1- Logique booléenne et instructions conditionnelles : principes et exemples. Applications

Orienté pour une classe de seconde

I- Algèbre de Boole et Logique Booléenne

George Boole, Kicésa ? C'est un logicien, mathématicien, 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 $A,B \in E$, 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
0	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
0	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
0	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)

Axiomes

Une algèbre de Boole doit vérifier

commutativité	$a+b=b+a$	$a.b=b.a$
associativité	$(a+b)+c=a+(b+c)$	$(ab)c=a(bc)$
distributivité	$a(b+c)=ab+ac$	$a+(bc)=(a+b)(a+c)$
éléments neutres	$a+0=a$	$a.1=a$
complémentation	$a+\bar{a}=1$	$a.\bar{a}=0$

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))

II- Les instructions conditionnelles

Tous les langages de programmation utilisent les instructions conditionnelles. Ces instructions exécutent différentes actions en fonction de l'évaluation d'une condition booléenne (VRAI ou FAUX).

Si … alors (If … then)

C'est l'instruction la plus connue. Noté en python :

```
Instruction
if [condition] :
    ...
    ...
Instruction
```

Quand un interpréteur tombe sur un IF dans le code, il évalue la condition. Si celle-ci est vraie, il exécute les lignes de code à l'intérieur du IF, sinon il passe à l'instruction suivante (après le bloc IF).

Les différentes conditions possibles :

Condition	Notation en python
a est égale à b	a == b
a est différent de b	a != b
a est inférieur à b	a < b
a est inférieur ou égal à b	a <= b
a est supérieur à b	a > b
a est supérieur ou égal à b	a >= b
a et b référence le même objet	a is b
a et b ne référence pas le même objet	a is not b
a in liste1	a est inclus dans liste1
a not in liste1	a n'est pas inclus dans liste1

Si .. alors … Sinon (If … then … else)

C'est une variante du IF. Noté en python:

```
Instruction
if [condition] :
    ...
    ...
else :
    ...
    ...
Instruction
```

Quand l'interpréteur tombe sur un IF … ELSE, il évalue la condition, si elle est vraie il exécute les lignes de code du bloc IF, mais si cette condition est fausse, il exécute le code du bloc ELSE. Il n'est pas possible d'exécuter les des blocs de code, c'est soit l'un soit l'autre en fonction de l'évaluation de la condition.

SI … Sinon Si … sinon (If … elseif… else)

```
Instruction
if [condition1] :
    ...
    ...
elif [condition2] :
    ...
    ...
elif [condition3] :
    ...
    ...
else :
    ...
    ...
Instruction
```

Il est possible de combiner plusieurs conditions. Seuls les énoncés suivant la première condition qui se trouve être vraie seront exécutés. Toutes les autres déclarations seront ignorées. Il n'est pas nécessaire d'avoir le ELSE final.

Tant que / faire … tant que … (While … / Do … while …)

```
Instruction
while [condition] :
    ...
    ...
Instruction
```

Le but de la boucle *while* est de répéter certaines instructions tant qu'une condition est respectée. Il n'est donc pas nécessaire de connaître le nombre de répétition à l'avance. Il faut par contre faire attention aux boucles infinies. (cas d'une boucle spéciale : la boucle d'événement)

La boucle do… while n'existe pas en python mais on peut la fabriquer. La différence est que la condition est vérifiée après l'instruction, donc celle-ci est exécutée au moins une fois. Il est possible d'émulé une fonction do…while en python en recopiant le code à l'intérieur de la boucle avant celle-ci.

Pour … (for …) :

```
Instruction
for [variable] in [intervalle]:
    ...
    ...
Instruction
```

La boucle for permet de répéter un nombre fini de fois une instruction en fonction du nombre d'élément d'une liste ou d'une séquence de nombre (type rang)

Switch case

```
switch (expr) {
  case 'Oranges':
    console.log('Oranges are $0.59 a pound.');
    break;
  case 'Mangoes':
    console.log('Mangoes are $5.78 a pound.');
    break;
  case 'Papayas':
    console.log('Mangoes and papayas are $2.79 a pound.');
    break;
  default:
    console.log('Sorry, we are out of ' + expr + '.');
}
```

L'instruction switch n'existe pas en python mais existe dans beaucoup d'autre langage (JAVA, C#, JS,…). C'est le même principe qu'un enchainement de if … elif … elif …else. L'avantage du switch est qu'il est plus lisible pour le lecteur et plus rapide quand le nombre de cas augmente.

III- Exemples et application

Devinette (dans les deux sens, dichotomie)
Exemple de boucles imbriquées
Tracer la courbe d'une fonction homographique
Tri
…