

**PYTHON** 

**TIC II 2º BACHILLERATO**

por Alberto Durán Pérez

# INTRODUCCIÓN



- Características Clave

-  **Filosofía:** Sencillez (Zen de Python).
-  **Ejecución:** Interpretado (línea a línea).
-  **Entorno:** Multiplataforma.

- ¿Para qué se usa?

-  Inteligencia Artificial (IA).
-  Ciencia de Datos.
-  Desarrollo Web.

- Fundamentos Técnicos

-  **Variables:** Cajas para guardar datos.
-  **Reglas:** Sintaxis estricta y limpia.

# VARIABLES EN PYTHON

## REGLAS, TIPOS Y BUENAS PRÁCTICAS

Fundamentos de Programación

# 1. ¿QUÉ ES UNA VARIABLE?

Es un lugar en la memoria para **almacenar un valor**.

- **Etiqueta:** El nombre ( myNum )
- **Contenido:** El valor ( 4 )

```
mynum = 4  
cliente = "Mauricio" # Texto entre comillas
```

**Nota:** La variable guarda información para consultarla o cambiarla después.



## 2. VALORES CAMBIANTES

Como un marcador de baloncesto 🏀, el contenido puede cambiar, pero el nombre de la variable se mantiene.





```
puntuacion = 86
print(puntuacion) # Muestra 86

puntuacion = 88    # Cambiamos el valor
print(puntuacion) # Ahora muestra 88
```

La función `print()`

Sirve para "enseñar" el valor actual al usuario por pantalla.

# 3. REGLAS DE NOMBRAMIENTO

1. **Inicio:** Debe empezar por **Letra** o **guion bajo** ( `_` ).
  -  `variable` , `_secreta`
  -  `1variable` (Nunca números al principio)
2. **Caracteres:** Solo letras, números y `_` .
  -  Nada de espacios, `@` , `-` , o tildes.
3. **Reservadas:** Prohibido usar palabras propias de Python.
  -  `if` , `for` , `while` , `class` .
4. **Case Sensitive:** Las mayúsculas importan.
  - `nombre`  $\neq$  `Nombre`

## 4. CONVENCIONES DE ESTILO 🎨

Estilo	Formato	Uso Principal
snake_case 🐍	mi_variable_total	Variables y Funciones
camelCase 🐪	miVariableTotal	Otros lenguajes / Librerías
PascalCase 🧑🔬	MiVariableTotal	Clases (Avanzado)
UPPERCASE 📢	GRAVEDAD	<b>Constantes</b> (No cambian)

# TIPOS DE DATOS (DATA TYPES)

¿QUÉ GUARDA LA CAJA?



# 1. NÚMEROS: ENTEROS Y DECIMALES

En Python, diferenciamos los números "completos" de los que tienen "coma".

Tipo	Nombre Técnico	Descripción	Ejemplo
Entero ●	int (Integer)	Sin decimales (+ o -)	5 , -3 , 42
Decimal ●	float (Float)	Con punto decimal	3.14 , -0.3

## ⚡ El Truco del Incremento

Aumentar un contador es muy común. Python tiene un atajo:

```
contador = 2
contador += 1 # Es igual a: contador = contador + 1
print(contador) # Resultado: 3
```

## 2. TEXTO

### Cadenas ( str )

Secuencias de caracteres entre comillas ( " o ' ).

- **Concatenar:** Unir textos con + .

```
nombre = "Elena"  
print("Hola " + nombre)
```

### 3. BOOLEANOS ( BOOL )

Solo tienen dos valores posibles (Lógica binaria).

- True (Verdadero)
- False (Falso)

¡Ojo! La primera letra siempre va en Mayúscula.

# 3. CONVERSIÓN DE TIPOS (CASTING)

A veces necesitamos transformar un dato. Python usa funciones con el nombre del tipo destino.

## Las Funciones Mágicas:

- `int()` : Convierte a entero (trunca decimales).
- `str()` : Convierte a texto (para imprimir mensajes).
- `float()` : Convierte a decimal.

```
# Ejemplo: De texto a número para sumar
edad_texto = "17"
edad_numero = int(edad_texto)
print(edad_numero + 1) # Resultado: 18 (¡Funciona!)
```

## 4. EL DETECTIVE TYPE()

¿No sabes qué hay en una variable? Pregúntale a Python.

```
x = 3.14  
print(type(x)) # <class 'float'>
```

### **Pregunta para pensar**

¿Qué pasa si intentas esto?

```
print("Tengo " + 15 + " años")
```

**Respuesta:**  **ERROR.**

Python no suma peras (texto) con manzanas (números).

**Solución:** `print("Tengo " + str(15) + " años")`

# RESUMEN RÁPIDO

Tipo	Función Python	Ejemplo Real
Entero	<code>int()</code>	Edad, Cantidad de hijos
Decimal	<code>float()</code>	Precio, Altura, Peso
Texto	<code>str()</code>	Nombre, Dirección
Lógico	<code>bool()</code>	¿Aprobado?, ¿Usuario activo?

**Recuerda:** Usa nombres de variables descriptivos ( `precio` , no `x` ).

# OPERACIONES MATEMÁTICAS

## ARITMÉTICA, MATH Y ESTADÍSTICA

Python para Ciencia de Datos

# 1. OPERADORES BÁSICOS

Python funciona como una calculadora potente.

Símbolo	Operación	Ejemplo	Resultado
+	Suma	$10 + 5$	15
-	Resta	$10 - 5$	5
*	Multiplicación	$10 * 5$	50
/	División	$10 / 3$	3.333...
//	División Entera	$10 // 3$	3 (Trunca)
%	Módulo (Resto)	$10 \% 3$	1
**	Potencia	$2 ** 3$	8



## 2. PRESENTAR RESULTADOS

No basta con calcular, hay que mostrarlo bien. La forma moderna es usar **f-strings**.

```
a = 10  
b = 5  
resultado = a + b
```

### FORMA ANTIGUA (POCO CLARA)

```
print("La suma de", a, "y", b, "es:", resultado)
```

### FORMA PRO (F-STRINGS)

```
print(f"La suma de {a} y {b} es: {resultado}")
```

# LA FUNCIÓN PRINT()

LA VOZ DE TU PROGRAMA

# PRINT()

- Es una función **incorporada** (siempre disponible).
- Se utiliza para **mostrar información** al usuario en la consola.

## Uso Básico

```
# Imprimir texto fijo
print("¡Hola, Mundo!")

# Imprimir variables y texto combinado (usa comas)
x = 10
print("El valor de x es", x)
```

**Recuerda:** La coma ( , ) añade un espacio automáticamente entre los argumentos.

# F-STRINGS: LA FORMA MODERNA ✨

Los `f-strings` (cadenas con formato) son la forma más legible y potente de combinar variables y texto.

- Se nombran por la letra `f` antes de la cadena ( `f"""` ).
- Permiten introducir variables o expresiones usando llaves `{}` .

## Ejemplo Básico

```
nombre = "Alberto"  
edad = 35
```

```
print(f"Hola, me llamo {nombre} y tengo {edad} años.")  
# Resultado: Hola, me llamo Alberto y tengo 35 años.
```

# F-STRINGS: PODER AVANZADO (I)

Los `f-strings` no solo insertan variables; también pueden ejecutar **código** o aplicar **formato** dentro de las llaves `{}` .

## 1. Operaciones Directas

```
x = 5
```

```
y = 3
```

```
# Ejecuta x + y DENTRO de la cadena.
```

```
print(f"La suma de {x} y {y} es {x + y}")
```

# F-STRINGS: PODER AVANZADO (II)

## 2. Formato Decimal

Útil para controlar la precisión de los números flotantes.

```
div = 124 / 45  # Resultado: 2.7555555...
```

```
# Sin formato
```

```
print(f"El valor es {div}")
```

```
# Con formato: {:.4f} muestra 4 decimales
```

```
print(f"El valor es {div:.4f}")
```

# CONTROLANDO LA PRESENTACIÓN

## ⬇ Saltos de Línea

El carácter especial `\n` (newline) fuerza un salto de línea.

```
# Imprime en tres líneas separadas
print("Línea 1 \nLínea 2 \nLínea 3")
```

## ➡ Parámetro `sep` (Separador)

Controla el texto que aparece **entre** los argumentos separados por comas.

```
# Sin sep: 'Alberto Sara Ana'
# Con sep: 'Alberto-Sara-Ana'
print("Alberto", "Sara", "Ana", sep="-")
```

# EL PARÁMETRO END

Por defecto, `print()` termina la línea con un salto de línea ( `\n` ). El parámetro `end` te permite cambiar esto:

## Unir Líneas

```
# Muestra "Hola " (nota el espacio al final)
```

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

```
# Muestra "Mundo" en la MISMA línea
```

```
print("Mundo")
```

```
# Salida: Hola Mundo
```



## Imprimir Colecciones

`print()` funciona directamente con listas, tuplas y diccionarios, mostrándolos en un formato legible.

```
frutas = ["manzana", "banana", "naranja"]  
print("Stock de frutas:", frutas)
```

