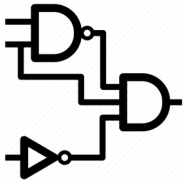
Les opérations Booléennes



Les fonctions booléennes sont utilisées partout : dans les langages de programmation, en architecture des ordinateurs, dans certains algorithmes cryptographiques... Nous nous concentrons dans ce chapitre sur les opérations NON, ET et OU qui permettent d'exprimer toutes les autres.

1. Expression des opérations Booléennes

Comme les opérations d'une variable réelle, les opérations booléennes peuvent s'exprimer de manière symbolique : l'expression $\bar{x}\cdot y + x\cdot z$ est bâtie sur le même modèle que l'expression (sin(x) × y) + (x × z), sauf que les variables x, y et z y représentent des booléens et non des nombres réels.
Contrairement aux fonctions d'une variable réelle, ces fonctions ne peuvent pas s'exprimer par des courbes. Néanmoins, elles peuvent s'exprimer par des tables car, à la différence des nombres réels, les booléens sont en nombre fini.

Les opérations Booléennes NON, ET, OU

<table>
<tr><th>Opérateur</th><th>Expression</th><th>Table de vérité</th><th>Commentaire</th></tr>
<tr><td>NON</td><td>$s = \bar{a}$
s = non a</td><td><table><tr><th>a</th><th>non a</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td><td>Le résultat de l'opérateur non a est égal à l'inverse de a</td></tr>
<tr><td>ET</td><td>s = a · b
s = a et b</td><td><table><tr><th>a</th><th>b</th><th>a et b</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td>Le résultat de l'opération a et b est vrai si a et b sont vraies</td></tr>
<tr><td>OU</td><td>s = a + b
s = a ou b</td><td><table><tr><th>a</th><th>b</th><th>a ou b</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td>Le résultat de a ou b est vrai si au moins a ou b est vrai</td></tr>
</table>

Propriétés des opérations Booléennes

Complémentarité	Commutativité	Associativité	Distributivité
$\bar{\bar{a}} = a$ $a + \bar{a} = 1$ $a \cdot \bar{a} = 0$	$a + b = b + a$ $a \cdot b = b \cdot a$	$a + (b + c) = a + b + c$ $a \cdot (b \cdot c) = a \cdot b \cdot c$	$a \cdot (b + c) = a \cdot b + a \cdot c$ $a + (b \cdot c) = (a + b) \cdot (a + c)$

2.Opérations Booléennes sous Python

Opérations Booléennes

Les opérations Booléennes NON, ET, OU sont définies par les instructions suivantes :

Priorité	Opérateur	Expression	Résultat
Haute	NON	`not x`	Si x est faux alors True sinon False
↓	ET	`x and y`	Si x est vrai, alors y sinon False
Faible	OU	`x or y`	Si x est faux, alors y sinon True

Lorsque Python évalue une expression booléenne, il le fait de façon **paresseuse**. C'est à dire que si la partie gauche d'un or est vraie, il n'évalue pas la partie droite. De même si la partie gauche d'un and est fausse, la partie droite n'est pas évaluée. Cela permet d'écrire les choses suivantes :

```
>>> x=0
>>> x==0 or 1/x<1
True
>>> x!=0 and 1/x<1
False
```

Si la division 1/x était évaluée, il y aurait une erreur, puisqu'on le peut pas diviser par 0. Mais dans les deux cas, l'évaluation n'est pas faite puisque le résultat de l'expression a déjà pu être déterminée avec la partie gauche.

Opérations Booléennes bit à bit des nombres entiers

Les opérations bit à bit ne sont pertinentes que sur les nombres entiers (types integer).

Expression	Résultat
`x | y`	OU logique effectué bit à bit
`x ^ y`	OU Exclusif effectué bit à bit
`x & y`	ET logique effectué bit à bit
`x << n`	x décalé vers la gauche de n bits
`x >> n`	x décalé vers la droite de n bits
`~ x`	Inversion des bits de x

```
>>> 9 | 3  #1001 OU 0011 = 1011 (11)
11
>>> 3 << 2  # 0011 (3) décalé de 2 donne 1100 (12)
12
```