Correction DS02

Thème 1 : Bases de la programmation
Thème 2 : Binaire - circuit - codage

Durée de l'épreuve : **01h50**

L'usage de la calculatrice n'est pas autorisée.
Le candidat répond sur feuilles doubles numérotées et garde l'énoncé.
Les traces de recherche, même incomplètes ou infructueuses, seront valorisées.
La qualité de la rédaction, la clarté et la précision des raisonnements seront prises en compte.

Exercice 1 (10 points)

1. Écrire la table de vérité des fonctions logiques :

a. $x\ And\ y$

b. $x\ Or\ y$

c. $Not(x)\ Or\ Not(y)$

d. $\big(x\ And\ Not(y)\big)\ Or\ \big(Not(x)\ And\ y\big)$

x	y	$x\ And\ y$	$x\ Or\ y$	$Not(x)\ Or\ Not(y)$	$\big(x\ And\ Not(y)\big)\ Or\ \big(Not(x)\ And\ y\big)$
0	0	0	0	1	0
0	1	0	1	0	1
1	0	0	1	0	1
1	1	1	1	0	0

2. Soit un guichet bancaire automatique qui possède :

- trois entrées pour demander de l'argent :
  - e_{100} pour demander 100 Dh
  - e_{200} pour demander 200 Dh
  - e_{300} pour demander 300 Dh
- deux sorties pour récupérer un billet :
  - S_{100} pour récupérer un billet de 100 Dh
  - S_{200} pour récupérer un billet de 200 Dh

Une activation simultanée de plusieurs entrées est une fausse manœuvre (et n'active donc aucune sortie).

a. Établir la table de vérité de ce guichet bancaire.

e_{100}	e_{200}	e_{300}	S_{100}	S_{200}
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

b. Donner les fonctions logiques (non simplifiées) de ses sorties.

$S_{100} = \overline{e_{100}} \cdot \overline{e_{200}} \cdot e_{300} \ + \ e_{100} \cdot \overline{e_{200}} \cdot \overline{e_{300}}$
$S_{200} = \overline{e_{100}} \cdot \overline{e_{200}} \cdot e_{300} \ + \ \overline{e_{100}} \cdot e_{200} \cdot \overline{e_{300}}$

3. Implémenter une fonction qui :

- prend en paramètre une liste de nombre
- renvoie le plus grand nombre de cette liste

```
assert maxListe([1, 2, 3]) == 3
assert maxListe([1, 2, 5, 4, 1, 3]) == 5
```

```
def maxListe(maListe) :
    max = maListe[0]  #initialisation du max avec le 1er élément de la liste
    # parcours de la liste du 2ème élément à la fin
    for i in range(1, len(maListe)) :
        if maListe[i] > max :
            # mis à jour du max le cas échéant
            max = maListe[i]
    return max
```

4. Vous avez développé un zellige avec *Turtle* dans le cadre de votre projet.
Ce zellige est un pavage du plan d'un motif lui même constitué par la répétition d'une forme de base.
Implémentez la fonction *motif()* de votre projet (inutile de gérer les couleurs)
Voir votre projet

5. Vous disposez maintenant d'une fonction *motif()*.
Implémenter la fonction *zellige(n, m, d)* qui trace le pavage du plan avec le motif (inutile de gérer les couleurs) :

- n (int) : le nombre de motifs sur une frise (largeur)
- m (int) : le nombre de motifs sur la hauteur
- d (float) : la distance entre deux motifs

Voir votre projet

Exercice 2 (5 points)

1. Coder tous les entiers de 0 à 20 (base 10) en binaire et en hexadécimal.

Décimal	0	1	2	3	4	5	6	7	8	9	10
Binaire	0	1	10	11	100	101	110	111	1000	1001	1010
Hexadécimal	0	1	2	3	4	5	6	7	8	9	A

Décimal	11	12	13	14	15	16	17	18	19	20
Binaire	1011	1100	1101	1110	1111	1 0000	1 0001	1 0010	1 0011	1 0100
Hexadécimal	B	C	D	E	F	10	11	12	13	14

2. Convertir $(0101\ 0101)_2$ en décimal et en hexadécimal ; puis $(1AE)_{16}$ en décimal et en binaire.
$(0101)_2 = (5)_{16} \Longrightarrow (0101\ 0101)_2 = (55)_16 = (5 \times 16 + 5)_2 = (85)_2$
$(1AE)_{16} = (1 \times 16^2 + 10 \times 16 + 14)_{10} = (256 + 160 + 14)_10 = (420)_{10}$
$(1)_{16} = (0001)_2$ et $(A)_{16} = (1010)_2$ et $(E)_{16} = (1110)_2 \Longrightarrow (1AE)_{16} = (0001\ 1010\ 1110)_2$

3. Implémenter une fonction qui convertit un nombre binaire donné sous forme de texte en nombre décimal :

```
assert conversion_binaire_decimal("101") == 5
```

voir votre TP

4. Réécrire et compléter la fonction ci-dessous qui convertit un nombre décimal en binaire :

```
def conversion_decimal_binaire(n):
    binaire = ""
    dividende = XXX
    while XXX != 0:
        quotient = dividende // XXX
        binaire = binaire + XXX
        dividende = XXX
    binaire_inv = ""
    for i in range(len(binaire)):
        binaire_inv = binaire_inv + XXX
    return XXX

assert conversion_decimal_binaire(5) == "101"
```

voir votre TP

Question Bonus : Implémenter une fonction qui convertit un nombre de la base n à la base m.

Exercice 3 (5 points)

1. Implémenter une fonction qui renvoie la moyenne de deux notes (les deux notes sont sur 20 et ont le même coefficient) :

```
assert moyenneDeuxNotes(10, 16) == 13
```

```
def moyenneDeuxNotes(n_1: float, n_2: float):
    return (n_1 + n_2) /2
```

2. Implémenter une fonction qui renvoie la moyenne d'une liste de notes (toutes les notes sont sur 20 et ont le même coefficient) :

```
assert moyenneListeNotes([10, 16]) == 13
assert moyenneListeNotes([10, 12, 14, 16, 18]) == 14
```

```
def moyenneListeNotes(listeNotes: list):
    m = 0
    for i in range(len(listeNotes)):
        m = m + listeNotes[i]
    return m / len(listeNotes)
```

3. Implémenter une fonction qui renvoie la moyenne pondérée de deux devoirs (les deux devoirs sont sur 20) :
   - devoir_1 (list) : le devoir 1 sous forme d'une liste à deux éléments (note et pondération)
   - devoir_2 (list) : le devoir 2 sous forme d'une liste à deux éléments (note et pondération)

```
assert moyennePondereeDeuxdevoirs([11, 1], [14, 2]) == 13
assert moyennePondereeDeuxdevoirs([10, 2], [16, 3]) == 13.6
```

```
def moyennePondereeDeuxdevoirs(d_1: list, d_2: list):
    n_1 = d_1[0]
    p_1 = d_1[1]
    n_2 = d_2[0]
    p_2 = d_2[1]
    return (n_1 * p_1 + n_2 * p_2) / (p_1 + p_2)
```

4. Implémenter une fonction qui renvoie la moyenne pondérée d'une liste de devoirs (tous sur 20) :
   - devoirs (list) : une liste de devoirs, chaque devoir étant une liste à deux éléments (note et pondération)

```
assert moyennePondereeListedevoirsSur20([[11, 1], [14, 2]]) == 13
assert moyennePondereeListedevoirsSur20([[10, 1], [13, 2], [14, 3]]) == 13
```

```
def moyennePondereeListedevoirsSur20(devoirs: list):
    points = 0
    ponderation_totale = 0
    for i in range(len(devoirs)):
        points += devoirs[i][0] * devoirs[i][1]
        ponderation_totale += devoirs[i][1]
    return points / ponderation_totale
```

5. Implémenter une fonction qui renvoie la moyenne pondérée sur 20 d'une liste de devoirs (chaque devoir étant maintenant une liste à trois éléments (note, pondération et note maximale du devoir)) :

```
assert moyennePondereeListedevoirs([[5, 1, 10], [15, 2, 20]]) == 14
```



```python
def moyennePondereeListedevoirs(devoirs: list):
    points = 0
    points_max = 0
    for i in range(len(devoirs)):
        points += devoirs[i][0] * devoirs[i][1]
        points_max += devoirs[i][1] * devoirs[i][2]
    return points / points_max * 20
```

Question Bonus : Réimplémenter la dernière fonction pour tenir compte du cadeau du père Noël : non prise en compte dans le calcul de la moyenne pondérée du devoir qui maximisera la moyenne résultante.

