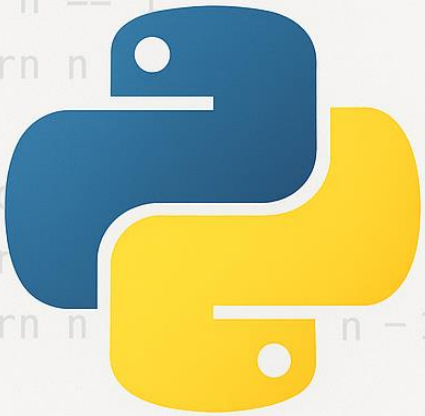
CURSO DE ESPECIALIZACIÓN

TEMA 1: INTRODUCCIÓN



python

1. Introducción

Python es uno de los lenguajes de programación más utilizados en la actualidad por su sencillez, versatilidad y amplia comunidad de usuarios. Su sintaxis clara y su orientación multiparadigma lo convierten en una herramienta idónea tanto para principiantes como para profesionales en ámbitos como la ciencia de datos, el desarrollo web, la inteligencia artificial, la automatización de procesos o la creación de aplicaciones de escritorio.
La primera versión de Python es de 1991, aunque su creador, Guido van Rossum, empezó a trabajar en Python en 1989. Su nombre es un homenaje a los Monty Python.

El objetivo de este tema es introducir al alumnado en los fundamentos del lenguaje, así como en los entornos de desarrollo y prácticas iniciales.

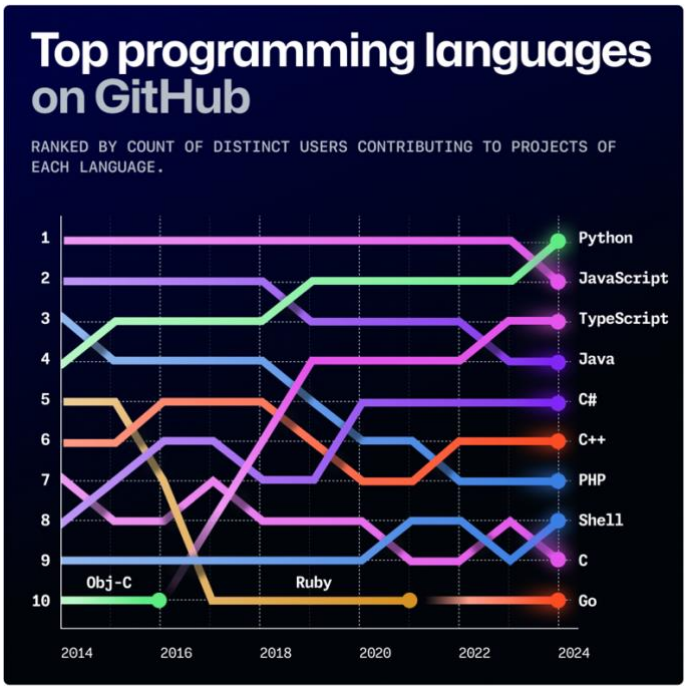
2. Fundamentos de la programación en Python

Python tiene una serie de características:
- Es un **lenguaje interpretado**: no hay que compilar el código, esto tiene ventajas y desventajas como veremos más adelante.
- Es de **alto nivel**: Usa estructuras fáciles de entender y una sintaxis que puedes leer casi como si fuera inglés.
- Puede usarse para desarrollo web, para crear interfaces, ciencia de datos, videojuegos, automatización, datos científicos y numéricos…
- Es un lenguaje de **propósito general**, puede utilizarse para crear cualquier programa.
- Es multiparadigma, soporta parcialmente la POO, programación imperativa, programación funcional… pudiéndose adaptar a tus intereses y gustos.
- Es **multiplataforma**, se puede usar en Linux, Windows, macOS…
- Tiene su propia biblioteca estándar, por lo que muchas veces no hará falta instalar ninguna biblioteca porque ya está incluida en su biblioteca estándar.

Python se fundamente en 4 pilares y sigue la filosofía que según su autor debe ser:
- Fácil, intuitivo y tan potente como sus principales competidores.
- Debe ser de código abierto.
- Debe ser tan comprensible como cualquier texto en inglés.
- Debe ser apto para actividades diarias, permitiendo crear prototipos en poco tiempo.

2.1 Por qué aprender Python

Python tiene una sintaxis legible, curva de aprendizaje rápida, se escribe menos código, amplio rango de aplicaciones, especialmente en IA, multiplataforma sin hacer cambios, gran comunidad. Podemos ver que, en la lista de los lenguajes más populares según GitHub, ha escalado a la primera posición.



¿Quién usa Python? Pues es usado por Netflix, Youtube en la parte servidor, la NASA…

Desventajas

Python también tiene desventajas:

- Su velocidad de ejecución no es la mejor, debido a que es interpretado.
- Python permite trabajar con múltiples hilos (multithreading), aunque debido a su diseño interno (el GIL, Global Interpreter Lock), estos no se ejecutan realmente en paralelo cuando realizan tareas intensivas de CPU. Sin embargo, el uso de hilos sí resulta útil para programas que realizan muchas operaciones de entrada y salida (como lectura de archivos o peticiones web).
- Tiene un consumo de memoria elevado en comparación con otros lenguajes.
- No es popular ni recomendable en aplicaciones móviles.
- Hay bibliotecas no muy maduras y a veces es difícil de elegir una librería para nuestras necesidades.

3. IDE y entorno de desarrollo

Un IDE (Integrated Development Environment) es un entorno que integra editor, depurador y gestor de proyectos, además que ofrece ayudas a la hora de escribir código, ya sea coloreando las palabras reservadas, indentando líneas o alertarnos de posibles errores.

Algunos IDEs y editores usados en Python son:

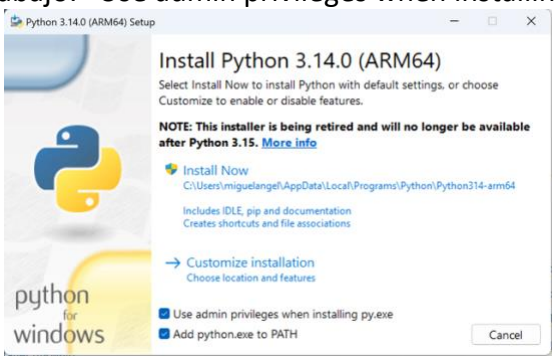
- VSCode: gratuito, multiplataforma, extensible.
- PyCharm: especializado en Python, muy completo.

3.1 Instalación Python y entorno de desarrollo

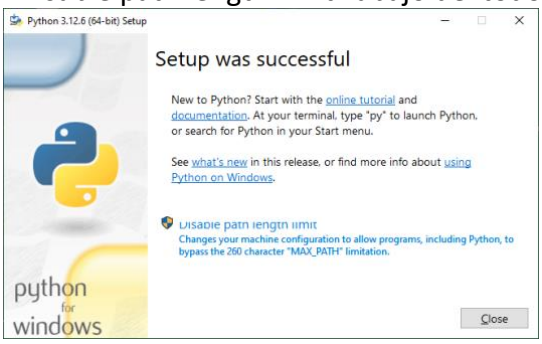
Primero hay que instalar Python en su ordenador, en macOS y Linux suele venir instalado, prueba a abrir el terminal buscando la aplicación terminal en Linux y MacOS. En Windows pulsa la tecla Win + R y escribe cmd (o en la barra de búsqueda abajo del

todo, escribe Ejecutar y cuando te aparezca la pantalla de ejecutar, escribe cmd), una vez en el terminal (pantalla negra), escribe: python --version y si no funciona escribe python3 --version y debe aparecerte la versión que tienes instalada.
Si no lo tienes instalado, ve a la página oficial de Python: https://www.python.org y elige debajo de Download la última versión de Python, te llevará una página (https://www.python.org/downloads/release/python-3137/) donde podrás descargar una versión de Python dependiendo de tu Sistema Operativo.

IMPORTANTE: En la instalación de Python, no te olvides marcar los dos cuadros de abajo: "Use admin privileges when installing py.exe" y "Add python.exe to PATH".



IMPORTANTE: Cuando te diga que la instalación ha sido exitosa, pulsa donde pone "Disable path length limit" abajo del todo y dile que sí a la instalación que te pida.

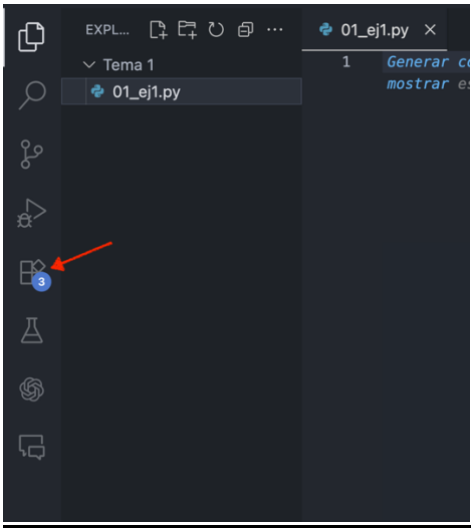


Ahora prueba a abrir el terminal de nuevo y escribir python --version y debería aparecerte la versión de Python.

IDE de Desarrollo

El editor que usaremos será VSCode que es multiplataforma y gratuito.
Os descargáis VSCode de su página oficial: https://code.visualstudio.com
Una vez instalado, vamos a añadirle algunas extensiones que nos facilitarán el trabajo con Python.

Para facilitarnos el trabajo de escribir código, instalaremos las siguientes extensiones en VSCode: Python, Python Debugger y Pylance (normalmente al instalar la extensión llamada Python se instalan las demás), estas extensiones se instalan buscándolas por su nombre pulsando sobre el menú "Extensiones".

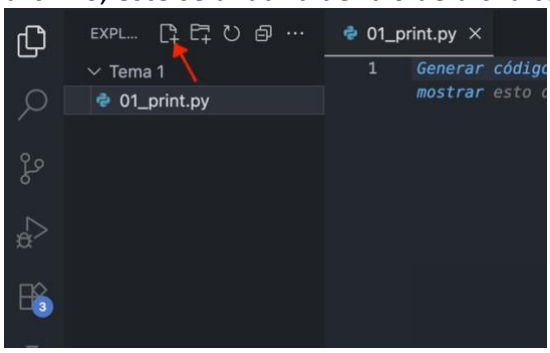


Si quieres tener VSCode en español, instala la extensión “Spanish Language Pack for Visual”.

3.2 Primer código en Python

Los archivos de Python utilizan la extensión .py y la convención de nombre es: nombre de archivos todo en minúsculas y si hay espacios poner guiones bajos (snake_case), por ejemplo: archivo_de_ejemplo.py.

VSCode tiene una peculiaridad, para empezar a trabajar necesitas una carpeta dentro de la cual irán todos los archivos que vayamos creando, así que crea una carpeta en el Escritorio y arrástrala a VSCode. Una vez arrastrada la carpeta, al darle a añadir un archivo, este se añadirá dentro de dicha carpeta.



Crea un archivo llamado 01_print.py y escribe dentro:
print("Hola mundo")
print es una función ya establecida dentro de Python. Se usan comillas dobles porque Hola mundo es una cadena de texto.

Ejecutar el archivo:

1. Primera forma: ir al terminal, navegar hasta la carpeta donde está el archivo Python, utilizando cd y luego ejecutar el archivo .py con el comando: python 01_print.py.
2. Segunda forma: en VSCode, arriba a la derecha hay un triángulo como si fuese el botón play de un vídeo o música. Se puede configurar una tecla rápida para ejecutar, se suele poner cmd + enter en macOS o ctrl+enter en Windows, para configurar dicha tecla:

- En macOS: Menú Code > ajustes > métodos abreviados de teclado.

- En Windows: Archivo > Preferencias > Métodos abreviados de teclado.
Una vez que te aparezca el listado de teclas rápidas, busca “Run Python File” y ahí pones la tecla rápida que hayas elegido.

Al ejecutar el archivo, se te abrirá la consola de VSCode por defecto en la parte inferior, si la cierras, puedes abrirla desde el menú Terminal > Nuevo terminal.

4. Sintaxis, operadores y tipos de datos

La sintaxis es el conjunto de reglas que determinan cómo debe escribirse el código para que el intérprete de Python lo entienda.
Características destacadas:

- Indentación obligatoria: en lugar de llaves {} o palabras clave begin/end, Python utiliza la indentación para delimitar bloques de código.
- Sensibilidad a mayúsculas y minúsculas: Variable y variable son identificadores distintos.
- Comentarios: con # para una sola línea, o con triples comillas """ para bloques.

Ejemplo:

```
# Esto es un comentario
if True:
    print("Hola")   # Sangría de 4 espacios
```

4.1 Comentarios

Los comentarios en Python son líneas de texto que el intérprete ignora al ejecutar el programa. Se utilizan para explicar el código, hacer anotaciones o desactivar temporalmente partes de un script.

Tipos de comentarios en Python

1. Comentario de una sola línea

Se escribe colocando el símbolo # delante del texto que queremos comentar.

```
# Esto es un comentario de una sola línea
x = 10  # También puede ir al final de una línea de código
```

2. Comentario multilínea o bloque de texto

Python no tiene una sintaxis específica para comentarios multilínea, pero se acostumbra a usar comillas triples (""" o ''') para escribir textos explicativos largos. Aunque se utilizan principalmente para docstrings (documentación de funciones o clases), también pueden servir como comentarios de varias líneas.

```
"""
Este es un comentario multilínea.
Se utiliza para explicar partes del código
o dejar notas extensas para otros programadores.
"""
```

Consejo: utiliza comentarios breves y claros. No describas lo obvio (por ejemplo, “sumo dos números” justo encima de suma = a + b), sino aquello que ayude a entender la intención o el porqué del código.

4.2 Variables

Las variables se utilizan para almacenar datos en memoria. Permiten asignarles un nombre para poder referirnos a ese valor en cualquier momento del programa.

Cómo se define una variable

En Python, una variable se define con el operador = (asignación):

```
variable = "hola"
print(variable)  # Muestra: hola
```

Aquí, el valor "hola" se guarda en memoria y podemos acceder a él utilizando el nombre variable.
En Python, las variables se crean en el momento en que se les asigna un valor por primera vez. No es necesario declararlas explícitamente con un tipo específico, ya que el tipo de dato se infiere del valor asignado. Esto hace que el código sea más limpio y fácil de escribir y entender.

Reglas para nombrar variables

1. Los nombres de variable solo pueden contener letras, números y guiones bajos (_).
Ejemplo válido: mi_variable1.
2. Deben comenzar por una letra o guion bajo, pero nunca por un número.
Incorrecto: 1variable
Correcto: variable1.
3. No se permiten espacios en los nombres de variable.
Usa _ en su lugar: nombre_usuario.
4. No uses palabras reservadas de Python, como print, if, for, class, etc.
Ejemplo: print = "hola" provocaría errores.
5. Los nombres deben ser claros y descriptivos, por ejemplo:
nombre es mejor que n.

Tipado dinámico

Python es un lenguaje de tipado dinámico, al igual que JavaScript.
Esto significa que el tipo de dato se determina en tiempo de ejecución, sin necesidad de declararlo previamente.

```
variable = "hola"
print(type(variable))  # <class 'str'>

variable = 5
print(type(variable))  # <class 'int'>
```

Como puedes ver, la misma variable puede cambiar de tipo en cualquier momento.

Tipado fuerte

Python es también de tipado fuerte, lo que significa que no convierte automáticamente entre tipos incompatibles.
Por ejemplo, no se puede sumar un número y una cadena directamente:

```
edad = 20
nombre = "Pepe"
# print(nombre + edad)  # ❌ Error: no se puede concatenar str e int
```

Para solucionarlo, hay que convertir (hacer un “casting”) el tipo de dato:

```
print(nombre + " tiene " + str(edad) + " años.")  # Correcto
```

Como puedes ver, para hacer un casting, solo hay que poner el tipo del dato y entre paréntesis el dato a convertir (ver tabla de la sección Tipos de datos más abajo).

Constantes

Python no tiene constantes reales, pero se pueden simular.
Una constante es un valor que no debería cambiar durante la ejecución del programa.
Por convención, se escriben en mayúsculas:

```
PI = 3.1416
MAX_USUARIOS = 50
```

Aunque técnicamente pueden modificarse, su escritura en mayúsculas indica al programador que no deben cambiarse.

4.3 Cadenas

Una cadena es una secuencia de caracteres, para delimitar una cadena, puedo usar comillas dobles (“) o comillas simples (‘) indistintamente, esto nos permite poder definir una cadena que tenga comillas dobles en su interior, por ejemplo:

```
print 'Bienvenidos a este "curso" de Python'.
```

Como podemos ver es fácil distinguir dónde empieza y donde acaba la cadena y dentro de ésta la palabra funciona lleva comillas dobles.

El operador “+” permite concatenar 2 cadenas. Ejemplo:

```
variable = "5"
print("Mi numero es: " + variable)
```

Hay que tener en cuenta que si intentamos concatenar una cadena y un número dará error, ya que solo se permiten concatenar cadenas.

print permite imprimir múltiples valores a la vez:

```
print ("Python", "es", "genial")
```

Al ejecutar este código, por defecto separa las cadenas por un espacio en blanco, para modificar este separador:

```
print ("Python", "es", "genial", sep="#")
```

Ahora el separador sería # en vez del espacio. sep le indica el nombre del parámetro de print que queremos cambiar, ya que su valor por defecto es el espacio.

Hay que tener en cuenta que entre print y print siempre hace un salto de línea por defecto, esto se puede modificar con:

```
print ("Python", "es", "genial", end=" ")
```

Ahora al finalizar un print, no salta de línea, añade un espacio en blanco al final.

Para ver los parámetros por defecto en VSCode, deja el ratón encima de la palabra “print” y aparecerá la ayuda en línea, donde podemos ver que print tiene los parámetros sep y end que ya hemos usado y además file y flush.

Cadenas formateadas (f-strings)

Una forma más moderna y legible de insertar variables en cadenas es usando f-strings.

Se añade una f delante de las comillas, y se usan {} para insertar variables o expresiones:

```
mi_nombre = "Pepe"
edad = 20
print(f"Hola {mi_nombre}, tengo {edad} años.")
```

Dentro de las llaves {} se puede incluir cualquier expresión válida:

```
print(f"El año que viene tendré {edad + 1} años.")
```

Si queremos que aparezda el valor númerico de una variable con 1 decimal, se hace de la siguiente forma:

```
precio = 19.945656
print(f"El precio del producto es: {precio:.1f}€")
```

Primeras funciones de las cadenas:

Podemos aplicar distintos métodos a las cadenas para cambiar su formato o contenido.

1. Poner la primera letra de cada palabra en mayúsculas

El método .title() convierte la primera letra de cada palabra en mayúscula:

```
nombre = "manuel perez"
print(nombre.title())  # Muestra: Manuel Perez
```

2. Convertir todo a mayúsculas o minúsculas

- .upper() convierte todos los caracteres en mayúsculas.
- .lower() convierte todos los caracteres en minúsculas.

```
nombre = "manuel perez"
print(nombre.upper())  # Muestra: MANUEL PEREZ
print(nombre.lower())  # Muestra: manuel perez
```

Caracteres especiales

A veces necesitamos incluir saltos de línea, tabulaciones u otros caracteres no visibles dentro de una cadena.

Para ello usamos caracteres de escape, que comienzan con una barra invertida \.

\n = salto de línea

\t = tabulación

Ejemplos:

```
print("Python")          # Muestra: Python
print("\tPython")        # Muestra:     Python (con tabulación)
print("Lenguajes:\nPython\nJavaScript")
# Muestra:
# Lenguajes:
# Python
# JavaScript
```

Hay que tener en cuenta que algo tan simple como este ejemplo puede dar error:

```
print("c:\documentos\nombres")
```

Ya que Python interpreta el \n de \nombres como un salto de línea, para evitar este error hay 2 opciones:
1. “Escapar” la barra para que no la interprete, esto se hace poniendo una \ delante, quedando así:

```
print("c:\\documentos\\nombres")
```

2. Poniendo una “r” antes de abrir comillas, para indicar que no interprete caracteres especiales:

```
print(r"c:\documentos\nombres")
```

Eliminar espacios en blanco

En ocasiones, una cadena puede tener espacios extra al principio o al final, lo que afecta a comparaciones o resultados.
Por ejemplo, "hola" y "hola " no son iguales.
Python ofrece tres métodos muy útiles:
.rstrip() = Elimina los espacios en blanco del final.
.lstrip() = Elimina los espacios del inicio.
.strip() = Elimina los espacios en ambos extremos.

```
lenguaje = "Python   "
lenguaje = lenguaje.rstrip()
print(lenguaje)  # Muestra: Python
```

También funciona con otros caracteres, no solo espacios:

```
texto = "----Hola----"
print(texto.strip("-"))  # Muestra: Hola
```

Eliminar prefijos o sufijos

En versiones modernas de Python (3.9 o superior) existen dos métodos muy prácticos:
.removeprefix(prefijo): Elimina un prefijo si existe.
.removesuffix(sufijo): Elimina un sufijo si existe.
Ejemplo:

```
url = "http://www.suarezdefigueroa.es"
print(url.removeprefix("http://"))
# Muestra: www.suarezdefigueroa.es
```

Y también se puede usar para eliminar sufijos, por ejemplo, extensiones de archivo:

```
archivo = "informe_final.pdf"
print(archivo.removesuffix(".pdf"))  # Muestra: informe_final
```

Contar la longitud de una cadena: len()

La función integrada len() (de length, “longitud”) devuelve el número de caracteres que contiene una cadena, incluidos los espacios.
Ejemplo:

```
nombre = "Python"
print(len(nombre))  # Muestra: 6
```

```
frase = "Hola mundo"
print(len(frase))   # Muestra: 10 (incluye el espacio)
```

Esto resulta muy útil, por ejemplo, para saber si una contraseña tiene la longitud mínima requerida o para contar letras en un texto.

Sustituir texto con replace

El método replace() permite reemplazar una parte de una cadena por otra.
Su sintaxis es:
cadena.replace(texto_a_buscar, texto_nuevo[, contador])

- texto_a_buscar → el texto que quieres reemplazar.
- texto_nuevo → el texto por el que se reemplazará.
- contador (opcional) → número máximo de reemplazos (por defecto, cambia todos).

```
frase = "Me gusta programar en Java"
nueva = frase.replace("Java", "Python")
print(nueva)  # Me gusta programar en Python
```

Por defecto replace cambia **todas las coincidencias**.
Puedes indicar un número máximo de sustituciones:

```
mensaje = "Python es divertido. Python es potente. Python es fácil."
nuevo = mensaje.replace("Python", "C++", 2)
print(nuevo)
# C++ es divertido. C++ es potente. Python es fácil.
```

Solo cambia las dos primeras apariciones.

4.4 Operadores

Python utiliza operadores para realizar distintas operaciones:

Operadores aritméticos

Operador	Significado	Ejemplo	Resultado
+	Suma o concatenación	3 + 2, "a" + "b"	5, "ab"
-	Resta	10 – 4	6
*	Multiplicación	2 * 3	6
/	División real	7 / 2	3.5
//	División entera (piso)	7 // 2	3
%	Resto o módulo	7 % 2	1
**	Potencia	2 ** 3	8

Orden de prioridad

1. Paréntesis ()
2. Potencias **
3. Multiplicación, división, resto, división entera

4. Suma y resta

Ejemplo:

```
(7 + 3) * 2 ** 3 + 1
# = 10 * 8 + 1 = 81
```

Operadores de comparación

Operador	Significado	Ejemplo	Resultado
==	Igualdad	5 == 5	True
!=	Distinto	5 != 3	True
<	Menor que	3 < 5	True
>	Mayor que	7 > 10	False
<=	Menor o igual que	3 <= 3	True
>=	Mayor o igual que	4 >= 5	False

Operadores lógicos

Operador	Significado	Ejemplo	Resultado
and	Y lógico	x > 0 and x < 10	True si ambas son verdaderas
or	O lógico	x < 0 or x > 10	True si una de las dos es verdadera
not	Negación	not (x > 0)	True si la condición es falsa

Asignación

Utiliza varios operadores: =, +=, -=, *=, /=, etc.

Ejemplo: x += 2 equivale a x = x + 2

- / → división real (siempre float). Ejemplo: 5 / 2 == 2.5
- // → división entera (piso). Ejemplo: 5 // 2 == 2
- % → resto. Ejemplo: 5 % 2 == 1

4.5 Tipos de datos

En Python, cada valor tiene un tipo que determina qué operaciones se pueden realizar con él. Los tipos de datos se dividen en básicos (simples) y complejos (estructurados).

Tipos básicos

Tipo	Descripción	Ejemplo
int	Números enteros	x = 10
float	Números decimales	pi = 3.1416
str	Cadenas de texto (strings)	nombre = "Python"
bool	Valores lógicos (verdadero o falso)	activo = True
NoneType	Valor nulo o sin valor definido	dato = None

Puedes comprobar el tipo de cualquier variable usando la función type():

```
x = 10
print(type(x))  # <class 'int'>
```

Tipos complejos o estructurados de datos (los veremos en detalle en los siguientes temas)

Tipo	Descripción	Ejemplo
list	Lista ordenada y modificable de elementos	frutas = ["manzana", "pera", "uva"]
tuple	Tupla ordenada e inmutable	coordenadas = (10, 20)
dict	Diccionario con pares clave:valor	persona = {"nombre": "Ana", "edad": 25}
set	Conjunto de elementos únicos (sin orden)	numeros = {1, 2, 3}

Ejemplos de uso:

```
# Lista
colores = ["rojo", "verde", "azul"]
print(colores[0])  # Accede al primer elemento: 'rojo'

# Tupla
punto = (3, 4)
print(punto[1])  # Accede al segundo elemento: 4

# Diccionario
alumno = {"nombre": "Juan", "nota": 8.5}
print(alumno["nombre"])  # Muestra 'Juan'

# Conjunto
numeros = {1, 2, 2, 3}
print(numeros)  # Muestra {1, 2, 3} (los duplicados se eliminan)
```

Tipado opcional (Type Hints)

Python permite anotar el tipo de las variables, aunque no lo impone.
Estas anotaciones sirven para documentar el código y ayudar a los editores a detectar errores.

```
mi_variable: bool = True
```

Aun así, Python lo permite cambiar:

```
mi_variable = 42  # No da error en tiempo de ejecución
```

Para que el editor te avise, activa la comprobación de tipos en VSCode:

1. Abre Configuración (Ctrl + , o Cmd + , en macOS) o dale a la rueda dentada de abajo a la izquierda llamada “Administrar” y elegir “Configuración”.
2. Busca “type checking mode”.
3. Selecciona “strict”.

De esta forma, el editor mostrará advertencias si usas un tipo incorrecto, aunque el programa siga funcionando.

Ejercicio 1:

Crea las siguientes variables y muestra en pantalla su tipo con la función type():

1. Tu nombre
2. Tu edad
3. Una lista con tus tres colores favoritos
4. Un valor booleano que indique si te gusta programar

4.6 Función input()

La función input() sirve para pedir información al usuario desde la consola.

```
print("Dime tu nombre:")
nombre = input()
print(f"Hola {nombre}")
```

También se puede escribir de forma más compacta:

```
nombre = input("Dime tu nombre:\n")
print(f"Hola {nombre}")
```

El carácter \n hace que el texto se escriba en una nueva línea.

Conversión de tipos con input()

Ten en cuenta que todo lo que introduce el usuario con input() se guarda como texto (str), aunque parezca un número.

Por tanto, si queremos realizar operaciones numéricas, hay que convertirlo (casting):

```
edad = input("Dime tu edad:\n")
print(f"Tu edad en 5 años será {int(edad) + 5}")
#O también:
edad = int(input("Dime tu edad:\n"))
print(f"Tu edad en 5 años será {edad + 5}")
```

Introducir varios valores a la vez

Podemos capturar varios datos a la vez separándolos por espacios y usando split():

```
nombre, edad = input("Dime tu nombre y edad:\n").split()
print(f"Hola {nombre}, tienes {edad} años.")
```

split() separa los valores por espacios.

Por ejemplo, si el usuario escribe Juan 20, se asignará:

- nombre = "Juan"
- edad = "20"

Si el usuario escribe algo como “Juan Antonio 20”, el programa dará error, porque split() separará en tres valores, y solo hay dos variables a las que asignar.

Ejercicio 2.

Escribe un programa que pida al usuario dos números, los sume y muestre el resultado junto con su tipo de dato.

Ejercicio 3. Haz un programa que pida el nombre y la edad del usuario, y muestre cuántos años tendrá dentro de 10 años. Usa input() y f-strings.

Ejercicio 4.

Crea un programa que:

1. Pida al usuario su nombre y apellido.
2. Muestre su nombre completo con la primera letra en mayúscula.
3. Indique cuántos caracteres tiene su nombre completo.

5. Tipos de datos estructurados

En Python, además de los tipos de datos básicos (int, float, str, bool, NoneType), existen estructuras que permiten almacenar múltiples valores en una sola variable.

Estas estructuras se llaman tipos de datos estructurados o colecciones y son muy útiles para manejar conjuntos de información, como listas de nombres, registros de alumnos o catálogos de productos.
Los principales tipos estructurados son:

- Listas (list).
- Tuplas (tuple).
- Diccionarios (dict).
- Conjuntos (set).

5.1 Listas

Una lista es una estructura que permite almacenar varios valores en un solo objeto y acceder a ellos por su posición (índice).
Las posiciones comienzan en 0 (no en 1), como en la mayoría de lenguajes de programación.

Equivalente a los arrays o vectores de otros lenguajes, las listas pueden contener cualquier tipo de dato: números, cadenas, valores booleanos o incluso otras listas.

Creación de una lista

```
lista = [3, 4, 9, 1, 4, 4, 3]
```

lista	**3**	**4**	**9**	**1**	**4**	**4**	**3**
Índice normal	0	1	2	3	4	5	6
Índice inverso	-7	-6	-5	-4	-3	-2	-1

Ejemplos de uso:

```
print(lista[2])   # Accede al tercer elemento: 9
lista[-1] = 5     # Cambia el último elemento por 5
print(lista)      # [3, 4, 9, 1, 4, 4, 5]
```

Operaciones comunes con listas

```
frutas = ["manzana", "pera", "uva"]
frutas.append("naranja")   # Agrega un elemento al final
frutas.insert(1, "plátano")  # Inserta en la posición 1
frutas.remove("pera")      # Elimina el elemento "pera"
print(frutas)
print(len(frutas))         # Número de elementos de la lista
```

Las listas son mutables, es decir, sus elementos pueden cambiar después de ser creadas.

5.2 Tuplas

Una tupla es una colección inmutable, es decir, no se puede modificar después de ser creada.
Se define entre paréntesis y también se accede por índices:

```
tupla = (3, 4, 9, 1, 4, 4, 3)
print(tupla[2])   # Muestra 9
```

Si intentamos modificar un elemento:

```
tupla[0] = 10    # ✕ Error: las tuplas son inmutables
```

Las tuplas son útiles cuando se quiere proteger la información frente a cambios accidentales.

También pueden desempaquetarse fácilmente:

```
coordenadas = (10, 20)
x, y = coordenadas
print(x, y)   # 10 20
```

5.3 Diccionarios

Un diccionario es una estructura que almacena pares clave:valor.
Cada elemento tiene una clave única que permite acceder rápidamente a su valor correspondiente.

```
alumno = {"Nombre": "María", "Edad": 15, "Curso": "4A"}
print(alumno["Nombre"])  # Muestra: María
```

Modificar y añadir datos

```
alumno["Edad"] = 16          # Modifica un valor existente
alumno["Asignatura"] = "Mates" # Añade un nuevo par clave:valor
print(alumno)
```

En resumen:

- Usa listas cuando necesites colecciones que cambian.
- Usa tuplas para datos que no deben modificarse.
- Usa diccionarios cuando necesites asociar un valor a un nombre o clave.

6. Python y la PEP 8

PEP 8 (Python Enhancement Proposal 8, o Propuesta de Mejora de Python nº 8) es una guía de estilo que define un conjunto de recomendaciones para escribir código Python limpio y legible.

La razón principal de seguir estas normas es que el código se lee muchas más veces de las que se escribe, por lo que mantener un estilo coherente facilita su comprensión y mantenimiento, tanto por uno mismo como por otros desarrolladores.

Cumplir la PEP 8 no es obligatorio, pero es una práctica ampliamente adoptada por la comunidad. La mayoría de los proyectos, librerías y frameworks populares de Python siguen estas convenciones.

Principales recomendaciones de la PEP 8

- Usar sangrías de 4 espacios, nunca tabuladores (Tab).
- Limitar la longitud de las líneas a 79 caracteres para mejorar la legibilidad en pantallas y editores.
- Separar funciones, clases y bloques grandes de código con líneas en blanco.
- Escribir comentarios claros y concisos, preferiblemente en una sola línea cuando sea posible.
- Utilizar docstrings (cadenas entre comillas triples) para documentar módulos, clases y funciones.
- Usar espacios alrededor de los operadores y después de las comas, pero no dentro de los paréntesis.

```
variable = (1 + 2) * (3 / 4)
```

- Nombrar las clases y funciones de forma coherente:
  - Clases → CamelCase → MiClaseEjemplo
  - Funciones, métodos y variables → snake_case → mi_funcion, mi_variable
- Evitar el uso de caracteres no ASCII en los identificadores, para garantizar la compatibilidad en todos los entornos y sistemas.

En VSCode y otros editores modernos puedes instalar extensiones como Pylint, que analizan tu código y aplican automáticamente las reglas de la PEP 8. Pylint señala errores o incumplimientos del estilo.