Aprenda Python en un día y aprenda bien
Python para principiantes con proyecto práctico
El único libro en el que necesitas empezar a codificar Python inmediatamente

Por Jamie Chan

http://www.learncodingfast.com/python

Prefacio

Este libro está escrito para ayudarlo a aprender a programar Python RÁPIDO y apréndalo BIEN. Si es un principiante absoluto en Programación, encontrará que este libro explica conceptos complejos de una manera fácil de entender conducta. Los ejemplos se eligen cuidadosamente para demostrar cada concepto que puede obtener una comprensión más profunda del idioma. Si eres un codificador experimentado, este libro le ofrece una buena base desde la cual explorar Python. Los apéndices al final del libro también proporcionarán usted con una referencia conveniente para algunos de los funciones en Python.

Además, como dice Richard Branson: "La mejor manera de aprender sobre

todo es haciendo ". Al final del curso, se le guiará a través de
proyecto que le da la oportunidad de poner en práctica lo que ha aprendido.

Puede descargar el código fuente del proyecto y los apéndices en
http://www.learncodingfast.com/python.

Tabla de contenido

Página 5

Página 6

Página 7

Capítulo 1: Python, ¿qué Python?

Bienvenido al apasionante mundo de la programación. Estoy tan contento de que hayas recogido este libro y espero sinceramente que este libro pueda ayudarlo a dominar Python lenguaje y experimente la emoción de la programación. Antes de que nosotros sumergirse en los aspectos prácticos de la programación en Python, primero respondamos una pocas preguntas.

Página 8

¿Qué es Python?

Python es un lenguaje de programación de alto nivel ampliamente utilizado creado por Guido van Rossum a finales de los 80. El idioma coloca fuerte

énfasis en la legibilidad y la simplicidad del código, lo que hace posible
programadores para desarrollar aplicaciones rápidamente.

Como todos los lenguajes de programación de alto nivel, el código Python se parece al
Idioma inglés que las computadoras no pueden entender. Códigos que
que escribimos en Python tienen que ser interpretados por un programa especial conocido como
el intérprete de Python, que tendremos que instalar antes de poder codificar, probar
y ejecutar nuestros programas Python. Veremos cómo instalar Python
intérprete en el Capítulo 2.

También hay una serie de herramientas de terceros, como Py2exe o
Pyinstaller que nos permite empaquetar nuestro código Python en un
programas ejecutables para algunos de los sistemas operativos más populares como
Windows y Mac OS. Esto nos permite distribuir nuestros programas Python
sin requerir que los usuarios instalen el intérprete de Python.

Página 9

¿Por qué aprender Python?

Hay una gran cantidad de lenguajes de programación de alto nivel disponibles,
como C, C ++ y Java. La buena noticia es toda la programación de alto nivel.
los idiomas son muy similares entre sí. Lo que difiere es principalmente el
la sintaxis, las bibliotecas disponibles y la forma en que accedemos a esas bibliotecas. UN
biblioteca es simplemente una colección de recursos y códigos preescritos que
podemos usar cuando escribimos nuestros programas. Si aprende bien un idioma,
puede aprender fácilmente un nuevo idioma en una fracción del tiempo que le llevó
aprender el primer idioma.

Si es nuevo en la programación, Python es un gran lugar para comenzar. Uno de
las características clave de Python es su simplicidad, lo que lo convierte en el lenguaje ideal
para que los principiantes aprendan. La mayoría de los programas en Python requieren considerablemente
menos líneas de código para realizar la misma tarea en comparación con otras
lenguajes como C. Esto conduce a menos errores de programación y

reduce el tiempo de desarrollo necesario. Además, Python viene con un extensa colección de recursos de terceros que amplían las capacidades de el idioma. Como tal, Python se puede utilizar para una gran variedad de tareas, como para aplicaciones de escritorio, aplicaciones de bases de datos, redes programación, programación de juegos e incluso desarrollo móvil. Último pero no menos importante, Python es un lenguaje multiplataforma, lo que significa que el código escrito para un sistema operativo, como Windows, funcionará bien en Mac OS o Linux sin realizar ningún cambio en el código Python.

¿Estás convencido de que Python es EL lenguaje para aprender? Empecemos...

Capítulo 2: Preparándose para Python

Instalación del intérprete

Antes de que podamos escribir nuestro primer programa Python, tenemos que descargar el intérprete apropiado para nuestras computadoras.

Usaremos Python 3 en este libro porque, como se indica en el oficial Sitio de Python " *Python 2.x es heredado, Python 3.x es el presente y el futuro de el idioma* ". Además, " *Python 3 elimina muchas peculiaridades que pueden hacer tropezar innecesariamente a los programadores principiantes* ".

Sin embargo, tenga en cuenta que Python 2 todavía se usa bastante. Python 2 y 3 son aproximadamente un 90% similares. Por lo tanto, si aprende Python 3, es probable que no tengo problemas para entender los códigos escritos en Python 2.

Para instalar el intérprete para Python 3, diríjase a https://www.python.org/downloads/ . La versión correcta debe ser indicado en la parte superior de la página web. Haga clic en la versión para Python 3 y el software comenzará a descargarse.

Alternativamente, si desea instalar una versión diferente, desplácese hacia abajo en la página

y verá una lista de otras versiones. Haga clic en la versión de lanzamiento que usted quiere. Usaremos la versión 3.4.2 en este libro. Serás redirigido a la página de descarga de esa versión.

Desplácese hacia abajo hasta el final de la página y verá una lista de tablas varios instaladores para esa versión. Elija el instalador correcto para su

Página 11

computadora. El instalador a utilizar depende de dos factores:

1. El sistema operativo (Windows, Mac OS o Linux) y
2. El procesador (32 bits frente a 64 bits) que está utilizando.

Por ejemplo, si está utilizando una computadora con Windows de 64 bits, probablemente utilizar el " instalador MSI de **Windows** x86- **64** ". Simplemente haga clic en el enlace para Descargalo. Si descarga y ejecuta el instalador incorrecto, no se preocupe. Tú recibirá un mensaje de error y el intérprete no se instalará. Simplemente descargue el instalador correcto y estará listo.

Una vez que haya instalado correctamente el intérprete, estará listo para comience a codificar en Python.

Pagina 12

Usando el Python Shell, IDLE y escribiendo nuestro PRIMER programa

Escribiremos nuestro código usando el programa IDLE que viene incluido con nuestro intérprete de Python.

Para hacer eso, primero lancemos el programa IDLE. Lanzas el IDLE programa como la forma en que inicia cualquier otro programa. Por ejemplo en Windows 8, puede buscarlo escribiendo "IDLE" en el cuadro de búsqueda. Una vez que lo encuentre, haga clic en IDLE (Python GUI) para iniciarlo. Tú serás presentado con el Python Shell que se muestra a continuación.

Python Shell nos permite usar Python en modo interactivo. Esta significa que podemos ingresar un comando a la vez. El Shell espera un comando del usuario, lo ejecuta y devuelve el resultado del ejecución. Después de esto, el Shell espera el siguiente comando.

Intente escribir lo siguiente en el Shell. Las líneas que comienzan con >>> son las comandos que debe escribir mientras las líneas después de los comandos muestran el resultados.

```
>>> 2 + 3
5
>>> 3> 2
Cierto
>>> print ('Hola mundo')
Hola Mundo
```


Cuando escribe 2 + 3, está emitiendo un comando al Shell, pidiéndole que evaluar el valor de 2 + 3. Por lo tanto, Shell devuelve la respuesta 5. Cuando escribe 3> 2, le pregunta al Shell si 3 es mayor que 2. El Shell responde True. Finalmente, print es un comando que le pide al Shell que muestre la línea Hello World.

Python Shell es una herramienta muy conveniente para probar comandos de Python,

especialmente cuando empezamos con el idioma. Sin embargo, si sale del Python Shell y lo vuelve a ingresar, todos los comandos que el tipo desaparecerá. Además, no puede utilizar Python Shell para crear un programa real. Para codificar un programa real, debe escribir su código en un archivo de texto y guárdelo con una extensión .py. Este archivo se conoce como Secuencia de comandos de Python.

Para crear una secuencia de comandos de Python, haga clic en Archivo> Nuevo archivo en el menú superior de nuestro Python Shell. Esto abrirá el editor de texto que vamos a usar para escriba nuestro primer programa, el programa "Hello World". Escribiendo el "Hola El programa mundial ”es una especie de rito de iniciación para todos los nuevos programadores. Usaremos este programa para familiarizarnos con el software IDLE.

Escriba el siguiente código en el editor de texto (no en el Shell).

```
#Imprime las palabras "Hola mundo"
imprimir ("Hola mundo")
```

Debería notar que la línea #Imprime las palabras "Hola mundo" está en rojo, mientras que la palabra "imprimir" está en púrpura y "Hola mundo" está en verde. Esta es la forma en que el software hace que nuestro código sea más fácil de leer. los las palabras "imprimir" y "Hola mundo" tienen diferentes propósitos en nuestra programa, por lo que se muestran con diferentes colores. Entraremos en más detalles en capítulos posteriores.

Página 14

La línea #Imprime las palabras "Hola mundo" (en rojo) en realidad no es parte del programa. Es un comentario escrito para hacer nuestro código más legible para otros programadores. Esta línea es ignorada por Python Interprete. Para agregar comentarios a nuestro programa, escribimos un signo # delante de cada línea de comentario, así:

```
#Esto es un comentario
# Esto también es un comentario
# Este es otro comentario más
```

Alternativamente, también podemos usar tres comillas simples (o tres dobles citas) para comentarios de varias líneas, como este:

```
" '
Este es un comentario
Esto también es un comentario
Este es otro comentario más
" '
```

Ahora haga clic en Archivo> Guardar como ... para guardar su código. Asegúrate de guardarlo con

la extensión .py.

¿Hecho? ¡Voilà! Acaba de escribir con éxito su primer Python
programa.

Finalmente, haga clic en Ejecutar> Ejecutar módulo para ejecutar el programa (o presione F5).
Debería ver las palabras Hola mundo impresas en su Python Shell.

Para ver estos pasos en acción, puede consultar este excelente tutorial de
mybringback:
https://www.youtube.com/watch?v=pEFr1eYIePw.

Sin embargo, tenga en cuenta que usó Python 2 en el video, por lo que algunos comandos
le dará un error. Si desea probar sus códigos, debe agregar () para

Página 15

las declaraciones impresas. En lugar de escribir impreso 'Hola mundo',
tienes que escribir print ('Hola mundo'). Además, tienes que
cambie raw_input () por input (). Cubriremos print () y input ()
en el Capítulo 5.

Página 16

Capítulo 3: El mundo de las variables y los operadores

Ahora que hemos terminado con las cosas introductorias, vayamos a lo real cosas. En este capítulo, aprenderá todo sobre variables y operadores. Específicamente, aprenderá qué son las variables y cómo nombrar y declarar ellos. También aprenderemos sobre las operaciones comunes que podemos realizar. en ellos. Listo? Vamonos.

Página 17

¿Qué son las variables?

Las variables son nombres que se dan a los datos que necesitamos almacenar y manipular. en nuestros programas. Por ejemplo, suponga que su programa necesita almacenar el edad de un usuario. Para hacer eso, podemos nombrar estos datos userAge y definir el variable userAge utilizando la siguiente declaración.

```
userAge = 0
```

Después de definir la variable userAge, su programa asignará una cierta área del espacio de almacenamiento de su computadora para almacenar estos datos. Usted puede luego acceda y modifique estos datos refiriéndose a ellos por su nombre, userAge. Cada vez que declara una nueva variable, debe darle un valor inicial. En este ejemplo, le dimos el valor 0. Siempre podemos cambiar este valor en nuestro programa más tarde.

También podemos definir múltiples variables a la vez. Para hacer eso simplemente escribe

```
userAge, userName = 30, 'Peter'
```

Esto es equivalente a

```
userAge = 30
userName = 'Peter'
```


Nombrar una variable

Un nombre de variable en Python solo puede contener letras (a - z, A - B), números o guiones bajos (_). Sin embargo, el primer carácter no puede ser un número. Por lo tanto, puede nombrar sus variables userName, user_name o userName2 pero no 2userName.

Además, hay algunas palabras reservadas que no puede utilizar como nombre de variable porque ya tienen significados preasignados en Pitón. Estas palabras reservadas incluyen palabras como imprimir, ingresar, si, while, *etc*. Aprenderemos sobre cada uno de ellos en los capítulos siguientes.

Por último, los nombres de las variables distinguen entre mayúsculas y minúsculas. nombre de usuario no es el mismo que nombre de usuario.

Hay dos convenciones al nombrar una variable en Python. Podemos utilice la notación de mayúsculas y minúsculas o utilice guiones bajos. El caso del camello es la práctica de escribir palabras compuestas con mayúsculas y minúsculas (p. ej. thisIsAVariableName). Esta es la convención que usaremos en el resto del libro. Alternativamente, otra práctica común es utilizar subrayados (_) para separar las palabras. Si lo prefiere, puede nombrar su variables como esta: this_is_a_variable_name.

Página 19

El letrero de asignación

Tenga en cuenta que el signo = en la declaración userAge = 0 tiene una es decir, del signo = que aprendimos en matemáticas. En programación, el signo = se conoce como señal de asignación. Significa que estamos asignando el valor a el lado derecho del signo = a la variable de la izquierda. Una buena manera de entender la declaración userAge = 0 es pensar en ella como userAge <- 0.

Las afirmaciones x = y e y = x tienen significados muy diferentes en programación.

¿Confuso? Un ejemplo probablemente aclarará esto.

Escriba el siguiente código en su editor IDLE y guárdelo.

```
x = 5
y = 10
x = y
imprimir ("x =", x)
imprimir ("y =", y)
```

Ahora ejecuta el programa. Debería obtener este resultado:

x = 10
y = 10

Aunque x tiene un valor inicial de 5 (declarado en la primera línea), la tercera
la línea x = y asigna el valor de y a x (x <- y), por lo tanto, cambia el
valor de xa 10 mientras que el valor de y permanece sin cambios.

A continuación, modifique el programa cambiando SÓLO UNA declaración: Cambiar

Página 20

la tercera línea desde x = y hasta y = x. Matemáticamente, x = y y y = x significan
la misma cosa. Sin embargo, esto no es así en la programación.

Ejecute el segundo programa. Ahora obtendrás

x = 5
y = 5

Puede ver que en este ejemplo, el valor de x permanece como 5, pero el valor
de y se cambia a 5. Esto se debe a que la declaración y = x asigna el
valor de xay (y <- x). y se convierte en 5 mientras que x permanece sin cambios como 5.

Página 21

Operadores básicos

Además de asignar un valor inicial a una variable, también podemos realizar la operaciones matemáticas habituales sobre variables. Operadores básicos en Python incluir +, -,,, /,% *y* * que representan suma, resta,
multiplicación, división, división de piso, módulo y exponente respectivamente.

Ejemplo:

Suponga que x = 5, y = 2

Suma: x + y = 7

Resta: x - y = 3

Multiplicación: x * y = 10

División: x / y = 2.5

División de piso: x // y = 2 (redondea hacia abajo la respuesta al entero más cercano número)
Módulo: x% y = 1 (da el resto cuando 5 se divide por 2)
Exponente: x ** y = 25 (5 elevado a 2)

Página 22

Más operadores de asignación

Además del signo =, hay algunos operadores de asignación más en Python
(y la mayoría de los lenguajes de programación). Estos incluyen operadores como + =, - =

y * =.

Supongamos que tenemos la variable x, con un valor inicial de 10. Si queremos Incrementar x en 2, podemos escribir
x = x + 2

El programa primero evaluará la expresión de la derecha (x + 2) y asigne la respuesta a la izquierda. Entonces, eventualmente, la declaración anterior se convierte en x <- 12.

En lugar de escribir x = x + 2, también podemos escribir x + = 2 para expresar el mismo significado. El signo + = es en realidad una abreviatura que combina el signo de asignación con el operador de suma. Por tanto, x + = 2 simplemente significa x = x + 2.

De manera similar, si queremos hacer una resta, podemos escribir x = x - 2 o x - = 2. Lo mismo funciona para los 7 operadores mencionados en la sección encima.

Capítulo 4: Tipos de datos en Python

En este capítulo, primero veremos algunos tipos de datos básicos en Python, específicamente el entero, flotante y cadena. A continuación, exploraremos el concepto de tipo de fundición. Finalmente, discutiremos tres tipos de datos más avanzados en Python: la lista, tupla y diccionario.

Página 24

Enteros

Los enteros son números sin partes decimales, como -5, -4, -3, 0, 5, 7, etc.

Para declarar un número entero en Python, simplemente escriba variableName = valor inicial

Ejemplo:
userAge = 20, mobileNumber = 12398724

Página 25

Flotador

Flotante se refiere a números que tienen partes decimales, como 1.234, -0.023, 12.01.

Para declarar un flotante en Python, escribimos variableName = initial valor

Ejemplo:
userHeight = 1,82, userWeight = 67,2

Página 26

Cuerda

Cadena se refiere al texto.

Para declarar una cadena, puede usar variableName = 'initial
valor '(comillas simples) o variableName = "valor inicial"
(doble comillas)

Ejemplo:
userName = 'Peter', userSpouseName = "Janet", userAge
= '30'

En el último ejemplo, debido a que escribimos userAge = '30', userAge es un
cuerda. Por el contrario, si escribió userAge = 30 (sin comillas),
userAge es un número entero.

Podemos combinar múltiples subcadenas usando el signo de concatenar (+).
Por ejemplo, "Peter" + "Lee" es equivalente a la cadena
“PeterLee”.

Funciones de cadena integradas

Python incluye una serie de funciones integradas para manipular cadenas. UN
La función es simplemente un bloque de código reutilizable que realiza una determinada tarea.
Analizaremos las funciones con mayor profundidad en el Capítulo 7.

Un ejemplo de una función disponible en Python es el método upper () para
instrumentos de cuerda. Lo usa para poner en mayúscula todas las letras de una cadena. Por ejemplo,
'Peter'.upper () nos dará la cadena “PETER”. Puede referirse a
Apéndice A para obtener más ejemplos y códigos de muestra sobre cómo utilizar
Métodos de cadena integrados de Python.

Página 27

Formateo de cadenas con el operador%

Las cadenas también se pueden formatear con el operador%. Esto te da
mayor control sobre cómo desea que se muestre y almacene su cadena.
La sintaxis para usar el operador% es

"Cadena a formatear"% (valores o variables a
insertado en una cadena, separado por comas)

Hay tres partes en esta sintaxis. Primero escribimos la cadena para que sea formateado entre comillas. Luego escribimos el símbolo%. Finalmente, tenemos un par de paréntesis () dentro de los cuales escribimos los valores o variables a ser insertado en la cuerda. Este paréntesis con valores dentro es en realidad conocido como tupla, un tipo de datos que cubriremos en el capítulo más adelante.

Escriba el siguiente código en IDLE y ejecútelo.

```
brand = 'Apple'
exchangeRate = 1.235235245

message = 'El precio de este% s portátil es% d USD y
el tipo de cambio es de% 4.2f USD a 1 EUR '% (marca,
1299, tasa de cambio)

imprimir (mensaje)
```

En el ejemplo anterior, la cadena 'El precio de esta computadora portátil% s es% d USD y el tipo de cambio es% 4.2f USD a 1 EUR ' es la cadena que queremos formatear. Usamos% s,% d y% 4.2f formateadores como marcadores de posición en la cadena.

Estos marcadores de posición se reemplazarán con la marca variable, el valor 1299 y la tasa de cambio variable respectivamente, como se indica en el

Página 28

entre paréntesis. Si ejecutamos el código, obtendremos el resultado a continuación.

```
El precio de este portátil de Apple es de 1299 USD y el
el tipo de cambio es de 1,24 USD a 1 EUR
```

El formateador% s se utiliza para representar una cadena ("Apple" en este caso) mientras el formateador% d representa un número entero (1299). Si queremos agregar espacios antes de un entero, podemos agregar un número entre% y d para indicar el longitud deseada de la cuerda. Por ejemplo, "% 5d"% (123) nos dará " 123 ”(con 2 espacios al frente y una longitud total de 5).

El formateador% f se utiliza para formatear flotantes (números con decimales). aquí lo formateamos como% 4.2f donde 4 se refiere a la longitud total y 2 se refiere a 2 lugares decimales. Si queremos agregar espacios antes del número, podemos el formato es% 7.2f, lo que nos dará "1.24" (con 2 lugares decimales, 3 espacios al frente y una longitud total de 7).

Formateo de cadenas usando el método format ()

Además de utilizar el operador% para formatear cadenas, Python también proporciona us con el método format () para formatear cadenas. La sintaxis es

"Cadena a formatear" .format (valores o variables para
insertarse en una cadena, separados por comas)

Cuando usamos el método de formato, no usamos% s,% f o% d como
marcadores de posición. En su lugar, usamos llaves, así:

message = 'El precio de esta computadora portátil **{0: s}** es **{1: d}** USD
y el tipo de cambio es de **{2: 4.2f}** USD a 1
EUR'.format ('Apple', 1299, 1.235235245)

Dentro del corchete, primero escribimos la posición del parámetro en

Página 29

utilizar, seguido de dos puntos. Después de los dos puntos, escribimos el formateador. Allí no debe haber espacios dentro de las llaves.

Cuando escribimos formato ('Apple', 1299, 1.235235245), estamos
pasando tres parámetros al método format (). Los parámetros son
datos que el método necesita para realizar su tarea. Los parametros
son 'Apple', 1299 y 1.235235245.

El parámetro 'Apple' tiene una posición de 0,
1299 tiene una posición de 1 y
1.235235245 tiene una posición de 2.

Las posiciones siempre comienzan desde CERO.

Cuando escribimos {0: s}, le pedimos al intérprete que reemplace {0: s} con el parámetro en la posición 0 y que es una cadena (porque el formateador es 's').

Cuando escribimos {1: d}, nos referimos al parámetro en la posición 1,
que es un número entero (el formateador es d).

Cuando escribimos {2: 4.2f}, nos referimos al parámetro en la posición 2,
que es un flotante y queremos que tenga el formato de 2 lugares decimales y
una longitud total de 4 (el formateador es 4.2f).

Si imprimimos el mensaje, obtendremos
El precio de este portátil de Apple es de 1299 USD y el
el tipo de cambio es de 1,24 USD a 1 EUR

Nota: Si no desea formatear la cadena, simplemente puede escribir

```
message = 'El precio de esta {} computadora portátil es de {} USD y
el tipo de cambio es de {} USD a 1 EUR'.format ('Apple',
```

Página 30

```
1299, 1.235235245)
```

Aquí no tenemos que especificar la posición de los parámetros. los
El intérprete reemplazará las llaves según el orden del
parámetros proporcionados. Nosotros recibiremos

```
El precio de este portátil de Apple es de 1299 USD y el
El tipo de cambio es de 1.235235245 USD a 1 EUR
```

El método format () puede resultar un poco confuso para los principiantes. De hecho,
El formato de cadena puede ser más imaginativo que lo que hemos cubierto aquí, pero
lo que hemos cubierto es suficiente para la mayoría de los propósitos. Para mejorar
comprensión del método format (), pruebe el siguiente programa.

```
message1 = '{0} es más fácil que {1}'. format ('Python',
'Java')
message2 = '{1} es más fácil que {0}'. format ('Python',
'Java')
mensaje3 = '{: 10.2f} y {: d}'. formato (1.234234234, 12)
mensaje4 = '{}'. formato (1.234234234)

imprimir (mensaje1)
# Obtendrás 'Python es más fácil que Java'

imprimir (mensaje2)
# Obtendrá 'Java es más fácil que Python'

imprimir (mensaje3)
# Obtendrás '1.23 y 12'
# No es necesario que indique las posiciones del
parámetros.

imprimir (mensaje4)
# Obtendrás 1.234234234. No se realiza el formateo.
```

Página 31

Puede usar Python Shell para experimentar con el método format ().
Intente escribir varias cadenas y vea lo que obtiene.

Página 32

Conversión de tipos en Python

A veces, en nuestro programa, es necesario convertir de uno
tipo de datos a otro, como de un entero a una cadena. Esto se conoce como
tipo de fundición.

Hay tres funciones integradas en Python que nos permiten escribir
fundición. Estas son las funciones int (), float () y str ().

La función int () en Python toma un flotante o una cadena apropiada y
lo convierte en un número entero. Para cambiar un flotante a un entero, podemos escribir

int (5.712987). Obtendremos 5 como resultado (cualquier cosa después del decimal se elimina el punto). Para cambiar una cadena a un entero, podemos escribir int ("4") y obtendremos 4. Sin embargo, no podemos escribir int ("Hola") o int ("4.22321"). Obtendremos un error en ambos casos.

La función float () toma un número entero o una cadena apropiada y lo cambia a un flotador. Por ejemplo, si escribimos float (2) o float ("2"), obtendremos 2.0. Si escribimos float ("2.09109"), obtendremos 2.09109 que es un flotante y no una cadena, ya que se eliminan las comillas.

La función str (), por otro lado, convierte un entero o un flotante en un cuerda. Por ejemplo, si escribimos str (2.1), obtendremos "2.1".

Ahora que hemos cubierto los tres tipos de datos básicos en Python y sus casting, pasemos a los tipos de datos más avanzados.

Página 33

Lista

Lista se refiere a una colección de datos que normalmente están relacionados. En lugar de Al almacenar estos datos como variables independientes, podemos almacenarlos como una lista. por Por ejemplo, supongamos que nuestro programa necesita almacenar la edad de 5 usuarios. En lugar de almacenarlos como user1Age, user2Age, user3Age, user4Age y user5Age, tiene más sentido almacenarlos como una lista.

Para declarar una lista, escribe listName = [valores iniciales]. Nota que usamos corchetes [] cuando declaramos una lista. Múltiples valores son separados por una coma.

Ejemplo:
userAge = [21, 22, 23, 24, 25]

También podemos declarar una lista sin asignarle ningún valor inicial. Nosotros simplemente escriba listName = []. Lo que tenemos ahora es una lista vacía con no hay elementos en él. Tenemos que usar el método append () mencionado a continuación para agregue elementos a la lista.

Los valores individuales de la lista son accesibles por sus índices e índices siempre comience desde CERO, no 1. Esta es una práctica común en casi todos lenguajes de programación, como C y Java. Por tanto, el primer valor tiene

un índice de 0, el siguiente tiene un índice de 1 y así sucesivamente. Por ejemplo, userAge [0] = 21, userAge [1] = 22

Alternativamente, puede acceder a los valores de una lista desde atrás. El último elemento de la lista tiene un índice de -1, el penúltimo tiene un índice de -2 y así sucesivamente. Por lo tanto, userAge [-1] = 25, userAge [-2] = 24.

Puede asignar una lista, o parte de ella, a una variable. Si escribe userAge2 = userAge, la variable userAge2 se convierte en [21, 22, 23, 24, 25].

Página 34

Si escribe userAge3 = userAge [2: 4], está asignando elementos con índice 2 al índice 4-1 de la lista userAge a la lista userAge3. En otra palabras, userAge3 = [23, 24].

La notación 2: 4 se conoce como rebanada. Siempre que usamos la notación de corte en Python, el elemento en el índice de inicio siempre se incluye, pero el elemento en el final siempre está excluido. Por tanto, la notación 2: 4 se refiere a elementos de índice 2 al índice 4-1 (es decir, índice 3), por lo que userAge3 = [23, 24] y no [23, 24, 25].

La notación de sector incluye un tercer número conocido como paso a paso. Si nosotros escriba userAge4 = userAge [1: 5: 2], obtendremos una sublista que consiste de cada segundo número desde el índice 1 al índice 5-1 porque el paso a paso es 2. Por lo tanto, userAge4 = [22, 24].

Además, las notaciones de sector tienen valores predeterminados útiles. El predeterminado para el primero el número es cero y el valor predeterminado para el segundo número es el tamaño de la lista siendo cortado. Por ejemplo, userAge [: 4] le da valores del índice 0 al índice 4-1 mientras que userAge [1:] le da valores del índice 1 al índice 5-1 (dado que el tamaño de userAge es 5, *es decir* , userAge tiene 5 elementos).

Para modificar elementos en una lista, escribimos listName [índice del elemento a ser modificado] = nuevo valor. Por ejemplo, si desea modificar el segundo elemento, escribe userAge [1] = 5. Su lista se convierte en userAge = [21, 5, 23, 24, 25]

Para agregar elementos, usa la función append (). Por ejemplo, si escribe userAge.append (99), agrega el valor 99 al final de la lista. Tu la lista ahora es userAge = [21, 5, 23, 24, 25, 99]

Para eliminar elementos, escriba del listName [índice del elemento a ser eliminado]. Por ejemplo, si escribe del userAge [2], su lista ahora se convierte en userAge = [21, 5, 24, 25, 99] (el tercer elemento es

Página 35

eliminado)
Para apreciar completamente el funcionamiento de una lista, intente ejecutar lo siguiente programa.

```
#declarando la lista, los elementos de la lista pueden ser de diferentes
tipos de datos myList = [1, 2, 3, 4, 5, "Hola"]

#imprima la lista completa.
imprimir (myList)
# Obtendrá [1, 2, 3, 4, 5, "Hola"]

# imprima el tercer elemento (recuerde: el índice comienza en
cero).
imprimir (myList [2])
# Obtendrás 3

#imprima el último elemento.
imprimir (myList [-1])
# Recibirás "Hola"

#asignar myList (del índice 1 al 4) a myList2 y
imprimir myList2
myList2 = myList [1: 5]
imprimir (myList2)
# Obtendrás [2, 3, 4, 5]

#modificar el segundo elemento en myList e imprimir el
lista actualizada myList [1] = 20
imprimir (myList)
# Obtendrá [1, 20, 3, 4, 5, 'Hola']

#anexar un nuevo elemento a myList e imprimir el actualizado
list myList.append ("¿Cómo estás?")
imprimir (myList)
# Obtendrás [1, 20, 3, 4, 5, 'Hola', 'Cómo estás']
```

Página 36

```
#remove el sexto elemento de myList e imprima el
lista actualizada del
myList [5]
imprimir (myList)
```

```
# Obtendrás [1, 20, 3, 4, 5, '¿Cómo estás?']
```

Hay un par de cosas más que puede hacer con una lista. Para muestra
códigos y más ejemplos sobre cómo trabajar con una lista, consulte el Apéndice B.

Página 37

Tupla

Las tuplas son como listas, pero no puede modificar sus valores. La inicial
los valores son los valores que permanecerán durante el resto del programa. Un
ejemplo donde las tuplas son útiles es cuando su programa necesita almacenar
los nombres de los meses del año.

Para declarar una tupla, escribe tupleName = (valores iniciales).
Observe que usamos corchetes () cuando declaramos una tupla. Múltiple
los valores están separados por una coma.

Ejemplo:

```
monthsOfYear = ("Ene", "Feb", "Mar", "Abr", "Mayo",
"Junio", "julio", "agosto", "septiembre", "octubre", "noviembre", "diciembre")
```

Accede a los valores individuales de una tupla usando sus índices, al igual que con una lista.

Por tanto, monthsOfYear [0] = "Jan", monthsOfYear [-1] = "Dic".

Para obtener más ejemplos de lo que puede hacer con una tupla, consulte el Apéndice C.

Página 38

Diccionario

El diccionario es una colección de PARES de datos relacionados. Por ejemplo, si queremos para almacenar el nombre de usuario y la edad de 5 usuarios, podemos almacenarlos en un diccionario.

Para declarar un diccionario, escribe dictionaryName = {dictionary key: data}, con el requisito de que las claves del diccionario deben ser únicas (dentro de un diccionario). Es decir, no puedes declarar un diccionario como este myDictionary = {"Peter": 38, "John": 51, "Peter": 13}.

Esto se debe a que "Peter" se usa dos veces como clave del diccionario. Tenga en cuenta que nosotros use llaves {} cuando declare un diccionario. Varios pares son separados por una coma.

Ejemplo:
userNameAndAge = {"Peter": 38, "John": 51, "Alex": 13, "Alvin": "No disponible"}

También puede declarar un diccionario usando el método dict (). Declarar el diccionario userNameAndAge anterior, escribe

userNameAndAge = dict (Peter = 38, John = 51, Alex = 13, Alvin = "No disponible")

Cuando usa este método para declarar un diccionario, usa round

corchetes () en lugar de corchetes {} y no pones comillas
marcas para las teclas del diccionario.

Para acceder a elementos individuales en el diccionario, usamos la tecla de diccionario,
que es el primer valor del par {clave de diccionario: datos}. por
Por ejemplo, para obtener la edad de John, escribe userNameAndAge ["John"].

Página 39

Obtendrás el valor 51.

Para modificar elementos en un diccionario, escribimos
dictionaryName [clave de diccionario del elemento a modificar]
= nuevos datos. Por ejemplo, para modificar el par "John": 51, escribimos
userNameAndAge ["John"] = 21. Nuestro diccionario ahora se convierte en
userNameAndAge = {"Peter": 38, "John": 21, "Alex": 13,
“Alvin”: “No disponible”}.

También podemos declarar un diccionario sin asignarle ningún valor inicial.
Simplemente escribimos nombreDiccionario = {}. Lo que tenemos ahora es un
diccionario vacío sin elementos.

Para agregar elementos a un diccionario, escribimos nombreDiccionario [diccionario
clave] = datos. Por ejemplo, si queremos agregar "Joe": 40 a nuestro
diccionario, escribimos userNameAndAge ["Joe"] = 40. Nuestro diccionario
ahora se convierte en userNameAndAge = {"Peter": 38, "John": 21,
"Alex": 13, "Alvin": "No disponible", "Joe": 40}

Para eliminar elementos de un diccionario, escribimos del
dictionaryName [clave de diccionario]. Por ejemplo, para eliminar el
“Alex”: 13 pares, escribimos del userNameAndAge [“Alex”]. Nuestro
el diccionario ahora se convierte en userNameAndAge = {"Peter": 38,
"John": 21, "Alvin": "No disponible", "Joe": 40}

Ejecute el siguiente programa para ver todo esto en acción.

```
#declarar el diccionario, las claves del diccionario y los datos
puede ser de diferentes tipos de datos
myDict = {"One": 1.35, 2.5: "Two Point Five", 3: "+",
7,9: 2}

#imprime el diccionario completo
```

Página 40

```
imprimir (myDict)
# Obtendrá {2.5: 'Dos punto cinco', 3: '+', 'Uno':
1,35, 7,9: 2}
# Tenga en cuenta que los elementos de un diccionario no se almacenan en el
mismo orden que la forma en que los declara.

#imprima el elemento con key = "One".
imprimir (myDict ["Uno"])
# Obtendrás 1.35

#imprima el artículo con key = 7.9.
imprimir (myDict [7.9])
# Obtendrás 2

# modificar el elemento con clave = 2.5 e imprimir el actualizado
diccionario
myDict [2.5] = "Dos y medio"
imprimir (myDict)
# Obtendrá {2.5: 'Dos y medio', 3: '+', 'Uno':
1,35, 7,9: 2}

# agregar un nuevo elemento e imprimir el diccionario actualizado
myDict ["New item"] = "Soy nuevo"
imprimir (myDict)
# Obtendrá {'Elemento nuevo': 'Soy nuevo', 2.5: 'Dos y un
Half ', 3:' + ',' One ': 1.35, 7.9: 2}

#remove el elemento con key = "One" e imprima el
diccionario actualizado
del myDict ["Uno"]
imprimir (myDict)
# Obtendrá {'Elemento nuevo': 'Soy nuevo', 2.5: 'Dos y un
Mitad ', 3:' + ', 7.9: 2}
```

Para obtener más ejemplos y códigos de muestra de cómo trabajar con un diccionario,

Página 41

puede consultar el Apéndice D.

Página 42

Capítulo 5: Cómo hacer que su programa sea interactivo

Ahora que hemos cubierto los conceptos básicos de las variables, escribamos un programa que hace uso de ellos. Volveremos a visitar el programa "Hola mundo" que escribimos en el Capítulo 2, pero esta vez lo haremos interactivo. En lugar de solo decir hola al mundo, queremos que el mundo sepa también nuestros nombres y edades. En Para hacer eso, nuestro programa debe poder solicitarnos información y mostrarlos en la pantalla.

Dos funciones integradas pueden hacer eso por nosotros: input () e print ().

Por ahora, escriba el siguiente programa en IDLE. Guárdelo y ejecútelo.

```
myName = input ("Por favor ingrese su nombre:")
myAge = input ("¿Qué pasa con tu edad?")

print ("Hola mundo, mi nombre es", myName, "y soy",
myAge, "años").
```

El programa debería pedirle su nombre.

Por favor, escriba su nombre:

Supongo que ingresaste a James. Ahora presione Entrar y le pedirá
tu edad.

Que hay sobre tu edad:

Digamos que ingresó 20. Ahora presione Enter nuevamente. Deberías conseguir el
la siguiente oración:

Hola mundo, mi nombre es James y tengo 20 años.

Página 43

Entrada()

En el ejemplo anterior, usamos la función input () dos veces para obtener nuestro nombre y edad del usuario.

myName = input ("Por favor ingrese su nombre:")
La cadena "Por favor ingrese su nombre:" es el mensaje que se mostrará en la pantalla para dar instrucciones al usuario. Una vez que el usuario ingresa información relevante, esta información se almacena **como una cadena** en la variable mi nombre. La siguiente declaración de entrada solicita al usuario su edad y almacena la información **como una cadena** en la variable myAge.

La función input () difiere ligeramente en Python 2 y Python 3. En Python 2, si desea aceptar la entrada del usuario como una cadena, debe usar el función raw_input () en su lugar.

Página 44

Impresión()

La función print () se utiliza para mostrar información a los usuarios. Acepta cero o más expresiones como parámetros, separadas por comas.

En la siguiente declaración, pasamos 5 parámetros a print () función. ¿Puedes identificarlos?

```
print ("Hola mundo, mi nombre es", myName, "y soy",
myAge, "años").
```

La primera es la cadena "Hola mundo, mi nombre es"
La siguiente es la variable myName declarada usando la función de entrada anteriormente.
La siguiente es la cadena "y yo soy", seguida de la variable myAge y finalmente la cadena "años".

Tenga en cuenta que no usamos comillas cuando nos referimos a las variables myName y myAge. Si usa comillas, obtendrá el resultado

```
Hola mundo, mi nombre es myName y tengo mis años de edad
antiguo.
```

en cambio, que obviamente no es lo que queremos.

Otra forma de imprimir una declaración con variables es usar el formateador% aprendimos en el Capítulo 4. Para lograr el mismo resultado que la primera impresión declaración anterior, podemos escribir

```
print ("Hola mundo, mi nombre es% sy tengo% s años
antiguo. "% (myName, myAge))
```

Finalmente, para imprimir la misma declaración usando el método format (),

Página 45

escribir

print ("Hola mundo, mi nombre es {} y tengo {} años
antiguo ".format (myName, myAge))

La función print () es otra función que difiere en Python 2 y Python 3. En Python 2, lo escribirás sin corchetes, así:

imprimir "Hola mundo, mi nombre es" + myName + "y soy
"+ myAge +" años ".

Página 46

Cotizaciones triples

Si necesita mostrar un mensaje largo, puede usar la comilla triple
símbolo (" 'o "" ") para distribuir su mensaje en varias líneas. Por ejemplo,
print (" 'Hola mundo.
Mi nombre es James y tengo 20 años " ').
Te regalaré
Hola Mundo.
Mi nombre es James y tengo 20 años.

Esto ayuda a mejorar la legibilidad de su mensaje.

Página 47

Personajes de escape

A veces es posible que necesitemos imprimir algunos caracteres especiales "no imprimibles" como una pestaña o una nueva línea. En este caso, debe usar la \ (barra invertida) carácter para escapar de los caracteres que de otro modo tienen un significado diferente.

Por ejemplo, para imprimir una pestaña, escribimos el carácter de barra invertida antes del letra t, así: \ t. Sin el carácter \, se imprimirá la letra t. Con se imprime una pestaña. Por lo tanto, si escribe print ('Hello \ tWorld'), obtener Hello World

Otros usos comunes del carácter de barra invertida se muestran a continuación. >>> muestra el comando y las siguientes líneas muestran el resultado.

\ n (imprime una nueva línea)

```
>>> print ('Hola \ nMundo')
Hola
Mundo
```

\\ (Imprime el carácter de barra invertida)

```
>>> imprimir ('\\')

\
```

\ ”(Imprime comillas dobles, de modo que las comillas dobles no señalen el final de la cuerda)

```
>>> print ("Mido 5'9 \" alto ")
Mido 5'9 ”de altura
```

Página 48

\ '(Escriba comillas simples, de modo que las comillas simples no indiquen el final de la cuerda)

```
>>> print ('Mido 5 \' 9 ”de altura ')
Mido 5'9 ”de altura
```

Si no desea que los caracteres precedidos por el carácter \ sean interpretados como caracteres especiales, puede usar cadenas sin procesar agregando una r antes de la primera cita. Por ejemplo, si no quieres que te interpreten como una pestaña, debe escribir print (r'Hola \ tWorld '). Conseguirás Hola \ tWorld como resultado.

Página 49

Capítulo 6: Toma de decisiones y elecciones

Felicitaciones, has llegado al capítulo más interesante. espero
has disfrutado el curso hasta ahora. En este capítulo, veremos cómo
haga que su programa sea más inteligente, capaz de tomar decisiones y tomar decisiones.
Específicamente, veremos la instrucción if, for loop y while
lazo. Estos se conocen como herramientas de control de flujo; ellos controlan el flujo de la
programa. Además, también veremos el intento, excepto la declaración de que
determina qué debe hacer el programa cuando ocurre un error.

Sin embargo, antes de entrar en estas herramientas de flujo de control, primero tenemos que mirar
en declaraciones de condición.

Página 50

Declaraciones de condición

Todas las herramientas de flujo de control implican evaluar una declaración de condición. los
El programa procederá de manera diferente dependiendo de si la condición es

reunió.

La declaración de condición más común es la declaración de comparación. Si queremos comparar si dos variables son iguales, usamos el == signo (doble =). Por ejemplo, si escribe x == y, le está preguntando al programa para comprobar si el valor de x es igual al valor de y. Si ellos estan igual, se cumple la condición y la declaración se evaluará como Verdadero. Más, la declaración se evaluará como Falsa.

Otros signos de comparación incluyen! = (No es igual), <(menor que),> (mayor que), <= (menor o igual que) y> = (mayor que o igual a). La siguiente lista muestra cómo se pueden utilizar estos signos y proporciona ejemplos de declaraciones que se evaluarán como Verdadero.

No es igual a: 5! = 2

Mayor que: 5> 2

Menor que: 2 <5

Mayor o igual a: 5> = 2
5> = 5

Menor o igual a: 2 <= 5
2 <= 2

También tenemos tres operadores lógicos, y, o, no que son útiles si desea combinar múltiples condiciones.

Página 51

El operador y devuelve True si se cumplen todas las condiciones. De lo contrario, volverá Falso. Por ejemplo, la declaración 5 == 5 y 2> 1 devolverá True ya que ambas condiciones son verdaderas.

El operador or devuelve Verdadero si se cumple al menos una condición. De lo contrario lo hará falso retorno. La declaración 5> 2 o 7> 10 o 3 == 2 será devuelve True ya que la primera condición 5> 2 es True.

El operador no devuelve Verdadero si la condición después de la palabra clave no es falso. De lo contrario, devolverá False. La declaración no 2> 5 devolverá True ya que 2 no es mayor que 5.

Página 52

Si declaración

La sentencia if es uno de los flujos de control más utilizados.
declaraciones. Permite al programa evaluar si se cumple una determinada condición,
y realizar la acción apropiada en función del resultado de la
evaluación. La estructura de una instrucción if es la siguiente:

```
si se cumple la condición 1: haga A
elif la condición 2 se cumple: hacer B
elif se cumple la condición 3: hacer C
si se cumple la condición 4: hacer D
más:
        hacer E
```

elif significa "else if" y puede tener tantas declaraciones elif como
te gusta.

Si ha codificado en otros lenguajes como C o Java antes, es posible que
sorprendido al notar que no se necesitan paréntesis () en Python después de la
if, elif y la palabra clave else. Además, Python no usa curly {}
corchetes para definir el inicio y el final de la instrucción if. Más bien, Python
usa sangría. Todo lo que esté sangrado se trata como un bloque de código que
se ejecutará si la condición se evalúa como verdadera.

Para comprender completamente cómo funciona la instrucción if, active IDLE e ingrese
el siguiente código.

```
userInput = input ('Ingrese 1 o 2:')
```

```
if userInput == "1": print ("Hola mundo") print ("¿Cómo
¿Es usted? ") elif userInput ==" 2 ": print (" Python
¡Rocas! ") Print (" Me encanta Python ") más:
        print ("No ingresaste un número válido")
```

El programa primero solicita al usuario una entrada utilizando la función de entrada.
El resultado se almacena en la variable userInput como una cadena.

Página 53

A continuación, la declaración if userInput == "1": compara el
variable userInput con la cadena “1”. Si el valor almacenado en userInput
es "1", el programa ejecutará todas las instrucciones que tengan sangría hasta que
termina la sangría. En este ejemplo, imprimirá "Hola mundo", seguido de
"¿Cómo estás?".

Alternativamente, si el valor almacenado en userInput es "2", el programa
imprimir "Python Rocks", seguido de "Me encanta Python".

Para todos los demás valores, el programa imprimirá "No ingresó un
Número válido".

Ejecute el programa tres veces, ingrese 1, 2 y 3 respectivamente para cada ejecución.
Obtendrá el siguiente resultado:

```
Ingrese 1 o 2: 1
Hola Mundo
¿Cómo estás?

Ingrese 1 o 2: 2
¡Python Rocks!
Me encanta python

Ingrese 1 o 2: 3
No ingresaste un número válido
```

Página 54

En línea si

Una instrucción if en línea es una forma más simple de una instrucción if y es más conveniente si solo necesita realizar una tarea simple. La sintaxis es:

haga la Tarea A si la condición es verdadera, de lo contrario, realice la Tarea B

Por ejemplo,

num1 = 12 si myInt == 10 más 13

Esta declaración asigna 12 a num1 (Tarea A) si myInt es igual a 10. De lo contrario asigna 13 a num1 (Tarea B).

Otro ejemplo es
print ("Esta es la tarea A" si myInt == 10 else "Esto es
tarea B ")

Esta declaración imprime "Esta es la tarea A" (Tarea A) si myInt es igual a 10. De lo contrario, imprime "Esta es la tarea B" (Tarea B).

En bucle

A continuación, veamos el bucle for. El bucle for ejecuta un bloque de código repetidamente hasta que la condición de la instrucción for deje de ser válida.

Recorriendo un iterable

En Python, un iterable se refiere a cualquier cosa sobre la que se pueda realizar un bucle, como una cadena, lista o tupla. La sintaxis para recorrer un iterable es como sigue:

```
para un in iterable:
    imprimir (a)
```

Ejemplo:

```
mascotas = ['gatos', 'perros', 'conejos', 'hámsters']

para myPets en mascotas:
    imprimir (misMascotas)
```

En el programa anterior, primero declaramos la lista de mascotas y le damos el miembros 'gatos', 'perros', 'conejos' y 'hámsters'. Siguiente el declaración para myPets en mascotas: recorre la lista de mascotas y asigna a cada miembro de la lista a la variable myPets.

La primera vez que el programa se ejecuta a través del ciclo for, asigna 'gatos' a la variable myPets. La declaración print (myPets) luego imprime el valor 'gatos'. La segunda vez que los programas recorren el declaración, asigna el valor 'perros' a myPets e imprime el valor 'perros'. El programa continúa recorriendo la lista hasta el final de se alcanza la lista.

Página 56

Si ejecuta el programa, obtendrá

```
gatos
perros
conejos
hámsters
```

También podemos mostrar el índice de los miembros en la lista. Para hacer eso, nosotros utilice la función enumerate ().

```
para el índice, myPets en enumerate (mascotas):
    imprimir (índice, myPets)
```

Esto nos dará la salida

```
0 gatos
1 perros
2 conejos
3 hámster
```

El siguiente ejemplo muestra cómo recorrer una cadena.

```
mensaje = 'Hola'

para yo en el mensaje:
        imprimir (i)
```

La salida es

```
H
mi
l
```

Página 57

```
l
o
```

Recorrer una secuencia de números

Para recorrer una secuencia de números, la función integrada range ()
Viene muy bien. La función range () genera una lista de números y
tiene el rango de sintaxis (inicio, final, paso).

Si no se da inicio, los números generados comenzarán desde cero.

Nota: un consejo útil para recordar aquí es que en Python (y la mayoría
lenguajes de programación), a menos que se indique lo contrario, siempre partimos de
cero.

Por ejemplo, el índice de una lista y una tupla comienza desde cero.
Cuando se usa el método format () para cadenas, las posiciones de
los parámetros comienzan desde cero.
Cuando se usa la función range (), si no se da inicio, los números
inicio generado desde cero.

Si no se da el paso, se generará una lista de números consecutivos (es decir,
paso = 1). Debe proporcionarse el valor final. Sin embargo, una cosa rara
acerca de la función range () es que el valor final dado nunca es parte de
la lista generada.

Por ejemplo,
range (5) generará la lista [0, 1, 2, 3, 4]

Página 58

range (3, 10) generará [3, 4, 5, 6, 7, 8, 9]
range (4, 10, 2) generará [4, 6, 8]

Para ver cómo funciona la función range () en una declaración for, intente ejecutar el siguiente código:

```
para i en el rango (5):
        imprimir (i)
```

Deberías conseguir
0
1
2
3
4

Página 59

Mientras bucle

La siguiente declaración de flujo de control que veremos es el while
lazo. Como sugiere el nombre, un ciclo while se ejecuta repetidamente
instrucciones dentro del bucle mientras una determinada condición sigue siendo válida. los
La estructura de una instrucción while es la siguiente:

```
mientras que la condición es verdadera:
        hacer una
```

La mayoría de las veces, cuando se usa un bucle while, primero debemos declarar un
variable para funcionar como un contador de bucle. Llamemos a esta variable
mostrador. La condición en la declaración while evaluará el valor
de contador para determinar si es menor (o mayor) que un cierto valor. Si se
es decir, se ejecutará el ciclo. Veamos un programa de muestra.

```
contador = 5

mientras que el contador> 0:
        print ("Contador =", contador) contador = contador -
1
```

Si ejecuta el programa, obtendrá el siguiente resultado

```
Contador = 5
Contador = 4
Contador = 3
Contador = 2
Contador = 1
```

A primera vista, una instrucción while parece tener la sintaxis más simple y
debería ser el más fácil de usar. Sin embargo, hay que tener cuidado al usar
bucles while debido al peligro de bucles infinitos. Note que en el
programa anterior, tenemos la línea contador = contador - 1? Esta línea
Es crucial. Disminuye el valor del contador en 1 y asigna este nuevo

valor de nuevo al contador, sobrescribiendo el valor original.

Necesitamos disminuir el valor del contador en 1 para que el bucle
condición mientras que el contador> 0 eventualmente se evaluará como Falso. Si nosotros
Olvidar hacer eso, el bucle seguirá funcionando sin fin, lo que resultará en un
Bucle infinito. Si desea experimentar esto de primera mano, simplemente elimine la línea
contador = contador - 1 e intente ejecutar el programa nuevamente. los
El programa seguirá imprimiendo el contador = 5 hasta que de alguna manera elimine el
programa. No es una experiencia agradable, especialmente si tienes una gran
programa y no tiene idea de qué segmento de código está causando el infinito
lazo.

Página 61

Rotura

Al trabajar con bucles, a veces es posible que desee salir de todo el bucle
cuando se cumple una determinada condición. Para hacer eso, usamos la palabra clave break.
Ejecute el siguiente programa para ver cómo funciona.

```
j = 0
para i en el rango (5): j = j + 2
        imprimir ('i =', i, ', j =', j) si j == 6: romper
```

Debería obtener el siguiente resultado.

```
i = 0, j = 2
i = 1, j = 4
i = 2, j = 6
```

Sin la palabra clave break, el programa debería recorrer un ciclo de i = 0 a i = 4
porque usamos la función range (5). Sin embargo con el descanso
palabra clave, el programa termina prematuramente en i = 2. Esto se debe a que cuando i =
2, j alcanza el valor de 6 y la palabra clave break hace que el bucle
fin.

En el ejemplo anterior, observe que usamos una instrucción if dentro de una for
lazo. Es muy común para nosotros 'mezclar y combinar' varias herramientas de control en

programación, como usar un bucle while dentro de una declaración if o
usando un bucle for dentro de un bucle while. Esto se conoce como control anidado
declaración.

Página 62

Seguir

Otra palabra clave útil para los bucles es la palabra clave continue. Cuando nosotros
use continue, el resto del ciclo después de que se omita la palabra clave para ese
iteración. Un ejemplo lo aclarará.

```
j = 0
para i en el rango (5): j = j + 2
        imprimir ('\ ni =', i, ', j =', j) si j == 6:
continue print ('Seré omitido si j = 6')
```

Obtendrá el siguiente resultado:

```
i = 0, j = 2
Seré omitido si j = 6

i = 1, j = 4
Seré omitido si j = 6

i = 2, j = 6

i = 3, j = 8
Seré omitido si j = 6

i = 4, j = 10
Seré omitido si j = 6
```

Cuando j = 6, la línea que sigue a la palabra clave continue no se imprime. Otro
que eso, todo funciona como de costumbre.

Página 63

Prueba, excepto

La declaración de control final que veremos es la declaración try, except.
Esta declaración controla cómo procede el programa cuando ocurre un error.
La sintaxis es la siguiente:

```
tratar:
    hacer algo
excepto:
    hacer algo más cuando ocurre un error
```

Por ejemplo, intente ejecutar el programa a continuación

```
tratar:
    respuesta = 12/0
    imprimir (responder)
excepto:
    imprimir ("Se produjo un error")
```

Cuando ejecute el programa, aparecerá el mensaje "Un error ocurrió". Esto se debe a que cuando el programa intenta ejecutar el declaración respuesta = 12/0 en el bloque try, se produce un error ya que no se puede dividir un número por cero. El resto del bloque try es ignorado y la instrucción en el bloque excepto se ejecuta en su lugar.

Si desea mostrar mensajes de error más específicos a sus usuarios dependiendo del error, puede especificar el tipo de error después de la excepción palabra clave. Intente ejecutar el programa a continuación.

```
tratar:
    userInput1 = int (input ("Introduzca un número:"))
userInput2 = int (input ("Introduzca otro número:
")) respuesta = userInput1 / userInput2
    print ("La respuesta es", respuesta) myFile =
open ("missing.txt", 'r') excepto ValueError:
    print ("Error: No ingresaste un número") excepto
ZeroDivisionError: print ("Error: No se puede dividir por
cero ") excepto la excepción como e:
```

Página 64

```
    print ("Error desconocido:", e)
```

La siguiente lista muestra las distintas salidas para diferentes entradas de usuario. >>> denota la entrada del usuario y => denota la salida.

>>> Por favor ingrese un número: m => Error: No lo hizo

Ingrese un numero
Razón: el usuario ingresó una cadena que no se puede convertir en un número entero. Esta es un ValueError. Por lo tanto, la declaración en la excepción ValueError se muestra el bloque.

>>> Introduzca un número: 12
>>> Introduzca otro número: 0
=> Error: no se puede dividir por cero
Razón: userInput2 = 0. Dado que no podemos dividir un número por cero, esto es un ZeroDivisionError. La declaración en el excepto Se muestra el bloque ZeroDivisionError.

>>> Introduzca un número: 12
>>> Introduzca otro número: 3
=> La respuesta es 4.0
=> Error desconocido: [Errno 2] No existe tal archivo o directorio: 'missing.txt'

Motivo: el usuario ingresa valores aceptables y la línea impresa ("El la respuesta es ", respuesta) se ejecuta correctamente. Sin embargo, la siguiente línea genera un error como missing.txt no se encuentra. Dado que esto no es un ValueError o ZeroDivisionError, el último bloque excepto es ejecutado.

ValueError y ZeroDivisionError son dos de los muchos pre tipos de error definidos en Python. ValueError se genera cuando una función operación o función recibe un parámetro que tiene el tipo correcto pero un valor inapropiado. ZeroDivisionError se genera cuando el programa intenta dividir por cero. Otros errores comunes en Python incluyen

Página 65

IOError:
Se genera cuando falla una operación de E / S (como la función open () incorporada) por una razón relacionada con E / S, por ejemplo, "archivo no encontrado".

ImportError:
Se genera cuando una declaración de importación no encuentra la definición del módulo.
IndexError:
Se genera cuando un índice de secuencia (por ejemplo, cadena, lista, tupla) está fuera de rango.

KeyError:
Se genera cuando no se encuentra una clave de diccionario.

NameError:
Se genera cuando no se encuentra un nombre local o global.

Error de tecleado:

Se genera cuando se aplica una operación o función a un objeto de tipo inapropiado.

Para obtener una lista completa de todos los tipos de errores en Python, puede consultar https://docs.python.org/3/library/exceptions.html.

Python también viene con mensajes de error predefinidos para cada uno de los diferentes tipos de errores. Si desea mostrar el mensaje, utilice el como palabra clave después del tipo de error. Por ejemplo, para mostrar el valor predeterminado Mensaje ValueError, escribe:

```
excepto ValueError como e: print (e)
```

e es el nombre de la variable asignado al error. Puedes darle alguna otra nombres, pero es una práctica común utilizar e. La última declaración excepto en Nuestro programa

```
excepto la excepción como e:
        print ("Error desconocido:", e)
```

es un ejemplo del uso del mensaje de error predefinido. Sirve como final

Página 66

Intente detectar cualquier error inesperado.

Página 67

Capítulo 7: Funciones y módulos

En nuestros capítulos anteriores, hemos mencionado brevemente funciones y módulos. En este capítulo, veámoslos en detalle. Para reiterar, toda la programación los idiomas vienen con códigos integrados que podemos usar para hacer nuestras vidas más fácil como programadores. Estos códigos constan de clases preescritas, variables y funciones para realizar ciertas tareas comunes y son guardado en archivos conocidos como módulos. Primero veamos las funciones.

Página 68

¿Qué son las funciones?

Las funciones son simplemente códigos preescritos que realizan una determinada tarea. Por un analogía, piense en las funciones matemáticas disponibles en MS Excel. A sumar números, podemos usar la función sum () y escribir sum (A1: A5) en lugar de escribir A1 + A2 + A3 + A4 + A5.

Dependiendo de cómo esté escrita la función, si es parte de una clase (un clase es un concepto en programación orientada a objetos que no cubriremos en este libro) y cómo lo importa, podemos llamar a una función simplemente por escribiendo el nombre de la función o usando la notación de puntos. Algunos Las funciones requieren que les pasemos datos para que puedan realizar sus tareas. Estas Los datos se conocen como parámetros y los pasamos a la función mediante encerrando sus valores entre paréntesis () separados por comas.

Por ejemplo, para usar la función print () para mostrar texto en el pantalla, lo llamamos escribiendo print ("Hola mundo") donde print es el nombre de la función y "Hello World" es el parámetro.

Por otro lado, para usar la función replace () para manipular texto cadenas, tenemos que escribir "Hola mundo" .replace ("Mundo", "Universo") donde reemplazar es el nombre de la función y "Mundo" y "Universo" son los parámetros. La cadena antes del punto (es decir “Hola mundo”) es la cadena que se verá afectada. Por lo tanto, "Hola Mundo ”se cambiará a“ Hola universo ”.

Definición de sus propias funciones

Podemos definir nuestras propias funciones en Python y reutilizarlas a lo largo el programa. La sintaxis para definir una función es la siguiente: def functionName (parámetros): código que detalla lo que la función debería devolver [expresión]

Aquí hay dos palabras clave, def y return.

def le dice al programa que el código sangrado de la siguiente línea en adelante es parte de la función. return es la palabra clave que usamos para devolver un respuesta de la función. Puede haber más de una vuelta declaraciones en una función. Sin embargo, una vez que la función ejecuta un retorno declaración, la función saldrá. Si su función no necesita regresar cualquier valor, puede omitir la declaración de retorno. Alternativamente, puede escribir retorno o retorno Ninguno.

Definamos ahora nuestra primera función. Suponga que queremos determinar si un el número dado es un número primo. Así es como podemos definir la función usando el operador de módulo (%) que aprendimos en el Capítulo 3 y el loop y if que aprendimos en el Capítulo 6.

def checkIfPrime (numberToCheck): para x en el rango (2, numberToCheck): if (numberToCheck% x == 0):
return False return True
En la función anterior, las líneas 2 y 3 usan un bucle for para dividir el parámetro numberToCheck por todos los números del 2 al numberToCheck - 1 para determinar si el resto es cero. Si el resto es cero, numberToCheck no es un número primo. La línea 4 devolverá False y el saldrá de la función.

Si por la última iteración del bucle for, ninguna de las divisiones da un resto de cero, la función llegará a la línea 5 y devolverá True. La función

Página 70

luego salga.

Para usar esta función, escribimos checkIfPrime (13) y lo asignamos a un variable como esta
answer = checkIfPrime (13)
Aquí estamos pasando 13 como parámetro. Entonces podemos imprimir el responda escribiendo print (respuesta). Obtendremos el resultado: Verdadero.

Página 71

Alcance variable

Un concepto importante para entender al definir una función es el concepto de alcance variable. Las variables definidas dentro de una función se tratan diferente a las variables definidas fuera. Hay dos diferencias principales.

En primer lugar, cualquier variable declarada dentro de una función solo es accesible dentro de la función. Estos se conocen como variables locales. Cualquier variable declarada fuera de una función se conoce como variable global y es accesible en cualquier parte del programa.

Para entender esto, pruebe el siguiente código:

```
message1 = "Variable global"

def myFunction ():
    print ("\ nDENTRO DE LA FUNCIÓN") #Variables globales
son accesibles dentro de una función de impresión (mensaje1)
    # Declarando una variable local message2 = “Local
Variable"
    imprimir (mensaje2)

# Llamar a la función myFunction ()

imprimir ("\ nFUERA DE LA FUNCIÓN")
# Las variables globales son accesibles fuera de la función
imprimir (mensaje1)

# Las variables locales NO son accesibles fuera de la función.
imprimir (mensaje2)
```

Si ejecuta el programa, obtendrá el resultado a continuación.

DENTRO DE LA FUNCIÓN

Página 72

Variable global
Variable local

FUERA DE LA FUNCIÓN
Variable global
NameError: el nombre 'mensaje2' no está definido
Dentro de la función, tanto las variables locales como las globales son accesibles.
Fuera de la función, la variable local message2 ya no es
accesible. Obtenemos un NameError cuando intentamos acceder a él fuera del
función.

El segundo concepto que hay que entender sobre el alcance variable es que si un local
variable comparte el mismo nombre que una variable global, cualquier código dentro de la
La función está accediendo a la variable local. Cualquier código externo está accediendo
la variable global. Intente ejecutar el código a continuación
message1 = "Variable global (comparte el mismo nombre que un
variable local)"

```
def myFunction ():
        message1 = "Variable local (comparte el mismo nombre que un
variable global)"
        imprimir ("\ nDENTRO DE LA FUNCIÓN") imprimir
(mensaje1)

# Llamar a la función myFunction ()

# Imprimir mensaje1 FUERA de la función imprimir
("\ NFUERA DE LA FUNCIÓN") imprimir (mensaje1)
```

Obtendrá el resultado de la siguiente manera:
DENTRO DE LA FUNCIÓN
Variable local (comparte el mismo nombre que una variable global)
FUERA DE LA FUNCIÓN
Variable global (comparte el mismo nombre que una variable local)
Cuando imprimimos message1 dentro de la función, imprime "Local

Página 73

Variable (comparte el mismo nombre que una variable global) "ya que

está imprimiendo la variable local. Cuando lo imprimimos afuera, está accediendo al variable global y, por lo tanto, imprime "Variable global (comparte la misma nombre como variable local) ".

Página 74

Importación de módulos

Python viene con una gran cantidad de funciones integradas. Estas funciones se guardan en archivos conocidos como módulos. Para usar los códigos integrados en Python módulos, primero tenemos que importarlos a nuestros programas. Hacemos eso por utilizando la palabra clave de importación. Hay tres formas de hacerlo.

La primera forma es importar todo el módulo escribiendo import Nombre del módulo.

Por ejemplo, para importar el módulo aleatorio, escribimos import random.

Para usar la función randrange () en el módulo aleatorio, escribimos rango aleatorio (1, 10).

Si le resulta demasiado problemático escribir al azar cada vez que utiliza el función, puede importar el módulo escribiendo import random as r (donde r es cualquier nombre de su elección). Ahora para usar el randrange () función, simplemente escribe r.randrange (1, 10).

La tercera forma de importar módulos es importar funciones específicas del módulo escribiendo desde moduleName import name1 [, name2 [, ... nombreN]].

Por ejemplo, para importar la función randrange () desde el módulo, escribimos desde el rango aleatorio de importación aleatoria. Si queremos importamos más de una función, las separamos con una coma. A importar las funciones randrange () y randint (), escribimos desde randrange de importación aleatoria, randint. Para usar la función ahora, ya no tienes que usar la notación de puntos. Simplemente escriba randrange (1, 10).

Página 75

Creando nuestro propio módulo

Además de importar módulos integrados, también podemos crear nuestros propios módulos. Esto es muy útil si tiene algunas funciones que desea reutilizar en otros proyectos de programación en el futuro.

Crear un módulo es sencillo. Simplemente guarde el archivo con una extensión .py y colóquelo en la misma carpeta que el archivo Python que lo va a importar desde.

Suponga que desea utilizar la función checkIfPrime () definida anteriormente en otro script de Python. Así es como lo haces. Primero guarda el código de arriba como prime.py en su escritorio. prime.py debería tener lo siguiente código.

```
def checkIfPrime (numberToCheck):
    para x en el rango (2, numberToCheck):
    si (numberToCheck% x == 0):
    falso retorno
    volver verdadero
```

A continuación, cree otro archivo Python y asígnele el nombre useCheckIfPrime.py. Guárdelo también en su escritorio. useCheckIfPrime.py debe tener el

siguiente código.

```
import prime
answer = prime.checkIfPrime (13)
imprimir (responder)
```

Ahora ejecute useCheckIfPrime.py. Debería obtener la salida True.
Simple como eso.

Sin embargo, suponga que desea almacenar prime.py y

Página 76

useCheckIfPrime.py en diferentes carpetas. Vas a tener que agregar
algunos códigos para usar CheckIfPrime.py para decirle al intérprete de Python
dónde encontrar el módulo.

Digamos que creó una carpeta llamada 'MyPythonModules' en su unidad C para
almacenar prime.py. Debe agregar el siguiente código en la parte superior de su
useCheckIfPrime.py (antes de la línea import prime).

```
importar sys

si 'C: \\ MyPythonModules' no está en sys.path:
sys.path.append ('C: \\ MyPythonModules')
```

sys.path se refiere a la ruta del sistema de Python. Esta es la lista de
directorios por los que pasa Python para buscar módulos y archivos. los
el código anterior agrega la carpeta 'C: \ MyPythonModules' a su sistema
camino.

Ahora puede poner prime.py en C: \ MyPythonModules y
checkIfPrime.py en cualquier otra carpeta de su elección.

Página 77

Capítulo 8: Trabajar con archivos

¡Frio! Llegamos al último capítulo del libro antes del proyecto. En
En este capítulo, veremos cómo trabajar con archivos externos.

En el Capítulo 5 anteriormente, aprendimos cómo obtener información de los usuarios mediante el función input (). Sin embargo, en algunos casos, lograr que los usuarios ingresen datos en nuestro programa puede no ser práctico, especialmente si nuestro programa necesita trabajar con grandes cantidades de datos. En casos como este, una forma más conveniente manera es preparar la información necesaria como un archivo externo y obtener nuestro programas para leer la información del archivo. En este capítulo, estamos voy a aprender a hacer eso. Listo?

Página 78

Abrir y leer archivos de texto

El primer tipo de archivo que vamos a leer es un archivo de texto simple con varias líneas de texto. Para hacer eso, primero creemos un archivo de texto con el

siguientes líneas.

Aprenda Python en un día y aprenda bien
Python para principiantes con proyecto práctico
El único libro que necesita para comenzar a codificar en Python inmediatamente
http://www.learncodingfast.com/python

Guarde este archivo de texto como myfile.txt en su escritorio. A continuación, encienda IDLE y escriba el código a continuación. Guarde este código como fileOperation.py en su escritorio también.

```
f = abierto ('miarchivo.txt', 'r')

primera línea = f.readline ()
secondline = f.readline ()
imprimir (primera línea)
imprimir (segunda línea)

f.close ()
```

La primera línea del código abre el archivo. Antes de que podamos leer de cualquier archivo, tenemos que abrirlo (al igual que necesitas abrir este libro electrónico en tu kindle dispositivo o aplicación para leerlo). La función open () hace eso y requiere dos parámetros:

El primer parámetro es la ruta al archivo. Si tu no salvaste fileOperation.py y myfile.txt en la misma carpeta (escritorio en este caso), debe reemplazar 'myfile.txt' con la ruta real donde almacenó el archivo de texto. Por ejemplo, si lo almacenó en una carpeta

Página 79

llamado 'PythonFiles' en su unidad C, debe escribir 'C: \\ PythonFiles \\ myfile.txt' (con doble barra invertida \\).

El segundo parámetro es el modo. Esto especifica cómo será el archivo usado. Los modos más utilizados son

modo 'r':
Solo para lectura.

modo 'w':
Solo para escribir.
Si el archivo especificado no existe, se creará.
Si el archivo especificado existe, se borrarán todos los datos existentes en el archivo.

modo 'a':
Para agregar.

Si el archivo especificado no existe, se creará.
Si el archivo especificado existe, cualquier dato escrito en el archivo se agrega automáticamente hasta el final

modo 'r +':
Tanto para leer como para escribir.

Después de abrir el archivo, la siguiente declaración firstline =
f.readline () lee la primera línea del archivo y la asigna al
primera línea variable.

Cada vez que se llama a la función readline (), lee una nueva línea de
el archivo. En nuestro programa, readline () se llamó dos veces. De ahí la primera
se leerán dos líneas. Cuando ejecute el programa, obtendrá el resultado:

Aprenda Python en un día y aprenda bien

Página 80

Python para principiantes con proyecto práctico

Notará que se inserta un salto de línea después de cada línea. Esto es porque
la función readline () agrega los caracteres '\ n' al final de cada
línea. Si no desea la línea adicional entre cada línea de texto, puede
imprimir (primera línea, fin = ''). Esto eliminará el '\ n'
caracteres. Después de leer e imprimir las dos primeras líneas, la última oración
f.close () cierra el archivo. Siempre debe cerrar el archivo una vez que
termine de leerlo para liberar los recursos del sistema.

Página 81

Uso de un bucle for para leer archivos de texto

Además de utilizar el método readline () anterior para leer un archivo de texto, también puede usar un bucle for. De hecho, el bucle for es más elegante y forma eficiente de leer archivos de texto. El siguiente programa muestra cómo es esto hecho.

```
f = abierto ('miarchivo.txt', 'r')
para la línea en f:
        imprimir (línea, fin = '')
f.close ()
```

El bucle for recorre el archivo de texto línea por línea. Cuando lo ejecutas usted obtendrá
Aprenda Python en un día y aprenda bien Python para
Principiantes con proyecto práctico El único libro que necesita
para comenzar a codificar en Python inmediatamente
http://www.learncodingfast.com/python

Página 82

Escribir en un archivo de texto

Ahora que hemos aprendido a abrir y leer un archivo, intentemos escribir en él. Para hacer eso, usaremos el modo 'a' (agregar). También puede utilizar la 'w' modo, pero borrará todo el contenido anterior en el archivo si el archivo ya existe. Intente ejecutar el siguiente programa.

```
f = open ('myfile.txt', 'a')
f.write ('\ nEsta oración se agregará.')
f.write ('\ n¡Python es divertido!')
f.close ()
```

Aquí usamos la función write () para agregar las dos oraciones 'This se agregará una oración. y "¡Python es divertido!" al archivo, cada uno comenzando en una nueva línea ya que usamos los caracteres de escape '\norte'. Usted obtendrá

```
Aprenda Python en un día y aprenda bien Python para
Principiantes con proyecto práctico El único libro que necesita
para comenzar a codificar en Python inmediatamente
http://www.learncodingfast.com/python Esta oración
se adjuntará.
¡Python es divertido!
```


Abrir y leer archivos de texto por tamaño de búfer

A veces, es posible que deseemos leer un archivo por tamaño de búfer para que nuestro programa no utiliza demasiados recursos de memoria. Para hacer eso, podemos usar el función read () (en lugar de la función readline ()) que nos permite para especificar el tamaño de búfer que queremos. Prueba el siguiente programa:

```
inputFile = open ('myfile.txt', 'r') outputFile = abierto
('myoutputfile.txt', 'w')
```

```
msg = inputFile.read (10)
while len (msg):
outputFile.write (msg) msg = inputFile.read (10)
inputFile.close () outputFile.close ()
```

Primero, abrimos dos archivos, inputFile.txt y outputFile.txt archivos para lectura y escritura respectivamente.

A continuación, usamos la declaración msg = inputFile.read (10) y un while bucle para recorrer el archivo 10 bytes a la vez. El valor 10 en el paréntesis le dice a la función read () que solo lea 10 bytes. los while condición while len (msg): comprueba la longitud de la variable msg. Siempre que la longitud no sea cero, el bucle se ejecutará.

Dentro del ciclo while, la declaración outputFile.write (msg) escribe el mensaje al archivo de salida. Después de escribir el mensaje, la declaración msg = inputFile.read (10) lee los siguientes 10 bytes y mantiene hacerlo hasta que se lea todo el archivo. Cuando eso sucede, el programa cierra ambos archivos.

Cuando ejecute el programa, aparecerá un nuevo archivo myoutputfile.txt creado. Cuando abras el archivo, notarás que tiene el mismo contenido. como su archivo de entrada myfile.txt. Para demostrar que solo se leen 10 bytes en un tiempo, puede cambiar la línea outputFile.write (msg) en el programa para outputFile.write (msg + '\ n'). Ahora ejecuta el programa otra vez. myoutputfile.txt ahora contiene líneas con un máximo de 10

Página 84

caracteres. Aquí hay un segmento de lo que obtendrá.

```
Aprende Pyth
en uno
Día y le
arn It Wel
```

Página 85

Abrir, leer y escribir archivos binarios

Los archivos binarios hacen referencia a cualquier archivo que no contenga texto, como imágenes o videos. archivos. Para trabajar con archivos binarios, simplemente usamos el modo 'rb' o 'wb'. Copiar un archivo jpeg en su escritorio y renómbrelo myimage.jpg. Ahora edite el programa anterior cambiando las dos primeras líneas

```
inputFile = open ('myfile.txt', 'r')
outputFile = open ('myoutputfile.txt', 'w')
```

a

```
inputFile = open ('myimage.jpg', 'rb')
outputFile = open ('myoutputimage.jpg', 'wb')
```

Asegúrese de cambiar también la declaración outputFile.write (msg + '\ n') de nuevo a outputFile.write (msg).

Ejecute el nuevo programa. Debería tener un archivo de imagen adicional llamado myoutputimage.jpg en su escritorio. Cuando abre el archivo de imagen, debería verse exactamente como myimage.jpg.

Página 86

Eliminación y cambio de nombre de archivos

Otras dos funciones útiles que debemos aprender al trabajar con archivos son las funciones remove () y rename (). Estas funciones están disponibles en el módulo del sistema operativo y deben importarse antes de que podamos usarlos.

La función remove () elimina un archivo. La sintaxis es eliminar (nombre de archivo). Por ejemplo, para eliminar myfile.txt, escribimos eliminar ('myfile.txt').

La función rename () cambia el nombre de un archivo. La sintaxis es renombrar (antiguo nombre, nuevo nombre). Para cambiar el nombre de oldfile.txt a newfile.txt, escriba rename ('archivo antiguo.txt', 'archivo nuevo.txt').

Página 87

Proyecto: Matemáticas y BODMAS

¡Felicidades! Ahora hemos cubierto suficientes fundamentos de Python (y programación en general) para comenzar a codificar nuestro primer programa completo. En esto capítulo, vamos a codificar un programa que prueba nuestra comprensión de la regla BODMAS de cálculo aritmético. Si no está seguro de qué BODMAS es, puedes visitar este sitio http://www.mathsisfun.com/operation-order-bodmas.html.

Nuestro programa establecerá aleatoriamente una pregunta aritmética para que la respondamos. Si obtenemos la respuesta incorrecta, el programa mostrará la respuesta correcta y preguntar si queremos probar una nueva pregunta. Si lo hacemos correctamente, el programa nos felicitará y nos preguntará si queremos una nueva pregunta. además, el El programa realizará un seguimiento de nuestros puntajes y guardará los puntajes en un Archivo de texto. Después de cada pregunta, podemos teclear "-1" para terminar el programa.

He dividido el programa en pequeños ejercicios para que pueda probar codificando el programa usted mismo. Pruebe los ejercicios antes de consultar el respuestas. Las respuestas se proporcionan en el Apéndice E o puede ir a http://www.learncodingfast.com/python para descargar los archivos de Python. yo Le recomendamos encarecidamente que descargue el código fuente como el el formateo en el Apéndice E puede resultar en la distorsión de algunas sangrías lo que dificulta la lectura del código.

Recuerde, aprender la sintaxis de Python es fácil pero aburrido. Problema resolver es donde reside la diversión. Si encuentra dificultades al hacer estos ejercicios, esfuércese más. Aquí es donde la recompensa es mayor.

Listo? ¡Vamonos!

Parte 1: myPythonFunctions.py

Escribiremos dos archivos para nuestros programas. El primer archivo es myPythonFunctions.py y el segundo es mathGame.py. La parte 1 céntrese en escribir el código para myPythonFunctions.py.

Para empezar, primero creemos el archivo myPythonFunctions.py. Bien ser definiendo tres funciones en este archivo.

Ejercicio 1: Importación de módulos
Necesitamos importar dos módulos para myPythonFunctions.py: el

módulo aleatorio y el módulo del sistema operativo.

Usaremos la función randint () del módulo aleatorio. los
La función randint () genera un número entero aleatorio dentro del rango
proporcionados por nosotros. Lo usaremos para generar números para nuestras preguntas.
luego.

Desde el módulo del sistema operativo, usaremos remove () y rename ()
funciones.

Intente importar estos dos módulos.

Ejercicio 2: Obtener la puntuación del usuario
Aquí definiremos nuestra primera función. Llamémoslo getUserPoint (). Esta
La función acepta un parámetro, userName. Luego abre el archivo
'userScores.txt' en modo 'r'.

userScores.txt se parece a esto:
Ann, 100
Benny, 102 años
Carol, 214

Página 89

Darren, 129

Cada línea registra la información de un usuario. El primer valor es el usuario
nombre de usuario y el segundo es la puntuación del usuario.

A continuación, la función lee el archivo línea por línea usando un bucle for. Cada línea es
luego dividir usando la función split () (consulte el Apéndice A para
ejemplo sobre el uso de la función split ()).

Guardemos los resultados de la función split () en el contenido de la lista.

A continuación, la función comprueba si alguna de las líneas tiene el mismo nombre de usuario que
el valor que se pasa como parámetro. Si lo hay, la función
cierra el archivo y devuelve la puntuación junto a ese nombre de usuario. Si no lo hay
la función cierra el archivo y devuelve la cadena '-1'.

¿Claro hasta ahora? Intente codificar la función.

¿Hecho?

Ahora tenemos que hacer algunas modificaciones a nuestro código. Al abrir
nuestro archivo anteriormente, usamos el modo 'r'. Esto ayuda a prevenir cualquier
cambios accidentales en el archivo. Sin embargo, al abrir un archivo en modo 'r',

se produce un IOError si el archivo aún no existe. Por lo tanto, cuando corremos el programa por primera vez, terminaremos con un error ya que el archivo userScores.txt no existe previamente. Para evitar este error, puede hacer cualquiera de las siguientes:
En lugar de abrir el archivo en modo 'r', podemos abrirlo en modo 'w'. Cuando abriendo en modo 'w', se creará un nuevo archivo si el archivo no existe previamente. El riesgo de este método es que podamos escribir accidentalmente al archivo, lo que da como resultado que se borre todo el contenido anterior. Sin embargo, desde nuestro programa es un programa pequeño, podemos revisar nuestro código cuidadosamente para evitar cualquier escritura accidental.

Página 90

El segundo método es usar una declaración try, except para manejar el IOError. Para hacer eso, necesitamos poner todos nuestros códigos anteriores en el intento block, luego use excepto IOError: para manejar el error 'Archivo no encontrado'.
En el bloque excepto, informaremos a los usuarios que el archivo no se encuentra y luego proceda a crear el archivo. Usaremos la función open () con el modo 'w' para crearlo. La diferencia aquí es que usamos el modo 'w' solo cuando el archivo está extraviado. Dado que el archivo no existe inicialmente, no hay riesgo de que se borre cualquier contenido anterior. Después de crear el archivo, cierre el archivo y devuelva el cadena “-1”.

Puede elegir cualquiera de los métodos anteriores para completar este ejercicio.
La respuesta proporcionada utiliza el segundo método. Una vez que haya terminado, vamos continúe con el ejercicio 3.

Ejercicio 3: actualización de la puntuación del usuario
En este ejercicio, definiremos otra función llamada updateUserPoints (), que toma tres parámetros: newUser, userName y puntuación.

newUser puede ser Verdadero o Falso. Si newUser es True, la función abrirá el archivo userScores.txt en modo anexar y agregará el nombre de usuario del usuario y puntuación en el archivo cuando sale del juego.

si newUser es False, la función actualizará la puntuación del usuario en el archivo.
Sin embargo, no hay ninguna función en Python (o la mayoría de los lenguajes de programación para el caso) que nos permite actualizar un archivo de texto. Solo podemos escribir o agregarlo, pero no actualizarlo.

Por lo tanto, necesitamos crear un archivo temporal. Esta es una práctica en programación. Llamemos a este archivo userScores.tmp y lo abrimos en modo 'w'. Ahora, necesitaremos recorrer userScore.txt y copiar los datos línea por línea a userScores.tmp. Sin embargo, antes de copiar, compruebe si el nombre de usuario en esa línea es el mismo que el proporcionado como El parámetro. Si es el mismo, cambiaremos la puntuación a la nueva puntuación.

Página 91

antes de escribirlo en el archivo temporal.

Por ejemplo, si los parámetros proporcionados a la función son falsos, 'Benny' y '158' (es decir, updateUserPoints (False, 'Benny', '158')), la siguiente tabla muestra la diferencia entre el original userScores.txt y el nuevo userScores.tmp.

userScores.txt

Ann, 100
Benny, 102 años
Carol, 214
Darren, 129

userScores.tmp

Ann, 100
Benny, 158
Carol, 214
Darren, 129

Una vez que terminemos de escribir en userScore.tmp, cerraremos ambos archivos y eliminar userScores.txt. Finalmente, cambiaremos el nombre de userScores.tmp a userScores.txt.

¿Claro? Intenta codificarlo ...

Ejercicio 4: Generación de preguntas
Ahora llegamos a la parte más importante del programa, generando las preguntas matemáticas. Listo?

Para generar las preguntas, primero declaremos tres variables: dos listas y un diccionario.

Página 92

Denominaremos las dos listas operandList y operatorList.

operandList debe almacenar cinco números, con 0 como sus valores iniciales.

operatorList debe almacenar cuatro cadenas, con "" como valores iniciales.

El diccionario consta de 4 pares, con números enteros del 1 al 4 como diccionario.
claves y “+”, “-”, “ ”, “ *” como datos. Llamemos a este operatorDict.

[Ejercicio 4.1: Actualización de operandList con números aleatorios]

Primero necesitamos reemplazar los valores iniciales de nuestra lista de operandos con números aleatorios generados por la función randint ().

El randint () toma dos parámetros, inicio y final, y devuelve un
entero aleatorio N tal que start <= N <= end.

Por ejemplo, si se llama a randint (1, 9), devolverá aleatoriamente un número entero de los números 1, 2, 3, 4, 5, 6, 7, 8, 9.

Para actualizar nuestra variable operandList con números aleatorios, podemos hacer éste uno por uno ya que operandList solo tiene cinco miembros. Podemos
escribir
operandList [0] = randint (1, 9) operandList [1] =
randint (1, 9) operandList [2] = randint (1, 9)
operandList [3] = randint (1, 9) operandList [4] =
randint (1, 9)
Cada vez que se llama a randint (1, 9), devolverá aleatoriamente un número entero de los números 1, 2, 3, 4, 5, 6, 7, 8, 9.

Sin embargo, esta no es la forma más elegante de actualizar nuestra operandList. Imagínese lo engorroso que será si operandList tiene 1000 miembros.

Página 93

La mejor alternativa es usar un bucle for.

Intente usar un bucle for para realizar la misma tarea.

¿Hecho? ¡Excelente!

[Ejercicio 4.2: Actualización de operatorList con símbolos matemáticos]

Ahora que tenemos los números con los que operar, necesitamos
generar los símbolos matemáticos (+, - , *) para nuestras preguntas. Que hacer
eso, usaremos la función randint () y el operatorDict
diccionario.

randint () generará la clave del diccionario, que luego se mapeará
al operador correcto utilizando el diccionario operatorDict. Por ejemplo,

para asignar el símbolo a operatorList [0], escribimos
operatorList [0] = operatorDict [randint (1, 4)]

Al igual que en el ejercicio 4.1, debe utilizar un bucle for para completar esta tarea. Sin embargo, hay un problema que hace que este ejercicio sea más difícil que Ejercicio 4.1.

¿Recuerda que en Python, ** representa exponente (es decir, 2 ** 3 = 2 ^ 3)?

El problema es que cuando tenemos dos operadores exponentes consecutivos en Python, como 2 ** 3 ** 2, Python lo interpreta como 2 ** (3 ** 2) en lugar de (2 ** 3) ** 2. En el primer caso, la respuesta es 2 elevado a 9 (es decir, 2^9) que es 512. En el segundo caso, la respuesta es 8 elevado a 2 (es decir, 8^2) que es 64. Por lo tanto, cuando presentamos una pregunta como 2 ** 3 ** 2, el usuario obtendrá la respuesta incorrecta si la interpreta como (2 ** 3) ** 2.

Página 94

Para evitar este problema, vamos a modificar nuestro código para que no obtener dos signos ** consecutivos. En otras palabras, operatorList = ['+', '+', '-', '**'] está bien, pero operatorList = ['+', '-', '**', '**'] no es.

Este ejercicio es el más difícil de todos los ejercicios. Intenta pensar una solución para prevenir dos signos ** consecutivos. Una vez que haya terminado, puede continuar con el ejercicio 4.3.

Sugerencia: si está atascado, puede considerar usar una instrucción if dentro del en bucle.

[Ejercicio 4.3: Generación de una expresión matemática]

Ahora que tenemos nuestros operadores y operandos, vamos a intentar generar la expresión matemática como una cadena. Esta expresión usuarios los cinco números de nuestra lista de operandos y los cuatro números matemáticos símbolos de nuestra operatorList para formar una pregunta.

Tenemos que declarar otra variable llamada questionString y asigne la expresión matemática a questionString. Ejemplos de questionString incluye
6 - 2 * 3 - 2 ** 1
4 + 5 - 2 * 6 + 1
8 - 0 * 2 + 5 - 8

Intente generar esta expresión usted mismo.

Sugerencia: puede usar un bucle for para concatenar las subcadenas individuales de operandList y operatorList para obtener la matemática expresión.

[Ejercicio 4.4: Evaluación del resultado]

Página 95

Ahora deberíamos tener una expresión matemática como una cadena, asignada a la variable questionString. Para evaluar el resultado de esta expresión, vamos a utilizar una función incorporada brillante que viene con Python, eval ().

eval () interpreta una cadena como un código y ejecuta el código. por Por ejemplo, si escribimos eval ("1 + 2 + 4"), obtendremos el número 7.

Por lo tanto, para evaluar el resultado de nuestra expresión matemática, pasamos questionString a la función eval () y asigne el resultado a una nueva variable denominada resultado.

Este ejercicio es bastante sencillo y se puede completar en un solo paso.

[Ejercicio 4.5: Interacción con el usuario]

Finalmente, vamos a interactuar con nuestro usuario. En este ejercicio, estaremos haciendo algunas cosas:
Paso 1: Mostrar la pregunta al usuario Paso 2: Solicitar al usuario una respuesta Paso 3: Evaluar la respuesta, mostrando el mensaje y devolviendo la puntuación del usuario.

Para el paso 1, necesitamos usar una función incorporada para manipular cadenas. Como mencionado anteriormente, en Python, el símbolo ** significa exponente. Es decir, 2 ** 3 = 8. Sin embargo, para la mayoría de los usuarios, ** no tiene ningún significado. Por lo tanto, si mostramos una pregunta como 2 ** 3 + 8 -5, es probable que el usuario se confunda. Para evitar eso, reemplazaremos cualquier símbolo ** en questionString con el símbolo ^.

Para hacer eso, usaremos la función incorporada replace (). Usarlo es bonito sencillo, solo escribe questionString = questionString.replace ("**", "^"). Ahora puede imprimir el expresión resultante para el usuario.

Página 96

Para el paso 2, puede utilizar la función input () para aceptar la entrada del usuario.

Para el paso 3, debe usar una declaración if para evaluar la respuesta y
mostrar el mensaje correcto. Si el usuario lo acierta, lo felicitaremos
el usuario y devuelve el valor 1. Si el usuario se equivoca, mostraremos el
respuesta correcta y devuelve el valor 0.

¿Recuerda que la función input () devuelve la entrada del usuario como una cadena? Por lo tanto,
cuando compara la entrada del usuario con la respuesta correcta (obtenida en
Ejercicio 4.4), tienes que hacer algún tipo de conversión para cambiar la entrada del usuario
a un entero. Al cambiar la entrada del usuario a un número entero, debe usar
un intento, excepto una declaración para verificar si el usuario ingresó un número. Si el
usuario escribió una cadena en su lugar, el programa debe informar al usuario de la
error y pedirle al usuario que escriba un número.

Puede usar un ciclo while True para seguir solicitando al usuario un
número siempre que no lo haga. Escribir mientras True es
equivalente a escribir algo como while 1 == 1. Dado que 1 es siempre
es igual a 1 (por lo tanto, siempre verdadero), el ciclo se ejecutará indefinidamente.

Aquí hay una sugerencia sobre cómo puede usar un ciclo while True para esto
ejercicio.

```
mientras es cierto:
    tratar:
        enviar la respuesta del usuario a un número entero y evaluar
    la respuesta devuelve la puntuación del usuario basada en la respuesta
    excepto:
        Imprimir mensaje de error si falla la transmisión.
    que el usuario vuelva a introducir la respuesta
```

El bucle while True seguirá repitiendo ya que la condición while es
siempre cierto. El ciclo saldrá solo cuando se ejecute el bloque try.
correctamente y llega a la declaración de devolución.

Página 97

Prueba este ejercicio. Una vez que haya terminado, podemos pasar a la Parte 2 donde
escribimos el programa real.

Página 98

Parte 2: mathGame.py

Felicitaciones por completar la Parte 1 y bienvenido a la Parte 2. La Parte 2 es
será muy sencillo, ya que principalmente llamaremos a las funciones que
definido anteriormente.

Ejercicio 5: redacción del programa principal
Primero, adjuntemos nuestro programa principal en un intento, excepto declaración. Nosotros
desea manejar cualquier error imprevisto al ejecutar el programa principal.

Comenzaremos escribiendo el código para el bloque try.

En primer lugar, necesitamos importar el módulo myPythonFunctions. A continuación, vamos
solicitar al usuario su nombre de usuario y asignar el valor a la variable
nombre de usuario. Pase esta variable como parámetro a la función
getUserScore ().

getUserScore () devolverá la puntuación del usuario o devolverá '-1' (si
el usuario no se encuentra). Vamos a convertir este resultado en un número entero y asignarlo a

la variable userScore.

Ahora, necesitamos establecer el valor de otra variable newUser. Si el usuario es no encontrado, newUser = True, de lo contrario newUser = False. Si newUser = Es cierto que tenemos que cambiar userScore de -1 a 0.

La siguiente parte de nuestro programa implica un ciclo while. Específicamente, nuestro El programa solicitará la entrada de nuestro usuario para determinar si debe terminar el programa o hacer otra cosa.

Paso 1:
Necesita declarar otra variable userChoice y darle una inicial valor de 0.

Página 99

Paso 2:
Luego, usando un ciclo while, compare userChoice con una cadena de su opción, diga “-1”. Si userChoice no es lo mismo que "-1", llama a la función generateQuestion () para generar una nueva pregunta.

Paso 3:
generateQuestion () devolverá la puntuación que el usuario obtuvo por ese pregunta. Utilice este resultado para actualizar la variable userScore.

Paso 4:
Finalmente, para evitar un bucle infinito, necesitamos usar input () funcionar de nuevo dentro del ciclo while para aceptar la entrada del usuario y usarla para actualice el valor de userChoice.

¿Lo tengo? Intente codificarlo. Hacer la codificación real hará que todo más claro.

Finalmente, después de que termina el ciclo while, el siguiente paso es actualizar el archivo userScores.txt. Para hacer eso, simplemente llamamos al función updateUserPoints ().

Eso es todo por el bloque try. Ahora para el bloque excepto, simplemente informamos el usuario que se ha producido un error y el programa se cerrará.

¡Eso es! Una vez que termine este paso, tendrá un programa completo, su primer programa en Python. Intente ejecutar el programa mathGame.py. Lo hace funciona como se esperaba? ¿Emocionado? Espero que estés tan emocionado como yo. a.m. :)

Página 100

Retarte a ti mismo

Hemos llegado al final de este capítulo y esperamos que haya
codificó con éxito su primer programa. Si tiene problemas para completar
cualquier ejercicio, puede estudiar las respuestas en el Apéndice E. Aprenderá
mucho estudiando los códigos de otras personas.

En esta sección, tengo tres ejercicios adicionales para desafiar
usted mismo.

Ejercicio de desafío 1

En el programa que hemos codificado hasta ahora, evité usar la división
operador. ¿Puede modificar el programa para que genere preguntas con
¿el signo de división también? ¿Cómo compararía la respuesta del usuario con
¿La respuesta correcta?

Sugerencia: compruebe la función round ().

Ejercicio de desafío 2

A veces, la pregunta generada puede resultar en una respuesta muy
grandes o muy pequeños. Por ejemplo, la pregunta 6 * (8 ^ 9/1) ^ 3 dará el
responda 145071098537555009647411 2.

Es muy inconveniente para los usuarios calcular e ingresar un tamaño tan grande
número. Por lo tanto, queremos evitar respuestas que sean demasiado grandes o pequeñas. lata
Modificas el programa para evitar preguntas que resulten en respuestas.
mayor que 50 000 o menor que -50000?

Ejercicio de desafío 3

Página 101

El último ejercicio de desafío es el más difícil.

Hasta ahora faltan corchetes en las preguntas generadas. Puedes modificar el programa para que las preguntas también utilicen corchetes? Un ejemplo de La pregunta será 2 + (3 * 7 -1) + 5.

Diviértete con estos ejercicios. La solución sugerida se proporciona en Apéndice E.

Página 102

Gracias

Llegamos al final del libro. Gracias por leer este libro y yo Espero que hayas disfrutado del libro. Más importante aún, espero sinceramente este libro le ha ayudado a dominar los fundamentos de la programación Python.

Sé que podrías haber elegido entre una docena de libros sobre Python Programación, pero se arriesgó con este libro. Gracias una vez nuevamente por descargar este libro y leerlo hasta el final. Pruebe los ejercicios y los desafíos. Aprenderás mucho haciendo.

Ahora me gustaría pedir un "pequeño" favor. ¿Podrías tomar algunos

minutos o dos para dejar una reseña de este libro en Amazon?

Estos comentarios me ayudarán enormemente y me ayudarán a continuar escribir más guías sobre programación. Si te gusta el libro o tienes alguna sugerencias de mejora, hágamelo saber. Estaré profundamente agradecido. :)

Por último, pero no menos importante, recuerde que puede descargar el código fuente del proyecto y los apéndices en http://www.learncodingfast.com/python .

También puedes contactarme al jamie@learncodingfast.com.

Página 103

Apéndice A: Trabajar con cadenas

Nota: La notación [inicio, [fin]] significa que el *inicio* y el *final* son opcionales parámetros. Si solo se proporciona un número como parámetro, se toma para *empezar* .

marca el comienzo de un comentario
" 'marca el inicio y el final de un comentario de varias líneas El código real está en fuente monotype.
=> marca el inicio de la salida
contar (sub, [inicio, [final]])

Devuelve el número de veces que la subcadena *sub* aparece en la cadena. Esta función distingue entre mayúsculas y minúsculas.

[Ejemplo]

En los ejemplos siguientes, 's' aparece en los índices 3, 6 y 10

contar toda la cadena
'Esto es una cadena'.count (' s ')
=> 3

contar desde el índice 4 hasta el final de la cadena 'Esto es una cadena'.count (' s ', 4) => 2

contar desde el índice 4 hasta 10-1
'Esto es una cadena'.count (' s ', 4, 10) => 1

cuente 'T'. Solo hay 1 'T' ya que la función distingue entre mayúsculas y minúsculas.
'Esto es una cadena'.count (' T ')
=> 1

Página 104

termina con (sufijo, [inicio, [final]])
Devuelve True si la cadena termina con el *sufijo* especificado ; de lo contrario, devuelve Falso.
sufijo también puede ser una tupla de sufijos para buscar.
Esta función distingue entre mayúsculas y minúsculas.

[Ejemplo]

'hombre' aparece en el índice 4 a 6

revisa toda la cadena
'Cartero'.endswith (' hombre ')
=> Verdadero

revisar desde el índice 3 hasta el final de la cadena 'Postman'.endswith (' man ', 3)
=> Verdadero

comprobar del índice 2 al 6-1
'Postman'.endswith (' man ', 2, 6) => Falso

comprobar del índice 2 al 7-1
'Postman'.endswith (' man ', 2, 7) => Verdadero

Usando una tupla de sufijos (verifique del índice 2 al 6-1)
'Postman'.endswith ((' man ',' ma '), 2, 6) => Verdadero

buscar / índice (sub, [inicio, [final]])
Devuelve el índice en la cadena donde la primera ocurrencia de la subcadena se encuentra *sub* .
find () devuelve -1 si no se encuentra *sub* .
index () devuelve ValueError is *sub* no se encuentra.
Esta función distingue entre mayúsculas y minúsculas.

[Ejemplo]

Página 105

```
# revisa toda la cadena
'Esto es una cadena'.find (' s ')
=> 3

# comprobar desde el índice 4 hasta el final de la cadena 'Esto es una cadena'.find (' s ', 4) => 6

# comprobar del índice 7 al 11-1
'Esto es una cadena'.find (' s ', 7,11) => 10

# Sub no se encuentra
'Esto es una cadena'.find (' p ')
=> -1

'Esto es una cadena'.index (' p ')
=> ValueError
```

isalnum ()

Devuelve verdadero si todos los caracteres de la cadena son alfanuméricos y hay en
al menos un carácter, falso en caso contrario.
Alfanumérico no incluye espacios en blanco.

[Ejemplo]

```
'abcd1234'.isalnum ()
=> Verdadero

'a b c d 1 2 3 4'.isalnum ()
=> Falso

'abcd'.isalnum ()
=> Verdadero
```

Página 106

```
'1234'.isalnum ()
=> Verdadero
```

isalpha ()

Devuelve verdadero si todos los caracteres de la cadena son alfabéticos y hay en al menos un carácter, falso en caso contrario.

[Ejemplo]

'abcd'.isalpha ()
=> Verdadero

'abcd1234'.isalpha ()
=> Falso

'1234'.isalpha ()
=> Falso

'a b c'.isalpha ()
=> Falso

isdigit ()

Devuelve verdadero si todos los caracteres de la cadena son dígitos y hay al menos un carácter, falso en caso contrario.

[Ejemplo]

'1234'.isdigit ()
=> Verdadero

Página 107

'abcd1234'.isdigit ()
=> Falso

'abcd'.isdigit ()
=> Falso

'1 2 3 4'. Dígitos ()
=> Falso

es bajo()

Devuelve verdadero si todos los caracteres en mayúscula de la cadena están en minúsculas y hay al menos un carácter en mayúscula, falso en caso contrario.

[Ejemplo]

'abcd'.islower ()
=> Verdadero

'Abcd'.islower ()
=> Falso

'ABCD'.islower ()
=> Falso

isspace ()

Devuelve verdadero si solo hay espacios en blanco en la cadena y
es al menos un carácter, falso en caso contrario.

[Ejemplo]

Página 108

" .isspace ()
=> Verdadero

'a b'.isspace ()
=> Falso

istitle ()

Devuelve verdadero si la cadena es una cadena con título y hay al menos una
personaje
[Ejemplo]

'Esto es una cadena'.istitle ()
=> Verdadero

'Esto es una cadena'.istitle ()
=> Falso

isupper ()

Devuelve verdadero si todos los caracteres en mayúsculas de la cadena están en mayúsculas y no
es al menos un carácter en mayúsculas, de lo contrario es falso.

[Ejemplo]

'ABCD'.isupper ()
=> Verdadero

'Abcd'.isupper ()
=> Falso

'abcd'.isupper ()

Página 109

=> Falso

unirse()

Devuelve una cadena en la que el parámetro proporcionado está unido por un separador.

[Ejemplo]

```
sep = '-'
myTuple = ('a', 'b', 'c')
myList = ['d', 'e', 'f']
myString = "Hola mundo"

sep.unión (myTuple)
=> 'ab-c'

sep.unión (myTuple)
=> 'de-f'

sep.join (myString)
=> 'Hola- -Worl-d' '
```

inferior()

Devuelve una copia de la cadena convertida a minúsculas.

[Ejemplo]

```
'Hola Python'.lower ()
=> 'hola pitón'
```

reemplazar (antiguo, nuevo [, contar])

Página 110

Devuelve una copia de la cadena con todas las apariciones de la subcadena anterior reemplazada por nuevo.
el recuento es opcional. Si se proporciona, solo se reemplazan las primeras apariciones del *recuento* .
Esta función distingue entre mayúsculas y minúsculas.

[Ejemplo]

```
# Reemplazar todas las ocurrencias
'Esto es una cadena'.replace (' s ',' p ') =>' Thip ip a
ptring '

# Reemplazar las 2 primeras ocurrencias
'Esto es una cadena'.replace (' s ',' p ', 2) =>' Thip ip a
cuerda'
```

dividir ([sep [, maxsplit]])

Devuelve una lista de las palabras de la cadena, usando *sep* como cadena delimitadora.
sep y *maxsplit* son opcionales.
Si no se proporciona *sep* , se utiliza un espacio en blanco como delimitador.
Si se da *maxsplit* , como máximo se *realizan* divisiones de *maxsplit* .
Esta función distingue entre mayúsculas y minúsculas.

[Ejemplo]

```
" '
Dividir usando una coma como delimitador Observe que hay un espacio antes del
palabras 'es', 'a' y 'cadena' en la salida.
" '
'Esto, es, una, cadena'.split (', ') => [' Esto ',' es ','
una cuerda']

# Dividir usando espacios en blanco como delimitador 'Esto es una cadena'. Dividir ()
```

Página 111

```
=> ['Esto', 'es', 'a', 'cadena']

# Solo haz 2 divisiones
'Esto, es, una, cadena'.split (', '2) => [' Esto ',' es ',
' una cuerda']
```

splitlines ([keepends])

Devuelve una lista de las líneas de la cadena, rompiendo en los límites de las líneas.
Los saltos de línea no se incluyen en la lista resultante a menos que se indique *keepends*
y verdadero.

[Ejemplo]

Líneas divididas separadas por \ n
'Esta es la primera línea. \ NEsta es la segunda
line'.splitlines () => ['Esta es la primera línea.',
'Esta es la segunda línea.']

Dividir cadena de varias líneas (por ejemplo, cadena que usa la marca " ')' " Esta es la primera línea.
Esta es la segunda línea. " '. Splitlines () => [' Esto es
la primera línea. ',' Esta es la segunda línea. ']

Dividir y mantener saltos de línea
'Esta es la primera línea. \ NEsta es la segunda
line. '. splitlines (True) => [' Este es el primer
line. \ n ',' Esta es la segunda línea. ']

" 'Esta es la primera línea.
Esta es la segunda línea. " '. Splitlines (True) => [' This
es la primera línea. \ n ',' Esta es la segunda línea. ']

empieza con (prefijo [, inicio [, final]])

Página 112

Devuelve True si la cadena comienza con el prefijo; de lo contrario, devuelve False.
prefijo también puede ser una tupla de prefijos a buscar.
Esta función distingue entre mayúsculas y minúsculas.

[Ejemplo]

'Publicar' ocurre en el índice 0 a 3

revisa toda la cadena
'Postman'.startswith (' Publicar ')
=> Verdadero

comprobar desde el índice 3 hasta el final de la cadena 'Cartero'. comienza con ('Publicar', 3) => Falso

comprobar del índice 2 al 6-1
'Postman'.startswith (' Publicar ', 2, 6) => Falso

comprobar del índice 2 al 6-1
'Postman'.startswith (**' stm '** , 2, 6) => Verdadero

Usando una tupla de prefijos (verifique desde el índice 3 hasta el final de la cadena)
'Postman'.startswith ((' Publicar ',' tma '), 3) => Verdadero

tira ([caracteres])

Devuelve una copia de la cadena con los caracteres *iniciales* y finales *char*
remoto.
Si no se proporciona *char* , se eliminarán los espacios en blanco.
Esta función distingue entre mayúsculas y minúsculas.

[Ejemplo]

Página 113

Eliminar espacios en blanco
'Esto es una cadena' .strip () => 'Esto es una cadena'

Tiras'. No se elimina nada ya que 's' no está al principio o al final de la
cadena 'Esto es una cadena'.strip (' s ')
=> 'Esto es una cadena'

Pele 'g'.
'Esto es una cadena'.strip (' g ')
=> 'Esto es un strin'

Superior()

Devuelve una copia de la cadena convertida a mayúsculas.

[Ejemplo]

'Hola Python'.upper ()
=> 'HOLA PYTHON'

Página 114

Apéndice B: Trabajar con listas

=> marca el inicio de la salida
adjuntar( )

Agregar elemento al final de una lista
[Ejemplo]

myList = ['a', 'b', 'c', 'd']
myList.append ('e') print (myList)
=> ['a', 'b', 'c', 'd', 'e']

del

Quitar elementos de una lista
[Ejemplo]

myList = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l']

eliminar el tercer elemento (índice = 2) de myList [2]
imprimir (myList)
=> ['a', 'b', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l']

eliminar elementos del índice 1 al 5-1
del myList [1: 5]
imprimir (myList)
=> ['a', 'g', 'h', 'i', 'j', 'k', 'l']

eliminar elementos del índice 0 al 3-1
del myList [: 3]
imprimir (myList)
=> ['i', 'j', 'k', 'l']

Página 115

#delete items from index 2 to end del myList [2:]
imprimir (myList)
=> ['i', 'j']

extender ()

Combinar dos listas

[Ejemplo]

myList = ['a', 'b', 'c', 'd', 'e']
myList2 = [1, 2, 3, 4]
myList.extend (myList2) print (myList)
=> ['a', 'b', 'c', 'd', 'e', 1, 2, 3, 4]

En

Compruebe si un artículo está en una lista
[Ejemplo]

myList = ['a', 'b', 'c', 'd']
'c' en myList
=> Verdadero

'e' en myList
=> Falso

insertar ()

Agregar artículo a una lista en una posición particular
[Ejemplo]

Página 116

myList = ['a', 'b', 'c', 'd', 'e']
myList.insert (1, 'Hola') print (myList)
=> ['a', 'Hola', 'b', 'c', 'd', 'e']

len ()

Encuentra la cantidad de elementos en una lista
[Ejemplo]

myList = ['a', 'b', 'c', 'd']
print (len (myList)) => 4

pop ()

Obtener el valor de un elemento y eliminarlo de la lista Requiere índice de elemento como parámetro
[Ejemplo]

```
myList = ['a', 'b', 'c', 'd', 'e']

# eliminar el tercer elemento miembro = myList.pop (2) print (miembro)
=> c

imprimir (myList)
=> ['a', 'b', 'd', 'e']

#remove the last item member = myList.pop () print (miembro)
=> e

imprimir (myList)
=> ['a', 'b', 'd']
```

eliminar( )

Página 117

Eliminar un elemento de una lista. Requiere el valor del artículo como parámetro.

[Ejemplo]

```
myList = ['a', 'b', 'c', 'd', 'e']

#remove el elemento 'c'
myList.remove ('c') print (myList)
=> ['a', 'b', 'd', 'e']
```

contrarrestar()

Invertir los elementos de una lista
[Ejemplo]

```
myList = [1, 2, 3, 4]
myList.reverse ()
imprimir (myList)
=> [4, 3, 2, 1]
```

ordenar()

Ordenar una lista alfabética o numéricamente
[Ejemplo]

```
myList = [3, 0, -1, 4, 6]
myList.sort ()
imprimir (myList)
=> [-1, 0, 3, 4, 6]
```

ordenado ()

Página 118

Devuelve una nueva lista ordenada sin ordenar la lista original.
Requiere una lista como parámetro
[Ejemplo]

```
myList = [3, 0, -1, 4, 6]
myList2 = ordenado (myList)
# La lista original no está ordenada imprimir (myList)
=> [3, 0, -1, 4, 6]

# La nueva lista está ordenada print (myList2)
=> [-1, 0, 3, 4, 6]
```

Operador de adición: +

Lista de concatenar

[Ejemplo]

```
myList = ['a', 'b', 'c', 'd']
print (myList + ['e', 'f']) => ['a', 'b', 'c', 'd',
'e', 'f']

imprimir (myList)
=> ['a', 'b', 'c', 'd']
```

Operador de multiplicación: *

Duplicar una lista y concatenarla al final de la lista
[Ejemplo]

```
myList = ['a', 'b', 'c', 'd']
imprimir (myList * 3)
```

Página 119

=> ['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']

imprimir (myList)
=> ['a', 'b', 'c', 'd']

Nota:
Los símbolos + y * no modifican la lista. La lista permanece como ['a', 'b', 'c', 'd'] en ambos casos.

Página 120

Apéndice C: Trabajar con tuplas

=> marca el inicio de la salida
del

Eliminar toda la tupla
[Ejemplo]

myTuple = ('a', 'b', 'c', 'd') del myTuple
print (myTuple) => NameError: el nombre 'myTuple' no es definido
en

Compruebe si un elemento está en una tupla
[Ejemplo]

myTuple = ('a', 'b', 'c', 'd') 'c' en myTuple => Verdadero

'e' en myTuple => False

len ()

Encuentra la cantidad de elementos en una tupla
[Ejemplo]

myTuple = ('a', 'b', 'c', 'd') print (len (myTuple)) => 4

Operador de adición: +

Página 121

Concatenar tuplas
[Ejemplo]

myTuple = ('a', 'b', 'c', 'd') print (myTuple + ('e', 'f')) => ('a', 'b', 'c', 'd', 'e', 'f')
print (myTuple) => ('a', 'b', 'c', 'd')
Operador de multiplicación: *

Duplica una tupla y concatenala al final de la tupla
[Ejemplo]

myTuple = ('a', 'b', 'c', 'd') print (myTuple * 3) => ('a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'discos compactos')
print (myTuple) => ('a', 'b', 'c', 'd')

Nota: Los símbolos + y * no modifican la tupla. La tupla se queda como ['a', 'b', 'c', 'd'] en ambos casos.

Apéndice D: Trabajar con diccionarios

=> marca el inicio de la salida
claro( )

Elimina todos los elementos del diccionario y devuelve un diccionario vacío.
[Ejemplo]

```
dic1 = {1: 'uno', 2: 'dos'}
imprimir (dic1)
=> {1: 'uno', 2: 'dos'}

dic1.clear ()
imprimir (dic1)
=> {}
```

del

Eliminar todo el diccionario
[Ejemplo]

```
dic1 = {1: 'uno', 2: 'dos'}
del dic1
imprimir (dic1)
=> NameError: el nombre 'dic1' no está definido
```

obtener( )

Devuelve un valor para la clave dada.
Si no se encuentra la clave, devolverá la palabra clave None.
Alternativamente, puede indicar el valor que se devolverá si no se encuentra la clave.

[Ejemplo]

Página 123

dic1 = {1: 'uno', 2: 'dos'}
dic1.get (1)
=> 'uno'

dic1.get (5)
=> Ninguno

dic1.get (5, "No encontrado") => 'No encontrado'

En

Compruebe si un elemento está en un diccionario
[Ejemplo]

dic1 = {1: 'uno', 2: 'dos'}

basado en la clave 1 en dic1
=> Verdadero

3 en dic1
=> Falso

basado en el valor 'uno' en dic1.values () => Verdadero

'tres' en dic1.values () => Falso

artículos( )

Devuelve una lista de pares de diccionario como tuplas.
[Ejemplo]

Página 124

```
dic1 = {1: 'uno', 2: 'dos'}
dic1.items ()
=> dict_items ([(1, 'uno'), (2, 'dos')])
```

llaves( )

Devuelve la lista de claves del diccionario.
[Ejemplo]

```
dic1 = {1: 'uno', 2: 'dos'}
dic1.keys ()
=> dict_keys ([1, 2])
```

len ()

Encuentra la cantidad de elementos en un diccionario
[Ejemplo]

```
dic1 = {1: 'uno', 2: 'dos'}
imprimir (len (dic1)) => 2
```

actualizar ()

Agrega los pares clave-valor de un diccionario a otro. Los duplicados son
remoto.

[Ejemplo]

```
dic1 = {1: 'uno', 2: 'dos'}
dic2 = {1: 'uno', 3: 'tres'}

dic1.update (dic2) imprimir (dic1)
=> {1: 'uno', 2: 'dos', 3: 'tres'}

print (dic2) #no change => {1: 'uno', 3: 'tres'}
```

Página 125

valores( )

Devuelve la lista de los valores del diccionario.
[Ejemplo]

```
dic1 = {1: 'uno', 2: 'dos'}
dic1.values ()
```

```
=> dict_values (['uno', 'dos'])
```


Apéndice E: Respuestas del proyecto

Ejercicio 1

```
de randint de importación aleatoria
de la importación del sistema operativo eliminar, cambiar el nombre
```

Ejercicio 2

```
def getUserScore (nombre de usuario):


    tratar:
        input = open ('puntuaciones de usuario.txt', 'r')
        para entrada de línea:
            contenido = línea.split (',')
            si el contenido [0] == nombre de usuario:
                input.close ()
                devolver contenido [1]
        input.close ()
```

```
            devuelve "-1"
        excepto IOError:
            print ("\ nFile userScores.txt no encontrado. Un nuevo
        se creará el archivo. ") input =
        abierto ('userScores.txt', 'w')
            input.close ()
            devuelve "-1"
```

Ejercicio 3

```
def updateUserPoints (newUser, userName, score):
    si es newUser:
        input = open ('userScores.txt', 'a')
        input.write ('\ n' + nombre de usuario + ',' + puntuación)
    input.close ()
```

Página 127

```
    más:
        input = open ('puntuaciones de usuario.txt', 'r')
        salida = abierto ('userScores.tmp', 'w')
    para entrada de línea:
        contenido = línea.split (',')
        si el contenido [0] == nombre de usuario:
            contenido [1] = puntuación
            línea = contenido [0] + ',' + contenido [1] +
        '\norte'

        output.write (línea)
    input.close ()
    output.close ()


    eliminar ('userScores.txt')
    renombrar ('userScores.tmp', 'userScores.txt')
```

Ejercicio 4

```
def generateQuestion ():


    operandList = [0, 0, 0, 0, 0]
    operatorList = ['', '', '', '']
    operatorDict = {1: '+', 2: '-', 3: ' ', 4: ' *'}

    para el índice en el rango (0, 5):
        operandList [índice] = randint (1, 9)

    para índice en rango (0, 4):
        if index> 0 y operatorList [index-1]! =
'**': operator = operatorDict [randint (1, 4)]
    else: operator =
```

operatorDict [randint (1, 3)]

Página 128

```
        operatorList [índice] = operador
        questionString = str (operandList [0])

        para índice en rango (1, 5):
        questionString = questionString +
operatorList [índice-1] + str (operandList [índice])
        resultado = eval (questionString)

        questionString = questionString.replace ("**", "^")
        print ('\ n' + questionString)

        userResult = input ('Respuesta:')

        mientras es cierto:
        tratar:
        if int (userResult) == resultado:
print ("So Smart") return 1
        else: print ("Lo siento, mal
responder. La respuesta correcta es ", resultado)
volver 0
        excepto Exception como e: print ("Lo hiciste
no ingrese un número. Inténtalo de nuevo.")
userResult = input ('Respuesta:')
```

[Explicación del ejercicio 4.2]

A partir del segundo elemento (es decir, índice = 1) en operatorList, la línea if index> 0 y operatorList [index-1]! = '**': comprueba si el elemento anterior en operatorList es el símbolo '**'.

Si no es así, la instrucción operator = operatorDict [randint (1, 4)] se ejecutará. Dado que el rango dado a la función randint es de 1 a 4, se generarán los números 1, 2, 3 o 4. Por lo tanto, los símbolos '+', '-', ' ' *o* ' *' se asignará al operador de variable.

Página 129

Sin embargo, si el símbolo anterior es '**', la instrucción else (operador = operatorDict [randint (1, 3)]) se ejecutará. En este caso, el rango dado a la función randint es de 1 a 3. Por lo tanto, el símbolo '**', que tiene una clave de 4 en operatorDict NO se asignará al variable de operador.

Ejercicio 5

tratar:

```
        importar myPythonFunctions como m

        userName = input (" 'Por favor ingrese su nombre de usuario o
crea uno nuevo si es la primera vez que estás
ejecutando el programa: " ')

        userScore = int (m.getUserScore (nombre de usuario))
        si userScore == -1:
        newUser = True userScore = 0
        más:
        newUser = Falso
        userChoice = 0

        while userChoice! = '-1':

        userScore + = m.generateQuestion () imprimir
("Puntuación actual =", userScore) userChoice =
entrada ("Presione Enter para continuar o -1 para salir:")
        m.updateUserPoints (newUser, userName,
str (userScore))
excepto la excepción como e:
        print ("Ocurrió un error inesperado. El programa
salir. ")
```

Página 130

Retarte a ti mismo

Solo necesita cambiar la función generateQuestion () para todos los desafíos. Aquí está la solución sugerida.

```
def generateQuestion ():
        operandList = [0, 0, 0, 0, 0]
        operatorList = ['', '', '', '']
        operatorDict = {1: '+', 2: '-', 3: ' ', 4: '/',
5: ' *'}

        resultado = 500001
```

```
        while result> 50000 or result <-50000: for index
en rango (0, 5): operandList [índice] = randint (1,
9)
        para índice en rango (0, 4): si índice> 0
y operatorList [index-1]! = '**':
operador = operatorDict [randint (1, 4)]
        else: operator =
operatorDict [randint (1, 5)]
        operatorList [índice] = operador
                " '
                Genere aleatoriamente las posiciones de (y)
                Por ejemplo, si openBracket = 2, el símbolo (será
                colocado delante del tercer número Si
                closeBracket = 3, se colocará el símbolo)
                detrás del cuarto número Desde el cierre
                el soporte no puede estar antes del soporte de apertura,
                tenemos que generar la posición para el
                soporte de cierre de openBracket + 1 en adelante
                " '

        openBracket = randint (0, 3) closeBracket =
randint (openBracket + 1, 4)
```

Página 131

```
        si openBracket == 0: questionString =
'(' + str (operandList [0]) else: questionString =
str (operandList [0])
        para índice en rango (1, 5): si índice ==
openBracket: questionString =
questionString + operatorList [index-1] + '(' +
str (operandList [índice]) elif índice ==
closeBracket: questionString =
questionString + operatorList [índice-1] +
str (operandList [índice]) + ')'
        else: questionString =
questionString + operatorList [índice-1] +
str (operandList [índice])
        resultado = ronda (eval (questionString), 2)
        #Fin del bucle while
        questionString = questionString.replace ("**", "^")
        print ('\ n' + questionString)


        userResult = input ('Responder (corregir a 2 dp si
no es un entero): ')
        mientras es cierto:
        tratar:
        if float (userResult) == resultado:
print ("So Smart") return 1
        else: print ("Lo siento, mal
responder. La respuesta correcta es ", resultado)
```

```
volver 0
        excepto Exception como e: print ("Lo hiciste
no ingrese un número. Inténtalo de nuevo.")
userResult = input ('Responder (corregir a 2 dp si
no es un entero): ')
```

Página 132

Una última cosa…

Cuando pase la página, Amazon le pedirá que califique este libro y comparte tus pensamientos en Facebook y Twitter.

Si esta guía le ha ayudado, le agradecería profundamente que tómese unos segundos para que sus amigos lo sepan.

Para mí, la programación es un arte y una ciencia. Es muy adictivo y agradable. Espero compartir esta pasión con tantas personas como posible.

Además, espero que no dejes de aprender aquí. Si usted está interesado en más desafíos de programación, puede consultar el sitio https://projecteuler.net/ . ¡Que te diviertas!