SISTEMA DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Python
comando pip
IDE PyCharm
palabras reservadas
print() e input()
operadores

¿Qué es Python?

Python fue creado a finales de los ochenta por Guido Van Rossum. A priori, se puede pensar que el nombre proviene de la serpiente pitón (del inglés *python*), pero no es así. Realmente, se debe a la afición que su creador tenía hacia los *Monty Python*, un grupo británico de humoristas con películas como "*La vida de Brian*" (1979).

Python es un lenguaje de programación:

- Interpretado.
- Multiparadigma.
- De alto nivel.
- Tipado dinámico y fuerte.

Características de Python

Python es un lenguaje de programación:

- Interpretado, es decir, que a diferencia de los compilados que requieren de un paso previo antes de ser ejecutado (la compilación), para convertir el lenguaje de alto nivel en código máquina, en los interpretados se va **convirtiendo a medida que se va ejecutando**.
- Multiparadigma, puesto que soporta **programación orientada a objetos**, **imperativa** y **funcional**.
- De alto nivel, ya que expresa los algoritmos de una manera **fácilmente comprensible** por el ser humano.
- Tipado dinámico y fuerte, puesto que la comprobación de la tipificación se hace durante la ejecución y además **no permite violaciones de los tipos de datos**, por ejemplo, no permite operar un carácter con un número. Si tenemos un número, las operaciones deben ser con números y lo mismo con caracteres.

El **núcleo del lenguaje es su gramática**, definida oficialmente en el siguiente vínculo:

https://docs.python.org/3/reference/grammar.html

A la hora de preguntarnos **por qué usar Python**, podemos llegar a las siguientes consideraciones:

- Es open source.
- Es multiplataforma (Mac, Windows y Linux).
- Es parecido a leer y escribir en inglés.

• Viene con una **librería estándar** (https://docs.python.org/3/library/index.html) muy completa que le permite **interaccionar con otros lenguajes**, bases de datos, etc.

• Dispone de un gran abanico de **librerías de terceros** (muchas empaquetadas en https://pypi.org/) para afrontar problemas como la **inteligencia artificial** (Keras, Tensorflow), **Big Data** (Pydoop), **Data Science** (Pandas, Numpy), **Testing** (Pytest, Robot), **web** (Django), **Scrapping** (Urllib, selenium).

• Es el lenguaje con **mayor tasa de crecimiento** en los últimos años. Según el informe anual de GitHub, lo que ellos llaman el "Estado del Octoverso", Python **superó a Java en 2019** y se colocó en segunda posición, justo por detrás de JavaScript.

Instalación de Python

Para instalar Python en Windows, bastará con descargarlo desde su web oficial (https://www.python.org/downloads/) y, seguidamente, ejecutarlo.

En cambio, en **Mac** ya existe una **versión preinstalada**, ya que **macOS lo usa** para su propio funcionamiento. No obstante, también **es posible instalar otra versión**, siendo los pasos idénticos que en Windows.

En **GNU/Linux**, al igual que pasaba en Mac, aquí también existe una **versión preinstalada**, porque lo usan también como **recurso propio**. Para **instalar otra versión**, lo más fácil es usar el **administrador de paquetes**.

```
sudo apt-get install python3
```

Como nota, indicar que se puede tener varias versiones de Python instalada en el mismo PC, no existe contraindicaciones sobre esto.

Una vez instalado, por ejemplo, en **Windows**, bastará con abrir '**símbolo del sistema**' y empezar a usar Python. El terminal tiene dos modos de trabajo:

1. Interactivo: las **instrucciones** que vayamos escribiendo, las irá ejecutando **una a una**. Para ello, deberemos ejecutar el **comando python (o python3 en Linux)**. Un mensaje con la versión de Python abierta y los '>>>' nos indicarán que estamos en Python y ya podremos realizar acciones desde el terminal:

Ejemplo de sesión de Python interactivo en Linux

```
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Para salir de Python, bastará con ejecutar el comando exit() o Ctrl + d.

2. Ejecutando scripts: los scripts son archivos con **extensión ".py"** que se pueden editar con cualquier editor de textos, como, por ejemplo, Notepad, y en cuyo interior está el código implementado. Bastará con ejecutarlos usando el comando:

‘**python**’ + **<nombre del archivo>**

Creación y ejecución de script Python desde el terminal en Linux

```
marc@debian-12:~$ cat > script.py
print('Hola Mundo')
marc@debian-12:~$ python3 script.py
Hola Mundo
marc@debian-12:~$
```

Instalando Python en Windows

Describiremos cómo instalar Python en nuestro PC con un sistema operativo Windows:

Para ello, en primer lugar, deberemos entrar en la web oficial de Python, que es python.org, y pinchar en la pestaña de descargas, concretamente en All Release, porque desde aquí podremos ver todas las versiones de Python y descargarnos la que más nos interese. Están ordenadas por fecha de más moderna a más antigua.

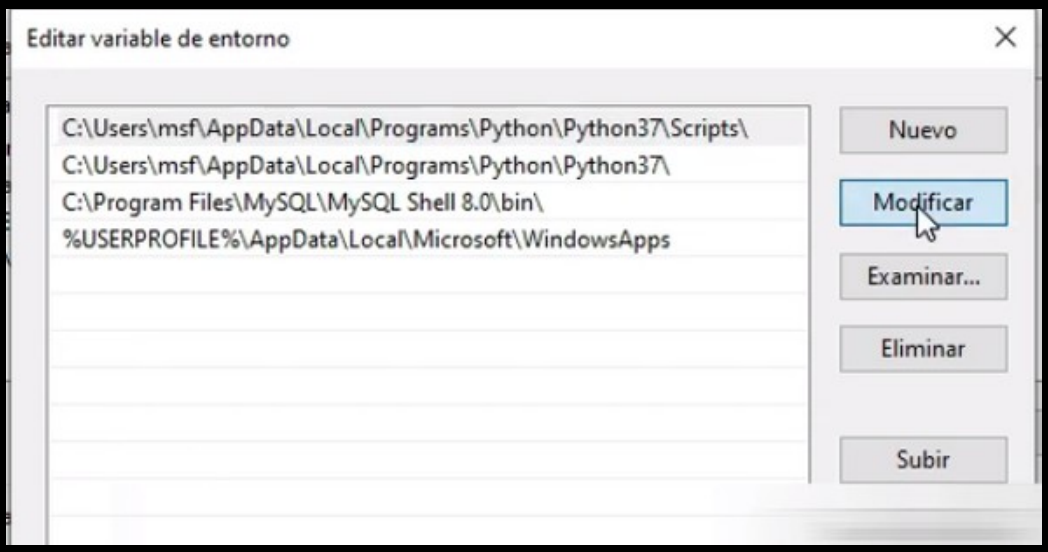
En nuestro caso, nos vamos a descargar la versión más actual para Windows. Como nuestro sistema operativo es de 64-bit, podremos pinchar sobre Windows, que es nuestro sistema operativo, y aquí en la última versión que es la 3.8.3, podremos bajárnosla, la versión de 64-bit.

Una vez descargada, la abriremos y seguiremos los pasos del instalador. Podremos tener **diferentes versiones de Python instaladas** en el mismo PC. Esperamos que termine la instalación y una vez finalizado, cerraremos la ventana.

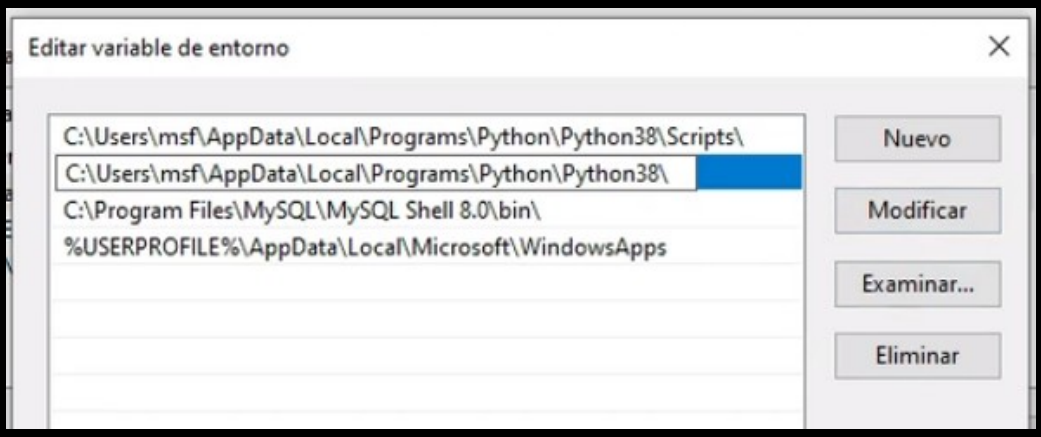
Tras esto, ya podremos abrir la consola de Windows y escribir Python. En este caso, nos ha abierto la versión 3.7.4, que la tendríamos también instalada en nuestro ordenador.

Si quisiéramos abrir la versión 3.8.3, es decir la última que hayamos instalado que es la que actualmente acabamos de abrir, (en vuestro caso será otra versión más reciente), para ello tendríamos que entrar a modificar las variables del entorno:

> Entraríamos en las propiedades de este equipo → en configuraciones avanzadas del sistema → variables del entorno → path.
>
> Aquí podemos ver que la versión que tenemos es la 3.7, tanto la versión en script como sin script.



Pues tendríamos que modificar y ponerle un 8 y aquí igual, porque como vemos en la misma ruta está tanto la versión 3.7 como la versión 3.8 y guardamos los cambios.



Tras guardar los cambios, tendremos que abrir el terminal de nuevo, reinicializarlo y ahora si ejecutamos el comando Python ya si vemos que es la versión 3.8.3 y desde aquí ya podremos realizar las operaciones oportunas.

IDE PyCharm

Usaremos la versión gratuita (**Community Edition**) del IDE PyCharm. Este está desarrollado por JetBrains y es multiplataforma. Para instalarlo, bastará con descargarlo de su **página oficial**:

https://www.jetbrains.com/es-es/pycharm/

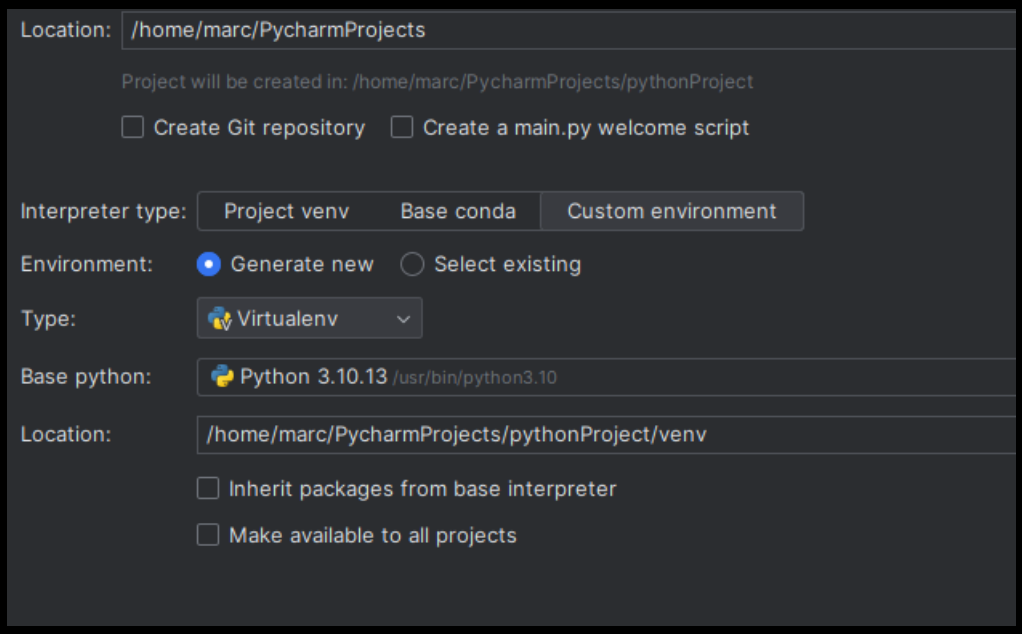
Este IDE nos permitirá, entre otras cosas:

- **completar código** con atajos de teclado,
- **navegar** por el código (por ejemplo, entre las clases),
- **refactorizarlo** (o limpiarlo),
- **detectar errores** o advertencias,
- usar **plugins** que permitirán, por ejemplo, interaccionar con otros lenguajes y frameworks (como Node JS),
- un **acceso a base de datos** más fácil,
- opción de **debugging**,
- etc.

Para comenzar a trabajar con PyCharm, crearemos un nuevo proyecto:

A la hora de crear un proyecto nuevo, PyCharm nos da la opción de seleccionar un **nuevo entorno** usando Virtualenv o un intérprete existente. La diferencia entre una y otra opción es que si se usa el entorno Virtualenv para el proyecto, todos los **paquetes que se agreguen, modifiquen o eliminen solo afectarán a ese proyecto**. En cambio, si se usa el intérprete existente, los cambios que se hagan **afectarán a todos los proyectos** que use dicho interprete.

Nuevo proyecto Python en PyCharm



En nuestros proyectos, por ahora, usaremos el entorno Virtualenv.

Podremos encontrar más información sobre la configuración del intérprete en el siguiente enlace:

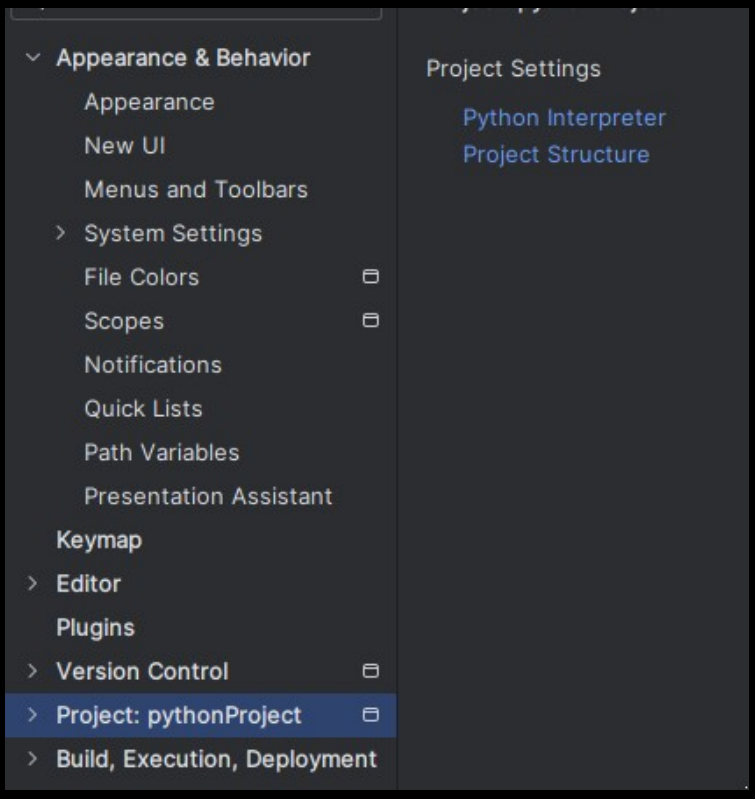
https://www.jetbrains.com/help/pycharm/configuring-python-interpreter.html#add-existing-interpreter

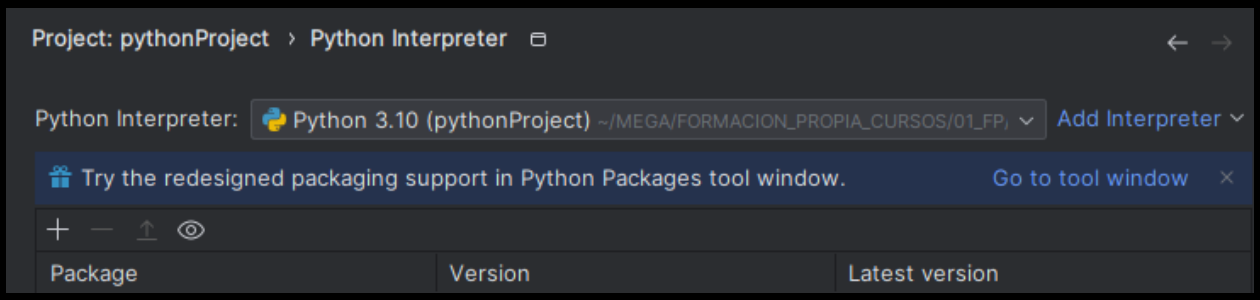
Hola mundo

Se deberá **crear el fichero Python** en nuestro proyecto. Para ello, bastará con pulsar el botón derecho sobre la carpeta del proyecto, y seguidamente, en 'nuevo' y 'archivo Python'. Esto nos creará un archivo '.py'.

A continuación, habrá que **configurar el intérprete** para que PyCharm sepa dónde está el archivo a ejecutar y con qué interprete ejecutarlo. Para ello, arriba a la derecha, junto al icono de 'play', se pulsará en '**editar configuración**' e informaremos de la ruta del archivo Python y el intérprete.

Configurando el intérprete



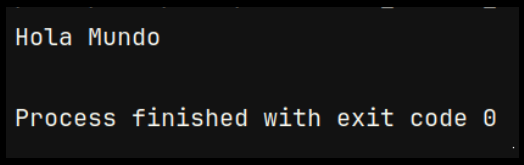


Con esto, ya estaremos en disposición de crear y ejecutar nuestra primera práctica. Para ello, usaremos el comando para que nos muestre por pantalla “Hola mundo”:

```
print('Hola Mundo')
```

Finalmente, para ver el resultado, pulsaremos sobre el icono de ‘play’ , que nos mostrará el resultado en la consola de PyCharm:

Visualizando el resultado de la ejecución



El comando pip

Es el acrónimo de '**Pip Instalador de Paquetes**' o 'Pip Instalador de Python', y es un **sistema de gestión de paquetes** usado para **instalar paquetes escritos en Python**.

En el siguiente enlace, se podrá encontrar toda la información referente a pip: https://pip.pypa.io/en/stable/

De esta forma, por ejemplo, para instalar librerías de terceros, una vez que conozcamos el nombre de la librería, bastará con usar el comando pip en el terminal (fuera de Python) con la instrucción install.

Por ejemplo, para instalar la librería redis, bastará con escribir el siguiente comando en el terminal:

Instalando una librería con el comando pip

```
pip install redis
```

El uso de **librerías de terceros**, dada la gran penetración del lenguaje entre el sector del desarrollo, hace que Python se convierta en una de las **opciones más potentes y más completas** a la hora de afrontar **cualquier proyecto de programación**.

Palabras reservadas y comentarios

En Python, nos encontramos con **treinta y tres palabras reservadas**:

https://www.w3schools.com/python/python_ref_keywords.asp

and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To return a list of values from a generator

Con respecto a los comentarios, estos pueden ser mono-línea, usando **#**, o multi-línea, usando **tres comillas dobles (“””)** al inicio y al final del comentario.

```
# Este es un comentario mono-línea

"""
Y este es un comentario multi-línea
"""
```

Tipos de datos

En primer lugar, indicar que, en todo momento, podremos **saber el tipo de dato** de una variable usando la **función type()**:

Código:

```
varObjStr = 'Hola'
varObjInt = 0

print(type(varObjStr))
print(type(varObjInt))
```

Resultado en consola:

```
<class 'str'>
<class 'int'>
```

Las variables y constantes pueden ser de tipo:

- Numéricas:
  - **Int** → número = 0
  - **Float** → número = 0.0
  - **Complex** → número 3+1j
- Cadenas: **str** → texto = “hola mundo”
- Booleano: **bool** → booleano = True (o False)
- Listas: **list** → lista = [“dato1”, “dato2”…]
- Tuplas: **tuple** → tupla = (“dato1”, “dato2”, …)

La diferencia entre las listas y las tuplas:

En las tuplas, el **contenido no es mutable**, es decir, no se puede cambiar su valor. Además, debido a esto, las tuplas ocupan **menos memoria**.

Ambas (lista y tuplas) son **heterogéneas**, es decir, pueden tener, por ejemplo, números y cadenas mezclados en su interior.

- Rangos: **range** → rango = range(4)

- Diccionarios o json (almacenan clave y valor diccionario):

  dic → diccionario = {“nombre” : ”Miguel”, “edad” : ”25”}

- Set (se trata de una colección que no está ordenada ni indexada):

  set → set = ((“dato1”, “dato2”...))

Casting

Hay ocasiones en las que nos interesa que **una variable sea de un tipo concreto**, aunque estas se hayan **declarado de otro tipo**. Para ello, se usan las siguientes funciones:

• int(): construye un número entero a partir de un literal entero, un literal flotante (redondeando al número entero anterior) o un **literal de cadena** (**siempre que la cadena represente un número entero**).

• float(): construye un número flotante a partir de un literal entero, un literal flotante o un literal de cadena (siempre que la **cadena represente un flotante o un entero**). Ojo.

• str(): construye una cadena a partir de una amplia variedad de tipos de datos, incluidas **cadenas**, literales **enteros** y literales **flotantes**.

Castings

```
print("tipo str " + varStr)
print(type(int(varStr))) # casting a int desde String

print("tipo int " + str(varInt))
print(type(float(varInt))) # casting a float desde int

print("tipo float " + str(varFloat))
print(type(str(varFloat))) # casting a String desde float
```

salida consola

```
tipo str 2
<class 'int'>

tipo int 2
<class 'float'>

tipo float 2.0
<class 'str'>
```

Las funciones de print e input

Hablaremos de las funciones Input y Print. Para ello, haremos uso de un pequeño programa que hemos hecho en PyCharm, del cual iremos comentando y descomentando el código para realizar diferentes pruebas.

1. Función Input:

    En primer lugar, hablaremos del Input, que se utiliza para leer la información que el usuario introduce por teclado. Pausa el programa hasta que el usuario pulsa la tecla ENTER y siempre devuelve un string. Por ejemplo, ejecutamos la práctica y asignamos la entrada del usuario a una variable de test, luego la imprimimos. Si el usuario ingresa "123" y presiona ENTER, imprimirá "123". Sin embargo, podemos solicitar información al usuario incluyendo un parámetro en el Input, como en el caso de pedir el nombre.

    Ejemplo:

```
nombre = input("¿Cuál es tu nombre?")
print(nombre)
```

2. Trabajando con Números:

    El Input siempre devuelve un string, por lo que si tratamos con números, necesitamos convertirlos. Si preguntamos la edad y operamos directamente, se producirá un error. Utilizamos un casting para convertir la variable a entero antes de operar.

    Ejemplo:

```
edad = int(input("¿Cuál es su edad?"))
print(edad + 1)
```

3. Función Print:

La función Print se utiliza para imprimir por pantalla. Podemos utilizarla separando diferentes variables o textos por coma. Si algún elemento no es un string, **Python lo convierte automáticamente**.

Ejemplo:

```
nombre = "Miguel"
apellido = "Sánchez"
edad = 25
print(nombre, apellido, edad)
```

4. Parámetros de la Función Print:

La función Print tiene parámetros como end y sep. Por ejemplo, podemos especificar un separador y un finalizador.

```
print("Hola", "Adiós", sep="...", end="---")
```

Salida en consola:

```
Hola...Adiós---
```

5. F-Strings:

Desde la versión 3.6, se introdujo el fstring. Se coloca una F antes del texto y permite incluir expresiones directamente dentro del string.

Ejemplo:

```
fruta = "manzana"
precio = 2.0
cantidad = 8

print(f"El coste de la {fruta} es {precio}. \nPrecio total de 8 {fruta}s: {precio * cantidad}")
```

Salida en consola:

```
El coste de la manzana es 2.0.
Precio total de 8 manzanas: 16.0
```

Variables

Las variables son como contenedores para almacenar valores. En Python, no hay un comando para declarar variables, estas **se crean en el momento en el que se le asigna un valor** y **pueden cambiar de tipo** en el transcurso del código, por ejemplo:

Variables en Python

```
# Tipado dinámico
var = 1 # aquí la variable es de tipo numérico
var = "Hola mundo" # aquí la variable pasa a ser de tipo cadena
```

Para **declarar variables**, hay que seguir las siguientes reglas:

- Deben **empezar** por letras o por '_', **no por números**.
- Solo pueden tener **letras, números y el símbolo '_'**.
- Python es case **sensitive**, es decir, sensible a las mayúsculas y minúsculas (por ejemplo, test es distinto de Test).

Constantes

Una constante es una variable que siempre **mantiene el mismo valor**. En Python, **no existen constantes como tal**, lo que se hace es declarar una **variable en mayúsculas** para indicar que es una constante, por ejemplo:

```
CONSTANTE = 3
```

Errores

En Python, podemos encontrarnos con errores:

• De sintaxis: cuando escribimos mal el código y el intérprete no lo entiende, por ejemplo, no cerrar un paréntesis.

Error de sintaxis

```
1 + ("z"
      ^
SyntaxError: '(' was never closed

Process finished with exit code 1
```

• De ejecución: no tiene error de sintaxis, pero el intérprete no lo puede ejecutar, por ejemplo, sumar un número y un carácter.

Error de ejecución

```
1 + "z"
TypeError: unsupported operand type(s) for +: 'int' and 'str'

Process finished with exit code 1
```

Leer la información que nos ofrecen los errores nos ayudará a subsanarlos.

¡Ojo! input() siempre devuelve un string

Vamos a crear un programa que **sume dos operandos** que el usuario introduce por pantalla y luego **los multiplique**.

El programa preguntará por el Operando1: y el Operando 2:, y luego devolverá la suma de ambos y la multiplicación.

Tendremos que hacer uso de las funciones input() y print().

En primer lugar, **guardaremos en las variables "op1" y "op2" los operandos** que el usuario informe por pantalla:

```
op1 = input("Operando 1: ")
op2 = input("Operando 2: ")
```

Y, a continuación, imprimimos por pantalla el **resultado de la suma y la multiplicación**:

```
print(op1 + op2)
print(op1 * op2)
```

Ahora, si ejecutamos el programa e introducimos los valores 2 y 3, los resultados esperados son 5 para la suma y 6 para la multiplicación:

En su lugar, nos ha devuelto 23 para la suma y para la multiplicación, un **error indicando que no se pueden multiplicar tipos cadenas**.

Y esto es debido a que la **función input() siempre devuelve un string**, y claro, el operador '+' sí funciona en los strings como concatenación, pero el '*', no.

Para poder operar con los operandos, debemos hacer un **casting a tipo int**. Por ello, para recoger el valor adecuadamente, podríamos implementar la siguiente instrucción:

Casting

```
op1 = int(input("Operando 1: "))
op2 = int(input("Operando 2: "))
```

Y, finalmente, este sería el resultado:

Operando 1: 1

Operando 2: 2

3

2

Operadores

En Python, podemos encontrarnos con:

Aritméticos:

para operaciones matemáticas.

Operador Nombre Ejemplo

+ suma x + y

- resta x – y

* multiplicación x * y

/ división x / y

% resto x % y

** exponente x ** y

// división con redondeo a la baja x // y

De asignación:

para asignar valores a variables.

OPERADOR - EJEMPLO - ES LO MISMO QUE

= x = 1 x = 1

+= x += 1 x = x + 1

-= x -= 1 x = x - 1

*= x *= 1 x = x * 1

/= x /= 1 x = x / 1

%= x %= 1 x = x % 1

//= x //= 1 x = x // 1

**= x **= 1 x = x ** 1

De comparación:

para comparar valores.

OPERADOR – NOMBRE – EJEMPLO

== igual que x == y

!= distinto de x != y

> o >= mayor/mayor o igual que x > y o x >= y

< o <= menor/menor o igual que x < y o x <= y

Lógicos:

para combinar sentencias condicionales.

OPERADOR – DESCRIPCIÓN - EJEMPLO

and Devuelve verdadero si ambos x operandos son verdadero < 1 and x < 3

or Devuelve verdadero si al menos uno de los operandos es verdadero x < 1 or x < 3

not devuelve lo contrario de lo que tenga el operando not(x < 1)

De identidad:

para comparar objetos, pero, concretamente, si son el **mismo objeto con la misma posición de memoria**, no si tienen el mismo valor.

OPERADOR – DESCRIPCIÓN - EJEMPLO

is Devuelve verdadero si ambas variables son el mismo objeto x is y

is not Devuelve verdadero si ambas variables no son el mismo objeto x is not y

De pertenencia: si un objeto está presente en otro.

OPERADOR – DESCRIPCIÓN – EJEMPLO

in Devuelve verdadero si un objeto está presente en otro. x in y

not in Devuelve verdadero si un objeto no está presente en otro. x not in y

bit a bit: se usan con números binarios.

OPERADOR – NOMBRE – DESCRIPCIÓN

& AND Devuelve 1 si ambos bits son 1

| OR Devuelve 1 si uno de los bits es 1

^ XOR Devuelve 1 si solo 1 de los bits es 1

~ NOT Invierte el bit

<< Rellena con ceros por la izquierda

>> Rellena con ceros por la derecha

Ejemplo de código Python usando input() y casting

Vamos a crear un programa en Python que pregunte por pantalla la edad del usuario y a continuación le muestre un mensaje indicándole los **años que le restan para llegar a los 100**, que será una constante.

El programa preguntará "¿Cuál es su edad?" y al introducirla, si ésta es, por ejemplo, 25, entonces mostrará el mensaje "Le faltan 75 años para alcanzar los 100.".

Para dar solución al caso práctico inicial, tendremos que hacer uso de las funciones input() y print(). En primer lugar, crearemos la constante de la edad a alcanzar con valor 100:

Creación de la constante

```
EDAD_A_ALCANZAR = 100
```

Seguidamente, teniendo en cuenta que la función input() siempre devuelve un string, para poder operar con ella como un número, deberemos hacer un casting a tipo int. Por ello, para recoger el valor de la edad, podríamos implementar la siguiente instrucción:

Recoger valor de la edad

```
edad = int(input("¿Cuál es su edad?"))
```

A continuación, deberemos restarle a 100 dicha edad, guardando el resultado en la misma variable o en una nueva:

Resta

```
falta = EDAD_A_ALCANZAR - edad
```

Y, finalmente, mostrar el mensaje por pantalla:

Mensaje por pantalla

```
print("Le faltan", falta, "años para alcanzar los", EDAD_A_ALCANZAR)
```

Ahora, sí ejecutamos el programa:

Ejecución del programa

```
¿Cuál es su edad?46
Le faltan 54 años para alcanzar los 100

Process finished with exit code 0
```