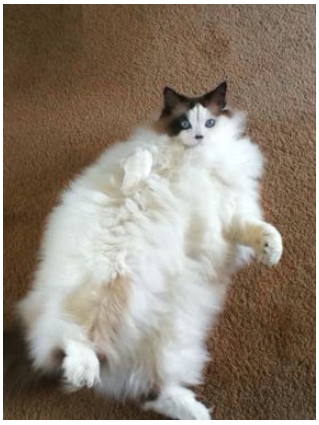
8 Les booléens

Extrait du programme

Thème : Types et valeurs de base



Un Chat boule, hein !!! hein ? !!

Contenus :

Valeurs booléennes : 0, 1.
Opérateurs booléens : and, or, not.
Expressions booléennes.

Capacités attendus :

Dresser la table d'une expression booléenne.

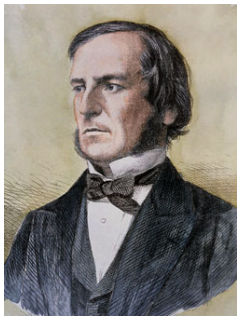
Commentaires :

Le ou exclusif (xor) est évoqué.
Quelques applications directes comme l'addition binaire sont présentées.
L'attention des élèves est attirée sur le caractère séquentiel de certains opérateurs booléens.

Source de tous les schémas présents dans ce cours :
NSI, Numérique & Sciences Informatique, Mickaël Barraud, pdf

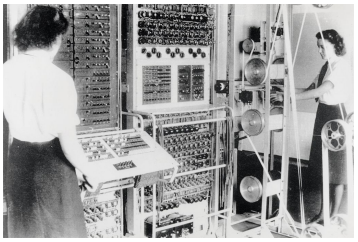
Vous pouvez retrouver le cours complet à l'adresse suivante :
`https://github.com/NaturelEtChaud/NSI-Premiere/tree/main/08%20Les%20bool%C3%A9enss`

I. Les exposés en lien avec ce cours



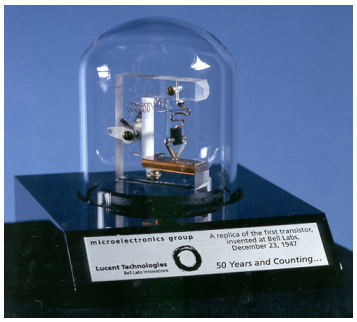
George Boole
(2 novembre 1815-8 décembre 1864)

George Boole est un logicien, mathématicien et philosophe britannique. Il est le créateur de la logique moderne, fondée sur une structure algébrique et sémantique, que l'on appelle algèbre de Boole en son honneur. Il s'agit d'une algèbre binaire, dite booléenne, n'acceptant que deux valeurs numériques : 0 et 1. Cette algèbre aura de nombreuses applications en téléphonie et en informatique.



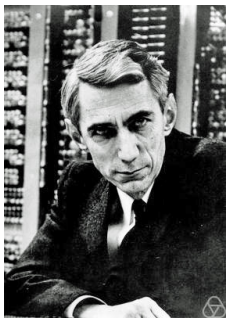
Le Colossus Mark II
(juin 1944-détruit peu après la fin de la guerre)

Colossus est une série de calculateurs électroniques fondé sur le système binaire. Le premier, Colossus Mark 1, est construit en l'espace de onze mois et opérationnel en décembre 1943, par une équipe dirigée par Thomas Flowers et installé près de Londres, à Bletchley Park : constitué de 1 500, puis 2 400 tubes à vide, il accomplissait 5 000 opérations par seconde. Il était utilisé pendant la Seconde Guerre mondiale pour la cryptanalyse .



Une réplique du premier transistor
(23 décembre 1947)

A la suite des travaux sur les semi-conducteurs, le transistor a été inventé le 23 décembre 1947 par les Américains John Bardeen, William Shockley et Walter Brattain, chercheurs des Laboratoires Bell. Ces chercheurs ont reçu pour cette invention le prix Nobel de physique en 1956.
Sources Wikipédia.



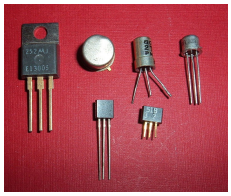
Claude Shannon
(30 avril 1916 - 24 février 2001)

Claude Elwood Shannon (30 avril 1916 à Petoskey, Michigan - 24 février 2001 à Medford, Massachusetts) est un ingénieur en génie électrique et mathématicien américain. Il est l'un des pères, si ce n'est le père fondateur, de la théorie de l'information.

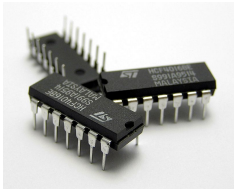
II. Le transistor

Le transistor est une sorte d'interrupteur qui permet, ou non, de laisser passer le courant, donc aboutissant à deux états possibles : 0 ou 1. C'est donc naturellement qu'avec ce nouvel outil, les pionniers de l'informatique et de l'électronique ont conçu des machines travaillant uniquement avec des nombres binaires [1].

Les premiers transistors étaient des transistors à lampe. Très encombrant et très fragiles, mais toujours utilisés par exemple dans des amplis pour guitare car le son est plus « chaud ».



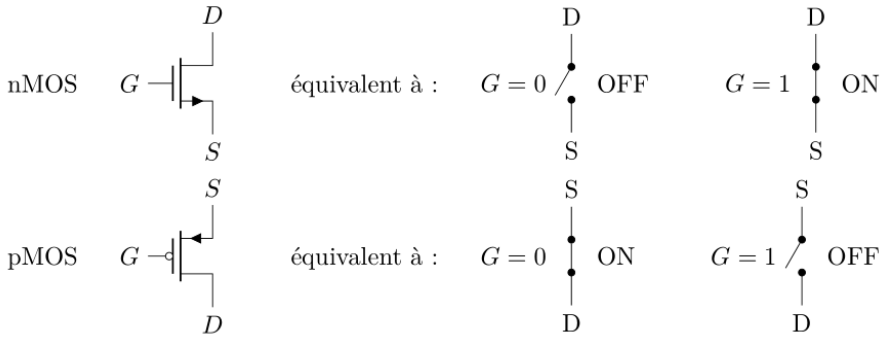
Les transistors ont ensuite étaient miniaturisés pour devenir un simple composant électronique.



En 1958, l'Américain Jack Kilby invente le premier circuit intégré, jetant ainsi les bases du matériel informatique moderne. Ces puces en silicium contenant plusieurs transistors interconnectés en circuits microscopiques dans un même bloc, permettaient la réalisation de mémoires, ainsi que d'unités logiques et arithmétiques. Ce concept révolutionnaire concentrait dans un volume incroyablement réduit, un maximum de fonctions logiques. Cette découverte a valu à Kilby un prix Nobel de physique en 2000.

Enfin, le microprocesseur est le composant électronique le plus important d'un ordinateur. Depuis le premier microprocesseur, le Intel 4004, créé en 1971, le nombre transistors devient de plus en plus impressionnant. Ainsi, le Core i7 Haswell a atteint 2,6 milliards de transistors en 2014.

Le fonctionnement d'un transistor est simple. C'est un interrupteur commandé par un courant électrique. Si on passe ou non un courant, l'interrupteur s'ouvre ou se ferme. Il existe deux types de transistor, le nMOS et le pMOS, représentés par les schémas ci-dessous.
Pour les transistors nMOS, si on ne passe pas de courant l'interrupteur est ouvert, si on passe un courant l'interrupteur est fermé. Par défaut l'interrupteur est donc ouvert ; l'information ne circule pas.
Pour les transistors pMOS, c'est exactement le contraire, si on ne passe pas de courant l'interrupteur est fermé, si on passe un courant l'interrupteur est ouvert. Par défaut l'interrupteur est donc fermé ; l'information circule.



1. Même si on vu dans le chapitre 3 qu'il pouvait y avoir des exceptions, comme par exemple avec le Setun russe.

III. Les booléens

Comme le transistor ne peut avoir que deux états ouvert ou fermé, les pionniers de l'informatique, comme Claude Shannon, se sont intéressé à l'algèbre de Boole qui n'utilise que deux nombres, le 0 et le 1.
Dans cet esprit, le type de variable **booléen** ne peut avoir que deux états **faux** ou **vrai**. Le faux est associé à 0 et le vrai, associé à 1.

Dans le langage Python, les variables de type `bool` peuvent avoir deux états `False` ou `True`.

Dans le langage Python, tout ce qui n'est pas Faux (ou `0` ou vide) est Vrai.

- Attention petit scarabée, et je sais que tu adore ça, il y a une petite pythonnerie : tout ce qui n'est pas faux est vrai.
- Euh, sauf votre respect, vous me prenez pour un imbécile ou quoi ?
- Non pas du tout. Voyons le petit code suivant, que je te conseille d'écrire sur ton ordinateur pour l'essayer :

```
1 test = # à toi de compléter, mon petit scarabée ...
2 if test :
3     print('Le test est vrai')
4 else :
5     print('Le test est faux')
```

- Ok, c'est fait.
- Très bien. Maintenant complète la ligne 1 avec les différentes propositions suivantes :

```
1 test = True
```

Que remarques-tu ?
- .....

```
1 test = False
```

- Que remarques-tu ?
- .....

```
1 test = 0
```

- Que remarques-tu ?
- .....

```
1 test = None #None est le vide en Python
```

- Que remarques-tu ?
- .....

```
1 test = (1<=5)
```

- Que remarques-tu ?
- .....

```
1 test = (1<=-5)
```

- *Que remarques-tu ?*

- .....

```
1 test = 'un texte, juste comme ça'
```

- *Que remarques-tu ?*

- .....

```
1 test = (1, 2, 3, 4, 8)
```

- *Que remarques-tu ?*

- .....

C'est quand même bizarre ce que l'on peut faire avec ce langage Python !

IV. Opérateurs booléens

Il n'y a que deux valeurs booléennes, donc pour décrire n'importe quelle fonction ou opérateur booléen, il suffit de dresser son tableau de valeurs donnant toutes les possibilités. On appelle cette table une **table de vérité**.

a) La négation

La **négation** permet d'échanger le faux et le vrai.
Le circuit combinatoire correspondant est une « porte non ». Elle est obtenue avec deux transistors, un nMOS et un pMOS.

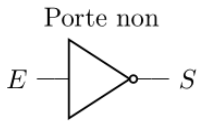
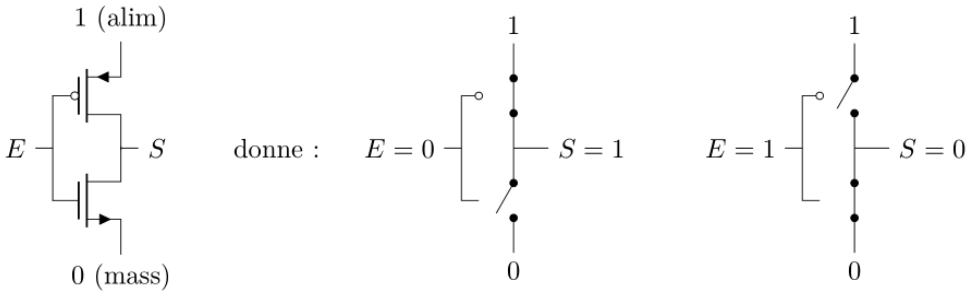


Table de vérité :

E	S
0	1
1	0



En Python, la négation d'un booléen est obtenue avec la commande `not`.

La table de vérité est alors la suivante :

`a`	`False`	`True`
`not a`	`True`	`False`

On peut effectuer le programme suivant qui permet d'obtenir la table de vérité de la négation.

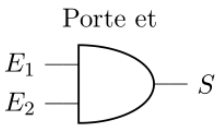
```python
# -*- coding: utf-8 -*-

def negation(a):
    return not a

def tableVerite():
    print('-'*30) #une ligne de  30 symboles -
    tupleBool = (False,True) #un tuple contenant les deux possibilites

    for a in tupleBool:
        print("La negation de",a,"est",negation(a))

    print('-'*30) #une ligne de  30 symboles -
```

Python/negation.py

On obtient dans la console, le résultat suivant :

```
>>tableVerite()
------------------
La negation de False est True
La negation de True est False
------------------
```

b) Et



`S = E1 & E2`

E_1	E_2	S
0	0	0
0	1	0
1	0	0
1	1	1

L'opérateur **et** renvoie vrai si et seulement si les deux booléens sont eux aussi également vrai.
Le circuit combinatoire correspondant est une « porte et ».

En Python, le ET est obtenue avec la commande `and`.

La table de vérité est alors la suivante :

a	False	True	False	True
b	False	False	True	True
a and b	False	False	False	True

On peut effectuer le programme suivant qui permet d'obtenir la table de vérité de la négation.

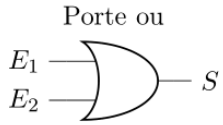
```python
# -*- coding: utf-8 -*-

def ET(a,b):
    return (a and b)

def tableVerite():
    print('-'*40) #une ligne de  40 symboles -
    tupleBool = (False,True) #un tuple contenant les deux possibilites

    for a in tupleBool:
        for b in tupleBool :
            print(a,"\t et \t",b,"\t = \t",ET(a,b))

    print('-'*40) #une ligne de  40 symboles -
```

Python/et.py

On obtient dans la console, le résultat suivant :

```
>>tableVerite()
----------------------------------------
False et False = False
False et True = False
True et False = False
True et True = True
----------------------------------------
```

c) Ou




```
S = E1 | E2
```

E_1	E_2	S
0	0	0
0	1	1
1	0	1
1	1	1

L'opérateur **ou** renvoie vrai si l'un ou l'autre ou les deux booléens sont vrai.
Le circuit combinatoire correspondant est une « porte ou ».

En Python, le OU est obtenue avec la commande `or`.

La table de vérité est alors la suivante :

a	False	True	False	True
b	False	False	True	True
a or b	False	True	True	True

On peut effectuer le programme suivant qui permet d'obtenir la table de vérité de la négation.

```python
# -*- coding: utf-8 -*-

def OU(a,b):
    return (a or b)

def tableVerite():
    print('-'*40) #une ligne de  40 symboles -
    tupleBool = (False,True) #un tuple contenant les deux possibilites

    for a in tupleBool:
        for b in tupleBool :
            print(a,"\t ou \t",b,"\t = \t",OU(a,b))

    print('-'*40) #une ligne de  40 symboles -
```

Python/ou.py

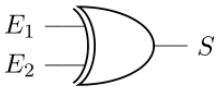
On obtient dans la console, le résultat suivant :

```
>>tableVerite()
--------------------
False ou False = False
False ou True = True
True ou False = True
True ou True = True
--------------------
```

d) Ou exclusif

L'opérateur **ou exclusif** renvoie vrai uniquement si l'un ou l'autre des deux booléens est vrai [2].
Le circuit combinatoire correspondant est une « porte ou exclusif » en anglais, on parle de « XOR ».

Porte ou exclusif




```
S = E1 ^ E2
```

E_1	E_2	S
0	0	0
0	1	1
1	0	1
1	1	0

X-OR, le shérif, shérif de l'espace ...
`https://www.youtube.com/watch?time_continue=73&v=-l3EGGfxIz0&feature=emb_logo`

En Python, le OU exclusif est obtenue avec la commande ... ben en fait il n'y en a pas.

La table de vérité est alors la suivante :

`a`	`False`	`True`	`False`	`True`
`b`	`False`	`False`	`True`	`True`
`a xor b`	`False`	`True`	`True`	`False`

Exercice 1
Écrire en Python, une fonction `xor()` prenant en arguments 2 booléens `a` et `b` et qui renvoie le booléen correspondant à `a xor b`.
Vous l'utiliserez ensuite pour obtenir la table de vérité de l'opérateur OU Exclusif.

e) D'autres opérateurs

Il existe d'autres opérateurs qui peuvent parfois être utile.

• Porte NOR : NOR signifie NOT OR, autrement dit, `a NOR b` = `NOT (a OR b)`.

Exercice 2
Établir la table de vérité de la porte NOR :

`a`	`False`	`True`	`False`	`True`
`b`	`False`	`False`	`True`	`True`
`a or b`	.....	.....	.....	.....
`a NOR b = not(a or b)`	.....	.....	.....	.....

• Porte NAND : NAND signifie NOT AND, autrement dit, `a NAND b` = `NOT (a AND b)`.

Exercice 3
Établir la table de vérité de la porte NAND :

`a`	`False`	`True`	`False`	`True`
`b`	`False`	`False`	`True`	`True`
`a and b`	.....	.....	.....	.....
`a NAND b = not(a and b)`	.....	.....	.....	.....

2. C'est un peu comme au restaurant quand on voit *fromage ou dessert*, c'est l'un ou l'autre, mais pas les deux. Alors que chez moi, quand j'invite, c'est le ou inclusif, c'est donc fromage ou dessert ou les deux !

• Porte XNOR : XNOR signifie NOT XOR, autrement dit, `a XNOR b` = `NOT (a XOR b)`.

Exercice 4
Établir la table de vérité de la porte XNOR :

a	False	True	False	True
b	False	False	True	True
a xor b	.....	.....	.....	.....
a XNOR b = not(a xor b)	.....	.....	.....	.....

f) D'autres exercices

Exercice 5 (Loi de Morgan)
Montrer que `not (a or b)` est équivalent à `(not a) and (not b)`.

Exercice 6 (Loi de Morgan)
Montrer que `not (a and b)` est équivalent à `(not a) or (not b)`.

Exercice 7 (xor)
Vérifier l'égalité : `a xor b = (a and not b) or (not a and b)`.

Exercice 8 (nand)
Dans chacun des cas, écrire la table de vérité puis trouver à l'opérateur correspondant :

1. `nand(a,a)` ;
2. `nand(nand(a,b),nand(a,b))` ;
3. `nand(nand(a,a),nand(b,b))`.

Exercice 9 (nor)
Dans chacun des cas, écrire la table de vérité puis trouver à l'opérateur correspondant :

1. `nor(a,a)` ;
2. `nor(nor(a,b),nor(a,b))` ;
3. `nor(nor(a,a),nor(b,b))`.

V. Caractère séquentiel des opérateurs logiques

`a and b` et `b and a` n'ont pas la même signification. Pour évaluer `a and b` , Python évalue d'abord `a` :

- Si `a` est `False`, comme quelque soit `b` la réponse sera `False`, pour gagner du temps `b` n'est pas évalué.
  Ce peut être très pratique, par exemple quand on veut évaluer une valeur d'un tableau :
  `if i < len(T) and T[i] > 0` : si l'indice `i` sort du tableau, `T[i]` ne sera pas évalué.
  Alors que `if T[i] > 0 and i < len(T)` pourra poser un problème si l'indice `i` sort du tableau. (`out of range`)

- Si `a` est `True`, `b` détient la réponse : c'est sa valeur qui est renvoyée (sans être testée).
  Cela permet un autre raccourci d'écriture :
  `variable = (0 <= i < len(T)) and T[i]` vaut `False` si `i` sort du tableau et la valeur de `T[i]` sinon.

Il en est de même pour `a or b`, si `a` est `True`, l'expression prend la valeur de `a` sinon elle prend la valeur de `b`.

On peut ainsi obtenir des instructions très courtes, mais plutôt à éviter :

```
1 parite = x % 2 == 0 and "pair" or "impair"
```

Exercice 10
Reprenons la fonction `ET` codée au début de ce cours :

```
1 def ET(a,b):
2     return a and b
```

Que renvoie :
- `ET(10,25)` .....
- `ET(25,10)` .....
- Pourquoi ? .....
  .....
  .....
  .....

- Facile, monsieur, c'est parce que ce n'est pas faux !
- C'est pas faux.

Livre II, épisode 79, La botte secrète II :

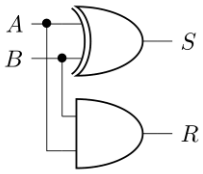
VI. Application au calcul binaire

Nous allons écrire ici 1 pour `True` et 0 pour `False` et on va réécrire les opérations binaires avec des formules logiques.

a) Demi additionneur (half adder)

Posons les additions à un bit `a` + `b`, avec `s` le résultat de la somme et `r` la retenue :

a	+	b	r	s
0	+	0	...	...
0	+	1	...	...
1	+	0	...	...
1	+	1	...	...



Autrement dit :

- `r = a ..... b`
- `s = a ..... b`

Décidément, ce cours s'adresse vraiment aux personnes de plus de 40 ans... Si vous en avez autour de vous demandez leur...

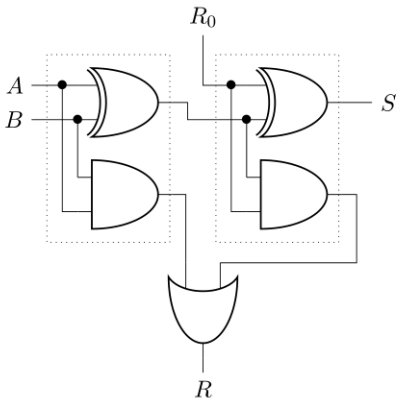
b) Plein additionneur (full adder)

Pour additionner deux entiers écrits en binaire, il faut aussi tenir compte de la retenue précédente.

	0	1	1	
	1	0	1	1
+	0	0	1	1
	1	1	1	0

On pose alors les additions à un bit `a` + `b` + r_0, où r_0 est la retenue du calcul précédent. `s` est toujours le résultat de la somme et `r` la retenue :

a	+	b	+	r_0	r	s
0	+	0	+	0	...	...
0	+	1	+	0	...	...
1	+	0	+	0	...	...
1	+	1	+	0	...	...
0	+	0	+	1	...	...
0	+	1	+	1	...	...
1	+	0	+	1	...	...
1	+	1	+	1	...	...

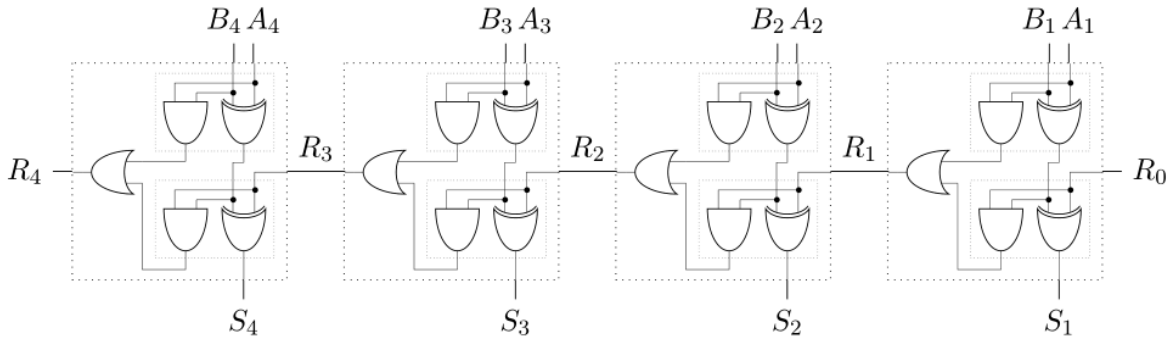


Autrement dit [3] :

- `r = (a and b) or (`r_0 `and (a xor b))`
- `s = a xor b xor` r_0

Exercice 11
Vérifier à la main ou avec un programme que `r = (a and b) or (`r_0 `and (a xor b))` et `s = a xor b xor` r_0.

Enfin, pour additionner des nombres binaires, il suffit de mettre plusieurs *plein additionneurs* en cascade :



3. Je donne la réponse directement, tellement elle est évidente ...

VII. Correction des exercices

Correction de l'exercice 1

```python
# -*- coding: utf-8 -*-

def xor(a,b):
    if a:
        return not b
    else :
        return b

def tableVerite():
    print('-'*40) #une ligne de  40 symboles -
    tupleBool = (False,True) #un tuple contenant les deux possibilites

    for a in tupleBool:
        for b in tupleBool :
            print(a,"\t ou ex  \t",b,"\t = \t",xor(a,b))

    print('-'*40) #une ligne de  40 symboles -
```

Python/ouEx.py

On obtient dans la console, le résultat suivant :

```
>>tableVerite()
--------------------
False ou ex False = False
False ou ex True = True
True ou ex False = True
True ou ex True = False
--------------------
```

Correction de l'exercice 2

a	False	True	False	True
b	False	False	True	True
a or b	False	True	True	True
a XOR b = not(a or b)	True	False	False	False

Correction de l'exercice 3

a	False	True	False	True
b	False	False	True	True
a and b	False	False	False	True
a NAND b = not(a and b)	True	True	True	False

Correction de l'exercice 4

a	False	True	False	True
b	False	False	True	True
a xor b	False	True	True	False
a XNOR b = not(a xor b)	True	False	False	True

Correction de l'exercice 5 (Loi de Morgan)
`not (a or b)` = `(not a) and (not b)`.

Correction de l'exercice 6 (Loi de Morgan)
`not (a and b)` = `(not a) or (not b)`.

Correction de l'exercice 7 (xor)
`a xor b = (a and not b) or (not a and b)`.

Correction de l'exercice 8 (nand)

1. `nand(a,a)` = `not a`;
2. `nand(nand(a,b),nand(a,b))` = `a and b`;
3. `nand(nand(a,a),nand(b,b))` = `a or b`.

Correction de l'exercice 9 (nor)

1. `nor(a,a)` = `not a`;
2. `nor(nor(a,b),nor(a,b))` = `a or b`;
3. `nor(nor(a,a),nor(b,b))` = `a and b`.