



Cofinanciado por
la Unión Europea



Fondos Europeos

Python Inicio

Índice

- 1.Instalar Python
- 2.Gestión de paquetes
- 3.Tipos de datos
- 4.Conversión de tipos
 - Explícita
 - Implícita
- 5.variables en Python
- 6.Ámbito de variables
 - variables locales
 - variables globales
- 7.Variables no definidas
- 8.Variables mutables e inmutables
- 9.Asignación de múltiples variables
- 10.Variables de clase e instancia
- 11.Operadores
 - Aritméticos
 - Relacionales
 - De asignación
 - Lógicos
 - De identidad
 - De pertenencia



1. Instalar Python

Descargar Python desde: <https://www.python.org/downloads/>

Existen dos formas de ejecutar código Python:

- **Sesión interactiva:** podemos escribir líneas de código en el intérprete y obtener una respuesta. Para iniciar el intérprete en línea de comandos, escribimos python3. Para salir de él, escribimos exit() o bien usamos Control + D.
- **Archivo:** podemos escribir el código de un programa en un archivo de texto con la extensión .py y ejecutarlo.

Ejecutar Python en Windows

Dado que los archivos .py están asociados al intérprete de Python, en algunos casos basta con hacer doble clic en el archivo para ejecutarlo. Sin embargo, si el programa se ejecuta en una ventana de consola, esta puede cerrarse rápidamente. Para evitar esto, es recomendable abrir el **Símbolo del sistema** (cmd), navegar hasta la carpeta del script con cd y ejecutarlo con:

```
python archivo.py
```

O bien desde PowerShell:

```
python archivo.py
```

Si la variable de entorno PATH no está configurada correctamente, es posible que sea necesario usar la ruta completa del ejecutable de Python.

Ejecutar Python en Linux

Para que el sistema operativo ejecute un archivo .py con el intérprete adecuado, es necesario añadir una línea especial al principio del archivo llamada **shebang**.

Shebang en Python

La primera línea del archivo Python debe ser:

```
#!/usr/bin/env python3
```





El par de caracteres `#!` indica al sistema operativo que dicho script se debe ejecutar utilizando el intérprete especificado. `env` busca automáticamente el ejecutable de Python en el sistema, evitando problemas con diferentes rutas de instalación.

Después de añadir el shebang, es necesario otorgar permisos de ejecución al archivo con:

```
chmod +x hola.py
```

Luego, podemos ejecutar el script directamente en la terminal con:

```
./hola.py
```

2. Gestión de paquetes en Python

Python incluye muchos paquetes y módulos por defecto, lo que se conoce como la **Librería Estándar**. Sin embargo, es posible instalar muchos más según nuestras necesidades.

Para gestionar los paquetes, utilizamos la herramienta **pip**, una utilidad incluida en la instalación de Python que permite instalar, desinstalar y actualizar paquetes.

Ejemplo de instalación de un paquete con pip:

```
pip install pandas
```

Ejecutar un script en Linux

Si tenemos un archivo Python (`hola.py`), podemos otorgarle permisos de ejecución con:

```
chmod +x hola.py
```

Luego, podemos ejecutarlo directamente en la terminal con:

```
./hola.py
```

3. Tipos de datos en Python

Python tiene un sistema de **tipado dinámico y fuertemente tipado**, lo que significa:

Tipado dinámico: No es necesario declarar el tipo de dato de una variable. Su tipo se determina en tiempo de ejecución según el valor asignado. Además, el tipo de una variable puede cambiar si se le asigna un valor de otro tipo.

```
x = 10      # x es un entero (int)
x = "Hola"  # Ahora x es una cadena de texto (str)
```

Fuertemente tipado: Python no permite tratar una variable como si fuera de otro tipo sin una conversión explícita. Si intentamos operar con tipos incompatibles, obtendremos un error.

```
x = "10"
y = 5
print(x + y)  # ❌ Error: no se puede sumar una cadena con un entero
```

Para operar con distintos tipos, es necesario hacer una conversión explícita:

```
print(int(x) + y)  # ✅ Correcto: convierte "10" a entero antes de sumar
```

Tipo de Dato	Descripción	Ejemplo
int	Números enteros (positivos, negativos o cero)	x = 10
float	Números decimales (punto flotante)	y = 3.14
complex	Números complejos con parte real e imaginaria	z = 2 + 3j
bool	Valores lógicos (Verdadero o Falso)	es_mayor = True
str	Cadenas de texto (entre comillas simples o dobles)	mensaje = "Hola"
bytes	Secuencia de bytes inmutable	b = b"texto"
bytearray	Secuencia de bytes modificable	ba = bytearray(5)
memoryview	Vista en memoria de un objeto de bytes	mv = memoryview(b"Hola")
NoneType	Representa la ausencia de valor (None)	resultado = None

Estos son los tipos básicos, pero Python también tiene estructuras de datos como listas, tuplas, diccionarios y conjuntos, que permiten manejar colecciones de valores.



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.ª ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



Uso de type() en Python

La función type() se usa para conocer el tipo de dato de una variable en Python.

Ejemplo:

```
entero = 23
print(type(entero)) # Output: <class 'int'>
```

Otros ejemplos con distintos tipos de datos:

```
decimal = 3.14
texto = "Hola"
booleano = True

print(type(decimal)) # <class 'float'>
print(type(texto)) # <class 'str'>
print(type(booleano)) # <class 'bool'>
```

Comprobar el tipo de una variable con isinstance()

Además de type(), en Python podemos usar la función isinstance() para verificar si una variable es de un tipo específico.

Sintaxis:

```
isinstance(variable, tipo)

x = 10
print(isinstance(x, int)) # True

y = 3.14
print(isinstance(y, float)) # True

texto = "Hola"
print(isinstance(texto, str)) # True

# Comprobación con múltiples tipos
print(isinstance(x, (int, float))) # True
```

◊ Diferencia entre type() e isinstance():

- type() devuelve el tipo exacto de la variable.
- isinstance() permite comprobar si una variable pertenece a un tipo o a varios.



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.ª ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



4. Conversión de tipos en Python

En Python, hay dos formas de convertir datos de un tipo a otro:

1. **Conversión implícita** (automática).
2. **Conversión explícita** (manual, usando funciones).

Conversión implícita

Python convierte automáticamente los valores al tipo de **mayor rango** cuando se mezclan diferentes tipos de datos en una operación.

Tipo menor → Tipo mayor	Ejemplo	Resultado
bool → int	True + 1	2 (True se convierte en 1)
int → float	5 + 3.2	8.2 (5 se convierte en 5.0)
bool → float	False + 2.5	2.5 (False se convierte en 0.0)

- ◊ Python **nunca convierte automáticamente** un número a str, ya que esto puede causar ambigüedad.

Ejemplo de conversión implícita:

```
num_entero = 10
num_flotante = 5.5
resultado = num_entero + num_flotante # Python convierte 10 a 10.0 automáticamente
print(resultado, type(resultado)) # 15.5 <class 'float'>
```

Conversión explícita

Para convertir datos manualmente, usamos las siguientes funciones:

Función	Descripción	Ejemplo	Resultado
bool()	Convierte a booleano	bool(0)	False
int()	Convierte a entero	int(3.9)	3
float()	Convierte a flotante	float(5)	5.0
str()	Convierte a cadena	str(123)	"123"

Ejemplo de conversión explícita:

```
edad = 25
mensaje = "Tengo " + str(edad) + " años."
print(mensaje) # Tengo 25 años.
```

Nota: Usaremos mucho str() cuando queramos mostrar en pantalla un número dentro de un texto con print().



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.^a ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



5. Variables en Python

En Python, los nombres de variables deben seguir una serie de reglas:

1. Carácteres permitidos:

- Letras minúsculas (a-z) y mayúsculas (A-Z).
- Dígitos (0-9).
- Guiones bajos (_).

2. Restricciones:

- Deben comenzar con una letra o un guion bajo (_), pero **nunca** con un dígito.
- No pueden ser una palabra reservada del lenguaje (**keywords**).

3. Sensible a mayúsculas y minúsculas:

- Python distingue entre mayúsculas y minúsculas. Es decir, edad y Edad son variables distintas.

Ejemplos de nombres válidos e inválidos:

 Válidos:	 Inválidos:
<pre>mi_variable = 10 _variable = "Hola" Variable123 = 3.14</pre>	<pre>2variable = 5 # ✗ Error: empieza con un número class = "Python" # ✗ Error: 'class' es una palabra reservada mi-variable = 20 # ✗ Error: contiene un guion (-) en lugar de un guion bajo</pre>

Las variables son fundamentales en Python, ya que permiten asignar nombres a los valores en memoria para su uso en el programa. **No es necesario indicar el tipo de la variable al declararla**, ya que Python usa tipado dinámico.

Ejemplo de declaración de variables:

```
nombre = "Ana"
edad = 25
precio = 19.99
```





6. Ámbito de las variables

El **ámbito** se refiere al contexto en el que una variable es accesible. Hay diferentes ámbitos en Python, y es importante comprender cómo funcionan:

- **Ámbito global:** Las variables definidas fuera de cualquier función son accesibles en todo el programa.
- **Ámbito local:** Las variables definidas dentro de una función sólo son accesibles dentro de esa función.

Ejemplo de ámbito:

```
x = 10 # Variable global

def funcion():
    x = 5 # Variable Local
    print("Dentro de la función:", x)

funcion()
print("Fuera de la función:", x)
```

Salida:

Dentro de la función: 5
Fuera de la función: 10

Esto muestra que las variables locales y globales pueden tener el mismo nombre, pero se refieren a diferentes valores dependiendo del ámbito.

Variables globales

Son aquellas definidas fuera de cualquier función y pueden ser usadas dentro de las funciones sin necesidad de declararlas.

Ejemplo de acceso a una variable global dentro de una función:

```
mensaje = "Hola, mundo" # Variable global

def saludar():
    print(mensaje) # Se puede acceder sin problemas

saludar() # Output: Hola, mundo
```





Variables locales

Si asignamos un valor a una variable dentro de una función, **se creará una nueva variable local en lugar de modificar la global**.

Ejemplo de variable local:

```
mensaje = "Hola, mundo" # Variable global

def cambiar_mensaje():
    mensaje = "Hola desde la función" # Nueva variable local
    print(mensaje)

cambiar_mensaje() # Output: Hola desde la función
print(mensaje) # Output: Hola, mundo (la variable global no se modificó)
```

Nota: La variable mensaje dentro de cambiar_mensaje() es **local** y no afecta la variable global con el mismo nombre.

Modificar una variable global dentro de una función

Para modificar una variable global dentro de una función, debemos usar la palabra clave global.

Ejemplo usando global:

```
contador = 0 # Variable global

def incrementar():
    global contador # Permite modificar la variable global
    contador += 1

incrementar()
print(contador) # Output: 1
```

Importante:

- El uso de global debe hacerse con cuidado, ya que puede hacer que el código sea más difícil de mantener.
- Es preferible devolver valores desde funciones y asignarlos fuera de ellas en lugar de modificar variables globales.





7. Variables no definidas (Error NameError)

Si intentas usar una variable que no ha sido definida previamente en su ámbito correspondiente, obtendrás un error de tipo NameError.

Ejemplo de error NameError:

```
def mi_funcion():
    print(variable_no_definida) # NameError: name 'variable_no_definida'

mi_funcion()
```

8. Variables mutables e inmutables

En Python, los tipos de datos también se dividen en **mutables** (que pueden ser modificados después de su creación) e **inmutables** (que no pueden ser modificados).

- **Tipos mutables:** Listas (list), Diccionarios (dict), Conjuntos (set)
- **Tipos inmutables:** Cadenas de texto (str), Tuplas (tuple), Números (int, float, bool)

Ejemplo de variable mutable:

```
mi_lista = [1, 2, 3]
mi_lista.append(4) # Modificando la lista (mutable)
print(mi_lista) # Output: [1, 2, 3, 4]
```

Ejemplo de variable inmutable:

```
mi_numero = 10
mi_numero += 5 # Crear un nuevo número en lugar de modificar el anterior
print(mi_numero) # Output: 15
```

9. Asignación múltiple de variables

En Python, puedes asignar valores a varias variables en una sola línea. Esto es útil cuando necesitas inicializar varias variables a la vez.

Ejemplo de asignación múltiple:

```
a, b, c = 5, 10, 15
print(a, b, c) # Output: 5 10 15
```





10. Variables de clase y de instancia

Variables de clase en Python

Son compartidas por todas las instancias de la clase y se definen dentro de la clase, pero fuera de los métodos.

```
class Persona:
    especie = "Humano" # Variable de clase (compartida por todas las instancias)

    def __init__(self, nombre):
        self.nombre = nombre # Variable de instancia

p1 = Persona("Alice")
p2 = Persona("Bob")

print(p1.especie) # Humano
print(p2.especie) # Humano

# Modificar la variable de clase afecta a todas las instancias
Persona.especie = "Superhumano"
print(p1.especie) # Superhumano
print(p2.especie) # Superhumano
```

Variables de instancia en Python

Son únicas para cada objeto y se definen dentro del **constructor `__init__`**, usando `self`.

```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre # Variable de instancia

p1 = Persona("Alice")
p2 = Persona("Bob")

print(p1.nombre) # Alice
print(p2.nombre) # Bob

# Modificar una variable de instancia solo afecta a ese objeto
p1.nombre = "Charlie"
print(p1.nombre) # Charlie
print(p2.nombre) # Bob
```





Diferencias clave

Característica	Variable de Clase	Variable de Instancia
Se declara en	La clase (fuera de los métodos)	El método __init__
Se accede con	Clase.variable o self.variable	self.variable
Compartida entre objetos	<input checked="" type="checkbox"/> Sí	<input type="checkbox"/> No
Se modifica con	Clase.variable = valor	self.variable = valor

Nota: Tipos de variables: Constantes (Convención)

Python no tiene soporte nativo para constantes, pero se puede seguir una convención: las variables que no deben cambiar se escriben en mayúsculas para indicar que son constantes.

Ejemplo de constante:

```
PI = 3.14159 # Convención: constante
```

Resumen:

- **Ámbito:** Las variables pueden ser **locales** o **globales**, y su accesibilidad depende de dónde se definan.
- **Mutabilidad:** Diferenciar entre variables **mutables** e **inmutables** es clave para entender cómo se comportan los datos en Python.
- **Asignación múltiple:** Python facilita la asignación de varios valores a múltiples variables en una sola línea.



11. Operadores

Operadores Aritméticos (operaciones matemáticas)

Operador	Descripción	Ejemplo	Resultado
+	Suma	5 + 3	8
-	Resta	10 - 4	6
*	Multiplicación	6 * 2	12
/	División (resultado flotante)	7 / 2	3.5
//	División entera	7 // 2	3
%	Módulo (resto de la división)	10 % 3	1
**	Potencia (exponente)	2 ** 3	8

Operadores Relacionales (de comparación) (devuelven True o False)

Operador	Descripción	Ejemplo (a = 10, b = 3)	Resultado
==	Igual a	a == b	False
!=	Diferente de	a != b	True
>	Mayor que	a > b	True
<	Menor que	a < b	False
>=	Mayor o igual	a >= b	True
<=	Menor o igual	a <= b	False

Operadores de Asignación (modifican variables)

Operador	Equivalente a	Ejemplo (x = 5)
=	x = valor	x = 10
+=	x = x + valor	x += 2 → x = 7
-=	x = x - valor	x -= 3 → x = 2
*=	x = x * valor	x *= 4 → x = 20
/=	x = x / valor	x /= 2 → x = 2.5
//=	x = x // valor	x //= 2 → x = 2
%=	x = x % valor	x %= 2 → x = 1
**=	x = x ** valor	x **= 3 → x = 125



Operadores Lógicos

Operador	Descripción	Ejemplo	Resultado
and	Devuelve True si ambas condiciones son True	True and False	False
or	Devuelve True si al menos una condición es True	True or False	True
not	Invierte el valor lógico	not True	False

Operadores de Identidad (is, is not)

Verifican si dos variables hacen referencia al mismo objeto en memoria.

```
a = [1, 2, 3]
b = a
c = [1, 2, 3]

print(a is b)    # True (b apunta al mismo objeto que a)
print(a is c)    # False (c es una nueva Lista, aunque con los mismos valores)
print(a is not c) # True
```

Operadores de Pertenencia (in, not in)

Verifican si un valor está dentro de una secuencia (lista, cadena, etc.).

```
frutas = ["manzana", "pera", "uva"]

print("manzana" in frutas)    # True
print("banana" not in frutas) # True
```

Resumen

Python tiene los siguientes tipos de operadores:

- Aritméticos (+, -, *, /, //, %, **)
- Relacionales (==, !=, >, <, >=, <=)
- Lógicos (and, or, not)
- De asignación (=, +=, -=, etc.)
- De identidad (is, is not)
- De pertenencia (in, not in)

