DAVID GUAMANGA GÓMEZ

www.gelenia.com

FUNDAMENTOS DE PYTHON

Guía práctica y didáctica para aprender desde cero

★ **Explicaciones claras · Ejemplos · Retos · Buenas prácticas**

Python 3.14+ · Primera edición 2026

01

Cómo usar esta guía

Una ruta visual para aprender sin memorizar de forma mecánica.

Esta guía fue diseñada como material didáctico independiente para aprender los fundamentos de Python mediante explicaciones breves, ejemplos pequeños y ejercicios progresivos. El objetivo no es memorizar comandos: es comprender qué ocurre cuando un programa se ejecuta.

La estructura sigue una progresión pedagógica propia: primero construimos el modelo mental de Python, después trabajamos con datos y control del programa, y finalmente llegamos a funciones, archivos, módulos, errores, objetos y herramientas del ecosistema.

ⓘ**Referencia técnica**

Python 3.14.7. La documentación oficial separa tutorial, referencia del lenguaje, funciones incorporadas, tipos y biblioteca estándar.

Ruta de aprendizaje

✓ **Autoría**

El contenido, redacción, ejemplos y estructura de esta edición fueron creados para David Guamanga Gómez; no reproduce el texto ni el diseño de la guía de referencia proporcionada.

02

Pensar como programador

Antes del código: problema → algoritmo → solución.

¿Qué es programar?

Programar consiste en describir un proceso de forma que una máquina pueda ejecutarlo. Un programa combina datos, reglas y decisiones para transformar una entrada en una salida.

Algoritmo

Un algoritmo es una secuencia finita de pasos para resolver un problema. Antes de escribir código, expresa la solución con palabras: entrada → proceso → salida.

ENTRADA	PROCESO	SALIDA
Datos que recibimos	Reglas y operaciones	Resultado esperado

02 · Tu primer programa

El código se puede ejecutar desde un archivo `.py` o desde el intérprete interactivo. Python usa sangría para delimitar bloques.

Tu primer programa

```
print("Hola, Python")
```

Variables

Una variable es un nombre asociado a un objeto. Python tiene tipado dinámico.

Ejemplo

```
nombre = "David"
edad = 30
print(nombre, edad)
```

> **★ Reto**
>
> Crea variables para tu nombre, una ciudad y un año, y muestra una frase que las combine.

03 Datos y operadores

Reconoce qué tipo de información estás manejando.

Tipos de datos

Tipo	Idea clave
int	Números enteros
float	Números de punto flotante
complex	Números complejos
bool	True / False
str	Texto; secuencia inmutable
None	Ausencia de valor; su tipo es NoneType

Colecciones

Python incorpora `list`, `tuple`, `set`, `frozenset`, `dict` y `range`, además de tipos binarios como `bytes`, `bytearray` y `memoryview`.

> ✓ **Identidad y pertenencia**
>
> `is` comprueba identidad; `in` comprueba pertenencia. Para comprobar tipos usa `isinstance()`.

Operadores esenciales

Aritméticos	`+, -, *, /, //, %, **`
Comparación	`==, !=, <, >, <=, >=`
Lógicos	`and, or, not`
Asignación	`=, +=, -=, *=, /=`
Pertenencia	`in, not in`
Identidad	`is, is not`

Ejemplo

```
precio = 100
descuento = 0.15
final = precio * (1 - descuento)
print(final)
```

04

Decisiones y bucles

Controla qué hace el programa y cuántas veces lo hace.

Condicionales

`if`, `elif` y `else` permiten ejecutar bloques según una condición. Python también dispone de `match`.

Condición

```
edad = 20
if edad >= 18:
    print("Mayor de edad")
else:
    print("Menor de edad")
```

Repetición

`for`	Recorre elementos de un iterable.
`while`	Repite mientras una condición sea verdadera.
`break`	Termina el bucle.
`continue`	Pasa a la siguiente iteración.
`pass`	No realiza ninguna acción.

> **★ Reto**
>
> Recorre los números del 1 al 20 e imprime solamente los pares.

05 · Colecciones

Listas

Una `list` es ordenada y mutable. Puedes agregar, eliminar, modificar y recorrer elementos.

Lista

```
frutas = ["manzana", "pera", "mango"]
frutas.append("uva")
print(frutas[0])
```

Tuplas · Conjuntos · Diccionarios

- ✓ `tuple`: ordenada e inmutable.
- ✓ `set`: contiene elementos únicos. `set.pop()` devuelve un elemento arbitrario.
- ✓ `dict`: asocia claves con valores.

Diccionario

```
persona = {"nombre": "David", "edad": 30}
print(persona["nombre"])
persona["edad"] = 31
```

06 Funciones

Divide problemas en piezas reutilizables.

Qué es una función

Una función agrupa instrucciones reutilizables. Se define con `def`, recibe argumentos y puede devolver un resultado. Python también admite parámetros por nombre, valores predeterminados, `*args`, `**kwargs`, lambdas y anotaciones.

Función con parámetros y retorno

```
def calcular_total(precio, impuesto):
    """Devuelve el precio con impuesto."""
    return precio * (1 + impuesto)

print(calcular_total(100, 0.19))
```

Funciones incorporadas

`len()`	`type()`	`print()`	`input()`
`sum()`	`min()`	`max()`	`sorted()`
`enumerate()`	`zip()`	`isinstance()`	

! Importante

`reduce()` no es una función incorporada global: pertenece a `functools` y se importa con `from functools import reduce`.

★ Reto

Escribe una función que reciba una lista de números y devuelva el promedio.

07 · Errores y archivos

Las excepciones permiten manejar problemas durante la ejecución de forma controlada.

Manejo de errores

```
try:
    numero = int(input("Número: "))
    print(10 / numero)
except ValueError:
    print("Debes escribir un entero.")
except ZeroDivisionError:
    print("No puedes dividir entre cero.")
```

08

Módulos y paquetes

Organiza, reutiliza y aísla dependencias.

Módulos

Un módulo es un archivo Python con código reutilizable. Se incorpora con `import` o `from ... import ...`.

Biblioteca estándar

Python incluye una biblioteca estándar amplia como `pathlib`, `os`, `json`, `csv`, `datetime`, `math`, `statistics` y `logging`.

Trabajar con rutas

```
from pathlib import Path
ruta = Path("notas.txt")
print(ruta.exists())
```

Paquetes externos

PyPI reúne paquetes de terceros. Para proyectos reales conviene aislar dependencias mediante entornos virtuales.

Entorno virtual

```
python -m venv .venv
source .venv/bin/activate
pip install requests
```

09 · Objetos y buenas prácticas

En Python, los valores que utilizas son objetos. Tienen identidad, tipo y valor. Una clase define estructura y comportamiento para objetos.

Clase y objeto

```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

    def saludar(self):
        return f"Hola, soy {self.nombre}"

p = Persona("David")
print(p.saludar())
```

ⓘ **Detalles útiles**

`id(objeto)` devuelve un identificador entero del objeto durante su vida. `lista.copy()` o `copy.copy(objeto)` son formas de copiar objetos; `copy()` no es una función incorporada global.

10

Python en la práctica

Convierte los fundamentos en pequeños proyectos.

Proyectos sugeridos

01 **Calculadora de consola**
Practica entradas, operaciones y manejo de errores.

02 **Lista de tareas**
Guarda tareas y usa un archivo como persistencia.

03 **Analizador de texto**
Cuenta palabras y líneas de un texto.

04 **Procesamiento de CSV**

Lee y transforma datos tabulares.

05 **API + JSON**

Consulta una API pública y procesa su respuesta.

> **✓ Regla de oro**
>
> No memorices todo: aprende a leer documentación, dividir problemas, probar partes pequeñas y construir soluciones iterativamente.

Glosario esencial

argumento	Valor entregado al llamar una función.
atributo	Dato asociado a un objeto.
clase	Definición a partir de la cual se crean objetos.
iterable	Objeto que puede recorrerse elemento a elemento.
iterador	Objeto que produce valores sucesivos.
método	Función asociada a un objeto o tipo.
módulo	Archivo Python con código reutilizable.
objeto	Entidad con identidad, tipo y valor.
paquete	Conjunto organizado de módulos.
excepción	Evento que altera el flujo normal de ejecución.
mutable	Objeto cuyo contenido puede cambiar.
inmutable	Objeto cuyo valor no puede cambiar una vez creado.

> **ⓘ Referencia técnica**
>
> La referencia técnica principal para verificar lenguaje, funciones, tipos, control de flujo y biblioteca estándar es la documentación oficial de Python 3.14.7.

Autor: David Guamanga Gómez · Sitio web: www.gelenia.com · Edición: 2026