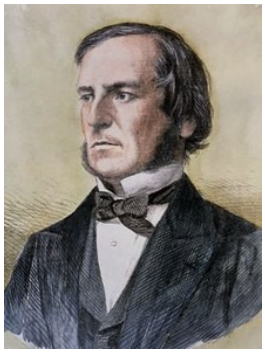
CH1 - LES BOOLEENS

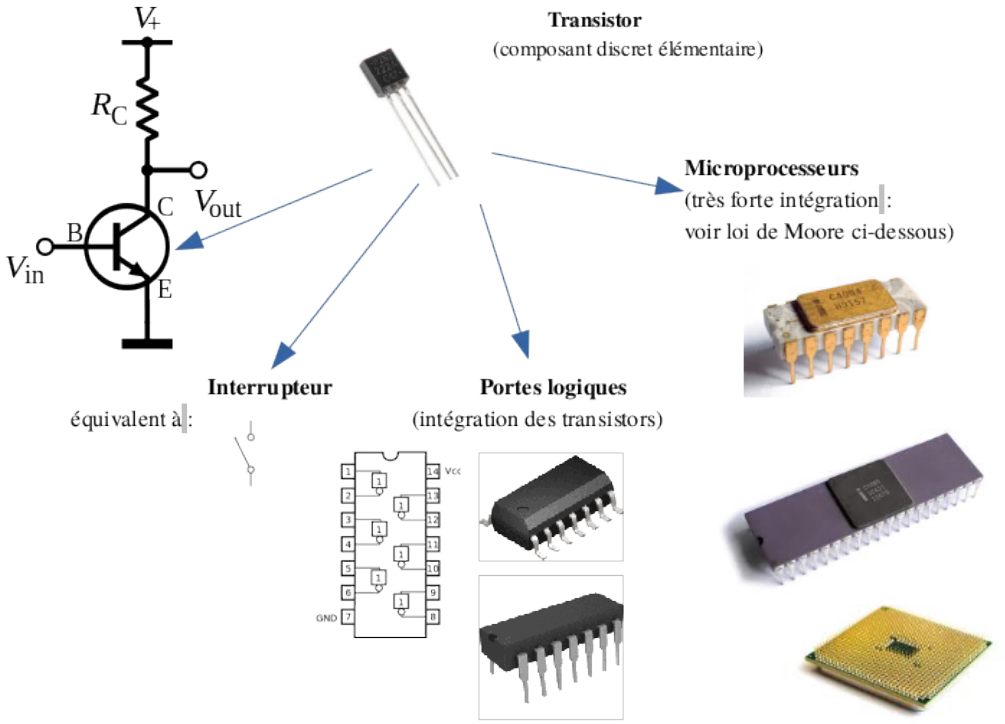
1. C'EST QUOI UN BOOLEEN ?



Le terme *booléen* vient du nom du mathématicien britannique George Boole.
Il est le créateur de la logique moderne qui s'appuie sur l'algèbre qui porte désormais son nom : l'*algèbre de Boole*.

Un booléen est une donnée dont la valeur ne peut prendre que deux états, soit l'état *vrai* soit à l'état *faux* (ou l'état fermé/ouvert en lien avec l'électronique). On utilise également le bit pour représenter des booléens : ainsi un 0 représente la valeur *faux* et un 1 représente la valeur *vrai*.

Dans les machines actuelles (PC, smartphones, etc.) le « 0 » ou « 1 » se traduit par « tension 0V» ou « tension +5V ». L'intermédiaire se fait à l'aide d'un petit composant appelé ***transistor***.



A voir pour info :
Documentaire vintage de 1994 (pour les 46s d'intro) https://youtu.be/W20d8Pj2fy8?start=0&end=46
Démontage PC https://youtu.be/Fk2kYo2E61A

A retenir :
Lorsque vous utilisez l'instruction **SI** (condition) … **ALORS** (faire), vous utilisez un booléen qui prend la valeur *VRAI* si la condition est vérifiée, et la valeur *FAUX* si elle ne l'est pas.

Exemples : En console

```
>>> a = 2
```

On assigne la valeur 2 à la variable a

```
>>> a == 2
```

On teste si a est bien égal à 2

→ Renvoie VRAI

```
>>> a = 2
>>> if a == 2 : print(”c’est bon !”)
 c’est bon !
```

On effectue une action (print) si la condition est vraie

```
>>> a = 3
>>> if a == 2 : print(”c’est bon !”)
```

→ Renvoie VRAI si a est bien égal à 2, FAUX sinon.

2. OPERATEURS BOOLEENS

ACT 1 Space Alarm !
ACT 2 Fonctions logiques

(a) Les opérations de base

On définit sur ces valeurs booléennes trois opérations : la négation, la conjonction et la disjonction, également appelées le NON, le ET et le OU logiques.

Le NON logique : la négation

Le NON logique d'un booléen A se définit par :

NON A vaut VRAI si et seulement si A vaut FAUX. Cet opérateur peut également être défini par sa table de vérité présentée ci-contre.

A	Non A
1	0
0	1

→ Le NON logique peut se noter : **~** ou ¬ ou ¯ (lire « barre ») **NOT**.

Le ET logique : la conjonction

Le ET logique entre deux booléens A et B se définit par :

A ET B vaut VRAI si et seulement si A vaut VRAI et B vaut VRAI. Remplir la table de vérité ci-contre.

A	B	A ET B
0	0	0
1	0	0
0	1	0
1	1	1

→ Le ET logique peut se noter **&** ou ∧ ou **AND**.

Le OU logique : la disjonction

Le OU logique entre deux booléens A et B se définit par :

A OU B vaut VRAI si et seulement si A vaut VRAI ou B vaut VRAI. Remplir la table de vérité ci-contre.

Cet opérateur peut également être défini par sa table de vérité présentée ci-contre.

A	B	A OU B
0	0	0
1	0	1
0	1	1
1	1	1

→ Le OU logique peut se noter **|** ou **v** ou **OR**.

(b) Et Python dans tout ça ?

En Python, les deux valeurs booléennes possibles sont True et False, qui sont de type bool. Vous pouvez le vérifier en exécutant les instructions ci-contre.

```
>>> v = True
>>> print(v)
 True
>>> print(type(v))
 <class 'bool'>
```

Le NON logique se note not, le ET logique se note and et le OU logique se note or.

Vous pouvez également créer une fonction permettant d'obtenir tous les résultats.

Exemples :

- Vérifier à l'aide de Python les tables de vérité des ET, NOT et OR logiques.

```
>>> not(True)
 False
>>> not(False)
 True
```

- fonction ET(a, b) en codant 1 pour True et 0 pour False.

```
def ET(a, b):
    if (a == 1 and b == 1):
        return 1
    else:
        return 0
```

Python **évalue les expressions logiques** de manière paresseuses (on parle de *lazy evaluation*).

Les opérateurs sont de type court-circuit :

- OR n'évalue le deuxième argument que si le premier est faux.
- AND n'évalue le deuxième argument que si le premier est vrai.

Autrement dit, l'évaluation se fait de la gauche vers la droite. Celle-ci est stoppée si la sortie est déterminée.

Exemples :

- True or x est vrai quelle que soit la valeur de x ; la valeur de x n'est pas lue dans la mémoire et l'opération or n'est pas effectuée en entier.
- False and not y est faux quelle que soit la valeur de y ; la valeur de y n'est pas lue dans la mémoire et l'opération and n'est pas effectuée en entier, l'opération not n'est pas effectuée du tout.

Remarque : À l'instar des opérateurs arithmétiques, les **opérateurs logiques ont une priorité** :

- Not
- And
- Or

L'utilisation des parenthèses qui peut parfois être logiquement inutile améliore la lisibilité du code et donc sa maintenance.

(c) Les autres opérations booléennes

Toutes les opérations booléennes peuvent se définir à partir des trois opérateurs ci-dessus.

Le OU exclusif

Une variante du OU : le **OU exclusif** entre deux booléens A et B se définit par :

A XOR B est vrai si A vaut VRAI et B vaut FAUX ou si A vaut FAUX et B vaut VRAI. Remplir la table de vérité ci-contre.

A	B	A XOR B
0	0	
1	0	
0	1	
1	1	

→ Le OU exclusif se note **XOR**.

Les lois de De Morgan

Augustus De Morgan est un mathématicien et logicien britannique. Il est le fondateur avec Boole de la logique moderne (1842-1864). Il a notamment formulé les lois qui portent son nom.
Les opérateurs booléens suivent un certain nombre de lois. Parmi elles, les lois de De Morgan, qui peuvent être utiles pour simplifier des opérations booléennes complexes.

NON (A ET B) = (NON A) OU (NON B)
NON (A OU B) = (NON A) ET (NON B)

Applications :

1. A, B et C étant trois booléens, déterminer à la main la table de vérité de l'expression : (A OU B) ET C. Vérifier à l'aide de Python.

2. L'expression A OU (NON A ET B) OU (NON B) vaut-elle toujours VRAI ?

3. Vérifier que A XOR B= (A ET (NON B)) OU ((NON A) ET B)

4. Couleurs :

Chaque pixel d'une image matricielle est composé de 3 composantes rouge, verte, bleue. L'addition de ces couleurs permet de recréer toutes les couleurs (synthèse additive).

La valeur de chaque composante R, V et B est codée sur 8 bits.

Pour simplifier, nous considérerons que chaque composante est codée sur 1 bit.

Imaginons un dispositif dans lequel 3 lampes de couleurs rouge, verte, bleue sont dirigées vers le même endroit et peuvent être allumées ou éteintes.

- Justifier que l'on ne peut pas créer plus de 8 couleurs différentes et donner leurs codes binaires. (ces couleurs sont dans le tableau ci-dessous)

Couleur	R	V	B
Noir			
Bleu			
Vert			
Cyan			
Rouge			
Magenta			
Jaune			
Blanc			

- Le complément d'une couleur est obtenu en allumant les lampes éteintes et en éteignant les lampes allumées. Déterminer les couleurs complémentaires des huit couleurs précédentes.

- Quelle est la couleur obtenue en effectuant chacune des opérations suivantes ?

Bleu | Rouge *Magenta & Cyan* *Vert ^ Blanc*

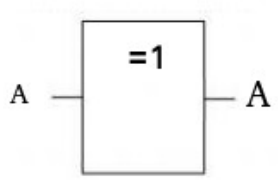
3. VERS L'ELECTRONIQUE ET LE CALCUL

A quelle opération mathématique vous fait penser l'opérateur ET ?
A quelle opération mathématique vous fait penser l'opérateur OU ?

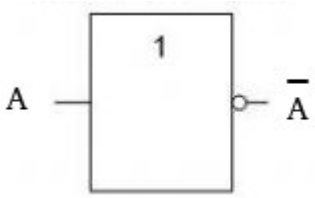
A chaque opération booléenne est associée un comportement électronique que l'on peut traduire en terme de tensions électriques : on parle de ***porte logique***.

Voici les représentations graphiques pour les portes logiques usuelles :

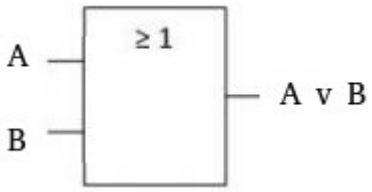
Identité



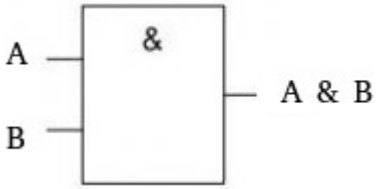
NON



OU



ET

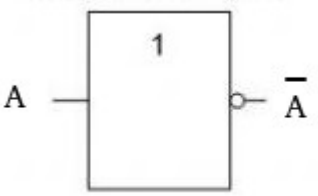


Les opérations logiques évoquées ci-dessus sont mises en œuvre en électronique sous forme de portes logiques. Ainsi les circuits électroniques calculent des fonctions logiques de l'algèbre de Boole. Pour chacun des opérateurs logiques évoquées ci-dessus (et d'autres) il existe donc des portes logiques appelés *porte ET*, *porte NON*, etc.

Les valeurs *vrai* et *faux* sont représentées par deux niveaux de tension, *haut* et *bas*. Un circuit de type *porte ET* dispose donc de deux entrées et une sortie et la valeur du niveau de tension en sortie dépend des niveaux de tension appliquées à chaque entrée, en respectant la table de vérité du ET.

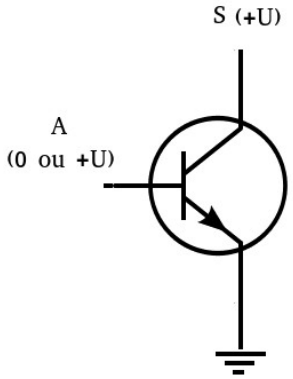
Par exemple, la porte NON est représentée par un transistor:

Porte logique NON :



- Si l'entrée A est à 0 V (*tension nulle*), le courant ne passe pas, alors la sortie S est à +U (*tension haute*).

Transitor :



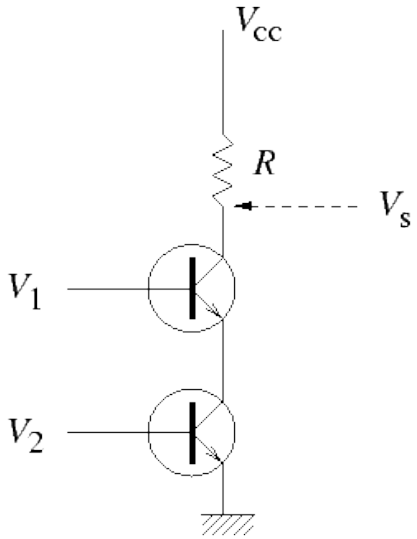
- Si l'entrée A est à +U (*tension haute*), le courant passe, alors la sortie est à 0 V (*tension nulle*).

On retrouve bien **S = NON(A)**

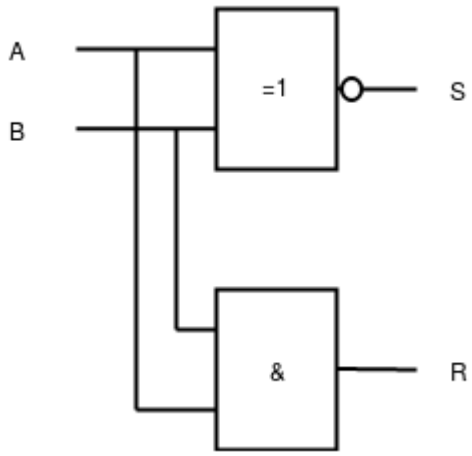
Les portes peuvent être connectées entre elles pour réaliser des circuits logiques et on peut ainsi réaliser des calculs.

Exemple :

Porte logique NON ET (ou NAND) avec 2 transistors en série.



Applications : Un circuit particulier, le demi-additionneur



Il est appelé ***demi-additionneur*** car il réalise l'addition de 2 bits (**A** et **B**), le résultat de cette somme est représentée par **S** et la retenue éventuelle par **R**.

Vérifiez, avec la table de vérité, que **S** et **R** correspondent bien aux valeurs de la somme et de la retenue sur 1 bit de **A** et **B**.

On rappelle que en binaire :

0 + 0 = 0 ; 0 + 1 = 1 ; 1 + 0 = 1 ; 1 + 1 = 10

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

→ Si l'on doit additionner 7 (111) avec 1 (001), que constate-t-on ?

A partir de ce circuit on peut en construire d'autres plus complexes permettant d'additionner des nombres de plusieurs bits. Voir sur cette page par exemple.

Et dans le même esprit, l'utilisation combinée des différentes portes de base permet de construire des circuits intégrés de plus en plus complexes, jusqu'au micro-processeur qui réalise les calculs au sein d'un ordinateur. Il suffit de trouver la bonne organisation. C'est un peu comme les Légo en somme... Vous pourrez trouver ici quelques compléments.