1 Comenzando el viaje

Instalando Python

Para instalar Python en nuestro ordenador, sólo tendremos que ir a la web https://www.python.org/ y descargar el instalador.

Los pasos varían algo entre sistemas operativos.

¡Atención! En Windows, no olvides añadir Python al PATH en los primeros pasos de instalación:



Imagen - Instalando Python

Una vez hecho eso ya tendremos accesible Python desde la terminal.

Copyright 2026

https://www.youtube.com/embed/9fNKy9zOPkg

¿Por qué debería aprender a escribir programas?

Escribir programas (o programar) es una actividad muy creativa y gratificante. Puedes escribir programas por muchas razones, desde ganarte la vida, resolver un problema difícil de análisis de datos, hasta divertirte o ayudar a alguien más a resolver un problema. Este libro asume que **todos** debemos saber cómo programar, y que una vez que sepa cómo programar, descubrirás qué quieres hacer con tus nuevas habilidades.

Estamos rodeados en nuestra vida diaria de ordenadores que van desde portátiles hasta teléfonos móviles. Podemos pensar en estos ordenadores como nuestros "asistentes personales" que pueden hacerse cargo de muchas cosas en nuestro nombre. El hardware en nuestros ordenadores de hoy en día está esencialmente construido para hacernos continuamente la pregunta "¿Qué quieres que haga ahora?"

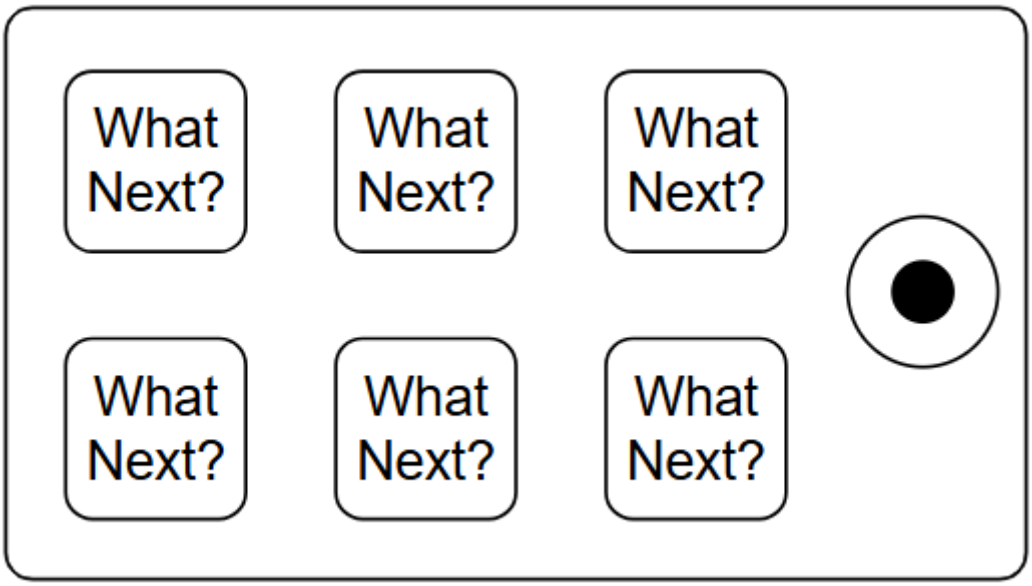


Imagen - Asistente personal digital

Los programadores agregan un sistema operativo y un conjunto de aplicaciones al hardware y terminamos con un Asistente digital personal que es bastante útil y capaz de ayudarnos a hacer muchas cosas diferentes.

Nuestros ordenadores son rápidos y tienen una gran cantidad de memoria y podrían ser muy útiles para nosotros si supiéramos el idioma con el que decirle al ordenador lo que nos gustaría "hacer a continuación". Si conociéramos este idioma, podríamos decirle al ordenador que realice tareas en nuestro nombre que son repetitivas. Curiosamente, las cosas que los ordenadores pueden hacer mejor son a menudo el tipo de cosas que a los humanos nos parecen aburridas.

Por ejemplo, mira los primeros tres párrafos de este capítulo, dime la palabra que se usa con más frecuencia y cuántas veces se usa dicha palabra. Si bien pudiste leer y entender las palabras en unos pocos segundos, contarlas es casi doloroso porque no es el tipo de problema que las mentes humanas están diseñadas para resolver. Para una ordenador es todo lo contrario, leer y entender texto de una hoja de papel es difícil, pero contar las palabras y decirle cuántas veces se usó la palabra más usada es muy fácil:

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

Nuestro "asistente de análisis de información personal" rápidamente nos dijo que la palabra "to" se usó dieciséis veces en los primeros tres párrafos de este capítulo (válido para la versión inglesa de este libro).

Para usar ese poder de la máquina, necesitamos adquirir la habilidad para hablar el "lenguaje del ordenador". Una vez que aprendas este nuevo idioma, puedes delegar tareas mundanas a su compañero (el ordenador), lo que te deja más tiempo para hacer las cosas para las que estás preparado de manera única. Traes creatividad, intuición e inventiva a esta asociación.

Creatividad y motivación

Si bien este libro no está dirigido a programadores profesionales, la programación profesional puede ser un trabajo muy gratificante tanto a nivel financiero como personal. Crear programas útiles, elegantes e inteligentes para que otros puedan usar es una actividad muy creativa. Su ordenador usualmente contiene muchos programas diferentes, de diferentes grupos de programadores, cada uno compitiendo por su atención e interés. Hacen todo lo posible para satisfacer sus necesidades y le brindan una excelente experiencia de usuario en el proceso. En algunas situaciones, cuando elige una pieza de software, los programadores reciben una

Copyright 2026

compensación directa debido a su elección.

Si pensamos en los programas como la producción creativa de grupos de programadores, tal vez la siguiente figura sea una versión posible de nuestra PDA:

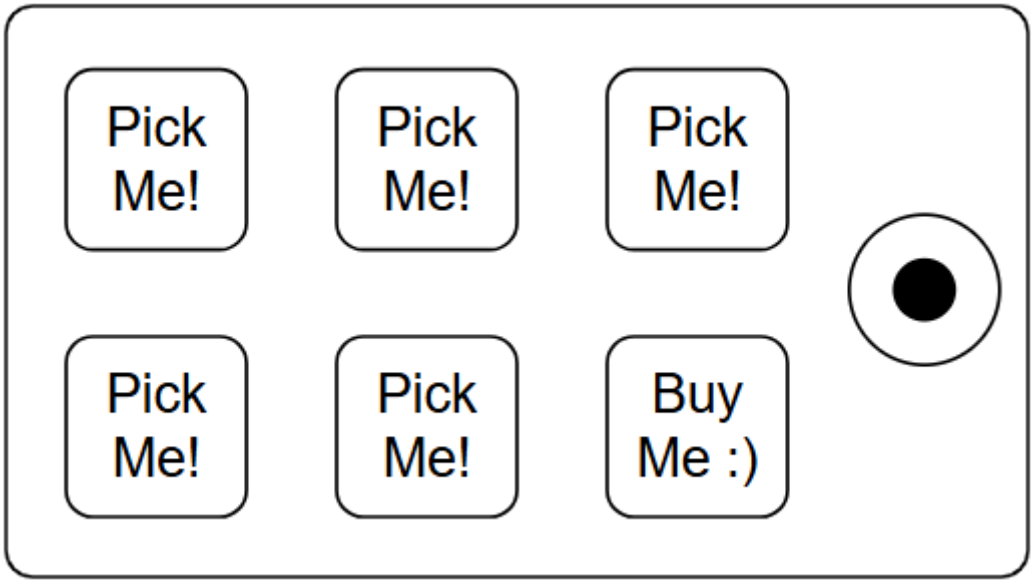
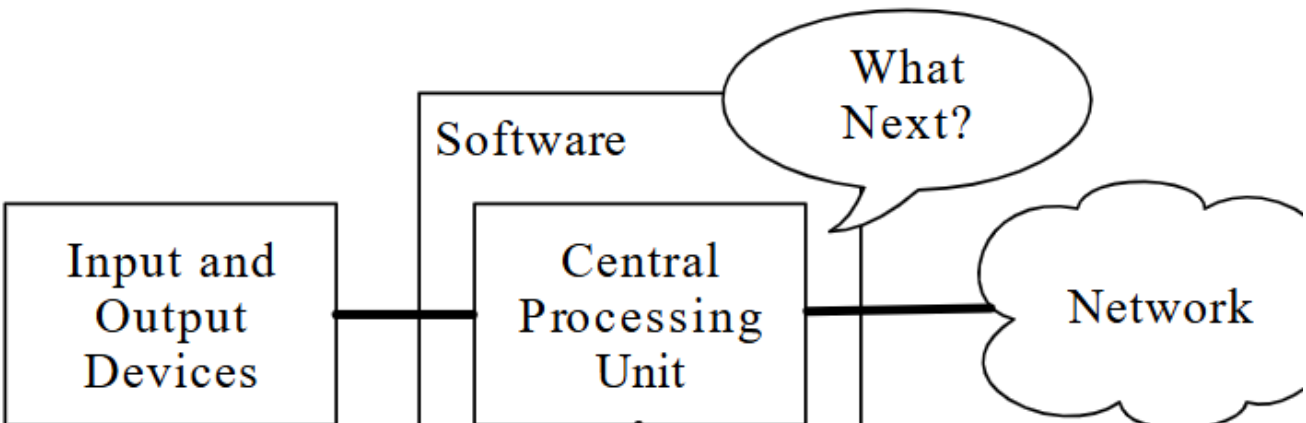


Imagen - Programadores hablando contigo

Por ahora, nuestra principal motivación no es ganar dinero o complacer a los usuarios finales, sino que seamos más productivos en el manejo de los datos y la información que encontraremos en nuestras vidas. Cuando comiences, serás el programador y el usuario final de tus programas. A medida que adquieras destreza como programador, tus pensamientos pueden dirigirse hacia el desarrollo de programas para otros.

Arquitectura de hardware del ordenador

Antes de comenzar a aprender el idioma que hablamos para crear software, necesitamos aprender un poco sobre cómo se construyen los ordenadores. Si tuviera que desarmar su ordenador o teléfono celular y mirar adentro, encontraría las siguientes partes:



Copyr

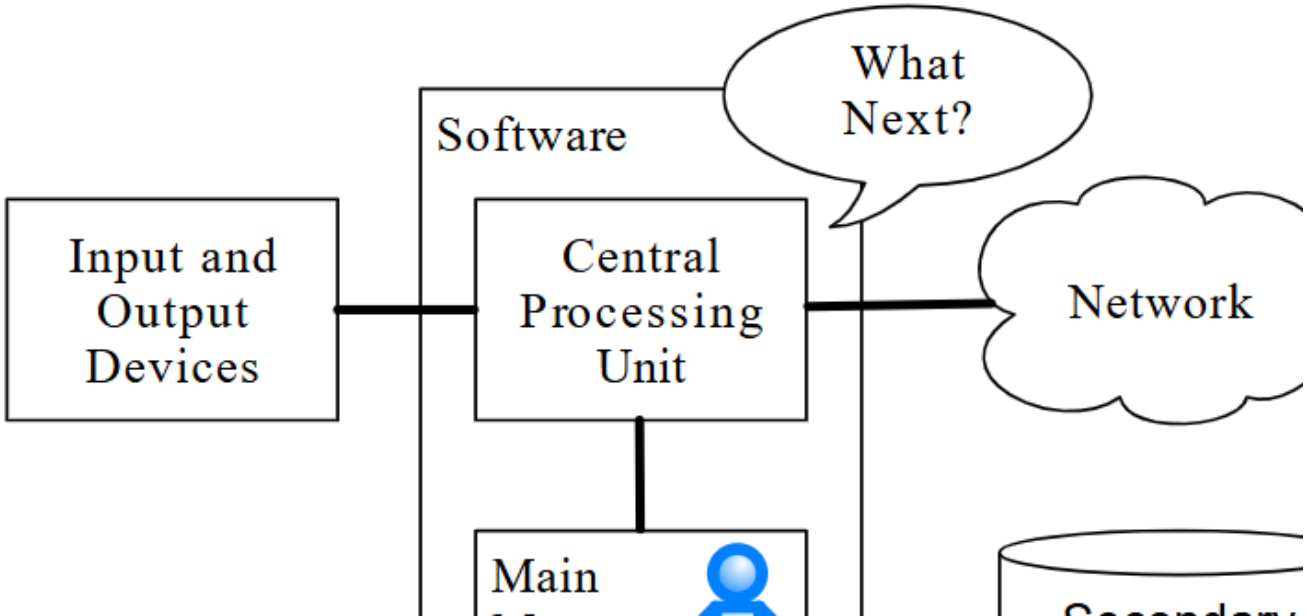
Imagen - Arquitectura de hardware

Las definiciones de alto nivel de estas partes son las siguientes:

- La **Unidad central de procesamiento** (o CPU) es la parte de el ordenador que está diseñada para estar obsesionada con "¿qué sigue?" Si tu ordenador tiene una clasificación de 3.0 Gigahercios, significa que la CPU preguntará "¿Qué sigue?" Tres mil millones de veces por segundo. Tendrás que aprender a hablar rápido para mantenerse al día con la CPU.
- La **memoria principal** se utiliza para almacenar información que la CPU necesita a toda prisa. La memoria principal es casi tan rápida como la CPU. Pero la información almacenada en la memoria principal se desvanece cuando se apaga el ordenador.
- La **memoria secundaria** también se usa para almacenar información, pero es mucho más lenta que la memoria principal. La ventaja de la memoria secundaria es que puede almacenar información incluso cuando no hay energía en el ordenador. Algunos ejemplos de memoria secundaria son las unidades de disco o la memoria flash (que generalmente se encuentran en memorias USB)
- Los **Dispositivos de entrada y salida** son simplemente nuestra pantalla, teclado, mouse, micrófono, altavoz, panel táctil, etc. Son todas las formas en que interactuamos con el ordenador.
- **Conexión de red** para recuperar información a través de una red. Podemos pensar en la red como un lugar muy lento para almacenar y recuperar datos que pueden no estar siempre "arriba". Entonces, en cierto sentido, la red es una forma más lenta y en ocasiones poco confiable de **Memoria secundaria**.

Si bien la mayoría de los detalles de cómo funcionan estos componentes es mejor dejarlos en manos de los desarrolladores, es útil tener un poco de terminología para que podamos hablar sobre estas partes diferentes a medida que escribimos nuestros programas.

Como programador, tu trabajo consiste en utilizar y organizar cada uno de estos recursos para resolver el problema que sea y analizar los datos que obtienes de la solución. En general, estarás "hablando" con la CPU y le dirás qué hacer a continuación. A veces le dirás a la CPU que use la memoria principal, la memoria secundaria, la red o los dispositivos de entrada/salida.



Copyr

Imagen - ¿Dónde estás?

Necesitas ser la persona que responde a la CPU "¿Qué sigue?". Pero sería muy incómodo reducirte a una altura de 5 mm e insertarte en el ordenador solo para que puedas emitir un comando tres mil millones de veces por segundo. Así que, en lugar de eso, debes escribir tus instrucciones con anticipación. Llamamos a estas instrucciones almacenadas un **programa**, y al hecho de escribir estas instrucciones para que sean correctas, **programación**.

Entendiendo la programación

En el resto de este libro, trataremos de convertirte en una persona experta en el arte de la programación. Al final, serás un **programador** - quizás no un programador profesional, pero al menos tendrás las habilidades para analizar un problema de análisis de datos/información y desarrollar un programa para resolver el problema.

En cierto sentido, necesitas dos habilidades para ser un programador:

- Primero, necesitas saber el lenguaje de programación (Python) - necesitas conocer el vocabulario y la gramática. Debes poder escribir correctamente las palabras en este nuevo idioma y saber cómo construir "oraciones" bien formadas en este nuevo idioma.
- Segundo, necesitas "contar una historia". Al escribir una historia, combinas palabras y oraciones para transmitir una idea al lector. Hay una habilidad y arte en la construcción de la historia, y la habilidad en la escritura de la historia se mejora al escribir un poco y obtener algunos comentarios. En programación, nuestro programa es la "historia" y el problema que intenta resolver es la "idea".

Una vez que aprendas un lenguaje de programación como Python, te resultará mucho más fácil aprender un segundo lenguaje de programación como JavaScript o C++. El nuevo lenguaje de programación tiene un vocabulario y gramática muy diferentes, pero las habilidades de resolución de problemas serán las mismas en todos los lenguajes de programación.

Aprenderás el "vocabulario" y las "oraciones" de Python con bastante rapidez. Te llevará más tiempo poder escribir un programa coherente para resolver un problema completamente nuevo. Enseñamos la programación como enseñamos a escribir. Comenzamos a leer y explicar programas, luego escribimos programas simples y luego escribimos programas cada vez más complejos a lo largo del tiempo. En algún momento, "obtienes tu musa" y ves los patrones por tu cuenta y puedes ver más naturalmente cómo tomar un problema y escribir un programa que resuelva ese problema. Y una vez que llegas a ese punto, la programación se convierte en un proceso muy agradable y creativo.

Copyright 2026

Comenzamos con el vocabulario y la estructura de los programas de Python.

Palabras y oraciones

A diferencia de las lenguas humanas, el vocabulario de Python es bastante pequeño. Llamamos a este "vocabulario" las "palabras reservadas". Estas son palabras que tienen un significado muy especial para Python. Cuando Python ve estas palabras en un programa de Python, tienen un solo significado para Python. Luego, a medida que escribes programas, inventarás sus propias palabras que tienen un significado para ti llamadas **variables**. Tendrás gran libertad para elegir tus nombres para sus variables, pero no puedes usar ninguna de las palabras reservadas de Python como nombre para una variable.

Cuando entrenamos a un perro, usamos palabras especiales como "sentarse", "quedarse" y "buscar". Cuando le hablas a un perro y no usas ninguna de las palabras reservadas, solo te miran con una expresión burlona hasta que dices una palabra reservada. Por ejemplo, si dices: "Me gustaría que más gente caminara para mejorar su salud general", lo que la mayoría de los perros escuchan es "bla bla bla **caminar** bla bla bla bla". Esto se debe a que "caminar" es una palabra reservada en lenguaje canino. Muchos podrían sugerir que el lenguaje entre humanos y gatos no tiene palabras reservadas[1].

Las palabras reservadas en el idioma donde los humanos hablan con Python incluyen lo siguiente:

```
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
```

Eso es todo, y a diferencia de un perro, Python ya está completamente entrenado. Cuando dices "try", Python lo intentará cada vez que lo digas.

Aprenderemos estas palabras reservadas y cómo se usan a tiempo, pero por ahora nos enfocaremos en el equivalente de Python de "hablar" (en lenguaje humano a perro). Lo bueno de decirle a Python que hable es que incluso podemos decirle qué decir dándole un mensaje entre comillas:

Copyright 2026

```
print('Hello world!')
```

E incluso hemos escrito nuestra primera oración de Python sintácticamente correcta. Nuestra oración comienza con la función **print** (imprimir) seguida de una cadena de texto de nuestra elección entre comillas simples.

Conversando con Python

Ahora que tenemos una palabra y una oración simple que conocemos en Python, necesitamos saber cómo iniciar una conversación con Python para probar nuestras nuevas habilidades lingüísticas.

Antes de poder conversar con Python, primero debes instalar el software de Python en tu ordenador y aprender a iniciar Python. Esto es demasiado detallado para este capítulo, por lo que te sugiero que consulte [www.py4e.com] (http://www.py4e.com), donde tengo instrucciones detalladas y recomendaciones para configurar e iniciar Python en sistemas Macintosh y Windows . En algún momento, estarás en una ventana de comandos o terminal y escribirás **python** y el intérprete de Python comenzará a ejecutarse en modo interactivo. Aparecerá de la siguiente manera:

```
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.
>>>
```

El indicador `>>>` es la forma en que el intérprete de Python te pregunta: "¿Qué quieres que haga ahora?" Python está listo para tener una conversación contigo. Todo lo que tienes que saber es cómo hablar el idioma Python.

Digamos, por ejemplo, que no conocías ni las palabras ni las oraciones más simples del lenguaje Python. Es posible que desees utilizar la línea estándar que usan los astronautas cuando aterrizan en un planeta lejano y tratan de hablar con los habitantes del planeta:

```
>>> I come in peace, please take me to your leader
  File "<stdin>", line 1
    I come in peace, please take me to your leader
         ^
```

```
SyntaxError: invalid syntax
>>>
```

Esto no va tan bien. A menos que pienses en algo rápido, es probable que los habitantes del planeta te apuñalen con sus lanzas, te pongan a escupir, te asen al fuego y te coman para la cena.

Afortunadamente, trajiste una copia de este libro en tus viajes, y accedes a esta misma página e intentas nuevamente:

```
>>> print('Hello world!')
Hello world!
```

Esto se ve mucho mejor, así que intenta comunicar algo más:

```
>>> print('You must be the legendary god that comes from the sky')
You must be the legendary god that comes from the sky
>>> print('We have been waiting for you for a long time')
We have been waiting for you for a long time
>>> print('Our legend says you will be very tasty with mustard')
Our legend says you will be very tasty with mustard
>>> print 'We will have a feast tonight unless you say
  File "<stdin>", line 1
    print 'We will have a feast tonight unless you say
                                                      ^
SyntaxError: Missing parentheses in call to 'print'
>>>
```

La conversación transcurrió muy bien durante un tiempo y luego cometiste el más mínimo error al usar el lenguaje y Python sacó las lanzas.

En este punto, también debes darte cuenta de que, si bien Python es increíblemente complejo, poderoso y muy exigente con la sintaxis que utilizas para comunicarte con él, Python no es **inteligente**. Realmente solo estás teniendo una conversación contigo mismo, pero usando la sintaxis adecuada.

En cierto sentido, cuando utilizas un programa escrito por otra persona, la conversación es entre usted y los otros programadores con Python que actúa como intermediario. Python es una forma para que los creadores de programas expresen cómo se supone que la conversación debe continuar. Y en unos pocos capítulos más, serás uno de esos programadores que usan Python para

hablar con los usuarios de tu programa.

Antes de dejar nuestra primera conversación con el intérprete de Python, probablemente debas saber la forma correcta de decir "adiós" cuando interactúas con los habitantes de Planet Python:

```
>>> good-bye
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'good' 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()
```

Notarás que el error es diferente para los dos primeros intentos incorrectos. El segundo error es diferente porque **if** es una palabra reservada y Python vio la palabra reservada y pensó que estábamos tratando de decir algo, pero te equivocaste en la sintaxis de la oración.

La forma correcta de decir "adiós" a Python es ingresar **quit()** en el indicador interactivo `>>>`. Probablemente te haya tomado bastante tiempo adivinarlo, por lo que tener un libro a la mano probablemente te resulte útil.

Terminología: intérprete y compilador

Python es un lenguaje **de alto nivel** destinado a ser relativamente sencillo para que los humanos lean y escriban y para que los ordenadores lean y procesen. Otros lenguajes de alto nivel incluyen Java, C ++, PHP, Ruby, Basic, Perl, JavaScript y muchos más. El hardware real dentro de la Unidad Central de Procesamiento (CPU) no comprende ninguno de estos lenguajes de alto nivel.

La CPU entiende un idioma que llamamos **lenguaje de máquina**. El lenguaje de máquina es muy simple y francamente muy tedioso de escribir porque está representado en ceros y unos:

```
001010001110100100101010000001111
11100110000011101010010101101101
...
```

El lenguaje de máquina parece bastante simple en la superficie, dado que solo hay ceros y unos, pero su sintaxis es aún más compleja y mucho más compleja que Python. Muy pocos programadores escriben lenguaje de máquina. En su lugar, creamos varios traductores para permitir que los programadores escriban en lenguajes de alto nivel como Python o JavaScript y estos traductores convierten los programas al lenguaje de máquina para su ejecución real por parte de la CPU.

Dado que el lenguaje de la máquina está vinculado al hardware de el ordenador, el lenguaje de la máquina no es **portátil** a través de diferentes tipos de hardware. Los programas escritos en lenguajes de alto nivel se pueden mover entre diferentes ordenadors utilizando un intérprete diferente en la nueva máquina o compilando el código para crear una versión del programa en la máquina.

Estos traductores de lenguaje de programación se dividen en dos categorías generales: (1) intérpretes y (2) compiladores.

Un **intérprete** lee el código fuente del programa como está escrito por el programador, analiza el código fuente e interpreta las instrucciones sobre la marcha. Python es un intérprete y cuando ejecutamos Python de forma interactiva, podemos escribir una línea de Python (una oración) y Python la procesa de inmediato y está lista para que escribamos otra línea de Python.

Algunas de las líneas le dicen a Python que quieres que recuerde algún valor para más adelante. Necesitamos elegir un nombre para que se recuerde ese valor y podemos usar ese nombre simbólico para recuperar el valor más adelante. Utilizamos el término **variable** para referirnos a las etiquetas que usamos para estos 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 poder recuperar el valor más adelante. Verificamos que Python realmente haya recordado el valor

utilizando **print**. Luego le pedimos a Python que recupere **x** y lo multiplique por siete y ponga el nuevo valor calculado en **y**. Luego le pedimos a Python que imprima el valor actualmente en **y**.

Aunque estamos escribiendo estos comandos en Python una línea a la vez, Python los trata como una secuencia ordenada de declaraciones, con declaraciones posteriores que pueden recuperar datos creados anteriomente. Estamos escribiendo nuestro primer párrafo simple con cuatro oraciones en un orden lógico y significativo.

Está en la naturaleza de un **intérprete** poder tener una conversación interactiva como se muestra arriba. A un **compilador** debemos entregarle todo el programa en un archivo, y luego ejecuta un proceso para traducir el código fuente de alto nivel al lenguaje de máquina y luego el compilador coloca el lenguaje de máquina resultante en un archivo para su posterior ejecución.

Si tienes un sistema Windows, a menudo estos programas de lenguaje de máquina ejecutable tienen un sufijo de ".exe" o ".dll" que significa "ejecutable" y "biblioteca de enlace dinámico" respectivamente. En Linux y Macintosh, no hay ningún sufijo que marque de forma única un archivo como ejecutable.

Si fueras a abrir un archivo ejecutable en un editor de texto, se vería completamente loco y sería 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 es fácil leer o escribir lenguaje de máquina, por lo que es bueno que tengamos **intérpretes** y **compiladores** que nos permiten escribir en lenguajes de alto nivel como Python o C.

Ahora, en este punto de nuestra discusión sobre compiladores e intérpretes, deberías preguntarte un poco sobre el intérprete de Python. ¿En qué idioma está escrito? ¿Está escrito en un lenguaje compilado? Cuando escribimos "python", ¿qué está sucediendo exactamente?

El intérprete de Python está escrito en un lenguaje de alto nivel llamado "C". Puede consultar el código fuente real del intérprete de Python en [www.python.org] (http://www.python.org). Entonces, Python es un programa en sí mismo y está compilado en código de máquina. Cuando instaló Python en su ordenador (o el proveedor lo instaló), hizo una copia en código de máquina del programa Python traducido en su sistema. En Windows, el código de máquina ejecutable para Python en sí mismo es probablemente un archivo con un nombre como:

```
C:\Python37\python.exe
```

Eso es más de lo que realmente necesitas saber para ser un programador de Python, pero a veces vale la pena responder esas pequeñas y molestas preguntas desde el principio.

Escribiendo un programa

Escribir comandos en el intérprete de Python es una excelente manera de experimentar con las funciones de Python, pero no se recomienda para resolver problemas más complejos.

Cuando queremos escribir un programa, usamos un editor de texto para escribir las instrucciones de Python en un archivo, que se denomina un **script**. Por convención, los scripts de Python tienen nombres que terminan con `.py`.

Para ejecutar el script, debes decirle al intérprete de Python el nombre del archivo. En una ventana de comandos de Unix o Windows, escribirías `python hello.py` de la siguiente manera:

```
csev$ cat hello.py
print('Hello world!')
csev$ python hello.py
Hello world!
csev$
```

El "csev $" es el indicador del sistema operativo y el "cat hello.py" nos muestra que el archivo "hello.py" tiene un programa de Python de una línea para imprimir una cadena.

Llamamos al intérprete de Python y le decimos que lea el código fuente del archivo "hello.py" en lugar de pedirnos líneas interactivas de código de Python.

Notarás que no había necesidad de tener **quit()** al final del programa Python en el archivo. Cuando Python está leyendo el código fuente de un archivo, sabe que debe detenerse cuando llega al final del archivo.

¿Qué es un programa?

La definición de un **programa** en su forma más básica es una secuencia de sentencias de Python que se han creado para hacer algo. Incluso nuestro sencillo script **hello.py** es un programa. Es un programa de una línea y no es particularmente útil, pero en la definición más estricta, es un programa de Python.

Podría ser más fácil entender qué es un programa pensando en un problema para el cual se podría construir un programa, y luego mirando un programa que solucione ese problema.

Digamos que estás haciendo una investigación de Computación Social en las publicaciones de Facebook y estás interesado en la palabra más utilizada en una serie de publicaciones. Podrías imprimir la corriente de publicaciones de Facebook y estudiar detenidamente el texto buscando la palabra más común, pero eso llevaría mucho tiempo y sería muy propenso a errores. Sería inteligente escribir un programa Python para manejar la tarea de manera rápida y precisa, de modo que puedas pasar el fin de semana haciendo algo divertido.

Por ejemplo, mira el siguiente texto sobre un payaso y un automóvil. Calcula la palabra más común y cuántas veces aparece.

```
the clown ran after the car and the car ran into the tent
and the tent fell down on the clown and the car
```

Luego imagina que estás haciendo esta tarea mirando millones de líneas de texto. Francamente, sería más rápido aprender Python y escribir un programa Python para contar las palabras que escanearlas manualmente.

La noticia aún mejor es que ya se me ocurrió un programa simple para encontrar la palabra más común en un archivo de texto. Lo escribí, lo probé y ahora te lo doy para que lo uses y puedas ahorrar algo de tiempo.

https://trinket.io/embed/python3/d51dc614a8

Ni siquiera necesitas saber Python para usar este programa. Deberás leer el Capítulo 10 de este libro para comprender completamente las impresionantes técnicas de Python que se utilizaron para crear el programa. Tú eres el usuario final, simplemente utiliza el programa y disfruta de su inteligencia y de cómo te ahorró tanto esfuerzo manual. Simplemente escribe el código en un archivo llamado **words.py** y ejecútalo o descárgate el código fuente de https://www.py4e.com/code3/ y ejecútalo.

Este es un buen ejemplo de cómo Python actúa como intermediario entre tú (el usuario final) y yo (el programador). Python es una forma de intercambiar secuencias de instrucciones útiles (es decir, programas) en un lenguaje común que puede ser utilizado por cualquier persona que instale Python en su ordenador. Así que ninguno de los dos está hablando **con Python**, en lugar de eso, nos estamos comunicando unos con otros **a través de** Python.

Los componentes básicos de los programas

En los siguientes capítulos, aprenderemos más sobre el vocabulario, la estructura de las oraciones, la estructura de los párrafos y la estructura de la historia de Python. Aprenderemos sobre las potentes capacidades de Python y cómo usarlas para crear programas útiles.

Hay algunos patrones conceptuales de bajo nivel que utilizamos para construir programas. Estas construcciones no son solo para programas de Python, sino que forman parte de todos los lenguajes de programación, desde el lenguaje de máquina hasta los lenguajes de alto nivel.

entrada (input)

Obtener datos del "mundo exterior". Esto podría ser leer datos de un archivo, o incluso algún tipo de sensor como un micrófono o un GPS. En nuestros programas iniciales, nuestra entrada provendrá del usuario que escribe datos en el teclado.

salida (output)

Muestra los resultados del programa en una pantalla o guárdalos en un archivo o tal vez envíalos a un dispositivo como un altavoz para reproducir música o leer texto.

ejecución secuencial

Realiza las instrucciones una tras otra en el orden en que se encuentran en el script.

ejecución condicional

Verifica ciertas condiciones y luego ejecuta u omite una secuencia de sentencias.

ejecución iterativa

Realizar algún conjunto de declaraciones repetidamente, generalmente con alguna variación.

reutilizar

Escribe un conjunto de instrucciones una vez y dales un nombre. Luego reutiliza esas instrucciones según sea necesario en todo el programa.

Suena casi demasiado simple para ser verdad, y por supuesto nunca es tan simple. Es como decir que caminar es simplemente "poner un pie delante del otro". El "arte" de escribir un programa es componer y tejer estos elementos básicos juntos muchas veces para producir algo que sea útil para sus usuarios.

El programa de conteo de palabras de arriba usa directamente todos estos patrones excepto uno.

¿Qué podría salir mal?

Como vimos en nuestras primeras conversaciones con Python, debemos comunicarnos con mucha precisión cuando escribimos el código de Python. La desviación o error más pequeño hará que Python deje de mirar tu programa.

Los programadores principiantes a menudo toman el hecho de que Python no deja espacio para errores como evidencia de que Python es malo, odioso y cruel. Si bien a Python parece gustarle todo el mundo, Python en realidad los conoce personalmente y guarda rencor contra ellos. Debido a este rencor, Python toma nuestros programas perfectamente escritos y los rechaza como "no aptos" solo para atormentarnos.

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

>>> I hate you Python!
  File "<stdin>", line 1
```

```
    I hate you Python!
         ^
SyntaxError: invalid syntax
>>> if you come out of there, I would teach you a lesson
  File "<stdin>", line 1
    if you come out of there, I would teach you a lesson
               ^
SyntaxError: invalid syntax
>>>
```

Hay poco que ganar discutiendo con Python. Es solo una herramienta. No tiene emociones y está listo para servirte cuando lo necesite. Sus mensajes de error suenan ásperos, pero son solo la llamada de ayuda de Python. Ha examinado lo que escribiste y simplemente no puede entender lo que has ingresado.

Python es mucho más como un perro, te ama incondicionalmente, tiene algunas palabras clave que entiende, te mira con una mirada dulce en su cara (`>>>`), y te espera para que digas algo que entiendas. Cuando Python dice "SyntaxError: sintaxis no válida", simplemente mueve la cola y dice: "Parecías decir algo, pero no entiendo lo que quisiste decir, pero sigue hablando (`>>>`)."

A medida que sus programas se vuelven cada vez más sofisticados, encontrará tres tipos generales de errores:

Errores sintácticos

Estos son los primeros errores que cometerás y los más fáciles de corregir. Un error de sintaxis significa que has violado las reglas de "sintaxis" de Python. Python hace todo lo posible para señalar la línea y el carácter en el que notó que estabas confundido. La única parte difícil de los errores de sintaxis es que, a veces, el error que debe solucionarse es, en realidad, más temprano en el programa que en el que Python notó que estabas confundido. Por lo tanto, la línea y el carácter que Python indica en un error de sintaxis pueden ser solo un punto de partida para tu investigación.

Errores lógicos

Un error lógico es cuando su programa tiene una buena sintaxis pero hay un error en el orden de las declaraciones o quizás un error en la forma en que las declaraciones se relacionan entre sí. Un buen ejemplo de un error lógico podría ser: "tome un trago de su botella de agua, colóquelo en su mochila, camine hasta la biblioteca y luego vuelva a colocar la tapa en la botella".

Copyright 2026

Errores semánticos

Un error semántico es cuando tu descripción de los pasos a seguir es sintácticamente perfecta y en el orden correcto, pero simplemente hay un error en el programa. El programa es perfectamente correcto, pero no hace lo que tú **querías** que haga. Un ejemplo simple sería si le dieras a una persona instrucciones para llegar a un restaurante y dijeras: "... cuando llegue a la intersección con la estación de servicio, gire a la izquierda y avance una milla y el restaurante es un edificio rojo a su izquierda". Su amigo llega tarde y lo llama para decirle que está en una granja y que está caminando detrás de un granero, sin ninguna señal de restaurante. Entonces dices "¿giraste a la izquierda o a la derecha en la estación de servicio?" y dicen: "Seguí tus instrucciones a la perfección, las tengo escritas, dice: gira a la izquierda y ve una milla en la gasolinera". Luego dices: "Lo siento mucho, porque aunque mis instrucciones fueron sintácticamente correctas, tristemente contenían un error semántico pequeño pero no detectado".

Nuevamente, en los tres tipos de errores, Python simplemente está haciendo todo lo posible por hacer exactamente lo que tú pediste.

El viaje del aprendizaje

A medida que avance en el resto del libro, no tengas miedo si los conceptos no parecen encajar bien a la primera. Cuando estabas aprendiendo a hablar, no fue un problema durante los primeros años que hicieras ruidos de gorgoteo. Y estuvo bien si tardaste seis meses en pasar de un vocabulario simple a oraciones simples y tardaste 5-6 años más en pasar de las oraciones a los párrafos, y algunos años más para poder escribir un cuento completo completo interesante en tú sólo.

Queremos que aprendas Python mucho más rápido, por lo que enseñamos todo al mismo tiempo en los próximos capítulos. Pero es como aprender un nuevo idioma que toma tiempo absorber y entender antes de que se sienta natural. Esto nos lleva a cierta confusión a medida que visitamos y revisamos los temas para tratar de que veas el panorama general mientras definimos los pequeños fragmentos que conforman ese panorama general. Si bien el libro está escrito de manera lineal, y si estás tomando un curso, progresarás de manera lineal, no dudes en ser muy no lineal en la forma en que abordas el material. Mira hacia adelante y hacia atrás y lee con ligereza. Al hojear material más avanzado sin comprender completamente los detalles, puedes comprender mejor el "¿por qué?" de la programación. Al revisar el material anterior e incluso rehacer ejercicios anteriores, te darás cuenta de que realmente aprendiste mucho, incluso si el material que estás mirando actualmente parece un poco impenetrable.

Copyright 2026

Por lo general, cuando estás aprendiendo tu primer lenguaje de programación, hay algunos maravillosos momentos "¡Ah Hah!" en los que puedes mirar desde una roca con un martillo y un cincel, alejarte, y ver que de hecho estás construyendo una hermosa escultura.

Si algo parece particularmente difícil, generalmente merece la pena permanecer despierto toda la noche y mirarlo fijamente. Toma un descanso, toma una siesta, toma un refrigerio, explícale a alguien (o quizás a tu perro) con qué está teniendo problemas y luego vuelve con los ojos frescos. Te aseguro que una vez que aprendas los conceptos de programación en el libro, mirarás hacia atrás y verás que todo fue realmente fácil y elegante, y que simplemente te tomó un poco de tiempo absorberlo.

Ejercicios (voluntarios y sin corrección)

Ejercicio 1: ¿Cuál es la función de la memoria secundaria en un ordenador?

a) Ejecutar todos los cálculos y la lógica del programa
b) Recuperar páginas web a través de Internet
c) Almacenar información a largo plazo, incluso más allá de un ciclo de alimentación
d) Tomar información del usuario

Solución
a

Ejercicio 2: ¿Cuál de los siguientes contiene "código de máquina"?

a) El intérprete de Python
b) El teclado
c) Archivo fuente de Python
d) Un documento de procesamiento de texto

Solución
a

Ejercicio 3: ¿Qué está mal en el siguiente código?

```
>>> primt 'Hello world!'
  File "<stdin>", line 1
    primt 'Hello world!'
                        ^
SyntaxError: invalid syntax
>>>
```

Solución

primt tendría que ser print

Ejercicio 4: ¿En qué parte de el ordenador hay una variable como "x" almacenada después de que termina la siguiente línea de Python?

```
x = 123
```

a) Unidad de procesamiento central
b) Memoria principal
c) Memoria secundaria
d) Dispositivos de entrada
e) Dispositivos de salida

Solución

c

Ejercicio 5: ¿Qué imprimirá el siguiente programa?

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

a) 43
b) 44
c) x + 1

Copyright 2026

d) Error porque x = x + 1 no es posible matemáticamente

https://trinket.io/embed/python/47afa56dd668

Ejercicio 6: Explique cada uno de los siguientes ejemplos con un ejemplo de una capacidad humana: (1) Unidad central de procesamiento, (2) Memoria principal, (3) Memoria secundaria, (4) Dispositivo de entrada y (5) Dispositivo de salida. Por ejemplo, "¿Cuál es el equivalente humano a una unidad central de procesamiento?"

Ejercicio 7 ¿Cómo arreglas un "Error de sintaxis"

Revision #5
Created 2025-04-05 08:35:27 CEST by Javier Quintana
Updated 2025-04-05 12:28:05 CEST by Javier Quintana

Copyright 2026