Chapter 1

¿Por qué deberías aprender a escribir programas?

Escribir programas (o programar) es una actividad muy creativa y gratificante. Puedes escribir programas por muchas razones, que pueden ir desde mantenerte activo resolviendo un problema de análisis de datos complejo hasta hacerlo por pura diversión ayudando a otros a resolver un enigma. Este libro asume que *todo el mundo* necesita saber programar, y que una vez que aprendas a programar ya encontrarás qué quieres hacer con esas habilidades recién adquiridas.

En nuestra vida diaria estamos rodeados de computadoras, desde equipos portátiles (laptops) hasta teléfonos móviles (celulares). Podemos pensar en esas computadoras como nuestros "asistentes personales", que pueden ocuparse de muchas tareas por nosotros. El hardware en los equipos que usamos cada día está diseñado esencialmente para hacernos la misma pregunta de forma constante, "¿Qué quieres que haga ahora?"

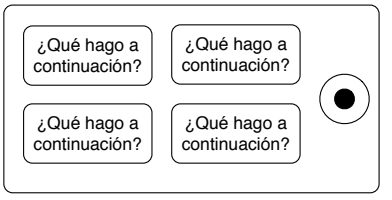

Figure 1.1: Personal Digital Assistant

Los programadores suelen añadir un sistema operativo y un conjunto de aplicaciones al hardware y así nos proporcionan un Asistente Digital Personal que es bastante útil y capaz de ayudarnos a realizar una gran variedad de tareas.

Nuestros equipos son rápidos y tienen grandes cantidades de memoria. Podrían resultarnos muy útiles si tan solo supiéramos qué idioma utilizar para explicarle a la computadora qué es lo que queremos que "haga ahora". Si conociéramos ese idioma, podríamos pedirle al aparato que realizase en nuestro lugar, por ejemplo, tareas repetitivas. Precisamente el tipo de cosas que las computadoras saben hacer mejor suelen ser el tipo de cosas que las personas encontramos pesadas y aburridas.

Por ejemplo, mira los primeros tres párrafos de este capítulos y dime cuál es la palabra que más se repite, y cuántas veces se ha utilizado. Aunque seas capaz de leer y comprender las palabras en pocos segundos, contarlas te resultará casi doloroso, porque la mente humana no fue diseñada para resolver ese tipo de problemas. Para una computadora es justo al revés, leer y comprender texto de un trozo de papel le sería difícil, pero contar las palabras y decirte cuántas veces se ha repetido la más utilizada le resulta muy sencillo:

```
python words.py
Enter file:words.txt
to 16
```

Nuestro "asistente de análisis de información personal" nos dirá enseguida que la palabra "que" se usó nueve veces en los primeros tres párrafos de este capítulo.

El hecho de que los computadores sean buenos en aquellas cosas en las que los humanos no lo son es el motivo por el que necesitas aprender a hablar el "idioma de las computadoras". Una vez que aprendas este nuevo lenguaje, podrás delegar tareas mundanas a tu compañero (la computadora), lo que te dejará más tiempo para ocuparte de las cosas para las que sólo tú estás capacitado. Tú pondrás la creatividad, intuición y el ingenio en esa alianza.

1.1 Creatividad y motivación

A pesar de que este libro no va dirigido a los programadores profesionales, la programación a nivel profesional puede ser un trabajo muy gratificante, tanto a nivel financiero como personal. Crear programas útiles, elegantes e inteligentes para que los usen otros, es una actividad muy creativa. Tu computadora o Asistente Digital Personal (PDA[1]), normalmente contiene muchos programas diferentes pertenecientes a distintos grupos de programadores, cada uno de ellos compitiendo por tu atención e interés. Todos ellos hacen su mejor esfuerzo por adaptarse a tus necesidades y proporcionarte una experiencia de usuario satisfactoria. En ocasiones, cuando elijes un software determinado, sus programadores son directamente recompensados gracias a tu elección.

Si pensamos en los programas como el producto de la creatividad de los programadores, tal vez la figura siguiente sea una versión más acertada de nuestra PDA:

Figure 1.2: Programadores Dirigiéndose a Ti

Por ahora, nuestra principal motivación no es conseguir dinero ni complacer a los usuarios finales, sino simplemente conseguir ser más productivos a nivel personal

[1] Personal Digital Assistant en inglés (N. del T.).

en el manejo de datos e información que encontremos en nuestras vidas. Cuando se empieza por primera vez, uno es a la vez programador y usuario final de sus propios programas. A medida que se gana habilidad como programador, y la programación se hace más creativa para uno mismo, se puede empezar a pensar en desarrollar programas para los demás.

1.2 Arquitectura hardware de las computadoras

Antes de que empecemos a aprender el lenguaje que deberemos hablar para darle instrucciones a las computadoras para desarrollar software, tendremos que aprender un poco acerca de cómo están construidos esas máquinas. Si desmontaras tu computadora o *smartphone* y mirases dentro con atención, encontrarías los siguientes componentes:

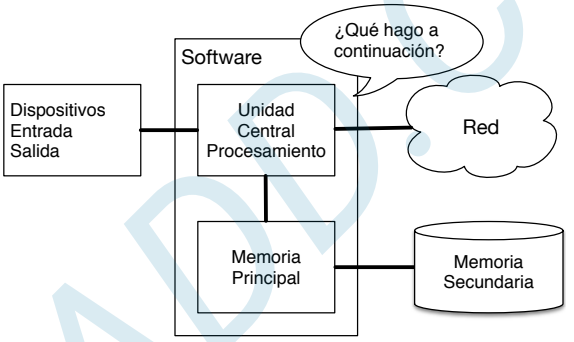

Figure 1.3: Arquitectura Hardware

Las definiciones de alto nivel de esos componentes son las siguientes:

- La *Unidad Central de Procesamiento* (o CPU[2]) es el componente de la computadora diseñado para estar obsesionado con el "¿qué hago ahora?". Si tu equipo está dentro de la clasificación de 3.0 Gigahercios, significa que la CPU preguntará "¿Qué hago ahora?" tres mil millones de veces por segundo. Vas a tener que aprender a hablar muy rápido para mantener el ritmo de la CPU.

- La *Memoria Principal* se usa para almacenar la información que la CPU necesita de forma inmediata. La memoria principal es casi tan rápida como la CPU. Pero la información almacenada en la memoria principal desaparece cuando se apaga el equipo.

- La *Memoria Secundaria* también se utiliza para almacenar información, pero es mucho más lenta que la memoria principal. La ventaja de la memoria secundaria es que puede almacenar la información incluso cuando el equipo está apagado. Algunos ejemplos de memoria secundaria serían las unidades de disco o las memorias flash (que suelen encontrarse en los *pendrives* USB y en los reproductores de música portátiles).

[2]Central Processing Unit en inglés (N. del T.).

- Los *Dispositivos de Entrada y Salida* son simplemente la pantalla, teclado, ratón, micrófono, altavoz, *touchpad*, etc. Incluyen cualquier modo de interactuar con una computadora.

- Actualmente, casi todos los equipos tienen una *Conexión de Red* para recibir información dentro de una red. Podemos pensar en una red como en un lugar donde almacenar y recuperar datos de forma muy lenta, que puede no estar siempre "activo". Así que, en cierto sentido, la red no es más que un tipo de *Memoria Secundaria* más lenta y a veces poco fiable.

Aunque la mayoría de los detalles acerca de cómo funcionan estos componentes es mejor dejársela a los constructores de equipos, resulta útil disponer de cierta terminología para poder referirnos a ellos a la hora de escribir nuestros programas.

Como programador, tu trabajo es usar y orquestar cada uno de esos recursos para resolver el problema del que tengas que ocuparte y analizar los datos de los que dispongas para encontrar la solución. Como programador estarás casi siempre "hablando" con la CPU y diciéndole qué es lo siguiente que debe hacer. A veces le tendrás que pedir a la CPU que use la memoria principal, la secundaria, la red, o los dispositivos de entrada/salida.

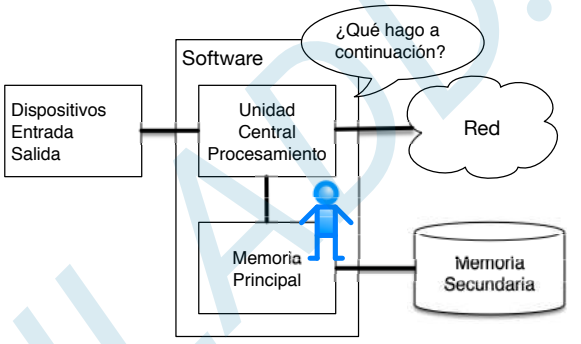

Figure 1.4: ¿Dónde estás?

Tú deberás ser la persona que responda a la pregunta "¿Qué hago ahora?" de la CPU. Pero sería muy incómodo encogerse uno mismo hasta los 5 mm. de altura e introducirse dentro de la computadora sólo para poder dar una orden tres mil millones de veces por segundo. Así que en vez de eso, tendrás que escribir las instrucciones por adelantado. Esas instrucciones almacenadas reciben el nombre de *programa* y el acto de escribirlas y encontrar cuáles son las instrucciones adecuadas, *programar*.

1.3 Comprendiendo la programación

En el resto de este libro, intentaremos convertirte en una persona experta en el arte de programar. Al terminar, te habrás convertido en un *programador* - tal vez no uno profesional, pero al menos tendrás la capacidad de encarar un problema de análisis de datos/información y desarrollar un programa para resolverlo.

En cierto modo, necesitas dos capacidades para ser programador:

- Primero, necesitas saber un lenguaje de programación (Python) - debes conocer su vocabulario y su gramática. Debes ser capaz de deletrear correctamente las palabras en ese nuevo lenguaje y saber construir "frases" bien formadas.
- Segundo, debes "contar una historia". Al escribir un relato, combinas palabras y frases para comunicar una idea al lector. Hay una cierta técnica y arte en la construcción de un relato, y la habilidad para escribir relatos mejora escribiendo y recibiendo cierta respuesta. En programación, nuestro programa es el "relato" y el problema que estás tratando de resolver es la "idea".

Una vez que aprendas un lenguaje de programación como Python, encontrarás mucho más fácil aprender un segundo lenguaje como JavaScript o C++. Cada nuevo lenguaje tiene un vocabulario y gramática muy diferentes, pero la técnica de resolución de problemas será la misma en todos ellos.

Aprenderás el "vocabulario" y "frases" de Python bastante rápido. Te llevará más tiempo el ser capaz de escribir un programa coherente para resolver un problema totalmente nuevo. Se enseña programación de forma muy similar a como se enseña a escribir. Se empieza leyendo y explicando programas, luego se escriben programas sencillos, y a continuación se van escribiendo programas progresivamente más complejos con el tiempo. En algún momento "encuentras tu musa", empiezas a descubrir los patrones por ti mismo y empiezas a ver casi de forma instintiva cómo abordar un problema y escribir un programa para resolverlo. Y una vez alcanzado ese punto, la programación se convierte en un proceso muy placentero y creativo.

Comenzaremos con el vocabulario y la estructura de los programas en Python. Ten paciencia si la simplicidad de los ejemplos te recuerda a cuando aprendiste a leer.

1.4 Palabras y frases

A diferencia de los lenguajes humanos, el vocabulario de Python es en realidad bastante reducido. Llamamos a este "vocabulario" las "palabras reservadas". Se trata de palabras que tienen un significado muy especial para Python. Cuando Python se encuentra estas palabras en un programa, sabe que sólo tienen un único significado para él. Más adelante, cuando escribas programas, podrás usar tus propias palabras con significado, que reciben el nombre de *variables*. Tendrás gran libertad a la hora de elegir los nombres para tus variables, pero no podrás utilizar ninguna de las palabras reservadas de Python como nombre de una variable.

Cuando se entrena a un perro, se utilizan palabras especiales como "siéntate", "quieto" y "tráelo". Cuando te diriges a un perro y no usas ninguna de las palabras reservadas, lo único que consigues es que se te quede mirando con cara extrañada, hasta que le dices una de las palabras que reconoce. Por ejemplo, si dices, "Me gustaría que más gente saliera a caminar para mejorar su salud general", lo que la mayoría de los perros oirían es: "bla bla bla *caminar* bla bla bla bla.". Eso se debe a que "caminar" es una palabra reservada en el lenguaje del perro. Seguramente habrá quien apunte que el lenguaje entre humanos y gatos no dispone de palabras reservadas[3].

[3] http://xkcd.com/231/

Las palabras reservadas en el lenguaje que utilizan los humanos para hablar con Python son, entre otras, las siguientes:

```
and       del       global      not       with
as        elif      if          or        yield
assert    else      import      pass
break     except    in          raise
class     finally   is          return
continue  for       lambda      try
def       from      nonlocal    while
```

Es decir, a diferencia de un perro, Python ya está completamente entrenado. Cada vez le digas "inténtalo", Python lo intentará una vez tras otra sin desfallecer[4].

Aprenderemos cuáles son las palabras reservadas y cómo utilizarlas en su momento, pero por ahora nos centraremos en el equivalente en Python de "habla" (en el lenguaje humano-perro). Lo bueno de pedirle a Python que hable es que podemos incluso indicarle lo que debe decir, pasándole un mensaje entre comillas:

```
print('¡Hola, mundo!')
```

Y ya acabamos de escribir nuestra primera oración sintácticamente correcta en Python. La frase comienza con la función *print* seguida de la cadena de texto que hayamos elegido dentro de comillas simples. Las comillas simples y dobles cumplen la misma función; la mayoría de las personas usan las comillas simples, excepto cuando la cadena de texto contiene también una comilla simple (que puede ser un apóstrofo).

1.5 Conversando con Python

Ahora que ya conocemos una palabra y sabemos cómo crear una frase sencilla en Python, necesitamos aprender a iniciar una conversación con él para comprobar nuestras nuevas capacidades con el lenguaje.

Antes de que puedas conversar con Python, deberás instalar el software necesario en tu computadora y aprender a iniciar Python en ella. En este capítulo no entraremos en detalles sobre cómo hacerlo, pero te sugiero que consultes https://es.py4e.com/ , donde encontrarás instrucciones detalladas y capturas sobre cómo configurar e iniciar Python en sistemas Macintosh y Windows. Si sigues los pasos, llegará un momento en que te encuentres ante una ventana de comandos o terminal. Si escribes entonces *python*, el intérprete de Python empezará a ejecutarse en modo interactivo, y aparecerá algo como esto:

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:54:25)
[MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

[4] En inglés "inténtalo" es "try", que es también una palabra reservada dentro del lenguaje Python (N. del T.).

El indicador >>> es el modo que tiene el intérprete de Python de preguntarte, "¿Qué quieres que haga ahora?". Python está ya preparado para mantener una conversación contigo. Todo lo que tienes que saber es cómo hablar en su idioma.

Supongamos por ejemplo que aún no conoces ni las palabras ni frases más sencillas de Python. Puede que quieras utilizar el método clásico de los astronautas cuando aterrizan en un planeta lejano e intentan hablar con los habitantes de ese mundo:

```
>>> Vengo en son de paz, por favor llévame ante tu líder
  File "<stdin>", line 1
    Vengo en son de paz, por favor llévame ante tu líder
          ^
SyntaxError: invalid syntax
>>>
```

Esto no se ve bien. A menos que pienses en algo rápidamente, los habitantes del planeta sacarán sus lanzas, te ensartarán, te asarán sobre el fuego y al final les servirás de cena.

Por suerte compraste una copia de este libro durante tus viajes, así que lo abres precisamente por esta página y pruebas de nuevo:

```
>>> print('¡Hola, mundo!')
¡Hola, mundo!
```

Esto tiene mejor aspecto, de modo que intentas comunicarte un poco más:

```
>>> print('Usted debe ser el dios legendario que viene del cielo')
Usted debe ser el dios legendario que viene del cielo
>>> print('Hemos estado esperándole durante mucho tiempo')
Hemos estado esperándole durante mucho tiempo
>>> print('La leyenda dice que debe estar usted muy rico con mostaza')
La leyenda dice que debe estar usted muy rico con mostaza
>>> print 'Tendremos un festín esta noche a menos que diga
  File "<stdin>", line 1
    print 'Tendremos un festín esta noche a menos que diga
                                                         ^
SyntaxError: Missing parentheses in call to 'print'
>>>
```

La conversación fue bien durante un rato, pero en cuanto cometiste el más mínimo fallo al utilizar el lenguaje Python, Python volvió a sacar las lanzas.

En este momento, te habrás dado cuenta que a pesar de que Python es tremendamente complejo y poderoso, y muy estricto en cuanto a la sintaxis que debes usar para comunicarte con él, Python *no* es inteligente. En realidad estás solamente manteniendo una conversación contigo mismo; eso sí, usando una sintaxis adecuada.

En cierto modo, cuando utilizas un programa escrito por otra persona, la conversación se mantiene entre tú y el programador, con Python actuando meramente de

intermediario. Python es una herramienta que permite a los creadores de programas expresar el modo en que la conversación supuestamente debe fluir. Y dentro de unos pocos capítulos más, serás uno de esos programadores que utilizan Python para hablar con los usuarios de tu programa.

Antes de que abandonemos nuestra primera conversación con el intérprete de Python, deberías aprender cual es el modo correcto de decir "adiós" al interactuar con los habitantes del Planeta Python:

```
>>> adiós
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'adiós' is not defined
>>> if you don't mind, I need to leave
  File "<stdin>", line 1
    if you don't mind, I need to leave
              ^
SyntaxError: invalid syntax
>>> quit()
```

Te habrás fijado en que el error es diferente en cada uno de los dos primeros intentos. El segundo error es diferente porque *if* es una palabra reservada, y cuando Python la ve, cree que estamos intentando decirle algo, pero encuentra la sintaxis de la frase incorrecta.

La forma correcta de decirle "adiós" a Python es introducir *quit()* en el símbolo indicador del sistema >>>. Seguramente te hubiera llevado un buen rato adivinarlo, así que tener este libro a mano probablemente te haya resultado útil.

1.6 Terminología: intérprete y compilador

Python es un lenguaje *de alto nivel*, pensado para ser relativamente sencillo de leer y escribir para las personas, y fácil de leer y procesar para las máquinas. Otros lenguajes de alto nivel son Java, C++, PHP, Ruby, Basic, Perl, JavaScript, y muchos más. El hardware real que está dentro de la Unidad Central de Procesamiento (CPU), no entiende ninguno de esos lenguajes de alto nivel.

La CPU entiende únicamente un lenguaje llamado *lenguaje de máquina* o *código máquina*. El código máquina es muy simple y francamente muy pesado de escribir, ya que está representado en su totalidad por solamente ceros y unos:

```
00101000111010010010101010000001111
11100110000011101010010101101101
...
```

El código máquina parece bastante sencillo a simple vista, dado que sólo contiene ceros y unos, pero su sintaxis es incluso más compleja y mucho más enrevesada que la de Python, razón por la cual muy pocos programadores escriben en código máquina. En vez de eso, se han creado varios programas traductores para permitir a los programadores escribir en lenguajes de alto nivel como Python o Javascript,

y son esos traductores quienes convierten los programas a código máquina, que es lo que ejecuta en realidad la CPU.

Dado que el código máquina está ligado al hardware de la máquina que lo ejecuta, ese código no es *portable* (trasladable) entre equipos de diferente tipo. Los programas escritos en lenguajes de alto nivel pueden ser trasladados entre distintas máquinas usando un intérprete diferente en cada una de ellas, o recompilando el código para crear una versión diferente del código máquina del programa para cada uno de los tipos de equipo.

Esos traductores de lenguajes de programación forman dos categorías generales: (1) intérpretes y (2) compiladores.

Un *intérprete* lee el código fuente de los programas tal y como ha sido escrito por el programador, lo analiza, e interpreta sus instrucciones sobre la marcha. Python es un intérprete y cuando lo estamos ejecutando de forma interactiva, podemos escribir una línea de Python (una frase), y este la procesa de forma inmediata, quedando listo para que podamos escribir otra línea.

Algunas de esas líneas le indican a Python que tú quieres que recuerde cierto valor para utilizarlo más tarde. Tenemos que escoger un nombre para que ese valor sea recordado y usaremos ese nombre simbólico para recuperar el valor más tarde. Utilizamos el término *variable* para denominar las etiquetas que usamos para referirnos a esos datos almacenados.

```
>>> x = 6
>>> print(x)
6
>>> y = x * 7
>>> print(y)
42
>>>
```

En este ejemplo, le pedimos a Python que recuerde el valor seis y use la etiqueta x para que podamos recuperar el valor más tarde. Comprobamos que Python ha guardado de verdad el valor usando *print*. Luego le pedimos a Python que recupere x, lo multiplique por siete y guarde el valor calculado en y. Finalmente, le pedimos a Python que escriba el valor actual de y.

A pesar de que estamos escribiendo estos comandos en Python línea a línea, Python los está tratando como una secuencia ordenada de sentencias, en la cual las últimas frases son capaces de obtener datos creados en las anteriores. Estamos, por tanto, escribiendo nuestro primer párrafo sencillo con cuatro frases en un orden lógico y útil.

La esencia de un *intérprete* consiste en ser capaz de mantener una conversación interactiva como la mostrada más arriba. Un *compilador* necesita que le entreguen el programa completo en un fichero, y luego ejecuta un proceso para traducir el código fuente de alto nivel a código máquina, tras lo cual coloca ese código máquina resultante dentro de otro fichero para su ejecución posterior.

En sistemas Windows, a menudo esos ejecutables en código máquina tienen un sufijo o extensión como ".exe" o ".dll", que significan "ejecutable" y "librería de

enlace dinámico" (*dynamic link library*) respectivamente. En Linux y Macintosh, no existe un sufijo que identifique de manera exclusiva a un fichero como ejecutable.

Si abrieras un fichero ejecutable con un editor de texto, verías algo complemente disparatado e ilegible:

```
^?ELF^A^A^A^@^@^@^@^@^@^@^@^B^@^C^@^A^@^@^@\xa0\x82
^D^H4^@^@^@\x90^]^@^@^@^@^@^@4^@ ^@^G^@(^@$^@!^@^F^@
^@^@4^@^@^@4\x80^D^H4\x80^D^H\xe0^@^@^@\xe0^@^@^@^E
^@^@^@^D^@^@^@^C^@^@^@^T^A^@^@^T\x81^D^H^T\x81^D^H^S
^@^@^@^S^@^@^@^D^@^@^@^A^@^@^@^A\^D^HQVhT\x83^D^H\xe8
....
```

No resulta fácil leer o escribir código máquina, pero afortunadamente disponemos de *intérpretes* y *compiladores* que nos permiten escribir en lenguajes de alto nivel, como Python o C.

Llegados a este punto en la explicación acerca de los compiladores e intérpretes, seguramente te estarás preguntando algunas cosas acerca del propio intérprete de Python. ¿En qué lenguaje está escrito? ¿Está escrito en un lenguaje compilado? Cuando escribimos "python", ¿qué ocurre exactamente?

El intérprete de Python está escrito en un lenguaje de alto nivel llamado "C". En realidad, puedes ver el código fuente del intérprete de Python acudiendo a www.python.org e incluso modificarlo a tu gusto. Quedamos, pues, en que Python es en sí mismo un programa y que está compilado en código máquina. Cuando instalaste Python en tu computadora (o el vendedor lo instaló), colocaste una copia del código máquina del programa Python traducido para tu sistema. En Windows, el código máquina ejecutable para el intérprete de Python es probablemente un fichero con un nombre como:

```
C:\Python35\python.exe
```

En realidad, eso ya es más de lo que necesitas saber para ser un programador en Python, pero algunas veces vale la pena contestar estas pequeñas dudas recurrentes justo al principio.

1.7 Escribiendo un programa

Escribir comandos en el intérprete de Python es un buen modo de experimentar con las capacidades de Python, pero no es lo más recomendado a la hora de resolver problemas más complejos.

Cuando queremos escribir un programa, usamos un editor de texto para escribir las instrucciones de Python en un fichero, que recibe el nombre de *script*. Por convención, los scripts de Python tienen nombres que terminan en `.py`.

Para ejecutar el script, hay que indicarle al intérprete de Python el nombre del fichero. En una ventana de comandos de Unix o Windows, escribirías `python hola.py`, de este modo:

```
csev$ cat hola.py
print('¡Hola, mundo!')
csev$ python hola.py
¡Hola, mundo!
csev$
```

"csev$" es el indicador (*prompt*) del sistema operativo, y el comando "cat hola.py" nos muestra que el archivo "hola.py" contiene un programa con una única línea de código que imprime en pantalla una cadena de texto.

Llamamos al intérprete de Python y le pedimos que lea el código fuente desde el archivo "hola.py", en vez de esperar a que vayamos introduciendo líneas de código Python de forma interactiva.

Habrás notado que cuando trabajamos con un fichero no necesitamos incluir el comando *quit()* al final del programa Python. Cuando Python va leyendo tu código fuente desde un archivo, sabe que debe parar cuando llega al final del fichero.

1.8 ¿Qué es un programa?

Podemos definir un *programa*, en su forma más básica, como una secuencia de declaraciones o sentencias que han sido diseñadas para hacer algo. Incluso nuestro sencillo script "hola.py" es un programa. Es un programa de una sola línea y no resulta particularmente útil, pero si nos ajustamos estrictamente a la definición, se trata de un programa en Python.

Tal vez resulte más fácil comprender qué es un programa pensando en un problema que pudiera ser resuelto a través de un programa, y luego estudiando cómo sería el programa que solucionaría ese problema.

Supongamos que estás haciendo una investigación de computación o informática social en mensajes de Facebook, y te interesa conocer cual es la palabra más utilizada en un conjunto de mensajes. Podrías imprimir el flujo de mensajes de Facebook y revisar con atención el texto, buscando la palabra más común, pero sería un proceso largo y muy propenso a errores. Sería más inteligente escribir un programa en Python para encargarse de la tarea con rapidez y precisión, y así poder emplear el fin de semana en hacer otras cosas más divertidas.

Por ejemplo, fíjate en el siguiente texto, que trata de un payaso y un coche. Estúdialo y trata de averiguar cual es la palabra más común y cuántas veces se repite.

```
el payaso corrió tras el coche y el coche se metió dentro de la tienda
y la tienda cayó sobre el payaso y el coche
```

Ahora imagina que haces lo mismo pero buscando a través de millones de líneas de texto. Francamente, tardarías menos aprendiendo Python y escribiendo un programa en ese lenguaje para contar las palabras que si tuvieras que ir revisando todas ellas una a una.

Pero hay una noticia aún mejor, y es que se me ha ocurrido un programa sencillo para encontrar cuál es la palabra más común dentro de un fichero de texto. Ya lo escribí, lo probé, y ahora te lo regalo para que lo puedas utilizar y ahorrarte mucho tiempo.

```
name = input('Enter file:')
handle = open(name, 'r')
counts = dict()

for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in list(counts.items()):
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)

# Código: https://es.py4e.com/code3/words.py
```

No necesitas ni siquiera saber Python para usar este programa. Tendrás que llegar hasta el capítulo 10 de este libro para entender por completo las impresionantes técnicas de Python que se han utilizado para crearlo. Ahora eres el usuario final, sólo tienes que usar el programa y sorprenderte de sus habilidades y de cómo te permite ahorrar un montón de esfuerzo. Tan sólo tienes que escribir el código dentro de un fichero llamado *words.py* y ejecutarlo, o puedes descargar el código fuente directamente desde https://es.py4e.com/code3/ y ejecutarlo.

Este es un buen ejemplo de cómo Python y el lenguaje Python actúan como un intermediario entre tú (el usuario final) y yo (el programador). Python es un medio para que intercambiemos secuencias de instrucciones útiles (es decir, programas) en un lenguaje común que puede ser usado por cualquiera que instale Python en su computadora. Así que ninguno de nosotros está hablando *con Python*, sino que estamos comunicándonos uno con el otro *a través de* Python.

1.9 Los bloques de construcción de los programas

En los próximos capítulos, aprenderemos más sobre el vocabulario, la estructura de las frases y de los párrafos y la estructura de los relatos en Python. Aprenderemos cuáles son las poderosas capacidades de Python y cómo combinar esas capacidades entre sí para crear programas útiles.

Hay ciertos patrones conceptuales de bajo nivel que se usan para estructurar los programas. Esas estructuras no son exclusivas de Python, sino que forman parte de cualquier lenguaje de programación, desde el código máquina hasta los lenguajes de alto nivel.

entrada Obtener datos del "mundo exterior". Puede consistir en leer datos desde un fichero, o incluso desde algún tipo de sensor, como un micrófono o un GPS.

En nuestros primeros programas, las entradas van a provenir del usuario, que introducirá los datos a través del teclado.

salida Mostrar los resultados del programa en una pantalla, almacenarlos en un fichero o incluso es posible enviarlos a un dispositivo como un altavoz para reproducir música o leer un texto.

ejecución secuencial Ejecutar una sentencia tras otra en el mismo orden en que se van encontrando en el *script*.

ejecución condicional Comprobar ciertas condiciones y luego ejecutar u omitir una secuencia de sentencias.

ejecución repetida Ejecutar un conjunto de sentencias varias veces, normalmente con algún tipo de variación.

reutilización Escribir un conjunto de instrucciones una vez, darles un nombre y así poder reutilizarlas luego cuando se necesiten en cualquier punto de tu programa.

Parece demasiado simple para ser cierto, y por supuesto nunca es tan sencillo. Es como si dijéramos que andar es simplemente "poner un pie delante del otro". El "arte" de escribir un programa es componer y entrelazar juntos esos elementos básicos muchas veces hasta conseguir al final algo que resulte útil para sus usuarios.

El programa para contar palabras que vimos antes utiliza al mismo tiempo todos esos patrones excepto uno.

1.10 ¿Qué es posible que vaya mal?

Como vimos en nuestra anterior conversación con Python, debemos comunicarnos con mucha precisión cuando escribimos código Python. El menor error provocará que Python se niegue a hacer funcionar tu programa.

Los programadores novatos a menudo se toman el hecho de que Python no permita cometer errores como la prueba definitiva de que es perverso, odioso y cruel. A pesar de que a Python parece gustarle todos los demás, es capaz de identificar a los novatos en concreto, y les guarda un gran rencor. Debido a ello, toma sus programas perfectamente escritos, y los rechaza, considerándolos como "inservibles", sólo para atormentarlos.

```
>>> primt '¡Hola, mundo!'
  File "<stdin>", line 1
    primt '¡Hola, mundo!'
                         ^
SyntaxError: invalid syntax
>>> primt ('Hola, mundo')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'primt' is not defined

>>> ¡Te odio, Python!
  File "<stdin>", line 1
    ¡Te odio, Python!
     ^
```

```
SyntaxError: invalid syntax
>>> si sales fuera, te daré una lección
  File "<stdin>", line 1
    si sales fuera, te daré una lección
               ^
SyntaxError: invalid syntax
>>>
```

No es mucho lo que se gana discutiendo con Python. Solo es una herramienta. No tiene emociones, es feliz y está preparado para servirte en el momento que lo necesites. Sus mensajes de error parecen crueles, pero simplemente se trata de una petición de ayuda del propio Python. Ha examinado lo que has tecleado, y sencillamente no es capaz de entender lo que has escrito.

Python se parece mucho a un perro, te quiere incondicionalmente, entiende algunas pocas palabras clave, te mira con una mirada dulce en su cara(>>>), y espera que le digas algo que él pueda comprender. Cuando Python dice "SyntaxError: invalid syntax" (Error de sintaxis: sintaxis inválida), tan solo está agitando su cola y diciendo: "Creo que has dicho algo, pero no te entiendo; de todos modos, por favor, sigue hablando conmigo (>>>)."

A medida que tus programas vayan aumentando su complejidad, te encontrarás con tres tipos de errores generales:

Errores de sintaxis (Syntax errors) Estos son los primeros errores que cometerás y también los más fáciles de solucionar. Un error de sintaxis significa que has violado las reglas "gramaticales" de Python. Python hace todo lo que puede para señalar el punto exacto, la línea y el carácter donde ha detectado el fallo. Lo único complicado de los errores de sintaxis es que a veces el error que debe corregirse está en realidad en una línea anterior a la cual Python *detectó* ese fallo. De modo que la línea y el carácter que Python indica en un error de sintaxis pueden ser tan sólo un punto de partida para tu investigación.

Errores lógicos Se produce un error lógico cuando un programa tiene una sintaxis correcta, pero existe un error en el orden de las sentencias o en la forma en que están relacionadas unas con otras. Un buen ejemplo de un error lógico sería: "toma un trago de tu botella de agua, ponla en tu mochila, camina hasta la biblioteca y luego vuelve a enroscar la tapa en la botella."

Errores semánticos Un error semántico ocurre cuando la descripción que has brindado de los pasos a seguir es sintácticamente perfecta y está en el orden correcto, pero sencillamente hay un error en el programa. El programa es correcto, pero no hace lo que tú *pretendías* que hiciera. Un ejemplo podría ser cuando le das indicaciones a alguien sobre cómo llegar a un restaurante, y le dices "...cuando llegues a la intersección con la gasolinera, gira a la izquierda, continúa durante otro kilómetro y el restaurante es el edificio rojo que encontrarás a tu izquierda.". Tu amigo se retrasa y te llama para decirte que está en una granja dando vueltas alrededor de un granero, sin rastro alguno de un restaurante. Entonces le preguntas "¿giraste a la izquierda o la derecha?", y te responde "Seguí tus indicaciones al pie de la letra, dijiste que girara a la izquierda y continuar un kilómetro desde la gasolinera.", entonces le respondes "Lo siento mucho, porque a pesar de que mis indica-

ciones fueron sintácticamente correctas, tristemente contenían un pequeño pero indetectado error semántico.".

Insisto en que, ante cualquiera de estos tres tipos de errores, Python únicamente hace lo que está a su alcance por seguir al pie de la letra lo que tú le has pedido que haga.

1.11 Depurando los programas

Cuando Python muestra un error, u obtiene un resultado diferente al que esperabas, empieza una intensa búsqueda de la causa del error. Depurar es el proceso de encontrar la causa o el origen de ese error en tu código. Cuando depuras un programa, y especialmente cuando tratas con un *bug* algo difícil de solucionar, existen cuatro cosas por hacer:

leer Revisar tu código, leerlo de nuevo, y asegurarte de que ahí está expresado de forma correcta lo que quieres decir.

ejecutar Prueba haciendo cambios y ejecutando diferentes versiones. Con frecuencia, si muestras en tu programa lo correcto en el lugar indicado, el problema se vuelve obvio, pero en ocasiones debes invertir algo de tiempo hasta conseguirlo.

pensar detenidamente ¡Toma tu tiempo para pensar!, ¿A qué tipo de error corresponde: sintaxis, en tiempo de ejecución, semántico?, ¿Qué información puedes obtener de los mensajes de error, o de la salida del programa?, ¿Qué tipo de errores podría generar el problema que estás abordando?, ¿Cuál fue el último cambio que hiciste, antes de que se presentara el problema?

retroceder En algún momento, lo mejor que podrás hacer es dar marcha atrás, deshacer los cambios recientes hasta obtener de nuevo un programa que funcione y puedas entender. Llegado a ese punto, podrás continuar con tu trabajo.

Algunas veces, los programadores novatos se quedan estancados en una de estas actividades y olvidan las otras. Encontrar un *bug* requiere leer, ejecutar, pensar detenidamente y algunas veces retroceder. Si te bloqueas en alguna de estas actividades, prueba las otras. Cada actividad tiene su procedimiento de análisis.

Por ejemplo, leer tu código podría ayudar si el problema es un error tipográfico, pero no si es uno conceptual. Si no comprendes lo que hace tu programa, puedes leerlo 100 veces y no encontrarás el error, puesto que dicho error está en tu mente.

Experimentar puede ayudar, especialmente si ejecutas pequeñas pruebas. Pero si experimentas sin pensar o leer tu código, podrías caer en un patrón que llamo "programación de paseo aleatorio", que es el proceso de realizar cambios al azar hasta que el programa logre hacer lo que debería. No hace falta mencionar que este tipo de programación puede tomar mucho tiempo.

Debes tomar el tiempo suficiente para pensar. Depurar es como una ciencia experimental. Debes plantear al menos una hipótesis sobre qué podría ser el problema. Si hay dos o más posibilidades, piensa en alguna prueba que pueda ayudarte a descartar una de ellas.

Descansar y conversar ayuda a estimular el pensamiento. Si le explicas el problema a alguien más (o incluso a tí mismo), a veces encontrarás la respuesta antes de terminar la pregunta.

Pero incluso las mejores técnicas de depuración fallarán si hay demasiados errores, o si el código que intentas mejorar es demasiado extenso y complicado. Algunas veces la mejor opción es retroceder, y simplificar el programa hasta que obtengas algo que funcione y puedas entender.

Por lo general, los programadores novatos son reacios a retroceder porque no soportan tener que borrar una línea de código (incluso si está mal). Si te hace sentir mejor, prueba a copiar tu programa en otro archivo antes de empezar a modificarlo. De esa manera, puedes recuperar poco a poco pequeñas piezas de código que necesites.

1.12 El camino del aprendizaje

Según vayas avanzando por el resto del libro, no te asustes si los conceptos no parecen encajar bien unos con otros al principio. Cuando estabas aprendiendo a hablar, no supuso un problema que durante los primeros años solo pudieras emitir lindos balbuceos. Y también fue normal que te llevara seis meses pasar de un vocabulario simple a frases simples, y que te llevara 5-6 años más pasar de frases a párrafos, y que todavía tuvieran que transcurrir unos cuantos años más hasta que fuiste capaz de escribir tu propia historia corta interesante.

Pretendemos que aprendas Python rápidamente, por lo que te enseñaremos todo al mismo tiempo durante los próximos capítulos. Aún así, ten en cuenta que el proceso es similar a aprender un idioma nuevo, que lleva un tiempo absorber y comprender antes de que te resulte familiar. Eso produce cierta confusión, puesto que revisaremos en distintas ocasiones determinados temas, y trataremos que de esa manera puedas visualizar los pequeños fragmentos que componen esta obra completa. A pesar de que el libro está escrito de forma lineal, no dudes en ser no lineal en la forma en que abordes las materias. Avanza y retrocede, y lee a veces por encima. Al ojear material más avanzado sin comprender del todo los detalles, tendrás una mejor comprensión del "¿por qué?" de la programación. Al revisar el material anterior e incluso al realizar nuevamente los ejercicios previos, te darás cuenta que ya has aprendido un montón de cosas, incluso si el tema que estás examinando en ese momento parece un poco difícil de abordar.

Normalmente, cuando uno aprende su primer lenguaje de programación, hay unos pocos momentos "¡A-já!" estupendos, en los cuales puedes levantar la vista de la roca que estás machacando con martillo y cincel, separarte unos pasos y comprobar que lo que estás intentando construir es una maravillosa escultura.

Si algo parece particularmente difícil, generalmente no vale la pena quedarse mirándolo toda la noche. Respira, toma una siesta, come algo, explícale a alguien (quizás a tu perro) con qué estás teniendo problemas, y después vuelve a observarlo con un perspectiva diferente. Te aseguro que una vez que aprendas los conceptos de la programación en el libro, volverás atrás y verás que en realidad todo era fácil, elegante y que simplemente te ha llevado un poco de tiempo llegar a absorberlo.

1.13 Glosario

bug Un error en un programa.

código fuente Un programa en un lenguaje de alto nivel.

código máquina El lenguaje de más bajo nivel para el software, es decir, el lenguaje que es directamente ejecutado por la unidad central de procesamiento (CPU).

compilar Traducir un programa completo escrito en un lenguaje de alto nivel a un lenguaje de bajo nivel, para dejarlo listo para una ejecución posterior.

error semántico Un error dentro de un programa que provoca que haga algo diferente de lo que pretendía el programador.

función print Una instrucción que hace que el intérprete de Python muestre un valor en la pantalla.

indicador de línea de comandos (*prompt*) Cuando un programa muestra un mensaje y se detiene para que el usuario teclee algún tipo de dato.

interpretar Ejecutar un programa escrito en un lenguaje de alto nivel traduciéndolo línea a línea.

lenguaje de alto nivel Un lenguaje de programación como Python, que ha sido diseñado para que sea fácil de leer y escribir por las personas.

lenguaje de bajo nivel Un lenguaje de programación que está diseñado para ser fácil de ejecutar para una computadora; también recibe el nombre de "código máquina", "lenguaje máquina" o "lenguaje ensamblador".

memoria principal Almacena los programas y datos. La memoria principal pierde su información cuando se desconecta la alimentación.

memoria secundaria Almacena los programas y datos y mantiene su información incluso cuando se interrumpe la alimentación. Es generalmente más lenta que la memoria principal. Algunos ejemplos de memoria secundaria son las unidades de disco y memorias flash que se encuentran dentro de los dispositivos USB.

modo interactivo Un modo de usar el intérprete de Python, escribiendo comandos y expresiones directamente en el indicador de la línea de comandos.

parsear Examinar un programa y analizar la estructura sintáctica.

portabilidad Es la propiedad que poseen los programas que pueden funcionar en más de un tipo de computadora.

programa Un conjunto de instrucciones que indican cómo realizar algún tipo de cálculo.

resolución de un problema El proceso de formular un problema, encontrar una solución, y mostrar esa solución.

semántica El significado de un programa.

unidad central de procesamiento El corazón de cualquier computadora. Es lo que ejecuta el software que escribimos. También recibe el nombre de "CPU" por sus siglas en inglés (*Central Processsing Unit*), o simplemente, "el procesador".

1.14 Ejercicios

Ejercicio 1: ¿Cuál es la función de la memoria secundaria en una computadora?

a) Ejecutar todos los cálculos y la lógica del programa

b) Descargar páginas web de Internet
c) Almacenar información durante mucho tiempo, incluso después de ciclos de apagado y encendido
d) Recolectar la entrada del usuario

Ejercicio 2: ¿Qué es un programa?

Ejercicio 3: ¿Cuál es la diferencia entre un compilador y un intérprete?

Ejercicio 4: ¿Cuál de los siguientes contiene "código máquina"?

a) El intérprete de Python
b) El teclado
c) El código fuente de Python
d) Un documento de un procesador de texto

Ejercicio 5: ¿Qué está mal en el siguiente código?:

```
>>> primt '¡Hola, mundo!'
  File "<stdin>", line 1
    primt '¡Hola, mundo!'
                ^
SyntaxError: invalid syntax
>>>
```

Ejercicio 6: ¿En qué lugar del computador queda almacenada una variable, como en este caso "X", después de ejecutar la siguiente línea de Python?:

```
x = 123
```

a) Unidad central de procesamiento
b) Memoria Principal
c) Memoria Secundaria
d) Dispositivos de Entrada
e) Dispositivos de Salida

Ejercicio 7: ¿Qué mostrará en pantalla el siguiente programa?:

```
x = 43
x = x + 1
print(x)
```

a) 43
b) 44
c) x + 1
d) Error, porque x = x + 1 no es posible matemáticamente.

Ejercicio 8: Explica cada uno de los siguientes conceptos usando un ejemplo de una capacidad humana: (1) Unidad central de procesamiento, (2) Memoria principal, (3) Memoria secundaria, (4) Dispositivos de entrada, y (5) Dispositivos de salida. Por ejemplo, "¿Cuál sería el equivalente humano de la Unidad central de procesamiento?".

Ejercicio 9: ¿Cómo puedes corregir un "Error de sintaxis"?.