Tableaux à deux dimensions

RÉSUMÉ

Beaucoup d'informations s'organisent naturellement en lignes et colonnes : une image est une grille de pixels, un tableur est un tableau de cellules, un plateau de jeu est une grille de cases. En Python, on représente ces structures à l'aide de **listes de listes**. Maîtriser les tableaux à deux dimensions, c'est savoir parcourir, modifier et exploiter ces grilles, ce qui est indispensable dès que l'on travaille sur des images, des cartes, des matrices ou des jeux.

① MÉMO

Qu'est-ce qu'un tableau 2D ?

Un **tableau à deux dimensions** (ou matrice) est une liste de listes. Chaque sous-liste représente une ligne :

```
grille = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]
```

Accès `grille[i][j]` donne l'élément à la ligne `i`, colonne `j`. Les indices commencent à 0.
Nombre de lignes `len(grille)`.
Nombre de colonnes `len(grille[0])` (en supposant que toutes les lignes ont la même taille).

Créer un tableau 2D

Par compréhension (méthode correcte) :

```
# Matrice n lignes, p colonnes, remplie de 0
M = [[0] * p for _ in range(n)]
```

Piège avec la multiplication :

```
# INCORRECT : crée n références vers la MÊME ligne !
M = [[0] * p] * n
```

Avec la méthode incorrecte, modifier `M[0][0]` modifie aussi `M[1][0]`, `M[2][0]`, etc., car toutes les lignes sont le même objet en mémoire.

Parcourir un tableau 2D

Double boucle par indice :

```python
for i in range(len(grille)):
    for j in range(len(grille[i])):
        print(grille[i][j], end=' ')
    print()  # retour à la ligne
```

Double boucle par valeur :

```python
for ligne in grille:
    for element in ligne:
        print(element, end=' ')
    print()
```

Le parcours par indice est indispensable dès que l'on doit modifier les éléments ou connaître leur position.

Opérations courantes

Ligne i `grille[i]` renvoie la liste correspondant à la ligne entière.
Colonne j `[grille[i][j] for i in range(len(grille))]` construit la colonne par compréhension.
Diagonale principale Les éléments `grille[i][i]` pour i de 0 à $n-1$ (matrice carrée).
Transposée `[[grille[i][j] for i in range(n)] for j in range(p)]`.

Pièges fréquents

- Inverser lignes et colonnes : `grille[i][j]` désigne ligne i, colonne j (pas l'inverse).
- Créer le tableau avec `[[0]*p]*n` au lieu de la compréhension (alias de lignes).
- Oublier que `grille[i]` renvoie une **référence** vers la ligne, pas une copie : modifier le résultat modifie la grille.
- Confondre le nombre de lignes (`len(grille)`) et le nombre de colonnes (`len(grille[0])`).

Erreurs classiques

Code erroné	Code correct	Explication
`M = [[0]*3]*4` puis `M[0][0] = 5`	`M = [[0]*3 for _ in range(4)]`	Avec `*4`, les quatre lignes sont le même objet : modifier l'une modifie toutes les autres.
`grille[j][i]` pour (ligne i, colonne j)	`grille[i][j]`	Le premier indice est toujours la ligne, le second la colonne.
`for i in range(len(grille[0]))` pour les lignes	`for i in range(len(grille))`	`len(grille)` donne le nombre de lignes, `len(grille[0])` le nombre de colonnes.
Modifier `ligne = grille[i]` puis `ligne[0] = 99`	Travailler directement sur `grille[i][0] = 99`	`ligne` est une référence : la modification affecte la grille originale dans les deux cas, mais le piège est de croire que `ligne` est indépendante.

② EXEMPLES

Programme 6 · Créer et afficher une grille

```python
n, p = 3, 4
grille = [[0] * p for _ in range(n)]

# Remplir avec des valeurs
for i in range(n):
    for j in range(p):
        grille[i][j] = i * p + j

# Afficher proprement
for ligne in grille:
    print(ligne)
# [0, 1, 2, 3]
# [4, 5, 6, 7]
# [8, 9, 10, 11]
```

Programme 7 · Extraire une colonne

```python
matrice = [[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]

colonne_1 = [matrice[i][1] for i in range(len(matrice))]
print(colonne_1)  # [2, 5, 8]
```

Programme 8 · Somme de tous les éléments

```python
matrice = [[1, 2, 3],
           [4, 5, 6]]

total = 0
for ligne in matrice:
    for element in ligne:
        total += element
print(total)  # 21
```

Programme 9 · Image en niveaux de gris (tableau 2D)

```python
# Une image en niveaux de gris est un tableau 2D
# où chaque valeur est comprise entre 0 (noir) et 255 (blanc)
image = [[0,   0,   0,   0,   0],
         [0, 255, 255, 255,   0],
         [0, 255,   0, 255,   0],
         [0, 255, 255, 255,   0],
         [0,   0,   0,   0,   0]]

# Négatif : inverser chaque pixel
negatif = [[255 - image[i][j]
            for j in range(len(image[0]))]
           for i in range(len(image))]
```

③ EXERCICES

Exercice 1 *Manipulations de base*

1. Écrire une fonction `somme_ligne(M, i)` qui renvoie la somme des éléments de la ligne i d'une matrice `M`.
2. Écrire une fonction `somme_colonne(M, j)` qui renvoie la somme des éléments de la colonne j.
3. Écrire une fonction `maximum(M)` qui renvoie la plus grande valeur d'une matrice, ainsi que sa position `(i, j)`.

Exercice 2 *Transposée* Écrire une fonction `transposee(M)` qui renvoie la transposée d'une matrice (les lignes deviennent les colonnes et réciproquement). Ne pas utiliser de compréhension de liste.

Exercice 3 *Damier* Écrire une fonction `damier(n)` qui crée une matrice $n \times n$ représentant un damier : `0` pour les cases blanches, `1` pour les cases noires. La case $(0, 0)$ est blanche.

Exemple. `damier(4)` donne :

```
[[0, 1, 0, 1],
 [1, 0, 1, 0],
 [0, 1, 0, 1],
 [1, 0, 1, 0]]
```

Exercice 4 *Voisins dans une grille*

1. Écrire une fonction `voisins(M, i, j)` qui renvoie la liste des valeurs des voisins (haut, bas, gauche, droite) de la case (i, j) dans une matrice `M`, en excluant les voisins hors limites.
2. Écrire une fonction `moyenne_voisins(M, i, j)` qui renvoie la moyenne des voisins de (i, j) (utile pour le floutage d'image).

SOLUTIONS DES EXERCICES

Corrigé de l'exercice 1.

```python
def somme_ligne(M, i):
    s = 0
    for j in range(len(M[i])):
        s += M[i][j]
    return s

def somme_colonne(M, j):
    s = 0
    for i in range(len(M)):
        s += M[i][j]
    return s

def maximum(M):
    maxi = M[0][0]
    pos = (0, 0)
    for i in range(len(M)):
        for j in range(len(M[i])):
            if M[i][j] > maxi:
                maxi = M[i][j]
                pos = (i, j)
    return maxi, pos
```

Vérification : pour $M = \begin{pmatrix} 1 & 9 & 3 \\ 4 & 5 & 6 \end{pmatrix}$:

- `somme_ligne(M, 0)` $= 1 + 9 + 3 = 13$;
- `somme_colonne(M, 1)` $= 9 + 5 = 14$;
- `maximum(M)` $= (9, (0, 1))$.

Corrigé de l'exercice 2.

```python
def transposee(M):
    n = len(M)        # nombre de lignes
    p = len(M[0])     # nombre de colonnes
    T = [[0] * n for _ in range(p)]
    for i in range(n):
        for j in range(p):
            T[j][i] = M[i][j]
    return T
```

Principe : l'élément en position (i, j) dans M se retrouve en position (j, i) dans T. La transposée d'une matrice $n \times p$ est une matrice $p \times n$.

Vérification : si $M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$, alors $T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$. ✓

Corrigé de l'exercice 3.

```python
def damier(n):
    M = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            M[i][j] = (i + j) % 2
    return M
```

Principe : la parité de $i + j$ détermine la couleur. Si $i + j$ est pair, la case est blanche (0) ; si $i + j$ est impair, la case est noire (1).

Vérification : $(0,0) : 0 + 0 = 0$ pair $\longrightarrow$ 0. $(0,1) : 0 + 1 = 1$ impair $\longrightarrow$ 1. $(1,1) : 1 + 1 = 2$ pair $\longrightarrow$ 0. ✓

Corrigé de l'exercice 4.

```python
def voisins(M, i, j):
    n = len(M)
    p = len(M[0])
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    result = []
    for di, dj in directions:
        ni, nj = i + di, j + dj
        if 0 <= ni < n and 0 <= nj < p:
            result.append(M[ni][nj])
    return result

def moyenne_voisins(M, i, j):
    v = voisins(M, i, j)
    return sum(v) / len(v)
```

Principe : on teste les quatre directions. La condition `0 <= ni < n and 0 <= nj < p` garantit qu'on reste dans les limites de la matrice. Une case dans un coin n'a que deux voisins, une case sur un bord en a trois, les autres en ont quatre.

Vérification : pour $M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ et $(i, j) = (1, 1)$, les voisins sont 2, 8, 4, 6, de moyenne $\frac{20}{4} = 5$. ✓