si estamos en Windows), usamos la otra URL de webapp, `/getuser`, para recuperar el valor actual del `usuario` de la aplicación web. Sin embargo, cuando lo hicimos, nos recibieron con un mensaje de error bastante intimidante:



¡Vaya! Ese es el mensaje de error, ¿no? El bit importante está en la parte superior: tenemos un "KeyError", ya que no hemos utilizado Safari para establecer un valor para "usuario" todavía. (Recuerda: establecimos un valor de "usuario" con Chrome y Opera, no con Safari).

Vamos a utilizar Safari para establecer el valor del usuario a Chuck:



Ahora que hemos utilizado Safari para establecer un valor para "usuario", la webapp felizmente responde con un mensaje confirmando que "Chuck" se ha agregado al diccionario "session".



## Test Drive, 2 of 2

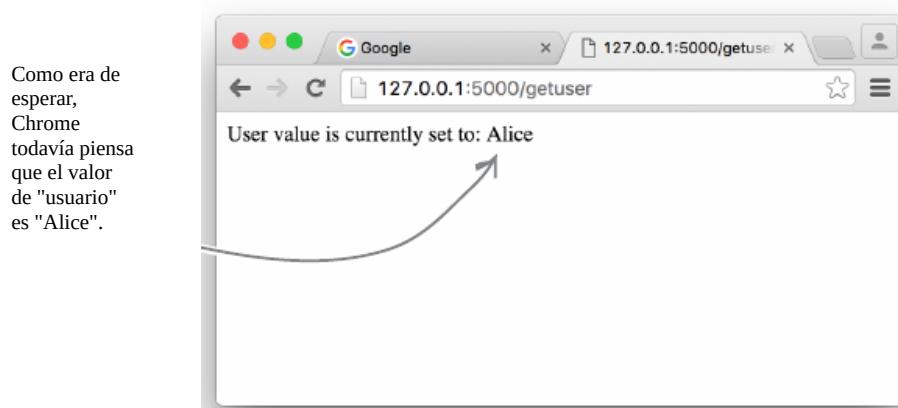
Ahora que hemos utilizado los tres navegadores para establecer los valores para el usuario, vamos a confirmar que la aplicación web (gracias a nuestro uso de la sesión) está deteniendo el valor de cada navegador de usuario de interferir con los datos de cualquier otro navegador. Aunque hemos utilizado Safari para establecer el valor del usuario para Chuck, veamos cuál es su valor en Opera usando la URL `/getuser`:



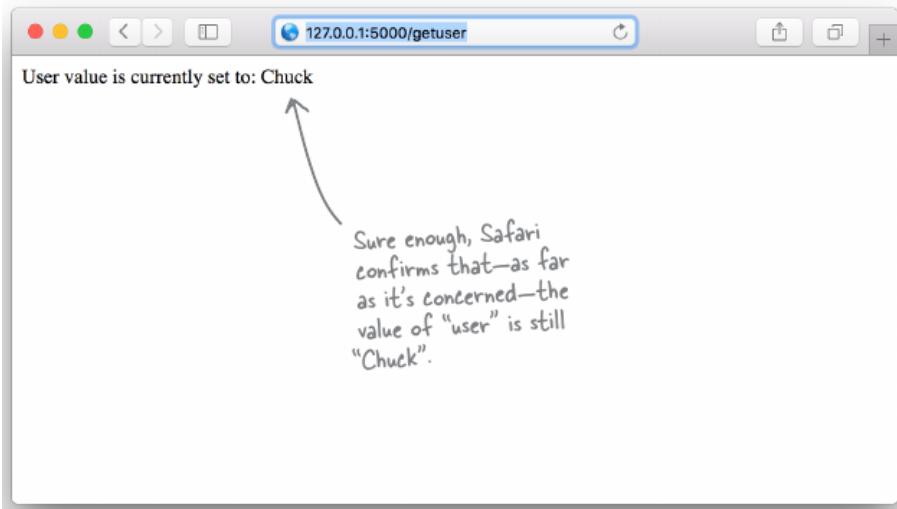
Traducimos:

A pesar de que Safari acaba de establecer "usuario" a "Chuck", el navegador de la ópera confirma que todavía piensa que el valor de "usuario" es "Bob".

Una vez confirmado que Opera muestra el valor del usuario como Bob, volvamos a la ventana del navegador Chrome y emitamos la URL de `/getuser` allí. Como era de esperar, Chrome confirma que, en lo que a ella se refiere, el valor del usuario es Alice:



Acabamos de utilizar Opera y Chrome para acceder al valor del usuario mediante la URL /getuser, que acaba de salir de Safari. Esto es lo que vemos cuando publicamos /getuser en Safari, que no produce un mensaje de error esta vez, ya que el usuario tiene un valor asociado ahora (así, no más KeyError):



Traducido:

Seguramente, Safari confirma que -por lo que a ella se refiere- el valor de "usuario" sigue siendo "Chuck".

*Así que ... cada navegador mantiene su propia copia del valor "usuario", ¿verdad?*

**No, no completamente, todo ocurre en la aplicación web.**

El uso del diccionario de sesión en la aplicación web permite el comportamiento que está viendo aquí. Al configurar automáticamente una cookie única dentro de cada navegador, la aplicación web (gracias a la sesión) mantiene un valor de usuario identificable por navegador para cada navegador.

Desde la perspectiva de la aplicación web, es como si hubiera varios valores de usuario en el diccionario de sesión (codificado por cookie). Desde la perspectiva de cada navegador, es como si sólo hay un solo valor de usuario (el asociado con su cookie individual y única).

## ***Managing Logins with Sessions***

### ***Gestión de inicio de sesión con sesiones***

Basándonos en nuestro trabajo con `quick_session.py`, sabemos que podemos almacenar el estado específico del navegador en la sesión. No importa cuántos navegadores interactúen con nuestra aplicación web, los datos del servidor de cada navegador (estado a.k.a.) son administrados por Flask cada vez que se utiliza la `sesión`.

Utilizaremos este nuevo know-how para volver al problema de controlar el acceso web a páginas específicas dentro de la aplicación web `vsearch4web.py`. Recuerde que queremos llegar al punto en el que podemos restringir quién tiene acceso a la URL `/viewlog`.

En lugar de experimentar en nuestro código de trabajo `vsearch4web.py`, vamos a poner ese código a un lado por ahora y trabajar con algún otro código, con lo que vamos a experimentar con el fin de resolver lo que tenemos que hacer. Volveremos al código `vsearch4web.py` una vez que hayamos calculado la mejor manera de abordar esto. Podemos entonces enmendar con confianza el código `vsearch4web.py` para restringir el acceso a `/viewlog`.

Aquí está el código para otra aplicación web basada en Flask. Como antes, tome un tiempo para leer este código antes de nuestra discusión de él. Esto es `simple_webapp.py`:



## Ready Bake Code

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello from the simple webapp.'

@app.route('/page1')
def page1() -> str:
    return 'This is page 1.'

@app.route('/page2')
def page2() -> str:
    return 'This is page 2.'

@app.route('/page3')
def page3() -> str:
    return 'This is page 3.'

if __name__ == '__main__':
    app.run(debug=True)
```

Esto es "simple\_webapp.py". En esta etapa de este libro, no debería tener dificultad para leer este código y entender lo que hace esta aplicación web.



## Let's Do Login

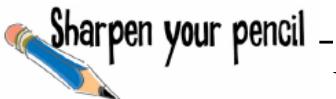
### Iniciar sesión

El código `simple_webapp.py` es sencillo: todas las URL son públicas y pueden ser accedidas por cualquiera que use un navegador.

Además del valor predeterminado / URL (que da como resultado la ejecución de la función hello), hay otras tres URL, / page1, / page2 y / page3 (que invocan funciones denominadas de forma similar cuando se accede). Todas las URL de la webapp devuelven un mensaje específico al navegador.

Como webapps ir, éste es realmente sólo una cáscara, pero hará para nuestros propósitos. Nos gustaría llegar al punto en el que / page1, / page2 y / page3 sólo son visibles para usuarios registrados, pero restringidos a todos los demás. Vamos a usar la tecnología de sesión de Flask para habilitar esta funcionalidad.

Comencemos proporcionando una URL realmente simple /login. Por ahora, no nos vamos a preocupar por proporcionar un formulario HTML que solicite un ID de inicio de sesión y una contraseña. Todo lo que vamos a hacer aquí es crear algún código que ajuste la sesión para indicar que se ha producido un inicio de sesión satisfactorio.



Vamos a escribir el código para la URL /login de inicio de sesión a continuación. En el espacio mostrado, proporcione un código que ajuste la sesión estableciendo un valor para la clave logged\_in en True. Además, haga que la función de la URL devuelva el mensaje "Ahora está conectado" al explorador en espera:

Add the new code here.

```
@app.route('/login')
def do_login() -> str:
    .....
    return .....
```



Set the "logged\_in" key in the "session" dictionary to "True".

```
@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    .....
    return 'You are now logged in.'
```

Return this message to the waiting browser.

traducimos:

1. Establezca la clave "logged\_in" en el diccionario "session" en "True".
2. Devuelva este mensaje al navegador en espera. 'you are now logged in'

Además de crear el código para la URL `/login`, necesitaba realizar otros dos cambios en el código para habilitar las sesiones. Usted fue a detalle lo que piensa que estos cambios fueron:

- 1 We need to add 'session' to the import line at the top of the code.
- 2 We need to set a value for this webapp's secret key.
- } ← Let's not forget to do these.

traducimos:

1. Necesitamos agregar "sesión" a la línea de importación en la parte superior del código.
2. Necesitamos establecer un valor para la clave secreta de esta aplicación web.
3. No olvidemos hacer esto.

## ***Amend the webapp's code to handle logins***

## ***Modificar el código de la webapp para manejar los inicio de sesión***

Vamos a detener la prueba de este nuevo código hasta que hemos añadido otras dos URL:

`/logout` y `/status`. Antes de continuar, asegúrese de que la copia de `simple_webapp.py` ha sido modificada para incluir los cambios que se muestran a continuación. Nota: no mostramos todo el código de la aplicación web aquí, solo los nuevos bits (que están resaltados):

```
from flask import Flask, session
app = Flask(__name__)
...
@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    return 'You are now logged in.'

app.secret_key = 'YouWillNeverGuessMySecretKey'
if __name__ == '__main__':
    app.run(debug=True)
```

Agregue el código para la URL "/login". →

Remember to import "session".

Establecer un valor para la clave secreta de esta aplicación web (que permite el uso de sesiones).

## **Let's Do Logout and Status Checking**

### **Registro y comprobación de estado**

La adición del código para las URL `/logout` y `/status` es nuestra próxima tarea.

Cuando se trata de cerrar la sesión, una estrategia es establecer la clave `logged_in` del diccionario de sesión en `False`. Otra estrategia es eliminar completamente la clave `logged_in` de la sesión. Vamos a ir con la segunda opción; La razón por la que se hará claro después de que el código URL de `/status`.



Escribamos el código para la URL de `/logout`, que necesita quitar la clave `logged_in` del diccionario de sesión, y luego devolver el mensaje "Ahora estás desconectado" en el navegador en espera. Agregue su código en los espacios abajo:

Add the  
logout  
code here.

```
@app.route('/logout')  
def do_logout() -> str:  
    .....  
    return .....
```

Añada el código de cierre de sesión  
aquí.

Sugerencia: si ha olvidado  
cómo eliminar una clave de un  
diccionario, escriba "dir(dict)"  
en el indicador >>> para  
obtener una lista de los métodos  
de diccionario disponibles.

Con `/logout` escrito, ahora volvemos nuestra atención a `/status`, que devuelve uno de los dos mensajes al navegador web en espera.

El mensaje "Has iniciado sesión actualmente" o "You are currently logged in" se devuelve cuando está registrado o `logged_in` como un valor en el diccionario de sesión (y, por definición, se establece en `True`).

El mensaje "You are NOT logged in" se devuelve cuando el diccionario de sesión no tiene una clave `logged_in`. Tenga en cuenta que no podemos comprobar `login_in` para `False`, ya que la URL `/logout` elimina la clave del diccionario de sesión en lugar de cambiar su valor. (No hemos olvidado que todavía tenemos que explicar por qué estamos haciendo las cosas de esta manera, y vamos a llegar a la explicación en un tiempo. Por ahora, confía en que esta es la forma en que tiene que codificar esta funcionalidad).

Vamos a escribir el código de la URL /status en el espacio de abajo:

```
@app.route('/status')
def check_status() -> str:
    if .....  
        return .....  
    return .....
```

Put your status-checking code here.

Check if the "logged\_in" key exists in the "session" dictionary, then return the appropriate message.

Traductor:

1. Ponga su código de verificación de estado aquí.
2. Compruebe si existe la clave "logged\_in" en el diccionario "session" y luego devuelva el mensaje correspondiente.



Debía escribir el código para la URL de /logout, que necesitaba quitar la clave `logged_in` del diccionario de sesión, y luego devolver el mensaje "Ahora está desconectado" en el buscador en espera:

```
@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')  
    .....  
    return 'You are now logged out.'
```

Use the "pop" method to remove the "logged\_in" key from the "session" dictionary.

Traducimos:

Utilice el método "pop" para quitar la clave "logged\_in" del diccionario "session".

Con /logout escrito, debías dirigir tu atención a la URL /status, que devuelve uno de los dos mensajes al navegador web en espera.

El mensaje "Has iniciado sesión actualmente" o "You are currently logged in" se devuelve cuando está registrado o `logged_in` como un valor en el diccionario de sesión (y, por definición, se establece en True).

El mensaje "You are NOT logged in" se devuelve cuando el diccionario de sesión no tiene una clave `logged_in`.

Debes escribir el código para /status en el espacio de abajo:

You were to write the code for /status in the space below:

```
@app.route('/status')
def check_status() -> str:
    if 'logged_in' in session:
        return 'You are currently logged in.'
    return 'You are NOT logged in.'
```

Does the "logged\_in" key exist in the "session" dictionary?

If yes, return this message.

If no, return this message.

Traducimos:

1. Debes escribir el código para / status en el espacio de abajo:
2. ¿Existe la clave "logged\_in" en el diccionario "session"?
3. En caso afirmativo, devuelva este mensaje.
4. Si no, devuelva este mensaje.

### ***Amend the webapp's code once more***

### ***Modificar el código de la webapp una vez más***

Todavía nos quedamos a prueba en la prueba de esta nueva versión de la aplicación web, pero aquí (a la derecha) es una versión resaltada del código que necesita agregar a su copia de `simple_webapp.py`.

Asegúrese de que ha modificado su código para que coincida con el nuestro antes de llegar a la próxima Prueba de unidad, que se viene a la derecha después de hacer bien en una promesa anterior.

```
...
@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')
    return 'You are now logged out.'

@app.route('/status')
def check_status() -> str:
    if 'logged_in' in session:
        return 'You are currently logged in.'
    return 'You are NOT logged in.'

app.secret_key = 'YouWillNeverGuessMySecretKey'

if __name__ == '__main__':
    app.run(debug=True)
```

Two new URL routes

## Why Not Check for False?

### ¿Por qué no comprobar si hay falso?

Cuando codificó URL `/loggin` de inicio de sesión, estableció la clave `logged_in` en True en el diccionario de sesión (lo que indicaba que el explorador estaba registrado en la aplicación web). Sin embargo, cuando codificó la URL `/logout`, el código no estableció el valor asociado con la clave `logged_in` a False, ya que preferimos eliminar todo rastro de la clave `logged_in` del diccionario de sesión. En el código que manejaba la URL `/status`, comprobamos el "estado de inicio de sesión" determinando si existía o no la clave `logged_in` en el diccionario de sesión; No verificamos si logueado es falso (o cierto, para esa materia). Lo que plantea la pregunta: ¿por qué la aplicación web no utiliza Falso para indicar "no conectado"?

La respuesta es sutil, pero importante, y tiene que ver con la forma en que los diccionarios funcionan en Python. Para ilustrar el problema, vamos a experimentar en el prompt `>>>` y simular lo que puede suceder con el diccionario de sesión cuando se usa con la aplicación web. Asegúrese de seguir esta sesión y lea cuidadosamente cada una de las anotaciones:

The screenshot shows a Python 3.5.1 Shell window with the following interaction:

```
>>> session = dict()
>>> if session['logged_in']:
...     print('Found it.')
...
Traceback (most recent call last):
  File "<pyshell#47>", line 1, in <module>
    if session['logged_in']:
KeyError: 'logged_in'
>>> if 'logged_in' in session:
...     print('Found it.')
...
>>> session['logged_in'] = True
>>> if 'logged_in' in session:
...     print('Found it.')
...
Found it.
>>> if session['logged_in']:
...     print('Found it.')
...
Found it.
>>> |
```

Annotations explain the behavior:

- Create a new, empty dictionary called "session".
- Try to check for the existence of a "logged\_in" value using an "if" statement.
- Whoops! The "logged\_in" key doesn't exist yet, so we get a "KeyError", and our code has crashed as a result.
- However, if we check for existence using "in", our code doesn't crash (there's no "KeyError") even though the key has no value.
- Let's assign a value to the "logged\_in" key.
- Checking for existence with "in" still works, although this time around we get a positive result (as the key exists and has a value).
- Checking with an "if" statement works too (now that the key has a value associated with it). However, if the key is removed from the dictionary (using the "pop" method) this code is once again vulnerable to "KeyError".

<traducimos:

1. Cree un nuevo diccionario vacío denominado "session".
2. Intente comprobar la existencia de un valor "logged\_in" usando una instrucción "if".
3. ¡Vaya! La clave "logged\_in" no existe todavía, por lo que recibimos un "KeyError", y nuestro código se ha estrellado como resultado.
4. Sin embargo, si comprobamos la existencia usando "in", nuestro código no se bloquea (no hay "KeyError") aunque la clave no tiene valor.
5. Asigne un valor a la clave "logged\_in".
6. Comprobar la existencia con "in" sigue funcionando, aunque esta vez obtenemos un resultado positivo (ya que la clave existe y tiene un valor).
7. Comprobar con una instrucción "if" funciona también (ahora que la clave tiene un valor asociado con ella). Sin embargo, si la clave se elimina del diccionario (utilizando el método "pop") este código es de nuevo vulnerable a "KeyError".

La experimentación anterior muestra que no es posible comprobar un diccionario para el valor de una clave hasta que existe un apareamiento de clave / valor. Intentar hacer tan resultados en un KeyError. Como es una buena idea evitar errores como este, el código `simple_webapp.py` comprueba la existencia de la clave `logged_in` como prueba de que el explorador ha iniciado sesión, en lugar de comprobar el valor real de la clave, evitando así la posibilidad de un KeyError.

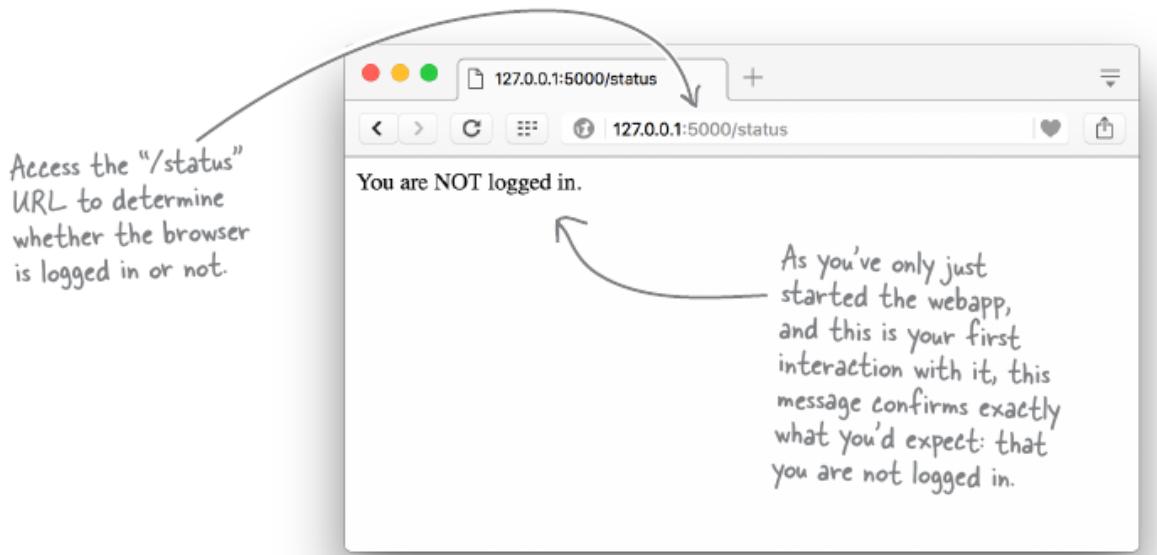


Echemos un vistazo a la aplicación web `simple_webapp.py` para ver qué tan bien realizan las URLs de `/login`, `/logout` y `/status`. Al igual que con el último Test Drive, vamos a probar esta aplicación web utilizando más de un navegador para confirmar que cada navegador mantiene su propio "estado de inicio de sesión" o "login state" en el servidor. Comencemos la aplicación web desde el terminal de nuestro sistema operativo:

On Linux and Mac OS X: `python3 simple_webapp.py`

On Windows: `py -3 simple_webapp.py`

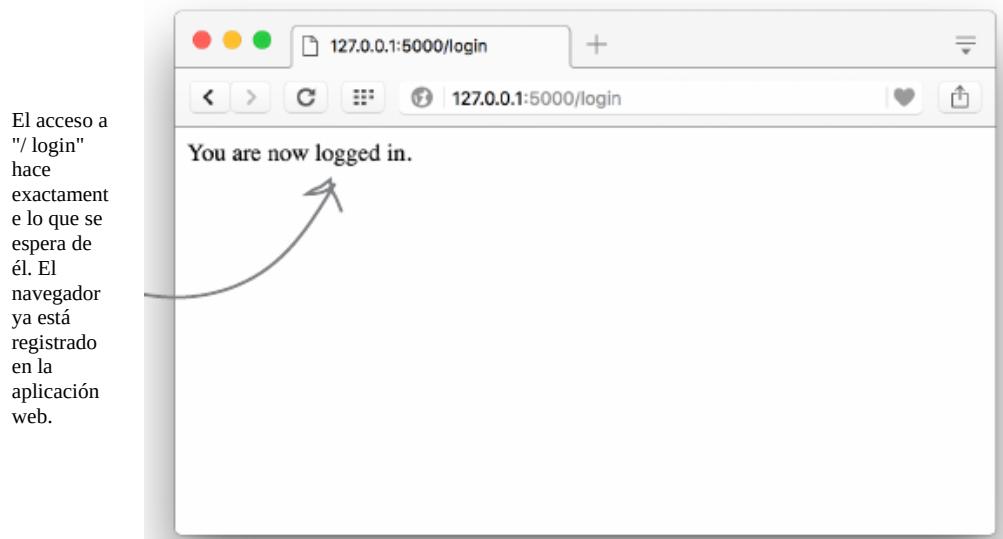
Vamos a encender Opera y comprobar su estado inicial de inicio de sesión mediante el acceso a URL de `/status`. Como se esperaba, el navegador no ha iniciado sesión:



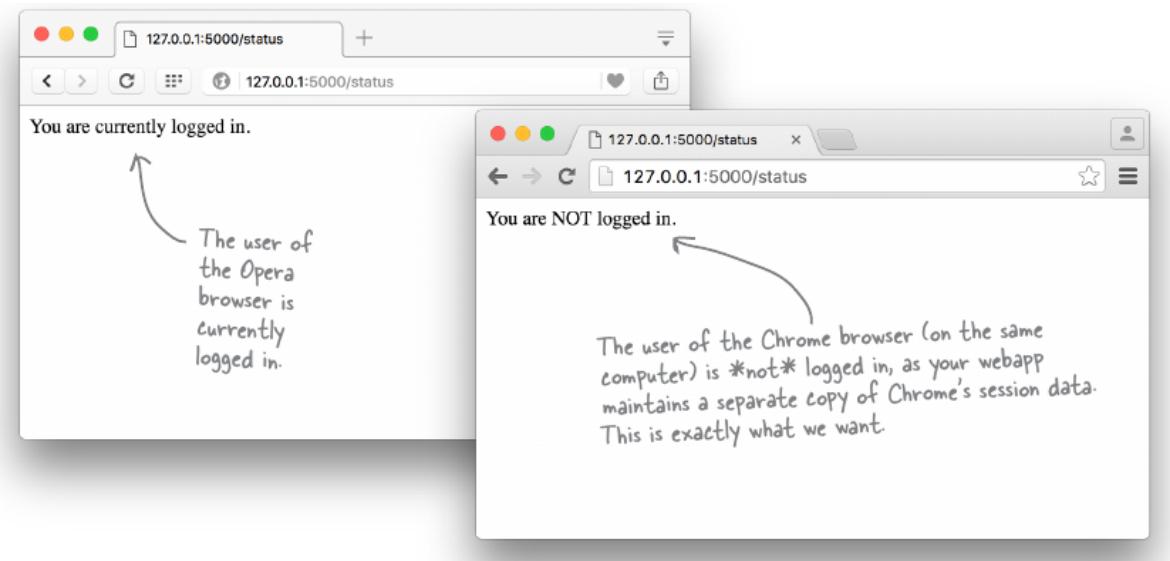
Traducimos:

1. Acceda a la URL "/ status" para determinar si el explorador está conectado o no.
2. Como acabas de iniciar la aplicación web, y esta es tu primera interacción con ella, este mensaje confirma exactamente lo que esperabas: que no has iniciado sesión.

Vamos a simular el inicio de sesión, accediendo a la URL `/login`. El mensaje cambia para confirmar que el inicio de sesión se ha realizado correctamente:



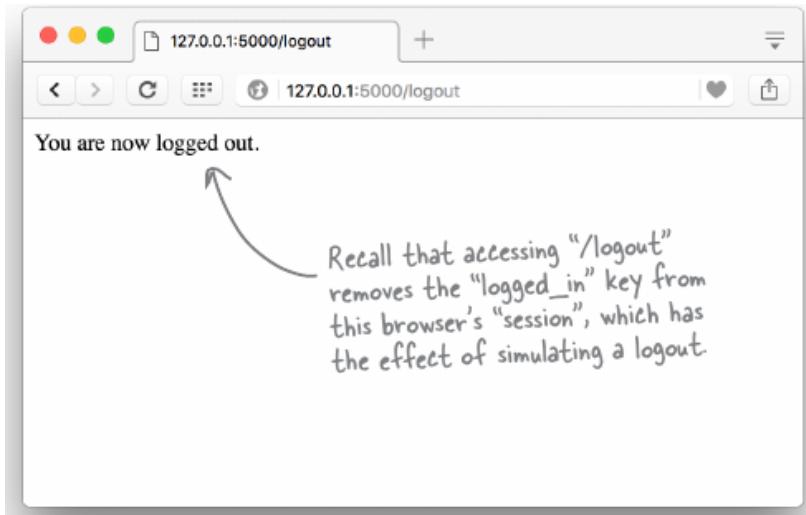
Ahora que ha iniciado sesión, vamos a confirmar el cambio de estado al acceder a la URL `/status` dentro de Opera. De este modo, se confirma que el usuario del navegador Opera está conectado. Si utiliza Chrome para comprobar el estado, también verá que el usuario de Chrome no está conectado, lo que es exactamente lo que queremos (ya que cada usuario de la webapp-cada navegador-tiene su propio estado mantenido por la aplicación web):



Traducmos:

1. El usuario del navegador Opera está conectado.
2. El usuario del navegador Chrome (en el mismo equipo de su webapp) no se ha conectado \*, ya que su aplicación web mantiene una copia separada de los datos de sesión de Chrome. Esto es exactamente lo que queremos.

Para concluir, vamos a acceder a la URL de `/logout` dentro de Opera para decirle a la aplicación web que estamos cerrando sesión logging out de la sesión:



traduciendo:

Recuerde que el acceso a `/ logout` elimina la clave `"logged_in"` de este navegador `"session"`, que tiene el efecto de simular un logout.

Aunque no hemos pedido a ninguno de los usuarios de nuestro navegador una ID de inicio de sesión o una contraseña, las URLs `/login`, `/logout` y `/status` nos permiten simular lo que sucedería con el diccionario de sesión de la webapp si creamos el formulario HTML requerido. Y luego conectar los datos del formulario a una base de datos de "credenciales". Los detalles de cómo esto puede suceder son muy específicos de la aplicación, pero el mecanismo básico (es decir, manipular la `session`) es el mismo, independientemente de lo que una aplicación web específica pueda desear.

¿Estamos listos ahora para restringir el acceso a las URLs `/page1`, `/page2` y `/page3`?

### **Listo para restringir**

### ***¿Ahora podemos restringir el acceso a las URL?***

*Echa un vistazo a este código de inicio de sesión, chicos. Creo que es bastante claro lo que tengo que hacer ...*

**Jim:** Oye, Frank ... ¿qué estás atrapado?

**Frank:** Necesito encontrar una manera de restringir el acceso a las URLs `/page1`, `/page2` y `/page3` ...

**Joe:** No puede ser tan difícil, ¿verdad? Ya tienes el código que necesitas en la función que maneja `/status` ...

**Frank:** ... y sabe si el navegador de un usuario está conectado o no, ¿verdad?

**Joe:** Sí, lo hace. Por lo tanto, todo lo que tiene que hacer es copiar y pegar el código de comprobación de la función que maneja `/status` en cada una de las URL que desea restringir, y luego estás en home y seco!

**Jim:** ¡Oh, hombre! Copia y pega ... el talón de Aquiles del desarrollador web. Realmente no quieres copiar y pegar código así ... solo puede conducir a problemas en el camino.

**Frank:** ¡Por supuesto! CS 101 ... Crearé una función con el código de `/status`, luego llamaré esa función según sea necesario dentro de las funciones que manejan las URLs `/page1`, `/page2` y `/page3`. Problema resuelto.

**Joe:** Me gusta esa idea ... y creo que funcionará. (Sabía que había una razón por la que nos sentábamos a través de todas esas aburridas conferencias de CS).

**Jim:** Espera ... no tan rápido. Lo que estás sugiriendo con una función es mucho mejor que tu idea de copiar y pegar, pero todavía no estoy convencido de que sea la mejor manera de hacerlo.

**Frank y Joe** (juntos, e incrédulos): ¿Qué es no gustar?!?!

**Jim:** Me fastidia que planeas agregar código a las funciones que manejan las URLs / page1, / page2 y / page3 que no tienen nada que ver con lo que realmente hacen esas funciones. Por supuesto, usted necesita comprobar si un usuario está conectado antes de conceder el acceso, pero agregar una llamada de función para hacer esto a cada URL no se siente muy bien conmigo ...

**Frank:** Entonces, ¿cuál es tu gran idea, entonces?

**Jim:** Si fuera yo, crearía, y luego usaría, un decorador.

**Joe:** ¡Por supuesto! Esa es una idea aún mejor. Vamos a hacer eso.

## ***Copy-and-Paste Is Rarely a Good Idea*** ***Copiar y Pegar Es Raramente Un Buen Idea***

Vamos a convencernos de que las ideas sugeridas en la última página no son la mejor manera de abordar el problema en cuestión, es decir, la mejor manera de restringir el acceso a páginas web específicas.

La primera sugerencia fue copiar y pegar parte del código de la función que maneja la URL /status (es decir, la función `check_status`). Aquí está el código en cuestión:

This is the  
code to copy  
and paste.

```
@app.route('/status')
def check_status() -> str:
    if 'logged_in' in session:
        return 'You are currently logged in.'
    return 'You are NOT logged in.'
```

Este código devuelve un  
mensaje diferente basado en si  
el explorador del usuario está  
conectado o no.

A continuación se muestra la función page1:

```
@app.route('/page1')
def page1() -> str:
    return 'This is page 1.' ← Esta es la funcionalidad específica de la página.
```

Si copiamos y pegamos el código resaltado de `check_status` en `page1`, el código de este último terminaría parecido a esto:

```
@app.route('/page1')
def page1() -> str:
    if 'logged_in' in session:
        return 'This is page 1.'
    return 'You are NOT logged in.'
```

Compruebe si el navegador del usuario está conectado ...  
... entonces haga la funcionalidad específica de la página.  
De lo contrario, informe al usuario que no ha iniciado sesión.

El código anterior funciona, pero si tuvieras que repetir esta actividad de copiar y pegar para las URLs /page2 y /page3 (así como cualquier otra URL que debieras agregar a tu aplicación web), crearías rápidamente una pesadilla de mantenimiento. Especialmente si consideras todas las modificaciones que tendrías que hacer si decides cambiar la forma en que funciona tu código de inicio de sesión (por ejemplo, si compruebas un ID de usuario y una contraseña enviados a los datos almacenados en una base de datos).

## ***Put shared code into its own function***

## **Poner código compartido en su propia función**

Cuando tiene un código que debe utilizar en muchos lugares diferentes, la solución clásica al problema de mantenimiento inherente a cualquier "solución rápida" de copiar y pegar es poner el código compartido en una función que se invoca cuando sea necesario.

Como tal estrategia resuelve el problema de mantenimiento (ya que el código compartido existe en un solo lugar en lugar de ser copiado y pegado de manera voluntaria), veamos qué es lo que crea una función de comprobación de inicio de sesión para nuestra aplicación web.

## **use a function**

### ***Creating a Function Helps, But...***

*Creación de una función ayuda, pero ...*

Vamos a crear una nueva función llamada `check_logged_in`, que, cuando se invoca, devuelve `True` si el navegador del usuario está conectado actualmente y `False` en caso contrario.

No es un gran trabajo (la mayor parte del código ya está en `check_status`); Aquí es cómo escribiríamos esta nueva función:

```
def check_logged_in() -> bool:  
    if 'logged_in' in session:  
        return True  
    return False
```

En lugar de devolver un mensaje, este código devuelve un booleano basado en si el explorador del usuario está conectado o no.

Con esta función escrita, vamos a usarla en la función page1 en lugar de ese código copiado y pegado:

```
@app.route('/page1')
def page1() -> str:
    if not check_logged_in():
        return 'You are NOT logged in.'
    return 'This is page 1.'
```

The diagram shows the Python code for the `page1` function. There are three annotations with arrows pointing to specific parts of the code:

- An annotation on the first line of the function body (`if not check_logged_in():`) says: "We're checking if we are \*not\* logged in."
- An annotation on the line `return 'You are NOT logged in.'` says: "Call the 'check\_logged\_in' function to determine the login status, then act accordingly."
- An annotation on the line `return 'This is page 1.'` says: "This code only ever runs if the user's browser is logged in."

Traducimos:

1. Estamos revisando si no estamos conectados.
2. Llame a la función "check\_logged\_in" para determinar el estado de inicio de sesión y, a continuación, actúe en consecuencia.
3. Este código sólo se ejecuta si el explorador del usuario inicia sesión.

Esta estrategia es un poco mejor que copiar y pegar, ya que ahora puede cambiar la forma en que funciona el proceso de inicio de sesión haciendo cambios en la función `check_logged_in`. Sin embargo, para usar la función `check_logged_in` todavía tiene que hacer cambios similares a las funciones `page2` y `page3` (así como a cualquier nueva URL que cree), y lo hace copiando y pegando este nuevo código de `page1` en las otras funciones. De hecho, si comparas lo que hiciste con la función `page1` de esta página con lo que hiciste con `page1` en la última página, es casi la misma cantidad de trabajo, y sigue copiando y pegando. Además, con ambas de estas "soluciones", el código añadido está oscureciendo lo que `page1` realmente hace.

Sería bueno si de alguna manera puede comprobar si el navegador del usuario está conectado sin tener que modificar ninguno de los códigos de su función existente (para no ocultar nada). De esta forma, el código en cada una de las funciones de su webapp puede permanecer directamente relacionado con lo que hace cada función, y el código de estado de inicio de sesión no se interpondrá. ¿Si sólo hubiera una manera de hacer esto?

Como hemos aprendido de nuestros tres desarrolladores amigables -Frank, Joe y Jim- unas pocas páginas atrás, Python incluye una característica de lenguaje que puede ayudar aquí, y va por el decorador de nombre. Un decorador le permite aumentar una función existente con código extra, y lo hace al permitirle cambiar el comportamiento de la función existente sin tener que cambiar su código.

Si estás leyendo esa última frase y diciendo: "¿Qué?!?!", No te preocupes: suena extraño la primera vez que lo oyes. Después de todo, ¿cómo puedes cambiar el funcionamiento de una función sin cambiar el código de la función? ¿Tiene incluso sentido intentar?

Vamos a averiguar por aprender acerca de los decoradores.

## **You've Been Using Decorators All Along**

### **Has estado usando decoradores todo el tiempo**

Has estado usando decoradores durante todo el tiempo que has escrito webapps con Flask, el cual volviste al capítulo 5.

Esta es la versión más antigua de la `hello_flask.py` webapp de ese capítulo, que destaca el uso de un decorador llamado `@ app.route`, que viene con Flask. El decorador `@ app.route` se aplica a una función existente (hola en este código), y el decorador aumenta la función que precede organizando para llamar hola cada vez que la webapp procesa la URL `/`. Los decoradores son fáciles de detectar; Están prefijados con el símbolo `@`:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello() -> str:
    return 'Hello world from Flask!'
app.run()
```

Aquí está el decorador, que al igual que todos los decoradores- está prefijado con el símbolo `@`.

Tenga en cuenta cómo, como usuario del decorador `@ app.route`, no tiene idea de cómo funciona el decorador su magia. Todo lo que te preocupa es que el decorador haga lo que promete: enlaza una URL determinada con una función. Todos los detalles básicos, detrás de los escenarios de cómo funciona el decorador se oculta de usted.

Cuando usted decide crear un decorador, usted necesita mirar bajo las cubiertas y (al igual que cuando creó un administrador de contexto en el último capítulo) enganchar en la maquinaria del decorador de Python. Hay cuatro cosas que usted necesita saber y entender para escribir un decorador:

1. **Cómo crear una función**
2. **Cómo pasar una función como argumento a una función**
3. **Cómo devolver una función de una función**
4. **Cómo procesar cualquier número y tipo de argumentos de función**

Has estado creando y usando tus propias funciones desde el Capítulo 4, lo que significa que esta lista de "cuatro cosas que debes saber" es realmente sólo tres. Tomemos un cierto tiempo para trabajar con los artículos 2 a 4 de esta lista mientras que progresamos hacia escribir un decorador de los nuestros.

## **Las funciones son objetos**

### **Pass a Function to a Function**

#### **Pasar una función a una función**

- Pass a function to a function.
- Return a function from a function.
- Process any number/type of arguments.

↑  
Comprobaremos cada tema completado mientras trabajamos a través de este material.

Ha pasado un tiempo, pero en el capítulo 2 introducimos la noción de que todo es un objeto en Python. A pesar de que puede sonar contraintuitivo, el "todo" incluye funciones, lo que significa que las funciones son objetos, también.

Evidentemente, cuando se invoca una función, se ejecuta. Sin embargo, como todo lo demás en Python, las funciones son objetos y tienen un ID de objeto: piensa en funciones como "objetos de función".

Echa un vistazo a la breve sesión IDLE siguiente. Una cadena se asigna a una variable denominada msg, y luego su ID de objeto se reporta a través de una llamada a la función integrada de identificación (BIF). Una pequeña función, llamada hola, se define entonces. La función hello se pasa a la identificación BIF que informa el ID de objeto de la función. El tipo BIF confirma que msg es una cadena y hello es una función, y finalmente se invoca hello e imprime el valor actual de msg en la pantalla:

The screenshot shows a Python 3.5.1 Shell window. The session starts with defining a string 'msg' and its ID. Then, a function 'hello' is defined which prints 'msg'. The ID of 'hello' is checked, followed by its type being checked against both 'str' and 'function'. Finally, 'hello()' is called, printing the current value of 'msg'.

```
>>>
>>> msg = "Hello from Head First Python 2e"
>>> id(msg)
4385961264
>>> def hello():
    print(msg)

>>> id(hello)
4389417984
>>> type(msg)
<class 'str'>
>>> type(hello)
<class 'function'>
>>> hello()
Hello from Head First Python 2e
>>>
```

The "id" BIF reports the unique object identifier for any object provided to it.

The "type" BIF reports on an object's type.

Estábamos un poco tortuosos en no llamar su atención a esto antes de que tuvimos que mirar la sesión de IDLE por encima, pero ... ¿notó cómo pasamos hola a la identificación y

tipo BIFs? No invocamos `hola`; Pasamos su nombre a cada una de las funciones como un argumento. Al hacerlo, pasamos una función a una función.

### ***Functions can take a function as an argument***

### ***Las funciones pueden tomar una función como un argumento***

Las llamadas a `id` y `type` anteriores demuestran que algunas de las funciones integradas de Python aceptan una función como argumento (o para ser más precisos: un objeto de función). Lo que una función hace con el argumento depende de la función. Ni `id` ni `type` invoca la función, aunque podría tener. Veamos cómo funciona.

### ***Invoking a Passed Function***

### ***Invocación de una función pasada***

Cuando un objeto de función se pasa como un argumento a una función, la función de recepción puede invocar el objeto de función pasado.

Aquí hay una pequeña función (llamada `apply`) que toma dos argumentos: un objeto de función y un valor. La función `apply` invoca el objeto de función y pasa el valor a la función invocada como un argumento, devolviendo los resultados de invocar la función en el valor al código de llamada:

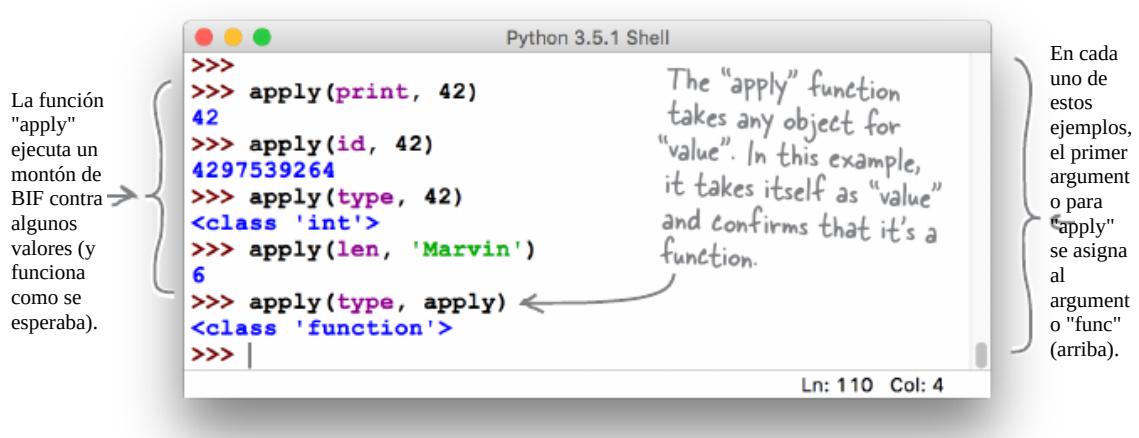
La función "apply" acepta un objeto de función como un argumento. La anotación "objeto" ayuda a confirmar nuestra intención aquí (y el uso del nombre del argumento "func" es una convención común).

```
func.py - /Users/paul/Documents/func.py (3.5.1)
def apply(func: object, value: object) -> object:
    return func(value)
```

Cualquier valor (de cualquier tipo) se puede pasar como el segundo argumento. En contra, las anotaciones indican lo que se permite como un tipo de argumento aquí: cualquier objeto.

Se invoca la función (pasada como un argumento), con el "valor" que se le pasa como su único argumento. El resultado de esta llamada de función se devuelve desde la función "apply".

Observe cómo las anotaciones de `apply` sugieren que acepta cualquier objeto de función junto con cualquier valor, y luego devuelve algo (que es todo muy genérico). Una prueba rápida de `aplicar` en el prompt >>> confirma que se aplican las obras como se esperaba:



Traducimos:

La función "aplicar" toma cualquier objeto para "valor". En este ejemplo, se toma como "valor" y confirma que es una función.

Si estás leyendo esta página y te preguntas cuándo necesitarías hacer algo como esto, no te preocunes: llegaremos a eso cuando escribamos a nuestro decorador. Por ahora, concentrarse en entender que es posible pasar un objeto de función a una función, que el último puede invocar.

### *functions inside functions*

### *Funciones dentro de funciones*

#### ***Functions Can Be Nested Inside Functions***

#### ***Funciones pueden ser anidadas dentro de funciones***

- Pass a function to a function.
- Return a function from a function.
- Process any number/type of arguments.

Por lo general, cuando se crea una función, se toma algún código existente y se vuelve reutilizable dándole un nombre y utilizando el código existente como suite de la función. Este es el caso de uso de función más común. Sin embargo, lo que a veces viene como una sorpresa es que, en Python, el código en la suite de una función puede ser cualquier código, incluido el código que define otra función (a menudo denominada función anidada o interna). Aún más sorprendente es que la función anidada puede ser devuelta desde la función envolvente externa; En efecto, lo que se devuelve es un objeto de función. Veamos algunos ejemplos que demuestran estos otros casos de uso de funciones menos comunes.

Primero es un ejemplo que muestra una función (llamada interna) anidada dentro de otra función (llamada externa). No es posible invocar interior desde cualquier lugar que no sea dentro de la suite exterior, ya que el interior es local en alcance a exterior:

```

def outer():
    def inner():
        print('This is inner.')
    print('This is outer, invoking inner.')
    inner()

```

The "inner" function is defined within the enclosing function's suite.

The "inner" function is invoked from "outer".

Traducimos:

1. La función "interna" se define dentro de la suite de funciones adjunta.
2. La función "interna" se invoca desde "exterior".

Cuando exterior u outer se invoca, ejecuta todo el código en su suite: inner se define, la llamada a la impresión BIF en exterior o outer se ejecuta, y luego se invoca la función interna inner(que llama a la impresión BIF dentro). Esto es lo que aparece en pantalla:

This is outer, invoking inner.  
This is inner.

The printed messages appear in the order: "outer" first, then "inner".

Traducimos:

Los mensajes impresos aparecen en el orden: "exterior" u "outer" primero, luego "interior" o "inner".

## ***When would you ever use this?***

### ***¿Cuándo usarías esto?***

Mirando este ejemplo simple, puede ser difícil pensar en una situación en la que crear una función dentro de otra función sería útil. Sin embargo, cuando una función es compleja y contiene muchas líneas de código, abstraer parte del código de la función en una función anidada a menudo tiene sentido (y puede facilitar la lectura del código de la función adjunta).

Un uso más común de esta técnica organiza para que la función de inclusión devuelva la función anidada como su valor, utilizando la instrucción return. Esto es lo que le permite crear un decorador.

Así pues, veamos qué sucede cuando devolvemos una función de una función.

## **Return a Function from a Function**

### **Devolver una función de una función**

Nuestro segundo ejemplo es muy similar al primero, pero por el hecho de que la función externa ya no invoca el interior, sino que lo devuelve. Echa un vistazo al código:

<input checked="" type="checkbox"/>	Pass a function to a function.
<input type="checkbox"/>	Return a function from a function.
<input type="checkbox"/>	Process any number/type of arguments.

```
def outer():
    def inner():
        print('This is inner.')
    print('This is outer, returning inner.')
    return inner
```

The "inner" function is still defined within "outer".

The "return" statement does not invoke "inner"; instead, it returns the "inner" function object to the calling code.

Traducimos:

1. La función "interna" todavía se define dentro de "exterior".
2. La instrucción "return" no invoca "inner"; En su lugar, devuelve el objeto de función "inner" al código de llamada.

Vamos a ver qué hace esta nueva versión de la función externa, volviendo a la shell IDLE y teniendo exterior para un giro.

Observe cómo asignamos el resultado de invocar exterior a una variable, llamada i en este ejemplo. A continuación, utilizamos i como si fuera un objeto de función, primero comprobando su tipo invocando el tipo BIF, invocando i como lo haríamos con cualquier otra función (añadiendo paréntesis). Cuando invocamos i, la función interna se ejecuta. En efecto, i es ahora un alias para la función interna como creada en el exterior:

The screenshot shows a Python 3.5.1 Shell window. The code is:

```
>>> i = outer()
>>> type(i)
<class 'function'>
>>> i()
This is inner.
```

Annotations explain:

- "The result of calling "outer" is assigned to a variable called "i"."
- "The "outer" function is invoked."
- "We check that "i" is, in fact, a function."
- "We invoke "i" and—voilà!—the "inner" function's code executes."

Traducimos:

1. El resultado de llamar "exterior" se asigna a una variable llamada "i".
2. Invocamos "i" y—voilà!—los. Ejecuta el código de la función "interna"
3. Comprobamos que "i" es, de hecho, una función.
4. Se invoca la función "externa".

Hasta aquí todo bien. Ahora puede devolver una función de una función, así como enviar una función a una función. Usted está casi listo para poner todo esto en su búsqueda para crear un decorador. Sólo hay una cosa más que debes entender: crear una función que pueda manejar cualquier número y tipo de argumentos. Veamos cómo hacerlo ahora.

## *Accepting a List of Arguments*

### *Aceptar una lista de argumentos*

Imagine que tiene un requisito para crear una función (que llamaremos `myfunc` en este ejemplo) que se puede llamar con cualquier número de argumentos. Por ejemplo, puede llamar `myfunc` de esta manera:

`myfunc(10)` *One argument*

<input checked="" type="checkbox"/>	Pass a function to a function.
<input checked="" type="checkbox"/>	Return a function from a function.
<input type="checkbox"/>	Process any number/type of arguments.

Ya casi estás allí. Un tema más a cubrir, y entonces usted estará listo para crear un decorador.

O puedes llamar a `myfunc` de esta manera:

`myfunc()` *No arguments*

O puedes llamar a `myfunc` de esta manera:

`myfunc(10, 20, 30, 40, 50, 60, 70)` *Muchos argumentos (que en este ejemplo, son todos, números, pero podría ser cualquier cosa: números, cadenas booleanas, lista.)*

De hecho, podrías llamar a myfunc con cualquier número de argumentos, con la salvedad de que no sabes de antemano cuántos argumentos van a ser proporcionados.

Como no es posible definir tres versiones distintas de myfunc para manejar cada una de las tres invocaciones anteriores, la pregunta es: ¿es posible aceptar cualquier número de argumentos en una función?

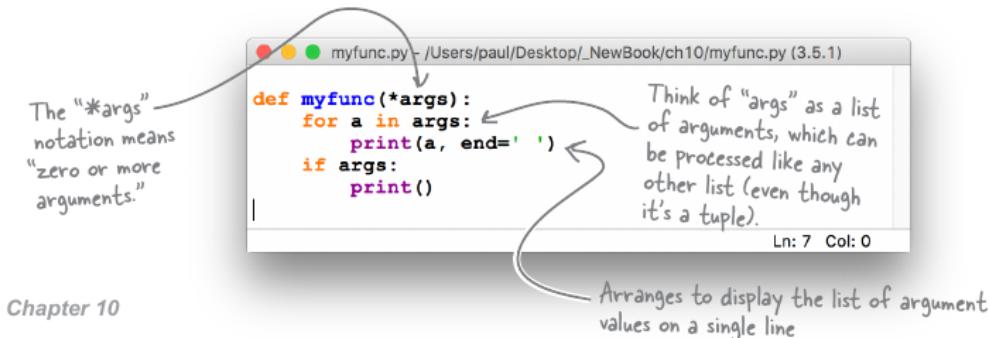
### **Use \* to accept an arbitrary list of arguments**

**Utilice \* para aceptar una lista arbitraria de argumentos**

**Piense en \* como el significado de "expandir a una lista de valores".**

Python proporciona una notación especial que le permite especificar que una función puede tomar cualquier número de argumentos (donde "cualquier número" significa "cero o más"). Esta notación utiliza el carácter `*` para representar cualquier número, y se combina con un nombre de argumento (por convención, `args` se utiliza) para especificar que una función puede aceptar una lista arbitraria de argumentos (aunque `* args` es técnicamente una tupla).

Esta es una versión de `myfunc` que usa esta notación para aceptar cualquier número de argumentos cuando se invoca. Si se proporcionan argumentos, `myfunc` imprime sus valores en la pantalla:



Traducimos:

1. La notación `* args` significa "cero o más argumentos".
2. Piense en `"args"` como una lista de argumentos, que pueden ser procesados como cualquier otra lista (aunque sea una tupla).
3. Se organiza para mostrar la lista de valores de argumento en una sola línea

### **Processing a List of Arguments**

**Procesamiento de una lista de argumentos**

Ahora que myfunc existe, vamos a ver si puede manejar las invocaciones de ejemplo de la última página, a saber:

```
myfunc(10)
myfunc()
myfunc(10, 20, 30, 40, 50, 60, 70)
```

Aquí hay otra sesión IDLE que confirma que `myfunc` está a la altura de la tarea. No importa cuántos argumentos suministramos (incluyendo ninguno), `myfunc` los procesa en consecuencia:

The screenshot shows a Python 3.5.1 Shell window. The code entered and its output are as follows:

```
>>> myfunc(10)
10
>>> myfunc()
>>> myfunc(10, 20, 30, 40, 50, 60, 70)
10 20 30 40 50 60 70
>>>
>>> myfunc(1, 'two', 3, 'four', 5, 'six', 7)
1 two 3 four 5 six 7
>>> |
```

Annotations on the left side of the shell window explain the behavior:

- "No matter the number of arguments provided, → 'myfunc' does the right thing (i.e., processes its arguments, no matter how many)."
- "When provided with no arguments, 'myfunc' does nothing."
- "You can even mix and match the types of the values provided, and 'myfunc' still does the right thing."

Ln: 37 Col: 4

Traducimos:

1. No importa el número de argumentos proporcionados, "myfunc" hace lo correcto (es decir, procesa sus argumentos, no importa cuántos).
2. Cuando se proporciona sin argumentos, "myfunc" no hace nada.
3. Incluso puede mezclar y combinar los tipos de los valores proporcionados, y "myfunc" sigue haciendo lo correcto.

### **\* works on the way in, too**

### **\* Funciona en el camino, también**

Si proporciona una lista a `myfunc` como argumento, la lista (a pesar de que contiene potencialmente muchos valores) se trata como un elemento (es decir, es una lista). Para instruir al intérprete a expandir la lista para que se comporte como si cada uno de los elementos de la lista fueran un argumento individual, prefija el nombre de la lista con el carácter \* al invocar la función.

Otra breve sesión IDLE demuestra la diferencia de \* puede tener:

The list is processed as a single argument to the function.

A screenshot of the Python 3.5.1 Shell. The code shown is:

```
>>> values = [1, 2, 3, 5, 7, 11]
>>> myfunc(values)
[1, 2, 3, 5, 7, 11]
>>> myfunc(*values)
1 2 3 5 7 11
>>>
>>>
```

Annotations in the image:

- An arrow points from the text "The list is processed as a single argument to the function." to the line `myfunc(values)`.
- An annotation next to the line `values = [1, 2, 3, 5, 7, 11]` says "A list of six integers".
- An annotation next to the line `myfunc(*values)` says "When a list is prefixed with "\*", it expands to a list of individual arguments."
- The status bar at the bottom right shows "Ln: 15 Col: 4".

Traducimos:

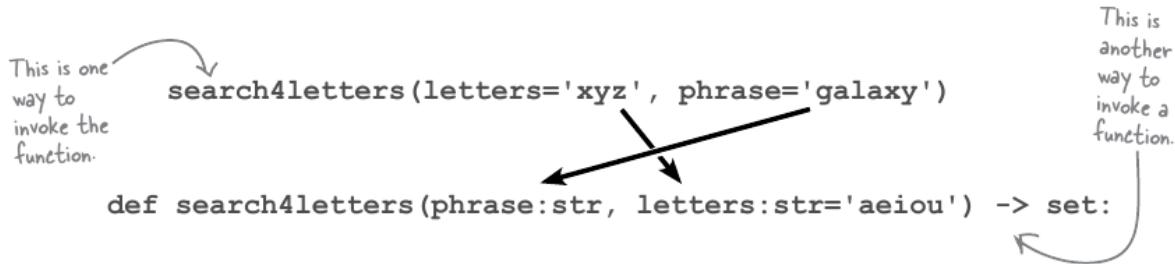
1. La lista se procesa como un único argumento a la función.
2. Una lista de seis enteros
3. Cuando una lista es prefijada con "\*", se expande a una lista de argumentos individuales.

## Accepting a Dictionary of Arguments

### Aceptación de un Diccionario de Argumentos

Cuando se trata de enviar valores a las funciones, también es posible proporcionar los nombres de los argumentos junto con sus valores asociados, y luego confiar en el intérprete para que coincida con las cosas en consecuencia.

Primero vio esta técnica en el capítulo 4 con la función `search4letters`, que -podrá recordar- espera dos valores de argumento, uno para la frase y otro para las letras. Cuando se utilizan argumentos de palabras clave, el orden en el que se proporcionan los argumentos a la función `search4letters` no importa:



Traducimos:

1. Esta es una forma de invocar la función.
2. Esta es otra forma de invocar una función.

Al igual que con las listas, también es posible organizar una función para aceptar un número arbitrario de argumentos de palabra clave, es decir, las teclas con los valores asignados a ellos (como con la frase y las letras en el ejemplo anterior).

## Use \*\* to accept arbitrary key word arguments

## **Utilice \*\* para aceptar arbitrariamente los argumentos de la palabra clave**

Además de la notación \*, Python también proporciona \*\*, que se expande a una colección de argumentos de palabras clave. Donde \* usa args como su nombre de variable (por convención), \*\* usa kwargs, que es abreviatura de "argumentos de palabra clave." (Nota: puede usar nombres distintos a args y kwargs dentro de este contexto, pero muy pocos programadores de Python. )

Veamos otra función, llamada myfunc2, que acepta cualquier número de argumentos de palabra clave:

*Piense en \*\* como significado "expanda a un diccionario de llaves y valores."*

The screenshot shows a Python script in IDLE with two functions:

```
def myfunc(*args):
    for a in args:
        print(a, end=' ')
if args:
    print()

def myfunc2(**kwargs):
    for k, v in kwargs.items():
        print(k, v, sep='->', end=' ')
if kwargs:
    print()
```

Annotations:

- A handwritten note on the left says: "Dentro de la función, "kwargs" se comporta como cualquier otro diccionario."
- An annotation above the first function says: "The '\*\*' tells the function to expect keyword arguments."
- An annotation on the right says: "Tome cada clave y el emparejamiento de valores en el diccionario, y muéstrela en la pantalla."

Traduciendo:

El "\*\*\*" indica a la función que espera argumentos de palabras clave.

## **Processing a Dictionary of Arguments**

### **Procesamiento de un Diccionario de Argumentos**

El código dentro de la suite de myfunc2 toma el diccionario de argumentos y los procesa, mostrando todos los pares clave / valor en una sola línea.

Aquí hay otra sesión IDLE que demuestra myfunc2 en acción. No importa cuántos pares clave / valor se proporcionan (incluyendo ninguno), myfunc2 hace lo correcto:

The screenshot shows the Python 3.5.1 Shell window. It contains the following text:

```

>>> myfunc2(a=10, b=20)
b->20 a->10
>>> myfunc2()
>>> myfunc2(a=10, b=20, c=30, d=40, e=50, f=60)
b->20 f->60 d->40 c->30 e->50 a->10
>>>
>>>

```

Annotations with arrows explain the code:

- An arrow from the text "Two keyword arguments provided" points to the first two arguments in the first call: `a=10, b=20`.
- An arrow from the text "Providing no arguments isn't an issue." points to the empty call `myfunc2()`.
- An arrow from the text "You can provide any number of keyword arguments, and 'myfunc2' does the right thing." points to the last call with six arguments: `a=10, b=20, c=30, d=40, e=50, f=60`.

Bottom status bar: Ln: 24 Col: 4

Traducción:

1. Dos argumentos de palabra clave proporcionados
2. Puede proporcionar cualquier número de argumentos de palabras clave y "myfunc2" hace lo correcto.
3. Proporcionar argumentos no es un problema.

***\*\* works on the way in, too***

***\*\* funciona en el camino, también***

Probablemente supuse que esto vendría, ¿no? Al igual que con \* args, cuando se usa \*\* kwargs también es posible usar \*\* cuando se invoca la función myfunc2. En lugar de demostrar cómo funciona esto con myfunc2, vamos a recordarle un uso previo de esta técnica anteriormente en este libro. En el Capítulo 7, cuando aprendió a usar la API de DB de Python, definió un diccionario de características de conexión de la siguiente manera:

```

dbconfig = {
    'host': '127.0.0.1',
    'user': 'vsearch',
    'password': 'vsearchpasswd',
    'database': 'vsearchlogDB', }

```

Un diccionario de pares clave / valor

Cuando llegó el momento de establecer una conexión con su servidor de base de datos MySQL (o MariaDB) en espera, utilizó el diccionario dbconfig de la siguiente manera. ¿Nota algo sobre la forma en que se especifica el argumento dbconfig?

`conn = mysql.connector.connect(**dbconfig)`

¿Le parece familiar?

Prefijando el argumento dbconfig con \*\*, le decimos al intérprete que trate el único diccionario como una colección de claves y sus valores asociados. En efecto, es como si invocara conectar con cuatro argumentos de palabras clave individuales, como esto:

```
conn = mysql.connector.connect('host='127.0.0.1', 'user='vsearch',
                               'password='vsearchpasswd', 'database='vsearchlogDB')
```

## *Accepting Any Number and Type of Function Arguments*

### *Aceptar cualquier número y tipo de argumentos de función*

Al crear sus propias funciones, es genial que Python le permita aceptar una lista de argumentos (usando \*), además de cualquier número de argumentos de palabras clave (usando \*\*). Lo que es aún más nítido es que puedes combinar las dos técnicas, que te permite crear una función que puede aceptar cualquier número y tipo de argumentos.

Aquí está una tercera versión de myfunc (que va por el nombre sorprendentemente imaginativo de myfunc3). Esta función acepta cualquier lista de argumentos, cualquier número de argumentos de palabra clave o una combinación de ambos:

The original "myfunc" works with any list of arguments.

The "myfunc2" function works with any amount of key/value pairs.

The "myfunc3" function works with any input, whether a list of arguments, a bunch of key/value pairs, or both.

```
def myfunc(*args):
    for a in args:
        print(a, end=' ')
    if args:
        print()

def myfunc2(**kwargs):
    for k, v in kwargs.items():
        print(k, v, sep='->', end=' ')
    if kwargs:
        print()

def myfunc3(*args, **kwargs):
    if args:
        for a in args:
            print(a, end=' ')
        print()
    if kwargs:
        for k, v in kwargs.items():
            print(k, v, sep='->', end=' ')
        print()
```

Both "\*args" and "\*\*kwargs" appear on the "def" line.

Traducción:

1. El "myfunc" original funciona con cualquier lista de argumentos.
2. La función "myfunc2" funciona con cualquier cantidad de pares clave / valor.
3. La función "myfunc3" funciona con cualquier entrada, ya sea una lista de argumentos, un montón de pares clave / valor o ambos.
4. Ambos "\* args" y "\*\* kwargs" aparecen en la línea "def".

Esta breve sesión IDLE muestra myfunc3:

The screenshot shows the Python 3.5.1 Shell window. On the left, there are two annotations: 'Funciona sin argumentos' with an arrow pointing to the first line of code, and 'Funciona con una combinación de una lista y argumentos de palabras clave' with an arrow pointing to the second line of code. The code itself is:

```

Python 3.5.1 Shell
>>> myfunc3()
>>> myfunc3(1, 2, 3) ← Works with a list
1 2 3
>>> myfunc3(a=10, b=20, c=30) ← Works with keyword arguments
a->10 b->20 c->30
>>> myfunc3(1, 2, 3, a=10, b=20, c=30)
1 2 3
a->10 b->20 c->30
>>>

```

Ln: 68 Col: 4

## **A Recipe for Creating a Function Decorator**

### **Una receta para crear un decorador de funciones**

- Pass a function to a function.
- Return a function from a function.
- Process any number/type of arguments.

Estamos listos para tener un ir en escribir su propio decorador.

Con tres elementos marcados en la lista de la derecha, ahora tiene una **comprendión** de las características del lenguaje Python que le permiten crear un decorador. Todo lo que usted necesita saber ahora es cómo usted toma estas características y las combina para crear el decorador que usted necesita.

Al igual que cuando creó su propio gestor de contexto (en el último capítulo), crear un decorador se ajusta a un conjunto de reglas o de receta. Recuerde que un decorador le permite aumentar una función existente con código extra, sin necesidad de cambiar el código de la función existente (que, admitiremos, todavía suena extraño).

Para crear un decorador de funciones, debe saber que:

#### **1. Un decorador es una función**

De hecho, en lo que respecta al intérprete, su decorador es simplemente otra función, aunque sea una que manipule una función existente. Hagamos esto hasta aquí en este libro, sabes que crear una función es fácil: usar la palabra clave `def` de Python.

#### **2. Un decorador toma la función decorada como un argumento**

Un decorador debe aceptar la función decorada como un argumento. Para hacer esto, simplemente pase la función decorada como un objeto de función a su decorador. Ahora que has trabajado en las últimas 10 páginas, sabes que esto también es fácil: llegas a un objeto de función haciendo referencia a la función sin paréntesis (es decir, usando sólo el nombre de la función).

### 3. Un decorador devuelve una nueva función

Un decorador devuelve una nueva función como su valor de retorno. Al igual que cuando exterior volvió interior (unas pocas páginas atrás), su decorador va a hacer algo similar, excepto que la función que devuelve debe invocar la función decorada. Hacer esto es atrevernos a decirlo? Fácil, pero para una pequeña complicación, que es lo que el punto 4 se trata.

### 4. Un decorador mantiene la firma de la función decorada

Un decorador debe asegurarse de que la función que devuelve toma el mismo número y tipo de argumentos como se esperaba por la función decorada. El número y tipo de argumentos de cualquier función se conoce como su firma (ya que la línea def de cada función es única).

Es hora de tomar un lápiz y poner esta información para trabajar creando su primer decorador.

## **Recapitulación: necesitamos restringir el acceso a determinadas URL**

*DE ACUERDO. Creo que estoy consiguiendo la mayor parte de esto. Pero recuerda ... ¿por qué estoy haciendo esto de nuevo?*

*Intentamos evitar copiar y pegar todo ese código de comprobación de estado de inicio de sesión.*

Hemos estado trabajando con el código simple\_webapp.py y necesitamos que nuestro decorador compruebe si el navegador del usuario está conectado o no. Si está conectado, las páginas web restringidas son visibles. Si el explorador no ha iniciado sesión, la aplicación web debería aconsejar al usuario que inicie sesión antes de ver las páginas restringidas. Vamos a crear un decorador para manejar esta lógica. Recuerde la función check\_status, que demuestra la lógica que queremos que nuestro decorador imite:

```
Queremos evitar copiar y pegar este código. ↗ @app.route('/status') def check_status() -> str:    if 'logged_in' in session:        return 'You are currently logged in.'    return 'You are NOT logged in.'
```

← Recuerde: este código devuelve un mensaje diferente basado en si el explorador del usuario está conectado o no.

## **Creating a Function Decorator**

## **Creación de un decorador de funciones**

Para cumplir con el punto 1 de nuestra lista, tuvo que crear una nueva función. Recuerda:

## **1. Un decorador es una función**

De hecho, en lo que respecta al intérprete, su decorador es simplemente otra función, aunque sea una que manipule una función existente. Hagamos referencia a esta función existente como la función decorada de aquí en adelante. Sabes que crear una función es fácil: usa la palabra clave def de Python.

El cumplimiento del punto 2 implica asegurar que su decorador acepte un objeto de función como argumento. De nuevo, recuerde:

## **2. Un decorador toma la función decorada como un argumento**

Su decorador debe aceptar la función decorada como un argumento. Para hacer esto, simplemente pase la función decorada como un objeto de función a su decorador. Llegará a un objeto de función haciendo referencia a la función sin paréntesis (es decir, utilizando el nombre de la función).



Vamos a poner su decorador en su propio módulo (para que pueda reutilizar más fácilmente). Comience creando un nuevo archivo llamado `checker.py` en su editor de texto.

Vas a crear un nuevo decorador en `checker.py` llamado `check_logged_in`. En el espacio de abajo, proporcione la línea de def de su decorador. Sugerencia: use func como el nombre de su objeto de función argumento:

Put the decorator's  
"def" line here.



<sup>there are no</sup>  
Dumb Questions

**P:** ¿Importa dónde en mi sistema creo `checker.py`?

**R:** Sí. Nuestro plan es importar `checker.py` en `webapps` que lo necesiten, por lo que necesita asegurarse de que el intérprete pueda encontrarlo cuando su código incluya la línea de import `checker`. Por ahora, ponga `checker.py` en la misma carpeta que `simple_webapp.py`.



Decidimos poner tu decorador en su propio módulo (para que puedas reutilizarlo más fácilmente).

Comenzó por crear un nuevo archivo denominado checker.py en el editor de texto. Tu nuevo decorador (en checker.py) se llama `check_logged_in` y, en el espacio siguiente, debes proporcionar la línea `def` de tu decorador:

`def check_logged_in(func):`



El decorador "check\_logged\_in" toma un solo argumento: el objeto de función de la función decorada.

***That's almost too easy, isn't it?***

***Eso es casi demasiado fácil, ¿no?***

Recuerde: un decorador es sólo otra función, que toma un objeto de función como un argumento (func en la línea def anterior).

Vamos a pasar al siguiente tema en nuestra receta "crear un decorador", que es un poco más involucrado (pero no por mucho). Recuerde lo que necesita su decorador para hacer:

### **3. Un decorador devuelve una nueva función**

Su decorador devuelve una nueva función como su valor devuelto. Al igual que cuando interior exterior volvió (algunas páginas hacia atrás), su decorador va a hacer algo similar, excepto que la función que devuelve debe invocar la función decorada.

Anteriormente en este capítulo, conoció la función externa outer, que, cuando se invoca, devolvió la función interna inner. Aquí está el código del exterior outer una vez más:

Todo este código está en la suite de función "outer".

```
def outer():
    def inner():
        print('This is inner.')
    print('This is outer, returning inner.')
    return inner
```

La función interna "inner" está anidada dentro de "outer".

El objeto de función "inner" se devuelve como resultado de invocar "outer". Tenga en cuenta la falta de paréntesis después de "inner", ya que estamos devolviendo un objeto de función. No estamos invocando el "inner".



Ahora que ha escrito la línea `def` de su decorador, vamos a añadir un código a su suite. Necesitas hacer cuatro cosas aquí.

1. Defina una función anidada llamada `wrapper` que es devuelta por `check_logged_in`. (Puede usar cualquier nombre de función aquí, pero, como verá en un poco, el envoltorio es una buena opción.)
  2. En el contenedor o wrapper, agregue parte del código de su función de `check_status` existente que implemente uno de los dos comportamientos en función de si el explorador del usuario está conectado o no. Para guardar el flip de la página, aquí está el código `check_status` una vez más (con los bits importantes resaltados):

```
@app.route('/status')
def check_status() -> str:
    if 'logged_in' in session:
        return 'You are currently logged in.'
    return 'You are NOT logged in.'
```

3. Según el elemento 3 de nuestra receta de creación de decorador, necesita ajustar el código de la función anidada para que invoque la función decorada (en lugar de devolver el mensaje "Estás conectado actualmente" o "You are currently logged in").
  4. Con la función anidada escrita, necesita devolver su objeto de función de `check_logged_in`.

Agregue el código requerido a la suite de `check_logged_in` en los espacios proporcionados a continuación:



Traducir:

1. Defina su función anidada.
  - 2 y 3. Agregue el código que desea ejecutar la función anidada.
  4. No olvide devolver la función anidada.



Con la línea de def de tu decorador escrita, debías añadir algún código a su suite. Necesitabas hacer cuatro cosas:

1. Defina una función anidada llamada wrapper que es devuelta por check\_logged\_in.
2. En el contenedor o wrapper, agregue parte del código de su función de estado de comprobación check\_status existente que implemente uno de los dos comportamientos en función de si el explorador del usuario está conectado o no.
3. Según el elemento 3 de nuestra receta de creación de decorador, ajuste el código de la función anidada para que invoque la función decorada (en contraposición a la devolución del mensaje "Estás conectado actualmente" o "You are currently logged in" message").
4. Con la función anidada escrita, devuelva su objeto de función de check\_logged\_in.

Debes agregar el código requerido a la suite de check\_logged\_in en los espacios proporcionados:

```
def check_logged_in(func):  
    A nested  
    "def" line  
    starts the  
    "wrapper"  
    function.  
  
    def wrapper():  
        if 'logged_in' in session:  
            return func()  
        else:  
            return 'You are NOT logged in.'  
  
    return wrapper  
  
Did you remember  
to return the  
nested function?  
  
If the user's  
browser is logged in...  
...invoke the  
decorated function.  
  
If the user's  
browser isn't  
logged in, return an  
appropriate message.
```

The diagram shows a hand-drawn style code snippet for a Python decorator. It starts with a regular function definition for 'check\_logged\_in'. Inside it, a nested 'def' line begins the 'wrapper' function. The code then branches: if 'logged\_in' is in 'session', it calls 'func()' and returns its result; otherwise, it returns the string 'You are NOT logged in.'. Finally, the outer function returns the 'wrapper' function itself. There are several handwritten annotations with arrows pointing to specific parts of the code:

- An arrow from the text "A nested 'def' line starts the 'wrapper' function." points to the start of the nested 'def wrapper():' line.
- An arrow from the text "Did you remember to return the nested function?" points to the final line 'return wrapper'.
- An arrow from the text "If the user's browser is logged in..." points to the condition 'if 'logged\_in' in session:'.
- An arrow from the text "...invoke the decorated function." points to the line 'return func()'.
- An arrow from the text "If the user's browser isn't logged in, return an appropriate message." points to the line 'return 'You are NOT logged in.''.

traducción:

1. Una línea "def" anidada inicia la función "wrapper".
2. ¿Recordó devolver la función anidada?
3. Si el navegador del usuario ha iniciado sesión ...
4. ... invocar la función decorada.
5. Si el navegador del usuario no está conectado, devuelva un mensaje apropiado.

**Can you see why the nested function is called "wrapper"?**  
**¿Puedes ver por qué la función anidada se llama "wrapper"?**

Si usted toma un momento para estudiar el código del decorador (hasta ahora), verá que la función anidada no sólo invoca la función decorada (almacenada en func), sino que también la aumenta añadiendo código adicional alrededor de la llamada. En este caso, el código

extra está comprobando si la clave `logged_in` existe en la sesión de su webapp. Criticamente, si el explorador del usuario no está conectado, la función decorada nunca es invocada por wrapper.

### ***The Final Step: Handling Arguments***

### ***El Paso Final: Manejando Argumentos***

Estamos casi allí - las "tripas" del código del decorador están en su lugar. Lo que queda es asegurar que el decorador maneje los argumentos de la función decorada correctamente, no importa lo que puedan ser. Recordar el punto 4 de la receta:

#### **4. Un decorador mantiene la firma de la función decorada**

Su decorador debe asegurarse de que la función que devuelve toma el mismo número y tipo de argumentos como se esperaba por la función decorada.

Cuando se aplica un decorador a una función existente, las llamadas a la función existente se reemplazan por llamadas a la función devuelta por el decorador. Como se vio en la solución de la página anterior, para cumplir con el punto 3 de nuestra receta de creación de decorador, devolvemos una versión envuelta de la función existente, que implementa código adicional según sea necesario. Esta versión envuelta adorna la función existente.

Pero hay un problema con esto, ya que hacer el envoltorio por sí solo no es suficiente; Las características de llamada de la función decorada también deben mantenerse. Esto significa, por ejemplo, que si su función existente acepta dos argumentos, su función wrapped también tiene que aceptar dos argumentos. Si pudiera saber de antemano cuántos argumentos esperar, entonces podría planificar en consecuencia. Por desgracia, no se puede saber esto con antelación, ya que su decorador puede aplicarse a cualquier función existente, que podría tener, literalmente, cualquier número y tipo de argumentos.

¿Qué hacer? La solución es ir "genérico", y organizar la función wrapper para soportar cualquier número y tipo de argumentos. Ya sabes cómo hacerlo, ya que ya has visto lo que `* args` y `** kwargs` pueden hacer.

***Recuerde: \* args y \*\* kwargs soportan cualquier número y tipo de argumentos.***

 **Sharpen your pencil** -Vamos a ajustar la función de contenedor wrapper para aceptar cualquier número y tipo de argumentos. Asegúrese también de que cuando se invoca `func`, utiliza el mismo número y tipo de argumentos que se pasaron a wrapper. Agregue el código de argumento en los espacios proporcionados a continuación:

What do you  
need to add to  
the "wrapper"  
function's  
signature?

```
def check_logged_in(func):
    def wrapper( ..... ):
        if 'logged_in' in session:
            return func( ..... )
        return 'You are NOT logged in.'
    return wrapper
```

traducimos:

¿Qué necesita agregar a la firma de la función "wrapper"?



Deberías ajustar la función wrapper para aceptar cualquier número y tipo de argumentos, así como asegurar que, cuando se invoca func, utiliza el mismo número y tipo de argumentos que se pasaron a wrapper:

El uso de una firma genérica hace el truco aquí, ya que soporta cualquier número y tipo de argumentos. Tenga en cuenta cómo invitamos "func" con los mismos argumentos suministrados a "wrapper", no importa lo que sean.

```
def check_logged_in(func):
    def wrapper( *args, **kwargs ):
        if 'logged_in' in session:
            return func( *args, **kwargs )
        return 'You are NOT logged in.'
    return wrapper
```

**We're done...or are we?**

**Ya terminamos ... ¿o estamos?**

Si comprueba nuestra receta de creación de decorador, se le perdonará por creer que hemos terminado. Estamos ... casi. Hay dos cuestiones que todavía tenemos que tratar: uno tiene que ver con todos los decoradores, mientras que el otro tiene que ver con este particular.

Vamos a sacar el tema específico fuera del camino primero. Como el decorador `check_logged_in` está en su propio módulo, debemos asegurarnos de que cualquier módulo al que hace referencia su código también se importe en `checker.py`. El decorador `check_logged_in` utiliza `session`, que debe importarse desde Flask para evitar errores. Manejar esto es sencillo, ya que todo lo que necesita hacer es agregar esta declaración de importación a la parte superior de `checker.py`:

```
from flask import session
```

La otra cuestión, que afecta a todos los decoradores, tiene que ver con cómo las funciones se identifican con el intérprete. Cuando se decoran, y si no se toma el debido cuidado, una función puede olvidar su identidad, lo que puede conducir a problemas. La razón por la que esto sucede es muy técnica y un poco exótica, y requiere un conocimiento de los internos de Python que la mayoría de la gente no necesita (o quiere) saber. Por lo tanto, la biblioteca estándar de Python viene con un módulo que maneja estos detalles para usted (por lo que nunca debe preocuparse por ellos). Todo lo que tienes que hacer es recordar importar el módulo requerido (`functools`), luego llamar a una sola función (`wraps`).

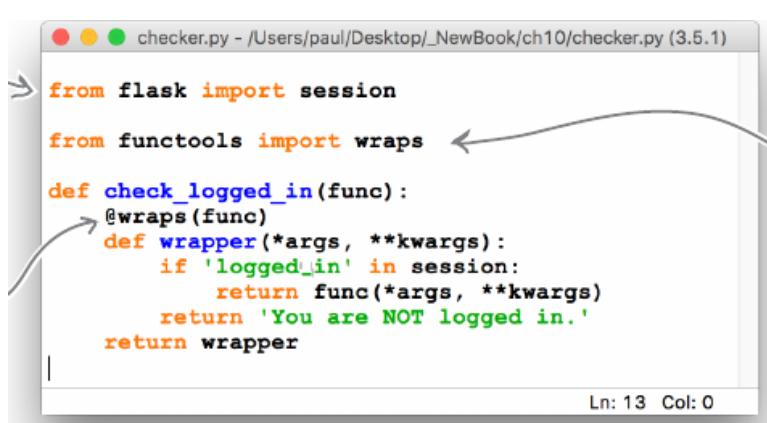
Tal vez algo irónico, la función de envolturas se implementa como un decorador, por lo que en realidad no lo llaman, sino que lo utilizan para decorar su función de envoltura dentro de su propio decorador. Ya hemos avanzado y hemos hecho esto por ti, y encontrarás el código en el decorador `check_logged_in` completado en la parte superior de la página siguiente.

*Al crear sus propios decoradores, siempre importar, a continuación, utilizar, "functools" del módulo "wraps" la función.*

## Your Decorator in All Its Glory

### Su decorador en toda su gloria

Antes de continuar, asegúrese de que su código de decorador coincide exactamente con el nuestro:



Asegúrese de importar "session" del módulo "flask".

Decora la función "wrapper" con el decorador "wraps" (asegúrandose de pasar "func" como argumento).

Importe la función "wraps" (que es un decorador) del módulo "functools" (que forma parte de la biblioteca estándar).

```
from flask import session
from functools import wraps

def check_logged_in(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        if 'logged_in' in session:
            return func(*args, **kwargs)
        return 'You are NOT logged in.'
    return wrapper
```

Ahora que el módulo `checker.py` contiene una función `check_logged_in` completada, vamos a usarla en `simple_webapp.py`. Esta es la versión actual del código de esta aplicación web (que mostramos aquí sobre dos columnas):

```

from flask import Flask, session

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello from the simple webapp.'

@app.route('/page1')
def page1() -> str:
    return 'This is page 1.'

@app.route('/page2')
def page2() -> str:
    return 'This is page 2.'

@app.route('/page3')
def page3() -> str:
    return 'This is page 3.'

@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    return 'You are now logged in.'

@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')
    return 'You are now logged out.'

@app.route('/status')
def check_status() -> str:
    if 'logged_in' in session:
        return 'You are currently logged in.'
    return 'You are NOT logged in.'

app.secret_key = 'YouWillNeverGuess...'

if __name__ == '__main__':
    app.run(debug=True)

```

Recall that our goal here is to restrict access to the /page1, /page2, and /page3 URLs, which are currently accessible to any user's browser (based on this code).

traducimos:

Recuerde que nuestro objetivo aquí es restringir el acceso a las URLs / page1, / page2 y / page3, que son actualmente accesibles para el navegador de cualquier usuario (basado en este código).

## ***Putting Your Decorator to Work***

### ***Ponga su decorador a trabajar***

Ajustar el código `simple_webapp.py` para usar el decorador `check_logged_in` no es difícil. Aquí está una lista de lo que debe suceder:

#### ***1. Importar el decorador***

El decorador `check_logged_in` debe importarse desde el módulo `checker.py`. Añadir la declaración de importación necesaria a la parte superior del código de nuestra webapp hace el truco aquí.

#### ***2. Eliminar cualquier código innecesario***

Ahora que el decorador `check_logged_in` existe, ya no tenemos ninguna necesidad de la función `check_status`, por lo que se puede eliminar de `simple_webapp.py`.

#### ***3. Utilice el decorador según sea necesario***

Para usar el decorador `check_logged_in`, aplícalo a cualquiera de las funciones de nuestra webapp usando la sintaxis `@`.

Aquí está el código a `simple_webapp.py` una vez más, con los tres cambios enumerados arriba aplicado. Observe cómo las URLs de / page1, / page2 y / page3 ahora tienen dos

decoradores asociados a ellas: `@ app.route` (que viene con Flask) y `@check_logged_in` (que acaba de crear):

```
from flask import Flask, session

from checker import check_logged_in

app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello from the simple webapp.'

@app.route('/page1')
@check_logged_in
def page1() -> str:
    return 'This is page 1.'

@app.route('/page2')
@check_logged_in
def page2() -> str:
    return 'This is page 2.'

@app.route('/page3')
@check_logged_in
def page3() -> str:
    return 'This is page 3.'

@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    return 'You are now logged in.'

@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')
    return 'You are now logged out.'

app.secret_key = 'YouWillNeverGuess...'

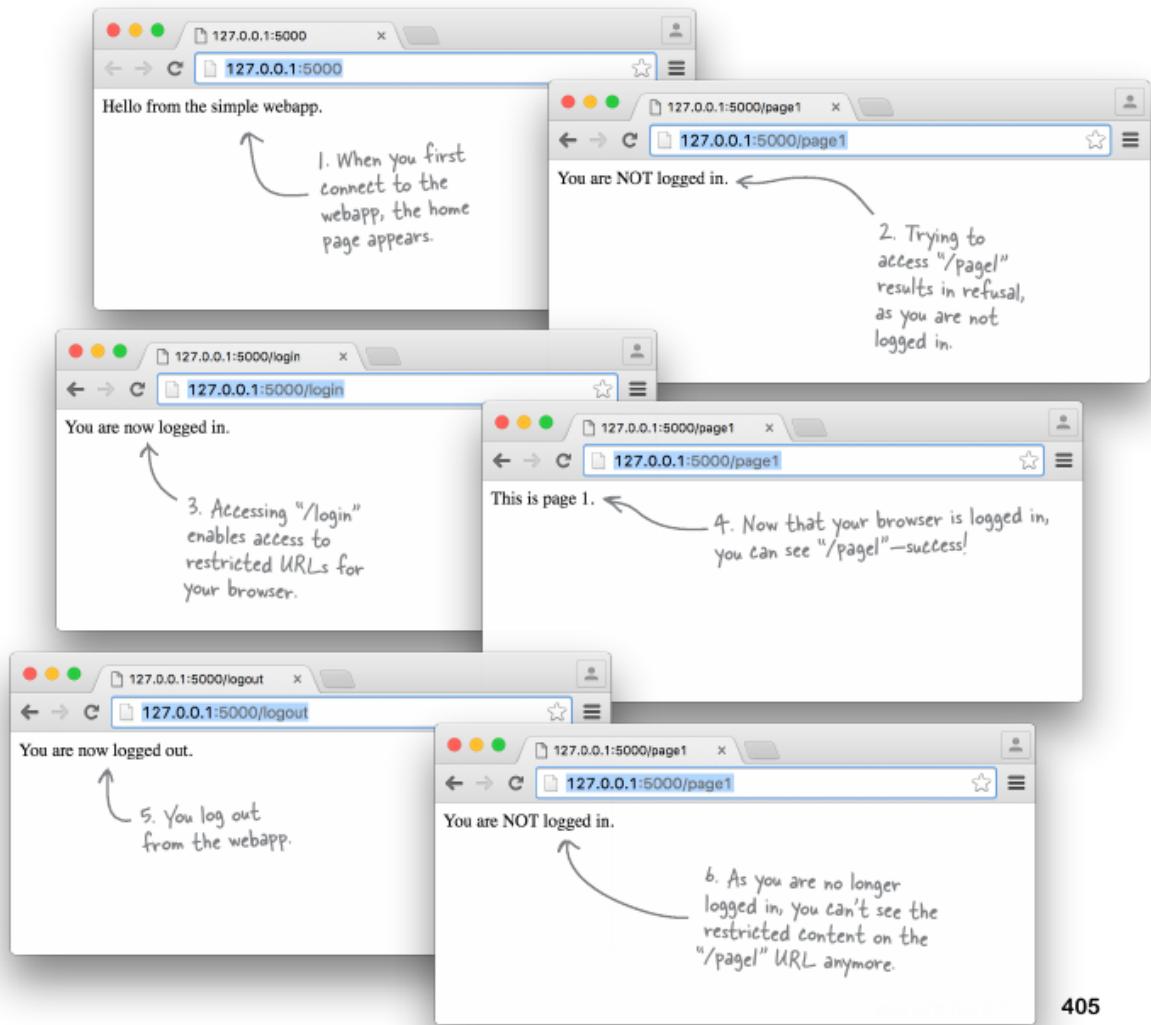
if __name__ == '__main__':
    app.run(debug=True)
```

No olvide aplicar estas ediciones resaltadas a  
su g. Webapp \* antes de continuar

## Test Drive

Para convencernos de que nuestro decorador de comprobación de inicio de sesión está funcionando según sea necesario, tomemos la versión habilitada para decoradores de simple\_webapp.py para darle un giro.

Con la aplicación Web en ejecución, utilice un navegador para intentar acceder a / page1 antes de iniciar sesión. Después de iniciar sesión, intente acceder a / page1 de nuevo y, después de cerrar la sesión, intente acceder al contenido restringido una vez más. Veamos qué pasa:



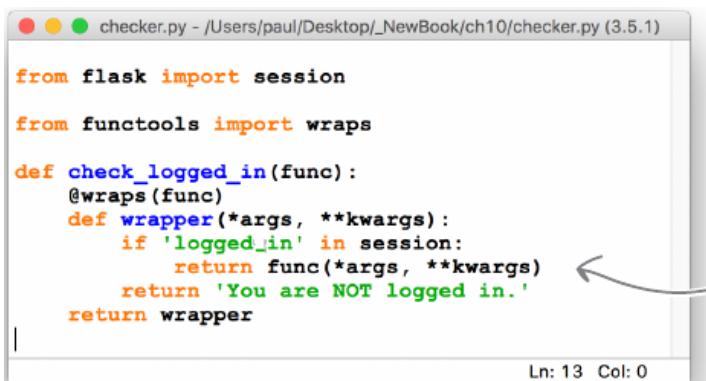
Traducimos:

1. 1. Cuando se conecta por primera vez a la aplicación web, aparece la página de inicio.
2. 2. El intento de acceder a "/ page1" resulta en la negativa, ya que no está conectado.
3. 3. El acceso a "/ login" permite acceder a URLs restringidas para su navegador.
4. 4. Ahora que su navegador está conectado, puede ver "/ page1" -success!
5. 5. Salga de la aplicación web.
6. 6. Ya que ya no ha iniciado sesión, ya no podrá ver el contenido restringido en la URL "/ page1".

## **Tengo que amar a los decoradores**

### **La belleza de los decoradores**

Eche otro vistazo al código para su decorador `check_logged_in`. Observe cómo se abstrae la lógica utilizada para comprobar si el explorador de un usuario está conectado, poniendo este código (potencialmente complejo) en un lugar, dentro del decorador, y luego haciéndolo disponible en todo el código, gracias a la sintaxis `@check_logged_in` decorator:



```
checker.py - /Users/paul/Desktop/_NewBook/ch10/checker.py (3.5.1)

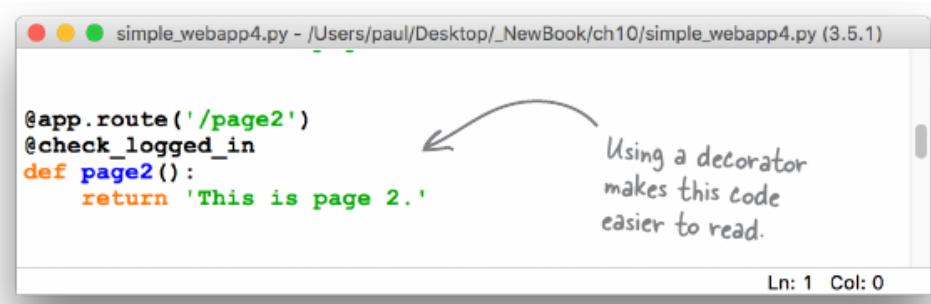
from flask import session
from functools import wraps

def check_logged_in(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        if 'logged_in' in session:
            return func(*args, **kwargs)
        return 'You are NOT logged in.'
    return wrapper

|                                     ←
Ln: 13 Col: 0
```

Este código parece extraño, pero no es realmente.

El código de resumen en un decorador hace que el código que lo usa sea más fácil de leer. Considere este uso de nuestro decorador en la URL /page2:



```
simple_webapp4.py - /Users/paul/Desktop/_NewBook/ch10/simple_webapp4.py (3.5.1)

@app.route('/page2')
@check_logged_in
def page2():
    return 'This is page 2.'                                     ←
                                                               Using a decorator
                                                               makes this code
                                                               easier to read.

Ln: 1 Col: 0
```

Observe cómo el código de la función `page2` sólo se ocupa de lo que necesita hacer: mostrar el contenido / page2. En este ejemplo, el código `page2` es una instrucción simple y simple; Sería más difícil de leer y comprender si también contenía la lógica necesaria para

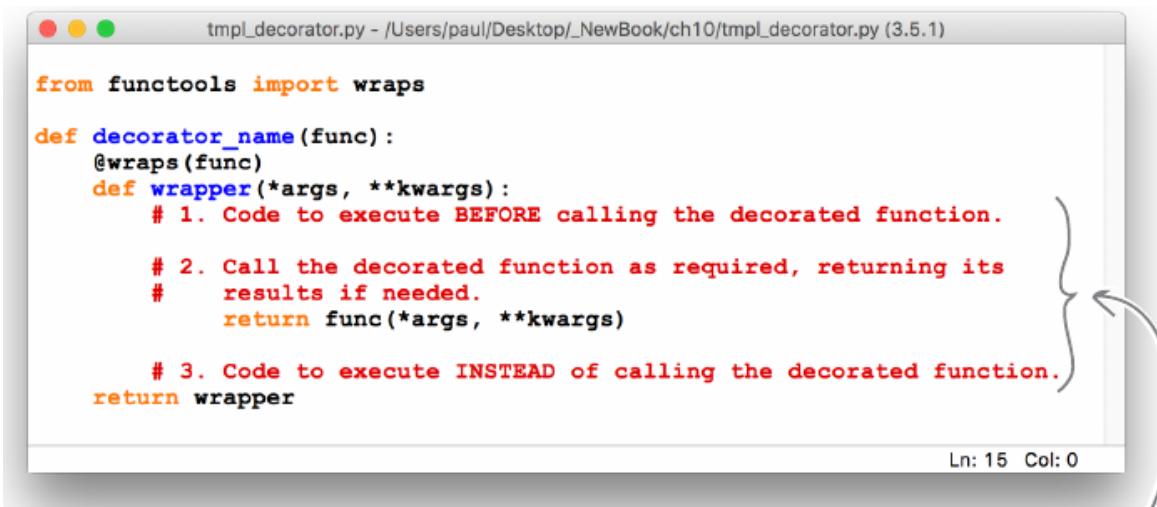
comprobar si el navegador de un usuario está conectado o no. El uso de un decorador para separar el código de inicio de sesión es un gran triunfo.

Esta "abstracción lógica" es una de las razones por las que el uso de decoradores es popular en Python. Otro es que, si lo piensas, al crear el `check_logged_in` en decorador, has logrado escribir código que aumenta una función existente con código extra, cambiando el comportamiento de la función existente sin cambiar su código. Cuando se introdujo por primera vez en este capítulo, esta idea fue descrita como "extraña". Pero, ahora que lo has hecho, realmente no hay nada, ¿verdad?

### ***Creación de más decoradores***

Con el proceso de crear el decorador `check_logged_in` detrás de usted, puede utilizar su código como la base de cualquier nuevo decorador que cree desde aquí.

Para facilitarte la vida, he aquí una plantilla de código genérica (en el archivo `tmpl_decorator.py`) que puedes usar como base de cualquier nuevo decorador que escribas:



```
from functools import wraps

def decorator_name(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        # 1. Code to execute BEFORE calling the decorated function.

        # 2. Call the decorated function as required, returning its
        #    results if needed.
        return func(*args, **kwargs)

        # 3. Code to execute INSTEAD of calling the decorated function.
    return wrapper
```

Sustituya estos comentarios por el código de su nuevo decorador.

Esta plantilla de código se puede ajustar según sea necesario para satisfacer sus necesidades. Todo lo que necesita hacer es dar a su nuevo decorador un nombre apropiado, a continuación, reemplazar los tres comentarios en la plantilla con el código específico de su decorador.

Si tiene sentido para su nuevo decorador para invocar la función decorada sin devolver los resultados, que está bien. Después de todo, lo que usted pone en su función de envoltura es su código, y usted es libre de hacer lo que quiera.

## there are no Dumb Questions

**P:** ¿No son decoradores como el gestor de contexto del capítulo anterior en que ambos me dejan envolver código con funcionalidad adicional?

**R:** Es una gran pregunta. La respuesta es sí y no. Sí, tanto los decoradores como los administradores de contexto aumentan el código existente con lógica adicional. Pero no, no son lo mismo. Los decoradores están específicamente interesados en aumentar las funciones existentes con funcionalidad adicional, mientras que los gestores de contexto están más interesados en asegurar que su código se ejecute dentro de un contexto específico, organizando el código para ejecutarse antes de una instrucción `with` y asegurándose de que el código siempre se ejecuta después de una sentencia `with`. Puedes hacer algo similar con los decoradores, pero la mayoría de los programadores de Python te consideran un poco loco si lo intentas. Además, tenga en cuenta que su código decorador no tiene ninguna obligación de hacer nada después de que invoca la función decorada (como es el caso con el decorador `check_logged_in`, que no hace nada). Este comportamiento decorador es muy diferente del protocolo que los gestores de contexto se espera que se adhieran a.

***Back to Restricting Access to /viewlog***

***Volver a restringir el acceso a /viewlog***

*Ah ah Ahora que puedo restringir las páginas de "simple\_webapp.py" puedo hacer mucho lo mismo para "vsearch4web.py", también, ¿verdad?*

*No es un caso de "lo mismo": es EXACTAMENTE el mismo. Es el mismo código; Solo reutilice las funciones de decorador, do\_login y do\_logout.*

Ahora que ha creado un mecanismo que le permite restringir el acceso a determinadas URL en `simple_webapp.py`, no es obvio que aplique el mismo mecanismo a cualquier otra aplicación web.

Esto incluye `vsearch4web.py`, donde tenía el requisito de restringir el acceso a la URL /`viewlog`. Todo lo que necesitas hacer es copiar las funciones `do_login` y `do_logout` desde `simple_webapp.py` en `vsearch4web.py`, importar el módulo `checker.py` y luego decorar la función `view_the_log` con `check_logged_in`. Por supuesto, es posible que desee añadir algo de sofisticación a `do_login` y `do_logout` (por, tal vez, verificando las credenciales de los usuarios con respecto a las almacenadas en una base de datos), pero, en lo que respecta a restringir el acceso a ciertas URL, el decorador `check_logged_in` realiza la mayor parte del trabajo pesado .

## **What's Next?**

### **¿Que sigue?**

En lugar de pasar un montón de páginas haciendo a vsearch4web.py lo que acabas de pasar un montón de tiempo haciendo a simple\_webapp.py, vamos a dejar el ajuste vsearch4web.py para que lo hagas por tu cuenta. Al comienzo del siguiente capítulo, presentaremos una versión actualizada de la aplicación web vsearch4web.py para compararla con la suya, ya que nuestro código actualizado se utiliza para enmarcar la discusión en el siguiente capítulo.

Hasta la fecha, todo el código de este libro se ha escrito bajo el supuesto de que nada malo nunca sucede, y nada nunca va mal. Esta fue una estrategia deliberada de nuestra parte, ya que queríamos asegurarnos de que tuviera una buena comprensión de Python antes de abordar temas como la corrección de errores, la eliminación de errores, la detección de errores, el manejo de excepciones y similares.

Hemos llegado al punto en que ya no podemos seguir esta estrategia. Los entornos en los que se ejecuta nuestro código son reales, y las cosas pueden (y no) salir mal. Algunas cosas son fijables (o evitables), y otras no. Si es posible, usted querrá que su código para manejar la mayoría de las situaciones de error, sólo resultando en un accidente cuando sucede algo realmente excepcional que está fuera de su control. En el próximo capítulo, analizamos varias estrategias para decidir qué es algo razonable que hacer cuando algo va mal.

## **Chapter 10's Code, 1 of 2**

This is  
"quick\_session.py".

```
from flask import Flask, session

app = Flask(__name__)

app.secret_key = 'YouWillNeverGuess'

@app.route('/setuser/<user>')
def setuser(user: str) -> str:
    session['user'] = user
    return 'User value set to: ' + session['user']

@app.route('/getuser')
def getuser() -> str:
    return 'User value is currently set to: ' + session['user']

if __name__ == '__main__':
    app.run(debug=True)
```

```
from flask import session  
  
from functools import wraps  
  
def check_logged_in(func):  
    @wraps(func)  
    def wrapper(*args, **kwargs):  
        if 'logged_in' in session:  
            return func(*args, **kwargs)  
        return 'You are NOT logged in.'  
    return wrapper
```

Se trata de "checker.py", que contiene el código del decorador de este capítulo: "check\_logged\_in".

Se trata de "tmpl\_decorator.py", que es una plantilla de creación de decorador práctica para que pueda reutilizar como mejor le parezca.

```
from functools import wraps  
  
def decorator_name(func):  
    @wraps(func)  
    def wrapper(*args, **kwargs):  
        # 1. Code to execute BEFORE calling the decorated function.  
  
        # 2. Call the decorated function as required, returning its  
        #     results if needed.  
        return func(*args, **kwargs)  
  
        # 3. Code to execute INSTEAD of calling the decorated function.  
    return wrapper
```

## ***Chapter 10's Code, 2 of 2***

```

from flask import Flask, session
from checker import check_logged_in
app = Flask(__name__)

@app.route('/')
def hello() -> str:
    return 'Hello from the simple webapp.'

@app.route('/page1')
@check_logged_in
def page1() -> str:
    return 'This is page 1.'

@app.route('/page2')
@check_logged_in
def page2() -> str:
    return 'This is page 2.'

@app.route('/page3')
@check_logged_in
def page3() -> str:
    return 'This is page 3.'

@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    return 'You are now logged in.'

@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')
    return 'You are now logged out.'

app.secret_key = 'YouWillNeverGuessMySecretKey'

if __name__ == '__main__':
    app.run(debug=True)

```

This is "simple\_webapp.py", which pulls all of this chapter's code together. When you need to restrict access to specific URLs, base your strategy on this webapp's mechanism.

We think the use of decorators makes this webapp's code easy to read and understand. Don't you? ☺

#### Traducimos:

1. Se trata de "simple\_webapp.py", que tira de todo el código de este capítulo junto. Cuando necesite restringir el acceso a URL específicas, base su estrategia en el mecanismo de esta aplicación web.
2. Creemos que el uso de los decoradores hace que el código de esta webapp sea fácil de leer y entender. ¿No es cierto?

# 11 exception handling

**Qué hacer cuando las cosas van mal**

**Programación funcional. Funciones de orden superior**

Python tiene algunas características de la programación funcional, un paradigma de programación basado en un concepto de función más cercano al matemático que al que habitualmente estamos habituados en la programación imperativa.

Python proporciona algunas funciones de orden superior y permite también definir otras a medida. El siguiente ejemplo construye un conversor de números anidando varias [funciones](#) en una **función de orden superior**.

```
def conversor(sis):
    def sis_bin(numero):
        print('dec:', numero, 'bin:', bin(numero))

    def sis_oct(numero):
        print('dec:', numero, 'oct:', oct(numero))

    def sis_hex(numero):
        print('dec:', numero, 'hex:', hex(numero))

    sis_func = {'bin': sis_bin, 'oct': sis_oct, 'hex': sis_hex}

    return sis_func[sis]

# Crea una instancia del conversor hexadecimal
```

```
conversorhex = conversor('hex')
```

```
# Convierte 100 de decimal a hexadecimal
```

```
conversorhex(100)
```

```
# Otro modo de usar el conversor.
```

```
# Convierte 9 de decimal a binario
```

```
conversor('bin')(9)
```

## La función map()

La función de orden superior **map()** aplica una función a una lista de datos y devuelve un [iterador](#) que contiene todos los resultados para los elementos de la lista.

En el siguiente ejemplo la función **cuadrado** calcula el cuadrado de un número. La **lista1** contiene una lista de datos numéricos. Con **map(cuadrado, lista1)** se aplica la función cuadrado a cada elemento de la lista.

```
def cuadrado(numero):
```

```
    return numero ** 2
```

```
lista1 = [-2, 4, -6, 8]
```

```
# Convierte a lista el iterador obtenido
```

```
lista2 = list(map(cuadrado, lista1))
```

```
# Muestra elementos de la lista
```

```
print(lista2) # 4, 16, 36, 64
```

Para calcular el área de un círculo necesitamos el número pi que está disponible, con un número suficiente de decimales, en el módulo de la biblioteca estándar math. En el siguiente ejemplo la función **area\_círculo** calcula el área de un círculo. La **lista3** tiene una lista de datos numéricos que son longitudes de radios diferentes. Con **map(area\_círculo, lista3)** se aplica la función a cada elemento de la lista.

```
import math

def area_circulo(radio):
    return math.pi * radio ** 2
```

```
lista3 = [1, 2, 3]

# Devuelve iterador que es convertido a lista
lista4 = list(map(area_circulo, lista3))

print(lista4)
```

Para convertir el iterador a una lista hemos empleado la función **list()**. También, podríamos haber recorrido el iterador con un **for...in**:

```
for resultado in map(area_círculo, lista3):
    print(resultado)
```

## La función filter()

La función **filter()** aplica un filtro a una lista de datos y devuelve un iterador con los elementos que superan el filtro.

```
# La función verifica si un número es negativo
```

```
def esneg(numero):
```

```

# Devuelve True/False según sea o no nº negativo

return (numero < 0)

lista5 = [-3, -2, 0, 1, 9, -5]

# Muestra los números negativos de la lista

# La función esneg() es llamada para comprobar,
# uno a uno, todos los números de la lista

print(list(filter(esneg, lista5)))

```

## La función reduce()

La función **reduce()** aplica una función a una lista de datos evaluando los elementos por pares. La primera vez se aplica al primer y segundo elemento, la siguiente, se aplicará al valor devuelto por la función junto al tercer elemento y así, sucesivamente, reduciendo la lista hasta que quede un sólo elemento.

A partir de Python 3 si queremos utilizar `reduce()` debemos importar el módulo `functools`:

```

import functools

def multiplicar(x, y):
    print(x * y) # muestra el resultado parcial
    return x * y

lista = [1, 2, 3, 4]
valor = functools.reduce(multiplicar, lista)
print(valor) # muestra el resultado final

```

## La función lambda

La función **lambda** se utiliza para declarar funciones que sólo ocupan una línea. Son objetos que se pueden asignar a variables para usar con posterioridad.

```
cuadrado = lambda x: x*x
```

```
lista = [1,2,3,5,8,13]
```

```
for elemento in lista:  
    print(cuadrado(elemento))
```

```
# Lambda, con 2 argumentos:
```

```
area_triangulo = (lambda b,h: b*h/2)
```

```
medidas = [(34, 8), (26, 8), (44, 18)]
```

```
for datos in medidas:  
    base = datos[0]  
    altura = datos[1]  
    print(area_triangulo(base, altura))
```

## Comprensión de listas

Es un tipo de construcción que consta de una expresión que determina cómo modificar los elementos de una lista, seguida de una o varias cláusulas **for** y, opcionalmente, una o varias cláusulas **if**. El resultado que se obtiene es una lista.

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```

# Cada elemento de la lista se eleva al cubo

cubos = [valor ** 3 for valor in lista]

print('Cubos de 1 a 10:', cubos)

numeros = [135, 154, 180, 193, 210]

divisiblespor3 = [valor for valor in numeros if valor % 3.0 == 0]

# Muestra lista con los números divisibles por 3

print(divisiblespor3)

# Define función devuelve el inverso de un número

def funcion(x):

    return 1/x

L = [1, 2, 3] # declara lista

# Muestra lista con inversos de cada número

print([funcion(i) for i in L])

```

## Generadores

Los generadores funcionan de forma parecida a la comprensión de listas pero no devuelven listas sino generadores. Un generador es una clase especial de función que genera valores sobre los que iterar. La sintaxis usada es como la

usada en la comprensión de listas pero en vez de usar corchetes se utilizan paréntesis. Para devolver los valores se utiliza **yield** en vez de **return**.

```
# Define generador

def generador(inicio, fin, incremento):

    while(inicio <= fin):

        yield inicio # devuelve valor

        inicio += incremento

# Recorre los valores del generador

for valor in generador(0, 6, 1):

    # Muestra valores, uno a uno:

    print(valor) # 0 1 2 3 4 5 6

# Obtiene una lista del generador

lista = list(generador(0, 8, 2))

# Muestra lista

print(lista) # [0,2,4,6,8]
```

## La función Decorador

Es una función que recibe una función como parámetro y devuelve otra función como valor de retorno. Se utiliza cuando es necesario definir varias funciones que son muy parecidas. La función devuelta actúa como un envoltorio (wrapper) resolviendo lo que sería común a todas las funciones. También se aplica a clases.

```
# Define decorador

def decorador1(funcion):
```

```
# Define función decorada

def funciondecorada(*param1, **param2):

    print('Inicio', funcion.__name__)

    funcion(*param1, **param2)

    print('Fin', funcion.__name__)

return funciondecorada
```

```
def suma(a, b):

    print(a + b)

suma2 = decorador1(suma)

suma2(1,2)

suma3 = decorador1(suma)

suma3(2,2)
```

```
# Otra forma más elegante, usando @:
```

```
@decorador1

def producto(arg1, arg2):

    print(arg1 * arg2)

producto(5,5)
```

```
# El siguiente decorador genera tablas de sumas

# y multiplicaciones. El código que es común a todas
```

```
# las funciones se declara en la función 'envoltura':
```

```
def tablas(funcion):

    def envoltura(tabla=1):

        print("Tabla del %i:" %tabla)

        print('-' * 15)

        for numero in range(0, 11):

            funcion(numero, tabla)

        print('-' * 15)

    return envoltura
```

```
@tablas
```

```
def suma(numero, tabla=1):

    print("%2i + %2i = %3i" %(tabla, numero, tabla+numero))
```

```
@tablas
```

```
def multiplicar(numero, tabla=1):

    print("%2i X %2i = %3i" %(tabla, numero, tabla*numero))
```

```
# Muestra la tabla de sumar del 1
```

```
suma()
```

```
# Muestra la tabla de sumar del 4
```

```
suma(4)
```

```
# Muestra la tabla de multiplicar del 1
```

```
multiplicar()
```

```
# Muestra la tabla de multiplicar del 10
```

```
multiplicar(10)
```

# 11 exception handling

***Las cosas van mal, todo el tiempo, no importa lo bueno que sea su código.***

Ha ejecutado correctamente todos los ejemplos de este libro y es probable que confíe en que todo el código presentado hasta el momento funciona. ¿Pero esto significa que el código es robusto? Probablemente no. Escribir código basado en el supuesto de que nada malo nunca sucede es (en el mejor de los casos) ingenuo. En el peor de los casos, es peligroso, como ocurren las cosas imprevistas. Es mucho mejor si te cuidas al codificar, en lugar de confiar. El cuidado es necesario para asegurar que su código haga lo que usted desea, así como reacciona correctamente cuando las cosas van al sur. En este capítulo, no sólo verá qué puede salir mal, sino también aprender qué hacer cuando (y, a menudo, antes) las cosas.

## **Long Exercise**

Estamos empezando este capítulo por buceo. A continuación se presenta el código más reciente de la `vsearch4web.py` webapp. Como veremos, hemos actualizado este código para usar el decorador `check_logged_in` del último capítulo para controlar cuándo la información presentada por la URL `/viewlog` es (y no es) visible para los usuarios.

Tómese el tiempo que necesite para leer este código y, a continuación, utilice un lápiz para circular y anotar las partes que crea que podrían causar problemas al operar en un entorno de producción. Resalta todo lo que piensas que puede causar un problema, no solo problemas potenciales de ejecución o errores.

```

from flask import Flask, render_template, request, escape, session
from vsearch import search4letters

from DBcm import UseDatabase
from checker import check_logged_in

app = Flask(__name__)

app.config['dbconfig'] = {'host': '127.0.0.1',
                        'user': 'vsearch',
                        'password': 'vsearchpasswd',
                        'database': 'vsearchlogDB', }

@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    return 'You are now logged in.'

@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')
    return 'You are now logged out.'

def log_request(req: 'flask_request', res: str) -> None:
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """insert into log
                  (phrase, letters, ip, browser_string, results)
                 values
                  (%s, %s, %s, %s, %s)"""
        cursor.execute(_SQL, (req.form['phrase'],
                            req.form['letters'],
                            req.remote_addr,
                            req.user_agent.browser,
                            res,))

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    log_request(request, results)
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

@app.route('/viewlog')
@check_logged_in
def view_the_log() -> 'html':
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """select phrase, letters, ip, browser_string, results
                  from log"""
        cursor.execute(_SQL)
        contents = cursor.fetchall()
    titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents,)

app.secret_key = 'YouWillNeverGuessMySecretKey'

if __name__ == '__main__':
    app.run(debug=True)

```

```
from flask import Flask, render_template, request, escape
from vsearch import search4letters

from DBcm import UseDatabase

app = Flask(__name__)

app.config['dbconfig'] = {'host': '127.0.0.1',
                         'user': 'vsearch',
                         'password': 'vsearchpasswd',
                         'database': 'vsearchlogDB', }

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """insert into log
                  (phrase, letters, ip, browser_string, results)
                  values
                  (%s, %s, %s, %s, %s)"""
        cursor.execute(_SQL, (req.form['phrase'],
                             req.form['letters'],
                             req.remote_addr,
                             req.user_agent.browser,
                             res, ))


@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
```

```

"""Extract the posted data; perform the search; return results."""

phrase = request.form['phrase']

letters = request.form['letters']

title = 'Here are your results:'

results = str(search4letters(phrase, letters))

log_request(request, results)

return render_template('results.html',
                      the_title=title,
                      the_phrase=phrase,
                      the_letters=letters,
                      the_results=results,)

@app.route('/')
@app.route('/entry')

def entry_page() -> 'html':

    """Display this webapp's HTML form."""

    return render_template('entry.html',
                          the_title='Welcome to search4letters on the web!')


@app.route('/viewlog')

def view_the_log() -> 'html':

    """Display the contents of the log file as a HTML table."""

    with UseDatabase(app.config['dbconfig']) as cursor:

        _SQL = """select phrase, letters, ip, browser_string, results
                  from log"""

        cursor.execute(_SQL)

        contents = cursor.fetchall()

    titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')

    return render_template('viewlog.html',

```

```

        the_title='View Log',
        the_row_titles=titles,
        the_data=contents,)

if __name__ == '__main__':
    app.run(debug=True)

```

## Long Exercise Solution

Debes tomar el tiempo que necesitas para leer el código que se muestra a continuación (que es una versión actualizada de la `vsearch4web.py` webapp). Luego, utilizando un lápiz, debía circular y anotar las partes que pensaba que podrían causar problemas al operar en un entorno de producción. Debía resaltar todo lo que pensaba que podría causar un problema, no sólo potenciales problemas de tiempo de ejecución o errores. (Hemos numerado nuestras anotaciones para facilitar la consulta.)

```

from flask import Flask, render_template, request, escape, session
from vsearch import search4letters

from DBcm import UseDatabase
from checker import check_logged_in

app = Flask(__name__)
app.config['dbconfig'] = {'host': '127.0.0.1',
                        'user': 'vsearch',
                        'password': 'vsearchpasswd',
                        'database': 'searchlogDB', }

1. What happens
if the database
connection fails?

@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    return 'You are now logged in.'

@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')
    return 'You are now logged out.'

def log_request(req: 'flask.Request', res: str) -> None:
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """insert into log
                  (phrase, letters, ip, browser_string, results)
                  values
                  (%s, %s, %s, %s)"""
        cursor.execute(_SQL, (req.form['phrase'],
                            req.form['letters'],
                            req.remote_addr,
                            req.user_agent.browser,
                            res, ))

```

2. Are these SQL statements protected from nasty web-based attacks such as SQL injection or Cross-site scripting?

3. What happens if executing these SQL statements takes a long time?

```

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    log_request(request, results) ← 4. What happens if this call fails?
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

@app.route('/viewlog')
@check_logged_in
def view_the_log() -> 'html':
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """select phrase, letters, ip, browser_string, results
                  from log"""
        cursor.execute(_SQL)
        contents = cursor.fetchall()
    titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                           the_title='View Log',
                           the_row_titles=titles,
                           the_data=contents)

app.secret_key = 'YouWillNeverGuessMySecretKey'

if __name__ == '__main__':
    app.run(debug=True)

```

Traducción:

1. 1. ¿Qué sucede si falla la conexión a la base de datos?
2. 2. ¿Están estas sentencias de SQL protegidas contra ataques desagradables de la web-base tales como la inyección del SQL o el scripting del sitio de la cruz?
3. 3. ¿Qué sucede si ejecuta estas sentencias de SQL toma mucho tiempo?
4. 4. ¿Qué sucede si falla esta llamada?

## **Databases Aren't Always Available**

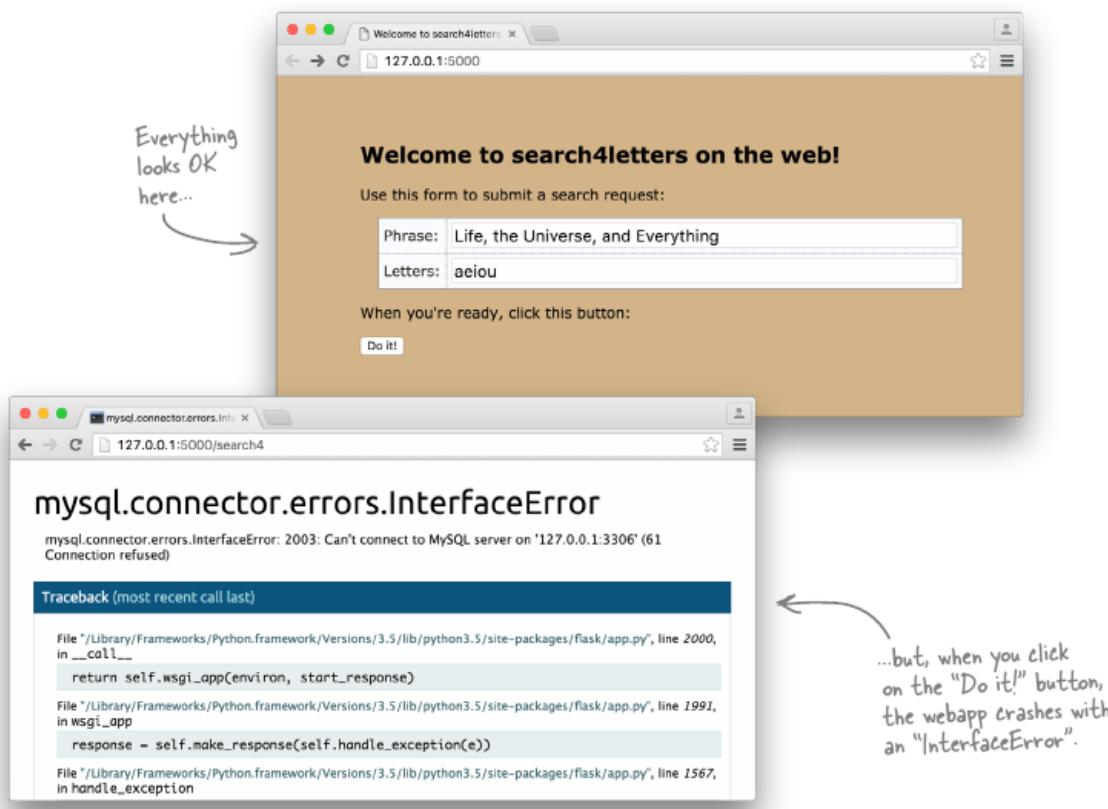
### **Las bases de datos no siempre están disponibles**

Hemos identificado cuatro problemas potenciales con el código vsearch4web.py, y admitimos que puede haber muchos más, pero ahora nos preocuparemos por estos cuatro problemas. Consideremos cada uno de los cuatro temas con mayor detalle (lo que hacemos aquí y en las siguientes páginas, simplemente describiendo los problemas, trabajaremos en soluciones más adelante en este capítulo). En primer lugar es preocupante acerca de la base de datos backend:

## 1. ¿Qué sucede si falla la conexión a la base de datos?

Nuestra webapp asume que la base de datos backend siempre está operativa y disponible, pero puede que no sea (por muchas razones). Por el momento, no está claro qué ocurre cuando la base de datos está inactiva, ya que nuestro código no considera esta eventualidad.

Veamos qué sucede si desactivamos temporalmente la base de datos backend. Como se puede ver a continuación, nuestra webapp carga bien, pero tan pronto como hacemos algo, aparece un mensaje de error intimidante:



... pero, al hacer clic en el botón "¡Hazlo!", La webapp se bloquea con un "InterfaceError".

## Web Attacks Are a Real Pain

### Los ataques Web son un verdadero dolor

Además de preocuparse por los problemas con su base de datos backend, también debe preocuparse por las personas desagradables que tratan de hacer cosas desagradables a su aplicación web, lo que nos lleva a la segunda cuestión:

## 2. ¿Está protegida nuestra webapp de los ataques web?

Las frases SQL inyección (SQLi) y Cross-site scripting (XSS) deben golpear el miedo en el corazón de cada desarrollador web. El primero permite a los atacantes explotar su base de datos back-end, mientras que el segundo les permite explotar su sitio web. Hay otras hazañas de la tela que usted necesitará preocuparse alrededor, pero éstos son los "dos grandes."

Al igual que con el primer número, vamos a ver qué sucede cuando tratamos de simular estas hazañas en contra de nuestra aplicación web. Como puede ver, parece que estamos listos para los dos:

The image contains four screenshots of a web application interface. The top-left window shows the search form with 'Phrase: x; show tables ;' and 'Letters: aeiou'. A callout points to the results page: 'If you try to inject SQL into the web interface, it has no effect (other than the expected "search4letters" output.)'. The top-right window shows the results page with the submitted data and the returned letters '{'a', 'e', 'o'}'. The bottom-left window shows the search form with 'Phrase: <script type='text/javascript'>alert('Hello!');</script>' and 'Letters: aeiou'. A callout points to the results page: 'Any attempt to exploit XSS by feeding JavaScript to the webapp has no effect.' The bottom-right window shows the results page with the submitted data and the returned letters '{'a', 'e', 'l', 'o'}'. A callout points to this result: 'The JavaScript isn't executed (thankfully); it's treated just like any other textual data sent to the webapp.'

Traducimos:

1. Cualquier intento de explotar XSS al alimentar JavaScript a la aplicación web no tiene ningún efecto.
2. Si intenta inyectar SQL en la interfaz web, no tiene ningún efecto (aparte de la salida esperada de "search4letters").
3. El JavaScript no se ejecuta (gracias); Se trata como cualquier otro dato textual enviado a la aplicación web.

## ***Input-Output Is (Some times) Slow***

### ***La entrada-salida es (a veces) lenta***

Por el momento, nuestra webapp se comunica con nuestra base de datos de backend de una manera casi instantánea, y los usuarios de nuestra webapp aviso poco o ningún retraso como la aplicación web interactúa con la base de datos. Pero imagínese si las interacciones con la base de datos backend tardaron un tiempo, quizás segundos:

#### ***3. ¿Qué pasa si algo lleva mucho tiempo?***

Tal vez la base de datos backend está en otra máquina, en otro edificio, en otro continente ... ¿qué pasaría entonces?

Las comunicaciones con la base de datos backend pueden tomar tiempo. De hecho, cada vez que el código tiene que interactuar con algo que es externo a él (por ejemplo: un archivo, una base de datos, una red, o lo que sea), la interacción puede tomar cualquier cantidad de tiempo, la determinación de que por lo general está fuera de su control . A pesar de esta falta de control, usted tiene que ser consciente de que algunas operaciones pueden ser largas.

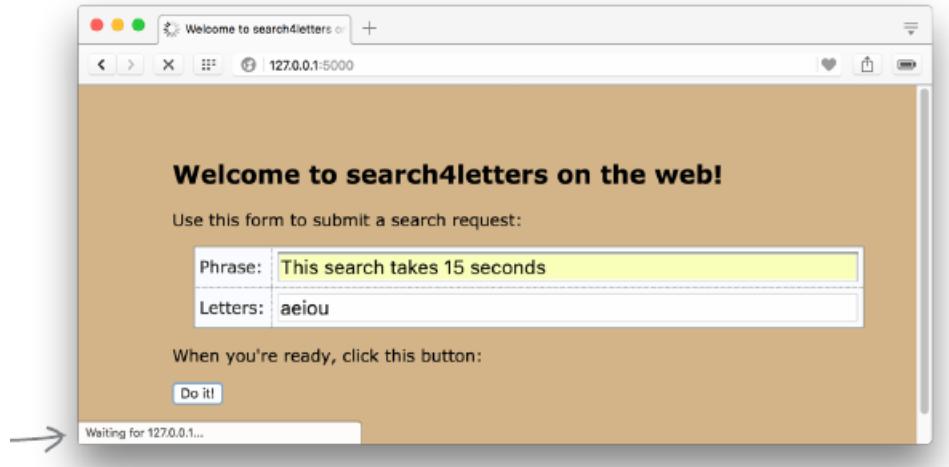
Para demostrar este problema, vamos a añadir un retraso artificial a nuestra aplicación web (utilizando la función de suspensión o sleep, que es parte del módulo de time de la biblioteca estándar). Agregue esta línea de código a la parte superior de su webapp (cerca de otras declaraciones de importación):

```
from time import sleep
```

Con la instrucción de importación anterior insertada, edite la función `log_request` e inserte esta línea de código antes de la instrucción `with`:

```
sleep(15)
```

Si reinicia su aplicación web, inicie una búsqueda, hay un retraso muy distinto mientras su navegador web espera que su aplicación web se ponga al día. A medida que vayan los retrasos web, 15 segundos se sentirán como una vida, lo que inducirá a la mayoría de los usuarios de su webapp a creer que algo se ha estrellado:



## **Your Function Calls Can Fail**

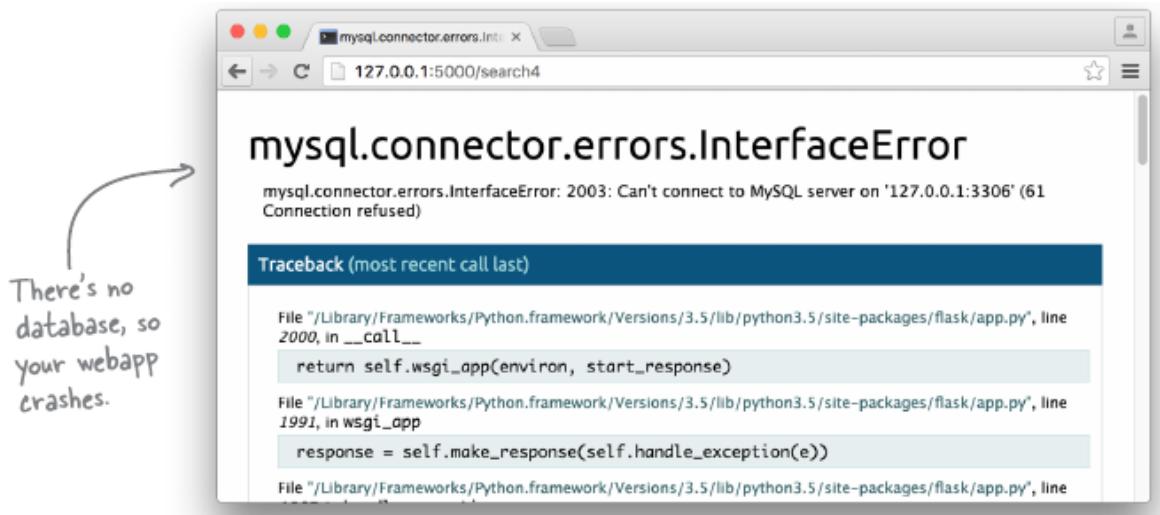
### **Sus llamadas de función pueden fallar**

El problema final identificado durante el ejercicio de apertura de este capítulo se relaciona con la llamada de función a `log_request` dentro de la función `do_search`:

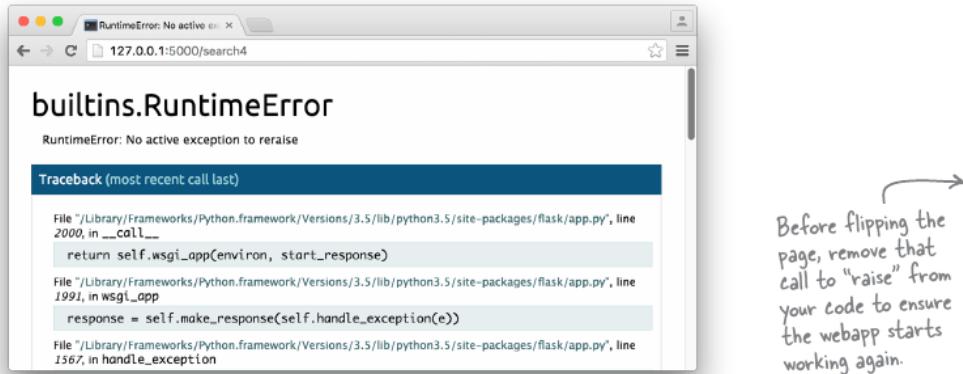
#### **4. ¿Qué sucede si falla una llamada de función?**

Nunca hay garantía de que una llamada de función tendrá éxito, especialmente si la función en cuestión interactúa con algo externo a su código.

Ya hemos visto lo que puede suceder cuando la base de datos backend no está disponible, la aplicación web se bloquea con un interfaz de error `InterfaceError`:



Otros problemas también pueden surgir. Para simular otro error, busque la línea de suspensión `sleep(15)` que agregó desde la discusión del problema 3 y reemplácela por una sola instrucción: `raise`. Cuando se ejecuta por el intérprete, el aumento fuerza un error de tiempo de ejecución. Si intenta su webapp de nuevo, un error diferente se produce esta vez:



Traducimos:

1. Otra cosa salió mal, y tu webapp se bloquea de nuevo.
2. Antes de volver a mover la página, elimine esa llamada para "aumentar" de su código para asegurarse de que la aplicación Web empieza a funcionar de nuevo.

### *¿qué hacer?*

#### **Teniendo en cuenta los problemas identificados**

Hemos identificado cuatro problemas con el código `vsearch4web.py`. Vamos a revisar cada uno y considerar nuestros próximos pasos.

#### **1. Su conexión a la base de datos falla**

Los errores se producen cada vez que un sistema externo en el que se basa su código no está disponible. El intérprete reportó un `InterfaceError` cuando ocurrió esto. Es posible detectar y reaccionar a estos tipos de errores utilizando el mecanismo de gestión de excepciones incorporado de Python. Si puede detectar cuando se produce un error, entonces está en condiciones de hacer algo al respecto.

#### **2. Su aplicación está sujeta a un ataque**

Aunque se puede argumentar que preocuparse por los ataques a su aplicación sólo preocupa a los desarrolladores web, las prácticas de desarrollo que mejoran la robustez del código que escribe son siempre dignas de consideración. Con `vsearch4web.py`, que se ocupa de los "dos grandes" vectores de ataque web, la inyección de SQL (SQLi) y Cross-site scripting (XSS), parece estar bien en la mano. Esto es más bien un accidente feliz que por diseño de su parte, ya que la biblioteca `Jinja2` está construida para protegerse contra XSS de forma

predeterminada, escapando de cualquier cadena potencialmente problemática (recuerde que el JavaScript que intentamos engañar a nuestra aplicación web no tuvo ningún efecto). En lo que respecta a SQLi, nuestro uso de las cadenas SQL parametrizadas de DB-API (con todos esos "placeholders") asegura -una vez más, gracias a la forma en que estos módulos fueron diseñados- que su código está protegido de toda esta clase de ataque.

*Si quieres saber más sobre SQLi y XSS, Wikipedia es un gran lugar para comenzar. Vea [https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection) y [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting), respectivamente. Y recuerde, hay todo tipo de otros tipos de ataques que pueden causar problemas para su aplicación; Estos son sólo los dos biggies.*

### **3. Su código tarda mucho tiempo en ejecutarse**

Si su código tarda mucho tiempo en ejecutarse, debe tener en cuenta el impacto en la experiencia de su usuario. Si su usuario no se da cuenta, es probable que esté bien. Sin embargo, si su usuario tiene que esperar, puede que tenga que hacer algo al respecto (de lo contrario, su usuario puede decidir que la espera no vale la pena, y vaya a otro lugar).

### **4. Su llamada de función falla**

No son sólo los sistemas externos los que generan excepciones en el intérprete; su código también puede generar excepciones. Cuando esto sucede, debe estar preparado para detectar la excepción y, a continuación, recuperarla según sea necesario. El mecanismo que utiliza para habilitar este comportamiento es el mismo que se insinúa en la discusión del número 1, anterior.

Entonces ... ¿por dónde empezamos cuando tratamos estos cuatro temas? Es posible utilizar el mismo mecanismo para tratar los temas 1 y 4, por lo que es donde comenzaremos.

## ***Always Try to Execute Error-Prone Code***

### ***Siempre trate de ejecutar código de error***

Cuando algo va mal con tu código, Python plantea una excepción de tiempo de ejecución. Piense en una excepción como un bloqueo de programa controlado activado por el intérprete.

Como usted ha visto con los números 1 y 4, las excepciones se pueden plantear en muchas circunstancias diferentes. De hecho, el intérprete viene con toda una serie de tipos de excepción incorporados, de los cuales RuntimeError (de la edición 4) es sólo un ejemplo. Además de los tipos de excepción incorporados, es posible definir sus propias excepciones personalizadas, y ha visto un ejemplo de esto también: la excepción InterfaceError (del número 1) está definida por el módulo MySQL Connector.

Para detectar (y, esperamos, recuperar) una excepción de tiempo de ejecución, implemente la sentencia Python try, que puede ayudarle a administrar las excepciones a medida que se producen en tiempo de ejecución.

*Para obtener una lista completa de las excepciones integradas, consulte <https://docs.python.org/3/library/exceptions.html>.*

Para detectar (y, esperamos, recuperar) una excepción de tiempo de ejecución o runtime, implemente la sentencia Python try, que puede ayudarle a administrar las excepciones a medida que se producen en tiempo de ejecución.

Para ver cómo se intenta en acción, consideremos primero un fragmento de código que podría fallar cuando se ejecuta. Aquí hay tres líneas de código inocentes, pero potencialmente problemáticas, para que usted considere:

No hay nada raro o maravilloso pasando aquí: el archivo nombrado y abierto, y sus datos se obtienen y luego se muestran en la pantalla.

```
try_examples.py - /Users/paul/Desktop/_NewBo...
with open('myfile.txt') as fh:
    file_data = fh.read()
print(file_data)
|
Ln: 5 Col: 0
```

No hay nada malo con estas tres líneas de código y -como se escribe actualmente- se ejecutarán. Sin embargo, este código podría fallar si no puede tener acceso a myfile.txt. Tal vez el archivo falte, o su código no tiene los permisos de lectura de archivos necesarios. Cuando el código falla, se genera una excepción:

Cuando se produce un error de ejecución o runtime, Python muestra un "rastreo o traceback", que detalla lo que salió mal y dónde. En este caso, el intérprete piensa que el problema está en la línea 2.

```
Whoops!
>>> ===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples.py =====
Traceback (most recent call last):
  File "/Users/paul/Desktop/_NewBook/ch11/try_examples.py", line 2, in <module>
    with open('myfile.txt') as fh:
FileNotFoundError: [Errno 2] No such file or directory: 'myfile.txt'
>>>
>>>
|
Ln: 119 Col: 4
```

A pesar de ser feo para mirar, el mensaje de seguimiento es útil.

Comencemos a aprender lo que puede hacer intentar ajustar el fragmento de código anterior para protegerlo contra esta excepción de `FileNotFoundException`.

## ***Catching an Error Is Not Enough***

### ***Detectar un error no es suficiente***

Cuando se produce un error de tiempo de ejecución, se genera una excepción. Si ignora una excepción generada, se le denomina sin captura y el intérprete terminará su código, luego mostrará un mensaje de error en tiempo de ejecución (como se muestra en el ejemplo de la parte inferior de la última página). Dicho esto, también se pueden capturar las excepciones planteadas (es decir, tratarlas) con la sentencia try. Tenga en cuenta que no es suficiente para detectar errores de ejecución, también tiene que decidir lo que va a hacer a continuación.

Tal vez usted decida ignorar deliberadamente la excepción planteada, y seguir adelante ... con los dedos firmemente cruzados. O tal vez intente ejecutar algún otro código en lugar del código que se estrelló, y seguir adelante. O quizás la mejor cosa a hacer es registrar el error antes de terminar su aplicación tan limpiamente como sea posible. Sea lo que sea que decida hacer, la sentencia try puede ayudar.

En su forma más básica, la sentencia try le permite reaccionar siempre que la ejecución de su código da lugar a una excepción elevada o raised. Para proteger el código con try, coloque el código dentro de la suite de try. Si se produce una excepción, se termina el código en la suite del try y, a continuación, se ejecuta el código en el try excepto el conjunto. La suite de excepción es donde se define lo que desea que suceda a continuación.

Actualizar el fragmento de código de la última página para mostrar un mensaje corto cada vez que se genera la excepción `FileNotFoundException`. El código de la izquierda es lo que tenías anteriormente, mientras que el código de la derecha ha sido modificado para aprovechar lo que intenta o `try` y `except` tiene que ofrecer:

***Cuando se genera un error de tiempo de ejecución, puede ser detectado o no capturado: "try" le permite detectar un error elevado o raised, y "except" le permite hacer algo al respecto.***

The diagram illustrates the evolution of Python code from a simple try block to one that includes an except clause. It shows two code snippets side-by-side:

```
try_examples.py - /Users/paul/Desktop/_NewBo...
with open('myfile.txt') as fh:
    file_data = fh.read()
print(file_data)
Ln: 5 Col: 0
```

Note how the entire code snippet is indented under the "try" statement.

```
try:
    with open('myfile.txt') as fh:
        file_data = fh.read()
    print(file_data)
except FileNotFoundError:
    print('The data file is missing.')
Ln: 8 Col: 0
```

The "except" statement is indented to the same level as its associated "try", and has its own suite.

This code is indented under the "except" clause and only executes if the "FileNotFoundException" exception is raised.

Traducimos:

1. Observe cómo el fragmento de código completo está sangrado en la instrucción "try".
2. La sentencia "except" está sangrada al mismo nivel que su "try" asociado, y tiene su propia suite.
3. Este código está sangrado en la cláusula "except" y sólo se ejecuta si se genera la excepción "FileNotFoundException".

Tenga en cuenta que lo que era tres líneas de código es ahora seis, lo que puede parecer inútil, pero no lo es. El fragmento de código original todavía existe como una entidad; compone la suite asociada con la sentencia try. La sentencia except y su suite es código nuevo. Veamos qué diferencia hacen estas enmiendas.



Vamos a tomar `try ... except` la versión de su fragmento de código para darle un giro. Si `myfile.txt` existe y es legible por su código, su contenido aparecerá en la pantalla. De lo contrario, se genera una excepción en tiempo de ejecución. Ya sabemos que `myfile.txt` no existe, pero ahora, en lugar de ver el feo mensaje de rastreo desde antes, se dispara el código de manejo de excepciones y se nos presenta un mensaje más amigable (a pesar de que nuestro fragmento de código se ha estrellado):

La primera vez que ejecutó el fragmento de código, el intérprete generó este feo traceback.

A screenshot of a Mac OS X window titled "Python 3.5.1 Shell". The code in the shell is:

```
>>> ===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples.py ======Traceback (most recent call last):  File "/Users/paul/Desktop/_NewBook/ch11/try_examples.py", line 2, in <module>    with open('myfile.txt') as fh:FileNotFoundException: [Errno 2] No such file or directory: 'myfile.txt'>>>>> ===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples2.py ======The data file is missing.>>> |
```

The line "The data file is missing." is highlighted in blue. Arrows point from the text "La primera vez que ejecutó el fragmento de código, el intérprete generó este feo traceback." to the line "FileNotFoundException: [Errno 2] No such file or directory: 'myfile.txt'" and from the text "La nueva versión del código produce mensajes mucho más amigables gracias a "try" y "except"" to the line "The data file is missing.". The status bar at the bottom right shows "Ln: 17 Col: 4".

La nueva versión del código produce mensajes mucho más amigables gracias a "try" y "except".

### **Puede haber más de una excepción planteada o raised...**

Este nuevo comportamiento es mejor, pero ¿qué pasa si myfile.txt existe, pero su código no tiene permiso para leer de él? Para ver qué sucede, creamos el archivo, luego establecemos sus permisos para simular esta eventualidad. Realizar el nuevo código produce este resultado:

¡Vaya! Estamos de vuelta a ver un feo mensaje de rastreo, como un "PermissionError" se lanza.

A screenshot of a Mac OS X window titled "Python 3.5.1 Shell". The code in the shell is:

```
>>> ===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples2.py ======The data file is missing.>>> ===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples2.py ======Traceback (most recent call last):  File "/Users/paul/Desktop/_NewBook/ch11/try_examples2.py", line 3, in <module>    with open('myfile.txt') as fh:PermissionError: [Errno 13] Permission denied: 'myfile.txt'>>>>> |
```

Arrows point from the text "multiple excepts" to the line "The data file is missing." and from the text "try Once, but except Many Times" to the line "PermissionError: [Errno 13] Permission denied: 'myfile.txt'". The status bar at the bottom right shows "Ln: 24 Col: 4".

multiple excepts

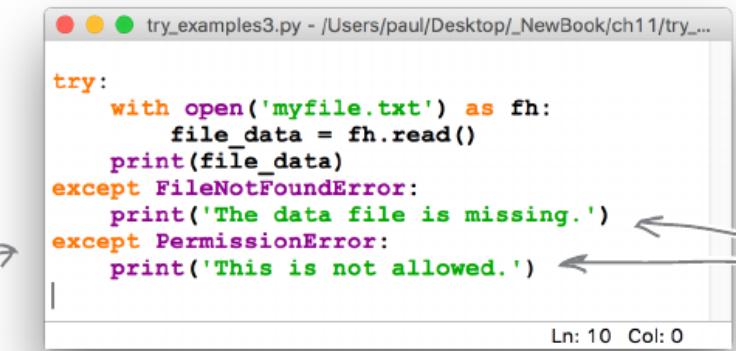
try Once, but except Many Times

### **Excepciones múltiples**

#### **Try una vez, pero except muchas veces**

Para protegerse contra otra excepción que se plantea, simplemente añada otra excepción a su sentencia try, identificando la excepción que le interesa y proporcionando cualquier código que considere necesario en la nueva suite except. He aquí otra versión actualizada del código que maneja la excepción PermissionError (si se lanza):

Además de excepciones "FileNotFoundException", este código también maneja un "PermissionError".

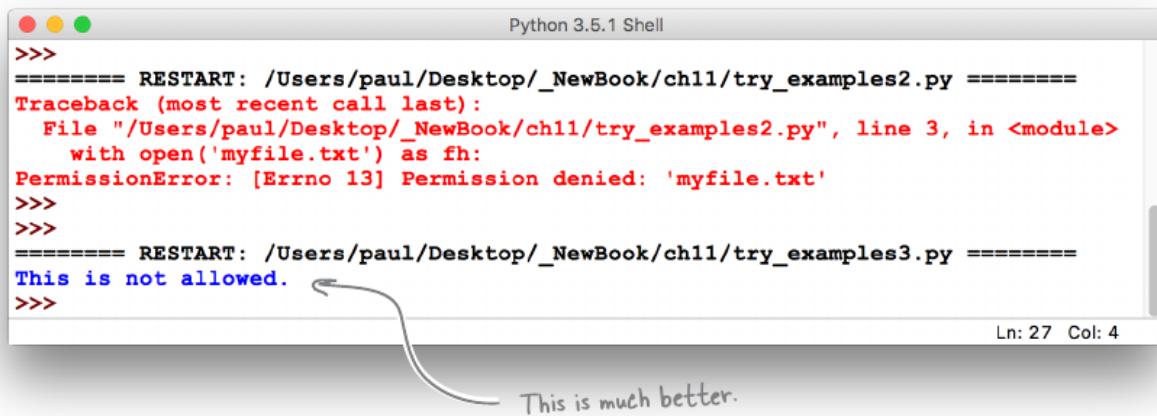


```
try:
    with open('myfile.txt') as fh:
        file_data = fh.read()
    print(file_data)
except FileNotFoundError:
    print('The data file is missing.')
except PermissionError:
    print('This is not allowed.')
```

Ln: 10 Col: 0

El código de las suites "except" puede hacer cualquier cosa. Por ahora, cada uno muestra un mensaje amistoso.

La ejecución de este código modificado todavía resulta en la excepción `PermissionError` que se lanza o raised. Sin embargo, a diferencia de antes, el feo traceback ha sido reemplazado por un mensaje mucho más amigable:



```
>>>
=====
RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples2.py =====
Traceback (most recent call last):
  File "/Users/paul/Desktop/_NewBook/ch11/try_examples2.py", line 3, in <module>
    with open('myfile.txt') as fh:
PermissionError: [Errno 13] Permission denied: 'myfile.txt'
>>>
>>>
=====
RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples3.py =====
This is not allowed.
```

Ln: 27 Col: 4

This is much better.

Esto se ve bien: has logrado ajustar lo que sucede siempre que el archivo con el que esperas trabajar no esté allí (no existe), o es inaccesible (no tienes los permisos correctos). ¿Pero qué sucede si se levanta o raised una excepción que usted no esperaba?

## *A Lot of Things Can Go Wrong*

### *Un montón de cosas pueden ir mal*

Antes de responder a la pregunta que se plantea al final de la última página, ¿qué sucede si se plantea una excepción que no esperaba? - eche un vistazo a algunas de las excepciones integradas de Python 3 (que se copian directamente de la documentación de Python) . No se sorprenda si usted está sorprendido por cuántos hay:



Traducimos:

1. Todas las excepciones incorporadas heredan de una clase llamada "Excepción".
2. Hay una gran cantidad de éstos, ¿no?
3. Aquí están las dos excepciones que nuestro código maneja actualmente.

Sería una locura tratar de escribir una suite aparte distinta para cada una de estas excepciones de tiempo de ejecución, ya que algunas de ellas nunca pueden ocurrir. Dicho esto, algunos pueden ocurrir, por lo que tiene que preocuparse por ellos un poco. En lugar de intentar manejar cada excepción individualmente, Python le permite definir un `except catch-all`, que se activa siempre que ocurre una excepción de tiempo de ejecución que no ha identificado específicamente.

catching all exceptions

The Catch-All Exception Handler

### **Cogiendo todas las excepciones**

#### **El controlador de excepción Catch-All**

Veamos qué sucede cuando ocurre algún otro error. Para simular tal ocurrencia, hemos cambiado myfile.txt de un archivo a una carpeta. Vamos a ver qué pasa cuando ejecutamos el código ahora:

A screenshot of a Mac OS X window titled "Python 3.5.2 Shell". The window contains a command-line interface. The user has typed "python try\_examples3.py" and the script has run. The output shows a traceback for an exception:

```
>>>
=====
RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples3.py =====
Traceback (most recent call last):
  File "/Users/paul/Desktop/_NewBook/ch11/try_examples3.py", line 3, in <module>
    with open('myfile.txt') as fh:
IsADirectoryError: [Errno 21] Is a directory: 'myfile.txt'
>>>
```

The cursor is at the end of the line "Is a directory: 'myfile.txt'". A red arrow points from the word "IsADirectoryError" in the error message to the explanatory text below.

Otra excepción ha ocurrido.

Otra excepción se lanza. Puede crear un extra, excepto la suite que se dispara cuando se produce esta excepción de `IsADirectoryError`, pero vamos a especificar un error de tiempo de ejecución catch-all en su lugar, que se activa cada vez que se produce alguna excepción (distinta de las dos ya especificadas). Para ello, agregue una sentencia `except` catch-all al final del código existente:

A screenshot of a Mac OS X window titled "try\_examples4.py - /Users/paul/Desktop/\_NewBook/ch11/tr...". The window contains a code editor with the following Python script:

```
try:
    with open('myfile.txt') as fh:
        file_data = fh.read()
    print(file_data)
except FileNotFoundError:
    print('The data file is missing.')
except PermissionError:
    print('This is not allowed.')
except:
    print('Some other error occurred.')
```

Annotations on the left side of the code explain the purpose of the "except" clause:

Esta sentencia "except" es "descubierta": no se refiere a una excepción específica.

Annotations on the right side of the code explain the purpose of the catch-all clause:

Este código proporciona un controlador de excepción catch-all.

Ejecutar esta versión modificada de su código se deshace del feo traceback, mostrando un mensaje amistoso en su lugar. No importa qué otra excepción ocurra, este código lo maneja gracias a la adición de la sentencia `except` catch-all :

Esto se ve mejor.

A screenshot of the Python 3.5.2 Shell window. The title bar says "Python 3.5.2 Shell". The code entered was:

```
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples3.py ======
Traceback (most recent call last):
  File "/Users/paul/Desktop/_NewBook/ch11/try_examples3.py", line 3, in <module>
    with open('myfile.txt') as fh:
IsADirectoryError: [Errno 21] Is a directory: 'myfile.t.txt'
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples4.py ======
Some other error occurred.
>>> |
```

The output shows a detailed traceback for the first error, identifying the file name and line number, and then a general message "Some other error occurred." at the bottom. A red arrow points from the word "IsADirectoryError" in the first error message to the word "IsADirectoryError" in the title of the slide. The status bar at the bottom right shows "Ln: 16 Col: 4".

## ¿Acaso no hemos perdido algo?

*DE ACUERDO. Tengo lo que está pasando aquí. ¿Pero este código ahora no oculta el hecho de que acabamos de tener un "IsADirectoryError"? ¿No es importante saber exactamente qué error ha encontrado?*

### Ah, sí ... buena captura.

Este último código ha arreglado la salida (en la que ha desaparecido el rastreo feo), pero también ha perdido alguna información importante: ya no sabe cuál es el problema específico con su código.

Saber qué excepción se planteó a menudo es importante, por lo que Python le permite obtener los datos asociados con la información de excepción más reciente, ya que se está manejando. Hay dos formas de hacerlo: utilizar las instalaciones del módulo `sys` y usar una extensión de la sintaxis `try / except`.

Echemos un vistazo a ambas técnicas.

there are no  
Dumb Questions

**P:** ¿Es posible crear un controlador de excepción catch-all que no hace nada?

**R:** Sí. A menudo es tentador añadir esto, en el suite `except`, al final de una sentencia `try`:

```
except:
    pass
```

Por favor, trate de no hacerlo. Esto, a excepción de la suite, implementa un catch-all que ignora cualquier otra excepción (presumiblemente en la esperanza errónea de que si algo se ignora podría desaparecer). Esta es una práctica peligrosa, ya que, como mínimo, una excepción inesperada debería producir un mensaje de error en la pantalla. Por lo tanto, asegúrese de escribir siempre código de comprobación de errores que maneja excepciones, en lugar de ignorarlas.

## ¿Qué salió mal?

### Aprendiendo sobre las excepciones de "sys"

La biblioteca estándar viene con un módulo llamado `sys` que proporciona acceso a los intérpretes, internos (un conjunto de variables y funciones disponibles en tiempo de ejecución).

Una de estas funciones es `exc_info`, que proporciona información sobre la excepción que se está manejando actualmente. Cuando se invoca, `exc_info` devuelve una tupla de tres valores donde el primer valor indica el tipo de excepción, el segundo detalla el valor de la excepción y el tercero contiene un objeto de rastreo que proporciona acceso al mensaje de rastreo (si lo necesita). Cuando no hay ninguna excepción disponible actualmente, `exc_info` devuelve el valor nulo de Python para cada uno de los valores de la tupla, que tiene este aspecto: (None, None, None).

Sabiendo todo esto, vamos a experimentar en la shell >>>. En la sesión IDLE que sigue, hemos escrito algún código que siempre va a fallar (como dividir por cero nunca es una buena idea). Un catch-all except suite utiliza la función `sys.exc_info` para extraer y mostrar datos relacionados con la excepción de disparo actual:

The screenshot shows a Python 3.5.2 Shell window. The code runs as follows:

```
>>> ===== RESTART: Shell =====
>>> import sys
>>>
>>> try:
>>>     1/0
>>> except:
>>>     err = sys.exc_info()
>>>     for e in err:
>>>         print(e)
```

Annotations explain the code and output:

- An arrow points from the `import sys` line to the text: "Be sure to import the "sys" module."
- An arrow points from the `1/0` line to the text: "Dividing by zero is \*never\* a good idea...and when your code divides by zero an exception occurs"
- An arrow points from the `err = sys.exc_info()` line to the text: "Let's extract and display the data associated with the currently occurring exception."
- An arrow points from the brace under the `print(e)` line to the text: "Here's the data associated with the exception, which confirms that we have an issue with divide-by-zero."
- The status bar at the bottom right shows "Ln: 117 Col: 4".

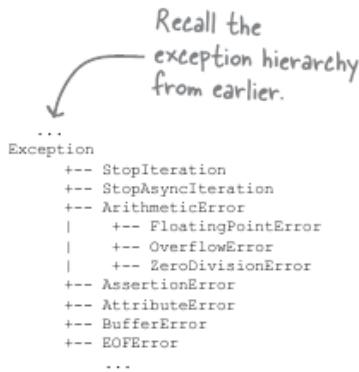
Traducimos:

1. Asegúrese de importar el módulo "sys".
2. Dividir por cero es \* nunca \* una buena idea ... y cuando su código divide por cero se produce una excepción
3. Vamos a extraer y mostrar los datos asociados con la excepción que se está produciendo actualmente.
4. Aquí están los datos asociados con la excepción, lo que confirma que tenemos un problema con la división por cero.

Es posible profundizar más en el objeto de rastreo para aprender más sobre lo que acaba de suceder, pero esto ya se siente como demasiado trabajo, ¿no? Todo lo que realmente queremos saber es qué tipo de excepción se produjo.

Para hacer esto (y tu vida) más fácil, Python extiende la sintaxis `try / except` para que sea conveniente obtener la información devuelta por la función `sys.exc_info()`, y lo hace sin tener que recordar importar el módulo `sys` o Wrangle con la tupla devuelta por esa función.

Recuerde de unas pocas páginas que el intérprete organiza excepciones en una jerarquía, con cada excepción heredando de una llamada `Exception`. Tomemos ventaja de esta disposición jerárquica a medida que reescribimos nuestro manejador de excepción `catch-all`.



Recuerde la jerarquía de excepciones de anterior.

Comprobamos el código:

```
>>> import sys
>>>
>>> try:
...     1/0
except:
    err = sys.exc_info()
    for e in err:
        print(e)

<class 'ZeroDivisionError'>
division by zero
<traceback object at 0xb5677c84>

>>> try:
...     1/0
except:
    err = sys.exc_info()[1]
    print(err)

division by zero
>>> try:
...     1/0
except:
    err = sys.exc_info()[0]
    print(err)

<class 'ZeroDivisionError'>
```

```

>>> try:
    1/0
except:
    err = sys.exc_info()[2]
    print(err)

<traceback object at 0xb5681504>
>>>

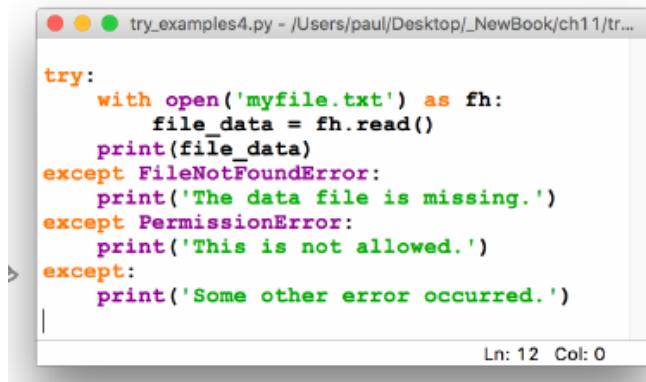
```

## ***The Catch-All Exception Handler, Revisited***

### ***El controlador de excepción Catch-All, Revisited***

Considere su código actual, que identifica explícitamente las dos excepciones que desea manejar (`FileNotFoundException` y `PermissionError`), así como proporciona un genérico catch-all, excepto suite (para manejar todo lo demás):

Este código funciona, pero en realidad no le dice mucho cuando se produce alguna excepción inesperada.



```

try:
    with open('myfile.txt') as fh:
        file_data = fh.read()
        print(file_data)
except FileNotFoundError:
    print('The data file is missing.')
except PermissionError:
    print('This is not allowed.')
except:
    print('Some other error occurred.')

```

Ln: 12 Col: 0

Tenga en cuenta cómo, al referirse a una excepción específica, hemos identificado la excepción por nombre después de la palabra clave `except`. Además de identificar excepciones específicas después de la excepción, también es posible identificar clases de excepciones utilizando cualquiera de los nombres de la jerarquía.

Por ejemplo, si sólo está interesado en saber que se ha producido un error aritmético (en oposición a un error de división por cero), puede especificar `excepto ArithmeticError`, que capturará un `FloatingPointError`, un `OverflowError` y un `ZeroDivisionError` si se producen. De manera similar, si especifica `excepto Excepción`, detectará cualquier error.

¿Pero cómo esto ayuda ... seguramente usted ya está cogiendo todos los errores con un "desnudo" excepto la declaración? Es verdad: lo eres. Pero puede extender la excepción `Exception` con la palabra clave `as`, que le permite asignar el objeto de excepción actual a una variable (err siendo un nombre muy popular en esta situación) y crear un mensaje de error más informativo. Echa un vistazo a otra versión del código, que utiliza `excepto Exception as:`

Recall that all the exceptions inherit from "Exception".

```

...
Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
        |    +-- FloatingPointError
        |    +-- OverflowError
        |    +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError

```

```

try:
    with open('myfile.txt') as fh:
        file_data = fh.read()
        print(file_data)
except FileNotFoundError:
    print('The data file is missing.')
except PermissionError:
    print('This is not allowed.')
except Exception as err:
    print('Some other error occurred:', str(err))

```

Ln: 12 Col: 0

Unlike the "bare" except catch-all shown above, this one arranges for the exception object to be assigned to the "err" variable.

The value of "err" is then used as part of the friendly message (as it's always a good idea to report all exceptions).

you are here ▶ 431

Traducir:

1. A diferencia de la "desnudo" excepto captura-todo mostrado arriba, éste organiza para que el objeto de excepción sea asignado a la variable "err".
2. El valor de "err" se utiliza entonces como parte del mensaje amistoso (ya que siempre es una buena idea reportar todas las excepciones).



## Test Drive -

Con esto: el último de los cambios realizados en el código code try / except, confirmemos que todo funciona de la manera esperada antes de volver a vsearch4web.py y aplicar lo que ahora se sabe sobre las excepciones a su aplicación web.

Comencemos por confirmar que el código muestra el mensaje correcto cuando falta el archivo:

```

Python 3.5.2 Shell
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
The data file is missing.
>>>

```

Ln: 7 Col: 4

← "myfile.txt" doesn't exist.

Si el archivo existe, pero no tiene permiso para acceder a él, se genera una excepción diferente:

```

Python 3.5.2 Shell
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
The data file is missing.
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
This is not allowed.
>>> |
Ln: 10 Col: 4

```

The file exists, but you can't read it.

Cualquier otra excepción es manejada por el catch-all, que muestra un mensaje amigable:

```

Python 3.5.2 Shell
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
The data file is missing.
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
This is not allowed.
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
Some other error occurred: [Errno 21] Is a directory: 'myfile.txt'
>>>
Ln: 23 Col: 4

```

Ha ocurrido alguna otra excepción. En este caso, lo que usted pensaba que era un archivo es de hecho una carpeta.

Finalmente, si todo está bien, el paquete try se ejecuta sin error y el contenido del archivo aparece en la pantalla:

```

Python 3.5.2 Shell
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
The data file is missing.
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
This is not allowed.
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
Some other error occurred: [Errno 21] Is a directory: 'myfile.txt'
>>>
===== RESTART: /Users/paul/Desktop/_NewBook/ch11/try_examples5.py ======
Empty (well... except for this line).

>>>
Ln: 27 Col: 4

```

¡Éxito! No se producen excepciones, por lo que la suite "try" se ejecuta hasta su finalización.

## Volver a nuestro código Webapp

Recordemos desde el comienzo de este capítulo que identificamos un problema con la llamada a `log_request` dentro de la función `do_search` de `vsearch4web.py`. En concreto, nos preocupa qué hacer cuando falla la llamada a `log_request`:

```

...
@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = search(letters, phrase)
    log_request(request, results)
    return render_template('results.html',
                          the_title=title,
                          the_phrase=phrase,
                          the_letters=letters,
                          the_results=results,
...

```

4. ¿Qué sucede si falla esta llamada?

Aqui tenemos al codigo completo:

```
from flask import Flask, render_template, request, escape
from vsearch import search4letters

from DBcm import UseDatabase

app = Flask(__name__)

app.config['dbconfig'] = {'host': '127.0.0.1',
                         'user': 'vsearch',
                         'password': 'vsearchpasswd',
                         'database': 'vsearchlogDB', }

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """insert into log
                  (phrase, letters, ip, browser_string, results)
                  values
                  (%s, %s, %s, %s, %s)"""
        cursor.execute(_SQL, (req.form['phrase'],
                             req.form['letters'],
                             req.remote_addr,
                             req.user_agent.browser,
                             res, ))


@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
```

```

"""Extract the posted data; perform the search; return results."""
phrase = request.form['phrase']
letters = request.form['letters']
title = 'Here are your results:'
results = str(search4letters(phrase, letters))

log_request(request, results)

return render_template('results.html',
                      the_title=title,
                      the_phrase=phrase,
                      the_letters=letters,
                      the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    """Display this webapp's HTML form."""
    return render_template('entry.html',
                          the_title='Welcome to search4letters on the web!')

@app.route('/viewlog')
def view_the_log() -> 'html':
    """Display the contents of the log file as a HTML table."""
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """select phrase, letters, ip, browser_string, results
                  from log"""
        cursor.execute(_SQL)
        contents = cursor.fetchall()
    titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
    return render_template('viewlog.html',
                          the_title='View Log',

```

```

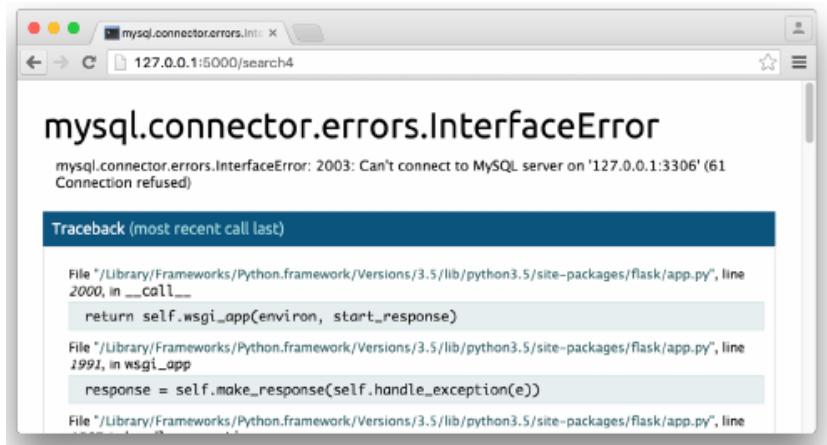
        the_row_titles=titles,
        the_data=contents,)

if __name__ == '__main__':
    app.run(debug=True)

```

Sobre la base de nuestras investigaciones, nos enteramos de que esta llamada podría fallar si la base de datos backend no está disponible, o si se produce algún otro error. Cuando se produce un error (de cualquier tipo), la aplicación web responde con una página de error poco amigable, que es probable que confunda (en lugar de iluminar) a los usuarios de su aplicación web:

Esto no es algo que quieras que vean tus usuarios de la aplicación web.



Aunque es importante para nosotros, el registro de cada solicitud web no es algo que nuestros usuarios de webapp realmente se preocupan. Todo lo que quieren ver es el resultado de su búsqueda. Por lo tanto, vamos a ajustar el código de la aplicación web para que se ocupe de los errores dentro de `log_request` mediante el manejo silencioso de cualquier excepción lanzada.

### *No hagas un sonido*

### **Manejo Silencioso de Excepciones**

*¿Seríamente? ¿Está planeando manejar las excepciones planteadas por "log\_request" en silencio? ¿No es eso simplemente otra variante de ignorar las excepciones y esperar que se vayan?*

**No: "silenciosamente" no significa "ignorar".**

Cuando sugerimos manejar excepciones silenciosamente en este contexto, nos referimos a manejar cualquier excepción planteada o lanzada de tal manera que los usuarios de su aplicación web no se den cuenta. Por el momento, sus usuarios lo notan, ya que la aplicación web se bloquea con una página de error confusa y -hagamos honesto-.

Los usuarios de su aplicación web no necesitan preocuparse de que `log_request` falla, pero lo hace. Así que vamos a ajustar su código para que las excepciones planteadas por `log_request` no se notan por sus usuarios (es decir, se silencian), pero son notados por usted.

## there are no Dumb Questions

**P:** ¿No todo esto `try/except` cosas simplemente hacer mi código más difícil de leer y entender?

**R:** Es cierto que el código de ejemplo de este capítulo comenzó como tres líneas fáciles de entender de código Python, y luego añadimos siete líneas de código que, a primera vista, no tienen nada que ver con lo que el primero Tres líneas de código que están haciendo. Sin embargo, es importante proteger el código que potencialmente puede generar una excepción, y `try / except` se considera generalmente como la mejor manera de hacerlo. Con el tiempo, su cerebro aprenderá a detectar las cosas importantes (el código en realidad haciendo el trabajo) que vive en la suite `try`, y filtrar las suites `except` que están ahí para manejar las excepciones. Al tratar de entender el código que usa `try / except`, siempre lee la suite `try` primero para aprender qué hace el código, luego mira las suites `except` si necesitas entender lo que sucede cuando las cosas salen mal.

## Sharpen your pencil .

Vamos a añadir un poco de `try / except` código a la invocación de `do_search` de la función `log_request`. Para mantener las cosas claras, vamos a agregar un manejador de excepción catch-all alrededor de la llamada a `log_request`, que, cuando se dispara, muestra un mensaje útil en la salida estándar (utilizando una llamada al BIF de impresión `print`). Al definir un manejador de excepciones catch-all, puede suprimir el comportamiento estándar de manejo de excepciones de su webapp, que actualmente muestra la página de error no amigable.

Aquí está el código de `log_request` como está escrito actualmente:

```

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    log_request(request, results) ←
    return render_template('results.html',
                          the_title=title,
                          the_phrase=phrase,
                          the_letters=letters,
                          the_results=results,)

```

Esta línea de código necesita ser protegida en caso de que falla (Generando un error de tiempo de ejecución).

En los espacios siguientes, proporcione el código que implementa un controlador de excepción catch-all alrededor de la llamada a `log_request`:

```

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))

.....
.....
```

No olvide llamar "log\_request" como parte del código que agrega.



```

.....
```

return render\_template('results.html',
 the\_title=title,
 the\_phrase=phrase,
 the\_letters=letters,
 the\_results=results,)



El plan era añadir algún código `try / except` a la invocación de `do_search` de la función `log_request`. Para mantener las cosas claras, decidimos agregar un controlador de excepción catch-all alrededor de la llamada a `log_request`, que, cuando se activa, muestra un mensaje útil en la salida estándar (mediante una llamada al BIF de impresión).

Aquí está el código de `log_request` como se escribe actualmente:

```

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    log_request(request, results)
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results)

```

En los espacios a continuación, debía proporcionar el código que implementa un manejador de excepción catch-all alrededor de la llamada a log\_request:

```

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))

    try:
        .....  

        log_request(request, results)  

    .....  

    except Exception as err:  

        .....  

        print('***** Logging failed with this error:', str(err))  

    .....  

    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results)

```

*This is the catch-all.*

*The call to "log\_request" is moved into the suite associated with a new "try" statement.*

*When a runtime error occurs, this message is displayed on screen for the admin only. Your user sees nothing.*

Traducción:

1. Esta es la captura de todos.
2. La llamada a "log\_request" se traslada al conjunto asociado con una nueva instrucción "try".
3. Cuando se produce un error de tiempo de ejecución, este mensaje se muestra en la pantalla sólo para el administrador. Su usuario no ve nada.



— (Extended) Test Drive, 1 of 3 —

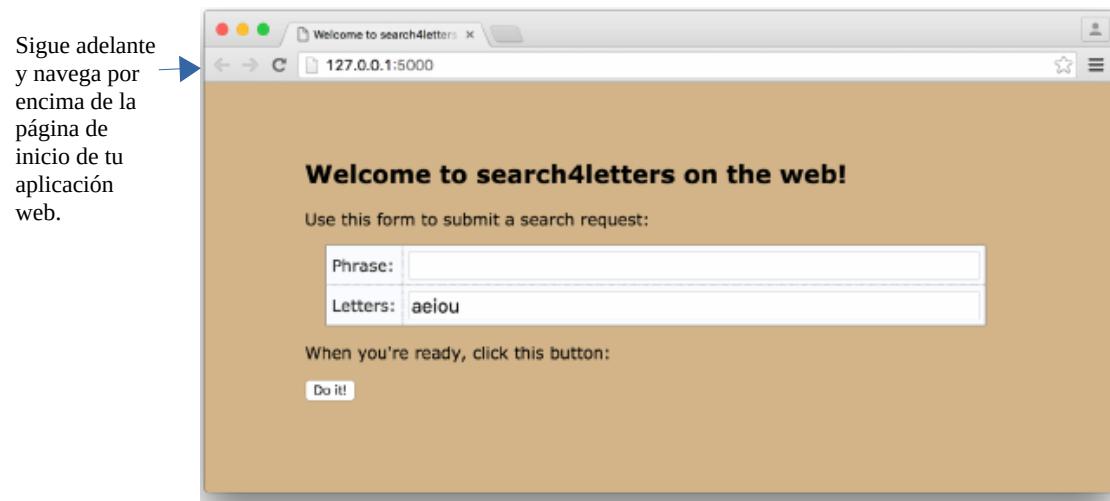
Con el código de gestión de excepciones de catch-all añadido a vsearch4web.py, vamos a tomar su webapp para un giro extendido (en las siguientes páginas) para ver la diferencia que este nuevo código hace. Anteriormente, cuando algo salía mal, su usuario recibió una página de error poco amistosa. Ahora, sin embargo, el error es manejado "silenciosamente" por el código catch-all. Si todavía no lo ha hecho, ejecute vsearch4web.py y, a continuación, utilice cualquier navegador para navegar por la página principal o home de su webapp:

```
$ python3 vsearch4web.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with fsevents reloader
* Debugger is active!
* Debugger pin code: 184-855-980
```

Your webapp is up and running,  
waiting to hear from a browser...

traducimos;

Tu aplicación web está lista y en funcionamiento esperando para escuchar de un navegador ...



En el terminal que está ejecutando su código, debería ver algo como esto:

```
...
* Debugger pin code: 184-855-980
127.0.0.1 - - [14/Jul/2016 10:54:31] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Jul/2016 10:54:31] "GET /static/hf.css HTTP/1.1" 200 -
127.0.0.1 - - [14/Jul/2016 10:54:32] "GET /favicon.ico HTTP/1.1" 404 -
```

These 200s confirm that your webapp is up and running (and serving up its home page). All is good at this point.

BTW: Don't worry about this 404...we haven't defined a "favicon.ico" file for our webapp (so it gets reported as not found when your browser asks for it).

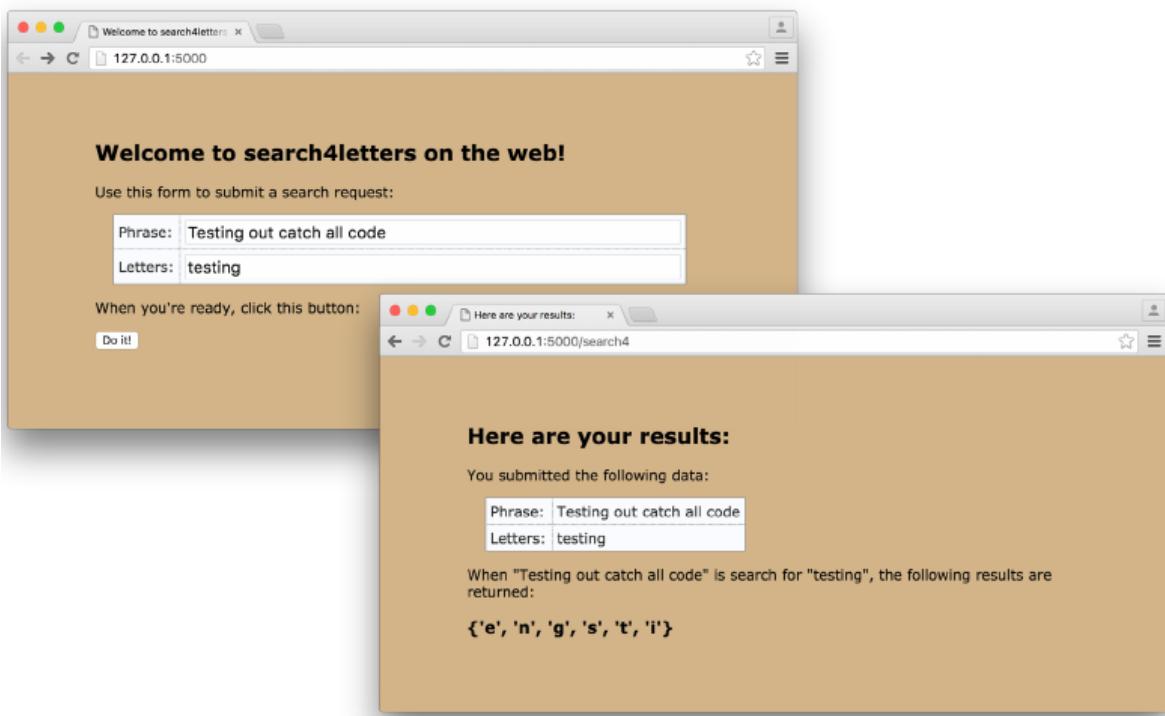
Traducimos:

1. Estos 200 confirmar que su aplicación web está en marcha (y servir su página de inicio). Todo es bueno en este punto.

- BTW: No te preocupes por este 404 ... no hemos definido un archivo "favicon.ico" para nuestra webapp (por lo que se informa como no se encuentra cuando su navegador lo pide).

## - (Extended) Test Drive, 2 of 3 -

Con el fin de simular un error, hemos desactivado nuestra base de datos backend, lo que debería dar lugar a un error que se produce cada vez que la webapp trata de interactuar con la base de datos. Como nuestro código captura silenciosamente todos los errores generados por log\_request, el usuario de la aplicación web no es consciente de que el registro no se ha producido. El código catch-all se ha organizado para generar un mensaje en la pantalla que describe el problema. Cuando inserte una frase y haga clic en el botón "¡hazlo!", La aplicación web muestra los resultados de tu búsqueda en el navegador, mientras que la pantalla del terminal de la aplicación web muestra el mensaje de error "silenciado". Tenga en cuenta que, a pesar del error de tiempo de ejecución, la aplicación web continúa ejecutando y atendiendo con éxito la llamada a /search:



Welcome to search4letters on the web!

Use this form to submit a search request:

Phrase:	Testing out catch all code
Letters:	testing

When you're ready, click this button:

Do it!

Here are your results:

You submitted the following data:

Phrase:	Testing out catch all code
Letters:	testing

When "Testing out catch all code" is search for "testing", the following results are returned:

```
{'e', 'n', 'g', 's', 't', 'i'}
```

```
127.0.0.1 - - [14/Jul/2016 10:54:32] "GET /favicon.ico HTTP/1.1" 404 -
***** Logging failed with this error: 2003: Can't connect to MySQL server on '127.0.0.1:3306'
(61 Connection refused)
127.0.0.1 - - [14/Jul/2016 10:55:55] "POST /search4 HTTP/1.1" 200 -
```

This message is generated by your catch-all exception-handling code. The webapp user doesn't see it.

Even though an error occurred, the webapp didn't crash. In other words, the search worked (but the webapp user isn't aware that the logging failed).

Traducimos:

1. Este mensaje es generado por su código de manejo de excepciones. El usuario de la aplicación web no lo ve.
2. Aunque se produjo un error. La aplicación web no se bloqueó. En otras palabras, la búsqueda funcionó (pero el usuario no es consciente de que el registro falló).



## —(Extended) Test Drive, 3 of 3 —

De hecho, no importa qué error ocurre cuando se ejecuta `log_request`, el código catch-all lo controla.

Reiniciamos nuestra base de datos backend, luego intentamos conectarnos con un nombre de usuario incorrecto. Puede aumentar este error cambiando el diccionario `dbconfig` en `vsearch4web.py` para utilizar `vsearchwrong` como valor para el `user`:

```
...  
app.config['dbconfig'] = {'host': '127.0.0.1',  
    'user': 'vsearchwrong', ←  
    'password': 'vsearchpasswd',  
    'database': 'vsearchlogDB', }  
....
```

When your webapp reloads and you perform a search, you'll see a message like this in your terminal:

```
***** Logging failed with this error: 1045 (28000): Access denied for user 'vsearchwrong'@  
'localhost' (using password: YES)
```

traducción:

Cuando tu webapp se recargue y realices una búsqueda, verás un mensaje como este en tu terminal

Cambie el valor para el usuario de nuevo a `vsearch` y, a continuación, intente acceder a una tabla inexistente cambiando el nombre de la tabla en la consulta SQL utilizada en la función `log_request` para que sea `logwrong` (en lugar de `log`):

```
def log_request(req: 'flask_request', res: str) -> None:  
    with UseDatabase(app.config['dbconfig']) as cursor:  
        _SQL = """insert into logwrong ←  
            (phrase, letters, ip, browser_string, results)  
            values  
            (%s, %s, %s, %s, %s)"""  
....
```

When your webapp reloads and you perform a search, you'll see a message like this in your terminal:

```
***** Logging failed with this error: 1146 (42S02): Table 'vsearchlogdb.logwrong' doesn't exist
```

Traducimos:

Cuando tu webapp se recargue y realices una búsqueda, verás un mensaje como este en tu terminal

Cambiar el nombre de la tabla de nuevo a `log` y, a continuación, como un ejemplo final, vamos a agregar una instrucción `raise` a la función `log_request` (justo antes de la instrucción `with`), que genera una excepción personalizada:

```
def log_request(req: 'flask_request', res: str) -> None:  
    raise Exception("Something awful just happened.") ←  
    with UseDatabase(app.config['dbconfig']) as cursor:  
        ...
```

When your webapp reloads one last time, and you perform one last search, you'll see the following message in your terminal:

```
...  
***** Logging failed with this error: Something awful just happened. ←
```

Traducimos: Cuando la aplicación web se vuelve a cargar una última vez y realice una última búsqueda, verá el siguiente mensaje en su terminal

### ***Manejo de más errores***

#### ***Manejo de otros errores de base de datos***

La función `log_request` hace uso del gestor de contexto `UseDatabase` (como proporcionado por el módulo `DBcm`). Ahora que ha protegido la llamada a `log_request`, puede estar tranquilo, sabiendo que cualquier problema relacionado con problemas con la base de datos será capturado (y manejado) por su código de captura de excepción.

Sin embargo, la función `log_request` no es el único lugar donde interactúa su aplicación web con la base de datos. La función `view_the_log` captura los datos de registro de la base de datos antes de mostrarlos en la pantalla.

Recuerde el código de la función `view_the_log`:

Todo este código también necesita ser protegido.

```
...  
@app.route('/viewlog')  
@check_logged_in  
def view_the_log() -> 'html':  
    with UseDatabase(app.config['dbconfig']) as cursor:  
        _SQL = """select phrase, letters, ip, browser_string, results  
                from log"""  
        cursor.execute(_SQL)  
        contents = cursor.fetchall()  
    titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')  
    return render_template('viewlog.html',  
                           the_title='View Log',  
                           the_row_titles=titles,  
                           the_data=contents)  
...
```

Este código también puede fallar, ya que interactúa con la base de datos backend. Sin embargo, a diferencia de `log_request`, la función `view_the_log` no se llama desde el código en `vsearch4web.py`; Es invocado por Flask en su nombre. Esto significa que no puede escribir código para proteger la invocación de `view_the_log`, ya que es el marco de Flask que llama a la función, no usted.

Si no puede proteger la invocación de `view_the_log`, lo siguiente mejor es proteger el código en su suite, específicamente el uso del gestor de contexto `UseDatabase`. Antes de considerar cómo hacerlo, consideremos lo que puede salir mal:

- Es posible que la base de datos de back-end no esté disponible.
- Es posible que no pueda iniciar sesión en una base de datos de trabajo.
- Despues de un inicio de sesión exitoso, la consulta de la base de datos podría fallar.
- Puede suceder algo más (inesperado).

Esta lista de problemas es similar a los que tenía que preocuparse con `log_request`.

## ***Does “More Errors” Mean “More excepts”? ¿"Más Errores" significa "más excepts"?***

Sabiendo lo que ahora sabemos acerca de `try / except`, podríamos agregar algún código a la función `view_the_log` para proteger el uso del gestor de contexto `UseDatabase`:

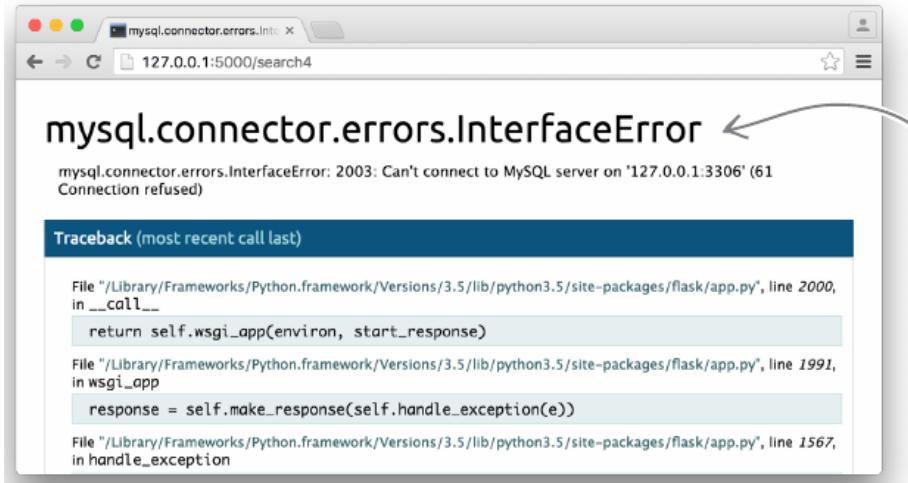
```
...  
@app.route('/viewlog')  
@check_logged_in  
def view_the_log() -> 'html':  
    try:  
        with UseDatabase(app.config['dbconfig']) as cursor:  
            ...  
    except Exception as err:  
        print('Something went wrong:', str(err))  
  
Another catch-all exception handler  
  
The rest of the function's code goes here.  
→ ↙
```

traducción:

1. Otro manejador de excepciones catch-all
2. El resto del código de la función va aquí.

Esta estrategia catch-all sin duda funciona (después de todo, eso es lo que utilizó con `log_request`). Sin embargo, las cosas pueden complicarse si decide hacer algo más que implementar un manejador de excepción catch-all. ¿Qué pasa si decide que necesita

reaccionar ante un error de base de datos específico, como "Base de datos no encontrada"? Recuerde desde el principio de este capítulo que MySQL informa una excepción `InterfaceError` cuando esto sucede:



Esta excepción se produce cuando su código no puede encontrar la base de datos de backend.

Podría agregar una instrucción `except` que apunte a la excepción `InterfaceError`, pero para hacer esto su código también tiene que importar el módulo `mysql.connector`, que define esta excepción en particular.

En la cara de las cosas, esto no parece ser un gran problema. Pero es.

### Mira tus importaciones

### Evite el código firmemente acoplado

Supongamos que ha decidido crear una sentencia `except` que proteja contra la base de datos backend que no está disponible. Podrías ajustar el código en `view_the_log` para que parezca algo así:

```
...
@app.route('/viewlog')
@check_logged_in
def view_the_log() -> 'html':
    try:
        with UseDatabase(app.config['dbconfig']) as cursor:
            ...
    except mysql.connector.errors.InterfaceError as err:
        print('Is your database switched on? Error:', str(err))
    except Exception as err:
        print('Something went wrong:', str(err))
    ...
```

The rest of  
the function's  
code still goes  
in here.

Agregar otra instrucción  
"except" para manejar una  
excepción específica

Traducción: El resto del código de la función sigue entrando aquí.

Si también recuerda agregar `import mysql.connector` a la parte superior de su código, esta instrucción adicional `except` funciona. Cuando no se puede encontrar su base de datos de backend, este código adicional le permite recordarle que su aplicación web está encendida.

Este nuevo código funciona, y usted puede ver lo que está pasando aquí ... ¿qué no es como?

El problema con abordar el problema de esta manera es que el código en `vsearch4web.py` ahora está muy estrechamente acoplado a la base de datos MySQL, y específicamente al uso del módulo MySQL Connector. Antes de agregar esta segunda sentencia `except`, el código `vsearch4web.py` interaccionó con su base de datos backend a través del módulo `DBcm` (desarrollado anteriormente en este libro). Específicamente, el gestor de contexto `UseDatabase` proporciona una abstracción conveniente que desacopla el código en `vsearch4web.py` de la base de datos backend. Si en algún momento en el futuro necesitas reemplazar MySQL con PostgreSQL, los únicos cambios que necesitarías hacer serían el módulo `DBcm`, no todo el código que usa `UseDatabase`. Sin embargo, al crear código como el que se muestra arriba, se enlaza firmemente (es decir, se acopla) el código de su webapp a la base de datos de MySQL debido a esa instrucción `import mysql.connector`, además de su nueva referencia a `mysql.connector.errors .InterfaceError`.

Si necesita escribir código que se acopla firmemente a su base de datos backend, considere siempre poner ese código en el módulo `DBcm`. De esta manera, su aplicación web se puede escribir para utilizar la interfaz genérica proporcionada por `DBcm`, a diferencia de una interfaz específica que se dirige (y te bloquea) en una base de datos de backend específico.

Ahora consideremos qué mover el código anterior excepto en `DBcm` hace para nuestra aplicación web.

## ***The DBcm Module, Revisited***

### ***El módulo DBcm, revisitado***

La última vez que consultó `DBcm` en el capítulo 9, cuando creó ese módulo con el fin de proporcionar un gancho en la declaración con cuando se trabaja con una base de datos MySQL. En ese entonces, evitamos cualquier discusión sobre el manejo de errores (ignorando convenientemente el problema). Ahora que has visto lo que hace la función `sys.exc_info`, deberías tener una mejor idea de lo que significan los argumentos para el método `__exit__` de `UseDatabase`:

```

import mysql.connector

Este es el
código del
gestor de
contexto en
"DBcm.py"
    class UseDatabase:

        def __init__(self, config: dict) -> None:
            self.configuration = config

        def __enter__(self) -> 'cursor':
            self.conn = mysql.connector.connect(**self.configuration)
            self.cursor = self.conn.cursor()
            return self.cursor

        def __exit__(self, exc_type, exc_value, exc_trace) -> None:
            self.conn.commit()
            self.cursor.close()
            self.conn.close()

```



Ahora que ha visto "exc\_info", debería quedar claro a qué se refieren estos argumentos de método: datos de excepción.

Recuerde que `UseDatabase` implementa tres métodos:

- `__init__` proporciona una oportunidad de configuración antes de ejecutar,
- `__enter__` se ejecuta cuando empieza la sentencia `with`, y
- `__exit__` está garantizado para ejecutarse cuando termine la suite de `with`.

Al menos, ese es el comportamiento esperado siempre que todo vaya al plan. Cuando las cosas van mal, este comportamiento cambia.

Por ejemplo, si se produce una excepción mientras `__enter__` se está ejecutando, la instrucción `with` termina y se cancela cualquier procesamiento posterior de `__exit__`. Esto tiene sentido: si `__enter__` se encuentra en problemas, `__exit__` ya no puede suponer que el contexto de ejecución se inicializa y configura correctamente (por lo que es prudente no ejecutar el código del método `__exit__`).

El gran problema con el código del método `__enter__` es que la base de datos de backend no esté disponible, así que tomemos algún tiempo para ajustar `__enter__` para esta posibilidad, generando una excepción personalizada cuando no se pueda establecer la conexión de la base de datos. Una vez que hayamos hecho esto, ajustaremos `view_the_log` para comprobar nuestra excepción personalizada en lugar de la base de datos específica `mysql.connector.errors.InterfaceError`.

### *Tu propia excepción*

#### *Creación de excepciones personalizadas*

Crear sus propias excepciones personalizadas no podría ser más fácil: decidir sobre un nombre apropiado, y luego definir una clase vacía que hereda de la clase de `Exception`

incorporada de Python. Una vez que haya definido una excepción personalizada, se puede generar con la palabra clave `raise`. Y una vez que se levanta una excepción, es capturado (y tratado) por `try / except`.

Un rápido viaje a la indicación `>>>` de IDLE demuestra excepciones personalizadas en acción. En este ejemplo, estamos creando una excepción personalizada llamada `ConnectionError`, que luego subimos (con `raise`), antes de capturar con `try / except`. Lea las anotaciones en orden numerado y (si lo está siguiendo) ingrese el código que hemos escrito en el prompt `>>>`:

```

Python 3.5.2 Shell
>>>
>>>
>>> class ConnectionError(Exception):
>>>     pass
>>>
>>>
>>> raise ConnectionError('Cannot connect... is it time to panic?')
Traceback (most recent call last):
  File "<pyshell#74>", line 1, in <module>
    raise ConnectionError('Cannot connect... is it time to panic?')
ConnectionError: Cannot connect... is it time to panic?
>>>
>>>
>>> try:
>>>     raise ConnectionError('Whoops!')
>>> except ConnectionError as err:
>>>     print('Got:', str(err))
Got: Whoops!
>>>
>>> |

```

1. Create a new class called "ConnectionError" that inherits from the "Exception" class.

2. "pass" is Python's empty statement that creates the empty class.

3. Raising our new exception (with "raise") results in a traceback message.

4. Catch the "ConnectionError" exception using "try/except".

5. The "ConnectionError" was caught, allowing us to customize the error message.

Ln: 148 Col: 4

traducir:

1. Cree una nueva clase llamada "ConnectionError" que hereda de la clase "Exception".
2. "pase" es vacío de Python que crea la clase vacía.
3. Aumentar nuestra nueva excepción (con "raise") da como resultado un mensaje de rastreo o traceback.
4. Coja la excepción "ConnectionError" utilizando "try / except".
5. El "ConnectionError" fue capturado, lo que nos permite personalizar el mensaje de error.

*La clase vacía no está muy vacía ...*

Al describir la clase `ConnectionError` como "empty", le contamos una pequeña mentira. Por supuesto, el uso de `pass` asegura que no hay ningún código nuevo asociado con la clase `ConnectionError`, pero el hecho de que `ConnectionError` hereda de la clase `Exception` integrada de Python significa que todos los atributos y comportamientos de `Exception` también están disponibles en `ConnectionError` (Pero vacío). Esto explica por qué `ConnectionError` funciona igual que lo esperaría con `raise` y `try / except`.



Vamos a ajustar el módulo de `Dbcm` para crear un `ConnectionError` personalizado cuando una conexión a la base de datos backend falla.

**1** Aquí está el código actual de `Dbcm.py`. En los espacios proporcionados, agregue el código necesario para elevar un `ConnectionError`.

Define your custom exception.

```
import mysql.connector
```

.....

```
class UseDatabase:
```

def \_\_init\_\_(self, config: dict) -> None:  
 self.configuration = config

def \_\_enter\_\_(self) -> 'cursor':  
 .....  
 self.conn = mysql.connector.connect(\*\*self.configuration)  
 self.cursor = self.conn.cursor()  
 return self.cursor

Add code to "raise" a "ConnectionError".

.....

```
    def __exit__(self, exc_type, exc_value, exc_trace) -> None:  
        self.conn.commit()  
        self.cursor.close()  
        self.conn.close()
```

## Traducción:

1. Defina su excepción personalizada.
  2. Agregue código para "elevar" un "ConnnectionError".

2. Con el código del módulo `DBcm` modificado, utilice su lápiz para detallar los cambios que haría en este código de `vsearch4web.py` para aprovechar la excepción `ConnectionError` recién definida:

Use your pencil to show the changes you'd make to this code now that the "ConnectionError" exception exists.

```

from DBcm import UseDatabase
import mysql.connector
...
    the_row_titles,
    the_data=contents,)
except mysql.connector.errors.InterfaceError as err:
    print('Is your database switched on? Error:', str(err))
except Exception as err:
    print('Something went wrong:', str(err))
return 'Error'

```

Traducción:

Utilice su lápiz para mostrar los cambios que haría a este código ahora que existe la excepción "ConnectionError".

## *Levantando un error de conexión*



1. Debía ajustar el módulo de DBcm para crear un ConnectionError personalizado siempre que fallara una conexión a la base de datos backend. Debía ajustar el código actual a DBcm.py para agregar el código necesario para generar o raise un ConnectionError.

Defina la excepción personalizada como una clase "vacía" que hereda de "Exception".

```

import mysql.connector
class ConnectionError(Exception):
    pass
class UseDatabase:
    def __init__(self, config: dict) -> None:
        self.configuration = config
    def __enter__(self) -> 'cursor':
        try:
            self.conn = mysql.connector.connect(**self.configuration)
            self.cursor = self.conn.cursor()
            return self.cursor
        except mysql.connector.errors.InterfaceError as err:
            raise ConnectionError(err)
    def __exit__(self, exc_type, exc_value, exc_trace) -> None:
        self.conn.commit()
        self.cursor.close()
        self.conn.close()

```

Una nueva construcción "try / except" protege el código de conexión a la base de datos.

Within the "DBcm.py" code, refer to the backend database-specific exceptions by their full name.

Raise the custom exception.

Traducción:

1. Dentro del código "DBcm.py", refiérase a las excepciones específicas de la base de datos backend por su nombre completo.
2. Elevar o generar la excepción personalizada.

Con el código del módulo DBcm modificado, debías detallar los cambios que realizarías en este código desde vsearch4web.py para aprovechar la excepción ConnectionError recién definida:

Ya no es necesario importar "mysql.connector" (como "DBcm" lo hace para usted).

```
from DBcm import UseDatabase, ConnectionError
import mysql.connector
...
ConnectionError
except mysql.connector.errors.InterfaceError as err:
    print('Is your database switched on? Error:', str(err))
except Exception as err:
    print('Something went wrong:', str(err))
return 'Error'
```

Be sure to import the "ConnectionError" exception from "DBcm".

Cambie la primera instrucción "except" para buscar un "ConnectionError" en lugar de un "InterfaceError".

Traducir: Asegúrese de importar la excepción "ConnectionError" de "Dbcm".

## Test Drive -

Veamos qué diferencia hace este nuevo código. Recuerde que ha movido el código de manejo de excepciones específico de MySQL de vsearch4web.py en DBcm.py (y lo reemplazó por un código que busca la excepción ConnectionError personalizada). ¿Esto ha hecho alguna diferencia?

Estos son los mensajes que la versión anterior de vsearch4web.py generó cuando no se pudo encontrar la base de datos backend:

```
...
Is your database switched on? Error: 2003: Can't connect to MySQL server on '127.0.0.1:3306'
(61 Connection refused)
127.0.0.1 - - [16/Jul/2016 21:21:51] "GET /viewlog HTTP/1.1" 200 -
```

Y aquí están los mensajes que la versión más reciente de vsearch4web.py genera cada vez que no se encuentra la base de datos de backend:

```
...
Is your database switched on? Error: 2003: Can't connect to MySQL server on '127.0.0.1:3306'
(61 Connection refused)
127.0.0.1 - - [16/Jul/2016 21:22:58] "GET /viewlog HTTP/1.1" 200 -
```

*Estás tratando de engañarme, ¿no? ¡Estos mensajes de error son iguales!*

**Sí. A la vista de las cosas, éstas son las mismas.**

Sin embargo, aunque la salida de la versión actual y anterior de vsearch4web.py aparece idéntica, detrás de las escenas las cosas son muy diferentes.

Si decide cambiar la base de datos backend de MySQL a PostgreSQL, ya no tendrá que preocuparse por cambiar el código de vsearch4web.py, ya que todo su código específico de base de datos reside en DBcm.py. Siempre y cuando los cambios que realice en DBcm.py mantengan la misma interfaz que las versiones anteriores del módulo, puede cambiar las bases de datos SQL con la frecuencia que desee. Esto puede no parecer una gran cosa ahora, pero si vsearch4web.py crece a cientos, miles o decenas de miles de líneas de código, es realmente un gran problema.

### **Más problemas de base de datos**

**¿Qué más puede ir mal con "Dbcm"?**

Incluso si su base de datos de backend está funcionando, las cosas pueden ir mal.

Por ejemplo, las credenciales utilizadas para tener acceso a la base de datos pueden ser incorrectas. Si lo son, el método `__enter__` fallará de nuevo, esta vez con un `mysql.connector.errors.ProgrammingError`.

O bien, el conjunto de código asociado con su gestor de contexto `UseDatabase` puede generar una excepción, ya que nunca garantiza que se ejecute correctamente. A `mysql.connector.errors.ProgrammingError` también se plantea cada vez que su consulta de base de datos (el SQL que está ejecutando) contiene un error.

El mensaje de error asociado con un error de consulta SQL es diferente al mensaje asociado con el error credenciales, pero la excepción generada es la misma: `mysql.connector.errors.ProgrammingError`. A diferencia de los errores de credenciales, los errores en los resultados de SQL generan una excepción mientras se ejecuta la instrucción `with`. Esto significa que tendrá que considerar la protección contra esta excepción en más de un lugar. La pregunta es: ¿dónde?

Para responder a esta pregunta, echemos un vistazo al código de `Dbcm`:

```

import mysql.connector

class ConnectionError(Exception):
    pass

class UseDatabase:
    def __init__(self, config: dict):
        self.configuration = config

    def __enter__(self) -> 'cursor':
        try:
            self.conn = mysql.connector.connect(**self.configuration)
            self.cursor = self.conn.cursor()
            return self.cursor
        except mysql.connector.errors.InterfaceError as err:
            raise ConnectionError(err)

    def __exit__(self, exc_type, exc_value, exc_traceback):
        self.conn.commit()
        self.cursor.close()
        self.conn.close()

```

Pero ¿qué pasa con las excepciones que se producen dentro de la suite "with"? Estos suceden \* después\* de que el método `__enter__` finaliza pero \* antes\* de que se inicie el método `__exit__`.

Es posible que se sienta tentado a sugerir que las excepciones planteadas dentro de `with` la suite se deben manejar con una sentencia `try / except` dentro de la `with`, pero tal estrategia te devuelve a escribir código firmemente acoplado. Pero considere esto: cuando una excepción se plantea dentro de la suite y no se captura, la instrucción `with` organiza para pasar detalles de la excepción no captada al método `__exit__` del gestor de contexto, donde tiene la opción de hacer algo al respecto.

## ***Creating More Custom Exceptions***

### ***Creación de excepciones personalizadas***

Extendamos `DBcm.py` para informar dos excepciones adicionales personalizadas.

El primero se llama `CredentialsError` y se activa cuando se produce un `ProgrammingError` dentro del método `__enter__`. El segundo se llama `SQLLError` y se eleva cuando se informa de un `ProgrammingError` al método `__exit__`.

La definición de estas nuevas excepciones es fácil: añada dos nuevas clases de excepción vacías a la parte superior de `DBcm.py`:

```

import mysql.connector

class ConnectionError(Exception):
    pass

class CredentialsError(Exception):
    pass

class SQLLError(Exception):
    pass

class UseDatabase:
    def __init__(self, configuration: dict):
        self.config = configuration
        ...

```

Un `CredentialsError` puede ocurrir durante `__enter__`, así que vamos a ajustar el código de ese método para reflejar esto. Recuerde que un nombre de usuario o contraseña MySQL incorrecto resulta en un error de programación que se plantea o eleva:

```
...  
try:  
    self.conn = mysql.connector.connect(**self.config)  
    self.cursor = self.conn.cursor()  
    return self.cursor  
except mysql.connector.errors.InterfaceError as err:  
    raise ConnectionError(err)  
except mysql.connector.errors.ProgrammingError as err:  
    raise CredentialsError(err)  
  
→ def __exit__(self, exc_type, exc_value, exc_traceback):  
    self.conn.commit()  
    self.cursor.close()  
    self.conn.close()
```

Agregue este código al método `"__enter__"` para tratar cualquier problema de inicio de sesión.

Estos cambios de código ajustan `DBcm.py` para aumentar una excepción de `CredentialsError` cuando proporciona un nombre de usuario o una contraseña incorrecta del código a su base de datos backend (MySQL). Ajustar el código de `vsearch4web.py` es su próxima tarea.

### ***Cogiendo más excepciones***

#### ***¿Las credenciales de la base de datos son correctas?***

Con estos últimos cambios realizados en `DBcm.py`, vamos ahora a ajustar el código en `vsearch4web.py`, prestando especial atención a la función `view_the_log`. Sin embargo, antes de hacer cualquier otra cosa, agregue `CredentialsError` a la lista de importaciones de `DBcm` en la parte superior de su código `vsearch4web.py`:

```
...  
from DBcm import UseDatabase, ConnectionError, CredentialsError  
...  
Asegúrese de importar su nueva  
excepción.
```

Con la línea de importación enmendada, debe agregar una nueva suite `except` a la función `view_the_log`. Al igual que cuando se agregó soporte para `ConnectionError`, se trata de una edición directa:

```

@app.route('/viewlog')
@check_logged_in
def view_the_log() -> 'html':
    try:
        with UseDatabase(app.config['dbconfig']) as cursor:
            _SQL = """select phrase, letters, ip, browser_string, results
                      from log"""
            cursor.execute(_SQL)
            contents = cursor.fetchall()
            titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
            return render_template('viewlog.html',
                                   the_title='View Log',
                                   the_row_titles=titles,
                                   the_data=contents,)
    except ConnectionError as err:
        print('Is your database switched on? Error:', str(err))
    except CredentialsError as err:
        print('User-id/Password issues. Error:', str(err)) ←
    except Exception as err:
        print('Something went wrong:', str(err))
    return 'Error'

```

Agregue este código a "view\_the\_log" para capturar cuando su código usa el nombre de usuario o contraseña incorrecto con MySQL.

Realmente no hay nada nuevo aquí, ya que todo lo que estás haciendo es repetir lo que hiciste para `ConnectionError`. De hecho, si intenta conectarse a su base de datos backend con un nombre de usuario incorrecto (o contraseña), su aplicación web muestra ahora un mensaje apropiado, como esto:

```

...
User-id/Password issues. Error: 1045 (28000): Access denied for user 'vsearcherror'@'localhost'
(using password: YES)
127.0.0.1 - - [25/Jul/2016 16:29:37] "GET /viewlog HTTP/1.1" 200 -

```



Ahora que su código sabe todo acerca de "CredentialsError", genera un mensaje de error específico de la excepción.

### **Manejar error SQL es diferente**

Tanto `ConnectionError` como `CredentialsError` se generan debido a problemas con la ejecución del código del método `__enter__`. Cuando se produce alguna excepción, no se ejecuta el conjunto de la sentencia correspondiente statement with.

Si todo va bien, tu suite se ejecuta como de costumbre.

Recuerde esto con la sentencia de la función `log_request`, que utiliza el gestor de contexto `UseDatabase` (proporcionado por `DBcm`) para insertar datos en la base de datos backend:

```

with UseDatabase(app.config['dbconfig']) as cursor:
    _SQL = """insert into log
              (phrase, letters, ip, browser_string, results)
              values
              (%s, %s, %s, %s, %s)"""
    cursor.execute(_SQL, (req.form['phrase'],
                         req.form['letters'],
                         req.remote_addr,
                         req.user_agent.browser,
                         res, ))

```

Tenemos que preocuparnos por lo que sucede si algo va mal con este código (es decir, el código dentro de la suite "with").

Si (por alguna razón) su consulta SQL contiene un error, el módulo MySQL Connector genera un ProgrammingError, al igual que el generado durante el método `__enter__` del gestor de contexto. Sin embargo, como esta excepción se produce dentro de su gestor de contexto (es decir, dentro de la instrucción `with`) y no se captura allí, la excepción se devuelve al método `__exit__` como tres argumentos: el tipo de la excepción, el valor de la excepción y el rastreo asociado con la excepción.

Si echa un vistazo rápido al código existente de DBcm para `__exit__`, verá que los tres argumentos están listos y en espera de ser utilizados:

```
def __exit__(self, exc_type, exc_value, exc_traceback):
    self.conn.commit()
    self.cursor.close()
    self.conn.close()
```

Los tres argumentos de excepción están listos  
para su uso.

Cuando se genera una excepción dentro de el suite `with` y no se captura, el gestor de contexto termina el código con el suite, y salta al método `__exit__`, que luego se ejecuta. Sabiendo esto, puede escribir código que comprueba las excepciones de interés para su aplicación. Sin embargo, si no se produce ninguna excepción, los tres argumentos (`exc_type`, `exc_value` y `exc_traceback`) están todos establecidos en `None`. De lo contrario, se llenan con detalles de la excepción generada o `raised`.

Explotemos este comportamiento para generar un `SQLLError` siempre que algo va mal dentro del gestor de contexto `UseDatabase` de la suite `with`.

*"None" es el valor nulo de Python.*

**`exc_type` `exc_value` `exc_traceback`**

### **Tenga cuidado con el posicionamiento de código**

Para comprobar si se ha producido una excepción no detectada dentro de la sentencia de su código, compruebe el argumento `exc_type` en el método `__exit__` dentro de `__exit__`, teniendo cuidado de considerar exactamente dónde agrega su nuevo código.

*Usted no va a decirme que hace una diferencia donde pongo mi código de comprobación "exc\_type", ¿verdad?*

## Sí, hace una diferencia.

Para entender por qué, considere que el método `__exit__` de su gestor de contexto proporciona un lugar donde se puede colocar el código que se garantiza que se ejecute una vez que finalice su suite `with`. Ese comportamiento es parte del protocolo de gestor de contexto, después de todo.

Este comportamiento debe mantenerse incluso cuando se generan excepciones en el gestor de contexto de suite `with`. Lo que significa que si planea agregar código al método `__exit__`, lo mejor es ponerlo después de cualquier código existente en `__exit__`, de esa manera usted todavía garantiza que el código existente del método se ejecute (y preserve la semántica del protocolo del gestor de contexto).

Echemos un vistazo al código existente en el método `__exit__` a la luz de esta discusión sobre la colocación de código. Tenga en cuenta que cualquier código que agregue necesita generar una excepción `SQLLError` si `exc_type` indica que se ha producido un error de programación `ProgrammingError`:

```
def __exit__(self, exc_type, exc_value, exc_traceback):
    self.conn.commit()
    self.cursor.close()
    self.conn.close()
```

Traducimos:

1. Si agrega código aquí y ese código genera una excepción, las tres líneas de código existentes no se ejecutarán.
2. Agregando código \*después de\* las tres líneas existentes de código asegura que `"__exit__"` hace su cosa \*antes de\* cualquier pasado-en excepto se trata.

## Raising an `SQLLError`

### Levantamiento de un `SQLLError`

En esta etapa, ya ha agregado la clase de excepción `SQLLError` a la parte superior del archivo `DBcm.py`:

```
import mysql.connector

class ConnectionError(Exception):
    pass

class CredentialsError(Exception):
    pass

class SQLLError(Exception):
    pass
```

Aquí es donde agregó en la excepción "SQLLError".

```
class UseDatabase:
    def __init__(self, config: dict):
        self.configuration = config
    ...
```

Con la clase de excepción `SQLError` definida, todo lo que necesita hacer ahora es agregar algún código al método `__exit__` para comprobar si `exc_type` es la excepción que le interesa y, si es así, eleve o `raise` un `SQLError`. Esto es tan sencillo que estamos resistiendo el impulso habitual de Head First de convertir la creación del código requerido en un ejercicio, ya que nadie quiere insultar a la inteligencia de nadie en esta etapa de este libro. Así que aquí está el código que necesita para añadir al método `__exit__`:

```
def __exit__(self, exc_type, exc_value, exc_traceback):
    self.conn.commit()
    self.cursor.close()
    self.conn.close()
    if exc_type is mysql.connector.errors.ProgrammingError:
        raise SQLError(exc_value)
```

Si se produce un error  
"ProgrammingError", eleve un  
"SQLError".

Si quieres ser más seguro y *hacer algo sensato con cualquier otra excepción inesperada enviada a \_\_exit\_\_*, puedes agregar una suite `elif` al final del método `__exit__` que reordena la excepción inesperada al código de llamada:

```
...
self.conn.close()
if exc_type is mysql.connector.errors.ProgrammingError:
    raise SQLError(exc_value)
elif exc_type:
    raise exc_type(exc_value)
```

← This "elif" raises  
any other exception  
that might occur.

Traduciendo:

Este "elif" plantea cualquier otra excepción que podría ocurrir.



Con la compatibilidad con la excepción `SQLError` añadida a `DBcm.py`, agregue otra excepción a su función `view_the_log` para capturar cualquier `SQLErrors` que se produzca:

Agregue este código  
a la función "view\_  
the\_log" dentro de su  
aplicación web  
"vsearch4web.py".

```
...
except ConnectionError as err:
    print('Is your database switched on? Error:', str(err))
except CredentialsError as err:
    print('User-id/Password issues. Error:', str(err))
except SQLError as err:
    print('Is your query correct? Error:', str(err))
except Exception as err:
    print('Something went wrong:', str(err))
return 'Error'
```

Una vez que guarde vsearch4web.py, su aplicación web debería volver a cargarse y estar lista para las pruebas. Si intenta ejecutar una consulta SQL que contiene errores, la excepción se controla mediante el código anterior:

```
...  
Is your query correct? Error: 1146 (42S02): Table 'vsearchlogdb.logerror' doesn't exist  
127.0.0.1 - - [25/Jul/2016 21:38:25] "GET /viewlog HTTP/1.1" 200 -
```

No more generic "ProgrammingError" exceptions from MySQL Connector, as your custom exception-handling code catches these errors now.

Traduciendo:

No más excepciones genéricas de "ProgrammingError" del código MySQL Connector, ya que su código de manejo de excepciones personalizado capta estos errores ahora.

Igualmente, si sucede algo inesperado, el código catch-all de su webapp comienza a funcionar, mostrando un mensaje apropiado:

```
...  
Something went wrong: Some unknown exception.  
127.0.0.1 - - [25/Jul/2016 21:43:14] "GET /viewlog HTTP/1.1" 200 -
```

Si sucede algo inesperado, su código lo maneja.

Con el código de manejo de excepciones añadido a tu aplicación web, no importa qué error de tiempo de ejecución ocurra, tu webapp sigue funcionando sin mostrar una página de error espeluznante o confusa a tus usuarios.

*Y lo realmente agradable de esto es que este código toma la excepción genérica "ProgrammingError" proporcionada por el módulo MySQL Connector y lo convierte en dos excepciones personalizadas que tienen un significado específico para nuestra aplicación web.*

**Sí, lo hace. Y esto es muy poderoso.**

Aquí tenemos el código completo de **vsearch4web.py**

```
from flask import Flask, render_template, request, escape, session
```

```
from vsearch import search4letters

from DBcm import UseDatabase, ConnectionError, CredentialsError, SQLERror

from checker import check_logged_in

from time import sleep

app = Flask(__name__)

app.config['dbconfig'] = {'host': '127.0.0.1',
                        'user': 'vsearch',
                        'password': 'vsearchpasswd',
                        'database': 'vsearchlogDB', }

@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    return 'You are now logged in.'

@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')
    return 'You are now logged out.'

def log_request(req: 'flask_request', res: str) -> None:
    """Log details of the web request and the results."""
    pass
```

```

# raise Exception("Something awful just happened.")

with UseDatabase(app.config['dbconfig']) as cursor:

    _SQL = """insert into log

(phrase, letters, ip, browser_string, results)

values

(%s, %s, %s, %s, %s)"""

cursor.execute(_SQL, (req.form['phrase'],

                     req.form['letters'],

                     req.remote_addr,

                     req.user_agent.browser,

                     res, ))

```

```

@app.route('/search4', methods=['POST'])

def do_search() -> 'html':

    """Extract the posted data; perform the search; return results."""

    phrase = request.form['phrase']

    letters = request.form['letters']

    title = 'Here are your results:'

    results = str(search4letters(phrase, letters))

    try:

        log_request(request, results)

    except Exception as err:

        print('***** Logging failed with this error:', str(err))

    return render_template('results.html',

                           the_title=title,

                           the_phrase=phrase,

```

```
    the_letters=letters,
    the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    """Display this webapp's HTML form."""
    return render_template('entry.html',
                           the_title='Welcome to search4letters on the web!')

@app.route('/viewlog')
@check_logged_in
def view_the_log() -> 'html':
    """Display the contents of the log file as a HTML table."""
    try:
        with UseDatabase(app.config['dbconfig']) as cursor:
            _SQL = """select phrase, letters, ip, browser_string, results
                      from log"""
            cursor.execute(_SQL)
            contents = cursor.fetchall()
        # raise Exception("Some unknown exception.")
        titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
        return render_template('viewlog.html',
                               the_title='View Log',
                               the_row_titles=titles,
                               the_data=contents,
)
    except ConnectionError as err:
```

```

        print('Is your database switched on? Error:', str(err))

    except CredentialsError as err:

        print('User-id/Password issues. Error:', str(err))

    except SQLError as err:

        print('Is your query correct? Error:', str(err))

    except Exception as err:

        print('Something went wrong:', str(err))

    return 'Error'

app.secret_key = 'YouWillNeverGuessMySecretKey'

if __name__ == '__main__':
    app.run(debug=True)

```

## **A Quick Recap: Adding Robustness**

### **Una recapitulación rápida: agregando robustez**

Tomemos un minuto para recordarnos lo que nos propusimos hacer en este capítulo. Al tratar de hacer nuestro código de webapp más robusto, tuvimos que responder a cuatro preguntas relacionadas con cuatro temas identificados. Repasemos cada pregunta y observemos cómo lo hicimos:

#### **1. ¿Qué sucede si falla la conexión a la base de datos?**

Ha creado una nueva excepción llamada ConnectionError que se genera cada vez que no se encuentra su base de datos backend. A continuación, utilizó try / except para controlar un ConnectionError si se produjera.

#### **2. ¿Está protegida nuestra webapp de los ataques web?**

Fue un "accidente feliz", pero su elección de Flask además de Jinja2, junto con la especificación DB-API de Python, protege su aplicación web de los ataques de web

más notorios. Por lo tanto, sí, su webapp está protegido de algunos ataques web (pero no todos).

### **3. ¿Qué pasa si algo lleva mucho tiempo?**

Aún no hemos respondido a esta pregunta, aparte de demostrar lo que sucede cuando su aplicación web tarda 15 segundos en responder a una solicitud de usuario: su usuario web tiene que esperar (o, más probablemente, su usuario web se cansa de esperar y se va) .

### **4. ¿Qué sucede si falla una llamada de función?**

Utilizó try / except para proteger la llamada de función, lo que le permitió controlar lo que el usuario de su aplicación web ve cuando algo sale mal.

## ***¿Qué pasa si algo lleva mucho tiempo?***

Cuando realizó el ejercicio inicial al comienzo de este capítulo, esta pregunta resultó de nuestro examen de las llamadas `cursor.execute` que ocurrieron en las funciones `log_request` y `view_the_log`. Aunque ya has trabajado con ambas funciones al responder a las preguntas 1 y 4 anteriores, todavía no has terminado con ellas.

Tanto `log_request` como `view_the_log` utilizan el gestor de contexto `UseDatabase` para ejecutar una consulta SQL. La función `log_request` escribe los detalles de la búsqueda enviada a la base de datos backend, mientras que la función `view_the_log` se lee desde la base de datos.

*La pregunta es: ¿qué haces si este escribir o leer toma mucho tiempo?*

*Bueno, como con muchas cosas en el mundo de la programación, depende.*

### ***Esperar a esperar***

### ***¿Cómo hacer frente a esperar? Depende...***

Cómo decides lidiar con el código que hace que tus usuarios esperen, ya sea en una lectura o en una escritura, pueden ser complejos. Así que vamos a detener esta discusión y aplazar una solución hasta el siguiente capítulo, corto.

De hecho, el siguiente capítulo es tan corto que no garantiza su propio número de capítulo (como verás), pero el material que presenta es lo suficientemente complejo como para justificar la separación de la discusión principal de este capítulo, que presentó el try / Excepto de python el mecanismo. Así que, vamos a esperar un poco antes de poner a descansar problema 3: ¿qué pasa si algo lleva mucho tiempo?

*¿Te das cuenta de que nos estás pidiendo que esperemos para tratar con el código que espera?*

## Sí. La ironía no se pierde en nosotros.

Le estamos pidiendo que espere para aprender a manejar "espera" en su código.

Pero usted ya ha aprendido mucho en este capítulo, y creemos que es importante tomar un poco de tiempo para dejar que el material try / except se hunda en su cerebro.

Por lo tanto, nos gustaría que hiciera una pausa, y tomara un breve descanso ... después de haber echado el ojo sobre el código visto hasta ahora en este capítulo.

## Chapter 11's Code, 1 of 3

Se trata de  
"try\_example.py"

```
try:
    with open('myfile.txt') as fh:
        file_data = fh.read()
    print(file_data)
except FileNotFoundError:
    print('The data file is missing.')
except PermissionError:
    print('This is not allowed.')
except Exception as err:
    print('Some other error occurred:', str(err))
```

```
import mysql.connector

class ConnectionError(Exception):
    pass

class CredentialsError(Exception):
    pass

class SQLError(Exception):
    pass

class UseDatabase:
    def __init__(self, config: dict):
        self.configuration = config

    def __enter__(self) -> 'cursor':
        try:
            self.conn = mysql.connector.connect(**self.configuration)
            self.cursor = self.conn.cursor()
            return self.cursor
        except mysql.connector.errors.InterfaceError as err:
            raise ConnectionError(err)
        except mysql.connector.errors.ProgrammingError as err:
            raise CredentialsError(err)

    def __exit__(self, exc_type, exc_value, exc_traceback):
        self.conn.commit()
        self.cursor.close()
        self.conn.close()
        if exc_type is mysql.connector.errors.ProgrammingError:
            raise SQLError(exc_value)
        elif exc_type:
            raise exc_type(exc_value)
```

Ésta es la  
versión de  
"DBcm.py"  
de excepción.

## Chapter 11's Code, 2 of 3

Esta es la versión de "vsearch4web.py" que hace que los usuarios esperen ..."

```
from flask import Flask, render_template, request, escape, session
from flask import copy_current_request_context

from vsearch import search4letters

from DBcm import UseDatabase, ConnectionError, CredentialsError, SQLERror
from checker import check_logged_in

from time import sleep

app = Flask(__name__)

app.config['dbconfig'] = {'host': '127.0.0.1',
                         'user': 'vsearch',
                         'password': 'vsearchpasswd',
                         'database': 'vsearchlogDB', }

@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    return 'You are now logged in.'

@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')
    return 'You are now logged out.'
```

It's probably a good idea to protect this "with" statement in much the same way as you protected the "with" statement in "view\_the\_log" (on the next page).

```
@app.route('/search4', methods=['POST'])
def do_search() -> 'html':

    @copy_current_request_context
    def log_request(req: 'flask_request', res: str) -> None:
        sleep(15) # This makes log_request really slow...
        with UseDatabase(app.config['dbconfig']) as cursor:
            _SQL = """insert into log
                      (phrase, letters, ip, browser_string, results)
                      values
                      (%s, %s, %s, %s, %s)"""
            cursor.execute(_SQL, (req.form['phrase'],
                                 req.form['letters'],
                                 req.remote_addr,
                                 req.user_agent.browser,
                                 res, ))
```

The rest of "do\_search" is at the top of the next page.

Traduciendo:

1. Probablemente es una buena idea proteger esta sentencia "with" de la misma manera que protegió la sentencia "with" en "view\_the\_log" (en la página siguiente).
2. El resto de "do\_search" está en la parte superior de la página siguiente.

## Chapter 11's Code, 3 of 3

```
results = str(search4letters(phrase, letters))
try:
    log_request(request, results)
except Exception as err:
    print("***** Logging failed with this error:", str(err))
return render_template('results.html',
                      the_title=title,
                      the_phrase=phrase,
                      the_letters=letters,
                      the_results=results)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                          the_title='Welcome to search4letters on the web!')

@app.route('/viewlog')
@check_logged_in
def view_the_log() -> 'html':
    try:
        with UseDatabase(app.config['dbconfig']) as cursor:
            _SQL = """select phrase, letters, ip, browser_string, results
                      from log"""
            cursor.execute(_SQL)
            contents = cursor.fetchall()
        # raise Exception("Some unknown exception.")
        titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
        return render_template('viewlog.html',
                              the_title='View Log',
                              the_row_titles=titles,
                              the_data=contents,
)
    except ConnectionError as err:
        print('Is your database switched on? Error:', str(err))
    except CredentialsError as err:
        print('User-id/Password issues. Error:', str(err))
    except SQLError as err:
        print('Is your query correct? Error:', str(err))
    except Exception as err:
        print('Something went wrong:', str(err))
    return 'Error'

app.secret_key = 'YouWillNeverGuessMySecretKey'

if __name__ == '__main__':
    app.run(debug=True)
```

This is the  
rest of the  
"do\_search"  
function.

**11¾ a little bit of threading**

## \* **Dealing with Waiting** \*

### **Tratamiento de la espera**

**A veces, su código puede tardar mucho tiempo en ejecutarse. ?**

Dependiendo de quién sepa, esto puede o no ser un problema. Si algún código toma 30 segundos para hacer su cosa "detrás de las escenas", la espera no puede ser un problema. Sin embargo, si su usuario está esperando a que su aplicación responda, y tarda 30 segundos, todo el mundo nota. Lo que debe hacer para solucionar este problema depende de lo que está tratando de hacer (y quién está haciendo la espera). En este breve capítulo, discutiremos brevemente algunas opciones, luego veremos una solución al problema en cuestión: ¿qué sucede si algo toma demasiado tiempo?

### **Esperando: ¿Qué hacer?**

Cuando se escribe un código que tiene el potencial de hacer que los usuarios esperen, es necesario pensar cuidadosamente en lo que está tratando de hacer. Consideremos algunos puntos de vista.

- *Escucha, amigo, si tienes que esperar, no hay nada más para eso: esperas ...*
- *Tal vez, ¿cómo esperas es diferente cuando estás escribiendo que cuando estás leyendo?*
- *Como todo lo demás, todo depende de lo que estás tratando de hacer, y la experiencia de usuario que estás disparando ...*

Tal vez es el caso de que la espera de una escritura es diferente de esperar una lectura, especialmente en lo que se refiere a cómo funciona su webapp?

Echemos un vistazo a las consultas SQL en `log_request` y `view_the_log` para ver cómo las está utilizando.

### **How Are You Querying Your Database?**

## ¿Cómo estás consultando tu base de datos?

En la función `log_request`, estamos utilizando un `INSERT` de SQL para agregar detalles de la solicitud a nuestra base de datos backend. Cuando se llama `log_request`, espera mientras el `INSERT` es ejecutado por `cursor.execute`:

```
def log_request(req: 'flask_request', res: str) -> None:
    with UseDatabase(app.config['dbconfig']) as cursor:
        _SQL = """insert into log
                  (phrase, letters, ip, browser_string, results)
              values
                  (%s, %s, %s, %s, %s)"""
        cursor.execute(_SQL, (req.form['phrase'],
                             req.form['letters'],
                             req.remote_addr,
                             req.user_agent.browser,
                             res, ))
```

At this point, the webapp "blocks" while it waits for the backend database to do its thing.

Traducimos:

En este punto, la webapp "bloquea" mientras espera a que la base de datos backend haga su cosa.

El código que espera que algo externo se complete se denomina "código de bloqueo", en el que se bloquea la ejecución de su programa hasta que la espera haya terminado. Como regla general, el código de bloqueo que toma un tiempo notable es malo.

Lo mismo ocurre con la función `view_the_log`, que también espera cuando se ejecuta la consulta SQL `SELECT`:

```
@app.route('/viewlog')
@check_logged_in
def view_the_log() -> 'html':
    try:
        with UseDatabase(app.config['dbconfig']) as cursor:
            _SQL = """select phrase, letters, ip, browser_string, results
                      from log"""
            cursor.execute(_SQL)
            contents = cursor.fetchall()
            titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
            return render_template('viewlog.html',
                                  the_title='View Log',
                                  the_row_titles=titles,
                                  the_data=contents,
                                  )
    except ConnectionError as err:
        ...
To save on space, we're not showing all of the code for "view_the_log". The exception-handling code still goes here.
```

Your webapp "blocks" here, too, while it waits for the database.

Traducimos:

1. Su webapp "bloquea" aquí, también, mientras espera la base de datos.
2. Para ahorrar espacio, no mostramos todo el código de "view\_the\_log". El código de manejo de excepciones sigue aquí.

Ambas funciones bloquean. Sin embargo, mirar de cerca lo que sucede después de la llamada a cursor.execute en ambas funciones. En log\_request, el cursor.execute llama es lo último que hace la función, mientras que en view\_the\_log, los resultados de cursor.execute son utilizados por el resto de la función.

Consideremos las implicaciones de esta diferencia.

### ***Los INSERT y los SELECT de la base de datos son diferentes***

Si estás leyendo el título de esta página y pensando "¡Por supuesto que lo son!", Tenga la seguridad de que (esta tarde en este libro) no hemos perdido nuestros mármoles.

Sí: un SQL INSERT es diferente de un SQL SELECT, pero, como se relaciona con su uso de ambas consultas en su webapp, resulta que el INSERT en log\_request no necesita bloquear, mientras que el SELECT en view\_the\_log hace lo que Hace que las consultas sean muy diferentes.

Esta es una observación clave.

Si SELECT en view\_the\_log no espera a que los datos vuelvan de la base de datos backend, el código que sigue a cursor.execute probablemente fallará (ya que no tendrá datos para trabajar). La función view\_the\_log debe bloquearse, ya que tiene que esperar a que los datos se actualicen.

Cuando la aplicación web llama a log\_request, quiere que la función registre los detalles de la solicitud web actual en la base de datos. El código de llamada realmente no importa cuando esto sucede, sólo que lo hace. La función log\_request no devuelve ningún valor, ni datos; El código de llamada no está esperando una respuesta. Todo el código de llamada se preocupa es que la solicitud web se registra con el tiempo.

Lo que plantea la pregunta: ¿por qué log\_request obligar a sus llamadores a esperar?

*¿Está a punto de sugerir que el código "log\_request" podría de alguna manera ejecutarse simultáneamente con el código de la webapp?*

**Sí. Esa es nuestra idea loca.**

Cuando los usuarios de tu aplicación web inician una nueva búsqueda, no les importa que los detalles de la solicitud se registren en alguna base de datos de backend, así que no hagámoslos esperar mientras tu aplicación web funciona.

En su lugar, vamos a arreglar para algún otro proceso para hacer el registro con el tiempo e independientemente de la función principal de la webapp (que es permitir a sus usuarios realizar búsquedas).

### ***Doing More Than One Thing at Once***

#### ***Haciendo más de una cosa a la vez***

Aquí está el plan: usted va a arreglar para que la función `log_request` se ejecute independientemente de su webapp principal. Para hacer esto, usted va a ajustar el código de su webapp para que cada llamada a `log_request` se ejecute simultáneamente. Esto significa que su webapp ya no tiene que esperar a que `log_request` se complete antes de atender otra solicitud de otro usuario (es decir, no más demoras).

Si `log_request` toma un instante, unos segundos, un minuto, o incluso horas para ejecutar, su webapp no le importa (y tampoco su usuario). Lo que te importa es que el código eventualmente se ejecuta.

#### ***Código concurrente: tiene opciones***

Cuando se trata de organizar algunos de los códigos de su aplicación para ejecutar simultáneamente, Python tiene algunas opciones. Además de una gran cantidad de soporte de módulos de terceros, la biblioteca estándar incluye algunas funciones integradas que pueden ayudar aquí.

Una de las más conocidas es la biblioteca de subprocesos o `threading`, que proporciona una interfaz de alto nivel a la implementación de subprocesos proporcionada por el sistema operativo que aloja su aplicación web. Para usar la biblioteca, todo lo que necesita hacer es importar la clase `Thread` desde el módulo de subprocesamiento o `threading` cerca de la parte superior de su código de programa:

```
from threading import Thread
```

Adelante y agregue esta línea de código cerca de la parte superior de su archivo `vsearch4web.py`.

Ahora empieza la diversión.

Para crear un nuevo subproceso, cree un objeto `Thread`, asignando el nombre de la función que desea que el subproceso ejecute a un argumento con nombre denominado `target` y

proporcionando cualquier argumento como una tupla a otro argumento denominado `args`. A continuación, el objeto `Thread` creado se asigna a una variable de su elección.

Como ejemplo, supongamos que tiene una función llamada `execute_slowly`, que toma tres argumentos, que asumiremos son tres números. El código que invoca `execute_slowly` ha asignado los tres valores a variables llamadas `glacial`, `plodding`, y `leaden`. A continuación, se explica cómo se invoca normalmente a `execute_slowly` (es decir, sin preocuparnos por la ejecución simultánea):

```
execute_slowly(glacial, plodding, leaden)
```

Si `execute_slowly` toma 30 segundos para hacer lo que tiene que hacer, el código de llamada bloquea y espera 30 segundos antes de hacer cualquier otra cosa. Gorrón.

gotta love threads

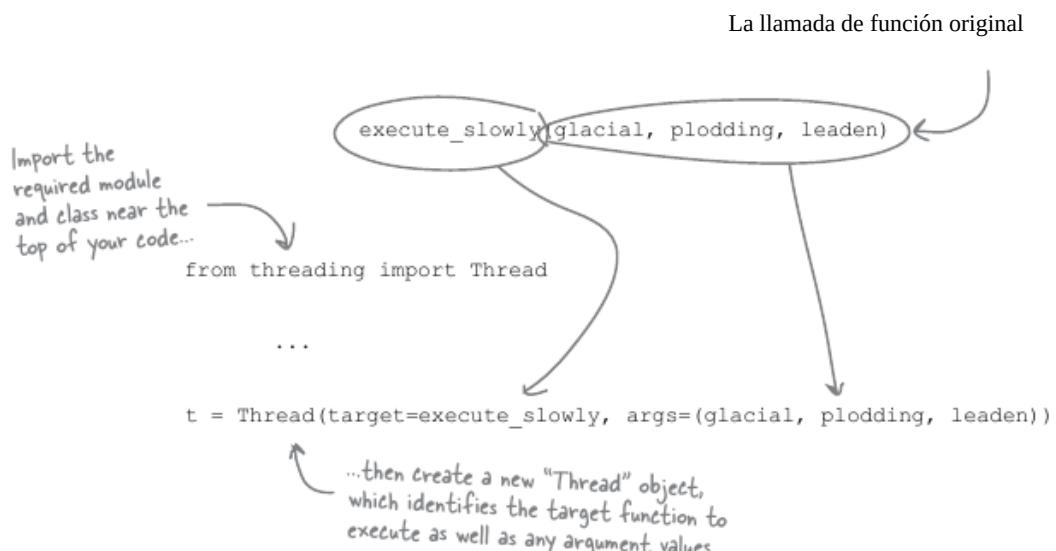
Don't Ge t Bummed Out: Use Thre ads

### **Tengo que amar hilos**

#### **No se vuelva loco: use hilos**

En el gran esquema de cosas, esperar 30 segundos para que la función `execute_slowly` complete no suena como el fin del mundo. Pero, si su usuario está sentado y esperando, se preguntarán qué ha ido mal.

Si su aplicación puede continuar ejecutándose mientras `execute_slowly` se ocupa de su negocio, puede crear un `Thread` para ejecutar `execute_slowly` simultáneamente. Esta es la llamada de función normal una vez más, junto con el código que convierte la llamada de función en una solicitud de ejecución de subprocessos:



Traducimos:

1. Importe el módulo y la clase necesarios cerca de la parte superior de su código ...

2. ... entonces cree un nuevo objeto "Thread" que identifique el función objetivo, ejecute así como cualquier argumento sobre los valores.

Por supuesto, este uso de Thread se ve un poco extraño, pero no es realmente. La clave para entender lo que está pasando aquí es notar que el objeto Thread se ha asignado a una variable (t en este ejemplo), y que la función execute\_slowly todavía no se ha ejecutado. Asignar el objeto Thread a t lo prepara para su ejecución. Para pedir a la tecnología de enrutamiento de Python que ejecute execute\_slowly, inicie el hilo como este:

t.start() ←

Cuando llama a "start", la función asociada con el hilo "t" está programada para su ejecución por el módulo "threading".

En este punto, el código que llamó t.start continúa ejecutándose. La espera de 30 segundos que resulta de ejecutar execute\_slowly no tiene ningún efecto en el código de llamada, ya que la ejecución de execute\_slowly es manejada por el módulo de enrutamiento de Python, no por usted. El módulo de subprocessamiento threading conspira con el intérprete de Python para ejecutar run\_slowly eventualmente.



Cuando se trata de llamar a log\_request en su código de webapp, sólo hay un lugar donde usted necesita buscar: en la función do\_search. Recuerde que ya ha puesto su llamada a log\_request dentro de un try / except para protegerse contra errores de ejecución inesperados.

Tenga en cuenta, también, que hemos añadido un retraso de 15 segundos -utilizando sleep(15) -a nuestro código log\_request (lo que hace que sea lento). Aquí está el código actual para do\_search:

```
@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    try:
        log_request(request, results)
    except Exception as err:
        print('***** Logging failed with this error:',
              str(err))
    return render_template('results.html',
                          the_title=title,
                          the_phrase=phrase,
                          the_letters=letters,
                          the_results=results,)
```

Así es como invoca  
actualmente  
"log\_request".

Vamos a suponer que ya ha añadido `from threading import Thread` a la parte superior del código de su aplicación web.

Coge el lápiz y, en el espacio proporcionado a continuación, escribe el código que insertas en `do_search` en lugar de la llamada estándar a `log_request`.

Recuerde: debe utilizar un objeto `Thread` para ejecutar `log_request`, tal como lo hicimos con el ejemplo `execute_slowly` de la última página.

Agregue el código de subprocesamiento que utilizaría para ejecutar eventualmente "log\_request".



*threading at work*



Cuando se trata de llamar a `log_request` en su código de webapp, sólo hay un lugar donde usted necesita buscar: en la función `do_search`. Recuerde que ya ha puesto su llamada a `log_request` dentro de un `try / except` para protegerse contra errores inesperados en tiempo de ejecución.

Tenga en cuenta, también, que hemos añadido un retraso de 15 segundos - utilizando `sleep(15)` - a nuestro código `log_request` (haciéndolo lento). Aquí está el código actual para `do_search`:

```

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    try:
        log_request(request, results)
    except Exception as err:
        print('***** Logging failed with this error:',
              str(err))
    return render_template('results.html',
                           the_title=title,
                           the_phrase=phrase,
                           the_letters=letters,
                           the_results=results)

```

*Here's how you currently invoke "log\_request".*

Asumimos que ya habías agregado `from threading import Thread` a la parte superior del código de tu webapp. En el espacio proporcionado a continuación, debía escribir el código que insertaría en `do_search` en lugar de la llamada estándar a `log_request`. Usted debía usar un objeto `Thread` para ejecutar `log_request`, al igual que lo hicimos con el ejemplo `execute_slowly` reciente.

Estamos manteniendo la declaración "try" (por ahora).

The diagram illustrates the modification of the code. It shows the original code with handwritten annotations. A large arrow points from the original `log_request` call to the new line where `t.start()` is called. The text "try:" is annotated with a bracket under it. The line `t = Thread(target=log_request, args=(request, results))` is annotated with "t.start()". A note says "The 'except' suite is unchanged, so we aren't showing it here." Another note says "Just like the earlier example, identify the target function to run, supply any arguments it needs, and don't forget to schedule your thread to run."

```

try:
    t = Thread(target=log_request, args=(request, results))
    t.start()
except ...

```

Traducimos:

1. La suite "except" no ha cambiado, por lo que no lo estamos mostrando aquí.
2. Al igual que en el ejemplo anterior, identifique la función de destino a ejecutar, proporcione cualquier argumento que necesite y no olvide programar el hilo para que se ejecute.



Con estas ediciones aplicadas a vsearch4web.py, está listo para otra ejecución de prueba. Lo que esperas ver aquí es el siguiente paso a esperar cuando ingresas una búsqueda en la página de búsqueda de tu webapp (como el código log\_request está siendo ejecutado simultáneamente por el módulo de subprocesamiento o threading). Adelante y darle un ir.

Efectivamente, en el instante en que haga clic en el botón "¡Hazlo!", Tu webapp devuelve tus resultados. La suposición es que el módulo de threading está ejecutando log\_request, y esperando el tiempo que tarda en ejecutar el código de esa función hasta completarse (aproximadamente 15 segundos).

Estás a punto de darte una palmadita en la parte posterior (por un trabajo bien hecho) cuando, de la nada y después de unos 15 segundos, la ventana de terminal de su webapp entra en erupción con mensajes de error, no a diferencia de estos:

Take a look at this message.

The last request was a success.

```
127.0.0.1 - - [29/Jul/2016 19:43:31] "POST /search4 HTTP/1.1" 200 -
Exception in thread Thread-6:
Traceback (most recent call last):
  File "vsearch4web.not_slow.with_threads.but_broken.py", line 42, in log_request
    cursor.execute(SQL, (req.form['phrase'],
  File "/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/
werkzeug/local.py", line 343, in __getattr__
...
    raise RuntimeError(_request_ctx_err_msg)
RuntimeError: Working outside of request context. ← Whoops! An uncaught exception.

Lots (!!) more traceback messages here
This typically means that you attempted to use functionality that needed
an active HTTP request. Consult the documentation on testing for
information about how to avoid this problem.

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/threading.py",
line 914, in _bootstrap_inner
    self.run()
...
RuntimeError: Working outside of request context. ← And another one...yikes!

This typically means that you attempted to use functionality that needed
an active HTTP request. Consult the documentation on testing for
information about how to avoid this problem.
```

Si comprueba su base de datos backend, aprenderá que los detalles de su solicitud web no se registraron. Con base en los mensajes anteriores, parece que el módulo de subprocesamiento threading no está nada contento con su código. Mucho del segundo grupo de mensajes de rastreo hace referencia a threading.py, mientras que el primer grupo de mensajes de rastreo hace referencia a código en las carpetas werkzeug y flask. Lo que está claro es que la adición en el código de subprocesos ha resultado en un gran desastre. ¿Qué está pasando?

## ***First Things First: Don't Panic***

### ***Primero las cosas primero: no se preocupe***

Su primer instinto puede ser para retroceder el código que ha agregado para ejecutar log\_request en su propio hilo (y obtener de nuevo a un buen estado conocido). Pero no entremos en pánico, y no hagamos eso. En su lugar, echemos un vistazo a ese párrafo descriptivo que apareció dos veces en los mensajes de seguimiento o traceback:

```
...
This typically means that you attempted to use functionality that needed
an active HTTP request. Consult the documentation on testing for
information about how to avoid this problem.
```

traducir:

“Esto normalmente significa que intentó utilizar la funcionalidad que necesitaba una solicitud HTTP activa. Consulte la documentación sobre pruebas para obtener información acerca de cómo evitar este problema.”

Este mensaje proviene de Flask, no del módulo de threading. Sabemos esto porque el módulo de threading no podría importarle menos sobre lo que usted lo utiliza para, y definitivamente no tiene ningún interés en lo que usted está intentando hacer con HTTP.

Echemos otro vistazo al código que programa el hilo para la ejecución, que sabemos que tarda 15 segundos en ejecutarse, ya que es lo que demora log\_request. Mientras está mirando este código, piense en lo que sucede durante esos 15 segundos:

```
@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    try:
        t = Thread(target=log_request, args=(request, results))
        t.start()
    except Exception as err:
        print('***** Logging failed with this error:', str(err))
    return render_template('results.html',
                          the_title=title,
                          the_phrase=phrase,
                          the_letters=letters,
                          the_results=results,)
```

What happens while this thread takes 15 seconds to execute?



El instante en que el hilo o thread está programado para su ejecución, el código de llamada (la función do\_search) continúa ejecutándose. La función render\_template se ejecuta (en un parpadeo de un ojo) y, a continuación, la función do\_search finaliza.

Cuando do\_search termina, todos los datos asociados con la función (su contexto) son recuperados por el intérprete. Las variables solicitud, frase, letras, título y resultados dejan de ser. Sin embargo, la solicitud y las variables de resultados se pasan como argumentos a log\_request, que trata de acceder a ellos 15 segundos más tarde. Lamentablemente, en ese momento, las variables ya no existen, ya que do\_search ha terminado. Gorrón.

### ***Don't Get Bummed Out: Flask Can Help***

#### ***No se vuelva loco: Flask puede ayudar***

Basado en lo que acabas de aprender, parece que la función log\_request (cuando se ejecuta dentro de un subprocesso) ya no puede "ver" sus datos de argumento. Esto se debe al hecho de que el intérprete ha sido limpiado después de sí mismo y recuperado la memoria usada por estas variables (como do\_search ha terminado). Específicamente, el objeto de petición ya no está activo, y cuando log\_request va a buscarlo, no se puede encontrar.

¿Entonces, qué puede hacerse? No se preocupe: la ayuda está a mano.

*Sólo te voy a escribir a lápiz para la próxima semana, cuando sé que me vas a pedir que reescriba la función "log\_request". ¿OKAY?*

#### **Realmente no hay necesidad de una reescritura.**

A primera vista, podría parecer que tendría que reescribir log\_request para confiar de alguna manera menos en sus argumentos ... asumiendo que es posible. Pero resulta que Flask viene con un decorador que puede ayudar aquí.

El decorador, copy\_current\_request\_context, garantiza que la petición HTTP que está activa cuando se llama a una función permanece activa incluso cuando la función se ejecuta posteriormente en un subprocesso. Para usarlo, debe agregar copy\_current\_request\_context a la lista de importaciones en la parte superior del código de su aplicación web.

Como con cualquier otro decorador, se aplica a una función existente utilizando la sintaxis @ habitual. Sin embargo, hay una advertencia: la función que se está decorando tiene que ser definida dentro de la función que lo llama; La función decorada debe anidarse dentro de su llamador (como una función interna).



Esto es lo que queremos que hagas (después de actualizar la lista de importaciones de Flask):

1. Tome la función log\_request y anótela dentro de la función do\_search.
2. Decore log\_request con @copy\_current\_request\_context.
3. Confirme que los errores de tiempo de ejecución de la última unidad de prueba han desaparecido.



Te pedimos que hiciesas tres cosas:

1. Tome la función log\_request y anótela dentro de la función do\_search.
2. Decore log\_request con @copy\_current\_request\_context.
3. Confirme que los errores de tiempo de ejecución de la última unidad de prueba han desaparecido.

Esto es lo que parece nuestro código do\_search después de realizar las tareas 1 y 2 (nota: discutiremos la tarea 3 sobre la página):

```
@app.route('/search4', methods=['POST'])
def do_search() -> 'html':
    def log_request(req: 'flask_request', res: str) -> None:
        sleep(15) # This makes log_request really slow...
        with UseDatabase(app.config['dbconfig']) as cursor:
            _SQL = """insert into log
                      (phrase, letters, ip, browser_string, results)
                  values
                  (%s, %s, %s, %s, %s)"""
            cursor.execute(_SQL, (req.form['phrase'],
                                  req.form['letters'],
                                  req.remote_addr,
                                  req.user_agent.browser,
                                  res,))

    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'
    results = str(search4letters(phrase, letters))
    try:
        t = Thread(target=log_request, args=(request, results))
        t.start()
    except Exception as err:
        print('***** Logging failed with this error:', str(err))
    return render_template('results.html',
                          the_title=title,
                          the_phrase=phrase,
                          the_letters=letters,
                          the_results=results,)
```

Task 1. The "log\_request" function is now defined (nested) inside the "do\_search" function.

Task 2. The decorator has been applied to "log\_request".

All of the rest of this code remains unchanged.

Traducir:

Tarea 1. La función "log\_request" ahora está definida (anidada) dentro de la función "do\_search".

Tarea 2. El decorador se ha aplicado a "log\_request".

Todo el resto de este código permanece sin cambios.

there are no  
Dumb Questions

P: ¿Todavía tiene sentido proteger la invocación de thread de log\_request con try / except?

**R:** No si esperas reaccionar a un problema de ejecución con log\_request, ya que el try / except habrá terminado antes de que se inicie el subprocesso. Sin embargo, su sistema puede fallar al intentar crear un nuevo hilo, por lo que consideramos que no puede resultar doloroso dejar try / except in do\_search.

## Test Drive

Tarea 3: Tomando esta última versión de vsearch4web.py para un giro confirma que los errores de tiempo de ejecución de la última unidad de prueba son una cosa del pasado. La ventana de terminal de su webapp confirma que todo está bien:

```
***  
127.0.0.1 - - [30/Jul/2016 20:42:46] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [30/Jul/2016 20:43:10] "POST /search4 HTTP/1.1" 200 -  
127.0.0.1 - - [30/Jul/2016 20:43:14] "GET /login HTTP/1.1" 200 -  
127.0.0.1 - - [30/Jul/2016 20:43:17] "GET /viewlog HTTP/1.1" 200 -  
127.0.0.1 - - [30/Jul/2016 20:43:37] "GET /viewlog HTTP/1.1" 200 -
```



No más alarmantes excepciones de tiempo de ejecución. Todos esos 200 significan que todo está bien con tu webapp. Y, 15 segundos después de enviar una nueva búsqueda, su aplicación web finalmente registra los detalles en su base de datos backend sin necesidad de que su usuario de la aplicación web de esperar.

*De acuerdo con esta carta, tengo que hacer una última pregunta. ¿Hay algún inconveniente en definir "log\_request" dentro de "do\_search"?*

### **No. No en este caso.**

Para esta aplicación web, la función log\_request sólo fue llamada por do\_search, por lo que nesting log\_request dentro de do\_search no es un problema.

Si después decide invocar log\_request de alguna otra función, puede que tenga un problema (y tendrá que repensar las cosas). Pero, por ahora, eres de oro.

### **Una última revisión**

#### **¿Es su Webapp robusto ahora?**

Aquí están las cuatro preguntas planteadas al comienzo del Capítulo 11:

- 1 ¿Qué sucede si falla la conexión a la base de datos?
- 2 ¿Está protegida nuestra webapp de los ataques web?
- 3 ¿Qué pasa si algo lleva mucho tiempo?
- 4 ¿Qué sucede si falla una llamada de función?

Su aplicación web maneja ahora una serie de excepciones de tiempo de ejecución, gracias a su uso de try / except ya algunas excepciones personalizadas que puede raise y capturar según sea necesario.

Cuando sepa que algo puede salir mal en tiempo de ejecución, fortifique su código contra cualquier excepción que pueda ocurrir. Esto mejora la solidez general de su aplicación, lo cual es una buena cosa.

Nótese que hay otras áreas en las que se podría mejorar la robustez. Pasaste mucho tiempo añadiendo try / except code al código de view\_the\_log, que aprovechó el gestor de contexto UseDatabase. UseDatabase también se utiliza dentro de log\_request, y probablemente debería ser protegido, también (y hacerlo se deja como un ejercicio de tarea para usted).

Su aplicación web es más sensible debido a su uso de threading para manejar una tarea que tiene que realizarse con el tiempo, pero no de inmediato. Esta es una buena estrategia de diseño, aunque es necesario tener cuidado de no ir por la borda con los hilos: el ejemplo de subprocessos en este capítulo es muy sencillo. Sin embargo, es muy fácil crear código de subprocessos que nadie pueda entender, y que te enloquecerá cuando tengas que depurarlo. Utilice los hilos o thread con cuidado.

En respuesta a la pregunta 3-¿qué sucede si algo lleva mucho tiempo? -el uso de subprocessos o threads mejoró el rendimiento de la escritura de la base de datos, pero no la de la base de datos. Es un caso de sólo tener que esperar a que los datos lleguen después de la lectura, no importa ahora el tiempo que tarda, ya que la aplicación web no fue capaz de proceder sin los datos.

Para hacer que la lectura de la base de datos sea más rápida (suponiendo que en realidad es lenta en primer lugar), puede que tenga que ver utilizando una configuración de base de datos alternativa (más rápido). Pero eso es una preocupación para otro día que no nos ocuparemos más adelante en este libro.

Sin embargo, habiendo dicho eso, en el próximo y último capítulo, de hecho consideramos el rendimiento, pero lo estaremos haciendo al discutir un tema que todos entienden y que ya hemos discutido en este libro: bucle.

## Chapter 11¾'s Code, 1 of 2

Esta es la última y mejor versión de "vsearch4web.py".

```

from flask import Flask, render_template, request, escape, session
from flask import copy_current_request_context
from vsearch import search4letters

from DBcm import UseDatabase, ConnectionError, CredentialsError, SQLError
from checker import check_logged_in

from threading import Thread
from time import sleep

app = Flask(__name__)

app.config['dbconfig'] = {'host': '127.0.0.1',
                        'user': 'vsearch',
                        'password': 'vsearchpasswd',
                        'database': 'vsearchlogDB', }

@app.route('/login')
def do_login() -> str:
    session['logged_in'] = True
    return 'You are now logged in.'

@app.route('/logout')
def do_logout() -> str:
    session.pop('logged_in')
    return 'You are now logged out.'

@app.route('/search4', methods=['POST'])
def do_search() -> 'html':

    @copy_current_request_context
    def log_request(req: 'flask_request', res: str) -> None:
        sleep(15) # This makes log_request really slow...
        with UseDatabase(app.config['dbconfig']) as cursor:
            _SQL = """insert into log
                      (phrase, letters, ip, browser_string, results)
                  values
                  (%s, %s, %s, %s, %s)"""
            cursor.execute(_SQL, (req.form['phrase'],
                                req.form['letters'],
                                req.remote_addr,
                                req.user_agent.browser,
                                res,))

    phrase = request.form['phrase']
    letters = request.form['letters']
    title = 'Here are your results:'

    log_request(request, title)

```

The rest of "do\_search" is  
at the top of the next page. →

## Chapter 11¾'s Code, 2 of 2

```

results = str(search4letters(phrase, letters))
try:
    t = Thread(target=log_request, args=(request, results))
    t.start()
except Exception as err:
    print('***** Logging failed with this error:', str(err))
return render_template('results.html',
                      the_title=title,
                      the_phrase=phrase,
                      the_letters=letters,
                      the_results=results,)

@app.route('/')
@app.route('/entry')
def entry_page() -> 'html':
    return render_template('entry.html',
                          the_title='Welcome to search4letters on the web!')

@app.route('/viewlog')
@check_logged_in
def view_the_log() -> 'html':
    try:
        with UseDatabase(app.config['dbconfig']) as cursor:
            _SQL = """select phrase, letters, ip, browser_string, results
                      from log"""
            cursor.execute(_SQL)
            contents = cursor.fetchall()
        # raise Exception("Some unknown exception.")
        titles = ('Phrase', 'Letters', 'Remote_addr', 'User_agent', 'Results')
        return render_template('viewlog.html',
                              the_title='View Log',
                              the_row_titles=titles,
                              the_data=contents,)
    except ConnectionError as err:
        print('Is your database switched on? Error:', str(err))
    except CredentialsError as err:
        print('User-id/Password issues. Error:', str(err))
    except SQLError as err:
        print('Is your query correct? Error:', str(err))
    except Exception as err:
        print('Something went wrong:', str(err))
    return 'Error'

app.secret_key = 'YouWillNeverGuessMySecretKey'

if __name__ == '__main__':
    app.run(debug=True)

```

↗ This is the  
rest of the  
"do\_search"  
function.

## \* Looping Like Crazy \*

### **12 iteración avanzada**

#### **Bucle como loco**

Acabo de tener la idea más maravillosa: ¿qué pasa si puedo hacer mis bucles ir más rápido?

A menudo es increíble cuánto tiempo pasan nuestros programas en bucles.

Esto no es una sorpresa, ya que la mayoría de los programas existen para realizar algo rápidamente un montón de veces. Cuando se trata de optimizar bucles, hay dos enfoques: (1) mejorar la sintaxis de bucle (para facilitar la especificación de un bucle), y (2) mejorar cómo se ejecutan los bucles (para hacerlos ir más rápido). A principios de la vida de Python 2 (es decir, hace mucho, mucho tiempo), los diseñadores de lenguaje agregaron una característica de lenguaje único que implementa ambos enfoques, y pasa por un nombre bastante extraño: comprensión. Pero no dejes que el nombre extraño te desanime: en el momento en que hayas trabajado en este capítulo, te preguntarás cómo lograste vivir sin comprensiones durante tanto tiempo.

#### **Datos de vuelos**

#### **Bahamas Buzzers tienen lugares para visitar**

Para aprender qué pueden hacer las comprensiones de bucles para usted, usted va a tomar una ojeada algunos datos "verdaderos".

*Operando desde Nassau en la isla de Nueva Providencia, Bahamas Buzzers ofrece vuelos en isla a algunos de los aeropuertos más grandes de la isla. La aerolínea ha sido pionera en la programación de vuelos just-in-time: basada en la demanda del día anterior, la aerolínea predice (cuál es un término elegante para "conjeturas") cuántos vuelos necesitan al día siguiente. Al final de cada día, la oficina central del BB genera la programación de vuelo del día siguiente, que termina en un archivo CSV basado en texto (valor separado por comas).*

Esto es lo que contiene el archivo CSV de mañana:

Se trata de un archivo CSV estándar, con la primera línea dada a la información del encabezado. Todo parece bien excepto por el hecho de que todo es MAYÚSCULAS (que es un poco "old school").

→ TIME, DESTINATION  
09:35, FREEPORT  
17:00, FREEPORT  
09:55, WEST END  
19:00, WEST END  
10:45, TREASURE CAY  
12:00, TREASURE CAY  
11:45, ROCK SOUND  
17:55, ROCK SOUND ←

El encabezado nos dice que esperamos dos columnas de datos: una representa los tiempos, los otros destinos.

El resto del archivo CSV contiene los datos de vuelo reales.

La oficina central llama a este archivo CSV `buzzers.csv`.

Si se le pidió que leyera los datos del archivo CSV y lo mostrara en la pantalla, usaría una instrucción `with`. Esto es lo que hicimos en el prompt `>>>` de IDLE, después de usar el módulo `os` de Python para cambiar a la carpeta que contiene el archivo:

The screenshot shows a Python 3.5.2 Shell window. The code reads a CSV file named 'buzzers.csv' located in the directory '/Users/paul/buzzdata'. The output shows several lines of CSV data: TIME,DESTINATION, followed by specific entries like 09:35,FREREPORT, 17:00,FREREPORT, etc. A callout points to the 'TIME,DESTINATION' part of the code and the resulting output, with the text 'Set this to the folder you're using.' A bracket below the output indicates 'The raw CSV data from the file.' Another callout points to the 'read()' method in the 'raw\_data' variable with the text 'The "read" method slurps up all of the characters in the file in one go.' To the right is a 'Geek Bits' box containing a cartoon character and the text 'Learn more about the CSV format here: [https://en.wikipedia.org/wiki/Comma-separated\\_values](https://en.wikipedia.org/wiki/Comma-separated_values)'.

```
>>> import os  
>>> os.chdir('/Users/paul/buzzdata')  
>>>  
>>> with open('buzzers.csv') as raw_data:  
    print(raw_data.read())  
  
TIME,DESTINATION  
09:35,FREREPORT  
17:00,FREREPORT  
09:55,WEST END  
19:00,WEST END  
10:45,TREASURE CAY  
12:00,TREASURE CAY  
11:45,ROCK SOUND  
17:55,ROCK SOUND  
  
>>> |  
Ln: 59 Col: 4
```

Set this to the folder you're using.

The "read" method slurps up all of the characters in the file in one go.

The raw CSV data from the file.

Geek Bits

Learn more about the CSV format here: [https://en.wikipedia.org/wiki/Comma-separated\\_values](https://en.wikipedia.org/wiki/Comma-separated_values)

traducimos:

1. Establezca esto en la carpeta que está utilizando.
2. El método "read" slurps todos los caracteres en el archivo de una sola vez.

### Lectura de datos CSV como listas

Los datos CSV, en su forma cruda, no son muy útiles. Sería más útil si pudiera leer y separar cada línea en la coma, facilitando la obtención de los datos.

Aunque es posible hacer esto "rompiendo" con el código Python hecho a mano (aprovechando el método de división del objeto de cadena), trabajar con datos CSV es una actividad tan común que la biblioteca estándar viene con un módulo llamado `csv` que puede ayudar .

Aquí hay otro pequeño para loop que muestra el módulo `csv` en acción. A diferencia del último ejemplo, donde se utilizó el método de lectura para capturar todo el contenido del archivo de una vez, en el código que sigue, `csv.reader` se utiliza para leer el archivo CSV una línea a la vez dentro del bucle `for`. En cada iteración, el bucle `for` asigna cada línea de datos CSV a una variable (llamada `linea`), la cual se muestra en la pantalla:

The screenshot shows a Python 3.5.2 Shell window. The code reads a CSV file named 'buzzers.csv' using the 'with' statement and 'csv.reader'. It prints each line from the file. The output shows the first few lines of the CSV data, which consists of two columns: 'TIME' and 'DESTINATION'. Arrows point from the explanatory text to the 'with' statement and the 'print(line)' line.

```

Python 3.5.2 Shell
>>>
>>> import csv
>>>
>>> with open('buzzers.csv') as data:
    for line in csv.reader(data):
        print(line)

['TIME', 'DESTINATION']
['09:35', 'FREEPORT']
['17:00', 'FREEPORT']
['09:55', 'WEST END']
['19:00', 'WEST END']
['10:45', 'TREASURE CAY']
['12:00', 'TREASURE CAY']
['11:45', 'ROCK SOUND']
['17:55', 'ROCK SOUND']
>>>
>>> |
Ln: 77 Col: 4

```

Abra el archivo utilizando "with" ...  
... entonces lea los datos una línea a la vez con "csv.reader".

Esto se ve mejor: cada línea de datos del archivo CSV se ha convertido en una lista.

El módulo `csv` está haciendo un poco de trabajo aquí. Cada línea de datos sin procesar se está leyendo del archivo, entonces "mágicamente" se convirtió en una lista de dos artículos.

Además de la información de encabezado (desde la primera línea del archivo) que se devuelve como una lista, cada par de tiempo de vuelo y par de destino también obtiene su propia lista. Tome nota del tipo de los datos individuales devueltos: todo es una cadena, aunque el primer elemento de cada lista (claramente) representa un tiempo.

El módulo `csv` tiene algunos trucos más en la manga. Otra función interesante es `csv.DictReader`. Vamos a ver lo que eso hace por usted.

### ***csv al diccionario***

#### ***Lectura de datos CSV como diccionarios***

Aquí hay código similar al último ejemplo, pero por el hecho de que este nuevo código utiliza `csv.DictReader` en lugar de `csv.reader`. Cuando se utiliza `DictReader`, los datos del archivo CSV se devuelven como una colección de diccionarios, con las claves para cada diccionario tomadas de la línea de encabezado del archivo CSV y los valores tomados de cada una de las líneas siguientes. Aquí está el código:

The keys

```

Python 3.5.2 Shell
>>>
>>>
>>> with open('buzzers.csv') as data:
    for line in csv.DictReader(data):
        print(line)
    
```

Usar "csv.DictReader" es un cambio simple, pero hace una gran diferencia. Lo que eran líneas de listas (la última vez) son ahora líneas de diccionarios.

The values

Lr: 79 Col: 19

```

TIME,DESTINATION
09:35,FREEPOR
17:00,FREEPOR
09:55,WEST END
19:00,WEST END
10:45,TREASURE CAY
12:00,TREASURE CAY
11:45,ROCK SOUND
17:55,ROCK SOUND
    
```

Recordar: los datos sin procesar en el archivo tienen este aspecto.

No hay duda de que esto es algo poderoso: con una sola llamada a DictReader, el módulo csv ha transformado los datos sin procesar en su archivo CSV en una colección de diccionarios Python.

Pero imagine que le han asignado la tarea de convertir los datos sin procesar en el archivo CSV basándose en los siguientes requisitos:

- 1. Convertir los tiempos de vuelo del formato de 24 horas al formato AM / PM**
- 2. Convierte los destinos de MAYÚSCULAS a Title Case**

En sí mismos, estas tareas no son difíciles. Sin embargo, cuando se consideran los datos brutos como una colección de listas o una colección de diccionarios, pueden ser. Por lo

tanto, vamos a escribir un bucle `for` para leer los datos en un solo diccionario que luego se puede utilizar para realizar estas conversiones con mucho menos alboroto.

## ***Let's Back Up a Little Bit***

### ***Vamos a retroceder un poco***

En lugar de utilizar `csv.reader` o `csv.DictReader`, vamos a rodar nuestro propio código para convertir los datos en bruto en el archivo CSV en un solo diccionario, que luego podemos manipular para realizar las conversiones necesarias.

Hemos tenido una charla con la gente de la oficina central en Bahamas Buzzers, y nos han dicho que están muy contentos con las conversiones que tenemos en mente, pero todavía quisiera que los datos guardados en su "forma cruda", como Que es cómo su tablero anticuado de las salidas espera que sus datos lleguen: formato de 24 horas para los tiempos de vuelo, y todo MAYÚSCULAS para los destinos.

Podría realizar conversiones en los datos sin procesar en su diccionario único, pero asegúrese de que las conversiones se realicen en copias de los datos, no en los datos iniciales reales como se leen. Aunque no está totalmente claro en este momento, los ruidos que salen de La oficina central parece indicar que cualquier código que crees puede tener que interactuar con algunos sistemas existentes. Por lo tanto, en lugar de enfrentar la perspectiva de convertir los datos de nuevo en su forma en bruto, vamos a leerlo en un solo diccionario como es, a continuación, convertir a copias según sea necesario (dejando los datos sin procesar en el diccionario original sin tocar).

No es un montón de trabajo (por encima de lo que tenía que ver con el módulo `csv`) para leer los datos sin procesar en un diccionario. En el código de abajo, el archivo se abre y la primera línea se lee e ignora (ya que no necesitamos la información del encabezado). Un bucle `for` entonces lee cada línea de datos sin procesar, dividiéndola en dos en la coma, con el tiempo de vuelo utilizado como clave de diccionario y el destino utilizado como valor de diccionario.

The raw data

```
TIME, DESTINATION
09:35, FREEPORT
17:00, FREEPORT
09:55, WEST END
19:00, WEST END
10:45, TREASURE CAY
12:00, TREASURE CAY
11:45, ROCK SOUND
17:55, ROCK SOUND
```

Can you break each line in two, using the comma as the delimiter?

¿Puedes romper cada línea en dos, usando la coma como el delimitador?

```

Python 3.5.2 Shell
>>> with open('buzzers.csv') as data:
    ignore = data.readline() ← Ignore the header info.
    flights = {}
    for line in data:
        k, v = line.split(',') ← Assign destination to flight time.
        flights[k] = v

>>> flights
{'12:00': 'TREASURE CAY\n', '09:35': 'FREEPORT\n', '17:00': 'FREEPORT\n', '19:00': 'WEST END\n', '17:55': 'ROCK SOUND\n', '10:45': 'TREASURE CAY\n', '09:55': 'WEST END\n', '11:45': 'ROCK SOUND\n'}
>>>
>>> import pprint
>>> pprint.pprint(flights)
{'09:35': 'FREEPORT\n',
 '09:55': 'WEST END\n',
 '10:45': 'TREASURE CAY\n',
 '11:45': 'ROCK SOUND\n',
 '12:00': 'TREASURE CAY\n',
 '17:00': 'FREEPORT\n',
 '17:55': 'ROCK SOUND\n',
 '19:00': 'WEST END\n'}
>>>
```

Annotations on the left side of the code:

- Open the file as before.
- Create a new, empty dictionary called "flights".
- Display the contents of the dictionary, which looks a little messed up until...
- ...the "pretty-printing" library produces more human-friendly output.

Annotations on the right side of the code:

- Ignore the header info.
- Process each line.
- Assign destination to flight time.
- The inclusion of the newline character looks a little strange, doesn't it?

Bottom right corner: Ln: 486 Col: 4

Break apart the line at the comma, which returns two values: the key (flight time) and value (destination).

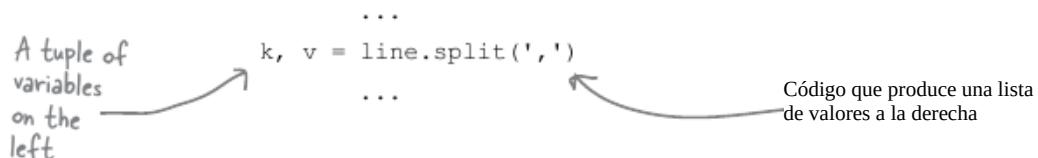
Traducimos:

1. Abra el archivo como antes.
2. Cree un nuevo diccionario vacío llamado "vuelos".
3. Mostrar el contenido del diccionario, que se ve un poco desordenado hasta ...
4. ... la biblioteca "pretty-printing" produce una salida más amigable con el ser humano.
5. Ignora la información del encabezado.
6. Separar la línea en la coma, que devuelve dos valores: la clave (tiempo de vuelo) y el valor (destino).
7. Asigne el destino al tiempo de vuelo.
8. La inclusión de la nueva linea se ve un poco extraño, ¿no?

## No a la nueva línea

### Separar, luego dividir, sus datos sin procesar

La última versión with la sentencia utilizó el método de split(incluido con todos los objetos de cadena) para romper la línea de datos sin procesar en dos. Lo que se devuelve es una lista de cadenas, que el código asigna individualmente a las variables k y v. Esta asignación multivariable es posible debido al hecho de que tiene una tupla de variables a la izquierda del operador de asignación, así como el código que produce una lista de valores a la derecha del operador (recuerde: las tuplas son listas inmutables):



Otro método de cadena, `strip`, elimina espacios en blanco desde el principio y el final de una cadena existente. Vamos a usarlo para eliminar la línea de salida no deseada de los datos en bruto antes de preformar la división o `split`.

Aquí hay una versión final de nuestro código de lectura de datos. Creamos un diccionario llamado `vuelos`, que utiliza los tiempos de vuelo como keys y los destinos (sin la nueva línea) como valores:

*Espacio en blanco: los siguientes caracteres se consideran espacios en blanco en cadenas: space, \t, \n y \r.*

```

Python 3.5.2 Shell
>>>
>>> with open('buzzers.csv') as data:
    ignore = data.readline()
    flights = {}
    for line in data:
        k, v = line.strip().split(',')
        flights[k] = v

>>> pprint.pprint(flights)
{'09:35': 'FREEPORT',
 '09:55': 'WEST END',
 '10:45': 'TREASURE CAY',
 '11:45': 'ROCK SOUND',
 '12:00': 'TREASURE CAY',
 '17:00': 'FREEPORT',
 '17:55': 'ROCK SOUND',
 '19:00': 'WEST END'}
>>>
>>> |

```

This code strips the line, then splits it, to produce the data in the format required.

Es posible que no haya detectado esto, pero el orden de las filas en el diccionario diferente de lo que está en el archivo de datos. Esto sucede porque los diccionarios NO mantienen el orden de inserción. No te preocunes por esto por ahora.

TIME, DESTINATION

- 09:35, FREEPORT
- 17:00, FREEPORT
- 09:55, WEST END
- 19:00, WEST END
- 10:45, TREASURE CAY
- 12:00, TREASURE CAY
- 11:45, ROCK SOUND
- 17:55, ROCK SOUND

¿Qué pasa si cambia el orden de los métodos en su código, así:

`line.split(',') .strip()`

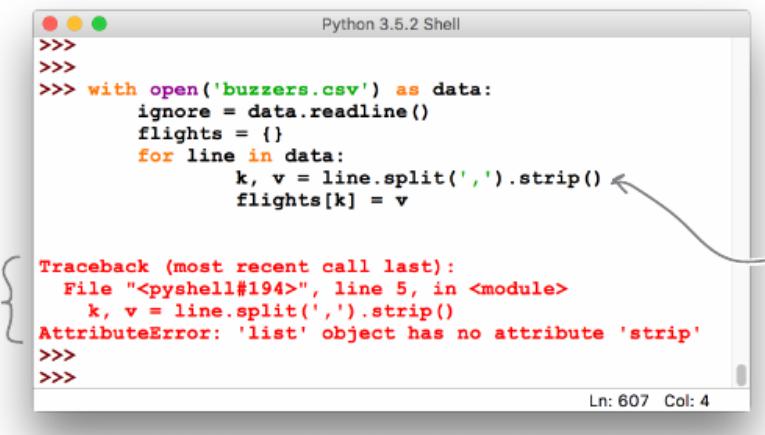
Cuando se encadenan métodos juntos de esta manera, se denomina "cadena de métodos".

¿Qué crees que pasaría?

### ***Be Careful When Chaining Method Calls***

### ***Tenga cuidado al encadenar las llamadas de método***

A algunos programadores no les gusta el hecho de que las llamadas al método de Python se pueden encadenar (como `strip` y `split` en el último ejemplo), porque tales cadenas pueden ser difíciles de leer la primera vez que las veas. Sin embargo, el encadenamiento de métodos es popular entre los programadores de Python, por lo que es probable que ejecute a través de código que utiliza esta técnica "en el medio silvestre". Sin embargo, se necesita cuidado, ya que el orden de las llamadas de método no es intercambiable.



Para entender lo que está sucediendo aquí, considere el tipo de datos a la derecha del operador de asignación como se ejecuta la cadena de método anterior.

Antes de que algo suceda, la línea es una cadena. La llamada `split` en una cadena devuelve una lista de cadenas, utilizando el argumento para dividir o `split` como un delimitador. Lo que comenzó como una cadena (línea) se ha transformado dinámicamente en una lista, que luego tiene otro método invocado en su contra. En este ejemplo, el siguiente método es `strip`, que espera ser invocado en una cadena, no una lista, por lo que el intérprete plantea un `AttributeError`, ya que las listas no tienen un método llamado `strip`.

La cadena de métodos de la página anterior no sufre de este problema:

```
    ...
line.strip().split(',')
    ...
```

Con este código, el intérprete comienza con una cadena (en línea), que tiene cualquier espacio en blanco / arrastrado eliminado por strip (que da otra cadena), que luego se divide en una lista de cadenas basadas en el delimitador de coma. No hay AttributeError, ya que la cadena de métodos no infringe ninguna regla de tipado.

*Transformando datos*

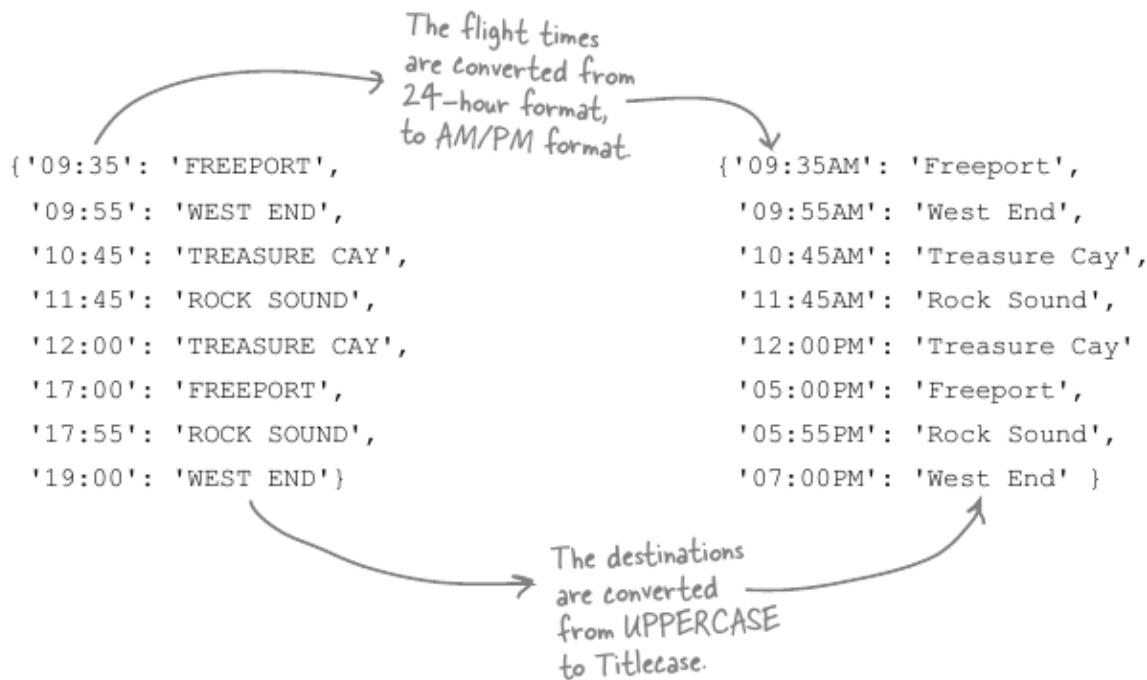
## **Transformación de datos en el formato que necesita**

Ahora que los datos están en el diccionario de vuelos, consideremos las manipulaciones de datos que BB Head Office le ha pedido que realice.

La primera es realizar las dos conversiones identificadas anteriormente en este capítulo, creando un nuevo diccionario en el proceso:

- 1.-Convertir los tiempos de vuelo del formato de 24 horas al formato AM / PM**
- 2.- Convierte los destinos de MAYÚSCULAS a Title Case**

Aplicar estas dos transformaciones al diccionario de vuelos le permite girar el diccionario a la izquierda en el de la derecha:



Traducción:

- Los tiempos de vuelo se convierten del formato de 24 horas al formato AM / PM.
- Los destinos se convierten de MAYÚSCULAS a Caso de Título.

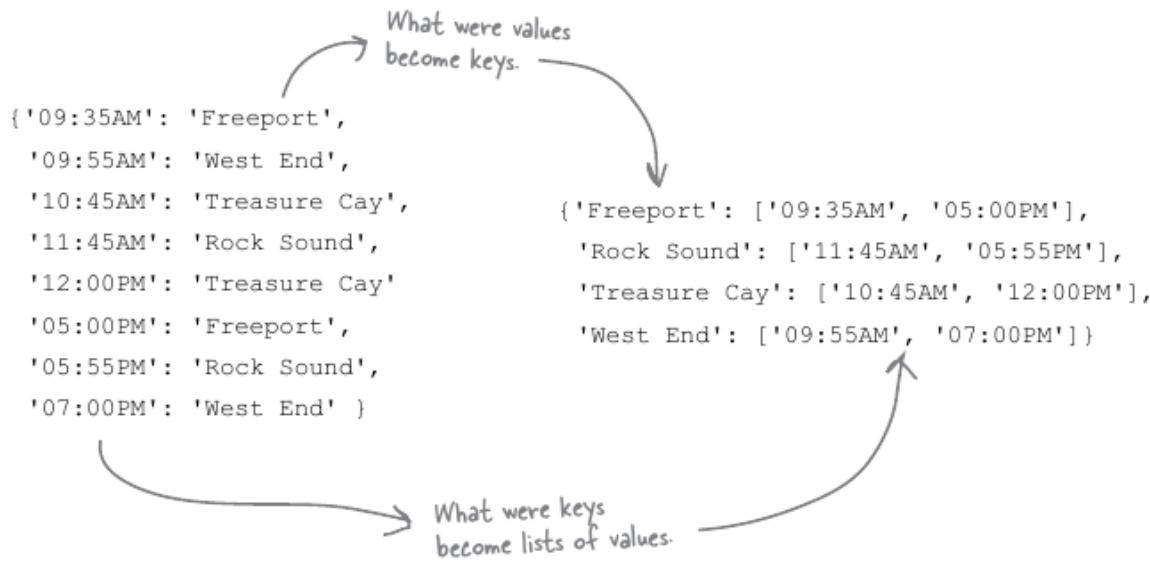
Tenga en cuenta que los datos en ambos diccionarios tienen el mismo significado, es sólo la representación que ha cambiado. La oficina central necesita el segundo diccionario, ya que sienten que sus datos son más universalmente comprensibles, así como más amigables; La oficina central piensa que todo - MAYÚSCULAS es semejante al grito.

Actualmente, los datos en ambos diccionarios tienen una sola línea para cada combinación de tiempo de vuelo / destino. Aunque la oficina central estará encantada cuando haya transformado el diccionario de la izquierda en el diccionario de la derecha, también han sugerido que sería realmente útil si los datos pudieran presentarse con destinos únicos como keys y una lista de valores de vuelos, es decir, una sola fila de datos para cada destino. Veamos cómo aparecería ese diccionario antes de embarcarse en la codificación de las manipulaciones requeridas.

## ***Transforming into a Dictionary Of Lists***

### ***Transformando en Diccionario de Listas***

Una vez que los datos en los vuelos se han transformado, la oficina central quiere que usted realice esta segunda manipulación (discutido abajo de la última página):



traducción:

1. Los valores se convirtieron en claves.
2. Las claves se convierten en listas de valores.

## ***Think about the data wrangling that's needed here...***

### ***Piensa en la manipulación de datos que se necesita aquí ...***

Hay un poco de trabajo requerido para obtener de los datos sin procesar en el archivo CSV al diccionario de las listas mostradas arriba a la derecha. Tómese un momento para pensar en cómo podría hacer esto usando el Python que ya conoce.

Si eres como la mayoría de los programadores, no te tomará mucho tiempo averiguar que el bucle for es tu amigo aquí. Como mecanismo de bucle principal de Python, ya le ha ayudado a extraer los datos sin procesar del archivo CSV y llenar el diccionario de vuelos:

Este es un uso clásico de "for",  
y un lenguaje de programación  
enormemente popular en  
python.

```
with open('buzzers.csv') as data:  
    ignore = data.readline()  
    flights = {}  
    for line in data:  
        k, v = line.strip().split(',')  
        flights[k] = v
```

Es tentador sugerir que este código sea modificado para realizar las transformaciones a los datos sin procesar, ya que se lee desde el archivo CSV, es decir, antes de agregar filas de datos a los vuelos. Pero recuerde la solicitud de la Sede de que los datos en bruto permanezcan intactos en los vuelos: cualquier transformación debe aplicarse a una copia de los datos. Esto hace las cosas más complejas, pero no por mucho.

basic converts first

Let's Do the Basic Conversions

### **Conversiones básicas primero**

#### **Hagamos las Conversiones Básicas**

Por el momento, el diccionario de vuelos contiene los tiempos de vuelo en formato de 24 horas como sus claves, con cadenas de MAYÚSCULAS que representan destinos como sus valores. Tiene dos conversiones iniciales para realizar:

- 1.- Convertir los tiempos de vuelo del formato de 24 horas al formato AM / PM**
- 2.- Convierte los destinos de MAYÚSCULAS a Title Case**

Conversión # 2 es fácil, así que hagamos eso primero. Una vez que los datos están en una cadena, simplemente llame al método de título de la cadena, como muestra esta sesión IDLE:

```
>>> s = "I DID NOT MEAN TO SHOUT."  
>>> print(s)  
I DID NOT MEAN TO SHOUT.  
El método "title"  
devuelve una copia de los  
datos en "s". → >>> t = s.title()  
>>> print(t)  
I Did Not Mean To Shout.  
Esto es mucho más amigable que  
antes.
```

La conversión # 1 implica un poco más de trabajo.

Si usted piensa en ello por un minuto, las cosas se involucran bastante cuando se trata de convertir 19:00 en 7:00 PM. Sin embargo, esto es sólo el caso cuando se miran los datos de

19:00 como una cadena. Tendrías que escribir un montón de código para hacer la conversión.

Si en su lugar considera que 19:00 es un tiempo, puede aprovechar el módulo de fecha y hora que se incluye como parte de la biblioteca estándar de Python. La clase datetime de este módulo puede tomar una cadena (como 19:00) y convertirla a su formato AM / PM equivalente utilizando dos funciones preconfiguradas y lo que se conoce como especificadores de formato de cadena. Aquí hay una pequeña función, llamada convert2ampm, que utiliza las instalaciones del módulo datetime para realizar la conversión que necesita:

Para obtener más información sobre los especificadores de formato de cadena, consulte <https://docs.python.org/3/library/datetime.html#strftime-and-strptime-comportamiento>.



### Ready Bake Code

```
from datetime import datetime

def convert2ampm(time24: str) -> str:
    return datetime.strptime(time24, '%H:%M').strftime('%I:%M%p')
```

Dado un tiempo en el formato de 24 horas (como una cadena), esta cadena de método lo convierte en una cadena en formato AM / PM.



Vamos a poner las técnicas de conversión de la última página para trabajar. A continuación se muestra el código que lee los datos sin procesar del archivo CSV, llenando el diccionario de vuelos a medida que avanza. También se muestra la función convert2ampm.

Su trabajo consiste en escribir un bucle for que toma los datos en los vuelos y convierte las claves al formato AM / PM, y los valores a Titlecase. Un nuevo diccionario, llamado flights2, se crea para contener los datos convertidos. Utilice su lápiz para agregar el código de bucle for en el espacio proporcionado.

Sugerencia: al procesar un diccionario con un bucle for, recuerde que el método items devuelve la clave y el valor de cada fila (como una tupla) en cada iteración.

```
Define the conversion function.  
from datetime import datetime  
import pprint  
  
def convert2ampm(time24: str) -> str:  
    return datetime.strptime(time24, '%H:%M').strftime('%I:%M%p')  
  
Grab the data from the file.  
with open('buzzers.csv') as data:  
    ignore = data.readline()  
    flights = {}  
    for line in data:  
        k, v = line.strip().split(',')  
        flights[k] = v  
  
    pprint.pprint(flights)  
    print()  
  
    flights2 = {} ← The new dictionary, called "flights2", starts out empty.  
  
    Pretty-print the "flights" dictionary prior to performing the conversions.  
  
Add your "for" loop here.  
.....  
.....  
pprint.pprint(flights2)  
  
Pretty-print the "flights2" dictionary to confirm that the conversions are working.
```

Traducción:

1. Defina la función de conversión.
2. Coge los datos del archivo.
3. Añada su bucle "for" aquí.
4. Pretty-print a el diccionario de "vuelos" antes de realizar conversiones.
5. El nuevo diccionario, llamado. "Vuelos2" comienza vacío
6. Pretty-print el diccionario "flights2" para confirmar que las conversiones están funcionando.



Tu trabajo consistía en escribir un bucle for que toma los datos en los vuelos y convierte las claves al formato AM / PM, y los valores a Titlecase. Debías crear un nuevo diccionario,

llamado flights2, para mantener los datos convertidos, y debías agregar el código de bucle for en el espacio proporcionado.

Guardamos todo este código en un archivo llamado "do\_convert.py".

```
from datetime import datetime
import pprint

def convert2ampm(time24: str) -> str:
    return datetime.strptime(time24, '%H:%M').strftime('%I:%M%p')

with open('buzzers.csv') as data:
    ignore = data.readline()
    flights = {}
    for line in data:
        k, v = line.strip().split(',')
        flights[k] = v

pprint.pprint(flights)
print()

flights2 = {}

for k, v in flights.items():
    flights2[convert2ampm(k)] = v.title()

pprint.pprint(flights2)
```

El método "items" devuelve cada fila del diccionario "flights".

On each iteration, the key (in "k") is converted to AM/PM format, then used as the new dictionary's key.

The value (in "v") is converted to Titlecase, then assigned to the converted key.

Traducción:

1. En cada iteración, la clave (en "k") se convierte en formato AM / PM y se utiliza como clave del nuevo diccionario.
2. El valor (en "v") se convierte en Title Case y se asigna a la clave convertida.



Si ejecuta el programa anterior, aparecerán dos diccionarios en la pantalla (que mostramos a continuación, uno al lado del otro). Las conversiones funcionan, aunque el orden en cada diccionario difiere, ya que el intérprete no mantiene el orden de inserción cuando rellena un nuevo diccionario con datos:

```

This is "flights".
{'09:35': 'FREEPORT',
 '09:55': 'WEST END',
 '10:45': 'TREASURE CAY',
 '11:45': 'ROCK SOUND',
 '12:00': 'TREASURE CAY',
 '17:00': 'FREEPORT',
 '17:55': 'ROCK SOUND',
 '19:00': 'WEST END'}

{'05:00PM': 'Freeport',
 '05:55PM': 'Rock Sound',
 '07:00PM': 'West End',
 '09:35AM': 'Freeport',
 '09:55AM': 'West End',
 '10:45AM': 'Treasure Cay',
 '11:45AM': 'Rock Sound',
 '12:00PM': 'Treasure Cay'}

```

The raw data is transformed.

488 Chapter 12

## ¿Detectó el patrón en su código?

Echa un vistazo al programa que acabas de ejecutar. Hay un patrón de programación muy común utilizado dos veces en este código. ¿Puedes distinguirlo?

```

from datetime import datetime
import pprint

def convert2ampm(time24: str) -> str:
    return datetime.strptime(time24, '%H:%M').strftime('%I:%M%p')

with open('buzzers.csv') as data:
    ignore = data.readline()
    flights = {}
    for line in data:
        k, v = line.strip().split(',')
        flights[k] = v

pprint pprint(flights)
print()

flights2 = {}
for k, v in flights.items():
    flights2[convert2ampm(k)] = v.title()

pprint pprint(flights2)

```

Si respondió: "el bucle for", sólo tiene la mitad de la razón. El bucle for es parte del patrón, pero echa otro vistazo al código que lo rodea. ¿Algo más?

```

from datetime import datetime
import pprint

def convert2ampm(time24: str) -> str:
    return datetime.strptime(time24, '%H:%M').strftime('%I:%M%p')

with open('buzzers.csv') as data:
    ignore = data.readline()
    flights = {}
    for line in data:
        k, v = line.strip().split(',')
        flights[k] = v

pprint.pprint(flights)
print()

flights2 = {}
for k, v in flights.items():
    flights2[convert2ampm(k)] = v.title()

pprint.pprint(flights2)

```

Each of the "for" loops is preceded by the creation of a new, empty data structure (e.g., a dictionary).

Each of the "for" loop's suites contains code that adds data to the new data structure, based on the processing of some existing data.

Traducción:

1. Cada uno de los bucles "for" está precedido por la creación de una nueva estructura de datos vacía (por ejemplo, un diccionario).
2. Cada una de las suites del bucle "for" contiene código que agrega datos a la nueva estructura de datos, basada en el procesamiento de algunos datos existentes.

### ***Detectar el patrón***

### ***Spotting the Pattern with Lists***

### ***Detección del patrón con listas***

Los ejemplos de la última página resaltaron el patrón de programación en relación con los diccionarios: empiece con un nuevo diccionario vacío y luego use un bucle for para procesar un diccionario existente, generando datos para un nuevo diccionario a medida que vaya:

The diagram shows a code snippet with annotations:

- The new, initially empty, dictionary:** Points to the line `flights2 = {}`.
- A regular "for" loop processes the existing data:** Points to the loop `for k, v in flights.items():`.
- The existing dictionary:** Points to the line `flights`.
- The existing data is used to generate keys and values, which are inserted into the new dictionary:** Points to the assignment `flights2[convert2ampm(k)] = v.title()`.

```

flights2 = {}
for k, v in flights.items():
    flights2[convert2ampm(k)] = v.title()

```

Este patrón también hace una aparición con listas, donde es más fácil de detectar. Eche un vistazo a esta sesión IDLE, donde las keys(es decir, los tiempos de vuelo) y los valores (es decir, los destinos) se extraen del diccionario de vuelos como listas y luego se convierten en nuevas listas usando el patrón de programación En las anotaciones):

```

1. Start with a new, empty list.
>>>
>>>
>>> flight_times = []
>>> for ft in flights.keys():
    flight_times.append(convert2ampm(ft))

2. Iterate through each of the flight times.
>>> print(flight_times)
['05:00PM', '09:55AM', '11:45AM', '10:45AM', '07:00PM', '05:55PM',
 '12:00PM', '09:35AM']

3. Append the converted data to the new list.
>>>
>>> destinations = []
>>> for dest in flights.values():
    destinations.append(dest.title())

2. Iterate through each of the destinations.
>>> print(destinations)
['Freeport', 'West End', 'Rock Sound', 'Treasure Cay', 'West End',
 'Rock Sound', 'Treasure Cay', 'Freeport']

3. Append the converted data to the new list.
>>>
>>> |

```

Annotations:

- 1. Start with a new, empty list.
- 4. View the new list's data.
- 1. Start with a new, empty list.
- 4. View the new list's data.
- 2. Iterate through each of the flight times.
- 3. Append the converted data to the new list.
- 2. Iterate through each of the destinations.
- 3. Append the converted data to the new list.

Este patrón se utiliza con tanta frecuencia que Python proporciona una notación taquigráfica conveniente para que se llama la comprensión. Veamos qué implica crear una comprensión.

## *Converting Patterns into Comprehensions*

### *Convertir patrones en comprensiones*

Tomemos el bucle más reciente para procesar los destinos como nuestro ejemplo. Aquí está de nuevo:

```

1. Start with a new, empty list.
destinations = []
for dest in flights.values():
    destinations.append(dest.title())

```

Annotations:

1. Start with a new, empty list.
2. Iterar a través de cada uno de los destinos.
3. Añada los datos convertidos a la nueva lista.

La función de comprensión integrada de Python le permite volver a trabajar las tres líneas de código anteriores como una sola línea.

Para convertir las tres líneas anteriores en una comprensión, vamos a pasar por el proceso, construyendo hasta la comprensión completa.

Comience por comenzar con una nueva lista vacía, que se asigna a una nueva variable (que estamos llamando `more_dests` en este ejemplo):

```
more_dests = []
```

1. Comience con una nueva lista vacía (y déle un nombre).

Especifique cómo se itera los datos existentes (en los vuelos de este ejemplo) usando el familiar anotación `for` y coloque este código dentro de los corchetes de la nueva lista (observe la ausencia de dos puntos al final del código `for`):

```
more_dests = [for dest in flights.values()]
```

2. Iterar a través de cada uno de los destinos.

Tenga en cuenta que aquí no hay colon o dos puntos.

Para completar la comprensión, especifique la transformación a aplicar a los datos (en `dest`), y ponga esta transformación antes de la palabra clave `for` (observe la ausencia de la llamada a `append`, que es asumida por la comprensión):

```
more_dests = [dest.title() for dest in flights.values()]
```

3. Append the converted data to the new list, without actually calling "append".

traducimos:

3. Agregue los datos convertidos a la nueva lista, sin llamar realmente "append".

Y eso es. La única línea de código en la parte inferior de esta página es funcionalmente equivalente a las tres líneas de código en la parte superior. Sigue adelante y ejecuta esta línea de código en tu solicitud >>> para convencerte de que la lista `more_dests` contiene los mismos datos que la lista de destinos.

### ***Alternativa a for***

### ***Take a Closer Look at the Comprehension***

### **Echa un vistazo a la comprensión**

Echemos un vistazo a la comprensión con un poco más de detalle. Aquí están las tres líneas originales de código, así como la comprensión de una sola línea que realiza la misma tarea.

Recuerde: ambas versiones producen nuevas listas (`destinations` y `more_dests`) que tienen exactamente los mismos datos:

```

destinations = []
for dest in flights.values():
    destinations.append(dest.title())

```

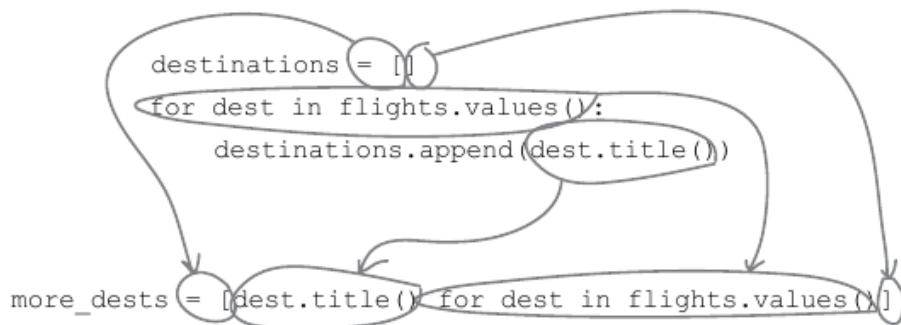
↓

```

more_dests = [dest.title() for dest in flights.values()]

```

También es posible seleccionar las partes de las tres líneas originales de código y ver dónde se han utilizado en la comprensión:



Si detecta este patrón en otro código, puede convertirlo fácilmente en comprensión. Por ejemplo, aquí hay algún código anterior (que produce la lista de tiempos de vuelo de AM / PM) reelaborados como una comprensión:

```

flight_times = []
for ft in flights.keys():
    flight_times.append(convert2ampm(ft))

```

↓

```

fts2 = [convert2ampm(ft) for ft in flights.keys()]

```

These do the  
\*same thing\*.

traducción: Estos hacen lo mismo.

**What's the Big Deal?**  
**¿Cuál es el problema?**

*Estas comprensiones parecen difíciles de entender. Estoy bastante feliz usando un bucle "for" cuando necesito hacer algo como esto. Es el aprendizaje de cómo escribir comprensiones realmente vale la pena el esfuerzo?*

**Sí. Creemos que merece la pena el esfuerzo.**

Hay dos razones principales por las que tomar el tiempo para entender comprensiones vale la pena.

En primer lugar, además de requerir menos código (lo que significa que las comprensiones son más fáciles en los dedos), el intérprete de Python está optimizado para ejecutar comprensiones lo más rápido posible. Esto significa que las comprensiones se ejecutan más rápido que el equivalente para el código de bucle for.

En segundo lugar, las comprensiones se pueden utilizar en lugares donde los bucles no pueden. De hecho, ya has visto esto, ya que todas las comprensiones presentadas hasta ahora en este capítulo han aparecido a la derecha del operador de asignación, algo que un loop regular no puede hacer. Esto puede ser sorprendentemente útil (como verá a medida que avance este capítulo).

***Comprehensions aren't just for lists***

***Las comprensiones no son sólo para listas***

Las comprensiones que has visto hasta ahora han creado nuevas listas, por lo que cada una se conoce como una lista de comprensión (o listcomp para abreviar). Si su comprensión crea un nuevo diccionario, se lo conoce como comprensión de diccionario (dictcomp). Y, para no dejar ninguna estructura de datos fuera, también puede especificar una comprensión de conjunto (setcomp).

No hay tal cosa como una comprensión de la tupla; Explicaremos por qué más adelante en este capítulo. Primero, sin embargo, echemos un vistazo a la comprensión de un diccionario.

***dictionary comprehensions***

***Specifying a Dictionary Comprehension***

***Comprensiones de diccionario***

***Especificación de una comprensión de diccionario***

Recuerde el código anterior de este capítulo que lee los datos sin procesar del archivo CSV en un diccionario llamado vuelos. Estos datos se transformaron en un nuevo diccionario denominado flights2, que está codificado por los tiempos de vuelo AM / PM y utiliza destinos "titulados" o "titlecased" como valores:

```

...
flights2 = {}
for k, v in flights.items():
    flights2[convert2ampm(k)] = v.title()
...

```

Este código se ajusta al "patrón de comprensión".

Vamos a replantear estas tres líneas de código como comprensión de diccionario.

Comience asignando un nuevo diccionario vacío a una variable (que llamamos `more_flights`):

```
more_flights = {}
```

1. Start with a new, empty dictionary.

Especifique cómo se itera los datos existentes (en los vuelos) con el uso de la notación de bucle `for` (asegurándose de no incluir los dos puntos habituales):

```
more_flights = {for k, v in flights.items() }
```

2. Iterate through each of the keys and values from the existing data.

Note that there's NO colon here.

Traduciendo:

- 2. Iterar a través de cada una de las claves y valores de los datos existentes.
- Tenga en cuenta que aquí no hay colon o dos puntos.

Para completar el dictcomp, especifique cómo se relacionan entre sí las claves y los valores del nuevo diccionario. El bucle `for` en la parte superior de la página produce la clave convirtiéndola a un tiempo de vuelo AM / PM usando la función `convert2ampm`, mientras que el valor asociado se convierte en `titlecase` gracias al método `title` del string. Un dictcomp equivalente puede hacer lo mismo y, como con listcomps, esta relación se especifica a la izquierda de la palabra clave `dictcomp`. Observe la inclusión de los dos puntos que separan la nueva clave del nuevo valor:

```
more_flights = {convert2ampm(k): v.title() for k, v in flights.items()}
```

3. Associate the converted key with its "titlecased" value (and note the use of the colon here).

traducido: 3. Asociar la clave convertida con su valor "titulado" (y observe el uso de los dos puntos aquí).

Y ahí está: tu primera comprensión de diccionario. Adelante y tomarlo para una vuelta para confirmar que trabaja.

## ***Extend Comprehensions with Filters***

### ***Extender comprensiones con filtros***

Imaginemos que sólo necesita los datos de vuelo convertidos para Freeport.

Volviendo al bucle for original, probablemente extendería el código para incluir una instrucción if que filtra basándose en el valor actual en v (el destino), produciendo código como este:

Los datos de vuelo sólo se convierten y se agregan al diccionario "just\_freeport" si se refiere al destino de Freeport.

```
just_freeport = {}
for k, v in flights.items():
    if v == 'FREEPORT':
        just_freeport[convert2ampm(k)] = v.title()
```

TIME	DESTINATION
09:35	FREEPORT
17:00	FREEPORT
09:55	WEST END
19:00	WEST END
10:45	TREASURE CAY
12:00	TREASURE CAY
11:45	ROCK SOUND
17:55	ROCK SOUND

The raw data

Si ejecuta el código de bucle anterior en el prompt >>>, terminará con sólo dos filas de datos (que representan los dos vuelos programados a Freeport como contenidos en el archivo de datos sin procesar). Esto no debería ser sorprendente, ya que usar un if de esta manera para filtrar datos es una técnica estándar. Resulta que estos filtros se pueden utilizar con comprensiones, también. Simplemente tome la sentencia if (menos los dos puntos) y tápelo al final de su comprensión. Aquí está el dictcomp de la parte inferior de la última página:

```
more_flights = {convert2ampm(k): v.title() for k, v in flights.items()}
```

Y he aquí una versión del mismo dictcomp con el filtro añadido:

```
just_freeport2 = {convert2ampm(k): v.title() for k, v in flights.items() if v == 'FREEPORT'}
```

Los datos de vuelo sólo se convierten y se añaden al diccionario "just\_freeport2" si se refiere al destino de Freeport.

Si ejecuta este dictcomp filtrado en su prompt >>>, los datos en el diccionario recién creado just\_freeport2 son idénticos a los datos en just\_freeport. Los datos de just\_freeport y just\_freeport2 son una copia de los datos originales en el diccionario de vuelos.

Por supuesto, la línea de código que produce just\_freeport2 parece intimidante. Muchos programadores nuevos en Python se quejan de que las comprensiones son difíciles de leer. Sin embargo, recuerde que la regla usual de fin de línea de fin de instrucción de Python se desactiva cuando el código aparece entre un par de paréntesis, por lo que puede reescribir cualquier comprensión sobre varias líneas para que sea más fácil de leer, así:

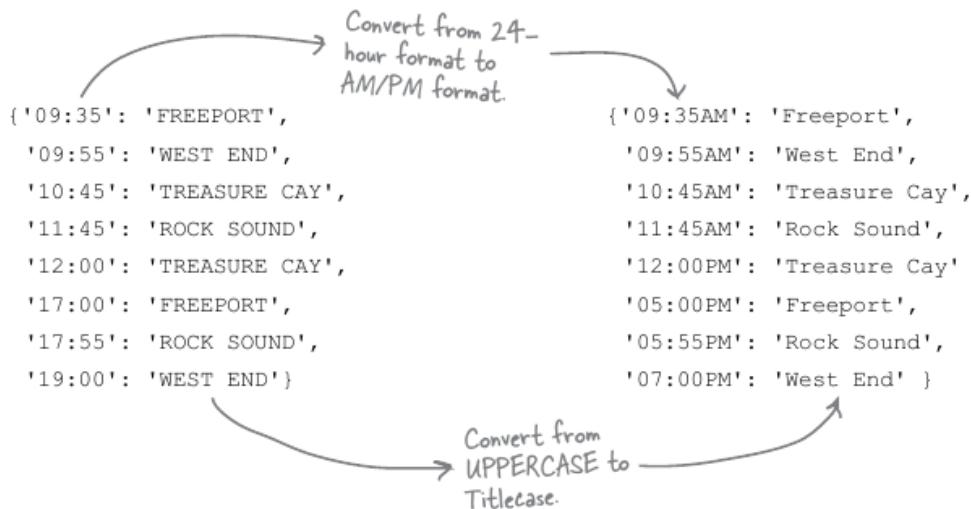
```
just_freeport3 = {convert2ampm(k): v.title()
                  for k, v in flights.items()
                  if v == 'FREEPORT'}
```

Tendrá que acostumbrarse a leer las comprensiones de una línea. Dicho esto, los programadores de Python están escribiendo cada vez más comprensiones en varias líneas (por lo que también verá esta sintaxis).

## **Una revisión rápida**

### **Recuerde lo que se propuso hacer**

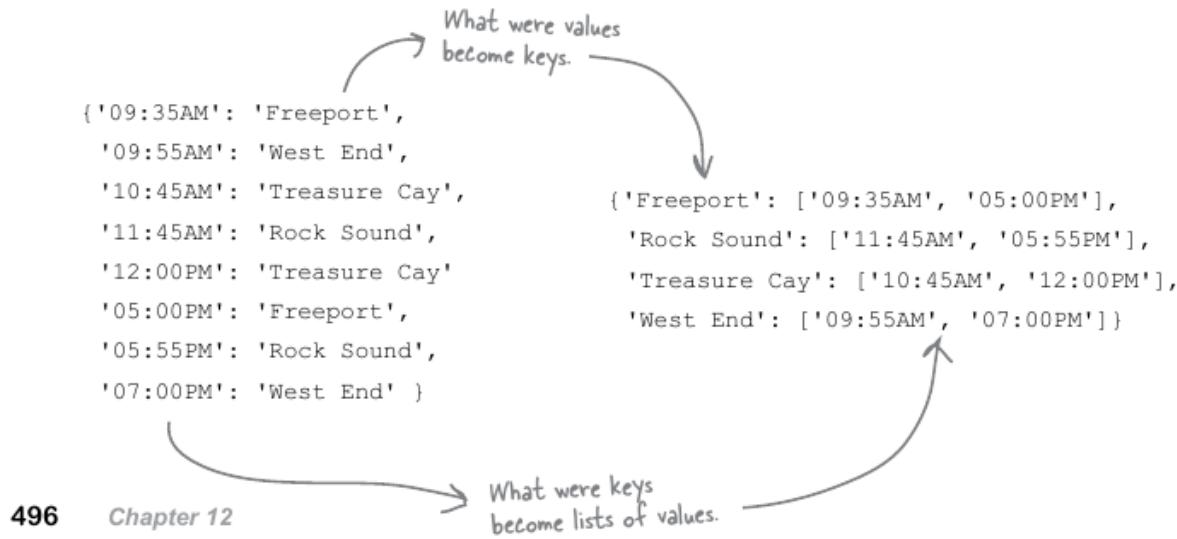
Ahora que ha visto lo que las comprensiones pueden hacer por usted, vamos a revisar las manipulaciones de diccionario requeridas anteriormente de este capítulo para ver cómo estamos haciendo. Aquí está el primer requisito:



Teniendo en cuenta los datos del diccionario de vuelos, ha visto que la siguiente comprensión de diccionario realiza las conversiones anteriores en una línea de código, asignando los datos copiados a un nuevo diccionario llamado `fts` aquí:

```
fts = {convert2ampm(k): v.title() for k, v in flights.items()}
```

La segunda manipulación (enumerando los tiempos de vuelo por destino) es un poco más complicada. Hay un poco más de trabajo debido al hecho de que las manipulaciones de datos son más complejas:



Antes de comenzar a trabajar en la segunda manipulación, vamos a hacer una pausa para ver un poco cómo el material de comprensión está penetrando en su cerebro.

Te han encargado transformar los tres bucles de esta página en comprensiones. Como usted lo hace, no olvide probar su código en IDLE (antes de voltear la página y mirar a escondidas nuestras soluciones). De hecho, antes de intentar escribir las comprensiones, ejecute estos bucles y vea lo que hacen. Escriba sus soluciones de comprensión en los espacios proporcionados.

1

```
data = [ 1, 2, 3, 4, 5, 6, 7, 8 ]
evens = []
for num in data:
    if not num % 2: ←
        evens.append(num)
```

The % operator is Python's modulo operator, which works as follows: given two numbers, divide the first by the second, then return the remainder.

2

```
data = [ 1, 'one', 2, 'two', 3, 'three', 4, 'four' ]
words = []
for num in data:
    if isinstance(num, str): ←
        words.append(num)
```

The "isinstance" BIF checks to see whether a variable refers to an object of a certain type.

3

```
data = list('So long and thanks for all the fish'.split())
title = []
for word in data:
    title.append(word.title())
```

Traducimos:

- El operador% es el operador del módulo Python, que funciona de la siguiente manera: dado dos números, divida el primero por segundo dato, luego devuelva el resto.
- El "isinstance" BIF comprueba si una variable se refiere a un objeto de cierto tipo.



Tú cogiste el lápiz y desplegaste la gorra de pensar. Para cada uno de estos tres bucles for, se encargó de transformarlos en comprensiones, asegurándose de probar su código en IDLE.

1

```
data = [ 1, 2, 3, 4, 5, 6, 7, 8 ]
evens = []
for num in data:
    if not num % 2:
        evens.append(num)
```

These four lines of loop code (which populate "evens") become one line of comprehension.

*evens = [num for num in data if not num % 2]*

2

```
data = [ 1, 'one', 2, 'two', 3, 'three', 4, 'four' ]
words = []
for num in data:
    if isinstance(num, str):
        words.append(num)
```

Again, this four-line loop is reworked as a one-line comprehension.

*words = [num for num in data if isinstance(num, str)]*

3

```
data = list('So long and thanks for all the fish'.split())
title = []
for word in data:
    title.append(word.title())
```

You should find this one the easiest of the three (as it contains no filter).

*title = [word.title() for word in data]*

Traducimos:

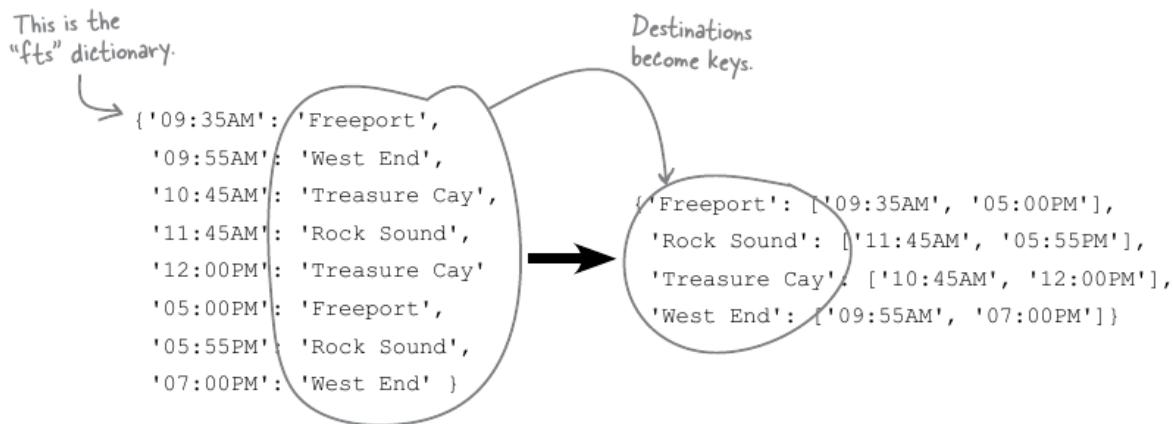
- Estas cuatro líneas de código de bucle (que pueblan "evans") se convierten en una línea de comprensión.
- Una vez más, este bucle de cuatro líneas se vuelve a trabajar como una comprensión de una línea. Una vez más, este bucle de cuatro líneas se vuelve a trabajar como una comprensión de una línea.
- Usted debe encontrar este uno de los más fáciles de los tres (ya que no contiene ningún filtro).

De al with Comple xit y the Python Way

## Tratar con la complejidad de la forma Python

Con su sesión de práctica de comprensión detrás de usted, vamos a experimentar en el prompt >>> para averiguar qué tiene que pasar con los datos en el diccionario `fts` para transformarlo en lo que se requiere.

Antes de escribir cualquier código, eche un vistazo a la transformación requerida. Observe cómo las claves del nuevo diccionario (a la derecha) son una lista de destinos únicos tomados de los valores del diccionario `fts` (a la izquierda):



Resulta que producir esos cuatro destinos únicos es muy sencillo. Dado que usted tiene los datos a la izquierda en un diccionario llamado `fts`, puede acceder a todos los valores utilizando `fts.values`, a continuación, alimentar a que el conjunto BIF para eliminar duplicados. Vamos a almacenar los destinos únicos en una variable llamada `dests`:

Coge todos los valores en "fts", a continuación, alimentar a los "set" BIF Esto le proporciona los datos que necesita.

```
>>> dests = set(fts.values())
>>> print(dests)
{'Freeport', 'West End', 'Rock Sound', 'Treasure Cay'}
```

Estos son los cuatro destinos únicos, que puede utilizar como las claves del nuevo diccionario.

Ahora que usted tiene una manera de conseguir los destinos únicos, es hora de agarrar los tiempos de vuelo asociados con esos destinos. Estos datos también se encuentran en el diccionario `fts`.

Antes de girar la página, piense en cómo podría extraer los tiempos de vuelo dados cada destino único.

De hecho, no se preocupe por extraer todos los tiempos de vuelo para cada destino; Sólo tienes que averiguar cómo hacerlo para West End primero.

west end only

Extract a Single Destination's Flight Times

### **Extremo oeste solamente**

### **Extraer los tiempos de vuelo de un solo destino**

Empecemos por extraer los datos de tiempo de vuelo para un solo destino, es decir, West End. Estos son los datos que necesita extraer:

```
{'09:35AM': 'Freeport',
 '09:55AM': 'West End',
 '10:45AM': 'Treasure Cay',
 '11:45AM': 'Rock Sound',
 '12:00PM': 'Treasure Cay',
 '05:00PM': 'Freeport',
 '05:55PM': 'Rock Sound',
 '07:00PM': 'West End'}
```

traducción:

Es necesario activar estas keys en una lista de valores.

Como antes, levante el prompt >>> y vaya al trabajo. Dado el diccionario fts, puede extraer los tiempos de vuelo de West End utilizando un código como este:

```
>>> wests = []
>>> for k, v in fts.items():
    if v == 'West End':
        wests.append(k)
>>> print(wests)
['09:55AM', '07:00PM']
```

Traducir:

- 1. Comience con una nueva lista vacía.
- 2. Extraiga las claves y los valores del diccionario "fts".
- 3. Filtre los datos en el destino "West End".
- 4. Agregue los tiempos de vuelo "West End" a la lista de "wests".

- ¡Funcionó! Aquí tienes los datos que necesitas.

Al ver este código, debe oír pequeñas campanas de alarma sonando en su cerebro, ya que este bucle es sin duda un candidato para volver a trabajar como una lista de comprensión, ¿verdad?

Lo que era cuatro líneas de código es ahora uno, gracias a su uso de un listcomp.

```
>>> wests2 = [k for k, v in fts.items() if v == 'West End']

>>> print(wests2)
['09:55AM', '07:00PM']
```

¡También funcionó!  
Aquí tienes los  
datos que necesitas.

**Ahora que sabe cómo extraer estos datos para un destino específico, vamos a hacerlo para todos los destinos.**

### **Extract Flight Times for All Destinations**

### **Extraiga los horarios de vuelo para todos los destinos**

Ahora tiene este código, que extrae el conjunto de destinos únicos:

```
dests = set(fts.values()) ← The unique destinations
```

Y también tiene esta lista listcomp, que extrae la lista de tiempos de vuelo para un destino determinado (en este ejemplo, ese destino es West End):

```
wests2 = [k for k, v in fts.items() if v == 'West End'] ← Los tiempos de vuelo para el destino "West End"
```

Para extraer la lista de tiempos de vuelos para todos los destinos, debe combinar estas dos instrucciones (dentro de un bucle for).

En el código que sigue, hemos dispensado la necesidad de los dests y las variables west2, prefiriendo usar el código directamente como parte del bucle for. Ya no somos codificadores de West End, ya que el destino actual está en dest (dentro del listcomp):

```

>>> for dest in set(fts.values()):
    print(dest, '->', [k for k, v in fts.items() if v == dest])

```

The unique destinations

Treasure Cay -> ['10:45AM', '12:00PM']  
West End -> ['07:00PM', '09:55AM']  
Rock Sound -> ['05:55PM', '11:45AM']  
Freeport -> ['09:35AM', '05:00PM']

The flight times for the destination referred to by the current value of "dest".

traducir: Los tiempos de vuelo para el destino referido por el valor actual de "dest".

El hecho de que acabamos de escribir un bucle para que aparezca conforme a nuestro patrón de comprensión comienza la pequeña campana de nuestro cerebro que vuelve a sonar. Vamos a tratar de suprimir ese timbre por ahora, como el código que acaba de experimentar con su prompt >>> muestra los datos que necesitamos ... pero lo que realmente necesita es almacenar los datos en un nuevo diccionario. Vamos a crear un nuevo diccionario (llamado when) para mantener estos datos recién extraídos. Regrese a su solicitud >>> y ajuste el bucle anterior para usarlo cuando:

```

1. Start with
a new, empty
dictionary.
>>> when = {}

2. Extract the unique
set of destinations.
>>> for dest in set(fts.values()):
    when[dest] = [k for k, v in fts.items() if v == dest]

3. Update the
"when" dictionary
with the flight
times.
>>> pprint.pprint(when)
{'Freeport': ['09:35AM', '05:00PM'],
 'Rock Sound': ['05:55PM', '11:45AM'],
 'Treasure Cay': ['10:45AM', '12:00PM'],
 'West End': ['07:00PM', '09:55AM']}

```

Here it is: the data you need, in a dictionary called "when".

traduciendo:

- 1. Comience con un nuevo diccionario vacío.
- 2. Extraer el conjunto único de destinos.
- 3. Actualizar el diccionario "when" con los tiempos de vuelo.
- Aquí está: los datos que necesita, en un diccionario llamado "when".

Si usted es como nosotros, su pequeña campana del cerebro (que usted ha estado intentando suprimir) es probable que suena fuerte y conducirle loco mientras que usted mira este código.

gotta love comprehensions  
That Feeling You Ge t...

**Tengo que amar comprensiones**

**Esa sensación que usted consigue ...**

... cuando una sola línea de código comienza a parecerse a la magia. Apague su campana del cerebro, luego eche otra mirada al código que compone su bucle más reciente:

```
when = {}
for dest in set(fts.values()):
    when[dest] = [k for k, v in fts.items() if v == dest]
```

Este código se ajusta al patrón que lo convierte en un blanco potencial para volver a trabajar como comprensión. Esto es lo anterior para el código de bucle reelaborado como un dictcomp que extrae una copia de los datos que necesita en un nuevo diccionario llamado when2:

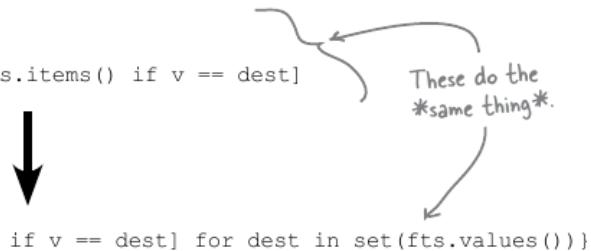
```
when2 = {dest: [k for k, v in fts.items() if v == dest] for dest in set(fts.values())}
```

Parece mágica, ¿no?

Esta es la comprensión más compleja que has visto hasta ahora, debido principalmente al hecho de que el dictcomp exterior contiene un listcomp interior. Dicho esto, este dictcomp muestra una de las características que establecen las comprensiones aparte del equivalente para el código de bucle: puede poner una comprensión casi en cualquier parte de su código. Lo mismo no ocurre para los bucles for, que sólo pueden aparecer como declaraciones en su código (es decir, no como parte de las expresiones).

Por supuesto, eso no quiere decir que siempre debe hacer algo como esto:

```
when = {}
for dest in set(fts.values()):
    when[dest] = [k for k, v in fts.items() if v == dest]
when2 = {dest: [k for k, v in fts.items() if v == dest] for dest in set(fts.values())}
```



Tenga en cuenta: una comprensión del diccionario que contiene una comprensión de la lista incrustada es difícil de leer la primera vez que lo vea.

Sin embargo, con la exposición repetida, las comprensiones se hacen más fáciles de leer y entender, y como se dijo anteriormente en este capítulo, los programadores de Python las usan mucho. Si usted utiliza comprensiones depende de usted. Si estás más contento con el código de bucle for, usa eso. Si te gusta la apariencia de comprensiones, úsalas ... simplemente no sientas que tienes que hacerlo.

## —Test Drive—

Antes de seguir adelante, pongamos todo este código de comprensión en nuestro archivo `do_convert.py`. Entonces podemos ejecutar el código en este archivo (usando IDLE) para ver que las conversiones y transformaciones que son requeridas por Bahamas Buzzers están ocurriendo como sea necesario. Confirme que su código es el mismo que el nuestro, luego ejecute el código para confirmar que todo funciona según las especificaciones.



```

do_convert.py - /Users/paul/Desktop/_NewBook/ch12/do_convert.py (3.5.2)
from datetime import datetime
import pprint

def convert2ampm(time24: str) -> str:
    return datetime.strptime(time24, '%H:%M').strftime('%I:%M%p')

with open('buzzers.csv') as data:
    ignore = data.readline()
    flights = {}
    for line in data:
        k, v = line.strip().split(',')
        flights[k] = v

pprint.pprint(flights)
print()

fts = {convert2ampm(k): v.title() for k, v in flights.items()}

pprint.pprint(fts)
print()

when = [dest: [k for k, v in fts.items() if v == dest] for dest in set(fts.values())]

pprint.pprint(when)
print()

```

1. The original raw data, as read in from the CSV data file. This is "flights".

2. The raw data, copied and transformed into AM/PM format and Titlecase. This is "fts".

3. The list of flight times per destination (extracted from "fts"). This is "when".

you are here ➤ 503

Traducimos:

- 1. Los datos originales sin procesar, como se leen desde el archivo de datos CSV. Esto es "vuelos".
- 2. Los datos brutos, copiados y transformados en formato AM / PM y Titlecase. Esto es "fts".
- 3. La lista de tiempos de vuelo por destino (extraída de "fts"). Esto es "when".

## there are no Dumb Questions

P: Así que ... déjame aclarar esto: ¿una comprensión es sólo una abreviatura sintáctica para una construcción de bucle estándar?

R: Sí, específicamente el bucle for. Un estándar para el bucle y su comprensión equivalente hacen lo mismo. Es sólo que la comprensión tiende a ejecutarse mucho más rápido.

P: ¿Cuándo sabré cuándo utilizar la comprensión de la lista?

R: No hay reglas duras y rápidas aquí. Por lo general, si está produciendo una nueva lista de una existente, tenga una buena mirada en su código de bucle. Pregúntese si el bucle es un candidato para la conversión a una comprensión equivalente. Si la nueva lista es "temporal" (es decir, se utiliza una vez, luego se desecha), pregúntese si una comprensión de la lista incrustada sería mejor para la tarea a mano. Como regla general, debe evitar la introducción de variables temporales en su código, especialmente si sólo se utilizan una vez. Pregúntese si en su lugar puede utilizarse una comprensión.

P: ¿Puedo evitar las comprensiones por completo?

R: Sí, usted puede. Sin embargo, tienden a ver un poco de uso dentro de la comunidad más amplia de Python, así que a menos que su plan sea nunca mirar el código de otra persona, sugerimos tomar el tiempo para familiarizarse con la tecnología de comprensión de Python. Una vez que te acostumbras a verlos, te preguntarás cómo viviste sin ellos. ¿Mencionamos que son rápidos?

P: Sí, lo entiendo, pero ¿la velocidad es tan grande hoy en día? Mi computadora portátil es super rápido y se ejecuta mi para bucles lo suficientemente rápido.

R: Es una observación interesante. Es verdad que hoy tenemos computadoras que son mucho más poderosas que cualquier cosa que haya venido antes. También es cierto que pasamos mucho menos tiempo tratando de agotar cada último ciclo de la CPU de nuestro código (porque, seamos realistas: ya no tenemos que hacerlo). Sin embargo, cuando se presenta con una tecnología que ofrece un impulso de rendimiento, ¿por qué no utilizarlo? Es un poco de esfuerzo para un gran retorno en el rendimiento.

*Me parece una taza de café muy fuerte (con un poco de algo en ella) me ayuda a conseguir mi cabeza alrededor de la mayoría de las comprensiones. Por cierto, ¿trabajan con conjuntos y tuplas?*

### ***Esa es una gran pregunta.***

Y la respuesta es: sí y no. Sí, es posible crear y utilizar un conjunto de comprensión (aunque, para ser honesto, se encontrará con ellos muy rara vez). Y, no, no hay tal cosa como una "comprensión de la tupla." Vamos a llegar a por qué esto es después de que hemos demostrado que establecer comprensiones en acción.

## ***The Set Comprehension in Action***

### ***La comprensión del conjunto en acción***

Un conjunto de comprensiones (o setcomp para abreviar) le permite crear un nuevo conjunto en una línea de código, utilizando una construcción que es muy similar a la sintaxis de comprensión de la lista.

Lo que establece un setcomp aparte de un listcomp es que la comprensión del conjunto está rodeada de llaves (a diferencia de los corchetes alrededor de un listcomp). Esto puede ser confuso, ya que los dictcomps están rodeados de llaves, también. (Uno se pregunta qué vino sobre los reveladores de la base de Python cuando decidieron hacer esto.)

Un conjunto literal está rodeado de llaves, al igual que los diccionarios literales. Para distinguirlos, busque el carácter de dos puntos que se utiliza como delimitador en los diccionarios, ya que los dos puntos no tienen significado en los conjuntos. El mismo consejo se aplica para determinar rápidamente si una comprensión rizada es un dictcomp o un setcomp: busque el colon. Si está allí, usted está mirando un dictcomp. Si no, es un setcomp.

Aquí hay un ejemplo de comprensión rápida (que se remite a un ejemplo anterior en este libro). Dado un conjunto de letras (en vocales), y una cadena (en mensaje), el bucle for así como su conjunto equivalente produce el mismo resultado: un conjunto de vocales encontradas en el mensaje:

```
vowels = {'a', 'e', 'i', 'o', 'u'}
message = "Don't forget to pack your towel."

found = set()
for v in vowels:
    if v in message:
        found.add(v)
```

The setcomp follows  
the same pattern as  
the listcomp.



```
found2 = { v for v in vowels if v in message }
```

Note the use of curly braces here,  
as this comprehension produces a set  
when executed by the interpreter

traducimos:

- El setcomp sigue el mismo patrón que el listcomp.
- Tenga en cuenta el uso de llaves aquí como esta comprensión produce un conjunto cuando se ejecuta por el intérprete.

Tómese unos momentos para experimentar con el código de esta página en su solicitud >>>. Debido a que ya sabes lo que puede hacer listcomps y dictcomps, conseguir su cabeza conjunto set comprehensions no es tan complicado. Realmente no hay nada más que lo que hay en esta página.

### ***Comprender esa comprensión***

### ***Cómo detectar una comprensión***

A medida que se familiariza con la apariencia del código de comprensión, se vuelven más fáciles de detectar y entender. Esta es una buena regla general para detectar las comprensiones de la lista:

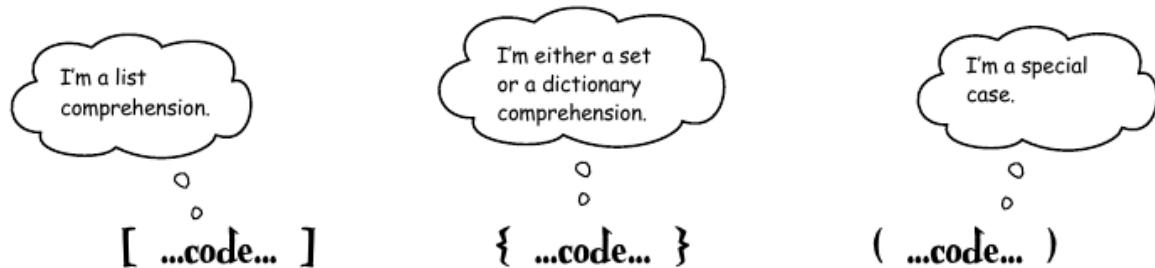
***Si localiza el código rodeado por [y], entonces está buscando una comprensión de la lista.***

Esta regla puede generalizarse de la siguiente manera:

***Si localiza el código rodeado por paréntesis (rizado o cuadrado), entonces es probable que busque una comprensión.***

¿Por qué el uso de la palabra "probable"? Además de que el código está rodeado por [], las comprensiones también pueden estar rodeadas por {}. Cuando el código está rodeado por [y], está buscando una comprensión de la lista. Cuando el código está rodeado por {y}, está buscando un conjunto o una comprensión del diccionario. Un dictcomp es fácil de detectar gracias a su uso del carácter de dos puntos como delimitador.

Sin embargo, el código también puede aparecer entre (y), que es un caso especial, aunque se le perdonaría por sugerir que el código rodeado de paréntesis debe ser seguramente una comprensión de la tupla. Usted sería perdonado, pero mal: "tuples comprehensions" no existen, aunque usted puede poner el código entre (y). Después de la "diversión" que ha estado teniendo con las comprensiones hasta ahora en este capítulo, puede estar pensando: ¿podría ser más extraño?



Vamos a concluir este capítulo (y este libro) explorando lo que está pasando con el código que aparece entre (y). No es una "comprensión de la tupla", pero obviamente se permite, así que ¿qué es?

### ***What About “Tuple Comprehensions”?***

### ***¿Qué pasa con “Tuple Comprehensions”?***

Las cuatro estructuras de datos integradas de Python (tuplas, listas, conjuntos y diccionarios) se pueden poner a muchos usos. Sin embargo, todos excepto las tuplas pueden ser creados a través de una comprensión.

¿Por qué es esto? Resulta que la idea de una "comprensión de tupla" realmente no tiene sentido. Recuerde que las tuplas son inmutables: una vez que se crea una tupla, no se puede cambiar. Esto también significa que no es posible generar los valores de una tupla en el código, como muestra esta breve sesión IDLE:

A screenshot of the Python 3.5.2 Shell window. The code entered is:

```
>>> names = ()
>>> for n in ('John', 'Paul', 'George', 'Ringo'):
...     names.append(n)
```

Annotations explain the behavior:

- An arrow points to the first line: "Create a new, empty tuple."
- An arrow points to the line `names.append(n)`: "Try to dynamically add data to the tuple."
- An arrow points to the error message: "You can't append to an existing tuple, as it is immutable."

Details from the shell window:

- Line: 44 Col: 4
- Traceback (most recent call last):
 File "<pyshell#17>", line 2, in <module>
 names.append(n)
 AttributeError: 'tuple' object has no attribute 'append'

No hay nada extraño o maravilloso pasando aquí, ya que este es el comportamiento esperado de las tuplas: una vez que existe, no se puede cambiar. Este hecho solo debería ser suficiente para descartar el uso de una tupla dentro de cualquier tipo de comprensión. Pero eche un vistazo a esta interacción en el prompt >>>. El segundo bucle se diferencia del

primero en el más pequeño de los modos: los corchetes alrededor del listcomp (en el primer bucle) han sido reemplazados por paréntesis (en el segundo):

What gives? Both loops generate the same results.

This for loop and list comprehension combination displays each of the list's values tripled. You know this is a listcomp, as that's code inside square brackets.

But look at this. The parentheses makes this look like a "tuple comprehension"—but you know such a thing is impossible. Yet the loop still produces the expected output. Weird, eh?

traduciendo:

- ¿Lo que da? Ambos bucles generan los mismos resultados.
- Esta combinación de comprensión de bucle y lista muestra cada uno de los valores de la lista triplicados. Usted sabe que esto es un listcomp, ya que es el código dentro de corchetes.
- Pero mira esto. El paréntesis hace que esto parezca una "comprensión de la tupla", pero usted sabe que tal cosa es imposible. Sin embargo, el bucle todavía produce la salida esperada. Extraño, ¿eh?

generate your data

Parentheses Around Code == Generator

### **Genere sus datos**

### **Paréntesis alrededor del código == Generador**

Cuando te encuentras con algo que parece un listcomp pero está rodeado de paréntesis, estás mirando un generador:

This looks like a listcomp, but isn't: it's a generator.

for i in (x\*3 for x in [1, 2, 3, 4, 5]):  
 print(i)

Un generador se puede utilizar en cualquier lugar donde se use un listcomp, y produce los mismos resultados.

Como se vio en la parte inferior de la última página, cuando se reemplazan los corchetes circulares de un listcomp con paréntesis, los resultados son los mismos; Es decir, el generador y el listcomp producen los mismos datos.

Sin embargo, no se ejecutan de la misma manera.

Si se está rascando la cabeza en la frase anterior, considere esto: cuando un listcomp se ejecuta, produce todos sus datos antes de que se produzca cualquier otro procesamiento. Tomado en el contexto del ejemplo en la parte superior de esta página, el bucle for no comienza a procesar ninguno de los datos producidos por el listcomp hasta que se complete el listcomp. Esto significa que un listcomp que tarda mucho tiempo en producir datos retrasa cualquier otro código de ejecución hasta que el listcomp concluya.

Con una pequeña lista de elementos de datos (como se muestra arriba), esto no es un gran problema.

Pero imagine que su listcomp está obligado a trabajar con una lista que produce 10 millones de elementos de datos. Ahora tienes dos problemas: (1) tienes que esperar a que el listcomp procese esos 10 millones de elementos de datos antes de hacer cualquier otra cosa, y (2) tienes que preocuparte de que el ordenador que ejecuta tu listcomp tenga suficiente memoria RAM para contener todos los datos en la memoria mientras se ejecuta el listcomp (10 millones de piezas individuales de datos). Si su listcomp se queda sin memoria, el intérprete termina (y su programa es tostado).

***Listcomps y generadores producen los mismos resultados, pero operan de una manera muy diferente.***

### ***Los generadores producen elementos de datos uno a la vez ...***

Cuando reemplaza los corchetes de su listcomp con paréntesis, el listcomp se convierte en un generador y su código se comporta de manera diferente.

A diferencia de un listcomp, que debe concluir antes de que cualquier otro código pueda ejecutarse, un generador libera datos tan pronto como los datos son producidos por el código del generador. Esto significa que si usted genera 10 millones de elementos de datos, el intérprete sólo necesita suficiente memoria para contener un elemento de datos (a la vez) y cualquier código que esté esperando para consumir los elementos de datos producidos por el generador se ejecuta inmediatamente; Es decir, no hay espera.

No hay nada como un ejemplo para entender la diferencia que usa un generador, así que realizemos una tarea simple dos veces: una vez con un listcomp, luego de nuevo con un generador.

## Using a Listcomp to Process URLs

### Utilizar un Listcomp para procesar URL

Para demostrar la diferencia usando un generador puede hacer, vamos a realizar una tarea usando un listcomp (antes de volver a escribir la tarea como un generador).

Como ha sido nuestra práctica a lo largo de este libro, vamos a experimentar con algún código en el prompt >>> que utiliza la biblioteca de peticiones (que le permite interactuar mediante programación con la Web). Esta es una pequeña sesión interactiva que importa la biblioteca de peticiones, define una tupla de tres elementos (urls) y luego combina un bucle for con un listcomp para solicitar la página de destino de cada URL antes de procesar la respuesta web devuelta.

Para entender lo que está pasando aquí, usted necesita seguir a lo largo de su computadora.

The screenshot shows a Python 3.5.2 Shell window. The code is as follows:

```
>>>
>>> import requests
>>>
>>> urls = ('http://headfirstlabs.com', 'http://oreilly.com', 'http://twitter.com')
>>>
>>> for resp in [requests.get(url) for url in urls]:
...     print(len(resp.content), '>', resp.status_code, '>', resp.url)
```

The output is:

```
31590 > 200 > http://headfirstlabs.com/
78722 > 200 > http://www.oreilly.com/
128244 > 200 > https://twitter.com/
>>> |
```

Annotations with arrows pointing to specific parts of the code and output:

- An annotation points to the line `urls = ('http://headfirstlabs.com', 'http://oreilly.com', 'http://twitter.com')` with the text: "Define a tuple of URLs. Feel free to substitute your own URLs here. Just be sure to define at least three."
- An annotation points to the list comprehension part of the loop with the text: "The 'for' loop contains a listcomp, which, for each of the URLs in 'urls', gets the website's landing page."
- An annotation points to the output line `31590 > 200 > http://headfirstlabs.com/` with the text: "Nothing weird or wonderful here. The output produced is exactly what's expected."
- An annotation points to the status code part of the output with the text: "With each response received, display the size of the returned landing page (in bytes), the HTTP status code, and the URL used."

Traducimos:

- Definir una tupla de URL. Siéntase libre de sustituir sus propias URL aquí. Sólo asegúrese de definir al menos tres.
- El bucle "for" contiene un listcomp, que, para cada una de las URL en "urls", obtiene la página de destino del sitio web.
- Nada raro o maravilloso aquí. La producción producida es exactamente lo que se espera.
- Con cada respuesta recibida, muestre el tamaño de la página de destino devuelta (en bytes), el código de estado HTTP y la URL utilizada.

### **Descargar "requests" de PyPI utilizando el comando "pip".**

Si sigue a lo largo de su computadora, experimentará un retraso notable entre entrar en el código de bucle for y ver los resultados. Cuando aparecen los resultados, se muestran de una sola vez (todos a la vez). Esto se debe a que el listcomp funciona a través de cada una de las URL en la tupla de urls antes de hacer cualquier resultado disponible para el bucle for. ¿El resultado? Usted tiene que esperar para su salida.

Tenga en cuenta que no hay nada malo en este código: hace lo que quiere, y la salida es correcta. Sin embargo, vamos a volver a trabajar este listcomp como un generador para ver la diferencia que hace. Como se mencionó anteriormente, asegúrese de seguir a lo largo de su computadora a medida que trabaja a través de la siguiente página (para que pueda ver lo que sucede).

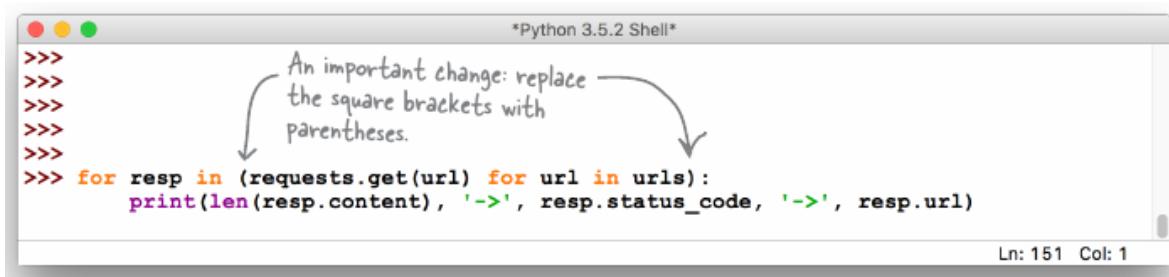
*gotta love generators*

### **Using a Generator to Process URLs**

*Tengo que amar generadores*

### **Uso de un generador para procesar URL**

Aquí está el ejemplo de la última página reelaborada como un generador. Hacerlo es fácil; Simplemente reemplace los corchetes del listcomp con paréntesis:



The screenshot shows a Python 3.5.2 Shell window. The code is as follows:

```
>>>
>>>
>>>
>>>
>>> An important change: replace
>>> the square brackets with
>>> parentheses.
>>> for resp in (requests.get(url) for url in urls):
>>>     print(len(resp.content), '->', resp.status_code, '->', resp.url)
```

A callout bubble points to the line `for resp in (requests.get(url) for url in urls):` with the text: "An important change: replace the square brackets with parentheses." The status bar at the bottom right of the shell window shows "Ln: 151 Col: 1".

traducción:

- Un cambio importante: reemplace los corchetes con paréntesis.

Un breve momento después de entrar en el loop anterior, aparece el primer resultado:

```
*Python 3.5.2 Shell*
>>>
>>>
>>> for resp in (requests.get(url) for url in urls):
    print(len(resp.content), '->', resp.status_code, '->', resp.url)

31590 -> 200 -> http://headfirstlabs.com/ ← The first URL's response
|                                              Ln: 153 Col: 0
```

Luego, un momento después, aparecerá la siguiente línea de resultados:

```
*Python 3.5.2 Shell*
>>>
>>> for resp in (requests.get(url) for url in urls):
    print(len(resp.content), '->', resp.status_code, '->', resp.url)

31590 -> 200 -> http://headfirstlabs.com/
78722 -> 200 -> http://www.oreilly.com/ ← The second URL's response
|                                              Ln: 154 Col: 0
```

Entonces-finalmente-unos momentos después, aparece la última línea de resultados (y termina el bucle for):

```
Python 3.5.2 Shell
>>> for resp in (requests.get(url) for url in urls):
    print(len(resp.content), '->', resp.status_code, '->', resp.url)

31590 -> 200 -> http://headfirstlabs.com/
78722 -> 200 -> http://www.oreilly.com/
128244 -> 200 -> https://twitter.com/ ← The third, and final, URL's response
>>> |
|                                              Ln: 156 Col: 4
```

### ***Using a Generator: What Just Happened?***

### ***Usando un generador: ¿Qué acaba de suceder?***

Si comparas los resultados producidos por tu listcomp con los producidos por tu generador, son idénticos. Sin embargo, el comportamiento de su código no lo es.

El listcomp espera que todos sus datos se produzcan antes de alimentar cualquier dato al bucle de espera, mientras que el generador libera datos tan pronto como esté disponible. Esto significa que el bucle for que utiliza el generador es mucho más sensible, en contraposición al listcomp (que te hace esperar).

Si estás pensando que esto no es realmente tan grande un trato, imagínese si la tupla de URL se definió con cien, mil, o un millón de URL. Además, imagine que el código que procesa la respuesta está alimentando los datos procesados a otro proceso (tal vez una base de datos en espera). A medida que aumenta el número de URL, el comportamiento del listcomp empeora en comparación con el del generador.

*Así que ... ¿esto significa que siempre debería usar un generador sobre un listcomp?*

#### **No. No diríamos eso.**

No malinterpreten: el hecho de que los generadores existen es grande, pero esto no significa que usted querrá reemplazar todos sus listcomps con un generador equivalente. Como un montón de cosas en la programación, el enfoque que utiliza depende de lo que está tratando de hacer.

Si usted puede permitirse el lujo de esperar, entonces listcomps están bien; De lo contrario, considere el uso de un generador.

Un uso interesante de los generadores es integrarlos dentro de una función. Echemos un vistazo a encapsular su generador creado en una función.

#### ***generator functions rock***

#### ***Define What Your Function Needs to Do***

#### ***Funciones del generador rock***

#### ***Definir lo que su función necesita hacer***

Imaginemos que usted quiere tomar su generador de peticiones y convertirlo en una función. Ha decidido empaquetar el generador dentro de un pequeño módulo que está escribiendo, y desea que otros programadores puedan utilizarlo sin tener que conocer o entender a los generadores.

Aquí está su código del generador una vez más:

```

import requests Import any required libraries.

urls = ('http://headfirstlabs.com', 'http://oreilly.com', 'http://twitter.com')

for resp in (requests.get(url) for url in urls):
    print(len(resp.content), '->', resp.status_code, '->', resp.url)

```

*Define a tuple of URLs.*

*Process the generated data.*

*The generator (remember: looks like a listcomp, but is surrounded by parentheses)*

Traducimos:

- Importe todas las bibliotecas necesarias.
- Definir una tupla de URL.
- Procesar los datos generados.
- El generador (recuerda: se parece a un listcomp, pero está rodeado de paréntesis)

Vamos a crear una función que encapsula este código. La función, que se denomina `gen_from_urls`, toma un único argumento (una tupla de URL) y devuelve una tupla de resultados para cada URL. La tupla devuelta contiene tres valores: la longitud del contenido de la URL, el código de estado HTTP y la URL de la que procede la respuesta.

Suponiendo que `gen_from_urls` existe, desea que otros programadores puedan ejecutar su función como parte de un bucle `for`, como esto:

```

from url_utils import gen_from_urls Import the function from your module.

urls = ('http://headfirstlabs.com', 'http://oreilly.com', 'http://twitter.com')

for resp_len, status, url in gen_from_urls(urls): Call the function on each iteration of the "for" loop.
    print(resp_len, status, url)

```

*Define a tuple of URLs.*

*Process the data.*

Traducimos:

- Definir una tupla de URL.
- Importe la función de su módulo.
- Procesar los datos.
- Llame a la función en cada iteración del bucle "for".

Aunque este nuevo código no parece tan diferente del código de la parte superior de la página, tenga en cuenta que los programadores que utilizan `gen_from_urls` no tienen ni idea (ni necesitan saber) que están utilizando solicitudes para hablar con la Web. Tampoco

necesitan saber que está usando un generador. Todos los detalles y opciones de implementación están ocultos detrás de esa llamada de función fácil de entender.

Veamos qué implica escribir gen\_from\_urls para que pueda generar los datos que necesita.

### ***Yield to the Power of Generator Functions***

#### ***Rendimiento a la potencia de las funciones del generador***

Ahora que ya sabes lo que tiene que hacer la función gen\_from\_urls, vamos a escribirla. Comience creando un nuevo archivo llamado url\_utils.py. Edite este archivo, luego agregue peticiones de importación como su primera línea de código.

La línea de def de la función es sencilla, ya que toma una única tupla en el camino y devuelve una tupla en la salida (observe cómo hemos incluido anotaciones de tipo para hacer esto explícito para los usuarios de nuestra función de generador). Adelante y agregue la línea def de la función al archivo, así:

```
import requests  
  
def gen_from_urls(urls: tuple) -> tuple:
```



Después de importar "solicitudes" o "requests", defina su nueva función.

La suite de la función es el generador de la última página, y la línea for es una simple copia y pega:

```
import requests  
  
def gen_from_urls(urls: tuple) -> tuple:  
    for resp in (requests.get(url) for url in urls):
```



Agregue su línea de bucle "for" con el generador.

La siguiente línea de código necesita "devolver" el resultado de esa solicitud GET tal y como la realiza la función requests.get. Aunque es tentador agregar la siguiente línea como la suite de for, por favor no haga esto:

```
return len(resp.content), resp.status_code, resp.url
```



Cuando una función ejecuta una instrucción return, la función termina. No quiere que esto suceda aquí, ya que la función gen\_from\_urls se llama como parte de un bucle for, que está esperando una tupla diferente de resultados cada vez que se llama a la función.

Pero, si no puedes ejecutar el retorno, ¿qué debes hacer?

Utilice el rendimiento o **yield** en su lugar. La palabra clave **yield** se añadió a Python para soportar la creación de funciones de generador, y se puede usar en cualquier lugar donde se use una devolución. Cuando lo hace, su función se transforma en una función de generador que puede ser "llamada" desde cualquier iterador, que, en este caso, es desde dentro de su bucle for:

```
import requests

def gen_from_urls(urls: tuple) -> tuple:
    for resp in (requests.get(url) for url in urls):
        yield len(resp.content), resp.status_code, resp.url
```

Utilice "yield" para devolver cada línea de resultados de la respuesta GET al bucle de espera "for". Recuerde: NO use "return".

Echemos un vistazo más de cerca a lo que está sucediendo aquí.

*generator functions at work*

## **Tracing Your Generator Function, 1 of 2**

*Funciones del generador en el trabajo*

### **Seguimiento de la función del generador, 1 de 2**

Para entender lo que sucede cuando se ejecuta la función de generador, vamos a rastrear la ejecución del siguiente código:

```
from url_utils import gen_from_urls

urls = ('http://talkpython.fm', 'http://pythonpodcast.com', 'http://python.org')

for resp_len, status, url, in gen_from_urls(urls):
    print(resp_len, '->', status, '->', url)
```

Import your generator function.

Define a tuple of URLs.

Use your generator function as part of a "for" loop.

traduciendo:

- Importe su función de generador.
- Definir una tupla de URL.
- Utilice su función de generador como parte de un bucle "for".

Las dos primeras líneas de código son bastante simples: la función se importa y se define una tupla de URL.

La diversión comienza en la siguiente línea de código, cuando se invoca la función generador `gen_from_urls`. Hagamos referencia a esto para el bucle como "el código llamante":

```
for resp_len, status, url, in gen_from_urls(urls): ←  
    El bucle de código de  
    llamada "for" se comunica  
    con las funciones del  
    generador "for".
```

El intérprete salta a la función `gen_from_urls` y comienza a ejecutar su código. La tupla de URL se copia en el único argumento de la función, y luego se ejecuta la función del generador para loop:

```
def gen_from_urls(urls: tuple) -> tuple:  
    for resp in (requests.get(url) for url in urls): ←  
        yield len(resp.content), resp.status_code, resp.url  
    El bucle de código de  
    llamada "for" se comunica  
    con las funciones del  
    generador "for".
```

El bucle for contiene el generador, que toma la primera URL en la tupla de urls y envía una solicitud GET al servidor identificado. Cuando se devuelve la respuesta HTTP desde el servidor, se ejecuta la declaración de rendimiento o `yield`.

Aquí es donde las cosas se ponen interesantes (o extraño, dependiendo de su punto de vista).

En lugar de ejecutar, pasar a la siguiente URL en la tupla de urls (es decir, continuar con la siguiente iteración de `gen_from_urls` para loop), el rendimiento `yield` pasa sus tres piezas de datos al código de llamada. En lugar de terminar, el generador de funciones `gen_from_urls` ahora espera, como si estuviera en una animación suspendida ...

## ***Tracing Your Generator Function, 2 of 2***

### ***Seguimiento de la función del generador, 2 de 2***

Cuando los datos (como devueltos por rendimiento) llegan al código de llamada, se ejecuta el conjunto de bucle for. Como la suite contiene una sola llamada al BIF de impresión, esa línea de código se ejecuta y muestra los resultados de la primera URL en la pantalla:

```
print(resp_len, '->', status, '->', url)
```

```
34591 -> 200 -> https://talkpython.fm/
```

El código de llamada para el bucle entonces itera, llamando `gen_from_urls` otra vez ... ordenarlo.

Esto es casi lo que sucede. Lo que realmente sucede es que `gen_from_urls` se despierta de su animación suspendida, y luego continúa ejecutándose. El bucle `for` dentro de `gen_from_urls` itera, toma la siguiente URL de la tupla de `urls` y contacta al servidor asociado con la URL. Cuando se devuelve la respuesta HTTP desde el servidor, se ejecuta la instrucción de rendimiento, pasando sus tres datos al código de llamada (que la función accede a través del objeto `resp`):

```
yield len(resp.content), resp.status_code, resp.url
```

Las tres piezas obtenidas de datos se toman del objeto "resp" devuelto por el método "get" de la biblioteca "requests".

Como antes, en lugar de terminar, la función generador `gen_from_urls` ahora espera una vez más, como si estuviera en una animación suspendida ...

Cuando los datos (como devueltos por rendimiento `yield`) llegan al código de llamada, la suite del bucle `for` ejecuta la impresión una vez más, mostrando el segundo conjunto de resultados en la pantalla:

```
34591 -> 200 -> https://talkpython.fm/
19468 -> 200 -> http://pythonpodcast.com/
```

El código de llamada para el bucle itera, "llamando" `gen_from_urls` una vez más, lo que da como resultado que su función de generador se vuelve a despertar. La instrucción de rendimiento `yield` se ejecuta, los resultados se devuelven al código de llamada y la pantalla se actualiza de nuevo:

```
34591 -> 200 -> https://talkpython.fm/
19468 -> 200 -> http://pythonpodcast.com/
47413 -> 200 -> https://www.python.org/
```

En este punto, has agotado la tupla de URL, por lo que la función del generador y el código de llamada para el bucle terminan. Es como si las dos piezas de código se turnaran para ejecutar, pasando datos entre sí en cada turno.

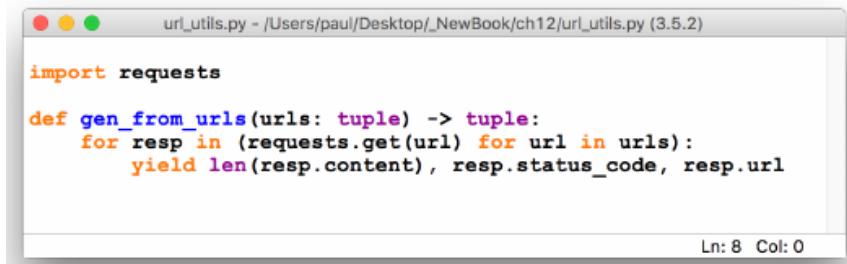
Vamos a ver esto en acción en el prompt `>>>`. Ahora es el momento para una última prueba de unidad.

*don't be sad*



**Test Drive**

En esto, el último Test Drive de este libro, vamos a tomar su función de generador para un giro. Como siempre ha sido nuestra práctica, cargue su código en una ventana de edición IDLE, luego presione F5 para ejercitarse la función en el prompt >>. Sigue con nuestra sesión (abajo):



Aquí está la función del generador "gen\_from\_urls" en el módulo "url\_utils.py".

```

import requests

def gen_from_urls(urls: tuple) -> tuple:
    for resp in (requests.get(url) for url in urls):
        yield len(resp.content), resp.status_code, resp.url

```

Ln: 8 Col: 0

El primer ejemplo a continuación muestra `gen_from_urls` que se llama como parte de un bucle `for`. Como era de esperar, la salida es la misma que se obtuvo unas pocas páginas atrás.

El segundo ejemplo a continuación muestra `gen_from_urls` que se utiliza como parte de un dictcomp. Observe cómo el nuevo diccionario sólo necesita almacenar la URL (como clave) y el tamaño de la página de destino (como el valor). El código de estado HTTP no es necesario en este ejemplo, por lo que decimos al intérprete que lo ignore utilizando el nombre de variable predeterminado de Python (que es un solo carácter de subrayado):



Each line of results appears, after a short pause, as the data is generated by the function.

This dictcomp associates the URL with the length of its landing page.

Pass the tuple of URLs to the generator function.

The underscore tells the code to ignore the yielded HTTP status code value.

Pretty-printing the "url\_res" dictionary confirms that the generator function can be used within a dictcomp (as well as within a "for" loop).

```

Python 3.5.2 Shell
>>>
>>>
>>> for resp_len, status, url in gen_from_urls(urls):
    print(resp_len, '->', status, '->', url)
31590 -> 200 -> http://headfirstlabs.com/
78722 -> 200 -> http://www.oreilly.com/
128244 -> 200 -> https://twitter.com/
>>>
>>> urls_res = {url: size for size, _, url in gen_from_urls(urls)}
>>>
>>> import pprint
>>>
>>> pprint.pprint(urls_res)
{'http://headfirstlabs.com/': 31590,
 'http://www.oreilly.com/': 78722,
 'https://twitter.com/': 128244}
>>>
>>>

```

Ln: 271 Col: 0

Traducimos:

- Cada línea de resultados aparece, después de una breve pausa, a medida que los datos son generados por la función.
- Este dictcomp asocia la URL con la longitud de su página de destino.
- Pase la tupla de URL a la función del generador.
- El carácter de subrayado le indica al código que ignore el valor de código de estado HTTP generado.
- pretty- imprimir el diccionario "url\_res" confirma que la función del generador se puede utilizar dentro de un dictcomp (así como dentro de un bucle "for").

## ***Observaciones finales***

El uso de funciones de comprensión y generador es considerado a menudo como un tema avanzado en el mundo de Python. Sin embargo, esto se debe principalmente al hecho de que estas características están ausentes de otros lenguajes de programación, lo que significa que los programadores que se mueven a Python a veces luchan con ellos (ya que no tienen punto de referencia).

Dicho esto, en Head First Labs, el equipo de programación de Python ama las comprensiones y los generadores, y cree que con la exposición repetida, especificando las construcciones de bucle que las utilizan se convierte en segunda naturaleza. No pueden imaginar tener que prescindir de ellos.

Incluso si usted encuentra la sintaxis de la comprensión y del generador extraña, nuestro consejo es pegarse con ellos. Incluso si se descarta el hecho de que son más eficaces que el equivalente para el bucle, el hecho de que usted puede utilizar comprensiones y generadores en lugares donde no se puede utilizar un bucle for es razón suficiente para tomar una mirada seria a estas características de Python. Con el tiempo, ya medida que se familiarice con su sintaxis, las oportunidades de explotar comprensiones y generadores se presentarán tan naturalmente como las que le dicen a su cerebro de programación que utilice una función aquí, un bucle allí, una clase por aquí, etc. He aquí una revisión de lo que le presentaron en este capítulo:



Cuando se trata de trabajar con datos en archivos, Python tiene opciones. Además del estándar BIF abierto, puede utilizar las funciones del módulo csv de la biblioteca estándar para trabajar con datos con formato CSV.

Las cadenas de métodos permiten realizar el procesamiento de datos en una línea de código. El string.strip (). La cadena split () se ve mucho en el código Python.

Tenga cuidado con cómo ordena sus cadenas de métodos. Específicamente, preste atención al tipo de datos devueltos de cada método (y asegúrese de que se mantiene la compatibilidad de tipo).

Un bucle for utilizado para transformar datos de un formato a otro puede ser reelaborado como una comprensión.

Las comprensiones se pueden escribir para procesar las listas existentes, los diccionarios y los conjuntos, con las comprensiones de listas siendo la variante más popular "en estado salvaje". Los programadores de Python experimentados se refieren a estas construcciones como listcomps, dictcomps y setcomps.

Un listcomp es un código rodeado de corchetes, mientras que un dictcomp es un código rodeado de llaves (con delimitadores de dos puntos). Un setcomp es también código rodeado de llaves (pero sin los dos puntos del dictcomp).

No hay tal cosa como una "comprensión de la tupla", ya que las tuplas son inmutables (por lo que no tiene sentido intentar crear dinámicamente una).

Si detecta un código de comprensión rodeado de paréntesis, está buscando un generador (que puede convertirse en una función que utiliza el rendimiento yield para generar datos según sea necesario).

Como concluye este capítulo (y, por definición, el contenido principal de este libro), tenemos una última pregunta que hacerle. Tome una respiración profunda, luego voltear la página.

## **One Final Question**

### **Una pregunta final**

DE ACUERDO. Aquí va, nuestra última pregunta para usted: en esta etapa de este libro, ¿incluso notan el uso de Python de espacios en blanco significativos?

La queja más común oída de los programadores nuevos a Python es su uso del espacio en blanco para significar bloques del código (en vez de, por ejemplo, los apoyos rizados). Pero, después de un tiempo, su cerebro tiende a no notar más. Esto no es un accidente: el uso de Python de espacios en blanco significativos fue intencional por parte del creador del lenguaje.

Esto no es un accidente: el uso de Python de espacios en blanco significativos fue intencional por parte del creador del lenguaje.

Fue deliberadamente hecho de esta manera, porque el código se lee más de lo que está escrito. Esto significa que el código que se ajusta a una apariencia consistente y conocida es más fácil de leer. Esto también significa que el código de Python escrito hace 10 años por un desconocido completo es todavía legible por usted hoy debido al uso de Python de espacios en blanco.

Esta es una gran victoria para la comunidad de Python, lo que hace que sea una gran victoria para usted, también.

## Chapter 12's Code

This is "do\_convert.py".

```
from datetime import datetime
import pprint

def convert2ampm(time24: str) -> str:
    return datetime.strptime(time24, '%H:%M').strftime('%I:%M%p')

with open('buzzers.csv') as data:
    ignore = data.readline()
    flights = {}
    for line in data:
        k, v = line.strip().split(',')
        flights[k] = v

pprint.pprint(flights)
print()

fts = {convert2ampm(k): v.title() for k, v in flights.items()}

pprint.pprint(fts)
print()

when = {dest: [k for k, v in fts.items() if v == dest] for dest in set(fts.values())}

pprint.pprint(when)
print()
```

This is "url\_utils.py".

```
import requests

def gen_from_urls(urls: tuple) -> tuple:
    for resp in (requests.get(url) for url in urls):
        yield len(resp.content), resp.status_code, resp.url
```





