

# 1.4

## Les séquences

NSI 1ÈRE - JB DUTHOIT

Il est possible de "stocker" plusieurs grandeurs dans une même structure, ce type de structure est appelé une **séquence**.

### Définition

De façon plus précise, nous définirons une séquence comme un ensemble fini et ordonné d'éléments indicés de 0 à n-1 (si cette séquence comporte n éléments).

Nous commencerons par étudier un premier "type" de séquence, les **tuples**, puis nous enchaînerons sur les **tableaux**.

### 1.4.1 Les tuples en Python

```
mon_tuple = (5, 8, 6, 9)
```

```
type(mon_tuple)
```

Chaque élément du tuple est indicé.

☞ Retrouver comment accéder au premier élément, au second..etc...

⚠ dans les séquences les indices commencent toujours à 0 (le premier élément de la séquence a pour indice 0)

☞ Compléter le programme suivant :

```
mon_tuple = (Duthoit, Jean-Baptiste)
print(f"Mon nom est {mon_tuple[...]} et mon prénom {mon_tuple
[...]}")
```

### Propriété

Un **tuple** est une séquence **immuable** (c'est à dire qui ne peut pas être modifiée), pouvant contenir plusieurs autres objets.

#### Exercice 1.51

Créer une fonction nommée **distance**, qui prend en argument deux tuples (Dans chaque tuple, se trouvent les coordonnées (x;y) d'un point) et qui renvoie la distance entre ces deux points, dans un repère ortho-normé.

☞ On rappelle la formule qui permet de calculer la distance AB connaissant les coordonnées de A et de B :  $AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$

Par exemple, si on considère les points A = (3,4) et B = (6,9), l'appel **distance(A,B)** va renvoyer 5.

### Exercice 1.52

On considère un tuple de tuple :

(('Alim',17),('Pierre',34),('Sophie',19),('Angel',23))

Écrire 2 lignes de code python qui permettent d'afficher :

```
'bonjour Alim tu as 17 ans'
'bonjour Pierre tu as 34 ans'
'bonjour Sophie tu as 19 ans'
'bonjour Angel tu as 23 ans'
```

## 1.4.2 Les tableaux en Python

### Création d'un tableau vide

Il existe deux moyens :

Avec la commande `list` :

```
mon_tableau = list()
```

ou bien

```
mon_tableau = []
```

$\triangle$  Python utilise le terme "list", mais il s'agit en réalité d'un tableau (tableau dynamique en réalité)

### Création d'une tableau non vide

```
mon_tableau_1 = [1, 2, 3, 4]
mon_tableau_2 = [36, "voiture", 2.8, True]
mon_tableau_3 = [36, "texte", 2.8, mon_tableau_1]
```

### Accès au éléments

```
mon_tableau_1[2] # pour obtenir 3
```

### Modification de tableau

$\triangle$  Ce que ne permet pas un tuple :-)

```
mon_tableau_1[2] = 120 # pour remplacer 3 par 120
```

### Ajout d'un élément

On utilisera ici la méthode `append` :

```
mon_tableau = [1,2,3,4]
mon_tableau.append(5)
```

$\triangle$  La méthode "append" est très utilisée et permet d'ajouter un élément **à la fin** du tableau.

## Suppression d'éléments

On utilise ici la commande **del** :

```
mon_tableau = [1,2,3,4,5,6]
del mon_tableau[1] # suppression du second élément
\end{listing}

\subsubsection{Longueur d'un tableau}

On utilise ici la commande \cmd{len}:
\begin{lstlisting}[language=python, style=mystyle]
mon_tableau = [1,2,3,4,5,6]
len(mon_tableau) # affiche 6
```

## Parcourir un tableau version 1

```
mon_tableau = [1,2,3,4,5,6,7,8,9,10]
for element in mon_tableau:
    print(element)
```

## Parcourir un tableau version 2

```
mon_tableau = [1,2,3,4,5,6,7,8,9,10]
for i in range(len(mon_tableau)):
    print(mon_tableau[i])
```

### 1.4.3 Des tableaux et des chaînes

#### Des tableaux aux chaînes

☛ On utilise ici la commande **join** :

```
mon_tableau = ['Bonjour', 'à', 'tous']
a = " ".join(mon_tableau)
```

☛ Vérifiez que a est un str

#### Des chaînes aux tableaux

☛ On utilise ici la commande **split** :

```
ma_chaine = " Bonjour tout le monde !"
a = ma_chaine.split(" ")
```

☛ Vérifiez que a est du type **list** ☛ La méthode **split()** découpe donc la chaîne en fonction du paramètre donné.

## Propriété

Un tableau (list en Python) est une séquence **mutable** pouvant contenir plusieurs autres objets.

### Exercice 1.53

Écrire une fonction `ajout(tableau, entier)` qui prend un tableau d'entiers et un entier en argument et qui renvoie le tableau où l'entier a été ajouté à la fin.

```
assert tableau([1,2,3,4],5) == [1,2,3,4,5]
```

⚠ Remarquez bien que la liste a été modifiée en place.

### Exercice 1.54

Écrire une fonction `liste()` (sans paramètre) qui renvoie un tableau avec les entiers de 0 à 20

### Exercice 1.55

Écrire une fonction `liste_hasard()`(sans paramètre) qui renvoie un tableau avec 20 entiers choisis au hasard entre 0 et 100

### Exercice 1.56

Écrire une fonction `liste_hasard_n(n)` prenant en argument un entier n et qui renvoie un tableau avec n entiers choisis au hasard entre 0 et 100.

☞ Ne pas oublier : `from random import *`

### Exercice 1.57

Écrire une fonction `echange12(tableau)` qui prend en argument `tableau`, de type `list`, et contenant des entiers (au moins 2 éléments, de type `int`) et qui retourne le même tableau dans lequel on a échangé le premier et le second élément.

```
assert echange12([1,2,3,4]) == [2,1,3,4]
```

### Exercice 1.58

Écrire une fonction `echange12tri(tableau)` qui prend en argument `tableau`, de type `list`, et contenant des entiers (au moins 2 éléments, de type `int`) et qui retourne le même tableau dans lequel on a mis dans l'ordre croissant les deux premiers éléments.

```
assert echange12tri([1,2,3,4]) == [1,2,3,4] et assert echange12tri([11,2,3,4,1]) == [2,11,3,4,1]
```

### Exercice 1.59

Écrire une fonction `echange(tableau)` qui prend en argument `tableau`, de type `list`, et contenant des entiers (au moins 2 éléments, de type `int`) et qui retourne le tableau dans lequel on a échangé le premier et le dernier élément.

```
assert echange([1,2,3,4]) == [4,2,3,1]
```

### Exercice 1.60

Écrire une fonction nommée `liste(n)` qui a pour paramètre un entier `n` non nul et qui renvoie le tableau contenant les entiers de 1 à `n`.

Par exemple, `liste(10)` renvoie `[1,2,3,4,5,6,7,8,9,10]`

### Exercice 1.61

Écrire une fonction Python nommée `diviseurs(n)` qui prend en argument un entier naturel non nul `n` (type `int`) et qui renvoie la liste de ses diviseurs triés par ordre croissant.  
On pourra utiliser l'exercice précédent et le modifier!  
Par exemple, `diviseur(10)` renvoie `[1,2,5,10]`

### Exercice 1.62

Écrire une fonction qui permet d'écrire une liste d'entiers aléatoires positifs qui s'arrête dès que la somme des nombres dépasse une certaine valeur. La fonction `makeList(deb,fin,maxi)` prend en entrée 3 valeurs entières `deb`, `fin` et `maxi` correspondant respectivement à la plus petite valeur des nombres aléatoires, à la plus grande et à la valeur à ne pas dépasser en faisant la somme de ces nombres.

### Exercice 1.63

Écrire une fonction nommée `liste_1(n)` qui a pour paramètre un entier positif `n` non nul et qui renvoie le tableau contenant tous les carrés parfaits inférieurs à `n`, et dans l'ordre.  
Par exemple, `liste_1(10)` renvoie `[0,1,4,9]` et `liste_1(35)` renvoie `[0,1,4,9,16,25]`

### Exercice 1.64

Écrire une fonction nommée `liste_2(n)` qui a pour paramètre un entier positif `n` non nul et qui renvoie le tableau contenant tous les nombres impairs inférieurs à `n`, et dans l'ordre.  
Par exemple, `liste_2(10)` renvoie `[1,3,5,7,9]` et `liste_2(15)` renvoie `[1,3,5,7,9,11,13,15]`

### Exercice 1.65

Écrire une fonction nommée `liste_3(n)` qui a pour paramètre un entier positif `n` non nul et qui renvoie le tableau contenant tous les multiples positifs de 7 inférieurs à `n`, et dans l'ordre.  
Par exemple, `liste_3(10)` renvoie `[0,7]` et `liste_3(15)` renvoie `[0,7,14]`

### Exercice 1.66

Écrire une fonction nommée `liste_4(n)` qui a pour paramètre un entier positif `n` non nul et qui renvoie le tableau contenant tous carrés parfaits inférieurs à  $n^2$ , et dans l'ordre.  
Par exemple, `liste_4(10)` renvoie `[0,1,2,4,9,16,25,36,49,64,81,100]` et `liste_4(5)` renvoie `0,1,2,4,9,16,25`

### Exercice 1.67

Écrire une fonction `permute(tableau,i,j)` qui permute dans `tableau` les éléments d'index `i` et `j`.  
Par exemple, `permute([0,1,2,4,9,16,25,36,49,64,81,100],3,4)` renvoie `[0,1,2,9,4,16,25,36,49,64,81,100]`

## 1.4.4 Le sujet -délicat- des copies de tableaux

Étudiez soigneusement les lignes suivantes :

```
>>> x = [1, 2, 3]
>>> y = x
>>> y
[1, 2, 3]
>>> x[1] = -15
>>> x
[1, -15, 3]
>>> y
[1, -15, 3]
```

⚠ Vous voyez que la modification de `x` modifie `y` aussi ! Pour comprendre ce qui se passe, rendez-vous dans python Tutor !

## Copie explicite de la liste initiale

```
>>> x = [1, 2, 3]
>>> y = list(x)
>>> x[1] = -15
>>> y
[1, 2, 3]
```

## Deepcopy

⚠️ Attention, l' astuce précédentes ne fonctionne que pour les listes à une dimension, autrement dit les listes qui ne contiennent pas elles-mêmes d'autres listes.

La méthode qui fonctionne à tous les coups est la méthode de la copie profonde avec la commande **deepcopy** :

```
>>> import copy
>>> x = [[1, 2], [3, 4]]
>>> x
[[1, 2], [3, 4]]
>>> y = copy.deepcopy(x)
>>> x[1][1] = 99
>>> x
[[1, 2], [3, 99]]
>>> y
[[1, 2], [3, 4]]
```