Travaux pratiques d'introduction 5

# Listes/Tableaux – 1

Informatique tronc commun MPSI et PCSI

## I Introduction

### I.1 Stocker plusieurs valeurs sous une seule variable

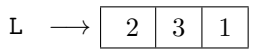
Des séances précédentes nous avons retenus que nous pouvions mémoriser une valeur – un nombre entier, un nombre réel, une chaine de caractère, etc – en l'associant à une variable.

Est-ce toujours concevable si nous devons stocker un très grand nombre de données comme par exemple celles récoltées à partir d'un capteur utilisé en physique expérimentale ? À l'évidence, non.

La *liste*, ou *tableau*, est une structure de données *ordonnée* et *modifiable* ou *mutable* qui répond à cette question en permettant d'associer à une seule variable un grand nombre de données.

En `Python`, on construit un tableau/liste en énumérant ses valeurs entre crochets séparées par des virgules.
Par exemple : `L1 = [2, 3, 1]` ou encore `L2 = ['Nom', 20.2, 2002, [2, 3, 1], True]`

On peut représenter l'association entre la variable et les valeurs d'un tableau comme ceci :




**À retenir : Syntaxe d'une liste/tableau**

Un *tableau*, dans le langage `Python`, permet de stocker plusieurs valeurs dans une seule variable et d'y accéder facilement.
Il est construit en énumérant ses valeurs entre crochets séparées par des virgules.
Par exemple : `L1 = [2, 3, 1]` ou encore `L2 = ['Nom', 20.2, 2002, [2, 3, 1], True]`

La *liste*, ou *tableau*, est une structure de données *ordonnée* et *modifiable* ou *mutable* qui associe à une seule variable un grand nombre de données.

### I.2 Objectifs

Dans ce TP nous découvrons la notion de tableau (ou de liste) à travers les thématiques suivantes :

1. Accéder à un ou plusieurs éléments d'un tableau.
2. Modifier les valeurs d'un tableau donné. Conséquence pratique sur la copie d'un tableau.
3. Parcours d'un tableau avec une boucle `for`.

## I.3 La liste, série finie et ordonnée de valeurs

Chaque position dans la liste possède un indice
Pour la liste `t = [-10, 2, 23, 8, 5, 4, 1, 6, 12, 7]`

| Valeurs → | -10 | 2 | 23 | 8 | 5 | 4 | 1 | 6 | 12 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| Indice → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**À retenir : Une liste/tableau est une séquence**

Ces valeurs forment une séquence, série *ordonnée* et *mutable* où chacune est associée à un indice de position `i` allant de `0` à `n - 1`, pour une séquence de longueur `n`.
Ces indices permettent d'accéder aux valeurs contenues dans ce tableau pour les consulter, les modifier, les extraire. On peut utiliser une boucle pour examiner tour à tour les différentes valeurs contenues dans un tableau.

## I.4 Comment accéder à une valeur du tableau/liste ?

| Valeurs → | -10 | 2 | 23 | 8 | 5 | 4 | 1 | 6 | 12 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| Indice → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

```
>>> t[0]
-10
>>> t[1]
2
```

**À retenir : Syntaxe**

`t[i]` où `i` est un indice entre 0 et `n - 1`
`n` est la longueur de la liste donnée par `len(t)`

## I.5 Comment accéder à toutes les valeurs du tableau/liste ?

| Valeurs → | -10 | 2 | 23 | 8 | 5 | 4 | 1 | 6 | 12 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| Indice → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Par exemple, comment faire la somme des valeurs du tableau ?
L'idée est de répéter la lecture des valeurs de la liste en modifiant les indices, d'où l'intérêt des boucles.

```
n = len(t)
somme = 0
for i in range(n):
    somme += t[i]
```

**Exercice 1 - Somme des nombres d'une liste**
Construire une fonction `somme(L)` qui prend en argument une liste `L` d'entiers ou de flottants et renvoie la somme de ses éléments.
Appliquer cette fonction à la liste `t = [-10, 2, 23, 8, 5, 4, 1, 6, 12, 7]`.

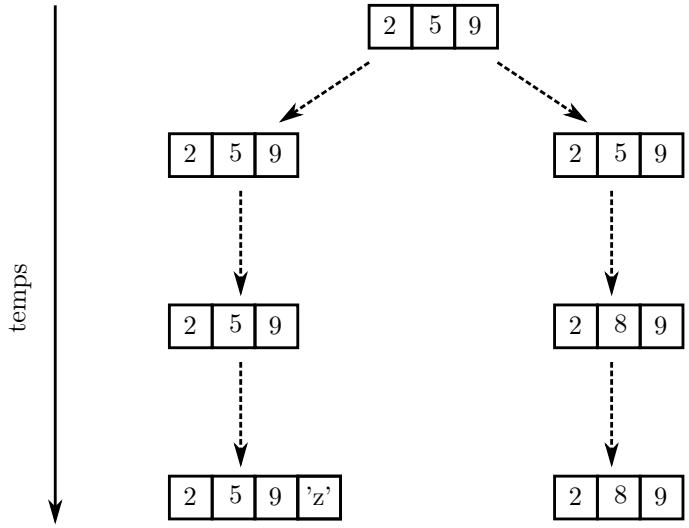
```
>>> somme(t)
58
```

**Exercice 2 - Moyenne des nombres d'une liste**
Construire une fonction `moyenne(L)` qui prend en argument une liste `L` d'entiers ou de flottants et renvoie la moyenne arithmétique de ses éléments.
Appliquer cette fonction à la liste `t = [-10, 2, 23, 8, 5, 4, 1, 6, 12, 7]`.

```
>>> moyenne(t)
5.8
```

## I.6 Copier une liste ? Passe si simple.

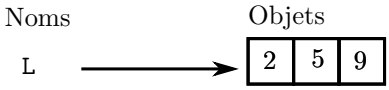
Considérons une liste `L = [2, 5, 9]` que nous souhaitons copier pour, ensuite, faire un usage de la copie qui n'affecte pas l'originale et réciproquement [1].



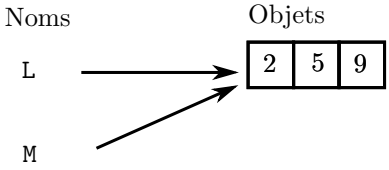

### I.6.a Comment réaliser des copies indépendantes de listes en `Python` ?

A partir de ce que nous connaissons pour les affectations des variables, écrivons le code suivant.

```
L = [2, 5, 9]
```




```
L = [2, 5, 9]
M = L
```

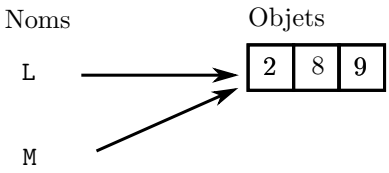



Le nom `M` pointe vers le même objet, vers la même référence en mémoire où est stockée la liste. La référence pointe donc vers un espace physique où se trouve l'objet liste. `M = L` n'a fait que donner

1. Ce travail s'inspire en grande partie du travail présenté par Jean-Loup Carré sur sa chaine YouTube : www.youtube.com/watch?v=I3dbeflpI-Q

un second nom au même objet. Vérifions-le.

```
L = [2, 5, 9]
M = L
M[1] = 8
>>> L
[2, 8, 9]
```




L'affectation `M[1] = 8` ne fait que modifier une valeur dans la liste qui reste toujours accessible par un de ses deux noms : `L` ou `M`.

**À retenir : Affectation d'une liste**

Dans le cas des listes, l'affectation `L = M` ne crée pas une copie.
L'affectation créé un second pointeur vers le même espace mémoire.

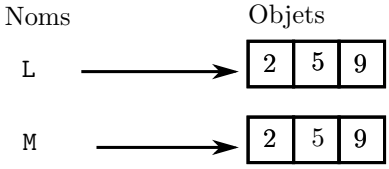
### I.6.b Comment alors copier une liste ?

Si on veut dupliquer une liste le moyen recommandé est d'utiliser une fonction à importer qui réalise une copie profonde d'une liste.

```
from copy import deepcopy
```

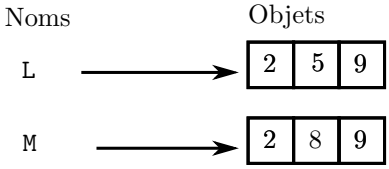
La fonction `deepcopy` est alors disponible et `M = deepcopy(L)` crée une nouvelle liste `M` qui a les mêmes valeurs que `L` mais qui est indépendante.

```
L = [2, 5, 9]
M = deepcopy(L)
```




Le nom `M` pointe vers un objet différent, vers une nouvelle référence en mémoire où est stockée la copie de la liste. La référence pointe donc vers un espace physique nouveau. Vérifions-le.

```
L = [2, 5, 9]
M = deepcopy(L)
M[1] = 8
>>> L
[2, 5, 9]
```




**À retenir : Faire la copie d'une liste**

Pour copier une liste `L`, utiliser la commande dédiée du module `copy`, `deepcopy`.

```
from copy import deepcopy

L = [2, 5, 9]
M = deepcopy(L)
```

## I.7 Modification d'une liste dans une fonction

Une fonction peut avoir pour but de modifier les valeurs d'une liste passée en paramètre. Dans ce cas la variable utilisée dans la fonction est associée à la liste initiale et les modifications seront conservées à la fin de l'exécution de la fonction. Aucun nouvelle liste ne sera créée.
Par conséquent, l'instruction **`return`** est inutile. La liste a été modifiée en place, un simple rappel de la liste le montre.

### I.7.a Exemple

Prenons l'exemple d'une fonction qui élève au carré les valeurs de la liste entrée en argument.

```
def carre(L):
    n = len(L)
    for i in range(n):
        L[i] = L[i]**2
```

La fonction ne retourne rien. En revanche, elle a modifié le liste entrée en argument.

```
L1 = [2, 5, 8]
carre(L1)
>>> L1
[4, 25, 64]
```

**Exercice 3 - Permutation de deux éléments d'une liste**
Construire une fonction `permute(L,i,j)` qui permute `L[i]` et `L[j]`. On admet que les indices `i` et `j` sont tels que `0 <= (i, j)< n` où `n = len(L)`.

# II Création de liste

Pour manipuler une liste, il faut l'avoir créée. Plusieurs méthodes sont possibles

## II.1 Création par extension

### II.1.a A partir de sa définition

On peut créer une liste en écrivant tous ses éléments séparés par des virgules et encadrés par des crochets.
`L = [1, 4, 9, 16, 25, 36, 49]`
Cette méthode n'est possible que pour les petites longueurs.

### II.1.b Opération algébrique de l'addition : concaténation

Un élément peut être ajouté en fin de liste de la façon suivante.
`L1 = L + [10]`
ou en début de liste
`L2 = [24] + L1`
L'élément ajouté est ici lui-même une liste – valeur entre crochet `[10]`.
Attention, cette concaténation de listes pour son augmentation nécessite à chaque fois une recopie de la liste complète. Son coût est donc une fonction de sa taille. Il existe en `Python` une méthode plus efficace qui évite cette recopie.

### II.2 Création par remplissage

Une méthode plus efficace consiste à créer une liste de `n` zéro pour ensuite substituer ces `0` par la valeur désirée :

1. on crée une liste "neutre" : la création d'une liste de taille `n` peut se faire par `[0]*n`.
   On a là une seconde opération algébrique, la multiplication.
   Ajouter une liste plusieurs fois à elle-même revient à la multiplier par un entier.

   ```
   >>> zero = [0]
   >>> zero*3
   [0, 0, 0]
   ```

2. on remplit cette liste de `n` zéro pas-à-pas à l'aide d'une boucle qui permet des opérations sur les valeurs de la liste, liste modifiée à chaque tour de boucle.

Par exemple, on choisit de créer la liste des carrés des entiers naturels en créant la fonction suivante :

```
def listeCarres(n):
    """ Entrée : un entier n
    Sortie : la liste des carrés des entiers de 1 à n"""
    carres = [0]* n
    for i in range (n):
        carres[i] = (i+1) ** 2
    return carres
```

### II.3 Limite de ces méthodes

1. La création par extension pose le problème de la recopie de la liste à chaque ajout. Nous verrons l'usage de la méthode spécifique à `Python`, `append()` qui ajoute l'élément sans la recopie de la liste.
2. La création par remplissage nécessite de connaître la taille de la liste à fabriquer, elle n'est pas toujours connue à l'avance.

## III Exercices

**Exercice 4 - Générer une liste d'entiers**
Définir une fonction `entier(n)` qui prend un entier `n` et renvoie la liste des entiers compris entre 0 et `n`.
Remarque : La fonction `list` permet de réaliser cette fonction : `list(range(n+1))`

**Exercice 5 - La suite de Fibonacci**
En mathématiques, la très célèbre *suite de Fibonacci* est une séquence infinie d'entiers définie de la façon suivante : on part des deux entiers 0 et 1 puis on construit à chaque fois l'entier suivant comme la somme des deux entiers précédents.

0, 1, 1, 2, 3, 5, ...

Écrire une fonction `fibo(n)` qui renvoie le tableau contenant les `n` premiers éléments de la suite.
Vérifier que le dernier terme de `fibo(30)` est 514229.

**Exercice 6 - Copie d'un tableau**
Écrire une fonction `copie(L)`, qui n'utilise pas `deepcopy` qui prend en paramètre un tableau `L` et renvoie une copie de ce tableau. Que test pouvez-vous imaginer pour vous assurer que votre copie est correcte ?
On pourra faire la copie de la liste suivante `t = [-10, 2, 23, 8, 5, 4, 1, 6, 12, 7]`

**Exercice 7 - Algorithme de Knuth**
Pour mélanger les éléments d'un tableau aléatoirement, il existe un algorithme très simple qui procède ainsi : on parcourt le tableau de la gauche vers la droite et, pour chaque élément à l'indice `i`, on l'échange avec un élément situé à un indice tiré aléatoirement entre `0` et `i` (inclus). Écrire une fonction `melange(tab)` qui réalise cet algorithme en modifiant en place `tab`. On pourra se servir à nouveau de la fonction `echange` créée précédemment.
Le module `random` possède une fonction `randint` qui permet un tirage aléatoire d'un entier.

```
from random import randint
al = randint(0,10)
```

`al` est un entier aléatoirement situé entre 0 et 10 inclus.

**Exercice 8 - Recherche d'un maximum/minimum dans une liste**

1. Sans utiliser les fonctions `max` ou `min` de Python, proposer une fonction `monMax(L)` qui renvoie sa valeur maximale d'une liste de nombre `L`.
2. Que faut-il changer pour que cette fonction renvoie le minimum ?
3. Comment modifier ces deux fonctions pour qu'elles renvoient seulement l'indice de position dans la liste du maximum ou du minimum.

**Exercice 9 - Calcul de tous les nombres premiers inférieurs à un entier $n$ donné**
Au III$^{e}$ siècle avant Jesus-Christ, Eratosthène, mathématicien, astronome et philosophe, invente une méthode efficace pour énumérer tous les nombres premiers inférieurs à un entier $n$ donné. Cette méthode est connue sous le nom de *crible d'Eratosthène*.
L'algorithme procède par élimination : il s'agit de supprimer d'une liste des entiers de 0 à $n$ tous les multiples d'un entier autres que lui-même. On commence par retirer de la liste les multiples de 2, puis les multiples de 3 restants, puis les multiples de 5 restants, et ainsi de suite en éliminant à chaque fois tous les multiples du plus petit entier restant.
Écrire une fonction `erato(n)` qui met la valeur de l'entier à retirer à 0. Par exemple pour $n = 11$, la fonction renvoie : `[0,0,2,3,0,5,0,7,0,0,0,11]`.

# IV Solutions

### Solution de l'exercice 1 - Somme des nombres d'une liste

```
def somme(L):
    n = len(L)
    som = 0
    for i in range(n):
        som += L[i]
    return som
```

### Solution de l'exercice 2 - Moyenne des nombres d'une liste

```
def moyenne(L):
    n = len(L)
    som = 0
    for i in range(n):
        som += L[i]
    moy = som / n
    return moy
```

### Solution de l'exercice 3 - Permutation de deux éléments d'une liste

```
def echange (L, i, j):
    """ Entree : une liste et deux indices i, j
    Requis : 0 <= i,j < len ( liste )
    Sortie : les termes i et j sont échangés"""
    tmp = L[i]
    L[i] = L[j]
    L[j] = temp
```

Cette fonction modifie la liste en place, il est donc inutile que la fonction retourne la liste. De fait, il y a pas de **return** à cette fonction.

### Solution de l'exercice 4 - Générer une liste d'entiers

```
def entier(n):
    ent = [0]*(n+1)
    for i in range(n+1):
        ent[i] = i
    return ent
```

### Solution de l'exercice 5 - La suite de Fibonacci

```
def fibo(n):
    ufib = [0]*n
    ufib[0] = 0
    ufib[1] = 1
    for i in range(2,n):
        ufib[i] = ufib[i - 1] + ufib[i - 2]
    return ufib
```

**Solution de l'exercice 6 - Copie d'un tableau**

```
def copie(L):
    n = len(L)
    L_copie = [0]*n
    for i in range(n):
        temp = L[i]
        L_copie[i] = temp
    return L_copie
```

Pour le test. On nomme la copie `t2 = copie(t)`. On fait une substitution sur un élément de la liste `t2[2] = 'a'`, puis on lit `t` qui ne doit pas avoir été modifié.

**Solution de l'exercice 7 - Algorithme de Knuth**

```
def melange(tab):
    n = len(tab)
    for i in range(1,n):
        al = randint(0,i)
        temp = tab[i]
        tab[i] = tab[al]
        tab[al] = temp

t = [-10, 2, 23, 8, 5, 4, 1, 6, 12, 7]
print('t avant',t)
melange(t)
print('t apres',t)

def melange2(tab):
    n = len(tab)
    for i in range(1,n):
        echange(tab,i,randint(0,i))
```

Pourquoi **`range()`** début à 1 ?

**Solution de l'exercice 8 - Recherche d'un maximum/minimum dans une liste**

```
def monMax(L):
    max_var = L[0]
    n = len(L)
    for i in range(n):
        if L[i] > max_var:
            max_var = L[i]
    return max_var

def monMin(L):
    min_var = L[0]
    n = len(L)
    for i in range(n):
        if L[i] < min_var:
            min_var = L[i]
    return min_var

#Complexité : la longueur de la liste : O(n)

def monMaxInd(L):
    max_var = L[0]
    indice = 0
    n = len(L)
    for i in range(n):
        if L[i] > max_var:
            max_var = L[i]
            indice = i
    return indice
```

**Solution de l'exercice 9 - Calcul de tous les nombres premiers inférieurs à un entier $n$ donné**

```
def erato(n):
    L = list(range(n+1))
    for i in range(2,n+1):
        for j in range(i+1,n):
            if L[j] != 0 and L[j] % i == 0:
                L[j] = 0
    return L
```