Operadores lógicos

Los operadores lógicos o *logical operators* nos **permiten trabajar con valores de tipo booleano**. Un valor booleano o `bool` es un tipo que solo puede tomar valores `True` o `False`. Por lo tanto, estos operadores nos permiten realizar diferentes operaciones con estos tipos, y su resultado será otro booleano. Por ejemplo, `True and True` usa el operador `and`, y su resultado será `True`. A continuación lo explicaremos mas en detalle.

Operador	Nombre	Ejemplo
and	Devuelve True si ambos elementos son True	True and True = True
or	Devuelve True si al menos un elemento es True	True or False = True
not	Devuelve el contrario, True si es Falso y viceversa	not True = False

Operador and

El operador `and` evalúa si el valor a la izquierda **y** el de la derecha son `True`, y en el caso de ser cierto, devuelve `True`. Si uno de los dos valores es `False`, el resultado será `False`. Es realmente un operador muy lógico e intuitivo que incluso usamos en la vida real. Si hace sol `y` es fin de semana, iré a la playa. Si ambas condiciones se cumplen, es decir que la variable `haceSol=True` y la variable `finDeSemana=True`, iré a la playa, o visto de otra forma `irALaPlaya=(haceSol and finDeSemana)`.

```
print(True and True)   # True
print(True and False)  # False
print(False and True)  # False
print(False and False) # False
```

Operador or

El operador `or` devuelve `True` cuando al menos uno de los elementos es igual a `True`. Es decir, evalúa si el valor a la izquierda **o** el de la derecha son `True`.

```
print(True or True)   # True
print(True or False)  # True
print(False or True)  # True
print(False or False) # False
```

Es importante notar que varios operadores pueden ser usados conjuntamente, y salvo que existan paréntesis que indiquen una cierta prioridad, el primer operador que se evaluará será el `and`. En el ejemplo que se muestra a continuación, podemos ver que tenemos dos operadores `and`. Para calcular el resultado final, tenemos que empezar por el `and` calculando el resultado en grupos de dos y arrastrando el resultado hacia el siguiente grupo. El resultado del primer grupo sería `True` ya que estamos evaluando `True and True`. Después, nos guardamos ese `True` y vamos a por el siguiente y último elemento, que es `False`. Por lo tanto hacemos `True and False` por lo que el resultado final es `False`

```
print(True and True and False)
#     |-----------|
#           True  and  False
#         |------------------|
#                False
```

También podemos mezclar los operadores. En el siguiente ejemplo empezamos por `False and True` que es `False`, después `False or True` que es `True` y por último `True or False` que es `True`. Es decir, el resultado final de toda esa expresión es `True`.

```
print(False and True or True or False)
#     False and True = False
#               Fase or True = True
#                       True or False = True
# True
```

Operador not

Y por último tenemos el operador `not`, que simplemente invierte `True` por `False` y `False` por `True`. También puedes usar varios `not` juntos y simplemente se irán aplicando uno tras otro. La verdad que es algo difícil de ver en la realidad, pero simplemente puedes contar el número de `not` y si es par el valor se quedará igual. Si por lo contrario es impar, el valor se invertirá.

```
print(not True)  # False
print(not False) # True
print(not not not not True) # True
```

Dado que estamos tratando con booleanos, hemos considerado que usar `True` y `False` es lo mejor y más claro, pero es totalmente válido emplear `1` y `0` respectivamente para representar ambos estados. Y por supuesto los resultados no varían

```
print(not 0) # True
print(not 1) # False
```

Bibliografía:

- Extraído de <<Operadores Lógicos>>. 26 de septiembre de 2022. Consultado el 28 de noviembre del 2022