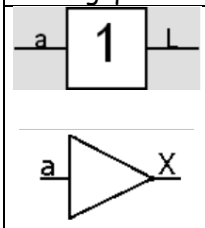
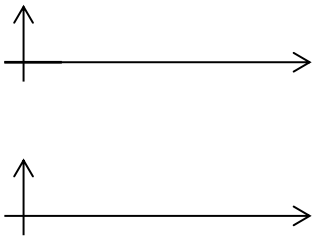
FONCTIONS LOGIQUES ELEMENTAIRES

Les fonctions logiques (ou portes logiques, opérateurs logiques) permettent d'associer une variable de sortie à une ou plusieurs variables d'entrée.
Quatre fonctions logiques élémentaires permettent de définir toutes les fonctions logiques complexes plus le ou exclusif qui est très utilisé pour le codage.
On différencie les opérateurs :

- booléens qui calculent des expressions avec opérandes booléens True et False
- logiques qui calculent des expressions sur les 0 et 1 pour chaque bit des opérandes entiers

1. La fonction «OUI» (ou EGALITE) :

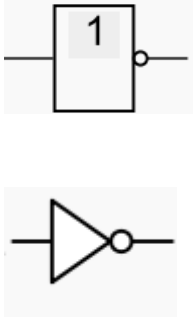
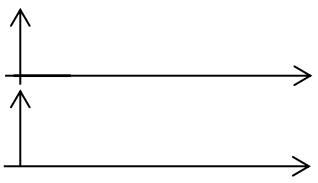
L'état de la variable de sortie est identique à celui de la variable d'entrée.

Symbole logique	Table de vérité	Équation logique	Schéma électrique à contacts	Chronogramme
				

a = True S = a print(S)	a = False S = a print(S)
a = 0b1 S = a print(bin(S)) =	a = 0b0 S = a print(bin(S)) =

2. La fonction «NON» (ou COMPLÉMENT) not / ~:

L'état de la variable de sortie est l'inverse de la variable d'entrée.

Symbole logique	Table de vérité	Equation logique	Schéma électrique à contacts	Chronogramme
				

a = True S = not a print(S)	a = False S = not a print(S)

En théorie

Valeur	binaire	décimal
a	10011100_{2}	156_{10}
~a	1100011_{2}	99_{10}

Bien que l'opérateur NON semble être le plus simple de tous, vous devez faire preuve d'une extrême prudence lorsque vous l'utilisez en Python. Tout ce que vous avez lu jusqu'à présent est basé sur l'hypothèse que les nombres sont représentés avec des entiers **non signés**. Cependant nativement Python utilise des nombres signés en complément à 2.

Petit rappel :

Pour écrire un nombre signé (négatif) en complément à deux on :

- Fixe un format : ex 8 bits
- On complémente à 1 le nombre positif (on applique la fonction NON) bit à bit sur le nombre de départ
- On complémente à deux en ajoutant au complément à 1 un 1 supplémentaire

Ex a = 0b 0100 1110
$\bar{a}$ = 0b 1011 0001

$$-a = \bar{a} + 1 \text{ = 0b 10110010}$$

En effet si on prend un nombre sur un format et que l'on lui ajoute son complément on obtient autant de 1 que le format choisi ex 8 Bit : $a + \bar{a} =$ 0b1111 1111.

Si de nouveau on ajoute à cette somme 1 on dépasse le format ex $a + \bar{a} + 1 =$ 0b 1 0000 0000.
Comme la machine physique qui effectue les calculs ne peut pas dépasser son format alors :

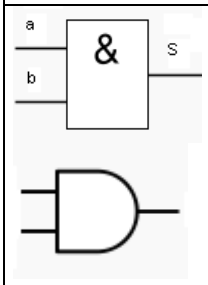
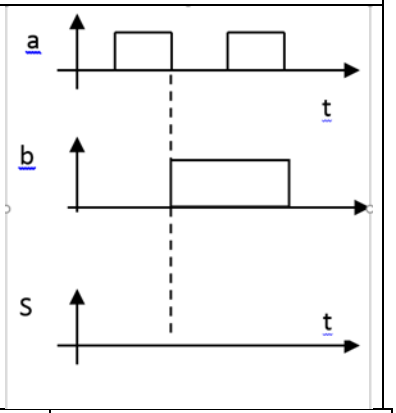
$$a + \bar{a} + 1 = 0b0 \ (sur\ les\ 8\ bits\ d'exemple)\ ou\ encore - \boldsymbol{a} = \boldsymbol{\bar{a} + 1}$$

C'est aussi ce que fait l'opérateur NON « ~ » bit à bit du python

Valeur	Valeur décimale	bin(~A)	int(bin(~A))
A = 0b 1		-0b10	-2
A = 0b 10101010	170	-0b10011101	-171
A = 0b 00001111	15	-0b10000	-16
A = 0b 1111 0000	240	-0b11110001	-241

3. La fonction «ET» (ou INTERSECTION) and / & :

L'état de la variable de sortie sera à l'état 1 si et seulement si les deux variables d'entrées sont à l'état 1. La fonction **ET** correspond à une liaison électrique série

Symbole logique	Table de vérité	Équation logique	Schéma électrique à contacts	Chronogramme
				

a = False b = False S = a and b print(S) =	a = True b = False S = a and b print(S) =	a = False b = True S = a and b print(S) =	a = True b = True S = a and b print(S) =
a = 0b 0 **b = 0b 0** **S = a & b =**	**a = 0b 1** **b = 0b 0** **S = a & b =**	**a = 0b 0** **b = 0b 1** **S = a & b =**	**a = 0b 1** **b = 0b 1** **S = a & b =**

Essayer:

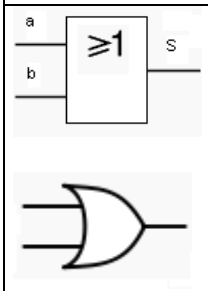
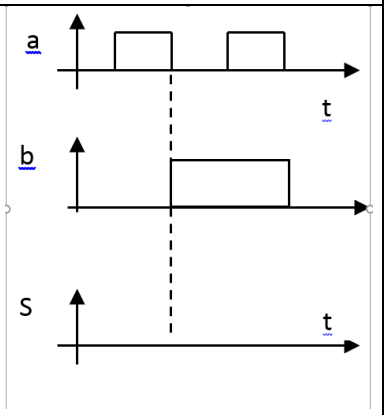
a = **0b10101010**
masque = **0b11110000**
a & masque = **0b**

Que font les 0 du masque au nombre de départ

Que font les 1 du masque au nombre de départ

4. **La fonction « OU » (ou UNION) or / | :**

L'état de la variable de sortie sera à l'état 1 si l'une des entrées est à l'état 1 ou bien si toutes les entrées sont à l'état 1. La fonction **OU** correspond à une liaison électrique parallèle

Symbole logique	Table de vérité	Équation logique	Schéma électrique à contacts	Chronogramme
				

a = False b = False S = a or b print(S) =	a = True b = False S = a or b print(S) =	a = False b = True S = a or b print(S) =	a = True b = True S = a or b print(S) =
a = 0b 0 **b = 0b 0** **S = a | b =**	**a = 0b 1** **b = 0b 0** **S = a | b =**	**a = 0b 0** **b = 0b 1** **S = a | b =**	**a = 0b 1** **b = 0b 1** **S = a | b =**

Essayer:

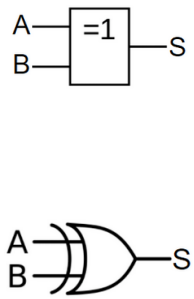
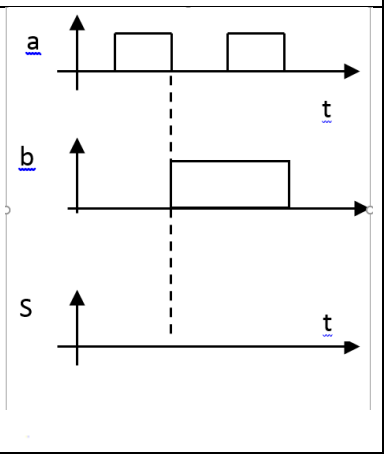
a = **0b10101010**
masque = **0b11110000**
a | masque = **0b**

Que font les 0 du masque au nombre de départ

Que font les 1 du masque au nombre de départ

5. La fonction «OU EXCLUSIF » :

L'état de la variable de sortie sera à l'état 1 si l'une des entrées est à l'état 1 ou bien si toutes les entrées sont à l'état 1. La fonction **OU** correspond à une liaison électrique parallèle

Symbole logique	Table de vérité	Equation logique	Schéma électrique à contacts	Chronogramme
				

a = 0b 0 **b = 0b 0** **S = a^b =**	**a = 0b 1** **b = 0b 0** **S = a^b =**	**a = 0b 0** **b = 0b 1** **S = a^b =**	**a = 0b 1** **b = 0b 1** **S = a^b =**

Rappels les caractères sont le plus souvent codés en UTF-8 qui reprend le code ASCII des années 1970 pour les caractères latins non spéciaux :

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	0001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	0010	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	0101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	0110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	0111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

EX : « A » = 0b 0100 0001 = $ 41 = $(65)_{10}$

Pour obtenir le code décimal d'un caractère ord() ex ord("A") >>> 65

Pour faire afficher ce code en binaire : bin(ord("A")) >>> '0b1000001'

Pour transformer un code ascii (utf-8) en caractère : chr(65) >>> "A"

```
1  en_clair = ord("A")
2  print(bin(en_clair))
3  cle = 0b11110000
4  en_code = en_clair ^ cle
5  print(bin(en_code))
6  print(chr(en_code))
7  mystere = en_code^cle
8  print(bin(en_code))
9  print(mystere)
10 print(chr(mystere))
```

Que fait ce programme ?

5 *PROPRIETES DE L'ALGEBRE DE BOOLE :*

5.1 Commutativité :

5.2 Associativité:

5.3 Distributivité

5.4 Identités remarquables:

Equation	Schéma électrique à contacts
a+0=a	
a.0=0	
a+1=1	
a.1=a	
a+a=a	
a.a=a	
$a.\bar{a} = 0$	
$a + \bar{a} = 1$	
$a + (\bar{a}.b) = a + b$	

Exercices:

Représenter les schémas électriques à contacts des équations suivantes :

$S1 = \bar{a} + (b.c)$

S2= a+b.c

S3 = (a+b).c

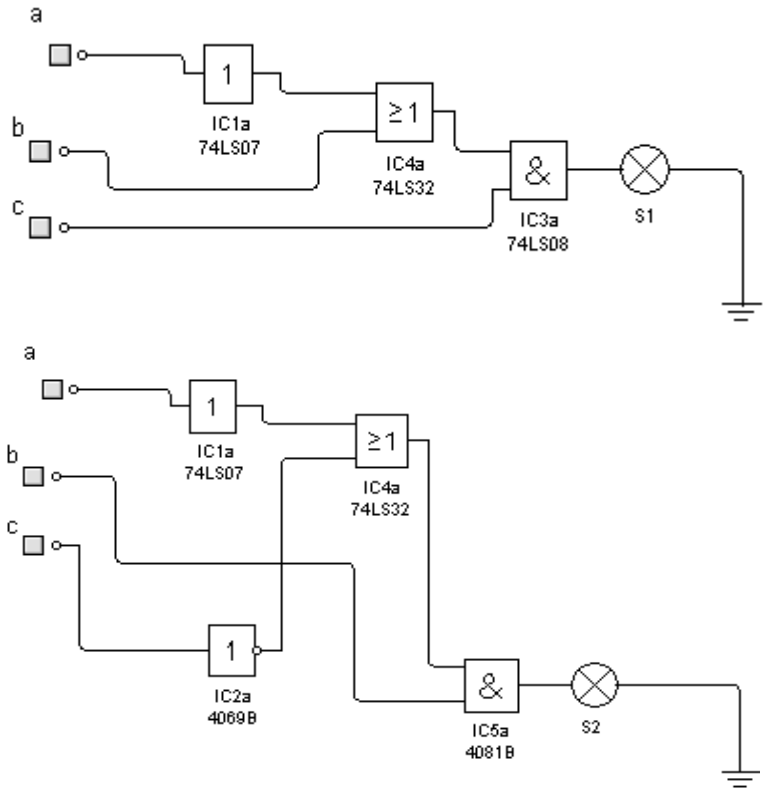
Représenter le logigramme des équations suivantes :

$S1 = \bar{a} + (b.c)$

S2= a+b.c

S3 = (a+b).c

Etablir l'équation de S1, S2



Donner l'équation de S1, S2

S1=----------

S2=----------

Exercice python et adresse IP

On représente une adresse IP sous forme de liste

IP1 = [192,168,1,10] masque M = [255,255,255,0]

Utiliser l'opérateur bin() pour transformer les élément de la liste et le masque en nombres binaires

Définir une fonction « reseau » qui donne l'adresse d'un réseau en lui fournissant adresse ip et masque comme définis ci-dessus en appliquer l'opérateur logique bit à bit va bien.

Applique les opérateurs logiques bit à bit qui permettent d'obtenir le « numéro » de la machine.

On définira une fonction « **complement** » qui prend un octet et complémente les 0 par 1 et les 1 par zero de l'octet. (Rappeler vous que la fonction ~ NON du python ne complémente pas bit à bit , en revanche une autre peut le faire facilement)

Exercice python et codage

On désire coder un mot de passe : « mot de passe » à l'aide de la fonction XOR. Ecrire une fonction « codage » qui prend une chaine de caractère et la code. Ecrire une fonction « decodage » qui prend une chaine de caractère et la décode.