I. Les booléens

L'algèbre booléenne ou algèbre de Boole (George Boole, mathématicien et philosophe britannique 1815/1854) est une branche des mathématiques traitant des opérations logiques. On ne manipule pas des nombres mais des propositions qui peuvent être vraies ou fausses.

Un booléen est une variable qui ne prend que 2 valeurs : Vrai (`True`) ou Faux (`False`). En programmation, on s'en sert énormément : entre autres, dans les tests (`if`) et dans les boucles (`while`), mais pas seulement !

Exercice 1.1 : SANS ORDINATEUR ! Compléter les affichages console suivants :

```
>>> a = (3<2)
>>> a
...
>>> "N" in "NSI"
...
```

```
>>> "n" in "NSI"
...
>>> x = 3
>>> x == 0
...
```

```
>>> b = (x != 3)
>>> b
...
>>> 3**2 == 6
...
```

Exercice 1.2 : Python fait un parallèle entre les booléens et certains entiers.

- Dans la console, taper `True*True` puis d'autres opérations avec `True` et `False`.
- En déduire les correspondances entre booléens et entiers `True` = ... `False` = ...

II. Les opérateurs booléens

Exercice 2.1 : On considère la fonction `aff` ci-contre qui nécessite un paramètre `h` (de type `float`)

```
def affichage(h) :
  if (h < 12) :
    return "matin"
  else :
    return "aprem"
```

- Que renvoie l'appel `affichage(10)` ?
- Que renvoie l'appel `affichage(12)` ?
- Que renvoie l'appel `affichage(12.5)` ?
- Résumer en une phrase ce que fait cette fonction :

Réponse :

- On souhaite intervertir les deux instructions d'affichage. Trouver 2 manières de reformuler la condition pour que le programme soit correct

1ère manière	2ème manière

`def affichage(h) :`
`  if .........`
`  return "aprem"`
`  else :`
`  return "matin"`
	`def affichage(h) :`
	`  if .........`
	`  return "aprem"`
	`  else :`
	`  return "matin"`

- On souhaite modifier le programme précédent pour qu'il affiche maintenant "jour" ou "nuit". On considèrera que la période "jour" est comprise entre 7h et 20h.
Trouver au moins 4 versions de ce programme !

`def affichage(h) :`
`  if .........`
`  return "jour"`
`  else :`
`  return "nuit"`
	`def affichage(h) :`
	`  if .........`
	`  return "........."`
	`  else :`
	`  return "........."`

`def affichage(h) :`
`  if .........`
`  return "........."`
`  else :`
`  return "........."`
	`def affichage(h) :`
	`  if .........`
	`  return "........."`
	`  else :`
	`  return "........."`

Les booléens

1) L'opérateur de négation : NON (not)

Exercice 2.2 :

- Déterminer la table de vérité de l'opérateur NON (not)

Taper le code

```
for A in (True,False) :
    print(A, not A)
```

Compléter la table de vérité ci-contre

Entrée	Sortie
A	not A
False	
True	

- Vérifier que la fonction définie par $f(x) = 1 - x$ joue le rôle de l'opérateur NON

Réponse : $f(0) =$ ... et $f(1) =$ ...

2) L'opérateur de disjonction : OU (or)

Exercice 2.3 :

- Quel code faut-il écrire en Python pour obtenir la table de vérité de l'opération OU ?

Réponse :

Entrée		Sortie
A	B	A ou B
0	0	
0	1	
1	0	
1	1	

- Compléter la table ci-contre.
- Vérifier que les fonctions suivantes jouent le rôle de l'opérateur OU

$f(x,y) = \max(x,y)$	$g(x,y) = x + y - xy$
$f(0,0) =$ $f(0,1) =$ $f(1,0) =$ $f(1,1) =$	$g(0,0) =$ $g(0,1) =$ $g(1,0) =$ $g(1,1) =$

3) L'opérateur de conjonction : ET (and)

Exercice 2.4 :

- Quel code faut-il écrire en Python pour obtenir la table de vérité de l'opération ET ?

Réponse :

Entrée		Sortie
A	B	A et B
0	0	
0	1	
1	0	
1	1	

- Compléter la table ci-contre.
- Vérifier que les fonctions suivantes jouent le rôle de l'opérateur ET

$f(x,y) = \min(x,y)$	$g(x,y) = xy$
$f(0,0) =$ $f(0,1) =$ $f(1,0) =$ $f(1,1) =$	$g(0,0) =$ $g(0,1) =$ $g(1,0) =$ $g(1,1) =$

Exercice 2.5 : `a = True ; b = False ; c = True ; d = False`

- ➔ `a and not b = ...`
- ➔ `not (a and c) = ...`
- ➔ `not c or not b = ...`
- ➔ `(a and b) or not d = ...`

4) Exercices

Exercice 2.6 : On considère 2 variables booléennes :

- `AmpouleGrillee` : l'ampoule électrique est grillée ou pas
- `InterrupteurON` : l'interrupteur est en position ON ou pas

1) On définit une autre variable booléenne : `LumiereAllumee`.
   De quelle manière pourrait-on affecter cette variable ?
   Réponse :

2) On définit une autre variable booléenne : `LumiereEteinte`.
   De quelle manière pourrait-on affecter cette variable ? (donner 2 réponses)
   Réponse :
   -
   -

Exercice 2.7 :

- Justifier que la table de vérité de (A and B) est la même que celle de (B and A).

- On considère les booléens `A = (x>=0)` et `B = (sqrt(x)<5)` où x est un nombre réel.
  On accepte un nombre s'il vérifie à la fois les conditions A et B.
  En Python, pour utiliser la racine carrée (sqrt pour **sq**uare **r**oo**t**), il faut l'importer du module math en écrivant, au début du programme : `from math import sqrt`
  On a écrit, en Python, 2 fonctions qui permettent de savoir si un nombre est accepté :

```
def reponse1(x):
    if (x>=0) and (sqrt(x)<5):
        return "accepté"
    else:
        return "refusé"
```

```
def reponse2(x):
    if (sqrt(x)<5) and (x>=0):
        return "accepté"
    else:
        return "refusé"
```

  Justifier qu'en Python, (`A and B`) n'est pas équivalent à (`B and A`) ?

  On dit que Python fait une évaluation paresseuse de l'opérateur and.

  Réponse :

- L'évaluation de Python pour l'opérateur `or` est-elle aussi paresseuse ?
  Réponse :

Exercice 2.8 : Donner la valeur de P dans chaque cas

`P = (2 > 3)`	`P = not(2 > 3)`
`P = (2 > 3) and (6 > 5)`	`P = (3 > 2) and (6 > 5)`
`P = (3 > 2) or (7 > 6)`	`P = (3 > 2) or (5 > 6)`

`P = (3 > 4) or (5 > 6)`	`P = (4 == 4) and False`
`P = (2**3 >= 8) and (3**2 <= 9)`	`P = not(2+3 != 5) and (7+2 == 9)`
`x = 0` `P = (x > 0) and (x < 10)`	`x = 5` `P = (x > 0) and (x < 10)`
`x = 5` `P = (x < 0) or (x > 10)`	`x = 20` `P = (x < 0) or (x > 10)`
`P = False and (False or True)`	`P = (False and False) or True`
`P = (False or (not (True and False))) and (not(True or False) or not(False and True))`	

III. Enchainement d'opérateurs booléens

Exercice 3.1 : A faire en commun

Objectif : Ecrire la table de vérité d'un enchainement d'opérateurs booléens : `"A et non B"`

- Méthode 1 : décomposer en plusieurs éléments simples : `"C = non B"` puis `"A et C"`

Entrée			
A	B	C = non B	A et C
0	0		
0	1		
1	0		
1	1		

donc

Entrée		Sortie
A	B	A et non B
0	0	
0	1	
1	0	
1	1	

- Méthode 2 : avec Python

```
def operateur(x,y):
  return (x and not y)

for A in (True,False):
  for B in (True,False):
    print(A,"avec",B,"donne",operateur(A,B))
```

Résultat

```
True avec True donne
True avec False donne
False avec True donne
False avec False donne
```

L'énorme avantage de la première méthode sur la seconde, c'est qu'elle ne nécessite aucun matériel informatique !!

Exercice 3.2 : distributivité

Soit A, B et C trois propositions.

- Démontrer que "A et (B ou C)" est équivalent à "(A et B) ou (A et C)"

A	B	C	B ou C	A et (B ou C)

A	B	C	A et B	A et C	(A et B) ou (A et C)

- Trouver une proposition équivalente à : "A ou (B et C)"

A	B	C	B et C	A ou (B et C)

A	B	C			

- Justifier que "(A ou B) et C" n'est pas équivalent à "A ou (B et C)" ?
  Donner un contre-exemple :

Exercice 3.3 :

1) On définit la fonction Z (constante égale à 0) par Z(0) = 0 et Z(1) = 0.
   Exprimer Z(x) à l'aide des opérateurs booléens ET, OU, NON.
   Réponse :
2) On définit la fonction U (constante égale à 1) par U(0) = 1 et U(1) = 1.
   Exprimer la fonction U à l'aide des fonctions booléennes ET, OU, NON.
   Réponse :

Exercice 3.4 : le multiplexeur mux et le ou-exclusif

On définit le multiplexeur par $mux(x, y, z) = \begin{cases} y \; si \; x = 0 \\ z \; si \; x = 1 \end{cases}$

1) Ecrire la table de vérité de cet opérateur

x	y	z	mux(x,y,z)

2) Justifier que mux(x, y, z) = (non(x) et y) ou (x et z)

x	y	z	non(x) et y	x et z	(non(x) et y) ou (x et z)

3) Utilisation du multiplexeur pour l'opérateur ou-exclusif

On rappelle la définition du ou-exclusif : $\text{oux}(x,y) = \begin{cases} 0 & \text{si } x = y \\ 1 & \text{si } x \neq y \end{cases}$ noté aussi : x xor y

a) Justifier que $\text{oux}(x,y) = \text{mux}(y, \text{oux}(x,0), \text{oux}(x,1))$.

Réponse :
Si $y = 0$, alors $\text{mux}(y, \text{oux}(x,0), \text{oux}(x,1)) = \ldots$
Si $y = 1$, alors $\text{mux}(y, \text{oux}(x,0), \text{oux}(x,1)) = \ldots$

b) Vérifier que $\text{oux}(x,0) = x$.

Réponse :
Si $x = 0$, $\text{oux}(x,0) =$
Si $x = 1$, $\text{oux}(x,1) =$
Donc $\text{oux}(x,0) =$

c) Vérifier que $\text{oux}(x,1) = \text{non}(x)$.

Réponse :
Si $x = 0$, $\text{oux}(x,1) =$
Si $x = 1$, $\text{oux}(x,1) =$
Donc $\text{oux}(x,0) =$

d) En déduire $\text{oux}(x,y)$ à l'aide des fonctions booléennes de base.

Réponse :

e) Une autre méthode :

- Soit A est un booléen, déterminer une écriture plus simple de A × A.

  Réponse :
  $True \times True =$
  $False \times False =$
  Dans tous les cas : A × A =

- <u>Rappels</u> : $\text{et}(x,y) = xy$; $\text{non}(x) = 1 - x$; $\text{ou}(x,y) = x + y - xy$
  Trouver une expression pour $\text{oux}(x,y)$ en fonction de x et de y

  Réponse :
  Soit $A = \text{non}(y)$ et x, donc $A =$
  Soit $B = y$ et $\text{non}(x)$, donc $B =$
  Ainsi, $\text{oux}(x,y) = A$ ou B, donc $\text{oux}(x,y) =$

Les booléens

Historiquement, les premiers ordinateurs étaient énormes car constitués de tubes électroniques (ENIAC : $160m^2$ et 30 tonnes). En 1947, des américains ont inventé le transistor, sorte d'interrupteur qui laisse passer ou pas le courant. Cela a permis de réduire la taille des ordinateurs tout en augmentant la fiabilité de ceux-ci. Ces transistors sont modélisés par des fonctions booléennes et l'assemblage de transistors revient à combiner des fonctions booléennes. Les transistors sont regroupés sur des circuits imprimés et actuellement, les processeurs sont composés de près d'un milliard de transistors. Ces transistors permettent donc de fabriquer les portes logiques ET, OU, NON (qui sont les basiques) ainsi que toutes les autres à partir de ces trois-là.
Globalement, tous les 2 ans et ce, depuis 1970, le nombre de transistors d'un processeur double : on parle de la **loi de Moore**. Cela a été possible grâce à la miniaturisation des composants : en 2016, certains transistors avaient une taille proche d'une vingtaine ou d'une trentaine d'atomes. La loi de Moore aura donc du mal à être encore valable pour les années à venir.

Exercice n°3.5 : Addition en binaire et fonctions booléennes

1) Addition de 2 nombres d'un bit chacun
   a) Rappeler les 4 calculs possibles dans ce cas
      Réponse :
   b) On peut donc écrire dans le cas général que : $a + b = xy$ (où a, b, x, y valent 0 ou 1)
      Si $a = 0$ et $b = 0$, alors $x =$ … et $y =$ …
      Si $a = 0$ et $b = 1$, alors $x =$ … et $y =$ …
      Si $a = 1$ et $b = 0$, alors $x =$ … et $y =$ …
      Si $a = 1$ et $b = 1$, alors $x =$ … et $y =$ …
      En déduire une formule pour obtenir x et y en fonction de a et b
      Réponse : $x =$ …
      $y =$ …

2) Addition de 3 nombres d'un bit chacun
   Quand on ajoute 3 bits, on obtient un nombre de 2 chiffres maximum.
   Ainsi : $a + b + c = xy$
   Indication : $a + b + c = (a + b) + c$
   a) Ecrire y en fonction de a, b et c et avec des fonctions booléennes de base.
      Réponse :
   b) Ecrire x en fonction de a, b et c et avec des fonctions booléennes de base.
      Réponse :

3) Facultatif : addition de 2 nombres de 2 bits chacun : $ab + cd = xyz$
   Réponse :

	r ← retenue a b + c d x y z