Il est souvent utile de pouvoir travailler avec plusieurs nombres ou caractères. Il existe, en programmation, des types de variables permettant cela : les **types construits**, incluant les tableaux.
Un tableau est un type constitué d'un **ensemble d'éléments qui peuvent être modifiés** (on dit que la liste est un objet *mutable*), dont chaque élément est **indexé**.

Partie I : Les listes

Une liste est un tableau à une dimension. La syntaxe pour définir une liste *L* est :

L = [élément, élément,..., élément, élément]

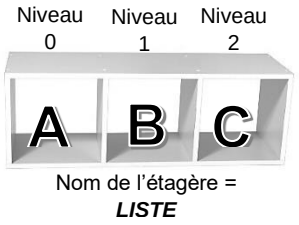
Chaque élément de la liste peut être retrouvé grâce à son indice : L[indice]

On peut se représenter une liste comme une étagère avec plusieurs cases qui sont indexées (numérotées).

Exemple :
Dans l'étagère, nommée LISTE, représentée ci-contre, au niveau 0, se trouve l'objet A, un niveau à côté, l'objet B et enfin, au niveau du bout, l'objet C.

Cette étagère est équivalente à la liste LISTE=[A,B,C].

Le terme d'indice 0 est la valeur A, celui d'indice 1 est la valeur B et celui d'indice 2 est la valeur C. On peut alors récupérer chaque élément de la liste en appelant son indice: LISTE[0] donnera la valeur A, LISTE[1] donnera la valeur B et LISTE[2] donnera la valeur C.



Nom de l'étagère = ***LISTE***

Si on considère la taille (le nombre d'élément) de la liste, on peut voir que les indices vont de *0* à *Taille-1*.

Il est aussi possible de lire la liste à partir de la fin ou de ne considérer qu'une seule partir de la liste, en donnant un domaine d'indice [début inclus : fin exclus].

Exemples :
- *LISTE[-1] correspond au dernier élément de la liste, soit C.*
- *LISTE[1:] isole les éléments d'indice 1 jusqu'à la fin de la liste, ce qui correspond à [B, C]*
- *LISTE[:2] isole les éléments du début jusqu'à l'indice 2 (exclus) de la liste, ce qui correspond à [A, B]*

Vérifions que nous avons compris les bases : Exercice flash !

Modification de liste (=CQFR !)

➢ Chaque élément de la liste est modifiable, en lui assignant une nouvelle valeur tout comme on le ferait pour une variable.
Exemple : Si on fait LISTE[0]=Z, la liste devient LISTE= [Z, B, C]

Attention : Écrire L[indice]=valeur ne marche QUE pour modifier un élément déjà existant d'une liste.

➢ Il est possible de répéter une liste, ou de concaténer (mettre ensemble) plusieurs listes.
Exemples :
- *LISTE*2 répètera 2 fois la liste et on aura alors [A, B, C, A, B, C].*
- *LISTE + [D, E, F] ajoutera les éléments D, E et F à la suite de Liste et on aura alors [A, B, C, D, E, F]*

Attention : Ecrire L+[elem] pour ajouter un élément ne marche QUE si L est déjà créée en tant que liste (même une liste vide du type L=[ ])

➢ Pour tester l'appartenance d'un élément à une liste, on peut utiliser : element in Liste
Cette commande renvoie True si l'élément est dans la liste et False dans le cas contraire

➢ Il existe aussi différentes fonctions et méthodes permettant de manipuler une liste. Une liste (non-exhaustive) de quelques-unes de ces commandes est donnée ci-dessous :

• len(L)	Renvoie la longueur de la liste L
• element in L	Indique si element appartient à la liste L (True/False)
• del(L[i])	Supprime l'élément d'indice *i* de la liste L
• L.append(element)	Ajoute element en dernière position de la liste L
• L.index(element)	Renvoie l'indice de la première occurrence de element dans la liste L
• L.remove(element)	Enlève, le cas échéant, la première occurrence de element dans la liste L

Lecture de listes **(=CQFR !)**

Il existe deux manières de lire une liste de manière itérative.

Lecture indice par indice

Dans la boucle POUR, la variable *i* prend successivement les différentes valeurs de l'**indice**.

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

Lecture élément par élément

Dans la boucle POUR, la variable *elem* prend successivement les différentes valeurs de l'**élément**.

```
for elem in L :
    print(elem)
```

Listes par compréhension **(=CQFR !)**

Il est possible de manipuler efficacement une liste en utilisant `for` et/ou `if` dans la définition de la liste : c'est la **compréhension de liste**. La syntaxe est la suivante :

`L = [fonction(i) for i in XX]` ou `L = [fonction(i) for i in XX if condition]`

où `XX` est soit un domaine de valeurs (`in range`), soit une liste précédemment définie.

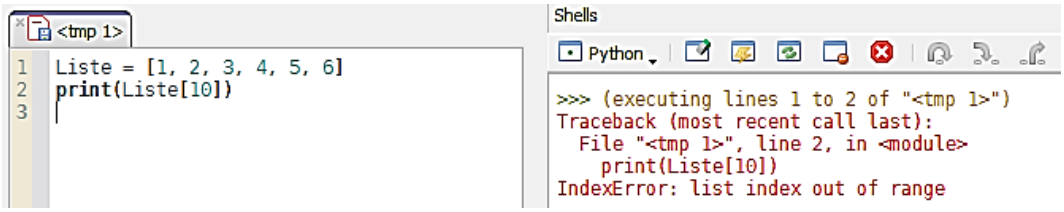
Exemples :

- *`L = [i for i in range(5)]` permet de créer la liste `L=[0, 1, 2, 3, 4]`*
- *`L = [i for i in range(10) if i%2 == 0]` permet de créer la liste des multiples de deux, `L=[0, 2, 4, 6, 8]`*
- *Étant donné une liste `A=[1, 2, 3, 4, 5]`, `L = [elem for elem in A if elem>2]` permet de créer la liste `L=[3, 4, 5]`*

Débogage

Lors de la lecture ou l'utilisation d'une liste, si l'indice demandé est en dehors de la taille de la liste, le compilateur affiche un message d'erreur.

Exemple : La liste a 6 éléments, et on demande l'élément de rang 10 (donc le 11ème) qui n'existe pas.



A. Pour débuter ensemble

Vérifions que nous avons compris les bases : Exercice flash !

I-A.1. Par répétition, créer une liste `Zeros` contenant 150 éléments ayant tous la valeur 0.

I-A.2. Que fait la fonction suivante :

```
1  def ListeEntiers(N) :
2      L = []
3      for i in range(N):
4          L = L + [i]
5      return L
```

I-A.3. Écrire une version modifiée la fonction `ListeEntiers` ci-dessus, que l'on appellera `ListeEntiers2`, permettant de créer la liste demandée par compréhension.

I-A.4. Écrire une fonction `AfficherParIndice` prenant en argument une liste et permettant d'afficher de manière itérative les valeurs de celle-ci, d'indice en indice, en indiquant à chaque fois l'indice de l'élément.

I-A.5. Écrire une fonction `AfficherParElem` prenant en argument une liste et permettant d'afficher de manière itérative les valeurs de celle-ci, d'élément en élément, en indiquant à chaque fois l'indice de l'élément.

B. Exercices pour s'entrainer

Pour chacun des exercices suivants, écrire le code qui réalise la tâche demandée puis le tester.

I-B.1. Écrire une fonction `Dec1` prenant en argument une liste d'entier qui, pour chaque élément de la liste, décrémente de 1 (soustrait 1 à) la valeur de l'élément et renvoi la liste modifiée.

I-B.2. Ecrire une fonction `Rotation` prenant en argument une liste et qui décale d'un rang tous les éléments de la liste, le dernier élément devenant le premier. La fonction renvoie une nouvelle liste.

I-B.3. Créer par compréhension une liste nommée `Multiples11` contenant les dix premiers multiples de 11, zéro exclus.

I-B.4. A l'aide de la fonction `ListeEntiers` (ou `ListeEntiers2`), créer une liste `ListeNbre` contenant les entiers de 0 à 100 (inclus), puis, par compréhension, créer une nouvelle liste `ListeCarres` utilisant la liste `ListeNbre` et contenant les entiers qui sont le carré d'un autre entier (ex : 4 est le carré de 2, il sera donc dans la liste).
Aide : Par exemple, si un entier N est le carré d'un autre entier, alors $\sqrt{N}$ est aussi un entier qui sera dans la liste

I-B.5. La fonction `Premier` permettant de déterminer si un entier est un nombre premier est donnée ci-contre.
En utilisant cette fonction, écrire une fonction `Facteurs_premiers(n)` qui renvoie la liste de tous les facteurs premiers d'un entier $N > 2$.

Exemple : pour N= 12936, la décomposition en facteurs premiers est $N = 2^3 \times 3 \times 7^2 \times 11$. On peut donc tester la fonction avec :

```
assert Facteurs_premiers(12936) == [2, 2, 2, 3, 7, 7, 11]
```

```
def Premier(n):
    if n>1:
        d=2
        if n%d != 0:
            d=3
            while n%d != 0:
                d += 2
        if d == n:
            return True
        else:
            return False
```

Pour aller un peu plus loin…

Un nombre palindrome est un nombre dont l'écriture en base 10 est symétrique. Il peut être alors lu de la même façon dans les deux sens (exemples : 11, 242, 12521…).
On veut déterminer si un nombre *a* donné est un palindrome ou non. Pour cela, on décompose le nombre a dans une liste L, où chaque case correspond à un chiffre de a.

Exemple : On représente a = 17842 par L = [1, 7, 8, 4, 2]

Ecrire une fonction `Palindrome(a)` qui décompose le nombre *a* dans une liste puis vérifie si *a* est un palindrome ou non.

Partie II : Les matrices

Une matrice est un tableau multiplement indexé, donc à plusieurs dimensions. En python, il se représente comme une liste de liste (de liste de liste de liste… tout dépend du nombre de dimensions). Nous allons ici nous concentrer sur les matrices à 2 dimensions.

La syntaxe pour définir une matrice *M* à 2 dimensions est alors :

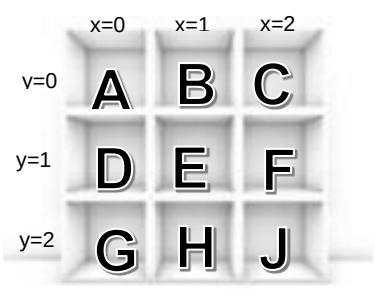
$$M = [L_0, L_1, \ldots, L_{n-1}, L_n]$$

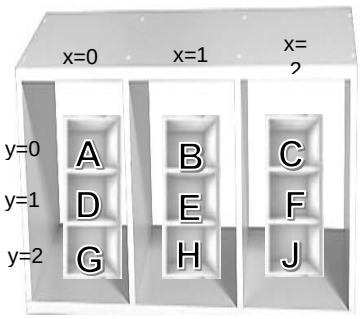
où L_n est une liste telle que : L_n = [élément, élément,..., élément, élément]

C'est donc une liste principale (M) contenant des listes secondaires (L).

Chaque élément de la matrice peut être retrouvé grâce à ses indices : `M[indiceM][indiceL]`

Exemple :
On peut se représenter une matrice en 2 dimensions comme l'étagère ci-dessous à gauche. Elle est équivalente à une étagère où chaque niveau contient lui-même une étagère. (la liste de listes), comme dans l'image de droite.





Cette étagère est équivalente à la matrice `MATRICE=[[A,D,G],[B,E,H],[C,F,J]]`.
On peut alors récupérer chaque élément de la liste de liste en appelant ses indices : `Matrice[0][0]` donnera la valeur A, `Matrice[1][2]` donnera la valeur H et `Matrice[2][2]` donnera la valeur J.

La lecture de matrice se déroule principalement comme la lecture de liste, mais il faut bien prendre en compte chaque dimension (et garder en tête que c'est une liste de liste !)

Exemples :

- *Matrice [-1] affichera le dernier élément de la liste, soit [C, F, J].*
- *Matrice[-1][-1] affichera le dernier élément de la matrice, soit J.*
- *Matrice[1:] isole les éléments d'indiceM 1 jusqu'à la fin de la liste, ce qui affichera [[B, E, H] , [C, F, J]]*

ATTENTION : les éléments A, B et C faisant partie de 3 listes différentes, on ne peut afficher [A, B, C] qu'en créant une nouvelle liste et en y insérant ces valeurs.

Il en est de même pour la répétition et la concaténation de matrice.

Exemples :

- *Matrice*2 répètera 2 fois la liste et on affichera alors [[A, D, G], [B, E, H], [C, F, J], [A, D, G], [B, E, H], [C, F, J]].*
- *Matrice + [X, Y, Z] ajoutera les éléments X, Y et Z à la suite de Matrice et on affichera alors [[A, D, G], [B, E, H], [C, F, J], X, Y, Z]*

Vérifions que nous avons compris les bases : Exercice flash !

A. Pour débuter ensemble

II-A.1. Écrire une fonction `AfficherMatrice` prenant en argument une matrice à 2 dimensions et permettant d'afficher de manière itérative les valeurs de celle-ci en indiquant à chaque fois les 2 indices de l'élément.

II-A.2. Par répétition, créer une liste `MatriceZeros` contenant 150×150 valeurs éléments ayant tous la valeur 0.

B. Exercices pour s'entrainer

Pour chacun des exercices suivants, écrire le code qui réalise la tâche demandée puis le tester.

II.B.1. On considère une matrice carrée (autant de liste que d'élément par liste) contenant des entiers. Écrire une fonction `SommeDiagonale` qui calcule la somme des éléments situés sur la diagonale principale de la matrice, en une seule et unique boucle. Il n'est pas nécessaire de vérifier que la matrice est bien carrée.

1	2	3
4	5	6
7	8	9

Exemple : Pour la matrice carrée de taille 3×3 ci-dessus, la fonction renvoie 1 + 5 + 9 = 15

II.B.2. La transposée d'une grille est la grille obtenue en intervertissant les lignes et les colonnes.
Écrire une fonction `TransposeGrille` prenant en argument une matrice carrée et renvoyant une matrice où les valeurs des colonnes deviennent des lignes (et vice-versa).

Note : pour créer une matrice *M* vide de taille *n*×*n*, utiliser `M = [[0]*n for i in range(n)]`

Ce qu'il faut retenir

❖ **Listes**

Une liste est un type constitué d'un **ensemble d'éléments qui peuvent être modifiés** (on dit que la liste est un objet *mutable*), dont chaque élément est **indexé**.

La syntaxe pour définir une liste *L* est : `L = [élément, élément,..., élément, élément]`

Chaque élément de la liste peut être retrouvé grâce à son indice : `L[indice]`

❖ **Manipulation de liste** *(Voir sujet)*

❖ **Lecture de liste** *(Voir sujet)*

❖ **Liste par compréhension** *(Voir sujet)*