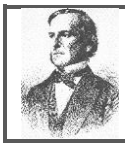
PREMIERE GENERALE

NUMERIQUE ET SCIENCES INFORMATIQUES

# FONCTIONS BOOLEENNES





**George Boole** (**1815-1864**), mathématicien anglais, est considéré créateur de la logique moderne. Son algèbre binaire, « **L'algèbre de Boole** », qu'il exprime en **1854** est utilisée de nos jours en électricité, en électronique et dans la mise de programmes informatiques

## 1 – FONCTIONS BOOLEENNES

En informatique, tous les **mécanismes de tests** sont basés sur la **logique booléenne** : Une expression est soit **vraie** (`True`) soit **fausse** (`False`). Les opérateurs de comparaison permettent de comparer deux valeurs et d'indiquer si la condition énoncée est vérifiée ou non (a est plus grand que b, a et b ont la même valeur, etc…). `True` correspond au niveau logique **1** et `False` au niveau logique **0**.

Une fonction booléenne est une fonction qui associe à une variable booléenne, « **Faux** » ou « **Vrai** » (`False` ou `True`, `0` ou `1`) à une ou plusieurs autres variables booléennes. Ces fonctions booléennes sont utilisées dans l'architecture des ordinateurs, dans les programmes informartiques, dans certains algorithmes...

Une fonction booléenne peut s'exprimer par des expressions utilisant des opérateurs booléens (non, ou, et ...) ou bien des tables de vérité.

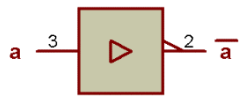
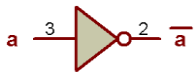
Ces fonctions peuvent être réalisées par des **instructions** dans des programmes informatiques ou bien encore réalisées physiquement par des portes **logiques contenues** dans des **circuits intégrés**.

# 2 – FONCTIONS BOOLEENNES ELEMENTAIRES

## 2.1 – Fontion NON (NOT)

| a | not(a) |
|---|---|
| False | |
| True | |

| a | not(a) |
|---|---|
| 0 | |
| 1 | |

| Porte logique | |
|---|---|
|  |  |



La fonction **non** (not) permet de **complémenter une variable booléenne**. Si la variable est "**Vrai**" la fonction **non** la passe à "**Faux**" et inversement si la variable est "**Faux**", la fonction **non** la passe à "**Vrai**".

`not(a) = ā`

**Exercice n°1**

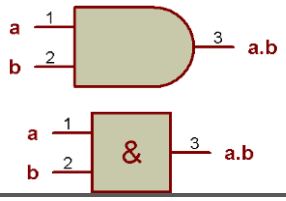
**Executer** les instructions suivantes permettant de vérifier la table de vérité ci-dessus :

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

## 2.2 – Fonction ET (AND)

| a | b | a and b |
|---|---|---|
| False | False | |
| False | True | |
| True | False | |
| True | True | |

| a | b | a and b |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

| Porte logique |
|---|
|  |



La fonction booléenne **et** (**and**) retourne un booléen qui est « **Vrai** » si **le booléen a et le booléen b sont** « **Vrai** ».

`a and b = a.b`

**Exercice n°2**

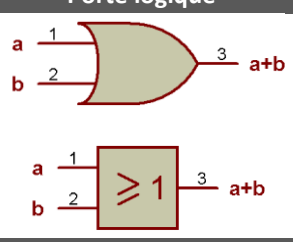
**Executer** les instructions suivantes permettant de vérifier la table de vérité ci-dessus :

```
>>> False and False
>>> False and True
>>> True and False
>>> True and True
```

## 2.3 – Fonction OR (OU)

| a | b | a or b |
|---|---|---|
| False | False | |
| False | True | |
| True | False | |
| True | True | |

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

| Porte logique |
|---|
|  |



La fonction booléenne **ou** (**or**) retourne un booléen qui est « **Vrai** » si **le booléen a est** « **Vrai** » **et/ou le booléen b est** « **Vrai** ».

`a or b = a + b`

**Exercice n°3**

**Executer** les instructions suivantes permettant de vérifier la table de vérité ci-dessus :

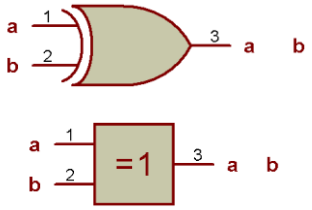
```
>>> False or False
>>> False or True
>>> True or False
>>> True or True
```

### 2.4 – Fonction XOR (OU-EXCLUSIF)

| a | b | a ^ b |
|---|---|---|
| False | False | |
| False | True | |
| True | False | |
| True | True | |

| a | b | a ^ b |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

**Porte logique**




La fonction booléenne **ou-exclusif** (**xor**) retourne un booléen qui est « Vrai » si **le booléen a est différent du booléen b**.

$$a \text{ ^ } b = a \oplus b$$

**Exercice n°4**

**Executer** les instructions suivantes permettant de vérifier la table de vérité ci-dessus.

```
>>> False ^ False
>>> False ^ True
>>> True ^ False
>>> True ^ True
```

## 3 – EVALUER UNE EXPRESSION BOOLEENNE

Comme dans une expression mathématique classique, certains opérateurs booléens sont prioritaires par rapport à d'autres et il est nécessaire de tenir compte des parenthèses. Les opérateurs booléens prioritaires sont **not** puis **and** et enfin **or** ou **xor(^)**.

**Exercice n°5**

**Vérifier** et **justifier** que les résultats des tests correspondent à ceux attendus.

```
>>> a = 1
>>> b = 2
>>> a == 1 and b < 3
>>> a == 1 and b == 3
>>> a == 1 or b > 3
>>> a < 1 and b > 3
```

Pour évaluer une expression booléenne comportant des variables booléennes, on peut dresser une table de vérité de chaque partie de l'expression en tenant compte des parenthèses et des priorités entre les expressions.

**Exemple** : `a or b and not a`

| a | b | not a | b and not a | a or b and not a |
|---|---|---|---|---|
| False | False | | | |
| False | True | | | |
| True | False | | | |
| True | True | | | |

Il est possible d'évaluer une expression comportant des variables booléennes, en remplaçant dans l'expression les variables par leur valeur. **Exemple** : Si **a** vaut `False` et **b** vaut `True`, que vaut l'expression `not a or b and a`

`not a or b and a` =
=
=
=

## Exercice n°6

1. **Compléter** la table de vérité de l'expression `not(a or not b) and b`. **Donner** les valeur de **a** et **b** permettant d'obtenir la valeur `True` pour l'expression précédente.

| a | b | | | | |
|---|---|---|---|---|---|
| False | False | | | | |
| False | True | | | | |
| True | False | | | | |
| True | True | | | | |

2. Si **a** vaut `False` et **b** vaut `True`, **donner** la valeur de l'expression `a and b or not a`.

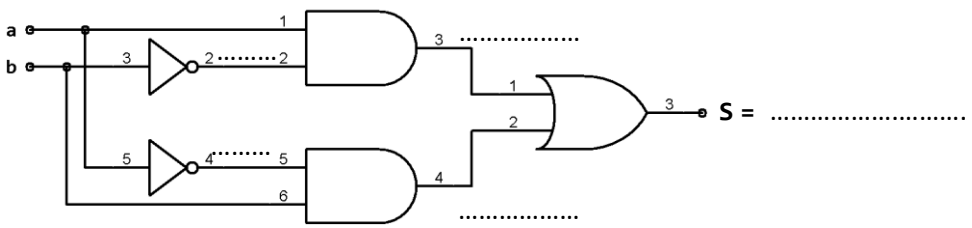
`a and b or not a` =
=
=
=

# 4 – CIRCUITS LOGIQUES

Dans un ordinateur, les opérations de calculs sont réalisés à de circuits logiques qui associent des portes logiques.

## Exercice n°7

1. **Compléter** les différentes sorties du circuit logique suivant. **Déduiser**-en l'équation de la sortie **S**.



## Exercice n°7

**2.** **Compléter** la table de vérité du circuit logique ci-dessus. En vous aidant des tables de vérité des opérateurs élémentaires, **indiquer** à quel opérateur logique élémentaire est équivalent ce circuit.

| a | b | | | | | |
|---|---|---|---|---|---|---|
| False | False | | | | | |
| False | True | | | | | |
| True | False | | | | | |
| True | True | | | | | |

## Exercice n°8

Un **additionneur 1 bit** est un circuit logique qui permet de faire l'addition de 2 bits.

Il possède en entrée :

- **a** : le premier bit
- **b** : le second bit
- **re** : la retenue entrante (éventuelle)

Il possède en sortie :

- **s** : la somme des 2 bits
- **rs** : la retenue sortante (éventuelle)


**1.** **Compléter** la table de vérité de l'additionneur 1 bit.

| a | b | re | S | rs |
|---|---|---|---|---|
| 0 | 0 | 0 | | |
| 0 | 0 | 1 | | |
| 0 | 1 | 0 | | |
| 0 | 1 | 1 | | |
| 1 | 0 | 0 | | |
| 1 | 0 | 1 | | |
| 1 | 1 | 0 | | |
| 1 | 1 | 1 | | |

A partir de cette table de vérité on peut déduire les équations de **S** et **re** suivantes :

```
S = (a ^ b) ^ re
rs = (re and (a ^ b)) or (a and b)
```

**2.** **Dessiner** le circuit logique permettant de réaliser cet additionneur 1 bit.