Matrices

I) Cours

Une matrice de nombres (entiers ou flottants) est une liste de listes de même longueur de nombres.

```
a = [[1,2,3,4],[4,5,6,7],[7,8,9,10]]      # On définit une matrice a de taille  3x4
print(a[1])    ⟶ [4, 5, 6, 7]               # Deuxième ligne de a
print(a[1][2]) ⟶ 6                        # Elément a23
```

Toutes les fonctions et méthodes utilisables sur les listes permettent de travailler sur les matrices.

```
print(len(a), len(a[0])) ⟶ 3,4  # Nombre de lignes et de colonnes de a
```

A Attention:

- A l'indiçage à partir de 0 pour les matrices informatiques et de 1 pour les matrices mathématiques
- A la copie d'une matrice : utiliser `copy.deepcopy()`, après avoir importé le module copy.
- `m = [[0] * n] * n` crée une matrice nulle dont les lignes ne sont pas indépendantes.

Pour créer une matrice de terme général donné

- `m = [[0] * n for i in range(n)]` crée une matrice m remplie de 0 et composée de lignes indépendantes,
- `m = [[f(i, j) for j in range(p)] for i in range(n)]` crée la matrice m de taille n×p et de terme général $m_{i,j} = f(i-1, j-1)$

II) Exercices

1) Affichage d'une matrice

Q1) Ecrire une fonction affiche_matrice(a) affichant une matrice ligne par ligne.

```
a = [[1,2,3,4],[4,5,6,7],[7,8,9,10]]
affiche_matrice(a) #renvoie
[1, 2, 3, 4]
[4, 5, 6, 7]
[7, 8, 9, 10]
```

---

2) Définir une matrices particulière

La fonction identite(n) suivante calcule la matrice identité d'ordre n :

```
def identite(n):
    """ Matrice identité d'ordre n """
    def f(i, j):        # f est une sous fonction
        if i == j:
            return 1
        return 0
    return [[f(i,j) for j in range(n)] for i in range(n)]
```

```
affiche_matrice(identite(2))  # renvoie
[1, 0]
[0, 1]
```

Q2) Sur le même modèle ou plus simplement encore, écrire (et tester) les fonctions calculant les matrices de taille n×n ($n \in \mathbb{N}^*$) :

$$\text{diag1(n)} = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ 2 & 1 & 2 & \ddots & \vdots \\ 3 & 2 & \ddots & \ddots & 3 \\ \vdots & \ddots & \ddots & \ddots & 2 \\ n & \cdots & 3 & 2 & 1 \end{pmatrix} \qquad \text{diag2(n)} = \begin{pmatrix} n & n-1 & \cdots & 2 & 1 \\ n-1 & \iddots & 2 & 1 & 2 \\ \vdots & \iddots & \iddots & \iddots & \vdots \\ 2 & \iddots & \iddots & \iddots & n-1 \\ 1 & 2 & \cdots & n-1 & n \end{pmatrix}$$

3) Opérations élémentaires sur les lignes

On écrit des fonctions effectuant les opérations élémentaires. On les écrit sous forme de fonctions “sans retour” (on dit parfois des procédures) pour pouvoir les exploiter dans le prochain TP où l’on calculera l’inverse d’une matrice carrée.

```
def dilatation(a, i, x):
    """ Dilatation L(i) <-- x * L(i) sur la matrice a """
    p = len(a[0])   # p est le nombre de colonnes de a
    for k in range(p):
        a[i][k] = x * a[i][k]
```

La complexité de cette fonction est $O(p)$. Pour la tester:

```
m = diag1(3)
dilatation(m, 1, -3)
affiche_matrice(m) # renvoie
[1, 2, 3]
[-6, -3, -6]
[3, 2, 1]
```

Q3) Sur le même modèle, écrire et tester les fonctions (sans retour) permutation(a, i, j) et transvection(a, i, j, x) effectuant les opérations élémentaires $L_i \longleftrightarrow L_j$ et $L_i \longleftarrow L_i + x * L_j$ sur la matrice a et préciser leur complexité.

III) Pour aller plus loin

1) Calcul matriciel

Q4) Ecrire des fonctions transpose(a), somme(a, b) , produit(a, b), (avec a, b des matrices) calculant si c’est possible les matrices ${}^t a$, a + b, a × b, Lorsque les matrices sont carrées $n \times n$, préciser leur complexité.

2) La biblothèque Numpy

Cette immense bibliothèque permet entre autres de manipuler les matrices.

```
import numpy as np      # Importation du module avec un alias
a = [[1, 2], [3, 4]]
b = np.array(a)     # Conversion en ndarray (objet de base numpy)
print(a, type(a))
print(b, type(b))
# renvoie
[[1, 2], [3, 4]] <class 'list'>
[[1 2]
 [3 4]] <class 'numpy.ndarray'>
```

Beaucoup de fonctions évoluées sont accessibles. On peut aussi directement opérer sur les matrices ou sur les lignes des matrices:

```
np.dot(b,c)    # Produit      np.linalg.matrix_power(b, 3)  # Puissance
np.linalg.det(b)  # Déterminant  np.linalg.inv(b) # Inverse
b[1] + 2*[2]- 3*b[0]          # Calcul sur les lignes de la matrice b
b - 2*c                          # Somme, produit par un nombre
```

Faire divers essais.