CHAPITRE

10 Opérateurs logiques

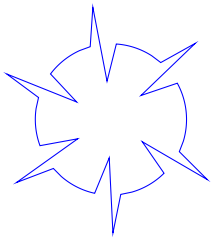

Table des matières

1 Introduction

Au travers du cours sur l'architecture des ordinateurs, nous avons vu comment des simples opérations logiques (un ET ou un Rotate Left...) pouvaient être à la base des programmes écrits en ***assembleur***.
Sans théorie excessive nous allons ici parler des ***opérateurs logiques*** qui agissent sur des ***bits***, sur des ***nombres***.

2 Les opérateurs logiques :

Il faut distinguer les opérateurs logiques des opérateurs booléens qui agissent sur les ***booléens***. Ce dernier point sera développé à la fin de ce cours.

2.1 Le OU logique

En python, l'opérateur `OU` s'écrit : `|`. La table de vérité s'écrit ainsi :

OU	0	1
0	0	1
1	1	1

Un exemple :

	1	1	0	0	0	(24)
OU	1	0	1	0	1	(21)
	1	1	1	0	1	(29)

En python, on obtient le résultat suivant :

```
>>> 24|21
29
```

> **Exercice n°1**
> Calculer les expressions suivantes à la main et vérifier ensuite avec la console python.
>
> ❶ `12|45`
>
> ❷ `202|189`

2.2 Le ET logique

En python, l'opérateur `ET` s'écrit : `&`(esperluette). La table de vérité s'écrit ainsi :

ET	0	1
0	0	0
1	0	1

Un exemple :

	1	1	0	0	0	(24)
ET	1	0	1	0	1	(21)
	1	0	0	0	0	(16)

En python, on obtient le résultat suivant :

```
>>> 24&21
16
```

Exercice n°2
Calculer les expressions suivantes à la main et vérifier ensuite avec la console python.

❶ `12&45`

❷ `202&189`

Exercice n°3
Que se passe t-il si on fait un ET logique avec `255` ? avec `192` ? avec `252` ?

2.3 Le OU EXCLUSIF logique

En python, l'opérateur `XOR` s'écrit : `^`. La table de vérité s'écrit ainsi :

XOR	0	1
0	0	1
1	1	0

Un exemple :

	1	1	0	0	0	(24)
XOR	1	0	1	0	1	(21)
	0	1	1	0	1	(13)

En python, on obtient le résultat suivant :

```
>>> 24^21
13
```

Exercice n°4
Calculer les expressions suivantes à la main et vérifier ensuite avec la console python.

❶ `12^45`

❷ `202^189`

Exercice n°5
Que se passe t-il si on applique deux fois de suite le même XOR logique à un nombre ?

2.4 Le NON logique

Le NON logique est l'opérateur qui inverse les bits. Sa table de vérité est donc :

	0	1
NON	1	0

En python, l'opérateur `NON` est `~`.
Voici ce que donne le langage Python sur quelques exemples :

```
>>> ~24
-25
>>> ~-121
120
```

Exercice n°6
Expliquer les résultats donnés précédemment.

2.5 Autres opérateurs :

Voici deux autres opérateurs, << et >> ainsi que leur effet dans les situations suivantes :

```
>>> 5<<3
40
>>> 10>>1
5
```

```
>>> 128>>1
64
>>> 128<<1
256
```

Exercice n°7
À quoi servent-ils ?

3 Les booléens

On retrouve des variables de type booléens dans de nombreux langages de programmation.

Une variable de type booléenne ne peut prendre que deux valeurs :`True` ou `False`

Elles apparaissent lors d'une déclaration (`flag = True` ou `flag = 1` par exemple) ou en évaluant une ***relation*** entre deux objets.

Les opérateurs sur les booléens sont `and`, `or` et `not`.

Les tables de vérités sont identiques à celles des opérateurs logiques ci-dessus.

```
#déclaration de booléen et de leur opérateur
a = True
b = False
c = True and False
d = True or False
e = not d
```

En Python, tout ce qui n'est pas égal à 0 pour les nombres, ou tout ce qui n'est pas vide pour les listes, chaînes de caractères... est considéré comme `True`.

```
if liste:
    print("la liste commence par", liste[0])
else:
    print("la liste est vide")
```

Exercice n°8

❶ Les variables `a`,`b` et `c` suivantes sont de type `bool`. Quelles sont leurs valeurs ?

❷ On peut aussi évaluer des chaînes de caractères et des listes. Que valent les variables `test1` et `test2` ?

```
a = 2 == 3
b = 5 > 8
c = 1 + 1 == 3 or 3**2 == 9
test1 = 'a' in "bonjour"
test2 = 2 in [2, 4, 5, 6]
```

Mais attention, parfois il faut se méfier !

Exercice n°9

❶ Que vaut le booléen `0.1 + 0.2 == 0.3` ? Testez dans une console Python.

❷ Que vaut l'expression `(80 > 50) + (60 > 50)` ? Testez dans une console Python.

Et enfin, le grand classique :

Exercice n°10

Que dire de la fonction Python suivante ?

```
def test_de_parite(n:int)->bool:
    if n%2 == 0:
        return True
    else:
        return False
```