# Manipulation de listes

## Déclaration d'une liste :

```
L = [1, 2.5, Twe, 47, 3, 1]
```

## Longueur d'une liste :

→ nb d'éléments dans la liste

```
n = len(L)        (ici n = 6)
```

## Indexation :

```
L = [1, 2.5, Twe, 47, 3, 1]
index     0    1    2    3   4   5
                                 ↳ n-1
"rétro-index"  -6   -5   -4   -3  -2  -1
```

```
L[1]   renvoie  2.5
L[4]      "     4
L[7]      "     list index out of range !
L[-1]     "     1
```

```
L[2 : 5]   renvoie  [Twe, 47, 4]
```

(2 : INCLU ; 5 : EXCLU)

```
L[3 : ]    renvoie  [47, 3, 1]
L[ : 3]       "     [1, 2.5, Twe]
L[-4 : ]      "     [Twe, 47, 3, 1]
L[0 : 6 : 2]  "     [1, Twe, 3]
```

pas de 2 (le pas peut être négatif)

## Modification d'un élément dans une liste

↳ L[0], L[2] = L[2], L[0]

L ⇒ [Twe, 2.5, 1, 47, 3, 1]

L = [1, 2.5, Twe, 47, 3, 1]

indice 0 1 2 3 4 5

L[3] = (5)

⇝ L est donc maintenant égale à :

k-1 k k+1

[1, [2.5, Twe, 5] 3, 1]

0 1 2 3 4 5=n-1

L[1 : 4] = [3.5, False, 8]

⇝ L est donc maintenant égale à :

[1, 3.5, False, 8, 3, 1]

Ajout / suppression d'un élément :

L = [1, 3.5, False, 8, 3, 1]

Ajout : L.append(3) : on rajoute l'élément 3 à la fin de la liste

⇝ L est de maintenant égale à

[1, 3.5, False, 8, 3, 1, 3]

Suppression : L.pop() renvoie le dernier élément de la liste

Si L = [1, 3.5, False, 8, 3, 1]

d = L.pop()

d ⇝ renvoie 1

L ⇝ [1, 3.5, False, 8, 3]

Concaténate de listes : L1 = [1, 2, 3]

L2 = [5, 6]

L = L1 + L2

L ⇝ renvoie [1, 2, 3, 5, 6]

On pourra aussi écrire :

L3 = 3 * L2 (ie qui revient à faire L2+L2+L2)

L3 renvoie [5, 6, 5, 6, 5, 6]

Liste vide : LV = []

Pour copier une liste : Il ne faut pas faire

L4 = L2 ... cela introduit des comportements non souhaités

Mais plutôt L4 = L2[:]

ou L4 = L2.copy()

Raisons des erreurs : une liste contient l'adresse de la zone mémoire où sont stockées les valeurs (pointeur) mais non pas les valeurs elles-mêmes.

Range : (0,5)

r1 = range(5)

r1 → renvoie [0, 1, 2, 3, 4]

r2 = range(2, 7) (INCL, EXCL)

r2 → renvoie [2, 3, 4, 5, 6]

r3 = range(2, 7, 2) → pas de 2

r3 → renvoie [2, 4, 6]

r4 = range(7, 2, -1) → pas négatif !

r4 → renvoie [7, 6, 5, 4, 3]

### 3.5 Boucle `while`

Ce type de boucle sera structurée de la manière suivante :

```
while expr_bool:
  bloc
```

Ajouter 1 à chaque elt de la liste tant que l'élément est positif.

```
L1 = [1, 2, 3]
k = 0
while k<8 and L1[k]>0:
    L1[k] = L1[k] + 1
    k += 1
```

Ne pas si on "inverse" l'ordre de la condition.

Les instructions contenues dans `bloc` seront exécutées tant que l'évaluation de `expr_bool` renvoie `True` ; dès qu'elle renvoie `False`, l'exécution se poursuit à la suite de `bloc`.

On pourra également utiliser les instructions `return ...` ou `break` si la sortie prématurée de la boucle est nécessaire.

### 3.6 Construction d'une liste par compréhension

Cette possibilité permet la construction d'une liste de manière très concise à l'aide d'une boucle `for` éventuellement suivie d'un test.

On donne par exemple la liste suivante construite par compréhension :

```
liste = [i for i in range (101) if i%2==0]
```

## 4 Tracé de courbes : `pyplot`

Les outils nécessaires au tracé de courbes sur `Python` sont disponibles dans la bibliothèque `matplotlib.pyplot`. Pour l'importer, on utilisera donc l'instruction `import matplotlib.pyplot as plt`.

La commande de base est `plt.plot(X,Y)` qui relie les points de coordonnées $(X[k], Y[k])$, $X$ et $Y$ étant deux tableaux de même taille (à une dimension). La répétition de cette instruction permettra de tracer plusieurs courbes sur le même graphique.

La commande `plt.show()` affiche le graphique à l'écran. Par défaut, la fenêtre d'affichage (les intervalles pour les valeurs des abscisses et des ordonnées) est déterminée automatiquement, de même que les unités et graduations sur les axes.

Pour ajuster les différents éléments du graphique, on pourra utiliser les commandes suivantes :

`plt.grid()` ajoute une grille en lignes pointillées à chaque graduation.
`plt.title('Titre')` définit le titre du graphique.
`plt.xlabel('Nom axe x')` définit l'étiquette de l'axe des abscisses (même syntaxe pour l'axe des ordonnées avec `plt.ylabel('Nom axe y')`).
`plt.xlim(a,b)` définit l'intervalle $[a, b]$ pour les valeurs de l'axe des abscisses (même syntaxe pour l'axe des ordonnées avec `plt.ylim(c,d)`).
`plt.axis('equal')` impose la même échelle sur l'axe des ordonnées et des abscisses.

Pour sauvegarder le graphique, on utilisera l'instruction `plt.savefig(nom_de_fichier)`. Le fichier devra prendre en compte l'extension `.pdf` (pour obtenir un fichier pdf) ou `.png` (pour obtenir une image au format png).

Il est également possible de rajouter des options sur chaque tracé. D'une manière générale, il faudra utiliser la syntaxe suivante : `plt.plot(X,Y,option1,option2,...)`. Les options les plus courantes sont les suivantes :

- `color='black'` permet d'obtenir un tracé en noir. On pourra aussi utiliser les couleurs suivantes : `'blue'`, `'green'`, `'red'`, `'cyan'`, `'magenta'`.