TP Info 3 | Listes

Ce deuxième TP a pour objectif principal la manipulation des listes.

1. Ce qu'il faut savoir

Voici quelques rappels de syntaxe.

Création d'une liste

▷ La commande `range(debut,fin,pas)` génère les entiers compris entre `debut` et `fin-1` avec un pas égal à `pas`.

▷ Exemple :

```
>>> L=[range(3,11,2)]
>>> L
[3,5,7,9]
```

▷ On peut créer la liste de longueur N et de terme général constant égal à l'expression *a* par la commande `[a]*N`.

▷ On peut générer une liste connaissant l'expression de son terme général en fonction d'un entier naturel au moyen de la commande `range` :

```
L=[expr for k in range(debut,fin,pas)]
```

où `expr` est une expression dépendant de `k`.

▷ Exemple :

```
>>> L=[k**2 for k in range(7)]
>>> L
[0,1,4,9,16,25,36]
```

Concaténation et longueur

▷ Les listes peuvent être *concaténées* (i.e. mises bout à bout) avec l'opérateur `+`.

▷ Exemple :

```
>>> entiers=[0,1,2,3]
>>> suivants=[4,5,6]
>>> tous=entiers+suivants
>>> tous
[0,1,2,3,4,5,6]
```

▷ La commande `len` permet de calculer la *longueur* d'une liste, i.e. son nombre d'éléments.

```
>>> len(tous)
7
```

Accès en lecture et en enregistrement

▷ On peut accéder directement à un élément d'une liste `L` connaissant sa position `i` dans la liste par `L[i]`.

▷ Le premier élément d'une liste est d'indice 0.

▷ Exemple :

```
>>> tous[len(tous)-1] 6
```

▷ Les termes de la listes sont modifiables par une affectation.

▷ Exemple :

```
>>> tous[3]=5
>>> tous
[0,1,2,5,4,5,6]
```

Slicing (*tranchage*)

▷ On peut extraire une sous-liste en déclarant l'indice de début et l'indice de fin, séparés par deux-points. Cette opération est appelée tranchage (ou *slicing*) :

```
nom[debut:fin]
```

▷ Exemple :

```
>>> tous[1:3]
[1,2]
```

▷ On peut également donner un troisième indice lors de l'extraction d'une sous-liste, qui correspond au *pas* de l'extraction.

▷ Exemple :

```
>>> entiers=[0,1,2,3,4,5,6,7,8]
>>> entiers[0:8:2]
[0,2,4,6]
```

Copies vs. alias

▷ Pour *copier* la liste `L` dans la liste `M`, on utilisera la commande `M = L[:]`.

▷ La commande `M=L` crée un *alias* : les noms `M` et `L` pointent vers un même objet. Ainsi, la modification de `M` entraîne la modification de `L`.

▷ Exemple :

```
>>> L=[3,True,2]
>>> L2=L
>>> L[1]=False
>>> L2
[3,False,2] >>> L=[3,True,2]
```

```
>>> L3=L[:]
>>> L3[1]=[False]
>>> L3
[3,False,2]
>>> L
[3,True,2]
```

▷ En dehors du type liste, le signe d'affectation `=` a pour effet de copier une variable dans une autre.

▷ Exemple :

```
>>> a=2
>>> b=a
>>> a,b
(2,2)
```

```
>>> a=4
>>> a,b
(4,2)
```

Ajout et suppression d'un élément

▷ La commande `nomliste.append(a)` ajoute la valeur de l'expression `a` à la fin de la liste `nomliste`.

▷ `L=[a]+L` ou `L=L+[a]`.

▷ La commande `nomliste.remove(a)` supprime la première occurrence de la valeur de l'expression `a` dans la liste `nomliste`.

▷ La commande `nomliste.pop()` renvoie la valeur du dernier terme de la liste `nomliste` et supprime celui-ci de la liste `nomliste`.

Itération sur une liste

▷ Dans la boucle `for`, l'argument `list` n'est pas nécessairement de type `range` mais peut également être de type liste.

▷ La fonction suivante renvoie le minimum d'une liste `L`.

```
def minimum1(L):
    m=L[0]
    for element in L:
        if element<m:
            m=element
    return m
```

2. Exercices

Les difficultés sont échelonnées de la manière suivante : aucune, ♪, ♪♪, ♪♪♪ et ♪♪♪♪. Certains énoncés sont tirés des annales des concours (oral et écrit) ; leur provenance est le plus souvent précisée. Les exercices notés ♪♪♪ et ♪♪♪♪ sont particulièrement délicats.

2.1. Listes

1 . [*Entraînement sur les listes*]
On considère les listes suivantes :

```
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): l4.append(l4[i]/3+28)
```

a) Les créer sous Python.

b) Quelles sont les longueurs de ces listes ?

c) Créer la liste des dix derniers éléments de `t3`.

d) Créer la liste constituée des éléments de `t3` sauf les 250 premiers et les 250 derniers.

e) Créer la liste constituée des cinq premiers éléments de `t4` suivie des cinq derniers de cette même liste.

f) Créer à l'aide de compréhensions de liste :
   - la liste constituée des termes d'indice pair de `t1` ;
   - celle constituée des termes pairs de `t1` ;
   - celle constituée de termes de `t3` congrus à 13 modulo 42.

2 . [*B.A.ba sur les listes*]
Donner le contenu de la liste `L` à l'issue des lignes de commandes suivantes (lues de haut en bas et de gauche à droite) :

```
L,n=[],3
for i in range(n):
    if i!=0:
        L.append(i)
for i in range(1,13,2*n):
    for j in range(0,n):
        L.append([i,j])
```

```
for i in range(1,len(L),5):
    for j in range(0,i,2):
        for k in range(0,i,2):
            if (i>j) and (j>k):
                L.append([i,j,k])
```

3 . [*Manipulation des listes*]

a) *Deviner* les valeurs des listes `k`, `l`, `m` et `n` aprèss avoir tapé les commandes suivantes :

```
k=[10,15,12]
l=k
m=l
n=m[:]
m[1]=17
n[0]=19
```

Vérifier vos résultats sous Python.

b) *Deviner* les valeurs de `t5`, `t6` et `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]
```

4 . [*Algorithmes élémentaires sur les listes de nombres réels*]
Écrire des fonctions `appartient(L,x)`, `maximum(L)`, `moyenne(L)` et `variance(L)` renvoyant respectivement :

a) `True` si x figure dans la liste `L`, `False` sinon ;

b) le plus grand élément de la liste `L` ;

c) la moyenne des termes de la liste `L` ;

d) la variance des termes de la liste `L`.

5 . [*Sommes cumulées d'une liste de nombres*]
Écrire une fonction `sommesCumulees(t)` prenant en entrée un tableau `t`, et renvoyant le tableau des sommes cumulées (depuis le premier terme). Par exemple, l'appel

```
t=[0,1,2,3,4,5,6,7]
sommesCumulees(t)
```

doit renvoyer `[0,1,3,6,10,15,21,28]`. Évaluer le nombre d'additions réalisées, en fonction de la longueur n du tableau. Si ce nombre est de l'ordre de n^2, essayer de réécrire le programme pour arriver à un nombre de l'ordre de n.

6 . [*Échange de deux éléments dans une liste*]
Écrire une fonction `echangeListe(t,i,j)` réalisant l'échange de deux valeurs dans un tableau. Cette fonction recevra comme arguments un tableau et deux indices (distincts) correspondant à des positions réelles dans ce tableau, et effectuera l'échange sans rien renvoyer.

7 . [*Listes croissantes*]
Écrire un programme renvoyant `True` si la liste donnée en argument est croissante, et `False` sinon.

8 . [*Présence d'un doublon dans une liste*]
Écrire une fonction `Doublon(L)` renvoyant, pour toute liste `L`, 0 si tous les éléments de `L` sont deux à deux distincts, 0 sinon.

9 . [*Recherche dichotomique dans une liste triée*]
En suivant l'algorithme de recheche dichotomique dans une liste triée, écrire une procédure `RechercheDicho(L,a)` renvoyant `True` si l'élément a appartient à la liste triée `L` et `False` sinon.

2.2. Pour aller plus loin

10 . [*Amnistie et arithmétique, Elements of mathematics, 1975, St-Louis*]
Dans la prison centrale de Sikinia, il y a 100 cellules numérotées 1, 2, 3, ..., 30, toutes occupées. Les portes des cellules peuvent être dans deux états : ouvertes ou fermées. On peut passer d'un état à l'autre en faisant faire un demi-tour au bouton de la porte. Au moment où commence l'histoire, toutes les portes sont fermées. Pour fêter le vingtième anniversaire de la république de Sikinia, le président décide d'une amnistie. Il donne au directeur de la prison les ordres suivants :

- ▷ Tournez successivement d'un demi-tour les boutons de toutes les portes, puis d'une porte sur deux à partir de la deuxième, puis d'une porte sur trois à partir de la troisième, puis d'une porte sur quatre à partir de la quatrième, et ainsi de suite jusqu'à la dernière cellule.
- ▷ Libérez alors les prisonniers dont la porte de la cellule est ouverte.

Écrire un programme calculant la liste des prisonniers libérés ainsi que leur nombre.

11 . [*Fusion de deux listes triées*]
Écrire une fonction `fusion(A,B)` prenant en argument deux listes de nombres triées `A`, `B` et renvoyant la concaténation triée de `A` et `B`.

12 . [*Mimimum d'une liste, valeur et occurrences*]

a) Écrire une fonction `minEtInd(L)` qui renvoie un couple contenant le minimum d'une liste `L` et l'indice de sa première occurrence. Par exemple, l'appel de fonction `minEtInd([4,5,6,2,5,3,2])` doit renvoyer `(2,3)`.

b) Écrire une fonction `occsMin(L)` qui renvoie la liste des occurrences du minimum d'une liste `L`. Par exemple, `occsMin([4,5,6,2,5,3,2]` doit renvoyer `[3,6]`.

13 . [*Distance maximale dans une liste*]
Écrire une fonction `distanceMax(t)` prenant en entrée un tableau `t` non vide, et renvoyant un triplet (d,i,j) tel que $i \leqslant j$ et $d = |t_i - t_j|$ est maximal.

14 . [*Doublons*]
Écrire une fonction `indicesDoublons(t)` prenant en entrée un tableau `t` non vide, et renvoyant la liste (éventuellement vide) des couples (i,j) tels que $i < j$ et $t_i = t_j$.

15 . [*Algorithme glouton du paiement*]
Considérons le problème suivant : payer une somme en euros avec des pièces. On étudie dans cet exercice l'algorithme glouton consistant à répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante donne une solution.

a) Écrire une fonction `glouton(x)` renvoyant une liste de couples `[valeur de la pièce, nombre de pièces]` correspondant à l'algorithme glouton pour la somme de `x` euros.

b) Cet algorithme renvoie-t-il toujours une solution optimale, c'est-à-dire comportant le moins possible de pièces ?

Plus généralement, un algorithme est dit *glouton* s'il suit le principe de faire, étape par étape, un choix optimum local, dans l'espoir d'obtenir un résultat optimum global. Un algorithme n'est pas nécessairement optimal.

16 . [*Les deux plus grands termes d'une suite*]
On souhaite trouver un algorithme renvoyant les deux plus grands termes d'un tableau `t` comportant au moins deux éléments.

a) Expliquer en quoi le problème précédent est mal spécifié.

b) Écrire une fonction `termesMax(t)` prenant en entrée un tableau `t` possédant au moins deux éléments, et renvoyant les deux derniers termes de la liste `t` triée par ordre croissant.

c) Écrire une fonction `maxMax(t)` prenant en entrée un tableau `t` possédant au moins deux éléments, et renvoyant les deux plus grands éléments *de l'ensemble des valeurs de* `t` si `t` n'est pas constant, et `False` sinon.

17 . [*Petits comptes entre amis*]
Des amis (au moins deux) font bourse commune pendant leurs vacances au bord de la mer. À la fin du séjour, arrive le temps des comptes. Les dépenses de chacun des amis sont résumées par une liste `depenses` de taille n, les amis étant numérotés de 0 à $n-1$.

a) Écrire une procédure `petitsComptes(depenses)` affichant le bilan sous la forme d'une suite de phrases du type : `i doit x euros à j`. On travaillera exclusivement sur des entiers avant d'arrondir une seule fois à la fin des calculs et on appliquera l'algorithme naïf consistant à calculer ce chacun doit (en positif ou en négatif) à la communauté et, pour chaque créditeur, le faire rembourser par les débiteurs.

b) Écrire une fonction `petitsComptesBis(noms,depenses)` prenant un argument supplémentaire, la liste des prénoms des amis, et affichant le bilan des comptes sous la forme `Prénom 1 doit x euros à prénom 2`, etc. Appliquer cette fonction aux cas de Sophie, Christophe, Pascal, Laurent et Julien qui ont aboutit à la liste suivante :

```
depenses=[230,290,346,235,123]
```