RND. Booléen

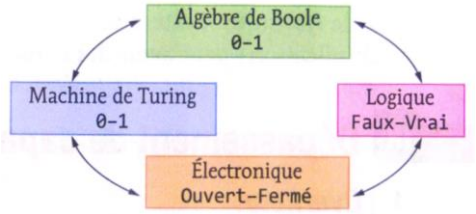
I/ Un peu d'histoire

George Boole (1815 – 1864) est issu d'un milieu peu fortuné et c'est en autodidacte qu'il apprend le grec et le latin. Pour vivre, il enseigne dans des écoles en milieu rural et pendant ses loisirs, il étudie les œuvres de grands mathématiciens, en particulier Lagrange et Laplace. Ses premières publications lui ouvrent les portes du *Queen's College* à Cork (Irlande) où il enseignera jusqu'à la fin de sa vie.

On le considère avec Augustus de Morgan comme le fondateur de la logique mathématique. Il définit des opérations sur les conditions, attribuant 1 à celles qui sont vraies et 0 à celles qui sont fausses. Cette algèbre binaire, qui porte désormais son nom (l'algèbre booléenne), est utilisée dans de nombreux domaines, en particulier celui de la conception des systèmes informatiques.

A savoir : On associe ainsi VRAI à 1 et FAUX à 0. D'autre part, une condition doit être VRAIE ou FAUSSE mais pas VRAIE et FAUSSE à la fois, c'est le principe du tiers-exclu.

En 1938, l'Américain Claude Shannon prouve que des circuits électriques peuvent résoudre tous les problèmes que l'algèbre booléenne peut résoudre. Avec les travaux d'Alan Turing en 1936, cela constitue de ce qui deviendra l'informatique.



Pour en savoir plus sur Boole : http://www.bibmath.net/bios/index.php?action=affiche&quoi=boole

Pour en savoir plus sur Lagrange : http://www.bibmath.net/bios/index.php?action=affiche&quoi=lagrange

Pour en savoir plus sur Laplace : http://www.bibmath.net/bios/index.php?action=affiche&quoi=laplace

II/ Opérations fondamentales de l'algèbre booléenne

1/ Opérations fondamentales

A savoir : les opérations fondamentales de l'algèbre booléenne sont les trois suivantes :

- **la conjonction, notée « ET *(AND en anglais)* » ou « & » ou « . » ,**
- **la disjonction, notée « OU *(OR en anglais)* » ou « | » ou « + » ,**
- **la négation, notée « NON *(NOT en anglais)* ou « ~ » ou avec une barre au-dessus de la condition**.

La conjonction se rapproche de la multiplication, la disjonction de l'addition.

Ces trois opérations suffisent pour décrire toute opération en algèbre booléenne, il suffira de les **combiner** pour arriver au résultat souhaité. Ces opérations fondamentales sont des exemples de **fonctions logiques**.

2/ Tables de vérité

L'algèbre booléenne ne contient que deux éléments : VRAI ou FAUX. On peut ainsi décrire toutes les possibilités de chaque fonction logique et regrouper les résultats sous forme d'un tableau, appelé **table de vérité**.

Exemple : soient a, b des conditions.
On peut représenter les opérations logiques fondamentales sous la forme des tableaux de vérité suivants :

Avec des « V » (VRAI) et des « F » (FAUX)

Conjonction

a	b	a AND b
V	V	V
V	F	F
F	V	F
F	F	F

Disjonction

a	b	a OR b
V	V	V
V	F	V
F	V	V
F	F	F

Négation

a	NOT a
V	F
F	V

Même représentation avec des « 0 » et des « 1 »

Conjonction

a	b	a AND b
1	1	1
1	0	0
0	1	0
0	0	0

Disjonction

a	b	a OR b
1	1	1
1	0	1
0	1	1
0	0	0

Négation

a	NOT a
1	0
0	1

On remarquera que pour **n conditions**, il y a **2^n lignes** dans une table de vérité.

A savoir : la disjonction exclusive, combinaison des opérations logiques fondamentales. Elle est notée « XOR » ou « ^ ».
Soient a et b deux conditions. a XOR b est vraie si et seulement si a et b ont des valeurs différentes.

Disjonction exclusive

a	b	a XOR b
V	V	F
V	F	V
F	V	V
F	F	F

Remarque : la disjonction exclusive correspond un peu à un choix à faire à Noël ou à un anniversaire : « soit je reçois le cadeau X, soit je reçois le cadeau Y mais pas les deux ».

3/ L'algèbre booléenne en Python

a) Le type « bool »

En Python, nous avons le type **« bool »** pour représenter les booléens. Une variable de type « bool » ne peut prendre que deux valeurs **True** ou **False**.

Important : il faut impérativement écrire **T**rue et **F**alse avec une **majuscule** en **première lettre** sinon Python ne reconnaîtra pas les valeurs et une erreur apparaîtra lors de l'exécution du programme.
Ils font partie des **mots clés** du langage qui sont colorés dans la plupart des environnements de développement.

b) Séquentialité des opérateurs « and « et « or »

A savoir :

- pour deux expressions a et b, en python**, leur conjonction** est le mot clé **and, leur disjonction** le mot clé **or** et **leur négation** le mot clé **not**,
- **une variable vide ou** contenant **0** sera évaluée comme **fausse, dans tous les autres cas,** elle sera évaluée comme **vraie.**

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 **celle 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 **celle de b.**
On remarquera que b est la dernière valeur évaluée à chaque fois.

Quant à l'expression not a, elle ne peut avoir que **True** ou **False** comme valeur.

Exemple :

Conjonction	Disjonction	Négation
0 and 5 = 0 car 0 est faux **2 and 5** = 5 car 2 est vrai et 5 aussi **5 and 0** = 0 car 5 est vrai et 0 faux	**0 or 5** = 5 car 0 est faux et 5 vrai **5 or 0** = 5 car 5 est vrai **5 or 2** = 5 car 5 est vrai	**not 6** = faux car 6 est vrai **not 0** = vrai car 0 est faux

Remarque : évaluer l'expression 0 and 5 est plus rapide qu'évaluer 5 and 0 car il y a un test en moins. Idem pour 5 or 0 et 0 or 5.

c) Opérateurs bit à bit « & « ; « | » et « ~ »

Les opérateurs **&** ; **|** et **~** signifient respectivement **and** ; **or** et **not mais pour des opérations bit à bit**. Ils sont donc utiles pour des opérations sur des nombres en écriture binaire.

Exemple 1 : 5 and 6 = 6 car 5 et 6 sont vrais mais 5 & 6 = 4.
Sur 4 bits :
$5 = 0101_2$
$6 = 0110_2$
donc 5 & 6 = (0101) & (0110) = (0 & 0)(1 & 1)(0 & 1)(1 & 0) = 0100_2= 4.

Exemple 2 : 5 or 6 = 5 car 5 est vrai mais 5 | 6 = 7.
Sur 4 bits :
$5 = 0101_2$
$6 = 0110_2$
donc 5 | 6 = (0101) | (0110) = (0 | 0)(1 | 1)(0 | 1)(1 | 0) = 0111_2= 7.

Exemple 3 : not 7 = false mais ~ 7 = 8.
Sur 4 bits :
$7 = 0111_2$
~ 7 = 1000_2= 8.
Remarque : L'opérateur ~ inverse simplement chaque bit.