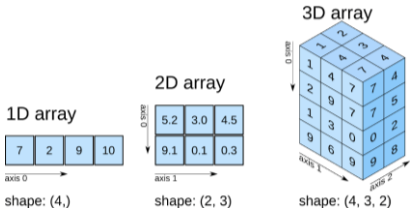
Cours – Les tableaux et les listes

Objectifs :
- ⇨ Comprendre la notion de tableau en programmation
- ⇨ Voir comment python gère les tableaux avec des listes



I – Les tableaux

Les tableaux sont des outils de base en programmation car ils permettent ..... et ..... les données. Ils permettent également de stocker et d'accéder à ..... de données (par exemple l'ensemble des pixels d'une image, ou tous les mots de la langue française).

1) Création d'un tableau

En python, un tableau est délimité par des crochets (`[ … ]`) et ses éléments sont séparés par des virgules.

Pour créer un tableau, on peut utiliser la syntaxe suivante :
```
tab = [valeur] * nbElements
```

Ceci crée un tableau nommé `tab` et contenant `nbElements` valant tous `valeur`.

Exemple :
`maListe = ['z'] * 8` crée le tableau `['z', 'z', 'z', 'z', 'z', 'z', 'z', 'z']`

On peut créer des tableaux d'entiers, de pseudo-réels (nombres à virgules), de textes, de booléens, … en réalité de n'importe quel type d'objet.

Il est possible de créer un tableau vide : `tabVide = []` ou bien de créer un tableau contenant déjà des données que l'on précise : `tabRempli = [valeur1, valeur2, valeur3, …]`

Exemples :
```
liste1 = []                     # Ce tableau est vide
tableau2 = [1,2,3,5,7,11]       # Ce tableau contient les 6 premiers nombres premiers
```

Dans un tableau les éléments sont placés ....., c'est-à-dire qu'ils ont ..... Ceci permet à l'interpréteur de placer tous les éléments les uns à la suite des autres en mémoire et ainsi d'y accéder plus rapidement.

2) Utilisation d'un tableau

Pour accéder aux éléments d'un tableau, il existe plusieurs méthodes :

a. On veut accéder à tous les éléments les uns après les autres

Dans ce cas, on utilise naturellement une boucle for. En effet, une liste en python est un **itérateur**. On pourra donc avoir une structure du type :

```
tableau = [1, 5, 3, -1, 2]
somme = 0
for element in tableau :
    print(element)
    somme = somme + element
print('total :', somme)
```

Le programme affichera :
```
1
5
3
-1
2
total : 10
```

b. On veut accéder à un élément en particulier

On peut accéder à un élément de tableau en indiquant son **indice**. On a vu précédemment que les données du tableau étaient stockées en ordre les unes derrière les autres. Elles ont ainsi toutes un numéro d'ordre en partant de 0 pour le premier élément du tableau jusqu'à n-1 pour un tableau de longueur n.

Python autorise également des indices négatifs qui partent de la fin du tableau (voir ci-dessous).

Exemple :

```
tableau = [8, 5, 13, -1, 72]
```

Indices positifs	Indices négatifs
`tableau[0] vaut 8`	`tableau[-5] vaut 8`
`tableau[1] vaut 5`	`tableau[-4] vaut 5`
`tableau[2] vaut 13`	`tableau[-3] vaut 13`
`tableau[3] vaut -1`	`tableau[-2] vaut -1`
`tableau[4] vaut 72`	`tableau[-1] vaut 72`

c. On veut connaitre la taille du tableau

Pour cela il existe une fonction built-in très simple et qui fonctionne sur la plupart des objets de python : la fonction `len()` (pour *length*, la longueur en anglais). Elle renvoie la taille du tableau qui lui est passé en argument.

Exemples :

```
t = [1, 2, 3, 4, 5]
taille = len(t)                 # taille vaut 5
tabVide = []
longueur = len(tabVide)         # longueur vaut 0
```

3) Mécanique interne

Voyons avec python tutor comment python gère les tableaux.

Application n°1 :

Copier-coller le code ci-contre dans python tutor et exécutez-le pas à pas.

1) Comment Python stocke-t-il les tableaux dans l'espace des noms et des valeurs ?

```
a = [7]*5
b = [0, 2, 8, -2, 3, 9, 12]
a[3] = 13
b[2] = a[1]
x = a[1]
c = b
y = x
c[4] = 18
y = 4
print(b, x)
```

2) Expliquez la différence entre ce qui se passe pour les variables b et c d'une part et les variables x et y d'autre part.

On dit que les variables de type tableau (`list` en python) sont ..... (ou plus précisément que le type `list` est mutable). Cela signifie que ce sont des variables dont on peut modifier le contenu.
Les entiers (`int`) par exemple sont des types non-mutables : Si on veut modifier une variable qui contient un entier, celle-ci va simplement pointer vers un entier différent, mais la valeur en mémoire n'est pas modifiée (voir python tutor en mode « Render all objects on the heap (Python) »). C'est le cas aussi des `float` ou des `bool` qui sont non-mutables également.

Le fait que le type `list` soit mutable implique qu'il faut faire attention lorsqu'on les utilise dans les fonctions.

Application n°2 :

1) Sans exécuter le code, donner la valeur du tableau `a` à la fin de l'exécution du programme ci-contre. La réponse change-t-elle si on place la ligne 2 (`a = [9, 18, 27]`) juste après la 5 (juste avant la fin de la fonction) ?

```
1  def f(tab, val):
2      a = [9, 18, 27]
3      for i in range(len(tab)):
4          if tab[i] > val:
5              tab[i] = tab[i]-val
6
7  a = [56, 33, 74, 25, 11]
8  f(a, 50)
9  print(a)
```

Copier-coller le code dans python tutor et exécutez-le pas à pas.

2) Comment se fait le passage d'argument du programme principal vers la fonction au début de l'exécution de cette dernière ?

Les flèches que l'on observe dans python tutor correspondent à des ..... : elles servent à indiquer où se trouve la donnée. Lorsqu'on passe un tableau en argument à une fonction on donne en réalité une *référence* vers ce tableau. Et comme celui-ci est mutable, il pourra être modifié par la fonction.

II – Les str, des « tableaux de caractères »

On a vu dans le TP que les chaînes de caractère peuvent être utilisées en grande partie comme des tableaux :

- On peut les initialiser avec l'opérateur « `*` »,
- On peut accéder à la lettre d'indice `i` de la chaîne de caractère `chaine` avec la notation `chaine[i]`.
- On peut déterminer la longueur d'une chaîne avec la fonction `len`.

Il y a cependant une grande différence avec les tableaux, c'est que le type `str` ..... . Il n'est donc pas possible par exemple de modifier la lettre d'indice `i` avec une instruction de la forme `chaine[i] = "A"`.
Si on tente de le faire on aura une erreur « `TypeError: 'str' object does not support item assignment` » qui indique qu'on ne peut pas modifier un caractère d'une chaine.

III – Tableaux à dimensions multiples

Il est possible de créer des tableaux … contenant des tableaux et d'avoir ainsi des tableaux à deux dimensions (avec des lignes et des colonnes). C'est particulièrement utile pour la manipulation d'images par exemple.

1) Exemple et problème

Application n°3 :

1) *[Sans exécuter le programme]* Combien de lignes et de colonnes contiendra le tableau `tab` crée par le programme ci-contre ?

```
1  ligne = [0]*5
2  tab = [ligne]*3
3  print(tab)
4  tab[0][2] = 8
5  print(tab)
```

2) *[Sans exécuter le programme]* Que fait la ligne 4 (soyez précis) ?

3) Copier-coller le code dans python et exécutez-le. Que remarquez-vous ?

4) Copier-coller le code dans python tutor et exécutez-le pas à pas. Expliquez alors le comportement découvert à la question précédente.

5) On voudrait pourtant créer un tableau à deux dimensions où toutes les cases sont indépendantes. Proposez une solution et testez-là.

2) Simulation d'une image

Application n°4 :

1) Ecrire une fonction `creation_tableau(n)` qui renvoie un tableau à deux dimensions carré de taille n. Les cases du tableau sont pour l'instant initialisées à 0. Créer le programme principal qui demande à l'utilisateur de rentrer la dimension n de l'image et crée un tableau `image` à l'aide de la fonction précédente.
2) Créer une procédure `affiche_image(image)` qui affiche le tableau `image` donné en paramètre. On affichera le caractère espace si la case du tableau vaut l'entier 0 et un caractère `'*'` dans le cas contraire.
3) Ecrire et tester une procédure `croix(image)` qui « trace » une ligne verticale et une ligne horizontale au centre de l'image en mettant à 1 les valeurs des cases du tableau concernées.

IV - Plus puissantes que les tableaux : les listes

On a vu les principales caractéristiques des tableaux tels qu'ils sont utilisés dans la plupart des langages, mais comme on l'a déjà dit, les tableaux sont implémentés en Python à l'aide d'un type d'objet bien plus polyvalent que les tableaux : le type `list`. Voyons ce que ce type apporte de plus que les tableaux

1) Tableaux composites

Les listes de python[1] peuvent contenir des données de type différents.
Ex : `t = [3, 'test', 8.2, True, 42]` est une liste composite et parfaitement valide.

Cela peut s'avérer très pratique, mais nécessite quelques précautions d'usage car une opération licite pour un des éléments du tableau (une mise au carré par exemple) peut être interdite pour la case suivante (si c'est un booléen dans l'exemple précédent).

2) Taille variable

Avec ses listes Python va plus loin que beaucoup de langage et permet d'utiliser des tableaux de tailles variables auxquels on peut facilement rajouter ou enlever des éléments avec les **méthodes** `append()` pour ajouter un élément, `remove()` ou `pop()` pour supprimer.

Les méthodes sont des fonctions attachées à un objet. Pour les appeler on écrit le nom de l'objet suivi d'un point puis du nom de la méthode et enfin des parenthèses avec les arguments comme toute fonction.
Exemple :

```
tab = [12, 5]
print(tab)
tab.append(18)
print(tab)
```

Le programme affichera :
```
[12, 5]
[12, 5, 18]
```

Le tableau suivant récapitule les instructions de bases pour gérer les listes.

Fonctions et méthodes des listes	

Les listes numérotées (type **`list`**) :
Une liste est une structure permettant de stocker plusieurs valeurs de tous types en les rangeant selon leur numéro
	`t = []  # crée une liste vide`
	`t = [42, 'abc']  # crée une liste de deux valeurs (le nombre 42 et la chaîne 'abc')`
	`t = [0]*25  # crée une liste de 25 éléments ne contenant que des zéros`

Ajouter un élément x à la fin d'une liste : **`append`**`(x)`	`t.append(b) # ajout l'élément b à la fin de la liste t`
Déterminer la taille de la liste : **`len`**`()`	`t = [4, 'kilo', 9, -1, 0]` `longueurListe = len(t)` longueurListe vaut 5
Accéder à un élément de la liste par son numéro `t[i]` i+1ème élément de la liste t (les indices vont de 0 pour le 1er terme à la taille de la liste - 1) indices 0 1 2 3 4 5 6 7 `t = ['a','b','c','d','e','f','g','h']` indices -8 -7 -6 -5 -4 -3 -2 -1 On peut utiliser des indices négatifs pour désigner les éléments de la liste en partant de la fin (-1 est le dernier élément)	`t = ['x', 3.5, 42, 'Ampère']` `print(t[2])` Affiche 42 `t[0] = 7` t vaut maintenant [7, 3.5, 42, 'Ampère'] `t[-2] = 'k'` t vaut maintenant [7, 3.5, 'k', 'Ampère']
Supprimer un élément de valeur x de la liste : **`remove`**`(x)`	`t = [12, 5, -3, 'abc']` `t.remove(-3)` t vaut maintenant [12, 5, 'abc'] et t[2] = 'abc'

Supprimer un élément au rang i : **`pop`**`(i)`	`t = ['a', 2.1, 'b', 42, 'abc', 11, 'def']`
	`t.pop(3)`
	t vaut maintenant ['a', 2.1, 'b', 'abc', 11, 'def'] et t[4] = 11

[1] Ce n'est pas le cas d'autres langages comme Java ou C qui n'acceptent que des données de même type dans les tableaux.