1. Logique booléenne et instructions conditionnelles : principes et exemples. Applications
2. **Algèbre de Boole et Logique Booléenne**

George Boole, Kicésa ? C’est un logicien, mathématiciens, philosophe anglais du XIXe siècle. Il est à l’origine de l’algèbre qui porte son nom, la partie des mathématiques qui s'intéresse à une approche algébrique de la logique.

La logique des prédicats/propositions

On appelle prédicat ou proposition une  « phrase » qui peut être soit vrai, soit fausse.

C’est une logique de type tout ou rien, une proposition ne pas être vrai et fausse.

Dans l’algèbre de Boole, on appelle E l’ensemble constitué de deux éléments appelés **valeurs de vérité** {VRAI, FAUX}, on peut le noter :

E = {1,0}

Sur cet ensemble, on peut définir deux lois, AND et OR et une loi complémentaire qu’on appelle la négation/complémentaire/inverse/contraire.

AND et OR sont des opérateurs binaires alors que la négation est un opérateur unaire.

Les opérateurs de base :

* Conjonction (AND)

Définit de la manière suivante, « A AND B » est vrai si et seulement si A est vrai et B est vrai aussi. On peut construire la table de cette loi avec A,B ∈ E, on appelle cela une table de vérité :

|  |  |  |
| --- | --- | --- |
| A | B | AND |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

En langage python, cet opérateur s’utilise avec la notation « and ».

*proposition 1 and proposition 2*

Certain autre langage utilise la notation « && ». Il ne faut pas confondre avec l’opérateur « & » qui correspond à un AND effectuer bit à bit.

* Disjonction (OR)

Définit de la manière suivante, « A OR B » est vrai si A est vrai ou B est vrai ou A et B est vrai. ( « l ’un, l’autre ou les deux »). On peut donc aussi construire la table de vérité de cette loi.

|  |  |  |
| --- | --- | --- |
| A | B | OR |
| O | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

En python, on va utiliser le mot clé « or » :

*proposition 1 or proposition 2*

Certain autre langage utilise la notation « || ». Il ne faut pas confondre avec le « | » de python qui va effectuer un « OU » bit à bit entre deux variables.

* Négation

Définit de la manière suivant, la négation de A est VRAI si et seulement si A est FAUX et inversement.

|  |  |
| --- | --- |
| A | Non A |
| 0 | 1 |
| 1 | 0 |

En python, on va placer le mot clé « not » devant la proposition. Dans la plupart des autres langage c’est « ! » qu’on va placer avant la proposition.

D’autres opérateurs (NOR,NAND et XOR)

* NOR

C’est le résultat de l’opérateur OR suivi d’une négation sur ce résultat. On peut le traduire par NON(A AND B).

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | OR | NOR |
| O | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

Il n’existe pas d’opérateur « nor » en python tel qu’on pourrait l’écrire :

*proposition 1 nor proposition 2*

Mais on peut quand même fabriquer cet opérateur en écrivant

*not (proposition 1 or proposition 2)*

* NAND

C’est le résultat de l’opérateur AND suivi d’une négation sur ce résultat. Peur le traduire par NON(A AND B).

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | AND | NAND |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Il n’existe pas d’opérateur « nand» en python tel qu’on pourrait l’écrire :

*proposition 1 nand proposition 2*

Mais on peut quand même fabriquer cet opérateur en écrivant

*not (proposition 1 and proposition 2)*

* XOR

Cette opérateur correspond au « OR » exclusif, c’est-à-dire qu’on peur le définir par « A XOR B » est vrai si est seulement A est vrai mais pas B ou B est vrai mais pas A (l’un, l’autre mais pas les deux)

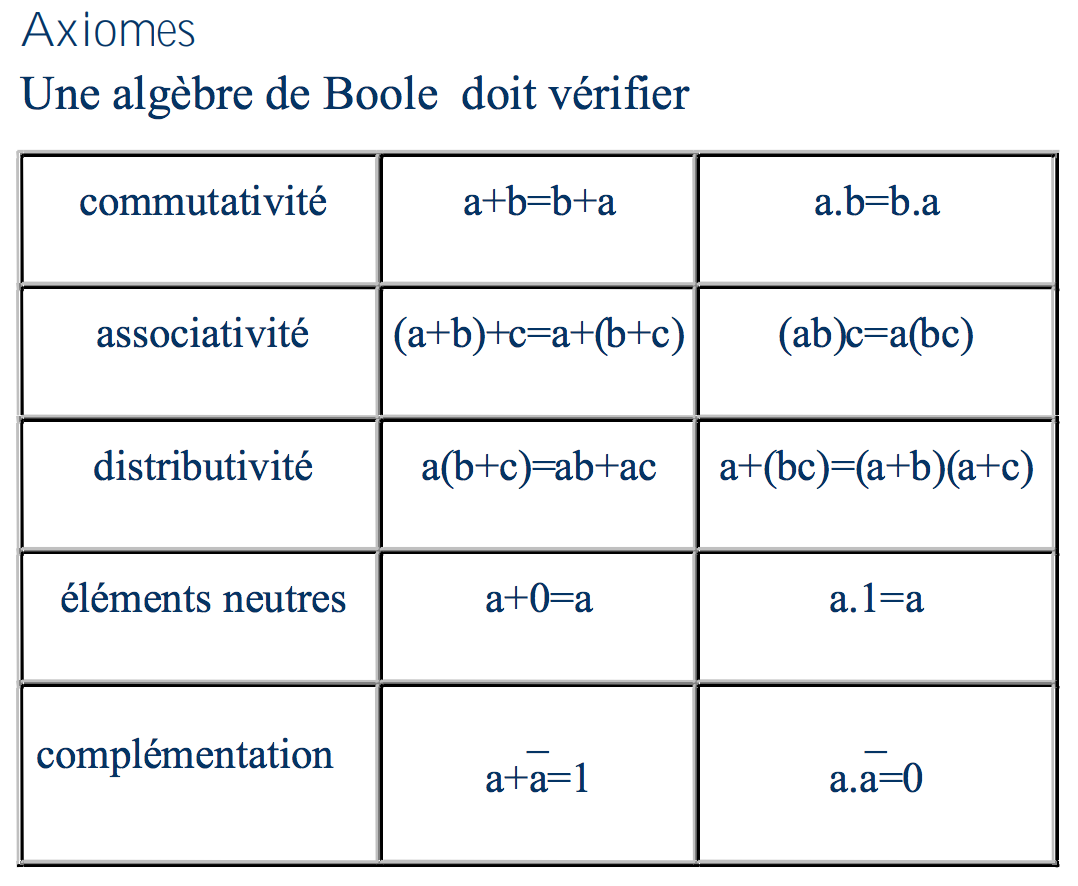
|  |  |  |
| --- | --- | --- |
| A | B | XOR |
| O | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Il n’existe pas d’opérateur « xor» en python tel qu’on pourrait l’écrire :

*proposition 1 xor proposition 2*

Mais on peut quand même fabriquer cet opérateur en écrivant

*(proposition 1 and not proposition 2) or (not proposition 1 and proposition 2)*



**Développement**: Coder les méthodes OR,AND, … seulement avec des instructions conditionnelles.

Proposition plus complexe : D = A.B.!C + !A.C

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | A.B. !C | !A.C | A.B.!C + !A.C |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |

Coder sur python par :

*def* **operation** (a,b,c):

*return* OR( (AND (AND(a , b) , NON(c))) , AND (NON(a) , c))

1. **Instruction conditionnelle**

If

If else

If elif else

For

While

Switch case

1. **Exemples et application**