C:\Users\LAURENT_ASUS\Notebooks JUPYTER\NSI\Q3\Laurent_LHERMET_matrices_exercices

October 28, 2020

0.0.1 Exercice 1

Ecrire une fonction `somme` qui prend en argument une matrice non vide de nombres entiers et renvoie la somme des nombres de cette matrice. Voici un exemple d'assertion qui doit être vérifiée par votre fonction :

```
assert(somme([[3, 8, 9, 11],
              [6, 8, 1, -4],
              [1, 1, 1, 0 ]]) == 45)
```

[1]:
```
%load_ext tutormagic
#integration de tutor magic dans le notebook
```

[9]:
```
%%tutor --lang python3
# votre code ici

def somme(M):
    total = 0
    for i in range (len(M)):                #pour l'ensemble des lignes de la␣
 ↪matrice
        for j in range (len(M[0])):         #pour l'ensemble des colonnes de la␣
 ↪matrice
            total += M[i][j]
    return total



liste_nb  = [[3,8,9,11],
           [6,8,1,-4],
           [1,1,1,0]]
print(somme(liste_nb))
```

```
<IPython.lib.display.IFrame at 0x2aa52032460>
```

0.0.2 Exercice 2

Ecrire une fonction `max_somme` qui prend en argument une matrice non vide et renvoie la plus grande somme des différentes colonnes. Voici deux exemples d'assertions qui doivent être vérifiées par votre fonction :

```
assert(max_somme([[3, 8, 9, 11],
              [6, 8, 1, -4],
              [1, 1, 1, 0 ]]) == 17)    #colonne d'indice 1 : 8 + 8 + 1 = 17

assert(max_somme([[-3, -8, -9, -11],
                  [-6, -8, -1, 4  ],
                  [-1, -1, -1, 0  ]]) == -7)    #colonne d'indice 3 : -11 + 4 + 0 = -7
```

[45]:
```
# %%tutor --lang python3 --tab


# Le total de la colonne peut être négatif ! On calcule le total de la première␣
 ↪colonne qui sert de valeur de réference
# et on met à jour la variable si une valeur supérieure est trouvée dans les␣
 ↪colonnes suivantes.

def max_somme(M):
    total_colonne = 0
    total_max = 0
    for lig in range (len(M)):   #total de la première colonne
        total_colonne += M[lig][0]
        total_max = total_colonne  #le total de la première colonne est la␣
 ↪valeur de référence
    #reinitialisation du total de la colonne pour passer aux suivantes
    total_colonne = 0
    #pour les colonnes suivantes de la matrice
    for col in range (1,len(M[0])):
        for lig in range (len(M)):     #pour l'ensemble des lignes de la matrice
            total_colonne += M[lig][col]
        if (total_colonne) > total_max:    #test si valeur colonne > valeur max␣
 ↪mémorisée
            total_max = total_colonne    # mémorisation de la nouvelle valeur␣
 ↪max
        total_colonne = 0           #reinitialisation de total_colonne
    return total_max

liste_nb = [[3, 8, 9, 11],
            [6, 8, 1, -4],
            [1, 1, 1, 0 ]]

liste_nb2 = [[-3, -8, -9, -11],
             [-6, -8, -1, 4  ],
```

```
            [-1, -1, -1, 0  ]]

print(max_somme(liste_nb))
print(max_somme(liste_nb2))
```

```
17
-7
```

[46]:
```
assert(max_somme([[3, 8, 9, 11],
                  [6, 8, 1, -4],
                  [1, 1, 1, 0 ]]) == 17)    #colonne d'indice 1 : 8 + 8 + 1 = 17

assert(max_somme([[-3, -8, -9, -11],
                  [-6, -8, -1, 4  ],
                  [-1, -1, -1, 0  ]]) == -7)    #colonne d'indice 3 : -11 + 4 +␣
↪0 = -7
```

0.0.3 Exercice 3

Question 1 :

Ecrire une fonction `compresse` qui prend en argument une matrice non vide dont les éléments sont des entiers et modifie les éléments de la façon suivante : - si l'élément est positif, on diminue l'élément de une unité, - si l'élément est négatif, on augmente l'élément de une unité, - si l'élément est nul, on ne le modifie pas.

La fonction devra **muter** la matrice en place et ne **renverra** donc aucune valeur.

[53]:
```
# %%tutor --lang python3 --tab

def compresse(M):
    for lig in range(len(M)):                       # Pour l'ensemble des lignes
        for col in range(len(M[0])):                # Pour l'ensemble des colonnes
            if M[lig][col]>0:
                M[lig][col]-= 1
            elif M[lig][col]<0:
                M[lig][col] += 1

ma_matrice = [[ 0,  1,  2,  3,  4, 5],
              [-1,  0,  1,  2,  3, 4],
              [-2, -1,  0,  1,  2, 3],
              [-3, -2, -1,  0,  1, 2],
              [-4, -3, -2, -1,  0, 1],
              [-5, -4, -3, -2, -1, 0]]

compresse(ma_matrice)
ma_matrice
```

[53]:
```
[[0, 0, 1, 2, 3, 4],
 [0, 0, 0, 1, 2, 3],
 [-1, 0, 0, 0, 1, 2],
 [-2, -1, 0, 0, 0, 1],
 [-3, -2, -1, 0, 0, 0],
 [-4, -3, -2, -1, 0, 0]]
```

[54]:
```
ma_matrice = [[ 0,  1,  2,  3,  4, 5],
              [-1,  0,  1,  2,  3, 4],
              [-2, -1,  0,  1,  2, 3],
              [-3, -2, -1,  0,  1, 2],
              [-4, -3, -2, -1,  0, 1],
              [-5, -4, -3, -2, -1, 0]]
```

[65]:
```
compresse(ma_matrice)
ma_matrice
```

[65]:
```
[[0, 1, 2, 3, 4, 5],
 [-1, 0, 1, 2, 3, 4],
 [-2, -1, 0, 1, 2, 3],
 [-3, -2, -1, 0, 1, 2],
 [-4, -3, -2, -1, 0, 1],
 [-5, -4, -3, -2, -1, 0]]
```

Question 2 :

Modifier la fonction précédente pour **qu'elle ne mute plus la matrice passée en argument qui devra rester intacte** mais qu'elle renvoie **une copie indépendante et compressée** de la matrice passée en argument.

[62]:
```
# %%tutor --lang python3 --tab
from copy import deepcopy

def compresse(M):
    N = deepcopy(M)
    for lig in range(len(N)):                    # Pour l'ensemble des lignes
        for col in range(len(N[0])):             # Pour l'ensemble des colonnes
            if N[lig][col]>0:
                N[lig][col]-= 1
            elif N[lig][col]<0:
                N[lig][col] += 1

ma_matrice = [[ 0,  1,  2,  3,  4, 5],
              [-1,  0,  1,  2,  3, 4],
              [-2, -1,  0,  1,  2, 3],
              [-3, -2, -1,  0,  1, 2],
              [-4, -3, -2, -1,  0, 1],
              [-5, -4, -3, -2, -1, 0]]
```

```
ma_matrice_compressee = compresse(ma_matrice)
ma_matrice_compressee
```

[66]:
```
ma_matrice = [[ 0,  1,  2,  3,  4, 5],
              [-1,  0,  1,  2,  3, 4],
              [-2, -1,  0,  1,  2, 3],
              [-3, -2, -1,  0,  1, 2],
              [-4, -3, -2, -1,  0, 1],
              [-5, -4, -3, -2, -1, 0]]
```

[67]:
```
ma_matrice_compressee = compresse(ma_matrice)
ma_matrice_compressee
```

0.0.4 Exercice 4

Programmer une fonction `sablier` qui prend en argument un entier `n` strictement positif et renvoie une matrice de `n` lignes et `n` colonnes dont : - le quart supérieur contient des 8, - le quart inférieur contient des 1, - le reste de la matrice (dont les diagonales) contient des 0.

Voici deux exemples d'assertions vérifiée par la fonction quatre couleurs:

```
assert(sablier(15) ==   [[0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0],
                         [0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0],
                         [0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0],
                         [0, 0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 8, 8, 8, 8, 8, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 8, 8, 8, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
                         [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
                         [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
                         [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]]

assert(sablier(8) ==    [[0, 8, 8, 8, 8, 8, 8, 0],
                         [0, 0, 8, 8, 8, 8, 0, 0],
                         [0, 0, 0, 8, 8, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 0, 0, 0, 0, 0],
                         [0, 0, 0, 1, 1, 0, 0, 0],
                         [0, 0, 1, 1, 1, 1, 0, 0],
                         [0, 1, 1, 1, 1, 1, 1, 0]]
```

[91]:
```
# %%tutor --lang python3 --tab

# pour n lignes et colonnes on va aller de 0 à n-1 donc dans range (0,n) !!


def sablier(n):
    ma_matrice = [[0 for _ in range(n)] for _ in range(n)]
    #placement des 8
    for lig in range (0,(n//2)):              #de la ligne 0 à lal ligne (n//
↪2)-1
        for col in range (lig+1,(n-1)-lig):  #de la colonne correspondant au␣
↪numéro de la lig+1 à n-numero de la ligne
            ma_matrice[lig][col] = 8
   #placement des 1
    for lig in range ((n//2)+1,n):            #de la moitié jusqu'en bas. Rappel␣
↪avec range s'arrête à n-1
        for col in range (n-lig,lig):         #idem avec range s'arrete à n-1
            ma_matrice[lig][col] = 1
    return ma_matrice

sablier(15)
```

[91]:
```
[[0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0],
 [0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0],
 [0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0],
 [0, 0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 8, 8, 8, 8, 8, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 8, 8, 8, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
 [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
 [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]]
```

[ ]:
```

```