Mathématiques supérieures — le 17 Novembre 2017

# TP n°5 - Les listes

## 1. Introduction aux structures de données linéaires

Nous allons dans ce TP étudier le type `list` de Python. Mais avant de comprendre ce que ce type représente en Python, il nous faut apprendre ce qu'est une *structure de données* en Informatique : une structure de données est en quelque sorte une façon de ranger et d'ordonner des objets. Il en existe plusieurs sortes, et celles qui vont nous intéresser aujourd'hui sont les **structures de données linéaires** qui permettent de représenter des suites finies ordonnées. Parmi les structures de données linéaires, on trouve :

— **Les tableaux :** Ceux-ci forment une suite de variables *de même type* stockées à des emplacements consécutifs de la mémoire. Chacun des emplacements occupe la même nombre de cases mémoire $C$ : ainsi, on peut accéder directement à l'emplacement d'indice $i$ du tableau en calculant simplement $D + iC$ où $D$ est l'adresse de la 1ere case du tableau (celle d'indice 0).
Ce type de structure est *statique* : une fois le tableau créé, on ne peut plus ajouter d'emplacement ni en supprimer ; c'est le principal désavantage de cette structure linéaire. Son principal avantage a été décrit plus haut : on peut lire et écrire sur chaque emplacement à la même "vitesse" : on dit qu'on accède aux emplacements d'un tableau **en coût constant**.

— **Les listes chaînées :** Celles-ci associent à chaque donnée (toujours de même type) un *pointeur* qui indique la localisation en mémoire de la donnée suivante - exceptée la dernière qui pointe vers une valeur désignant la fin de la structure).
L'avantage de cette structure est qu'elle est *dynamique* : on peut ajouter ou supprimer un emplacement du fait que sa localisation en mémoire n'importe pas. Son principal inconvénient est l'accès aux emplacements : pour lire ou modifier un emplacement, il faut parcourir tous les emplacements précédents pour obtenir son adresse : ainsi le $n$-ème emplacement est accessible en un temps proportionnel à $n$ !

Malgré son nom, la classe `list` n'est pas une liste chaînée : il s'agit d'une structure de données linéaire plus élaborée qui permet de combiner les avantages des deux classes décrites plus haut. À savoir que cette structure est dynamique et que le temps d'accès aux emplacements est constant. Un autre de ses avantages est qu'on peut stocker différents types de données parmi les emplacements d'une même liste.

## 2. Manipulation des listes en Python

### a. Définir une liste et accéder à ses éléments

Une liste (de type `list`) en Python est donc une suite d'objets de type quelconque (par exemple `int`, `complex`, `str` ou même `list` !).

Pour définir un tel objet, on délimite la suite d'objets par des crochets ouvrant `[` et fermant `]` et on sépare chacun des objets par une virgule `,`.

Essayons sur un exemple :

```
1 L=[3,12,'coucou',2+5j,6.2,'liste',8]
```

Comme dit précédemment, une liste peut contenir des listes ! Au passage, on peut définir une liste ne contenant rien : il suffit d'indiquer la fermeture immédiatement après l'ouverture de la liste.

```
M=[[1,1],[3],[3.1,3.4]] #Une liste de listes !

N=[] #Une liste vide
```

On peut appeler les variables `L`,`M` et `N` pour voir que Python comprend bien les listes comme on lui transmet :

```
L
M
N
```

On peut également définir une liste par compréhension de la même façon que les ensembles en mathématiques (enfin presque) : voici des exemples d'ensembles définis en compréhension et l'instruction qui permet de définir le liste qui leur correspond.

- $\{k^2 \mid k \in [\![0,5]\!]\}$ $\rightsquigarrow$ `[k**2 for k in range(6)]`
- $\{t \in [\![-6,6]\!] \mid t^4 \leq 70\}$ $\rightsquigarrow$ `[t for t in range(-6,7)if t**4<=70]`

Pour accéder aux éléments, on utilise leurs indice que l'ont désigne, comme on l'a vu pour les chaînes de caractères, immédiatement après le nom de la liste et entouré par des crochets. **ATTENTION :** Le **1er** élément d'une liste a pour indice **0**, le 2eme $\rightsquigarrow$ 1, etc...

Faisons un essai avec les listes qu'on a définies précédemment :

```
L[0]
L[2]
M[2]
M[2][0]
```

Les autres techniques que l'on a apprises pour l'accès aux chaînes de caractères sont aussi valables pour les listes : accès par indices négatifs, slicing et longueur de liste via la fonction `len`

```
L[-2] #Renvoie l'avant dernier élément de L
len(L) #Renvoie la longueur de L
L[2:4] #Renvoie les éléments d'indice 2 à !! 3 !! (=4-1)
M[2][0]
```

On peut alors parcourir une liste grâce à une boucle `for`. Il faut bien noter que `range(len(L))` énumère tous les indices de la liste `L` : les nombres de `0` à `len(L)-1`.
On rappelle bien sûr que `range(k)` est une énumération des nombres de 0 à $\mathbf{k-1}$.
Ici, on **affiche dans le shell** les éléments de `L` un par un :

```
for i in range(len(L)):
    print(L[i])
```

Une liste étant un objet énumérable pour Python (comme les chaînes de caractères), on peut la parcourir directement :

```
for element in L:
    print(element)
```

Dernier point de ce paragraphe, on peut créer une liste de nombre grâce à la fonction `range` et la fonction `list`. Le résultat de l'appel d'une fonction `range` **N'EST PAS** une liste pour Python ; pour transformer une énumération obtenue avec `range`, il faut lui appliquer la fonction `list` Voyons ceci sur quelques exemples :

```
list(range(5))
list(range(4,13))
list(range(3,-1))
list(range(1,30,2))
list(range(10,0,-1))
```

**Exercice 1.**

1. Définir une liste `I` contenant dont les éléments sont les chaînes de caractères suivantes (n'oubliez pas les espaces) `'je '`, `'déteste'`, `'suis fan de '` et `"l'Informatique"`
2. En utilisant l'accès aux éléments de `I` par leurs indices et grâce à la concaténation des chaînes de caractères, écrire la phrase `"je déteste l'Informatique"` puis la phrase `"je suis fan de l'Informatique"`.
3. Créer une liste `N` contenant les nombres allant de 1 à 82 de 3 en 3 i.e. $1, 4, 7, ..., 79, 82$
4. **Afficher** un par un chacun des nombres de cette liste.
5. Donner l'instruction qui permet de définir la liste `C` représentant l'ensemble suivant - ne pas oublier d'importer le module `math` ou le module `numpy` pour le cosinus (`cos`) :

$$\{x \in [\![0, 100]\!] \mid \cos(x) \geq 0\}$$

### b. Opérations sur les listes

Première opération, comme pour les chaînes de caractères : **la concaténation**. Grâce à l'opérateur `+`, on peut concaténer deux listes, c'est-à-dire les mettre bout à bout pour former une deuxième liste.
Observons le résultat de l'instruction suivante :

```
[3,4,5]+[1,1,1,1,2]
```

On peut également dupliquer (c'est-à-dire concaténer un certain nombre de fois) une liste autant que l'on veut pour former une nouvelle liste grâce à l'opérateur `*` : Pour `L` une liste et `k` un entier, `L*k` renvoie

la concaténation de k fois la liste L i.e. L+L+...+L.
Voyons ceci sur un exemple :

```
1 [1,6,8]*4
```

### c. Modification d'une liste

**Modifier un élément :**

Si L est une liste, alors L[k]=x remplace l'élément d'indice k par x.

```
1 L=[1,2,3,4]
2 L[2]=15
3 L
```

On peut remplacer une portion entière de liste grâce au slicing. Attention la portion à remplacer et la nouvelle liste qui la remplace doivent avoir la même taille !

```
1 L=list(range(11)) #liste des nombres de 0 à 10
2 L[2:5]=[12,34,61] #remplace [2,3,4] par [12,34,61]
3 L
```

**Copier une liste :**

ATTENTION ici ! La copie d'une liste est délicate. Autant pour copier une variable, rien de plus simple : si je veux copier la variable x dans une variable y, il me suffit de faire y=x. Puis si je modifie la valeur de y, cela n'affecte pas x.
**ATTENTION !!!!!** En Python, ceci ne fonctionne pas pour les listes ! L'instruction M=L ne fais pas une simple copie de la liste L, elle créé un *alias* de cette liste vers la liste M c'est-à-dire que toute modification de M affecte L et inversement !! Testons dans le shell :

```
1 L=[1,2,3,4]
2 M=L
3 L[0]=4
4 L
5 M
6
7 M[1]=1
8 L
9 M
```

Ainsi pour créer une copie indépendante d'une liste L, on doit utiliser la **méthode** copy() de l'objet L. Observons sur un exemple :

```
1 L=[1,2,3,4]
```

```
2 M=L.copy()
3 L[0]=4
4 L
5 M
```

*Aparté :* La méthode `copy()` ne fonctionne pas pour les sous-listes d'une liste... ainsi, si on modifie les sous-listes d'une liste dans un original, elles le seront aussi dans la copie... même avec la méthode `copy`. Pour pallier à ce problème, on utilise le module `copy` et la **fonction** `deepcopy` :

```
1 from copy import deepcopy
2 L=[[1,2],[3,4]]
3 M=deepcopy(L)
4 L[0][1]=4
5 L
6 M
```

**Supprimer un élément :**

Pour supprimer un élément d'une liste, on utilise la fonction `del`. Cela fonctionne également avec une portion de liste.
Testons :

```
1 L=[1,5,7,7,8,9,12,12]
2 del L[-1]
3 L
4
5 del L[2:4]
6 L
```

Bien sûr, la longueur de la liste diminue après l'utilisation de `del`

```
1 L=[1,5,7]
2 len(L)
3
4 del L[0]
5 len(L)
```

**Insertion d'un élément dans une liste :**

L'insertion d'un élément dans une liste peut se faire grâce à deux **méthodes** de la classe `list` :

— La méthode `append(x)` qui ajoute l'élément `x` à la fin de la liste ;
— La méthode `insert(i,x)` qui insère l'élément `x` à l'index `i` (et donc décale l'indice des éléments suivants).

Observons :

```
L=['aa','ee','ii','oo']
L.append('uu')
L

L.insert(2,'eeeiii')
L
```

**Autres méthodes :**

Voici quelques autres méthodes intéressantes pour interagir avec les listes :

- La méthode `remove(x)` supprime la première occurrence de `x` de la liste ;
- La méthode `pop(k)` supprime l'élément d'indice `i` et **retourne l'élément** en question ;
- La méthode `reverse()` inverse l'ordre des éléments de la liste ;
- La méthode `sort()` trie dans l'ordre croissant (si c'est possible) la liste ;

Exemples :

```
L=['z','r','t','a','b','g','d','p','k','h']
L.remove('b')
L

L.pop(6)
L

L.reverse()
L

L.sort()
L
```

**Exercice 2.**

1. Créer une liste `L` qui contient 250 chiffre 0 puis 342 chiffres 1 (la liste doit ressembler à `[000...00111...11]`).
2. Créer une liste `N` qui contient les nombres de 2 à 16. Remplacer le nombre 6 (chercher son indice) de la liste par 1000.
3. En utilisant la liste `N` modifiée dans la question précédente et une boucle `for`, remplacer tous les éléments de `N` par leur carré.
4. Ajouter dans `N` le nombre 33 entre les nombres 4 et 9.
5. Trier la liste `N` dans l'ordre décroissant.

### d. Quelques autres techniques

- La méthode `count(x)` compte le nombre de fois qu'apparaît `x` dans la liste ;
- La méthode `index(x)` retourne l'index de la première occurrence de `x` dans la liste ;

— La fonction `min(L)` renvoie le minimum des éléments de la liste `x`

— La fonction `max(L)` renvoie le maximum des éléments de la liste `x`

— La fonction `sum(L)` renvoie la somme des éléments de la liste `x`

— La fonction `sorted(L)` renvoie une copie triée dans l'ordre croissant de la liste `L` (contrairement à la méthode `sort()`, cette fonction ne modifie pas `L`)

```
1  L=[3,6,3,4,1,2,2,99,4,45,2,1,2]
2  L.count(2)
3
4  L.index(1)
5
6  min(L)
7  max(L)
8  sum(L)
9
10 M=sorted(L)
11 L
12 M
```

Et pour finir ; pour applique une fonction à tous les éléments d'un tableau, on peut utiliser la méthode vue pour la définition d'une liste par compréhension. Voyons ceci sur un exemple : j'ai une liste de nombres et je veux mettre tous les nombres de la liste à la puissance 3 puis leur ajouter 12, voici comment faire :

```
1 L=[3,6,3,4,1,2,2,99,4,45,2,1,2]
2
3 def f(x):
4     return x**3+12
5 M=[f(elem) for elem in L]
6 M
```

**Exercice 3.**

On cherche à estimer les racines du polynômes :

$$x^3 - 33,94x^2 + 141,548875x + 49,622625$$

1. Créer une liste `I` contenant les nombres de $-30$ à $30$ avec un pas de $0.01$
2. Créer une liste `M` contenant les éléments de `L` auxquels on a appliqué la fonction $f : x \mapsto x^3 - 33,94x^2 + 141,548875x + 49,622625$
3. Créer une liste `S` qui contient les indices $i$ de `M` qui vérifient :

$$(M[i] \leq 0 \text{ et } M[i+1] > 0) \text{ ou } (M[i] \geq 0 \text{ et } M[i+1] < 0)$$

4. En déduire une approximation des racines du polynôme.

## 3. La mise en pratique !

Voici une liste d'exercices afin de mieux comprendre et manipuler les listes en Python :

**Exercice 4.**

1. Écrire une fonction `somme(L)` qui prend pour argument une liste `L` de nombres et qui *renvoie* la somme des éléments de `L`. Si la liste `L` est vide, le résultat de `somme(L)` devra être 0.
2. 
3. Écrire une fonction `produit(L)` qui prend pour argument une liste `L` de nombres et qui *renvoie* le produit des éléments de `L`. Si la liste `L` est vide, le résultat de `produit(L)` devra être 1.
4. Que calcule la fonction `mystere(n)` suivante ?

```
1 def mystere(n):
2     return produit(list(range(1,n+1)))
```

5. Écrire une fonction `moyenne(L)` qui prend pour argument une liste `L` de nombres et qui *renvoie* la moyenne des éléments de `L`.
6. Écrire une fonction `variance(L)` qui prend pour argument une liste `L` de nombres et qui *renvoie* la variance des éléments de `L`.

   La variance $V$ d'une liste $x_1, ..., x_n$ de nombres de moyenne $m$ est définie par :

$$V = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - m)^2.$$

**Exercice 5.**

Écrire une fonction `separation(L,chaine)` qui prend pour argument une liste de chaînes de caractères `L` et une chaîne de caractères `chaine` et qui renvoie une liste `[M,N]` de deux listes : la liste `M` doit contenir tous les mots situés avant (dans l'ordre lexicographique) `chaine` inclus et la liste `N`, les mots situés après.

**Exercice 6.** *Liste des termes d'une suite*

Soit $(u_n)$ la suite récurrente :

$$\begin{cases} u_0 \in [-1, +\infty[ \\ u_{n+1} = \sqrt{u_n + 1} \end{cases}$$

1. Justifier que cette suite est bien définie.
2. Écrire une fonction `liste_termes(n,initial)` qui renvoie la liste des $n$ premiers termes $u_0, ..., u_{n-1}$ de la suite $(u_n)$ ayant pour terme initial $u_0 =$`initial`.

3. De même, écrire une fonction qui renvoie la liste des termes de la suite récurrente

$$\begin{cases} v_1 = 1 \\ v_{n+1} = \frac{n^2}{(n+1)^2} v_n \end{cases}$$

puis calculer la somme des $n$ premiers termes de la suite pour $n = 100$, 1000, 10000, 100000, 1000000. Que constate-t-on ?

**Exercice 7.** *Crible d'Érathostène*

On cherche dans cet exercice à écrire une fonction qui renvoie la liste des nombres premiers plus petits qu'un certain nombre `n`. Pour cela, on va utiliser la méthode du crible d'Érathostène :
- on part de la liste des nombres allant de 2 à $n$ ;
- on supprime de cette liste tous les nombres divisibles par 2 ;
- dans cette nouvelle liste, on supprime tous les nombres divisibles par 3 ;
- 4 ayant été enlevé, on passe au nombres divisibles par 5 ;
- etc...

Écrire une fonction `crible(n)` qui renvoie la liste des nombres premiers compris entre 0 et $n$ grâce à la méthode du crible d'Érathostène.

Combien y-a-t-il de nombres premiers entre 0 et 100 ? 1000 ? 10000 ?

**Exercice 8.**

Importer le module `numpy.random` i.e.

```
1 from numpy.random import *
```

Ce module est un sous-bibliothèque de Numpy qui contient des fonctions qui vont nous permettre de générer des nombres aléatoires. Ici, c'est la fonction `randint` qui va nous intéresser. Comme son nom anglais l'indique, cette fonction permet de générer un entier (de type `int`) tiré aléatoirement (ou presque) entre deux bornes qui dépendent des arguments passés à la fonction : on a le même système que pour la fonction `range` :
- `randint(k)` renvoie un nombre tiré aléatoirement entre 0 et $k-1$ ;
- `randint(j,k)` renvoie un nombre tiré aléatoirement entre $j$ et $k-1$ ;

1. Écrire une fonction `ZeroUn(n)` qui renvoie une liste de longueur `n` et dont chaque élément est un entier tiré aléatoirement entre 0 et 1.
2. Écrire une fonction `MaxBandeDeZero(L)` qui détermine la taille de la plus grande plage de 0 consécutifs dans la liste `L` constituée de 0 et de 1.
3. Écrire une fonction qui permet de déterminer la moyenne de la taille de la plus grande plage de 0 consécutifs parmi un échantillon de `N` listes de longueur `n` contenant de 0 et des 1 tirés aléatoirement.

   Conjecturer les valeurs des moyennes théoriques de la plus grande plage de 0 pour des listes de taille 10, 50, 100, 250 et 500.

**Exercice 9.**

Le tri par sélection est un algorithme de tri : il permet de trier une liste de nombres du plus petit au plus grand (par ordre croissant). Voilà une description d'une étape à répéter de notre algorithme de tri

— on cherche le plus petit élément du tableau

— on l'échange avec le premier élément du tableau

— on recommence le processus avec le sous-tableau restant (i.e. le tableau moins le premier élément)

1. Écrire une fonction `indice_minimum(L,k)` qui prend pour arguments une liste `L` de nombres et un indice `k` et qui renvoie l'indice du minimum des éléments `L[k]`,...,`L[n-1]` (où `n` est la longueur du tableau).

2. Écrire une fonction `tri_selection(L)` qui prend pour argument une liste `L` de nombres et renvoie la liste triée grâce à l'algorithme décrit plus haut (et qui bien sûr utilise la fonction `indice_minimum`!).

**Exercice 10.** *Mélange d'un jeu de cartes*

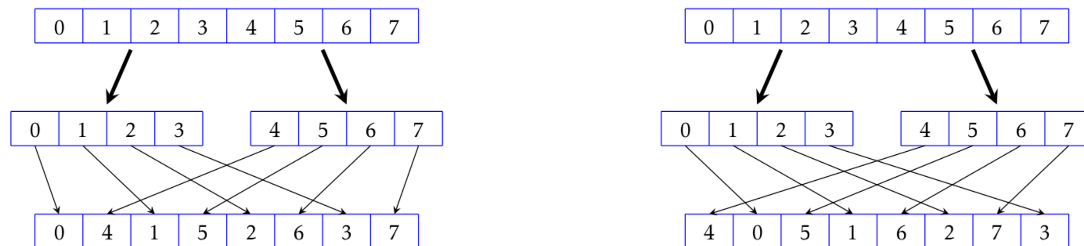
Une manière traditionnelle de mélanger un jeu de cartes consiste à couper le paquet en deux puis d'entrelacer les deux parties, comme on le voit souvent sur les tables de poker. Lorsque que le paquet de carte est coupée en deux parties égale, et que l'entrelacement se fait carte par carte, on dit que le mélange est parfait.

Il y a deux types de mélanges parfaits :

— Le **out shuffle** où l'entrelacement commence avec la première moitié du paquet ;

— Le **in shuffle** où l'entrelacement commence avec la deuxième moitié du paquet ;

Voici des exemples de out shuffle et in shuffle sur une liste de 8 éléments.




1. Soit `L` une liste de taille $n$ paire. Utiliser le slicing pour décrire :
   a. la première et la seconde moitié du tableau ;
   b. la liste contenant les éléments d'indices pairs, et la liste des éléments d'indices impairs

2. En déduire une fonction `out_shuffle(L)` qui renvoie la liste `L` de taille $n$ paire mélangée grâce à au out shuffle

3. En écrivant une fonction appropriée, déterminer au bout de combien de mélanges out shuffle une liste contenant les nombres de 1 à $n$ redevient elle même ?
   Application : donner le nombre de mélanges out shuffle d'un jeu de 52 cartes pour revenir à son état initial.

   Mêmes questions avec le in shuffle.