Boucles imbriquées

Tester et comprendre le code suivant :

```
for i in range(5,9):
  print(i)
  for j in range(14,18):
    print((i,j))#(i,j) est un couple de variables, appelée tuple
```

Exercice 1 (⋆). 1. Créer une fonction `SommePuissance(n,j)` à deux paramètres entiers naturels n et j renvoie $\sum_{i=1}^{n} i^j = 1^j+2^j+3^j+\ldots+n^j$.

2. Mathématiquement, calculer `SommePuissance(100,1)` ? Est-ce cohérent avec votre fonction ?
3. Sans utiliser la fonction précédente, écrire une fonction `SommeDoubleCarree`, à un paramètre n, qui renvoie $\sum_{i=1}^{n}\sum_{j=1}^{n} i^j$.
4. De même, écrire une fonction `SommeDoubleTriangle`, à un paramètre n, qui renvoie $\sum_{i=1}^{n}\sum_{j=i}^{n} i^j$.
5. Vérifier que `SommeDoubleTriangle(10)+SommeDoubleCarree(10)` ? vaut 32191290514

Exercice 2 (𝄞⋆⋆). Écrire une fonction `PlusProchesValeurs`, à un paramètre une liste, qui renvoie les deux valeurs les plus proches. Ainsi, `PlusProchesValeurs([1,3,8,24,19,9])` doit renvoyer `(8,9)`. Pour ce faire, à l'aide de deux boucles `for`, on va parcourir tous les couples `(i,j)` avec $i < j$ et on testera la valeur de $|L[j] - L[i]|$. La commande `abs(x)` calcule la valeur absolue de `x`.

Un premier exemple d'algorithme glouton

Exercice 3 (𝄞⋆⋆). Soit un système monétaire de pièces/billets : une liste ordonnée par valeur décroissante. Avec l'Euro, on a donc la liste `L = [500,200,100,50,20,10,5,2,1]`.

1. Si vous devez rendre en monnaie la somme de 349, comment feriez-vous ? Utiliser 349 pièces d'un euro n'est pas très pratique.

On veut donc un algorithme pour payer qui minimise le nombre de pièces.

2. Écrire une fonction `PlusGrandePiece` à deux paramètres : un montant à rendre et une liste représentant le système monétaire qui renvoie la plus grande pièce/billet que l'on peut utiliser pour payer ce montant.
3. Écrire une fonction `GloutonPieces` à deux paramètres : un montant et une liste représentant le système monétaire qui renvoie la liste des pièces à utiliser. Pour 349, la fonction renverra alors `[200,100,20,20,5,2,2]`. Pour cela on utilisera une boucle `while` : tant qu'on a pas fini de payer la somme, on utilise la fonction `PlusGrandePiece` pour payer une partie du montant. On rappelle que `L = []` crée la liste vide et que `L.append(3)` rajoute 3 à la liste `L`.
4. Tester `GloutonPieces` avec le système de l'Euro et 349.
5. Tester aussi avec le système `[10,9,1]` et le montant 27. Le nombre de pièces utilisées est-il optimal ?

Remarque 1. Cet algorithme n'est donc pas optimal, en effet, on a été trop gourmand en prenant directement une pièce de 10 (la plus grande pièce possible), et du coup on ne pourra plus tomber sur la solution optimale car il reste 7 à payer (on ne peut donc plus utiliser une pièce de 9).

Remarque 2. On dit qu'un algorithme est un **algorithme glouton** s'il est une succession d'étapes où à chaque étape

— On a fait un choix qui a réduit au maximum la taille du problème (ici le montant restant à rembourser).
— Et que ce choix est définitif (ici dès qu'on a choisi de mettre une pièce de 10, on ne revient pas en arrière).

Ainsi, le rendu de pièces de monnaies est un algorithme glouton qui ne donne pas une solution optimale, mais il propose une solution convenable au problème.

Boucles imbriquées : le tri à bulle

Remarque 3. Une fonction actualise les listes même si on ne retourne pas la liste en question contrairement aux nombres :

```
a = 3
b = 5
def f(b,a):
    a = a+b#Modification seulement locale de a (dans la fonction)
    b = 3*a
    return b
print(f(b,a))#exécution de f avec a=3 et b=5 et affichage résultat
print("a=",a)#L'exécution de la fonction f n'a pas modifié a
print("b=",b)#L'exécution de la fonction f n'a pas modifié b

L = [3,5]
b = 5
def g(b):
    L[0] = b+L[1]#Modification globale de L
    b = 3*L[0]
    return b
print(g(b))#exécution de g avec b=5 et affichage résultat
print("L=",L)#L a été modifié lors de l'exécution de la fonction g
print("b=",b)#L'exécution de la fonction f n'a pas modifié b
```

Exercice 4 (𝄞⋆⋆). Dans cet exercice, on va voir une méthode pour trier une liste de façon croissante :

1. Tapez le code ci-dessous qui va vous créer une liste à trier.

```
import np as np#Bibliothèque pour du calcul numérique
#Les chargements de bibliothèques sont à mettre en début de code
L = [np.random.rand() for i in range(20)]
print("Liste à trier :",L)
```

2. Commencer par créer une fonction `Changement(L)`, où `L` est une liste, qui va parcourir tous les indices possibles i et qui échangera les valeurs de `L[i]` et de `L[i+1]` si `L[i]>L[i+1]`. Cette fonction renverra `True` si elle fait au moins un échange de valeurs, `False` sinon.
3. Créer une fonction `TriABulles` à un paramètre une liste, cette fonction va appeler la fonction `Changement` tant que celle-ci trouve des changements à faire. À la fin, il n'y a plus de changement à faire donc la liste est triée [1].
4. Si n est la taille de la liste, au maximum combien y aura t-il d'opérations à effectuer ?
5. Cette méthode de tri s'appelle le tri à bulles, pourquoi ?
6. Pour un n donné, créer N listes au hasard et mesurer le temps nécessaire pour les trier. Puis tracer sur un graphique le temps nécessaire moyen en fonction de la taille des listes. Pour mesurer le temps, on peut utiliser le code suivant

```
import time as time#Bibliothèque gérant le temps
#Les chargements de bibliothèques sont à mettre en début de code
tic = time.time()#Temps de départ
#Un gros calcul à effectuer
tac = time.time()#Temps d'arrivée
temps = tac-tic#Temps mis pour le gros calcul
```

7. Cette fonction `TriABulles` utilise une autre fonction `Changement`, écrire une fonction `TriABulles2` qui ne l'utilise pas, ainsi, on verra mieux la structure d'imbrication. On pourra de plus afficher la liste à chaque étape pour faire un petit film (ceci fonctionnera sous Spyder ou Pyzo mais pas Capytale). On pourra utiliser les commandes :

```
import matplotlib.pyplot as plt#Bibilothèque pour graphique
#Les chargements de bibliothèques sont à mettre en début de code
plt.clf()#Efface la figure
plt.plot(L)#Affiche la liste sur un graphique
plt.pause(1)# Met une pause d'une seconde
```

Les questions suivantes sont optionnelles.

1. Noter que suite à la remarque 3, cette fonction n'a pas besoin de return.