Rapport du TP 1 de Maths 3

par HARTMANN Matthias, groupe 2A

Table of Contents

Mise en application :

Fonction affichage_matrice :

Code :

```
def affichageMatrice(M:np.ndarray):
    """!
    @brief Cette fonction affiche une Matrice M de la façon suivante :
        [a,b,c]
        [d,e,f]
        [. . .]

    Paramètres :
        @param M : np.ndarray => Une matrice M

    """
    for ligne in M :
        print(ligne)
```

Test de bon fonctionnement :

```
Matrice : np.ndarray = np.array([[1,2,3], [4,5,6]])
affichageMatrice(Matrice)
```

Sortie :

```
[1 2 3]
[4 5 6]
```

Fonction my_identite :

Code :

```
def my_identite(n:int) -> np.array:
    """!
    @brief Cette fonction génère une matrice identité ayant n colonnes et n
lignes.

    Paramètres :
        @param n : int => Le nombre de colonnes et de lignes
    Retour de la fonction :
        @return np.array => La matrice identitée générée

    """
    M : np.ndarray = np.zeros([n, n])
    for col in range(n):
        M[col, col] = 1

    return M
```

Test de bon fonctionnement :

```
affichageMatrice(my_identite(3))
```

Sortie :

```
[1. 0. 0.]
[0. 1. 0.]
[0. 0. 1.]
```

Fonction suppr_pairs:

Code :

```
def suppr_pairs(M:np.ndarray) -> np.ndarray:
    """!
    @brief Cette fonction remplace tous les nombres pairs d'une matrice par des
0

    Paramètres :
        @param M : np.ndarray => Une matrice M
    Retour de la fonction :
        @return np.ndarray => La matrice transformée

    """
    size : Tuple[int, int] = np.shape(M)
    for ligne in M:
        for col in range(size[1]):
            if ligne[col] % 2 == 0:
                ligne[col] = 0
    return M
```

Test de bon fonctionnement :

```
MATRICE : np.ndarray = np.array([[2,3,4,6], [3,5,8,1]])
affichageMatrice(suppr_pairs(MATRICE))
```

Sortie :

```
[0 3 0 0]
[3 5 0 1]
```

Fonction compte:

Code :

```
def compte(M:np.ndarray, L: List[int]) -> int:
    """!
    @brief Cette fonction comptabilise le nombre de fois qu'un élément de la
liste L est
    retrouvé dans la matrice M

    Paramètres :
        @param M : np.ndarray => Une matrice M
        @param L : List[int] => Une liste de nombre
    Retour de la fonction :
        @return int => Le nombre de fois où un élément de la liste L est
retrouvé dans la matrice M

    """
    compteur : int = 0
    size : Tuple[int, int] = np.shape(M)
    for ligne in M:
        for col in range(size[1]):
            if ligne[col] in L:
                compteur += 1
    return compteur
```

Test de bon fonctionnement :

```
MATRICE : np.ndarray = np.array([[2,3,4,6], [3,5,1,1]])
print(compte(MATRICE, [1,4,5]))
```

Sortie :

```
4
```

Fonction my_transpose :

Code :

```
def my_transpose(M: np.ndarray) -> np.ndarray :
    """!
    @brief Cette fonction renvoie la transposée de la matrice M

    Paramètres :
        @param M : np.ndarray => Une matrice M
    Retour de la fonction :
        @return np.ndarray => la transposée de la matrice M

    """

    [nbLignes, nbCol] = np.shape(M)

    NV : np.ndarray = np.zeros((nbCol, nbLignes))

    for ligne in range(nbLignes):
        for col in range(nbCol):
            NV[col, ligne] = M[ligne, col]

    return NV
```

Test de bon fonctionnement :

```
MATRICE : np.ndarray = np.array([[1,2,3], [4,5,6]])
affichageMatrice(my_transpose(MATRICE))
```

Sortie :

```
[1. 4.]
[2. 5.]
[3. 6.]
```

Fonction applatir:

Code :

```
def applatir(M: np.ndarray) -> List[int or float]:
    """!
    @brief Cette fonction renvoie une liste contenant l'ensemble des valeurs de
la matrice M

    Paramètres :
        @param M : np.ndarray => Une matrice M
    Retour de la fonction :
        @return List[int or float] => La liste des éléments de M

    """

    liste : List[int or float] = []
    for ligne in range(np.shape(M)[0]):
        liste += list(M[ligne,:])
    return liste
```

Test de bon fonctionnement :

```
MATRICE : np.ndarray = np.array([[1,2,3], [4,5,6]])
print(applatir(MATRICE))
```

Sortie :

```
[1, 2, 3, 4, 5, 6]
```

Fonction egals_mat:

Code :

```
def egales_mat(M1: np.ndarray, M2: np.ndarray) -> bool:
    """!
    @brief Cette fonction compare 2 matrices et renvoie vrai si elles sont
identiques

    Paramètres :
        @param M1 : np.ndarray => Une matrice M1
        @param M2 : np.ndarray => Une matrice M2
    Retour de la fonction :
        @return bool => Un booléen. Vrai si les matrices sont identiques, faux
sinon.

    """
    resulat : bool = True

    if(np.shape(M1) == np.shape(M2)):
        [nbLignes, nbCol] = np.shape(M1)

        col : int = 0
        ligne : int = 0

        while ligne < nbLignes and resulat:

            resulat = M1[ligne, col] == M2[ligne, col]

            col += 1
            if(col >= nbCol):
                ligne += 1
                col = 0

        return resulat
    else:
        return False
```

Test de bon fonctionnement :

```
M1 : np.ndarray = np.array([[1,2,3], [4,5,6]])
M2 : np.ndarray = np.array([[1,1,3], [4,3,6]])
print(egales_mat(M1, M2))

M1 : np.ndarray = np.array([[1,1,3], [4,3,6]])
print(egales_mat(M1, M2))
```

Sortie :

```
False
True
```

Fonction symetrique:

Code :

```
def symetrique(M1: np.ndarray, M2: np.ndarray) -> bool:
    """!
    @brief Cette fonction renvoie vrai si la matrice M2 est la transposée de la
matrice M1

    Paramètres :
        @param M1 : np.ndarray => Une matrice M1
        @param M2 : np.ndarray => Une matrice M2
    Retour de la fonction :
        @return bool => Vrai si la matrice M2 est la transposée de la matrice
M1

    """

    return egales_mat(M2, my_transpose(M1))
```

Test de bon fonctionnement :

```
M1 : np.ndarray = np.zeros([2,3])
M2 : np.ndarray = np.zeros([3,2])
print(symetrique(M1, M2))

M1 : np.ndarray = np.zeros([2,3])
M2 : np.ndarray = np.zeros([3,3])
print(symetrique(M1, M2))
```

Sortie :

```
True
False
```

Fonction antisymetrique:

Code :

```
def antisymetrique(M1: np.ndarray, M2: np.ndarray) -> bool:
    """!
    @brief Cette fonction renvoie vrai si la matrice M2 est l'opposée de la
transposée de la matrice M1

    Paramètres :
        @param M1 : np.ndarray => Une matrice M1
        @param M2 : np.ndarray => Une matrice M2
    Retour de la fonction :
        @return bool => Vrai si la matrice M2 est l'opposée de la transposée de
la matrice M1

    """

    return egales_mat(M2, my_transpose(M1)* -1)
```

Test de bon fonctionnement :

```
M1 : np.ndarray = np.zeros([2,3])
M2 : np.ndarray = np.zeros([3,2])
print(antisymetrique(M1, M2))

M1 : np.ndarray = np.zeros([2,3])
M2 : np.ndarray = np.zeros([3,3])
print(antisymetrique(M1, M2))

M1 : np.ndarray = np.ones([2,3])
M2 : np.ndarray = np.ones([3,2])
print(antisymetrique(M1, M2))
```

Sortie :

```
True
False
False
```

sympa