TP 1 Correction - Listes et tableaux

1 Minimum et du maximum d'une liste

On rappelle qu'on définit une liste `L` en Python grâce à la commande `L=[1,2,4,5]` par exemple. On accède à la composante numéro i de la liste (attention : on commence à compter à 0) par la commande `L[i]`. Par exemple, ici, `L[1]=2`.

1. Écrire une fonction `Maximum` qui prend en paramètre une liste de réels `L` et qui renvoie son maximum (sans utiliser la fonction `max` de Python).

```
def Maximum(L):
    M=L[0]
    for k in range(len(L)):
        if L[k]>M:
            M=L[k]
    return(M)
```

2. Faire de même une fonction `Minimum`.

```
def Minimum(L):
    m=L[0]
    for k in range(len(L)):
        if L[k]<m:
            m=L[k]
    return(m)
```

3. Écrire une fonction `PositionMax` qui renvoie la position du maximum dans la liste. S'il apparaît plusieurs fois, le programme doit renvoyer la liste des positions. Par exemple, pour `L=[1,2,0,1,2]`, on doit obtenir `[1,4]`.

```
def PositionMax(L):
    M=Maximum(L)
    T=[]
    for k in range(len(L)):
        if L[k]==M:
            T.append(k)
    return(T)
```

2 Statistiques d'une liste

4. Écrire une fonction `Occurence(L,x)` qui renvoie le nombre d'occurences (*le nombre d'apparitions*) de `x` dans la liste `L`.

```
 def Occurence(L,x):
    c=0 #compteur du nombre d'occurences
    n=len(L)
    for k in range(n):
        if L[k]==x:
            c+=1
    return(c)
```

5. On dispose d'une liste `L` regroupant les années de naissance d'un groupe de personnes. Créer une fonction `ClasseEffectif` qui prend en entrée la liste `L` et qui renvoie deux listes : la liste des classes (liste des années de naissance présentes, où elles n'apparaissent qu'une fois) et la liste des effectifs (liste du nombre de personnes du groupe nées cette année). Par exemple, si `L=[2004,2004,2003,2005,2004,2003,2004]`, la commande `ClasseEffectif(L)` doit renvoyer `[2004,2003,2005] [4,2,1]`.

```
def ClasseEffectif(L):
    #On crée nos listes de classes et d'effectifs
    C=[]
    E=[]
    for k in range(len(L)):
        if L[k] not in C:
            C.append(L[k])
            E.append(Occurence(L,L[k]))
    return C,E
```

6. Créer une fonction `Moyenne` qui prend en entrée une liste de réels `L` et renvoie la moyenne des valeurs de la liste.

```
def Moyenne(L):
        n=len(L)
        s=0 #on initialise la somme des termes à 0
        for k in range(n):
            s+=L[k]
        return s/n
```

On peut aussi utilise la fonction `sum` de Python.

```
def Moyenne2(L):
        return sum(L)/len(L)
```

7. L'utiliser pour créer une fonction `Variance` qui prend en entrée `L` et renvoie la variance des valeurs de la liste.

```
def Variance(L):
        n=len(L)
        m=Moyenne(L)
        s=0 #on initialise la somme à 0
        for k in range(n):
            s+=(L[k]-m)**2
        return s/n
```

La variance peut aussi être calculée en utilisant la formule donnée par le théorème de Kœnig-Huygens, à savoir :

$$\mathrm{V}(x) = \frac{1}{n}\sum_{k=1}^{n} x_k^2 - \text{moyenne}^2$$

On peut alors calculer la variance comme suit :

```
def Variance2(L) :
     L_carre = [element**2 for element in L]
     return Moyenne(L_carre)-Moyenne(L)**2
```

3 Fonctions usuelles sur les listes

8. Reprogrammer la fonction `index`. En Python, la commande `L.index(x)` renvoie l'indice de la première occurence de `x` dans la liste `L`. Si `x` n'est pas présent dans la liste `L`, la fonction renvoie `x not in the list`. On souhaite obtenir le même résultat en entrant la commande `indexbis(L,x)`.
Pour cette fonction, on introduit un compteur c qui va nous permettre de nous arrêter dès qu'on a rencontré la première occurence de `x`, mais aussi de savoir qu'on n'a pas rencontré de x si c'est le cas.

```
def indexbis(L,x):
    c=0
    i=0
    n=len(L)
    while c==0 and i<n:
        if L[i]==x:
            c=1
        else:
            i+=1
    if c==1:
        return(i)
    else:
        return(str(x)+' not in the list')
```

9. Reprogrammer la fonction `sum`. En Python, la commande `L.sum()` renvoie la somme des éléments de la liste `L`. On souhaite obtenir le même résultat en entrant la commande `sumbis(L)`.
Pour la fonction `sum`, on crée une variable `s` qu'on va modifier en lui ajoutant progressivement tous les éléments de la liste de nombres `L`.

```
def sumbis(L):
    s=0
    n=len(L)
    for k in range(n):
        s+=L[k]
    return(s)
```

10. Reprogrammer la fonction `remove`. En Python, la commande `L.remove(x)` modifie la liste `L` en lui retirant la première occurence de l'élément `x`. On souhaite que la commande `removebis(L,x)` nous renvoie la liste ainsi obtenue.
Pour la fonction `remove`, on construit une liste `T` qui contient les mêmes éléments que `L` jusqu'à ce qu'on rencontre le premier `x`. On "saute" alors cet élément et on termine la construction de la liste `T`.

```
def removebis(L,x):
    n=len(L)
    c=0
    i=0
    T=[]
    while c==0 and i<n:
        if L[i]==x:
            c=1
            for k in range(i+1,n):
                T.append(L[k])
        else:
            T.append(L[i])
            i+=1
    if c==1:
        return(T)
    else:
        return(str(x)+' not in the list')
```

11. Reprogrammer la fonction `reverse`. En Python, la commande `L.reverse()` modifie la liste `L` en inversant l'ordre de ses composantes. On souhaite que la commande `reversebis(L)` nous renvoie la liste ainsi obtenue.

```
def reversebis(L):
    n=len(L)
    T=[]
    for k in range(n):
        T.append(L[n-1-k])
    return(T)
```

12. Reprogrammer la fonction `choice` du module random. En Python, la commande `rd.choice(L)` (*si on a importé le module random comme* `rd`) renvoie un nombre au hasard choisi parmi les éléments de la liste `L`. On souhaite obtenir le même résultat en entrant la commande `choicebis(L)`.
Pour la fontion `choice`, on pioche au hasard un indice i puis on retourne `L[i]`.

```
import random as rd
def choicebis(L):
    n=len(L)
    i=rd.randint(0,n-1)
    return(L[i])
```

4 Autour de la notion de parité

On rappelle que si `a` et `b` sont des entiers naturels (avec `b` non nul), la commande `a%b` permet d'obtenir le reste de `a` dans la division euclidienne par `b`. Par exemple, `7%2` vaut `1` (puisque $7 = 2 \times 3 + 1$).

13. Écrire une fonction booléenne `NbPair` qui prend en entrée une liste de réels `L` et qui détermine si le nombre composantes de cette liste est pair.

```
def NbPair(L):
        return len(L)%2==0
```

14. Écrire une fonction booléenne `EstPair` qui prend en entrée une liste d'entiers et qui détermine si tous les éléments de cette liste sont pairs.

```
def EstPair(L):
    for k in range(len(L)):
        if L[k]%2==1:
            return False
    return True
```

15. Écrire une fonction `Creer` qui prend en entrée un entier `n` et qui renvoie la liste de longueur $2n$ dont les termes d'indices pairs sont des 1 et les autres une suite arithmétique de raison 2 et de premier terme 1. Par exemple, `Creer(3)` renvoie `[1,1,1,3,1,5]`.

```
def Creer(n):
    L=[]
    u=1
    for k in range(2*n):
        if k%2==0:
            L.append(1)
        else:
            L.append(u)
            u+=2
    return(L)
```

5 Produit scalaire de deux vecteurs de $\mathbb{R}^n$

On rappelle que si $u = (x_1, ..., x_n)$ et $v = (y_1, ..., y_n)$ sont deux vecteurs de $\mathbf{R}^n$ le produit scalaire de u et v est donné par

$$u.v = \sum_{i=1}^{n} x_i y_i.$$

16. Créer une fonction `ProdScalaire` qui prend en entrée deux listes de taille n de réels représentant les coordonnées de deux vecteurs u et v de $\mathbb{R}^n$ et qui renvoie leur produit scalaire.
Voilà un programme qui affiche un message d'erreur si les listes ne sont pas de même taille et renvoie le produit scalaire des vecteurs correspondants sinon.

```
def ProdScalaire(L1,L2):
    n=len(L1)
    m=len(L2)
    if n!=m:
        return('Taille de vecteurs incompatibles')
    else:
        p=0
        for k in range(n):
            p+=L1[k]*L2[k]
        return(p)
```

6 Tableaux

En Python, un tableau est codé comme une liste de liste. Par exemple, le tableau suivant correspond à la liste de listes `L` :

1	2	3	4
-1	-2	-3	-4
2	6	4	8

`L=[[1,2,3,4],[-1,-2,-3,-4],[2,6,4,8]]`

Les actions fondamentales sont les suivantes :

— on accède au coefficient situé en ligne `i` et colonne `j` par la syntaxe `L[i][j]`.

— on accède au nombre de lignes du tableau par `len(L)`

— on accède au nombre de colonnes par `len(L[0])`.

17. Écrire une fonction `EstPresent(L,x)` qui renvoie un booléen selon que `x` est présent ou non dans le tableau `L`.

```
def EstPresent(L,x):
    for i in range(len(L)): #on parcourt les lignes
        for j in range(len(L[0])): #on parcourt les colonnes
            if L[i][j]==x:
                return True
    return False
```

18. (a) Écrire une fonction `MoyenneLigne(L)` qui renvoie la liste contenant les moyennes de chaque ligne du tableau `L`.

```
def MoyenneLigne(L):
    M=[]
    for i in range(len(L)): #on parcourt les lignes
        c=0 #pour chaque ligne on initialise la somme à 0
        for j in range(len(L[0])): #on parcourt les colonnes
            c+=L[i][j]
        M.append(c/len(L[0]))
    return M
```

(b) Écrire une fonction `MoyenneColonne(L)` qui renvoie la liste contenant les moyennes de chaque colonne du tableau `L`.

```
def MoyenneColonne(L):
    M=[]
    for j in range(len(L[0])): #on parcourt les colonnes
        c=0 #pour chaque colonne on initialise la somme à 0
        for i in range(len(L)): #on parcourt les lignes
            c+=L[i][j]
        M.append(c/len(L))
    return M
```

19. Écrire une fonction `Min(L)` qui renvoie le minimum du tableau `L`.

```
def Min(L):
    m=L[0][0]
    for i in range(len(L)): #on parcourt les lignes
        for j in range(len(L[0])): #on parcourt les colonnes
            if L[i][j]<m:
                m=L[i][j]
    return m
```

20. Écrire une fonction `PositionMin(L)` qui renvoie la position du minimum dans le tableau `L`.

```
def PositionMin(L):
    m=L[0][0]
    pos=[0,0]
    for i in range(len(L)):
        for j in range(len(L[0])):
            if L[i][j]<m:
                m=L[i][j]
                pos=[i,j]
    return pos
```

21. Écrire une fonction `Frequence(L)` qui renvoie la liste des fréquences d'apparition de chacun des entiers compris entre -4 et 4 dans `L`. Sur le tableau donné en exemple, on attend en sortie `[0.083, 0.083, 0.083, 0.083, 0.0, 0.083, 0.17, 0.083, 0.17]`.

```
def Frequence(L):
    F=[]
    N=len(L)*len(L[0]) #Nb de cases du tableau
    for x in range(-4,5):
        c=0 #on initialise le compteur à 0
        for i in range(len(L)): #on parcourt les lignes
            for j in range(len(L[0])): #on parcourt les colonnes
                if L[i][j]==x:
                    c+=1
        F.append(c/N)
    return F
```

22. Écrire une fonction `LignePaire(L)` qui renvoie une liste de booléens précisant si les lignes du tableau `L` sont constituées uniquement d'entiers pairs ou non. Sur le tableau donné en exemple, on attend en sortie `[False,False,True]`.

```
def LignePaire(L):
    P=[]
    for i in range(len(L)):
        c=True
        j=0
        while c==True and j<len(L[0]):
            if L[i][j]%2==1:
                c=False
            j+=1
        P.append(c)
    return P
```

23. Écrire une fonction `Produit(L1,L2)` qui à partir de deux listes renvoie le tableau constitué des produits des élements de `L2` par ceux de `L1`. Par exemple, `Produit([2,3],[4,5,6])` doit renvoyer `[[8,10,12],[12,15,18]]`.

```
def Produit(L1,L2):
    T=[]
    for i in range(len(L1)):
        I=[]
        for j in range(len(L2)):
            I.append(L1[i]*L2[j])
        T.append(I)
    return T
```