Modulo 1

Bienvenido a Fundamentos de Programación en Python - Parte 1

Módulo 1
Introducción a Python y a la programación.

Módulo 2
Tipos de datos, variables, operaciones básicas de entrada y salida, operadores básicos.

Módulo 3
Valores booleanos, ejecución condicional, bucles, listas y procesamiento de listas, operaciones lógicas y bit a bit.

Módulo 4
Funciones, tuplas, diccionarios y procesamiento de datos.

Fundamentos de Programación en Python: Módulo 1

En este módulo, aprenderás sobre:

- Fundamentos de programación.
- Establecimiento de tu entorno de programación.
- Compilación vs. interpretación.
- Introducción a Python.

¿Cómo funciona un programa de computadora?

Este curso tiene como objetivo explicar el lenguaje Python y para que se utiliza. Vamos a comenzar desde los fundamentos básicos.

Un programa hace que una computadora sea utilizable. Sin un programa, una computadora, incluso la más poderosa, no es más que un objeto. Del mismo modo, sin un pianista, un piano no es más que una caja de madera.

Las computadoras pueden realizar tareas muy complejas, pero esta habilidad no es innata. La naturaleza de una computadora es bastante diferente.

Una computadora puede ejecutar solo operaciones extremadamente simples, por ejemplo, una computadora no puede evaluar el valor de una función matemática complicada por sí misma, aunque esto no está más allá de los límites posibles en un futuro próximo.

Las computadoras contemporáneas solo pueden evaluar los resultados de operaciones muy fundamentales, como sumar o dividir, pero pueden hacerlo muy rápido y pueden repetir estas acciones prácticamente cualquier cantidad de veces.

Imagina que quieres saber la velocidad promedio que has alcanzado durante un largo viaje. Sabes la distancia, sabes el tiempo, necesitas la velocidad.

Naturalmente, la computadora podrá calcular esto, pero la computadora no es consciente de cosas como la distancia, la velocidad o el tiempo. Por lo tanto, es necesario instruir a la computadora para que:

- Acepte un número que represente la distancia.
- Acepte un número que represente el tiempo de viaje.
- Divida el valor anterior por el segundo y almacene el resultado en la memoria.
- Muestre el resultado (representando la velocidad promedio) en un formato legible.

Estas cuatro acciones simples forman un **programa**. Por supuesto, estos ejemplos no están formalizados, y están muy lejos de lo que la computadora puede entender, pero son lo suficientemente buenos como para traducirlos a un idioma que la computadora pueda aceptar.

La palabra clave es el **lenguaje**.

Lenguajes naturales vs. Lenguajes de programación

Un lenguaje es un medio (y una herramienta) para expresar y registrar pensamientos. Hay muchos lenguajes a nuestro alrededor. Algunos de ellos no requieren hablar ni escribir, como el lenguaje corporal. Es posible expresar tus sentimientos más profundos de manera muy precisa sin decir una palabra.

Otro lenguaje que empleas cada día es tu lengua materna, que utilizas para manifestar tu voluntad y para pensar en la realidad. Las computadoras también tienen su propio lenguaje, llamado lenguaje **máquina**, el cual es muy rudimentario.

Una computadora, incluso la más técnicamente sofisticada, carece incluso de un rastro de inteligencia. Se podría decir que es como un perro bien entrenado, responde solo a un conjunto predeterminado de comandos conocidos.

Los comandos que reconoce son muy simples. Podemos imaginar que la computadora responde a órdenes como "Toma ese número, divídelo por otro y guarda el resultado".

Un conjunto completo de comandos conocidos se llama **lista de instrucciones**, a veces abreviada **IL** (por sus siglas en inglés de Instruction List). Los diferentes tipos de computadoras pueden variar según el tamaño de sus IL y las instrucciones pueden ser completamente diferentes en diferentes modelos.

Nota: los lenguajes máquina son desarrollados por humanos.

Ninguna computadora es actualmente capaz de crear un nuevo idioma. Sin embargo, eso puede cambiar pronto. Por otro lado, las personas también usan varios idiomas muy diferentes, pero estos idiomas se crearon ellos mismos. Además, todavía están evolucionando.

Cada día se crean nuevas palabras y desaparecen las viejas. Estos lenguajes se llaman **lenguajes naturales**.

¿Qué hace a un lenguaje?

Podemos decir que cada idioma (máquina o natural, no importa) consta de los siguientes elementos:

ALFABETO
Un conjunto de símbolos utilizados para formar palabras de un determinado idioma (por ejemplo, el alfabeto latino para el inglés, el alfabeto cirílico para el ruso, el kanji para el japonés, etc.).

LÉXICO
(También conocido como diccionario) un conjunto de palabras que el idioma ofrece a sus usuarios (por ejemplo, la palabra "computadora" proviene del diccionario en inglés, mientras que "abcde" no; la palabra "chat" está presente en los diccionarios de inglés y francés, pero sus significados son diferentes.

SINTAXIS
Un conjunto de reglas (formales o informales, escritas o interpretadas intuitivamente) utilizadas para precisar si una determinada cadena de palabras forma una oración válida (por ejemplo, "Soy una serpiente" es una frase sintácticamente correcta, mientras que "Yo serpiente soy una" no lo es).

SEMÁNTICA
Un conjunto de reglas que determinan si una frase tiene sentido (por ejemplo, "Me comí una dona" tiene sentido, pero "Una dona me comió" no lo tiene).

La IL es, de hecho, **el alfabeto de un lenguaje máquina**. Este es el conjunto de símbolos más simple y principal que podemos usar para dar comandos a una computadora. Es la lengua materna de la computadora.

Desafortunadamente, esta lengua está muy lejos de ser una lengua materna humana. Todos (tanto las computadoras como los humanos) necesitamos algo más, un lenguaje común para las computadoras y los seres humanos, o un puente entre los dos mundos diferentes.

Necesitamos un lenguaje en el que los humanos puedan escribir sus programas y un lenguaje que las computadoras puedan usar para ejecutar los programas, que es mucho más complejo que el lenguaje máquina y más sencillo que el lenguaje natural.

Tales lenguajes son a menudo llamados lenguajes de programación de alto nivel. Son algo similares a los naturales en que usan símbolos, palabras y convenciones legibles para los humanos. Estos lenguajes permiten a los humanos expresar comandos a computadoras que son mucho más complejas que las ofrecidas por las IL.

Un programa escrito en un lenguaje de programación de alto nivel se llama **código fuente** (en contraste con el código de máquina ejecutado por las computadoras). Del mismo modo, el archivo que contiene el código fuente se llama **archivo fuente**.

Compilación vs. Interpretación

La programación de computadora es el acto de establecer una secuencia de instrucciones con la cual se causará el efecto deseado. El efecto podría ser diferente en cada caso específico: depende de la imaginación, el conocimiento y la experiencia del programador.

Por supuesto, tal composición tiene que ser correcta en muchos sentidos, tales como:

- **Alfabéticamente:** Un programa debe escribirse en una secuencia de comandos reconocible, por ejemplo, el Romano, Cirílico, etc.
- **Léxicamente:** Cada lenguaje de programación tiene su diccionario y necesitas dominarlo; afortunadamente, es mucho más simple y más pequeño que el diccionario de cualquier lenguaje natural.
- **Sintácticamente:** Cada idioma tiene sus reglas y deben ser obedecidas.
- **Semánticamente:** El programa tiene que tener sentido.

Desafortunadamente, un programador también puede cometer errores en cada uno de los cuatro sentidos anteriores. Cada uno de ellos puede hacer que el programa se vuelva completamente inútil.

Supongamos que ha escrito correctamente un programa. ¿Cómo persuadimos a la computadora para que la ejecute? Tienes que convertir tu programa en lenguaje máquina. Afortunadamente, la traducción puede ser realizada por una computadora, haciendo que todo el proceso sea rápido y eficiente.

Hay dos formas diferentes de **transformar un programa de un lenguaje de programación de alto nivel a un lenguaje de máquina**:

COMPILACIÓN - El programa fuente se traduce una vez (sin embargo, esta ley debe repetirse cada vez que se modifique el código fuente) obteniendo un archivo (por ejemplo, un archivo .exe si el código está diseñado para ejecutarse en MS Windows) que contiene el código de la máquina; ahora puedes distribuir el archivo en todo el mundo; el programa que realiza esta traducción se llama compilador o traductor.

INTERPRETACIÓN - Tú (o cualquier usuario del código) puedes traducir el programa fuente cada vez que se ejecute; el programa que realiza este tipo de transformación se denomina intérprete, ya que interpreta el código cada vez que está destinado a ejecutarse; también significa que no puede distribuir el código fuente tal como está, porque el usuario final también necesita que el intérprete lo ejecute.

Debido a algunas razones muy fundamentales, un lenguaje de programación de alto nivel particular está diseñado para caer en una de estas dos categorías.

Hay muy pocos idiomas que se pueden compilar e interpretar. Por lo general, un lenguaje de programación se proyecta con este factor en la mente de sus constructores: ¿Se compilará o interpretará?

¿Qué hace realmente el intérprete?

Supongamos una vez más que has escrito un programa. Ahora, existe como un **archivo de computadora**: un programa de computadora es en realidad una pieza de texto, por lo que el código fuente generalmente se coloca en **archivos de texto**. Nota: debe ser **texto puro**, sin ninguna decoración, como diferentes fuentes, colores, imágenes incrustadas u otros medios. Ahora tienes que invocar al intérprete y dejar que lea el archivo fuente.

El intérprete lee el código fuente de una manera que es común en la cultura occidental: de arriba hacía abajo y de izquierda a derecha. Hay algunas excepciones: se cubrirán más adelante en el curso.

En primer lugar, el intérprete verifica si todas las líneas subsiguientes son correctas (utilizando los cuatro aspectos tratados anteriormente).

Si el compilador encuentra un error, termina su trabajo inmediatamente. El único resultado en este caso es un **mensaje de error**. El intérprete le informará dónde se encuentra el error y qué lo causó. Sin embargo, estos mensajes pueden ser engañosos, ya que el intérprete no puede seguir tus intenciones exactas y puede detectar errores a cierta distancia de tus causas reales.

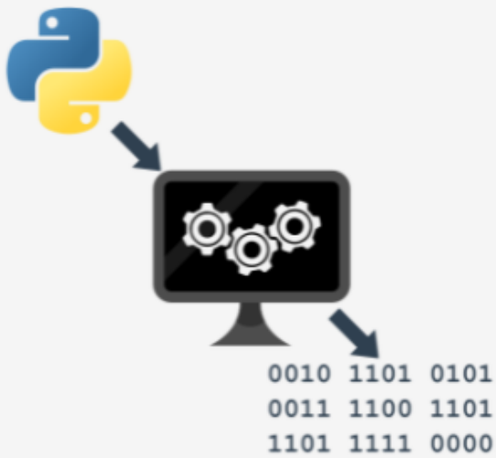
Por ejemplo, si intentas usar una entidad de un nombre desconocido, causará un error, pero el error se descubrirá en el lugar donde se intenta usar la entidad, no donde se introdujo el nombre de la nueva entidad.

En otras palabras, la razón real generalmente se ubica un poco antes en el código, por ejemplo, en el lugar donde se tuvo que informar al intérprete de que usaría la entidad del nombre.

Si la línea se ve bien, el intérprete intenta ejecutarla (nota: cada línea generalmente se ejecuta por separado, por lo que el trío "Lectura - Verificación - Ejecución", pueden repetirse muchas veces, más veces que el número real de líneas en el archivo fuente, como algunas partes del código pueden ejecutarse más de una vez).

También es posible que una parte significativa del código se ejecute con éxito antes de que el intérprete encuentre un error. Este es el comportamiento normal en este modelo de ejecución.

Puedes preguntar ahora: ¿Cuál es mejor? ¿El modelo de "compilación" o el modelo de "interpretación"? No hay una respuesta obvia. Si hubiera habido, uno de estos modelos habría dejado de existir hace mucho tiempo. Ambos tienen sus ventajas y sus desventajas.



Compilación vs. Interpretación - Ventajas y Desventajas

	COMPILACIÓN	INTERPRETACIÓN
VENTAJAS	• La ejecución del código traducido suele ser más rápida. • Solo el usuario debe tener el compilador; el usuario final puede usar el código sin él. • El código traducido se almacena en lenguaje máquina, ya que es muy difícil de entender, es probable que tus propios inventos y trucos de programación sigan siendo secreto.	• Puede ejecutar el código en cuanto lo complete; no hay fases adicionales de traducción. • El código se almacena utilizando el lenguaje de programación, no el de la máquina; esto significa que puede ejecutarse en computadoras que utilizan diferentes lenguajes máquina; no compila el código por separado para cada arquitectura diferente.
DESVENTAJAS	• La compilación en sí misma puede llevar mucho tiempo; es posible que no puedas ejecutar tu código inmediatamente después de cualquier modificación. • Tienes que tener tantos compiladores como plataformas de hardware en los que deseas que se ejecute su código.	• No esperes que la interpretación incremente tu código a alta velocidad: tu código compartirá la potencia de la computadora con el intérprete, por lo que no puede ser realmente rápido. • Tanto tú como el usuario final deben tener el intérprete para ejecutar su código.

¿Qué significa todo esto para ti?

- Python es un **lenguaje interpretado**. Esto significa que hereda todas las ventajas y desventajas descritas. Por supuesto, agrega algunas de sus características únicas a ambos conjuntos.
- Si deseas programar en Python, necesitarás el **intérprete de Python**. No podrás ejecutar tu código sin él. Afortunadamente, **Python es gratis**. Esta es una de sus ventajas más importantes.

Debido a razones históricas, los lenguajes diseñados para ser utilizados en la manera de interpretación a menudo se llaman **lenguajes de programación**, mientras que los programas fuente codificados que los usan se llaman **scripts**.

¿Qué es Python?

Python es un lenguaje de programación de alto nivel, interpretado, orientado a objetos y de uso generalizado con semántica dinámica, que se utiliza para la programación de propósito general.

Y aunque puede que conozcas a la pitón como una gran serpiente, el nombre del lenguaje de programación Python proviene de una vieja serie de comedia de la BBC llamada **Monty Python's Flying Circus**.

En el apogeo de su éxito, el equipo de Monty Python estaba realizando sus escenas para audiencias en vivo en todo el mundo, incluso en el Hollywood Bowl.

Dado que Monty Python es considerado uno de los dos nutrientes fundamentales para un programador (el otro es la pizza), el creador de Python nombró el lenguaje en honor del programa de televisión.

¿Quién creó Python?

Una de las características sorprendentes de Python es el hecho de que en realidad es el trabajo de una persona. Por lo general, los grandes lenguajes de programación son desarrollados y publicados por grandes compañías que emplean a muchos profesionales, y debido a las normas de derechos de autor, es muy difícil nombrar a cualquiera de las personas involucradas en el proyecto. Python es una excepción.

No hay muchos idiomas cuyos autores son conocidos por su nombre. Python fue creado por **Guido van Rossum**, nacido en 1956 en Haarlem, Países Bajos. Por supuesto, Guido van Rossum no desarrolló y evolucionó todos los componentes de Python.

La velocidad con la que Python se ha extendido por todo el mundo es el resultado del trabajo continuo de miles de (muy a menudo anónimos) programadores, evaluadores, usuarios (muchos de ellos no son especialistas en TI) y entusiastas, pero hay que decir que la primera idea (la semilla de la que brotó Python) llegó a una cabeza: la de Guido.

Un proyecto de programación por hobby

Las circunstancias en las que se creó Python son un poco desconcertantes. Según Guido van Rossum:

> En diciembre de 1989, estaba buscando un proyecto de programación de "pasatiempo" que me mantendría ocupado durante la semana de Navidad. Mi oficina (...) estaría cerrada, pero tenía una computadora en casa y no mucho más en mis manos. Decidí escribir un intérprete para el nuevo lenguaje de scripting en el que había estado pensando últimamente: un descendiente de ABC que atraería a los hackers de Unix / C. Elegí Python como un título de trabajo para el proyecto, estando en un estado de ánimo ligeramente irreverente (y un gran fanático de Monty Python's Flying Circus).
> — *Guido van Rossum*

Los objetivos de Python

En 1999, Guido van Rossum definió sus objetivos para Python:

- Un lenguaje **fácil e intuitivo** tan poderoso como los de los principales competidores.
- De **código abierto**, para que cualquiera pueda contribuir a su desarrollo.
- El código que es tan **comprensible** como el inglés simple.
- **Adecuado para tareas cotidianas**, permitiendo tiempos de desarrollo cortos.

Unos 20 años después, está claro que todas estas intenciones se han cumplido. Algunas fuentes dicen que Python es el lenguaje de programación más popular del mundo, mientras que otros afirman que es el tercero o el quinto.

De cualquier manera, todavía ocupa un alto rango en el top ten de la PYPL Popularity of Programming Language y la TIOBE Programming Community Index.

Python no es una lengua joven. **Es maduro y digno de confianza**. No es una maravilla de un solo golpe. Es una estrella brillante en el firmamento de programación, y el tiempo dedicado a aprender Python es una muy buena inversión.

¿Qué hace especial a Python?

¿Por qué los programadores, jóvenes y viejos, experimentados y novatos, quieran usarlo? ¿Cómo fue que las grandes empresas adoptaron Python e implementaron sus productos estrella al usarlo?

Hay muchas razones. Ya hemos enumerado algunas de ellas, pero vamos a enumerarlas de una manera más práctica:

- Es **fácil de aprender** - El tiempo necesario para aprender Python es más corto que en muchos otros lenguajes; esto significa que es posible comenzar la programación real más rápido.
- Es **fácil de enseñar** - La carga de trabajo de enseñanza es menor que la que necesitan otros lenguajes; esto significa que el profesor puede poner más énfasis en las técnicas de programación generales (independientes del lenguaje), no gastando energía en trucos exóticos, extrañas excepciones y reglas incomprensibles.
- Es **fácil de utilizar** - Para escribir software nuevo; a menudo es posible escribir código más rápido cuando se usa Python.
- Es **fácil de entender** - A menudo, también es más fácil entender el código de otra persona más rápido si está escrito en Python.
- Es **fácil de obtener, instalar y desplegar** - Python es gratuito, abierto y multiplataforma; No todos los lenguajes pueden presumir de eso.

Por supuesto, Python también tiene sus inconvenientes:

- No es un demonio de la velocidad; Python no ofrece un rendimiento excepcional.
- En algunos casos puede ser resistente a algunas técnicas de prueba más simples, lo que puede significar que la depuración del código de Python puede ser más difícil que con otros lenguajes. Afortunadamente, cometer errores siempre es más difícil en Python.

También debe señalarse que Python no es la única solución de este tipo disponible en el mercado de TI.

Tiene muchos seguidores, pero hay muchos que prefieren otros lenguajes y ni siquiera consideran Python para sus proyectos.

Rivales de Python

Python tiene dos competidores directos, con propiedades y predisposiciones comparables. Estos son:

- **Perl** - un lenguaje de scripting originalmente escrito por Larry Wall.
- **Ruby** - un lenguaje de scripting originalmente escrito por Yukihiro Matsumoto.

El primero es más tradicional, más conservador que Python, y se parece a algunos de los buenos lenguajes antiguos derivados del lenguaje de programación C clásico.

En contraste, este último es más innovador y está más lleno de ideas nuevas. Python se encuentra en algún lugar entre estas dos creaciones.

Internet está lleno de foros con discusiones infinitas sobre la superioridad de uno de estos tres sobre los otros, si deseas obtener más información sobre cada uno de ellos.

¿Dónde podemos ver a Python en acción?

Lo vemos todos los días y en casi todas partes. Se utiliza ampliamente para implementar complejos **servicios de Internet** como motores de búsqueda, almacenamiento en la nube y herramientas, redes sociales, etc. Cuando utilizas cualquiera de estos servicios, en realidad estás muy cerca de Python.

Muchas **herramientas de desarrollo** se implementan en Python. Cada vez se escriben mas **aplicaciones de uso diario** en Python. Muchos **científicos** han abandonado las costosas herramientas patentadas y se han cambiado a Python. Muchos **evaluadores** de proyectos de TI han comenzado a usar Python para llevar a cabo procedimientos de prueba repetibles. La lista es larga.

¿Por qué no Python?

A pesar de la creciente popularidad de Python, todavía hay algunos nichos en los que Python está ausente o rara vez se ve:

- **Programación de bajo nivel** (a veces llamada programación "cercana al metal"): si deseas implementar un controlador o motor gráfico extremadamente efectivo, no se usaría Python
- **Aplicaciones para dispositivos móviles**: este territorio aún está a la espera de ser conquistado por Python, lo más probable es que suceda algún día.

Hay más de un Python

Hay dos tipos principales de Python, llamados Python 2 y Python 3.

Python 2 es una versión anterior del Python original. Su desarrollo se ha estancado intencionalmente, aunque eso no significa que no haya actualizaciones. Por el contrario, las actualizaciones se emiten de forma regular, pero no pretenden modificar el idioma de manera significativa. Prefieren arreglar cualquier error recién descubierto y agujeros de seguridad. La ruta de desarrollo de Python 2 ya ha llegado a un callejón sin salida, pero Python 2 en sí todavía está muy vivo.

Python 3 es la versión más nueva (para ser precisos, la actual) del lenguaje. Está atravesando su propio camino de evolución, creando sus propios estándares y hábitos.

El primero es más tradicional, más conservador que Python, y se parece a algunos de los buenos lenguajes antiguos derivados del lenguaje de programación C clásico.

Estas dos versiones de Python no son compatibles entre sí. Las secuencias de comandos de Python 2 no se ejecutarán en un entorno de Python 3 y viceversa, por lo que si deseas que un intérprete de Python 3 ejecute el código Python 2 anterior, la única solución posible es volver a escribirlo, no desde cero, por supuesto. Como grandes partes del código pueden permanecer intactas, pero tienes que revisar todo el código para encontrar todas las incompatibilidades posibles. Desafortunadamente, este proceso no puede ser completamente automatizado.

Es demasiado difícil, consume mucho tiempo, es demasiado caro y es demasiado arriesgado migrar una aplicación Python 2 antigua a una nueva plataforma. Es posible que reescribir el código le introduzca nuevos errores. Es más fácil y mas sensato dejar estos sistemas solos y mejorar el intérprete existente, en lugar de intentar trabajar dentro del código fuente que ya funciona.

Python 3 no es solo una versión mejorada de Python 2, es un lenguaje completamente diferente, aunque es muy similar a su predecesor. Cuando se miran a distancia, parecen ser los mismos, pero cuando se observan de cerca, se notan muchas diferencias.

Si estás modificando una solución Python existente, entonces es muy probable que esté codificada en Python 2. Esta es la razón por la que Python 2 todavía está en uso. Hay demasiadas aplicaciones de Python 2 existentes para descartarlo por completo.

NOTA

Si se va a comenzar un nuevo proyecto de Python, **deberías usar Python 3, esta es la versión de Python que se usará durante este curso**.

Es importante recordar que puede haber diferencias mayores o menores entre las siguientes versiones de Python 3 (p. Ej., Python 3.6 introdujo claves de diccionario ordenadas de forma predeterminada en la implementación de CPython). La buena noticia es que todas las versiones más nuevas de Python 3 son **compatibles** con las versiones anteriores de Python 3. Siempre que sea significativo e importante, siempre intentaremos resaltar esas diferencias en el curso.

Todos los ejemplos de código que encontrarás durante el curso se han probado con Python 3.4, Python 3.6 y Python 3.7.

Python alias CPython

Además de Python 2 y Python 3, hay más de una versión de cada uno.

En primer lugar, están los Pythons que mantienen las personas reunidas en torno a PSF (Python Software Foundation), una comunidad que tiene como objetivo desarrollar, mejorar, expandir y popularizar Python y su entorno. El presidente del PSF es el propio Guido van Rossum, y por esta razón, estos Pythons se llaman **canónicos**. También se consideran **Pythons de referencia**, ya que cualquier otra implementación del lenguaje debe seguir todos los estándares establecidos por el PSF.

Guido van Rossum utilizó el lenguaje de programación "C" para implementar la primera versión de su lenguaje y esta decisión aún está vigente. Todos los Pythons que vienen del PSF están escritos en el lenguaje "C". Hay muchas razones para este enfoque y tiene muchas consecuencias. Una de ellos (probablemente la más importante) es que gracias a él, Python puede ser portado y migrado fácilmente a todas las plataformas con la capacidad de compilar y ejecutar programas en lenguaje "C" (virtualmente todas las plataformas tienen esta característica, lo que abre muchas expansiones y oportunidades para Python).

Esta es la razón por la que la implementación de PSF a menudo se denomina **CPython**. Este es el Python más influyente entre todos los Pythons del mundo.

Cython

Otro miembro de la familia Python es **Cython**.

Cython es una de las posibles soluciones al rasgo de Python más doloroso: la falta de eficiencia. Los cálculos matemáticos grandes y complejos pueden ser fácilmente codificados en Python (mucho más fácil que en "C" o en cualquier otro lenguaje tradicional), pero la ejecución del código resultante puede requerir mucho tiempo.

¿Cómo se reconcilian estas dos contradicciones? Una solución es escribir tus ideas matemáticas usando Python, y cuando estés absolutamente seguro de que tu código es correcto y produce resultados válidos, puedes traducirlo a "C". Ciertamente, "C" se ejecutará mucho más rápido que Python puro.

Esto es lo que pretende hacer Cython: traducir automáticamente el código de Python (limpio y claro, pero no demasiado rápido) al código "C" (complicado y hablador, pero ágil).

Jython

Otra versión de Python se llama **Jython**.

"J" es para "Java". Imagina un Python escrito en Java en lugar de C. Esto es útil, por ejemplo, si desarrollas sistemas grandes y complejos escritos completamente en Java y deseas agregarles cierta flexibilidad de Python. El tradicional CPython puede ser difícil de integrar en un entorno de este tipo, ya que C y Java viven en mundos completamente diferentes y no comparten muchas ideas comunes.

Jython puede comunicarse con la infraestructura Java existente de manera más efectiva. Es por esto que algunos proyectos lo encuentran útil y necesario.

Nota: la implementación actual de Jython sigue los estándares de Python 2. Hasta ahora, no hay Jython conforme a Python 3.

PyPy y RPython

Echa un vistazo al logo de abajo. Es un rebus. ¿Puedes resolverlo?

Es un logotipo de **PyPy** - un Python dentro de un Python. En otras palabras, representa un entorno de Python escrito en un lenguaje similar a Python llamado **RPython** (Restricted Python). En realidad es un subconjunto de Python. El código fuente de PyPy no se ejecuta de manera interpretativa, sino que se traduce al lenguaje de programación C y luego se ejecuta por separado.

Esto es útil porque si deseas probar cualquier característica nueva que pueda ser o no introducida en la implementación de Python, es más fácil verificarla con PyPy que con CPython. Esta es la razón por la que PyPy es más una herramienta para las personas que desarrollan Python que para el resto de los usuarios.

Esto no hace que PyPy sea menos importante o menos serio que CPython.

Además, PyPy es compatible con el lenguaje Python 3.

Hay muchos más Pythons diferentes en el mundo. Los encontrarás sí los buscas, pero **este curso se centrará en CPython**.

¿Cómo obtener Python y cómo usarlo?

Hay varias formas de obtener tu propia copia de Python 3, dependiendo del sistema operativo que utilices.

Es probable que los usuarios de Linux tengan Python ya instalado - este es el escenario más probable, ya que la infraestructura de Python se usa de forma intensiva en muchos componentes del sistema operativo Linux.

Por ejemplo, algunas distribuciones pueden unir sus herramientas específicas con el sistema y muchas de estas herramientas, como los administradores de paquetes, a menudo están escritas en Python. Algunas partes de los entornos gráficos disponibles en el mundo de Linux también pueden usar Python.

Si eres un usuario de Linux, abre la terminal/consola y escribe:

```
python3
```

En el indicador de shell, presiona Enter y espera.

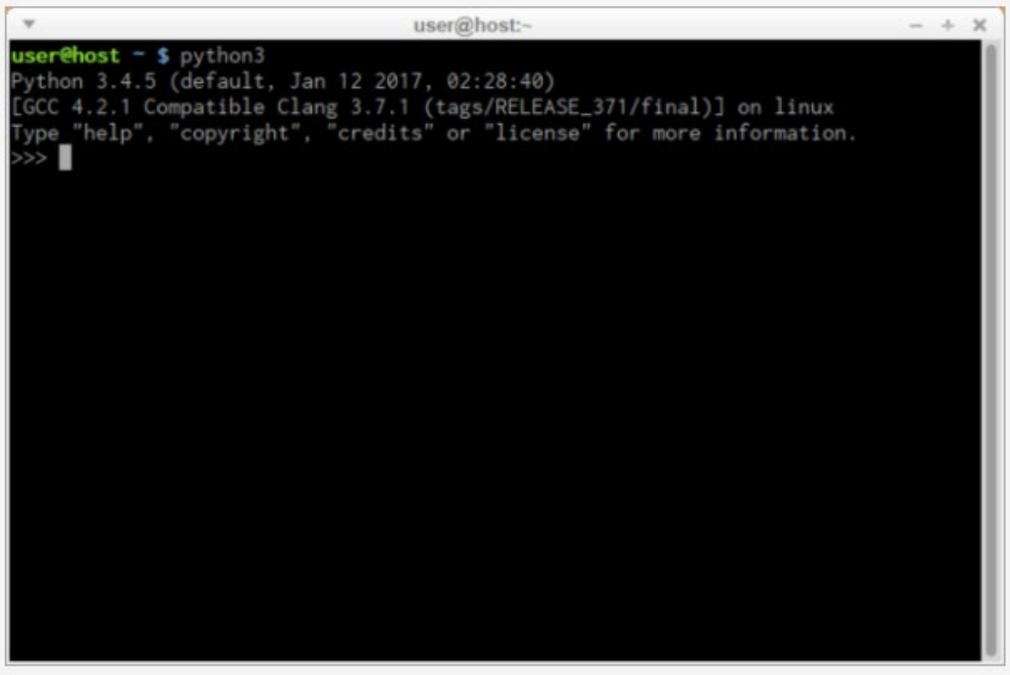
Si ves algo como esto:

```
Python 3.4.5 (default, Jan 12 2017, 02:28:40)
[GCC 4.2.1 Compatible Clang 3.7.1 (tags/RELEASE_371/final)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Entonces no tienes que hacer nada más.

Si Python 3 está ausente, consulta la documentación de Linux para saber cómo usar tu administrador de paquetes para descargar e instalar un paquete nuevo: el que necesitas se llama python3 o su nombre comienza con eso.

> Todos los usuarios que no sean Linux pueden descargar una copia en https://www.python.org/downloads/.



Descargando e instalando Python

Debido a que el navegador le dice al sitio web que se ingresó, el sistema operativo que se utiliza, el único paso que se debe seguir es hacer clic en la versión de Python que se desee.

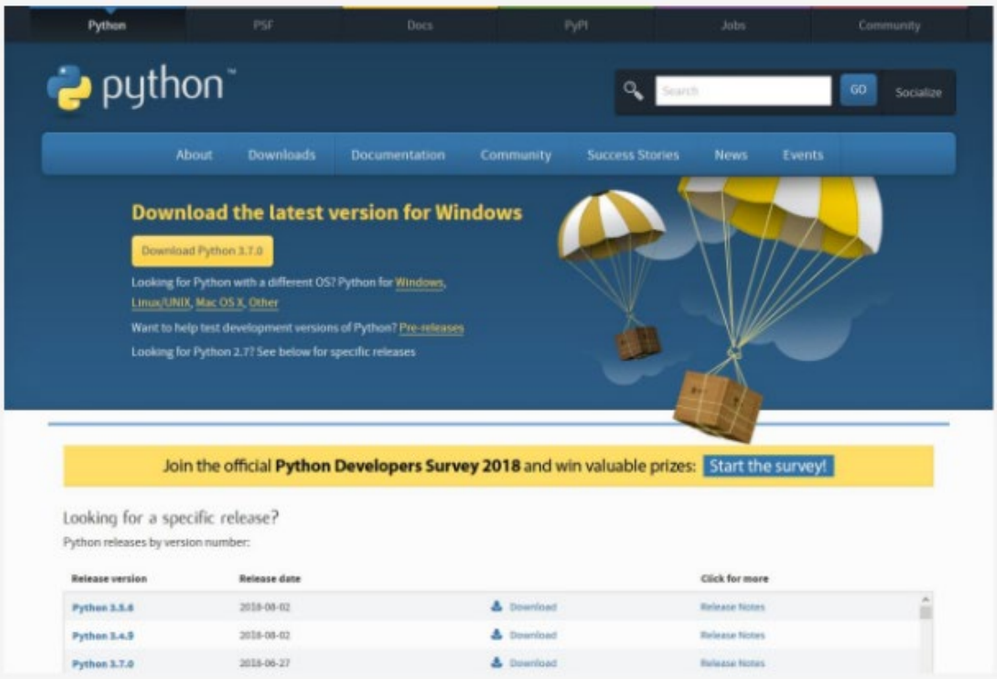
En este caso, selecciona Python 3. El sitio siempre te ofrece la última versión.

Si eres un **usuario de Windows** , utiliza el archivo .exe descargado y sigue todos los pasos.

Deja las configuraciones predeterminadas que el instalador sugiere por ahora, con una excepción: mira la casilla de verificación denominada **Agregar Python 3.x a PATH** y selecciónala.

Esto hará las cosas más fáciles.

Si eres un usuario de **macOS** , es posible que ya se haya preinstalado una versión de Python 2 en tu computadora, pero como estaremos trabajando con Python 3, aún deberás descargar e instalar el archivo .pkg correspondiente desde el sitio de Python.



Comenzando tu trabajo con Python

Ahora que tienes Python 3 instalado, es hora de verificar si funciona y de hacer el primer uso.

Este será un procedimiento muy simple, pero debería ser suficiente para convencerte de que el entorno de Python es completo y funcional.

Hay muchas formas de utilizar Python, especialmente si vas a ser un desarrollador de Python.

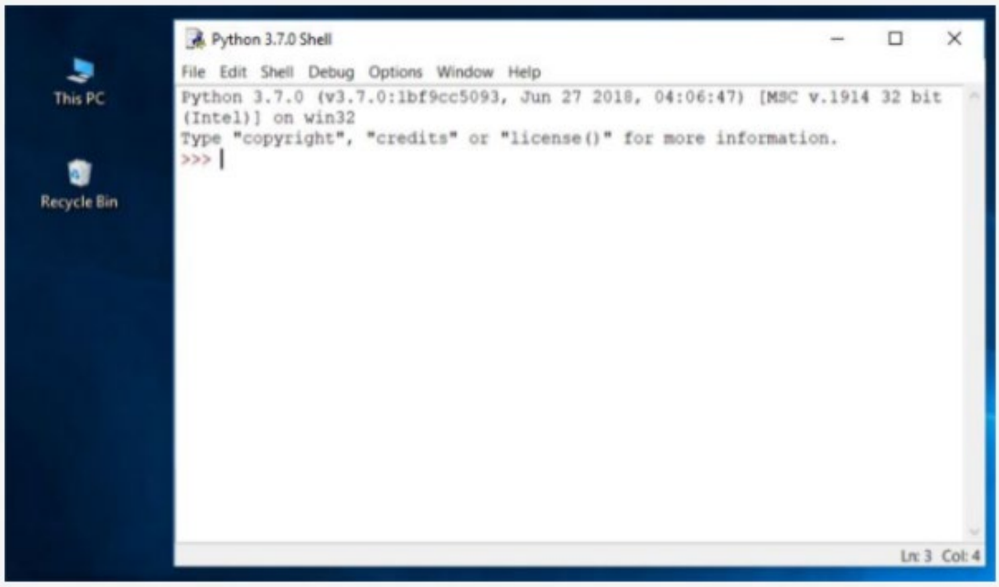
Para comenzar tu trabajo, necesitas las siguientes herramientas:

- Un **editor** que te ayudará a escribir el código (debes tener algunas características especiales, no disponibles en herramientas simples); este editor dedicado te dará más que el equipo estándar del sistema operativo.
- Una **consola** en la que puedes iniciar tu código recién escrito y detenerlo por la fuerza cuando se sale de control.
- Una herramienta llamada **depurador**, capaz de ejecutar tu código paso a paso y te permite inspeccionarlo en cada momento de su ejecución.

Además de sus muchos componentes útiles, la instalación estándar de Python 3 contiene una aplicación muy simple pero extremadamente útil llamada IDLE.

IDLE es un acrónimo de: Integrated Development and Learning Environment (Desarrollo Integrado y Entorno de Aprendizaje).

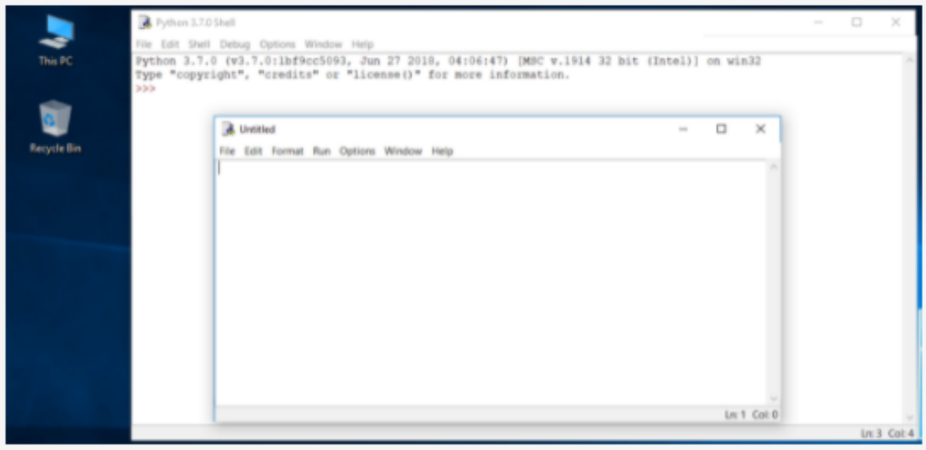
Navega por los menús de tu sistema operativo, encuentra IDLE en algún lugar debajo de Python 3.x y ejecútalo. Esto es lo que deberías ver:



¿Cómo escribir y ejecutar tu primer programa?

Ahora es el momento de escribir y ejecutar tu primer programa en Python 3. Por ahora, será muy simple.

El primer paso es crear un nuevo archivo fuente y llenarlo con el código. Haz clic en *File* en el menú del IDLE y elige *New File*.

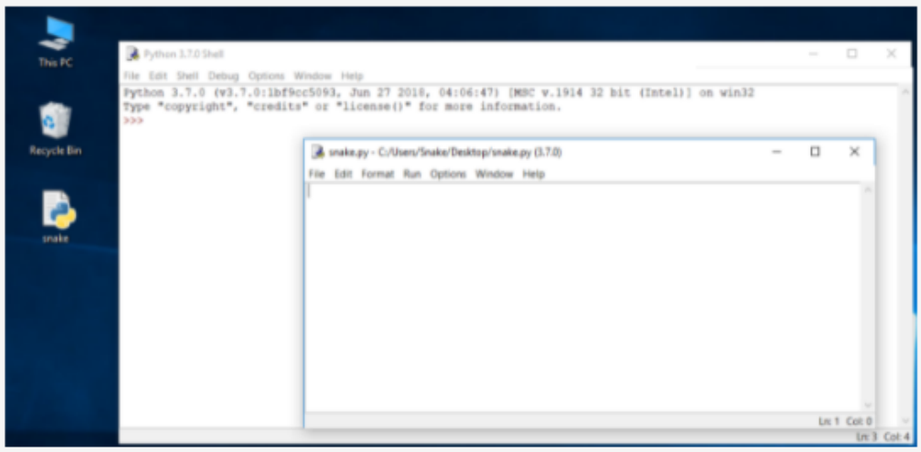


Como puedes ver, IDLE abre una nueva ventana para ti. Puedes usarla para escribir y modificar tu código.

Esta es la **ventana del editor**. Su único propósito es ser un lugar de trabajo en el que se trate tu código fuente. No confundas la ventana del editor con la ventana de shell. Realizan diferentes funciones.

La ventana del editor actualmente no tiene título, pero es una buena práctica comenzar a trabajar nombrando el archivo de origen.

Haz clic en *File* (en la nueva ventana), luego haz clic en *Save as ...* , selecciona una carpeta para el nuevo archivo (el escritorio es un buen lugar para tus primeros intentos de programación) y elige un nombre para el nuevo archivo.



Nota: no establezcas ninguna extensión para el nombre de archivo que vas a utilizar. Python necesita que sus archivos tengan la extensión *.py* , por lo que debes confiar en los valores predeterminados de la ventana de diálogo. El uso de la extensión *.py* estándar permite que el sistema operativo abra estos archivos correctamente.

¿Cómo escribir y ejecutar tu primer programa?

Ahora pon solo una línea en tu ventana de editor recién abierta y con nombre.

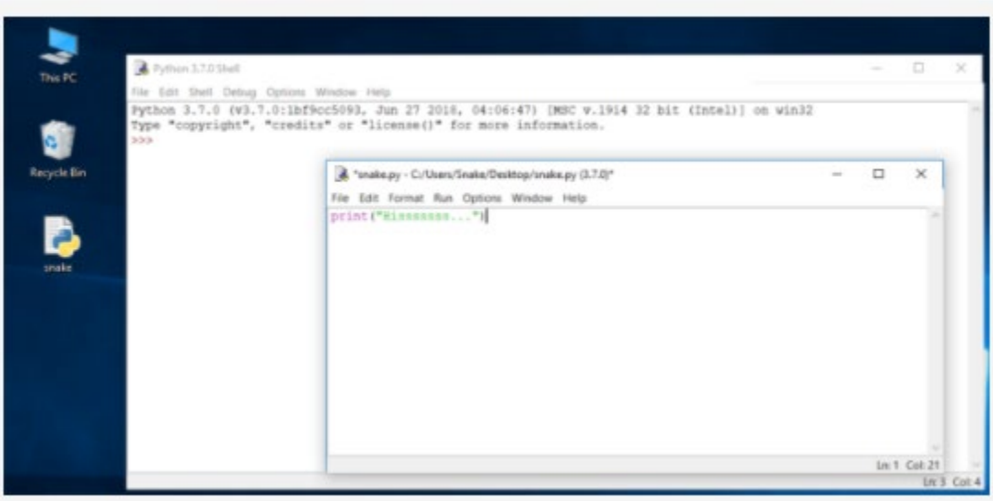
La línea se ve así:

```
print("Hisssssss...")
```

Puedes utilizar el portapapeles para copiar el texto en el archivo.

No vamos a explicar el significado del programa en este momento. Encontrarás una discusión detallada en el siguiente capítulo.

Echa un vistazo más de cerca a las comillas. Estas son la forma más simple de comillas (neutral, recta, etc.) que se usan comúnmente en los archivos de origen. No intentes utilizar citas tipográficas (curvadas, rizadas, etc.), utilizadas por los procesadores de texto avanzados, ya que Python no las acepta.

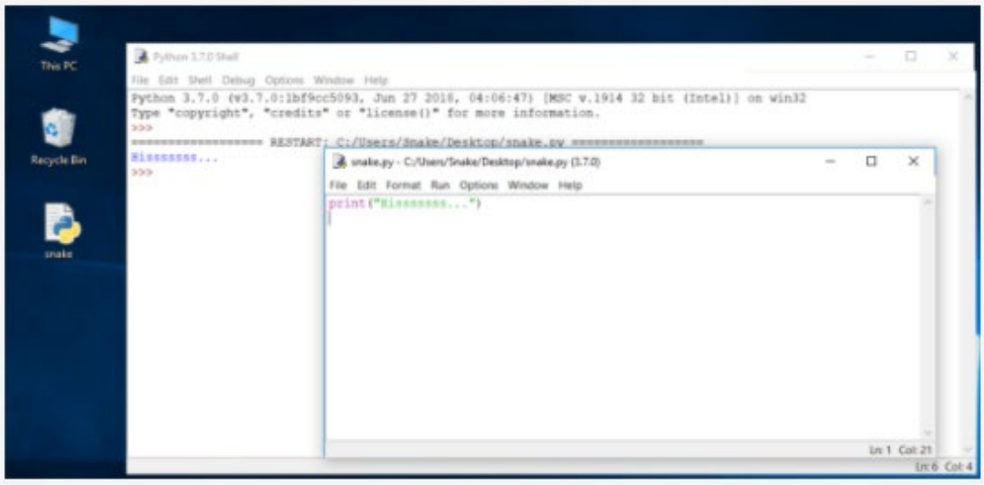


Si todo va bien y no hay errores en el código, la ventana de la consola mostrará los efectos causados por la ejecución del programa.

En este caso, el programa se ejecutara de manera correcta.

Intenta ejecutarlo una vez más. Y una vez más.

Ahora cierra ambas ventanas ahora y vuelve al escritorio.



¿Cómo estropear y arreglar tu código?

Ahora ejecuta IDLE otra vez.

Haz clic en *File* , *Open* , señala el archivo que guardaste anteriormente y deja que IDLE lo lea.

Intenta ejecutarlo de nuevo presionando *F5* cuando la ventana del editor está activa.

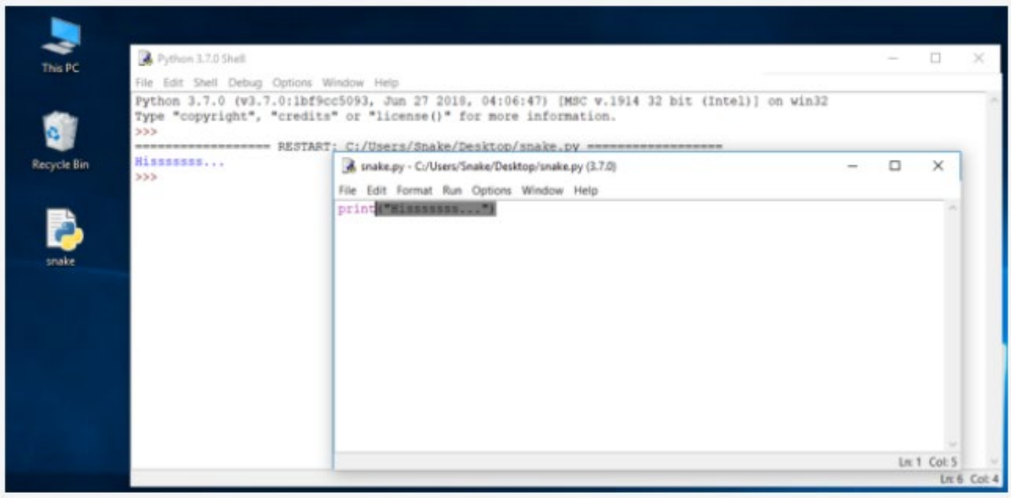
Como puedes ver, IDLE puede guardar tu código y recuperarlo cuando lo necesites de nuevo.

IDLE contiene una característica adicional y útil.

Primero, quita el paréntesis de cierre.

Luego ingresa el paréntesis nuevamente.

Tu código debería parecerse al siguiente:



Cada vez que coloques el paréntesis de cierre en tu programa, IDLE mostrará la parte del texto limitada con un par de paréntesis correspondientes. Esto te ayuda a recordar **colocarlos en pares**.

Retira nuevamente el paréntesis de cierre. El código se vuelve erróneo. Ahora contiene un error de sintaxis. IDLE no debería dejar que lo ejecutes.

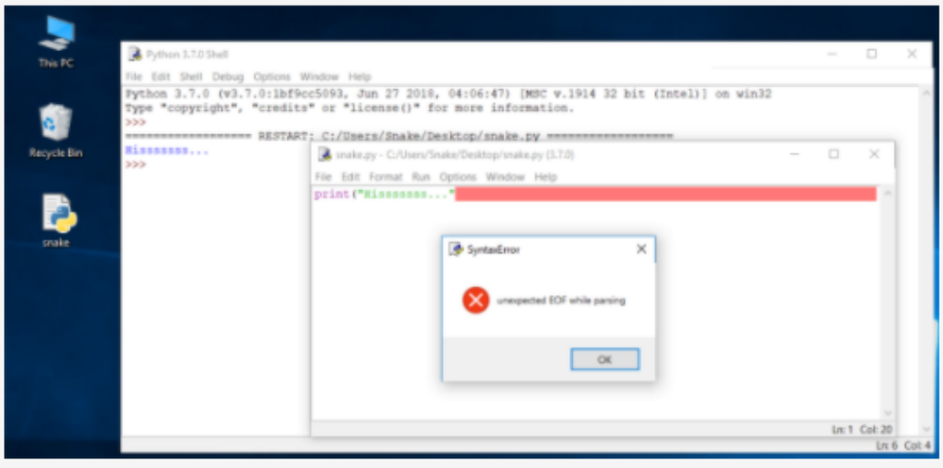
Intenta ejecutar el programa de nuevo. IDLE te recordará que guardes el archivo modificado. Sigue las instrucciones.

¿Cómo estropear y arreglar tu código?

Mira todas las ventanas con cuidado.

Aparece una nueva ventana: dice que el intérprete ha encontrado un EOF (*fin de archivo*).

La ventana del editor muestra claramente donde ocurrió.

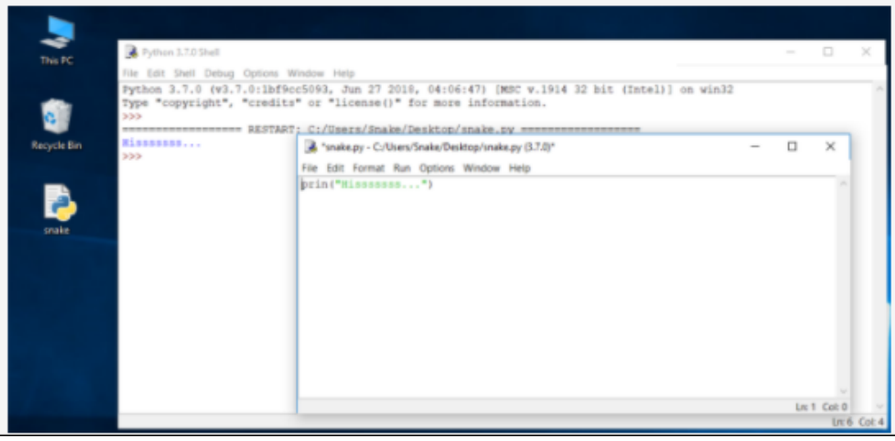


Arregla el código ahora. Debe verse así:

```
print("Hisssssss...")
```

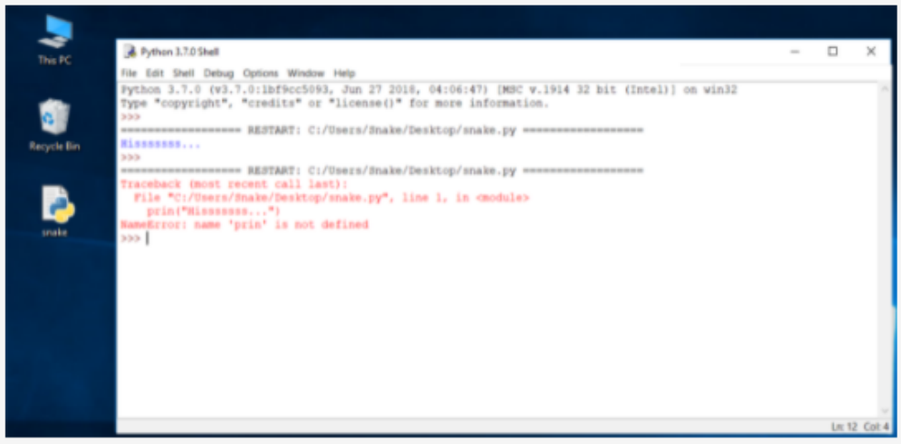
Ejecútalo para ver si "sisea" de nuevo.

Vamos a arruinar el código una vez más. Elimina una letra de la palabra `print`. Ejecuta el código presionando *F5*. Como puedes ver, Python no puede reconocer el error.



¿Cómo estropear y arreglar tu código?

Es posible que hayas notado que el mensaje de error generado para el error anterior es bastante diferente del primero.



Esto se debe a que la naturaleza del error es **diferente** y el error se descubre en una **etapa diferente** de la interpretación.

La ventana del editor no proporcionará ninguna información útil sobre el error, pero es posible que las ventanas de la consola si.

El mensaje (en rojo) muestra (en las siguientes líneas):

- El **rastreo** (que es la ruta que el código atraviesa a través de diferentes partes del programa, puedes ignorarlo por ahora, ya que está vacío en un código tan simple).
- La **ubicación del error** (el nombre del archivo que contiene el error, el número de línea y el nombre del módulo); nota: el número puede ser engañoso, ya que Python generalmente muestra el lugar donde se da cuenta por primera vez de los efectos del error, no necesariamente del error en sí.
- El **contenido de la línea errónea**: nota: la ventana del editor de IDLE no muestra números de línea, pero muestra la ubicación actual del cursor en la esquina inferior derecha; utilízalo para ubicar la línea errónea en un código fuente largo.
- El **nombre del error** y una breve explicación.

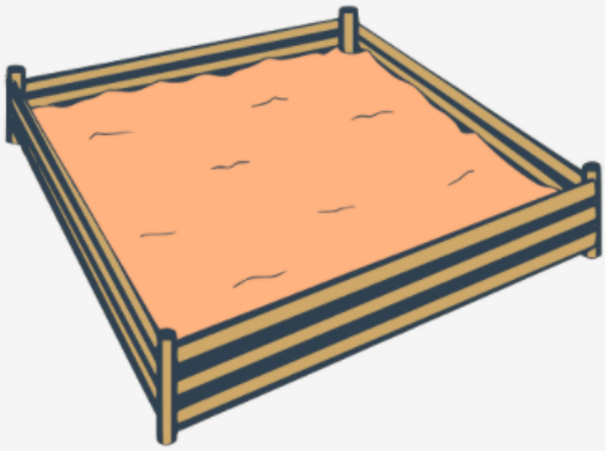
Experimenta creando nuevos archivos y ejecutando tu código. Intenta enviar un mensaje diferente a la pantalla, por ejemplo,¡rawr!, miau, o incluso tal vez un ¡oink! Intenta estropear y arreglar tu código, observa que sucede.

Sandbox

Este curso no requiere que instales ninguna aplicación de software para probar tu código y hacer los ejercicios.

Para probar o experimentar con tu código, puedes utilizar un entorno de programación en línea interactivo y dedicado.

Sandbox permite que el código Python se ejecute en un navegador de Internet.



Es una herramienta integrada dentro del curso, que se puede usar como un **Sandbox de Python basado en el navegador** que te permite probar el código discutido a lo largo del curso, así como **un intérprete que te permite iniciar, realizar y probar los ejercicios de laboratorio** diseñados específicamente para este curso.

La interfaz de Sandbox consta de tres partes principales:

- La ventana del **editor** que te permite escribir tu código.
- La ventana de **consola** que te permite ver el resultado de tus programas.
- Una herramienta llamada barra de **botones de acción** que te permite ejecutar tu código, actualizar la ventana del editor, descargar tu programa como un archivo .py, cargar un archivo .py que se mostrará en el editor, informar algún error (en caso de que detectes uno, ¡háznoslo saber!).
- El botón de **Configuración** que te permite ajustar la configuración de la pantalla y cambiar entre los entornos Python / C / C ++.

Ahora copia el siguiente código:

```
print("Hola!")
print("¡Bienvenido a Fundamentos de Programación en Python!")
print("ESTO ES EL MODO SANDBOX.")
```

... luego da clic en el botón **Sandbox** para ingresar al Modo Sandbox, pega el código en la ventana del editor y haz clic en el botón Ejecutar para ver que sucede.

Para volver a nuestro curso, haz clic en **Back to course** en la esquina superior derecha de la interfaz de Sandbox.

Interfaz de práctica

Este curso contiene cuatro tipos diferentes de interfaces.

Hasta ahora, haz visto la **Interfaz de estudio** (una o dos ventanas con texto e imágenes/animación) y la **Interfaz de Sandbox**, que puedes usar para probar tu propio código (haz clic en *Sandbox* para cambiar a la Interfaz de Sandbox).

Lo que ves ahora es la **Interfaz de práctica**, que te permite estudiar cosas nuevas y realizar tareas de codificación al mismo tiempo. Utilizarás este tipo de interfaz la mayor parte del tiempo durante el curso.

La Interfaz de práctica consiste en un área de texto a la izquierda y las ventanas del Editor/Consola a la derecha.

Otro tipo de interfaz que verás en el futuro es la Interfaz de prueba/examen, que te permitirá verificar tus conocimientos y habilidades para ver que tan bien has dominado el material de estudio.

¡Felicidades! Has completado el Módulo 1

¡Bien hecho! Has llegado al final del Módulo 1 y has completado una meta importante en tu educación de programación en Python. Aquí hay un breve resumen de los objetivos que has cubierto y con los que te has familiarizado en el Módulo 1:

- Los fundamentos de la programación de computadoras, es decir, como funciona la computadora, como se ejecuta el programa, como se define y construye el lenguaje de programación.
- La diferencia entre compilación e interpretación.
- La información básica sobre Python y cómo se posiciona entre otros lenguajes de programación, y qué distingue a sus diferentes versiones.
- Los recursos de estudio y los diferentes tipos de interfaces que utilizarás en el curso.

Ahora estás listo para tomar el cuestionario del módulo, que te ayudará a evaluar lo que has aprendido hasta ahora.