# Les listes en Python

*N'hésitez pas à vous référer chaque fois que nécessaire au « mémento » dédié aux listes en Python…*

## Parcours 0. Manipulation de listes (fenêtre interactive = Python Shell / Console Python)

Lorsqu'une liste L a été construite, ou modifiée, il est possible de vérifier que l'opération s'est bien passée en demandant à l'interpréteur (donc dans la fenêtre interactive) la valeur de la liste **L** :

```
>>> L=[1,3,5,7]
>>> L=L+L
>>> L
[1, 3, 5, 7, 1, 3, 5, 7]
```

Cette façon de faire est naturellement valide pour toute variable Python, quel que soit son type.

**Exercice 0.** Définir la liste **L = [17, 38, 10, 25, 72]**, puis effectuez les actions listées ci-dessous. Après chaque action, vérifiez que celle-ci s'est correctement effectuée.

- ajouter l'élément **12** en fin de la liste **L** ;
- mettre dans une variable **val** la valeur de l'élément d'indice **4** ;
- mettre dans une variable **ind** l'indice de l'élément **25** ;
- mettre dans une variable **long** la longueur de la liste **L** ;
- supprimer de la liste **L** l'élément **38** ;
- ajouter les éléments **[8, 14, 29]** en fin de la liste **L** ;
- supprimer de la liste **L** l'élément d'indice **2** ;
- remplacer, dans la liste **L**, l'élément d'indice **0** par l'élément **55** ;
- mettre dans une liste **L2** la sous-liste de **L** allant du 2e au 5e élément ;
- mettre dans une liste **L3** la sous-liste de **L** allant du début de la liste au 3e élément ;
- mettre dans une liste **L4** la sous-liste de **L** allant du 3e élément à la fin de la liste ;
- mettre dans une liste **L5** la concaténation des liste **L2**, **L3** et **L4** ;
- recopier la liste **L** dans une liste **LL**, ajouter l'élément **99** en fin de la liste **LL** ; quel est maintenant la valeur des listes **L** et **LL** ? Avez-vous réalisé une recopie ou une identification ?
- définir en compréhension la liste **L6**, composée des cubes des entiers de **5** à **11** ;
- définir en compréhension la liste **L7**, composée des images des entiers de **1** à **8** par la fonction $\mathbf{x \to 2x^2 - 3x + 5}$ ;
- définir en compréhension la liste **L8**, composée des carrés des entiers pairs compris entre **9** et **21** ;
- définir en compréhension la liste **L9**, composée des multiples de **3** compris entre **111** et **130** ;

## Parcours 1. Écriture de fonctions simples

*Nous allons maintenant utiliser les deux fenêtres de l'environnement Python. Dans la fenêtre « programme », nous écrirons les fonctions demandées (à la suite les unes des autres, en pensant à sauvegarder régulièrement notre travail !) et nous les testerons dans la fenêtre interactive. Avant de tester ces fonctions, il sera naturellement nécessaire « d'exécuter » le programme, afin que l'interpréteur Python les (re)connaisse… En cas d'oubli, un message d'erreur de la forme* « **NameError: name 'leNomDeLaFonction' is not defined** » *devrait s'afficher.*

*Une fonction Python peut retourner un nombre quelconque de résultats (voire aucun). Il est en effet possible d'utiliser l'instruction* return *avec plusieurs résultats à retourner, séparés par une virgule :*

```
>>> def fonctionStupide(n):
        return n-1,2*n

>>> a,b=fonctionStupide(43)
>>> a
42
>>> b
86
```

*On sera particulièrement attentif à la distinction entre identification et recopie lors des affectations de listes (voir mémento).*

**Exercice 1.** Écrire une fonction **echanger** qui prend comme arguments une liste et deux indices et qui échange les deux éléments correspondants. Par exemple, **echanger ([ 1, 2, 5, 8 ], 0, 3)** devra retourner **[ 8, 2, 5, 1 ]**. On supposera que les deux indices donnés sont valides (dans le cas contraire, l'appel de cette fonction engendrera une erreur).

**Exercice 2.** En utilisant la fonction **echanger**, écrire une fonction **inverser** qui prend comme argument une liste et renvoie la liste inversée (le 1er élément est devenu le dernier, le 2e l'avant-dernier, etc.). Attention, il est possible que la liste à inverser soit vide…

**Exercice 3.** Écrire une fonction **moyenne** qui prend comme argument une liste non vide de nombres et renvoie la moyenne des éléments de cette liste.

**Exercice 4.** Écrire une fonction **maximum** qui prend comme argument une liste non vide de nombres et renvoie la valeur maximale de cette liste.

**Exercice 5.** Écrire une fonction **indiceMinimum** qui prend comme argument une liste non vide de nombres et renvoie l'indice de la valeur minimale de cette liste.

**Exercice 6.** Écrire une fonction **triListe** qui prend comme argument une liste de nombres et renvoie une liste triée par ordre croissant contenant les mêmes éléments. Cette fonction procèdera de la façon suivante : (1) recopie de la liste argument dans une liste de travail **LT** et initialisation de la liste résultat **LR** à vide, (2) tant que la liste **LT** n'est pas vide, rechercher l'indice de la valeur minimale, rajouter l'élément correspondant en fin de **LR**, puis supprimer cet élément de **LT**. On pourra utiliser la fonction **indiceMinimum** précédemment définie. Attention, là encore il est possible que la liste à trier soit vide…

**Exercice 7.** Écrire une fonction **ecartType** qui prend comme argument une série statistique sous forme d'une liste non vide de nombres et renvoie l'écart-type de cette série (par exemple, **ecartType([1, 3, 5, 7, 9])** doit renvoyer **2.8284271247461903**). On pourra utiliser la fonction **moyenne** précédemment définie.

**Exercice 8.** Écrire une fonction **mediane** qui prend comme argument une liste non vide de nombres et renvoie la valeur médiane de cette liste. On pourra utiliser la fonction **triListe** précédemment définie.

**Exercice 9.** Écrire une fonction **statistiques** qui prend comme argument une liste non vide de nombres et renvoie les paramètres statistiques (moyenne, écart-type et médiane) de cette liste.

**Exercice 10.** Écrire une fonction **pairImpair** qui prend comme argument une liste de nombres entiers et renvoie deux listes : la première liste contient les nombres pairs, la seconde les nombres impairs. Il est possible que la liste argument soit vide.

**Exercice 11.** Écrire une fonction **moinsDeSix** qui prend comme argument une liste de mots (chaînes de caractères) et renvoie deux listes : la première contient les mots de moins de six lettres, la seconde les mots de six lettres ou plus (si **m** est une chaîne de caractères, **len(m)** renvoie le nombre de caractères de la chaîne **m**). Là encore, il est possible que la liste argument soit vide.

**Exercice 12.** Écrire une fonction **palindrome** qui prend comme argument une liste et qui vérifie si cette liste est un palindrome (par exemple, les listes **[ 1, 3, 5, 3, 1 ]** et **[ 'r', 'a', 'd', 'a', 'r' ]** sont des palindromes alors que les listes **[ 1, 2 ]** et **[ 'a', 'b', 'b' ]** n'en sont pas). La valeur renvoyée sera donc **True** ou **False**. Par convention, on supposera qu'une liste vide est un palindrome.
Vérifiez que la fonction **palindrome** peut être utilisée avec des listes d'entiers ou des listes de caractères (en fait, quel que soit le type des éléments de la liste). *Note : il est possible de transformer une chaîne de caractères en liste de caractères, en écrivant* **L = list('python')** *; la liste* **L** *vaut alors* **[ 'p', 'y', 't', 'h', 'o', 'n' ]**.

**Exercice 13.** Écrire une fonction **diviseurs** qui prend comme argument un nombre entier et renvoie la liste de ses diviseurs.

**Exercice 14.** Écrire une fonction **estPremier** qui prend comme argument un nombre entier et détermine si cet entier est premier ou non (la valeur renvoyée sera donc **True** ou **False**).

**Exercice 15.** Écrire une fonction **listeFacteursPremiers** qui prend comme argument un nombre entier et renvoie la liste de ses facteurs premiers (on pourra utiliser les fonctions **diviseurs** et / ou **estPremier**).

**Exercice 16.** Écrire une fonction **listeChiffres** qui prend comme argument un entier et renvoie la liste de ses chiffres. Par exemple, **listeChiffres(9384001)** devra renvoyer la liste **[ 9, 3, 8, 4, 0, 0, 1 ]**.

## Parcours 2. Pour aller plus loin (listes simples)


**Exercice 17.** Un code barre à 13 chiffres est constitué de trois parties. En lisant de gauche à droite, nous avons :

- les deux premiers chiffres constituent le « drapeau », qui correspond au pays d'origine (France = 3 par exemple) ;
- les dix chiffres suivants constituent la « zone utile de codage », qui permet de coder effectivement l'article ;
- le dernier chiffre est un caractère de contrôle, appelé la « clé ».

Cette clé est calculée de la manière suivante : *S étant la somme des termes de rang impair du code barre sans la clé, S' la somme des termes de rang pair, N prend la valeur S + 3S'. Soit alors R le reste de la division euclidienne de N par 10. Si R est non nul, alors la clé est égale à 10 – R, sinon la clé est 0.*

Écrire la fonction **cleCodeBarre** qui prend comme argument une liste constituée des douze premiers chiffres d'un code barre et renvoie la valeur de la clé.

Écrire la fonction **estUnCodeBarre** qui prend comme argument une liste constituée de treize chiffres et vérifie si elle correspond à un code barre valide (la valeur renvoyée sera donc **True** ou **False**).

**Exercice 18.** Écrire une fonction **insereElement** qui prend comme arguments une liste triée et un élément et renvoie la liste dans laquelle l'élément a été inséré en bonne place.
Vérifiez que cette fonction peut être utilisée sur différents types de listes.

**Exercice 19.** Écrire une fonction **triListeParInsertion** qui prend comme argument une liste et renvoie une liste triée par ordre croissant contenant les mêmes éléments. Cette fonction procèdera de la façon suivante : après avoir initialisé la liste résultat à « liste vide », on insère un à un les éléments de la liste argument en bonne place (on utilisera naturellement la fonction **insereElement** précédemment définie).

**Exercice 20.** Écrire une fonction **triListeParEchange** qui prend comme argument une liste et la trie par ordre croissant de la façon suivante : tant que la liste contient des éléments « mal rangés » (un « grand » qui précède un « petit »), échanger ces deux éléments. On pourra utiliser la fonction échanger précédemment définie.
Vérifiez que cette fonction peut être utilisée indifféremment sur des listes d'entiers, de nombres décimaux, de caractères ou de chaînes de caractères, voire des listes de listes (en fait, il suffit que les opérateurs de comparaison soient définis sur le type des éléments).

**Exercice 21.** Écrire une fonction **fusionListes** qui prend comme arguments deux listes triées, construit la liste triée obtenue par fusion de ces deux listes et la renvoie. Par exemple, à partir des listes **[1, 3, 6, 8, 9, 12]** et **[1, 5, 9, 10, 15, 16]**, cette fonction renverra la liste **[1, 1, 3, 5, 6, 8, 9, 9, 10, 12, 15, 16]**.
Cette fonction devra mettre en œuvre un « vrai » algorithme de fusion (qui consiste à parcourir les deux listes en rangeant dans la liste résultat le bon élément à chaque étape), sans tricher ! En d'autres termes, on ne se contentera pas de concaténer les deux listes, puis de les trier à l'aide d'une fonction déjà écrite…

Cette fonction **fusionListes** va nous permettre d'écrire une fonction (récursive) **triFusion**, qui prend comme argument une liste et la trie par ordre croissant de la façon suivante : si la longueur de la liste est 0 ou 1, la liste est déjà triée ; dans le cas contraire, on coupe la liste en deux listes de « même longueur » (à un près selon la parité de sa longueur), que l'on trie par un appel récursif à **triFusion** ; il ne reste plus qu'à fusionner les deux moitiés triées pour obtenir le résultat recherché ☺…

**Exercice 22.** Écrire une fonction **chercheParDichotomie** qui prend comme arguments une liste d'entiers triée et un entier, et renvoie l'indice de cet entier dans la liste, s'il est présent, ou la valeur **–1** dans le cas contraire. Cette recherche sera effectuée par dichotomie.

## Parcours 3. Listes de listes

**Exercice 23.** Écrire une fonction **moyenneCoeff** qui prend comme argument une liste non vide de listes de la forme **[ [note1, coef1], [note2, coef2], … ]** et renvoie la moyenne de cette liste de notes avec coefficients. Par exemple, **moyenneCoeff ( [ [12.50,2], [14.00,1], [17.50,3] ] )** devra renvoyer **15.25**.

**Exercice 24.** Écrire une fonction **calculeImages** qui prend comme argument une fonction **f** et deux entiers **a** et **b** (avec **a ≤ b**) et qui renvoie la liste composée des listes de la forme **[ x, f(x) ]**, pour **x** variant de **a** et **b** (inclus).
Par exemple, si **truc** est la fonction définie par **truc(x) = $2x^2 – 3x + 5$**, alors **calculeImages ( truc, 2, 6 )** devra renvoyer la liste **[ [2, 7], [3, 14], [4, 25], [5, 40], [6, 59] ]**.

**Exercice 25.** Écrire une fonction **maximumListeDeListes** qui prend comme argument une liste de listes d'éléments (contenant au moins un élément quelque part…) et renvoie le plus grand élément présent.
Par exemple, **maximumListeDeListes ( [ [1, 8, 2, 7, 15], [7, 21, 14], [21, 4, 11], [8, 11, 5 ] )** devra renvoyer **21**, alors que **maximumListeDeListes ( [ [ ], [7, 3], [ ] ] )** devra renvoyer **7**.

*Il est possible de représenter une matrice par une « liste de listes », par exemple ligne par ligne. Ainsi, la matrice M ci-contre sera représentée en Python par la liste L suivante :*

**[ [1, 2, 4], [5, 3, 1], [8, 2, 6] ]**

*L'élément* **L[2][0]** *est alors le 1er élément de la 3e liste, soit* **M[3,1] = 8**.

$$\begin{bmatrix} 1 & 2 & 4 \\ 5 & 3 & 1 \\ 8 & 2 & 6 \end{bmatrix}$$

*Pour l'ensemble des exercices suivants, on supposera que les matrices sont représentées ligne par ligne, donc sous forme d'une liste de listes, chaque sous-liste représentant une ligne de la matrice (voir exemple ci-dessus).*

**Exercice 26.** Écrire une fonction **estMatriceCarree** qui vérifie si une liste de listes d'entiers représente ou pas une matrice carrée (la valeur renvoyée sera donc **True** ou **False**).

**Exercice 27.** Écrire une fonction **matricesSommeCompatibles** qui prend comme arguments deux listes de listes représentant deux matrices et renvoie **True** si leurs dimensions sont identiques, **False** sinon.

**Exercice 28.** Écrire une fonction **matricesProduitCompatibles** qui prend comme arguments deux listes de listes représentant deux matrices et renvoie **True** si le nombre de colonnes de la 1re matrice et le nombre de lignes de la 2e matrice sont identiques, **False** sinon.

**Exercice 29.** Écrire une fonction **produitMatriceParScalaire** qui prend comme arguments une liste de listes représentant une matrice et un entier, et renvoie une liste de listes représentant le produit de cette matrice par cet entier.

**Exercice 30.** Écrire une fonction **produitMatriceParVecteur** qui prend comme arguments une liste de listes représentant une matrice et une liste représentant un vecteur colonne, et renvoie une liste de listes représentant le produit de cette matrice par ce vecteur (on supposera que les dimensions de la matrice et du vecteur sont compatibles).

**Exercice 31.** Écrire une fonction **produitVecteurParMatrice** qui prend comme arguments une liste représentant un vecteur ligne et une liste de listes représentant une matrice, et renvoie une liste de listes représentant le produit de ce vecteur par cette matrice (on supposera que les dimensions du vecteur et de la matrice sont compatibles).

**Exercice 32.** Écrire une fonction **sommeMatrices** qui prend comme arguments deux listes de listes représentant deux matrices de même dimension, et renvoie une liste de listes représentant la somme de ces deux matrices.

**Exercice 33.** Écrire une fonction **produitMatrices** qui prend comme arguments deux listes de listes représentant deux matrices de dimensions compatibles, et renvoie une liste de listes représentant le produit de ces deux matrices.