

TP2 MPSI ITC 2025 Matrices (tableaux bi-dimensionnels)

Objectif. Manipuler des matrices (listes de listes) en Python : création, recherche, addition, produit matriciel, translation circulaire à droite et à gauche.

Rappels et conventions

- On représente une matrice comme une liste de lignes : `[[1,2,3],[4,5,6]]`.
- Les indices commencent à 0.
- On n'utilise pas de bibliothèques externes (ni `numpy`, etc.).
- Pour chaque fonction, un **exemple d'appel** et son **résultat attendu** sont fournis ci-dessous.

1) Création et recherche

Question 1.

📌 `create(n, m)` créer une matrice de zéros

```
1 def create(n: int, m: int) -> list[list[int]]:
2     """Créer et retourner une matrice n x m remplie de zéros."""
3     # A ECRIRE
4     pass
```

```
>>> create(2, 3)
[[0, 0, 0], [0, 0, 0]]
```

Question 2.

📌 `search(mat, x)` positions d'un élément

```
1 def search(mat: list[list[int]], x: int) -> list[tuple[int, int]]:
2     """Retourner la liste des positions (i, j) où
3         x apparaît dans mat (indices 0-based)."""
4     # A ECRIRE
5     pass
```

```
>>> M = [[1, 2, 3],
...       [4, 2, 6],
...       [2, 8, 9]]
>>> search(M, 2)
[(0, 1), (1, 1), (2, 0)]
>>> search(M, 5)
[]
```

2) Opérations élémentaires sur les matrices

Question 3.

📌 `add(m1, m2)` addition terme à terme

```
1 def add(m1: list[list[int]], m2: list[list[int]]) -> list[list[int]]:
2     """Additionner deux matrices de mêmes dimensions et retourner le résultat."""
3     # A ECRIRE
4     pass
```

```
>>> add([[1, 2], [3, 4]], [[5, 6], [7, 8]])
[[6, 8], [10, 12]]
```

Question 4.

📌 `prod(m1, m2)` produit matriciel

```
1 def prod(m1: list[list[int]], m2: list[list[int]]) -> list[list[int]]:
2     """Calculer le produit matriciel m1 (n x p) par m2 (p x m) -> (n x m)."""
3     # A ECRIRE
4     pass
```

```
>>> prod([[1, 2, 3], [4, 5, 6]], [[7, 8], [9, 10], [11, 12]])
[[58, 64], [139, 154]]
```

Question 5.

📌 `transpose(m)` transposée

```
1 def transpose(m: list[list[int]]) -> list[list[int]]:
2     """Echanger lignes et colonnes de la matrice m."""
3     # A ECRIRE
4     pass
```

```
>>> transpose([[1, 2, 3], [4, 5, 6]])
[[1, 4], [2, 5], [3, 6]]
```

Question 6.

📌 `antitranspose(m)` antitransposée (symétrie seconde diagonale)

```
1 def antitranspose(m: list[list[int]]) -> list[list[int]]:
2     """Symétrie par rapport à la 'seconde diagonale' i+j=n-1
3     (définie pour les matrices n x n)."""
4     # A ECRIRE
5     pass
```

```
>>> antitranspose([[1, 2, 3],
...                [4, 5, 6],
...                [7, 8, 9]])
[[9, 6, 3],
 [8, 5, 2],
 [7, 4, 1]]
```

Question 7.

📌 `flat(m)` aplatir une matrice en liste

```
1 def flat(m: list[list[int]]) -> list[int]:
2     """Retourner la liste des éléments de la matrice m, lus ligne par ligne."""
3     # A ECRIRE
4     pass
```

```
>>> flat([[1, 2, 3], [4, 5, 6]])
[1, 2, 3, 4, 5, 6]
```

Question 8.

📌 `unflat(lst, n, m)` reconstruire une matrice depuis une liste

```
1 def unflat(lst: list[int], n: int, m: int) -> list[list[int]]:
2     """Transformer une liste de n*m nombres en matrice n x m."""
3     # A ECRIRE
4     pass
```

```
>>> unflat([1, 2, 3, 4, 5, 6], 2, 3)
[[1, 2, 3], [4, 5, 6]]
```

4) Pistes de tests

- Vérifier `add` sur des matrices 1×1 , vides (`[]`) et rectangulaires.
- Tester `prod` avec dimensions incompatibles (erreur attendue) puis compatibles
- Vérifier que `transpose(transpose(m)) == m`.
- Pour une matrice carrée $n \times n$, vérifier que `antitranspose(antitranspose(m)) == m`.
- `m==unflat(flat(m),len(m),len[m[0]])`