

# 4. Introducción a PYTHON

## Introducción a Python

# Qué es Python y sus aplicaciones

Python es un lenguaje de programación de **alto nivel**, **interpretado** y de **propósito general**.

Se utiliza en una gran variedad de campos, como desarrollo web, análisis de datos, inteligencia artificial, automatización y ciencia.

Su principal característica es su sintaxis sencilla y legible, lo que lo convierte en una excelente opción tanto para principiantes como para profesionales.

## Conceptos básicos de Python

- **Alto nivel:** Python se parece más al lenguaje humano que al lenguaje que entiende el ordenador.
- **Interpretado:** Python no necesita compilarse antes de ejecutarse. En lugar de convertir todo el programa en código máquina de una vez, el intérprete lee y ejecuta el código línea por línea. Esto hace que sea más fácil probar, depurar y modificar el programa rápidamente.
- **Propósito general:** Se puede usar para casi cualquier tipo de aplicación, no está limitado a una tarea específica.

## Librerías en python

En Python, las librerías (también llamadas módulos o paquetes) son conjuntos de código reutilizable que amplían las capacidades del lenguaje. Gracias a ellas, puedes realizar tareas complejas —como trabajar con archivos, conectarte a internet o analizar datos— sin tener que escribir todo desde cero.

- Librerías estándar (Standard Library)
- Librerías externas o de terceros
- Librerías propias (módulos personalizados)

## Librerías en python -> Librerías estandar

Vienen incluidas con la instalación de Python. Contienen módulos muy útiles para tareas comunes como:

- os → interactuar con el sistema operativo
- math → operaciones matemáticas avanzadas
- datetime → manejar fechas y horas
- json → trabajar con datos en formato JSON
- random → generar números aleatorios

```
import math  
print(math.sqrt(25))
```

## Librerías en python → Librerías externas → gestor de paquetes pip

El gestor de paquetes pip es una herramienta incluida en Python que permite instalar, actualizar y desinstalar librerías externas (también llamadas paquetes). Gracias a pip, puedes acceder fácilmente a miles de librerías desarrolladas por la comunidad y disponibles en el Python Package Index (PyPI), el repositorio oficial de Python.

```
> pip install requests
```

```
# Instalar un paquete  
> pip install nombre_paquete
```

```
# Actualizar un paquete  
> pip install --upgrade nombre_paquete
```

```
# Desinstalar un paquete  
> pip uninstall nombre_paquete
```

```
# Ver que librerías tienes instaladas  
> pip list
```

```
# Instalar diferentes librerías de un archivo  
> pip install -r requirements.txt
```

## Librerías en python -> Librerías propias (módulos personalizados)

También puedes crear tus propias librerías (archivos.py) y reutilizarlas dentro de tu proyecto. Por ejemplo, si tienes un archivo utilidades.py con funciones, puedes importarlo en otro script con:

```
import utilidades
```

```
mi_proyecto/  
|  
├─ venv/           # Entorno virtual  
├─ main.py         # Programa principal  
├─ utilidades.py   # Librería creada por ti  
├─ requirements.txt # Lista de librerías instaladas  
└─ data/           # Carpeta con datos o archivos
```

## Introducción a Python

# Introducción al entorno virtual “**VENV**”

El entorno virtual de Python (o virtual environment) es una herramienta que permite crear un espacio aislado para cada proyecto, donde puedes instalar librerías y dependencias sin afectar al resto del sistema ni a otros proyectos.

## Crear y activar un entorno virtual de python

```
# Crear un entorno virtual
python -m venv venv

# Activar el entorno (en Windows)
venv\Scripts\activate

# Activar el entorno (en macOS/Linux)
source venv/bin/activate
```

## Desactivar un entorno virtual de python

```
deactivate
```

# 4.1. Sintaxis básica

## Primer programa Hola, mundo!

Un script es un archivo de texto con extensión .py que contiene instrucciones escritas en Python.

Cuando lo ejecutas, el intérprete lee el código línea por línea y realiza las acciones indicadas.

```
#holamundo.py
```

```
print("Hola, mundo!")
```

```
> python holamundo.py  
Hola, mundo!
```

## Primer script, estructura básica de un script

# indica un comentario, texto que no se ejecuta.

import sirve para usar librerías.

Las funciones agrupan código reutilizable.

El bloque `if __name__ == "__main__":` indica el punto de inicio del programa cuando se ejecuta directamente (y no desde otro módulo).

```
# 1. Comentarios y documentación
# Este programa muestra un mensaje en pantalla

# 2. Importación de librerías (si son necesarias)
import math

# 3. Definición de variables y funciones
def saludar():
    print("¡Bienvenido al curso de Python!")

# 4. Ejecución principal del programa
if __name__ == "__main__":
    saludar()
    print("Fin del programa.")
```

## 4.2. Variables

# Introducción a Python

## Variables

Una variable en Python es un nombre que se usa para almacenar un valor en la memoria. Puedes imaginarla como una caja con una etiqueta: el nombre de la variable es la etiqueta, y dentro de la caja está el valor que guarda.

Cuando creas una variable, Python reserva espacio en la memoria y asocia ese espacio con el nombre que tú eliges. Luego puedes usar ese nombre para acceder o modificar el valor.

## Crear una variable

En Python no necesitas declarar el tipo de variable (como en otros lenguajes). Simplemente se asigna un valor con el signo **=**

Deben empezar con una letra o guion bajo, nunca con un número.

👉 **Correcto:** edad, \_usuario

❌ **Incorrecto:** 2dato, -nombre

Pueden contener letras, números y guiones bajos (\_).

Son sensibles a mayúsculas y minúsculas:

Edad y edad son variables diferentes.

Usa nombres descriptivos y claros.

✅ total\_compra

❌ t

```
nombre = "Ana"
edad = 25
altura = 1.68
```

## Asignación de variables

Una de las características de python es la optimización de código, por ejemplo permitir declarar varias variables a la vez:

```
x, y, z = 1, 2, 3
```

```
a = b = c = 0
```

El principal valor de las variables es que pueden “variar” su contenido durante la ejecución del programa:

```
contador = 0  
contador = contador + 1
```

# Asignación de variables

Ejercicio:

teniendo las variables:

```
x = 3  
y = 1
```

Cual es la mejor manera de intercambiar los valores entre ambas variables

*#opción 1*

```
a = x  
x = y  
y = a
```

*#opción 2*

```
x = y  
y = x
```

*#opción 3*

```
x, y = y, x
```

## Tipos básicos de variables

**int** → números enteros → edad = 30

**float** → números decimales → precio = 19.99

**str** → texto → nombre = "María tiene un rosal"

**bool** → valores lógicos → activo = True // False

`type(variable)`

La función `type()` retorna el tipo de variable entre los anteriores

```
nombre = input("¿Cómo te llamas? ")
edad = int(input("¿Cuántos años tienes? "))
print("Hola", nombre, "tienes", edad, "años.")
```

## Introducción a Python

# Ejercicios Python 1

## Ejercicio 1

Prueba la función **input()** en el intérprete o en un script, ¿que hace? No lo sabes? Busca en la web.

- Ahora con esta función haz un script que te pida tu nombre y te salude.
- A continuación, haz un script que te pida tu nombre y te salude, un color y un objeto o lugar y genera una frase con las 3 variables.

## Ejercicio 2

Explica (en 3-4 frases) qué hace cada parte del siguiente código:

```
# Programa de bienvenida
def bienvenida(nombre):
    print(f"Hola, {nombre}, bienvenido al curso de Python!")

if __name__ == "__main__":
    usuario = input("Introduce tu nombre: ")
    bienvenida(usuario)
```

## Ejercicio 3 - Libreria math

Haz un script que ejecute estas acciones por pasos:

### 1.- Cálculo de raíz cuadrada:

Pide al usuario un número y muestra su raíz cuadrada usando la función `math.sqrt()`.

### 2.- Área de un círculo:

Solicita al usuario el radio de un círculo y calcula su área utilizando `math.pi` y `math.pow()`.

### 3.- Seno y coseno de un ángulo:

Pide al usuario un ángulo en grados y muestra su seno y coseno empleando `math.sin()` y `math.cos()`.

*(Recuerda convertir los grados a radianes con `math.radians()`.)*

## Ejercicio 4 - Libreria datetime

Haz un script que ejecute estas acciones por pasos:

### **Fecha y hora actual:**

Muestra la fecha y hora actuales con `datetime.datetime.now()`.

### **Cálculo de edad:**

Pide al usuario su año de nacimiento y calcula su edad actual usando `datetime.date.today().year`.

### **Días restantes del año:**

Calcula cuántos días faltan para que termine el año con `datetime.date.today()` y `datetime.date.replace()`.

## Ejercicio 5 - Libreria os

Haz un script que ejecute estas acciones por pasos:

### **Directorio actual:**

Muestra el directorio de trabajo actual con `os.getcwd()`.

### **Crear carpeta:**

Crea una nueva carpeta llamada "prueba\_python" utilizando `os.mkdir()`.

### **Listar archivos:**

Muestra todos los archivos y carpetas en el directorio actual con `os.listdir()`.

## Ejercicio 6 - Libreria random

Escoge uno de los siguientes ejercicios y haz un script con el programa.

### **Juego de adivinar el número:**

Genera un número aleatorio entre 1 y 10 con `random.randint()` y pide al usuario que lo adivine. Muestra un mensaje indicando si ha acertado o no.

### **Promedio de números aleatorios:**

Genera tres números aleatorios entre 1 y 100 usando `random.randint()`, muéstralos por pantalla y calcula su promedio.

### **Simulación de un dado:**

Crea un programa que muestre un número aleatorio del 1 al 6 cada vez que se ejecuta, usando `random.randint()`.

## Ejercicio 7 - Libreria time

Escoge uno de los siguientes ejercicios y haz un script con el programa.

### **Temporizador simple:**

Crea un programa que muestre “3... 2... 1...” con pausas de un segundo entre cada número usando `time.sleep()` y luego imprima “¡Listo!”.

### **Medir tiempo de escritura:**

Mide cuánto tarda el usuario en escribir su nombre.

Guarda el tiempo antes y después de la entrada con `time.time()` y muestra la diferencia.

## Ejercicio 7 - len

¿Qué imprimirá por terminal este código?

```
if __name__ == "__main__":  
    print("introduce una palabra o frase")  
    text = input()  
  
    print(len(text))
```

## 4.2. Operadores y estructuras de decisión

## Introducción a Python

# Operadores aritméticos

Los operadores aritméticos son símbolos que permiten realizar operaciones matemáticas con números o variables en Python. Son la base para cualquier cálculo dentro de un programa y funcionan igual que en matemáticas, con algunas particularidades de programación.

# Introducción a Python - Operadores aritméticos

Operador	Nombre	Función	Ejemplo	Resultado
+	Suma	Suma dos valores	5 + 3	8
-	Resta	Resta un valor a otro	10 - 4	6
*	Multiplicación	Multiplica dos valores	2 * 6	12
/	División	Divide y devuelve un número decimal	7 / 2	3.5
//	División entera	Divide y devuelve solo la parte entera	7 // 2	3
%	Módulo o resto	Devuelve el <b>resto</b> de una división	7 % 2	1
**	Potencia	Eleva un número a otro	2 ** 3	8

# Introducción a Python - Operadores aritméticos

```
a = 10
b = 3
print(a + b)
print(a // b)
print(a % b)
print(a ** b)
```

## Introducción a Python

# Operadores de comparación

Los operadores de comparación son símbolos que permiten comparar dos valores y obtener como resultado un booleano:

True (verdadero)

False (falso)

Se usan muchísimo en condicionales (if) y en combinación con operadores lógicos.

# Introducción a Python - Operadores de comparación

Operador	Significado	Ejemplo	Resultado
<code>==</code>	Igual a	<code>5 == 5</code>	<code>True</code>
<code>!=</code>	Distinto de	<code>5 != 3</code>	<code>True</code>
<code>&gt;</code>	Mayor que	<code>7 &gt; 4</code>	<code>True</code>
<code>&lt;</code>	Menor que	<code>3 &lt; 1</code>	<code>False</code>
<code>&gt;=</code>	Mayor o igual que	<code>6 &gt;= 6</code>	<code>True</code>
<code>&lt;=</code>	Menor o igual que	<code>2 &lt;= 5</code>	<code>True</code>

# Introducción a Python - Operadores de comparación

## Comprobación sencilla

```
edad = 20
print(edad >= 18)    # True
```

## Comparación encadenada

```
print(10 < 20 < 30)    # True
```

## Comparando valores

```
a = 4
b = 7
```

```
print(a == b)    # False
print(a != b)    # True
print(a < b)     # True
```

## Introducción a Python - Operadores de comparación

Permiten evaluar condiciones que el programa usará para decidir qué hacer.

```
nota = 7
```

```
if nota >= 5:  
    print("Aprobado")  
else:  
    print("Suspendido")
```

## Introducción a Python - Operadores de comparación

¿Qué imprimirá por terminal si el numero es....? (9,10,11)

```
if __name__ == "__main__":  
    print("introduce un número")  
    numero = input()  
  
    print(numero == 10)  
    print(numero == "10")  
    print(numero > 10)  
    print(numero >= 10)  
    print(numero < 10)  
    print(numero <= 10)
```

## Introducción a Python - Operadores de comparación

¿Qué imprimirá por terminal si el numero es....? (9&10,10&10,10&11)

```
if __name__ == "__main__":  
    print("introduce un número")  
    numero1 = input()  
    print("introduce otro número")  
    numero2 = input()  
  
    print(numero1 == numero2)  
    print(numero1 > numero2)  
    print(numero1 >= numero2)  
    print(numero1 < numero2)  
    print(numero1 <= numero2)
```

## Introducción a Python

# Operadores lógicos

Los operadores lógicos son operadores que permiten combinar condiciones y/o trabajar con valores booleanos (True y False).

Se usan muchísimo junto con condicionales (if, elif, else) para tomar decisiones más complejas en un programa.

En Python existen tres operadores lógicos principales: **and**, **or** y **not**

## Operador lógico AND (Y lógico)

Devuelve **True** solo si se cumplen ambas condiciones.

*“Saldré en bicicleta si hace sol y tengo tiempo”*

Ejemplos:

```
(5 > 3) and (10 > 2)    # True  
(5 > 3) and (10 < 2)    # False
```

$(a > b)$  and  $(a > c)$  and  $(b > c)$  and...

## Operador lógico OR (O lógico)

Devuelve **True** si al menos una de las condiciones es verdadera.

*“Saldré en bicicleta llueva o haga sol”*

Ejemplos:

```
(5 > 10) or (10 > 2)    # True  
(5 > 10) or (10 < 2)    # False
```

## Operador lógico NOT (NO lógico)

Invierte el valor lógico:

not True → False

not False → True

***“NO** saldré en bicicleta”*

Ejemplos:

**not** (5 > 3)      # False

**not** False        # True