# TP - Tableaux

M. Tellene

## 1 Création de l'environnement de travail

Avant de commencer le TP, vous allez créer votre environnement de travail. Pour ce faire vous allez dans votre dossier personnel, puis « apprentissage_python ». Une fois arrivé, créer un dossier « TP_tableaux ». C'est dans ce dossier que vous sauvegarderez les exercices de ce TP.

## 2 Introduction

Nous avons vu qu'en Python il existait plusieurs types de donnée pour les variables que nous utilisons dans nos programme : `integer` (`int`), `float`, `bool`, `string` (`str`),... Seulement une variable simple ne peut contenir qu'une valeur à la fois.

Il existe cependant des **structures de données** qui permettent de stocker plusieurs variables en même temps. Un de ces structure s'appelle « tableau ».

Nous allons dans un premier temps voir en saisie console quelques lignes de commandes permettant d'appréhender le fonctionnement des listes.

## 3 Commande `Shell`

Les commandes suivantes sont à taper dans le `shell` (rappel : c'est la partie basse de l'éditeur de code). Un tableau se déclare **comme n'importe quelle variable** vue jusqu'à présent. Exécuter successivement dans la console les lignes suivantes :

```
>>> var = 42
>>> print(var)
>>> print(type(var))

>>> var = [42]
>>> print(var)
>>> print(type(var))
```

1. Quel est l'affichage obtenu lors de l'exécution de la ligne 6? .....
2. Quel résultat est obtenu lors de l'exécution de la ligne 7? .....
3. Quel(s) élément(s) de syntaxe permette(nt) d'avoir une variable de ce type? .....

   .....

Un tableau possède une taille qui peut évoluer au fil du temps :

```
>>> var = [42]
>>> var.append(84)
>>> print(var)

>>> var.remove(42)
>>> print(var)
```

EXERCICE 1

**Au fur et à mesure du TP, compléter le tableau suivant.** Ce tableau récapitule les fonctions usuelles associées aux tableaux :

| Avant la fonction | Appel de la fonction | Résultat |
|---|---|---|
| `L = [42, 84]` | `L.append(50)` | `L = [42, 84, 50]` |
| `L = [42, 84, 50]` | `L.remove(42)` | |
| `L = [42, 84, 50]` | `print(L[1])` | |
| `L = [42, 84, 50]` | `print(len(L))` | |
| Liste inexistante | `L = []` | |
| `L = [42, 84, 50]` | `L.index(42)` | |
| `L = [42, 84, 50]` | `50 in L` | |
| `L = [42, 84, 50]` | `max(L)` | |
| `L = [42, 84, 50]` | `min(L)` | |

Avec vos manipulations, vous avez pu remarquer qu'en appelant le tableau, son contenu s'affiche.

Exemple :

```
>>> var = [42, 84, 50]
>>> print(var)
```

Il est également possible, comme pour les chaînes de caractères, d'extraire certains éléments d'un tableau. Pour ce faire, il faut utiliser l'indice de l'élément dans celui-ci. Comme pour les chaînes de caractères, le premier élément d'un tableau a pour indice 0.

```
>>> var = [42, 84, 50]
>>> print(var)
>>> print(var[0])
>>> print(var[1])
>>> print(var[2])
>>> print(var[3])
```

Un tableau peut contenir tous les types de variables possibles (`int`, `str`, `float` et même nos propres types de variables [1] !). La seule chose à respecter est qu'il faut que le type soit unique : un tableau dont les éléments sont des `int` ne peut pas contenir d'éléments d'un autre type.

1. Terminale NSI

Comme pour les chaînes de caractères, encore, un tableau possède une taille récupérable avec la fonction `len()`.

```
>>> var = [42, 84, 50]
>>> print(len(var))
```

1. Quel est le résultat de l'instruction `print(var[3])` ? .....
2. Quelle est la longueur du tableau `var` ? .....
3. Retrouver chaque correspondance entre la commande et le résultat

- `>>> print(lettre)`
- `>>> print(lettre[3])`
- `>>> print(lettre[0]+lettre[6])`
- `>>> print([lettre[0]]+[lettre[6]])`
- `>>> print([lettre[0]+lettre[6]])`
- `>>> print(len(lettre)`
- `>>> print(lettre[-1])`

- `10`
- `['a','g']`
- `['d']`
- `['j']`
- `['ag']`
- `'ag'`
- `['a','b','c','d','e','f','g','h','i','j']`

## 4 A vous de jouer

EXERCICE 2

Pour cet exercice, il vous faudra créer un fichier **carte_identite.py**

Recopier et compléter le programme suivant pour obtenir l'affichage demandé :

```
# cette partie est le code
nom = "Martin"
prenom = ["Charles", "Jean", "Pierre"]
date = "01.03.1975"
departement = 75
taille = 1.70
identite = [nom,prenom,date,departement,taille]
for element in identite:
    print(element)
```

```
#cette partie est l'affichage demandé
Martin
['Charles', 'Jean', 'Pierre']
01.03.1975
75
1.7
Charles Martin
```

1. En utilisant la fonction `type()` et `print()`, compléter le tableau suivant :

| Objet | Type | Valeur |
|---|---|---|
| `identite[0]` | | |
| `identite[1]` | | |
| `identite[2]` | | |
| `identite[3]` | | |
| `identite[4]` | | |

2. Quelle commande (en se limitant à l'utilisation d'`identite`) permet d'afficher « Charles Martin » dans la console ? .....

EXERCICE 3
Pour cet exercice, il vous faudra créer un fichier **creation_tableau.py**

1. Créer une fonction `tab_cent_premiers_nombres` qui ne prend pas d'argument. Cette fonction devra renvoyer un tableau contenant les 100 premiers entiers
2. Créer une fonction `tab_cent_premiers_nombres_pairs` qui ne prend pas d'argument. Cette fonction devra renvoyer un tableau contenant les 100 premiers entiers pairs

# 5 Énumération

Nous avons vu la structure de boucle `for` pour créer un tableau mais il est aussi possible de l'utiliser pour itérer sur les éléments de celui-ci.

1. Créer un fichier `enumeration.py`
2. Recopier le code suivant :

```
# énumération par les éléments
liste = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
for var in liste:
    print(var,liste.index(var))

# énumération par les indices
for var in range(len(liste)):
    print(var,liste[var])
```

EXERCICE 4
Pour cet exercice, il faudra modifier le fichier **creation_tableau.py**

Écrire une fonction `creation_par_extraction` qui ne prend pas d'argument. La fonction devra, à partir de le tableau des 100 premiers entiers, renvoyer un tableau des 100 premiers entiers impairs par extraction d'éléments. Voici, dans l'ordre, ce que cette fonction devra faire :

1. Appeler la fonction `tab_cent_premiers_nombres` et mettre le résultat dans une variable (`tab_1` par exemple)
2. Créer un tableau vide
3. Itérer sur les éléments de `tab_1` et mettre les nombres impairs dans le deuxième tableau
4. Renvoyer le deuxième tableau

EXERCICE 5
Pour cet exercice, il faudra modifier le fichier **creation_tab.py**

Écrire une fonction `affichage` qui prend en argument un tableau. La fonction devra demander un nombre à l'utilisateur (cf. `int(input())`) et énumérer [2] les éléments du tableau entré en argument.

Les éléments du tableau étant un multiple du nombre entré devront être suivi d'une « * ».

2. Vous pourrez choisir la méthode d'énumération (par les éléments ou par les indices) que vous voulez

Exemple de rendu :

```
>>> tab1 = tab_cent_premiers_nombres()

>>> affichage(tab1)
Entrer un nombre : 6
0 *
1
2
3
4
5
6 *
7
...
```

## 6 Petite parenthèse chaînes de caractères

Les chaînes de caractères sont proches des tableaux. En effet, il est possible de :

— récupérer un caractère en particulier avec « [...] » :

```
>>> chaine = "salut"
>>> print(chaine[3])
'u'
```

— récupérer la longueur avec la fonction `len()` :

```
>>> chaine = "salut"
>>> print(len(chaine))
5
```

— énumérer les éléments de la chaîne avec un boucle `for` :

```
# énumération par les éléments
chaine = "salut"
for lettre in chaine:
    print(lettre, chaine.index(lettre))

's' 0
'a' 1
'l' 2
'u' 3
't' 4

# énumération par les indices
chaine = "salut"
for lettre in range(len(chaine)):
    print(lettre, chaine[lettre])

0 's'
1 'a'
2 'l'
3 'u'
4 't'
```

## 7 Tableau par compréhension

Au moment de créer un tableau, il faut lui attribuer une valeur initiale. Cette valeur est parfois le tableau vide (`tab = [ ]`) qui permet de donner le type `<class 'list'>` à la variable créée. Dans

un des exercices précédents, il vous était demandé de créer un tableau contenant les 100 premiers entiers. La solution la plus simple était de créer un tableau vide, d'utiliser une boucle `for` allant jusqu'à 100 et d'ajouter les éléments un à un.

Ces opérations peuvent être simplifier à une instruction : `tab = [n for n in range(100)]`. L'instruction précédente consiste à créer un tableau **par compréhension**.

Décomposition de l'instruction (il n'y a rien a remplir) :

- `tab = ..................` : création de la variable
- `tab = [..................]` : on spécifie que la variable contiendra un tableau, la variable sera donc de type `<class 'list'>`
- `tab = [... for n in range(100)]` : on spécifie la condition que devra respecter les éléments du tableau, ici les éléments iront de 0 à 99
- `tab = [n for n in range(100)]` : littéralement l'instruction `n for n in range(100)` dit : « je mets dans mon tableau tous les `n` allant de 0 à 99 »

EXERCICE 6
Compléter le tableau :

| Commande | Description | Nombre d'éléments du tableau |
|---|---|---|
| `tab = [n for n in range(100)]` | `Tableau des 100 premiers entiers` | `100` |
| `tab = [2*n for n in range(100)]` | | |
| `tab = [2*n+1 for n in range(100)]` | | |
| `tab = [n for n in range(100) if n%2 == 0]` | | |
| `tab = [n for n in range(100) if n%2 != 0]` | | |
| `tab = ['.' for n in range(100)]` | | |
| `tab = [5*['.'] for n in range(100)]` | | |

## 8 Les tableaux de tableaux

Nous avons vu qu'un tableau pouvait être lui-même un élément d'un tableau (`prenom` dans `identite` de Charles Martin). Ce mécanisme est très pratique pour recréer un tableau de tableaux de données.

EXERCICE 7
En vous aidant du dernier exemple des tableaux par compréhension, créer un tableau de tableaux de 10 lignes et 10 colonnes où chaque case contient le caractère « . »

| Commande | Description |
|---|---|
| `tab = .....` | Créer un tableau de 10 lignes × 10 colonnes |
| `print(tab)` | |
| `for ligne in tab:`<br>`  print(ligne)` | |
| `print(tab[7])` | |
| `print(tab[4][2])` | |

EXERCICE 8

Pour cet exercice, il faudra créer un fichier **multiplication.py**

Écrire une fonction `tables` qui ne prend pas d'argument. Cette fonction devra, avec la méthode de votre choix, créer et renvoyer un tableau de tableaux contenant les tables de multiplications de 0 à 9.

Rendu :

```
>>> tables()
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [0, 2,
    4, 6, 8, 10, 12, 14, 16, 18, 20], [0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
    [0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40], [0, 5, 10, 15, 20, 25, 30, 35,
    40, 45, 50], [0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60], [0, 7, 14, 21, 28,
    35, 42, 49, 56, 63, 70], [0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80], [0, 9,
    18, 27, 36, 45, 54, 63, 72, 81, 90]]
```

EXERCICE 9

Pour cet exercice, il faudra modifier le fichier **multiplication.py**

Écrire une fonction `avoir_resultat` qui prend en argument 2 nombres (`x` et `y` par exemple). Cette fonction devra, **sans calcul**, renvoyer le résultat de `x` × `y`. Pour ce faire vous ferez un accès dans le tableau que vous avez créé précédemment.

Exemple de rendu :

```
>>> avoir_resultat(5, 5)
25

>>> avoir_resultat(3, 4)
12
```

# 9 Algorithmes de bases

Les tableaux sont sujets à beaucoup d'algorithmes dits de bases. Nous verrons certains de ces algorithmes au cours de l'année. Dans les exercices suivants vous en verrez quelque uns.

Les fonctions demandées dans cette partie, devront être écrites dans un fichier **bases.py**.

## 9.1 Recherche d'une valeur dans une liste

EXERCICE 10

Écrire une fonction `est_present` qui prend en argument un tableau et un nombre. Le fonction devra renvoyer `True` si le nombre est dans le tableau, `False` sinon. **Vous n'avez pas le droit d'utiliser `in`.**

Exemple de rendu :

```
>>> dans_liste([1,2,3,4], 4)
True

>>> dans_liste([1,2,3,4], 5)
False
```

EXERCICE 11

1. Écrire une fonction `maximum_tab` qui prend en argument un tableau. Le fonction devra renvoyer l'élément du tableau qui possède la plus grande valeur. **Vous n'avez pas le droit d'utiliser la fonction `max()`**

   Exemple de rendu :

   ```
   >>> maximum_liste([1,2,3,4])
   4

   >>> maximum_liste([1,5,3,4])
   5
   ```

2. Écrire une fonction `maximum_tab_avec_indice` qui prend en argument un tableau. La fonction devra renvoyer l'élément du tableau qui possède la plus grande valeur **mais également** l'indice de cet élément dans le tableau. **Vous n'avez toujours pas le droit d'utiliser la fonction `max()`**

   Exemple de rendu :

   ```
   >>> maximum_liste_avec_indice([1,2,3,4])
   (4, 3)

   >>> maximum_liste_avec_indice([1,5,3,4])
   (5, 1)
   ```

EXERCICE 12

1. Écrire une fonction `minimum_tableau` qui prend en argument un tableau. Le fonction devra renvoyer l'élément du tableau qui possède la plus petite valeur. **Vous n'avez pas le droit d'utiliser la fonction `min()`**

   Exemple de rendu :

   ```
   >>> minimum_liste([1,2,3,4])
   1

   >>> minimum_liste([0,0,0,-1])
   -1
   ```

2. Écrire une fonction `minimum_tableau_avec_indice` qui prend en argument un tableau. La fonction devra renvoyer l'élément du tableau qui possède la plus petite valeur **mais également** l'indice de cet élément dans le tableau. **Vous n'avez toujours pas le droit d'utiliser la fonction** `min()`

   Exemple de rendu :

```
>>> minimum_liste_avec_indice([1,2,3,4])
(1, 0)

>>> minimum_liste_avec_indice([0,0,0,-1])
(-1, 3)
```