Corrigé du TP 05 Listes

Chenevois-Jouhet-Junier

1 Manipulations des tableaux/listes; slicing et compréhension

```
import math, random #imports de bibliothèques/modules utiles pour le TP
```

1.1 Exercice 1

```
t1=[10,30,42,2,17,5,30,-20]

t2 = []
for i in range(-3, 6):
    t2.append(i**2)

#t21 identique à t2
t21 = [i**2 for i in range(-3,6)]

t3 = []
for i in range(1000):
    if (i%5) in [0, 2, 4]:
        t3.append(i**3)

#t31 identique à t3
t31 = [i**3 for i in range(1000) if (i%5) in [0,2,4]]

t4 = [841.0]
for i in range(20):
    t4.append(t4[i]/3+28)



"""
>>> for i in ['0','00','000','1','2','3','4']:
...     L = eval('l'+i)
...     print 'Liste %s de longueur %s'%('l'+i,len(L))
...
Liste l0 de longueur 15
Liste l00 de longueur 15
Liste l000 de longueur 15
Liste l1 de longueur 8
Liste l2 de longueur 9
```

```
Liste l3 de longueur 600
Liste l3 de longueur 21
"""
```

1.2 Exercice 2

```
#renommons les listes de l'exo 1 en remplaçant t par l
t0,t1,t2,t3,t4 = l0,l1,l2,l3,l4
```

1.3 Exercice 3

```
# Dix derniers éléments de t3
t5 = t3[-10:]
# Éléments de t3 sauf les 250 premiers et les 250 derniers
t6 = t3[250:-250]
#cinq premiers éléments de t4 suivie des cinq derniers de t4
t7 = t4[:5]+t4[-5:]
```

1.4 Exercice 4

Recopier les commandes ci-dessous dans l'éditeur de la page web http://pythontutor.com/visualize.html

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

k,t,m sont des alias de la même liste et n pointe vers 1 autre liste

1.5 Exercice 5

```
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]

"""
>>> t2,t5
([9, 4, 1, 0, 1, 4, 9, 16, 25], [4, 0, 4])
t5 contient les trois premiers éléments de t2 d'indice impair
>>> t6
[81, 16, 1, 0, 1, 16, 81, 256, 625]
t6 contient les carrés des éléments de t2
>>> t7
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
t7 contient les couples (i,j) avec i in [1,2,3] et j in [3,1,4] et i!=j
"""
```

Compléments sur les listes en compréhension :

- Liste des classes scientifiques par année, puis par filière puis par numéro de classe :

```
"""
In [4]: [x + y + z for x in [800, 900] for y in [30, 40, 50] for z in range(1, 4)]
Out[4]:
[831,  832,  833, 841, 842, 843, ...   951, 952, 953]
"""
```

- Liste des classes scientifiques par filière, puis par année puis par numéro de classe :

```
"""
In [5]: [x + y + z for y in [30, 40, 50] for x in [800, 900]  for z in range(1, 4)]
Out[5]:
[831,  832,  833, 931, 932, 933, ...   951, 952, 953]
"""
```

1.6 Exercice 6

```
#listes des termes d'indice pair de t1 (len(t1)//2 tours de boucle for)
t8 = [t1[i] for i in range(0,len(t1),2)]
#idem, listes des termes d'indice pair de t1 (len(t1)//2 tours de boucle for)
#ATTENTION ne pas passer un flottant comme len(t1)/2 comme argument de range
t81 = [t1[2 * i] for i in range(len(t1)//2)]
#liste des termes pairs de t1 ( len(t1) tours de boucle for)
t9 = [terme for terme in t1 if terme%2 == 0]
#>>> t1,t8
#[10, 30, 42, 2, 17, 5, 30, -20],[10, 42, 17, 30]
#>>> t9
#[10, 30, 42, 2, 30, -20]
```

1.7 Exercice 7

```
def echange(tab, i, j):
    """echange les valeurs de tab[i] et tab[j]"""
    #Facultatif : on vérifie que i et j sont des entiers compris entre 0 et len(t)
    #sinon la fonction génère une erreur
    #assert isinstance(i,int) and isinstance(j,int) and 0<=i<len(tab) and 0<=j<len(tab)
    tab[i],tab[j] = tab[j],tab[i]
    #inutile de retourner t puisque les listes sont passées par référence

"""
>>> t=[i**3 for i in range(5)]
>>> echange(t,1,2)
>>> t
[0, 8, 1, 27, 64]
"""
```

2 Parcours de tableaux

2.1 Exercice 8

```
def somme(tab):
    """somme des éléments d'un tableau, redéfinition de sum"""
```

```
    s = 0
    for terme in tab:
        s = s + terme
        #s += terme   est un raccourci
    return s

def produit(tab):
    """produit des éléments d'un tableau"""
    p = 1
    for terme in tab:
        p *= terme
        #p *= terme est un raccourci
    return p
```

2.2 Exercice 9

```
def moyenne(tab):
    """moyenne des éléments d'un tableau"""
    return somme(tab)/len(tab)

def ecart_type(tab):
    """écart-type des éléments d'un tableau"""
    return math.sqrt(moyenne([e**2 for e in tab])-moyenne(tab)**2)

"""
>>> moyenne(t2)
7.666666666666667
>>> ecart_type(t2)
7.803133273813083
"""
```

2.3 Exercice 10

```
def maximum(tab):
    """retourne le maximum d'un tableau, redéfinition de max"""
    if tab == []:  #si la liste est vide
        return None
    m = tab[0]
    for terme in tab[1:]:
        if terme > m:
            m = terme
    return m

def maximum2(tab):
    """retourne le maximum d'un tableau, redéfinition de max"""
    if tab == []:  #si la liste est vide
        return None
    m = tab[0]
    for k in range(1, len(tab)):
        if tab[k] > m:
            m = tab[k]
    return m
```

```
def maximum3(tab):
    """retourne le maximum d'un tableau, marche pour les listes vides"""
    m = -float('inf')  #on initialise m à moins l'infini
    for terme in tab:
        if terme > m:
            m = terme
    return m
"""
>>> maximum(t1)
42
"""
def position_maximum(tab):
    """retourne la première position où le maximum est atteint"""
    if tab == []:  #si la liste est vide
        return None
    imaxi, maxi = 0, tab[0]
    for indice in range(1, len(tab)):
        terme = tab[indice]
        if terme > maxi:
            imaxi, maxi = indice, terme
    return imaxi

def position_maximum2(tab):
    """retourne la première position où le maximum est atteint"""
    imaxi, maxi = 0, tab[0]
    #l'itérateur enumrate permet de récupérer le couple indice,valeur
    for indice,terme in enumerate(tab):
        if terme > maxi:
            imaxi, maxi = indice, terme
    return imaxi

"""
>>> position_maximum(t1)
2
"""
```

2.4 Exercice 11

```
def liste_position_maximum(tab):
    """retourne le maximum et la liste des positions où il est atteint"""
    tmaxi, maxi = [0],tab[0]
    for indice in range(1, len(tab)):
        terme = tab[indice]
        if terme > maxi:
            tmaxi, maxi = [indice], terme
        elif terme == maxi:
            tmaxi.append(indice)
    return tmaxi, maxi

"""
>>> from random import randint
>>> t=[randint(1,5) for i in range(10)]
```

```
>>> t
[4, 4, 1, 5, 5, 4, 2, 2, 1, 2]
>>> liste_position_maximum(t)
([3, 4], 5)
"""
```

2.5 Exercice 12

```
def est_croissante(tab):
    """retourne True si tab est croissante et False sinon"""
    for i in range(1, len(tab)):
        if tab[i] < tab[i-1]:
            return False
    #si on arrive là le tableau était dans l'ordre croissant
    return True

def est_strict_decroissante(tab):
    """retourne True si tab est strictement décroissante et False sinon"""
    for i in range(1, len(tab)):
        if tab[i] >= tab[i-1]:
            return False
    #si on arrive là le tableau était dans l'ordre strictement décroissant
    return True

"""
>>> est_croissante([1,2,2,3,4])
True
>>> est_croissante([1,2,2,3,1])
False
>>> est_strict_decroissante([4,3,2,1])
True
>>> est_strict_decroissante([4,3,2,1,1])
False
"""
```

2.6 Exercice 13

```
def sommes_cumulees(tab):
    """retourne les sommes cumulées de tab depuis le premier terme"""
    cumul = [tab[0]]
    naddition = 0
    for terme in tab[1:]:
        cumul.append(cumul[-1]+terme)
        naddition += 1
    return cumul,'%s additions'%naddition

"""
>>> sommes_cumulees(t2)
([9, 13, 14, 14, 15, 19, 28, 44, 69], '8 additions')
"""
```

2.7 Exercice 14

```
def deux_gros(tab):
    """retourne les deux + gros éléments d'un tableau de longueur>=2"""
    assert len(tab) >= 2
    if tab[0] > tab[1]:
        gros = tab[:2]
    elif tab[0] < tab[1]:
        gros = tab[1::-1]
    else:
        gros = [tab[0]]
    for terme in tab[2:]:
        if len(gros) == 1:
            if terme > gros[0]:
                gros.insert(terme,0)
            else:
                gros.append(terme)
        else:
            if terme > gros[0]:
                gros[0],gros[1] = terme, gros[0]
            elif gros[0] > terme > gros[1]:
                gros[1] = terme
    return gros
```

2.8 Exercice 15

```
def deux_grosbis(tab):
    """idem mais répétition du plus gros s'il apparait 2 fois"""
    assert len(tab) >= 2
    if tab[0] > tab[1]:
        gros = tab[:2]
    elif tab[0] < tab[1]:
        gros = tab[1::-1]
    else:
        gros = [tab[0]]*2
    for terme in tab[2:]:
        if terme > gros[0]:
            gros[0], gros[1] = terme, gros[0]
        elif terme > gros[1]:
            gros[1] = terme
    return gros

"""
>>> deux_gros(t1)
[42, 30]
>>> deux_gros(t1+[42])
[42, 30]
>>> deux_grosbis(t1+[42])
[42, 42]
"""
```

2.9 Exercice 16

```
def difference(t):
    """Modifie le tableau t en substituant a chaque element sauf le premier
    sa difference avec l'element precedent"""
    n = len(t)
    for k in range(n - 1, 0, -1):
        t[k]= t[k] - t[k - 1]
    return t
```

Il faut modifier le tableau de droite à gauche afin de préserver les valeurs qu'on utilise dans les différences (celles des cellules précédentes du tableau).

```
def difference_reciproque(t):
    """Modifie t, fonction réciproque de difference"""
    n = len(t)
    for k in range(1, n):
        t[k]= t[k] + t[k - 1]
    return t

"""
In [3]: t = [4,5,3,7] ; difference(t)
Out[3]: [4, 1, -2, 4]

In [4]: difference_reciproque(t)
Out[4]: [4, 5, 3, 7]
"""
```

2.10 Exercice 17

```
def distincts(t):
    """Retourne un booleen, teste si les éléments d'un tableau sont distincts"""
    #on compare chaque élément avec ses successeurs
    for i in range(len(t)):
        for j in range(i+1, len(t)):
            if t[j] == t[i]:
                return False
    return True

"""
In [2]: t1, distincts(t1)
Out[2]: ([10, 30, 42, 2, 17, 5, 30, -20], False)

In [3]: t1[:-2], distincts(t1[:-2])
Out[3]: ([10, 30, 42, 2, 17, 5], True)
"""

def distincts2(t):
    """Retourne un booleen, teste si les éléments d'un tableau sont distincts"""
    i = 0
    while i < len(t):
```

```
        j = i + 1
        while j < len(t):
            if t[j] == t[i]:
                return False
            j += 1
        i += 1
    return True
```

Dans le pire des cas (tous les éléments distincts), n - 1 + n - 2 + ... + 1 comparaisons sont effectuées (soit n(n-1)/2), complexité quadratique

```
def est_permutation(t):
    """teste si un tableau est une permutation"""

    def est_permutation_aux(t):
        """Fonction auxiliaire, celle qui travaille"""
        for i in range(len(t)):
            if memo[t[i] - 1]:  #si valeur deja vu ce n'est pas une permutation
                return False
            memo[t[i] - 1] = True
        return True  #si on arrive ici c'est une permutation

    memo = [False]*len(t) #initialisation du tableau de memoization
    return est_permutation_aux(t)

"""
In [20]: t = [i for i in range(1, 11)] ; random.shuffle(t) ; t
Out[20]: [4, 2, 8, 10, 3, 9, 7, 6, 1, 5]

In [21]: est_permutation(t)
Out[21]: True

In [22]: tn = t + [2] ; est_permutation(tn)
Out[22]: False
"""
```

2.11 Exercice 18

```
def grand_et_petit(tab):
    n = len(tab)
    assert n%2 == 0, 'Le tableau doit etre de taille paire'
    tcopie = sorted(tab)
    tres = []
    n = len(tcopie)//2
    for k in range(n):
        tres.extend([tcopie[-1-k], tcopie[k]])
    return tres

def grand_et_petit2(tab):
    t = sorted(tab)
    return [k%2 == 0 and t[-1-k//2] or t[(k-1)//2] for k in range(len(t))]
```

```
"""
In [18]: tab = random.sample(range(1000), 10)

In [19]: tab
Out[19]: [708, 566, 84, 624, 691, 390, 560, 576, 485, 117]

In [20]: grand_et_petit(tab)
Out[20]: [708, 84, 691, 117, 624, 390, 576, 485, 566, 560]

In [21]: grand_et_petit2(tab)
Out[21]: [708, 84, 691, 117, 624, 390, 576, 485, 566, 560]
"""
```

2.12 Exercice 19

```
def lissage(t):
    newt = [t[0]]
    for k in range(1, len(t)-1):
        newt.append((t[k-1] + t[k+1])/2)
    newt.append(t[-1])
    return newt

def ecartmax(t):
    f = lambda tab,k : abs(tab[k] - tab[k+1])
    return max([f(t, k) for k in range(0, len(t)-1)])

def nblissages(tmes, diffmax, itermax):
    mesures = tmes
    nliss = 0
    while nliss <= itermax and ecartmax(mesures) >= diffmax:
        mesures = lissage(mesures)
        nliss += 1
    if nliss <= itermax:
        return nliss
    return None
```

2.13 Exercice 20

```
def maximum_intervalle(t, n):
    taille_tableau = len(t)
    assert n <= taille_tableau, 'len(sous-tableau) <= len(tableau) !'
    s = 0
    for k in range(n):
        s += t[k]
    debut = 0
    fin = n - 1
    smax = s
    for k in range(n, taille_tableau - n + 1):
         s -= t[debut]
         debut += 1
```

```
            fin += 1
            s += t[fin]
            if s > smax:
                smax = s
    return smax

"""
In [90]: maximum_intervalle([10,2,8,4,7,5], 3)
Out[90]: 20
"""
```

3 Extrait du sujet posé au concours Centrale 2015

3.1 Exercice 21 Sujet Centrale 2015

```
"""
In [6]: [1, 2, 3] + [4, 5, 6]
Out[6]: [1, 2, 3, 4, 5, 6]

In [7]: 2 * [1, 2, 3]
Out[7]: [1, 2, 3, 1, 2, 3]
"""
```

3.2 Exercice 22 Sujet Centrale 2015

```
def smul(k ,t):
    """Retourne une nouvelle liste où chaque élément de t est multiplié par le
    réel k"""
    n = len(t)
    v = [0]*n
    for i in range(n):
        v[i] = t[i] * k
    return v

def vsom(u ,v):
    """Retourne une nouvelle liste où chaque élément est la somme des éléments
    des listes u et v de meme index"""
    n = len(u)
    assert len(v) == n, "Les deux listes doivent etre de meme longueur"
    w = [0]*n
    for i in range(n):
        w[i] = u[i] + v[i]
    return w

def vdif(u ,v):
    """Retourne une nouvelle liste où chaque élément est la différence entre
     l'élement de v et l'élément de u de meme index"""
    n = len(u)
    assert len(v) == n, "Les deux listes doivent etre de meme longueur"
    return vsom(v, smul(-1, u))

def vprod(u ,v):
```

```
    """Retourne une nouvelle liste où chaque élément est le produit des éléments
    de v et v de meme index"""
    n = len(u)
    assert len(v) == n, "Les deux listes doivent etre de meme longueur"
    w = [0]*n
    for i in range(n):
        w[i] = u[i] * v[i]
    return w

def prodscal(u ,v):
    """Retourne le produit scalaire des vecteurs de coordonnées u et v"""
    n = len(u)
    assert len(v) == n, "Les deux listes doivent etre de meme longueur"
    w = vprod(u, v)
    s = 0
    for i in range(n):
        s += w[i]
    return s

"""
In [10]: smul(2, [1, 2, 3])
Out[10]: [2, 4, 6]

In [11]: vsom([1, 2, 3], [4, 5, 6])
Out[11]: [5, 7, 9]

In [12]: vdif([1, 2, 3], [4, 5, 6])
Out[12]: [3, 3, 3]

In [13]: vprod([1, 2, 3], [4, 5, 6])
Out[13]: [4, 10, 18]

In [14]: prodscal([1, 2, 3], [4, 5, 6])
Out[14]: 32
"""
```

3.3 Exercice 23 Sujet Centrale 2015

Force exercée sur le corps P_j par l'ensemble des autres corps P_k avec $k \neq j$

$$\overrightarrow{F_j} = \sum_{k \neq j} \overrightarrow{F_{k/j}}$$
$$\overrightarrow{F_j} = \sum_{k \neq j} G \frac{m_j m_k}{r_{jk}^3} \overrightarrow{P_j P_k}$$

```
G = 6.67*10**(-11)  #Constante gravitationnelle

def force2(m1, p1, m2, p2):
    """Retourne la liste de coordonnées de la force gravitationnelle exercée
    par le corps de masse m2 et de position p2 sur le corps de masse m1 et de position p1"""
    vecteur = vdif(p1, p2) #coordonnées du vecteur vect(p1, p2)
    norme = (prodscal(vecteur, vecteur))**0.5
```

```
    return smul(G*m1*m2/norme**3, vecteur)


def forceN(j, m, pos):
    """Prend en parametre l'indice j du corps sur lequel s'exerce
    la force, la liste m des masses des N corps, la liste pos des positions
    et retourne le vecteur force Fj exercé par tous les autres corps
    sur le corps j"""
    f = [0]*3 #vecteur force resultante
    n = len(m)
    assert n == len(pos), "Les listes m et pos doivent etre de meme taille"
    mj = m[j]
    pj = p[j]
    for k in range(n):
        if k != j:
            f = vsom(f, force2(mj, pj, m[k], p[k]))
    return f
```

4 Pour ceux qui s'ennuient

4.1 Exercice 24

```
def fibonacci(n):
    """retourne un tableau de (n+1) éléments contenant les termes f0,..,fn"""
    tab = [0,1]
    for i in range(2,n+1):
        tab.append(tab[i-1]+tab[i-2])
    return tab
"""
>>> fibonacci(10)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
"""
```

4.2 Exercice 25

```
def ligne_binome(n):
    """retourne la liste des C(n,k) avec 0<=k<=n"""
    assert isinstance(n,int) and n>=0,"n doit être un entier positif"
    if n==0:
        return  [1]
    else:
        ligneavant = [1,1]
        for i in range(2,n+1):
            ligne = [1]
            for k in range(1,i):
                ligne.append(ligneavant[k-1]+ligneavant[k])
            ligne.append(1)
            ligneavant = ligne
        return ligneavant
```

```
"""
>>> for i in range(6):
...     print(ligne_binome(i))
...
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
"""
```

4.3 Exercice 26

```
def plus_grand_bloc_croissant(t):
    """retourne un triplet (longueur,indice du t,plus grande sous-suite
    croissante constituée de termes  consécutifs du tableau)"""
    lmax = 0#longueur du plus grand bloc croissant
    bmax = None  #indices de début et de fin (exclu) du + gd  bloc croissant
    k = 0 #indice de départ d'exploration d'un nouveau bloc
    while k < len(t) - lmax:
        j = k
        #recherche de l'indice du + grand bloc croissant à partir de k
        while j < len(t)-1 and t[j+1] >= t[j]:
            j += 1
        lprov = j - k + 1
        if lprov > lmax:  #mise à jour des paramètres du maximum
            lmax = lprov
            bmax = (k, j + 1)
        k = j + 1
    return lmax, t[bmax[0]: bmax[1]]

"""
In [12]: plus_grand_bloc_croissant([2,10,1,3,5,7,6,8,9])
Out[12]: (4, [1, 3, 5, 7])
In [13]: plus_grand_bloc_croissant([2,10,1,3,5,6,7,8,9])
Out[13]: (7, [1, 3, 5, 6, 7, 8, 9])
In [24]: lmax, _ = plus_grand_bloc_croissant([i for i in range(10**6)])
In [25]: lmax
Out[25]: 1000000
"""
```

Algorithme de complexité linéaire (d'ordre n) dans le pire des cas qui serait un tableau croissant comme [0,11,2,3,4,5,...,999999]

```
def test_exo26():
    from random import randint
    resultats = []
    for _ in range(100):
        t = [randint(0,10**6) for _ in range(10**5)]
        resultats.append(plus_grand_bloc_croissant(t)[0])
    return moyenne(resultats),maximum(resultats)
```