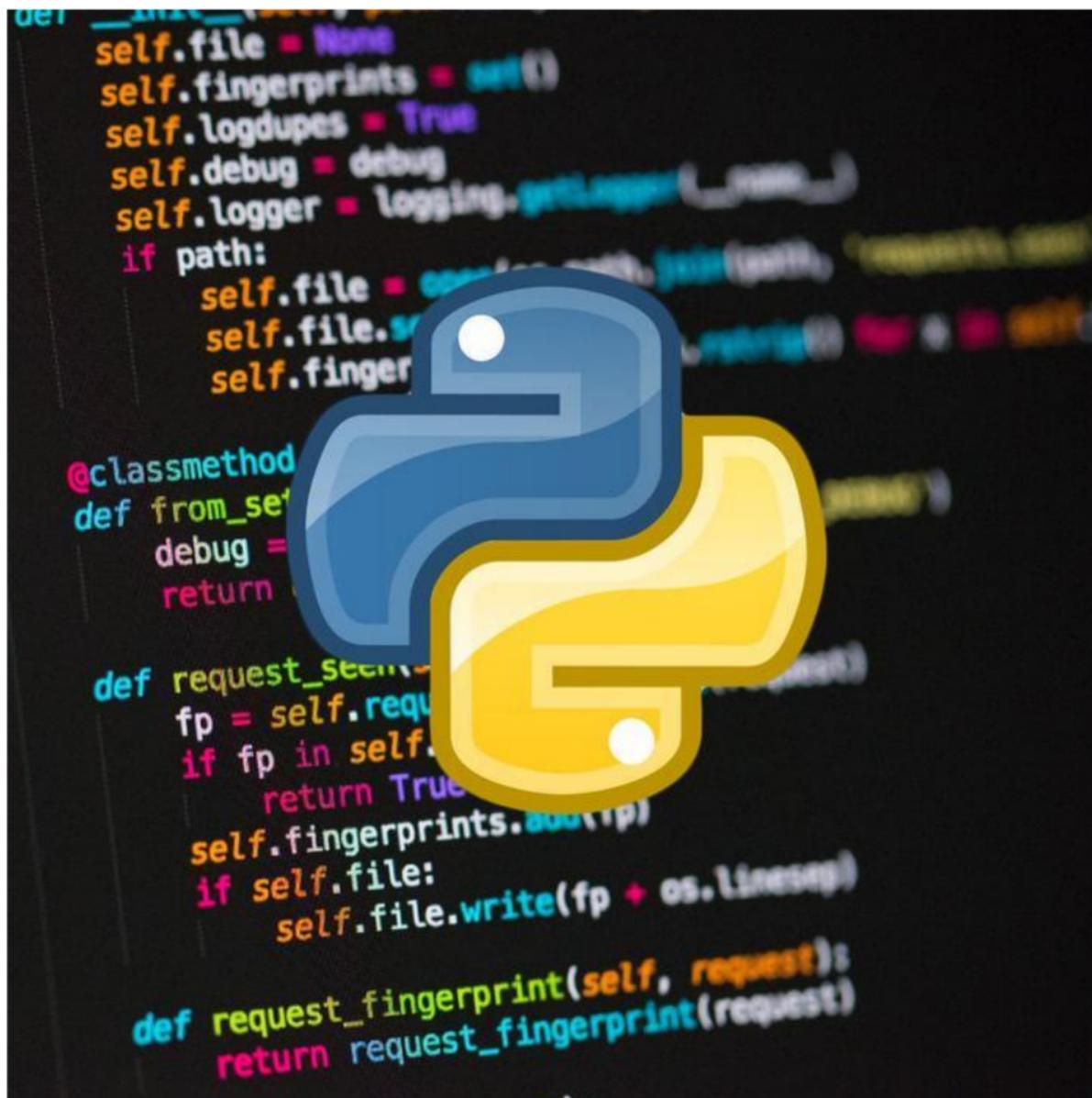


Python - inicios en la programación



Python, inicios en la programación

El siguiente material fue preparado para aquellas personas que quieran entrar en el mundo de la programación haciendo uso del lenguaje de programación Python.

El objetivo es que, al terminar todos los capítulos, usted pueda realizar distintos algoritmos y programas que resuelvan los diferentes problemas que se les presente.

Este material fue creado para el curso de “Python, inicios en la programación” que dicta la Academia Cisco – Oracle Taborin.

Material realizado por: Magni, Guillermo Tomas

Versión: 3.7.1

Ultima modificación: 19/10/2019

Capítulo 1: fundamentos en la programación

Hoy en día resulta bastante complicado pensar en la actualidad en la que viven las personas sin uno de los inventos más importantes de la historia: las computadoras. Y es que no solo revolucionaron la vida moderna, si no que en gran medida se ha convertido también en el sostén de la misma. Las computadoras se utilizan para una gran cantidad de actividades: desde llevar a la contabilidad de una empresa a crear material didáctico para los más chicos o crear software que ayuden a los médicos a tomar decisiones sobre un paciente mediante el análisis de un conjunto de estudios. Así, si nos ponemos a analizar a fondo podremos encontrar más que seguro que por detrás de casi cualquier actividad hay una computadora cumpliendo una función, permitiéndole a las personas contar con herramientas que, bien utilizadas, abren un nuevo abanico de posibilidades casi para cualquier profesión.

El tema surge que para que una computadora sea de utilidad se deben desarrollar los programas que resuelvan las distintas necesidades de las personas, y es ahí cuando entra la *programación*.

¿Qué es la programación?

La programación es el proceso de tomar un *algoritmo* y codificarlo en un *lenguaje de programación* formando así un *programa* que pueda ser ejecutado por una computadora.

1.1) Algoritmos y programas

Ahora bien, si volvemos un poco para atrás, vera que acabamos de mencionar dos conceptos nuevos y muy importantes en el mundo de la

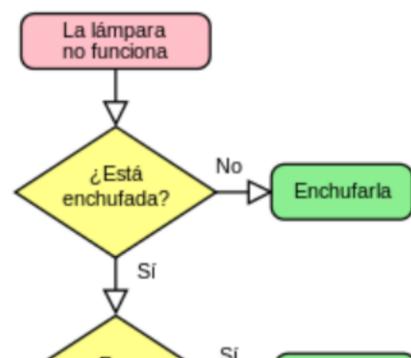
¿Qué es un algoritmo?

Un algoritmo es un conjunto finito de pasos que, ordenados de cierta manera, llevan a la resolución de un problema.

Si bien dijimos que los algoritmos son parte importante en el mundo de la computación, también lo son en nuestro día a día. Inconscientemente, las personas ejecutamos algoritmos para llevar adelante cientos de acciones. Por ejemplo: cambiar la rueda de un auto, preparar una pizza, sacar a pasear al perro, etc. Usted, al momento de preparar una pizza, no pone el queso y la salsa de tomate sobre la sartén y al final la masa. Lo que hace es:

- 1) Preparar la masa
 - a. Pone todos los ingredientes (harina, sal, aceite) dentro de una fuente y los mescla hasta formar la masa
- 2) Coloca la masa en una sartén y pone la salsa de tomate y el queso por arriba
- 3) Pone la pizza en el horno a cocinar

Y esto, como usted lo ve, es un algoritmo: una serie de pasos finitos que, ordenados de cierta manera, llevaron a la solución de su problema. Podríamos verlo más gráficamente, con otro ejemplo, de la siguiente manera:



1.2) Lenguajes de programación

Entonces, ya sabiendo lo que es un algoritmo, ya estamos en condiciones de decirle que una computadora es lo que se conoce como "*maquina algorítmica*", ya que su funcionalidad es, básicamente, ejecutar un algoritmo para obtener los resultados pedidos. El tema es que ahora todas las miradas apuntan hacia usted porque, como ya mencionamos anteriormente, todos los programas que corren en una computadora son creados por los programadores.

¿Cómo hace un programador para crear un algoritmo que la computadora pueda interpretar?

Usando lenguajes de programación. Un lenguaje de programación es un conjunto de símbolos, palabras y reglas para combinar esos símbolos y palabras de formas muy precisas y detalladas. Los lenguajes de programación cuentan con "palabras reservadas" que el lenguaje utiliza para funciones específicas y "operadores" que son un conjunto de símbolos para operaciones generalmente matemáticas y lógicas. Pero no cualquier combinación de esos operadores y palabras reservadas resulta en una orden válida para el lenguaje, de la misma forma que no cualquier combinación de palabras castellanas resulta en una frase comprensible en español. Los operadores y palabras reservadas deben combinarse siguiendo reglas específicas y muy puntuales. El conjunto de esas reglas se conoce como la *sintaxis* del lenguaje. Por ejemplo, cuando nosotros hacemos un calculo matemático, lo que hacemos es colocar el primer numero que vamos a utilizar en la operación seguido por el operador matemático y terminando con el segundo numero que vamos a utilizar. Esto en Python se vería de la siguiente manera:

```
|>>> 2 + 2
```

Fíjese que ahora cambiamos el orden de los elementos acomodándolo de la siguiente forma: + 2 2, a lo que el interprete de Python nos respondió con un error del tipo *SyntaxError*. El interprete nos esta diciendo que hay un error en la sintaxis, básicamente nos esta diciendo “la forma en la que me pediste que yo hiciera algo está mal escrita”.

¿Qué es un intérprete?

Recién hablábamos del *intérprete* de Python, un interprete es un programa que ejecuta línea a línea las instrucciones de un programa. Él es el encargado de cargar el código fuente y traducir las instrucciones a un lenguaje intermedio que pueda ser ejecutado luego.

¿Qué es el código fuente?

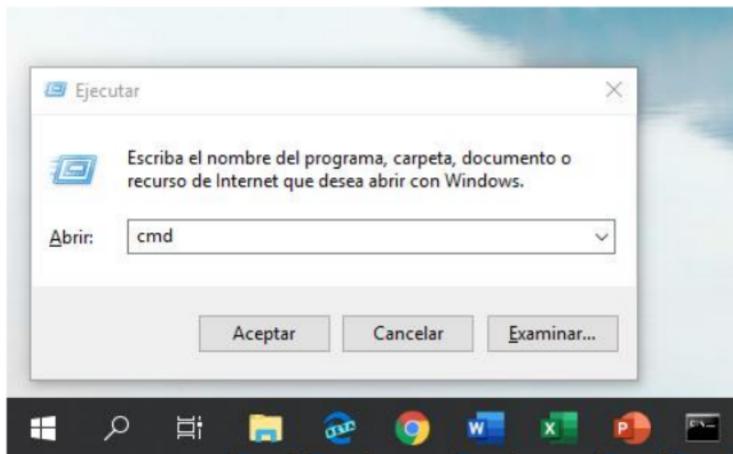
Se conoce como “código fuente” al código creado por un programador para resolver un algoritmo (esto que hará usted a lo largo del curso)

¿Cómo sería el esquema de trabajo en Python?



1.3) Uso del Shell Python

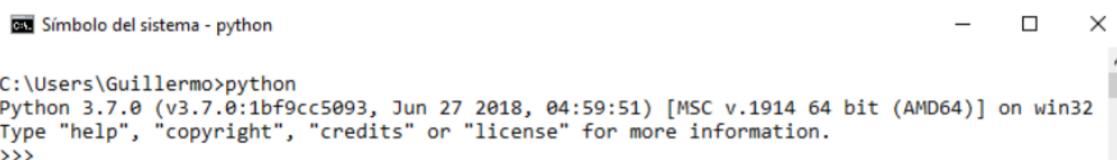
Una vez usted ya tenga instalado Python en su computadora (mostraremos como instalar Python y Visual Studio Code, que serán las herramientas a utilizar en este curso, en otro apunte. Puede seguir leyendo y luego pasar a ese apunte o parar la lectura para ir a realizar la instalación y así probar los ejemplos que veremos a continuación) podrá ejecutar el



Esto nos abrirá la ventana de comando de Windows:

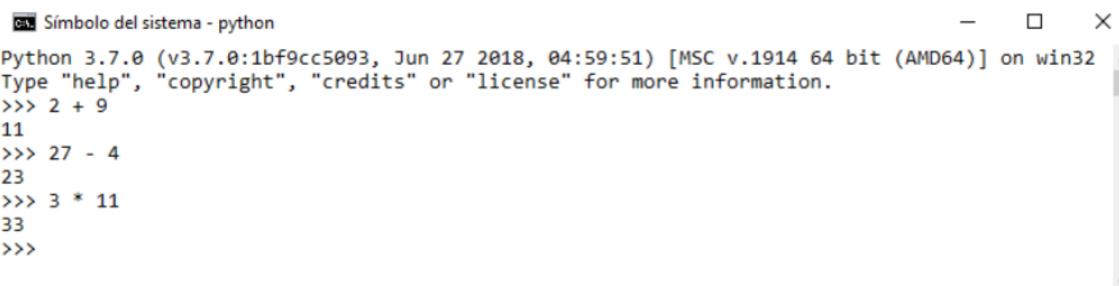


Si usted escribe el comando *python* y aprieta el enter, la ventana de comando dará inicio al interprete de Python, como puede ver a continuación:



es donde nosotros empezaremos a mandar instrucciones (que quede en claro que en estos momentos estamos usando el Shell solo para mostrar ejemplos, cuando llegue el momento del código trabajara casi en su totalidad en el IDE Visual Studio Code).

Habiendo pasado ya la parte más teórica del capítulo, empezamos con la programación. Python, como casi todos los lenguajes de programación, tiene una forma particular de trabajar la parte matemática. En lo que son las operaciones comunes como las sumas (+), restas (-) y multiplicación (*) no hay nada raro. Si usted ingresa las operaciones, Python las ejecutara sin problema:



```
Símbolo del sistema - python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> 2 + 9
11
>>> 27 - 4
23
>>> 3 * 11
33
>>>
```

Con la división ocurre un caso distinto, Python permite trabajar la división de tres formas:

1. La división común, para esta se usa el símbolo `/`, devolverá el valor exacto de la división. Por ejemplo: `3 / 2` nos devolverá como resultado `1.5` (en los lenguajes de programación, los números con coma (,) se hacen utilizando el punto (.))
2. La división solo con valores enteros, para esta se utiliza el símbolo `//`, devolverá el resultado, pero solo el valor entero, pasará por el alto los valores decimales. Por ejemplo: `3 // 2` nos devolverá como resultado `1`.

```
>>> 3 / 2  
1.5  
>>> 3 // 2  
1  
>>> 4 % 2  
0
```

Dependiendo el problema que tenga que resolver le puede convenir usar una forma o la otra. Por ejemplo, si quiere saber si un numero es par o impar, podría utilizar la división por resto para así ver cual es el resto de la división, si divide un numero por 2 y el resto da 0 es porque el numero es divisible por dos y, por ende, es par.

Las potencias por su parte se hacen utilizando el símbolo `**`. Así $2^{**} 3$ (2^3) nos devolverá como resultado 8:

```
>>> 2 ** 3  
8
```

Habiendo visto la parte de las matemáticas, pasamos a las *variables*. Las **variables** son espacios reservados en la memoria en las que se puede guardar información para utilizarla en el momento en que se la necesite y que, como su nombre indica, pueden cambiar de contenido a lo largo de la ejecución de un programa. En Python la forma de crear variables es la siguiente:

```
>>> numero = 17
```

Donde:

- “**numero**”: es el nombre de la variable. Toda variable necesita un nombre que la identifique y diferencia del resto de variables que tenga el programa. Por nombre puede llevar el que uno quiera, siempre y cuando cumpla las siguientes reglas:

- Python es **case sensitive**: Python diferencia mayúsculas de minúsculas. Así, aunque en esencia, el nombre de dos variables sea iguales, si alguno de sus caracteres está en minúscula o mayúsculas, Python lo tomará como dos variables totalmente distintas. Por ejemplo:

```
>>> nombre = "Guillermo"
>>> Nombre = "Tomas"
>>> nombre
'Guillermo'
>>> Nombre
'Tomas'
```

Fíjese que lo único que cambio entre las dos variables 'nombre' y 'Nombre', fue la 'n' del principio. Con eso basta para que Python considere que son dos variables distintas, aunque el significado sea el mismo. Esto ocurre si modifica cualquier letra del nombre de la variable:

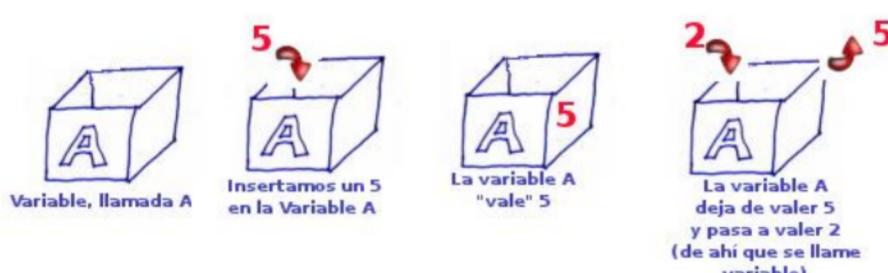
```
>>> aux = 1
>>> Aux = 2
>>> aAux = 3
>>> auX = 4
>>> aux
1
>>> Aux
2
>>> aAux
3
>>> auX
4
>>> -
```

- “**=**”: el **=** es lo que se conoce como el signo de asignación. Es la forma de decirle a Python: “guardarme en la variable x el valor y”
- “**17**”: es el valor que guardamos en la variable. Hay distintos tipos de datos que se pueden guardar en una variable (hablaremos de eso en otra ocasión). Es importante recordar que es diferente de otra cosa.

```
>>> y = 13  
>>> y  
13  
>>> y = "Hola Mundo!"  
>>> y  
'Hola Mundo!'  
>>> y = True  
>>> y  
True  
>>>
```

De este ejemplo podemos ver dos cosas:

1. Lo que le comentábamos recién, en una variable se puede guardar cualquier valor y después cambiar ese valor por otro que sea de un tipo de dato totalmente distinto.
2. Como puede observar, una variable solo puede contener un valor y solo uno. Por ende, si usted le da otro valor, el valor anterior se pierde.



Llegados acá, pasamos a explicar los tipos de datos que vio recién. En todo lenguaje de programación hay distintos tipos de datos que permiten llevar adelante los programas. Los que nosotros trabajaremos en este curso son los siguientes: bool (usados para comparaciones lógicas), int (recorte de

Tipo	Descripción	Byte por cada variable	Rango
bool	Valores lógicos	1	False, True
int	Valores enteros	Dinámico	Ilimitado
float	Valores reales	8	Hasta 15 decimales
str	Cadena de caracteres	2 * cantidad de caracteres	Unicode

Algunos ejemplos de los tipos de datos:

```
>>> a = 14
>>> b = 17.232
>>> c = "Esto es una cadena de caracteres"
>>> d = False
```

Sabiendo esto, usted puede usar las variables como se ve a continuación:

```
>>> a = 10
>>> b = a * 2
>>> b
20
>>> c = 107.11
>>> d = c - b
>>> print("El valor de d es:", d)
El valor de d es: 87.11
>>> 3 > 4
False
```

- **b = a * 2:** la variable **b**, va a guardar el resultado de multiplicar lo que tenga la variable **a** (10) por 2.
- **3 > 4:** acá podemos usar el sentido de los valores **bool**, cuando nosotros hablábamos de que se usan para comparaciones lógicas nos referíamos a esto. Sirven como una respuesta a una comparación que se le pida hacer a interprete de Python. Python

También tome nota en la siguiente línea:

```
>>> print("El valor de d es:", d)
El valor de d es: 87.11
```

Lo que usted ve ahí, **print()**, es una de las funciones mas utilizadas en Python. La función, como usted ya habrá entendido, sirve para enviar mensajes a la pantalla.

```
>>> print("Hola Mundo!")
Hola Mundo!
```

Todo lo que ponga entre “ ” se escribirá en la pantalla de su ventana de comando. Ahora bien, en el primer ejemplo, una vez terminado el mensaje lo que siguió fue: , **d**). La coma (,) permite “unir” valores a la cadena. De esta forma, podemos ver que el **print** unió “El valor de d es:” y el valor que estaba guardado en la variable **d**, que era **87.11**. Por eso el mensaje que devolvió el comando fue: *El valor de d es: 87.11*. Otros ejemplos:

```
>>>
>>> dolar = 56.64
>>> print("El valor del dolar es:", dolar)
El valor del dolar es: 56.64
>>>
```

```
>>> lampara = True
>>> print("¿Esta la lampara del patio prendida?", lampara)
¿Esta la lampara del patio prendida? True
>>> -
```

```
>>>
>>> numero_de_clase = 8
>>> print("¿Cuantas clases tiene el curso de Python?", numero_de_clase)
¿Cuantas clases tiene el curso de Python? 8
>>> -
```

Ingresar valores por teclado

Otra función totalmente útil en el mundo de la programación que es la que

```
py index.py > ...
1     nombre = input("Hola! Cual es su nombre?: ")
2     print("Encantado de conocerte,", nombre,"!")
```

Como podemos ver, el `input()` se relaciona con una variable. Esto es porque lo que sea que el usuario escriba en la ventada de comandos y le de enter debe ser guardado en algún lado. Entonces, si ejecutamos ese código:

```
PS C:\Users\Guillermo\Documents\curso python> .\index.py
Hola! Cual es su nombre?:
```

Podemos observar como el programa se para esperando a que el usuario ingrese algo. Nosotros vamos a ingresar el nombre “Guillermo” que se va a guardar en la variable **nombre**:

```
PS C:\Users\Guillermo\Documents\curso python> .\index.py
Hola! Cual es su nombre?: Guillermo
Encantado de conocerte, Guillermo !
```

Así, el nombre “Guillermo” es guardado en la variable `nombre`, que luego es utilizada por el `print()` para darnos un saludo.

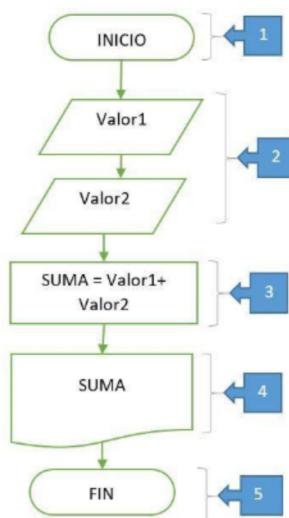
1.4) Primer programa

Llegados a este punto, hemos visto algunos conceptos y herramientas interesantes como para afrontar algunos problemas iniciales. Pensemos en un ejercicio simple que permita hacer uso de lo que hemos visto hasta ahora: crear un pequeño programa que reciba por teclado dos números, los sume y muestre el resultado al usuario. Pensemos el problema por partes:

1. La idea del ejercicio es crear un programa que, de respuesta al problema de querer sumar dos números, por ende, el **resultado** a

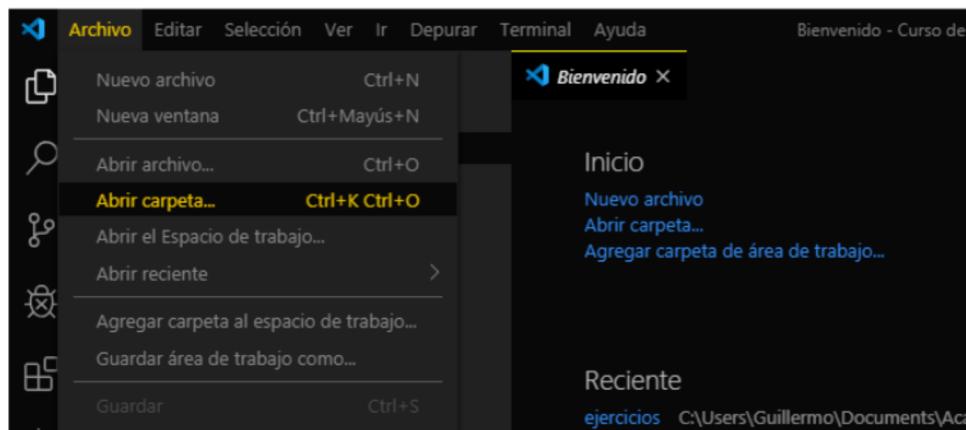
necesitaremos como mínimo de 2 variables (numero_1 y numero_2).

3. Ya llegados a este punto lo único que nos falta es definir el algoritmo para dar solución al problema:
 - a. El programa debería dar, antes que nada, un mensaje de bienvenida y avisar cual es su funcionalidad.
 - b. Una vez realizado el paso anterior, se deberían empezar a pedir los números a sumar, un numero a la vez.
 - c. Ya con los dos números ingresados, el programa debería realizar la suma. Para esto recomendamos crear una tercera variable llamada *resultado* que guarde el valor que devuelva la suma entre numero_1 y numero_2
 - d. Por último, queda mostrarle el usuario el resultado de la operación realizada. Gráficamente, el algoritmo se vería así:

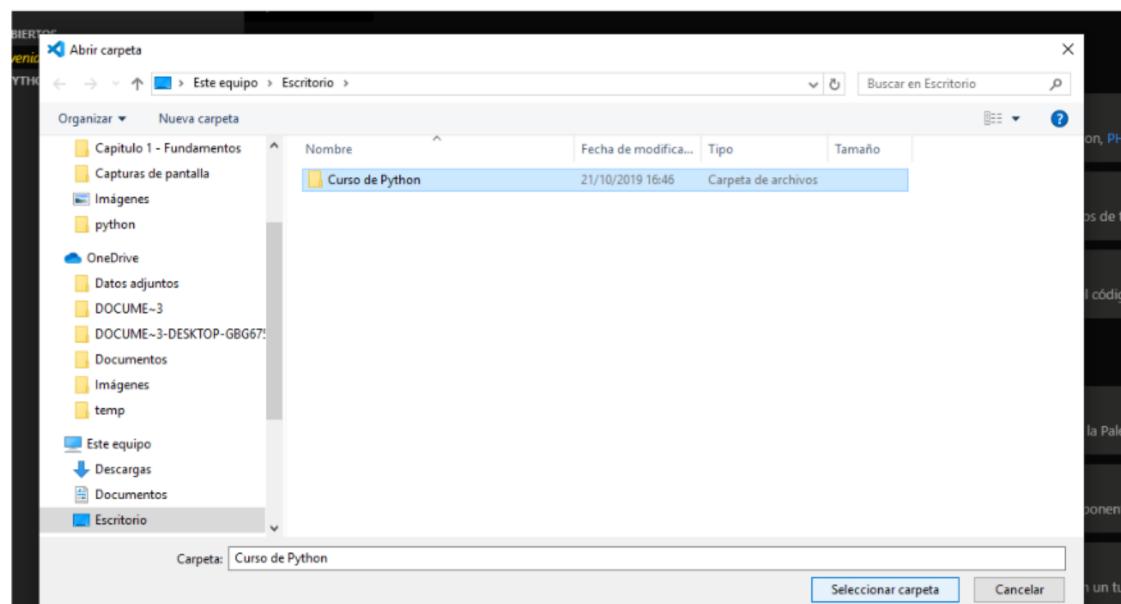


Habiendo terminado el análisis del problema y pensado el algoritmo para llegar a la solución del mismo, solos nos quedaría empezar a construir el código. Antes, vamos a ver como crear archivos en Visual Studio Code. Lo

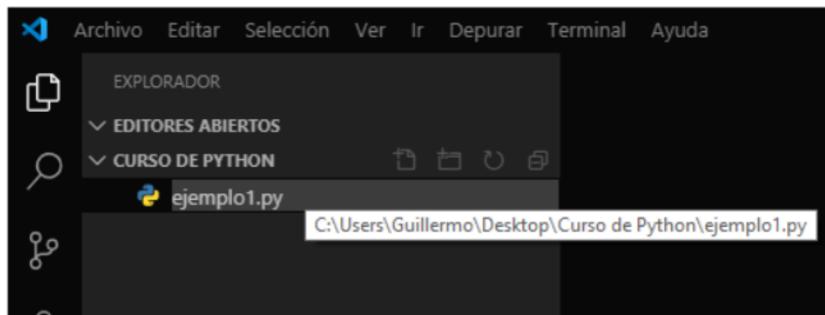
Luego debemos indicarle al Visual Studio que queremos que trabaje con los archivos de esa carpeta. Para eso vamos al menu **Archivos** y luego a la opción **Abrir carpeta...**



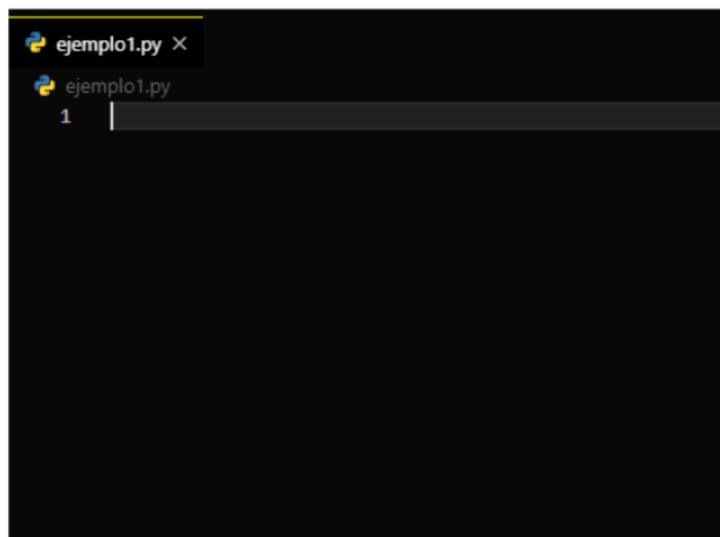
Y buscamos la carpeta que creamos anteriormente, en nuestro caso está en el Escritorio. Una vez la encontramos, la seleccionamos y damos al botón **Seleccionar carpeta**



De esta forma se crea un nuevo archivo y el programa nos pedirá que le demos un nombre. El nombre puede ser el que usted quiera, pero debe terminar con .py, de esta forma le indicamos al Visual Studio que el archivo tiene escrito código en Python (py). En nuestro caso le pusimos de nombre ejemplo1.py:



Esto le abrirá un editor de texto que le permitirá empezar a escribir el código:



Ahora si ya estamos en condiciones de ponernos a trabajar con el código:

1. Dar un mensaje de bienvenida:

- numeral, como podrá observar, y la única función de estos es ir describiendo el funcionamiento del programa. Los comentarios son ignorados por el intérprete, pasa a la siguiente línea apenas lee el #.
- II. Lo segundo que debería notar es el: `int()`. El `int()` es una función que ofrece Python para convertir los caracteres en números enteros. ¿Para qué usariamos una función `int()`? La función `input` permite guardar un valor que ingrese el usuario por teclado, pero el problema está en que esa función solo guarda una *cadena de caracteres (un str)*, es por eso que debemos convertir esa cadena en un numero para poder trabajar con él, y por eso utilizamos la función `int()`, para convertir la cadena en un número. Podemos verificar esto haciendo uso de otra función: la función `type()`. La función `type()` nos devuelve que tipo de dato es el valor que pongamos entre paréntesis. Veamos:

```
>>> type(5)
<class 'int'>
>>>
>>> type("Hola")
<class 'str'>
>>>
>>> type(True)
<class 'bool'>
>>>
```

¿Esto para que nos sirve? Veamos en el código que venimos escribiendo:

```
4  #Pedimos el primer numero a sumar
5  numero_1 = int(input("Ingrese el primer numero que quiera sumar:"))
6
7  #Controlamos que tipo de dato es el que se guardo en al variable numero_1
8  print(type(numero_1))
```

La línea 8 imprime en pantalla lo que devuelva el método `type(numero_1)`. Veamos:



```
4 #Pedimos el primer numero a sumar
5 numero_1 = input("Ingrese el primer numero que quiera sumar:")
6
7 #Controlamos que tipo de dato es el que se guardo en al variable numero_1
8 print(type(numero_1))
```

Ejecutamos el programa de nuevo:

```
C:\Users\Guillermo\Documents\curso python>suma.py
Bienvenido! Este es un pequeño programa para realizar una suma entre dos numeros.
Ingrese el primer numero que quiera sumar:6
<class 'str'>
```

Fíjese que ahora el valor que se guarda en la variable *numero_1* ya no es del tipo *int*, si no del tipo *str*. Esto ocurre porque el método *input()* esta programado para que cualquier tipo de dato que se le envíe por teclado lo convierta en una cadena de caracteres, luego Python ofrece los métodos para convertirlos de cadenas de caracteres a los datos que queramos. Para los valores con coma existe el método *float()*, como podemos ver en el siguiente ejemplo:

```
>>> peso = float(input("Ingrese su peso:"))
Ingrese su peso:15.6
>>> peso
15.6
>>> type(peso)
<class 'float'>
>>>
```

Ya sabiendo esto, damos marcha atrás y continuamos con el programa: pedimos que ingrese el segundo valor a sumar y lo guardamos en la variable *numero_2*, hacemos la suma y guardamos el resultado en la variable *resultado* y terminamos mostrar el resultado de la suma al usuario:

Ejecutamos el programa:

```
C:\ Símbolo del sistema
C:\Users\Guillermo\Documents\curso python>suma.py
Bienvenido! Este es un pequeño programa para realizar una suma entre dos numeros.
Ingrese el primer numero que quiera sumar:6
Ingrese el segundo numero que quiera sumar:13
El resultado de la suma es: 19
```

Acá podemos ver nuestro programa terminado y en funcionamiento.

Veamos algunos ejemplos más:

Ejercicio 2: cree un programa que calcule el IVA que tendrá un producto. Muestre en pantalla cuanto es el precio del IVA y cual seria el costo final del producto.

La resolución del ejercicio seria la siguiente:

```
#Guardamos en una variable el porcentaje del iva
iva = 0.21
#Guardamos en una variable el precio del producto
precioProducto = 716
#Calculamos cuento seria el monto extra del precio del producto por el iva
precioIVA = precioProducto * iva
#Mostramos en pantalla el monto extra por el iva
print("El precio del IVA es $",precioIVA)
#Mostramos en pantalla el coste final del producto
print("El precio final es $",(precioIVA + precioProducto))
```

Fíjese como decidimos hacer la suma para saber el costo final del producto directamente dentro del mensaje de la última línea, lo único que debemos hacer para que el programa nos permita llevar adelante esta forma de trabajo es encerrar la operación dentro de paréntesis (), para que este sepa que debe resolverse una operación. El resultado del código de arriba es el siguiente: