Introducción a Python: Características principales del lenguaje

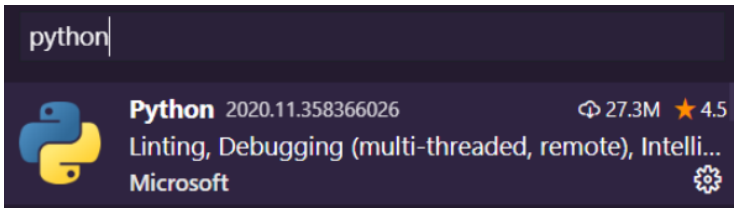
Python es un lenguaje de programación de alto nivel cuya máxima es la legibilidad del código. Las principales características de Python son las siguientes:

- Es multiparadigma, ya que soporta la programación imperativa, programación orientada a objetos y funcional.
- Es multiplataforma: Se puede encontrar un intérprete de Python para los principales sistemas operativos: Windows, Linux y Mac OS. Además, se puede reutilizar el mismo código en cada una de las plataformas.
- Es dinámicamente tipado: Es decir, el tipo de las variables se decide en tiempo de ejecución.
- Es fuertemente tipado: No se puede usar una variable en un contexto fuera de su tipo. Si se quisiera, habría que hacer una conversión de tipos.
- Es interpretado: El código no se compila a lenguaje máquina.

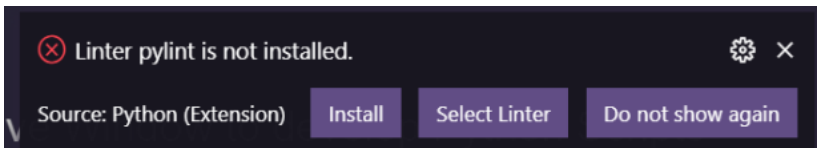
Instalación de Python y uso en VS Code

1) Descargar Python de https://www.python.org/downloads/ e instalar tildando la opción “Agregar Python al PATH”. Al finalizar tocar en “Disable path length limit”.

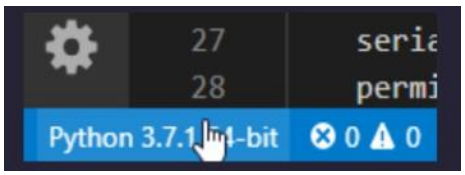
2) Descargar la extensión para VS Code.



3) Al crear el primer archivo .py es probable que muestre el siguiente mensaje para instalar pylint, un analizador de código cuyo fin es detectar errores antes de que sean interpretados y se recomienda instalar.



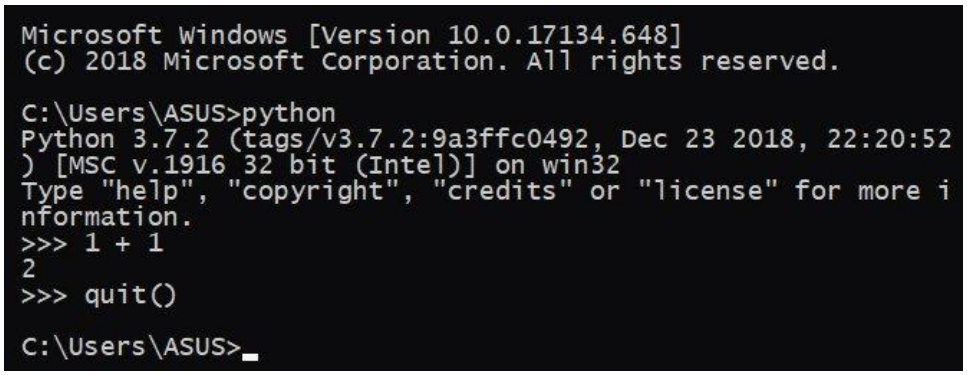
4) Si la versión de Python no fue detectada por visual code, seleccionar el intérprete en la barra de estado.



Ejecutar Python en modo Inmediato sin ningún IDE

Una vez que se instala Python, escribir Python en la línea de comando invocará al intérprete en modo inmediato. Podemos escribir directamente el código de Python y presionar Entrar para obtener el resultado.

Intente escribir 1 + 1 y presione enter. Obtenemos 2 como salida. Este indicador se puede utilizar como una calculadora. Para salir de este modo, escriba quit()y presione enter.



Palabras clave de Python

Las palabras clave son las palabras reservadas en Python.

No podemos usar una palabra clave como nombre de variable , nombre de función o cualquier otro identificador. Se utilizan para definir la sintaxis y la estructura del lenguaje Python.

En Python, las palabras clave distinguen entre mayúsculas y minúsculas.

Hay 33 palabras clave en Python 3.7. Este número puede variar ligeramente a lo largo del tiempo.

Todas las palabras clave excepto True, False y None están en minúsculas y deben escribirse tal cual. La lista de todas las palabras clave se proporciona a continuación.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Tratar de averiguar qué significa cada palabra clave puede ser abrumador. Si desea tener una visión general puede dirigirse al siguiente sitio donde verá una lista completa de palabras claves con ejemplos: https://www.programiz.com/python-programming/keyword-list

Identificadores de Python

Un identificador es un nombre dado a entidades como clase, funciones, variables, etc. Ayuda a diferenciar una entidad de otra. Todo el manejo de nombres y estilos en Python se basa en la guía de estilos PEP8 (https://peps.python.org/pep-0008/)

Reglas para escribir identificadores

1. Los identificadores pueden ser una combinación de letras en minúsculas **(a a z)** o mayúsculas **(A a Z)** o dígitos **(0 a 9)** o un guión bajo _. Nombres como `myClass`, `var_1` y `print_this_to_screen`, todos son ejemplos válidos.
2. Un identificador no puede comenzar con un dígito. `1variable` no es válido, pero `variable1` es un nombre válido.
3. Las palabras clave no se pueden utilizar como identificadores.

```
global = 1
```

Resultado

```
  Archivo "<entrada interactiva>", línea 1
    globales = 1
            ^
Error de sintaxis: sintaxis invalida
```

4. No podemos usar símbolos especiales como **!** , **@** , **#** , **$** , **%** etc. en nuestro identificador.
Resultado

```
a@ = 0
```

5. Un identificador puede tener cualquier longitud.

```
  Archivo "<entrada interactiva>", línea 1
    a@ = 0
     ^
Error de sintaxis: sintaxis invalida
```

Importante recordar

Python es un lenguaje que distingue entre mayúsculas y minúsculas. Esto significa, `Variable` y `variable` no son lo mismo. Siempre asigne a los identificadores un nombre que tenga sentido. Si bien `c = 10` es un nombre válido, escribir `count = 10` tendría más sentido y sería más fácil descubrir qué representa cuando observa su código después de un largo intervalo. Se pueden separar varias palabras con un guion bajo, como `this_is_a_long_variable`.

Declaraciones, identación y comentarios en Python

Las instrucciones que un intérprete de Python puede ejecutar se denominan declaraciones. Por ejemplo, `a = 1` es una instrucción de asignación. `If` declaración, `for` declaración, `while` declaración, etc. son otros tipos de declaraciones que se discutirán más adelante.

Identación

La mayoría de los lenguajes de programación como C, C++ y Java usan llaves{ } para definir un bloque de código. Python, sin embargo, usa identación.
Un bloque de código (cuerpo de una función , ciclo , etc.) comienza con identación y termina con la primera línea sin identación. La cantidad de identación depende de usted, pero debe ser consistente a lo largo de ese bloque.
En general, se utilizan cuatro espacios en blanco para la identación y se prefieren a las tabulaciones. Aquí hay un ejemplo.

```
for i in range(1,11):
    print(i)
    if i == 5:
        break
```

La aplicación de la identación en Python hace que el código se vea limpio y ordenado. Esto da como resultado programas de Python que se ven similares y consistentes.
La identación se puede ignorar en la continuación de línea, pero siempre es una buena idea identar. Hace que el código sea más legible. Por ejemplo:

```
if True:
    print('Hello')
    a = 5
```

Y

```
if True: print('Hello'); a = 5
```

ambos son válidos y hacen lo mismo, pero el estilo anterior es más claro.
Una identación incorrecta resultará en `IndentationError`.

Comentarios

Los comentarios son muy importantes al escribir un programa. Describen lo que sucede dentro de un programa, de modo que una persona que mira el código fuente no tenga dificultades para descifrarlo.
Es posible que olvide los detalles clave del programa que acaba de escribir en un mes. Así que tomarse el tiempo para explicar estos conceptos en forma de comentarios siempre es fructífero.
En Python, usamos el símbolo hash (**#**) para comenzar a escribir un comentario.
Se extiende hasta el carácter de nueva línea. Los comentarios son para que los programadores entiendan mejor un programa. Python Interpreter ignora los comentarios.

Docstrings

Los docstrings son un tipo de comentarios especiales que se usan para ***documentar*** un módulo, función, clase o método. En realidad, son la primera sentencia de cada uno de ellos y se encierran entre tres comillas simples o dobles.

Los docstrings son utilizados para generar la documentación de un programa. Además, suelen utilizarlos los entornos de desarrollo para mostrar la documentación al programador de forma fácil e intuitiva.

```
def double(num):
    """Función que duplica el valor"""
    return 2*num
```

Variables

Una variable es una ubicación con nombre utilizada para almacenar datos en la memoria. Es útil pensar en las variables como un contenedor que contiene datos que se pueden cambiar más adelante en el programa. Por ejemplo

```
number = 10
```

Aquí, hemos creado una variable llamada **número** y hemos asignado el valor 10 a la variable.

Python no tiene ningún comando para declarar una variable. Una variable se crea en el momento en que le asigna un valor por primera vez.

Se puede pensar en las variables como una bolsa para guardar libros y ese libro se puede reemplazar en cualquier momento (por un libro o cualquier cosa que pueda meter en la bolsa).

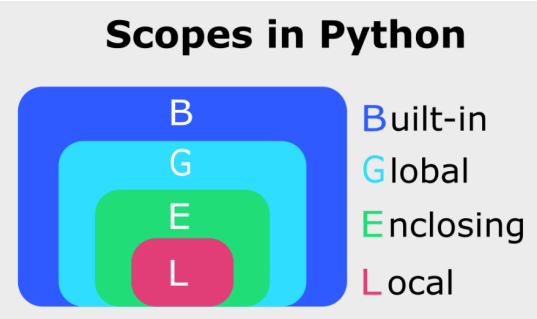
```
number = 10
number = 1.1
```

Inicialmente, el valor de número estaba 10. Más tarde, se cambió a 1.1.

No es necesario declarar las variables con ningún *tipo* en particular, e incluso pueden cambiar de tipo después de que se hayan establecido.

Puede obtener el tipo de datos de una variable con la función type().

La variable va a tener un scope (ámbito) que dependerá de donde sea declarada. El scope de una variable, es desde donde es accesible la misma y se puede modificar su valor. Una de las técnicas para recordar los scopes de las variables es recordar el seudónimo LEGB, que hace referencia a los 4 tipos de Scope (Local, Enclosing, Global, Built-in).




```python
#Global Scope
x = 0
def funcion():
    #Enclosed scope
    x = 1
    def funcion_interna():
        #Local scope
        x= 2
        print(f"Local scope x={x}")

    funcion_interna()
    print(f"Enclosed scope x={x}")

funcion()
print(f"Global scope x={x}")
```

Para más información, sugerimos observar el sitio web: https://realpython.com/python-scope-legb-rule/

Tipos de Datos

Las variables pueden almacenar datos de diferentes tipos, y diferentes tipos pueden hacer cosas diferentes.

Python tiene los siguientes tipos de datos integrados de forma predeterminada, en estas categorías:

Tipo de texto:	`str`
Tipos numéricos:	`int`, `float`, `complex`
Tipos de secuencia:	`list`, `tuple`, `range`
Tipo de mapeo:	`dict`
Establecer tipos:	`set`, `frozenset`
Tipo booleano:	`bool`
Tipos binarios:	`bytes`, `bytearray`, `memoryview`
Ninguno Tipo:	`NoneType`

Para repasar tipos de datos, strings, números, casting y demás, se recomienda seguir los tutoriales de w3schools (https://www.w3schools.com/python/python_datatypes.asp), o repasar los contenidos de FullStack donde se encuentran explicados en detalle.

Condicionales

Python admite las condiciones lógicas habituales de las matemáticas:

- Es igual a : a == b
- No es igual a: a != b
- Menos que: a < b
- Menor o igual que: a <= b
- Mayor que: a > b
- Mayor o igual que: a >= b

Ejemplo	Tipo de Dato
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Estas condiciones se pueden usar de varias maneras, más comúnmente en "sentencias if" y bucles.
Una "sentencia if" se escribe utilizando la palabra clave if

La palabra clave elif es la forma de Python de decir "si las condiciones anteriores no fueron ciertas, intente esta condición".
La palabra clave else captura cualquier cosa que no esté capturada por las condiciones anteriores.

```
a = 33
b = 33
if b > a:
    print("b es mayor que a")
elif a == b:
    print("a y b son iguales")
else:
    print("a es mayor que b")
```

Para mas detalles sobre condicionales visite la página:
https://www.w3schools.com/python/python_conditions.asp

Ciclos en Python

Python tienen dos comandos de ciclos primitivos

- Ciclos inexactos (no se la cantidad de veces que lo voy a ejecutar): while
- Ciclos exactos (sé cuántas veces lo voy a ejecutar): for

Ciclo While

El ciclo while en Python se usa para iterar sobre un bloque de código siempre que la expresión de prueba (condición) sea verdadera.
Generalmente usamos este ciclo cuando no sabemos la cantidad de veces que iterar de antemano.

```
valor_ingresado = input("Presione S para salir, cualquier tecla para sumar uno al total. ")
total = 0
while valor_ingresado != "S":
  total += 1
  valor_ingresado = input("Presione S para salir, cualquier tecla para sumar uno al total. ")

print("El total ha sido: " + str(total))
```

Para más información, visite: https://www.w3schools.com/python/python_while_loops.asp

Ciclo For

El ciclo for en Python se usa para iterar sobre una secuencia (lista , tupla , cadena) u otros objetos iterables. La iteración sobre una secuencia se llama recorrido.

```
# Programa para encontrar la suma de todos los números en una lista

# Lista de números
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable para guardar la suma
sum = 0

# iteramos sobre la lista
for val in numbers:
    sum = sum + val

print("La suma es: ", sum)
```

Para mas información visite: https://www-programiz-com.translate.goog/python-programming/for-loop

La función Range

Podemos generar una secuencia de números usando range(). range(10) generará números del 0 al 9 (10 números).
También podemos definir el tamaño de inicio, parada y paso como range(start, stop,step_size). step_size por defecto es 1 si no se proporciona.
El range objeto es "perezoso" en cierto sentido porque no genera todos los números que "contiene" cuando lo creamos. Sin embargo, no es un iterador ya que admite operaciones in, len y __getitem__.
Esta función no almacena todos los valores en la memoria; sería ineficiente. Por lo tanto, recuerda el inicio, la parada, el tamaño del paso y genera el siguiente número sobre la marcha.
Para obligar a esta función a generar todos los elementos, podemos usar la función list().
El siguiente ejemplo aclarará esto:

```
print(range(10))

print(list(range(10)))

print(list(range(2, 8)))

print(list(range(2, 20, 3)))
```

Resultado

```
rango (0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[2, 3, 4, 5, 6, 7]
[2, 5, 8, 11, 14, 17]
```

Podemos usar la función range() en bucles for para iterar a través de una secuencia de números. Se puede combinar con la función len() para iterar a través de una secuencia usando la indexación. Aquí hay un ejemplo:

```
# Programa para iterar a través de una lista usando indexación
genero = ['pop', 'rock', 'jazz']

# itera sobre la lista usando el índice
for i in range(len(genero)):
    print("Me gusta el ", genero[i])
```

Funciones

Características:

- Una función es un bloque de código que solo se ejecuta cuando se le llama.
- Se le pude pasar datos, definidos como parámetros, a una función.
- Una función puede devolver datos como resultado.

Definición y argumentos

Cuando se define una función, se debe especificar el nombre de esta, y el nombre de los parámetros en caso de que reciba. Cuando se le pasa el valor de un parámetro a una función, este recibe el nombre de argumento.

- En Python, una función se define usando la palabra clave def
- Para llamar a una función, use el nombre de la función seguido de paréntesis
- La información se puede pasar a funciones como argumentos.
- Los argumentos se especifican después del nombre de la función, entre paréntesis. Puede agregar tantos argumentos como desee, simplemente sepárelos con una coma.

En los siguientes ejemplos se ven dos funciones. La primera no retorna ningún valor, y recibe un parámetro pNombre, imprimiendo por pantalla un saludo al nombre cuando se la invoque. La segunda función recibe dos parámetros, los suma y los devuelve, siendo el valor de dicha devolución guardado en una variable para luego ser impreso por pantalla:

```
def saludar(pNombre):
  print("Hola " + pNombre)

saludar("Alejandro")

def sumar(a, b):
    return a + b

resultado = sumar(1, 2)
print("El resultado de la suma es: " + resultado)
```

Observe con atención la importancia de la identación, y que al no especificar el tipo de dato que se recibe, dichas funciones pueden recibir cualquier dato produciendo un error en caso de recibir como argumento un tipo de dato que no pueda manejar.

Para más repaso sobre funciones puede visitar el siguiente sitio o repasar los contenidos vistos en FullStack: https://www.programiz.com/python-programming/function

Módulos en Python

Los módulos hacen referencia a un archivo que contiene sentencias y definiciones de Python.

Un archivo que contiene código de Python, por ejemplo: **example.py**, se denomina módulo y su nombre de módulo sería **example**.

Usamos módulos para dividir programas grandes en pequeños archivos manejables y organizados. Además, los módulos proporcionan la reutilización del código.

Podemos definir nuestras funciones más utilizadas en un módulo e importarlo, en lugar de copiar sus definiciones en diferentes programas.

Vamos a crear un módulo. Escriba lo siguiente y guárdelo como **example.py**.

```
# Modulo de python example
def suma(a, b):
    """Este programa suma dos números y devuelve el resultado"""
    resultado = a + b
    return resultado
```

Acá hemos definido una función suma, dentro un módulo llamado example. La función toma dos números y devuelve el resultado.

¿Como importar módulos?

Podemos importar las definiciones dentro de un módulo a otro módulo o al intérprete interactivo en Python.

Usamos la palabra clave **import** para hacer esto. Para importar nuestro módulo previamente definido **example**, escribimos lo siguiente en el indicador de Python.

```
>>> import example
```

Esto no importa los nombres de las funciones definidas en **example** directamente en la tabla de símbolos actual. Solo importa el nombre del módulo **example**.

Usando el nombre del módulo podemos acceder a la función usando el operador punto. Por ejemplo:

```
>>> example.suma(4,5.5)
9.5
```

Para seguir repasando información de módulos, revise el material de FullStack o profundice en https://www.programiz.com/python-programming/modules

Entornos Virtuales

Las aplicaciones en Python usualmente hacen uso de paquetes y módulos que no forman parte de la librería estándar. Las aplicaciones a veces necesitan una versión específica de una librería, debido a que dicha

aplicación requiere que un bug particular haya sido solucionado o bien la aplicación ha sido escrita usando una versión obsoleta de la interfaz de la librería.

Esto significa que tal vez no sea posible para una instalación de Python cumplir los requerimientos de todas las aplicaciones. Si la aplicación A necesita la versión 1.0 de un módulo particular y la aplicación B necesita la versión 2.0, entonces los requerimientos entran en conflicto e instalar la versión 1.0 o 2.0 dejará una de las aplicaciones sin funcionar.

La solución a este problema es crear un **entorno virtual**, un directorio que contiene una instalación de Python de una versión en particular, además de unos cuantos paquetes adicionales.

Diferentes aplicaciones pueden entonces usar entornos virtuales diferentes. Para resolver el ejemplo de requerimientos en conflicto citado anteriormente, la aplicación A puede tener su propio entorno virtual con la versión 1.0 instalada mientras que la aplicación B tiene otro entorno virtual con la versión 2.0. Si la aplicación B requiere que actualizar la librería a la versión 3.0, esto no afectará el entorno virtual de la aplicación A

Creando entornos virtuales

El script usado para crear y manejar entornos virtuales es pyvenv. pyvenv normalmente instalará la versión mas reciente de Python que tengas disponible; el script también es instalado con un número de versión, con lo que si tienes múltiples versiones de Python en tu sistema puedes seleccionar una versión de Python específica ejecutando `python3` o la versión que desees.
Para crear un entorno virtual, decide en que carpeta quieres crearlo y ejecuta el módulo `venv` como script con la ruta a la carpeta:

```
python3 -m venv tutorial-env
```

Esto creará el directorio `tutorial-env` si no existe, y también creará directorios dentro de él que contienen una copia del intérprete de Python y varios archivos de soporte.
Una ruta común para el directorio de un entorno virtual es `.venv`. Ese nombre mantiene el directorio típicamente escondido en la consola y fuera de vista mientras le da un nombre que explica cuál es el motivo de su existencia. También permite que no colisione con los ficheros de definición de variables de entorno `.env` que algunas herramientas soportan.
Una vez creado el entorno virtual, podrás activarlo.
En Windows, ejecuta:

```
tutorial-env\Scripts\activate.bat
```

En Unix o MacOS, ejecuta:

```
source tutorial-env/bin/activate
```

(Este script está escrito para la consola bash. Si usas las consolas **csh** or **fish**, hay scripts alternativos `activate.csh` y `activate.fish` que deberá usar en su lugar.)
Activar el entorno virtual cambiará el prompt de tu consola para mostrar que entorno virtual está usando, y modificará el entorno para que al ejecutar `python` sea con esa versión e instalación en particular. Por ejemplo:

```
$ source ~/envs/tutorial-env/bin/activate
(tutorial-env) $ python
Python 3.5.1 (default, May  6 2016, 10:59:36)
  ...
>>> import sys
>>> sys.path
['', '/usr/local/lib/python35.zip', ...,
'~/envs/tutorial-env/lib/python3.5/site-packages']
>>>
```

Librerias

Puede instalar, actualizar y eliminar paquetes usando un programa llamado pip. De forma predeterminada, pip instalará paquetes del índice de paquetes de Python, <https://pypi.org>. Puede navegar por el índice de paquetes de Python yendo a él en su navegador web.

pip tiene varios subcomandos: «install», «uninstall», «freeze», etc. (Consulte la guía https://docs.python.org/es/3/installing/index.html#installing-index para obtener la documentación completa de pip).

Para repasar estos temas, podes referirte al material de FullStack o al siguiente sitio web: https://docs.python.org/es/3/tutorial/venv.html#