Python - Cours 2
Listes

I Listes et premières commandes

Une liste est une collection finie d'éléments pouvant être de types différents (entiers, flottants, chaînes de caractères, listes...) ; ils sont notés entre crochets et séparés par des virgules.

```
>>> L=[1,2,3]
>>> type(L)
>>> L==[1,3,2]     #L'ordre des éléments est important
>>> M=[1,2,"Truc", False, L]
>>> []     #Liste vide
>>> L[0]
>>> L[0]=0     #Permet de changer la valeur de l'élément indexé 0
>>> L[1]
>>> L[2]
>>> L[3]     #On accède aux éléments avec L[i], pour i allant de 0 à "nb d'éléments-1"
>>> L[-1]     #Numérotation négative possible... On parcourt alors la liste de droite à gauche.
>>> type(range(0,21))     #range() ne renvoie pas vraiment une liste, mais peut être converti en liste
>>> N=list(range(0,21))     #La commande list permet de convertir range() en liste
>>> N[2:6]     #Renvoie la liste composée des éléments indexés 2 à 6-1 de la liste N
>>> N[0:11:2]     #Renvoie la liste composée des éléments de N indexés de 0 à 10, de 2 en 2
>>> N[::2]     #Renvoie la liste composée des éléments de N indexés de 2 en 2
>>> N[::-1]     #Renvoie la liste donc l'indexation est inversée
>>> list("Truc")
```

Que dire des listes M et L après exécution de ces lignes ?

```
>>> L=list(range(0,11))
>>> M=L
>>> M[0]=43
```

Commentaires :

✘ Attention !
Attention donc à la copie d'une liste, qui reste liée à la liste initiale... Pour créer une copie indépendante d'une liste `L`, on peut écrire `M=L[:]` par exemple.

- La commande `len(L)` renvoie la longueur (*length* en anglais) de la liste `L`, c'est à dire son nombre d'éléments.
- La commande `in` permet : soit de parcourir les éléments d'une liste quand elle elle placée après une instruction `for`, soit de tester l'appartenance d'un élément à une liste.

Important !
L'indexation des éléments de `L` se fait de 0 à `len(L)-1`.

Python 1 – Parcourir une liste

```
1 L=[0,2,4,6,8,10]
2 for k in L:
3     x=k**2
4     print(x)
```

Petite remarque
Nous avions déjà vu ceci dans le premier cours, et nous le faisions déjà sur les chaînes de caractères.

```
>>> L=list(range(0,10))
>>> 8 in L
>>> 10 in L
```

- La commande `L.append(x)` ajoute, en fin de la liste `L`, l'élément `x`.

```
>>> L=[0,1,2]
>>> L.append(3)
```

- Les symboles + et * permettent : soit l'addition / la multiplication de nombres, soit la concaténation / la concaténation répétée de listes.

```
>>> L=[1,2,3]
>>> M=[4,5,6]
>>> L+M     #Renvoie la liste composée des éléments de la première puis ceux de la seconde
>>> M+L
>>> N=list(range(0,6))
>>> N+[5,5,5]
>>> L*3     #Renvoie la liste obtenue en concaténant 3 fois la liste L
```

✘ Attention !
Attention à l'ordre de la concaténation... Que nous avions déjà vue sur les chaînes de caractères.

Petite remarque
La commande * fonctionne de la même façon sur les chaînes de caractères (pour répéter un message par exemple).

Les commandes `L.append(x)` et `L=L+[x]` font donc exactement la même chose.

- La commande `del L[i]` supprime l'élément indexé `i` de la liste `L` et ré-indexe la liste.

```
>>> L=list(range(0,11))
>>> len(L)
>>> del L[4]
>>> len(L)
```

- La commande `L.count(x)` renvoie le nombre d'éléments de `L` ayant la valeur de `x`.

```
>>> L=[10,12,11,10,13,10]
>>> L.count(10)
>>> L.count(15)
```

- Si `L` est une liste de nombres, la commande `sum(L)` renvoie la valeur de la somme des éléments de `L`.
- Si `L` est une liste de nombres, les commandes `min(L)` et `max(L)` renvoient respectivement la valeur du minimum et du maximum des éléments de `L`.

Petite remarque
Si `L` est composée de chaînes de caractères, alors `min(L)` et `max(L)` renvoient respectivement les premiers et derniers éléments classés par ordre alphabétique.

II Modes de génération des listes

Dans le chapitre 0 du cours de mathématiques, nous avions vu qu'il y avait essentiellement deux façons différentes d'écrire un ensemble :

- **en extension :** cela revient à décrire les éléments qu'il contient : $E = \{1; 2; 3\}$
- **en compréhension :** cela revient à le définir à partir d'une condition : $E = \{x \in \mathbb{R} \,/\, 2x + 5 \geq 1\}$

C'est sensiblement identique pour générer des listes en `Python`. Pour générer une liste d'entiers consécutifs, on utilisera la commande `list(range(n,m))` qui génère la liste ordonnée des entiers de $[\![n; m-1]\!]$. Dans les autres cas, voici les méthodes possibles.

II.1 En extension

Cela revient en quelque sorte à écrire chaque élément de la liste ; ou à les ajouter à l'aide d'une boucle. Deux cas de figure :

```
>>> L=[0,1,4,9,16,25,36,49,64,81,100]
```

Python 2 – Liste en extension avec boucle `for`

```
L=[] #On commence par définir L, qui est une liste vide
for k in range(0,11):
    L.append(k**2) #On peut aussi écrire L=L+[k**2]
```

Petite remarque
Le principe est identique à ce que nous avions fait dans le TP1 pour calculer des sommes et produits. On commence soit avec une somme nulle (`S=0`, puis on ajoute tous les termes souhaités) soit avec un produit égal à 1 (`P=1`, puis on multiplie tous les facteurs souhaités).

II.2 En compréhension

Deux syntaxes possibles selon ce que l'on souhaite faire :

```
L=[f(x) for x in TRUC]

L=[f(x) for x in TRUC if x ...]
(permet de rajouter une condition à x pour qu'il soit ajouté à L)
```

où TRUC est soit une liste, soit une commande `range()`

```
>>> L=[k**2 for k in range(0,11)]
>>> M=[k for k in L]     #Permet de copier les éléments de la liste L dans M, sans créer de dépendance entre les listes
>>> [k**2 for k in range(0,11) if k!=2]
>>> L=[k**2 for k in range(0,11) if k!=2 if k!=3]
>>> [c*2 for c in "Test"]
>>> [i+j for i in range(0,3) for j in range(0,6)]
>>> [i+j for j in range(0,6) for i in range(0,3)]
>>> [i+j for i in range(0,3) for j in range(0,3) if i>=j]
```

●○○○ Exercice 1 - $\sum_{k=0}^{n} k^4$

1. Créer une liste L qui contient les nombres k^4, pour $k \in [\![0;n]\!]$, après avoir demandé à l'utilisateur de saisir un valeur pour l'entier naturel n.
2. En déduire un programme permettant de calculer $\sum_{k=0}^{n} k^4$, pour une valeur de n saisie par l'utilisateur.

```
n=int(input("Entrer une valeur d'un entier naturel n : "))
L=[k**4 for k in range(0,n+1)]
S=sum(L)
print(L)
print(S)
```

●○○○ Exercice 2 - Suite

Considérons la suite (u_n) définie par : $\begin{cases} u_0 = 0 \\ \forall n \in \mathbb{N},\ u_{n+1} = e^{-u_n} \end{cases}$. Créer une fonction Python telle que l'exécution de `listeU(n)` renvoie une liste contenant les termes u_0 à u_n de la suite (u_n).

```
import numpy as np

def listeU(n):
    u=0
    L=[u]
    for k in range(1,n+1):
        u=np.exp(-u)
        L.append(u) # ou L=L+[u]
    return L
```

●○○○ Exercice 3 - Inverse

Écrire une fonction qui prend en argument d'entrée une liste de réels et renvoie la liste obtenue en calculant, si possible, l'inverse de chaque réel de la liste.

```
def inverse(L):
    M=[1/k for k in L if k!=0]
    return M
```

●○○○ Exercice 4 - Indexation inversée

Écrire une fonction qui prend en argument d'entrée une liste et qui renvoie la liste obtenue en inversant l'ordre des éléments.

> **Petite remarque**
> La commande `L[::-1]` permet d'obtenir la liste inversée, mais il s'agit ici de ne pas lutiliser.

```
def listeinversee(L):
    M=[]
    for k in L:
        M=[k]+M #permet d'ajouter l'élement par la gauche
    return M
```

●●○○ Exercice 5 - Maximum dans une liste

Écrire une fonction telle que l'exécution de la commande `maximum(L)` renvoie le maximum des nombres contenus par la liste L.

> **Petite remarque**
> La commande `max(L)` permet d'obtenir le maximum, mais il s'agit ici de ne pas lutiliser (ni `min`).

```
def maximum(L):
    m=L[0]
    for x in L: #on parcourt les éléments (on peut aussi parcourir les indices)
        if x>m:
            m=x
    return m
```

●●○○ Exercice 6 - Échange dans une liste

Écrire une fonction qui prend en arguments d'entrée une liste ainsi que deux indices i et j, puis qui renvoie la nouvelle liste obtenue après échange des éléments d'indices i et j.

```
def echange(L,i,j): #programme qui modifie la liste L initiale...
    L[i],L[j]=L[j],L[i]
    return L

def echange_bis(L,i,j): #ne modifie pas la liste initiale
    M=L[:] #copie sans créer de dépendance entre M et L
    M[i],M[j]=M[j],M[i]
    return M
```

●●○○ Exercice 7 - Suppression

Écrire une fonction qui prend en arguments d'entrée une liste et un élément x, puis qui renvoie la liste obtenue après suppression de toutes les occurrences de x dans cette liste.

```
def suppression(L,x):
    M=[k for k in L if k!=x] #nouvelle liste avec les éléments différents de x
    return M

def suppression_bis(L,x):
    M=[k for k in L if k!=x] #nouvelle liste avec les éléments différents de x
    L=M #on modifie la liste initiale
    return L
```

Petite remarque

Le programme "naïf" consistant à supprimer `L[k]` lorsqu'il est égal à `x` pose souci... Et autant profiter des spécificités de Python !

●●○○ Exercice 8 - Insertion

Écrire une fonction qui prend en arguments d'entrées une liste, un indice i et un élément x, puis qui renvoie la liste obtenue après insertion de l'élément x à l'indice i, en décalant le reste vers la droite.

Petite remarque

La commande `L.insert(i,x)` permet de le faire, mais elle n'est pas à connaître.

```
def insertion(L,i,x):
    L1=L[0:i]
    L2=L[i:len(L)]
    L=L1+[x]+L2 #on modifie la liste initiale
    return L
```

●●○○ Exercice 9 - Ordonner une liste de 3 éléments

Écrire une fonction qui prend en arguments d'entrée trois réels et qui renvoie la liste de ces trois nombres dans l'ordre croissant.

Petite remarque

La commande `L.sort()` permet d'ordonner une liste, mais il s'agit ici de ne pas lutiliser.

```
#Avec 3 valeurs
def listeordonnee(a,b,c):
    if b<a:
        a,b=b,a
    if c<b:
        b,c=c,b
    if b<a:
        a,b=b,a
    L=[a,b,c]
    return L
```

```
#Avec un nombre quelconque de valeurs
def listeordonnee(L):
    print(L)
    for k in range(1,len(L)):
        for j in range(0,k):
            if L[j]>L[k]:
                L[j],L[k]=L[k],L[j]
                print(L)
    return L
```

●●●○ Exercice 10 - Somme double

1. Écrire un programme qui demande la valeur d'un entier naturel n à l'utilisateur et affiche ensuite la valeur de $\sum_{0\leq i<j\leq n} (i+j)$.

```
n=int(input("Entrer une valeur de n : "))
L=[i+j for i in range(0,n) for j in range(i+1,n+1)]
print(sum(L))
```

Petite remarque

i va ainsi de 0 à $n-1$...

2. Pour tout $n \in \mathbb{N}^*$, calculer $\sum_{0\leq i<j\leq n} (i+j)$.

Soit $n \in \mathbb{N}^*$. On a :

$$\begin{aligned}
\sum_{0\leq i<j\leq n}(i+j) &= \sum_{i=0}^{n-1}\sum_{j=i+1}^{n}(i+j) \\
&= \sum_{i=0}^{n-1}(n-i)i + \sum_{j=i+1}^{n} j \\
&= \sum_{i=0}^{n-1}(n-i)i + (n-i)\frac{i+1+n}{2} && \text{somme des termes d'une SA} \\
&= \sum_{i=0}^{n-1}(n-i)\frac{3i+1+n}{2} && \text{changement d'indice } k = n-i \\
&= \sum_{k=1}^{n} k\frac{4n+1-3k}{2} \\
&= 2n\sum_{k=1}^{n}k + \frac{1}{2}\sum_{k=1}^{n}k - \frac{3}{2}\sum_{k=1}^{n}k^2 \\
&= n^2(n+1) + \frac{n(n+1)}{4} - \frac{n(n+1)(2n+1)}{4} \\
&= \frac{n(n+1)(4n+1-(2n+1))}{4} \\
&= \frac{n^2(n+1)}{2}
\end{aligned}$$