Fiche 2

Gestions des tableaux

Exercice 1 :

1. Écrire la fonction `somme`, qui prend en argument un tableau d'entiers, et renvoie la somme des valeurs du tableau.
2. Écrire la fonction `moyenne` qui renvoie la moyenne des valeurs d'un tableau de nombres.
3. Écrire la fonction `min_max` qui renvoie le plus grand et le plus petit élément d'un tableau sous forme d'un couple.
4. Écrire la fonction `somme_positifs`, qui prend en argument un tableau d'entiers relatifs, et renvoie la somme des valeurs positives du tableau.
5. Écrire la fonction `somme_paire`, qui prend en argument un tableau d'entiers , et renvoie la somme des valeurs paires du tableau.

Correction

```
# Exercice 1

# Question 1

def somme(ma_liste) :
    s = 0
    for i in ma_liste :
        s = s + i
    return s


#Question 2
def moyenne(ma_liste) :
    """ Retourne la moyenne de la liste """
    s = somme(ma_liste)
    taille = len(ma_liste)
    return s/taille

def moyenne_2(ma_liste) :
    return (somme(ma_liste) / len(ma_liste))

#Question 3
def min_max(ma_liste) :
    mini =ma_liste[0]
    maxi =ma_liste[0]
    for i in ma_liste :
        if i < mini :
            mini = i
        if i > maxi :
            maxi = i
    return (mini, maxi )
```

```
#Question 4 :
def somme_positifs(tab) :
    s = 0
    for i in tab :
        if i >=0 :
            s = s + i
    return s

#Question 5 :
def somme_paires(tab) :
    s = 0
    for i in tab :
        if i%2 == 0 :
            s = s + i
    return s
```

Exercice 2 :

1. Écrire une fonction `triple` qui prend en argument un tableau de 3 entiers trié dans l'ordre croissant et un entiers, et renvoie le tableau des 3 valeurs les plus grandes stockées dans le tableau.
   Exemple :

```
>>> triple([8,10,12],7)
[8, 10, 12]
>>> triple([8,10,12],9)
[9, 10, 12]
>>> triple([8,10,12],11)
[10, 11, 12]
>>> triple([8,10,12],15)
[10, 12 , 15]
```

2. Écrire la fonction `triple_max` qui renvoie le tableau des trois plus grande valeurs d'un tableau de nombres. (rangés dans l'ordre croissant)

Correction

```
#Question 1

def triple(tab,n) :
    """ tab est trie, n est un entier, renvoie les 3 elements maximum """
    if n < tab[0] :
        return tab

    elif n < tab[1] :
        tab[0] = n
        return tab

    elif n < tab[2] :
        tab[0]=tab[1]
        tab[1] = n
```

```
        return tab

    else :
        tab[0]=tab[1]  # tab[1:] + [n]
        tab[1] = tab[2]
        tab[2] = n
        return tab


def triage_3(ma_liste) :
    if ma_liste[0] > ma_liste[1] :
        if ma_liste[0] > ma_liste[2] :
            if ma_liste[2] > ma_liste[1] :
                return [ ma_liste[1] , ma_liste[2] , ma_liste[0] ]
            else :
                return [ ma_liste[2] , ma_liste[1] , ma_liste[0] ]
        else :
            return [ ma_liste[1] , ma_liste[0] , ma_liste[2] ]
    else :
        if ma_liste[1] > ma_liste[2] :
            if ma_liste[2] > ma_liste[0] :
                return [ ma_liste[0] , ma_liste[2] , ma_liste[1]]
            else :
                return [ ma_liste[2] , ma_liste[0] , ma_liste[1]]
        else :
            return ma_liste



def triple_max(ma_liste ) :
    tab = triage_3(ma_liste[:3])
    for i in ma_liste[3:] :
        tab = triple(tab,i)
    return tab
```

Exercice 3 : On considère le tableau composé uniquement des valeurs 0, 1, 2 et 3.
Écrire la `compte`, qui prend en argument un tel tableau, et renvoie un autre tableau de taille 4, contenant en position i le nombre de i du tableau argument.

Correction

```
#Exercice 3
def compte(ma_liste) :
    resultat = [ 0 , 0 ,0 ,0 ] # [ 0 for i in range(4)]
    for i in ma_liste :
        resultat[i] = resultat[i]  +1
    return resultat
```

Exercice 4 : On considère le tableau suivant :

```
t =[[ i for i in range(7)] for j in range(10)]
```

1. Déterminer les valeurs suivantes (on vérifia ensuite sur l'ordinateur) :

```
>>> len(t)
>>> len(t[0])
>>> t[2][6]
>>> t[1][8]
>>> t[4][2] = 9
>>> t[2][5] == 5
```

2. Écrire la procédure que permet de changer les valeurs de tableau `t` en suivant les règles suivantes :
   - $t[i][j]$ devient 0 si $i = j$.
   - $t[i][j]$ devient -1 si $i < j$.
   - $t[i][j]$ devient 1 si $i > j$.

Correction

```
t = [[k for k in range(7)] for j in range(10)]


for ligne in range(10) :
    for colonne in range(7) :
        if ligne == colonne :
            t[ligne][colonne] =0
        if ligne < colonne :
            t[ligne][colonne] =-1
        if ligne > colonne :
            t[ligne][colonne] =1



for i in t :
    print(i)
```

Exercice 5 : On considère la matrice définie par (en utilisant la bibliothèque `random` :

```
carte = [[ randint(0,1) for i in range(100)] for i in range(100)]
```

1. Écrire la fonction `test`, qui prend en argument deux entiers i et j et un tableau t, et renvoie le booléen indiquant si la case $t[i][j]$ ainsi que les huit cases qui l'entoure sont tous des 1. (on veillera à ne pas travailler sur le "bord" de la matrice...)
2. Écrire une procédure qui compte le nombre de case vérifiant la propriété précédente.

Correction

```
# exercice 5
from random import randint

carte= [[ randint(0,1) for i in range(100)] for i in range(100)]
```

```
def test(t,i,j) :
    return t[i][j]+t[i-1][j-1]+ t[i-1][j]+t[i-1][j+1] +\
           t[i][j-1] + t[i][j+1]  +\
           t[i+1][j-1] + t[i+1][j] + t[i+1][j+1] ==9


def compte(t) :
    nb_ligne = len(t)
    nb_colonne = len(t[0])
    compteur = 0
    for ligne in range(1,nb_ligne -1) :
        for colonne in range(1 , nb_colonne -1 ) :
            if test(t,ligne,colonne) :
                compteur += 1
    return compteur

print(compte(carte))
```

Exercice 6 : On souhaite construire le jeu du démineur :

1. Écrire la fonction `creation_carte` qui prend en argument un entier n et renvoie une matrice de taille n^2 composée de (-2).
2. Écrire la fonction `bord` qui prend en argument une matrice et renvoie la même matrice en remplaçant les "bords" de la matrice par (-3).
3. Écrire la fonction `placement_bombe` qui prend en argument une matrice et un entiers nb et renvoie la même matrice en plaçant aléatoirement les nb bombes sur la carte (mais pas sur les bords...). Une bombe sera représentée par le nombre -1.
4. Écrire la fonction `jeu_i_j` qui prend en argument une matrice t et deux entiers i et j. Si $t_{i,j}$ représente une bombe, la fonction renvoie -1, sinon elle retourne le nombre de bombes au voisinage directe de $t_{i,j}$.
5. Écrire la fonction `affiche_jeu` qui prend en argument une matrice t et affiche + si $t_{i,j} < 0$ et la valeur de $t_{i,j}$ sinon.
6. En déduire la fonction `jeu` qui prend en argument deux entiers, n (la taille de la carte) et p (le nombre de bombes), et permet de jouer en assurant un affichage de la carte à chaque tour de jeu.

Correction

```
# Exercice 6
from random import randint

def creation_carte(n) :
    carte = [[ -2 for j in range(n) ] for i in range(n) ]
    return carte

def bord(mat) :
    nb_ligne = len(mat)
    nb_colonne = len(mat[0])
    for i in range(nb_ligne) :
```

```
        mat[i][0] = -3  # action sur la premiere colonne
        mat[i][nb_colonne-1]= -3 # action sur la derniere colonne
    for j in range(nb_colonne) :
        mat[0][j] = -3 # action sur la premiere ligne
        mat[nb_ligne-1][j] = -3 # action sur la derniere ligne
    return mat

def placement_bombe(mat,nb) :
    nb_ligne = len(mat)
    nb_colonne = len(mat[0])
    while not(nb == 0 ) :
        ligne_au_hasard = randint(1, nb_ligne -2)
        colonne_au_hasard = randint(1 , nb_colonne -2)
        if not(mat[ligne_au_hasard][colonne_au_hasard] == -1) :
            mat[ligne_au_hasard][colonne_au_hasard] = -1
            nb -= 1
    return mat

def jeu_i_j(mat,i,j):
    if mat[i][j] == -1 :
        return -1
    else :
        nb= 0
        liste_possible = [(i-1,j-1),(i-1,j),(i-1,j+1),\
                          (i,j-1) , (i,j+1) ,\
                          (i+1,j-1) , (i+1,j) ,(i+1,j+1)]
        for case in liste_possible :
            if mat[case[0]][case[1]]== -1 : #il y a une bombe
                nb +=1
        return nb

def affiche_jeu(mat) :
    nb_ligne = len(mat)
    nb_colonne = len(mat[0])
    print("*" , end = "␣")
    for i in range(nb_colonne) :
        print(i , end = "␣")
    print("")
    for i in range(nb_ligne) :
        print(i , end="␣")
        for j in range(nb_colonne ) :

            if mat[i][j]<0 :
                print("+" , end = "␣" )
            else :
                print(mat[i][j] , end = "␣")
        print("␣")
```

```
def jeu (n , p ) :
    """ n taille matrice, p nb de bombes """
    mat = creation_carte(n)
    mat_2 = bord(mat)
    mat_3 = placement_bombe(mat_2,p )
    explose = False
    affiche_jeu(mat_3)
    while not(explose) :
        ligne = int(input("entrer une ligne : "))
        colonne = int(input("entrer une colonne : "))
        resultat = jeu_i_j(mat_3 , ligne ,colonne)
        if resultat == -1 :
            explose = True
        else :
            mat_3[ligne][colonne] = resultat
        affiche_jeu(mat_3)
    print("explosion")
```