# Booléens - TD

## Objectifs

- Construire la table de vérité d'une expression booléenne relativement simple.
- Évaluer et construire des expressions booléennes en Python

## Démarche : construire une table de vérité

| `a` | `b` | `a or b` |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

1. compter le nombre d'entrées (= de lettres `a, b, c...`).

   Avec $n$ entrées, il y a une ligne pour les noms de colonnes et $2^n$ lignes pour les valeurs des bits.

   Toujours avec $n$ entrées, on a $n$ colonnes pour les entrées et au moins une pour le résultat.

2. Enumérer les valeurs des bits par **ordre croissant**. Par exemple pour 3 entrées `a, b, c` :

   - `0 0 0`
   - `0 0 1`
   - `...`
   - `1 1 1`

3. (optionnel) Si l'expression est composée de plusieurs morceaux, on peut ajouter une colonne intermédiaire par morceaux

4. Calculer les résultats de chaque colonne *résultat*.

Et voilà.

## Exercice 1

Construire la table de vérité de l'expression : `a OU (NON b)`

## Exercice 2

Construire la table de vérité de l'expression : `NON a et (b ou c)`

## Exercice 3

Construire la table de vérité de l'expression : `(a et NON b) ou (NON a et b)`

## Exercice 4

Construire la table de vérité de l'expression : `(a OU b) ET (a OU c)`

## Exercice 5

Trouver une expression équivalente à `a et b` construite uniquement à partir des opérateurs `NON` et `ou`

On vérifiera à l'aide d'une table de vérité.

## Exercice 6

Trouver une expression équivalente à `a OU b` construite uniquement à partir des opérateurs `NON` et `et`

On vérifiera à l'aide d'une table de vérité.

## Exercice 7 - simulateur

Vérifier chacun des résultats précédents dans le simulateur logique ci-dessous.

## Exercice 8 - Évaluer les booléens Python

Donner la valeur des expressions booléennes suivantes :

```
(1 > 2) and (3 < 5)
((4 - 7) >= 2) or (2 != 1 + 1)
a = 223
b = 455
a != (b // 2)
```

## Exercice 9 - parenthésage minimal

insérer le minimum de parenthèses dans les expressions suivantes pour les égalités soient correctes

```
2 + 3 * 5 + 4 == 21
5 + 2 * 3 + 4 == 25
4 + 5 * 2 + 3 == 29
```

## Exercice 10 - Clé de sécurité

La clé de vérification utilisée en France pour les numéros de sécurité sociale est égale à 97 moins le résidu modulo 97 du nombre formé par les autres chiffres : c'est-à-dire que la clé est l'unique entier entre 01 et 97 tel que la somme de la clé avec le reste du numéro fasse un entier divisible par 97.

Nous allons traduire ceci en Python

1. Voici le numéro de sécurité sociale de JP : `1 81 10 59 340 223`. Calculer sa clé de sécurité à l'aide d'une calculatrice.
2. Proposer une expression de la clé de sécurité.
3. Écrire cette expression en Python en supposant que le numéro est noté `numero`

   ```
   numero = 1811059340223
   cle = ???
   ```

4. Proposer une expression booléenne qui soit vraie si `cle` est bien la clé de sécurité de `numero` et fausse sinon.

Au passage, que signifie le numéro *1 81 10 59 . . .* ?

- Sexe : 1 c'est un homme, 2 si c'est une femme
- Année de naissance : 81, né en 1981
- Mois de naissance : 10, né en octobre
- Département de naissance : 59, dans le nord. . .
- Code commune Insee : 340 commune du nord donc 59340 qui correspond à Leffrinckoucke. Attention code insee != code postal.
- N° d'ordre de naissance : 223 ème personne née à Leffrinckoucke durant ce mois (j'ai inventé hein, n'allez pas chercher. . . )
- Clé de contrôle : voir plus haut.

source: justice.fr

## Exercice 11 - Demi additionneur binaire

### Représentation graphique

A chaque porte est associée une représentation graphique. Voici pour les portes ET et XOR :

- porte ET

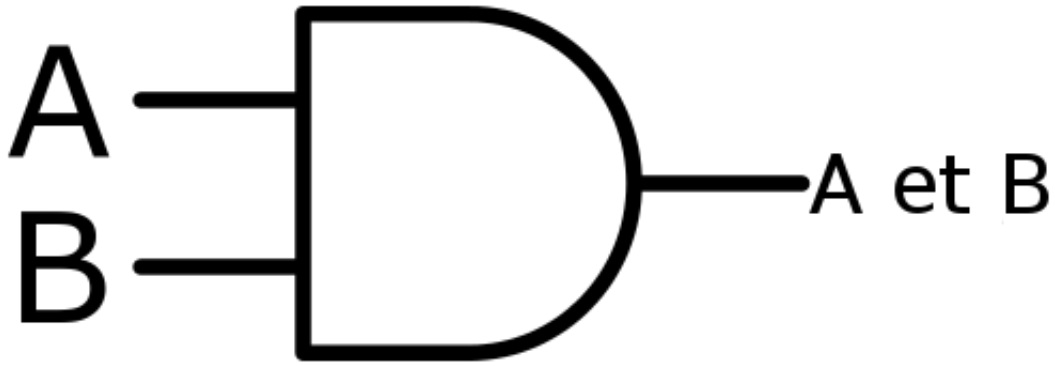


Figure 1: porte ET

- Porte XOR

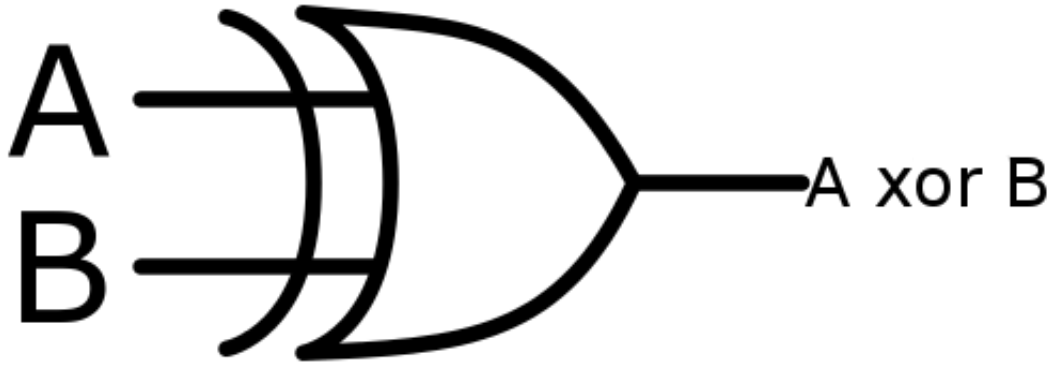


Figure 2: porte XOR

1. Rappeler les tables de vérité de ces deux portes.
2. On considère deux entiers $A$ et $B$ représentés sur un bit.
   - On les additionne et l'éventuelle retenue est perdue. Quels sont les résultats possibles ?

     Cette opération peut-elle être comparée à une des tables précédentes ?
   - On recommence, mais cette fois on décide de noter sur deux bits le résultat de la somme.

     Dans quel cas aura-t-on une retenue à écrire ? Quelle opération booléenne sur les bits permet d'obtenir ce résultat ?
3. Vérifier dans le simulateur de l'[exercice 7][#7]

### Portes logiques

Les opérations logiques évoquées ci-dessus sont mises en oeuvre en électronique sous forme de **portes logiques**.

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*.

## Exercice 12 - Demi additionneur

Un circuit de type *porte ET* dispose donc de deux entrées et une sortie.

La valeur du niveau de tension en sortie est obtenue avec la table de vérité du ET.

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

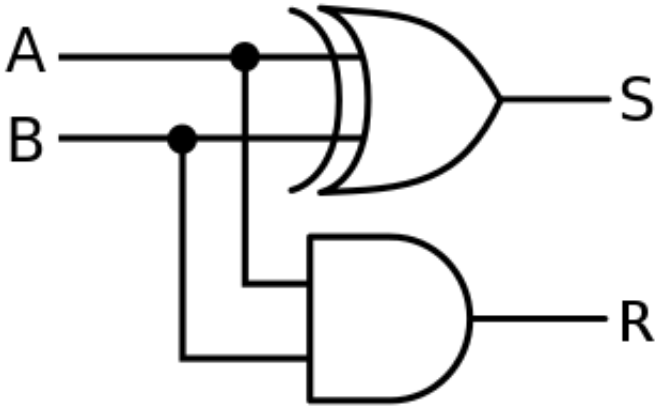


Figure 3: demi-additionneur

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

Construisez les tables de vérité de $S$ et $R$ et comparez à celle de l'addition de deux bits $A$ et $B$.

## Exercice 13 - Retrouver une expression booléenne

On considère la table de vérité de l'expression `Z` ci-dessous

| `x` | `Z(x)` |
|---|---|
| 0 | 0 |
| 1 | 0 |

Exprimer `Z` à l'aide des fonctions booléennes et, ou, non.

## Exercice 14 - Retrouver une expression booléenne

On considère la table de vérité de l'expression `U` ci-dessous

| `x` | `U(x)` |
|---|---|
| 0 | 1 |
| 1 | 1 |

Exprimer `U` à l'aide des fonctions booléennes et, ou, non.

## Exercice 15 - Programmer une table de vérité

Partons d'un exemple avec l'expression booléenne `Non (a ET Non b)`

On souhaite vérifier si cette expression peut s'écrire sous la forme `Non a OU b`.

Plusieurs approches sont possibles :

- démonstrations mathématique,
- table de vérité,

- utiliser un programme qui teste tous les cas.

Les deux derniers points sont équivalents d'un point de vue logique mais dans le second on fait faire les calculs à une machine.

**Démarche :**

1. on crée une fonction pour chaque expression booléenne
2. on crée une fonction qui prend ces expressions booléennes et teste chaque valeur possible des variables.

```
def exp1(a, b):
  return not (a and not b)

def exp2(a, b):
  return not a or b

def tester_egalite_2_variables(f, g):
  for a in (True, False):
    for b in (True, False):
      if f(a, b) != g(a, b):
        return False
  return True
```

   a. Lire attentivement le code de la fonction `tester_egalite_2_variables`.

   b. Les deux `return` ne sont pas indentés de la même manière. Expliquer.

3. On considère deux fonctions booléennes à trois entrées (`a, b, c`), écrire un programme python permettant de tester leur égalité.

def exp1(a, b, c): return (a and b) or (a and not b) or (a and c)

def exp2(a, b, c): return a

On pourra vérifier avec cet exemple :

**Expression 1 :**

`E1 = (a ET b) OU (a ET NON b) OU (a ET c)`

**Expression 2 :**

`E2 = a`

**Table de vérité de E1**

| a | b | c | NON b | a ET b | a ET NON b | a ET c | E1 | E2 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |