EXERCICES DE REVISION

Exercice 1 : Théorie

a) **Ecrivez le code et donnez les explications répondant aux requêtes.**

1. **Créez une liste « `li` » vide .**
2. **Ajoutez à « `li` » votre prénom (type str) comme élément .**
3. **Ajoutez à « `li` » en premier élément le dict `{1: 2}`**
4. **Sur base de la variable « `li` » (!), créez un tuple dans une variable « tu » qui contient les mêmes éléments que « `li` » .**
5. **Pouvez-vous supprimer le dernier élément de « `li` » ? Si oui, écrivez le code correspondant, si non, expliquez pourquoi.**
6. **Pouvez-vous supprimer le dernier élément de « `tu` » ? Si oui, écrivez le code correspondant, si non, expliquez pourquoi.**

b) Soit un dict « `positions` » :

```
positions = {'pos1': 'joueur2', 'pos2': 'joueurs1', 'pos3': 'joueur3'}
```

Ecrivez une boucle qui permet d'afficher en console les joueurs, triés sur base de leurs positions (de la position 1 à la position 3).

c) Soit le code suivant:

```
a = ([1, 2], [3, 4])
b = a
b[1][1] = 5
print a
```

Que va afficher le print? Et surtout, pourquoi?

Exercice 2 : Questions à choix multiples

A. Que produira l'instruction <print> du morceau de code suivant :

```
01: listeA = [1, 2, 3, 4, 5]
02: listeA.append(listeA[5 - 1] + 1)
03: listeA.sort()
04: listeA.append(listeA[5] - 1)
05: listeA.append("50")
06: listeA.reverse()
07: print listeA[1]
```

a. 4
b. 5
c. 50
d. ERREUR !

B. Quel est le résultat du fragment de code suivant :

```
01: a, b, c = 2, "", 6
02: while 1 == 1:
03:     b = str(a) + ","
04:     if a >= c:
05:         break
06:     for i in range(1, 3):
07:         a += 1
08: print b
```

a. 2,4,6,8,10,12,14, …
b. 2,4,6,8,
c. 2,4,6,
d. 6,

C. Que produira l'instruction <print> du morceau de code suivant :

```
01: fruits = {"poire": "vert", "tomate" : "rouge"}
02: fruits["exam"] = fruits.get("fraise", "absent")[:2]
03: one = fruits.get("exam", "absent")
04: two = fruits['poire'][-1:-3:-1]
05: three = "%s et %s"
06: print three % (one, two)
```

a. fra et
b. ab et tr
c. fra et tre
d. abs et tre

D. Que produira l'instruction <print> du morceau de code suivant :

```
01: a = [1, 0]
02: b = (5, 4, a)
03: c = (a, list(a), b)
04: a[1] = 1
05: print c[0][1] + c[1][1] + c[2][2][1]
```

a. ERREUR !
b. 1
c. 2
d. 3

E. Par quoi faut-il remplacer *XXXXXX* et *YYYYYY* pour que le code suivant affiche à l'écran :
0-0-0-2-2-2-2-4-4-4-4-6-6-6-6-8-8-8-8-10-

```
01: i = XXXXXX
02: st = ''
03: while (YYYYYY):
04:     st = st + str((i / 4) * 2) + '-'
05:     i = i + 1
06: print st
```

a. XXXXXX → 0 et YYYYYY → i <= 20
b. XXXXXX → 0 et YYYYYY → i< 20
c. XXXXXX → 1 et YYYYYY → i < 20
d. XXXXXX → 1 et YYYYYY → i <= 20

F. Que produira l'instruction <print> du morceau de code suivant :

```
01: list_ = [5, 4, 8, 8, 9]
02: list_[len(list_) - 1] = 0
03: list_[-1] = 1
04: list_[2:2] = list_[3:8]
05: list_[1:3] = list_[4:0:-2]
06: print list_
```

a. ERREUR !
b. [5, 8, 8, 1, 8, 1]
c. [5, 8, 8, 8, 8, 1]
d. [5, 8, 8, 1, 8, 8, 1]

Exercice 3 : Correction de code

Il vous est demandé de corriger un programme écrit en python qui répond aux spécifications ci-dessous. (Pour information, le code ci-dessous comporte 8 erreurs à retrouver et corriger.) Vos corrections doivent assurer que le programme soit syntaxiquement correct pour un interpréteur python 2.7 et qu'il fonctionne normalement. C'est-à-dire que le programme doit se lancer et produire les résultats attendus par les spécifications. Des « corrections » qui introduisent de nouvelles erreurs peuvent vous pénaliser. Voici l'énoncé du programme.

Un programme définit une fonction « `ajouterTroisEntiers` ». Cette fonction reçoit une liste d'entiers en paramètre et modifie cette liste en l'enrichissant de 3 éléments, soit les trois entiers consécutifs au dernier élément de la liste reçue en paramètre. La fonction renvoie également cette liste modifiée.

Exemple :

Si la fonction reçoit une liste `[2, 9, 4]`***, elle la modifie et renvoie*** `[2, 9, 4, 5, 6, 7]`.

Ce programme crée ensuite une liste « `liste` » contenant les entiers 1 et 2. Il utilise la fonction `ajouterTroisEntiers` ***pour enrichir*** `liste` ***de 3 éléments. Il affiche ensuite simplement en console le contenu de*** `liste`.

```
1 def ajouterTroisEntiers(li)
2     derElem += li[-1]
3     for i in range(derElem + 1, derElem + 3 ):
4         li = li.append(i)
5     return liste
6
7 liste = (1, 2)
8 ajouterTroisEntiers(liste, "int")
9   print liste  #affiche [1, 2, 3, 4, 5]
```

SOLUTION

01 : manque « : »

02 : mettre « = » au lieu de « += »

03 : mettre « + 4 » au lieu de « + 3 » (pour bien aller de derElem +1 à +3 compris)

04 : pas de « li = » (append est une fonction qui modifie une list mais renvoie None !)

05 : renvoyer « li » et non « liste » (qui n'est pas déclaré dans la fonction…)

07 : pour une liste « [1, 2] »

08 : un seul paramètre : « (liste) »

09 : cette instruction qui suit doit être bien alignée à gauche (sinon erreur)

Exercice 4 : Etoiles

Ecrivez un script Python en utilisant des boucles "*while*" ou "*for*" imbriquées qui affichera la figure suivante composée uniquement d'étoiles et d'espaces (les commentaires à droite ne sont que des indications pour vous aider):

```
01: **********              # 2 étoiles - 0 espace - 8 étoiles
02: **  *******             # 2 étoiles - 2 espaces - 7 étoiles
03: **    ******            # 2 étoiles - 4 espaces - 6 étoiles
04: **      *****           # 2 étoiles - 6 espaces - 5 étoiles
05: **        ****          # 2 étoiles - 8 espaces - 4 étoiles
06: **          ***         # 2 étoiles - 10 espaces - 3 étoiles
07: **            **        # 2 étoiles - 12 espaces - 2 étoiles
08: **              *       # 2 étoiles - 14 espaces - 1 étoile
```

Exercice 5 : Traduction d'algorithme

Une fonction `construireDictMoyennesEtudiants` reçoit deux listes en paramètres. Ces deux listes ordonnées sont de même taille, avec des indices correspondant. C'est-à-dire que le premier élément de la première liste correspond au premier élément de la seconde liste, etc…

La première liste `etudiants` contient des tuples, chacun contenant 3 éléments de type `str`: le matricule, le nom, le prénom.

La seconde liste `pointsEtudiants` contient les cotes correspondantes des étudiants. Ces cotes sont dans une chaine de caractères, séparées par un point-virgule « ; ». Les cotes avec décimales sont représentées au format float python, c'est-à-dire avec un point pour la virgule. Les cotes pour un étudiant absent sont représentées dans cette str par `abs` (et valent 0).

Cette fonction renvoie un dict, qui contient pour chaque étudiant de ces listes, le matricule de l'étudiant en clef et la moyenne (type str) en valeur.

Il vous est demandé d'écrire le code python 2.7 répondant à l'algorithme de cette fonction.

Exemple:

```
etudiants = [("E4467", "Chimé", "Henri"), ("E4758", "Louré", "Samuel")]

pointsEtudiants = ["10;10.5;9.5", "12;abs;12"]


construireDictMoyennesEtudiants(etudiants, pointsEtudiants)
```

renvoie un dict contenant:

```
{'E4467': '10.0', 'E4758': '8.0'}
```

Voici l'algorithme à implémenter en pyhton :

(1) Définir la fonction `construireDictMoyennesEtudiants`**, avec son nom et les paramètres qu'elle reçoit en input. Dans cette fonction:**

(2) Créer un dictionnaire vide `moyennesEtudiants`**.**

(3) Parcourir la liste `etudiants` **(de manière à connaitre l'indice dans cette liste des étudiants parcourus, pour pouvoir l'utiliser). Pour chaque étudiant parcouru:**

(4) Initialiser sa moyenne à 0 dans une variable `moy`**.**

(5) Créer une liste `pointsEtudiant` **qui contient tous les points de l'étudiant traité, grâce à :**

(5.a) la liste `pointsEtudiants`**,**

(5.b) l'indice de l'étudiant actuellement parcouru et

(5.c) la fonction `split()`**, qui aidera à obtenir directement une liste des points (type str), à partir de la** `str` **des points (en utilisant « ; » comme séparateur).**

(6) Parcourir cette liste `pointsEtudiant`**. Pour chaque cote:**

(7) Si la valeur est différente de "`abs`**", ajouter la cote à** `moy`**. Ne pas oublier de convertir la cote de** `str` **vers** `float` **avant de pouvoir l'additionner.**

(8) Lorsque toutes les cotes de l'étudiant sont parcourues, obtenir `moy` **en divisant le total obtenu jusqu'ici dans** `moy` **par le nombre de cotes de l'étudiant, grâce à la fonction** `len()` **appliquée sur la liste des points** `pointsEtudiant`**.**

(9) Ajouter cette moyenne obtenue dans le dictionnaire `moyennesEtudiants` **:**

(9.a) La clef est le matricule. Il peut être obtenu grâce à la liste `etudiants` **et à l'indice de l'étudiant actuellement parcouru, en récupérant le premier élément de son tuple.**

(9.b) La valeur doit être la moyenne obtenue dans `moy`**, avec un chiffre après la virgule, au format** `str`**. Utilisez la chaine** `"%.1f"` **pour formater** `moy`**.**

(10) Lorsque tous les étudiants sont parcourus, la fonction doit renvoyer le dict `moyennesEtudiants` **qui est maintenant rempli.**

En respectant les consignes, ce code devrait faire une petite douzaine de lignes.

Solution :

```
def construireDictMoyennesEtudiants(etudiants, pointsEtudiants):
    moyennesEtudiants = {}

    for indice in range(len(etudiants)):
        moy = 0.0
        pointsEtudiant = pointsEtudiants[indice].split()

        for cote in pointsEtudiant:
            if cote != "abs":
                moy += float(cote)

        moy /= len(pointsEtudiant)
        moyennesEtudiants[etudiants[indice][0]] = "%.1f" % (moy)

    return moyennesEtudiants
```