INTRODUCCIÓN A PYTHON

Introducción a la programación en Python por freeCodeCamp

Este tutorial lo he realizado sobre un magnifico tutorial realizado por Estefania Cassingena Navone.

PERE MANEL VERDUGO ZAMORA
pereverdugo@gmail.com

¿Por qué aprender a programar?

"Todo el mundo debería aprender a programar, porque te enseña a pensar", Steve Jobs (1995).

Cuando programamos resolvemos un problema y lo traducimos a una serie de pasos que un ordenador será capar de ejecutar.

Python es un lenguaje de programación que por sintaxis es ideal para aquellos que se quieren iniciar al mundo de la programación.

Un lenguaje de programación con reglas bien definidas que nos permite escribir una serie de instrucciones que puedan ser leídas y ejecutadas por un ordenador.

Python se utiliza en ciencia de datos, inteligencia artificial, aprendizaje automático, desarrollo Web y en enseñanza de programación.

Se utiliza en diversas áreas, desarrollo de juegos, en la medicina, en ciencias puras (Biología) y Astronomía entre otras.

Ventajas de Python:

- Fácil de aprender.
- Sintaxis clara y sencilla.
- Poderoso.
- Aplicaciones reales.
- Alta demanda.

Pregunta:

¿Verdadero o Falso?

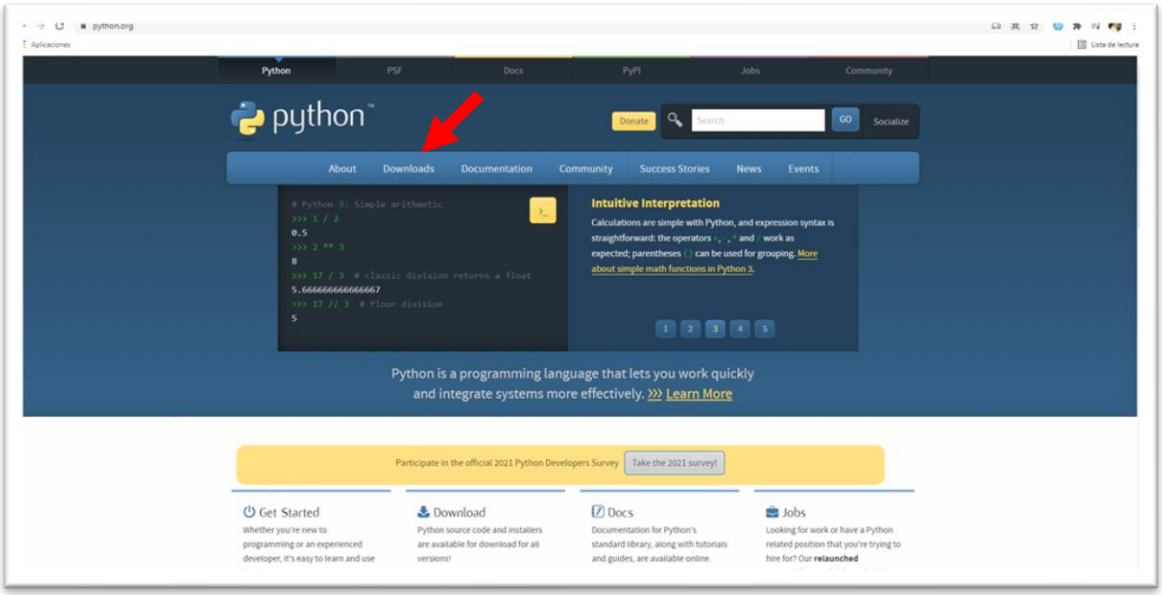
Un lenguaje de programación como Python nos permite escribir instrucciones que pueden ser leídas y ejecutadas por un ordenador?

A) Verdadero
B) Falso

Instalar Python

Lo primero que tenemos que hacer es ir a la página donde podremos descargar Python.

https://www.python.org/

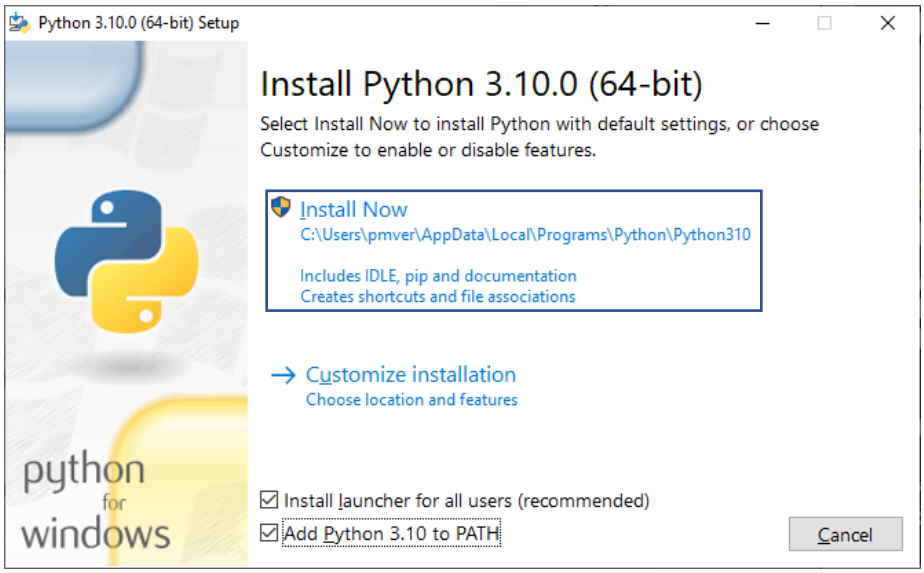


Buscaremos Downloads (Descargas).

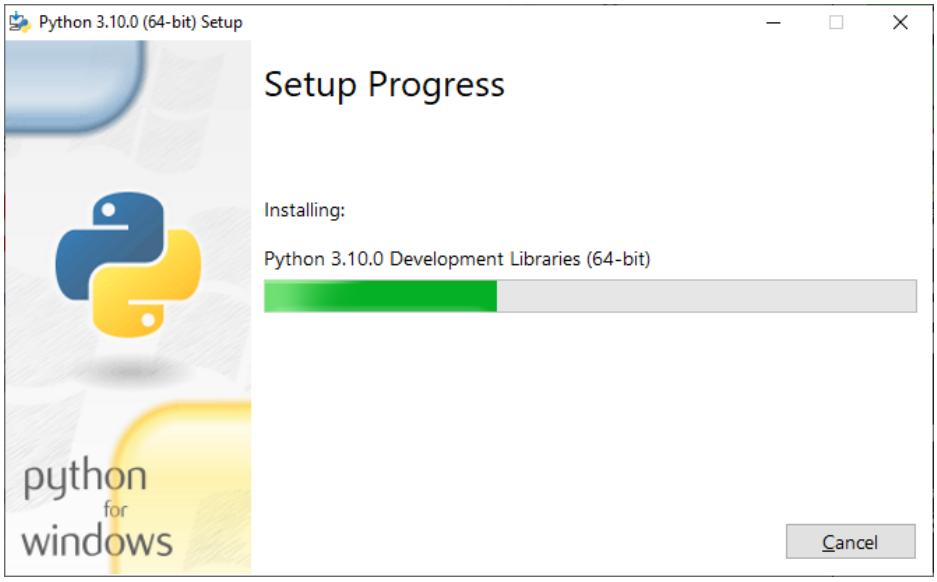
Si tu sistema operativo es Windows ya lo puedes descargar en dicho botón, si tienes otro sistema operativo en la parte izquierda lo puedes seleccionar.

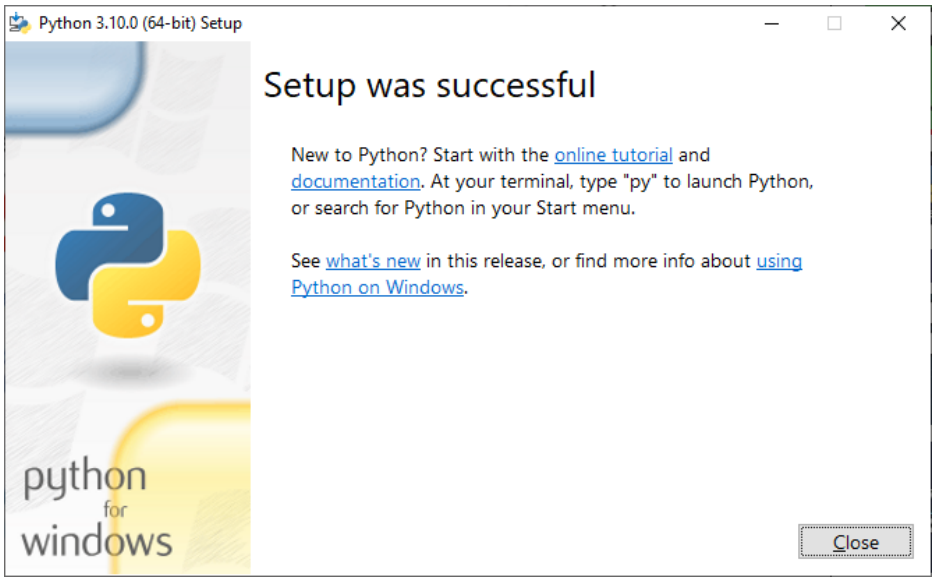
Una vez que hemos realizado la descarga tendremos este archivo.

Ya lo podemos ejecutar para realizar su instalación.



Déjalo como muestra la imagen y Seleccionamos “Install Now” la opción más recomendable.





Introducción a IDLE.

Entorno integrado de programación que se instala automáticamente cuando instalas Python.

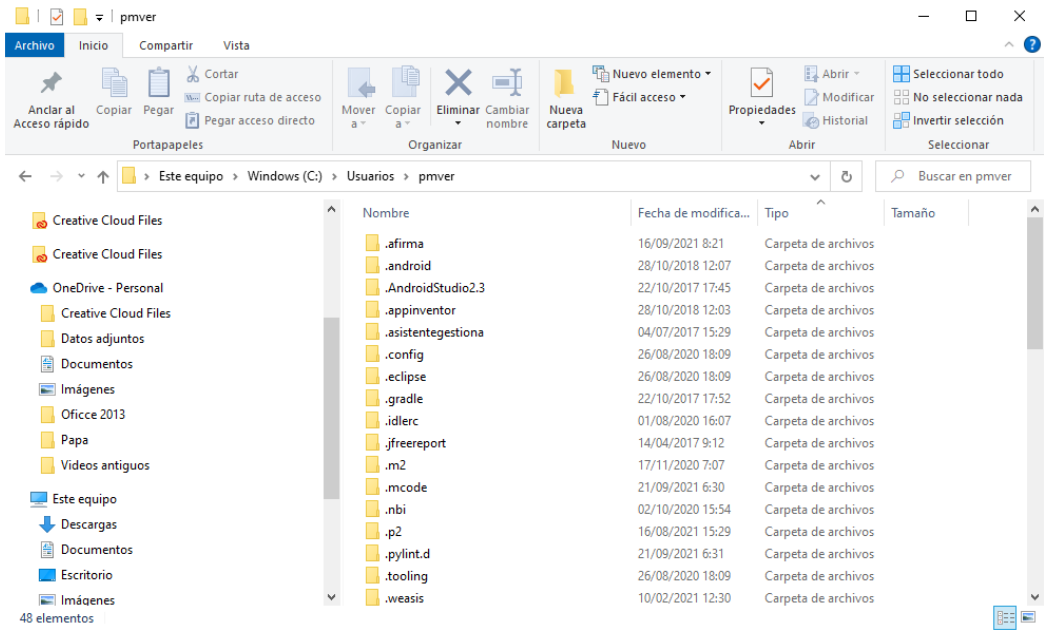
Con IDLE Puedes:

- Escribir
- Editar
- Ejecutar

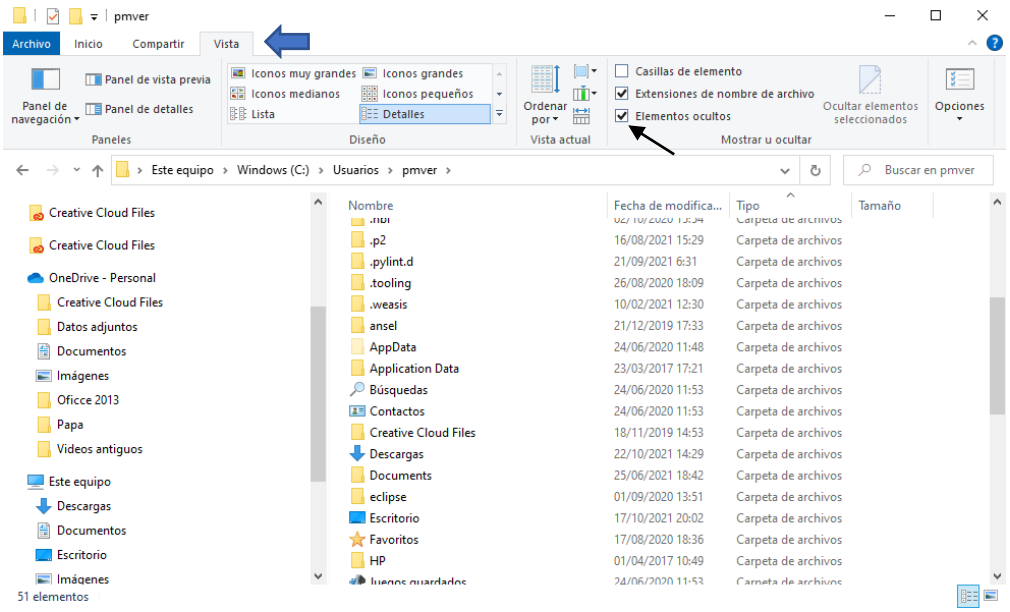
} Código

Vamos a buscar el archivo.

Con el administrador de archivos nos ubicaremos en C:\Users\pmver



Queremos buscar el siguiente directorio AppData, pero este está oculto.

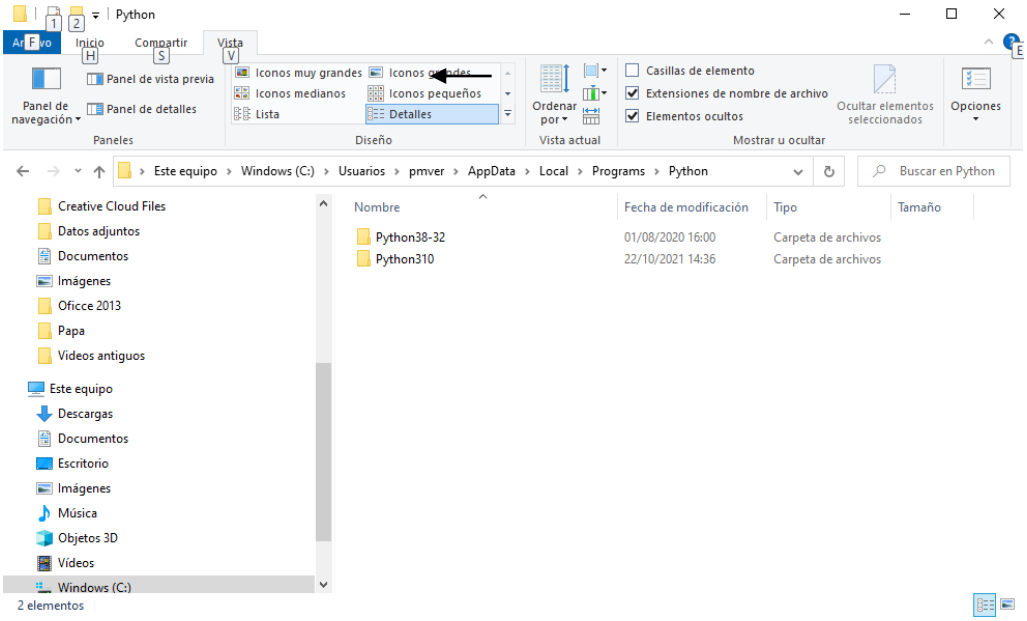


Seleccionamos la pestaña Vista y activamos la casilla “Elementos ocultos”.

Esta carpeta ya está visible lo único que se ve un poco más transparente.

Seguimos con el recorrido:

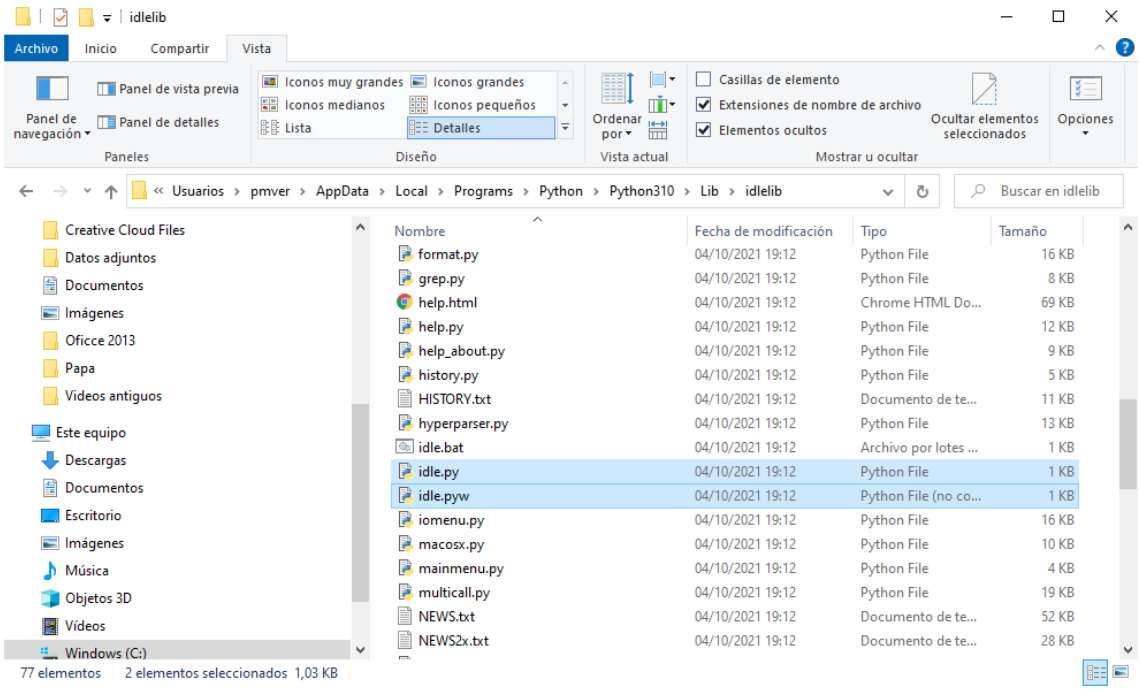
C:\Users\pmver\AppData\Local\Programs\Python



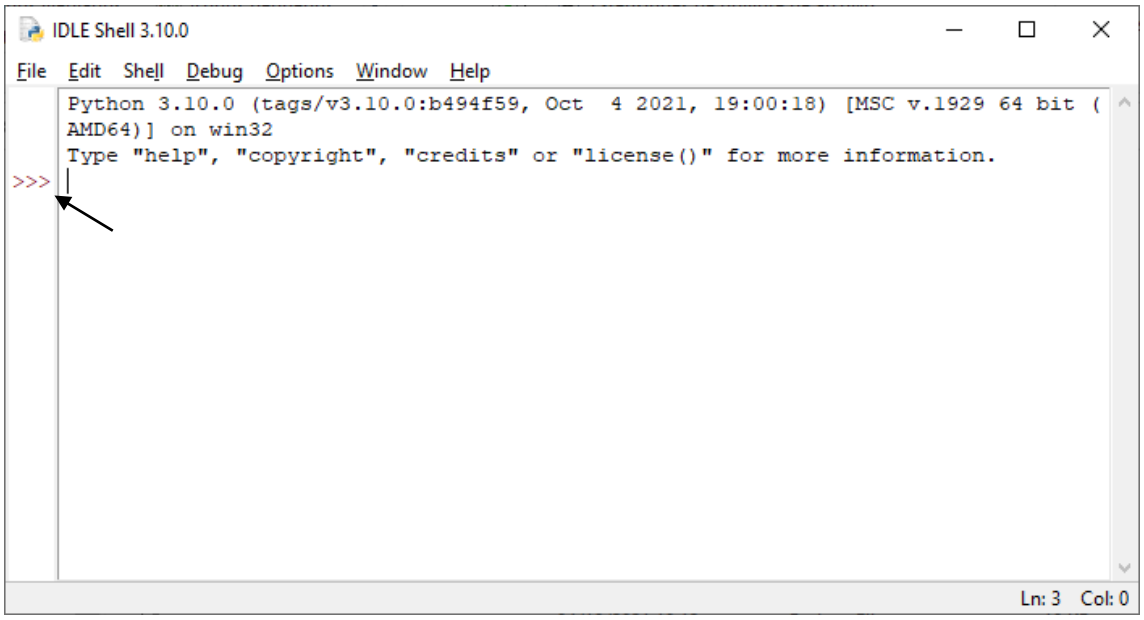
Se pueden tener varias versiones instaladas, la última y más reciente que he instalado es la Python310, versión 3.10.

Seguimos la ruta:

C:\Users\pmver\AppData\Local\Programs\Python\Python310\Lib\idlelib



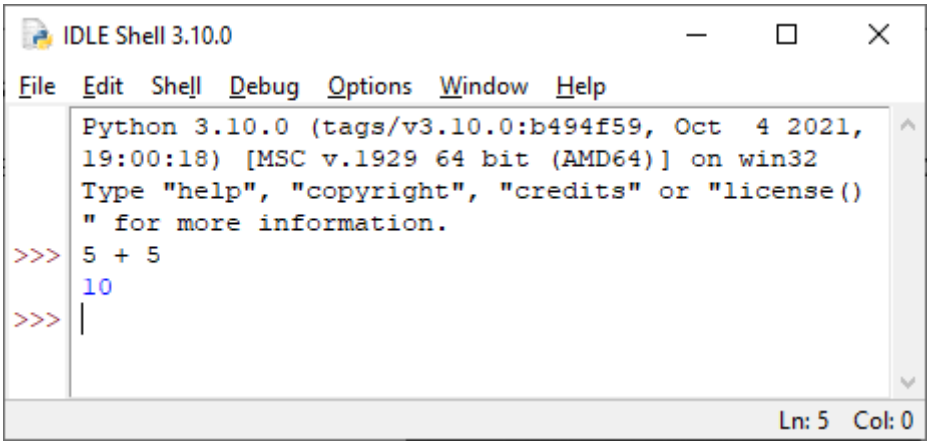
De los dos vamos a ejecutar el primero.



Esta será una de nuestras herramientas principales, la consola.

Si observamos >>> esto nos indica que la consola ya está preparada.

Vamos a introducir nuestro primer comando que será la suma de 5 + 5, al hacer intro nos mostrará el resultado.

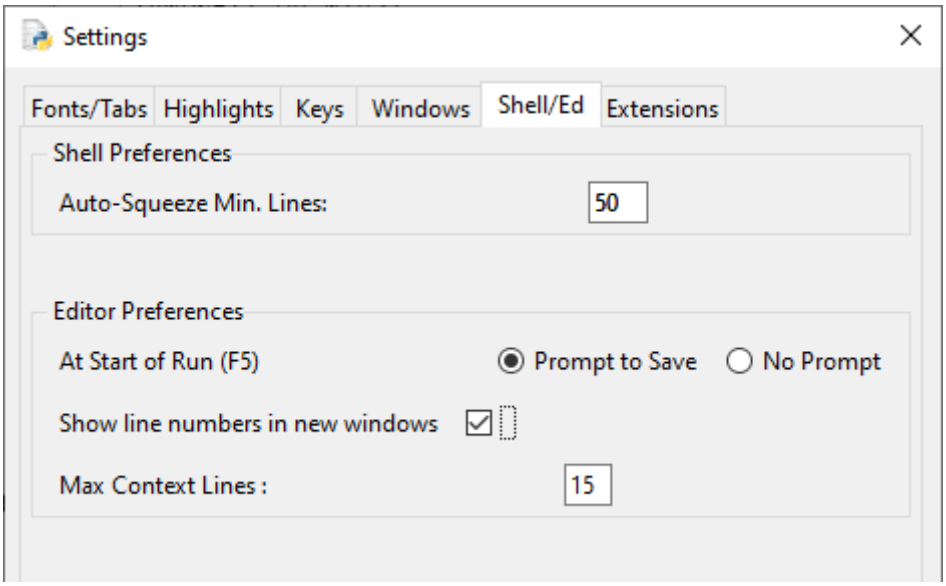


El resultado se mostrará de color azul.

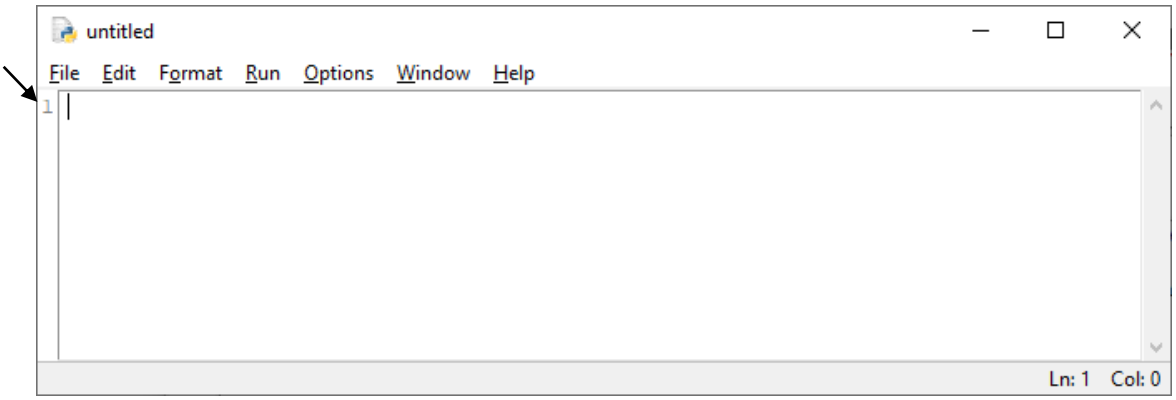
Antes de empezar un nuevo proyecto vamos a realizar la siguiente configuración.

Del menú Options seleccionaremos Configure IDLE.

Seleccionaremos la pestaña Shell/Ed y activaremos la opción “Show line number in new Windows” que nos muestre el número de línea cuando estemos programando.

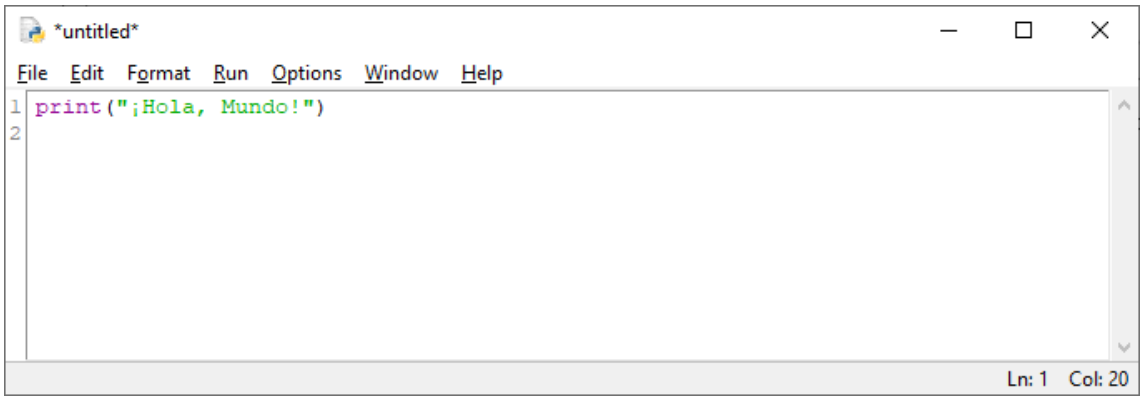


Vamos a crear nuestro primer archivo, del menú File seleccionaremos New File.

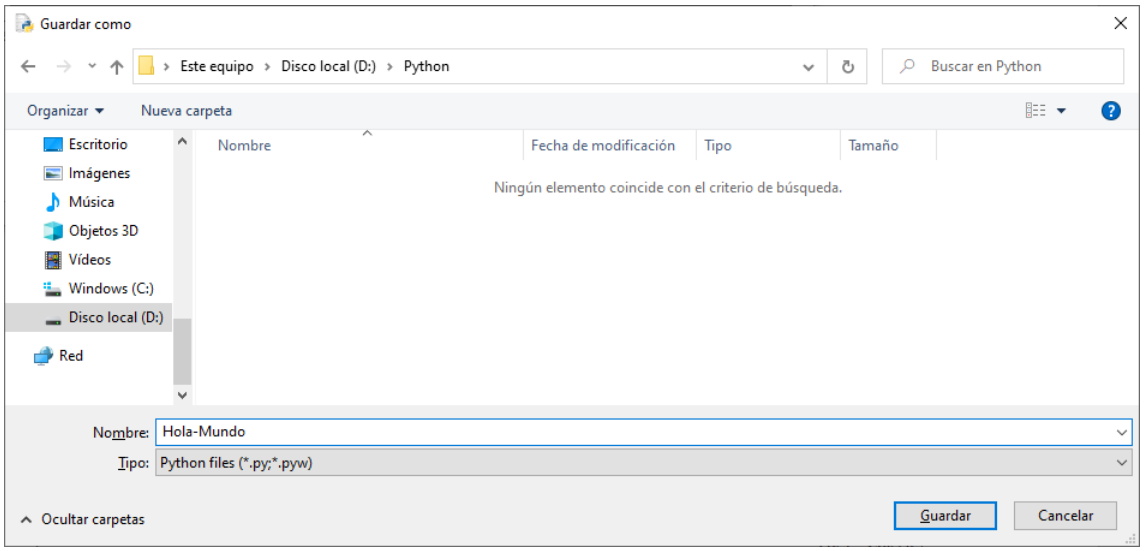


Si seleccionamos del menú la opción Run observaremos la opción Run Module (F5) que nos permitirá ejecutar el proyecto que estamos realizando.

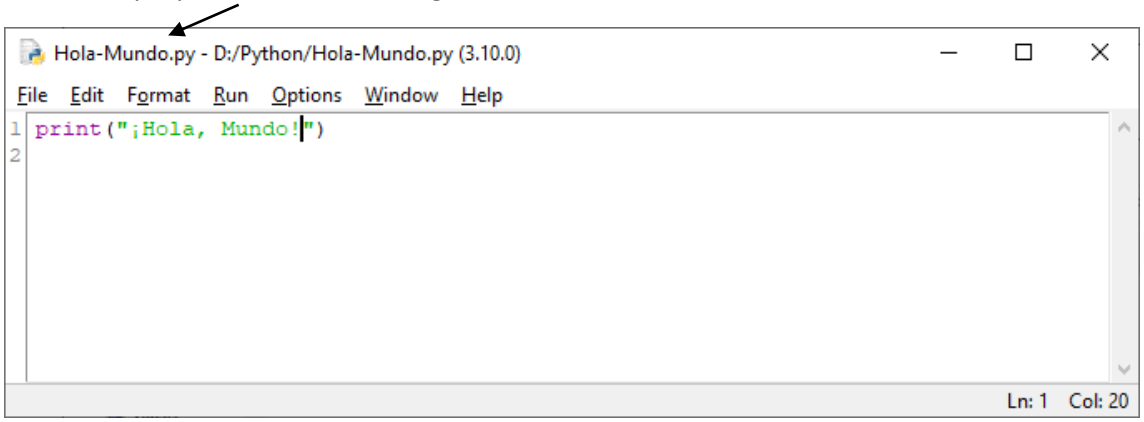
Tu primer programa



A continuación vamos a guardar el archivo, de menú File seleccionaremos Save.



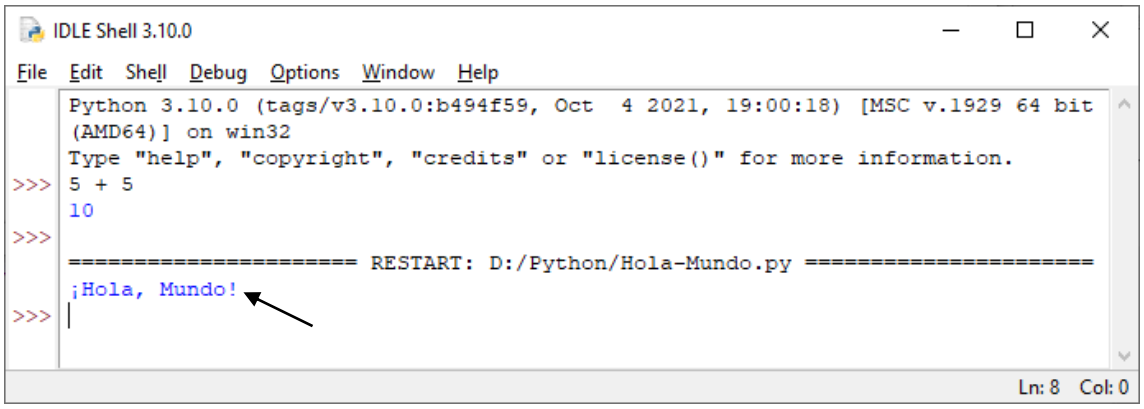
Seleccionaremos la carpeta donde queremos guardar nuestros proyectos y le pondremos nombre al proyecto Hola-Mundo seguido de Guardar.



Ahora nos muestra el nombre del proyecto y nos dice con su extensión que es un archivo Python.

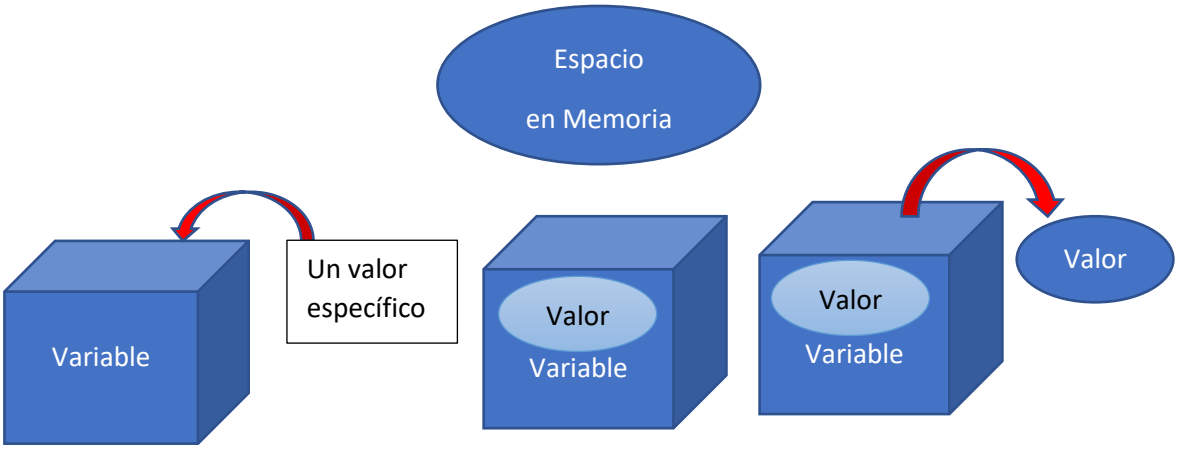
Ahora seleccionaremos del menú Run y de este Run Module, también podemos hacer F5.

Podemos ver el resultado en nuestra consola de Python.


IDLE Shell 3.10.0
File Edit Shell Debug Options Window Help
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 5 + 5
10
>>>
========== RESTART: D:/Python/Hola-Mundo.py ==========
¡Hola, Mundo!
>>>
Ln: 8 Col: 0

Variables

Variable es el nombre que se le asigna a un valor en el programa.



Una variable es un valor que se reserva en el espacio de la memoria que tiene un nombre y en cualquier momento podemos recuperar su valor utilizando dicha variable.

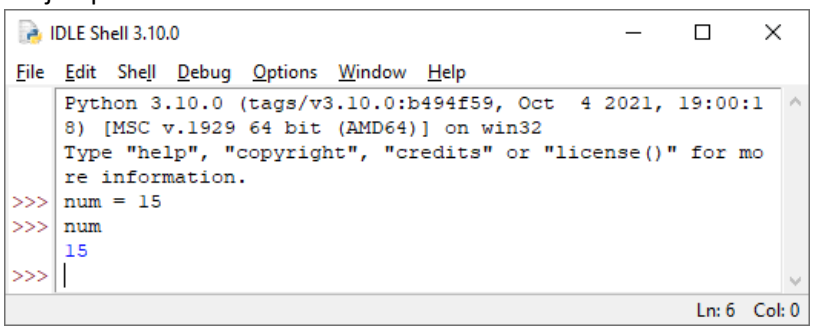
Esta variable se podrá ir actualizando durante la ejecución del programa.

Asignación:

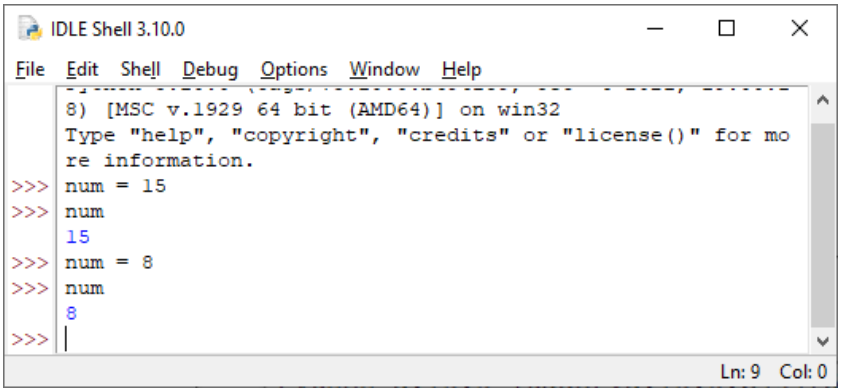
num = 5

Nombre de la variable num y su valor 5.

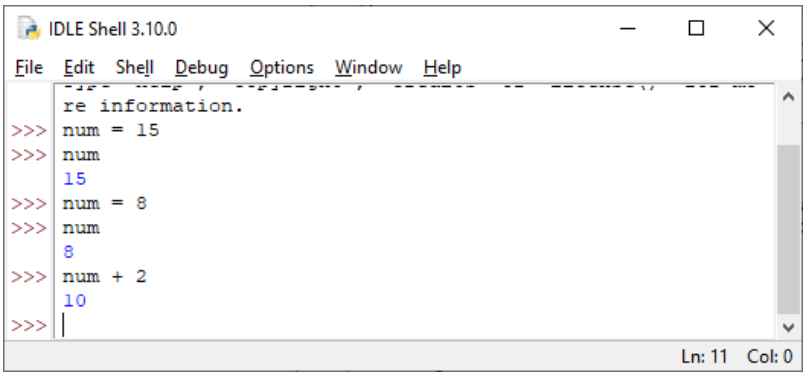
Vamos a ver un ejemplo desde la consola.



En este ejemplo vemos como asignamos a la variable num el valor 15, para poder ver el valor del mismos solo tienes que escribir el nombre de la variable seguido de enter.



Si queremos modificar el valor de la variable solo hay que asignarle otro valor.



Podemos utilizar la variable para sumarle otro valor.

Reglas para nombrar variables

Los nombres de variables deben empezar con una letra o un guion bajo.

mi_varaible o _mi_variable como ejemplos correctos.

9mi_variable no es correcto empieza con un número, este será el mensaje de error que saldrá.

SyntaxError: invalid syntax.

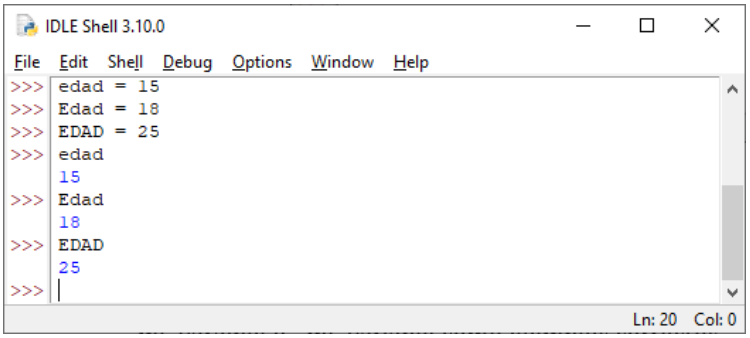
Las variables solo pueden tener caracteres alfanuméricos de (A-Z, a-z, 0-9 y gión bajo _).

Num_visitas = 12442 es un ejemplo correcto, cuando una variable tiene varias palabras la podremos separar guion bajo.

@variable = 34 es un ejemplo no correcto ya que el primer carácter la arroba no está permitida.

Se distinguen entre mayúsculas y minúsculas.

edad, Edad y EDAD son tres variables distintas.



Pregunta:

Selecciona el/los nombre€ válido(s) para una variable:

a) cantidad$_en_inventario
b) 0cantidad_en_inventario
c) cantidad_en_inventario
d) _cantidad_en_inventasrio

Tipos de datos

Numéricos:

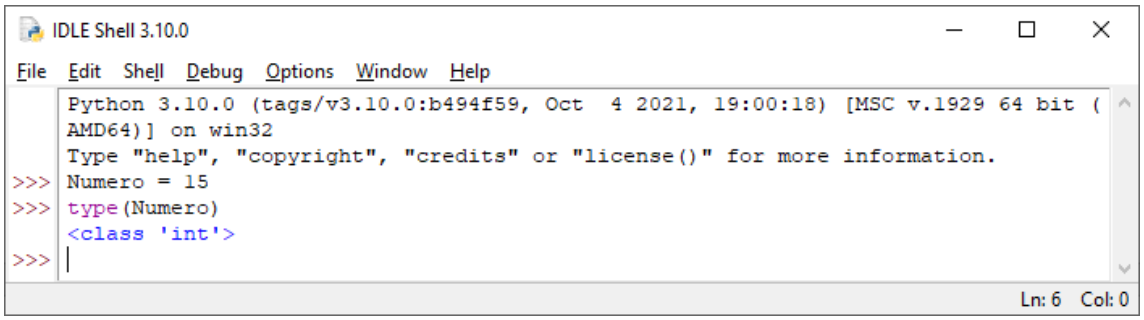
Entero

Números que no tienen decimales.

- Positivos
- Negativos
- Cero

Ejemplo: 5, -3, 0, 3, -8, 15, -1, 27 y -16.

Si usamos la función type() nos dirá que tipo de variable es.



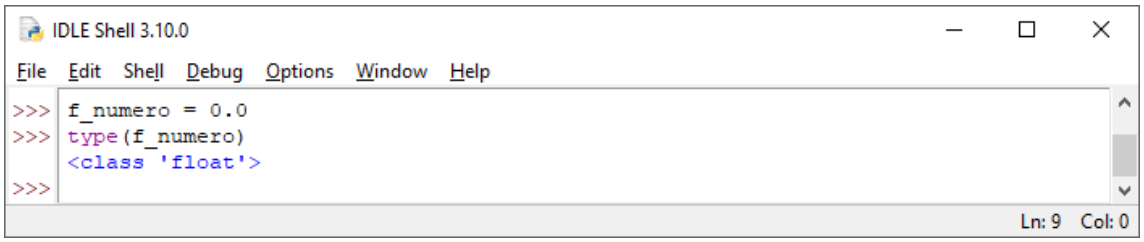
int de Integer (Entero).

En Python no existe un limite para el número máximo.

Números en Coma Flotante

Números que tienen decimales

- Positivos
- Negativos



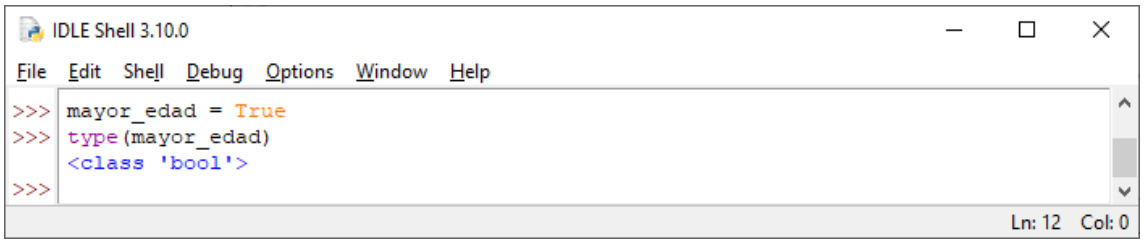
Booleanos

True (Verdadero)	False (Falso)

Importante para:

- Expresiones condicionales
- Ciclos (Bucles)

Recuerda que las palabras True y False son palabras reservadas que no podrás utilizar como nombre de variables.

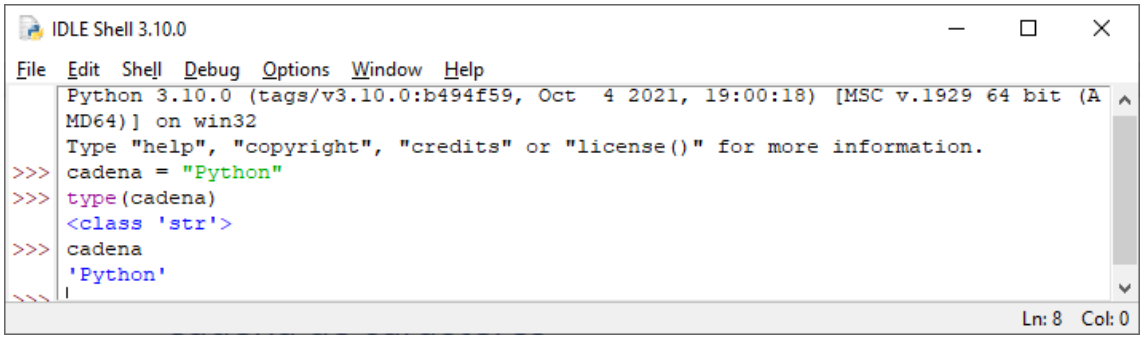


Cadena de caracteres

Cadena de caracteres (String)

Secuencia de caracteres encerrados entre comillas y usados para representar texto en el programa.

Ejemplo de una cadena de caracteres en Python → “Python” o ‘Python’.



Cuando queremos ver el valor de la cadena esta la muestra con comillas simples.

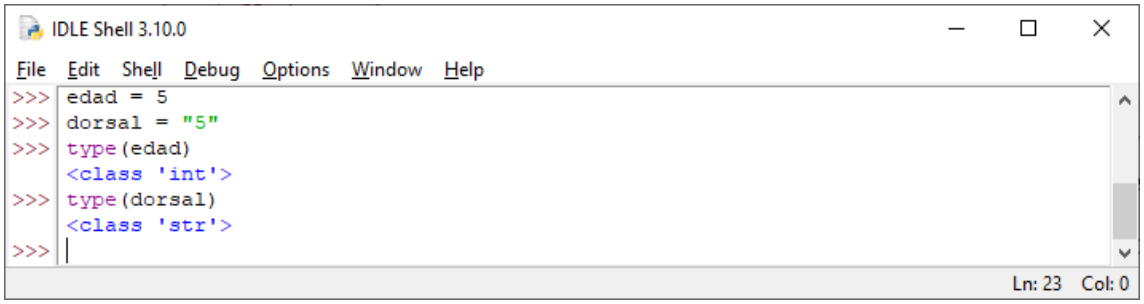
Lo que no se puede hacer es esto “Python’ o ‘Python”, tienes que utilizar siempre las mismas al inicio como al fin.

Esto puede provocar el siguiente error:

SyntaxError: EOL while scanning string literal.

Entero		Cadena de caracteres
5	≠	“5”

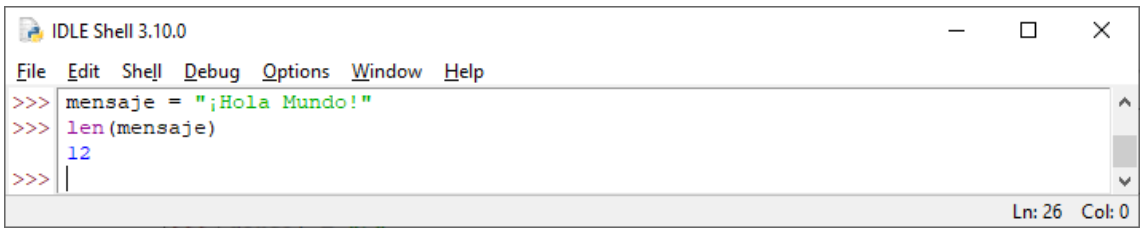
Un número entero no es igual que un número que es parte de una cadena de caracteres.



Con la variable int podrás realizar operaciones aritméticas en cambio con la de tipo str no.

Tamaño (Length): Número de caracteres que posee.

Esto se ve con la función len().

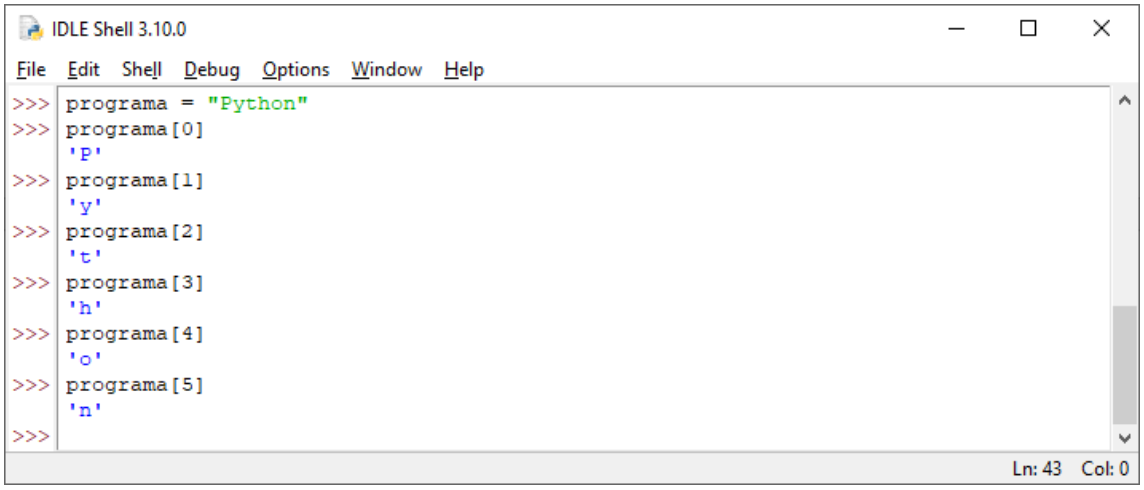


Estructura de una cadena de caracteres.

“	P	y	t	h	o	n	“
	0	1	2	3	4	5	

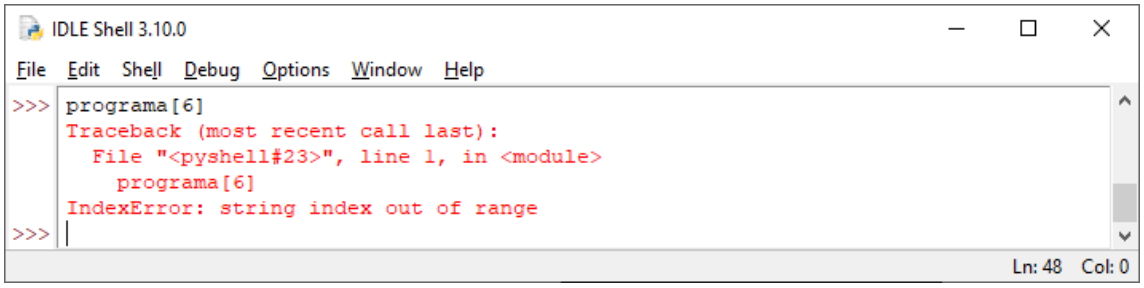
Índices

Indexación (Indexing)



Con el nombre de la variable más su índice podemos extraer el carácter.

Si intentamos acceder a un índice que no existe tendremos el siguiente error.



IndexError: Índice de cadena de caracteres fuera de rango.

Pregunta:

Selecciona el carácter ubicado en el índice 6 de esta cadena de caracteres:

“¡Hola, Mundo!”

a) “a”
b) “,”
c) “ “
d) “M”

Selecciona el carácter ubicado en el índice 8 de esta cadena de caracteres:

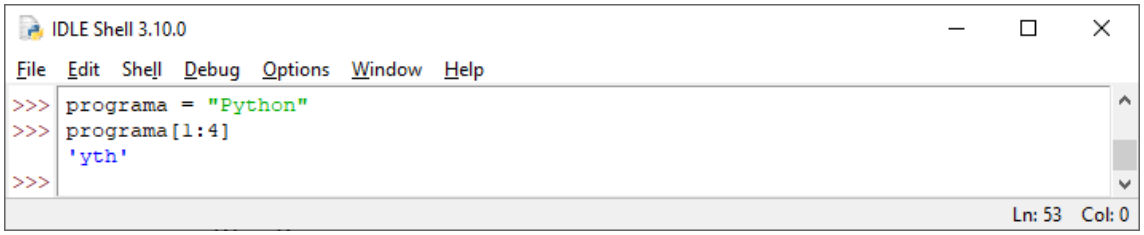
“¡Hola, Mundo!”

a) “o”
b) “M”
c) “u”
d) “d”

Rebanado (Slicing)

El rebanado nos permite obtener una rebanada (porción) de una cadena de caracteres.

De la cadena “Python” queremos extraer yth.



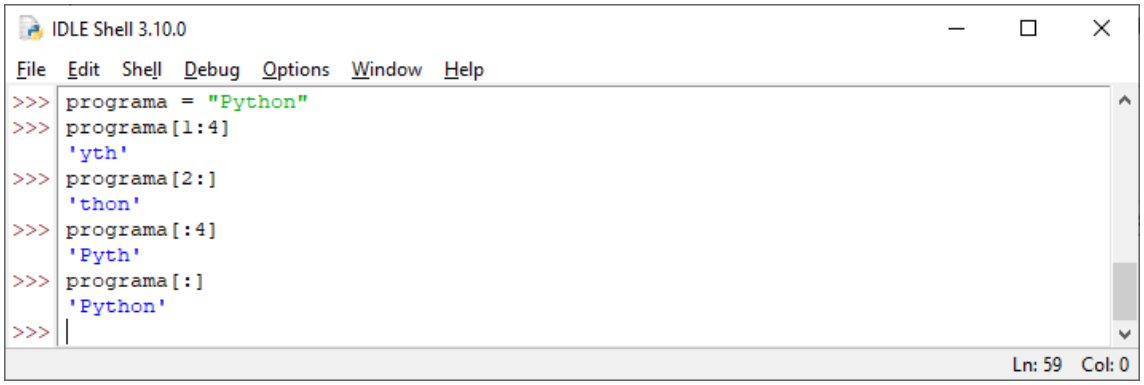
Variable y entre corchetes índice inicial : índice final (este último no lo emprime)].

Otras combinaciones:

programa[inicio:] Extrae desde la posición inicio hasta el final.

programa[:fin] Extrae desde el inicio hasta la posición fin -1.

Programa[:] Extrae toda la cadena.



Selecciona la rebanada resultante:

frase = “¡Hola, Mundo!”

frase[7:12]

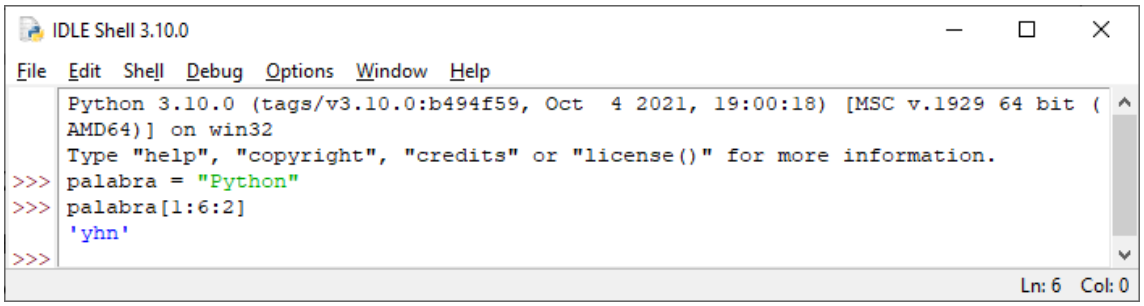
a) Hola
b) Mundo
c) , Mund
d) Mundo!

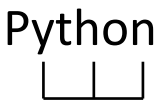
Hay un tercer parámetro que podemos utilizar.

<cadena>[inicio:fin:**paso**]

paso: Cómo “saltar” de un carácter al siguiente” si no se pone nada por defecto es 1.

Supongamos que queremos extraer de Python los caracteres subrayados, Python.





Pregunta:

Selecciona la rebanada resultante:

frase = “¡Hola, Mundo!”

frase[7:12:2]

a) Hla
b) MnD
c) Mno
d) Mdo

Métodos:

Los métodos de las cadenas de caracteres son operaciones comunes que vienen implementadas en Python.

¿Como se ejecutan?

<cadena>.<método>(<valores>)

Capitalize: Retorna una copia de la cadena con el primer carácter en mayúsculas y el resto en minúsculas.

<cadena>.capitalize()

Otros métodos importantes:

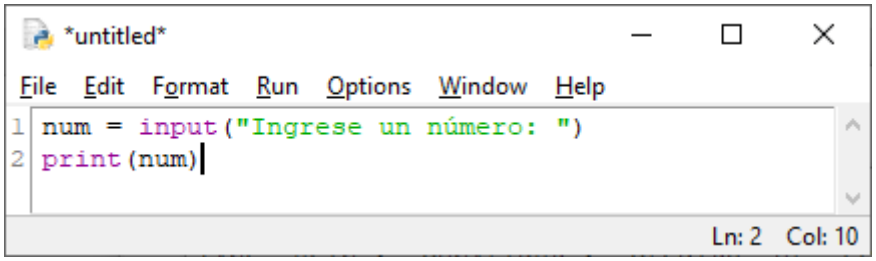
- find: Busca en que posición está un determinado carácter. palabra.find(“t”)
- index: Dice en que posición esta un determinado carácter. palabra.index(“t”)
- isalnum: controla si solo tiene caracteres alfanuméricos.
- Isalpha: controla si solo tiene caracteres alfabéticos.
- Isdecimal: controla si solo tiene caracteres decimales.
- Isdigit: controla si solo tiene dígitos.
- Islower: controla si solo contiene caracteres en minúscula.
- Isupper: controla si solo contiene caracteres en mayúscula.
- lower: retorna una copia en minúsculas.
- upper: retorna una copia en mayúsculas.

Todos aquellos métodos que empiezan por “is” nos devuelve un valor booleano, True o False.

Recibiendo datos del usuario

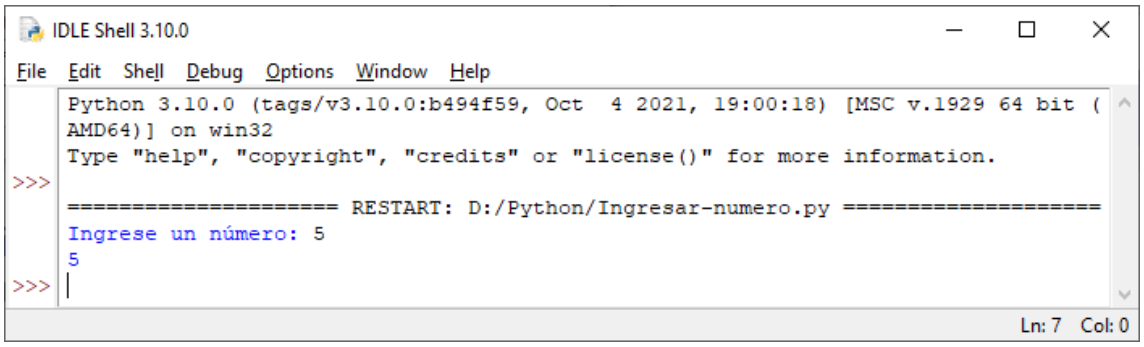
input(): Para poder introducir información por teclado.

<var> = input(<mensaje>)



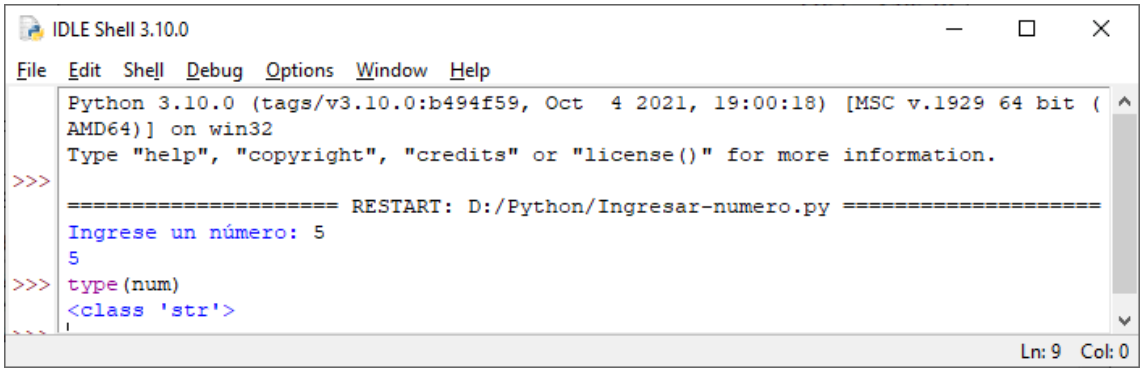
Vamos a guardar el archivo.

Y con F5 lo vamos a ejecutar.

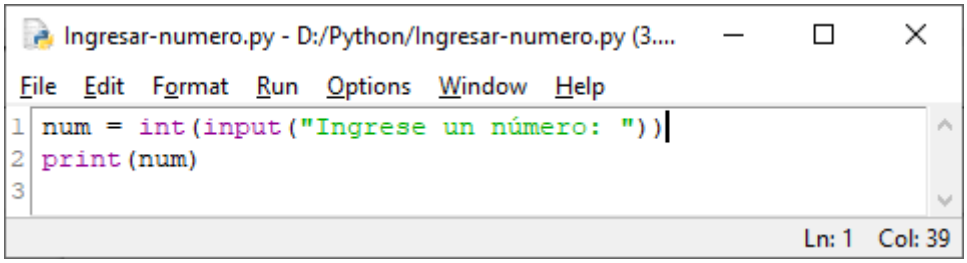


Advertencia:

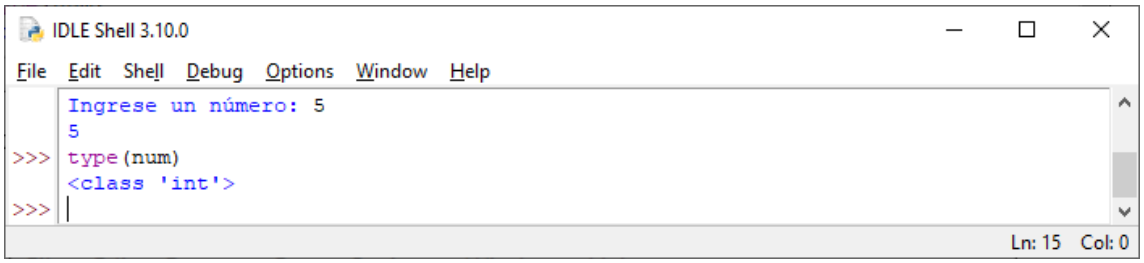
La función input() siempre retorna una cadena de caracteres.



Para conseguir que sea un número lo hemos de reconvertir en entero con int.



Lo guardamos y lo ejecutamos.



Ahora sí que es un valor entero.

Pregunta:

Selecciona el tipo de dato de la variable 'valor':

>>> valor = input("Ingrese 'True' o 'False': ")

Ingrese 'True' o 'False': True

a) Entero
b) Número de Coma Flotante
c) Cadena de caracteres
d) Booleano

Operadores

Un operador es un símbolo que realiza una operación.

Valores con los cuales se ejecuta la operación.

Operador + Operandos = Expresión

Una expresión es una combinación de valores, variables, y operadores que al ser evaluados resultan un valor.

Las expresiones se evalúan de izquierda a derecha, excepto cuando ciertos operadores tienen mayor prioridad para el orden de las operaciones. Esto se denomina jerarquía.

Operadores:

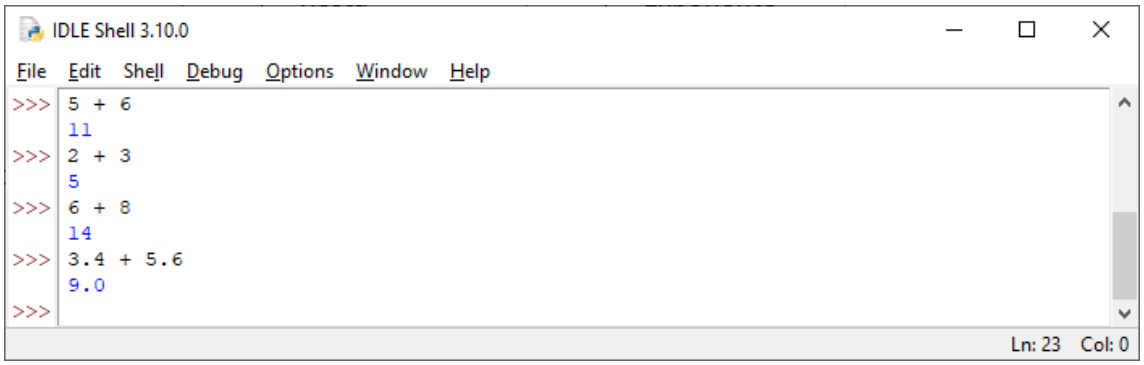
- Aritméticos
- Lógicos
- De Asignación
- Relacionales

Operadores aritméticos:

Operadores Aritméticos nos permiten realizar operaciones aritméticas en el programa.

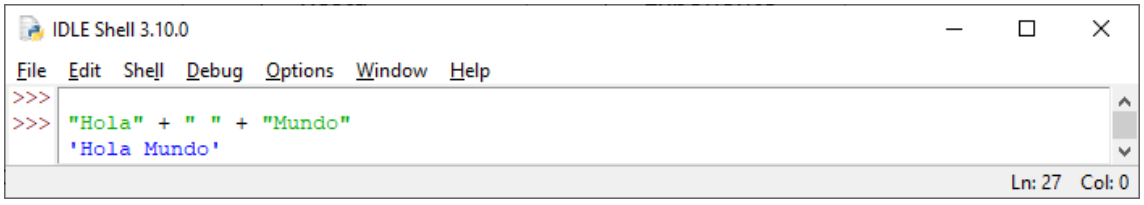
- Suma
- Resta
- Multiplicación
- División
- División Entera
- Exponente
- Módulo

Operador Suma:

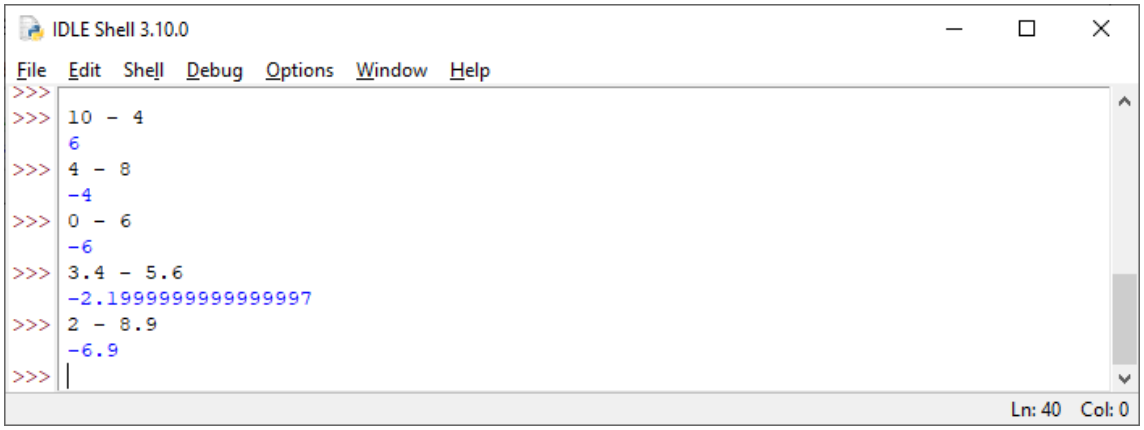


Se pueden sumar enteros y números de coma flotante.

Este operador también se puede utilizar para concatenar cadena de caracteres.

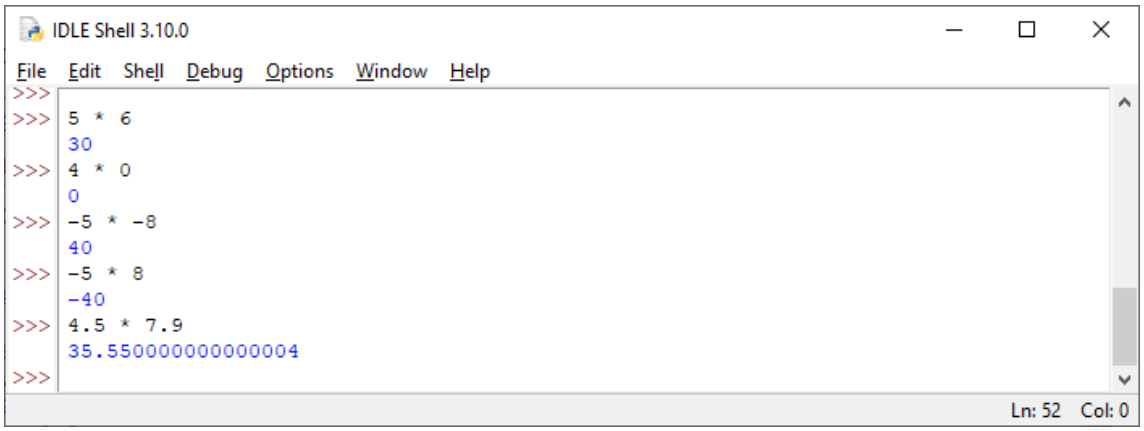


Operador resta:



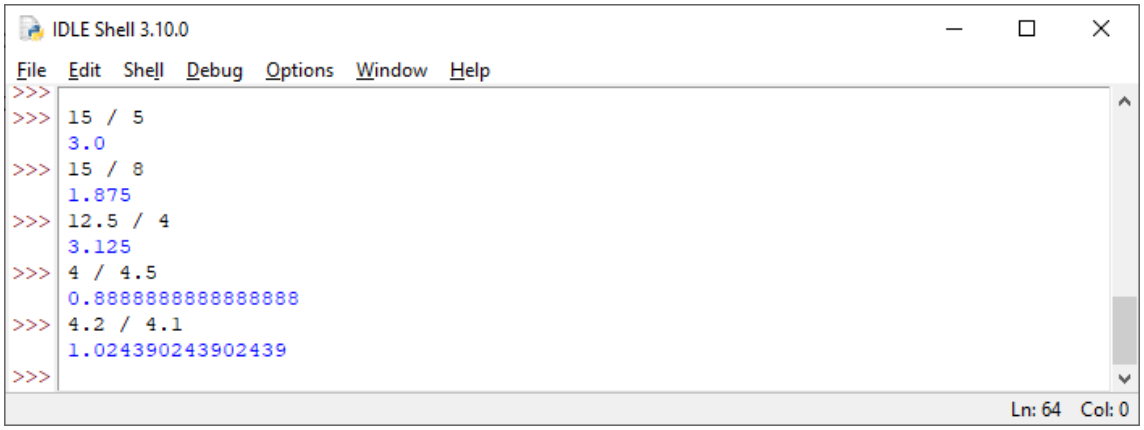
Tanto para valores enteros como de coma flotante.

Operador multiplicación:



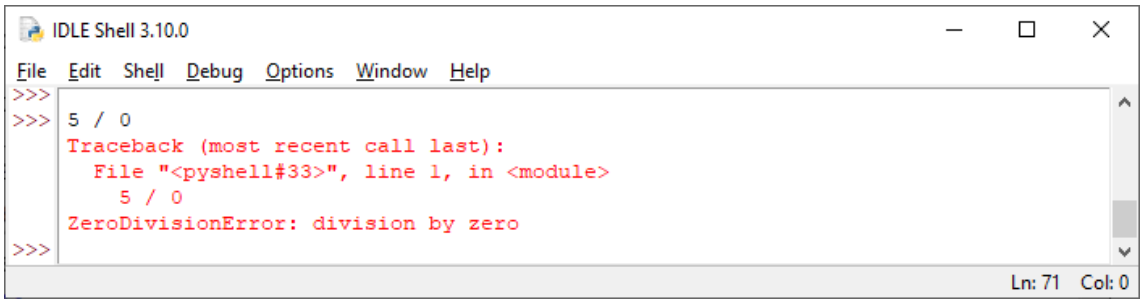
Tanto para valores enteros como de coma flotante.

Operador división:

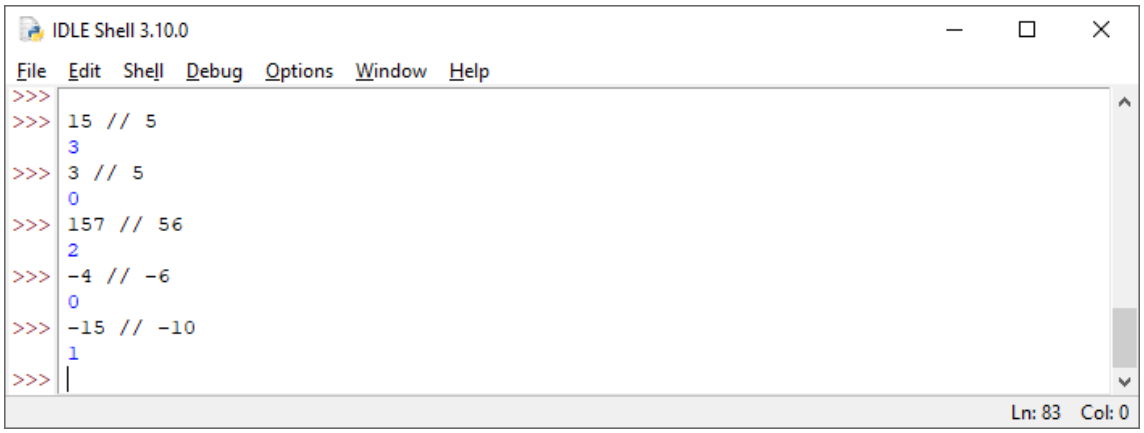


Tanto para valores enteros como de coma flotante. El resultado siempre será un número en coma flotante.

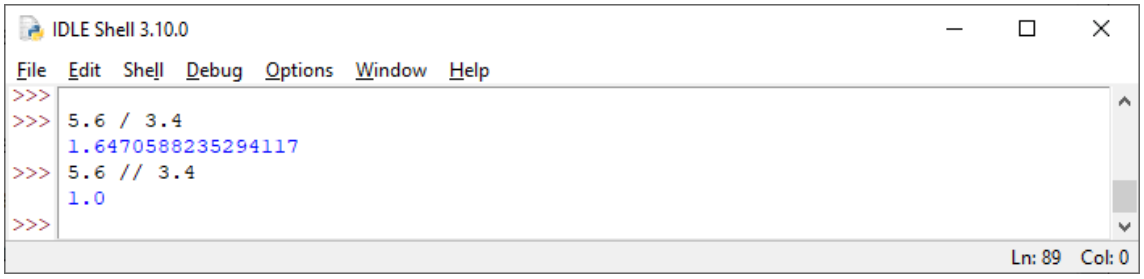
Que pasa si intentamos dividir por 0.



Error de división por Cero.
Operador División entera.

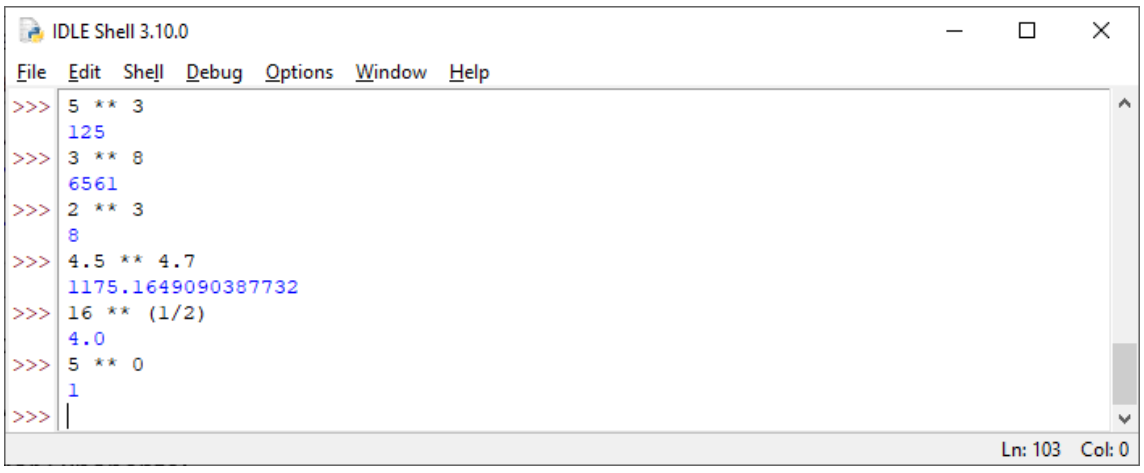


El operador es '// ' la doble barra inclinada.
En este caso si retorna un valor entero (int).
Este operador elimina toda la parte decimal, no redondea.



Aquí podemos ver la diferencia de una división a una división entera.
Si la división entera es de dos número de coma flotante retorna un valor del mismo tipo.

Operador Exponente:



Si los valores son enteros retorna un valor entero y si los valores son de coma flotante retorna un valor de coma flotante.

En el ejemplo 16 ** (1/2) hace lo mismo que la raíz cuadrada.

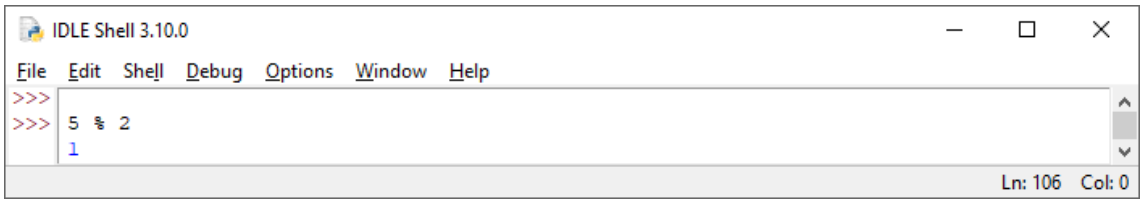
El ejemplo 5 ** 0 es igual a 1 cualquier número elevado a 0 es igual a la unidad.

Operador módulo:

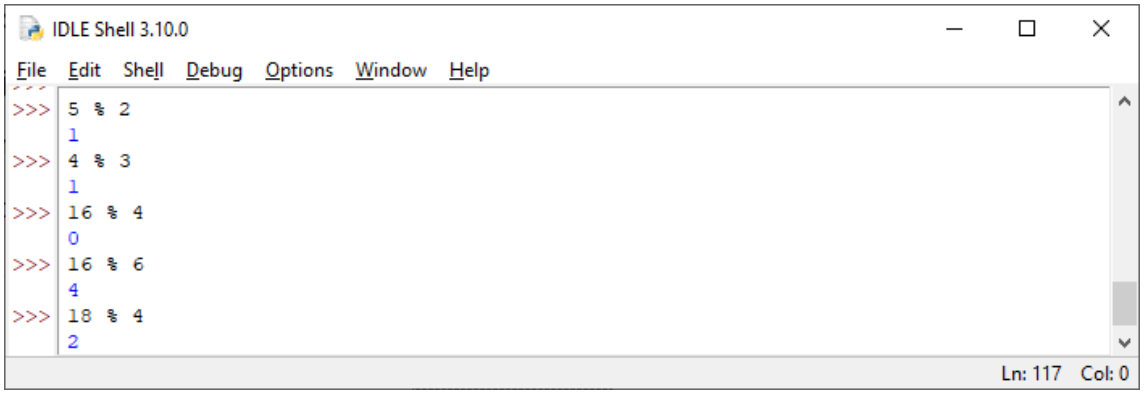
Retorna el resto de la división. Es utilizado para verifica si un número es par o impar.

5	2
1	2

Si dividimos 5 entre 2 nos dará una resultado de 2 con un resto de 1. Este es el valor que nos va a retornar el operador módulo.



Según este valor será 0 o 1 podremos deducir que este numero es par o impar.



Otros ejemplos.

Como se determina el orden de una operación (jerarquía).

PEMDAS: Paréntesis, Exponentes, Multiplicación, División, Sumas, Restas, de la misma jerarquía las operaciones se realizan de izquierda a derecha.

Operadores Lógicos:

Nos permiten trabajar con valores booleanos (True y False).

and or not

Operador and evalúa si el operando izquierdo y el derecho son verdaderos.

x and y

Si ambas expresiones x e y son verdaderas con el operador and al final también será verdadera.

X	Y	X and Y
True	True	True
True	False	False
False	True	False
False	False	False

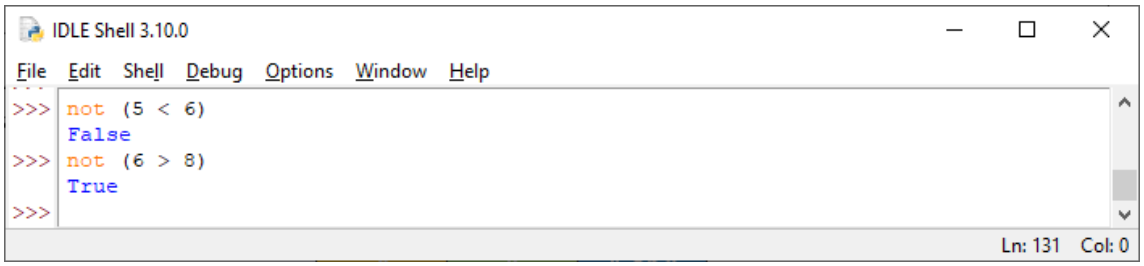
Como una de las dos condiciones es falsa el final también es falsa, ya que el operado and obliga que las dos condiciones sean verdaderas.

x or y

X	Y	X or Y
True	True	True
True	False	True
False	True	True
False	False	False

not x

x	not x
True	False
False	True



>>> not True → False

>>> not False → True

Prioridad:

- not
- and
- or

Mayor

Menor

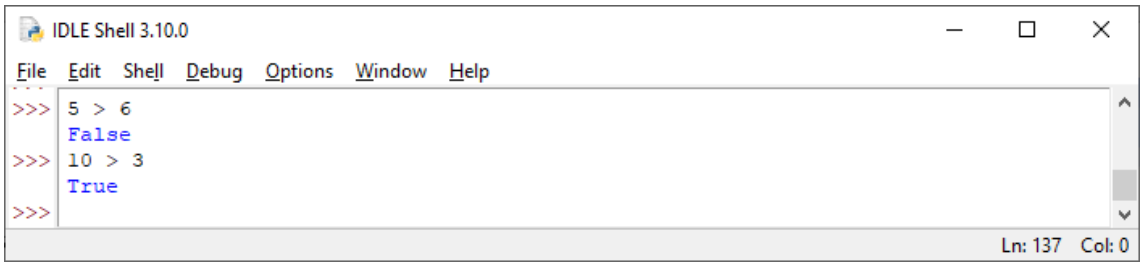
Si existen varios operadores iguales la prioridad es de izquierda a derecha.

Operadores relacionales:

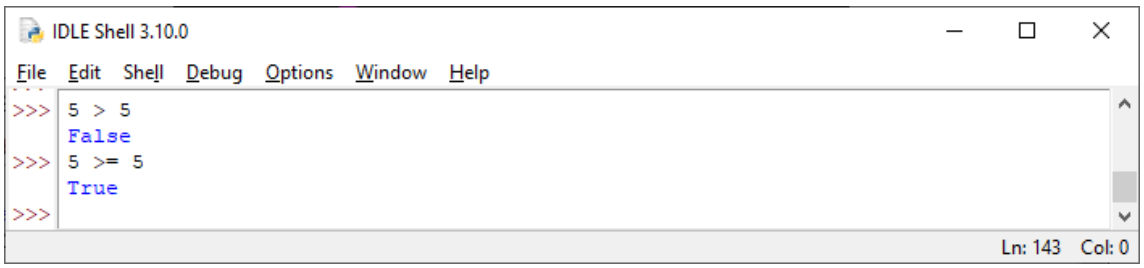
Son utilizados para comparar valores y retornar un valor booleano.

>	mayor	>=	mayor igual
<	menor	<=	menor igual
==	igual	!=	distinto

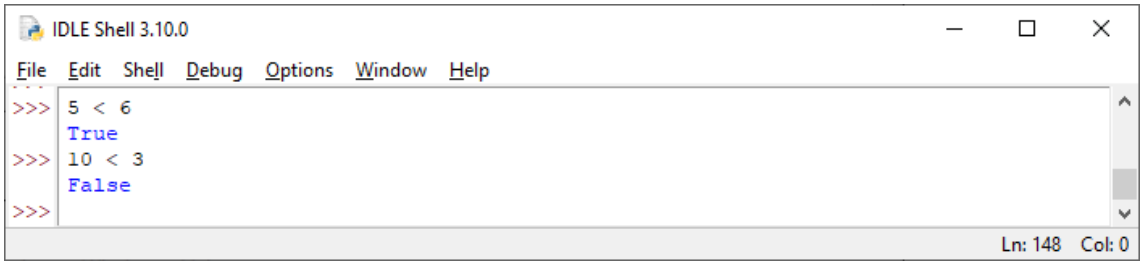
Con el operador mayor.



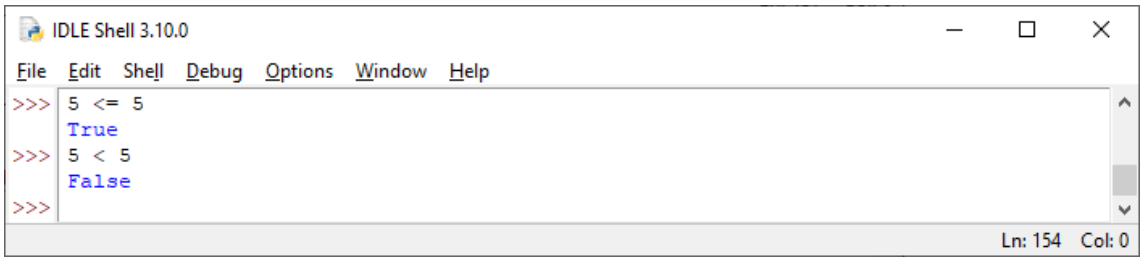
La diferencia de mayor y mayor igual.



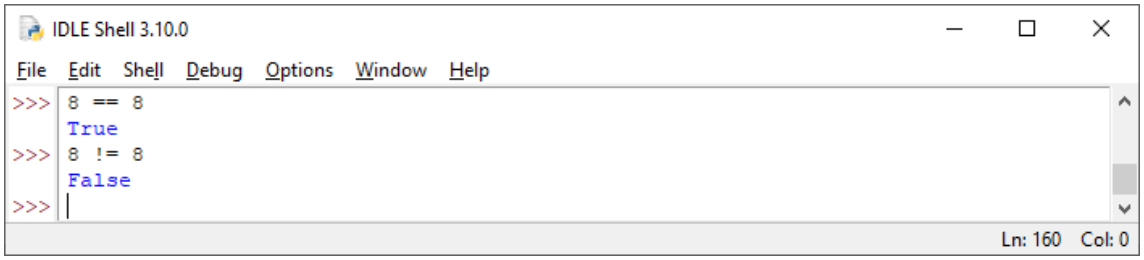
Con el operador menor.



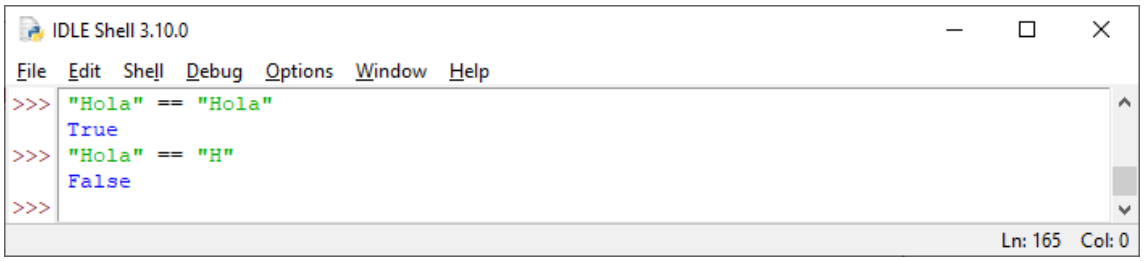
La diferencia entre el operador menor y el operador menor igual que.



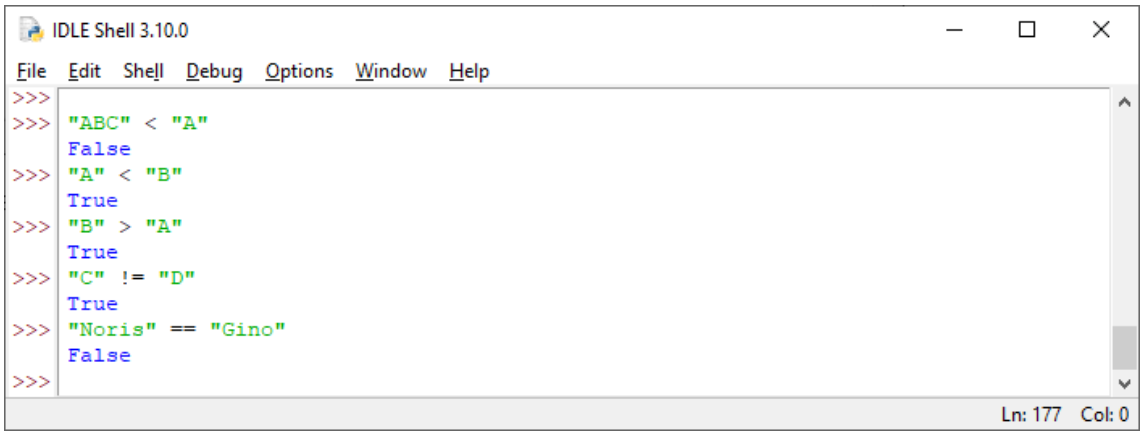
La diferencia entre igual y distinto.



Comparar cadenas de texto.



Para comparar el orden de las cadenas de texto, el orden que tienen las palabras en un diccionario (orden alfabético).

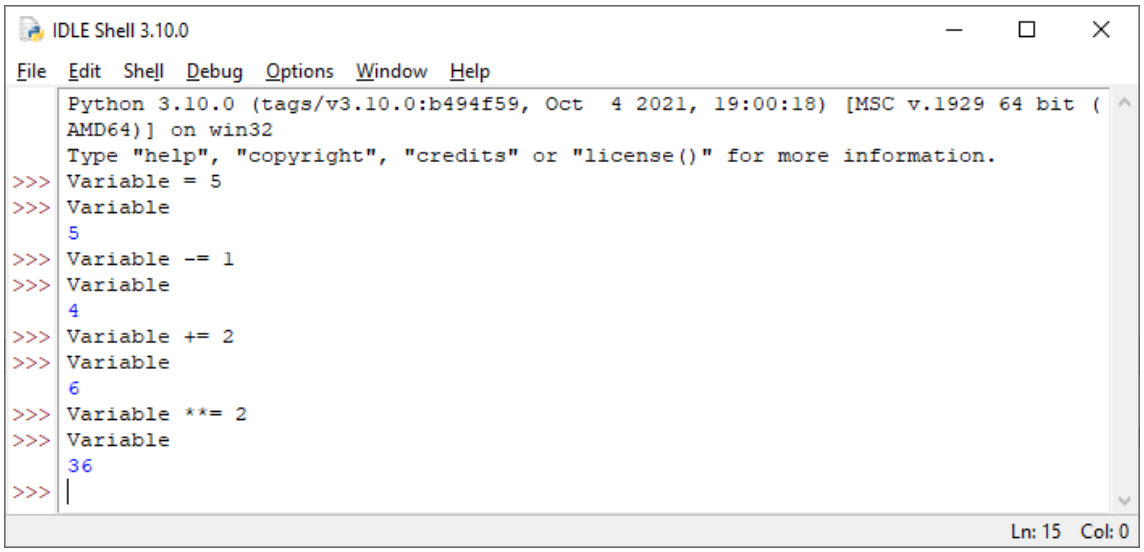


En los programas vamos a sustituir los valores por variables que contienen valores.

Operadores de asignación:

Son utilizados para asignar valores a variables del programa.

=	Variable = 5	+=	Variable += 1
-=	Variable -= 1	*=	Variable *= 5
/=	Variable /=5	**=	Variable **= 2
//=	Variable //=5	%=	Variable %= 6



Algunos ejemplos.

Pregunta:

Selecciona el valor final de num:

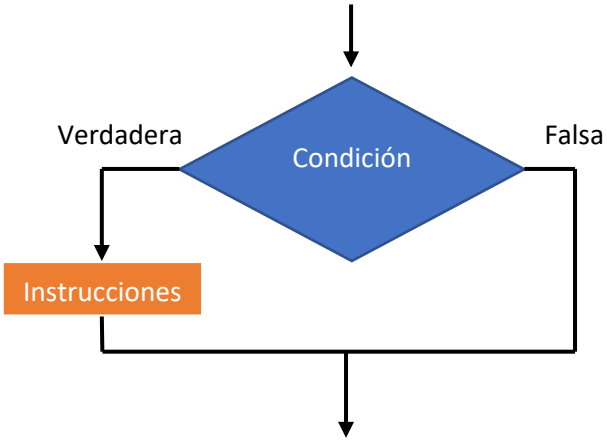
>>> num = 6

>>> num %= 2

a) 3
b) 2
c) 1
d) 0

Sentencias condicionales

Instrucción o un grupo de instrucciones cuya ejecución depende del valor de una condición booleana.

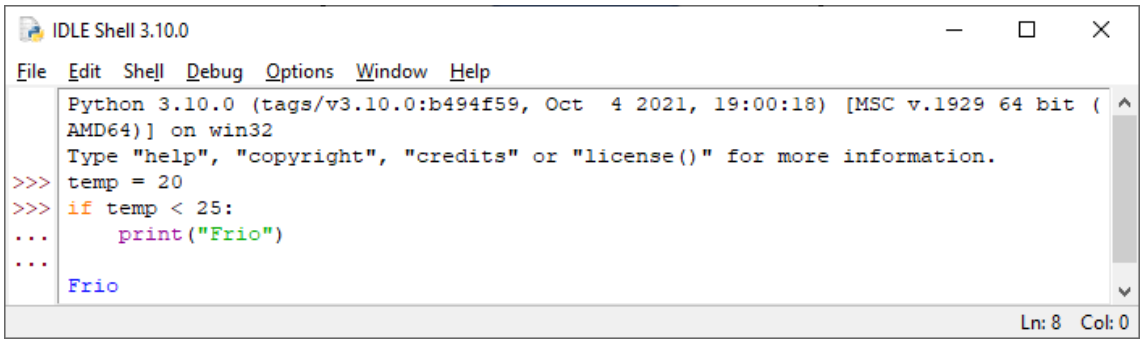


If <condición>:

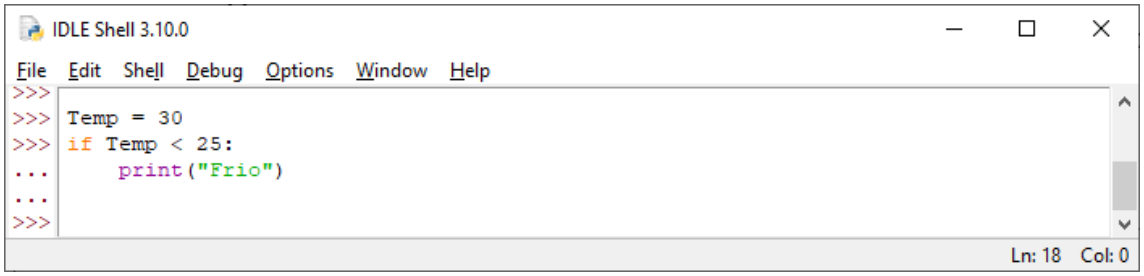
⟶ # Código

Recuerda que después de la condición tendrás que poner dos puntos ':' si no nos dará error, ya que la sintaxis obliga a poner estos dos puntos.

El codigo que se ha de ejecutar si se cumple la condición tiene que ir dentado, es decir unos cuatro espacios a la derecha, según el ejemplo anterior.

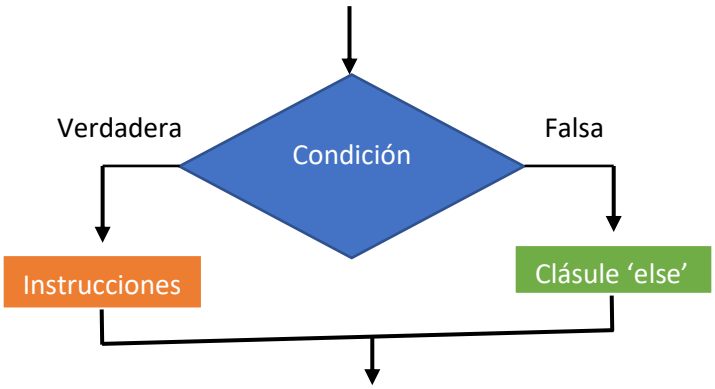


En este ejemplo le estamos diciendo si temperatura es menor de 25 que imprima Frio, como la condición se cumple ya que temperatura es igual a 20.



En este ejemplo como la condición no se cumple no se imprime el mensaje Frio.

Cláusula 'else'




```
if <condición>:
    # Código
else:
    # Código
```

Ejemplo:

untitled

File Edit Format Run Options Window Help

```
1 temp = 20
2 if temp < 25:
3     print("Frío")
4 else:
5     print("Calor")
```

Ln: 5 Col: 18

Vamos a guardar y ejecutar.

IDLE Shell 3.10.0

File Edit Shell Debug Options Window Help

```
>>> ========== RESTART: D:/Python/Temperatura.py ==========
Frío
>>>
```

Ln: 21 Col: 0

Como la temperatura es menor de 25 se cumple la sentencia if e ignora la sentencia else.

Temperatura.py - D:/Python/Temperatura.py (3.10.0)

File Edit Format Run Options Window Help

```
1 temp = 30
2 if temp < 25:
3     print("Frío")
4 else:
5     print("Calor")
6
```

Ln: 1 Col: 9

Guardamos y ejecutamos.

IDLE Shell 3.10.0

File Edit Shell Debug Options Window Help

```
>>> ========== RESTART: D:/Python/Temperatura.py ==========
Calor
>>>
```

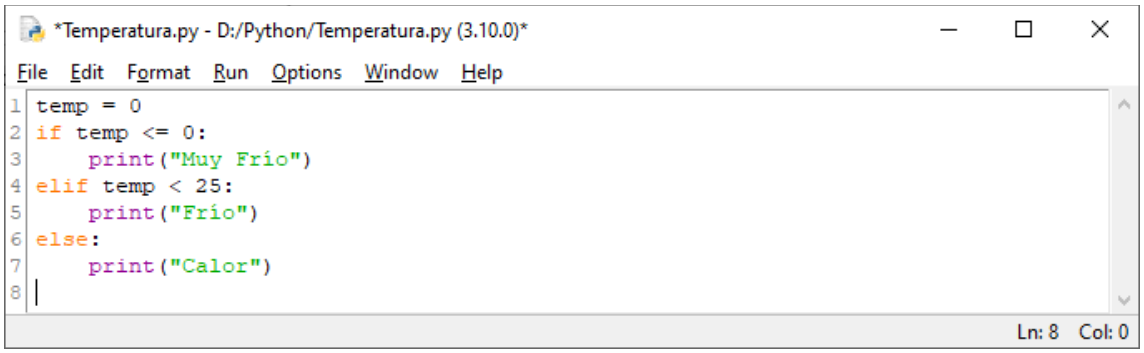
Ln: 24 Col: 0

Como la temperatura en mayor de 25 ignora el if y ejecuta el else.

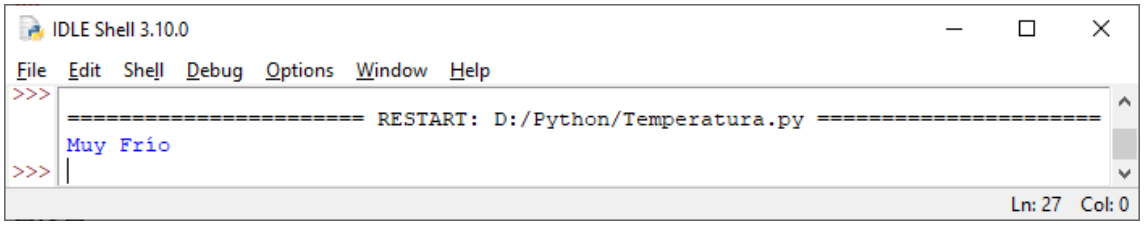
Cláusula 'elif'

```
if <condición1>:
        # Código1
elif <condición2>:
        # Código2
else:
        # Código3
```

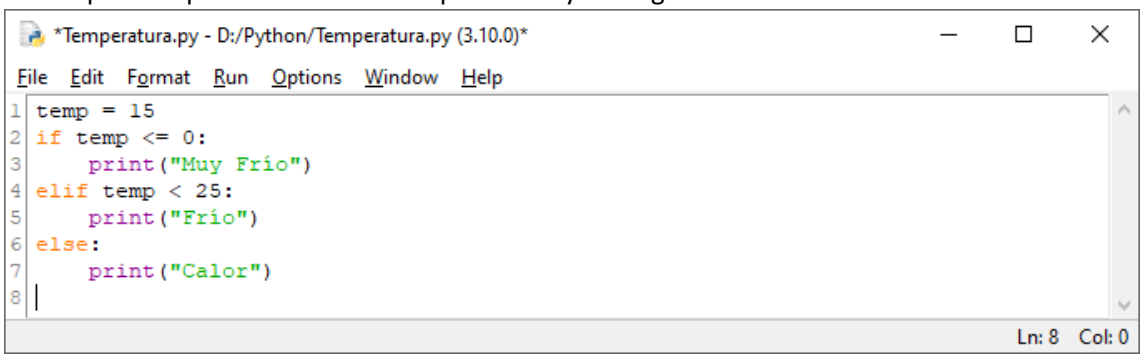
En este caso si se cumple la condición1 se ejecuta el código1 si no compara si se cumple la condición2 si es así ejecuta el código2 de lo contrario se ejecutará el código3.
Pueden haber varios elif pero de else solo uno.
Si tenemos varios elif cuando se cumpla uno se ejecutará el código de dicha condición y si hubieran más condiciones elif ya no se van a ejecutar.



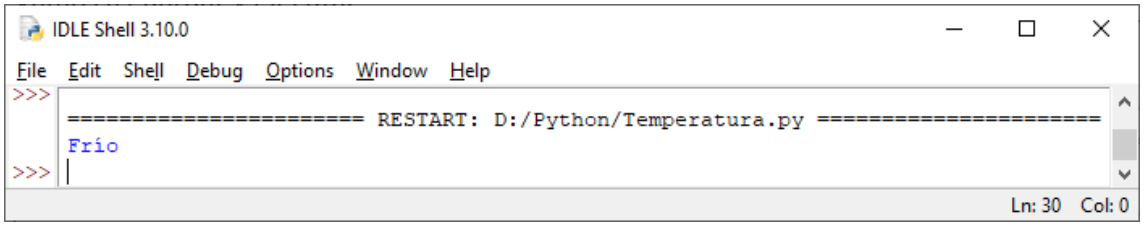
Vamos a guardar y ejecutar.



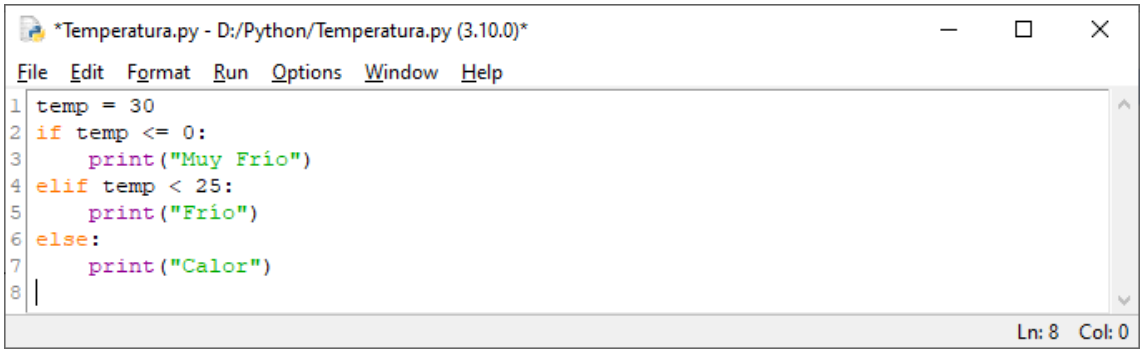
Al cumplirse la primera condición imprime 'Muy Frío' ignorando el resto de condiciones.



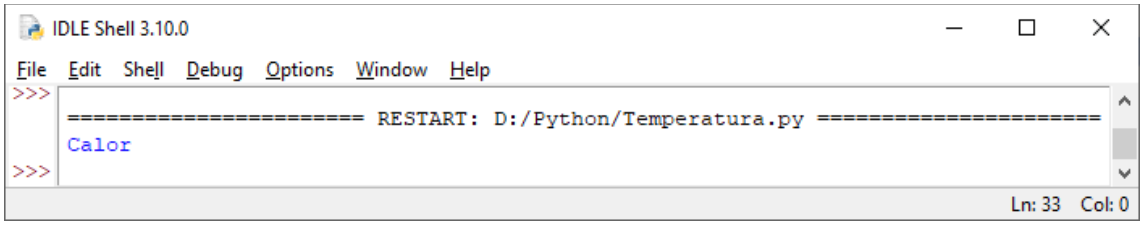
Vamos a guardar y ejecutar.



En este caso al no cumplir la primera condición, compara la segunda condición como se cumple imprime ‘Frío” ignorando la instrucción else.



Guardamos y ejecutamos.



En este caso la primera condición no se cumple, compara la segunda condición que tampoco se cumple pues ejecuta el código de la sentencia else.
Ejemplo con varios elif.

```
if <condición1>:
        # Código1
elif <condición2>:
        # Código2
elif <condición3>:
        # Código3
elif <condición4>:
        # Código4
elif <condición5>:
        # Código5
else:
        # Código 6
```

Comentarios

Texto que se escribe en el programa para facilitar su comprensión.

Estos comentarios nos ayudará a nosotros u otros programadores cuando tengan que seguir con un proyecto.

No es parte del código, cuando se ejecuta el programa estos comentarios son ignorados.

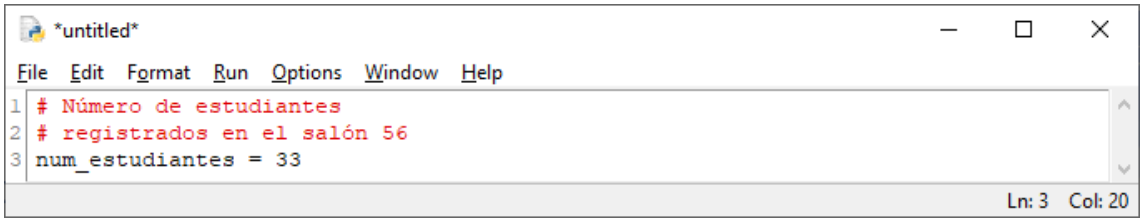
Propósito:

Describir la lógica de ciertas partes del código a otros programadores que lean el programa.

Comentario en Python.

comentario

Para crear un comentario tienes que empezar por almohadilla.



Hemos agregado dos líneas de comentario para explicar el valor de la variable.

Tenemos que evitar añadir más comentarios de los necesarios, no comentar en exceso.

No es conveniente explicar lo que hace cada línea, ya que un programador tiene que saber interpretar el código de programación.

Listas

Es una estructura de datos utilizada para almacenar múltiples valores en secuencia.

Ejemplo de una lista que contiene 4 números:

[1, 2, 3, 4]

Para definir una lista tenemos que usar corchetes y cada uno de los elementos separado por comas y seguido de un espacio.

Otro ejemplo de lista.

["a", "b", "c"]
[0] [1] [2]

También se denomina secuencia ordenada ya que cada elemento está ubicado en un índice específico.

Características:

- Secuencia ordenada de valores.
- Puede contener valores de cualquier tipo.
- Pueden contener valores de distintos tipos.
- Cada posición en la lista está asociada a un entero llamado "índice".
- Es mutable. Puede ser modificada.

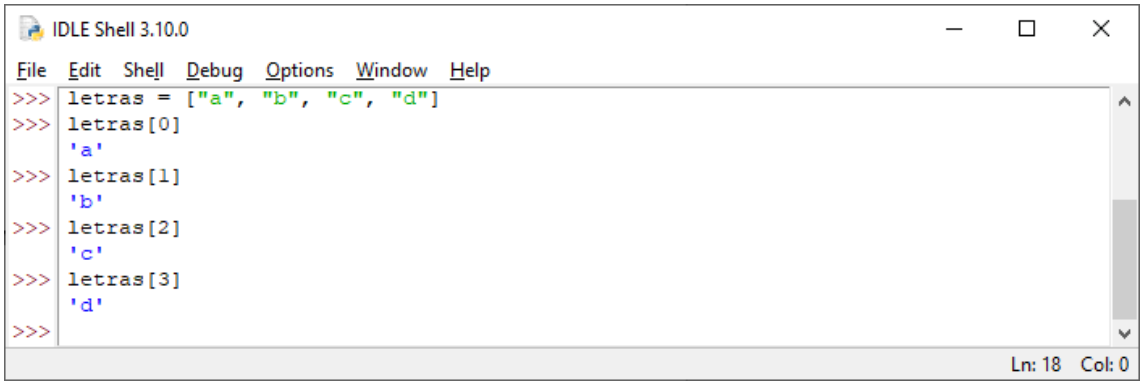
Acceder a un elemento.

Para acceder a un elemento de la lista, usamos su índice correspondiente.

<lista>[<índice>]

Tenemos una lista con las letras a, b, c y d.

["a", "b", "c", "d"]
[0] [1] [2] [3]



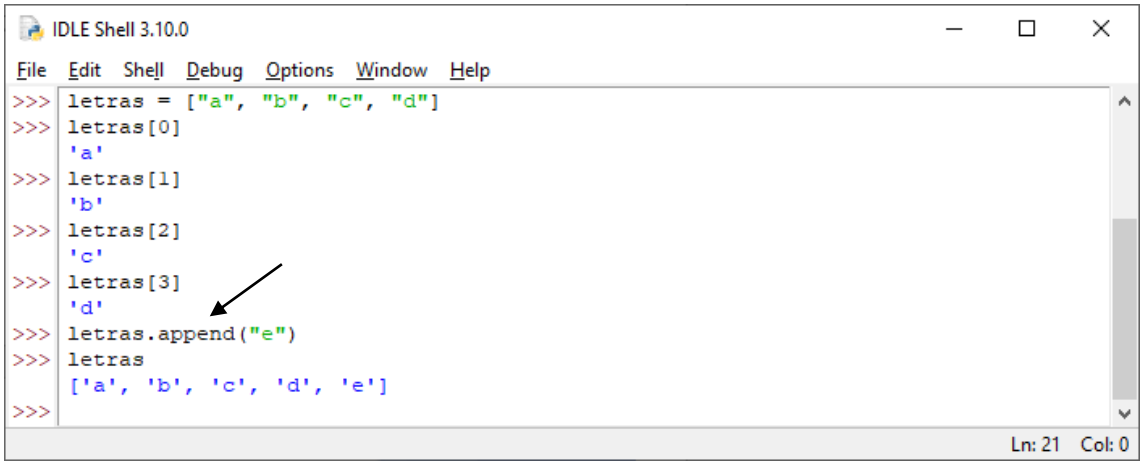
Podemos acceder a cada uno de los elementos de la lista agregando el número de su índice entre corchetes.

Agregar un elemento.

1. Al final de la lista
2. En un índice específico

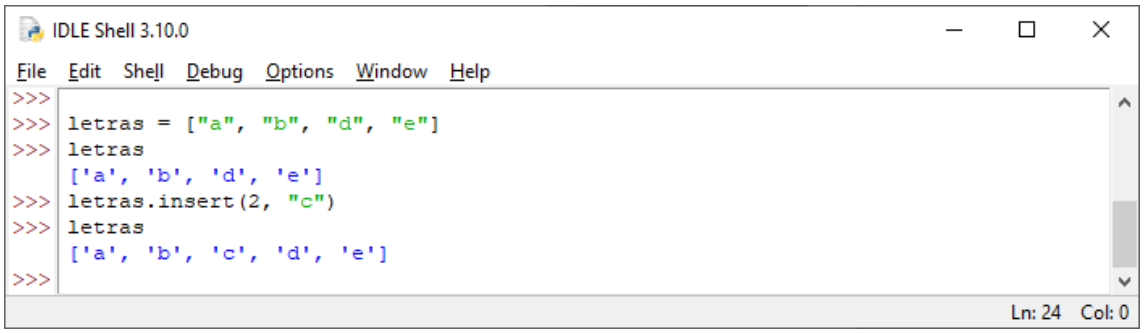
Al final de la lista.

<lista>.append(<elemento>)



En un índice específico

<lista>.insert(<índice>, <elemento>)



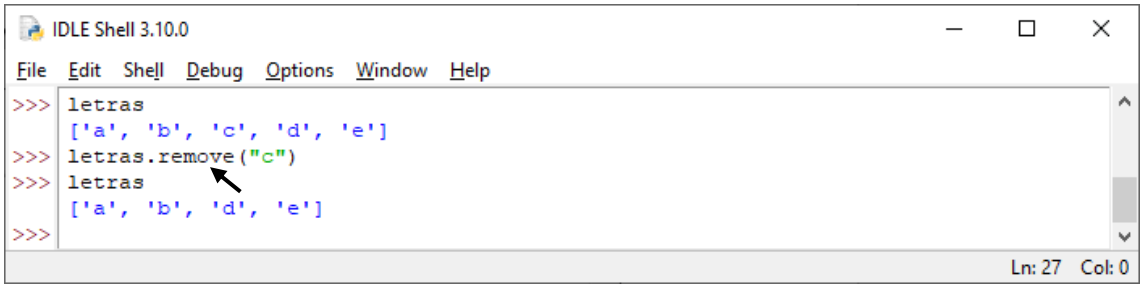
Pregunta:

Selecciona el método que agrega un elemento al final de la lista.

a) .index()
b) .append()
c) .extend()
d) .insert()

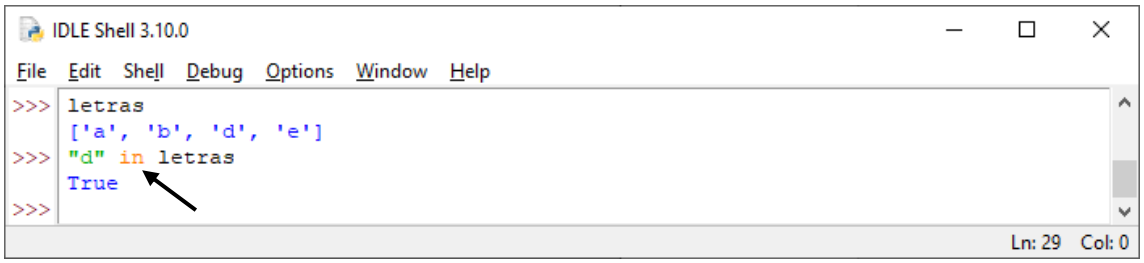
Borrar un elemento.

<lista>.remove(<elem>)

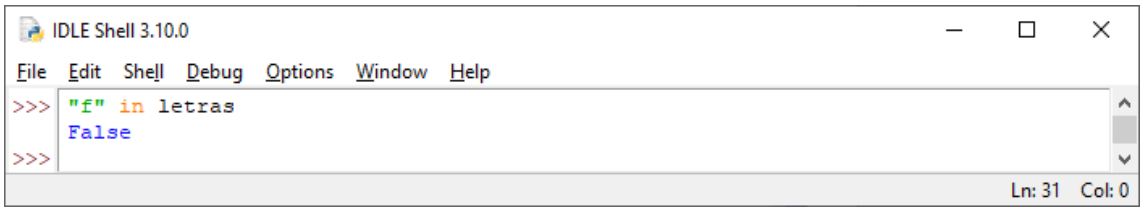


Elimina el primer valor encontrado.
Si intentamos eliminar un elemento que no existe en la lista tendremos el siguiente error.
ValueError: List.remove(x): x not in list. (el elemento no está en la lista).
¿Como comprobar si hay un determinado elemento antes de borrarlo?
Encontrar un elemento.

<elem> in <lista>

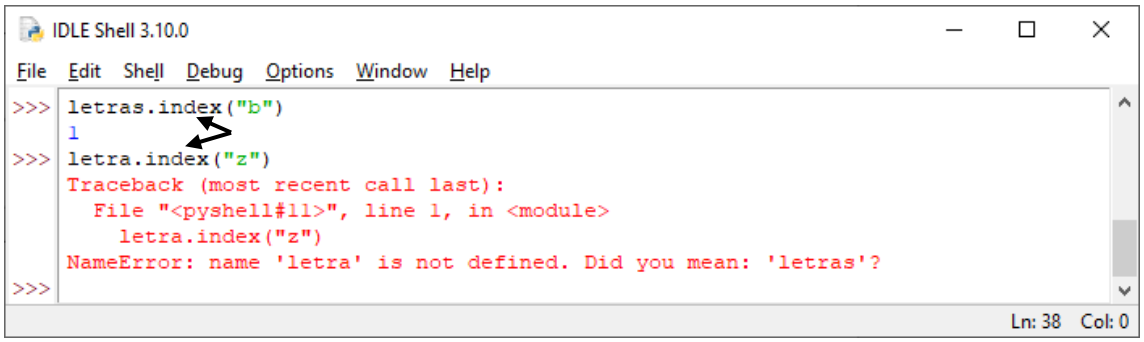


Si está en la lista el valor será True.



Si no está en la lista el valor será False.
.index(<elem>)
Retorna el índice de la primera ocurrencia del elemento en la lista. Si no se encuentra el elemento, ocurre un error.

<lista>.index(<elem>)



La letra “b” se encuentra en la lista en el índice 1 pero la “z” no está en la lista por este motivo aparece el siguiente mensaje de error.

Pregunta:

Selecciona el valor retornado por el operador ‘in’ en este código

>>> a = [1, 2, 3, 4]

>>> 6 in a

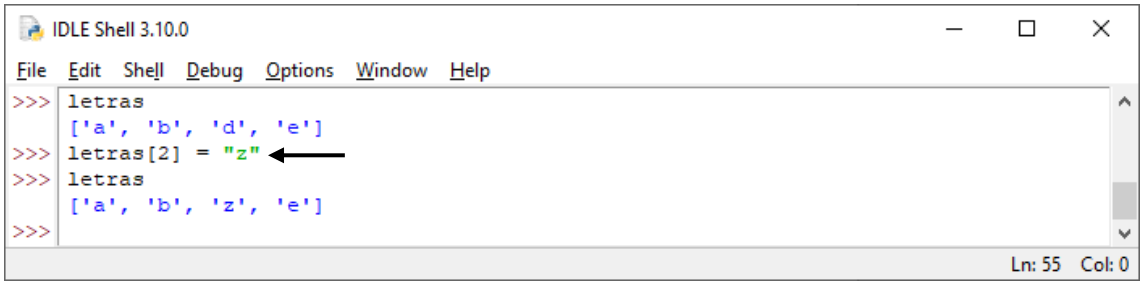
a) True
b) False

Cambiar un elemento

Mutable

Puede ser cambiadas. Sus elementos pueden ser modificados.

<lista>[<índice>] = <nuevo_valor>



Métodos de las listas.

Operaciones comunes de listas que ya están implementadas en Python.

<lista>.<método>(<parámetros>)

Métodos importantes:

- .count(<elem>) →Contar las veces que se repite un elemento.
- .extend(<lista>) → Extender una lista agregando elementos de otra lista.
- .pop() → Elimina y retorna un elemento de la lista.
- .reverse() → Que invierte el orden actual de la lista.
- .sort() → Ordena la lista en modo ascendente o descendente.

Pregunta:

Selecciona el valor final de la lista.

>>> a = [3, 4, 5, 6]

>>> a.pop()

6

>>> a.reverse()

>>> a.extend([8 ,9, 0])

a) [3, 4, 5, 6]
b) [6, 5, 4, 8, 9, 0]
c) [8, 9, 0, 6, 5, 3]
d) [5, 4, 3, 8, 9, 0]

Tuplas

Estructura de datos inmutable que contiene una secuencia ordenada de elementos.

Ejemplo de tupla.

(1, 2, 3, 4)

Las tuplas está rodeada por paréntesis y una serie de elementos separados por coma.

Son secuencias ordenadas igual que las listas.

("a", "b", "c")
[0] [1] [2] ← Índices

Características:

- Secuencia ordenada de valores.
- Puede contener valores de cualquier tipo de datos.
- Puede contener valores de distintos tipos de datos.
- Cada posición de la tupla se identifica con un entero denominado "índice".
- Son inmutables. No pueden modificarse.

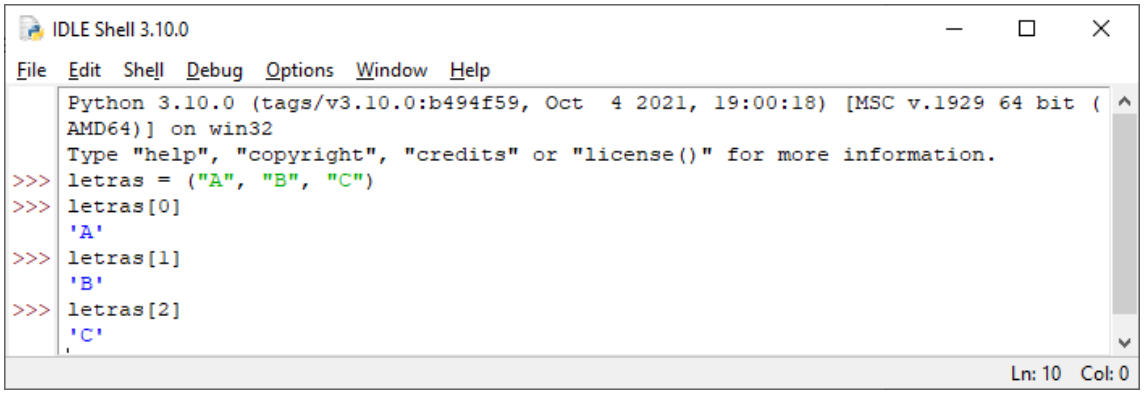
Esto es interesante si vamos crear una estructura de datos que no queremos que se modifique con la ejecución del programa.

Si intentamos modificar algún elemento de la tupla observaremos el siguiente error:

TypeError: 'tuple' object does not support item asignament.

Acceder a un elemento

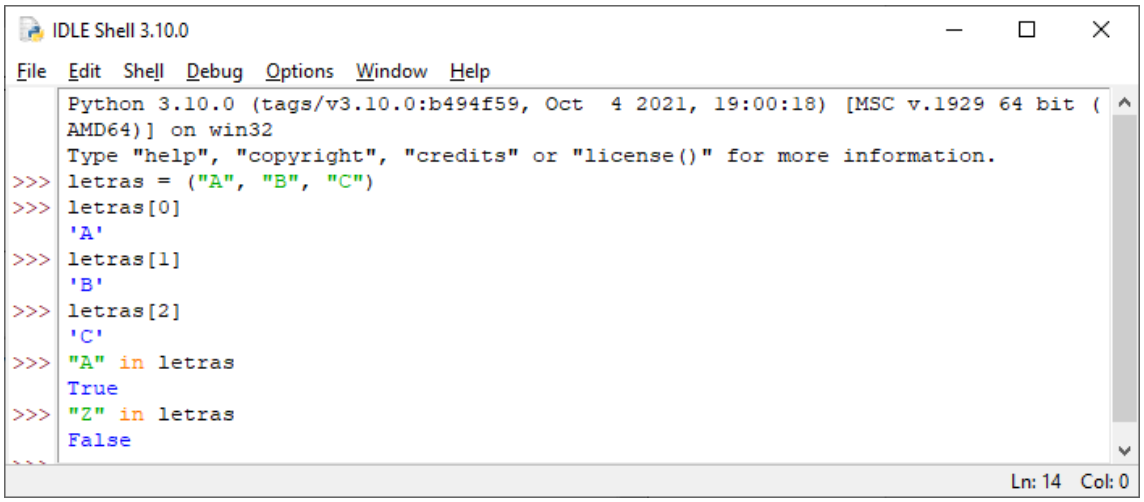
Podemos acceder a un elemento usando su índice.



En este aspecto funciona igual que una lista.

Encontrar un elemento

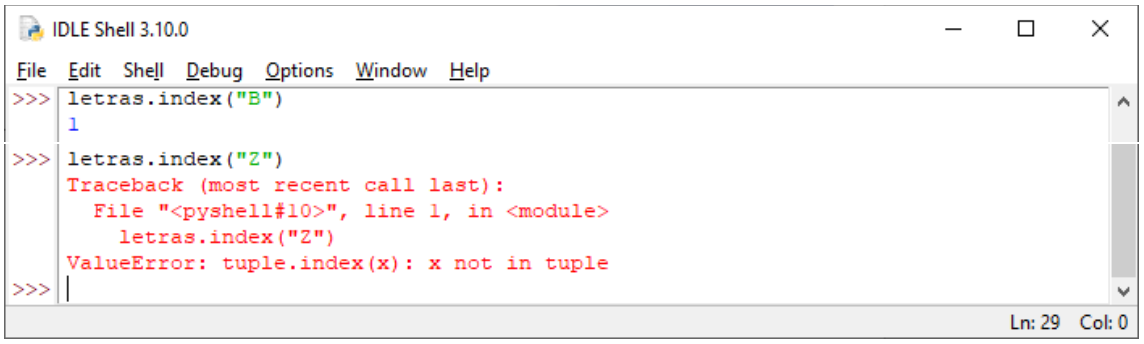
<elem> in <tupla>



Responde con un valor booleano si está el la tupla True y si no está en la tupla False.

También podemos localizar en que posición de índice se encuentra el elemento en la tupla.

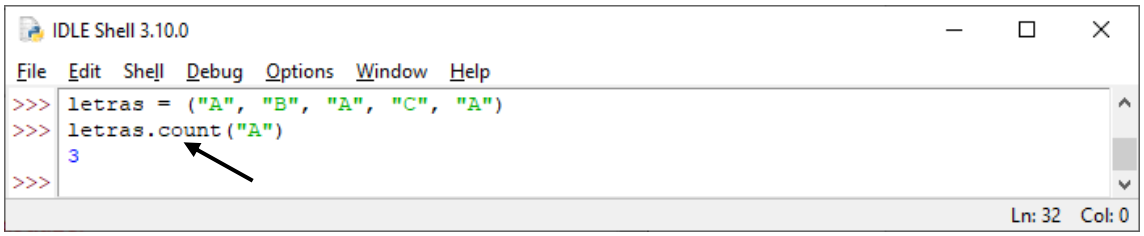
<tupla>.index(<elem>)



Si el elemento existe nos dirá su posición con respecto al índice, en cambio si el elemento no existe nos dará un mensaje de error.

Si queremos saber cuantos elementos iguales tenemos en una tupla:

<tupla>.count(<elem>)



Pregunta:

Completa la oración:

La principal diferencia entre las listas y las tuplas en Python es que las listas son mutables y las tuplas son inmutables.

Recuerda que las listas se definen con corchetes y las tuplas con paréntesis.

Diccionarios

Colección de pares clave-valor.

Ejemplo de un diccionario.

{“A”: 45, “B”: 30}

Par Clave-Valor

En Python un diccionario se suele definir con llaves.

El elemento de la izquierda se llama clave y el de la derecha valor.

Con la clave podemos acceder al valor.

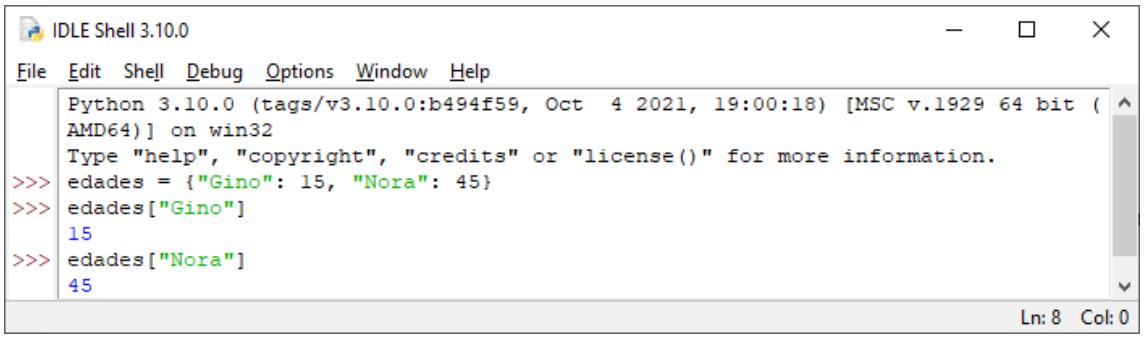
En el ejemplo anterior tenemos dos Clave-Valor, que están separados por una coma y un espacio.

Características

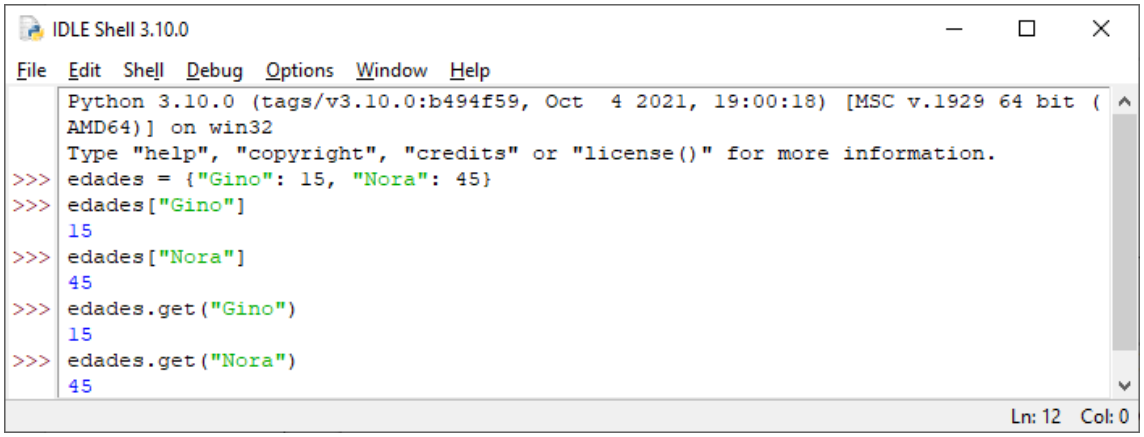
- Colección de Pares Clave-Valor.
- Las claves deben ser únicas e inmutables.
- Los valores asociados pueden ser de cualquier tipo.
- La clave se usa para acceder a su valor asociado.
- Los pares clave_valor pueden ser modificados, añadidos y eliminados.

Acceder a un valor

<diccionario>[<clave>]



Otra forma de obtener los datos.

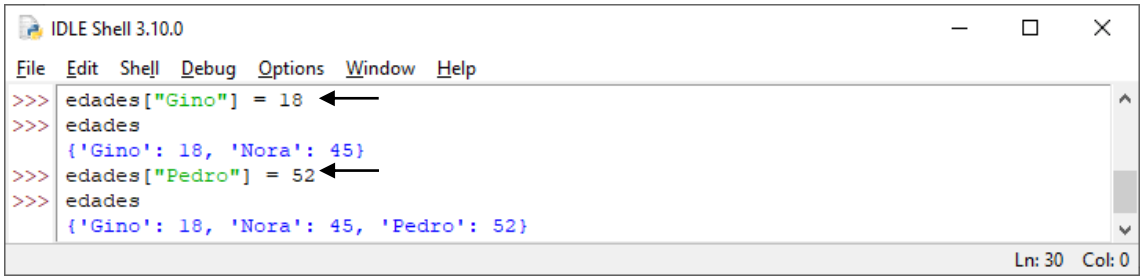


Con el método get(). Obtener.

Añadir y modificar

<diccionario>[<clave>] = <nuevo_valor>

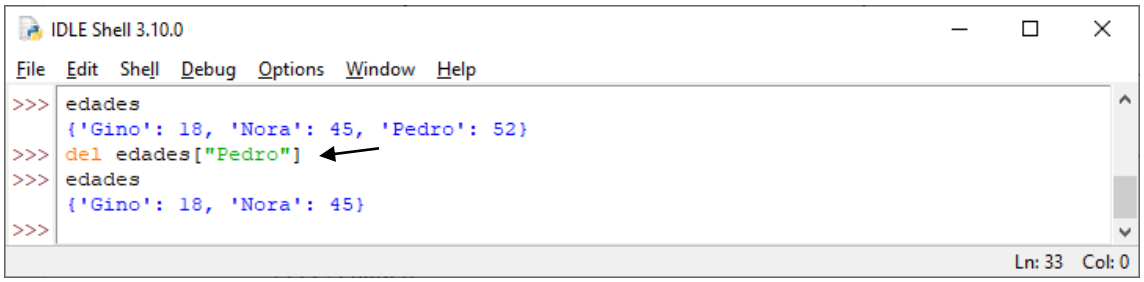
Si la clave ya existe se modifica el valor asociado, si la clave es nueva se crea un nuevo Clave-valor.



En el primer ejemplo como la clave de “Gino” ya existe lo que hacemos es cambiarle su valor, en cambio en el segundo ejemplo como “Pedro” no existe lo que hace es agregar un nuevo clave-valor.

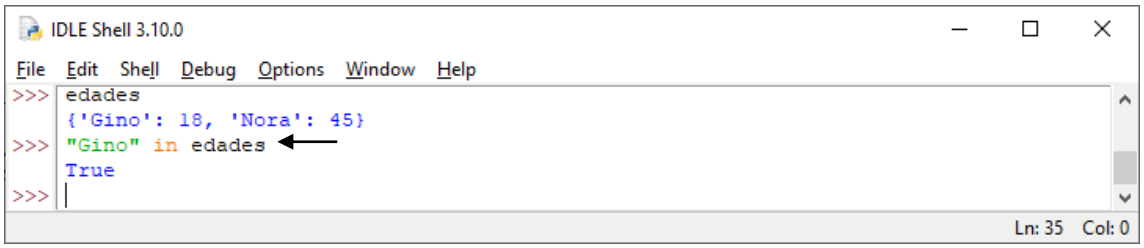
Borrar clave-valor:

del <dicc>[<clave>]



Revisar Existencia:

<elem> in <dicc>

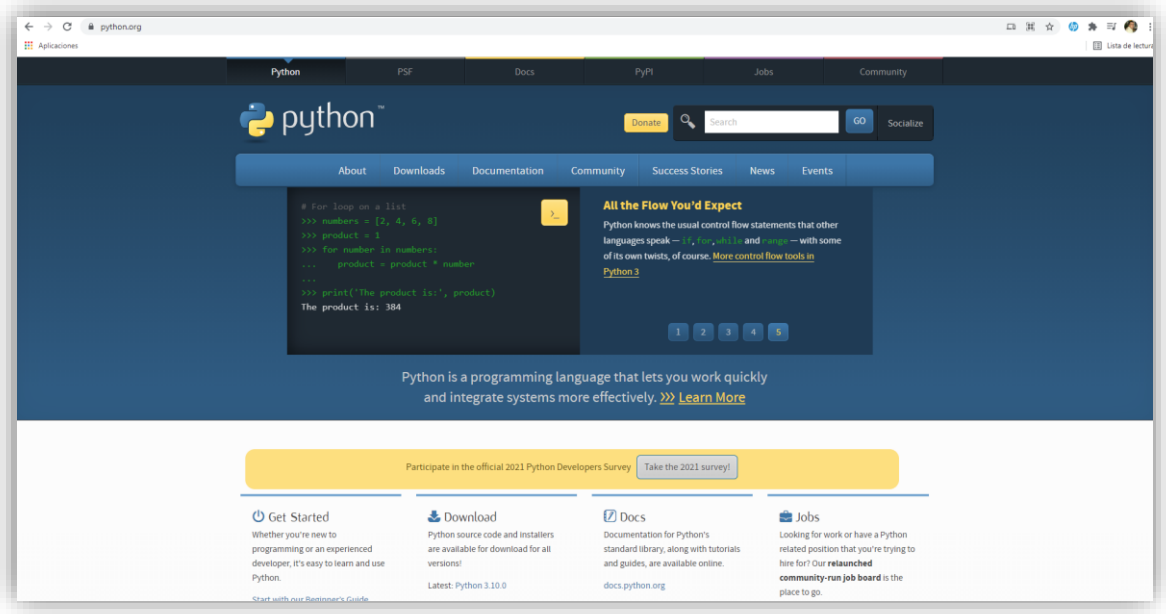


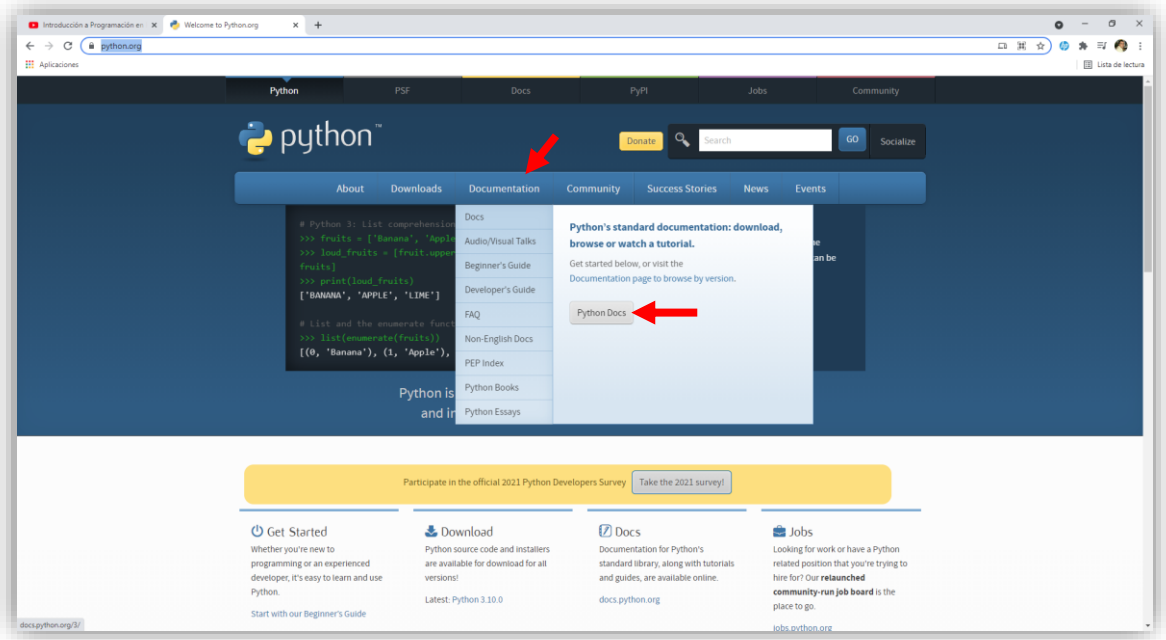
Si la clave existe retornará un valor booleano de True de lo contrario False.

Documentación

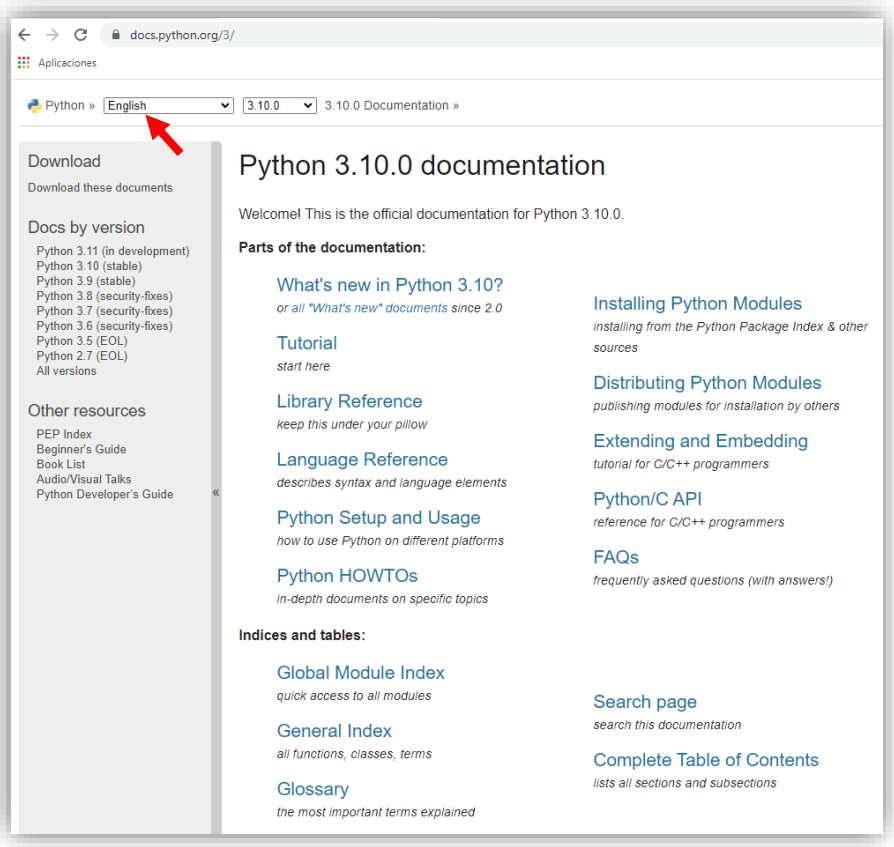
Para acceder a la documentación de Python vamos a la siguiente url:

https://www.python.org/

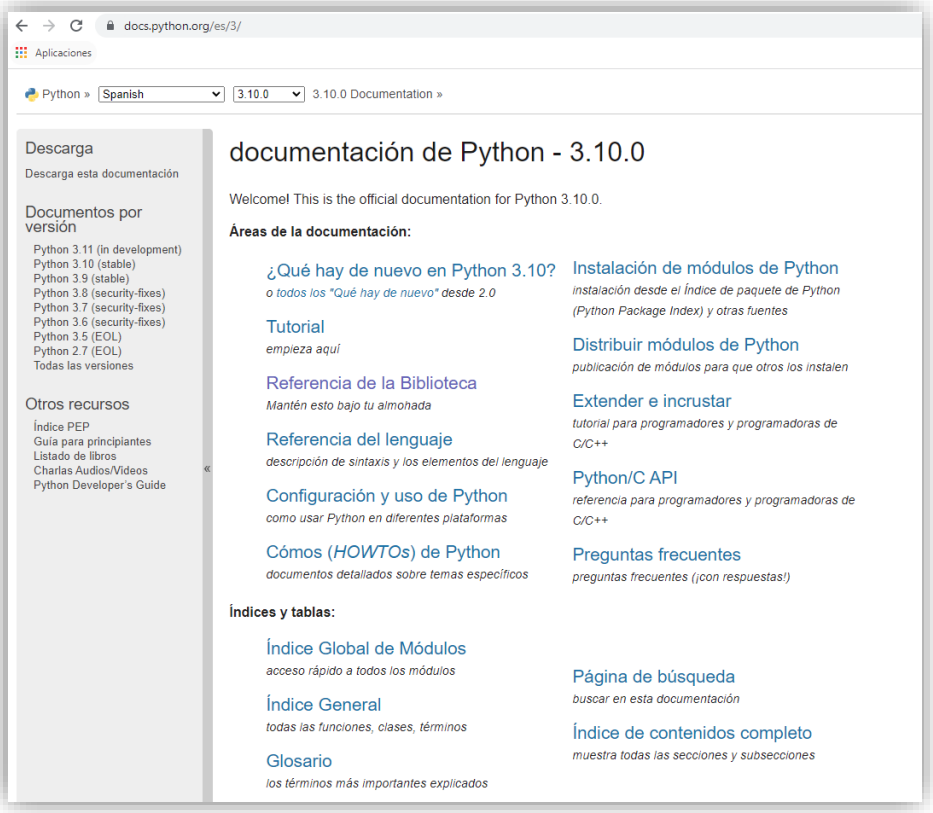




Seleccionaremos Documentation y de este el botón Pyton Docs.



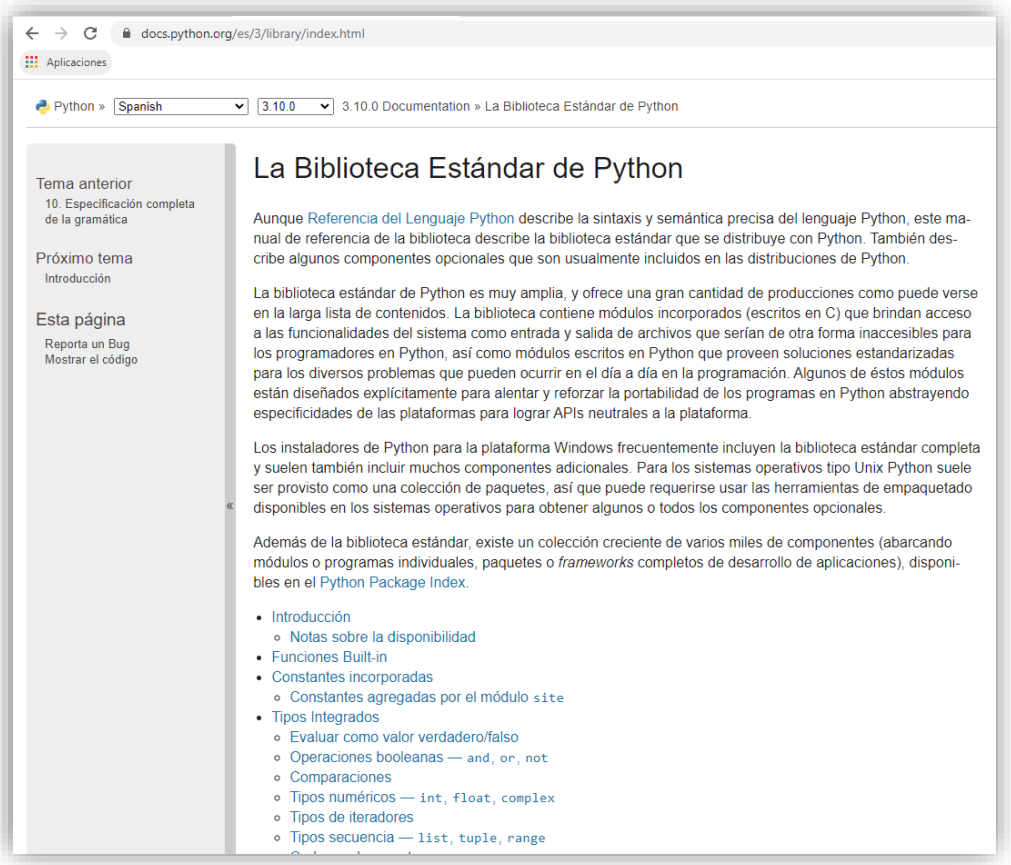
En la parte superior izquierda podemos seleccionar el idioma en Spanish.



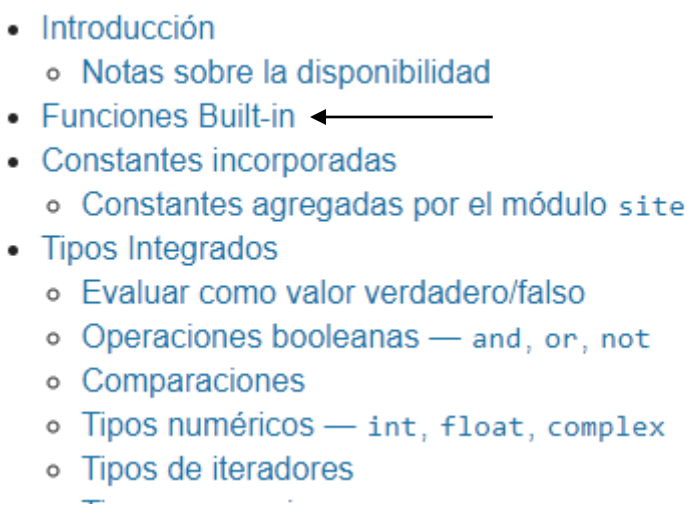
Unos de los lugares que vamos a ir muy a menudo será “Referencia de la Biblioteca” y la “Referencia del lenguaje” Descripción de sintaxis y los elementos del lenguaje.

También podemos acceder a Tutoriales y además ¿Qué hay de nuevo en Python 3.10?.

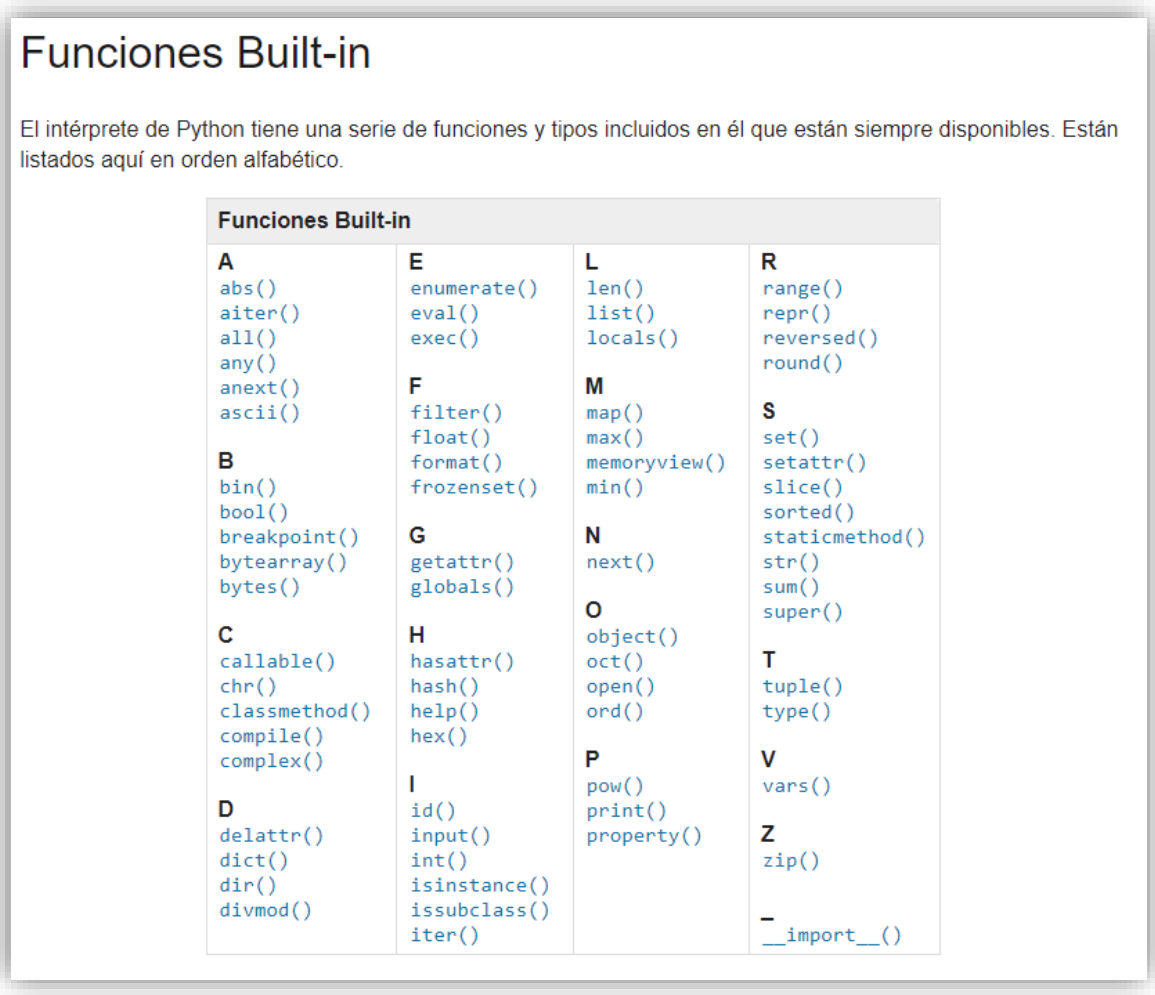
Pero por ahora vamos a ir a “Referencia de biblioteca”.



Uno de los apartado que vamos a utilizar “Funciones Buit-in”.



Vamos a seleccionarla.



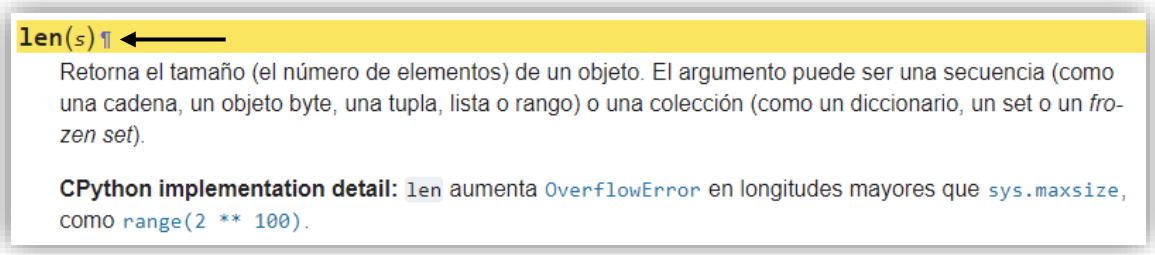
Funciones Built-in

El intérprete de Python tiene una serie de funciones y tipos incluidos en él que están siempre disponibles. Están listados aquí en orden alfabético.

Funciones Built-in

A	**E**	**L**	**R**
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
any()			round()
anext()	**F**	**M**	
ascii()	filter()	map()	**S**
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	**G**	**N**	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	**H**	object()	
callable()	hasattr()	oct()	**T**
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		**P**	**V**
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	**Z**
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		**_**
	iter()		__import__()

Haciendo un clic en cada una de ellas tendremos una descripción, vamos a hacer clic en len().



len(*s*) ¶

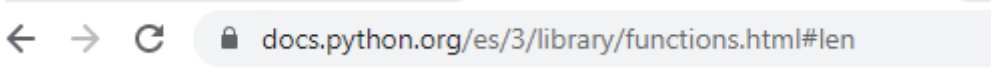
Retorna el tamaño (el número de elementos) de un objeto. El argumento puede ser una secuencia (como una cadena, un objeto byte, una tupla, lista o rango) o una colección (como un diccionario, un set o un *frozen set*).

CPython implementation detail: `len` aumenta `OverflowError` en longitudes mayores que `sys.maxsize`, como `range(2 ** 100)`.

Veremos una descripción de esta función.

Si queremos compartir esta información para que puedan hacer un link en esta parte seleccionaremos el símbolo que seleccionamos con una flecha.

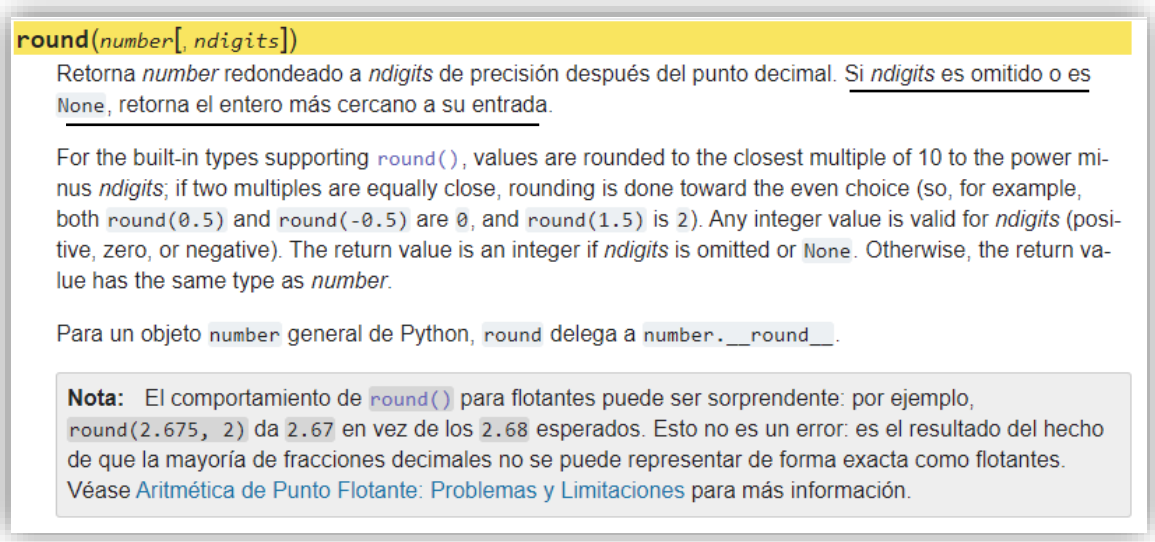
Luego verás en la barra de direcciones la ruta que has de compartir.



La puedes seleccionar copiar y compartirla.

Para probarlo una vez la has copiado abre otra página en tu navegador y pegas esta dirección url que has copiado, verás que llegas al mismo sitio.

Ahora vamos a consultar la función round().



round(*number*[, *ndigits*])

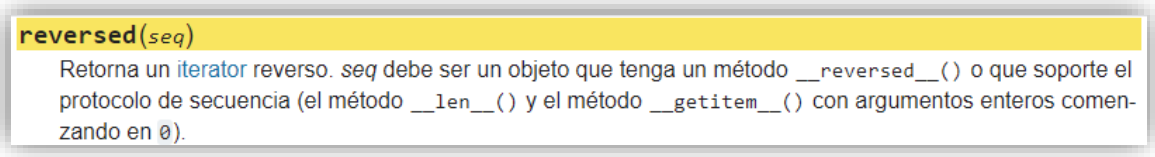
Retorna *number* redondeado a *ndigits* de precisión después del punto decimal. Si *ndigits* es omitido o es `None`, retorna el entero más cercano a su entrada.

For the built-in types supporting `round()`, values are rounded to the closest multiple of 10 to the power minus *ndigits*; if two multiples are equally close, rounding is done toward the even choice (so, for example, both `round(0.5)` and `round(-0.5)` are `0`, and `round(1.5)` is `2`). Any integer value is valid for *ndigits* (positive, zero, or negative). The return value is an integer if *ndigits* is omitted or `None`. Otherwise, the return value has the same type as *number*.

Para un objeto `number` general de Python, `round` delega a `number.__round__`.

Nota: El comportamiento de `round()` para flotantes puede ser sorprendente: por ejemplo, `round(2.675, 2)` da `2.67` en vez de los `2.68` esperados. Esto no es un error: es el resultado del hecho de que la mayoría de fracciones decimales no se puede representar de forma exacta como flotantes. Véase Aritmética de Punto Flotante: Problemas y Limitaciones para más información.

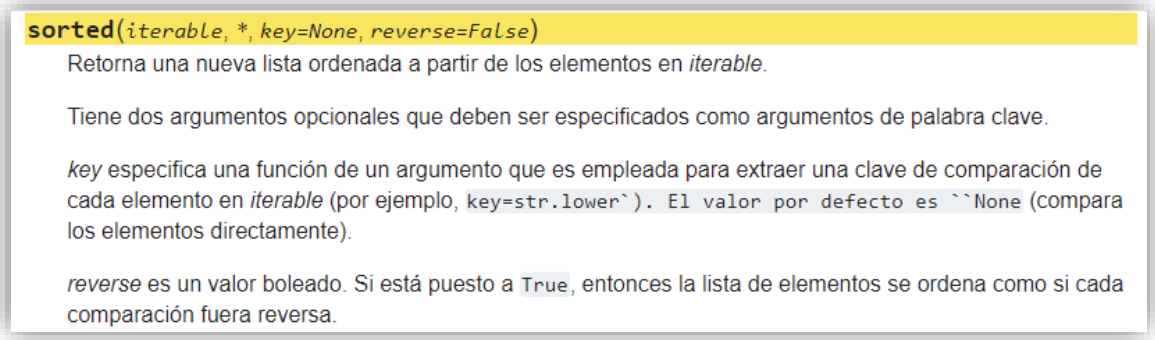
Esta función nos pide dos parámetros number numero que queremos redondear y ndigits número de dígitos que queremos que tenga.
Observarás que ndigits está entre corchetes, esto significa que este parámetro es opcional, observa el comentario que esta subrayado.
El primer párrafo suele tener la información más precisa y casi siempre iremos a consultar este primer párrafo.
Ahora vamos a la función reversed().



reversed(*seq*)

Retorna un iterator reverso. *seq* debe ser un objeto que tenga un método `__reversed__()` o que soporte el protocolo de secuencia (el método `__len__()` y el método `__getitem__()` con argumentos enteros comenzando en `0`).

Como podrás observar la información es muy escueta.
Ahora vamos a ir a consultar la función sorted().



sorted(*iterable*, *, *key=None*, *reverse=False*)

Retorna una nueva lista ordenada a partir de los elementos en *iterable*.

Tiene dos argumentos opcionales que deben ser especificados como argumentos de palabra clave.

key especifica una función de un argumento que es empleada para extraer una clave de comparación de cada elemento en *iterable* (por ejemplo, `key=str.lower`). El valor por defecto es ``None (compara los elementos directamente).

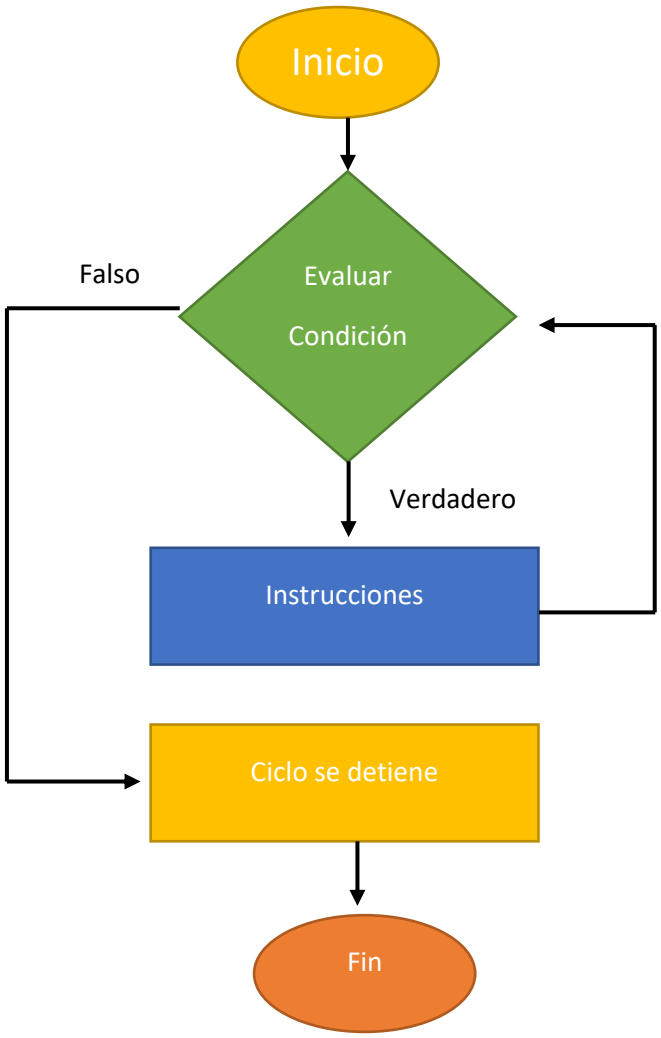
reverse es un valor boleado. Si está puesto a `True`, entonces la lista de elementos se ordena como si cada comparación fuera reversa.

Esta función tiene un parámetro reverse=False, si no la ponemos esta por defecto será false.

Ciclos For

Estructura de control en programación que permite ejecutar una o varias líneas de código múltiple de veces.

Los usamos cuando sabemos con antelación cuántas veces debemos repetir ciertas instrucciones.



Iteración es una repetición de una ejecución de un determinado grupo de código.

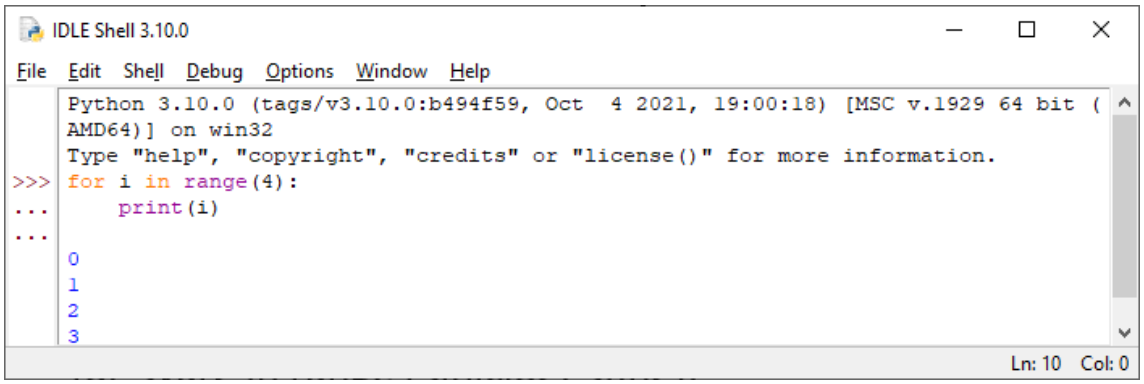
Los ciclos se denominan una estructura de control.

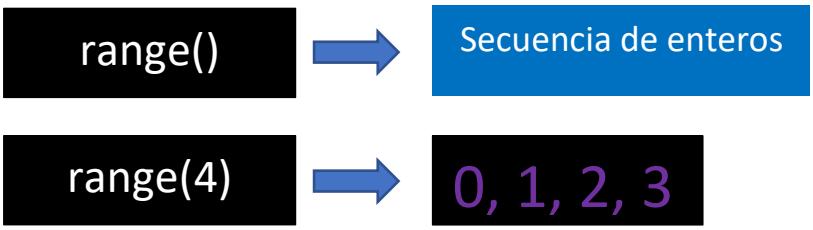
```
for <var> in range (<inicio>, <fin>):
    # Código
```

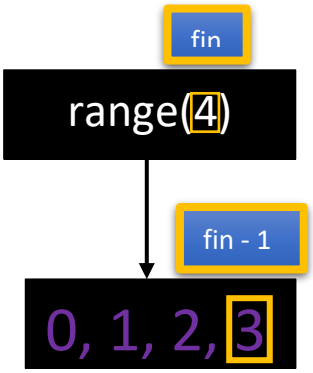
Variable de Control:

- Variable que puede ser utilizada en el código que se va a repetir.
- Se actualiza automáticamente antes de cada iteración.
- Debe tener un nombre descriptivo.

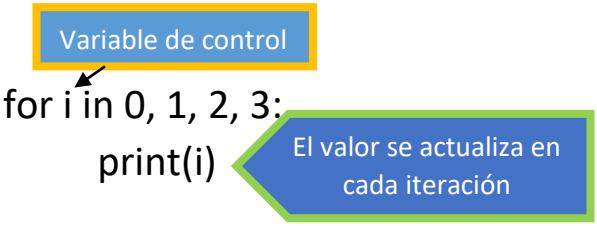
La variable se actualiza automáticamente dependiendo del rango “Range”, secuencia de valores.







Como no le pasamos el valor inicial, el valor por defecto es el 0.



Parámetros de la función range:

range(start, stop[, step])

start: El valor del parámetro start (0 si no se utiliza el parámetro).

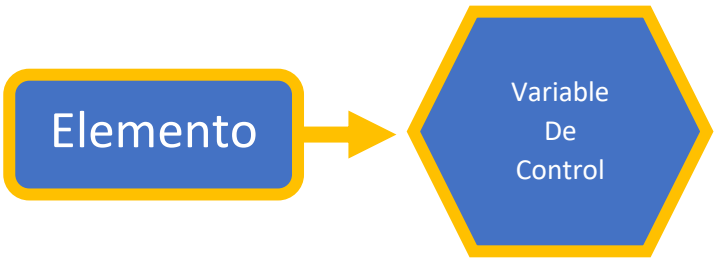
stop: El valor del parámetro stop.

step: El valor del parámetro step (1 si no se utiliza el parámetro).

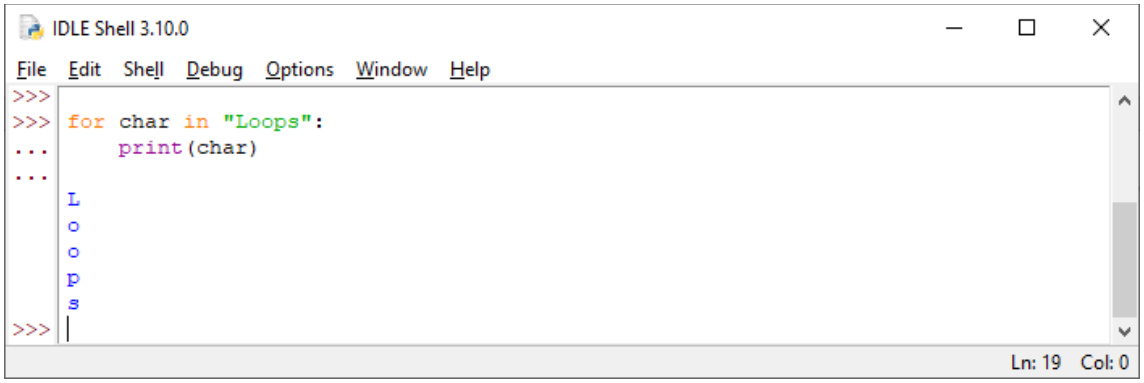
Ciclos sobre Iterables:

Para iterar sobre:

- Cadena de caracteres.
- Listas.
- Tuplas.
- Diccionarios.
- Otros.

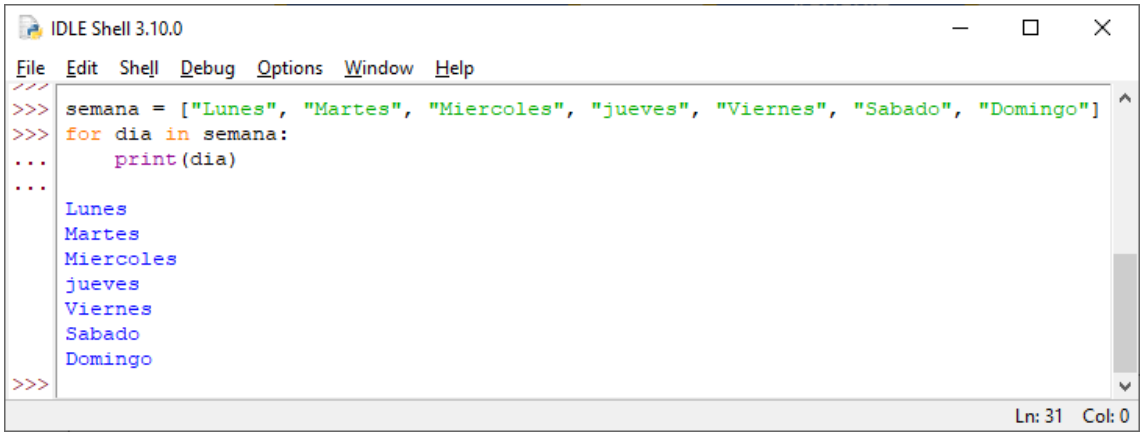



```
for <var> in <iterable>:
    # Código
```

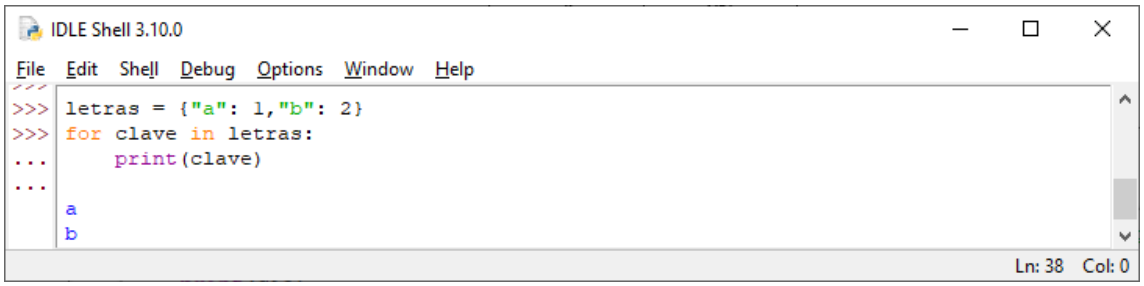


Iteración	char
1	'L'
2	'o'
3	'o'
4	'p'
5	's'

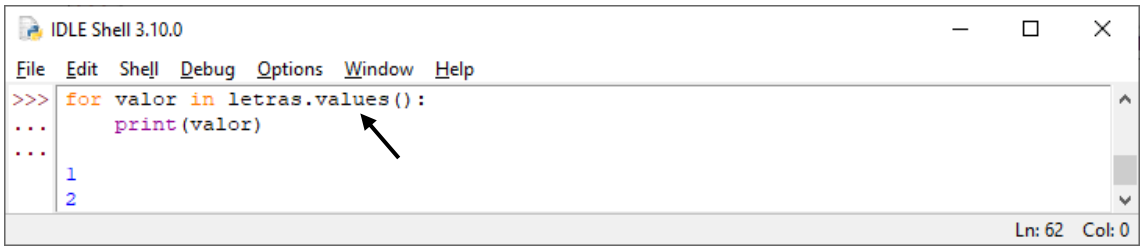
Ejemplo con listas.



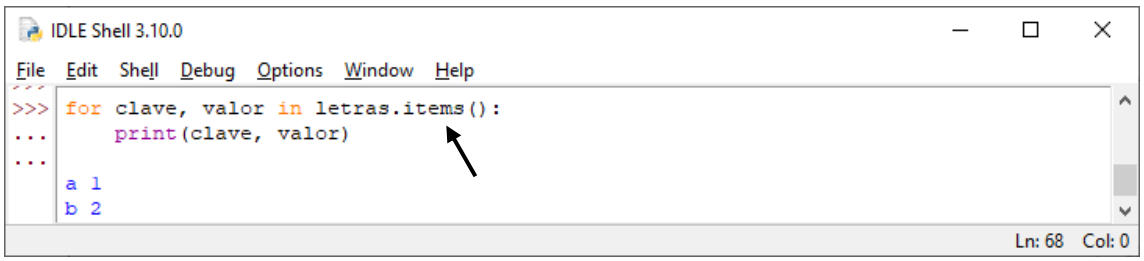
En el caso de los diccionarios hay variaciones.



También podemos iterar sobre los valores y los pares.



Este ejemplo es sobre valores.



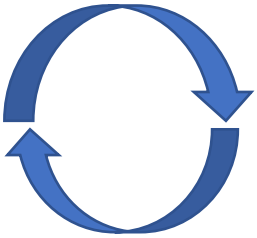
Este ejemplo es sobre clave y valor.

Ciclos While

El ciclo While (mientras) que continúa mientras una condición es verdadera y se detiene cuando esta es falsa.

Tú tienes que preocuparte de que en algún momento esta condición sea falsa, de este modo saldrá del bucle.

Ciclo infinito.



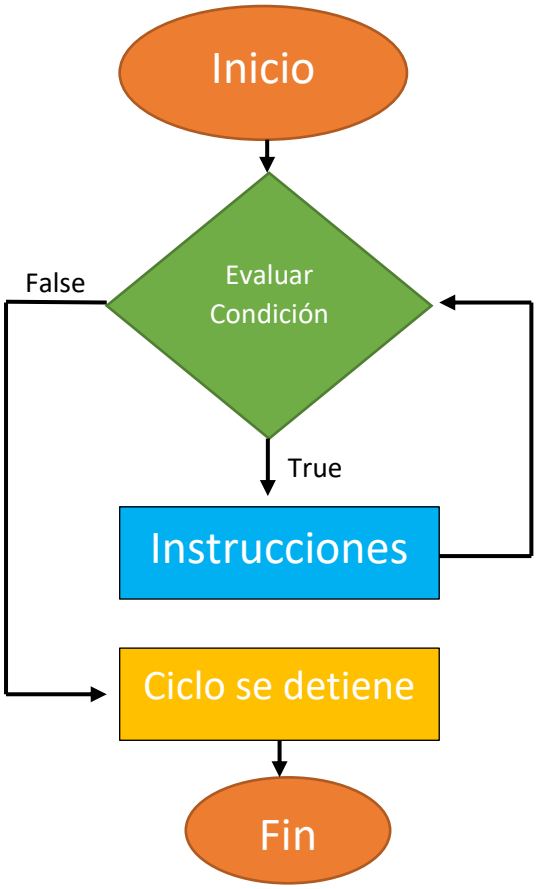
while <condición>:

➡ # Código

El código de ciclo while tiene que esta indentado (4 espacios)

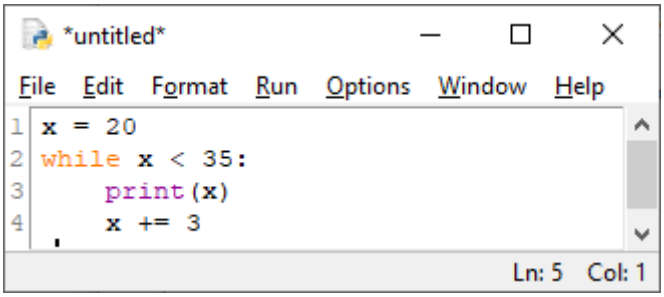
Iteraciones:

Un ciclo while no tiene un número fijo o predeterminado de iteraciones, se ejecuta hasta que la condición es falsa.



Variables de control

Los ciclos while no actualizan las variables de control automáticamente. Deben ser actualizadas en el cuerpo del ciclo.



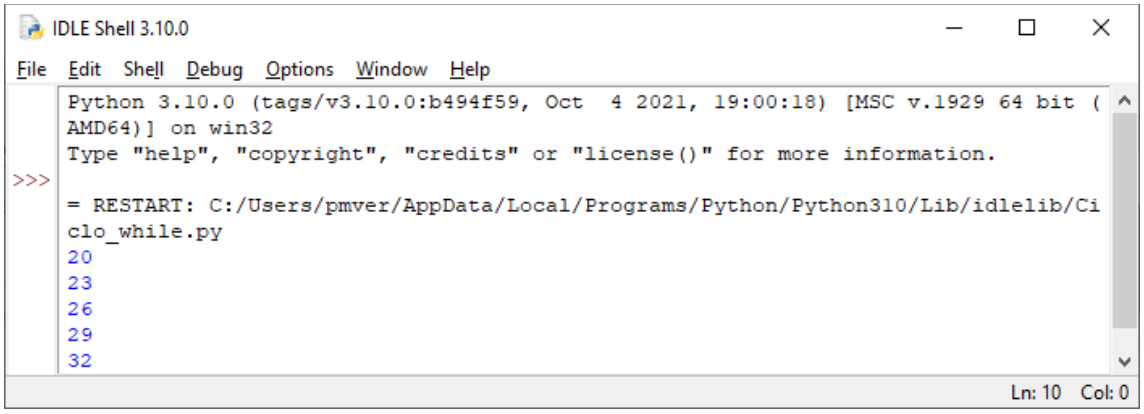
Iniciamos la variable x con el valor 20.

El ciclo while se irá cumpliendo mientras el valor de x sea menos de 35.

Mostrará por consola el valor de x.

A la variable x le hacemos un incremento de 3.

Vamos a guardar el proyecto y a ejecutarlo.



Si dentro del bucle no vamos actualizando el valor de la variable x, se creará un bucle infinito.

Si alguna vez te pasa, que sepas que con 'Ctrol + C' se interrumpe el programa.

Provocaremos un error llamado keyboardInterrupt.

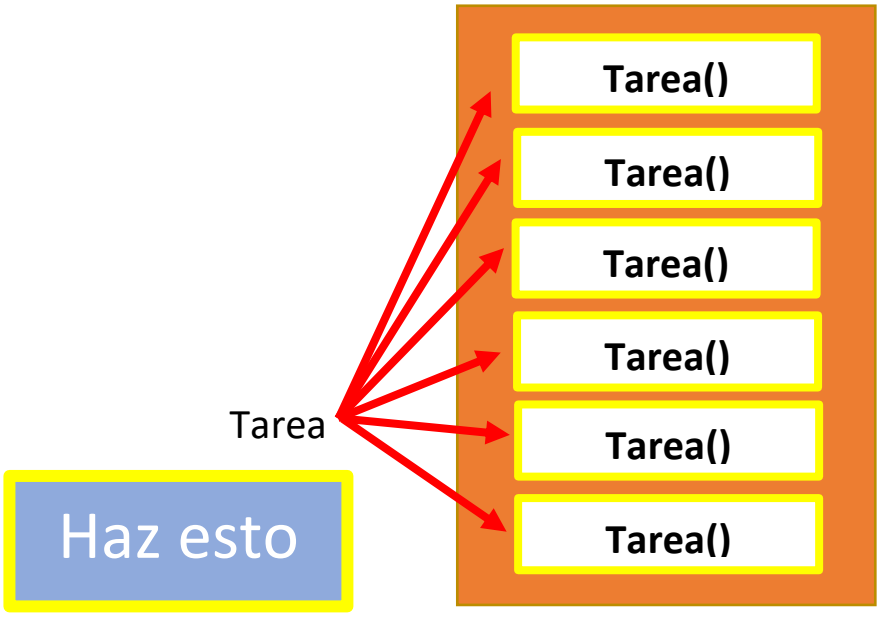
Funciones

Una función es un bloque de código reutilizable que realiza una sola tarea específica.

Cuando realizamos un programa de Python, normalmente puede haber tareas repetitivas, si necesitamos repetir esta tarea en diversas partes del programa.

Si a esta tarea se le tienen que realizar cambios el proceso puede ser tedioso y largo.

Hay un principio en la programación que dice “No debes repetir código”, (DRY).



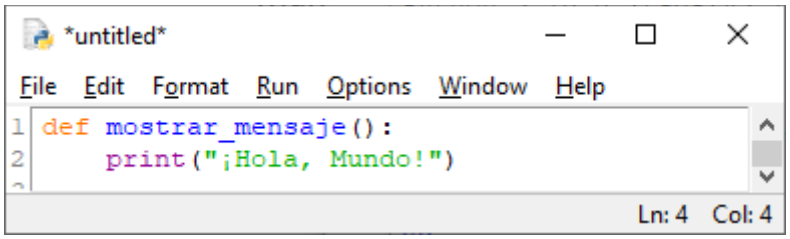
El código lo escribimos una sola vez y a donde alga falta, este código lo agregamos por mediación de una función.

Ventajas:

- Reutilizable.
- Conciso.
- Leíble.
- Mantenible.
- Comprobable.

Sintaxis

```
def <función>():
    # Código
```

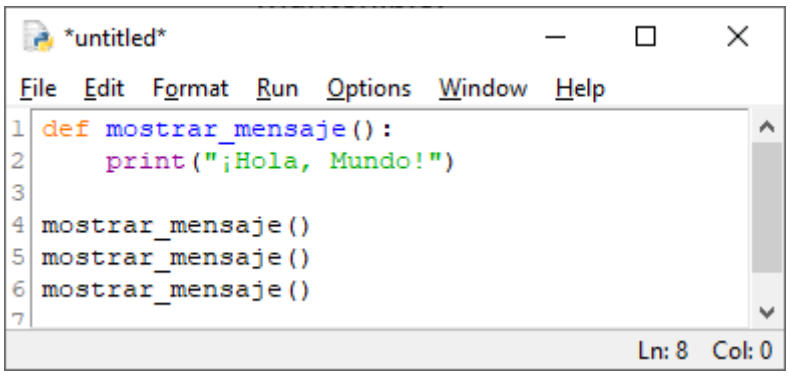


Llamar a una función.

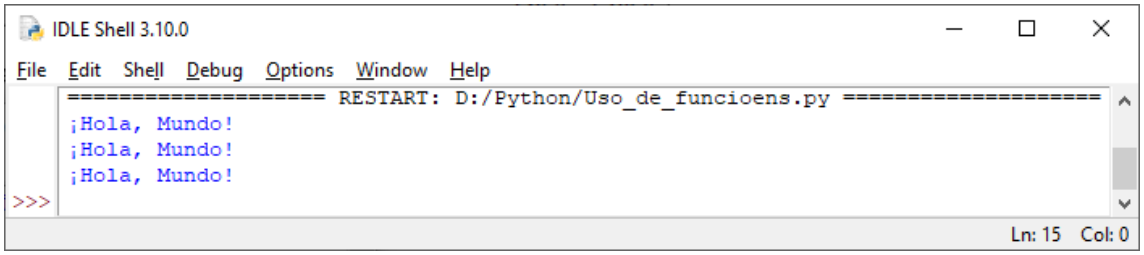
Ejecuta el código del cuerpo de la función.

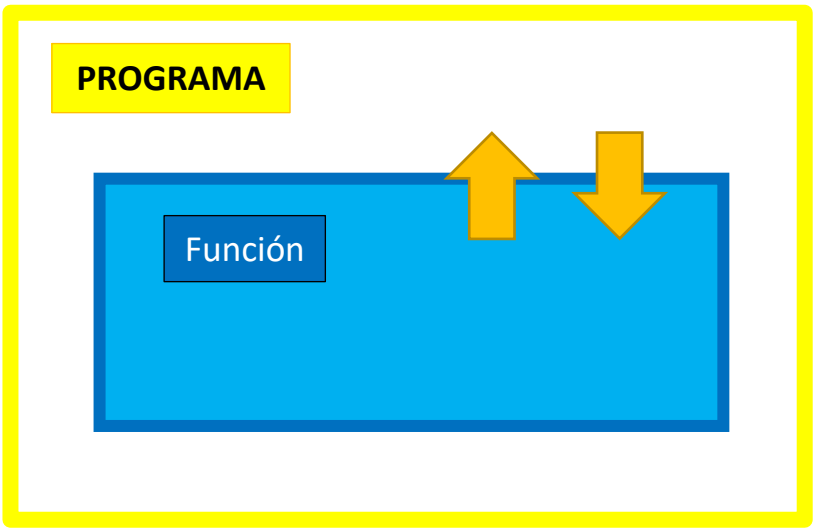
Sintaxis

```
<nombre_de_la_función>()
```



Vamos a guardar y a ejecutar.

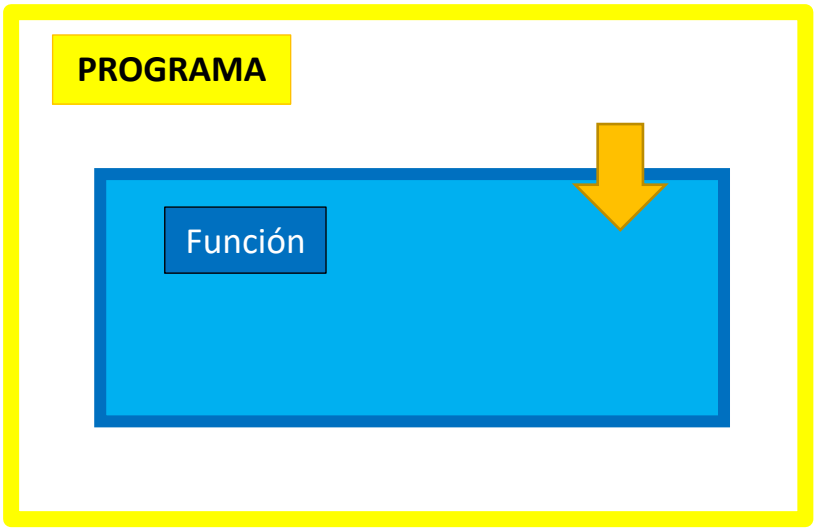




Dentro del programa se encuentran las funciones, pero dentro de estas hay un mini ambiente y espacio con sus propias propiedades y características, así que las puedes considerar como un compartimento dentro del programa.

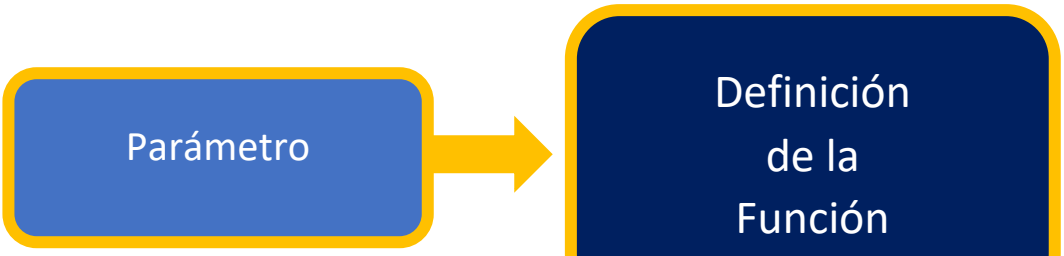
Estos compartimentos también pueden recibir información o enviar información.

¿Cómo puede recibir valores desde el programa principal?



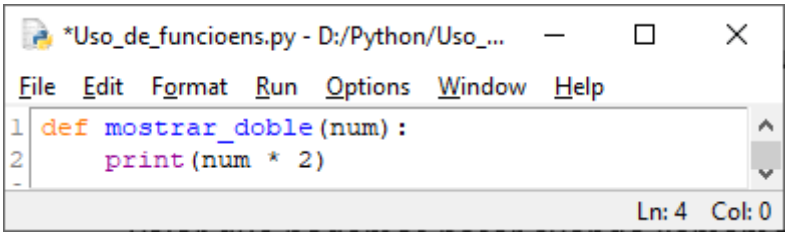
Estos valores que se pueden pasar a una función se denominan Parámetros y los valores que les pasamos se denominar argumentos.

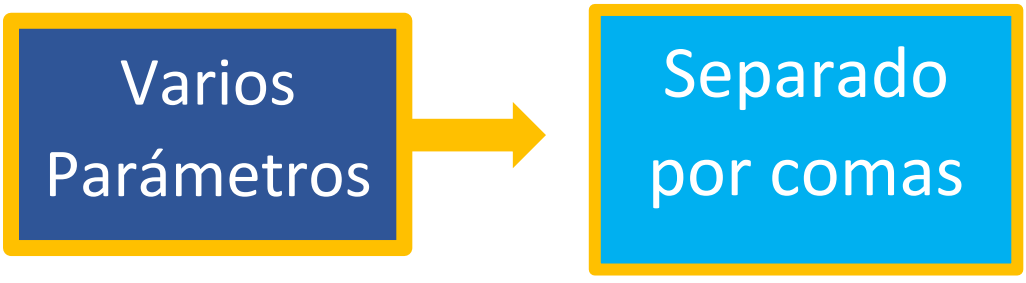
Parámetro: Variable que se incluye en la definición de la función para representar y guardar un valor que podemos pasar cuando llamamos a la función.

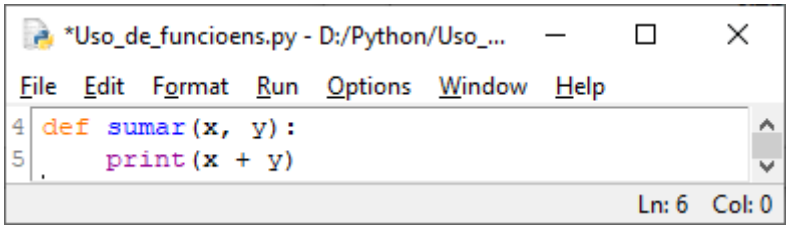



```
def <función>(<parámetro>):
    # Código
```

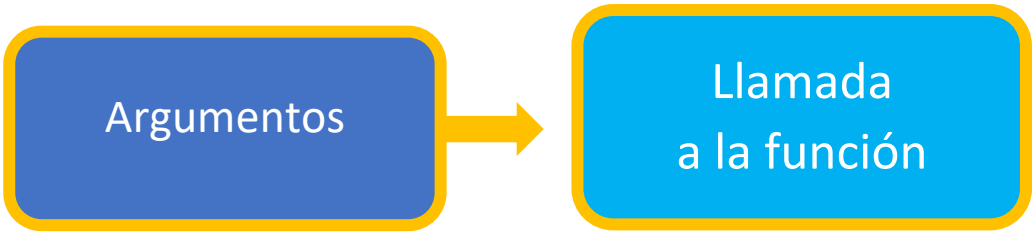
El parámetro se puede usar en el cuerpo de la función como cualquier otra variable.



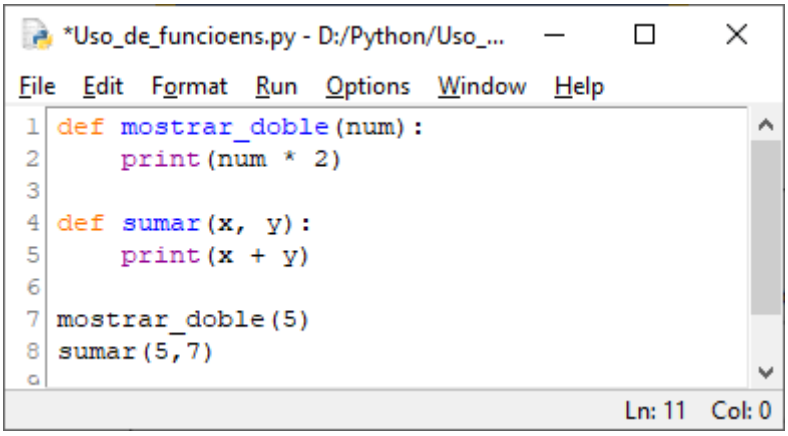




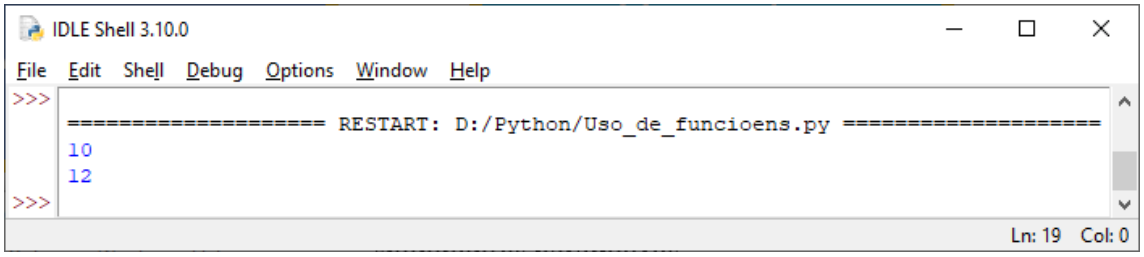
Argumentos: Valor que asignamos a un parámetro cuando llamamos a la función.



Los argumentos son asignados a sus correspondientes parámetros.



Guardamos y ejecutamos.

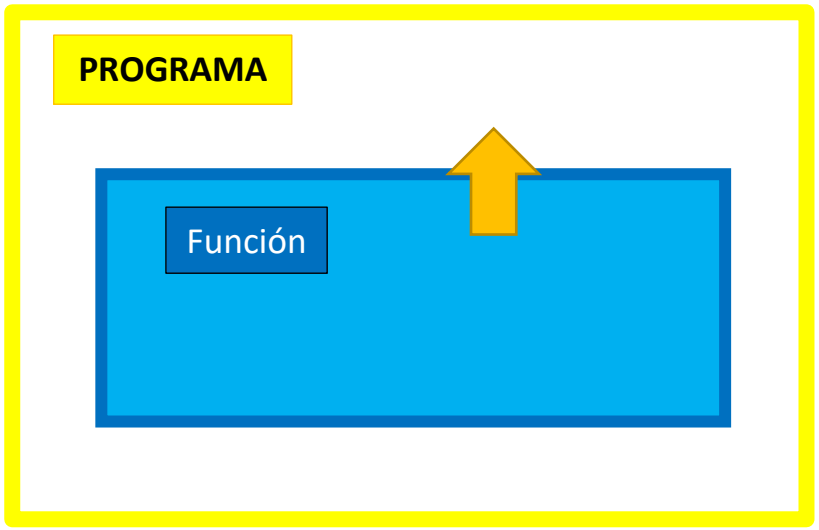


El primer valor es el doble de 5 y el segundo valor es la suma de 5 + 7.

Parámetros para su definición.

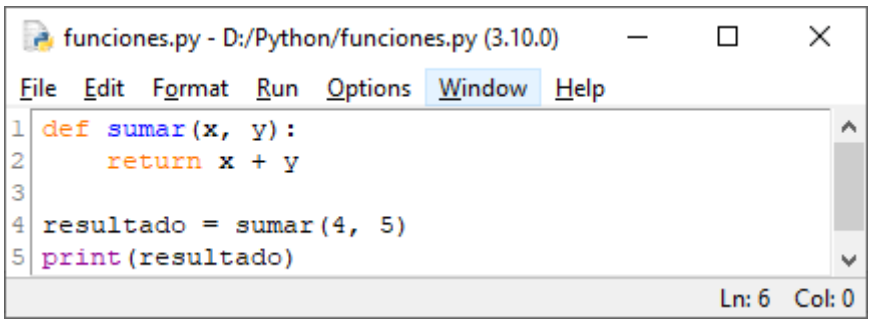
Argumentos para cuando llamamos a la función.

También una función puede retornar valores.



El propósito de esta comunicación es retornar un valor después de completar la tarea.

```
def <función>(<params>):
    # Código
    return <valor>
```

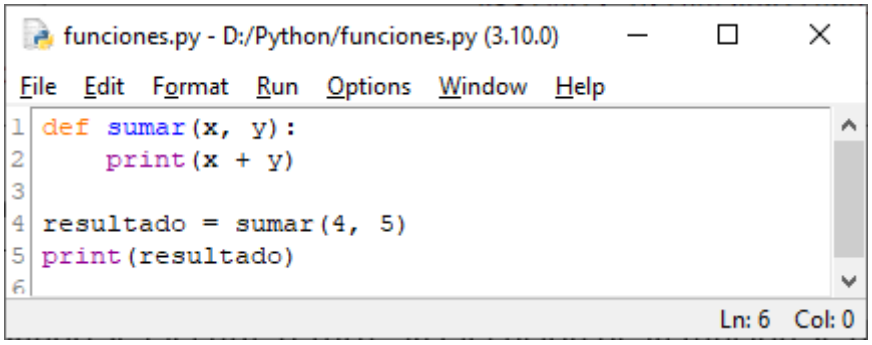


Una vez guardado y ejecutado este será el resultado:

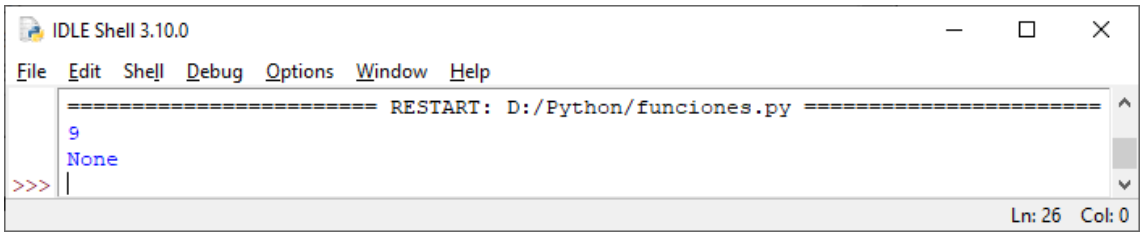
La diferencia es que en lugar de mostrar el resultado lo que hace es retornar el resultado al programa principal y de allí se muestra el resultado.

Cuando se ejecuta 'return', la ejecución de la función se detiene inmediatamente.

¿Qué pasa si no hay 'return' (None)?



Guardamos y ejecutamos.

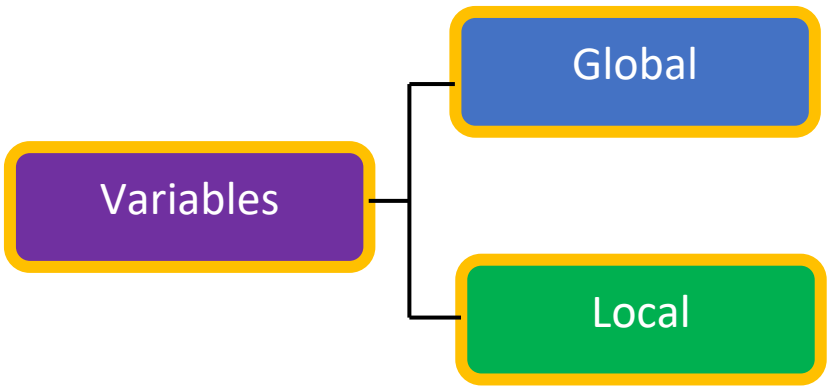


Retorna el valor None (palabra reservada en Python).

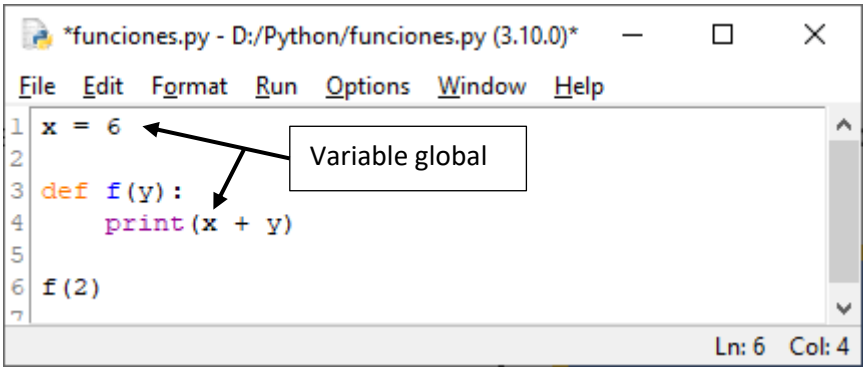
Alcance de una Variable (Scope)

Alcance que tendrá la variable en el programa. Dónde se podrá usar.

Determina a qué variables se tiene acceso en cada parte del programa.

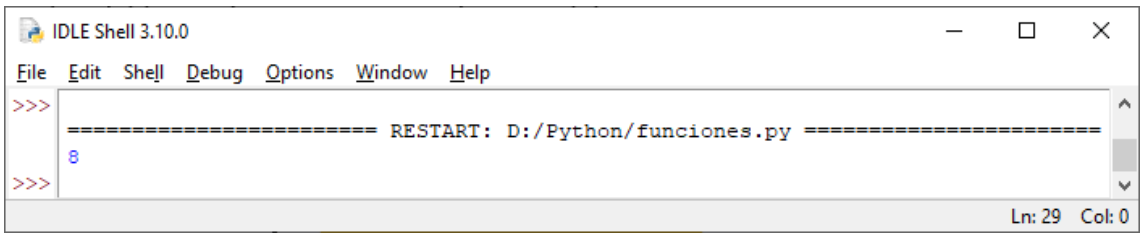


Las variables con scope Global son aquellas que definimos en el programa principal en cambio las variables con scope Local son aquellas que definimos en las funciones.

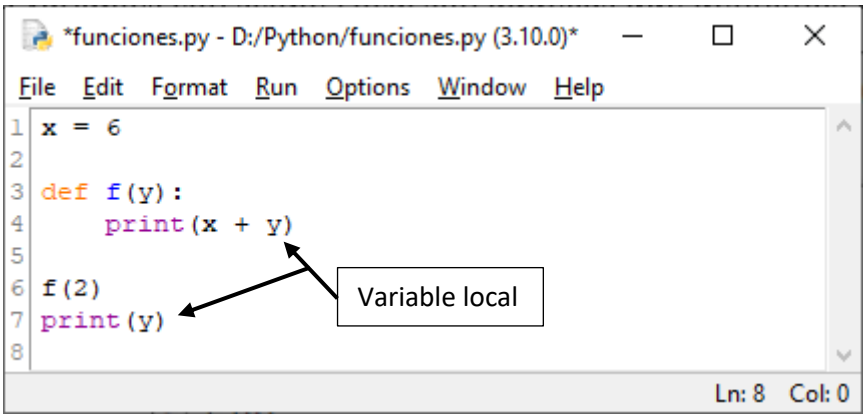


La variable x al ser local la podemos utilizar en la función f(), además la función f() nos pide un parámetro que luego sumará con la variable x.

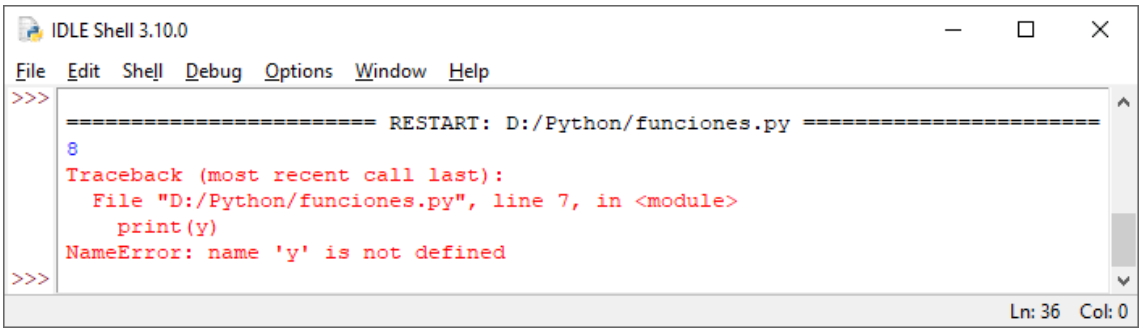
Cuando llamamos a la función F(2) agregando el argumento número 2, verás como realiza la suma del valor de x que es 6 más el 2 que hemos pasado como argumento.



Ahora queremos imprimir una variable local en el programa principal.



Guardamos y ejecutamos.

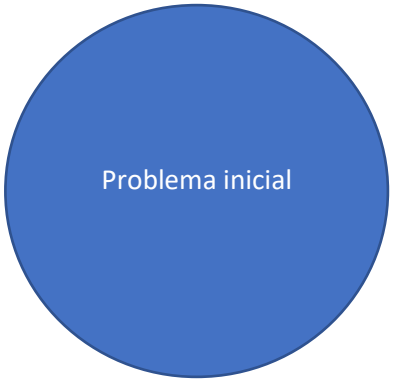


La variable y no la reconoce.

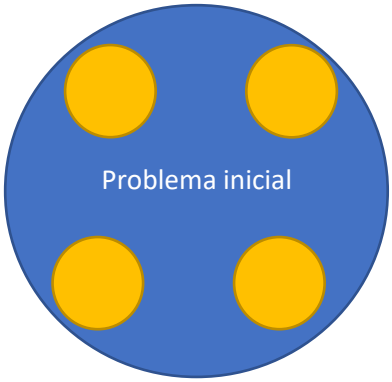
Recursión

Definir algo en términos de sí mismo.

Vamos a resolver un problema inicial.



Este problema inicial lo vamos a dividir en varios subproblemas.



Que se pueden resolver de la misma forma para después poder solucionar el problema inicial.

Vamos a ver un ejemplo con la sucesión de Fibonacci.

0, 1, 1, 2, 3, 5, 8, 13, 21, …

Una sucesión es una secuencia de número que exigen un patrón.

$$fib(n) = fib(n-1) + fib(n-2)$$

Es el resultado de sumar los dos números anteriores, la sucesión inicial con 0 y 1.

Función recursiva: es aquella función que se llama a sí misma.

Tiene dos elementos:

- Caso Base
- Caso Recursivo

El caso Base permite que el proceso termine.

El caso Recursivo no permite en componer un problema en un caso más pequeño de este mismo problema hasta llegar al caso base que detiene el proceso.

```
def Fibonacci(n):
    if n == 0 or n == 1:
        return n
else:
        return Fibonacci(n-1) + Fibonacci(n-2)
```

Caso recursivo

Los dos números Fibonacci previos en la sucesión.

n = posición en la sucesión (iniciando en 0).

Fibonacci.py - D:/Python/Fibonacci.py (3.10.0)

File Edit Format Run Options Window Help

```
1 def Fibonacci(n):
2     if n == 0 or n == 1:
3         return n
4     else:
5         return Fibonacci(n-1) + Fibonacci(n-2)
6
7 print(Fibonacci(2))
```

Ln: 8 Col: 0

Guardamos y ejecutamos.

IDLE Shell 3.10.0

File Edit Shell Debug Options Window Help

```
========== RESTART: D:/Python/Fibonacci.py ==========
1
>>>
```

Ln: 15 Col: 0

Este será el resultado.

Pregunta:

Selecciona el resultado de esta llamada a la función factorial:

```
def factorial(n):
        if n == 0 or  n == 1:
                return 1
        else:
                n * factorial(n)◄—(n-1)

factorial(5)
```

a) 120
b) 25
c) -1
d) Error

El factorial de un número es el producto de todos los números enteros positivos desde 1 hasta ese valor.
Es un error porque a llamada recursiva no se está disminuyendo, tendría que ser n-1.
Vamos a realizar el programa correctamente.

factorial.py - D:/Python/factorial.py (3.10.0)

File Edit Format Run Options Window Help

```
1 def factorial(n):
2     if n == 0 or n== 1:
3         return 1
4     else:
5         return n * factorial(n-1)
6
7 print(factorial(5))
```

Ln: 8 Col: 0

Este será el resultado:

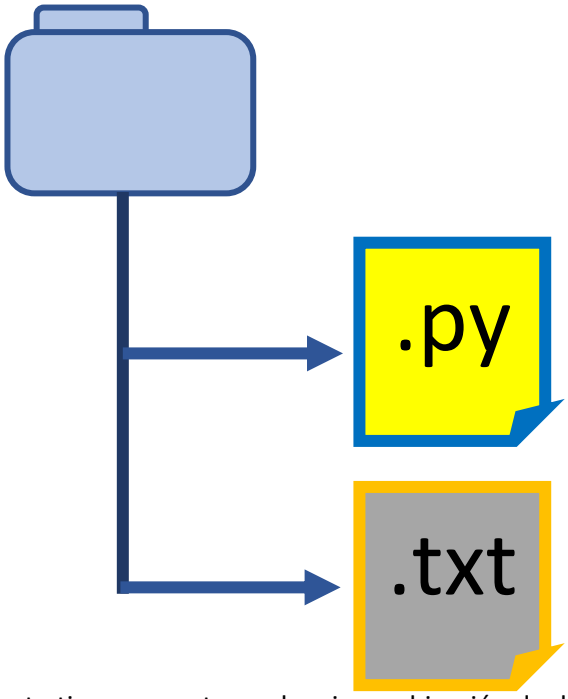
IDLE Shell 3.10.0

File Edit Shell Debug Options Window Help

```
========== RESTART: D:/Python/factorial.py ==========
120
>>>
```

Ln: 20 Col: 0

Archivos



El archivo de texto tiene que estar un la misma ubicación de donde se encuentra el archivo Python.

Sentencia 'with'

Nos permite abrir un archivo y luego cerrarlos automáticamente.

Leer archivos.

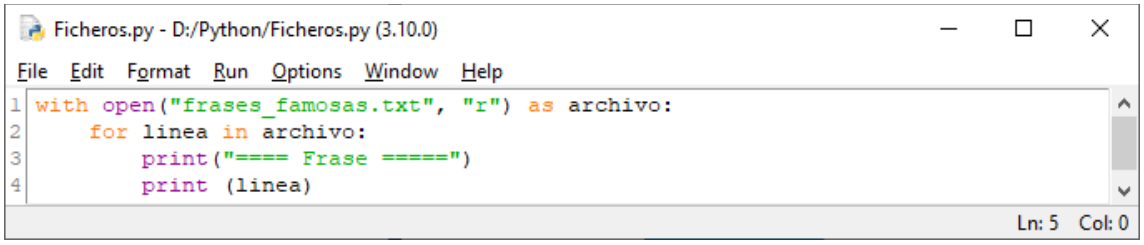
```
with open("<nombre_archivo>.txt", "r") as archivo:
    # Trabajar con el archivo.
```

Variable

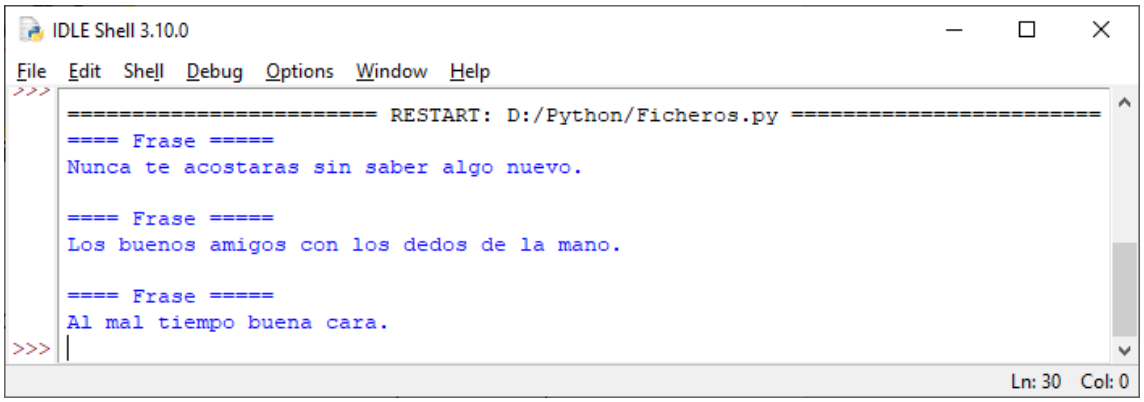
El código del como manejar el archivo tiene que esta dentada.

Vamos a realizar una práctica, para ello desde el bloc de Notas crea un pequeño archivo con frases famosas, lo tienes que guardar en la misma carpeta donde vas a guardar el archivo de Python.

Vamos a realizar el siguiente código:



Una vez guardado lo vamos a ejecutar.



Modos de apertura de archivos:

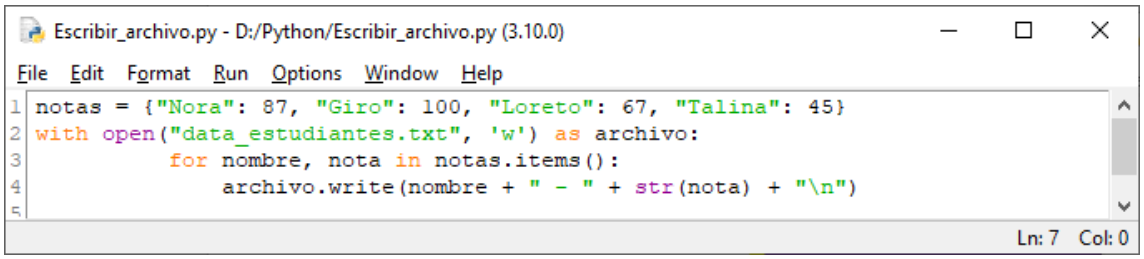
- r (read – leer)
- w (write – escribir)
- a (append – añadir)
- Agregar un + incluye leer. Por ejemplo, w+ es leer y escribir.

Modificar el contenido de estos archivos de texto.

With open("<nombre_archivo>.txt", "w") as archivo:
    # Trabajar con el archivo.

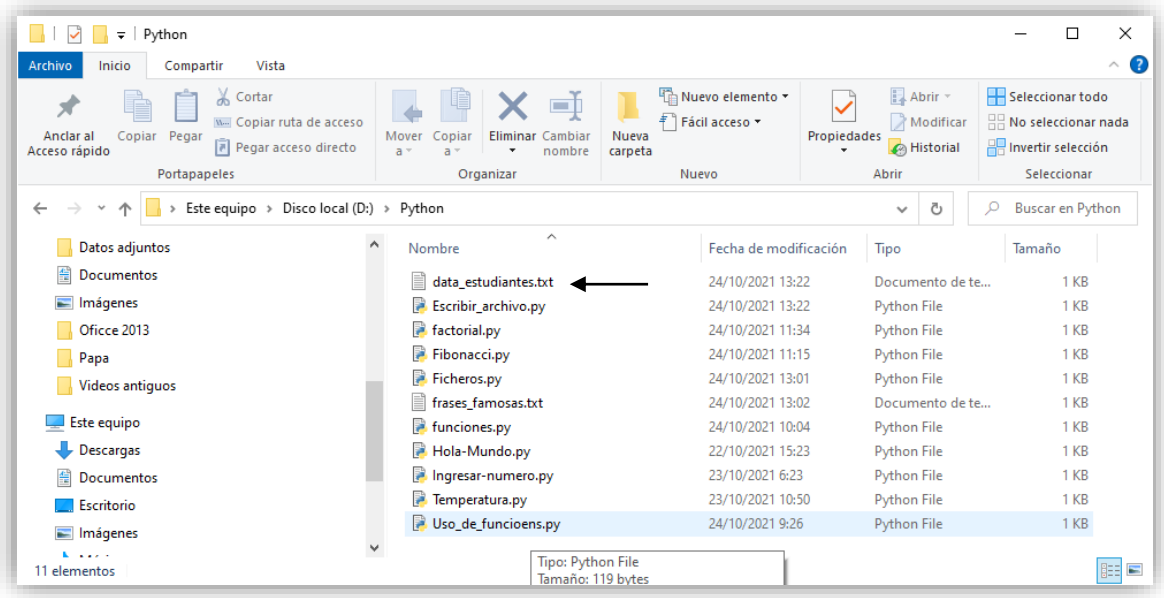
El modo "w" reemplaza el contenido del archivo.

    archivo.write(<contenido>)

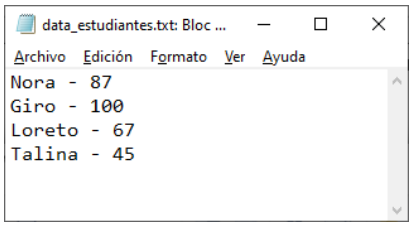


Guardamos y ejecutamos.

Podrás observar que en la consola no aparece nada, ahora vamos a ir a la carpeta donde guardamos el archivo de Python.

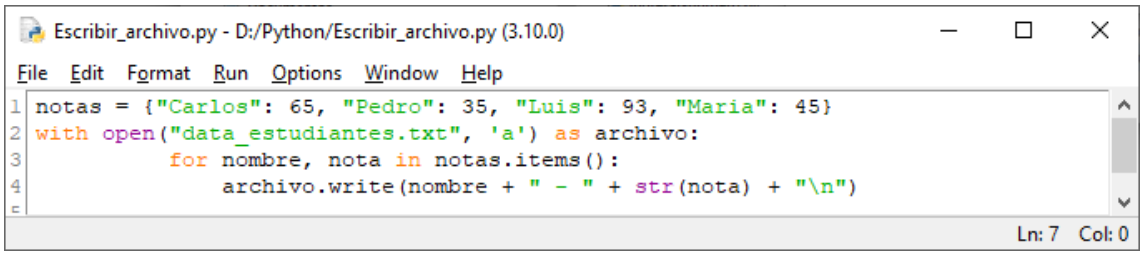


Lo vamos a abrir para poder ver el contenido.

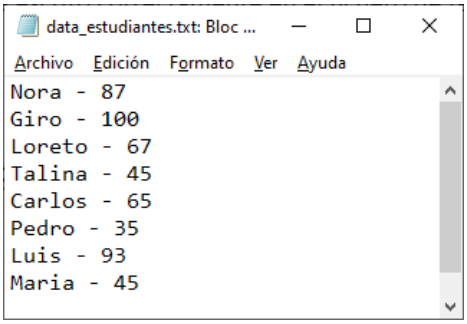


Verás que se ha escrito correctamente en el fichero de texto.
Si queremos agregar mas datos a este archivo.

With open(“<nombre_archivo.txt”, “a”) as archivo:
    # trabajar con el archivo.

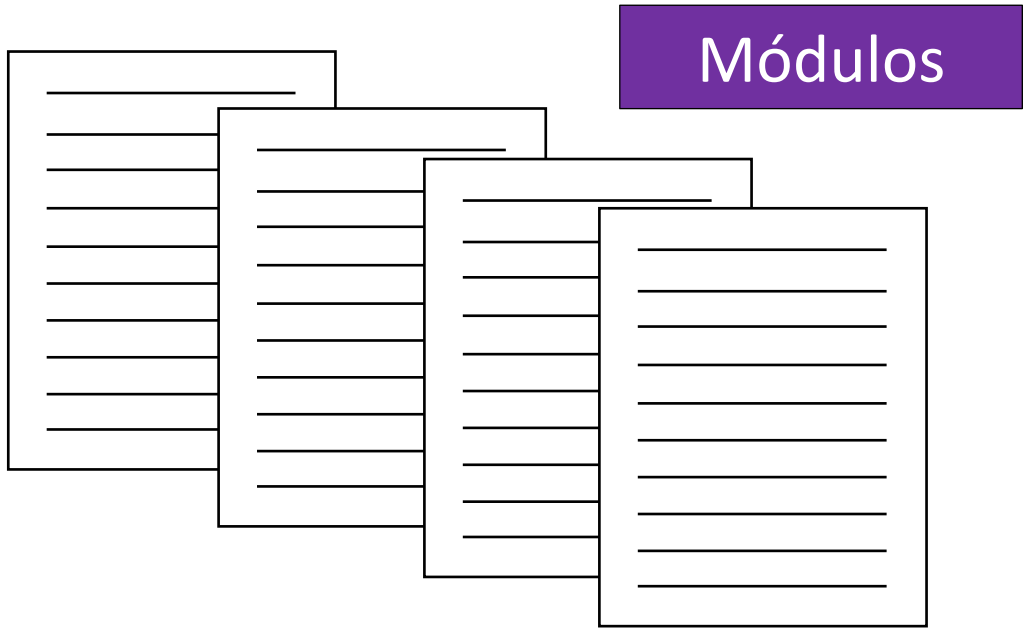


En la consola no sale nada, ahora vamos a ver el contenido del archivo.



Hemos añadido más información.

Importaciones



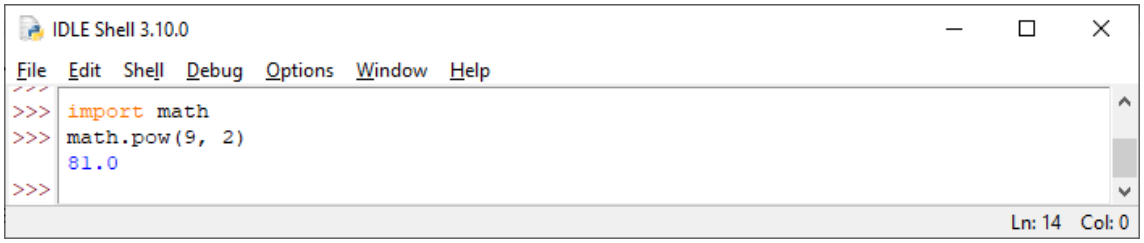
Para hacer más fácil la programación de aquellos proyectos que pueden ser muy grandes, consigue en trabajar por módulos que será llamados desde un archivo principal.
Módulo: Un archivo Python que contiene definiciones y sentencias, que estén relacionados para que pertenezcan al mismo módulo.
Importación: Sentencia que da acceso a las funciones y constantes definidas en el módulo importado.

import <módulo>

luego cuando necesites una función de este módulo.

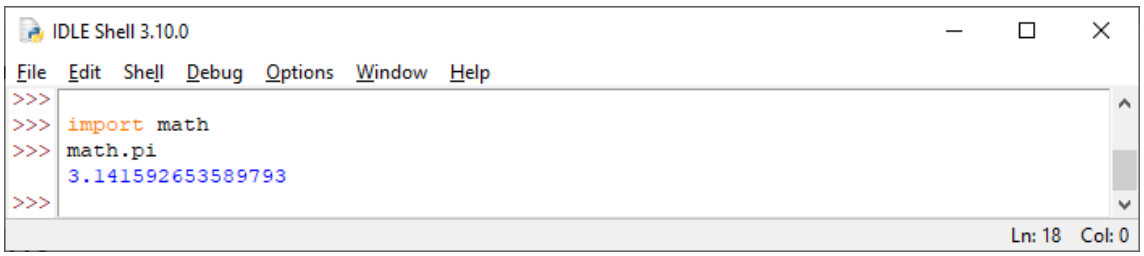
<módulo>.<función>(<args>)

Python tiene muchos módulos incorporados.



Eleva a una potencia.
Para acceder a una constante importada.

<módulo>.<constante>



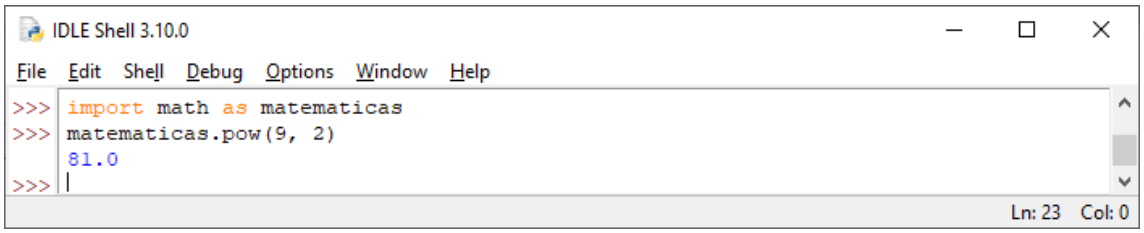
Podemos acceder a la constante Pi.

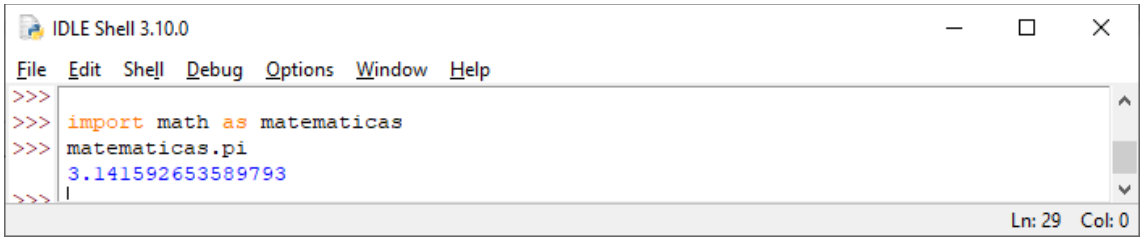
import <módulo> as <nombre_alternativo>

De este modo podremos llamar al nombre alternativo para utilizar funciones y constantes.

<nombre_alternativo>.<función>(<args>)

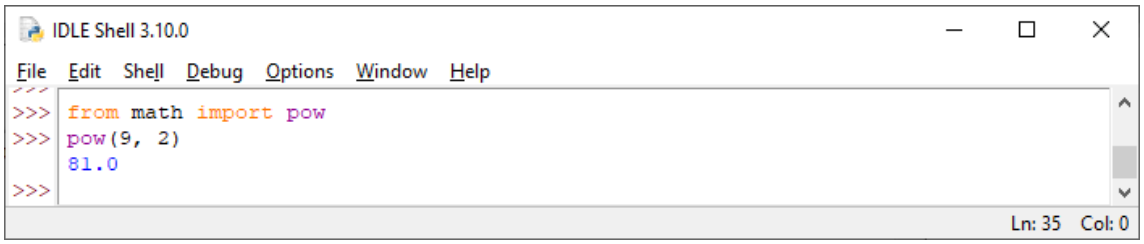
<nombre_alternativo>.<constantes>





from <módulo> import <elemento>

Esto permite importar solo un elemento del módulo sin la necesidad de importar el resto de módulos.

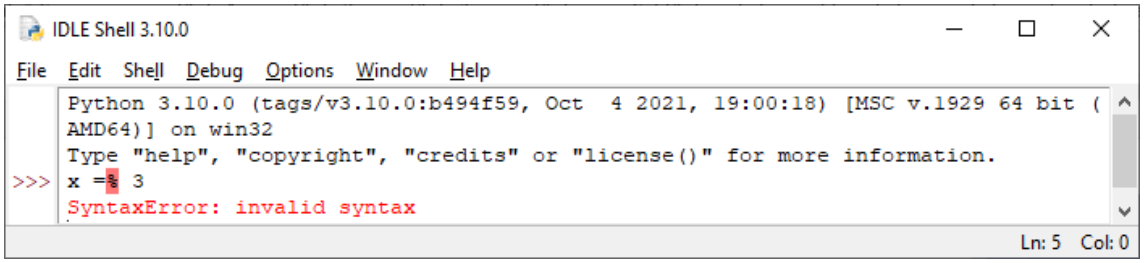


from <módulo> import *

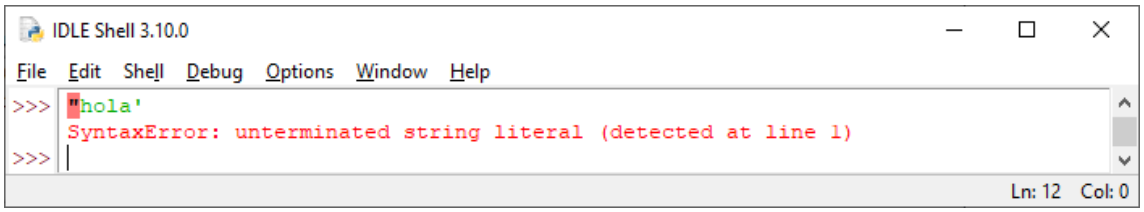
Esta en la forma de importar todos los elementos de un módulo, pero para poder utilizarlos en el programa no vamos a necesitar especificar el módulo del cual proviene sin que los podremos utilizar utilizando solamente su nombre, esto puede crear muchísimas confusiones con esta opción, con lo que se aconseja no utilizarlo.

Errores y Excepciones

SyntaxError: Error en la sintaxis del programa. Ocurre cuando no se siguen las reglas formales para escribir código en Python.



Lo que hemos puesto no es válido.



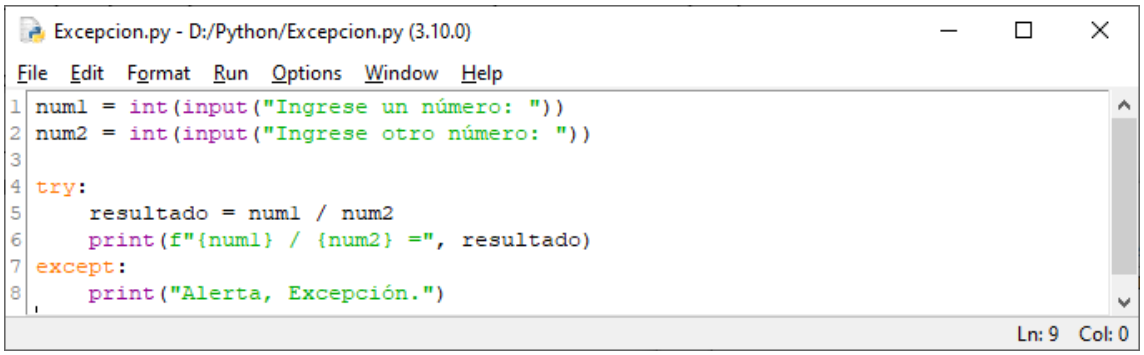
Cuando abrimos con un tipo de comillas y cerramos con el otro.

Hay muchos tipos de errores que no permite que se ejecute el programa.

Excepción: Son errores detectados durante la ejecución de un programa y este se detiene.

Hay una forma de evitar que estas excepciones terminen bruscamente, con la siguiente estructura:

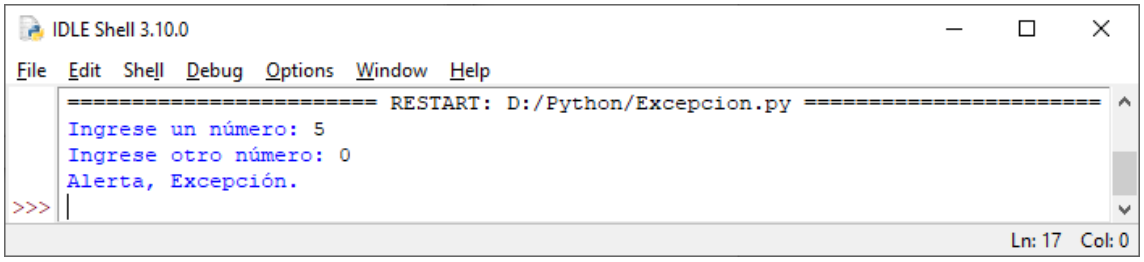
```
try:
    # intenta ejecutar este código
except:
    # si ocurre una excepción, detente
    # inmediatamente y ejecuta éste código
```



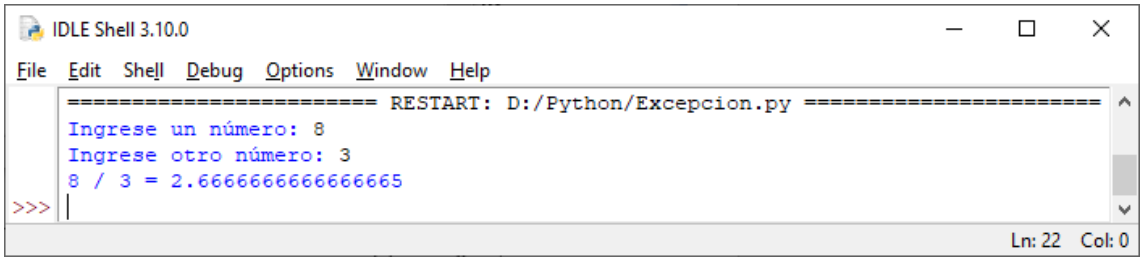
En este ejemplo al usuario se le pide que introduzca dos números que luego se calculará la división.

Supongamos que en el segundo numero el usuario introduce un 0, ya sabemos que la división por 0 dará un mensaje de error e interrumpirá el programa, para evitar esto dentro de la sentencia try: pasa la división a la variable resultado y después muestra el resultado.

Como esta división a generado un error el programa se interrumpe y pasa a la siguiente línea de except: donde nos dará un mensaje de "Alerta, Excepción.", pero el programa no se a interrumpido.

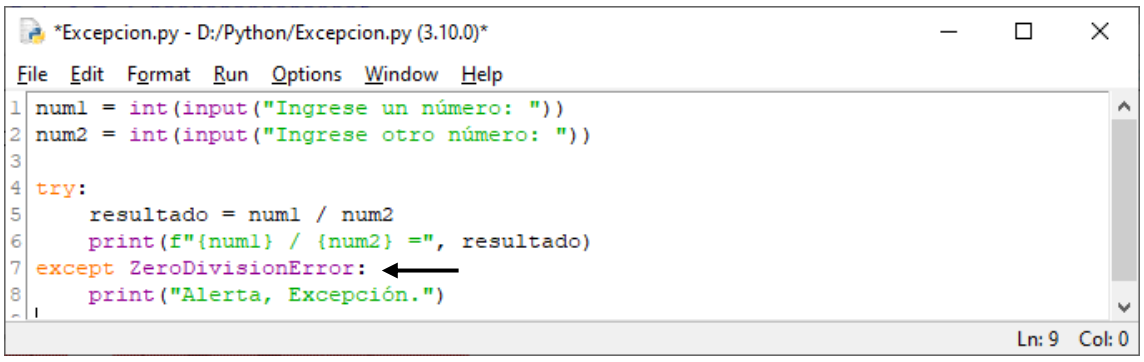


Si no se genera un error este será el resultado.



También podemos especificar que tipo de excepción queremos manejar.

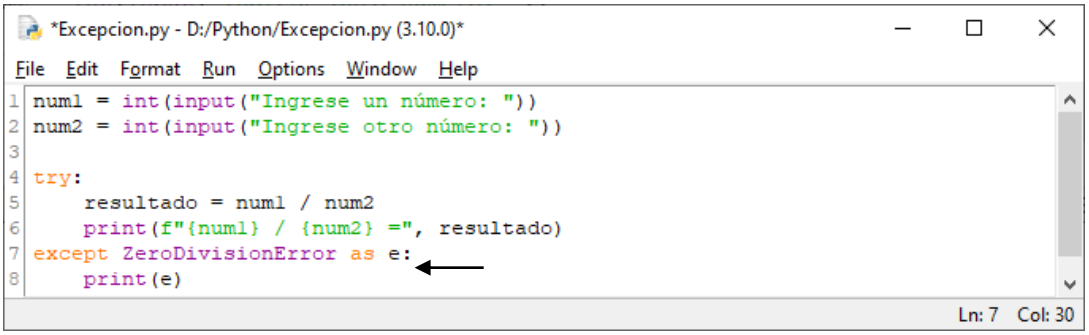
```
try:
    # Intenta ejecutar este código
except: <tipo_de_excepción>:
    # Si ocurre una excepción de este tipo,
    # detente inmediatamente y ejecuta
    # éste código
```



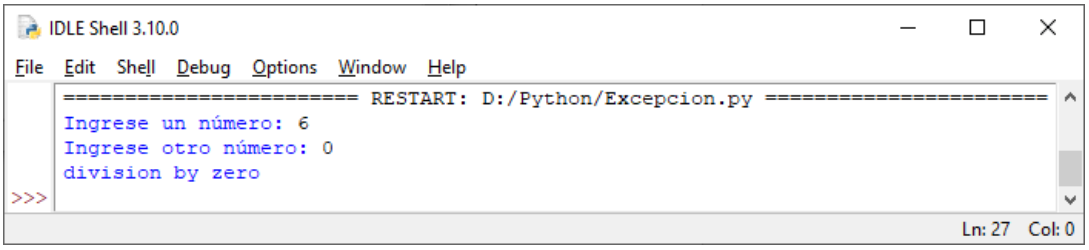
Pero solo vamos a controla este tipo de excepción.

También pueden haber múltiples except para controlar distintos tipos de error.

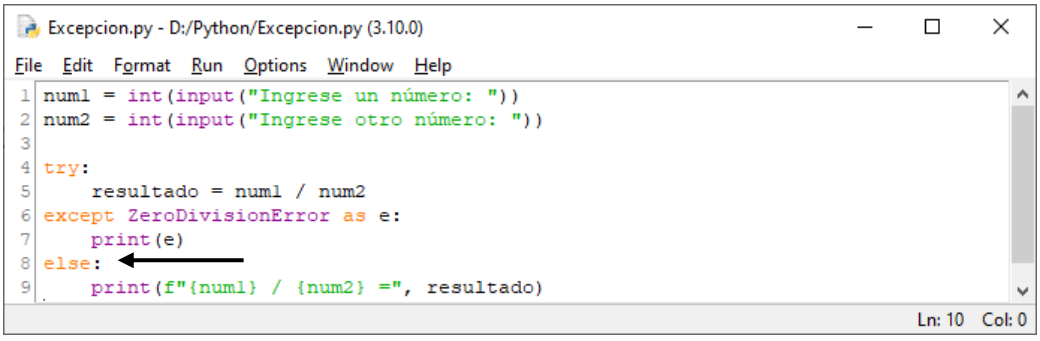
```
try:
    # Intenta ejecutar este código
except <tipo_de_excepción> as <var>:
    # Si ocurre una excepción de este tipo,
    # detente inmediatamente y ejecuta
    # éste código
```



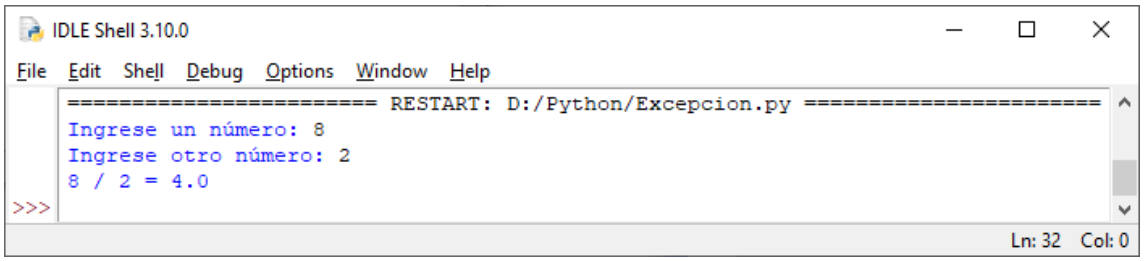
La variable e almacena el tipo de mensaje, que después será mostrado.




```
try:
    # Intenta ejecutar este código
except: <tipo_de_excepción> as <var>:
    # Si ocurre una excepción de este tipo,
    # detente inmediatamente y ejecuta
    # éste código
else:
    # Si no ocurrió una excepción en 'try'
    # ejecuta este código
```

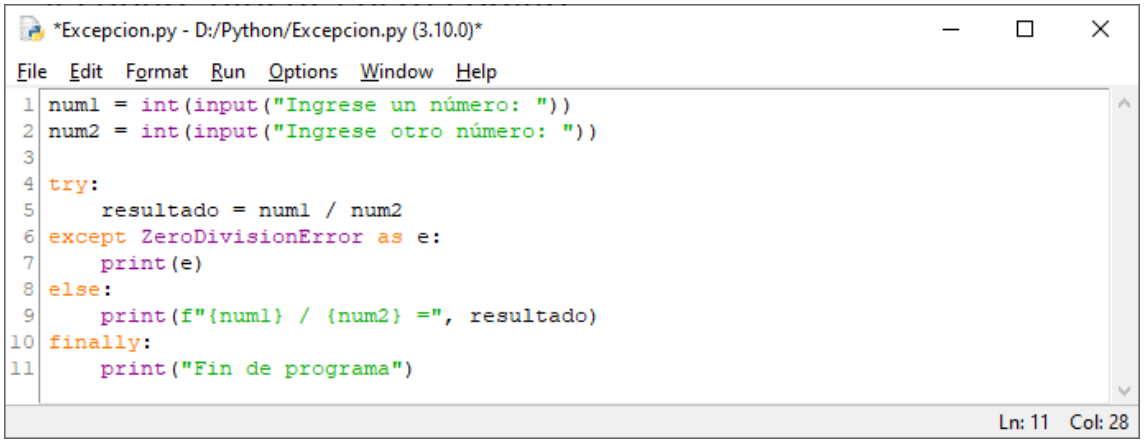


Vamos a guardar y a ejecutar.

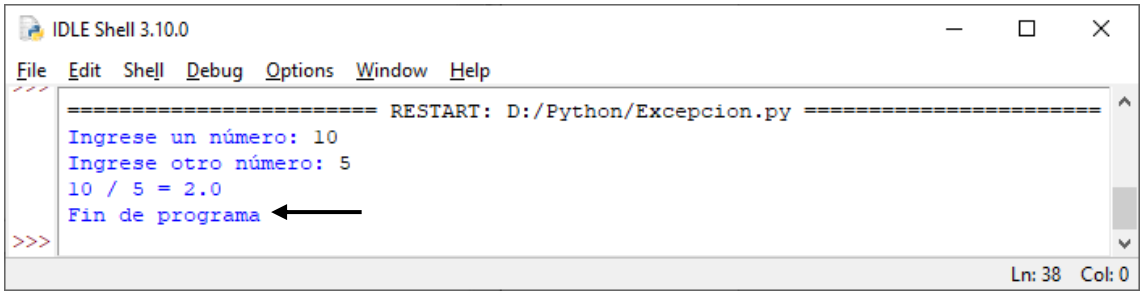


Si no ocurre el error muéstrame el resultado.

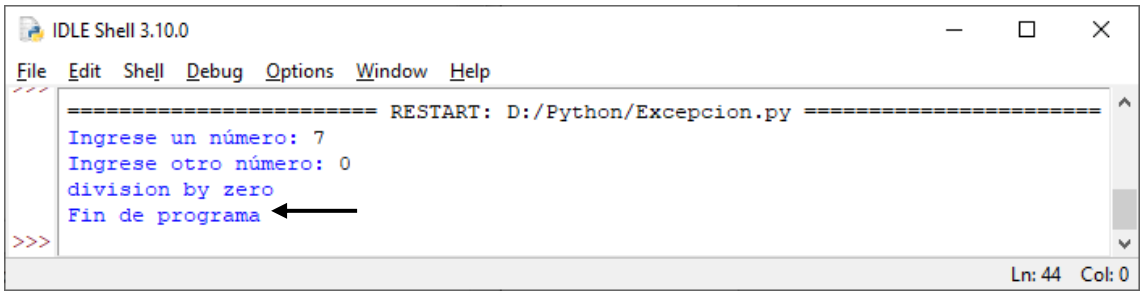
```
try:
    # Intenta ejecutar este código
except: <tipo_de_excepción>:
    # Si ocurre un excepción de este tipo,
    # detente inmediatamente y ejecuta
    # éste código
finally:
    # Luego, ejecuta este código.
```



La opción finally: se ejecutará tanto si se genera un error como si no.
Vamos a probar introduciendo valores correctos.



Lo vamos a ejecutar de nuevo introduciendo como segundo valor un 0 así provocamos un error.



Hay operaciones que siempre se tienen que hacer, como por ejemplo cerrar un archivo.
Esta es la estructura final.

```
try:
    # Intenta ejecutar este código
except: <tipo_de_excepción>:
    # Si ocurre un excepción de este tipo,
    # detente inmediatamente y ejecuta
    # éste código
else:
    # Si no ocurrió una excepción en 'try'
    # ejecuta este código
finally:
    # Luego, ejecuta este código.
```

Programación Orientada a Objetos (POO)

Hasta ahora hemos ejecutado programas que se ejecutan línea por línea incluyendo ciclos, condicionales, sencillo leer línea por línea.

Programación orientada a objetos es un paradigma de programación especifico, en el cual nosotros representamos objetos, entidades o conceptos de la vida real o abstractos.

Estos tipos de objetos se denominar clases.

Las clases nos permiten trabajar con atributos y funcionalidad de un objeto (métodos) específicos para cada clase.

Los métodos son muy similares a funciones pero van a estar relacionado con un objeto.

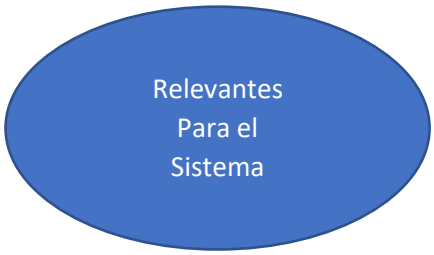
Los atributos son valores que van a tener estos objetos.

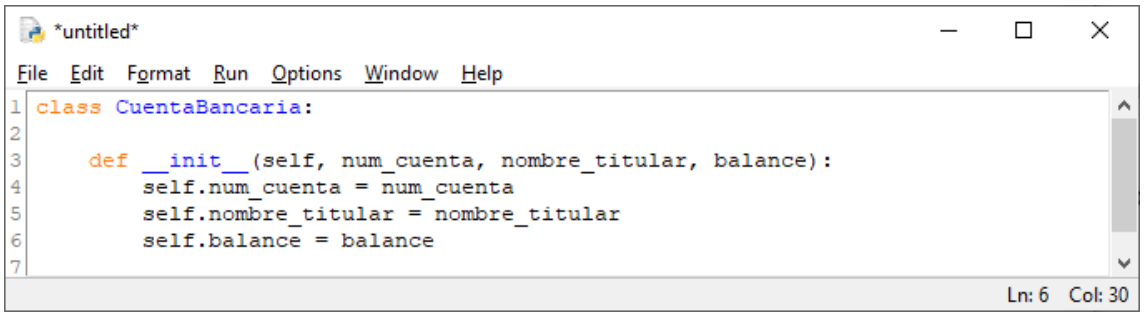
Vamos a realizar un ejemplo con una cuenta bancaria.

Tenemos que diseñar un plano para una cuenta bancaria muy sencilla.

Atributos:

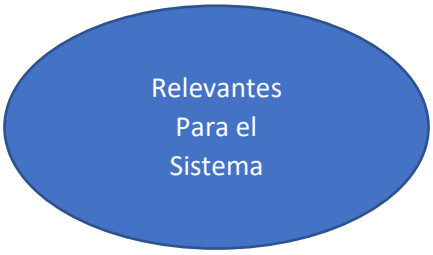
- Número de cuenta
- Nombre del titular
- Balance Inicial
- Balance Actual
- Fecha de Apertura
- …

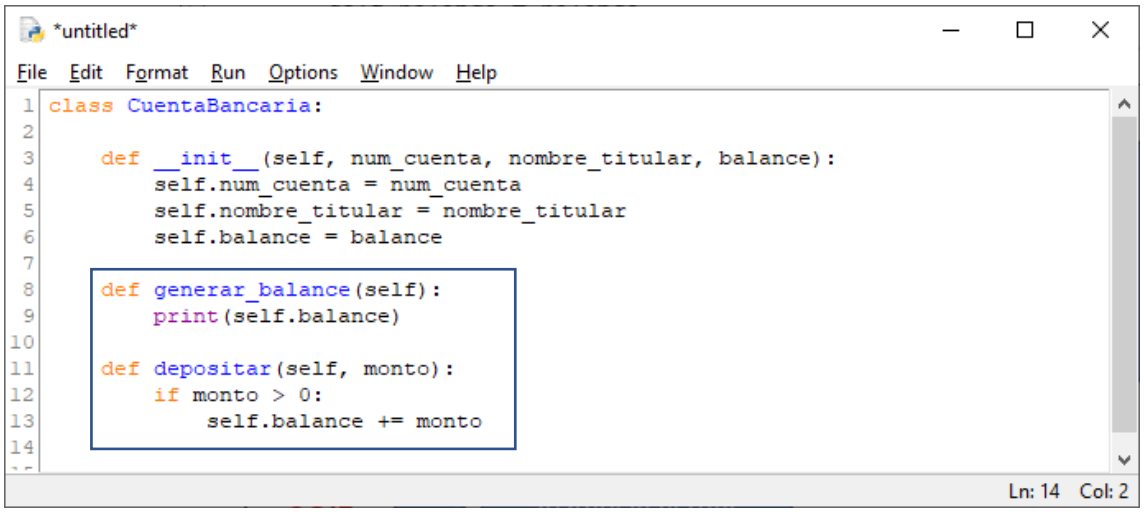




Funcionalidad:

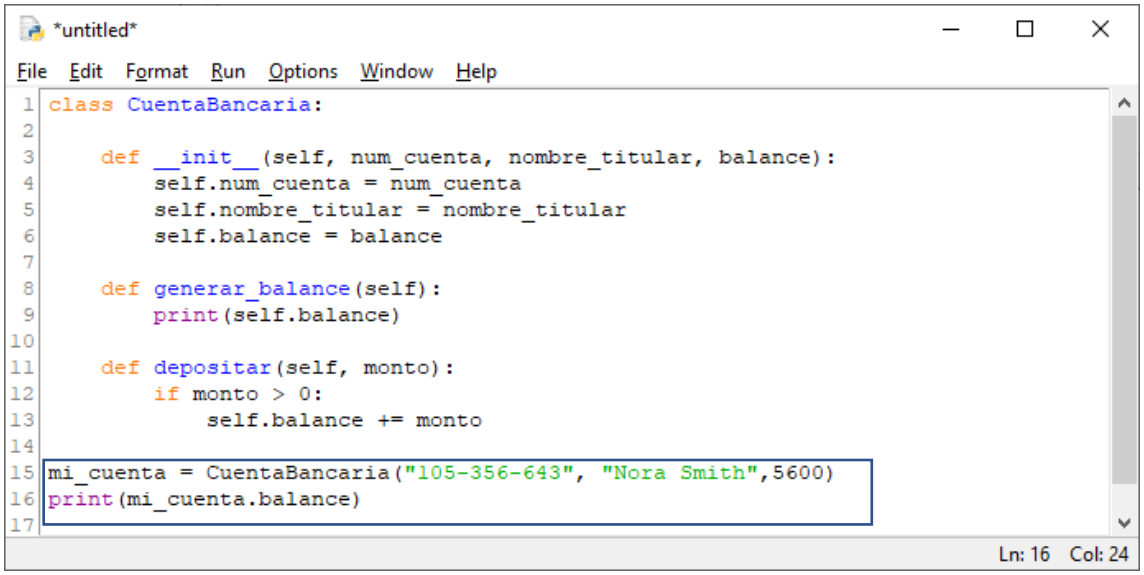
- Retirar
- Depositar
- Generar Balance
- Actualizar Datos
- …



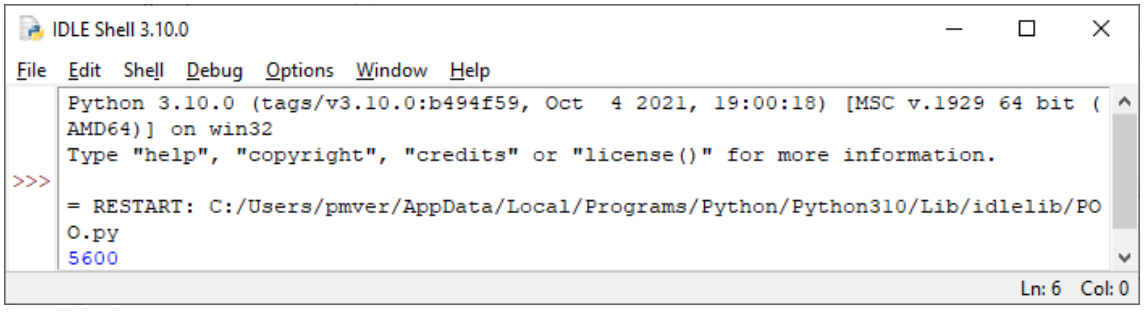


Tenemos 3 cuentas bancarias (Instancias).

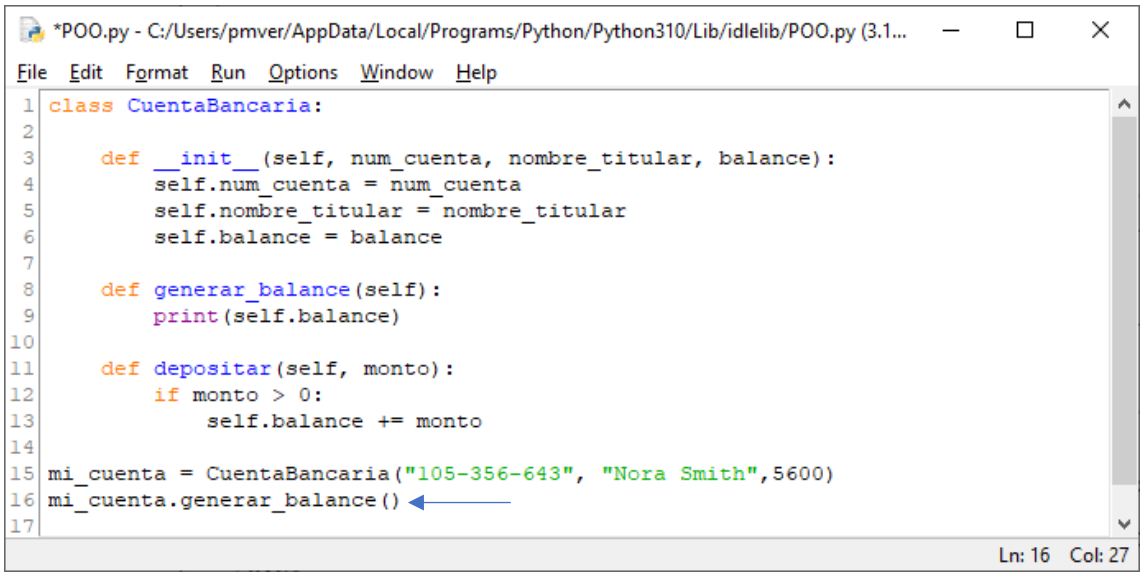
- 0103-231-234-232
- 0103-244-532-552
- 0103-654-234-242



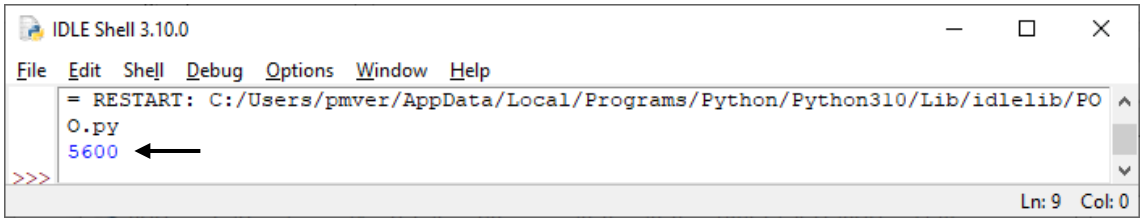
Vamos a guardar y ejecutar.



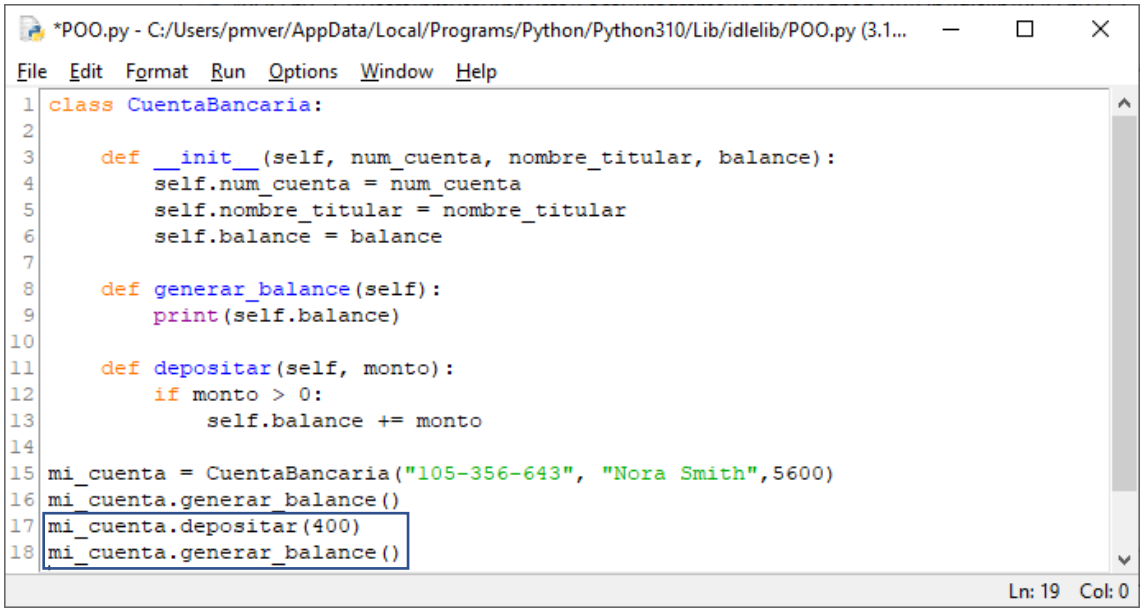
Vamos a llamar el método generar_balance.



Vamos a guardar y ejecutar.

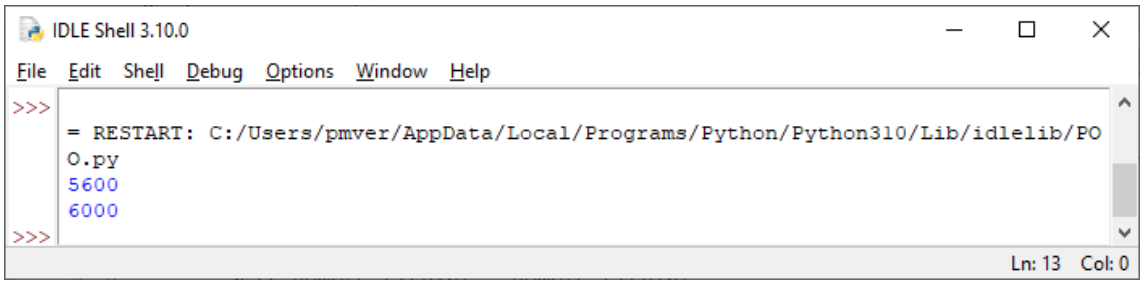


Vamos a llamar al método depositar.



Depositamos 400 euros y después volvemos a consultar el balance.

Guardamos y ejecutamos.



Antes del ingreso teníamos 5600 hemo ingresado 400, pues el balance actual es de 6000.

Contenido

¿Por qué aprender a programar? 1
Instalar Python 2
Introducción a IDLE. 4
Tu primer programa 8
Variables 10
Reglas para nombrar variables 11
Tipos de datos 12
Numéricos: 12
Cadena de caracteres 14
Rebanado (Slicing) 16
Recibiendo datos del usuario 19
Operadores 21
Sentencias condicionales 29
Comentarios 33
Listas 34
Tuplas 38
Diccionarios 40
Documentación 42
Ciclos For 47
Ciclos While 51
Funciones 53
Recursión 61
Archivos 64
Importaciones 68
Errores y Excepciones 70
Programación Orientada a Objetos (POO) 75