Los Booleanos: el Ying y el Yang

Los Booleanos son un tipo llamado así en honor a George Bool (https://en.wikipedia.org/wiki/George_Boole) un matemático autodidacta inglés, del siglo XIX.

Bool transformó la lógica de los antiguos griegos y le dió rigor matemático con la llamada *Algebra de Bool* en su libro The Laws of Thought (https://en.wikipedia.org/wiki/The_Laws_of_Thought) (nada menos). Sirvió para preparar los cimientos para la Ciencia de la Computación moderna.

En Python el tipo se llama `bool` y sólo tiene dos instancias:

- `True`
- `False`

Las operaciones que se pueden hacer sobre ellos, son las siguientes:

- `not` : negar o invertir un valor
- `and` : combinar dos booleanos de una cierta forma
- `or` : combinar dos booleanos de otra forma

Los veremos con detalle más abajo.

OJO, lo que viene en seguida puede ser un poco rollo, pero es muy importante y no queda más que aprenderlo. Además, cuando llegues al final, entenderás este chiste:

> Una mujer se acerca a su marido programador y le pregunta:
>
> - Esta noche, ¿me vas a llevar a cenar o vas a seguir frente a la pantalla?
>
> El marido,sin levantar la vista del teclado, contesta:
>
> - Sí

Para mejor entenderlos, usaremos algo llamado **Tabla de Verdad**

> Una tabla de verdad es un método utilizado en lógica y matemáticas para representar los posibles resultados de una operación lógica o proposición que involucre una o más variables.

`not`

`not` es un operador *unario* (sólo actúa sobre un elemento) que cuando se aplica a un booleano, devuelve el otro valor posible:

- `not True` devuelve `False`
- `not False` devuelve `True`

`not` tiene prioridad (se evalúa antes) que los demás operadores booleanos.

Se dice que es una *negación lógica* y es similar, aunque no igual, al *no* de nuestro lenguaje.

Tabla de Verdad

A	not A
True	False
False	True

`and`

Al contrario de `not`, los demás son *binarios* (operan sobre *dos* valores).

`and` sólo devuelve `True` cuando ambos operandos son `True`. En todos los demás casos, devuelve `False`.

Se parece al *y* de nuestro lenguaje.

Tabla de Verdad

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

`or`

El `or` es también un operador binario y se parece al *o* de nuestro lenguaje, pero hay diferencias sutiles.

Si al menos uno de los operandos es `True`, devuelve `True`. Si ambos son falsos, devuelve `False`.

Tabla de Verdad

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Mirando su tabla, ¿puedes ver en qué se distingue `or` de nuestro *o*?

Si estás en un restaurante y el camarero te dice:

De primero hay lentejas o ensaladilla rusa

y tú contestas *Sí*, es muy probable que el camarero no te entienda y te diga *Sí ¿qué?*. El *o* del camarero es exclusivo, tiene que ser una cosa o la otra, no pueden ser las dos a la vez.

Es decir, la diferencia entre nuestro *o* y el `or` de Python está en la primera linea de la tabla.

Repasa el chiste ahora. ¿Entiendes lo que contesta el programador? ¿Entiendes el cabreo de su sufrida señora?

xor

`xor` viene de *Exclusive Or* y es un operador lógico que funciona como nuestro *o*. No es muy usado, y no hace falta que te lo aprendas, pero es bueno que sepas que existe.

El operador ==

El operador `==` sirve para comparar dos cosas en Python y devuelve un booleano.

- `3 == 4` devuelve `False`
- `"Hola" == "Hola"` devuelve `True`

Se usa mucho en las *expresiones booleanas*.

> No lo confundas con el operador de asignación (`=`) que se utiliza para crear variables y cambiar su valor.

```python m = "Rigoberto" # asignación (definimos variable) p = 324 # asignación (definimos variable)

m == p # Comparación: qué devuelve esa expresión booleana?

same = (m == p) # Asignación Y comparación: qué está pasando y qué hay en same?
```

Leyes de De Morgan

Se trata de dos leyes relacionadas con el efecto de un `not` sobre una expresión que contiene un `and` o bien una expresión que contiene `or` .

Son muy útiles para simplificar expresiones lógicas en lenguajes de programación, y también para simplificar circuitos lógicos.

Ley 1: Negación del `and`

Suponiendo que A y B son variables booleanas (no importa su valor)

```
(not A) or (not B) = not(A and B)
```

Ley 2: Negación del `or`

```
(not A) and (not B) = not(A or B)
```

No hace que te las aprendas de memoria, pero es bueno que las tengas a mano, para que cunado tengas una expresión lógica compleja, la sepas simplificar acudiendo a nuestro amigo De Morgan.

Más operadores

Hay más operadores booleanos, además del `xor`, pero no hace falta que te los aprendas, porque no se usan a penas y el el fondo son combinaciones de los que ya hemos visto.

Antes de seguir, hagamos unos ejercicios en el lab correspondiente.