



Hechos

QUE

CONECTAN



El futuro digital
es de todos

MinTIC

FUNDAMENTOS DE PROGRAMACIÓN EN PYTHON

Universidad
Industrial de
Santander



‘Misión
TIC 2022’

1.1. Instalación de Python

Windows:

Ver video anexo.

Instalación de Python en Windows

Ubuntu:

Abrir la terminal (ctrl+alt+t) y ejecutar: `sudo apt-get install python3`

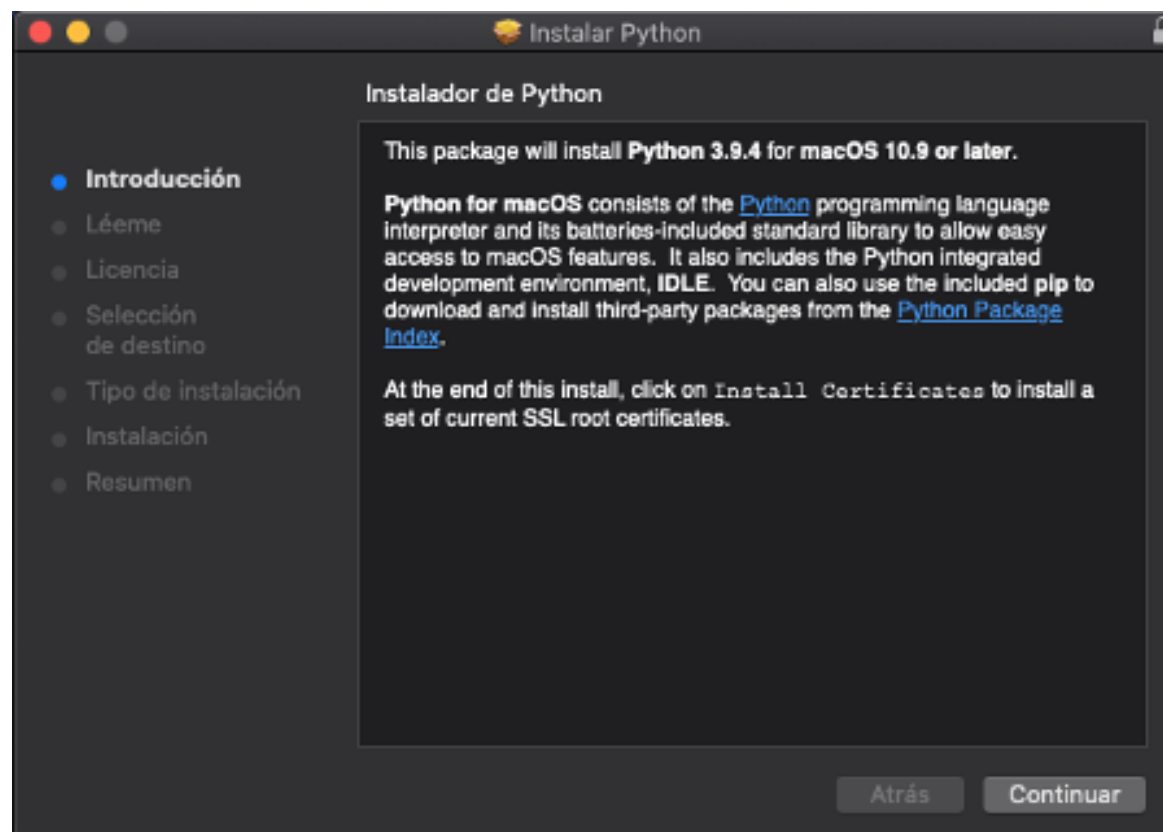
1. Podrá ver la versión de Python3 instalada ejecutando: `python3 -version`

MAC:

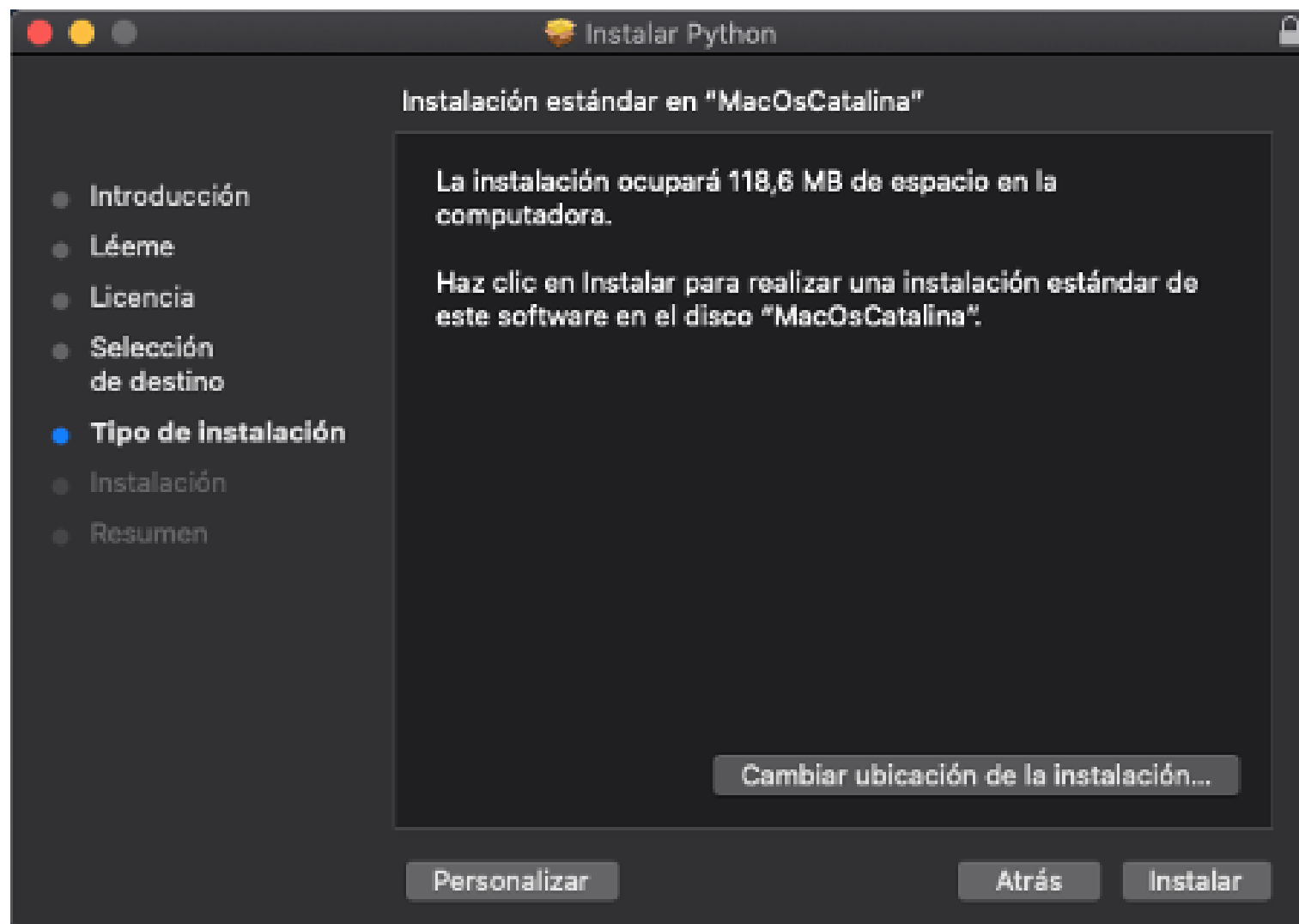
Desde su sitio oficial: <https://www.python.org/downloads/>

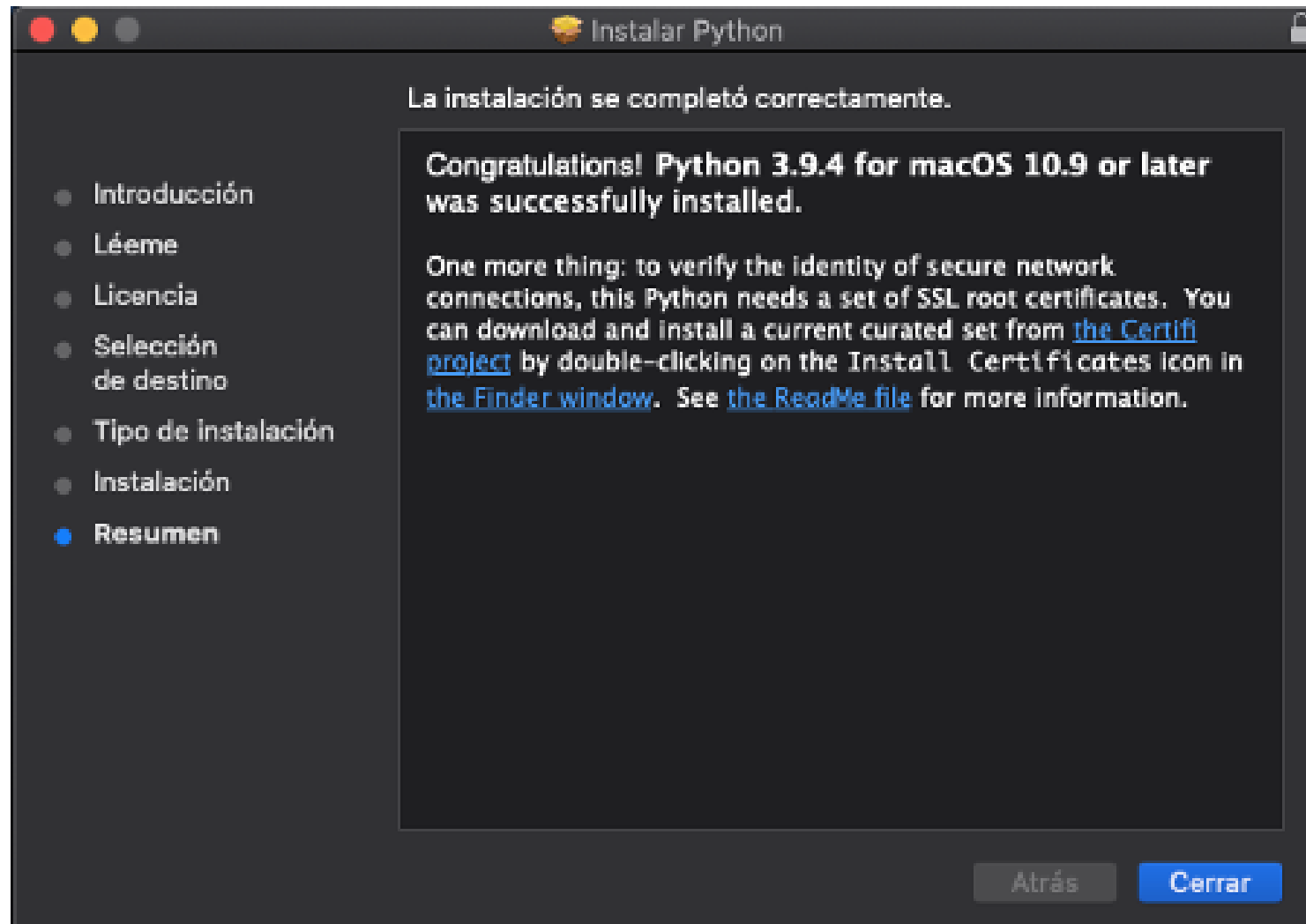
Download the latest version for Mac OS X

Download Python 3.9.4



Seguidamente, dar clic en Continuar; el tamaño del programa aproximadamente es de 118,6 MB.





Listo, vamos a cerrar y verificamos desde el shell:

```
>python3 --version
```

```
>Python 3.9.4
```

Listo para trabajar:

```
>python3
```

```
>Python 3.9.4 (v3.9.4:1f2e3088f3, Apr 4 2021, 12:32:44)
```

```
>[Clang 6.0 (clang-600.0.57)] on darwin
```

```
>Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> 3 +5
```

```
8
```

```
>>>
```

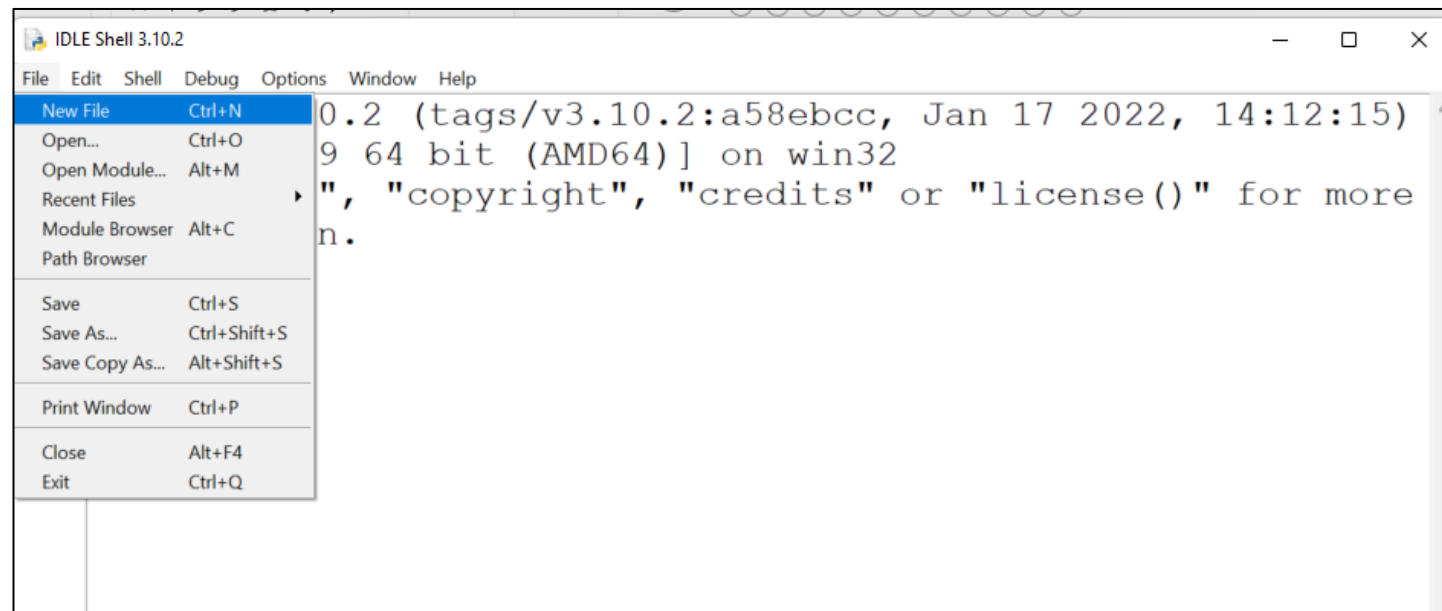
1.2. Creación de script en Python (¡ Hola Mundo !)

Para esta sección vamos a utilizar el IDE que viene incluido al realizar la instalación de Python llamado IDLE, pero también puedes utilizar tu editor de código favorito.

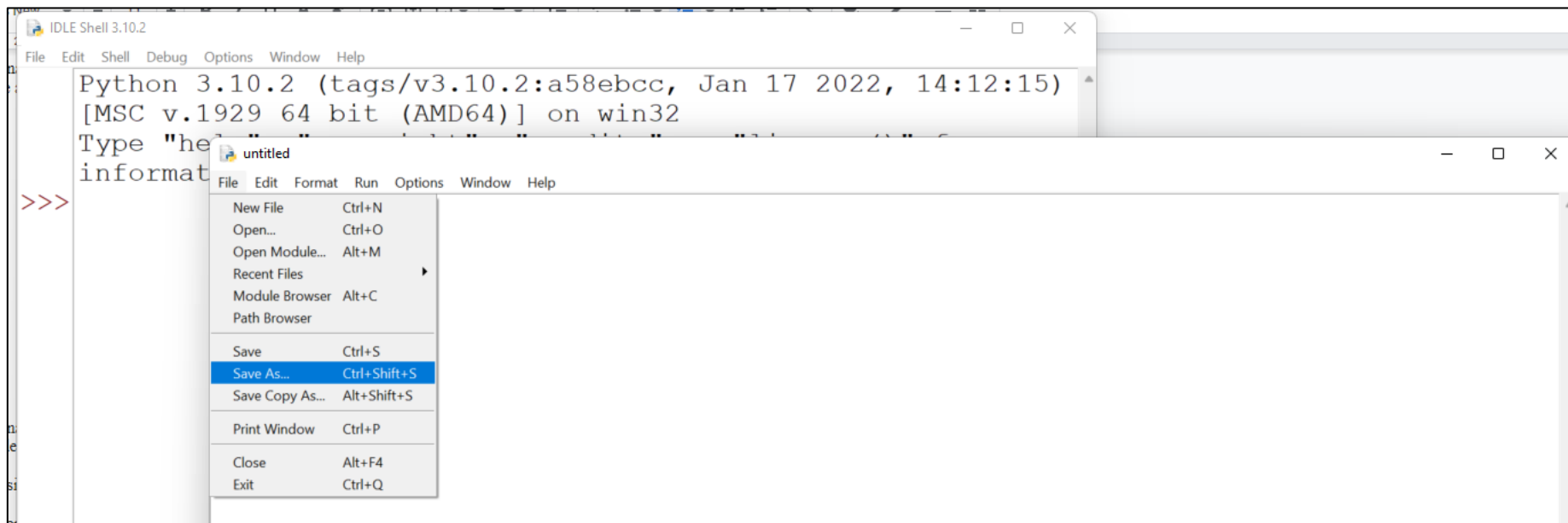
1. Lo primero que vamos a hacer es buscar la palabra IDLE en windows para abrir la aplicación.



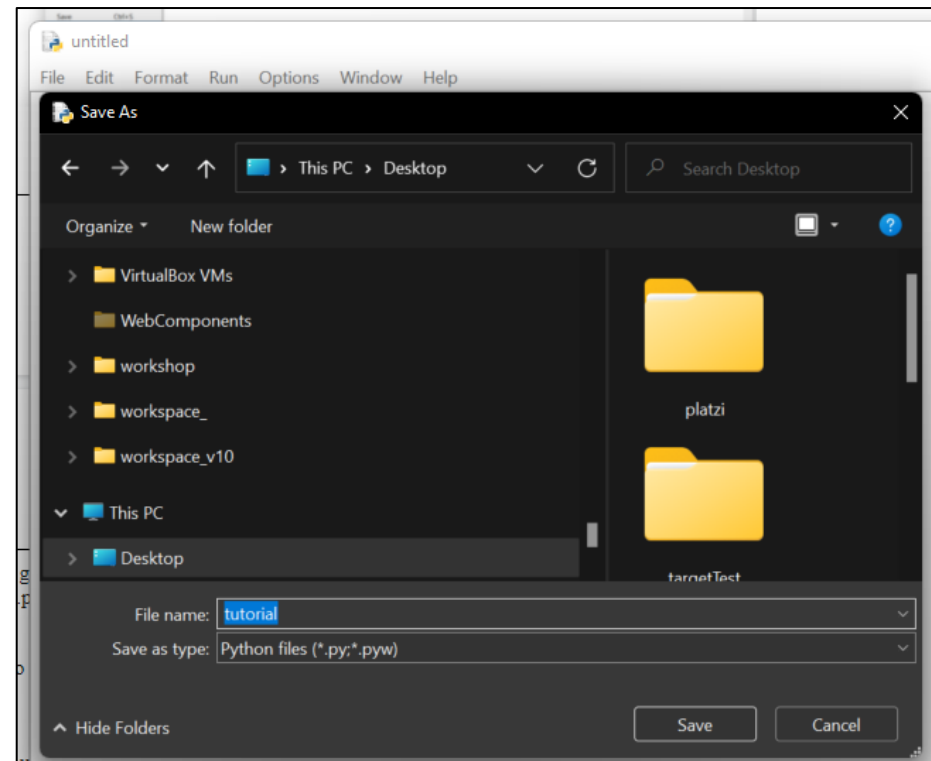
2. Una vez IDLE esté abierto, vamos a seleccionar File (archivo) > New File (nuevo archivo) para crear un nuevo script



3. Una vez esté abierto el editor con el script, notaremos que este aún no tiene nombre, por eso para guardarlo en alguna ubicación, vamos a seleccionar file > save as (guardar como).

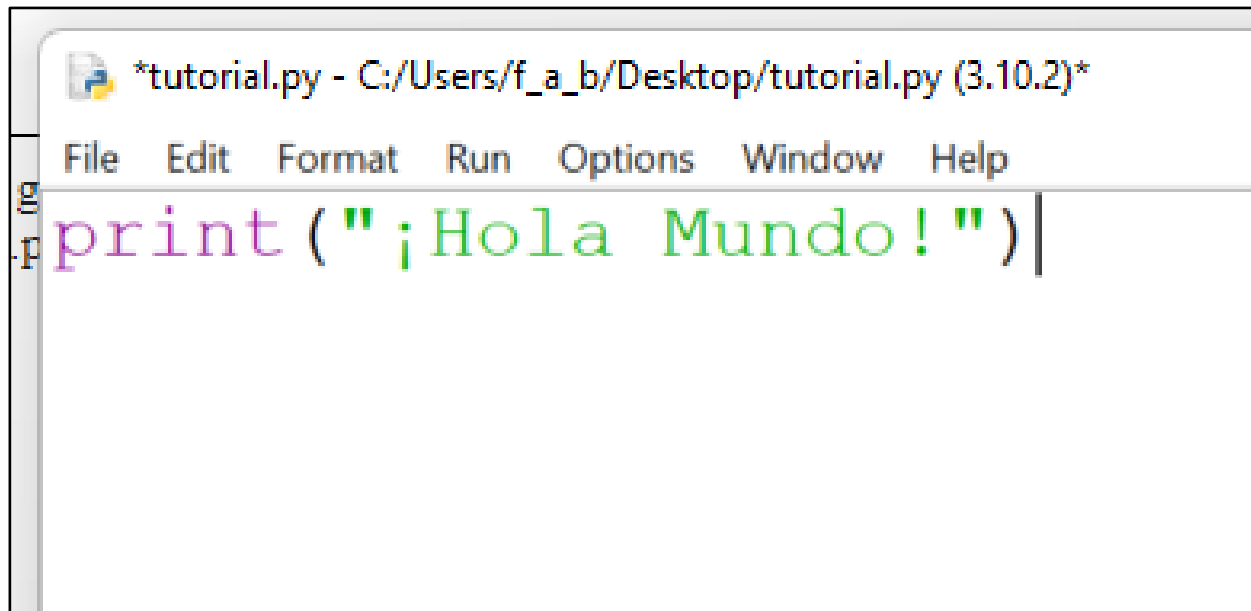


4. Asignaremos la ruta donde se guardará el archivo, colocaremos un nombre y daremos click en save (guardar).
Note que los scripts de python tienen la extensión de archivo .py.

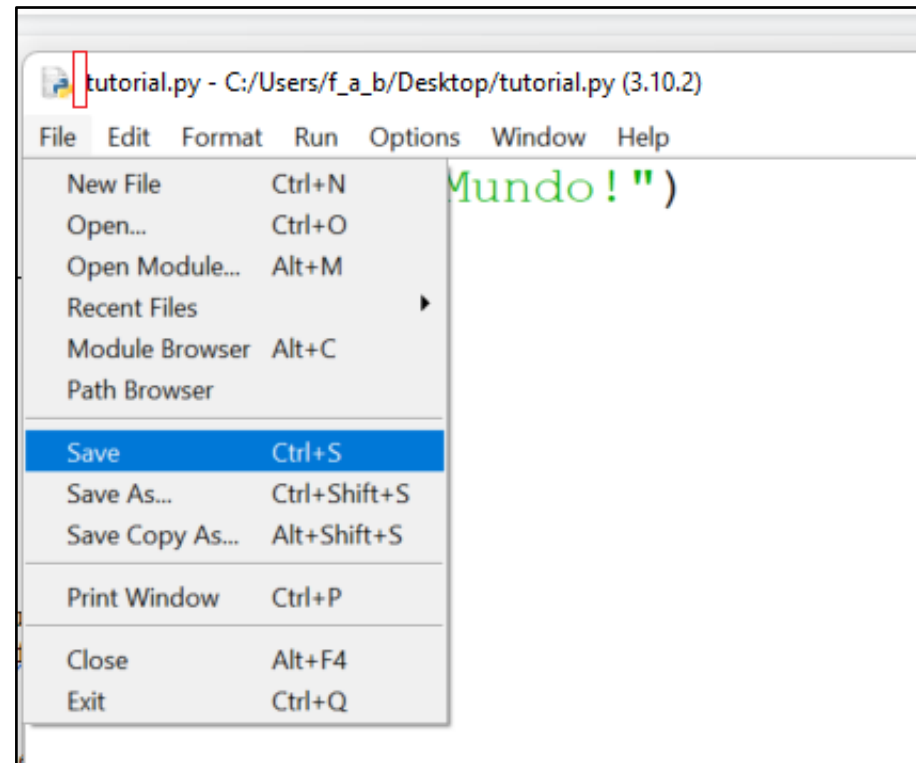


5. Con esto ya tenemos el entorno listo para empezar a escribir nuestro programa. En Python se utiliza la función `print()` para mostrar datos en la pantalla. Como vamos a escribir una frase, esta debe ir entre comillas; más adelante, se explicará con más detalle, por ahora escribiremos lo siguiente:

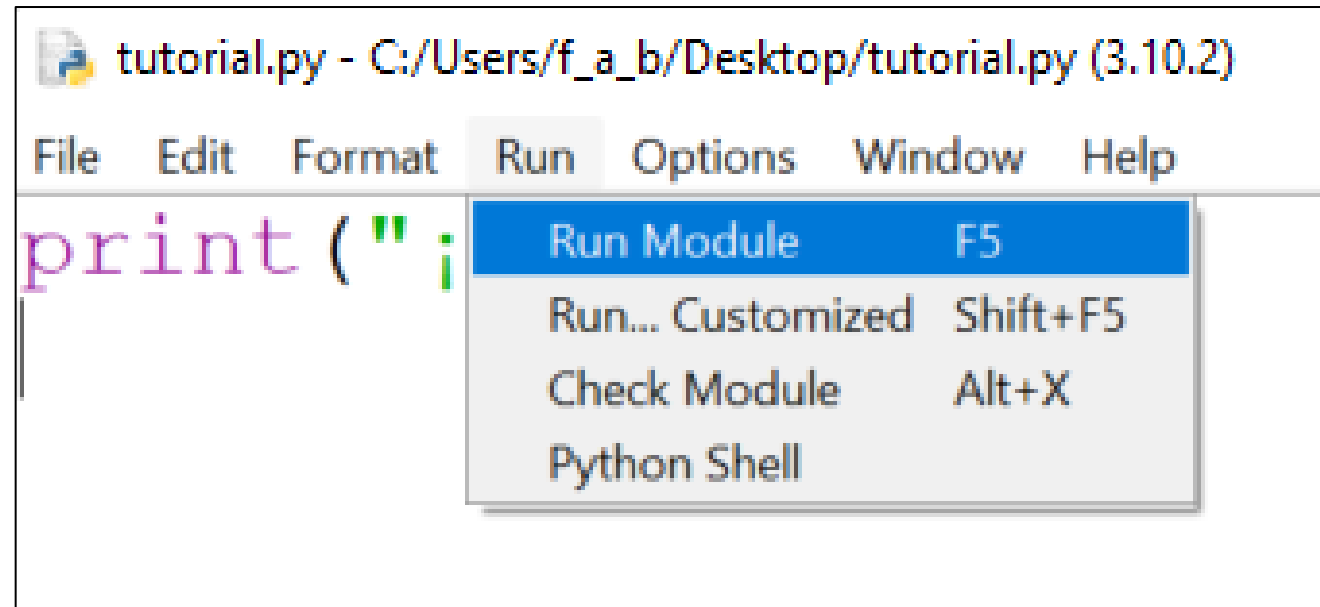
```
print("hola mundo")
```

A screenshot of a Python IDE window titled '*tutorial.py - C:/Users/f_a_b/Desktop/tutorial.py (3.10.2)*'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The main text area shows the code `print(';Hola Mundo!')` with a cursor at the end of the line. The code is color-coded: `print` is purple, `(` is green, `;` is green, `Hola` is green, `Mundo!` is green, and `)` is green. The background is white with a light gray border.

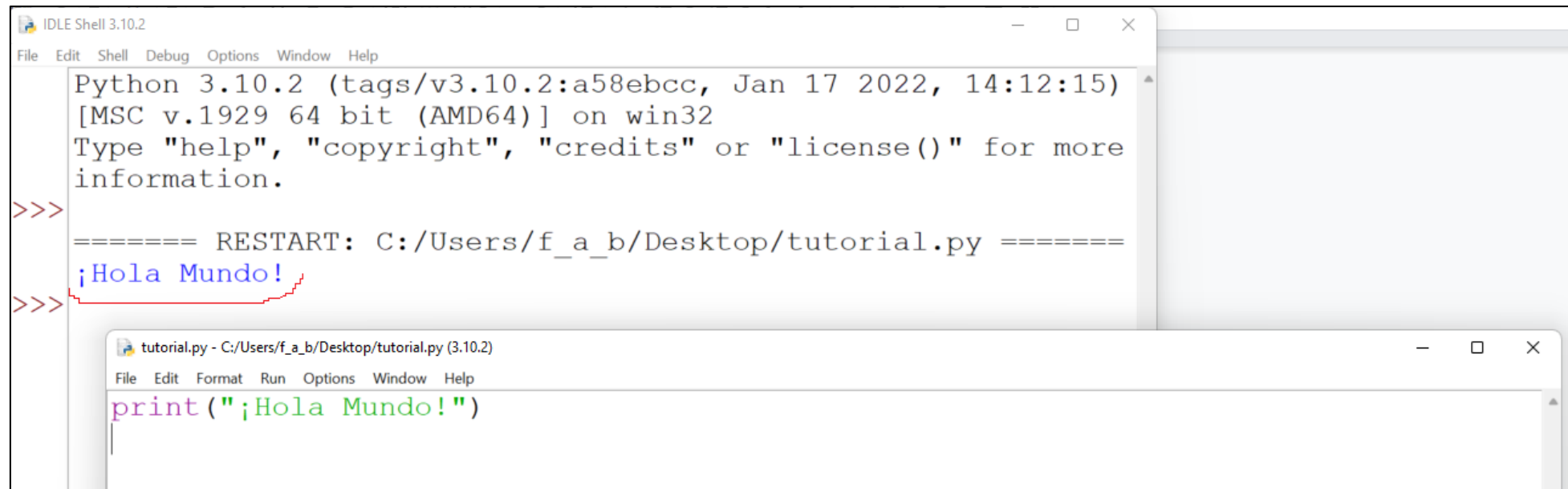
6. Siempre es bueno ir guardando los avances que vamos haciendo en el script, para ello podemos pulsar ctrl+s o ir a file > save. Note que el asterisco que acompaña el nombre tutorial.py en la parte superior, desaparece al momento de guardar.



7. Para ejecutar el programa que hemos escrito, basta con ir a run > run module o pulsar la tecla f5.



8. Veremos que en la shell (lugar donde se ejecuta el programa) se muestra la impresión de la palabra hola mundo.



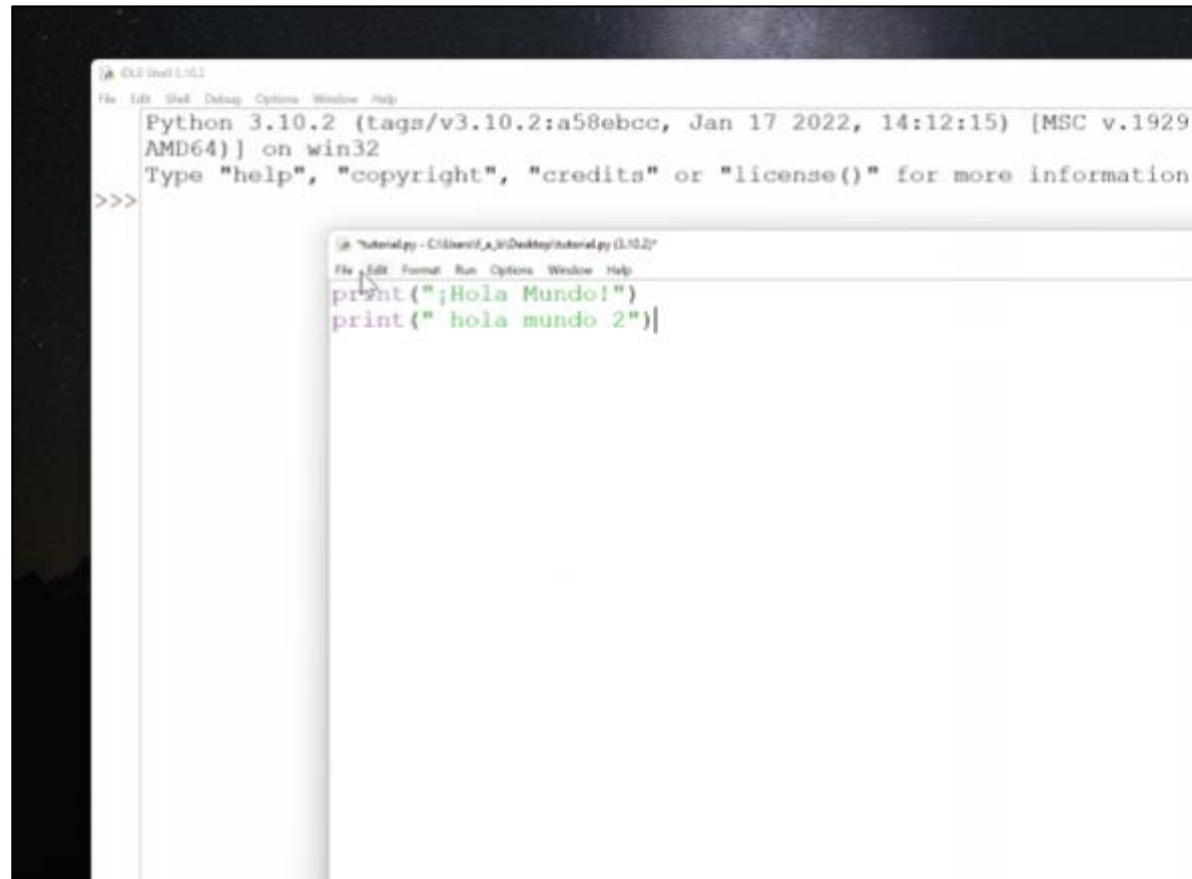
The image shows two windows from the Python IDLE 3.10.2 application. The top window is the 'IDLE Shell 3.10.2', which displays the Python startup banner and the execution of a script. The output shows a restart of the script 'tutorial.py' followed by the printed text '¡Hola Mundo!'. The bottom window is the 'tutorial.py' editor, showing the source code of the script: `print("¡Hola Mundo!")`.

```
IDLE Shell 3.10.2
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
===== RESTART: C:/Users/f_a_b/Desktop/tutorial.py =====
¡Hola Mundo!
>>>
```

```
tutorial.py - C:/Users/f_a_b/Desktop/tutorial.py (3.10.2)
File Edit Format Run Options Window Help
print("¡Hola Mundo!")
```

9. Supongamos que apagaste el computador y quieres continuar escribiendo el programa que tenemos escrito en el script, para abrir el script nuevamente basta con abrir IDLE, ir a file > open y buscar la ruta y el archivo que guardaste.

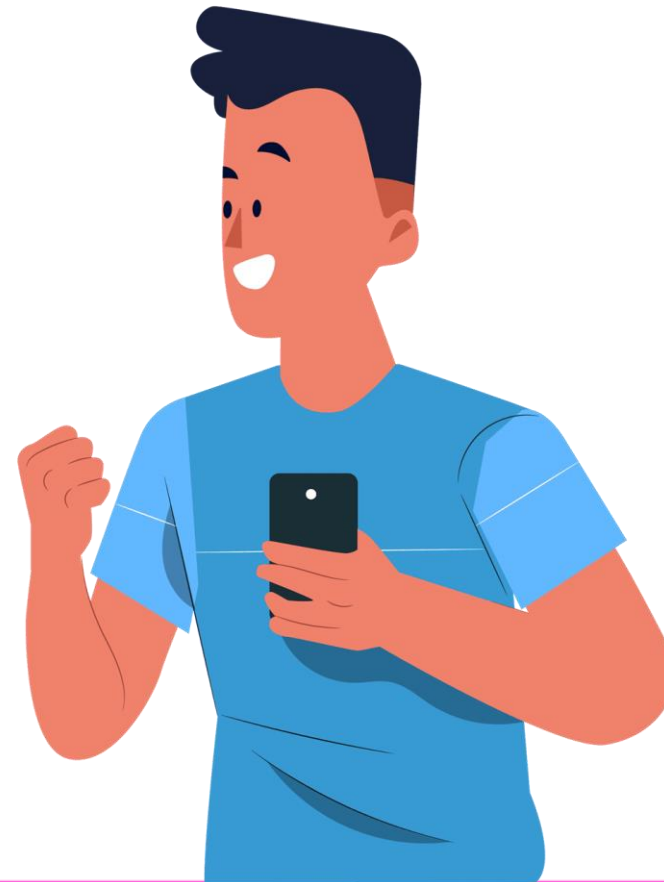
10. Podrías añadir un nuevo print debajo del anterior y ejecutar.

The image shows a screenshot of a Python IDLE environment. The background window is the IDLE Shell, titled 'Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 AMD64] on win32'. It displays the standard Python prompt 'Type "help", "copyright", "credits" or "license()" for more information.' and the prompt '>>>' is visible. Overlaid on top of the shell is a smaller window titled 'tutorial.py - C:\Users\user\Desktop\tutorial.py (3.10.2)'. This window shows a Python script with two lines of code: `print(";Hola Mundo!")` and `print(" hola mundo 2")`. The cursor is positioned at the end of the second line of code.

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

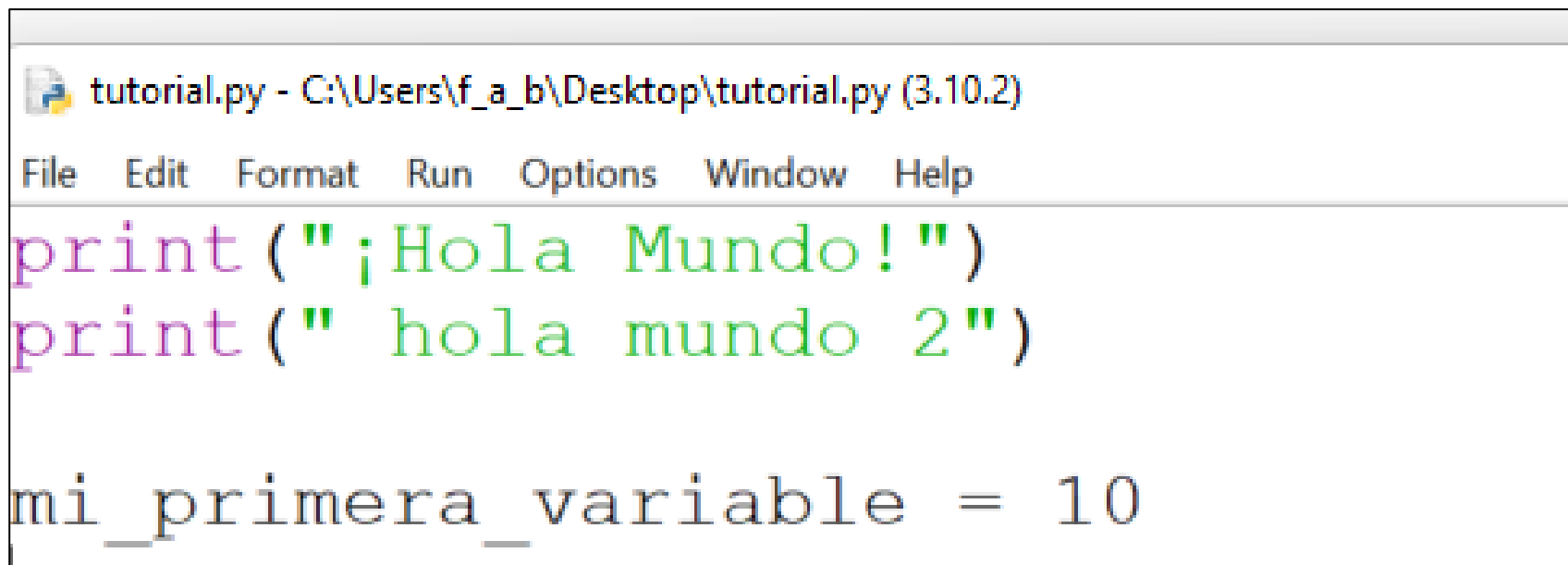
```
print(" hola mundo 2")
```

11. ¡Felicidades!, ya sabes como crear scripts de Python, guardarlos y ejecutarlos.



1.3. Declaración de variables en Python

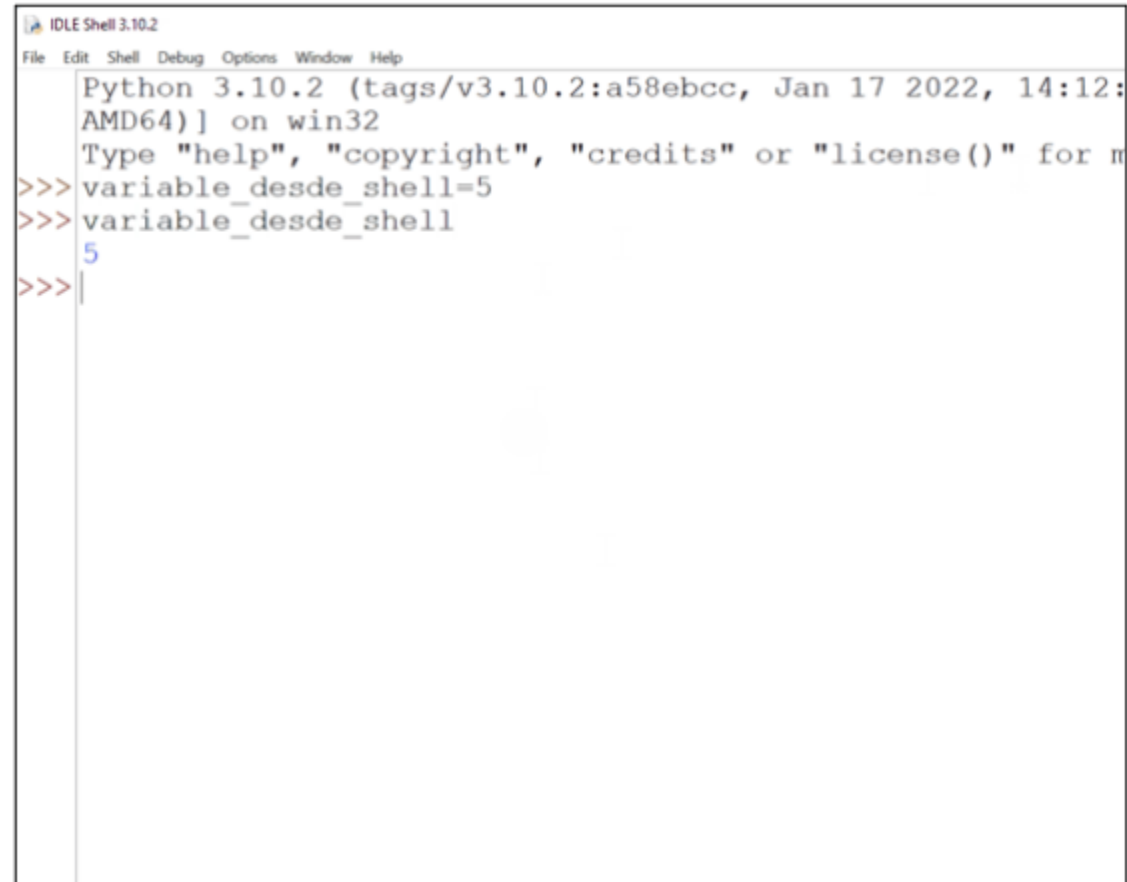
1. Continuando sobre el script anterior, para declarar una variable en python basta con escribir el nombre de la variable seguido del valor que va a almacenar la variable.



```
tutorial.py - C:\Users\f_a_b\Desktop\tutorial.py (3.10.2)
File Edit Format Run Options Window Help
print(";Hola Mundo!")
print(" hola mundo 2")

mi_primera_variable = 10
```

2. Es posible escribir líneas de código directamente sobre la Shell; para ello, ten en cuenta que lo que escribas allí no se almacenará en el script y no podrás recuperarlo cuando salgas de IDLE.



```
IDLE Shell 3.10.2
File Edit Shell Debug Options Window Help
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for m
>>> variable_desde_shell=5
>>> variable_desde_shell
5
>>> |
```

1.4. Tipos de datos e instrucciones básicas

En esta sección aprenderemos a declarar variables con diferentes tipos de valores.

1. Declarar variable entera: si al inicializar una variable, el valor es un número sin parte decimal, Python reconocerá que se trata de una variable de tipo entera (int).

2. Declarar variable flotante: si al inicializar una variable, el valor es un número con parte decimal, Python reconocerá que se trata de una variable de tipo flotante (float).

Junto con la función print, podemos mostrar el contenido de las variables en la pantalla; por ejemplo:



```
File Edit Format Run Options Window Help
print("¡Hola Mundo!")
print(" hola mundo 2")

I
variable_entera=10
4. print(variable_entera)
variable_flotante=10.5
En print(variable_flotante)|
```

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

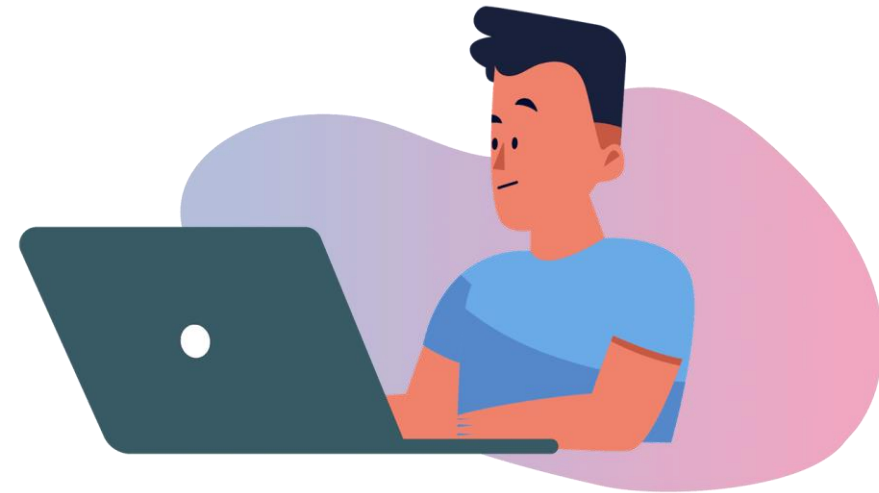
```
print(" hola mundo 2")
```

```
variable_entera=10
```

```
print(variable_entera)
```

```
variable_flotante=10.5
```

```
print(variable_flotante)
```



3. La función `type` nos ayuda a saber el tipo de variable, y junto con la función `print`, podemos mostrar el tipo de variable en pantalla:

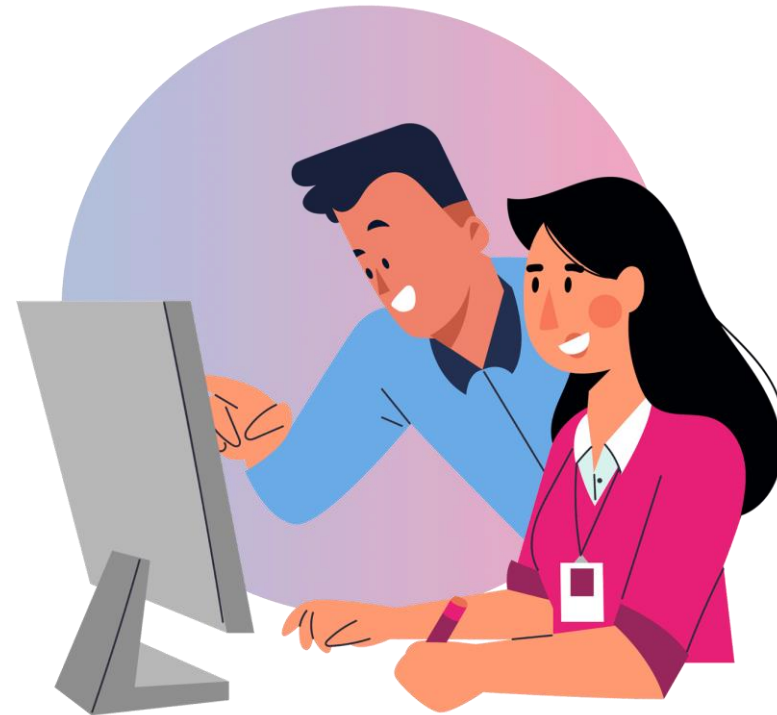


```
print("¡Hola Mundo!")
print(" hola mundo 2")

variable_entera=10
print(variable_entera)
print(type(variable_entera))

variable_flotante=10.5
print(variable_flotante)
print(type(variable_flotante))
```

```
print(";Hola Mundo!")  
  
print(" hola mundo 2")  
  
variable_entera=10  
print(variable_entera)  
print(type(variable_entera))  
  
variable_flotante=10.5  
print(variable_flotante)  
print(type(variable_flotante))
```



4. Para declarar una cadena de texto (string) el valor deberá ir encerrado entre comillas simples o doble, por ejemplo:



```
File Edit Format Run Options Window Help
print(";Hola Mundo!")
print(" hola mundo 2")

variable_entera=10
print(variable_entera)
print(type(variable_entera))

variable_flotante=10.5
print(variable_flotante)
print(type(variable_flotante))

cadena_texto="ejemplo cadena"
print(cadena_texto)
print(type(cadena_texto))
```



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

```
print(" hola mundo 2")
```

```
variable_entera=10
```

```
print(variable_entera)
```

```
print(type(variable_entera))
```

```
variable_flotante=10.5
```

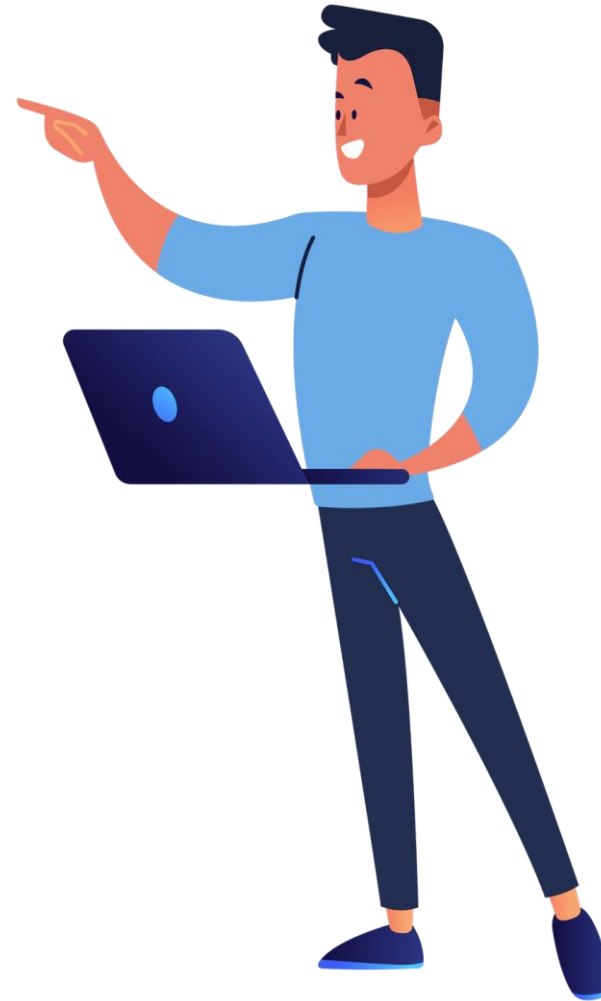
```
print(variable_flotante)
```

```
print(type(variable_flotante))
```

```
cadena_texto="ejemplo cadena"
```

```
print(cadena_texto)
```

```
print(type(cadena_texto))
```



1.5. Conversión entre variables:

1. Para convertir un número entero a texto (string), podemos hacer lo siguiente:

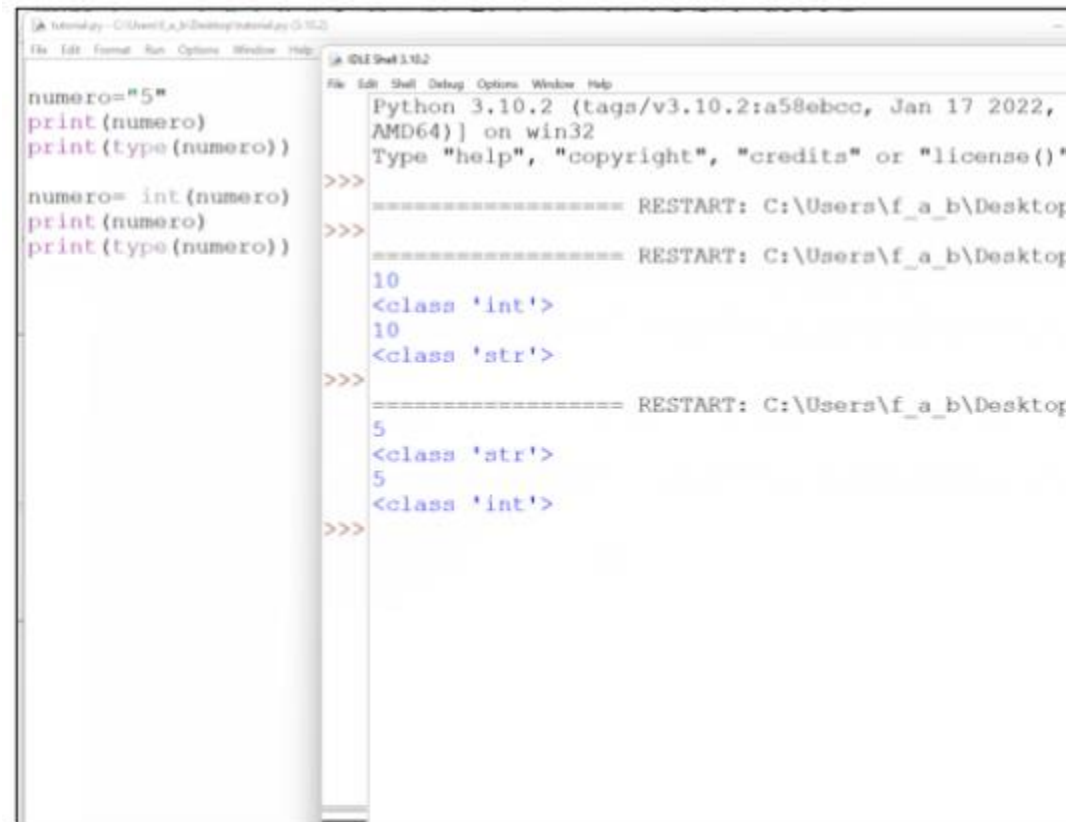


```
tutorial.py - C:\Users\fa_b\Desktop\tutorial.py (3.10.2)
File Edit Format Run Options Window Help

numero=10
print(numero)
print(type(numero))

numero=str(numero)
print(numero)
print(type(numero))
```

2. Para convertir un texto que contiene un número a una variable entera, podemos hacer lo siguiente:



```
numero="5"
print(numero)
print(type(numero))

numero= int(numero)
print(numero)
print(type(numero))
```

```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 1
AMD64)] on win32
Type "help", "copyright", "credits" or "license()"
>>>
===== RESTART: C:\Users\f_a_b\Desktop\
>>>
===== RESTART: C:\Users\f_a_b\Desktop\
10
<class 'int'>
10
<class 'str'>
>>>
===== RESTART: C:\Users\f_a_b\Desktop\
5
<class 'str'>
5
<class 'int'>
>>>
```

```
numero="5"
```

```
print(numero)
```

```
print(type(numero))
```

```
numero = int(numero)
```

```
print(numero)
```

```
print(type(numero))
```



```
numero=10
```

```
print(numero)
```

```
print(type(numero))
```

```
numero = str(numero)
```

```
print(numero)
```

```
print(type(numero))
```



3. Para convertir un número flotante a texto, podemos hacer lo siguiente:

A screenshot of a Python IDE window titled "Tutorial.py - C:\Users\F_A\Desktop\tutorial.py (3.10.2)". The code inside the editor is as follows:

```
numero=3.4
print(numero)
print(type(numero))

numero= str(numero)
print(numero)
print(type(numero))
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

The last line of code is highlighted in yellow. A cursor is positioned on the line following the last line of code.

```
numero=3.4
```

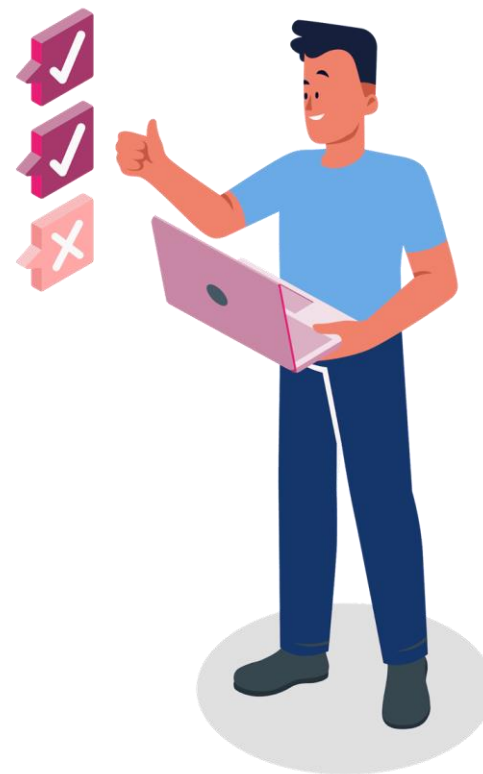
```
print(numero)
```

```
print(type(numero))
```

```
numero = str(numero)
```

```
print(numero)
```

```
print(type(numero))
```



4. Para convertir un número flotante a entero (perdiendo la parte decimal), podemos hacer lo siguiente:



```
Python - C:\Users\F_x\Desktop\tutorial.py (3.5.2)
File Edit Format Run Options Window Help
numero=10.5
print(numero)
print(type(numero))

numero = int(numero)
print(numero)
print(type(numero))
```



```
numero=10.5
```

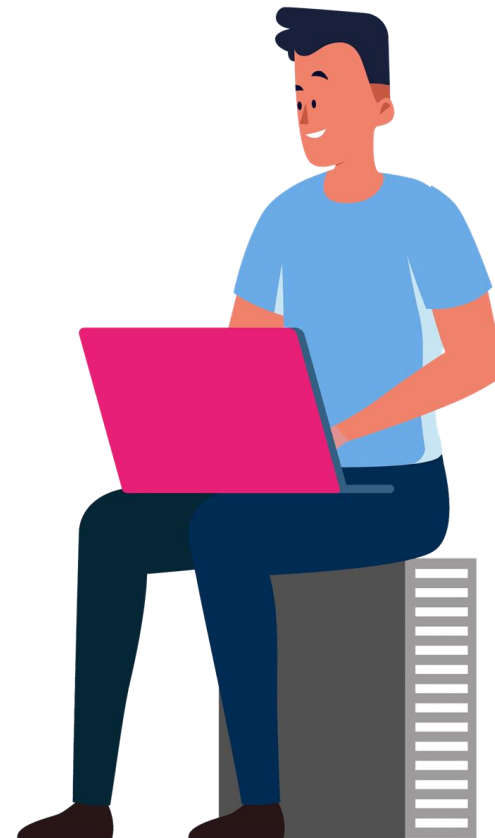
```
print(numero)
```

```
print(type(numero))
```

```
numero = int(numero)
```

```
print(numero)
```

```
print(type(numero))
```



1.6. Operadores aritméticos

1.6.1. Suma

- a. La sintaxis en la que se realiza una suma en Python es muy sencilla, basta con poner un número o una variable, seguido del símbolo suma (+) y otro número o variable, tal como se escribiría una suma en una línea de cuaderno. En el ejemplo en pantalla, le pedimos a Python que imprima el resultado de 10 más 20. Recordemos que para la suma el orden de los factores no altera el producto, por lo tanto, da igual si sumamos $20 + 10$ que $10 + 20$.

```
print (10 + 20)
```



```
30
```

- b. El operador suma, al igual que los demás operadores que veremos después, se puede emplear para realizar sumas entre dos variables, o variables con números. Por ejemplo, tenemos una variable llamada número de manzanas cuyo valor es cero. Si nosotros queremos añadirle una unidad, basta con sumar 1 a dicha variable. La forma en la que se escribe la sintaxis en Python es similar a la suma de dos números, basta con escribir el nombre de la variable, seguido del símbolo suma y luego la cantidad a sumar, como se muestra en pantalla.

```
numero_de_manzanas = 0  
print (numero_de_manzanas + 1)
```

```
1
```

- c. Si queremos guardar el resultado de la suma directamente en la variable que estamos usando, lo cual muy probablemente será el objetivo en la mayoría de las ocasiones, basta con declarar la variable nuevamente con el resultado de la operación suma, como se puede ver en pantalla.

```
numero_de_manzanas = 0  
numero_de_manzanas = numero_de_manzanas + 1  
print (numero_de_manzanas)
```

1

- d. Quizá hayas pensado en lo siguiente: ¿Y qué pasa si quiero sumar, ya no dos números o variables, sino tres, cuatro o quizá cinco? No hay problema, se sigue la misma sintaxis que ya hemos aprendido, basta con seguir añadiendo sumas en la misma línea, por ejemplo:

```
a1 = 10  
a2 = 20  
a3 = 30  
suma = a1 + a2 + a3 + 100
```

- e. Es posible sumar números flotantes con números enteros, debido a que los números flotantes son una extensión de los enteros. Por ejemplo: tenemos la variable `x1`, que vale 100, y la variable `x2` que vale 2.5, la forma en la que se sumaría es similar a como ya hemos aprendido, pero hay una ligera diferencia, el tipo de dato resultante de esa operación, será automáticamente modificado a flotante. Podemos comprobar el tipo de dato imprimiendo el resultado de la palabra reservada `type`.

```
x1 = 100
x2 = 2.5
print (type(x1+x2))

<class 'float'>
```

- f. Con las cadenas de texto o también llamadas *strings*, el operador suma lo que hace es juntar, pegar ambas cadenas. A este proceso se le llama concatenación. Por ejemplo, tenemos una variable que almacena una cadena que dice "hola" y otra variable con una cadena que dice "adiós". Si sumamos ambas variables obtendremos lo siguiente:

```
v1 = "hola"
v2 = "adiós"
print (v1 + v2)

holaadiós
```

1.6.2. Resta

- a. Lo primero y más importante a tener en cuenta, es que la resta, como lo es en las matemáticas, NO es una operación conmutativa. Es decir, el orden en el que se escriba la operación SÍ influye e importa, por ejemplo: tenemos la variable a igual a 5 y la variable b igual a 10. restando a-b nos dará -5, Por otra parte, restando b-a, el resultado será 5.

```
a = 5  
b = 10  
print (a - b)
```

-5

```
a = 5  
b = 10  
print (b - a)
```

5

- b. Con la resta ocurre algo similar a la suma al momento de operar entre un tipo entero y un flotante, el resultado será un número flotante, por ejemplo:

```
>>> a=2.5
>>> b=5
>>> print (type (b-a) )
<class 'float'>
```

- c. Por último, la resta NO funciona entre cadenas de texto.

1.6.3. Multiplicación

- a. Para utilizar la multiplicación basta con usar la sintaxis que ya hemos aprendido para hacer operaciones, usando el símbolo asterisco (*), por ejemplo:


```
variable1 = 2  
variable2 = 3  
print (variable1 * variable2)
```

6

- b. Como la multiplicación es conmutativa, no importa el orden en el que hagamos la operación, para un ejemplo demostrativo, podemos tomar el ejemplo anterior e invertir el orden de las variables a la hora de multiplicar:

```
variable1 = 2  
variable2 = 3  
print (variable2 * variable1)
```

6



- c. El comportamiento de la operación entre un tipo entero y uno flotante nos dará un resultado flotante, como era de esperarse.

1.6.4. División

- a. El operador división similar a la resta, NO es conmutativo. El orden de la operación influye en el resultado. La división sigue la sintaxis que ya hemos aprendido, utilizando el símbolo slash (/). Por ejemplo:

```
print (11 / 2)
```

```
5.5
```

- b. La división tiene una característica muy importante a resaltar: esta operación siempre resultará en un valor flotante. Haremos el siguiente ejemplo, declararemos dos variables, A y B, estas variables tendrán como valor 100 y 20, respectivamente. Luego, mediante el comando type, verificaremos el tipo de ambas variables. Por último, realizaremos la división de A dividido por B y veremos el tipo de dato del resultado.

```
A = 100
B = 20
print (type (A))
print (type (B))
print (type (A/B))

<class 'int'>
<class 'int'>
<class 'float'>
```

- c. Finalizando con el operador división, es muy importante saber que no es posible dividir entre cero, esto es debido a que, matemáticamente, la división por cero no está definida. Si tratas de dividir un número entre cero, ya sea por accidente o con intención, te saldrá un error llamado ZeroDivisionError.

1.6.5. División entera

- a. Este sigue la sintaxis que ya hemos aprendido, usando dos veces el símbolo de la división juntos, como se muestra en el ejemplo en pantalla:

```
print (10 // 2)
```

```
5
```

- b. Para divisiones cuyo resultado sea flotante, siempre será redondeado hacia abajo, es decir, supongamos que nuestro resultado nos da 1.98. Si ese resultado fue empleado por una división entera, lo que nos devolverá será 1. Hagamos otro ejemplo, imprimamos el resultado de 2 dividido entre 3:

```
print (2/3)
```

```
0.6666666666666666
```

- c. Vemos que el resultado es 0.66 periódico. Ahora, si repetimos la misma prueba, pero en este caso usando el operador de división entera, veremos que el resultado obtenido es cero:

```
print (2//3)
```

0

- d. Para finalizar respecto al operador de división entera, al igual que con una división normal, no es posible dividir entre cero, debido a que la división entre cero es matemáticamente indefinida.

1.6.6. Potenciación

- a. La potenciación no es conmutativa, por lo tanto, el orden importa. En Python, la operación de potenciación usa la misma sintaxis que ya hemos aprendido, usando dos veces el símbolo de multiplicar juntos, donde la base estará al lado izquierdo del operador y el exponente al lado derecho. Insisto, la potenciación no es conmutativa, por lo tanto, el orden importa.

```
print (2**3)
```

8

```
print (3**2)
```

9

- b. En Python, todo número cuyo exponente sea cero es igual a uno, incluyendo el cero, es decir, Python devuelve cero elevado a la cero igual a 1.

1.6.7. Módulo


- a. El módulo es el operador más desafiante de entender, pero es uno muy útil en determinadas situaciones, El operador módulo tiene como función devolver el residuo, o también llamado resto, de una división:

The diagram illustrates a division problem: $125 \div 5 = 25$ with a remainder of 0. The components are labeled as follows:

- DIVIDENDO** (Dividend): 125, indicated by a cyan arrow.
- DIVISOR** (Divisor): 5, indicated by an orange arrow.
- COCIENTE** (Quotient): 25, indicated by a green arrow.
- RESTO** (Remainder): 0, indicated by a purple arrow. The remainder is circled in red.

- b. El operador módulo no es conmutativo, el orden importa. En Python, este operador se escribe con la sintaxis que ya hemos aprendido, utilizando el símbolo porcentaje (%).

```
print (125%5)
```



- c. El módulo tiene muchas utilidades, una de ellas, de las más comunes, es determinar si un número es múltiplo de otro. Por ejemplo, si queremos saber si un número es par, es decir, que sea múltiplo de dos, podemos verificar que el módulo entre dos sea cero.

1.7. Operadores de comparación

1.7.1. Igualdad

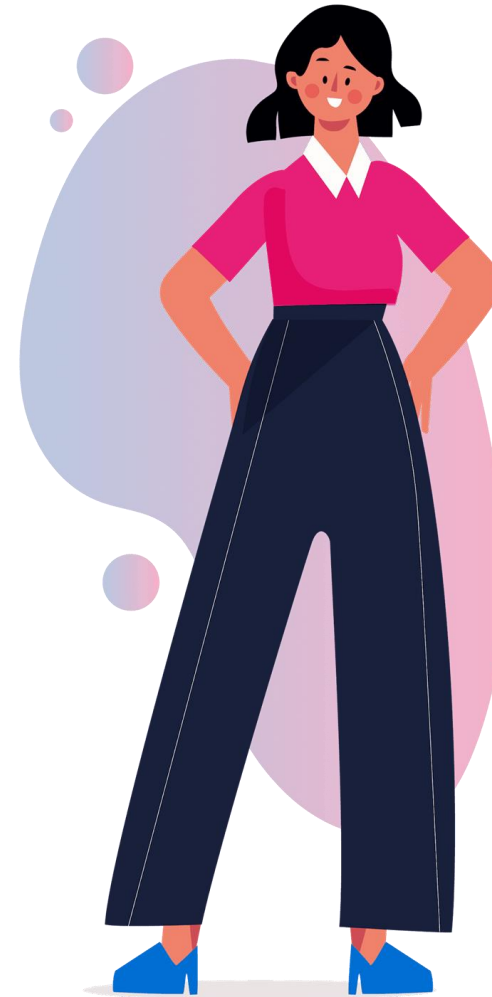
- a. Empecemos por el operador más sencillo, el de igualdad. Esto es bastante sencillo, lo que hace es comparar si ambos valores dados para el operador son iguales, en caso de serlo, devolverá True; en caso contrario, devolverá False. Veamos un ejemplo: tenemos dos variables, X y Y, ambas valen 5 y queremos saber si son iguales, la forma en la que en Python se realiza esta operación es usando el operador de igualdad, cuyo símbolo es dos iguales juntos (==). Entonces, quedaría así:

```
X = 5
Y = 5
print (X==Y)

True
```

- b. Ahora, cambiemos el valor de Y a 6, veremos que el resultado ahora será falso porque cinco no es igual a seis.

```
X = 5  
Y = 6  
print (X==Y)  
  
False
```



1.7.2. Diferencia

- a. Su símbolo es (!=) Por ejemplo, si tenemos dos variables, X y Y, con el valor de 5 y 6, respectivamente. Si imprimimos X es diferente a Y, nos devolverá True, porque efectivamente cinco es diferente a seis.

```
X = 5
Y = 6
print (X!=Y)

True
```

- b. Evidentemente, si ponemos algo, por ejemplo, "¿es cinco diferente de cinco?" nos devolverá falso.

1.7.3. Mayor que

- a. Este operador nos devuelve True si el número de la izquierda es mayor que el de la derecha, False en caso contrario. Su símbolo es idéntico al mayor que de las matemáticas (>). A continuación, se muestra si 20 es mayor que 10 y el resultado de la operación de comparación:

```
print (20 > 10)
```

```
True
```

1.7.4. Menor que

- a. El operador menor que es el opuesto al mayor que, es decir, devolverá True si el número de la izquierda es menor al de la derecha, False en caso contrario. Similar al operador anterior, su símbolo es idéntico al menor que el de las matemáticas (<).
- b. A continuación, se muestra cómo le preguntaremos a Python si la variable X, que vale 5, es menor que la variable Y, que vale 1 y que nos imprima en pantalla el resultado. Vemos que la respuesta es False, porque efectivamente cinco NO es menor que uno.

```
X = 5  
Y = 1  
print (X<Y)
```

```
False
```

1.7.5. Mayor o igual que

- a. Este operador es similar al mayor que, la única diferencia es que también devolverá True si ambos números son iguales. Su símbolo es el mayor que, junto con un igual. (\geq)
- b. Un ejemplo de su diferencia con el mayor que, es el siguiente: Tenemos dos variables, X y Y, ambas tienen el valor de 5. Queremos ver si X es mayor o igual que Y e imprimir el resultado de la operación.

```
X = 5
Y = 5
print (X>=Y)

True
```

- c. Vemos que el resultado es verdadero, True, esto es debido a que, aunque cinco no sea mayor que cinco, sí son iguales. Nótese que el operador se llama mayor o igual. Es por esto por lo que, si se cumple alguna de estas dos condiciones, el resultado es verdadero.

1.7.6. Menor o igual que

a. Al igual que pasa con el operador anterior, el menor o igual que, es similar al menor que, con la diferencia de que devolverá True si ambos números son iguales. Su símbolo es el menor que, junto con igual. (\leq)

b. Tenemos una variable llamada B cuyo valor es 10, queremos imprimir si B es menor o igual que 50. En pantalla se muestra el código.

```
B = 10
print (B<=50)

True
```

c. El resultado es True, verdadero, esto es debido a que, aunque 10 no sea igual a 50, sí es menor que 50. Recordemos que el operador es menor o igual que, por lo tanto, al cumplirse al menos una de las dos condiciones, el resultado es verdadero.

1.8. Operadores lógicos

1.8.1. Operador AND

- a. Para usar este operador basta con escribir un valor booleano, luego la palabra reservada "and" en minúscula y por último, otro valor booleano, por ejemplo:

```
print (True and False)  
False
```

- b. Podemos deducir que si escribimos en Python "falso y verdadero", nos devolverá falso:

```
print (False and True)  
False
```

- c. O, por el contrario, si escribimos "verdadero y verdadero", nos devolverá verdadero.

```
print (True and True)  
True
```

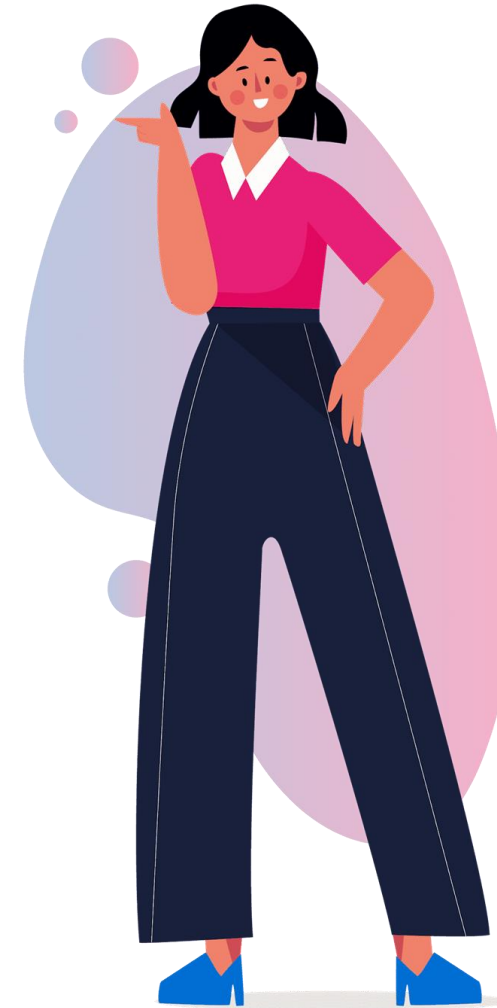

1.8.2. Operador OR

- a. Haremos como ejemplo lo que haría un operador "Mayor o igual que" sin usar directamente ese operador. Sabemos que los operadores de comparación devuelven valores booleanos, justo lo que necesitan nuestros operadores lógicos para funcionar, entonces podemos ponernos manos a la obra.
- b. Tenemos dos variables, X y Y, ambas con el valor de cinco. Vamos a imprimir en pantalla si X es mayor o igual que Y, sin usar el operador de comparación "mayor o igual". Primero, tendríamos que hacer la comparación de si X es mayor que Y, luego, concatenar dicha comparación con el operador OR junto con la comparación de si X es igual a Y, como se muestra en pantalla.

```
X = 5
Y = 5
print (X > 5 or X == 5)

True
```

- c. Vemos que el resultado de la operación lógica es True. Esto es porque, a pesar de que cinco no sea mayor que cinco, sí se cumple el operador lógico de la derecha, ya que cinco sí es igual a cinco. Todo gracias a que estamos usando el operador OR.
- d. Si lo hubiésemos hecho con el operador AND, nos hubiese devuelto False, porque recuerden, el operador AND solo devuelve verdadero si ambos valores son verdaderos.



1.8.3. Operador NOT

- a. Queremos que Python imprima en pantalla el inverso de False, es decir, que nos imprima True, sin imprimir la palabra reservada True. El código sería el siguiente:

```
print (not False)  
  
True
```

- b. Vemos que estamos negando el False, lo cual inmediatamente lo convierte en un True. Este peculiar comportamiento de invertir es muy útil en ciertos casos, donde trabajar con el valor original sea más difícil.

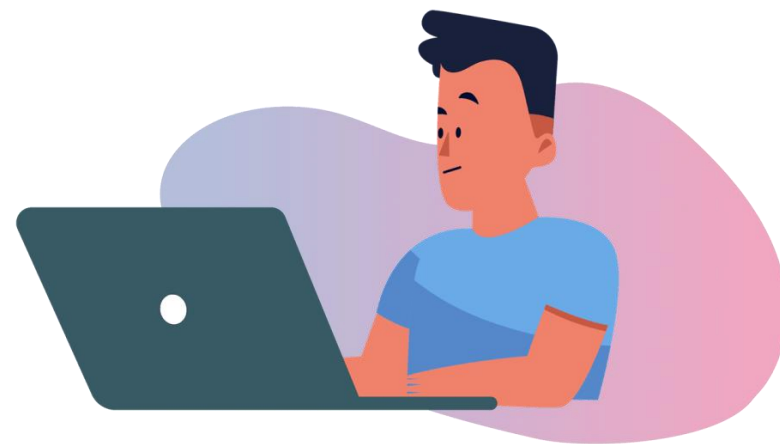
1.9. Precedencia de operadores

1. Vamos a tratar de mentalmente calcular el resultado de la siguiente operación:

$$7-2(13+5)$$

2. Si tu respuesta es 90 estás equivocado. La respuesta correcta es - 29. ¿Por qué? Por el orden en el que hay que realizar las operaciones.
3. Primero debemos resolver los paréntesis, la expresión quedaría así:

$$7-2(18)$$



4. Luego vienen los exponentes, que en este caso no tenemos ninguno, por lo tanto, podemos proceder con las multiplicaciones y divisiones. ¿Ves que el 2 está siendo multiplicado por 18? El resultado de esa multiplicación es 36. Nuestra expresión en este momento iría así:

7-36

5. Por último, resolvemos las sumas y las restas. Dando como resultado -29.
6. ¿Qué crees que se imprimirá en pantalla con el siguiente código? Trata de realizar la operación en tu cabeza.

```
X = True
Y = False
Z = False
print (not Z and (X or Y))
```

7. La respuesta es True. Fíjate en el paréntesis del "X or Y", esto hace que se realice esta operación primero que las demás. Como X es verdadero y Y es falso, el operador OR, como ya hemos aprendido, devolverá verdadero, luego, las demás operaciones tienen el mismo nivel de prioridad, por lo tanto, se resolverán de izquierda a derecha. El operador NOT invierte el valor booleano de Z, convirtiéndolo en True, luego se evalúa el operador AND, donde a la izquierda tenemos True y a la derecha, por el resultado del paréntesis, también. Verdadero y verdadero es verdadero
8. ¿Qué crees que se imprimirá en pantalla con el siguiente código? Trata de realizar la operación en tu cabeza.

```
X = True
Y = False
Z = False
print (not(not Z and not(X or Y)))
```

9. La respuesta es True.
10. Ahora que sabes aplicar los paréntesis a operaciones en Python, veamos el ejemplo visto previamente escrito en código Python.

```
print (7-2*(13+5))
```

-29

1.10. Entrada estandar INPUT

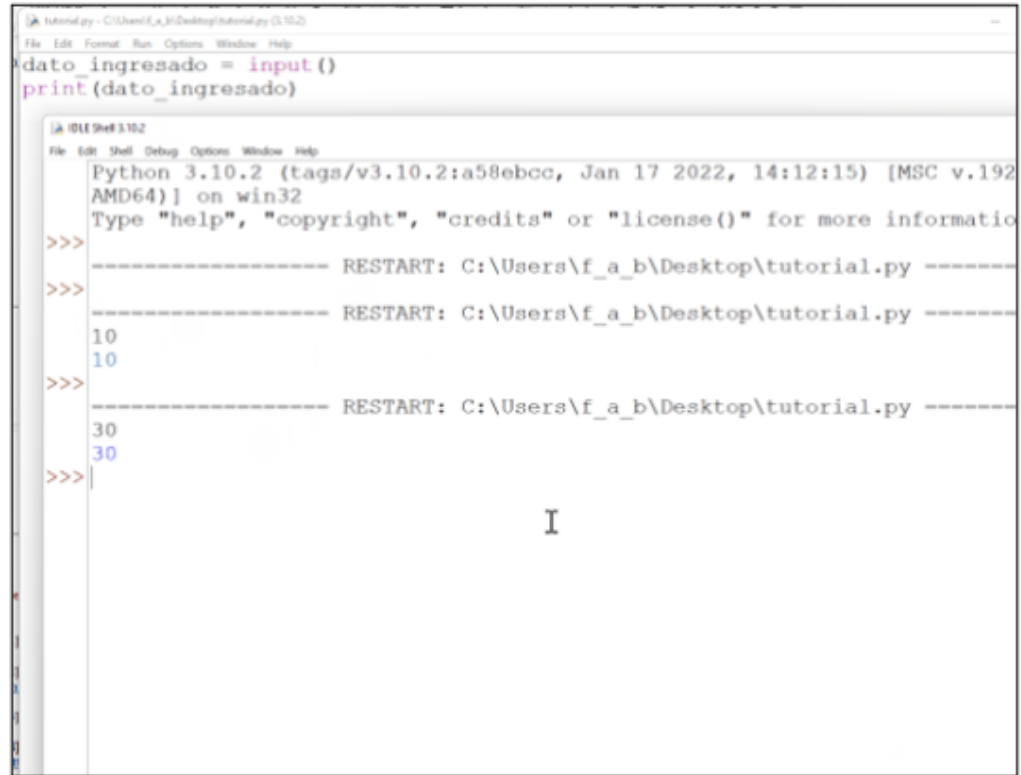
1. Así como con la función *print* podemos mostrar números, texto y otros tipos de variables a través de la terminal o Shell, también podemos ingresar datos a través de ella haciendo uso de la función *input*.
2. Si queremos ingresar algún dato mediante la terminal, podemos escribir en nuestro programa la función *input()*, aunque aún no hemos visto a fondo cómo se comportan las funciones, por ahora, debes tener en cuenta que la función *input()* va a devolver el dato ingresado por la terminal, por lo que este dato debe ser asignado a alguna variable, de la siguiente manera:

```
dato_ingresado = input()
```

3. Ahora podemos, a través de `print()`, mostrar el dato ingresado en la terminal, de la siguiente manera:

```
dato_ingresado = input()
```

```
print(dato_ingresado)
```



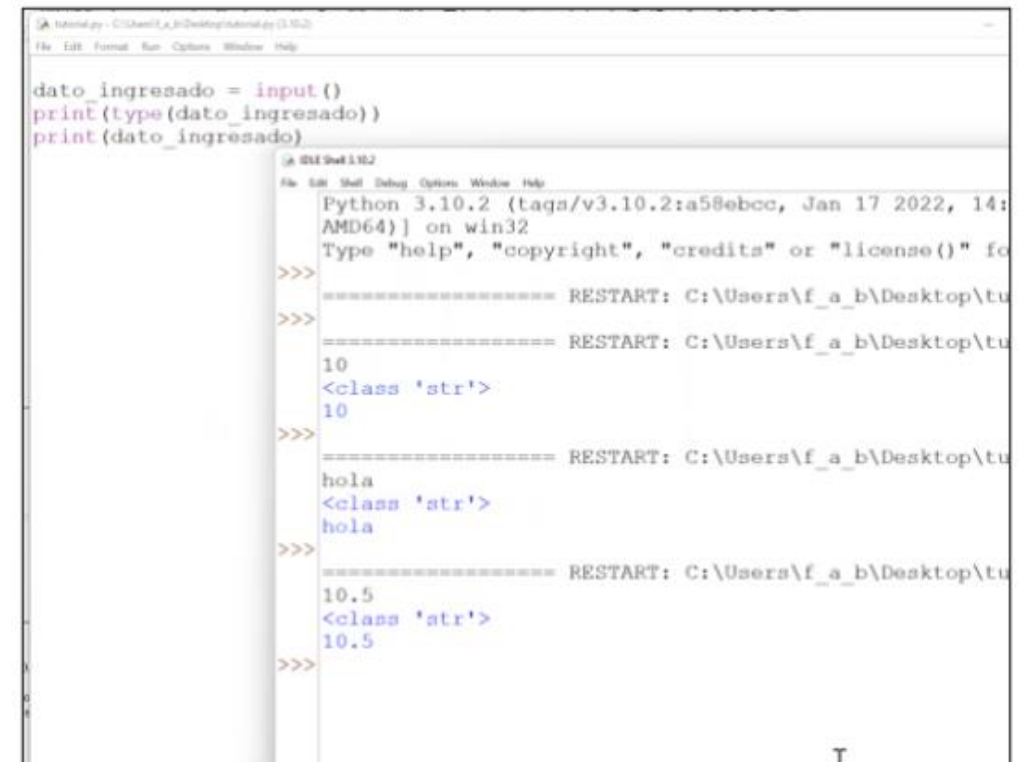
The screenshot shows a Python IDE window titled 'tutorial.py - C:\Users\fa\Desktop\tutorial.py (3.10.2)'. The script contains two lines of code: `dato_ingresado = input()` and `print(dato_ingresado)`. Below the script, a 'Python Shell 3.10.2' window displays the execution output. It shows the prompt `>>>` followed by the program's output. The first run shows the user inputting '10', which is then printed. Subsequent runs show the user inputting '30', which is also printed. The shell window also displays several 'RESTART' messages indicating the program was executed multiple times.

4. Como te pudiste dar cuenta, al ejecutar el programa se quedará esperando hasta que se ingrese algún dato, una vez se ingrese, el programa continúa con la siguiente operación, que en este caso es mostrar la variable usando `print (dato_ingresado)`.
5. Te preguntarán qué tipo de dato es `dato_ingresado`, podemos hacer uso de `type` para saberlo:

```
dato_ingresado = input()

print(type(dato_ingresado))

print(dato_ingresado)
```



```
dato_ingresado = input()
print(type(dato_ingresado))
print(dato_ingresado)
```

Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14: AMD64) on win32
Type "help", "copyright", "credits" or "license()" for more

>>> ===== RESTART: C:\Users\f_a_b\Desktop\tu
>>> 10
<class 'str'>
10

>>> ===== RESTART: C:\Users\f_a_b\Desktop\tu
>>> hola
<class 'str'>
hola


>>> ===== RESTART: C:\Users\f_a_b\Desktop\tu
>>> 10.5
<class 'str'>
10.5

6. Como te diste cuenta, el dato siempre será de tipo str, por lo que si quisiéramos realizar una suma aritmética con algún número ingresado, deberíamos **castear** la variable de *string* a entero o flotante; de lo contrario, dará un error, por ejemplo:

```
dato_ingresado = input()

dato_ingresado = dato_ingresado + 5

print(dato_ingresado)
```



```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" fo
----- RESTART: C:\Users\f_a_b\Desktop\tu
----- RESTART: C:\Users\f_a_b\Desktop\tu
10
<class 'str'>
10
----- RESTART: C:\Users\f_a_b\Desktop\tu
hola
<class 'str'>
hola
----- RESTART: C:\Users\f_a_b\Desktop\tu
10.5
<class 'str'>
10.5
----- RESTART: C:\Users\f_a_b\Desktop\tu
20
Traceback (most recent call last):
  File "C:\Users\f_a_b\Desktop\tutorial.py", line 2,
    dato_ingresado = dato_ingresado + 5
TypeError: can only concatenate str (not "int") to str
>>>
```

7. Podemos hacer más amigable el programa para el usuario, por ejemplo:

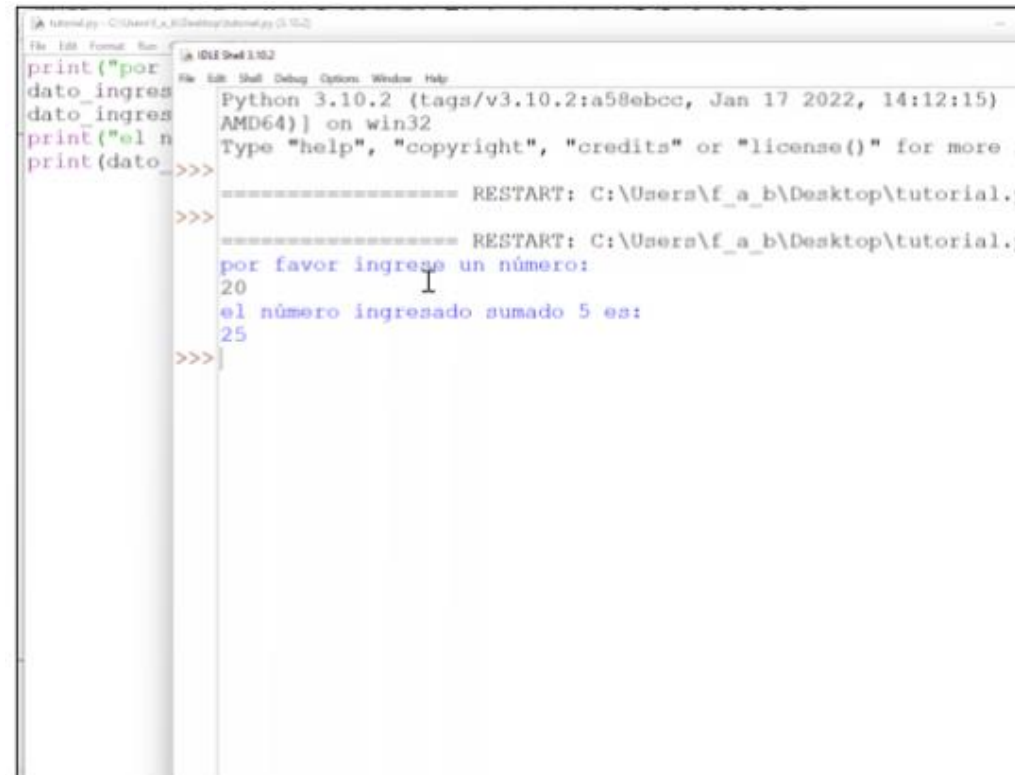
```
print("por favor ingrese un número: ")

dato_ingresado = input()

dato_ingresado = dato_ingresado + 5

print("el número ingresado sumado 5 es: ")

print(dato_ingresado)
```



```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15)
AMD64) on win32
Type "help", "copyright", "credits" or "license()" for more
>>>
===== RESTART: C:\Users\f_a_b\Desktop\tutorial.py
>>>
===== RESTART: C:\Users\f_a_b\Desktop\tutorial.py
por favor ingrese un número:
20
el número ingresado sumado 5 es:
25
>>>
```

8. Podemos también mostrar alguna frase al usuario a través de `input()`, esto nos ahorrará algunos `print`, de esta manera:

```
dato_ingresado = input("por favor ingrese un número: ")  
  
dato_ingresado = dato_ingresado + 5  
  
print("el número ingresado sumado 5 es: ")  
  
print(dato_ingresado)
```

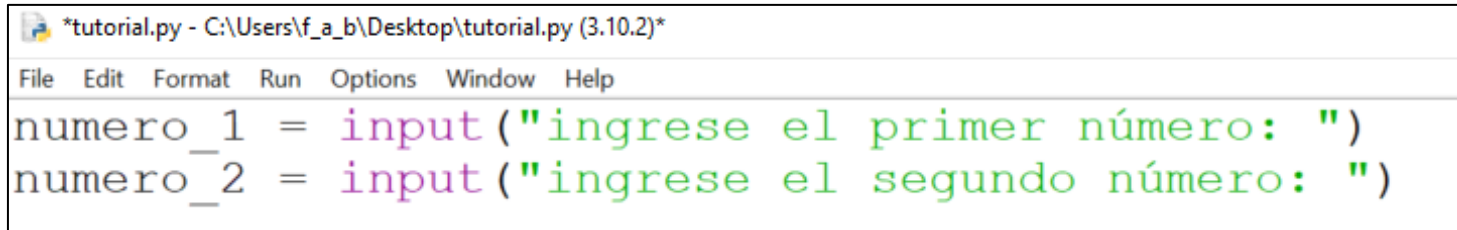


```
IDLE Shell 3.10.2  
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64-bit AMD64] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:\Users\f_a_b\Desktop\tutorial.py =====  
>>>  
por favor ingrese un número:10  
el número ingresado sumado 5 es:  
15  
>>>
```

1.11. Ejercicio final

Con todo lo que hemos visto, podemos hacer el siguiente ejercicio: pedir al usuario dos números enteros, calcular la suma y mostrar el 30% del resultado de la suma:

1. Lo primero que debemos hacer es solicitarle al usuario que ingrese dos números enteros de la siguiente manera:

A screenshot of a Python IDE window titled '*tutorial.py - C:\Users\f_a_b\Desktop\tutorial.py (3.10.2)*'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code area contains two lines of Python code: 'numero_1 = input("ingrese el primer número: ")' and 'numero_2 = input("ingrese el segundo número: ")'. The code is color-coded: 'input' is in purple, and the strings are in green.

```
*tutorial.py - C:\Users\f_a_b\Desktop\tutorial.py (3.10.2)*
File Edit Format Run Options Window Help
numero_1 = input("ingrese el primer número: ")
numero_2 = input("ingrese el segundo número: ")
```

2. Luego debemos convertir estos números a int:

```
numero_1 = int(numero_1)  
numero_2 = int(numero_2)
```

3. Ahora realizamos la suma de los dos números así:

```
suma = numero_1 + numero_2
```

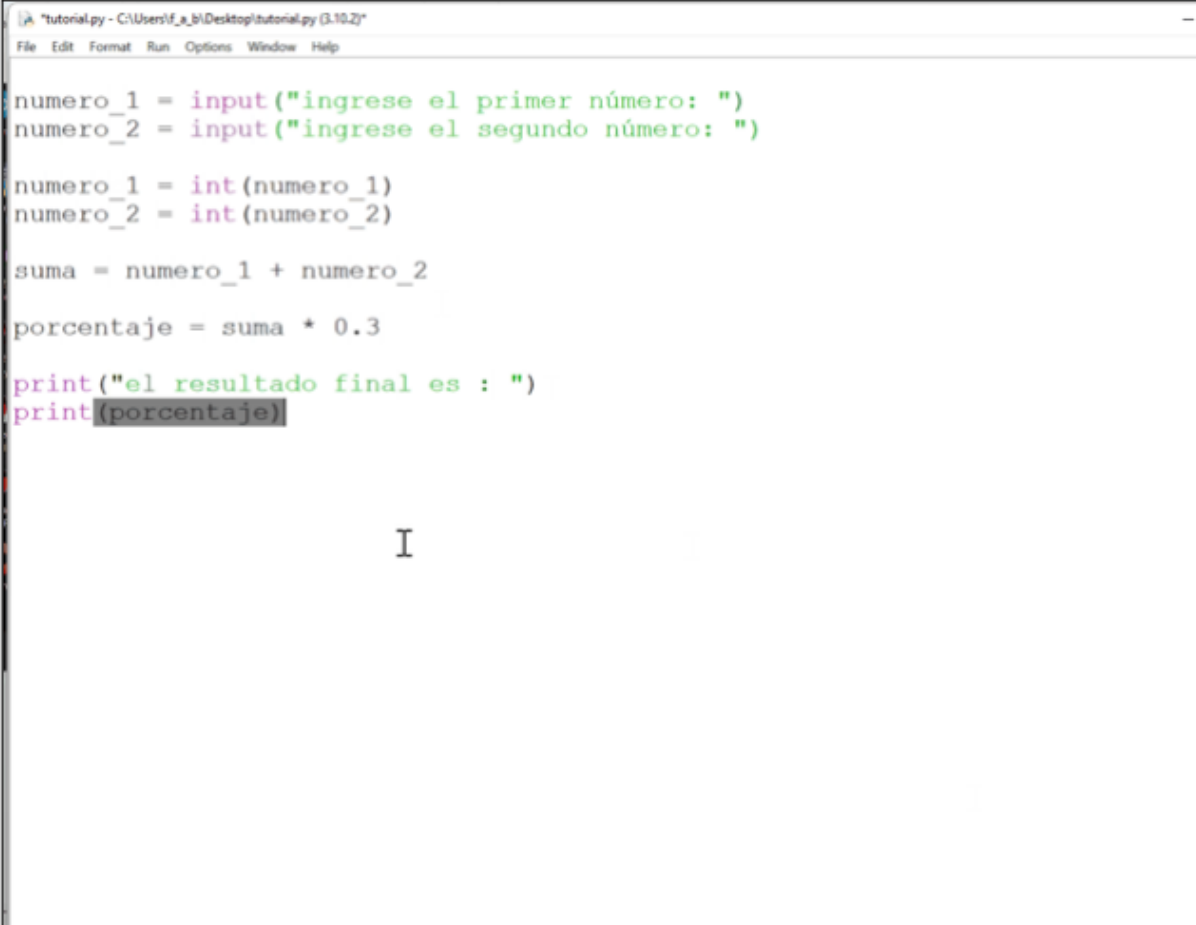
4. Debemos calcular el 30% del resultado de la suma:

```
porcentaje = suma * 0.3
```

5. Podríamos mostrar el resultado de la siguiente manera:

```
print("el resultado final es: ")  
print(porcentaje)|
```

6. Finalmente la ejecución:



```
numero_1 = input("ingrese el primer número: ")
numero_2 = input("ingrese el segundo número: ")

numero_1 = int(numero_1)
numero_2 = int(numero_2)

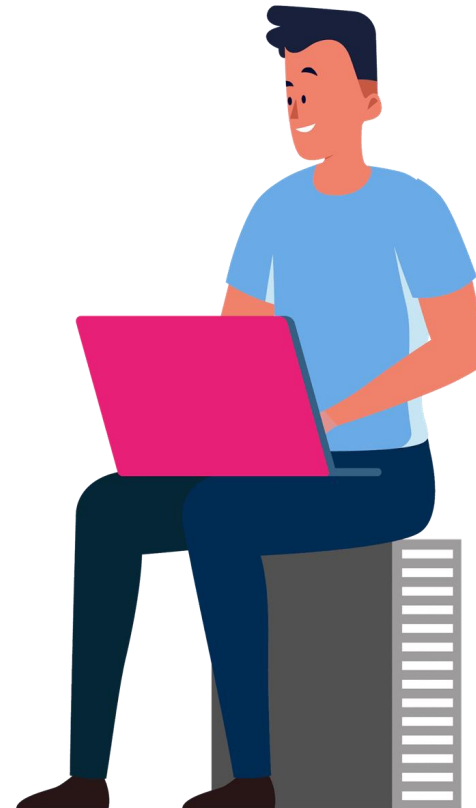
suma = numero_1 + numero_2

porcentaje = suma * 0.3

print("el resultado final es : ")
print(porcentaje)
```



```
numero_1 = input("ingrese el primer número: ")  
numero_2 = input("ingrese el segundo número: ")  
  
numero_1 = int(numero_1)  
numero_2 = int(numero_2)  
  
suma = numero_1 + numero_2  
  
porcentaje = suma * 0.3  
  
print("el resultado final es : ")  
print(porcentaje)
```



Referencias bibliográficas

[1]Python, (s.f.) Recuperado de: <https://www.python.org/about/>

[2]MALIK, F. (2019). Todo sobre Python: principiante a avanzado. Recuperado de: <https://medium.com/fintechexplained/everything-about-python-from-beginner-to-advance-level-227d52ef32d2>

[3]Principales idiomas informáticos. (2020) Recuperado de: <https://statisticstimes.com/tech/top-computer-languages.php>

[4]M. Kamaruzzaman, (2020). "Top 10 In-Demand programming languages to learn in 2020." [Online]. Recuperado de: <https://towardsdatascience.com/top-10-in-demand-programming-languages-to-learn-in-2020-4462eb7d8d3e>

[5]Python.org, "What is Python? Executive Summary." [Online]. Recuperado de: <https://www.python.org/doc/essays/blurb/>

[6] W3schools, "Introduction to Python." [Online]. Recuperado de: <https://www.w3schools.com/python/pythonintro.asp>

[7] DataFlair, "16 Facts about Python Programming that every Geek should know." [Online]. Recuperado de: <https://data-flair.training/blogs/facts-about-python-programming/>

[8] Ocado Group, "Python overtakes French as the most popular language taught in primary schools." [Online]. Recuperado de: <https://www.ocadogroup.com/media/press-releases/year/2015/python-overtakes-french-most-popular-language-taught-primary-schools>.

Material de estudio complementario

[Curso Python desde cero #6 | Operadores aritméticos en Python - YouTube](#)

[8. Programación en Python | Operadores Lógicos - YouTube](#)





El futuro digital
es de todos

MinTIC

Hechos

QUE

CONECTAN



CICLO 1

EJE TEMÁTICO 1

FUNDAMENTOS DE PROGRAMACIÓN EN PYTHON

Universidad
Industrial de
Santander



Misión
TIC 2022