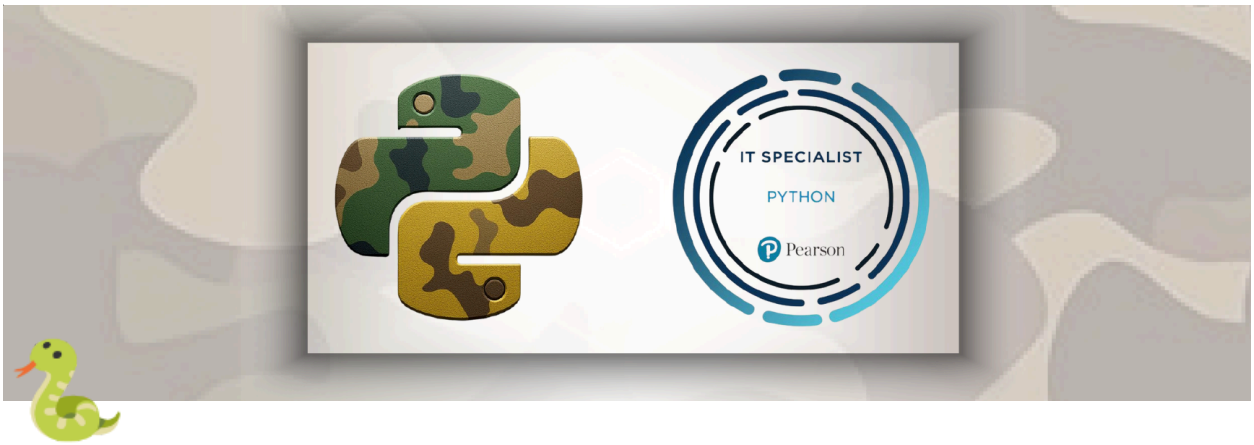

🐍 Clase 05 - Operadores Lógicos en Python

🔄 1. Conversión con bool()

Ejemplo 1.1: Comportamiento básico de bool()

```
# bool() convierte strings a True/False
respuesta = input("¿Te gusta Python? (escribe algo): ")
es_verdadero = bool(respuesta)

print(f"Tu respuesta: '{respuesta}'")
print(f"Convertido a bool: {es_verdadero}")
```

> **Comportamiento de bool() con strings:**
>
> - **String vacío** `""` → `False`
> - **Cualquier string con contenido** → `True`

Ejemplo 2.2: Conversión de números a booleano

```
numero = input("Ingresa un número: ")
numero_entero = int(numero)
es_verdadero = bool(numero_entero)

print(f"Número: {numero_entero}")
print(f"Como booleano: {es_verdadero}")
```

> **Comportamiento de bool() con números:**
>
> - `0` → `False`
> - **Cualquier otro número** (positivo o negativo) → `True`

💡 🔍 Puntos Clave sobre bool():

1. **String vacío** = `False`
2. **Cualquier string con contenido** = `True` (incluso `"0"` o `"False"`)
3. **Número 0** = `False`
4. **Cualquier otro número** = `True`
5. **Es útil para validar si el usuario ingresó algo**

🧠 2. Ejercicios para añadir a la practica 5

Ejercicio 2.1: Convertir número a booleano.

Escribe un programa que pida al usuario un número entero y un decimal luego que use bool() para convertirlos a booleano. Muestra los números ingresados y su valor booleano.

Ejercicio 2.2: Verificador de Datos

Crea un programa que solicite diferentes tipos de información al usuario y muestre cómo se comporta la conversión `bool()` con cada entrada.

Instrucciones:

Solicita al usuario que ingrese su **nombre** (puede dejarlo vacío)

Solicita que ingrese su **edad** como número

Solicita que ingrese un **comentario** sobre Python (puede dejarlo vacío)

Para cada entrada, muestra:

- El valor original que ingresó
- El resultado de convertirlo con `bool()`
- Un mensaje explicativo sobre el resultado

Ejemplo de salida esperada:

```
Ingresa tu nombre (puedes dejarlo vacío): María
Ingresa tu edad: 25
Escribe un comentario sobre Python (puedes dejarlo vacío):

=== RESULTADOS DE CONVERSIÓN bool() ===
Nombre: 'María' → True
Edad: 25 → True
Comentario: '' → False

=== EXPLICACIÓN ===
Tu nombre tiene contenido
Tu edad 25 es diferente de cero
Tu comentario está vacío
```

🧩 3. Operadores Lógicos: Definiciones

> Los **operadores lógicos** permiten combinar o evaluar expresiones booleanas (verdaderas o falsas). En Python, los tres operadores lógicos principales son:

Operador	Significado	Ejemplo	Resultado
`and`	Devuelve `True` si **ambas** condiciones son verdaderas	`True and True`	`True`
`or`	Devuelve `True` si **al menos una** condición es verdadera	`True or False`	`True`
`not`	Invierte el valor lógico	`not True`	`False`

💡 🧠 **Importante:** Python evalúa las expresiones de izquierda a derecha y puede usar "short-circuit" (deja de evaluar cuando ya sabe el resultado).

🧮 3. Ejercicios básicos de comprensión de conceptos

🧩 Ejercicio 1: Evaluación directa

Escribe y ejecuta las siguientes líneas una por una, y analiza el resultado:

```
print(True and True)
print(True and False)
print(False or True)
print(False or False)
print(not True)
print(not False)
```

> 👉 `and` necesita que **ambos valores sean verdaderos**.
> 👉 `or` necesita **al menos uno verdadero**.
> 👉 `not` cambia el valor lógico.

🧩 Ejercicio 2: Combinando expresiones con números

Usaremos operadores relacionales y lógicos juntos.

```
print(5 > 3 and 2 < 4)
print(10 == 10 or 5 != 5)
print(not (7 <= 2))
```

> 👉 `5 > 3` es `True`, `2 < 4` es `True`, → `True and True` = `True`.
> 👉 `10 == 10` es `True`, `5 != 5` es `False`, → `True or False` = `True`.
> 👉 `7 <= 2` es `False`, → `not False` = `True`.

🧩 Ejercicio 3: Lógica con cadenas y listas

```
print("hola" != "adiós" and len("hola") == 4)
print(len([1,2,3]) > 2 or len([]) > 0)
print(not len([]))
```

> 👉 Se pueden comparar cadenas y listas.
> 👉 `len()` devuelve un número, que puede usarse dentro de expresiones lógicas.

⚙️ 4. Ejercicios “intermedios” resueltos

💡 Ejercicio 1: Validar entrada del usuario

Pide un número y evalúa si está entre 1 y 10 (sin usar `if`).

```
num = int(input("Ingresa un número entre 1 y 10: "))
print("¿Está entre 1 y 10?", num >= 1 and num <= 10)
```

> 👉 La expresión `num >= 1 and num <= 10` devuelve un valor booleano que indica si el número cumple la condición.

💡 Ejercicio 2: Comparación múltiple

Verifica si un valor **NO** está fuera de rango.

```
edad = int(input("Ingresa tu edad: "))
print("¿Edad válida para registrarse?", not (edad < 18 or eda
d > 65))
```

> 👉 Si `edad < 18` o `edad > 65` es `True`, significa que **no** está en el rango válido.
> 👉 `not` invierte el resultado: edad válida → `True`.

🧠 Ejercicio 3: 🔍 Evaluador de requisitos para una app

> 💡 Queremos verificar si un usuario **puede acceder a una app**:
>
> - Debe tener **al menos 18 años**.
> - Su **país** debe ser “México” o “Colombia”.
> - Debe haber **aceptado los términos** (`True` o `False`).

```
edad = int(input("Edad: "))
pais = input("País: ")
acepta = input("¿Aceptas los términos? (True/False): ") == "T
rue"

puede_acceder = (edad >= 18) and (pais == "México" or pais ==
"Colombia") and acepta

print("¿Puede acceder a la app?", puede_acceder)
```

> 👉 `(edad >= 18)` → verifica mayoría de edad.
> 👉 `(pais == "México" or pais == "Colombia")` → valida país permitido.
> 👉 `and acepta` → requiere haber aceptado términos.
> El resultado final es `True` solo si **todas** las condiciones son verdaderas.

🧩5. Precedencia y selección de operadores en Python

🧩 1. Definiciones y conceptos clave

🔹 Precedencia de operadores

> La **precedencia** determina **en qué orden** se evalúan las operaciones dentro de una expresión.

Ejemplo:

```
resultado = 3 + 2 * 5
```

> Aunque `+` aparece antes, Python evalúa primero `*` porque tiene **mayor precedencia**.

🔹 Asociatividad

> La **asociatividad** define **la dirección de evaluación** cuando dos operadores tienen la misma prioridad.

```
print(2 ** 3 ** 2)
```

> 👉 Los operadores de potencia (`**`) se evalúan **de derecha a izquierda**, por lo tanto:
>
> `2 ** (3 ** 2)` → `512`.

🔹 Operadores de identidad (`is`, `is not`)

> Permiten verificar si **dos variables apuntan al mismo objeto en memoria**, no solo si tienen el mismo valor.

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

print(a is b)     # True, ambos apuntan al mismo objeto
print(a is c)     # False, tienen el mismo contenido pero son
objetos distintos
print(a is not c) # True
```

> 📌 **Importante:**
>
> - `==` compara **valores**.
> - `is` compara **identidad de objeto**.

🔹 Operadores de pertenencia (`in`, `not in`)

> Verifican si un elemento **existe dentro** de una secuencia (lista, cadena, tupla...).

```
letras = ["a","b","c"]
print("a" in letras)       # True
print("z" not in letras)   # True
```

También funcionan con cadenas:

```
print("py" in "python")      # True
```

🧮 5.2 Tabla de precedencia (simplificada)

De **mayor a menor prioridad**:

Prioridad	Tipo de operador	Ejemplo
1	Paréntesis	`( )`
2	Potencia	`**`

Prioridad	Tipo de operador	Ejemplo
3	Signo unario	`+x`, `-x`, `not x`
4	Multiplicativos	`*`, `/`, `//`, `%`
5	Aditivos	`+`, `-`
6	Comparaciones	`<`, `<=`, `>`, `>=`, `==`, `!=`
7	Identidad y pertenencia	`is`, `is not`, `in`, `not in`
8	Lógicos	`and`, `or`, `not`
9	Asignación	`=`, `+=`, `-=` ...

🧩 5.3 Ejercicios básicos de comprensión

🧠 Ejercicio 1: Orden de evaluación

```
print(3 + 2 * 5)
print((3 + 2) * 5)
print(10 - 2 ** 2)
```

> 👉Los operadores de potencia y multiplicación tienen prioridad sobre `+` y `-`.
> 👉Los paréntesis siempre cambian el orden de ejecución.

🧠 Ejercicio 2: Potencia y unarios

```
print(-2 ** 3)
print((-2) ** 3)
```

🧠 Ejercicio 3: Lógicos y relacionales

```
print(3 + 2 > 4 and not False)
print(3 + 2 > 4 or False)
```

🧠 Ejercicio 4: Identidad

```
x = [1,2]
y = x
z = [1,2]

print(x == z)    # True (mismo contenido)
print(x is z)     # False (objetos distintos)
print(x is y)    # True (misma referencia)
```

🧠 Ejercicio 5: Pertenencia

```
nombres = ["Ana","Luis","Carlos"]
print("Ana" in nombres)
print("Pedro" not in nombres)
print("a" in "programar")
```

⚙️ 5.4 Ejercicios intermedios resueltos

💡 Ejercicio 1: Comparación y pertenencia

Evalúa esta expresión y explica el orden:

```
print(3 * 2 + 5 in [5,6,11] or not False)
```

> **Paso a paso:**
>
> 1. `3 * 2` → `6`
> 2. `6 + 5` → `11`
> 3. `11 in [5, 6, 11]` → `True`
> 4. `True or not False` → `True or True` → `True`
>
> ✅ Resultado: `True`

💡 Ejercicio 2: Identidad con listas

```
lista1 = [10,20]
lista2 = [10,20]
lista3 = lista1

print(lista1 == lista2)   # True (valores iguales)
print(lista1 is lista2)   # False (objetos diferentes)
print(lista1 is lista3)   # True (misma referencia)
```

💡 Ejercicio 3: Evaluador de condiciones mixtas

Queremos saber si un valor está **dentro de una lista** y si esa lista es **la misma** que otra variable:

```
frutas = ["manzana","pera","uva"]
seleccion = frutas
busqueda = input("Ingresa una fruta: ")

resultado = (busqueda in frutas) and (seleccion is frutas)
print("¿Fruta encontrada en la lista original?", resultado)
```

> 👉 `busqueda in frutas` → evalúa pertenencia.
> 👉 `seleccion is frutas` → evalúa identidad.
> 👉 Ambas deben ser verdaderas (`and`).
