Valeurs, opérations et expressions booléennes

Valeurs booléennes



Cours

En informatique, une valeur booléenne (du nom du mathématicien George Boole) ne peut prendre que deux états :

- Faux (False) ou 0
- Vrai (True) ou 1

Ces valeurs sont à la base de toute la logique informatique et du fonctionnement des circuits électroniques. Un bit est la plus petite unité d'information et peut représenter une valeur booléenne.

En Python, ces deux valeurs s'écrivent False et True (avec une majuscule et sans guillemets). Elle sont du type bool:

```
>>> type(True)
<class 'bool'>
```

Opérateurs booléens

NOT



Cours

L'opérateur **NOT**, « NON » en français, inverse la valeur booléenne.

Table de vérité :

Α	NOT A
0	1
1	0

En Python, NOT s'écrit en minuscule not :

```
>>> not True
False
```

>>> not False

AND



L'opérateur AND, « ET » en français, renvoie 1 (Vrai) uniquement si toutes les valeurs sont à 1.

Table de vérité :

Α	В	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

En Python, AND s'écrit en minuscule and :

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

OR



- Cours

L'opérateur \mathbf{OR} , « \mathbf{OU} » en français, renvoie 1 (\mathbf{Vrai}) si \mathbf{au} moins une des valeurs est à 1.

Table de vérité :

Α	В	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

En Python, OR s'écrit en minuscule or :

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

XOR



Cours

L'opérateur XOR (eXclusive OR), « OU exclusif » en français, renvoie 1 uniquement si les deux valeurs sont différentes.

Table de vérité :

Α	В	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

En Python, XOR n'existe pas avec les valeurs de type bool ¹ mais on peut l'obtenir avec ^ sur des entiers :

```
>>> 1 ^ 0
>>> 1 ^ 1
>>> 0 ^ 0
```

Expressions booléennes



- Cours

Une expression booléenne est une combinaison de valeurs et d'opérateurs booléens.

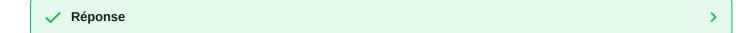
Prenons l'exemple de l'expression : (A AND B) OR (NOT C).

Si A = 1, B = 0 et C = 1, alors :

- A AND B = 1 AND 0 = 0
- NOT C = NOT 1 = 0
- (A AND B) OR (NOT C) = 0 OR 0 = 0

? Exercice corrigé

Que vaut l'expression NOT(A OR B) AND C si A = 0, B = 1, C = 1?



- A OR B = 0 OR 1 = 1
- NOT(A OR B) = NOT 1 = 0
- NOT(A OR B) AND C = 0 AND 1 = 0

Table de vérité d'une expression

Pour dresser la table de vérité d'une expression, on liste toutes les combinaisons possibles des variables d'entrée et on calcule le résultat.

Prenons l'exemple de l'expression booléenne A AND (B OR C). On commence par lister toutes les combinaisons possibles de A,B et C (23 = 8 lignes), on calcule les expressions intermédiaires (B OR C), puis le résultat final :

A	В	С	B OR C	A AND (B OR C)
0	0	0	0	0

Α	В	С	B OR C	A AND (B OR C)
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Application: l'addition binaire

L'addition binaire utilise directement les opérateurs booléens XOR et AND.

En effet pour additionner deux bits A et B :

- **Somme (S)** = A XOR B
- Retenue (Cout) = A AND B

Table de vérité :

A	В	Somme (S)	Retenue (C _{out})
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

On peut obtenir un additionneur complet, avec une retenue entrante (C_{in}), en utilisant les expressions suivantes :

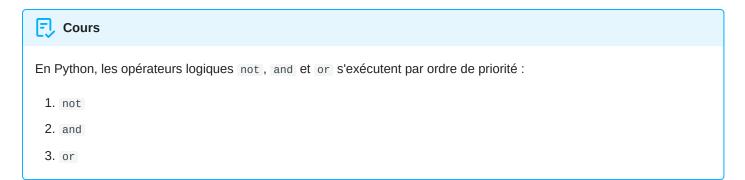
- Somme (S) = A XOR B XOR C_{in}
- Retenue (C_{out}) = (A AND B) OR (C_{in} AND (A XOR B))

Par exemple calculons, avec la retenue, l'addition: 1 + 1 + 1 (A = B = C_{in} = 1) :

• S = 1 XOR 1 XOR 1 = 0 XOR 1 = 1

- C_{out} = (1 AND 1) OR (1 AND 0) = 1 OR 0 = 1
- Résultat : 11 en binaire (3 en décimal)

Priorités et caractère séquentiel des opérateurs



Cela signifie que :

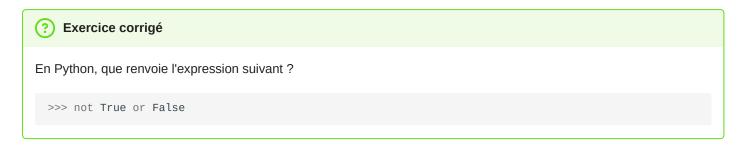
- · not s'applique avant and, et
- and s'évalue avant or.

Exemple:

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

Ici, Python calcule d'abord False and False, c'est False, puis True or False, c'est donc True.

Pour éviter les erreurs et améliorer la lisibilité du code on utilise des parenthèses : True or (False and False).





Cours

En Python, les opérateurs and et or ont un **comportement séquentiel** : ils évaluent les expressions de gauche à droite et **s'arrêtent** dès que le résultat est déterminé. L'évaluation est dite « paresseuse » (*lazy evaluation*).

Prenons l'exemple de ce programme :

```
x = 0
if x != 0 and 10/x > 2:
    print("Condition vraie")
```

Si Python évaluait 10/x, cela provoquerait une division par zéro. Mais comme x = 0 est faux, l'expression 10/x > 2 n'est jamais évaluée et le programme ne lève pas d'erreur.

De la même façon, voyons le programme suivant :

```
age = 25
if age > 18 or verifier_base_donnees():
    print("Accès autorisé")
```

Si la première condition est True, la seconde n'est pas évaluée et la fonction verifier_base_donnees() n'est pas appelée.

L'évaluation paresseuse est particulièrement utile pour optimiser les performances d'un programme et éviter certaines erreurs (division par zéro, accès à des valeurs non définies).

Prenons un exemple qui montre l'importance de l'ordre des conditions :

```
def fonction_lente():
    print("Fonction appelée")
    return True

# Version 1 : fonction appelée
resultat = fonction_lente() and True

# Version 2 : fonction appelée
resultat = True and fonction_lente()

# Version 3 : fonction NON appelée
resultat = False and fonction_lente()
```

À noter : L'évaluation séquentielle des opérations booléennes en programmation est une différence importante avec les circuits électroniques étudiés dans le chapitre « Transistors et circuits logiques » dans lesquels toutes les entrées sont évaluées simultanément.

1. On peut réaliser A XOR B en Python avec l'expression équivalente : (A OR B) AND NOT(A AND B). \leftarrow