Guía 3: Control de flujo

Profesor: John Serrano - john.serrano@usach.cl

1 Operadores de comparación

En Python, los operadores de comparación se utilizan para comparar dos valores y devolver un valor booleano (True o False).

1.0.1 Igualdad (==)

La igualdad compara dos elementos y verifica si son iguales. Hay que tener un poco de cuidado, ya que hay algunos casos donde a pesar del tipo de dato, Python retornará True en una comparación (Por ejemplo, comparan un número int con su contraparte float con parte decimal 0)

[1]:
```
3 == 3
```

[1]:
```
True
```

[2]:
```
3 == 4
```

[2]:
```
False
```

[3]:
```
3 == 3.0
```

[3]:
```
True
```

[4]:
```
0 == -0
```

[4]:
```
True
```

[5]:
```
# Notar que el segundo string tiene un espacio extra
"esto es un string" == "esto es un  string"
```

[5]:
```
False
```

[6]:
```
True == 1
```

[6]:
```
True
```

[7]:
```
True == 0
```

[7]:
```
False
```

1.0.2 Desigualdad (!=)

La desigualdad verifica si dos valores son diferentes. Cuidado al escribir el simbolo, ya que si se escribe al revés (=!), Python dará un error.

[8]:
```
5 != 3
```

[8]: True

[9]:
```
3 != 3
```

[9]: False

[10]:
```
3.0 != 3
```

[10]: False

[11]:
```
"Fundamentos de programación" != "Fundamentos de Computación y Programación"
```

[11]: True

1.0.3 Mayor que y menor que ($>$ o $<$)

Verifica si el valor de la izquierda es mayor o menor que el de la derecha.

[12]:
```
3 > 10
```

[12]: False

[13]:
```
3.0 > 3
```

[13]: False

[14]:
```
0 < 100
```

[14]: True

[15]:
```
# El resultado se debe al código ASCII que se ha mencionado durante clases
"A" < "a"
```

[15]: True

[16]:
```
# En casos como estos, primero se compara si h < m, lo cual es True. Dado eso, se obtiene True en la comparación.
"hola" < "mundo"
```

[16]: True

[17]:
```
"h" < "m"
```

[17]: True

[18]:
```
# Recordar que True = 1 y False = 0.
True < False
```

[18]: False

1.0.4 Mayor o igual que o menor o igual que (>= o <=)

Verifica si el valor de la izquierda es mayor o igual o menor o igual que el de la derecha. El símbolo de igual se escribe directamente después del símbolo de mayor o menor, ya que es imposible escribir algo abajo del símbolo de mayor o menor como lo hacemos en nuestro lenguaje natural.

[19]:
```
3 >= 3.0
```

[19]: True

[20]:
```
"hola" >= "hello"
```

[20]: True

[21]:
```
True >= 3
```

[21]: False

[22]:
```
10.00000000000 <= 10.0
```

[22]: True

2 Operadores booleanos

En Python, los operadores booleanos se utilizan para combinar o invertir expresiones booleanas (que son True o False). Debemos recordar las tablas de verdad vistas en Álgebra I.

NOTA: Debemos recordar que el True de booleanos es equivalente a cualquier elemento que no sea 0 y sus representaciones. Lo mismo con False, es equivalente a 0 y todas sus representaciones (0, 0.0, 0+0j)

2.0.1 And (y lógico)

Devuelve True si ambas expresiones son True. De lo contrario, devuelve False.

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

[23]:
```
False and False
```

[23]: False

[24]:
```
False and True
```

[24]: False

[25]:
```
True and False
```

[25]: False

[26]:
```
True and True
```

[26]: True

[27]:
```
2 and 1
```

[27]: 1

[28]:
```
1 and 0
```

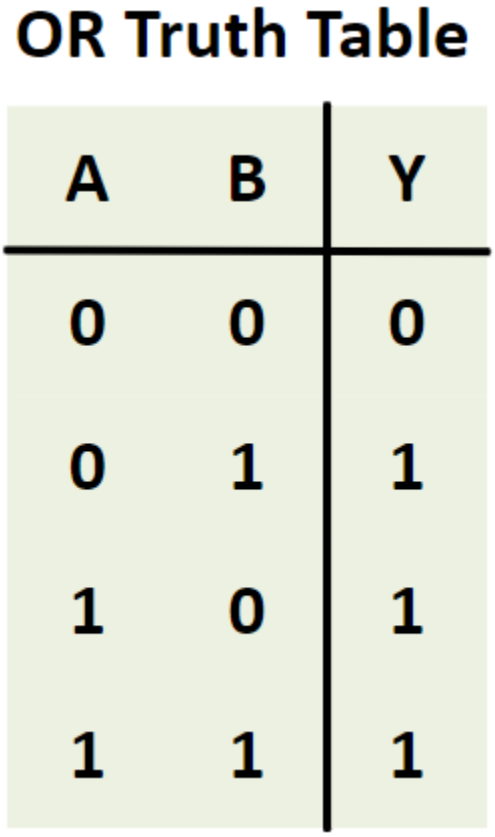
[28]: 0

[29]:
```
0 and True
```

[29]: 0

2.0.2 Or (o lógico)

Devuelve True si al menos una de las expresiones es True. Solo devuelve False si ambas son False.



[1]:
```
False or False
```

[1]: False

[2]:
```
False or True
```

[2]: True

[3]:
```
True or False
```

[3]: True

[5]:
```
True or True
```

[5]: True

```
[6]: 3 or 0
```

[6]: 3

```
[7]: 1 or 0
```

[7]: 1

```
[8]: 0 or 0
```

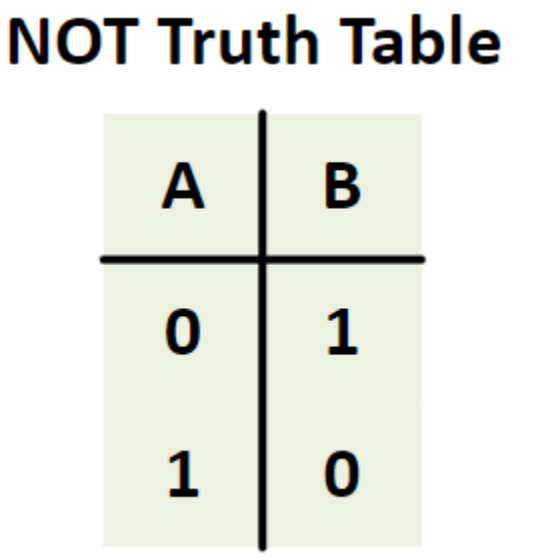
[8]: 0

```
[9]: 0 or True
```

[9]: True

2.0.3 Not (negación lógica)

Invierte el valor de la expresión booleana. Si es True, lo convierte en False, y viceversa.




```
[30]: not(False)
```

[30]: True

```
[31]: not(True)
```

[31]: False

```
[32]: not(1)
```

[32]:
```
False
```

[33]:
```
not(2)
```

[33]:
```
False
```

[34]:
```
not(0)
```

[34]:
```
True
```

[35]:
```
not("0")
```

[35]:
```
False
```

3 Decisiones (if, elif, else)

En Python, las estructuras de control condicionales como **if, elif y else** se utilizan para tomar decisiones en el flujo de un programa. Estas estructuras permiten que el código ejecute diferentes bloques de instrucciones en función de si una o más condiciones son verdaderas o falsas.

La declaración **if** evalúa una condición booleana (que puede ser True o False). Si la condición es True, el bloque de código dentro del if se ejecuta.

[36]:
```
# Procesamiento
if 2 > 0:
    print("El número 2 es mayor que 0")

# Salida
print("Esto va afuera, se imprime igual")
```

```
El número 2 es mayor que 0
Esto va afuera, se imprime igual
```

Se pueden tener múltiples if dentro de un if. El espacio que está antes del primer print es conocido como **indentación**, que es equivalente a 4 espacios. Esto simboliza que la instrucción que se está ejecutando está dentro de un bloque de if en el cual se cumplió la condición inicial (2 es mayor que 0). Todo lo que no está indentado se ejecuta igualmente ya que está fuera de la condición del if. A continuación, un ejemplo con más detalle:

[38]:
```
# Entrada
numero = int(input("Ingrese un numero entero: "))

# Procesamiento
if numero > 10:
    print("El numero es mayor que 10")
    if numero > 100:
        print("El numero es mayor que 100")
        if numero > 1000:
            print("El numero es mayor que 1000")
```

```
# Salida
print("Este print se ejecuta igual")
```

```
Ingrese un numero entero:  159

El numero es mayor que 10
El numero es mayor que 100
Este print se ejecuta igual
```

elif (abreviatura de "else if") se utiliza para verificar múltiples condiciones. Después de la primera condición if, se pueden usar uno o más bloques elif para comprobar otras condiciones.

[39]:
```
# Entrada
numero = int(input("Ingrese un numero entero: "))

# Procesamiento y salida
if numero == 0:
    print("El numero es 0")
elif numero == 1:
    print("El numero es 1")
elif numero > 1:
    print("El numero es mayor que 1")
```

```
Ingrese un numero entero:  3

El numero es mayor que 1
```

Pero, para el ejemplo anterior, ¿qué pasa con los números negativos? Al no haber ninguna condición asociada, no se imprimiria nada. En realidad, en nuestro código faltaría esta condición. Es ahí donde entra **else**, el cual se utiliza para definir un bloque de código que se ejecutará cuando ninguna de las condiciones anteriores sea verdadera. No necesita una condición porque actúa como el caso por defecto.

[40]:
```
# Entrada
numero = int(input("Ingrese un numero entero: "))

# Procesamiento y salida
if numero == 0:
    print("El numero es 0")
elif numero == 1:
    print("El numero es 1")
elif numero > 1:
    print("El numero es mayor que 1")
else:
    print("El numero no es igual a 0, a 1 o mayor que 1")
```

```
Ingrese un numero entero:  -3

El numero no es igual a 0, a 1 o mayor que 1
```

Ahora, **¿qué pasa si tenemos multiples ifs?** Podríamos considerar que cada if, elif y else es un bloque de código, por lo que cada vez que haya un nuevo if se debe verificar la condición, independiente si en el otro if se cumplió la condición de elif o se ejecutó el else. Por ejemplo:

[41]:
```
# Entrada
numero = int(input("Ingrese un numero entero: "))

# Procesamiento y salida
if numero > 0:
    print("El numero es mayor que 0")

if numero == "2":
    print("El numero es 2")
else:
    print("El numero no es 2")
```

```
Ingrese un numero entero:  3

El numero es mayor que 0
El numero no es 2
```

Dentro de un bloque de if no solo podemos tener prints, si no también cálculos, otros ifs, etc. Lo más importante es respetar el orden de los bloques if.