Les listes en Python

① MÉMO

Créer et accéder

Création `L = [3, 1, 4, 1, 5]` ou `L = []` (liste vide).
Accès `L[0]` est le premier élément, `L[-1]` le dernier.
Longueur `len(L)` renvoie le nombre d'éléments.
Tranche `L[a:b]` renvoie les éléments d'indice `a` à `b-1`.
Les indices vont de `0` à `len(L) - 1`. Un accès hors limites provoque une erreur `IndexError`.

Modifier une liste

`L.append(x)` Ajoute `x` à la fin.
`L.insert(i, x)` Insère `x` à l'indice `i`.
`L.pop()` Retire et renvoie le dernier élément.
`L.pop(i)` Retire et renvoie l'élément d'indice `i`.
`L.remove(x)` Retire la première occurrence de `x`.
`L[i] = y` Remplace l'élément d'indice `i` par `y`.
Les listes sont **mutables** : on peut modifier leur contenu sans créer de nouvelle liste.

Construction par compréhension

```
[expression for x in sequence]
[expression for x in sequence if condition]
```

Exemples :
- `[i**2 for i in range(5)]` donne `[0, 1, 4, 9, 16]`;
- `[x for x in L if x > 0]` filtre les éléments positifs de `L`.

Pièges fréquents

- `L2 = L` ne crée pas de copie : les deux variables désignent la même liste. Pour copier, utiliser `L2 = L[:]` ou `L2 = list(L)`;
- `L = [[0] * 3] * 3` crée trois références vers la même sous-liste! Utiliser `[[0]*3 for _ in range(3)]`;
- Ne pas modifier une liste pendant qu'on la parcourt avec `for`.

② EXEMPLES

Programme 2 · Parcours par valeur vs par indice

```
fruits = ["pomme", "banane", "cerise"]

for fruit in fruits:          # parcours par valeur
    print(fruit)

for i in range(len(fruits)):  # parcours par indice
    print(f"{i} : {fruits[i]}")
```

Programme 3 · Construction par compréhension

```
pairs = [x for x in range(20) if x % 2 == 0]
# [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Programme 4 · Piège de l'alias

```
A = [1, 2, 3]
B = A          # B est un alias de A, pas une copie
B[0] = 99
print(A)       # [99, 2, 3]  A est aussi modifié !

C = A[:]       # C est une vraie copie
C[0] = 0
print(A)       # [99, 2, 3]  A n'est pas modifié
```

Programme 5 · Méthodes utiles

```
notes = [12, 8, 15, 9, 14, 11]
notes.sort()           # trie en place : [8, 9, 11, 12, 14, 15]
triees = sorted(notes) # renvoie une NOUVELLE liste triée
notes.reverse()        # inverse en place
print(notes.count(12)) # nombre d'occurrences de 12
print(12 in notes)     # True (test d'appartenance)
```

Programme 6 · Matrice (liste de listes)

```
matrice = [[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]
print(matrice[1][2])  # 6 (ligne 1, colonne 2)
```

③ EXERCICES

Exercice 1 *Manipulations de base* **(3 points)**

1. Écrire une fonction `moyenne(L)` qui renvoie la moyenne des éléments d'une liste de nombres.
2. Écrire une fonction `sans_doublons(L)` qui renvoie une nouvelle liste contenant les éléments de `L` sans répétition, en préservant l'ordre de première apparition.

3. Écrire une fonction `interleave(L1, L2)` qui entrecroise deux listes de même longueur. Par exemple, `interleave([1, 2, 3], [4, 5, 6])` renvoie `[1, 4, 2, 5, 3, 6]`.

Exercice 2 *Compréhension de listes* **(2 points)** Réécrire chaque boucle sous forme de liste en compréhension :

```
# a)
resultat = []
for i in range(1, 11):
    resultat.append(i ** 3)

# b)
resultat = []
for mot in ["chat", "chien", "lapin", "hamster"]:
    if len(mot) > 4:
        resultat.append(mot.upper())
```

Exercice 3 *Matrice* **(3 points)**

1. Écrire une fonction `creer_matrice(n, p, val)` qui crée une matrice $n \times p$ remplie de la valeur `val`.
2. Écrire une fonction `transposee(M)` qui renvoie la transposée d'une matrice.
3. Écrire une fonction `somme_matrices(A, B)` qui renvoie la somme de deux matrices de mêmes dimensions.

SOLUTIONS DES EXERCICES

Corrigé de l'exercice 1.

```python
def moyenne(L):
    return sum(L) / len(L)

def sans_doublons(L):
    resultat = []
    for x in L:
        if x not in resultat:
            resultat.append(x)
    return resultat

def interleave(L1, L2):
    resultat = []
    for i in range(len(L1)):
        resultat.append(L1[i])
        resultat.append(L2[i])
    return resultat
```

Vérification :

- `moyenne([10, 12, 14])` = 12,0 ;
- `sans_doublons([3, 1, 4, 1, 5, 3])` = `[3, 1, 4, 5]` ;
- `interleave([1, 2, 3], [4, 5, 6])` = `[1, 4, 2, 5, 3, 6]`.

Corrigé de l'exercice 2.

```python
# a)
resultat = [i ** 3 for i in range(1, 11)]
# [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]

# b)
resultat = [mot.upper() for mot in ["chat", "chien", "lapin", "hamster"]
            if len(mot) > 4]
# ["CHIEN", "LAPIN", "HAMSTER"]
```

Corrigé de l'exercice 3.

```python
def creer_matrice(n, p, val):
    return [[val] * p for _ in range(n)]

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

def somme_matrices(A, B):
    n = len(A)
```

```
    p = len(A[0])
    return [[A[i][j] + B[i][j] for j in range(p)] for i in range(n)]
```

Attention : dans `creer_matrice`, on utilise une compréhension et non `[[val]*p]*n`, qui créerait n références vers la même liste.

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