

Scripting básico

Pragmatic Python



Presentación

www.medialab.com

-

Github



jmuniztraviesas@gmail.com

Índice

- ¿Qué es Python?
- Variables y Constantes
- Comentarios
- Tipos de datos
 - Tipos de datos numéricos
 - Otros tipos de datos
- Listas y Diccionarios
- Control de flujo de ejecución
- Bucles
- Input y Output
- Funciones
- Importar y descargar módulos
- Ejemplos de scripts
- Notas finales

¿Qué es python?

Lenguaje de “*scripting*”



- De **alto nivel** (fácil de escribir)
 - Relativamente ineficiente
- (Para cosas “**rápidas y fáciles**”)

Una de las grandes ventajas de python es su **extensa comunidad**.

Python es un lenguaje *interpretado*, por eso hemos de descargar la aplicación python, se trata del intérprete que analizará nuestro código



Descargas

Intérprete de python: Fundamental

Aplicación que permite ejecutar código y scripts de python.

IMPORTANTE: Agregar a PATH



Welcome to Python.org

The official home of the Python Programming Language

Python.org /

Terminal de windows: Opcional

Aplicación que facilita el uso y gestión para utilizar la línea de comandos (terminal)



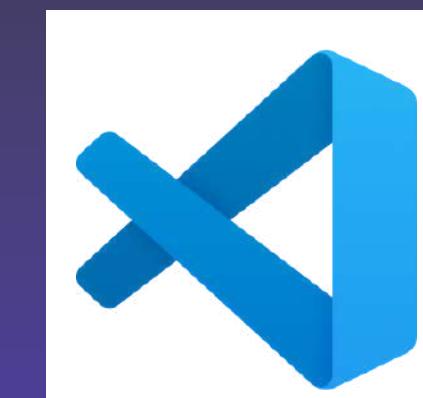
Windows Terminal: descarga e instalación gratuitas en Windows

Terminal Windows es una aplicación de terminal moderna, rápida, eficaz, eficiente y productiva par...

Microsoft Store

VS Code: Opcional

Editor de texto usado ampliamente para programar en varios lenguajes, muy ligero.



Visual Studio Code

Variables y Constantes

- El operador “=” se usa para **asignar valores** a las variables
- Se pueden asignar varias variables en una sola línea separadas mediante comas

```
manzanas = 3
# Asignación múltiple
x, y, z = 0, 1, 2
# Constantes (no existen, por convenio letras en mayúscula)
PI = 3.14159265
BUFFER_SIZE = 256
```

- **No existen las constantes** a nivel de lenguaje, pero por convenio MAYÚSCULA -> CONSTANTE
(Excepción: las tuplas)

- En python las variables suelen escribirse en **snake_case** (minúsculas con barra baja entre palabras)

Nota sobre notación de variables:

- **snake_case**: variables en python
- **UPPERCASE**: constantes en python
- **PascalCase**: Objetos (Clases)
- **notienennombre**: funciones en python

Comentarios

Para escribir **buen código** no es sólo necesario que funcione, si no que también sea **legible**.

Esto se consigue de varias maneras:

- Expresando claramente la intencionalidad a la hora de escribir código
 - Nombres de variables claros
 - Estilo de programación consistente
- De forma más directa: con **comentarios**

Los **comentarios** son porciones de texto en un fichero que el intérprete ignora, es decir:

No son código.

```
'''Comentario multilínea. Se suele poner
al principio del script para describir lo que hace'''

# El intérprete ignora todo texto detrás de un '#'
var1 = 37 # Mala práctica, el nombre de variable no dice nada
'''Los comentarios no suplen la información que
| debería estar en otras partes'''
η = 0.93 # En python 3 los nombres de variables pueden ser unicode
álgebra_lineal = True # Esto no significa que sea buena práctica usar acentos
```

- Hay dos tipos de comentarios:
- Comentario de una línea (#)
 - Comentario multilínea (""" ")

Tipos de datos

- En python **no se pueden** “pre-asignar” tipos de datos, el intérprete deduce el tipo de dato durante la ejecución (*dynamic typing*)
- Para dejar claras las intenciones puedes escribir tipos de datos, pero el **intérprete los ignorará**
- Ciertos módulos (e.g. *numpy*) **sí** tienen tipos de datos que el compilador no ignora, en este caso es **crucial** usarlos

```
plazas_curso : int = 15      # El intérprete ignorará el int
plazas_curso = 15             # Equivalente a lo anterior a ojos del intérprete
litros_cafe : float = 2.8    # El intérprete ignorará el float
cafe_rico : bool = False     # Hay más tipos de datos que números
# Ejemplo numpy
import numpy as np
byte = np.dtype(np.uint8)     # Un byte es un entero de 8 bits
```

Tipos de datos numéricos

- **int**: de **integer**, número entero (\mathbb{Z}).
Los números sin parte decimal, ya sean positivos o negativos:
1, 2, 3, 15, -2, -278536 ...
- **float**: de floating point, es un tipo de representación de los números reales (\mathbb{R}).
-2.5, 4.26, 0.0, 1.0, 327.2, ...
- **complex**: números complejos (\mathbb{C}).
Formados de parte real e imaginaria,
No trabajaremos con ellos.

```
# int (Número entero, integer)
entero : int = 5_000_000    # 5 millones, se pueden poner _ como separadores
# float (Número real, floating point)
flotante : float = 5.0
# complex
complejo = complex(2.5, 2.5)  # No usaremos complejos
```

Tipos de datos numéricos

Todos los tipos de dato numéricos tienen las operaciones básicas:

- + : Suma
- - : Resta
- * : Producto
- / : División

Además python cuenta con la operación potencia:

- ** : Potencia

```
# Operaciones con números
suma = 3 + 4
print("suma:", suma)
resta = 7.5 - 5
print("resta: %f" % resta)    # Hay distintas formas de hacer print, lo miraremos luego
producto = 4 * 4.5
print(f"producto: {producto}") # f-string es la forma preferida actualmente de ..
division = 640 / 20           #..dar formato a cadenas de texto
print("division:", division)
potencia = 3.1**4.5
print("potencia:", potencia)
```

Tipos de datos numéricos

Ejercicios:

Calcular las siguientes expresiones, asignando su valor a variables con nombres del tipo: “resultado_1, resultado_2, ...”:

- $2475 + 821$
- 4.65×26.3
- $963429 \div 22.5$
- 436^2
- $a^3; a = 45$
- $2\pi r^2; r = 2.5$
- $\frac{\sqrt[3.2]{427}}{4\pi^2}$

Recap I

Recapitulamos:

Expresividad y buenas prácticas:

- Variables con nombres **explicativos**
- Ayudar a la legibilidad con comentarios
- Expresar la **intencionalidad** del código es **clave**

Tipos de datos numéricos:

- Son pocos, enteros (**int**) y reales (**float**).
- Operaciones intuitivas de toda la vida: **suma** (+), **resta** (-), **multiplicación** (*), **división** (/), además es fácil hacer **potencias** (**)
- El ordenador calcula por ti, problemas que parecen difíciles son fáciles con código

Otros tipos de datos

Además de los tipos numéricos, existen otro tipo de datos en python:

bool: Variable *booleana*, es un tipo de variable usada en el álgebra de Boole que sólo puede tomar 2 valores posibles.

En nuestro caso verdadero (**True**) o falso (**False**).

str: de *string*, en español **cadena de texto**. Es texto *per se*, se puede construir (y codificar) de distintas maneras.

```
# bool (variable booleana, verdadero o falso)
true_flag = True
false_flag = False
# str (Cadenas de texto, string)
first_word = "hello"
second_word = "world"
message = f"{first_word} {second_word}" #Construcción de cadenas con format strings
```

Listas y Diccionarios

En python existen formas de agregar datos para formar *estructuras de datos*, dos tipos de estructuras en python son:

- **Listas []:** Conjunto Ordenado de datos, podemos introducir números, texto, ... Se acceden a través de un índice (número, int) que empieza en

CERO

Índices: off-by-one errors
Tuplas (): “*Listas constantes*”

- **Diccionarios {}:** Similares a las listas, pero los “índices” son arbitrarios. Se puede pensar en ellos como un par “llave - valor” (**key-value pairs**)

```
# ----- Listas (lists) -----
lista_compra = ["Huevos", "Leche", "Pan"]
print(lista_compra[0]) # Se acceden a través de índice
lista_2 = [2, 5.7, "Pedro"] # Las listas pueden tener varios tipos de datos distintos
print(lista_2) # Se pueden imprimir de forma íntegra
# ----- Diccionarios (dictionaries) -----
books = {"ISBN1": "Canción de Hielo y Fuego", "ISBN2": "Danza de dragones"}
print(books["ISBN1"]) # Se acceden a través de llaves (key)
desorden = { 37 : "treintaysiete", "cuatro" : 4} # También pueden combinar tipos de dato
print(desorden) # También se puede imprimir de forma íntegra
print(desorden[37])
# Trabajaremos más con ello después de dar bucles
```

- Similar: {
- Lista: Array (con tipos de dato arbitrario)
 - Diccionario: Mapa (lineal), JSON

Listas y Diccionarios

Ejercicios:

- Crea una lista con algunos nombres de tus compañeros (entre 3 y 5). Imprime en pantalla todos los elementos de la lista.
- Crea una lista con TODOS los nombres de tus compañeros. Imprime en pantalla todos los elementos de la lista.
- Crea un diccionario con los **colores del arco-iris** y su valor **RGB**. (buscarlos en google)
(Nota: los colores del arco-iris son: **rojo**, **naranja**, **amarillo**, **verde**, **azul**, **añil** y **violeta**)



Control de flujo de ejecución

Los lenguajes de programación integran varios métodos para controlar cómo se ejecuta un programa en función de varias **condiciones**. Se suele referir a esto como control de flujo (**flow control**)

Para hacer buen uso de las instrucciones de control de flujo es bueno saber un par de **operaciones booleanas**:

- **AND (&)**: Devuelve **True** si **todos** los elementos son **True**
- **OR (|)**: Devuelve **True** si **algún** elemento es **True**

```
# ----- Operaciones booleanas (verdadero / falso) -----
es_cierto = True
es_falso = False
# AND (&)
print("AND: ", es_cierto & es_falso)    # operación AND; "Tienen que darse
# OR (|)
print("OR: ", es_cierto | es_falso)     # operación OR; "Con que haya una ya
```

Recap II

Recapitulamos:

Tipos de datos no numéricos:

- Variables **booleanas** (**verdadero** | **falso**)
- Para introducir texto: **strings** (Cadenas de texto). Más adelante profundizaremos.

Operaciones con variables booleanas:

- Operación **AND** (&) : Si se dan **todas** las condiciones
- Operación **OR** (|) : Si se da **alguna** condición

Agregación de datos en una variable:

- **Listas** []: Agregar elementos de forma **ordenada**, **mutable**, acceso por **índices**.
- **Tuplas** (): Igual que las listas pero **inmutable**.
- **Diccionarios** { }: Asociar elementos de dos en dos; {**key** : **value**, ...}

Control de flujo de ejecución

- **Indentación:** la indentación es el nivel de separación del texto del borde izquierdo de la página.
- En python se aplica con **tabulación**.
- La **indentación** es importante en python porque nos diferencia entre **bloques de programación**
- Los bloques son unidades de código que se ejecutan como una sola unidad



```
# ----- Indentación -----
dia_semana = "Viernes" # Esto es un nivel de indentación
dia_mes = 14             # -
if (dia_semana == "Viernes"):
    print("Es viernes.") # Esto es otro nivel de indentación
    if (dia_mes == 14): # -
        print("Feliz San Valentín ❤️")# Esto es otro nivel de indentación
    else:                 # - Misma indentación que el segundo if
        print("Feliz Viernes")      # -
```

- Veremos que esto es importante para las instrucciones de flujo de ejecución, bucles y funciones.

Control de flujo de ejecución

- **if:** Si se da la condición establecida, ejecuta el código dentro del bloque que le acompaña.
- **else:** Acompaña a un if. Si no dió la condición del if, se ejecuta el código del else.
- **elif:** Es un abreviación de else if. Es equivalente a escribir un else con un if dentro.

```
# ----- Control de Flujo -----
es_viernes = True
es_14_de_febrero = True
if (es_viernes & es_14_de_febrero): # Si se dan las dos cosas
    print("Feliz San Valentín ❤")
elif(es_viernes):
    print("Feliz Viernes")
else:
    print("Feliz día aunque no sea viernes")
```

Control de flujo de ejecución

Ejercicios:

- Hacer un script que felicite San Valentín si es 14 de febrero.
 - Usando variables booleanas estáticas
 - Usando funciones para obtener el tiempo del sistema operativo

Ayuda:

```
# Usando el tiempo del sistema
from datetime import datetime # Importamos el objeto datetime del módulo datetime

ahora = datetime.now()      # Hora y fecha actual
```

- Hacer un script que compruebe si hay suficientes ingredientes para hacer una tarta (**3 huevos y 0.8 litros de leche**) en la nevera. Los alimentos dentro de la nevera están especificados en un diccionario:
- `lista_compra = {"Leche": 1.5, "Carne": 250.0, "Lechugas": 3, "Yogures": 6, "Huevos": 2}`

Bucles

- **while:** (“mientras”) El bucle while ejecuta un bloque de código continuamente mientras se cumpla una condición.
- **for:** (“por cada”) Ejecuta un bloque de código un número determinado de veces
- **break:** sirve para escapar prematuramente de un bucle

for “**range loops**” se hacen con la función **range** (rango), crea un rango de valores

```
# while
peligro = False
while(condition):
    if(peligro):
        print("este bucle es infinito, cuidado con el bucle while")
    else:
        break      # break sirve para salir de un bucle while
# for
N = 5
print("Bucle 1")
for i in range(N): # Fin
    print(i)
print("Bucle 2")
for i in range(10,20,2): # Principio, Fin, Salto
    print(i)
```

Bucles

```
# Los diccionarios pueden contener listas y vice-versa
# Diccionario que contiene tuplas con RGB como valores
colores_RGB = {"Rojo":(255,0,0), "Verde":(0,255,0), "Azul":(0,0,255)}
print(f"Rojo: {colores_RGB['Rojo']}") # OJO con ' '
# Imprimir en pantalla elementos de una lista
elementos = ["Hidrógeno", "Silicio", "Carbono", "Cloro", "Sodio"]
print("Todos los elementos de una lista: ")
for elemento in elementos:
    print(elemento)
# Imprimir ciertos elementos de una lista (índices)
print("Algunos elementos de una lista: ")
for x in range(2, 4, 1):
    print(elementos[x])
```

```
for index, item in enumerate(items):
    print(index,item)
```

Control de flujo de ejecución

Ejercicios:

- Contar los elementos de una lista.
 - ¿Por qué es mala idea hacer esto con un bucle? (función `len()`)
- Rehacer el script que compruebe si hay suficientes ingredientes para hacer una tarta mediante el uso de bucles (recorrer diccionario con bucle `for`)
- Imprimir cada elemento par de una lista (asumir 0 es par)
Nota: el operador módulo (%) te devuelve el resto de la división entre dos números.
- Construye un bucle que imprima los números del 1 al 5 sin usar el bucle `for`
- Imprimir los elementos de una lista separados por punto y coma (;)

Recap III

Recapitulamos:

Indentación:

- Python la usa para separar **bloques** de código.
- Pensad en los bloques como unidades lógicas independientes. (más adelante **funciones**)

Control de flujo:

- **if**(expresión): Si la expresión es **True**, se ejecuta el código en el bloque que lo acompaña.
- **else**: Acompaña a un **if**, y ejecuta el código **si el if no se cumplió**.
- **elif**(expresión): abreviación de **else if**.

Bucles:

- **while (expresión)**: Mientras se mantenga verdadera (**True**) la expresión, se ejecuta el bloque de código.
- **for**: Ejecuta el bloque de código **un número determinado de veces**.

Input y Output

Para formar **cadenas de texto**:

- **format strings:** ponemos f antes de una cadena de texto para indicar que es una format string. Entonces podemos introducirle variables con corchetes.

```
f"Texto {variable} texto..."
```

```
# Formato de cadenas de texto
PI = 3.14159265
diametro_mm = 5
texto_diametro = f"El valor del diámetro es {diametro_mm}mm." # También podemos introducir expresiones dentro de los corchetes
texto_resultado = f"El valor del área del círculo es: {(PI/4)*diametro_mm**2 : .2f}"
print(texto_resultado)
```

- Se puede dar formato a las variables introducidas con “.”

```
f"Texto {variable: .2f} texto..."
```

- Se pueden introducir expresiones dentro de los corchetes

Input y Output

- **print("...")**: Esta función *imprime* una cadena de texto en pantalla. Por defecto imprime un salto de línea ('\n') al final.

Si queremos cambiar esto, hemos de modificar el parámetro end:

```
print("...", end="...")
```

- **input("...")**: Esta función muestra en pantalla el texto introducido y espera por el input del usuario (**cadena de texto**).

```
# Print & Input
print("Whatever", end="") # Omitimos el salto de línea

num_input = input("Introduzca un número: ")
if(num_input.isnumeric()):
    print(f"\nNúmero introducido: {num_input}")
else:
    print("\nNo es un número")
```

Para introducir ciertos caracteres especiales (**secuencias de escape**):

- '\n' : Salto de línea
- '\t' : tabulación
- '\'' : comilla simple
- '\"' : comilla doble

Input y Output

En vez de imprimir a la consola, podemos imprimir datos a un **fichero** (archivo)

- **open**: Para utilizar un fichero es necesario abrirlo. Hay distintas formas de abrir un fichero (sólo lectura, escritura, ambas, ...)
- **read**: Permite leer un fichero abierto y almacenar los datos en una variable
- **write**: Escribe datos en un fichero.
- **close**: Cierra el fichero. Es CRUCIAL cerrar los ficheros para evitar errores.

```
# Archivos
file = open("fichero_1.txt", "r") # Abrimos como lectura
texto = file.read()    # Guardamos el texto como una variable
print(texto)
file.close() # cerramos el archivo (IMPORTANTE)

# En modo escritura ("w") se SOBREESCRIBE el fichero
file = open("fichero_2.txt","w") # Abrimos como escritura
file.write("TEXTO ESCRITO")
file.close()

# Sintaxis alternativa (más elegante)
with open("fichero_2.txt", 'a') as file:
    file.write("texto añadido") # El modo append añade texto al final
    print(texto)

# Al usar with el archivo se cierra solo al acabar el bloque
```

- ‘**r**’ : *read* (no crea fichero nuevo)
- ‘**w**’ : *write* (CUIDADO, SOBREESCRIBE)
- ‘**r+**’ :*read/write* (no crea fichero)
- ‘**a**’ : añade al final (*append*), crea fichero.

Input y Output

Ejercicios:

- Imprimir los elementos de una lista separados por punto y coma (;), omitir el punto y coma en el último elemento.
 - Imprimir los elementos de esa lista a un fichero llamado “**lista.txt**”
 - ¿Qué pasa si el fichero no existe? ¿Y si ya existe?
 - Añadir nuevos elementos al fichero.
- Pedirle al usuario que introduzca un número e imprimir por pantalla su cuadrado. (a^2) (*casting*)
 - ¿Qué pasa si el usuario no introduce un número? Solucionar los problemas que ello conlleva con un mensaje de error. (**sanear inputs**)
 - Cambiar la solución anterior para que siga pidiendo un número o un carácter de escape. (**recuperarse de errores**) -

Recap IV

Recapitulamos:

Manipular cadenas de texto:

- **format strings** para formar cadenas de texto `f"Texto {variable} texto..."`

Input & Output:

- **print()**: para imprimir texto en pantalla
- **input("prompt")**: para recibir input de usuario, se escribe un **prompt**.

Manejar archivos:

- **open()** y **close()** para abrir y cerrar archivos.
- **read()** y **write()** para las operaciones de lectura y escritura.
- **IMPORTANTE cerrar** los ficheros al acabar con ellos.
- Para asegurarnos que no nos olvidamos de cerrar podemos usar la sintaxis de **with**

Funciones

Es muy común que porciones de código se repitan en un mismo script, para abreviar la cantidad de código que escribimos podemos usar **funciones**:

- Son **bloques de código**.
- Permiten **reutilizar** código
- Permiten **organizar** el código en bloques lógicos
- Se pueden introducir datos mediante **argumentos**

```
def funcion_vacia():
    pass          # La instrucción pass hace que no haga nada
# Definimos una función que pide un número y lo eleva un número al cuadrado
def elevar_cuadrado():
    num = input("Introduzca un número: ")
    if(num.isnumeric()):
        result = num**2
        print(f"\nEl cuadrado de {num} es {result}")
        return result
    else:
        print("\nEl dato introducido no es un número.")
        return None
```

La sintaxis para definir una función es: **def nombrefuncion (argumento_1, argumento_2, ...):**

Funciones

Consideraciones sobre funciones:

- Una función ha de estar **definida antes** de ser **llamada** (“definirla más arriba”)

- Una función puede devolver un dato o no, esto se conoce como **retorno**

- Se puede especificar el tipo de valor de retorno de una función

- En python las funciones pueden aceptar un número de argumentos variable (**funciones variadicas**)

- Las funciones han de ser **concisas**. Evitar las funciones enormes y difíciles de entender.

```
# Podemos especificar el tipo de retorno
def shitty_cuadrado(input) -> (int | None):
    if(type(input) is int): # Si es un int
        return input**2
    else:
        return None
```

```
def nombrefuncion (arg1, arg2, ...): -> int
```

Funciones

Llamadas a funciones:

```
nomrefuncion(arg1, arg2, ...)
```

Llamadas a métodos:

```
objeto.metodo()
```

- Los métodos son como “funciones” pero referidas a instancias de objetos.
Podéis pensar en ellos como “funciones internas”
- No se puede llamar a un método si no existe un objeto asociado.
- No entraremos en lo que son objetos (clases). (*Consultar orientación a objetos*)

Funciones

Ejercicios:

- Crea una función que le pida al usuario sus datos (Nombre, edad, puesto y departamento) y los añada a un fichero llamado “personal.txt”. No sobreescribir este fichero.
 - **Nota:** Dividir el problema en problemas más pequeños
- Ampliar la función anterior para asegurarse de que el formato de los datos es correcto. (**nombre** la primera en mayúsculas sin espacios, **edad** en número entero, **puesto** y **departamento** la primera en mayúscula).

Importar y descargar módulos

No hace falta que reinventemos la rueda. Podemos importar las funciones que han hecho otras personas para facilitar nuestro código. Estas funciones suelen estar organizadas en ficheros llamados **módulos**

- Python tiene módulos incorporados que no hace falta descargar (*math*, *datetime*, ...)
- Puedes importar sólo parte de un módulo (funciones específicas)
- Existe un instalador de paquetes de python oficial **pip** (*package installer for python*) mediante el cual podemos descargar paquetes oficiales
- **No descarguéis cualquier cosa**, descargad paquetes ampliamente usados. Recordad que estás ejecutando código en vuestra máquina.

- **import:** importa un módulo
- **as:** le das un **alias** a un módulo importado
- **from:** sintaxis alternativa para importar parte de un módulo.

```
from matplotlib import pyplot
```

Nota sobre scripts que llaman a otros scripts:

- Si queremos que una porción de código de nuestro script sólo se ejecute si es el script principal (normalmente usado en aplicaciones): `if __name__ == "__main__":`

```
# Importar y descargar módulos
import math      #Importas el módulo math entero
PI = math.pi
exponential = pow(math.e, 2)    # e^2
import matplotlib.pyplot as plt #Importas sólo pyplot
# matplotlib no es un paquete incluido en python base

if __name__ == "__main__":
    pass #Evitas que esta porción de código se ejecute
          # si es llamado desde otro script
```

Importar y descargar módulos

Ejercicios:

- Importar el módulo de python de matemáticas e imprimir los valores de π y de e con 6 decimales.
- Elige una o varias de las tareas que te interesen e impleméntala (os ayudo yo):
 - Hacer una gráfica de la función: $\text{sen}(2\pi \cdot x)$, $x \in [0, 2\pi]$; usar mínimo 1000 valores
 - Nota: requiere importar `pyplot` de `matplotlib`, entonces hay que descargar `matplotlib`. (En terminal: `pip install matplotlib`)
 - Hacer un script que pulse un botón en pantalla cada segundo.
 - Nota: requiere importar `pyautogui`. (`pip install pyautogui`)
 - Hacer un generador de códigos QR. (`pip install qrcode`)

Aplicaciones

- Hacer una aplicación de reloj digital. (Empezar con terminal)
 - Nota: requiere `tkinter`, ya incluido en python.
- ~~Hacer una aplicación para jugar al piedra papel tijera con el ordenador~~

Notas sobre el uso de IA

A la hora de usar ChatGPT, Claude, Deepseek, CoPilot, etc :

Ten en cuenta tus objetivos:

- **¿Quieres aprender de verdad?**: “La letra con sangre entra”, si lo escribe el ordenador no vas a aprender realmente: Google, Stack Overflow, GitHub, ...
- **¿Quieres tener un script mediocre pero funcional?**: “Pillage and plunder”, saquea la red de todo el contenido que puedas: Stack Overflow, ChatGPT, **lo que sea**.

Consideraciones adicionales:

- Estás ejecutando código en tu máquina, intenta ENTENDER lo que estás ejecutando aunque sea por encima porque **puede ser peligroso** (raro pero puede ser)
- Todo lo que escribas en estas interfaces de IA son datos que estás enviando y que ellos usan, **no des datos importantes al chat**.

Siguientes pasos

En función de lo que quieras conseguir con python, has de familiarizarte con lo siguiente:

Python/Programación en general:

- Uso de **clases** y **métodos**. Encapsulación y demás conceptos de **OOP** (Object Oriented Programming).
- **Excepciones** (`try`, `except`, `finally`,...) Recuperación de errores no fatales.

Cálculos matemáticos:

- **numpy**: Librería para cálculo numérico más usada.
- **matplotlib**: Para plotear gráficas (imita a Matlab)

Data science (aparte de las anteriores):

- **pandas**: La librería de manipulación de datos por excelencia.
- **scikit-learn**: Para Machine Learning básico (y no tan básico)

Automatización:

- **pyautogui**: Para automatizar el uso de interfaces de usuario (clicks en pantalla).

Hardware (Terminal serie):

- **pyserial**: Para acceder al puerto serie de forma multiplataforma (conexión USB).

Visión por ordenador:

- **opencv**: Manipulación de imágenes/vídeo ya sea a tiempo real o no.

Interfaces de usuario (GUI):

- **tkinter** o **Kivy**: Más pensado para aplicaciones. (Básicos, sobre todo tkinter)
- **Qt**: Existen módulos de Qt para python. (Más potente y difícil de usar)