Listes I

Une liste est une façon de regrouper des éléments en un seul objet. Après avoir défini une liste, on peut récupérer un par un chaque élément de la liste, mais aussi en ajouter de nouveaux...

Cours 1 (Liste (1))**.**
Une ***liste*** est une suite d'éléments. Cela peut être une liste d'entiers, par exemple `[5,-7,12,99]`, ou bien une liste de chaînes de caractères, par exemple `["Mars","Avril","Mai"]` ou bien les objets peuvent être de différents types `[3.14,"pi",10e-3,"x",True]`.

- **Construction d'une liste.** Une liste se définit par des éléments entre crochets :
  — `liste1 = [5,4,3,2,1]` une liste de 5 entiers,
  — `liste2 = ["Vendredi","Samedi","Dimanche"]` une liste de 3 chaînes de caractères,
  — `liste3 = []` la liste vide (très utile pour la compléter plus tard !).
- **Accéder à un élément.** Pour obtenir un élément de la liste, il suffit d'écrire `liste[i]` où *i* est le rang de l'élément souhaité.
  Attention ! Le piège c'est que l'on commence à compter à partir du rang `0` !
  Par exemple après l'instruction `liste = ["A","B","C","D","E","F"]` alors
  — `liste[0]` renvoie `"A"`
  — `liste[1]` renvoie `"B"`
  — `liste[2]` renvoie `"C"`
  — `liste[3]` renvoie `"D"`
  — `liste[4]` renvoie `"E"`
  — `liste[5]` renvoie `"F"`

	"A"	"B"	"C"	"D"	"E"	"F"
rang :	0	1	2	3	4	5

- **Ajouter un élément.** Pour ajouter un élément à la fin de la liste, il suffit d'utiliser la commande `maliste.append(element)` (*to append* signifie « ajouter »). Par exemple si `premiers = [2,3,5,7]` alors `premiers.append(11)` rajoute 11 à la liste, si ensuite on exécute la commande `premiers.append(13)` alors maintenant la liste `premiers` vaut `[2,3,5,7,11,13]`.
- **Exemple de construction.** Voici comment construire la liste qui contient les premiers carrés :

```
liste_carres = []          # On part d'un liste vide
for i in range(10):
    liste_carres.append(i**2)    # On ajoute un carré
```

À la fin `liste_carres` vaut :

`[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]`

Cours 2 (Liste (2))**.**

- **Longueur d'une liste.** La longueur d'une liste est le nombre d'éléments qu'elle contient. La commande `len(liste)` renvoie la longueur (*length* en anglais). La liste `[5,4,3,2,1]` est de longueur 5, la liste `["Vendredi","Samedi","Dimanche"]` de longueur 3, la liste vide `[]` de longueur 0.
- **Parcourir une liste.** Voici la façon la plus simple de parcourir une liste (et ici d'afficher chaque élément) :

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

- **Parcourir une liste (bis).** Parfois on a besoin de connaître le rang des éléments. Voici une autre façon de faire (qui affiche ici le rang et l'élément).

```
n = len(liste)
for i in range(n):
    print(i,liste[i])
```

- Pour obtenir une liste à partir de `range()` il faut écrire :

```
list(range(n))
```

Activité 1 (Intérêts simples ou composés)**.**

Objectifs : créer deux listes afin de comparer deux types d'intérêts.

1. **Intérêts simples.** On dispose d'une somme S_0. Chaque année ce placement rapporte des intérêts en fonction de la somme initiale.

   Par exemple avec une somme initiale $S_0 = 1000$ et des intérêts simples de $p = 10\%$. Les intérêts sont de 100. Donc au bout d'un an, je dispose d'une somme de $S_1 = 1100$, au bout de deux ans $S_2 = 1200\ldots$

   Programme une fonction `interets_simples(S0,p,n)` qui renvoie les listes des sommes des n premières années. Par exemple `interets_simples(1000,10,3)` renvoie `[1000, 1100, 1200, 1300]`.
2. **Intérêts composés.** Une somme S_0 rapporte selon des intérêts composés. Cette fois les intérêts sont calculés chaque année sur la base de la somme de l'année précédente, c'est-à-dire selon la formule :

$$I_{n+1} = S_n \times \frac{p}{100}$$

   Programme une fonction `interets_composes(S0,p,n)` qui renvoie la liste des sommes des n premières années. Par exemple `interets_composes(1000,10,3)` renvoie `[1000, 1100, 1210, 1331]`.
3. J'ai le choix entre un placement à intérêts simples de 10% et un placement à intérêts composés de 7%. Quelle est la solution la plus avantageuse en fonction de la durée du placement ?

Cours 3 (Liste (3))**.**

- **Concaténer deux listes.** Si on a deux listes, on peut les fusionner par l'opérateur « + ». Par exemple avec `liste1 = [4,5,6]` et `liste2 = [7,8,9]`

```
liste1 + liste2   vaut   [4,5,6,7,8,9].
```

- **Ajouter un élément à la fin.** L'opérateur « + » fournit une autre méthode permettant d'ajouter un élément à une liste :

```
liste = liste + [element]
```

Par exemple `[1,2,3,4] + [5]` vaut `[1,2,3,4,5]`. Attention ! Il faut entourer l'élément à ajouter de crochets. C'est une méthode alternative à `liste.append(element)`.

- **Ajouter un élément au début.** Avec :

```
liste = [element] + liste
```

on ajoute l'élément en début de liste. Par exemple `[5] + [1,2,3,4]` vaut `[5,1,2,3,4]`.

- **Trancher des listes.** On peut extraire d'un seul coup toute une partie de la liste : `liste[a:b]` renvoie la sous-liste des éléments de rang a à $b-1$.

	"A"	"B"	"C"	"D"	"E"	"F"	"G"
rang :	0	1	2	3	4	5	6

Par exemple si `liste = ["A","B","C","D","E","F","G"]` alors
— `liste[1:4]` renvoie `["B","C","D"]`
— `liste[0:2]` renvoie `["A","B"]`
— `liste[4:7]` renvoie `["E","F","G"]`
Il faut encore une fois faire attention à ce que le rang d'une liste commence à 0 et que le tranchage `liste[a:b]` s'arrête au rang $b-1$.

Activité 2 (Manipulation de listes)**.**

Objectifs : programmer des petites routines qui manipulent des listes.

1. Programme une fonction `rotation(liste)` qui décale d'un rang tous les éléments d'une liste (le dernier élément devenant le premier). La fonction renvoie une nouvelle liste.
   Par exemple `rotation([1,2,3,4])` renvoie la liste `[4,1,2,3]`.
2. Programme une fonction `inverser(liste)` qui inverse l'ordre des éléments d'une liste.
   Par exemple `inverser([1,2,3,4])` renvoie la liste `[4,3,2,1]`.
3. Programme une fonction `supprimer_rang(liste,rang)` qui renvoie une liste formée de tous les éléments, sauf celui au rang donné.
   Par exemple `supprimer_rang([8,7,6,5,4],2)` renvoie la liste `[8,7,5,4]` (l'élément 6 qui était au rang 2 est supprimé).
4. Programme une fonction `supprimer_element(liste,element)` renvoyant une liste qui contient tous les éléments sauf ceux égaux à l'élément spécifié.
   Par exemple `supprimer_element([8,7,4,6,5,4],4)` renvoie la liste `[8,7,6,5]` (tous les éléments égaux à 4 ont été supprimés).

Cours 4 (Manipulation de listes)**.**
Tu peux maintenant utiliser les fonctions Python qui font certaines de ces opérations.

- **Inverser une liste.** Voici trois méthodes :
  — `maliste.reverse()` modifie la liste sur place (c'est-à-dire que `maliste` est maintenant renversée, la commande ne renvoie rien) ;
  — `list(reversed(maliste))` renvoie une nouvelle liste ;
  — `maliste[::-1]` renvoie une nouvelle liste.
- **Supprimer un élément.** La commande `liste.remove(element)` supprime la première occurrence trouvée (la liste est modifiée). Par exemple avec `liste = [2,5,3,8,5]` la commande

`liste.remove(5)` modifie la liste qui maintenant vaut `[2,3,8,5]` (le premier 5 a disparu).

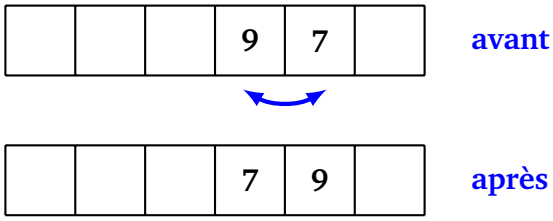
- **Supprimer un élément (bis).** La commande `del liste[i]` supprime l'élément de rang i (la liste est modifiée).

Activité 3 (Tri à bulles).

Objectifs : ordonner une liste du plus petit au plus grand élément.

Le tri à bulles est une façon simple d'ordonner une liste, ici ce sera du plus petit au plus grand élément. Le principe est le suivant :

- On parcourt la liste en partant du début. Dès que l'on rencontre deux éléments consécutifs dans le mauvais ordre, on les échange.
- À la fin du premier passage, le plus grand élément est à la fin et il ne bougera plus.
- On recommence du début (jusqu'à l'avant-dernier élément), cette fois les deux derniers éléments sont bien placés.
- On continue ainsi. Il y a en tout $n-1$ passages si la liste est de taille n.



Voici l'algorithme du tri à bulles :

Algorithme.

- — Entrée : une liste ℓ de n nombres
  — Sortie : la liste ordonnée du plus petit au plus grand
- Pour i allant de $n-1$ à 0 :
  Pour j allant de 0 à $i-1$:
  Si $\ell[j+1] < \ell[j]$ alors échanger $\ell[j]$ et $\ell[j+1]$.
- Renvoyer la liste ℓ.

Programme l'algorithme du tri à bulles en une fonction `trier(liste)` qui renvoie la liste ordonnée des éléments. Par exemple `trier([13,11,7,4,6,8,12,6])` renvoie la liste `[4,6,6,7,8,11,12,13]`.

Indications.

- Commence par définir `nouv_liste = list(liste)` et travaille uniquement avec cette nouvelle liste.
- Pour que l'indice i parcourt les indices à rebours de $n-1$ à 0, tu peux utiliser la commande :
  `for i in range(n-1,-1,-1):`
  En effet `range(a,b,-1)` correspond à la liste décroissante des entiers i vérifiant $a \geqslant i > b$ (comme d'habitude la borne de droite n'est pas incluse).

Cours 5 (Tri).
Tu peux maintenant utiliser la fonction `sorted()` de `Python` qui ordonne des listes.

python : sorted()

Usage : `sorted(liste)`
Entrée : une liste
Sortie : la liste ordonnée des éléments

Exemple : `sorted([13,11,7,4,6,8,12,6])` renvoie la liste `[4,6,6,7,8,11,12,13]`.

Attention ! Il existe aussi une méthode `liste.sort()` qui fonctionne un peu différemment. Cette commande ne renvoie rien, mais par contre la liste `liste` est maintenant ordonnée. On parle de modification *sur place*.

Activité 4 (Arithmétique)**.**

Objectifs : améliorer quelques fonctions de la fiche « Arithmétique – Boucle tant que – I ».

1. **Facteurs premiers.** Programme une fonction `facteurs_premiers(n)` qui renvoie la liste de tous les facteurs premiers d'un entier $n \geqslant 2$. Par exemple, pour $n = 12\,936$, dont la décomposition en facteurs premiers est $n = 2^3 \times 3 \times 7^2 \times 11$, la fonction renvoie `[2, 2, 2, 3, 7, 7, 11]`.
   Indications. Consulte la fiche « Arithmétique – Boucle tant que – I ». Le corps de l'algorithme est le suivant :
   Tant que $d \leqslant n$:
   Si d est un diviseur de n, alors :
   ajouter d à la liste,
   n devient n/d.
   Sinon incrémenter d de 1.
2. **Liste de nombres premiers.** Écris une fonction `liste_premiers(n)` qui renvoie la liste de tous les nombres premiers strictement inférieurs à n. Par exemple `liste_premiers(100)` renvoie la liste :
   `[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]`
   Pour cela, tu vas programmer un algorithme qui est une version simple du crible d'Ératosthène :

   Algorithme.
   - — Entrée : un entier $n \geqslant 2$.
     — Sortie : la liste des nombres premiers $< n$.
   - Initialiser `liste` qui contient tous les entiers de 2 à $n-1$.
   - Pour d allant de 2 à $n-1$:
     Pour k parcourant `liste` :
     Si d divise k et $d \neq k$, alors retirer l'élément k de `liste`
   - Renvoyer `liste`.

   Indications.
   - Pars de `liste = list(range(2,n))`.
   - Utilise `liste.remove(k)`.

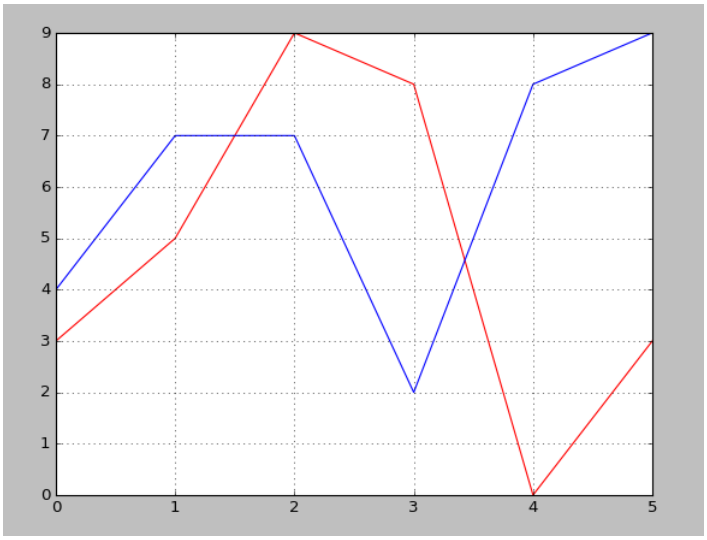
   Explications. Voyons comment fonctionne l'algorithme avec $n = 30$.

- Au départ la liste est

  $$[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29]$$

- On part avec $d = 2$, on élimine tous les nombres divisibles par 2, sauf si c'est le nombre 2 : on élimine donc 4, 6, 8,..., la liste est maintenant : $[2,3,5,7,9,11,13,15,17,19,21,23,25,27,29]$.
- On continue avec $d = 3$, on élimine les multiples de 3 (sauf 3), après ces opérations la liste est : $[2,3,5,7,11,13,17,19,23,25,29]$.
- Avec $d = 4$, on élimine les multiples de 4 (mais il n'y en a plus).
- Avec $d = 5$ on élimine les multiples de 5 (ici on élimine juste 25), la liste devient $[2,3,5,7,11,13,17,19,23,29]$.
- On continue (ici il ne se passe plus rien).
- À la fin, la liste vaut $[2,3,5,7,11,13,17,19,23,29]$.

Cours 6 (Visualiser une liste)**.**
Avec le module `matplotlib` il est très facile de visualiser les éléments d'une liste de nombres.




```
import matplotlib.pyplot as plt

liste1 = [3,5,9,8,0,3]
liste2 = [4,7,7,2,8,9]

plt.plot(liste1,color="red")
plt.plot(liste2,color="blue")
plt.grid()
plt.show()
```

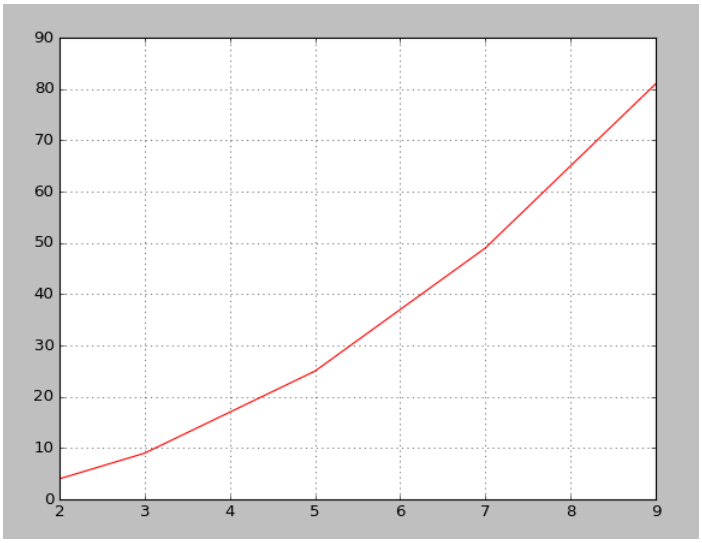
Explications.

- Le module s'appelle `matplotlib.pyplot` et on lui donne le nouveau nom plus simple de `plt`.
- Attention ! Le module `matplotlib` n'est pas toujours installé par défaut avec `Python`.
- `plt.plot(liste)` trace les points d'une liste (sous la forme (i, ℓ_i)) qui sont reliés par des segments.
- `plt.grid()` trace une grille.
- `plt.show()` affiche tout.

Pour afficher des points (x_i, y_i) il faut fournir la listes des abscisses puis la listes des ordonnées :

```
plt.plot(liste_x,liste_y,color="red")
```

Voici un exemple de graphe obtenu en affichant des points de coordonnées du type (x, y) avec $y = x^2$.

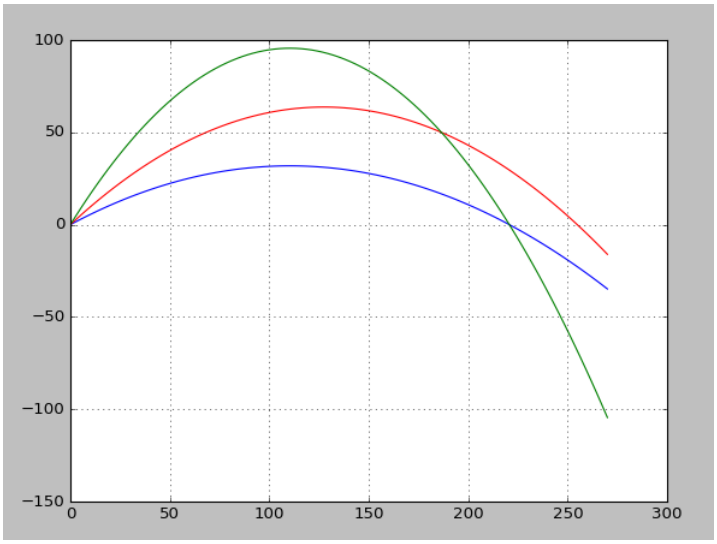



```
import matplotlib.pyplot as plt

liste_x = [2, 3, 5, 7, 9]
liste_y = [4, 9,25,49,81]
plt.plot(liste_x,liste_y,color="red")
plt.grid()
plt.show()
```

Activité 5 (Tir balistique)**.**

Objectifs : visualiser le tir d'un boulet de canon.

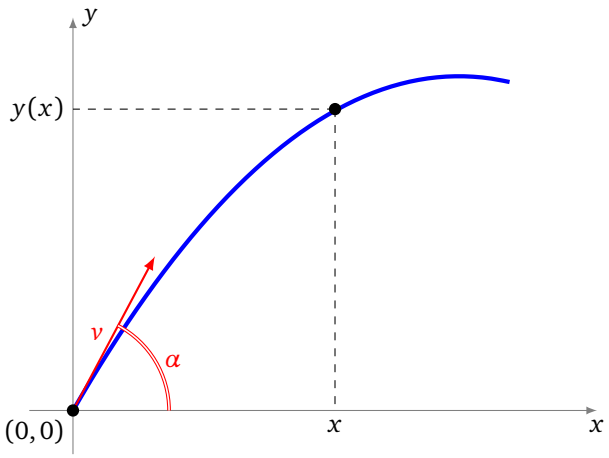


On tire un boulet de canon depuis l'origine $(0, 0)$. L'équation de la trajectoire est donnée par la formule :

$$y(x) = -\frac{1}{2} g \frac{1}{v^2 \cos^2(\alpha)} x^2 + \tan(\alpha) x$$

où

- α est l'angle du tir,
- v est la vitesse initiale,
- g est la constante de gravitation : on prendra $g = 9.81$.



1. Programme une fonction `tir_parabolique(x,v,alpha)` qui renvoie la valeur $y(x)$ donnée par la formule.

   Indication. Fais attention aux unités pour l'angle α. Si par exemple tu choisis que l'unité pour l'angle soit les degrés, alors pour appliquer la formule avec `Python` il faut d'abord convertir les angles en radians :

   $$\alpha_{\text{radians}} = \frac{2\pi}{360}\alpha_{\text{degrés}}$$

2. Programme une fonction `liste_trajectoire(xmax,n,v,alpha)` qui calcule la liste des ordonnées y des $n+1$ points de la trajectoire dont les abscisses sont régulièrement espacées entre 0 et $x_{\max}$.

   Méthode. Pour i allant de 0 à n :
   - calcule $x_i = i \cdot \frac{x_{\max}}{n}$,
   - calcule $y_i = y(x_i)$ par la formule de la trajectoire,
   - ajoute y_i à la liste.

3. Pour $v = 50$, $x_{\max} = 270$ et $n = 100$, affiche différentes trajectoires selon les valeurs de l'angle α. Quel angle α permet d'atteindre le point $(x, 0)$ au niveau du sol le plus éloigné possible du point de tir ?