M1102: Introduction à l'algorithmique et à la programmation

Feuille de TD n°7

Boucles imbriquées

INSTRUCTIONS: à la fin du TD, vous devez déposer sous Celene une archive zip contenant les fichiers Python correspondant à ce TD (fournis avec la feuille). Vous pourrez déposer une version améliorée de votre travail jusqu'à la **VEILLE** du TD suivant.

L'évaluation de la période 2 sera basée sur ces rendus.

Objectif

L'objectif principal de cette feuille de TD est de travailler sur la notion de boucles imbriquées c'est-à-dire des boucles qui contiennent d'autres boucles comme dans les exemples ci-dessous.

```
for i in range(N):
    for j in range(M):
        ici du code

while not trouve:
    for i in range(N):
        ici du code

for i in range(N):
    while ok:
        ici du code
...
```

Dans ce type de construction, la boucle la plus interne est exécutée à chaque tour de boucle de la boucle la plus externe. Cette notion apparaît naturellement dans beaucoup d'algorithmes qui travaillent sur des structures de données complexes (imbriquées elles aussi). Elle apparaît aussi dans de nombreux algorithmes classiques comme les tris.
Les exercices de cette feuille vont vous faire manipuler ce type de construction.

Exercice 1 *Comprendre l'imbrication des boucles*

Voici une fonction simple qui utilise des boucles imbriquées. Essayez de prédire ce que va afficher les différents appels ci-dessous puis vérifier votre prédiction avec un interprète Python3. Pour bien comprendre ce qui se passe vous pouvez utiliser le débogueur de Wing.

```
def boucles_imbliquees(N,M):
  for i in range(N):
      print("- j'effectue le tour",i,"de la boucle externe")
      for j in range(M):
          print(" ++ j'effectue le tour",j,"de la boucle interne")
```

1. Indiquez ce que va afficher l'appel `boucles_imbliquees(2,3)`
2. Indiquez ce que va afficher l'appel `boucles_imbliquees(3,2)`
3. Pour chacun de ces deux appels combien de fois l'instruction 5 sera-t-elle exécutée ?

4. Trouvez une formule sur `N` et `M` qui permettent de prédire le nombre de fois où cette instruction sera exécutée.
5. Modifiez la fonction `boucles_imbliquees` pour qu'elle compte le nombre de fois l'instruction 5 est exécutée et l'affiche. Vérifiez que votre formule est bien conforme à la réalité.

Exercice 2 *Des listes de listes*

Dans cet exercice, vous allez travailler sur une liste dont les éléments sont des listes comme par exemple la liste `[[1,2,5],[10,8],[],[7,8,1]]`. Le fichier à compléter s'appelle `listeDeListes.py`. Les fonctions à implémenter sont les suivantes :

1. On souhaite une fonction qui transforme une liste de listes en une chaine de caractères qui contient une ligne pour chaque sous liste de la grande liste, les éléments de chaque sous-liste étant accollé les uns autres (sans aucun séparateur). Par exemple pour la liste `[[1,2,5],[10,8],[],[7,8,1]]`, on veut obtenir la chaine `"125\n108\n\n781\n"` qui s'affiche comme suit :

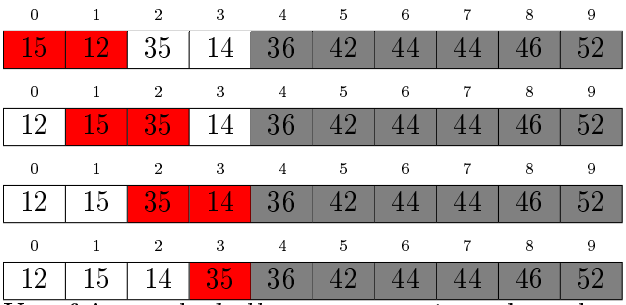
```
123
108

781
```

   Grâce à cette fonction vous pourrez visualisez les deux ASCII-arts fournis dans le script `listeDeListes.py`.
2. On souhaite une fonction qui recherche la valeur maximum contenue dans une liste de listes (dont les éléments sont des nombres). Si la liste est vide on souhaite obtenir la valeur `None`. Par exemple sur la liste `[[1,2,5],[10,8],[],[7,8,1]]` on souhaite obtenir 10
3. On souhaite une fonction qui retourne la liste des valeurs maximum pour chaque sous liste contenue dans la liste. Par exemple sur la liste `[[1,2,5],[10,8],[],[7,8,1]]` on souhaite obtenir `[5,10,None,8]`.

Exercice 3 *Le tri à bulle*

Cet exercice consiste à implémenter le tri à bulle dans le fichier `tri.py`. En Python, il existe la méthode `sort()` qui trie une liste mais le but de l'exercice est évidemment que vous implémentiez vous-même la méthode indiquée.
Le tri à bulle consiste à parcourir la partie non triée de la liste (cases non grisées) et de faire remonter le plus grand élément de cette partie (la *bulle*) en dernière position de la partie non triée. Pour ce faire on va comparer le premier élément avec le deuxième, s'ils ne sont pas dans le bon ordre, on les échange, puis on continue ainsi jusqu'à arriver à la fin de la partie triée comme illustré ci-dessous.



Une fois que la *bulle* est remontée en haut la partie non triée on sait que l'on vient de classer un nouvel élément de la liste

0	1	2	3	4	5	6	7	8	9
12	15	14	35	36	42	44	44	46	52

Il faudra recommencer le processus jusqu'à ce que la partie triée de la liste soit toute la liste. Cet algorithme nécessite deux boucles imbriquées : on vous demande d'écrire l'instruction `print(liste)` à la fin de chaque itération de la grande boucle. Par exemple pour l'appel `tribulle([12,5,-7,8,14,-6])` vous devez obtenir l'affichage

```
[5, -7, 8, 12, -6, 14]
[-7, 5, 8, -6, 12, 14]
[-7, 5, -6, 8, 12, 14]
[-7, -6, 5, 8, 12, 14]
```

Indiquez en commentaire dans la fonction combien de tours de boucle sont nécessaires pour trier une liste de N éléments.

Exercice 4 *Facture*

Dans cet exercice nous allons représenter une commande effectuée par un client sous la forme suivante.

```
(123,"Dupont",[("Verre",6,2.4),("Assiette",6,1.5)])
```

où `123` est le numéro de la commande, `"Dupont"` le nom du client, le troisième élément du triplet est la liste des produits commandés. Pour chaque produit, on a un triplet contenant le nom du produit, la quantité commandée et le prix unitaire du produit. À la fin de la journée, on récupère la liste des commandes effectuées dans cette journée sous la forme d'une liste Python.

```
[(123,"Dupont",[("Verre",6,2.4),("Assiette",6,1.5)]),(125,"Durand"
   ,[("vase",1,10.0)]),...]
```

1. On souhaiterait une fonction qui retourne la liste des montants des factures correspondant à cette liste de commande. Pour l'exemple ci-dessus on voudrait obtenir la liste suivante :

   ```
   [(123,"Dupont",23.4),(125,"Durand",10.0),...]
   ```

2. on souhaiterait aussi une fonction qui affiche la liste des factures comme ci-dessous.

   ```
   --------------------
   numéro: 123 Nom: Dupont
   produit                qte        prix       total
   Verre                    6 *      2.40 =     14.40
   Assiette                 6 *      1.50 =      9.00
                                             --------
   total                                        23.40
   --------------------
   numéro: 125 Nom: Durand
   produit                qte        prix       total
   vase                     1 *     10.00 =     10.00
                                             --------
   total                                        10.00
   ```

   Pour bien aligner les différentes informations, vous pouvez utiliser les méthodes `rjust` et `ljust` de l'API Python ainsi que la fonction `format` qui permet notamment transformer un `float` en une chaîne de caractères en imposant le nombre de chiffres après la virgule. Par exemple `format(8.2,'5.2f')` retourne `' 8.20'` c-à-d `8.2` représenté sur 5 caractères dont 2 après la virgule

On vous demande donc de compléter le fichier `facture.py` pour y implémenter les deux fonctions demandées.