CHAPITRE 15

Types construits - les tableaux

En poursuivant l'inaccessible, la simplicité se trouve en travers du chemin.

Alan Jay Perlis

Thème : types construits

Contenus	Capacités attendues	Commentaires
Tableau indexé, tableau donné en compréhension	Lire et modifier les éléments d'un tableau grâce à leurs index. Construire un tableau par compréhension. Utiliser des tableaux de tableaux pour représenter des matrices : notation a [i] [j]. Itérer sur les éléments d'un tableau.	Seuls les tableaux dont les éléments sont du même type sont présentés. Aucune connaissance des tranches (slices) n'est exigible. L'aspect dynamique des tableaux de Python n'est pas évoqué. Python identifie listes et tableaux.

I. Création d'un tableau

Déf. 1

Un tableau est une suite ordonnée d'éléments. Ces éléments sont séparés par des virgules et entourés par des crochets.

En Python, un tableau est de type `list`.

Exemple 1 (Un tableau d'entiers)

```
>>> t = [1, 2, 3, 4, 5]
>>> type(t)
list
```

t est un tableau composé de 5 éléments. Chacun de ces éléments est un nombre.

Exemple 2 (Un tableau de caractères)

```
>>> t = ['toi', 'moi', 'nous']
>>> type(t)
list
```

t est un tableau composé de 3 éléments. Chacun de ces éléments est une chaîne de caractères.

Un tableau peut aussi être composé de tuples ou de tableaux :

Exemple 3 (Un tableau de tuples et un tableau de tableaux)

```
>>> t1 = [(1, 2, 'moi'),(3, 4, 'toi'),(5, 6, 'nous')]
>>> t2 = [[0, 2, 4, 6, 8],[1, 3, 5, 7, 9]]
```

t1 est un tableau composé de 3 éléments. Chacun de ces éléments est un tuple.
t2 est un tableau composé de 2 éléments. Chacun de ces éléments est un tableau.

Activité 1.

1. Créer un tableau nommé *pairs* et contenant tous les entiers pairs positifs jusqu'à 10 inclus.
2. Créer un tableau nommé *impairs* et contenant tous les entiers impairs positifs jusqu'à 10 inclus.

II. Indexation des éléments d'un tableau

Si un tableau est non-vide, ses n éléments sont indexés par les entiers 0, 1, 2,. . . , n - 1. On accède à chacun de ses éléments en utilisant des crochets et l'indice de l'élément : `tableau[indice]`.

Attention

Les indices commencent à 0.

Exemple 4 (Indexation d'un tableau simple)

```
>>> t = [1,2,3,4,5]
>>> t[0]
1
>>> t[3]
4
```

Exemple 5 (Indexation d'un tableau emboîté)

```
>>> t = [[0,2,4,6,8],[1,3,5,7,9]]
>>> t[0][0]
0
>>> t[0][2]
4
>>> t[1][4]
9
```

Activité 2.
On considère le tableau suivant : mots = ['jambon', 'fromage', 'confiture', 'chocolat']. Indiquer le code Python renvoyant la chaîne "confiture".

Pour connaître le nombre d'éléments d'un tableau, on pourra utiliser l'instruction Python `len()`.

Exemple 6 (Nombre d'éléments d'un tableau simple)

```
>>> t = [1, 2, 3, 4, 5]
>>> len(t)
5
```

t est un tableau composé de 5 éléments. Chacun de ces éléments est un nombre.

Exemple 7 (Nombre d'éléments d'un tableau emboîté)

```
>>> t = [[0, 2, 4, 6, 8],[1, 3, 5, 7, 9]]
>>> len(t)
2
>>> len(t[1])
5
```

t est un tableau composé de 2 éléments. Chacun de ces éléments est un tableau.
t[1] est un tableau composé de 5 éléments. Chacun de ces éléments est un nombre.

À retenir

On peut affecter à une variable un élément d'un tableau. On pourra utiliser l'instruction Python `variable = tableau[indice]`.

Exemple 8 (Affectation d'un tableau)

```
>>> mots = ['jambon', 'fromage', 'confiture', 'chocolat']
>>> sujet = mots[0]
>>> sujet
'jambon'
```

III. Propriétés d'un tableau

Propriété 1

Dans la plupart des cas, on considérera que :

- les tableaux ont une taille fixe.
- les tableaux contiennent des objets du même type : que des entiers, que des chaînes de caractères, que des tuples ...

Propriété 2 (mutabilité d'un tableau)

Un tableau est mutable (on dit aussi muable), c'est à dire que l'on peut modifier l'un de ses éléments.

Exemple 9 (Modification d'un tableau d'entiers)

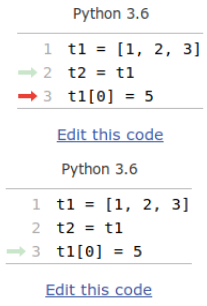
```
>>> t = [1, 2, 3, 4, 5]
>>> t[0] = 9
>>> t
[9, 2, 3, 4, 5]
```

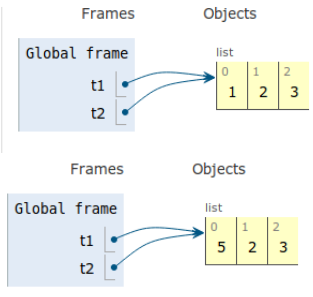
Propriété 3 (Identité de deux tableaux)

Deux tableaux qui sont liés par une relation d'égalité pointent vers la même adresse mémoire. **La modification de l'un entraîne la modification de l'autre**.

Exemple 10 (Identité de deux tableaux)

```
>>> t1 = [1, 2, 3]
>>> t2 = t1
>>> t2
[1, 2, 3]
>>> t1[0] = 5
>>> t1
[5, 2, 3]
>>> t2
[5, 2, 3]
```



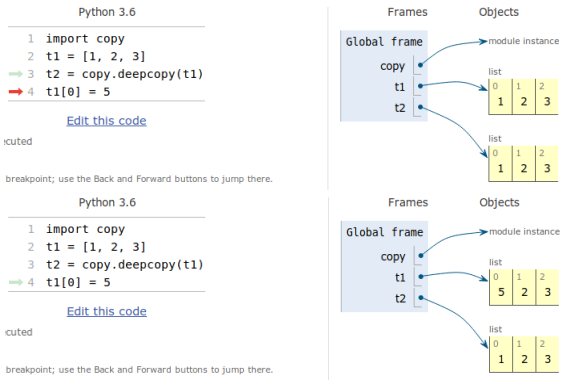


Lorsqu'on modifie le tableau t1 (ligne 5), le tableau t2 est lui aussi modifié.

Si l'on souhaite éviter cette modification, la solution consiste à copier le tableau dans la nouvelle variable. Pour cela on utilise la fonction `deepcopy()` du module `copy`.

Exemple 11 (Copie d'un tableau)

```
>>> import copy
t1 = [1, 2, 3]
>>> t2 = copy.deepcopy(t1)
>>> t2
[1, 2, 3]
>>> t1[0] = 5
>>> t1
[5, 2, 3]
>>> t2
[1, 2, 3]
```



Lorsqu'on modifie le tableau t1 (ligne 5), le tableau t2 n'est pas modifié.

IV. Itération sur les tableaux

Un tableau est un itérable. On peut donc itérer sur les éléments d'un tableau. On peut réaliser soit un parcours par index, soit un parcours par élément.

IV. 1. Parcours par index

L'indexation permet de remplir un tableau de manière automatique. La boucle *Pour* peut nous y aider. Le remplissage se passe alors en trois étapes :

1. Identifier la longueur du tableau.
2. Créer un tableau ne contenant que des 0 (pour un tableau de nombres) ou le caractère vide `''` (pour les tableaux de chaînes de caractères), ...
3. Utiliser une boucle bornée (*Pour i allant de ... à ...*) pour remplir le tableau.

Exemple 12 (Remplissage d'un tableau de 10 valeurs par le carré des n premiers nombres)

On cherche à obtenir le tableau suivant : [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

```
t = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
for i in range(0, len(t), 1):
    t[i] = i**2
```

Activité 3.
On considère une suite arithmétique définie par $U(n) = U(n-1) + 3n$ avec U(0) $= 0$. Écrire le code Python permettant de créer un tableau contenant les 20 premières valeurs.

IV. 2. Parcours par élément

Pour parcourir un ta bleau, on peut aussi parcourir le tableau élément par élément.

Exemple 13 (Affichage du contenu d'un tableau)

```
t = [1, 2, 3, 'bonjour', 8.5]

for element in t:
    print(element)
```

V. Opérations sur les tableaux

Les opérateurs + et * s'utilisent comme avec les chaînes de caractères et les tuples. Ce sont des opérateurs de concaténation. Le résultat est un tableau (un nouveau...).

Exemple 14 (Concaténation de tableaux)

```
>>> t1 = [0, 1, 2, 3]
>>> t2 = [4, 5, 6, 7]
>>> t1 + t2
[0, 1, 2, 3, 4, 5, 6, 7]
>>> 3*t1
[0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
```

Activité 4.
Créer une fonction *creation*() permettant de créer un tableau contenant n fois la valeur 0.
Par exemple,`creation(10)` renvoie `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

L'appartenance à un tableau se teste, comme pour les chaînes de caractères, avec l'opérateur `in`.
On pourra utiliser l'instruction Python `element in tableau`

Exemple 15 (Appartenance à un tableau)

```
>>> t1 = [0, 1, 2, 3]
>>> 0 in t1
True
>>> 5 in t1
False
```

VI. Les tableaux par compréhension

Activité 5.
On considère un tableau contenant des entiers défini de la façon suivante : `tableau_origine = [0, 1, 2, 3, 4, 5]`.

1. Indiquer un code Python permettant de renvoyer un nouveau tableau (`tableau_carre`) contenant le carré de chacun des nombres de `tableau_origine`.
2. Le code Python suivant réalise le même travail :

```
>>> tableau_origine = [0, 1, 2, 3, 4, 5]
>>> tableau_carre = [i**2 for i in tableau_origine]
```

   En quoi cette structure est-elle plus concise ?

Définition 2

Une instruction par compréhension se construit de la façon suivante :

- l'instruction est placée entre crochets `[...]`
- l'opération à réaliser
- le mot clé `for i in`
- le tableau à parcourir
- la condition

```
[instruction for i in tableau condition]
```

Exemple 16 (instruction qui renvoie le cube de tous les éléments présents dans table)

```
[i**3 for i in table]
```

Exemple 17 (instruction qui renvoie le cube de tous les éléments inférieurs à 10)

```
[i**3 for i in table if i < 10]
```

Exemple 18 (instruction qui affecte à nouvelle_table le cube de tous les éléments)

```
nouvelle_table = [i**3 for i in table if i < 10]
```

Activité 6.

1. Indiquer le code Python permettant d'initialiser par compréhension un tableau avec 10 valeurs.
2. Indiquer le code Python permettant d'initialiser par compréhension un tableau à deux dimensions avec 10 valeurs ([[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]).

Je retiens

- ▷ Un tableau est une collection d'éléments placés entre crochets, et séparés par une virgule.
- ▷ Un tableau est mutable (on dit aussi muable).
- ▷ L'instruction `len()` permet de connaître le nombre d'éléments (la longueur) d'un tableau.
- ▷ La longueur d'un tableau est fixe.
- ▷ L'initialisation d'un tableau se fait en utilisant une méthode par compréhension.
- ▷ Les n éléments d'un tableau sont indexés par les entiers $0, 1, 2, \ldots, n-1$. On accède à l'élément d'incide i par l'instruction `tableau[i]`.
- ▷ Un tableau est itérable, c'est à dire que l'on peut itérer sur chacun de ses éléments. On peut faire un parcours par index ou un parcours par élément.
- ▷ Les opérateurs + et * s'utilisent comme avec les chaînes de caractères et les tuples. Ce sont des opérateurs de concaténation. Le résultat est un tableau.
- ▷ L'appartenance d'un élément se teste avec le mot-clé `in`.

VII. Exercices

Exercice 1 (QCM).

1. Qu'affiche le programme python suivant ?

```
L=[3, 2, 2]
M=[1, 5, 1]
N=L + M
print(N)
```

- ☐ N
- ☐ $[4, 7, 3]$
- ☐ $[1, 5, 1, 3, 2, 2]$
- ☐ $[3, 2, 2, 1, 5, 1]$

2. Qu'affiche le programme python suivant ?

```
L=[0, 1, 2]
M=[3, 4, 5]
N=[L[i] + M[i] for i in range(len(L))]
print(N)
```

- ☐ $[3, 5, 7]$
- ☐ $[3, 5, 7, 9]$
- ☐ $[0, 1, 2, 3, 4, 5]$
- ☐ Erreur

3. On a saisi le code suivant :

```
liste = [1, [2, 7], 3, [4, 5, 8], 9]
```

Lequel des énoncés suivants est correct ?

- ☐ `print(liste[2]) #Affiche 2`
- ☐ `print(liste[0][0]) #Affiche 1`
- ☐ `print(liste[1][0]) #Affiche 2`
- ☐ `print(liste[3][1]) #Affiche 4`

Exercice 2.

1. On dispose d'un tableau `t = [15, 17, 12, 23]`. Quelle est la valeur de t après l'instruction `t[2] = 25` ?

2. On dispose d'un tableau t contenant `[[1,2,3],[4,5,6],[7,8,9]`. Quelle est la valeur de `t[1][2]` ?

3. Après l'instruction `t = [[i,i+1] for i in range (0,2,1)]`, quelle est la valeur de t ?

4. On construit une matrice (un tableau à n dimensions) avec le code suivant :

```
matrice = [3*[0] for i in range (3)]
for i in range (3):
    matrice[i][i] = i + 1
    matrice[0][i] = matrice[0][i] + i + 1
    matrice[i][2] = matrice[i][2] + i + 1
```

Quel est le résultat obtenu ?

Exercice 3 (Parcours par index ou tableau par compréhension).

1. On cherche à écrire une fonction *multiples_1*() qui prend en argument un entier n et renvoie un tableau des dix premiers multiples non nuls de n.
   Par exemple, `multiples\_1(5)` renvoie `[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]`.
   a. Écrivez le code Python de cette fonction en utilisant une solution par parcours d'index.
   b. Écrivez le code Python de cette fonction en utilisant une solution par compréhension
2. On cherche à écrire une fonction *multiples_2*(), prenant n en argument et renvoie un tableau des multiples non nuls de n strictement inférieurs à 100.
   Par exemple, `multiple\_2(20)` renvoie `[20, 40, 60, 80]`
   a. Écrivez le code Python de cette fonction en utilisant une solution par parcours d'index.
   b. Écrivez le code Python de cette fonction en utilisant une solution par compréhension
3. On cherche à écrire une fonction *produit*(), qui prend en argument un tableau *nombres* de nombres (entiers ou réels) et n un entier naturel non nul.
   Par exemple, `produit(nombres, 2)` lorsque nombres vait [4, 5, 6] renvoie `[8, 10, 12]`.
   a. Écrivez le code Python de cette fonction en utilisant une solution par parcours d'index.
   b. Écrivez le code Python de cette fonction en utilisant une solution par compréhension

Exercice 4 (Utilisation d'une fonction).
Écrire une fonction *map*(), qui étant donnés une fonction f et un tableau t en argument, renvoie la liste des f(x), où x décrit les éléments du tableau t. En particulier, le tableau renvoyé est de même longueur que le tableau entré en argument.
Exemple : `map(monome, nombres)` renvoie [5, 8, 11, 14] si nombres vaut `[1, 2, 3, 4]` et *monome* est défini par :

```
1 def monome(x) :
2     resultat = 3*x + 2
3     return resultat
```

1. Écrivez le code Python de cette fonction en utilisant une solution par parcours d'index.
2. Écrivez le code Python de cette fonction en utilisant une solution par compréhension