CHAPITRE 3

Valeurs booléennes

The Footman[1] *: Do you want coffee, or tea ?*
George Boole : Yes.
The Footman : You want coffee and tea ?
George Boole : No.
The Footman : So you want coffee ?
George Boole : No.
The Footman : So you want tea !
George Boole : Yes.
The Footman : "Be a footman !" they said. "You have the legs for it !" they said. "There's no thinking all !" they said.

Sydney Padua[2]

Thème : Représentation des données : types et valeurs de base

Contenus	**Capacités attendues**
Valeurs booléennes : 0, 1. Opérateurs booléens : and, or, not. Expressions booléennes	Dresser la table d'une expression booléenne.

I. Un peu d'histoire

Les machines traitent et mémorisent l'information au moyen de circuits logiques binaires : leurs entrées et sorties se caractérisent exclusivement par deux états : l'état logique bas et l'état logique haut.

1847	**Georges Boole** sort « Mathematical Analysis of Logic »
1930	**Claude Shannon** démontre qu'avec des interupteurs fermés pour VRAI et ouvert pour FAUX, on peut effectuer des opération logiques

II. Le type booléen

Le type booléen ou `bool` est un type de données. Un booléen ne peut prendre que 2 valeurs : vrai (`True`) ou faux (`False`).

Remarque. En Python les majuscules sont importantes : true et false ne sont pas reconnues par le langage, il faut écrire `True` et `False`.

Dans certaines représentations (comme par exemple dans l'algèbre de Boole), le booléen faux est représenté par un 0 tandis que le booléen vrai est représenté par un 1.

1. Valet
2. The Thrilling Adventures of Lovelace and Babbage, Sydney Padua, éd. Penguin Books, 2015

III. Opérateurs de comparaison

En python, les opérateurs de comparaison s'appliquent à des données (numériques, chaînes de caractères...) et produisent un résultat booléen. Ils permettent de faire des tests. Le résultat de l'opération est `True` si le test est vrai, `False` si le test est faux.

Opérateur en Python	Signification
==	est égal à
!=	est différent de
<	est inférieur à
<=	est inférieur ou égal à
>	est supérieur à
>=	est supérieur ou égal à

Exemple 1

```
>>> 4 == 4
True

>>> 4 == 1
False

>>>4 != 1
True

>>> 4*3 == 2*6
True
```

IV. Opérateurs booléens

Les principaux opérateurs booléens sont `not` (NON), `and` (ET) et `or` (OU). Le fonctionnement de chaque opérateur est résumé dans un tableau appelé table de vérité.

IV. 1. Opérateur NON (`not` en Python)

a	not a
False	True
True	False

Déf.1 L'opérateur `not` renvoie la valeur opposée de l'entrée.

IV. 2. Opérateur ET (`and` en Python)

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

Déf.2 L'opérateur `ET` renvoie la valeur `True` si toutes les entrées sont à `True`, et `False` sinon.

IV. 3. Opérateur OU (`or` en Python)

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

Déf.3 L'opérateur `OU` renvoie la valeur `True` si au moins une des entrées est à `True`, et `False` sinon.

V. Priorités des opérateurs booléens

Comme dans les expressions arithmétiques, des conventions de priorité définissent la manière dont doit être lue une expression booléenne en l'absence de de parenthèses. L'opérateur le plus prioritaire est le NON, puis vient le ET et enfin le OU.

L'expression :

```
a or not b and c
```

Doit être comprise de la manière suivante :

```
a or ((not b) and c)
```

- ▷ Première étape : on évalue `not b`
- ▷ Deuxième étape : on évalue `(resultat precedent and c)`
- ▷ Troisième étape : on évalue `a or resultat precedent`

VI. Séquentialité des opérateurs `and` et `or`

Python évalue les expressions logiques de manière paresseuses (on parle de « lazy evaluation »).

▷ **OU** n'évalue le deuxième argument que si le premier est faux.

▷ **ET** n'évalue le deuxième argument que si le premier est vrai.

Exemple 2

```
>>> True or x
True
```

L'expression est vraie quelle que soit la valeur de `x` : la valeur de `x` n'est pas lue dans la mémoire et l'opération `or` n'est pas effectuée en entier.

```
>>> x or True
Traceback (most recent call last):

File "<ipython-input-5-9574821e17a1>", line 1, in <module>
x or True

NameError: name'x' is not defined
```

Exemple 3

```
>>> False and not y
False
```

L'expression est fausse quelle que soit la valeur de y : la valeur de y n'est pas lue dans la mémoire et l'opération `and` n'est pas effectuée en entier, l'opération `not` n'est pas effectuée du tout.

```
>>> not y and False
Traceback (most recent call last):

File "<ipython-input-7-9da74e9fbbdb>", line 1, in <module>
not y and False

NameError: name 'y' is not defined
```

Je retiens

- ▷ Un booléen ne prend que deux valeurs : `VRAI` ou `FAUX` ;
- ▷ les principaux opérateurs booléens sont `NON`, `ET` et `OU` ;
- ▷ l'opérateur `NON` renvoie la valeur opposée de l'entrée ;
- ▷ l'opérateur `ET` renvoie la valeur `VRAI` si toutes les entrées sont à `VRAI`, et `FAUX` sinon ;
- ▷ l'opérateur `OU` renvoie la valeur `VRAI` si au moins une des entrées est à `VRAI`, et `FAUX` sinon ;
- ▷ l'ordre le priorité des opérateurs est : `NON` puis `ET` puis `OU` ;
- ▷ l'opérateur `OU` n'évalue pas le deuxième argument si le premier est `VRAI` ;
- ▷ l'ipérateur `ET` n'évalue pas le deuxième argument si le premier `FAUX`.

VII. Exercices

Exercice 1 (opérations booléennes).
Indiquez le résultat en détaillant l'opération (le cas échéant) des expressions ci-dessous lorsqu'elles sont exécutées dans une console python.

Expression	**Résultat et justification**
`>>> not True`	
`>>> not False`	
`>>> not 21>90`	
`>>> 21<90 or not 1>2`	
`>>> 8<9 and 2<1`	
`>>> x=36` `>>> x>13 and x<27`	

`>>> a=True`
`>>> b=False`
`>>> a and not b or not a and b`	

`>>> a=True`
`>>> b=True`
`>>> not a or b and a or not b`	

Exercice 2 (opérateur OU EXCLUSIF).

1. Établir la table de vérité de la fonction `xor` définie par :

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

2. Quelle est la différence avec un opérateur OU classique ?

Exercice 3 (opérateur `nand`).

1. Complétez la fonction Python *nand*() (pour `not and`) qui renvoie la valeur de `not (a and b)`.

```
def nand(a, b):
    ..........
```

2. Établissez la table de vérité de cette fonction.
3. Quelle est la différence avec un opérateur ET classique ?
4. Comparez l'expression `nand(a, a)` avec `not a` en dressant leur table de vérité.
5. Faîtes de même en comparant :

   a. `nand(nand(a,b), nand(a,b))` avec `a and b`

   b. `nand(nand(a,a), nand(b,b))` avec `a or b`