INFO-H-100 – Informatique – Prof. Th. Massart
1ère année du grade de Bachelier en Sciences de l'Ingénieur
Examen de août – ***Solutions***

Prénom : Nom : Matricule :

Remarques préliminaires

- On vous demande de répondre aux questions **sur les feuilles de l'énoncé**. Les réponses sur les feuilles de brouillon ne seront pas acceptées.
- N'oubliez pas d'inscrire votre nom, prénom et numéro de matricule sur **chaque** feuille.
- Vous disposez de trois heures et vous ne pouvez pas utiliser de notes.
- Vous pouvez utiliser le verso des feuilles pour répondre.
- Si du code vous est demandé,
  - la réponse à la question doit comprendre le code *Python* structuré et conforme aux règles de bonne pratique et conventions,
  - sauf mention contraire, vous ne pouvez utiliser aucune fonction de librairies (pas d'`import`)
  - veillez à découper votre réponse en fonctions de manière pertinente
  - veillez à utiliser des structures de données appropriées.

Question 1 - Fonctionnement de Python et complexité (7 points)

Un cube est un tableau à 3 dimensions. Dans le code donné ci-dessous, cube représente un tel cube d'entiers sous forme de listes imbriquées a trois niveaux d'imbrication où ici chaque dimension a deux valeurs.

En supposant que `my_foo` est appelée globalement avec un argument de type cube (tableau à 3 dimensions) d'entiers,

1. expliquez ce que fait cette fonction **et ce qui est imprimé lors de son exécution sur cet exemple précis** avec `cube = [[[1,3],[0,0]], [[0,3],[2,-1]]]`
2. pour cette même exécution, expliquez, grâce à 2 ou 3 diagrammes d'état représentatifs, l'état du programme lors des print. (Inutile de donner le diagramme d'état lors de chaque print)
3. cette fois en supposant que cube est de façon générale un tableau à trois dimensions d'entiers (liste de listes de listes d'entiers) avec n le paramètre donnant le nombre d'éléments pour chaque dimension (avec l'hypothèse que n a la même valeur pour chaque dimension), donnez la complexité **maximale** en O() de la fonction `my_foo` en temps d'exécution. Précisez bien les paramètres utilisés pour exprimer la complexité et justifiez bien vos réponses (le résultat seul ne suffit pas pour obtenir des points).

   Conseil : calculez la complexité maximale de `my_foo(s)` pour `s` de type entier (cas de base) ensuite, utilisez le résultat de la complexité avec un tableau à 0 (resp. 1, 2) dimension(s) pour calculer la complexité pour un tableau à 1 (resp. 2 , 3) dimensions.

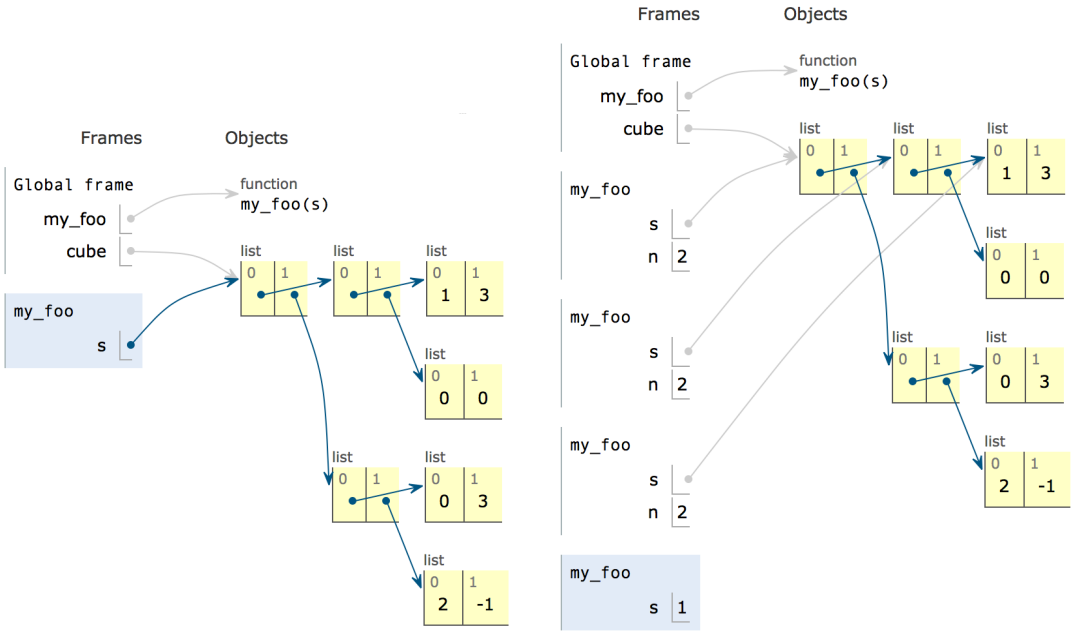
```
def my_foo(s):
   print(s) # traçage de l'exécution
   if isinstance(s,list) :
      n = len(s)
      my_foo(s[0])
      for i in range(1,n):
         e = my_foo(s[i])
         j = i-1
         while j>=0 and s[j] > e:
            s[j+1] = s[j]
            j=j-1
         s[j+1] = e
   return s

cube = [[[1,3],[0,0]],
        [[0,3],[2,-1]]]
print(my_foo(cube))
```

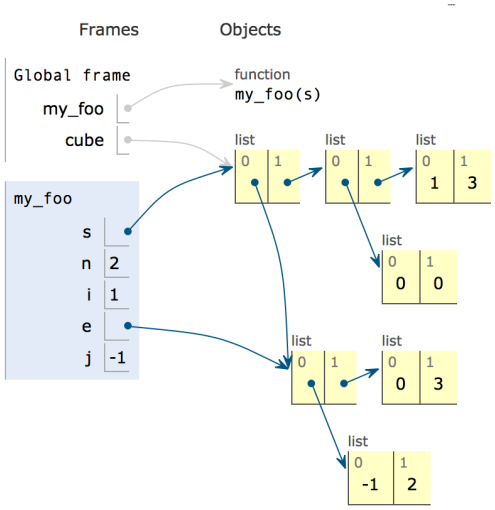
Solution

1. Que fait et imprime le code :
   — `my_foo` effectue une sorte de tri du cube plan par plan, ligne par ligne et valeur par valeur en fonction de la valeur (ou de la ligne ou du plan) la plus petite. La méthode de tri utilisée, est celle du tri par insertion.
   — Pour l'exemple, le code imprime :

```
[[[1, 3], [0, 0]], [[0, 3], [2, -1]]]
[[1, 3], [0, 0]]
[1, 3]
1
3
[0, 0]
0
0
[[0, 3], [2, -1]]
[0, 3]
0
3
[2, -1]
2
-1
résultat :  [[[-1, 2], [0, 3]], [[0, 0], [1, 3]]]
```

2. Donnons les diagrammes d'état
   — au début lors du premier print
   — lors du premier return, suite à 4 appels imbriqués à `my_foo`
   — et à la fin lors du dernier return avant l'impression du résultat final





3. Pour la complexité :
   — s est de type int

```
def my_foo(s):              # O(1)
    print(s)                # O(1)
    if isinstance(s,list) : # O(1)
        ...
    return s                # O(1)
```

   — s est une liste d'int

```
def my_foo(s):  # O(n^2)
    print(s)                        # O(n)
    if isinstance(s,list) :         # O(n^2) (O(1) pour le test)
        n = len(s)                  # O(1)
        my_foo(s[0])                # O(1)
        for i in range(1,n):        # O(n^2) (O(n) itér)
            e = my_foo(s[i])        # O(1)
            j = i-1                 # O(1)
            while j>=0 and s[j] > e: # O(n) (O(n) itér)
                s[j+1] = s[j]       # O(1)
                j=j-1               # O(1)
            s[j+1] = e              # O(1)
    return s                        # O(1)
```

   — s est une matrice : (liste de listes d'int)

```
def my_foo(s):  # O(n^3)
    print(s)                        # O(n^2)
    if isinstance(s,list) :         # O(n^3) (O(1) pour le test)
        n = len(s)                  # O(1)
        my_foo(s[0])                # O(n^2)
        for i in range(1,n):        # O(n^3) (O(n) itér)
            e = my_foo(s[i])        # O(n^2)
            j = i-1                 # O(1)
            while j>=0 and s[j] > e: # O(n^2) (O(n) itér et test en O(n))
                s[j+1] = s[j]       # O(1)
                j=j-1               # O(1)
            s[j+1] = e              # O(1)
    return s                        # O(1)
```

   — s est un cube (liste de listes de listes d'int)

```
def my_foo(s):  # O(n^4)
    print(s)                        # O(n^3)
    if isinstance(s,list) :         # O(n^4) (O(1) pour le test)
        n = len(s)                  # O(1)
        my_foo(s[0])                # O(n^3)
        for i in range(1,n):        # O(n^4) (O(n) itér)
            e = my_foo(s[i])        # O(n^3)
            j = i-1                 # O(1)
            while j>=0 and s[j] > e: # O(n^3) (O(n) itér et test en O(n^2))
                s[j+1] = s[j]       # O(1)
                j=j-1               # O(1)
            s[j+1] = e              # O(1)
    return s                        # O(1)
```

   — si s est un entier, $O(1)$ (juste une test et un return)
   — si s est une liste à une dimension on obtient : $O(n^2)$
   — si s est une liste à deux dimensions on obtient : $O(n^3)$ ($O(n)$ tris) ensuite un tri en $O(n^3)$ puisque $O(n^2)$ comparaisons où chaque comparaison est en $O(n)$
   — si s est une liste à trois dimensions on obtient : $O(n^4)$ ($O(n)$ tris de tableaux à deux dimensions en $O(n^3)$) ensite tri avec $O(n^2)$ comparaisons où chaque comparaison est en $O(n^2)$.

Question 2 - Opérations sur les fichiers (5 points)

Une société enregistre les achats de ses clients dans un fichier. La forme de ce fichier est la suivante : Chaque achat est mémorisé sous la forme d'une ligne contenant le nom du client, le nom de l'article acheté et le prix. Les 3 entrées sont séparées par un caractère '|'. Voici un exemple d'un tel fichier :

Achats.txt

```
Thierry | Apprendre Python | 23
Alain | Cormen | 12
Jean | Sedgewick | 45
Thierry | Mastering Python | 36
Jean | Knuth | 118
```

Vous avez la garantie que le fichier fourni est bien formé et ne contient donc que des lignes, chacune séparées en les trois parties mentionnées.

A intervalle régulier, la société veut exécuter une procédure de génération de factures. Il s'agit de lire le fichier et, pour chaque client, regrouper la liste de ses achats et le montant total. Le résultat doit être écrit dans un autre fichier. Par exemple :

Factures.txt

```
Alain  |  Cormen  | 12
Thierry  |  Apprendre Python  |  Mastering Python  | 59
Jean  |  Sedgewick  |  Knuth  | 163
```

Chaque ligne du fichier résultat commence par le nom du client. Ce nom est suivi de tous les articles achetés par ce client et finalement du prix total (la somme de tous les achats du client). Comme montré ci-dessus, chaque partie est séparée des autres sur la ligne par un '|'. Remarquez que dans le fichier résultat les clients n'apparaissent plus qu'une seule fois. Leur ordre d'apparition et l'ordre des articles est sans importance.

Écrivez une fonction `factures(fNameIn, fNameOut)` qui crée le fichier `fNameOut` au départ du fichier existant `fNameIn`. Par exemple, `factures("Achats.txt", "Factures.txt")` produirait le résultat montré ci-dessus.

Solution.

La solution la plus élégante utilise, dans `file_to_dict` la fonction (méthode) `setdefault` disponible pour les dictionnaires. Cette fonction renvoie la valeur associée à une clef du dictionnaire (`purchases` dans ce cas-ci) ou, si celle-ci n'existe pas, modifie le dictionnaire en y ajoutant cette clef (en premier paramètre) associée à la valeur par défaut fournie comme deuxième paramètre. `setdefault` renvoie ensuite la valeur (que ce soit celle qui préexistait dans le dictionnaire ou la "nouvelle" valeur par défaut). Un dictionnaire étant *mutable*, les deux assignations suivantes dans le code (`value[0]=...` et `value[1]=...`) modifient bien donc le dictionnaire.

```
def file_to_dict(file_name_in):
    fin  = open(file_name_in)
    purchases = {}
    for line in fin:
        pur = line.strip().split('|')
        client = pur[0]
        value = purchases.setdefault(client, [[], 0])
        value[0].append(pur[1])
        value[1] += int(pur[2])
    return purchases

def dict_to_file(d, file_name_out):
    fout = open(file_name_out, "w")
    for key, value in d.items():
        fout.write(key + " | ")
        for product_name in value[0]:
            fout.write(product_name + " | ")
        fout.write(str(value[1]) + '\n')
    fout.close()

def factures(file_name_in, file_name_out):
    purchases = file_to_dict(file_name_in)
    dict_to_file(purchases, file_name_out)
```

Solution alternative simplifiée pour la fonction `file_to_dict`.

Tout en étant moins élégante, la solution ci-dessous est plus simple à lire, car plus explicite sur les opérations. Le résultat reste évidemment le même.

```
def file_to_dict(file_name_in):
    fin  = open(file_name_in)
```

```
    purchases = {}
    for line in fin:
        pur = line.strip().split('|')
        client = pur[0]
        product_name = pur[1]
        price = int(pur[2])
        if client in purchases:
            purchases[client][0].append(product_name)
            purchases[client][1] += price
        else:
            purchases[client] = [[item], price]
    return purchases
```

Question 3 - Tri (4 points)

Dans une classe préparatoire pour rentrer à l'Ecole Polytechnique de Bruxelles, les étudiants sont évalués selon 3 cours différents : Mathématique, Physique et Anglais. Les mathématiques sont considérées comme étant la matière la plus importante. Une liste contient les évaluations de tous les étudiants. Ceux-ci sont représentés sous forme d'un tuple de longueur 4. Le premier élément est une chaine de caractère représentant le nom de l'étudiant, le second représente la note sur 20 en mathématique, la troisième la note sur 20 en physique et la quatrième la note sur 20 en anglais (cf. exemple ci-dessous).

Ecrire une fonction `tri_etudiants(ls)` qui trie les étudiants en mettant en premier ceux ayant la meilleure note en mathématique. En cas d'égalité, l'étudiant ayant la meilleure moyenne est mis en premier. Si cela n'arrive toujours pas à les départager, l'étudiant ayant la meilleure note en physique sera mis en premier.

Utilisez soit le tri par insertion soit le tri par selection.

Exemple

```
>>> ls=[('Jean',15,12,10),('Alain',19,17,9),('Stefan',15,18,13),('Thierry',15,17,14)]
>>> tri_etudiants(ls)
>>> print(ls)
[('Alain',19,17,9),('Stefan',15,18,13),('Thierry',15,17,14),('Jean',15,12,10)]
```

Solution.

```
def moyenne(etud):
    """ La moyenne de etud """
    return (etud[1] + etud[2] + etud[3]) / 3

def grade(etud):
    """
    Donne un grade à etud.
    Les facteurs 441 (= 21 * 21) et 21 assurent de donner un poids prioritaire
    d'abord aux maths (etud[1]) puis à la moyenne, sur la physique(etud[2])
    """
    return 441 * etud[1] + 21 * moyenne(etud) + etud[2]

def vient_avant(e1, e2):
    """ Détermine si le grade de e1 est meilleur que celui de e2 """
    return grade(e1) > grade(e2)

def echange(ls, i, j):
    ls[i], ls[j] = ls[j], ls[i]

def tri_etudiants(ls):
    """ Le tri par sélection classique """
    for i in range(len(ls) - 1):
        posMin = i
        for j in range(i + 1, len(ls)):
            if vient_avant(ls[j], ls[posMin]):
                posMin = j
        echange(ls, i, posMin)
```

De manière alternative, la fonction `vient_avant()` peut-être évaluée de façon différente en prenant tous les tests indiqués dans l'énoncé en compte :

```
def vient_avant(e1, e2):
    """ Détermine si le grade de e1 est meilleur que celui de e2 """
    return e1[1] > e2[1] or (                                   # La note de math est meilleure ou
           e1[1] == e2[1] and moyenne(e1) > moyenne(e2) or ( # elle est égale mais la moyenne est meilleure ou
           moyenne(e1) == moyenne(e2) and e1[2] > e2[2]      # elle est égale mais la note de physique est meilleure
           ))
```

Question 4 - Récursivité (4 points)

Soit une liste `ls`, dont chaque élément peut être un entier ou une sous-liste. Notez que chaque sous-liste peut, à son tour, contenir des entiers ou des sous-listes [1]. Écrire une fonction récursive `noEmptySubLists(ls)` qui fait une copie profonde en ignorant les sous-listes vides dans la copie. Il s'agit donc de copier chaque élément de la liste excepté les sous-listes vides.

Exemple

```
>>> noEmptySubLists([])
[]
>>> noEmptySubLists([1, 2])
[1, 2]
>>> noEmptySubLists([1, 2, []])
[1, 2]
>>> noEmptySubLists([1, [2, [[], []]], [3, [], 4]])
[1, [2], [3, 4]]
```

Solution.

```
# Deep-copy en supprimant toutes les sous-listes vides
def noEmptySubLists(ls):
    res = []
    for x in ls:
        if type(x) == list:
            y = noEmptySubLists(x)
            if y != []:
                res.append(y)
        else:
            res.append(x)
    return res
```

1. mais qu'aucune (sous-)liste n'est référencée cycliquement