1 Variable de type logique

On a déjà manipulé à travers les structures conditionnelles (if-elif-else) ou les boucles non bornées (while) la notion de test d'une condition, et donc d'opérateur et de variable logique.

> Un **booléen** est une variable logique qui ne peut prendre que **deux valeurs** : Vrai ou Faux (1 ou 0, True ou False, état haut ou état bas).

Ce nom provient de G. Boole, mathématicien qui a développé une algèbre de la logique (milieu XIXème).

On retrouve évidemment ici le principe au cœur de la construction électronique des ordinateurs : le courant passe ou ne passe pas !

2 Opérateurs booléens

2.1 Principe

Les booléens peuvent se combiner selon des **opérateurs logiques**. Un opérateur logique reçoit une ou deux variables booléennes en entrée pour donner une valeur de sortie.

2.2 Tables de vérité

On peut représenter une **table de vérité** pour chaque opérateur, qui indique la valeur de sortie en fonction des valeurs d'entrée.

2.2.1 Opérateur NON (NOT)

La sortie est le contraire de l'entrée :

A	S
0	1
1	0

2.2.2 Opérateur ET (AND)

La sortie est à Vrai seulement si les deux entrées sont à Vrai en même temps :

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

2.2.3 Opérateur OU (OR)

La sortie est à Vrai si l'une au moins des deux entrées est à Vrai :

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

2.2.4 Opérateur OU exclusif (XOR)

La sortie est à Vrai si une seule entrée est à Vrai :

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Ces opérateurs ont un **caractère séquentiel** :
Par exemple :

- (A et B) est équivalent à : Si (non A) alors Faux, Sinon B.
- (A ou B) est équivalent à : Si A alors Vrai, Sinon B.

3 Lien avec la programmation

On exploite couramment des expressions booléennes en programmation. À la suite d'un « Si » des structures conditionnelles ou d'un « Tant que » des boucles non bornées, on trouve une expression booléenne, qui **s'évalue** donc soit à « Vrai », soit à « Faux ».

En Python par exemple, une expression booléenne comme `"a > 5"` **vaut** soit `True`, soit `False`.
De la même manière `"2 in tab"`, ou bien `"a or b and c"` **valent** soit `True`, soit `False`.

Remarque : voici un **défaut** de programmation de débutant classique ; on rencontre souvent chez les débutants des fonctions comme :

```
def exemple(n, tab):
    """ A ne pas reproduire ! """
    if n > 0 and len(tab) <= 10:
        return True
    else:
        return False
```

Il suffit en réalité de simplement écrire :

```
def exemple(n, tab):
    """ Bonne pratique """
    return  n > 0 and len(tab) <= 10
```

4 Transistors et portes logiques

4.1 Le transistor

Au niveau matériel, tout repose sur un composant électronique fondamental : le **transistor**.

Avant l'invention du transistor en 1947, les ordinateurs étaient construits à base de tubes électroniques, ou tubes à vide (beaucoup plus encombrants, fragiles, et sujets aux pannes).

Le rôle d'un transistor peut se résumer à un interrupteur commandé électriquement : il permet donc d'assurer la fonction « le courant passe, ou ne passe pas ».

La miniaturisation des transistors permet d'en regrouper des milliards sur quelques mm^2.

À l'aide de quelques transistors, on peut réaliser des **portes logiques** qui sont l'équivalent matériel des opérateurs booléens.
Voir schématisation électronique sur le doc annexe. (doc de Fabrice Sincère). Apprendre les portes NON, ET, OU, XOR.

4.2 Repères historiques

D'après de Wikipédia :
À la suite des travaux sur les semi-conducteurs, **le transistor a été inventé en 1947** par les Américains J. Bardeen, W. Shockley et W. Brattain, chercheurs des Laboratoires Bellnote 1. Ces chercheurs ont reçu pour cette invention le **prix Nobel de physique en 1956**.

Le transistor est considéré comme un énorme progrès face au tube électronique : beaucoup plus petit, plus léger et plus robuste, fonctionnant avec des tensions faibles, il fonctionne presque instantanément une fois mis sous tension, contrairement aux tubes électroniques qui demandaient une dizaine de secondes de chauffage, généraient une consommation importante et nécessitaient une source de tension élevée (plusieurs centaines de volts).

À partir du milieu des années 50, on commence à utiliser le transistor dans les ordinateurs, les rendant assez fiables et relativement petits pour leur commercialisation.

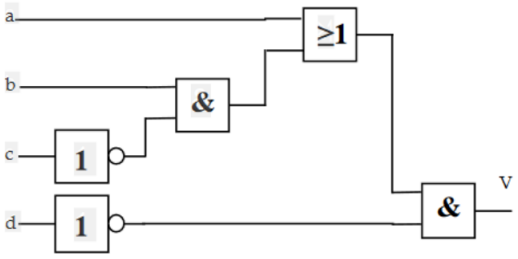
Après l'invention du **circuit intégré en 1958**, groupant en un petit volume plusieurs transistors et composants, **en 1969 est inventé le microprocesseur**, permettant à des milliers de transistors de fonctionner en harmonie sur un support, ce qui est encore une fois une révolution pour l'informatique moderne. Le 1er microprocesseur commercialisé par **Intel en 1971 (processeur 4004)** contient 2300 transistors : il est d'une **puissance comparable à l'ENIAC**, le premier ordinateur moderne dévoilé en 1946, qui occupait 167 m^2 pour un poids total de 30 tonnes.

De nos jours, le transistor est omniprésent dans la plupart des appareils de notre quotidien. Le nombre de transistor a considérablement augmenté pendant que sa taille diminuait, suivant en cela la Loi de Moore, avec par exemple 18 milliards de transistors pour 398 mm^2 en 2018. Aujourd'hui la gravure des transistors se réduit à quelques nanomètres.

4.3 Circuit combinatoire

Un **circuit combinatoire** est un module constitué de plusieurs portes logiques qui réalisent une **expression booléenne** avec plusieurs entrées pour fournir une ou plusieurs sorties.

Exemple : V = (a OU (b ET (NON c))) ET (NON d) :



Remarque : en exploitant la priorité des opérateurs (NON > ET > OU), on pourrait écrire l'expression booléenne précédente avec moins de parenthèses : V = (a OU b ET NON c) ET NON d

5 Exemple : l'addition binaire

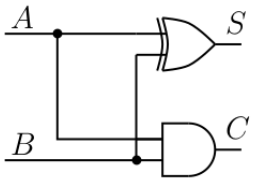
Attention : la fin de ce cours sera étudiée après avoir vu le chapitre P1-1 traitant de la numération binaire.

Avec les portes logiques, toutes les opérations mathématiques élémentaires peuvent être réalisées.

Rappel : addition de 2 bits A et B. On note S le bit de somme et C le bit de retenue (Carry) :
$0+0=\ 0$ (S, C) = (0, 0)
$0+1=\ 1$ (S, C) = (1, 0)
$1+0=\ 1$ (S, C) = (1, 0)
$1+1=10$ (S, C) = (0, 1)

5.1 Demi additionneur (half adder)

Un module demi additionneur est constitué d'une porte AND et une porte XOR. Il prend deux entrées et fournit deux sorties S et C :



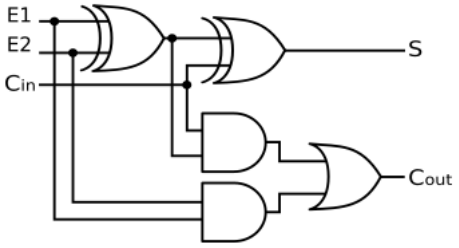
La table de vérité de ce circuit combinatoire est :

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

On a bien un moyen d'additionner les 2 bits A et B : S contient l'unité et C la retenue.

5.2 Additionneur complet (full adder)

L'additionneur complet doit pouvoir tenir compte d'une éventuelle retenue en entrée, en plus des 2 bits à ajouter.



La table de vérité de ce circuit combinatoire est :

E_1	E_2	C_{in}	S	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Pour réaliser l'addition binaire de 2 nombres de 4 bits par exemple, il faut 3 additionneurs complets et un demi-additionneur. Les **bits de poids faible** (LSB) des deux nombres sont les entrées du demi-additionneur et les autres bits sont en entrées des additionneurs complets et chaque C_{out} d'un additionneur est envoyée en entrée C_{in} de l'additionneur voisin. (la dernière retenue est "perdue").

Il existe des outils de simulation en ligne sur `https://logic.ly/demo`
ou `https://kazuhikoarase.github.io/simcirjs/`
Additionneur 4 bits : `https://kazuhikoarase.github.io/simcirjs/#-Lk3HmOoN11i2gJOohG5`