BOOLEENS

1. VARIABLES BOOLEENNES

Un booléen est un type de base, c'est même le plus simple puisqu'il prend deux valeurs : `TRUE` et `FALSE`. Ces deux valeurs sont facilement utilisables par les processeurs qui travaillent avec du binaire (le courant passe ou non). **TRUE** est d'ailleurs associé à **1** et **FALSE** à **0**.

On retrouve ces valeurs comme résultats de tests :

```
>>>a=7
>>>a<0
False
>>>a>0
True
```

Ce sont donc des booléens qui permettent de faire fonctionner les structures conditionnelles `if ... elif ... else ...` ou `while ...` : les blocs qui suivent les conditions sont réalisées si ces conditions sont à **True**.

2. FONCTIONS BOOLEENNES

Les fonctions booléennes de base sont `NOT`, `OR`, `AND`.

a. La fonction `NOT`

Cette fonction transforme **0** en **1** (**False** devient **True**), **1** en **0** (**True** devient **False**). Pour la fonction `NOT`, on a la table suivante :

A	NOT(A)
0	1
1	0

b. La fonction `OR`

Cet opérateur entre deux booléens permet de tester si l'un des deux booléens est vrai et donne dans ce cas vrai. On a la table suivante :

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

c. La fonction `AND`

Cet opérateur entre deux booléens permet de tester si les deux booléens sont vrais et donne dans ce cas vrai. On a la table suivante :

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

d. Les autres fonctions booléennes

D'autres opérateurs booléens existent mais s'obtiennent à partir des trois précédents. Exemple : l'opérateur `XOR` (OU exclusif) entre deux booléens renvoie vrai quand seulement l'un des booléens est vrai et faux sinon. Sa table est :

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

Si on fait la table de `(A OR B) AND (NOT(A) OR NOT(B))`, on obtient les mêmes résultats donc cette fonction correspond à `XOR`.

Ce principe de construction d'opérateurs plus complexes à partir de ces opérateurs booléens s'applique en électronique avec les circuits logiques.