

APUNTES DE INTRODUCCIÓN

Programación en Python

lo esencial

Sintaxis, estructuras de datos, funciones, clases, buenas prácticas y glosario

Autor: David M.
davmacsolutions.com

Octubre de 2026

ÍNDICE

1. Primeros pasos	2
2. Variables, tipos y operadores	2
3. Cadenas de texto	3
4. Estructuras de datos	4
5. Control de flujo	5
6. Comprensiones	6
7. Funciones	7
8. Errores y excepciones	8
9. Archivos	8
10. Módulos, paquetes y entornos	9
11. Programación orientada a objetos	10
12. Herramientas avanzadas	11
13. Buenas prácticas y errores frecuentes	13
14. Chuleta rápida	14
15. Glosario	15

1. PRIMEROS PASOS

Python es un lenguaje de programación interpretado, de propósito general y con una sintaxis pensada para ser legible. Se usa en análisis de datos, inteligencia artificial, desarrollo web, automatización de tareas, ciencia e ingeniería.

1.1 Instalación y ejecución

- Se descarga desde python.org; en Linux y macOS suele venir preinstalado. Trabaja siempre con Python 3.
- **Modo interactivo** (REPL): escribe `python` (o `python3`) en la terminal. El indicador `>>>` espera órdenes que se ejecutan al momento.
- **Script**: guarda el código en un archivo `programa.py` y ejecútalo con `python programa.py`.
- **Entornos de trabajo**: VS Code, PyCharm o Jupyter Notebook (muy usado en análisis de datos).

```
print("¡Hola, mundo!")
```

1.2 Reglas básicas de sintaxis

- **La indentación es obligatoria**: los bloques (cuerpos de `if`, `for`, funciones...) se delimitan con sangría, por convenio 4 espacios. No hay llaves.
- Las líneas que abren un bloque terminan en dos puntos `:`.
- Los comentarios empiezan con `#`.
- Python distingue mayúsculas de minúsculas: `edad` y `Edad` son variables distintas.

```
# Esto es un comentario
edad = 20
if edad >= 18:
    print("Mayor de edad")    # dentro del bloque (4 espacios)
print("Fin")                 # fuera del bloque
```

2. VARIABLES, TIPOS Y OPERADORES

Una variable es un nombre que apunta a un valor. No se declara su tipo: Python lo deduce (*tipado dinámico*).

```

nombre = "Ana"      # str (cadena de texto)
edad = 30           # int (entero)
altura = 1.68       # float (decimal, con punto)
activo = True       # bool (True / False)
nada = None         # NoneType (ausencia de valor)

type(edad)          # <class 'int'>
a, b = 1, 2         # asignación múltiple
a, b = b, a         # intercambio de valores

```

2.1 Conversiones y entrada de datos

```

int("42")          # 42
float("3.5")       # 3.5
str(10)            # '10'
bool(0)            # False (0, "", [], {}, None son "falsos")

texto = input("¿Cuántos años tienes? ") # input() siempre devuelve str
años = int(texto)

```

2.2 Operadores

Tipo	Operadores
Aritméticos	+ suma, - resta, * producto, / división (siempre float), // división entera, % resto, ** potencia
Comparación	== igual, != distinto, <, >, <=, >=
Lógicos	and, or, not
Pertenencia	in, not in ("a" in "casa" → True)
Identidad	is, is not (¿es el mismo objeto?)
Asignación compuesta	+=, -=, *=, /=, //=, %=, **=

```

7 / 2      # 3.5
7 // 2     # 3
7 % 2      # 1
2 ** 10    # 1024

```

3. CADENAS DE TEXTO

Las cadenas (`str`) se escriben entre comillas simples o dobles; con triples comillas pueden ocupar varias líneas. Son *inmutables*: sus métodos devuelven una cadena nueva.

```

s = "Python"
s[0]      # 'P'      (los índices empiezan en 0)
s[-1]     # 'n'      (índices negativos: desde el final)
s[0:3]    # 'Pyt'    (rebanada: desde 0 hasta 3, sin incluirlo)
s[::-1]   # 'nohtyP' (invertida)
len(s)    # 6

nombre, precio = "café", 1.5
f"El {nombre} cuesta {precio:.2f} €" # f-string: 'El café cuesta 1.50 €'

```

Método	Resultado
<code>s.upper(), s.lower()</code>	Mayúsculas, minúsculas.
<code>s.strip()</code>	Quita espacios al principio y al final.
<code>s.split(",")</code>	Divide en una lista usando el separador.
<code>", ".join(lista)</code>	Une una lista de cadenas con el separador.
<code>s.replace("a", "b")</code>	Sustituye un texto por otro.
<code>s.startswith("x"), s.endswith("x")</code>	Comprueba el inicio o el final.
<code>s.find("x")</code>	Posición de la primera aparición (−1 si no está).
<code>s.count("x")</code>	Número de apariciones.
<code>s.isdigit(), s.isalpha()</code>	¿Solo dígitos? ¿Solo letras?

4. ESTRUCTURAS DE DATOS

Tipo	Sintaxis	Ordenada	Mutable	Uso típico
Lista (list)	[1, 2, 3]	sí	sí	Secuencia de elementos que cambia.
Tupla (tuple)	(1, 2, 3)	sí	no	Datos fijos; varios valores de retorno.
Diccionario (dict)	{"a": 1}	sí*	sí	Asociar claves con valores.
Conjunto (set)	{1, 2, 3}	no	sí	Elementos únicos; pertenencia rápida.

* Los diccionarios conservan el orden de inserción desde Python 3.7.

4.1 Listas

```

frutas = ["manzana", "pera"]
frutas.append("uva")           # añade al final
frutas.insert(0, "kiwi")       # inserta en una posición
frutas.remove("pera")          # elimina por valor
ultima = frutas.pop()          # extrae y devuelve el último
frutas.sort()                  # ordena la propia lista
ordenada = sorted(frutas)      # devuelve una lista nueva ordenada
"kiwi" in frutas               # True
len(frutas)                    # número de elementos
numeros = [3, 1, 2]
sum(numeros), max(numeros), min(numeros) # (6, 3, 1)

```

4.2 Diccionarios

```
persona = {"nombre": "Ana", "edad": 30}
persona["edad"] # 30
persona["ciudad"] = "Madrid" # añade o modifica
persona.get("email", "-") # valor por defecto si la clave no existe
del persona["edad"] # elimina una clave

for clave, valor in persona.items():
    print(clave, valor)
```

4.3 Tuplas y conjuntos

```
punto = (3, 4)
x, y = punto # desempaquetado

a = {1, 2, 3}
b = {3, 4}
a | b # unión {1, 2, 3, 4}
a & b # intersección {3}
a - b # diferencia {1, 2}
set([1, 1, 2]) # {1, 2}: elimina duplicados
```

5. CONTROL DE FLUJO

5.1 Condicionales

```
nota = 7.5
if nota >= 9:
    calificacion = "Sobresaliente"
elif nota >= 7:
    calificacion = "Notable"
elif nota >= 5:
    calificacion = "Aprobado"
else:
    calificacion = "Suspenso"

# Expresión condicional (en una línea)
estado = "aprobado" if nota >= 5 else "suspenso"
```

5.2 Bucles

```

for i in range(5):          # 0, 1, 2, 3, 4
    print(i)

for i in range(2, 10, 2):    # 2, 4, 6, 8   (inicio, fin, paso)
    print(i)

for i, fruta in enumerate(["pera", "uva"]): # índice y valor
    print(i, fruta)

for nombre, nota in zip(["Ana", "Luis"], [8, 6]): # recorrer en paralelo
    print(nombre, nota)

n = 3
while n > 0:                # se repite mientras la condición sea cierta
    print(n)
    n -= 1

```

`break` sale del bucle; `continue` salta a la siguiente iteración; `pass` es una instrucción vacía que se usa cuando la sintaxis exige un bloque.

5.3 Coincidencia de patrones (*match*)

Disponible desde Python 3.10; útil para distinguir varios casos.

```

comando = "salir"
match comando:
    case "ayuda":
        print("Mostrando ayuda")
    case "salir" | "exit":
        print("Hasta luego")
    case _:
        # cualquier otro valor
        print("Comando desconocido")

```

6. COMPRESIONES

Permiten crear listas, diccionarios o conjuntos en una sola línea legible.

```

cuadrados = [x**2 for x in range(6)]          # [0, 1, 4, 9, 16, 25]
pares = [x for x in range(10) if x % 2 == 0]  # [0, 2, 4, 6, 8]
longitudes = {p: len(p) for p in ["sol", "luna"]} # {'sol': 3, 'luna': 4}
iniciales = {p[0] for p in ["ana", "alba", "eva"]} # {'a', 'e'}
total = sum(x**2 for x in range(6))           # generador: 55

```

Observación. Si una comprensión necesita varios `for` anidados y condiciones, suele ser más claro escribir un bucle normal.

7. FUNCIONES

```
def area_rectangulo(base, altura=1):
    """Devuelve el área de un rectángulo.""" # docstring
    return base * altura

area_rectangulo(4, 3) # 12 (argumentos posicionales)
area_rectangulo(altura=2, base=5) # 10 (argumentos con nombre)
area_rectangulo(4) # 4 (usa el valor por defecto)
```

7.1 Argumentos variables y retorno múltiple

```
def resumen(*numeros): # *args: número variable de posicionales (tupla)
    return min(numeros), max(numeros) # devuelve una tupla

minimo, maximo = resumen(4, 9, 1) # 1, 9

def ficha(**datos): # **kwargs: argumentos con nombre (diccionario)
    for clave, valor in datos.items():
        print(f"{clave}: {valor}")

ficha(nombre="Ana", edad=30)
```

7.2 Funciones lambda y de orden superior

```
doble = lambda x: x * 2 # función anónima de una sola expresión
alumnos = [("Ana", 8), ("Luis", 6), ("Eva", 9)]
sorted(alumnos, key=lambda a: a[1], reverse=True) # ordenar por nota
list(map(doble, [1, 2, 3])) # [2, 4, 6]
list(filter(lambda x: x > 1, [1, 2, 3])) # [2, 3]
```

7.3 Anotaciones de tipo

Son opcionales: Python no las comprueba al ejecutar, pero documentan el código y permiten que el editor o herramientas como mypy detecten errores.

```
def saludar(nombre: str, veces: int = 1) -> str:
    return ("Hola, " + nombre + ". ") * veces
```

7.4 Ámbito de las variables

Las variables creadas dentro de una función son *locales*: no existen fuera de ella. Una función puede leer variables *globales*, pero para modificarlas necesitaría declararlas con `global`, algo que conviene evitar: es mejor pasar valores como argumentos y devolver resultados con `return`.

8. ERRORES Y EXCEPCIONES

Cuando algo falla durante la ejecución, Python lanza una *excepción*. Si no se captura, el programa se detiene y muestra un mensaje (*traceback*) que indica el tipo de error y la línea donde ocurrió. Conviene leerlo de abajo arriba.

```
try:
    numero = int(input("Número: "))
    resultado = 10 / numero
except ValueError:
    print("No es un número válido")
except ZeroDivisionError:
    print("No se puede dividir entre cero")
else:
    print("Resultado:", resultado) # solo si no hubo error
finally:
    print("Fin") # se ejecuta siempre

def retirar(saldo, cantidad):
    if cantidad > saldo:
        raise ValueError("Saldo insuficiente") # lanzar una excepción
    return saldo - cantidad
```

Excepción	Causa habitual
SyntaxError	Código mal escrito: falta :, paréntesis o comillas.
IndentationError	Sangría incorrecta o mezcla de tabuladores y espacios.
NameError	Variable o función no definida (o mal escrita).
TypeError	Operación con tipos incompatibles, p. ej. "3" + 4.
ValueError	Tipo correcto pero valor no válido, p. ej. int("hola").
IndexError	Índice fuera de rango en una lista.
KeyError	Clave inexistente en un diccionario.
AttributeError	El objeto no tiene ese método o atributo.
ZeroDivisionError	División entre cero.
FileNotFoundError	El archivo no existe en la ruta indicada.
ModuleNotFoundError	Módulo no instalado o nombre incorrecto.

9. ARCHIVOS

La forma recomendada de abrir archivos es con `with`, que los cierra automáticamente aunque se produzca un error.

```
# Escribir ("w" sobrescribe, "a" añade al final)
with open("notas.txt", "w", encoding="utf-8") as f:
    f.write("Primera línea\n")

# Leer línea a línea
with open("notas.txt", encoding="utf-8") as f:
    for linea in f:
        print(linea.strip())
```

9.1 JSON, CSV y rutas

```
import json, csv
from pathlib import Path

datos = {"nombre": "Ana", "notas": [8, 9]}
with open("datos.json", "w", encoding="utf-8") as f:
    json.dump(datos, f, ensure_ascii=False, indent=2)
with open("datos.json", encoding="utf-8") as f:
    datos = json.load(f)

with open("tabla.csv", "w", newline="", encoding="utf-8") as f:
    escritor = csv.writer(f)
    escritor.writerow(["nombre", "nota"])
    escritor.writerow(["Ana", 8])
with open("tabla.csv", newline="", encoding="utf-8") as f:
    for fila in csv.DictReader(f):
        print(fila["nombre"], fila["nota"])

ruta = Path("datos") / "informe.txt" # rutas independientes del sistema
ruta.parent.mkdir(exist_ok=True)
ruta.write_text("Hola", encoding="utf-8")
ruta.exists() # True
```

10. MÓDULOS, PAQUETES Y ENTORNOS

Un *módulo* es un archivo `.py`; un *paquete* es una carpeta de módulos. La *biblioteca estándar* viene incluida con Python; el resto se instala con pip desde PyPI.

```
import math
math.sqrt(16) # 4.0
from random import randint, choice
randint(1, 6) # entero aleatorio entre 1 y 6
import datetime as dt # alias
dt.date.today()
```

10.1 Entornos virtuales

Cada proyecto debería tener su propio entorno virtual, para que sus dependencias no interfieran con las de otros proyectos.

```
python -m venv .venv          # crear el entorno
source .venv/bin/activate     # activar (Linux / macOS)
.venv\Scripts\activate       # activar (Windows)
pip install requests          # instalar un paquete
pip freeze > requirements.txt # guardar las dependencias
pip install -r requirements.txt # reinstalarlas en otro equipo
```

10.2 El punto de entrada de un script

```
def main():
    print("Programa principal")

if __name__ == "__main__": # solo se ejecuta si se lanza el archivo directamente,
    main()                 # no cuando se importa como módulo
```

11. PROGRAMACIÓN ORIENTADA A OBJETOS

Una *clase* es un molde que agrupa datos (*atributos*) y comportamiento (*métodos*); cada *objeto* creado a partir de ella es una *instancia*.

```
class Cuenta:
    def __init__(self, titular, saldo=0): # constructor
        self.titular = titular           # atributos de la instancia
        self.saldo = saldo

    def ingresar(self, cantidad):          # método
        self.saldo += cantidad

    def __str__(self):                    # texto al hacer print()
        return f"{self.titular}: {self.saldo} €"

c = Cuenta("Ana", 100)
c.ingresar(50)
print(c)                                # Ana: 150 €
```

11.1 Herencia

```
class CuentaAhorro(Cuenta):                                # hereda de Cuenta
    def __init__(self, titular, saldo=0, interes=0.02):
        super().__init__(titular, saldo)                  # llama al constructor padre
        self.interes = interes

    def aplicar_interes(self):
        self.ingresar(self.saldo * self.interes)

isinstance(CuentaAhorro("Luis"), Cuenta)                # True
```

11.2 Clases de datos

Para clases que solo almacenan datos, `dataclass` genera automáticamente el constructor, la representación y la comparación.

```
from dataclasses import dataclass

@dataclass
class Producto:
    nombre: str
    precio: float
    stock: int = 0

p = Producto("Libro", 12.5)
print(p)           # Producto(nombre='Libro', precio=12.5, stock=0)
```

Método especial	Se usa para
<code>__init__</code>	Inicializar el objeto al crearlo.
<code>__str__</code> , <code>__repr__</code>	Texto legible para el usuario y representación técnica.
<code>__len__</code>	Permitir <code>len(objeto)</code> .
<code>__eq__</code> , <code>__lt__</code>	Comparar con <code>==</code> y <code><</code> .
<code>__add__</code>	Definir el operador <code>+</code> .
<code>__iter__</code>	Hacer el objeto recorrible con <code>for</code> .

12. HERRAMIENTAS AVANZADAS

12.1 Iteradores y generadores

Un *generador* produce valores de uno en uno con `yield`, sin guardar todos en memoria. Es ideal para secuencias grandes o infinitas.

```
def cuenta_atras(n):
    while n > 0:
        yield n
        n -= 1

for i in cuenta_atras(3):
    print(i)                # 3, 2, 1
```

12.2 Decoradores

Un decorador es una función que recibe otra función y devuelve una versión ampliada de ella. Se aplica con @.

```
import time

def cronometrar(funcion):
    def envoltura(*args, **kwargs):
        inicio = time.perf_counter()
        resultado = funcion(*args, **kwargs)
        print(f"{funcion.__name__}: {time.perf_counter() - inicio:.4f} s")
        return resultado
    return envoltura

@cronometrar
def calcular():
    return sum(range(1_000_000))
```

12.3 Biblioteca estándar

Módulo	Para qué sirve
math, statistics	Funciones matemáticas; media, mediana, desviación típica.
random	Números aleatorios, barajar, elegir al azar.
datetime	Fechas, horas y diferencias de tiempo.
os, sys, pathlib	Sistema operativo, argumentos del programa, rutas de archivos.
json, csv	Leer y escribir esos formatos.
re	Expresiones regulares (búsqueda de patrones en texto).
collections	Counter, defaultdict, deque, namedtuple.
itertools, functools	Combinaciones, permutaciones; reduce, lru_cache.
logging	Registro de mensajes del programa (mejor que print en producción).
unittest	Pruebas automatizadas.
argparse	Opciones de línea de órdenes.

12.4 Bibliotecas externas más populares

Área	Bibliotecas
Cálculo y datos	NumPy (matrices y cálculo numérico), pandas (tablas de datos)
Gráficos	Matplotlib, Seaborn, Plotly
Aprendizaje automático	scikit-learn, PyTorch, TensorFlow
Web	Django, Flask, FastAPI
Peticiones HTTP y <i>scraping</i>	requests, httpx, BeautifulSoup
Pruebas y calidad	pytest, Ruff (estilo y errores), mypy (tipos), Black (formato)

13. BUENAS PRÁCTICAS Y ERRORES FRECUENTES

13.1 Estilo (PEP 8)

- Sangría de 4 espacios; líneas de longitud razonable (79–100 caracteres).
- `snake_case` para variables y funciones, `PascalCase` para clases, `MAYUSCULAS` para constantes.
- Nombres descriptivos: `precio_total` mejor que `pt`.
- Funciones cortas que hagan una sola cosa, con `docstring`.
- Usa un formateador automático (Black o Ruff) y un control de versiones (Git).

13.2 Trampas habituales

ATENCIÓN. Valores por defecto mutables. El valor por defecto se crea una sola vez y se comparte entre llamadas.

```
def añadir(x, lista=[]):      # MAL: la lista se reutiliza
    lista.append(x)
    return lista

def añadir(x, lista=None):   # BIEN
    if lista is None:
        lista = []
    lista.append(x)
    return lista
```

ATENCIÓN. Copiar no es asignar. `b = a` no copia la lista: ambos nombres apuntan al mismo objeto.

```
a = [1, 2, 3]
b = a          # mismo objeto: modificar b modifica a
c = a.copy()   # copia superficial (también a[:] o list(a))
import copy
d = copy.deepcopy([a, b]) # copia profunda (estructuras anidadas)
```

ATENCIÓN. Decimales en coma flotante. $0.1 + 0.2 == 0.3$ es False, porque los decimales se almacenan en binario de forma aproximada. Compara con `math.isclose(a, b)` y usa el módulo `decimal` para cálculos monetarios.

Error	Qué hacer
Usar <code>is</code> para comparar valores	<code>is</code> compara identidad; usa <code>==</code> . Reserva <code>is</code> para None.
Modificar una lista mientras se recorre	Recorre una copia o crea una lista nueva con una comprensión.
Olvidar que <code>input()</code> devuelve texto	Convierte con <code>int()</code> o <code>float()</code> .
Sumar texto y números	Usa f-strings: <code>f"Edad: {edad}"</code> .
<code>except</code> : sin tipo de error	Captura excepciones concretas; uno genérico oculta fallos.
Llamar a un archivo igual que un módulo	Un <code>random.py</code> propio impide importar el <code>random</code> estándar.
Instalar paquetes globalmente	Usa un entorno virtual por proyecto.

14. CHULETA RÁPIDA

Tarea	Código
Mostrar y leer	<code>print(a, b, sep=", ")</code> <code>x = input("Texto: ")</code>
Longitud	<code>len(x)</code>
Rango de números	<code>range(inicio, fin, paso)</code>
Recorrer con índice	<code>for i, v in enumerate(lista):</code>
Recorrer dos listas	<code>for a, b in zip(l1, l2):</code>
Recorrer diccionario	<code>for k, v in d.items():</code>
Ordenar	<code>sorted(lista, key=..., reverse=True)</code>
Invertir	<code>lista[::-1]</code> <code>reversed(lista)</code>
Contar apariciones	<code>from collections import Counter; Counter(lista)</code>
Redondear	<code>round(3.14159, 2) → 3.14</code>
Tipo de un objeto	<code>type(x)</code> <code>isinstance(x, int)</code>
Ayuda	<code>help(funcion)</code> <code>dir(objeto)</code>
Comprobar todos / alguno	<code>all(lista)</code> <code>any(lista)</code>
Valor absoluto, potencia	<code>abs(x)</code> <code>pow(x, y)</code> <code>x ** y</code>
Cociente y resto	<code>divmod(17, 5) → (3, 2)</code>
Formato numérico	<code>f"{x:.2f}"</code> <code>f"{n:,}"</code> <code>f"{p:.1%}"</code>
Fecha y hora actual	<code>from datetime import datetime; datetime.now()</code>
Leer archivo completo	<code>Path("a.txt").read_text(encoding="utf-8")</code>
Medir tiempo	<code>time.perf_counter()</code>

15. GLOSARIO

15.1 Símbolos y sintaxis

Símbolo	Significado
#	Inicio de comentario: Python ignora el resto de la línea.
:	Abre un bloque (if, for, def, class...); también separa clave y valor en diccionarios y los límites de una rebanada.
=	Asignación: guarda un valor en una variable.
==, !=	Comparación de igualdad y desigualdad.
//	División entera (descarta los decimales).
%	Resto de la división (módulo).
**	Potencia. En una llamada o definición de función, desempaqueta o recoge argumentos con nombre.
*	Producto. En funciones, recoge o desempaqueta argumentos posicionales; en listas, repite ([0] * 3).
:=	Operador morsa: asigna dentro de una expresión (if (n := len(x)) > 10:).
->	En una definición de función, indica el tipo del valor devuelto.
@	Aplica un decorador; entre matrices de NumPy, producto matricial.
.	Acceso a atributos y métodos (objeto.metodo()).
()	Llamada a función, tuplas, agrupación de expresiones.
[]	Listas, acceso por índice o clave, rebanadas, comprensiones.
{ }	Diccionarios, conjuntos; dentro de un f-string, expresiones a insertar.
[a:b:c]	Rebanada: desde a hasta b (sin incluir) con paso c.
_	Variable que se descarta (for _ in range(3)); en match, «cualquier valor»; separador de miles en números (1_000).
_nombre	Por convenio, atributo o función de uso interno.
__nombre__	Nombre especial (<i>dunder</i>) que Python usa internamente: __init__, __name__.
f"..."	Cadena formateada (f-string).
r"..."	Cadena cruda: las barras invertidas no se interpretan (útil en expresiones regulares y rutas de Windows).
b"..."	Secuencia de bytes.
"""..."""	Cadena de varias líneas; también docstring.
\n, \t	Salto de línea, tabulador.
>>>	Indicador del intérprete interactivo.
...	Ellipsis; marcador de código pendiente. También indicador de continuación en el intérprete.
\	Carácter de escape en cadenas; al final de una línea, la continúa en la siguiente.

15.2 Siglas

Sigla	Nombre completo	Significado
API	Application Programming Interface	Interfaz que una biblioteca o servicio ofrece para usarlo desde código.
CLI	Command Line Interface	Programa que se usa desde la terminal.
CPython	—	Implementación oficial de Python, escrita en C.
CSV	Comma-Separated Values	Formato de texto para tablas, con valores separados por comas.
DRY	Don't Repeat Yourself	Principio: evitar duplicar código.
EAFP	Easier to Ask Forgiveness than Permission	Estilo de Python: intentar la operación y capturar la excepción.
GIL	Global Interpreter Lock	Bloqueo de CPython que limita la ejecución simultánea de hilos de Python.
GUI	Graphical User Interface	Interfaz gráfica (ventanas, botones).
HTTP	HyperText Transfer Protocol	Protocolo de comunicación de la web.
IDE	Integrated Development Environment	Editor con herramientas de programación (PyCharm, VS Code).
JSON	JavaScript Object Notation	Formato de intercambio de datos parecido a diccionarios y listas.
LBYL	Look Before You Leap	Estilo opuesto a EAFP: comprobar antes de actuar.
OOP / POO	Object-Oriented Programming	Programación orientada a objetos.
PEP	Python Enhancement Proposal	Documento que propone o define cambios y normas del lenguaje (PEP 8: guía de estilo).
PyPI	Python Package Index	Repositorio oficial de paquetes, del que instala pip.
pip	Package Installer for Python	Gestor de paquetes de Python.
REPL	Read-Eval-Print Loop	Intérprete interactivo: lee, evalúa, muestra el resultado y repite.
SQL	Structured Query Language	Lenguaje para consultar bases de datos (módulo sqlite3).
UTF-8	Unicode Transformation Format	Codificación de texto estándar; admite tildes, eñes y emojis.
venv	virtual environment	Entorno virtual aislado para las dependencias de un proyecto.

15.3 Términos

Argumento. Valor que se pasa a una función al llamarla. El nombre que lo recibe en la definición es el *parámetro*.

Atributo. Dato asociado a un objeto (`cuenta.saldo`).

Biblioteca. Conjunto de módulos reutilizables. La *biblioteca estándar* viene incluida con Python.

Bucle. Estructura que repite un bloque de código (`for`, `while`).

Clase. Plantilla que define los atributos y métodos de un tipo de objeto.

Depurar. Encontrar y corregir errores (*bugs*) en un programa.

Diccionario. Colección de pares clave-valor.

Docstring. Cadena de documentación al inicio de una función, clase o módulo.

Excepción. Error que se produce durante la ejecución y que puede capturarse con `try`.

Expresión. Fragmento de código que produce un valor (`2 + 3`). Una *sentencia* es una instrucción completa.

Función. Bloque de código con nombre que realiza una tarea y puede devolver un resultado.

Generador. Función que produce valores de uno en uno con `yield`.

Herencia. Mecanismo por el que una clase hija reutiliza y amplía una clase padre.

Índice. Posición de un elemento en una secuencia; empieza en 0.

Inmutable. Objeto que no puede cambiar tras crearse (`int`, `str`, `tuple`).

Instancia. Objeto concreto creado a partir de una clase.

Intérprete. Programa que ejecuta el código Python línea a línea.

Iterable. Objeto que se puede recorrer con `for` (listas, cadenas, diccionarios, archivos...).

Método. Función que pertenece a un objeto o clase (`lista.append()`).

Módulo. Archivo `.py` cuyo contenido puede importarse.

Mutable. Objeto que puede modificarse tras crearse (`list`, `dict`, `set`).

Paquete. Carpeta que agrupa módulos relacionados.

Pythónico. Código que sigue las convenciones e idiomas propios de Python.

Rebanada. Porción de una secuencia (*slice*): `lista[1:4]`.

Script. Archivo de código que se ejecuta como programa.

Tipado dinámico. El tipo pertenece al valor, no a la variable, y se determina al ejecutar.

Traceback. Informe de error que muestra la secuencia de llamadas hasta el fallo.

Tupla. Secuencia ordenada e inmutable.

Variable. Nombre que hace referencia a un valor almacenado en memoria.