Type de document	Cours	Classe	1re	Durée	1h	Date	21/09/2022
Thème et contenu(s)	Représentation des données – Langages & programmation – Architecture						
Capacités attendues	Savoir identifier et utiliser les principales fonctions logiques						
Prérequis	Cours sur le binaire						
Description	Premiers pas en logique booléenne en vue d'une application sur machine						

I) Soyons logiques : fonctions de base

Nous avons vu qu'en base 2, il y avait seulement 2 valeurs possibles : 0 et 1. On peut, par convention, assimiler le **0** à quelque chose de **faux**, et le **1** à quelque chose de **vrai**. Dès lors, on peut pousser le raisonnement un peu plus loin et retrouver des **fonctions logiques** qu'on utilise dans la vie courante. Dans la suite de ce cours, les bits seront désignés par des lettres de l'alphabet. Chaque fonction possède un symbole associé (on peut ainsi dessiner des circuits logiques) mais leur étude fera l'objet d'un cours à part.

I - a) Table de vérité

L'idée est de comparer des bits en entrée et de regarder, selon la fonction qu'on leur applique, ce qui se passe en sortie. On crée des tableaux appelés **tables de vérité** dans lesquels figurent **toutes les combinaisons possibles des entrées**, la fonction appliquée, et le résultat en sortie.

I - b) L'identité

Entree	Sortie
a	**identite(a)**
0	**0**
1	**1**

Cette fonction est peu utilisée car elle ne fait rien. Elle est assimilable à f(x) = x. Elle se comporte de la manière suivante : si j'ai un 0 en entrée, ma sortie vaudra 0. Si j'ai un 1 en entrée, ma sortie vaudra 1. Ma sortie est donc égale à mon entrée.

I - c) L'inverseur

Entree	Sortie
a	**inverseur(a)**
0	**1**
1	**0**

Cette fonction consiste à **envoyer en sortie l'inverse de l'entrée**. Elle se comporte exactement comme dans la vie courante : le contraire de **faux** est **vrai**, et vice-versa. On note, par convention en algèbre booléenne, $\bar{a}$ l'inverse de a, on l'appelle « a barre ». En Python, cette fonction se nomme **not**.

I - d) La fonction ET

On l'appelle aussi AND et en Python elle s'écrit sans surprise **and**. Elle nécessite 2 paramètres (bits) pour fonctionner. Par convention, on note un ET logique entre 2 éléments avec un **point**, comme : **a.b** (se lit « a et b »). Cette fonction ne produit une sortie vraie (à 1), que si toutes les entrées sont vraies (on peut formuler ça autrement en disant que toutes les conditions doivent être réunies donc vraies pour que la conséquence soit vraie). Ex « réel » : « si tu veux passer en terminale, il faut que tu sois en première et que tu aies de bonnes notes ». Si je suis en première mais que je n'ai pas de bonnes notes, je ne passerai pas en terminale. Si je travaille très bien mais que je suis en CE1, je ne passerai pas en terminale non plus. Si je ne suis ni bon élève, ni en première, il est également certain que je ne passerai pas en terminale. Il suffit donc qu'une seule (à fortiori les deux) condition soit fausse pour que mon objectif (ma sortie) soit faux. Mathématiquement, on assimile ça à une multiplication : tout élément multiplié par 0 vaut 0.

Entree		Sortie
a	b	**a.b**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

I - e) La fonction OU

On l'appelle aussi OR et en Python elle s'écrit **or**. Elle nécessite elle aussi 2 paramètres (bits). Par convention, on note un OU logique entre 2 éléments avec un **+**, comme : **a+b**. Il suffit qu'au moins une des deux (à fortiori les deux) conditions soit vraie pour que la sortie soit vraie. Exemple « réel » : « Si tu parles trop ou que tu utilises ton téléphone, tu auras deux heures de colle ». Pour obtenir les deux heures de colle, il suffit d'utiliser son téléphone. Ou bien, il suffit de parler. Ou même de faire les deux. Seul le fait de ne faire **ni l'un ni l'autre** permet de ne pas se faire coller. Malgré son signe trompeur, elle **ne ressemble que très peu** à l'opération mathématique d'addition.

Entree		Sortie
a	b	**a+b**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

II) Fonctions avancées

II - a) La fonction NAND

On l'appelle en français fonction NON ET. Elle consiste à appliquer la fonction AND et à inverser le résultat (Not + AND = NAND) lors de la sortie. On l'écrit : $\overline{a \cdot b}$. En Python elle s'écrit comme un combinaison de **not** et de **and** : **not(a and b)**

Entree		Sortie
a	b	$\overline{a \cdot b}$
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

II - b) La fonction NOR

On l'appelle en français NON OU. Elle est la contraction de Not + OR. Elle consiste à appliquer la fonction OU sur les entrées puis à inverser ce qu'on trouve, pour ensuite l'envoyer en sortie. On l'écrit : $\overline{a + b}$. En Python elle s'écrit comme un combinaison de **not** et de **or** : **not(a or b)**

Entree		Sortie
a	b	$\overline{a + b}$
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

II - c) La fonction XOR

La fonction OU EXCLUSIF (eXclusive OR) possède plusieurs propriétés très intéressantes, et est très utilisée dans de nombreux domaines de l'informatique dont la cryptographie. Elle ne produit une sortie vraie que si ses deux entrées sont différentes. Plus généralement, elle ne produit 1 en sortie que si le nombre de ses entrées à 1 est impair. On la retrouve dans la vie courante dans l'exemple suivant : « un en-cas sera distribué à bord, vous avez le choix entre sucré ou salé ». On ne peut pas choisir les deux, il faut en choisir un et un seul pour avoir sa nourriture. Il se note : $a \oplus b$. Le XOR est l'exact équivalent d'une addition binaire (si on fait abstraction de la propagation de retenue). En Python elle se note **^**.

Entree		Sortie
a	b	$a \oplus b$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

III) Aller plus loin

En regardant les tables de vérité des fonctions avancées, et surtout en les manipulant, vous pouvez remarquer certaines équivalences : $\overline{a \cdot b} = \bar{a} + \bar{b}$ et $\overline{a + b} = \bar{a}.\bar{b}$. On appelle cela les Lois de De Morgan. Elles sont très utilisées pour simplifier des expressions booléennes et des circuits logiques.