Algo 2 - Logique en math et en info

I – Vrai, Faux et Non

1.1 Qu'est-ce que le FAUX et le VRAI ?

FAUX pour 0 VIDE, VRAI pour tout le reste

C'est la méthode qu'utilise Python et que nous avions déjà vu sur le transtypage avec **`bool()`**.
FAUX correspond à 0, None et tous les contenus vides : () [] {} ""
VRAI correspond à tout le reste.

FAUX pour 0, VRAI pour 1

On travaille uniquement avec 0 et 1.
FAUX correspond à 0 uniquement.
VRAI correspond à 1 uniquement.

Algèbre de Boole

L'algèbre de Boole est la branche des mathématiques qui traite de la logique avec une approche algébrique : on ramène la logique à un simple calcul à base d'additions, de multiplications… L'algèbre de Boole doit son nom à son créateur : le mathématicien britannique George Boole, en 1854.

1.2 Table de vérité du NON

Expression a	Sortie s
FAUX 0	
VRAI 1	

1.3 Le NON logique en Python

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

1.4 Equation mathématique du NON

Mathématiquement, cela revient à écrire :

`s =` ..... ou `s =` ..... avec `a ∈ {0,1}`

On dit que $\bar{a}$ ou ¬a est le **complément** de a car l'un des deux est nécessairement vrai.

$$a + \bar{a} = 1 \qquad a + \neg a = 1$$

Notons également que `NON(NON(a)` donne `a` :

$$\bar{\bar{a}} = a \qquad \neg\neg a = a$$

II – Conjonction : ET, AND : **VRAI** si et seulement si toutes les entrées sont VRAIES

2.1 Table de vérité du ET

Le ET est _____ si et seulement si toutes les expressions d'entrée sont _____

a	b	a AND b
0	0	
0	1	
1	0	
1	1	

2.2 Le ET logique en Python

```
>>> a = True
>>> b = False
>>> a and b
False
```

2.3 Equation mathématique du ET

Mathématiquement, cela revient à écrire :
`s = a ET b` et **`s = a.b`** avec `a,b ∈ {0,1}`

2.4 Evaluation paresseuse

Dans la plupart des langages de programmation (dont Python), on dit que l'évaluation des expressions logiques est **paresseuse** : on évalue les expressions de gauche à droite, et dès qu'on rencontre une expression évaluée à FAUX, on arrête d'évaluer : la réponse d'un ET est nécessairement FAUX !

CONCLUSION : l'ordre dans lequel on fournit les expressions à évaluer peut rendre l'évaluation d'un ET long ou rapide, provoquer une erreur ou pas.

```
1  def est_valide(note):
2      return type(note) == int
              and note >= 0
              and note <= 20
```

On commence par vérifier que la note est bien un entier AVANT de tenter de la comparer à 0 puis à 20.

III – Disjonction : OU OR : **FAUX** si et seulement si toutes les entrées sont FAUSSES

3.1 Table de vérité du OU

Le OU est _____ si et seulement si toutes les expressions d'entrée sont _____ .

a	b	a OR b
0	0	
0	1	
1	0	
1	1	

3.2 Le OU logique en Python

```
>>> a = True
>>> b = False
>>> a or b
True
```

3.3 Equation mathématique du OU

Mathématiquement, cela revient à écrire :
s = a OU b et **s = a+b** avec a,b ∈ {0,1}

3.4 Evaluation paresseuse

Dès qu'on rencontre une expression évaluée à VRAI, on arrête d'évaluer : la réponse est nécessairement VRAI !

CONCLUSION : l'ordre dans lequel on fournit les expressions à évaluer peut rendre l'évaluation d'un ET long ou rapide, provoquer une erreur ou pas.

Dans la fonction ci-dessous, on commence par vérifier que la note est bien un entier AVANT de tenter de la comparer à 0 puis à 20 :

```
1  def est_valide(note):
2      return note == "A"
       or (type(note) == int
            and note >= 0
            and note <= 20)
```

3.5 Interprétation du symbole + : addition sur des nombres, OU ou concaténation ?

Addition classique en décimal :
1 + 1 + 1 = ……..

Addition classique en binaire :
1 + 1 + 1 = ……..

OU en logique :
1 + 1 + 1 = ……..

Concaténation :
'1' + "1" + "1" = ……..

IV – Ou exclusif : XOR

4.1 Table de vérité du XOR

a	b	a XOR b
0	0	
0	1	
1	0	
1	1	

4.2 XOR n'existe pas en Python mais...

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

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

4.3 Equation mathématique du XOR

$$s = a.\overline{b} + \overline{a}.b = a \oplus b$$

V – a NAND b = NOT(a AND b) : **FAUX** si et seulement si toutes les entrées sont VRAIES

Loi de Morgan : $a \text{ NAND } b = \overline{a.b} =$

VI – a NOR b = NOT(a OR b) : **VRAI** si et seulement si toutes les entrées sont FAUSSES

Loi de Morgan : $a \text{ NOR } b = \overline{a+b} =$