Travail préparatoire en amont : lire le début du TP puis **Q.1** et **Q.2**.

L'objectif du TP est de manipuler les matrices en tant que listes de listes.

En mathématiques on appelle **matrice** un tableau de nombres (entiers, réels, complexes...).
Voici par exemple une matrice `A` à 2 lignes et 3 colonnes : $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.
Sous Python, on les représente par des **listes de listes** c'est-à-dire une liste dont les éléments sont eux-mêmes des listes :

```
A = [[1,2,3],[4,5,6]]
print(A)
```

L'algorithme précédent affiche `[[1,2,3],[4,5,6]]`.
Si on veut un affichage plus élégant, on peut utiliser le code suivant qui donne un affichage ligne par ligne.

```
def affiche(A) :
        for ligne in A :
                print(ligne)
        #la fonction ne renvoie rien
affiche(A)
```

Syntaxe de base pour les matrices (en tant que listes de listes) :

Opération	Syntaxe
Matrice (liste de listes)	`A=[[1,2,3],[4,5,6]]`
Afficher un élément d'une matrice	`A[1][2]` renvoie `6`.
Initialiser une matrice composée de zéros à n lignes et p colonnes	`M=[ [0 for j in range(p)] for i in range(n)]` Attention de ne pas inverser lignes et colonnes. Chaque ligne est composée de p éléments.
Modifier un élément d'une matrice	`A[1][2]=99` `print(A)` renvoie `A=[[1,2,3],[4,5,99]]`.
Nombre de lignes d'une matrice	`len(A)` renvoie `2`.
Afficher une ligne d'une matrice	`A[1]` renvoie `[4,5,6]`. Attention à l'indexation qui commence à 0.
Modifier une ligne d'une matrice	`A[1]=[7,8,9]` `print(A)` renvoie `A=[[1,2,3],[7,8,9]]`
Nombre de colonnes d'une matrice	`len(A[0])` renvoie `3`. On compte en fait le nombre d'élément de la première ligne.
Afficher une colonne d'une matrice	`[A[i][2] for i in range(len(A))]` renvoie `[3,6]`. C'est plus difficile de travailler avec les colonnes.
Modifier une colonne d'une matrice	`for i in range(len(A)): A[i][2]=9` `print(A)` renvoie `A=[[1,2,9],[4,5,9]]`.

Complément sur la création d'une matrice de 0 :

Pour créer une liste de 0, on peut procéder **par duplication**. Par exemple : `L = [0]*6`
Malheureusement avec une liste de listes, un problème survient :

```
A = [[0]*2]*3
print(A) # affiche [[0, 0], [0, 0], [0, 0]]
A[2][0]=12
print(A) # affiche [[12, 0], [12, 0], [12, 0]]
```

On pense ne modifier qu'un seul coefficient (le premier de la deuxième ligne) mais toutes les lignes ont été modifiées ! L'explication est que Python a fait pointer toutes les lignes vers la même adresse (en tapant `id(A[0])==id(A[1])` dans la console, on obtient `True`). Pour remédier à ce problème, on procède plutôt **par compréhension** :

```python
A = [[0]*2 for i in range(3)]
print(A) # affiche [[0, 0], [0, 0], [0, 0]]
A[2][0]=12
print(A) # affiche [[0, 0], [0, 0], [12, 0]]
id(A[0])==id(A[1]) or id(A[1])==id(A[2])# affiche False
```

Les lignes pointent dans la mémoire vers des adresses différentes et donc on peut modifier une ligne sans modifier les autres. La difficulté rencontrée par duplication est souvent la source de nombreux bugs. Vous avez simplement à retenir que **pour créer une matrice il ne faut pas procéder par duplication mais par compréhension**.

À vous de jouer. On testera les fonctions de **Q.1** *à* **Q.3** *avec* $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ *en laissant une trace dans le script.*

Q.1 Écrire une fonction `Somme` qui prend en argument une matrice `A` et renvoie la somme des tous les coefficients de la matrice `A`.

Q.2 Écrire une fonction `Maximum` qui prend en argument une matrice `A` et renvoie le maximum des coefficients de la matrice `A`.

Q.3 La transposée d'une matrice A est une matrice dont les lignes sont les colonnes de A.

Par exemple, si $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ alors la transposée de A est $\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$.

Écrire une fonction `Transpose` qui prend en argument une matrice `A` et renvoie la transposée de `A`.
Cette fonction ne doit pas modifier la variable `A`.
Si `A` possède `n` lignes et `p` colonnes, il faudra donc créer une matrice `B` à `p` lignes et `n` colonnes dont tous les coefficients sont initialisés à 0 puis les modifier avec l'instruction `B[i][j] = A[j][i]`.

Q.4 Compléter la fonction suivante qui a pour but de calculer le produit matriciel de `A` par `B`.

```python
def Produit(A,B) :
        n = ... # nombre de lignes de A
        p = ... # nombre de colonnes de A = nombre de lignes de B
        q = ... # nombre de colonnes de B
        C = ... # matrice de n lignes et q colonnes composée de zéros
        for i in range(...) :
                for j in range(...) :
                        somme = 0 # variable qui va contenir le coefficient d'indice (i,j)
                        for k in range(...) :
                                somme = somme + ...
                        C[i][j] = somme
        return C
```

On rappelle que le produit matriciel de $A = (a_{i,j}) \in \mathcal{M}_{n,p}(\mathbb{C})$ par $B = (b_{i,j}) \in \mathcal{M}_{p,q}(\mathbb{C})$ est la matrice $C = (c_{i,j}) \in \mathcal{M}_{n,q}(\mathbb{C})$ dont les coefficients sont données par la formule suivante : $c_{i,j} = \sum_{k=1}^{p} a_{i,k} b_{k,j}$.

Tester votre fonction avec $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ et $B = \begin{pmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{pmatrix}$. On doit trouver $AB = \begin{pmatrix} 16 & 22 \\ 34 & 49 \end{pmatrix}$.

Q.5 En utilisant la question précédente, écrire une fonction `Puissance` d'arguments une matrice carrée `A` et un entier naturel non nul `m` qui renvoie la puissance m-ième de la matrice A i.e. A^m.

Tester votre fonction pour $A = \begin{pmatrix} 2 & 4 \\ 1 & 1 \end{pmatrix}$, on doit trouver $A^4 = \begin{pmatrix} 100 & 156 \\ 39 & 61 \end{pmatrix}$.

Les trois prochaines questions sont pour les plus rapides. Ces derniers prendront soin de tester leurs fonctions.

Q.6 Écrire une fonction `MaxUn` qui prend en argument une matrice `A` de 0 et de 1 (par exemple une image en noir et blanc représentant un QR code) et renvoie le nombre maximal de 1 sur une ligne (le nombre maximal de pixels noirs sur une ligne).

Q.7 Écrire une fonction `IndLigne` qui prend en argument une matrice `A` de 0 et de 1 et renvoie en sortie l'indice d'une des lignes possédant le plus de 1.

Q.8 Écrire une fonction `NbUn` qui prend en argument une matrice `A` de 0 et de 1 et renvoie le nombre de lignes possédant au moins un 1.

Complément sur l'effet de bord :

Pour terminer, rappelons que les listes en Python peuvent être modifiées par une fonction dont elles sont l'argument : on parle **d'effet de bord**. En effet, la fonction ne reçoit pas la valeur de la variable en argument mais son adresse mémoire, ce qui lui permet de modifier la valeur stockée dans la variable.
On peut faire l'analogie avec une bibliothèque :

- si on emprunte une photocopie d'un livre et qu'on l'annote ou qu'on arrache une page, cela ne modifie pas le livre original,
- si on emprunte le livre original alors toute modification faite sur ce livre sera irréversible : c'est dans ce cas qu'on parle **d'effet de bord**.

On expose ci-dessous une fonction qui va modifier la matrice `A` par effet de bord.

```
def prog1(A):
        A[0][0] = 12
#cette fonction ne renvoie rien puisqu'il n'y pas de return! (en fait Python lui fait faire return None)
A = [ [1,2,3],[4,5,6] ]
prog1(A) # on ne doit pas exécuter A = prog1(A) puisque la fonction prog1() renvoie None
print(A) # affiche [[12,2,3],[4,5,6]]
```

Application à la résolution de systèmes via l'algorithme du pivot de Gauss :

Q.9 Écrire une fonction `Permutation` d'arguments une matrice `A` et deux indices `i` et `j` dont le but est d'échanger les lignes d'indices `i` et `j` de la matrice `A` (notation mathématique : $L_i \leftrightarrow L_j$). La matrice `A` doit être modifiée par effet de bord. Cette fonction ne renvoie rien.

Q.10 Écrire une fonction `Dilatation` d'arguments une matrice `A`, un indice `i`, et d'un scalaire `c` non nul dont le but est de multiplier la ligne d'indice i par c (notation mathématique : $L_i \leftarrow cL_i$). La matrice `A` doit être modifiée par effet de bord. Cette fonction ne renvoie rien.

Q.11 Écrire une fonction `Transvection` d'arguments une matrice `A`, deux indices `i` et `j`, et d'un scalaire `c` dont le but est d'ajouter c fois la ligne d'indice j à la ligne d'indice `i` (notation mathématique : $L_i \leftarrow L_i + cL_j$). La matrice `A` doit être modifiée par effet de bord. Cette fonction ne renvoie rien.

Q.12 Utiliser ces fonctions dans un script pour résoudre le système d'inconnues réelles suivant :

$$\begin{cases} 3x - y + 6z = -1 \\ -x + y + 2z = 1 \\ 2x + 7y + 4z = -2 \end{cases}$$

Pour ce faire, considérer l'écriture matricielle de ce système : $\begin{pmatrix} 3 & -1 & 6 & -1 \\ -1 & 1 & 2 & 1 \\ 2 & 7 & 4 & -2 \end{pmatrix}$.
Ensuite, utiliser les opérations élémentaires du pivot de Gauss sur les lignes (permutations, dilatations, transvections) qui préservent les équivalences afin d'obtenir une forme matricielle comme ci-contre : $\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \end{pmatrix}$.
Enfin, conclure en utilisant le fait que cette dernière écriture matricielle correspond à $\begin{cases} x = a \\ y = b \\ z = c \end{cases}$.

Q.13 *Question bonus :* Automatiser la résolution de l'algorithme du pivot de Gauss pour un système linéaire quelconque (mais en admettant qu'il admet une unique solution).
Vérifier votre programme sur le système suivant :

$$\begin{cases} x + 5z = -3 \\ 2x + y + 6z = 7 \\ 3x + 4y = 12 \end{cases}$$

Module NumPy

Le module NumPy est une librairie de calcul numérique permettant notamment de manipuler des tableaux de dimension quelconque. C'est le module qui est utilisé en pratique pour la manipulation de matrices. Afin de ne pas conserver une syntaxe trop lourde, la plupart du temps on l'importe en tant que `np`.

```
1 import numpy as np
```

Syntaxe de base pour les matrices (via NumPy) :

Opération	Syntaxe avec `NumPy`
Matrice (tableau `ndarray`)	`A = np.array([[1, 2, 3], [4, 5, 6]])`
Afficher un élément d'une matrice	`print(A[1, 2])` renvoie 6.
Initialiser une matrice composée de zéros à n lignes et p colonnes	`A = np.zeros((n, p))`
Modifier un élément d'une matrice	`A[1, 2] = 99`
Nombre de lignes d'une matrice	`A.shape[0]` renvoie 2.
Afficher une ligne d'une matrice	`print(A[1])` renvoie `[4 5 6]`
Modifier une ligne d'une matrice	`A[1] = [7, 8, 9]`
Nombre de colonnes d'une matrice	`A.shape[1]` renvoie 3.
Afficher une colonne d'une matrice	`print(A[:, 2])` renvoie `[3 6]`
Modifier une colonne d'une matrice	`A[:, 2] = [6,9]`
Produit matriciel de A par B	`np.dot(A,B)`

Vous serez amenés à croiser ce module dans les matières scientifiques. L'objectif de ce TP était de manipuler les matrices en tant que liste de listes, mais vous pouvez à l'avenir utiliser le module NumPy s'il n'y a pas de contre-indications.

Dès qu'on traite de grandes quantités de données, les opérations sur les tableaux NumPy sont considérablement plus rapides que sur les listes de listes : NumPy est le module adapté pour le calcul numérique en général. Cependant, les tableaux NumPy n'ont pas une portée aussi générale que les listes. En particulier, ils doivent contenir des éléments de même type, et ils ne sont pas conçus pour être redimensionnés de manière efficace une fois qu'on les a créés, contrairement aux listes. Les listes restent donc adaptées dans beaucoup de situations de programmation générale.