NSI	**Représentation numérique de l'information** Booléens	**Cours 6**

En Python une variable de type bool ne peut prendre que deux valeurs True et False (vrai et faux).

Une expression booléenne est composée de booléens et d'opérateurs. Nous utilisons dans la suite les opérateurs logiques **AND**, **OR** et **NOT** (soit **ET**, **OU** et **NON**).

- Pour l'opérateur **ET**, nous obtenons les résultats présentés dans le tableau suivant que nous pouvons résumer par : (a **ET** b) est vrai si et seulement si **a est vrai et b est vrai**.

b\a	**0**	**1**
0	0	0
1	0	1

En langage Python, l'expression a **ET** b s'écrit a **and** b. Remarquons que si nous remplaçons les valeurs **True** et **False** respectivement par **1** et **0**, l'expression **a and b** correspond à **a * b**, c'est-à-dire le produit.

*Note : En Python, l'opérateur **&** agit sur les bits, un à un. Par exemple **b1b2b3 & b4b5b6** a pour valeur **(b1 & b4)(b2 & b5)(b3 & b6)**. Donc **6 & 3** a pour valeur **2**, car en binaire **110 & 011** a pour valeur **010**.*

- Pour l'opérateur **OU**, nous obtenons les résultats présentés dans le tableau suivant et résumés par : (a **OU** b) est faux si et seulement si **a est faux et b est faux**.

b\a	**0**	**1**
0	0	1
1	1	1

En langage Python, l'expression a **OU** b s'écrit a **or** b. Si nous remplaçons les valeurs **True** et **False** respectivement par **1** et **0**, l'expression **a or b** correspond à **a + b - a * b**.

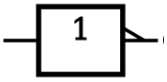
*Note : En Python, l'opérateur **|** agit sur les bits, un à un. Par exemple **b1b2b3 | b4b5b6** a pour valeur **(b1 | b4)(b2 | b5)(b3 | b6).** Donc **6 | 3** a pour valeur **7**, car en binaire **110 | 011** a pour valeur **111**.*

- Compléter le tableau suivant :

Base 10		**Base 2**		**Base 2**	**Base 10**
a	**b**	**a**	**b**	**a|b**	**a|b**
5	**2**	**0101**	**0010**	**0111**	**7**
4	**1**	**0100**	**0001**	**0101**	**5**
3	**13**	**0011**	**1101**	**1111**	**15**

- Pour l'opérateur **NON**, le résultat est simple : (**NON a**) est vrai si et seulement si a est faux. En langage Python, l'expression **NON** a s'écrit **not a**.

a	
0	1
1	0



*Note : En Python, l'opérateur **~** agit sur les bits, un à un. Quel que soit le nombre de bits utilisés, **~**0...0 vaut 1...1 , soit en décimal : **~**0 vaut -1. (voir plus tard avec les relatifs)*

Pour les tests, les opérateurs mathématiques de comparaison utilisés sont : **==**, **!=**, **<**, **<=**, **>**, **>=**.

Un nombre n'est pas égal au caractère qui le représente. Mais Python reconnaît si un entier et un flottant ont la "même valeur".

```
>>> 5 == "5"
False
>>> 5 == 5.0
True
```

Dans une expression booléenne, les opérandes peuvent être de différents types. Pour Python, ce qui est nul ou vide peut être interprété comme **False**, (les nombres 0 et 0.0, les chaînes de caractères vides, etc), le reste comme **True**. Certains langages n'ont pas de booléens et utilisent seulement **1** pour **True** et **0** pour **False**.

```
>>> if 2 and 5: print("c'est vrai")

c'est vrai
```

Séquentialité des opérateurs and et or

- Dans une expression **a** and **b**, **a** est évalué en premier. Si **a** est faux, l'expression prend la valeur de **a**, sinon elle prend la valeur de **b**.
- Dans une expression **a** or **b**, **a** est évalué en premier. Si **a** est vrai, l'expression prend la valeur de **a**, sinon elle prend la valeur de **b**.

```
>>> 0 and 5
0
>>> 2 and 5
5
>>> 2 and 0
0

>>> 0 or 5
5
>>> 2 or 5
2
>>> 2 or 0
2
```

L'expression **not a** ne peut avoir pour valeur que **False** ou **True**.

Table de vérité

Dans cette table, **F** signifie **False** et **T** signifie **True**.
Nous résumons ici les principaux résultats pour des expressions booléennes simples.

a	b	not a	not b	a and b	a or b	not a and not b	not a or not b
T	T	F	F	T	T	F	F
T	F	F	T	F	T	F	T
F	T	T	F	F	T	F	T
F	F	T	T	F	F	T	T

Nous pouvons déduire, en comparant les valeurs que :

- **not a** and **not b** est équivalente à **not (a or b)**
- **not a** or **not b** est équivalente à **not (a and b)**

Ces deux équivalences s'appellent les lois de De Morgan.

- Un autre opérateur joue un rôle majeur dans le calcul booléen et le fonctionnement d'une machine. Il s'agit de l'opérateur logique **XOR**, le **OU exclusif** :
    - **a** XOR **b** est équivalent à (**a** ET (**NON b**)) OU ((**NON a**) ET **b**).

b\a	**0**	**1**
0	0	1
1	1	0

Le ou exclusif peut être exprimé comme « **soit l'un soit l'autre** » mais pas les deux.
Ainsi, contrairement au OU, a XOR b est faux si a est vrai et b est vrai.

*Note : En Python, l'opérateur ^ agit sur les bits, un à un. Par exemple **b1b2b3 ^ b4b5b6** a pour valeur **(b1 ^ b4)(b2 ^ b5)(b3 ^ b6).** Donc **6 ^ 3** a pour valeur **5**, car en binaire **110 ^ 011** a pour valeur **101**.*

Remarque :

```
>>> 1 & 1
1
>>> 1 & 0
0
>>> 0 & 1
0
>>> 0 & 0
0
```

Dans chaque cas, l'expression **a & b** a pour valeur la **retenue** dans l'addition **a+b**.

```
>>> 1 ^ 1
0
>>> 1 ^ 0
1
>>> 0 ^ 1
1
>>> 0 ^ 0
0
```

De même, l'expression **a ^ b** a pour valeur **le chiffre** des unités dans l'addition **a+b**.

L'opérateur **^** s'utilise sur les bits mais il n'y a pas d'opérateurs logiques pour **XOR** comme and pour ET.
Une possibilité est d'écrire une fonction **xor** qui prend en paramètres deux variables **a** et **b** et renvoie **a** XOR **b**. Contrairement aux opérateurs **and** et **or**, l'évaluation de chacune des deux valeurs a et b est obligatoire dans tous les cas.

```
def xor(a, b):
    return (a and not(b)) or (not(a) and b)
```