

Fiche méthode : matrices

Matrices – Listes de listes de tuples – Tableaux 3D

1. Représenter une matrice

Une matrice (ou une image) de p lignes et q colonnes est une liste de p listes de longueur q . $M[i]$ est la ligne i et $M[i][j]$ le coefficient de la ligne i , colonne j .

```
M = [[1, 2, 3],
      [4, 5, 6]]
p = len(M)      # 2 lignes
q = len(M[0])   # 3 colonnes
M[1][2]         # 6
```

	j = 0	j = 1	j = 2
i = 0	1	2	3
i = 1	4	5	6

● Méthode

Toujours dessiner la matrice de l'exemple sous forme de tableau, avec les numéros de lignes et de colonnes : le premier indice est la **ligne**, le second la **colonne**.

2. Parcourir : deux boucles imbriquées

```
# par valeur
for ligne in M:
    for x in ligne:
        ... x ...
```

```
# par indices (si i ou j sont utiles)
for i in range(len(M)):
    for j in range(len(M[0])):
        ... M[i][j] ...
```

● Méthode

- La boucle extérieure choisit la ligne, la boucle intérieure parcourt cette ligne. Pour chaque valeur de i , j prend toutes ses valeurs.
- Un accumulateur qui concerne **toute la matrice** s'initialise avant les deux boucles ; un accumulateur qui concerne **chaque ligne** s'initialise entre les deux boucles.

On parcourt...	Boucle
la colonne j	<code>for i in range(len(M)):</code> avec <code>M[i][j]</code>
la diagonale (matrice carrée)	<code>for i in range(len(M)):</code> avec <code>M[i][i]</code>
les couples $i < j$ (au-dessus de la diagonale)	<code>for i in range(n):</code> puis <code>for j in range(i + 1, n):</code>

3. Construire une nouvelle matrice

On construit chaque ligne, puis on l'ajoute au résultat.

```
def nouvelle_matrice(M: list) -> list:
    '''Renvoie la matrice des f(M[i][j]).'''
    res = []
    for i in range(len(M)):
        ligne = []
        for j in range(len(M[0])):
            # nouvelle ligne à chaque tour
```

```

        ligne.append(f(M[i][j]))
    res.append(ligne)                # ligne terminée
return res

```

4. Les voisins d'une case

Dans une image ou un plateau de jeu, on a souvent besoin des cases voisines de (i, j) . Il faut **rester dans la matrice**.

```

def voisins(M: list, i: int, j: int) -> list:
    '''Renvoie la liste des couples (a, b) des cases voisines de (i, j)
    (haut, bas, gauche, droite) qui sont dans la matrice.'''
    p, q = len(M), len(M[0])
    res = []
    for (a, b) in [(i - 1, j), (i + 1, j), (i, j - 1), (i, j + 1)]:
        if 0 <= a < p and 0 <= b < q:      # on élimine les cases hors bords
            res.append((a, b))
    return res

```

● Attention !

Sans le test $0 \leq a < p$, l'indice p provoque une `IndexError`, mais l'indice -1 ne provoque pas d'erreur : $M[-1]$ est la dernière ligne. Le résultat est faux sans message.

5. Listes de listes de tuples

Dans une image en couleurs, chaque pixel est un triplet (R, G, B) : $img[i][j]$ est un tuple, $img[i][j][0]$ sa composante rouge.

```

img = [(255, 0, 0), (0, 0, 255)],
      [(0, 0, 0), (255, 255, 255)]
for ligne in img:
    for (r, g, b) in ligne:          # dépaquetage du triplet
        ... r ... g ... b ...

```

● Attention !

Un tuple n'est **pas modifiable** : $img[i][j][0] = 128$ provoque une `TypeError`. On remplace le pixel entier : $img[i][j] = (128, g, b)$.

6. Tableaux à trois dimensions

Un tableau 3D est une liste de matrices. Exemple : une vidéo en niveaux de gris est une liste d'images ; $V[t][i][j]$ est le pixel (i, j) de l'image numéro t .

```

nt = len(V)          # nombre d'images
p = len(V[0])        # lignes d'une image
q = len(V[0][0])     # colonnes

```

```

for t in range(nt):
    for i in range(p):
        for j in range(q):
            ... V[t][i][j] ...

```

● Méthode

- Chaque indice supplémentaire ajoute une boucle ; l'ordre des boucles suit l'ordre des indices.
- Pour construire un tableau 3D, construire une matrice à chaque tour de la boucle extérieure (et une ligne à chaque tour de la boucle du milieu).
- Si l'on ne travaille que sur une image, fixer t et se ramener à une matrice : $V[t]$.

7. Les pièges

● Attention !

1. `[[0] * q] * p` crée p fois la même ligne. Construire les lignes une à une.
2. Confondre `len(M)` (nombre de lignes) et `len(M[0])` (nombre de colonnes) : erreur invisible sur une matrice carrée, `IndexError` sur une matrice rectangulaire.
3. Écrire `M[j][i]` au lieu de `M[i][j]`.
4. Oublier de réinitialiser l'accumulateur ou la ligne entre deux lignes.
5. Sortir de la matrice en cherchant les voisins d'une case du bord.

8. Exemple rédigé

Écrire une fonction `negatif(img: list) -> list` qui renvoie le négatif d'une image en niveaux de gris (pixels entiers de 0 à 255) : chaque pixel v devient $255 - v$.

1. **Exemple** : `[[0, 100], [255, 55]]` $\rightarrow$ `[[255, 155], [0, 200]]`.
2. **Entrée / sortie** : une liste de listes d'entiers ; une *nouvelle* liste de listes de mêmes dimensions.
3. **Schéma** : construire une matrice, pixel par pixel.
4. **Code** :

```
def negatif(img: list) -> list:
    '''img : image de p lignes et q colonnes, pixels entre 0 et 255.
    Renvoie une nouvelle image dont chaque pixel v est remplacé par 255 - v.'''
    res = []
    for ligne in img:
        nouvelle = []
        for v in ligne:
            nouvelle.append(255 - v)
        res.append(nouvelle)
    return res
```