

TP d'informatique n°11

(Python Maths)

Manipulations de Listes

But du TP

- Réaliser des manipulations sur les listes.
- Introduire la notion de tableau

Vous penserez à **sauvegarder régulièrement** votre travail dans un fichier dans votre espace personnel. Pour le nom de fichier, n'utilisez ni accent, ni espace, ni caractères spéciaux (autres que `-` et `_`) : en effet, certains projets nécessitent l'ouverture de fichiers que l'on aura écrits au préalable : or cet appel peut s'avérer délicat si le nom de ce fichier possède des caractères "non habituels"

I Initialisation de quelques tableaux

1. Créer, dans l'éditeur Pyzo, les listes suivantes, qui pourront servir d'exemples pour les différentes fonctions qui seront écrites dans le TP :

```
Liste0 = [42] * 15
```

```
Liste1 = [42 for i in range(15)]
```

```
t1 = [10, 30, 42, 2, 17, 5, 30, -20]
```

```
t2 = [i**2 for i in range(-3,6)]
```

```
t3 = [i**3 for i in range(1000) if (i % 5) in {0,2,4}]
```

```
t4 = [841.0]
```

```
for i in range(20) :
```

```
    t4.append(t4[i]/3+28)
```

Vérifier dans l'interpréteur que ces listes ont bien été créés.

Quelles sont les longueurs de ces listes ?

II Manipulations des listes ; slicing et compréhension

2.
 - a) Créer la liste des dix derniers éléments de t3
 - b) Créer la liste constituée des éléments de t3 sauf les 250 premiers et les 250 derniers
 - c) Créer la liste constituée des cinq premiers éléments de t4 suivie des 5 derniers de cette même liste.
3. Deviner les valeurs des listes **k**, **t**, **m**, **n** après avoir tapé les commandes suivantes :

```
k = [10, 15, 12]
```

```
t = k
```

```
m = t
```

```
n = m[:]
```

```
m[1] = 17
```

```
n[0] = 19
```

Vérifier en exécutant effectivement ces commandes.

Pour mieux comprendre ce qui se passe, aller sur la page

<http://www.pythontutor.com/visualize.html>

entrer le code précédent, et visualiser pas-à-pas son exécution

4. Deviner les valeurs des listes **t5**, **t6**, **t7** après avoir tapé les commandes suivantes :

```
t5 = [t2[2*i + 1] for i in range(3)]
```

```
t6 = [x**2 for x in t2]
```

```
t7 = [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
```

Vérifier en exécutant effectivement ces commandes. *Ce mode de création de listes est appelé "compréhension de liste"*

5. Créer à l'aide de compréhension de listes, la liste constituée des termes d'indice pair de t1. Créer de même la liste des termes pairs de t1
6. Ecrire une fonction réalisant l'échange de deux valeurs dans une liste. Cette fonction recevra comme arguments une liste et deux indices (distincts) correspondant à des positions réelles dans cette liste, et effectuera l'échanger sans rien renvoyer.

```
>>> L = [4, 5, 6, 8, 9, 10, 11, 12, 1]
```

```
>>> echange(L,3,8)
```

```
>>> L
```

```
[4, 5, 6, 1, 9, 10, 11, 12, 8]
```

III Parcours de tableaux

7. Ecrire une fonction calculant la somme des éléments d'une liste.

```
def somme(tab) :
```

```
    ....
```

```
>>> somme(t1)
```

```
116
```

8. Ecrire une fonction calculant la moyenne des éléments d'une liste, puis une autre calculant l'écart-type. (l'écart-type est la racine carrée de la moyenne des carrés des écarts à la moyenne)

```
>>> moyenne(t2)
```

```
7.666666666666667
```

```
>>> ecart_type(t2)
```

```
7.803133273813083
```

9. Ecrire une fonction calculant le maximum des éléments d'un tableau. De même, écrire une fonction renvoyant la position de ce maximum

```
>>> maximum(t1)
42
>>> position_maximum(t1)
2
```

10. Dans la deuxième fonction écrite à l'exercice précédent, que se passe-t-il lorsque le maximum est pris plusieurs fois ? Comment changer ce comportement ?
11. Ecrire une fonction renvoyant True si la liste donnée en argument est croissante, et False sinon. Même chose pour la stricte décroissance. Tester ces fonctions
12. Ecrire une fonction prenant en entrée une liste, et renvoyant la liste des sommes cumulées (depuis le premier terme).

```
>>> sommes_cumulees(t2)
[9, 13, 14, 14, 15, 19, 28, 44, 69]
```

Evaluer le nombre d'additions réalisées en fonction de la longueur n de la liste. Si ce nombre est de l'ordre de n^2 , essayer de réécrire la fonction pour arriver à un nombre de l'ordre de n .

13. Ecrire une fonction prenant en entrée une liste contenant au moins deux éléments, et renvoyant les deux plus grands éléments de cette liste.

```
>>> deux_grands(t1)
[42, 30]
```

14. L'exercice précédent était mal formulé. Que se passe-t-il si le plus grand élément apparaît deux fois dans la liste ? Ecrire une nouvelle fonction pour que la réponse soit différente dans ce cas.

IV Manipulations avancées

- 15.a) Créer un tableau bidimensionnel (une liste de listes...) rectangulaire à 8 lignes et 7 colonnes constitué de 0
- b) Créer un tableau bidimensionnel rectangulaire à 8 lignes et 7 colonnes et dont le p -ième élément de la n -ième ligne est $2n + p^2$.
- c) Créer un tableau bidimensionnel triangulaire à 8 lignes et dont la p -ième ligne possède p fois le nombre 1.
- d) Créer une fonction qui prend comme argument un entier naturel n non nul et qui retourne le tableau bidimensionnel triangulaire correspondant au triangle de Pascal à n lignes.

16. Ecrire une fonction prenant comme argument une liste, et retournant la plus grande sous-suite croissante constituée de termes consécutifs de la liste. :

```
>>> plus_long_bloc_croissant([2, 10, 1, 3, 5, 7, 6, 8, 9])
[1, 3, 5, 7]
```

On pourra faire quelques tests sur des listes aléatoires :

```
from random import randint
```

```
resultats = []
```

```
for i in range(100) :
```

```
    t = [randint(0, 10**4) for j in range(10**3)]
```

```
    resultats.append( len( plus_long_bloc_croissant(t) ) )
```

```
>>> moyenne(resultats) , maximum(resultats)
```

17. A l'aide de la fonction clock du module time et de la fonction deepcopy du module copy, tester les différentes durées de création ou de copy de listes par les instructions :

```
from time import clock
```

```
from copy import deepcopy
```

```
debut = clock( )
```

```
T1 = tuple(" ")
```

```
for k in range(10**4) :
```

```
    T1 += (k,)
```

```
L0 = list(T1)
```

```
fin = clock( )
```

```
print(fin – debut)
```

```
debut = clock( )
```

```
L3 = [k for k in range(10**4) ]
```

```
fin = clock( )
```

```
print(fin – debut)
```

```
debut = clock( )
```

```
L1 = [ ]
```

```
for k in range(10**4) :
```

```
    L1 += [k]
```

```
fin = clock( )
```

```
print(fin – debut)
```

```
debut = clock( )
```

```
L4 = [0] * (10**4)
```

```
for k in range(10**4) :
```

```
    L4[k] = k
```

```
fin = clock( )
```

```
print(fin – debut)
```

```
debut = clock( )
```

```
L2 = [ ]
```

```
for k in range(10**4) :
```

```
    L2.append(k)
```

```
fin = clock( )
```

```
print(fin – debut)
```

Sous le même schéma, tester les durées de plusieurs méthodes de copie de listes, d'ajout d'une centaine de termes dans une longue liste sous différentes formes, de l'effet d'un changement de termes sur une copie d'une liste donnée...