Python 1
Rappels et algorithmes classiques sur les 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.

- Exécuter successivement les lignes suivantes dans la console.

```
>>> L=[1,17,43]
>>> type(L)
>>> L==[1,43,17]     #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
>>> 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
```

Les listes `L` et `M` sont toutes deux modifiées ! Il faudra donc procéder autrement si l'on souhaite copier une liste, puis modifier la copie sans modifier la liste initiale.

✘ 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`, autrement dit 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 (elle renvoie alors **`True`** ou **`False`** selon l'appartenance ou non).

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

```
>>> 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.

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`.

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

En mathématiques, il y a 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 : $E = \{x \in \mathbb{R} \ / \ 2x + 5 \geqslant 1\}$

C'est sensiblement identique pour générer des listes en `Python`...

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` 1 – 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]
```

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]
```

III Exercices

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 $(u_n)_{n\in\mathbb{N}}$ 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 $(u_n)_{n\in\mathbb{N}}$.

```
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 - ●○○ - Liste inversée

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

Remarque
La commande `L[::-1]` permet d'obtenir la liste inversée, mais il s'agit ici de ne pas l'utiliser.

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

Exercice 4 - ●○○ - Suppression

Écrire une fonction qui prend en arguments d'entrée une liste et un élément x, puis qui renvoie la nouvelle 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
```

Exercice 5 - ●○○ - Insertion

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

> **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] #ou L1=[L[k] for k in range(0,i)]
    L2=L[i:len(L)] #ou L2=[L[k] for k in range(i,len(L))]
    L=L1+[x]+L2 #on modifie la liste initiale
    return L
```

Exercice 6 - ●●○ Occurrences...

1. Écrire une fonction prenant en arguments d'entrée une liste `L` et un élément `x`, puis renvoyant en sortie le nombre d'occurrences de `x` dans `L`.

> **Remarque**
> Sans utiliser `L.count(x)` qui renvoie directement le résultat demandé.

```
def nombreocc(L,x):
    c=0
    for e in L:
        if e==x:
            c=c+1
    return c
```

2. Écrire une fonction prenant en arguments d'entrée une liste `L` et un élément `x`, puis renvoyant en sortie le rang de la première occurrence de `x` dans `L` si la liste `L` contient l'élément `x`, et `False` sinon.

```
def premiereocc(L,x):
    for k in range(0,len(L)):
        if L[k]==x:
            return k
    return False
```

3. Écrire une fonction prenant en arguments d'entrée une liste `L` et un élément `x`, puis renvoyant en sortie le rang de la dernière occurrence de `x` dans `L` si la liste `L` contient l'élément `x`, et `False` sinon.

```
def derniereocc(L,x):
    rang=False
    for k in range(0,len(L)):
        if L[k]==x:
            rang=k
    return k

def dernieroccbis(L,x):
    return len(L)-1-premiereocc(L[::-1],x)
```

Exercice 7 - ●●○ Maximum

> **Remarque**
> Sans utiliser `L.sort()` qui permet d'ordonner la liste `L`...

1. Sans utiliser la commande `max`, écrire une fonction prenant en argument d'entrée une liste de réels `L` puis renvoyant en sortie le maximum de `L`.

```
def monmax(L):
    maxi=L[0]
    for x in L:
        if x>maxi:
            maxi=x
    return maxi

def monmaxbis(L): #ou en parcourant les rangs
    maxi=L[0]
    for k in range(0,len(L)):
        if L[k]>maxi:
            maxi=L[k]
    return maxi
```

2. Sans utiliser la commande `max`, écrire une fonction prenant en argument d'entrée une liste de réels `L` puis renvoyant en sortie le maximum de `L` ainsi que son nombre d'occurrences.

```
def monmax(L):
    maxi=L[0]
    c=0
    for x in L:
        if x>maxi:
            maxi=x
            c=1
        elif x==maxi:
            c=c+1
    return maxi,c
```

3. Sans utiliser la commande `max`, écrire une fonction prenant en argument d'entrée une liste de réels `L` puis renvoyant en sortie le maximum de `L` ainsi que le rang de sa première occurrence.

```
def monmax(L):
    maxi=L[0]
    rang=0
    for k in range(1,len(L)):
        if L[k]>maxi:
            maxi=L[k]
            rang=k
    return maxi,rang
```

4. Sans utiliser la commande `max`, écrire une fonction prenant en argument d'entrée une liste de réels `L` puis renvoyant en sortie les deux plus grands nombres (éventuellement égaux) de cette liste.

```
def deux_plus_grands(L):
    if L[0]<L[1]:
        maxi1,maxi2=L[1],L[0]
    else:
        maxi1,maxi2=L[0],L[1]
    for k in L[2:len(L)]:
        if k>maxi1 and k>maxi2:
            maxi1,maxi2=k,maxi1
        elif maxi2<k<=maxi1:
            maxi2=k
    return maxi2
```

5. Sans utiliser la commande `max`, écrire une fonction prenant en argument d'entrée une liste de réels `L` dont au moins deux sont distincts puis renvoyant en sortie le second plus grand maximum (distinct du premier) de cette liste.

```
def second_max(L):
    k=1
    while L[k]==L[0]:
        k=k+1
    if L[0]<L[k]:
        maxi1,maxi2=L[k],L[0]
    elif L[0]>L[k]:
        maxi1,maxi2=L[0],L[k]
    for x in L[2:len(L)]:
        if x>maxi1 and x>maxi2:
            maxi1,maxi2=x,maxi1
        elif maxi2<x<maxi1:
            maxi2=x
    return maxi2
```

Exercice 8 - ●●○ Les plus proches

Écrire une fonction prenant en argument d'entrée une liste de réels `L` (contenant au moins deux éléments) puis renvoyant en sortie les deux éléments les plus proches.

```
def proches(L):
    ecart=abs(L[1]-L[0])
    a,b=L[0],L[1]
    for i in range(0,len(L)):
        for j in range(i+1,len(L)):
            if abs(L[i]-L[j])<ecart:
```

```
7                   ecart=abs(L[i]-L[j])
8                   a,b=L[i],L[j]
9       return a,b
```

Remarque

En mettant `if 0<abs(L[i]-L[j])<ecart` en ligne 6, on obtient les deux plus proches distincts.

Exercice 9 – ●●● – Ordonner une liste...

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

```
1  def listeordonnee(L):
2      for k in range(1,len(L)):
3          for j in range(0,k):
4              if L[j]>L[k]:
5                  L[j],L[k]=L[k],L[j]
6      return L
```

Remarque

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