TD 1 Python

▼ Subject

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/59062b1f-2a2c-4672-b1f9-bd8e988a72e3/TD_1_-_Matrices.pdf

1. Classiques

Exercice 1.1 (Print)

Question n°1 :

```
def printmat(M):
   for l in M:
      for e in l:
         print(e, end=" ")
      print()
```

Question n°2

```
pass
```

Exercice 1.2 : (Init & Load)

Question n°1

```
def init(l,c,val):
   M = []
   for i in range(l):
      lst = []
      for j in range(c):
         lst.append(val)
      M.append(lst)
   return M
```

Remarque :

```
def init_bad(l,c,val):
   l = []
   for i in range(c):
      l.append(val)
   M = []
   for j in range(l):
      M.append(line)
   return M
```

Même si ce code à une complexité inférieure au précédent, il pose un problème si l'on souhaite modifier la liste créée. Lorsque l'on modifiera une valeur à une certaine ligne, cette valeur sera également modifiée pour les autres lignes.

Exemple :

```
>>> M = init_bad(3,2,0)
>>> M
[[0,0],[0,0],[0,0]]
>>> M[1][0] = 42
>>> M
[[42,0],[42,0],[42,0]]
```

Question n°2

```
def load(filename):
   matrix = []
   with open(filename) as myfile:
      lines = myfile.readline
      for l in lines:
         lst = []
```

```
        val = ""
        for i in range(len(l)-1):
            if l[i] == " ":
                lst.append(int(val))
                val = ""
            else:
                val += l[i]
        matrix.append(lst)
    return matrix
```

```
# METHODE OPTI++
def load(filename):
    matrix = []
    with open(filename) as myfile:
        lines = myfile.readline
        while
    return matrix
```

▼ Exercice 1.3 : (Addition de matrices)

```
def add_matrices(A,B):
    (m1, m2) = (len(A), len(B))
    (n1, n2) = (len(A[0]), len(B[0]))
    M = []

    if m1 != m2 or n1 != n2:
        raise Exception()
    else
        for i in range(m1):
            lst = []
            for j in range(n1):
                lst.append(A[i][j] + B[i][j])
            M.append(lst)
        return M
```

▼ Exercice 1.4 (Produit de matrices)

```
def mult(A,B):
    m = len(A)
    n = len(A[0])
    p = len(B[0])

    if n != len(B):
        raise Exception
```

```
    M = []
    for i in range(m):
       lst = []
       for j in range(p):
          sum = 0
          for k in range(n):
             sum += (A[i][k] * B[k][j])
          lst.append(sum)
       M.append(lst)

   return M
```

▼ 2. Recherches et tests

▼ Exercice 2.1 : (Minimax)

```
def posMax(L):
    """ return the position of the max value in the mon-empty """
    mi = 0
    for i in range(1,len(L));
        if L[i] < L[mi]:
            mi = i
    return mi

def posMinimax(M):
    (mini, minj) = (0, posMax[0]))
    c = len(M[0])
    for i in range(1, len(M)):
        maxj = posMax (M[i])
        if M[mini][minj] > M[i][maxj]:
            (mini,minj) = (i,maxj)
    return (mini,minj)
```

▼ Exercice 2.2 : (Recherche)

```
def searchMatrix (M,x):
    fj = -1
    (l,c) = (len(M), lenM[0])
    i = 0
    while i < l and not fj == -1:
        j = 0
        while j < c and M[i][j] != x:
            j += 1
```

```
        if j < c:
            fj = j
        i += 1
    if fj != -1:
        return (i-1,fj)
    else
        return (-1,-1)

def searchMatrix (M,x):
    (l,c) = (len(M), lenM[0])
    i = 0
    j = c
    while i < l and j == c:
        j = 0
        while j < c and M[i][j] != x:
            j += 1
        i += 1
    if j < c:
        return (i-1,j)
    else
        return (-1,-1)
```

Exercice 2.3 : (Symétrique)

```
#V1 (Opti de code)
def symmetric(M):
    n = len(M) # car matrice carrée
    (i,j) = (0,n)
    while i < n and j == n:
        j = 0
        while j < n and M[i,j] == M[j,i]:
            j += 1
        i += 1
    return j == n # pas i == l car i++ meme si dernière ligne mauvaise

#V2 (Opti d'algo) : Partie supérieure
def symmetric(M):
    n = len(M)
    (i,j) = (0,n)
    while i < n and j == n:
        j = i + 1 # on commence après la diagonale
        while j < n - 1 and M[i,j] == M[j,i]: # on ignore la dernière ligne (vide)
            j += 1
        i += 1
    return j == n

#V2-bis (Opti d'algo) : Partie inférieure
def symmetric(M):
    n = len(M)
    (i,j) = (1,0)
```

```
    while i < n and j == i - 1:
        j = 0
        while j < i and M[i,j] == M[j,i]:
            j += 1
        i += 1
    return j == n-1 # j == i-1
```

3. A kind of magic

Exercice 3.1 : (Carré magique)

```
def magic_square(n):
    mat = matrix.init(n,n,None)
    (i,j) = (n-1, n//2)
    x = 0
    M[i][j] = x
    for _ in range(n):
        for _ in range(n-1):
            M[i][j] = x
            x += 1
        M[i][j] = x
        x += 1
        i -= 1
        if i < 0:
            i = n-1
    return mat
```

Exercice 3.2 : (Harry Potter)

Question n°1

```
### Glouton ###
def searchBestPath(l):
    ind = 0
    for i in range(1,len(l)):
        if l[i] < l[ind]:
            ind = i
    return ind

def harrypotter(T):
    w = len(M[0])
    jmax = 0
```

```
    for j in (1,w):
        if M[0][j] > M[0][jmax]:
            jmax = j
    nbstones = M[0][jmax]
    for i in range(1,len(M)):
        j = jmax
        if j > 0 and M[i][jmax-1] > M[i][j]:
            j = jmax - 1
        if j < w - 1 and M[i][jmax+1] > M[i][j]:
            j = jmax - 1
        jmax = j
        nbstones += M[i][jmax]
    return nbstones
```

```
def __brute(T):
    if i == len(M)-1:
        return M[i][j]
    else:
        bf = __brute(M,i+1,j)
        if j > 0:
            lf = __brute(M,i+1,j-1)
        if lf > bf:
            bf = lf
        if j < len(M[i]) - 1:
            rf = __brute(M,i+1,j+1)
            if rf > bf:
                bf = rf
        return M[i][j] + bf

def harrypotter(M):
    bf = 0
    for j in range(len(M[0])):
        f == __brute(M,0,j)
        if f > bf:
            bf = f
    return bf
```

```
def build_max_matrix1(T):
    (l,c) = (len(T),len(T[0])
    M = matrix.init(l,c,0)

    # cas d'arret : copie de la dernière ligne
    for j in range(c):
```

```
        M[l-1][j] = T[l-1][j]

    # cas général
    for i in range(l-2,-1,-1):
        M[i][0] = T[i][0] + max(M[i+1][0], M[i+1][1])
        for j in range(1, c-1):
            M[i][j] = T[i][j] + max(M[i+1][j-1],M[i+1][j], M[i+1][j+1]
        M[i][c-1] = T[i][c-1] + max(M[i+1][c-2], M[i+1][c-1])
    return M

def HarryPotter(T):
    l = len(T)
    M = build_max_matrix(T):
    m = 0
    for j in range(len(T[0])):
        m = max(m,M[0][j])
    return m



def build_max_matrix2(T):
    (l,c) = (len(T),len(T[0])
    M = matrix.init(l,c,0)

    # cas d'arret : copie de la première ligne
    for j in range(c):
        M[0][j] = T[0][j]

    # cas général
    for i in range(1,l):
        M[i][0] = T[i][0] + max(M[i-1][0], M[i-1][1])
        for j in range(1, c-1):
            M[i][j] = T[i][j] + max(M[i-1][j-1],M[i-1][j], M[i-1][j+1]
        M[i][c-1] = T[i][c-1] + max(M[i-1][c-2], M[i-1][c-1])
    return M

def HarryPotter(T):
    l = len(T)
    M = build_max_matrix(T):
    m = 0
    for j in range(len(T[0])):
        m = max(m,M[l-1][j])
    return m
```

Greedy : O(w + 3h)

Brut force : O(w * 3**h)

Dynamic : O(w * h)

▼ Question n°2

```
# Non traité
```