Cours

Nous avons déjà rencontré les booléens dans la première séquence.

En programmation informatique, un booléen est un type de variable à deux états (généralement notés vrai et faux), destiné à représenter les valeurs de vérité de la logique et l'algèbre booléenne. Il est nommé ainsi d'après George Boole (1815-1864), fondateur dans le milieu du XIXe siècle de l'algèbre portant son nom.

Nous avons vu qu'en Python, les deux valeurs booléennes sont notées `True` et `False`.

De manière équivalents, on adopte souvent une notation numérique en associant 1 à `True` et 0 à `False`.

1. Opérateurs booléens de base

Dans le cours sur les bases de Python, nous avons déjà vu les opérateurs `or`, `and` et `not`.

Opérateur OU

Définition

Soit a et b deux expressions :

$$a \text{ OU } b \text{ est vrai} \iff a \text{ est vrai ou } b \text{ est vrai}$$

Table de vérité de l'opérateur OU :

a	b	a OU b
1	1	1
1	0	1
0	1	1
0	0	0

Remarque

En logique l'opérateur OU est **inclusif** : cela signifie que a OU b est vrai aussi lorsque a est vrai et b est vrai. Dans la langue courant, le mot *ou* est le plus souvent **exclusif** : dans un menu, par exemple "fromage ou dessert" ne permet pas de prendre les deux.

Opérateur ET

Définition

Soit a et b deux expressions :

$$a \text{ ET } b \text{ est vrai} \iff a \text{ est vrai et } b \text{ est vrai}$$

Table de vérité de l'opérateur ET :

a	b	a ET b
1	1	1
1	0	0
0	1	0
0	0	0

Opérateur NON

Définition

Soit a une expression :

$$(\text{NON } a) \text{ est vrai} \iff a \text{ est faux}$$

Table de vérité de l'opérateur NON :

a	NON a
1	0
0	1

2. Expressions booléennes

Les opérateurs de base peuvent être combinés pour formuler des expressions booléennes plus complexes. Pour éviter des problèmes d'interprétation, il est préférable d'utiliser des parenthèses pour marquer les priorités.

Exercice

Recopier et compléter la table de vérité ci-dessous :

a	b	NON a	NON b	(NON a) ET (NON b)	NON((NON a) ET (NON b))	a OU b
1	1					
1	0					
0	1					
0	0					

Que peut-on constater ?

3. Le ou exclusif

Le OU logique étant inclusif, on définit un opérateur spécifique pour le ou exclusif, appelé opérateur XOR.

Définition

Soit a et b deux expressions :

$$a \text{ XOR } b \text{ est vrai} \iff (a \text{ est vrai et } b \text{ est faux}) \text{ ou } (a \text{ est faux et } b \text{ est vrai})$$

Table de vérité de l'opérateur XOR :

a	b	a XOR b
1	1	0
1	0	1
0	1	1
0	0	0

En Python, l'opérateur `xor` n'existe pas. Le ou exclusif est noté `^`.

```
>>> True ^ False
True
```

4. L'addition binaire en mode booléen

Lorsque nous posons l'addition de deux entiers écrits en base 2, nous avons besoin d'additionner des groupes de 3 bits (un pour chaque nombre et un pour la retenue).

Voyons ce que donne l'addition de trois bits :

a	b	c	$a+b+c$
0	0	0	0
1	0	0	1
0	1	0	1
0	0	1	1
1	1	0	10
1	0	1	10
0	1	1	10
1	1	1	11

Exercice

En assimilant 0 à `False` et 1 à `True` écrire une fonction `add_trois_bits(a, b, c)` qui retourne la somme $a+b+c$ en utilisant uniquement les opérateurs ET, OU et NON. On retournera la somme sous la forme d'une chaîne de deux caractères (`"01"` par exemple).

```
def add_3_bits(a, b, c) :
    unite = int(...)
    deuzaine = int(...)
    return str(deuzaine)+str(unite)

assert add_3_bits(0,0,0)=="00"
assert add_3_bits(1,0,0)=="01"
assert add_3_bits(0,1,0)=="01"
assert add_3_bits(0,0,1)=="01"
assert add_3_bits(1,1,0)=="10"
assert add_3_bits(1,0,1)=="10"
assert add_3_bits(0,1,1)=="10"
assert add_3_bits(1,1,1)=="11"
print("C'est parfait !")
```