

Booléens, portes logiques et expressions booléennes

17 septembre 2026

Valeurs booléennes

Vrai / faux

Principe

un ordinateur ne manipule que des 0 et des 1 (électricité ou pas)

=> vus aussi comme des **valeurs de vérité**

▶ 0 = faux = False

▶ 1 = vrai = True

=> une **valeur booléenne** (George Boole, 1815-1864)

le plus petit type de donnée possible : **1 bit**

D'où ça vient ?

réponse à une **question fermée** (oui / non)

- ▶ l'interrupteur est-il allumé ?
- ▶ 3 est-il plus grand que 2 ?
- ▶ la porte est-elle ouverte ?

Deux notations, un seul objet

circuits	tables	Python
tension haute (5V ou 3.3V)	1	True
tension basse (0V)	0	False

dans ce cours : surtout 0 et 1

Opérateurs booléens et portes logiques

Opérateurs et portes

Opérateurs booléens

combiner des booléens : **NON**, **ET**, **OU**, + **OU exclusif**

exemple : « la porte est-elle fermée **ET** la lumière allumée ? »

Portes logiques

porte logique = composant électronique qui réalise un opérateur

- ▶ 1 ou 2 signaux en entrée
- ▶ 1 signal en sortie
- ▶ un **symbole** pour les schémas

un processeur moderne : plusieurs **milliards** de portes

Lire un schéma

- ▶ entrées à gauche, sortie à droite
- ▶ les traits = les **fils**, ils transportent 0 ou 1
- ▶ un fil peut se **diviser** (alimenter plusieurs portes)
- ▶ deux fils n'arrivent **jamais** sur la même entrée

Table de vérité

table de vérité = sortie pour **toutes** les combinaisons d'entrées

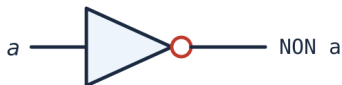
nombre de lignes : 2^n pour n entrées

- ▶ 1 entrée $\Rightarrow$ 2 lignes
- ▶ 2 entrées $\Rightarrow$ 4 lignes
- ▶ 3 entrées $\Rightarrow$ 8 lignes
- ▶ 4 entrées $\Rightarrow$ 16 lignes

Les portes de base

NON (NOT)

unaire : une seule entrée, on l'inverse



a	NON a
0	1
1	0

notations : NON a, $\bar{a}$, $\neg a$

le **petit rond** = « et on inverse »

ET (AND)

vaut 1 **si et seulement si** les deux entrées valent 1

« il faut le badge **et** le code »



notations : $a \wedge b$, $a \cdot b \Rightarrow$ comme une **multiplication**

a	b	a ET b
0	0	0
0	1	0
1	0	0
1	1	1

OU (OR)

vaut 1 dès qu'**au moins une** entrée vaut 1

attention : OU **inclusif** (les deux à 1 $\Rightarrow$ 1 aussi)



notations : $a \vee b$, $a + b$

a	b	a OU b
0	0	0
0	1	1
1	0	1
1	1	1

OU exclusif (XOR)

vaut 1 quand les entrées sont **différentes**

« fromage ou dessert », mais pas les deux



symbole du OU + **barre**

supplémentaire à l'arrière

notation : $a \oplus b$

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

XOR : deux façons de le voir

- ▶ **détecteur de différence**
- ▶ **addition binaire sur 1 bit**, dont on garde les unités
 - ▶ $0 + 0 = 0$
 - ▶ $0 + 1 = 1$
 - ▶ $1 + 0 = 1$
 - ▶ $1 + 1 = 1\underline{0} \Rightarrow$ on garde 0

à réutiliser plus loin (demi-additionneur)

Portes dérivées

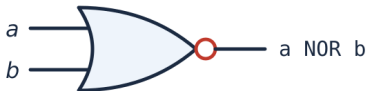
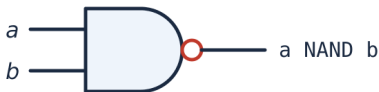
NAND et NOR

versions **niées** de ET et OU (plus simples à fabriquer)

▶ $a \text{ NAND } b = \text{NON } (a \text{ ET } b)$

▶ $a \text{ NOR } b = \text{NON } (a \text{ OU } b)$

symbole du ET / du OU + **petit rond**

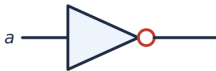


NAND et NOR : tables

a	b	a NAND b	a NOR b
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

culture : NAND est **universelle** (tout circuit se refait avec des NAND)

Récapitulatif des symboles



NON (NOT)

not a



ET (AND)

a and b



OU (OR)

a or b



OU exclusif (XOR)

a != b



NON-ET (NAND)

not (a and b)



NON-OU (NOR)

not (a or b)

Récapitulatif des tables de vérité

toutes les entrées à gauche, un opérateur par colonne

a	b	NON a	ET	OU	XOR	NAND	NOR
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

Comment lire ce tableau

- ▶ NON a ne dépend que de a (opérateur unaire)
- ▶ NAND = colonne ET **inversée**, NOR = colonne OU inversée
- ▶ OU et XOR ne diffèrent que sur la **dernière ligne** (les deux entrées à 1)
- ▶ ET : 1 fois sur 4 ; OU : 3 fois sur 4

=> un **aide-mémoire**, pas quelque chose à apprendre par cœur

Simulation

Simulateur

simulateur de circuits logiques (via **Capytale**)

- ▶ placer entrées, portes, sortie
- ▶ basculer les entrées, observer la sortie

attention : le simulateur **vérifie** une table de vérité, il ne la remplace pas

Combiner des portes

Méthode

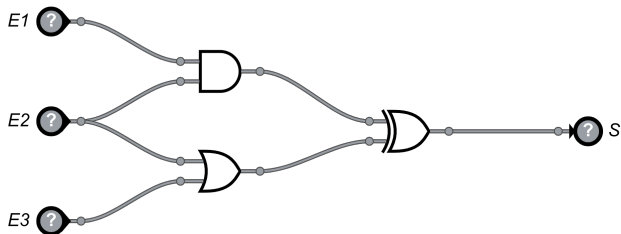
Dresser la table de vérité d'un circuit

1. compter les entrées $n \Rightarrow 2^n$ lignes
2. écrire **toutes** les combinaisons (= compter en binaire de 0 à $2^n - 1$)
3. une **colonne intermédiaire par porte**, dans l'ordre du signal
4. remplir colonne par colonne, en n'utilisant que ce qui est déjà rempli
5. la ou les dernières colonnes = la ou les sorties

$\Rightarrow$ on ne devine **jamais** la sortie, on avance porte par porte

Exemple

Le circuit



- ▶ ET sur E1,E2 $\Rightarrow$ A
- ▶ OU sur E2,E3 $\Rightarrow$ B
- ▶ XOR sur A,B $\Rightarrow$ S

E2 alimente deux portes : le fil se divise

Sa table de vérité

E1	E2	E3	A = E1 ET E2	B = E2 OU E3	S = A XOR B
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	1	0

=> sortie à 1 dans 5 cas sur 8

Exercices



Même travail (avec les colonnes intermédiaires) pour les circuits 2 et 3 de la feuille Exos portes logiques.

Expressions booléennes

Écriture

Du schéma à l'expression

expression booléenne = formule combinant des variables avec NON, ET, OU, XOR

le circuit 1 :

$$S = (E_1 \text{ ET } E_2) \text{ XOR } (E_2 \text{ OU } E_3)$$

deux écritures du **même** objet

- ▶ schéma $\Rightarrow$ expression : partir de la sortie, remonter
- ▶ expression $\Rightarrow$ schéma : une porte par opérateur, en commençant par les parenthèses internes

Exercices : schéma $\Leftrightarrow$ expression



Feuille Exos expressions booléennes :

- ▶ **exercice 1** : du schéma vers l'expression (3 circuits)
- ▶ **exercice 2** : de l'expression vers le schéma (3 expressions)

méthode : nommer la sortie de chaque porte (A, B, C), puis remplacer de proche en proche

Priorités

comme en maths ($2 + 3 \times 4 = 14$)

1. parenthèses
2. NON
3. ET
4. OU (et XOR)

à priorité égale : de gauche à droite

moyen de retenir : ET = multiplication, OU = addition

Attention aux parenthèses

- ▶ $a \text{ OU } b \text{ ET } c$ se lit $a \text{ OU } (b \text{ ET } c)$
- ▶ $\text{NON } a \text{ ET } b$ se lit $(\text{NON } a) \text{ ET } b$
- ▶ ... et **pas** $\text{NON } (a \text{ ET } b)$, qui est un NAND !

a	b	$(\text{NON } a) \text{ ET } b$	$\text{NON } (a \text{ ET } b)$
0	0	0	1
0	1	1	1
1	0	0	1
1	1	0	0

en cas de doute $\Rightarrow$ parenthéser

Tables et équivalences

Table d'une expression

même méthode : une colonne par opérateur, dans l'ordre des priorités

$$S = (a \text{ ET } \bar{b}) \text{ OU } (\bar{a} \text{ ET } b)$$

a	b	$\bar{a}$	$\bar{b}$	$a \cdot \bar{b}$	$\bar{a} \cdot b$	S
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

$\Rightarrow$ c'est la table du **XOR** : on l'a reconstruit avec NON, ET, OU

Exercice : tables d'expressions



Exercice 3 de la feuille Exos expressions booléennes : quatre expressions, table de vérité complète avec les colonnes intermédiaires

les deux premières sont chacune **une porte** du cours : lesquelles ?

Équivalence

deux expressions sont **équivalentes** si elles ont la **même table de vérité**

seul critère : ni l'écriture, ni le nombre de portes ne comptent

Règles utiles (pour info)



Non exigible

au programme : **dresser la table de vérité d'une expression**,
et passer du schéma à l'expression

le **calcul** sur les expressions (règles ci-dessous, simplification, De Morgan) n'est **pas exigible**, ce ne sera pas demandé en devoir

Règles utiles : la liste

simplifier une expression = fabriquer le circuit avec moins de portes

règle	écriture
élément neutre	$a \text{ ET } 1 = a ; a \text{ OU } 0 = a$
élément absorbant	$a \text{ ET } 0 = 0 ; a \text{ OU } 1 = 1$
idempotence	$a \text{ ET } a = a ; a \text{ OU } a = a$
complément	$a \text{ ET } (\text{NON } a) = 0 ; a \text{ OU } (\text{NON } a) = 1$
double négation	$\text{NON } (\text{NON } a) = a$
De Morgan	$\text{NON } (a \text{ ET } b) = (\text{NON } a) \text{ OU } (\text{NON } b)$ $\text{NON } (a \text{ OU } b) = (\text{NON } a) \text{ ET } (\text{NON } b)$

Application : le demi-additionneur

Le problème

Additionner deux bits

table d'addition binaire (chapitre B.1) :

- ▶ $0 + 0 = 0$
- ▶ $0 + 1 = 1$
- ▶ $1 + 0 = 1$
- ▶ $1 + 1 = 10$

dernier cas : ça ne tient plus sur un bit

=> il faut **deux sorties**

- ▶ la **somme** S (les unités)
- ▶ la **retenue** R (à reporter à gauche)

La table de vérité

E1	E2	résultat	R	S
0	0	00	0	0
0	1	01	0	1
1	0	01	0	1
1	1	10	1	0

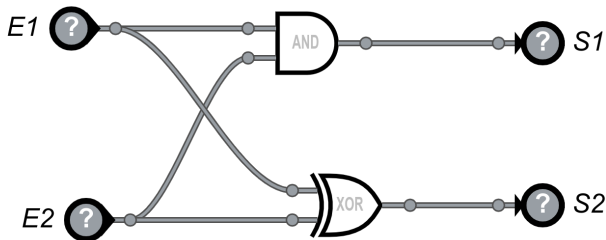
colonne par colonne :

- ▶ $R = 1$ seulement si les deux entrées sont à 1 $\Rightarrow$ **ET**
- ▶ $S = 1$ si les entrées diffèrent $\Rightarrow$ **XOR**

Le circuit

Deux portes suffisent

$$R = E_1 \text{ ET } E_2 \quad S = E_1 \text{ XOR } E_2$$



les deux entrées alimentent les deux portes : les fils se divisent

Demi-additionneur : exercice



Construire ce circuit dans le simulateur, vérifier les 4 lignes de sa table.

Pourquoi « demi » ?

sur plusieurs bits, chaque colonne additionne **trois** bits : les deux bits du rang + **la retenue précédente**

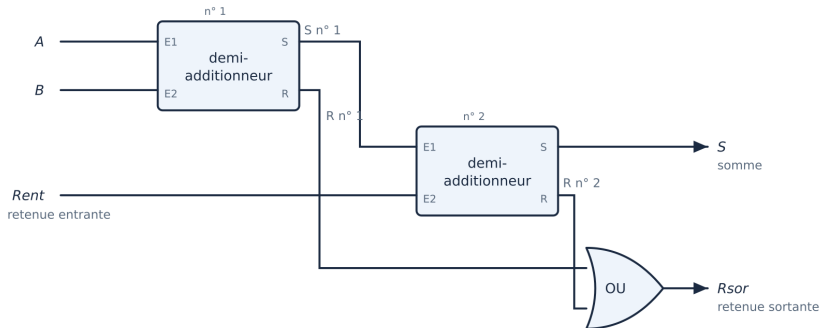
```

1 1 1 1      <- retenues
  1 0 1 1
+ 0 1 1 1
-----
1 0 0 1 0
    
```

▶ 3 bits en entrée => **additionneur complet** (*full adder*)

▶ = 2 demi-additionneurs + 1 porte OU

L'additionneur complet



- ▶ demi-add. n° 1 : $A + B$
- ▶ demi-add. n° 2 : ajoute la retenue entrante $\Rightarrow$ somme finale S
- ▶ porte OU : rassemble les deux retenues

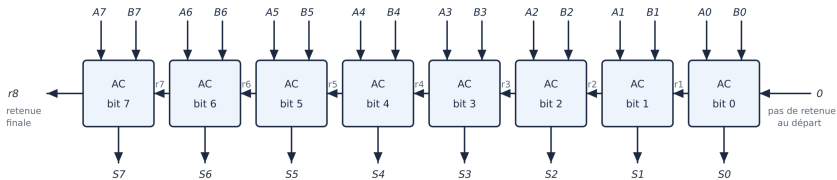
Pourquoi un simple OU ?

les deux retenues ne valent **jamais** 1 en même temps

- ▶ si $A = B = 1$, la somme du $n^{\circ} 1$ vaut 0
- ▶ donc le $n^{\circ} 2$ ne peut plus produire de retenue

$\Rightarrow$ le cas « $1 + 1$ » ne se présente jamais à l'entrée du OU

Additionner deux octets



AC = additionneur complet = 2 demi-additionneurs + 1 porte OU ; r₁ à r₇ = retenues intermédiaires

huit additionneurs complets, un par colonne

La chaîne des retenues

- ▶ retenue sortante d'un étage $\Rightarrow$ retenue entrante de l'étage de **gauche**
- ▶ bit 0 : pas de retenue entrante (entrée à 0)
- ▶ retenue finale $r_8 = 9\text{e bit} \Rightarrow$ **dépassement de capacité** si elle vaut 1

$\Rightarrow$ additionneur à **propagation de retenue** : chaque étage attend son voisin de droite

Exercice



Exercice 4 de la feuille Exos expressions booléennes : compter les portes de l'additionneur 8 bits, et y poser une addition qui déborde

Au final

8, 32, 64 additionneurs bout à bout $\Rightarrow$ l'unité de calcul du processeur

$\Rightarrow$ l'addition est entièrement ramenée à de la **logique booléenne**