1 Définition

Un **tableau indexé** est une **séquence ordonnée d'éléments identifiés par leur indice** qui est leur position dans le tableau.

Rappel : en informatique, **on compte** les séquences **à partir de l'indice 0** (comme pour les tuples).

Dans beaucoup de langages, comme en **C** par exemple, lorsqu'on crée un tableau, un espace mémoire contigu est réservé pour contenir les données du tableau. Le tableau peut donc être vu comme une succession de cellules-mémoire de tailles identiques permettant de stocker un élément du tableau (les éléments sont du même type).

2 Tableau en Python

En Python, les tableaux indexés sont implémentés par des **listes** (type `list`).
(*Remarque* : une liste au sens général de l'informatique est un concept un peu différent : cf cours Term)

2.1 Création d'un tableau (liste-Python)

Les listes-Python s'écrivent **entre crochets**.

```
>>> liste = ['Henri', 'Louis', 'ELISABETH']
```

valeurs	'Henri'	'Louis'	'ELISABETH'
indice (à l'endroit)	0	1	2
indice (à l'envers)	-3	-2	-1

On peut créer un tableau d'éléments identiques simplement avec la syntaxe suivante :

```
>>> init = [0] * 5
>>> init
[0, 0, 0, 0, 0]
```

2.2 Accès à un élément

On accède aux éléments par indice comme pour les tuples :

```
>>> liste[0]
'Henri'
>>> liste[-1]
'ELISABETH'
>>> liste[-3]
'Henri'
```

2.3 Modification d'un élément

Les listes de Python sont des objets **mutables**, contrairement aux tuples.

```
>>> liste[2]  = 'Elisabeth'
>>> liste
['Henri', 'Louis', 'Elisabeth']
```

2.4 Méthode `append` (hors programme, mais très utile !)

Le langage Python permet d'ajouter aisément un élément à la fin de la liste avec la méthode `append`. (*Remarque* : une **méthode** est une fonction qui peut s'appliquer **spécifiquement** à un certain **type** d'objet ; on y fait appel avec la notation "pointée").

```
>>> liste.append('Cesar')
>>> liste
['Henri', 'Louis', 'Elisabeth', 'Cesar']
```

3 Création d'un tableau par compréhension

Une structure de création de tableau est assez courante : on désire ajouter une série d'éléments à un tableau sous certaines conditions.

Voir l'exemple suivant :

```
1 liste ← tableau_vide
2 n ← borne
3 pour i de 1 à n faire
4 |  si i est pair alors
5 |  |  ajouter i à liste
6 |  fin si
7 fin pour
```

Cette structure de création de tableau peut être effectuée en une ligne dite « **en compréhension** ».

En Python, voici l'implémentation correspondante pour l'exemple précédent :

```
>>> liste = [i for i in range(1, n+1) if i % 2 == 0]
```

La structure générale de création de tableau en compréhension est donc :
`liste = [expression for compteur in range(etendue) if conditions]`

Exemple : on peut utiliser cette syntaxe par exemple pour extraire une copie d'une partie d'un tableau :

```
>>> tab = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
 # extraction des éléments d'indice 2 à 5 (exclu)
>>> tranche = [tab[i] for i in range(2, 5)]
>>> tranche
['c', 'd', 'e']
```

4 Représentation de matrices : tableau de tableaux

4.1 Principe

Une matrice est un objet mathématique qui peut être présenté comme un tableau de m lignes et n colonnes.

$$\begin{pmatrix} 00 & 01 & \dots & 0n_{-1} \\ 10 & 11 & \dots & 1n_{-1} \\ \vdots & \vdots & \dots & \vdots \\ m_{-1}0 & m_{-1}1 & \dots & m_{-1}n_{-1} \end{pmatrix}$$

En programmation, une matrice s'implémente facilement comme un tableau de tableaux. Le tableau principal possède m éléments, qui sont chacun un tableau de n éléments.

Exemple 1 : création de la matrice suivante :

$$matrice \longmapsto \begin{pmatrix} 6 & \underline{3} & 1 & 1 & 4 \\ 4 & 5 & 2 & 2 & 4 \\ 6 & 1 & 1 & 2 & 5 \end{pmatrix}$$

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

# chaque élément du tableau est aussi un tableau !
>>> matrice[0]
[6, 3, 1, 1, 4]
>>> matrice[1]
[4, 5, 2, 2, 4]
>>> matrice[2]
[6, 1, 1, 2, 5]
```

Pour accéder à un élément particulier de la matrice, on accède d'abord à une ligne de la matrice puis à un élément de cette ligne.

Exemple :
Le chiffre 3 souligné de l'exemple précédent est accessible par `matrice[0][1]` :

```
>>> matrice[0][1]    # rappel : matrice[0] vaut [6, 3, 1, 1, 4]
3
```

Remarque : Pour plus de lisibilité, on peut écrire la construction du tableau de tableaux sur plusieurs lignes :

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

Exemple 2 : création d'une matrice de nombres aléatoires entre 1 et 6 « en compréhension » :

```
m, n = 3, 5       # 3 lignes de 5 colonnes
matrice = [[randint(1, 6) for i in range(n)] for j in range(m)]
```

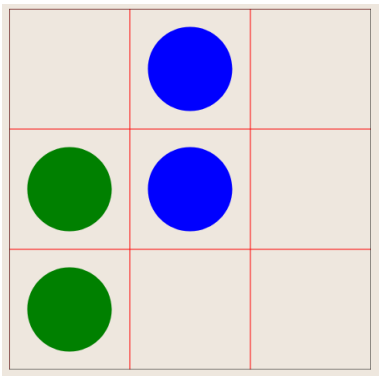
4.2 Application classique : grille de jeu

On désire représenter une grille du jeu Puissance 4.
Un choix de représentation assez classique peut être le suivant :

- chaque case est représentée par un nombre :
  Ex : case vide = 0, pion jaune = 1 , pion rouge = 2.
- chaque ligne est représentée par une liste de cases :
  Ex : `[0, 0, 1, 1, 0, 2, 1, 0]`
- la grille complète est une liste de lignes :
  Ex : `[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], ... , [0, 0, 1, 1, 0, 2, 1, 0], [0, 0, 1, 1, 0, 2, 1, 0], [0, 0, 1, 1, 0, 2, 1, 0]]`

Exercice :

1. Écrire la liste correspondant à la grille suivante du jeu de Morpion :



2. Comment accéder à la case du coin inférieur gauche ?
3. Écrire un code Python qui affiche ligne après ligne l'état de la grille.
   Par exemple, pour la grille donnée précédemment, l'affichage doit être :

```
0 1 0
2 1 0
2 0 0
```

5 Itérations sur les tableaux

5.1 Parcours par élément

Il est possible de parcourir les éléments d'un tableau un à un :

```
   Entrées : Tableau
1  pour chaque element de Tableau faire
2  |  écrire element
3  fin pour chaque
```

Implémentation en Python :

```
for element in tab:
    print(element)
```

5.2 Parcours par indice

Il est aussi possible d'effectuer un parcours séquentiel d'un tableau par indice :

```
   Entrées : Tableau
1  n ← taille(Tableau)
   /* les indices d'un tableau commencent à 0 et terminent à n-1 */
2  pour i de 0 à n-1 faire
3  |  écrire Tableau[i]
4  fin pour
```

Implémentation en Python :

```
for i in range(len(tab)):
    print(tab[i])
```