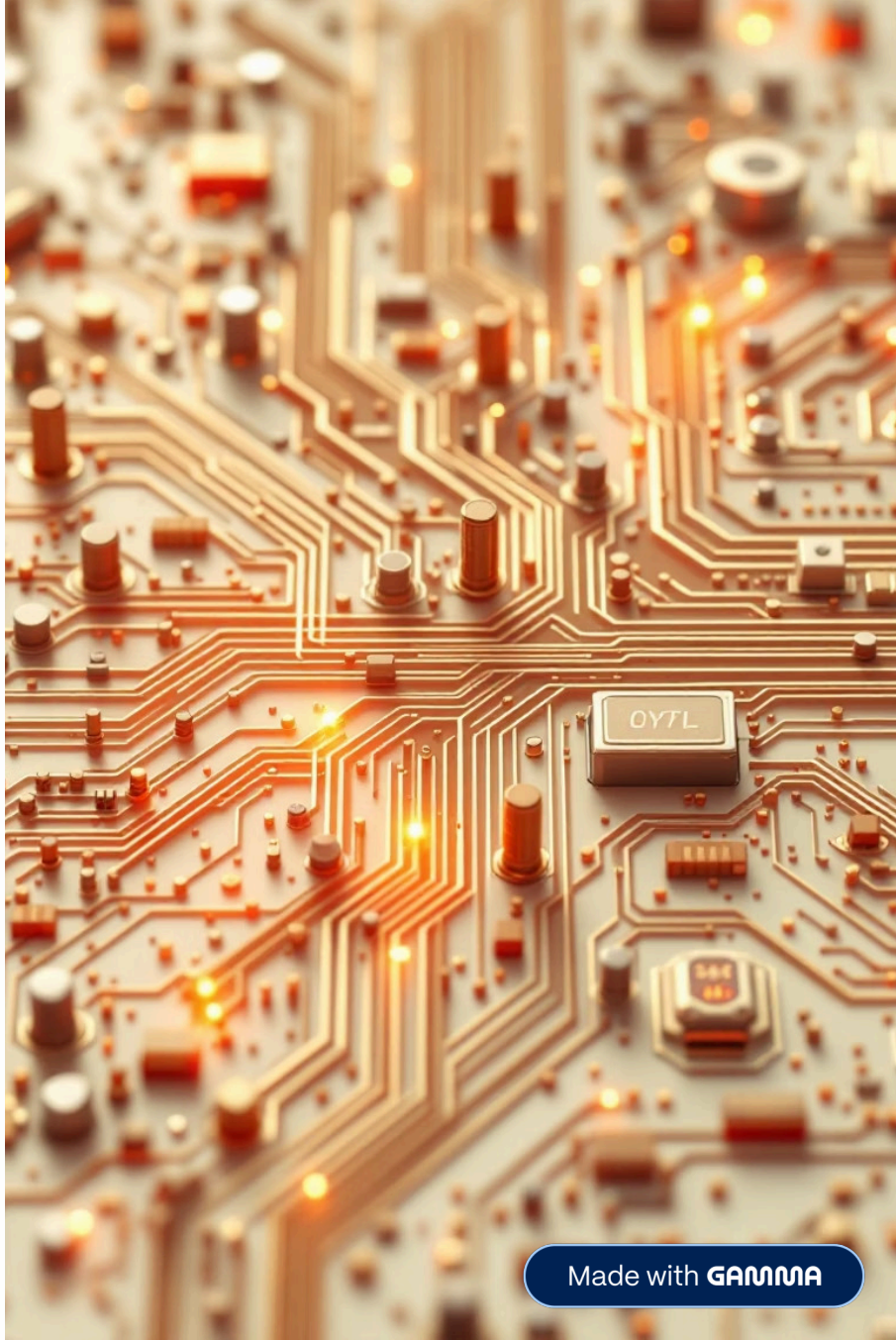
Logique pour les 3èmes : ET, OU, NON et OU Exclusif

Découvrons ensemble les opérateurs logiques qui sont à la base de l'informatique moderne.

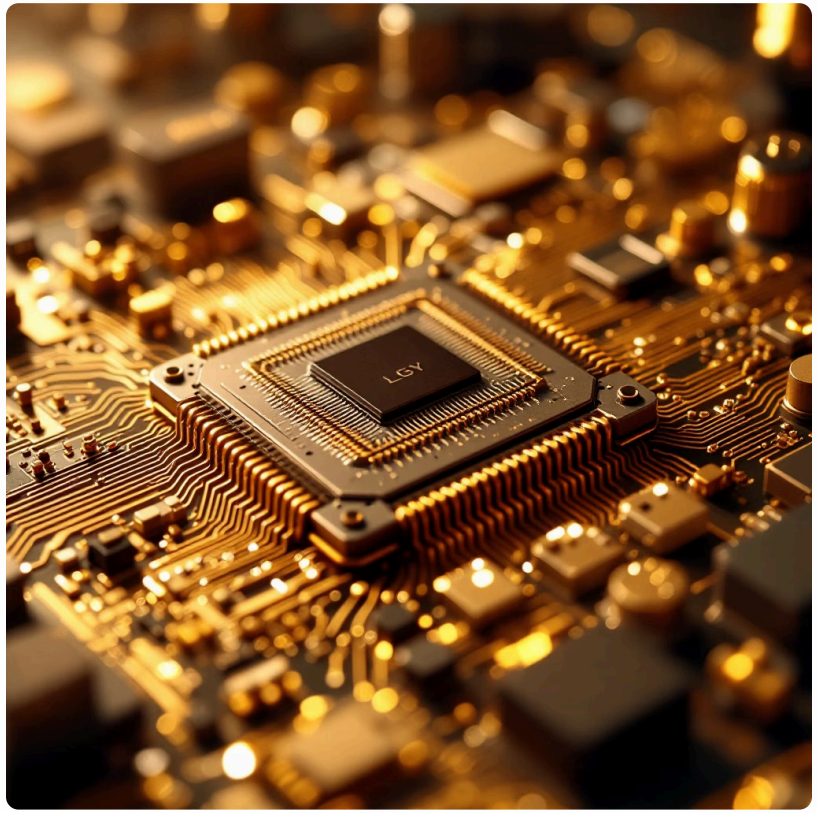


La Logique : Fondamentaux de l'Informatique

La logique est le langage que parlent les ordinateurs. Ces opérateurs sont essentiels pour comprendre comment fonctionnent les programmes et les circuits électroniques.

Les opérateurs logiques agissent comme des **portes** : ils prennent des entrées (VRAI ou FAUX, 1 ou 0) et produisent un résultat selon des règles précises.

Aujourd'hui, nous allons explorer quatre opérateurs fondamentaux qui permettent de construire toute la logique informatique.



NON : L'Inversion

Principe Simple

L'opérateur NON inverse la valeur d'une entrée. Si c'est VRAI (1), ça devient FAUX (0), et vice-versa.

Exemple Concret

Affirmation : La porte est ouverte (VRAI)

Négation : La porte n'est pas ouverte (FAUX)

Table de Vérité

A	NON A
0	1
1	0

En code bloc

```
def non(a):
 return not a

print(non(True)) # Résultat : False
print(non(False)) # Résultat : True
```

ET : La Conjonction

1

Principe de Base

L'opérateur ET ne donne VRAI que si **TOUTES** les entrées sont VRAIES. Une seule entrée fausse suffit pour que le résultat soit FAUX.

2

Exemple du Quotidien

Pour entrer dans le club, il faut **avoir 18 ans** ET **avoir une invitation**. Les deux conditions doivent être remplies simultanément.

Table de Vérité

A	B	A ET B
0	0	0
0	1	0
1	0	0
1	1	1

En code bloc

```
def et(a, b):
 return a and b

print(et(True, True)) # Résultat : True
print(et(True, False)) # Résultat : False
print(et(False, True)) # Résultat : False
print(et(False, False)) # Résultat : False
```

OU : La Disjonction Inclusive

Principe Flexible

L'opérateur OU donne VRAI si **AU MOINS une** des entrées est VRAIE. Le mot « inclusif » signifie que les deux peuvent être vraies en même temps.

Exemple Gourmand

Au restaurant : « Je prendrai un sandwich OU une salade. » On peut prendre l'un, l'autre, ou même les deux !

Table de Vérité

A	B	A OU B
0	0	0
0	1	1
1	0	1
1	1	1

En code bloc

```
def ou(a, b):
 return a or b

print(ou(True, True)) # Résultat : True
print(ou(True, False)) # Résultat : True
print(ou(False, True)) # Résultat : True
print(ou(False, False)) # Résultat : False
```

OU Exclusif (XOR) : La Disjonction Exclusive

Exclusivement Un

Le XOR est VRAI si **EXACTEMENT une** entrée est VRAIE, mais pas les deux en même temps. C'est le « ou » du choix exclusif.

Dessert OU Café

« Je prendrai un dessert OU un café » — mais pas les deux ! Il faut choisir. Si on prend les deux, le XOR devient FAUX.

Pour en savoir plus sur le OU exclusif : [Article Wikipédia]

Table de Vérité

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

En code bloc

```
def xor(a, b):
 return (a or b) and not (a and b)

print(xor(True, True)) # Résultat : False
print(xor(True, False)) # Résultat : True
print(xor(False, True)) # Résultat : True
print(xor(False, False)) # Résultat : False
```

Application : Chiffrement Simple (XOR)

Le OU Exclusif (XOR) est un outil puissant en cryptographie pour protéger les messages secrets. Son principe est élégant : combiner un message avec une clé secrète pour le rendre illisible.

01

Message Original

On prend notre message en binaire : **1 0 1 0**

02

Clé Secrète

On utilise une clé connue seulement de nous : **0 1 1 0**

03

Chiffrement

On applique XOR bit par bit : **1 1 0 0**

04

Déchiffrement

En appliquant XOR à nouveau avec la même clé, on retrouve le message original !

Propriété magique du XOR : Si on applique XOR deux fois avec la même clé, on retrouve la valeur de départ. C'est pourquoi il est si utile en cryptographie !

En savoir plus : **Wikipédia - Fonction OU exclusif**

Démonstration en code

```
message = [1, 0, 1, 0]
cle = [0, 1, 1, 0]
chiffre = [1, 1, 0, 0]

# Chiffrement
for i in range(len(message)):
 chiffre[i] = message[i] ^ cle[i]

# Déchiffrement (même opération !)
for i in range(len(chiffre)):
 message[i] = chiffre[i] ^ cle[i]
```