TD n°7 : Les types construits : tuples et tableaux	Thème 4 : Langages et Programmation
	COURS

Lien Capytale

1 Les séquences en Python

Séquences

nous avons parlé de **séquences d'instructions**, quand on exécute l'une après l'autre les instructions écrites sur des lignes successives en Python, par exemple. Ici, c'est différent. Les séquences dont on va parler sont des **types** particuliers, plus évolués que les types simples abordées jusque là comme les entiers, flottants, booléens...*

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

Rassurez-vous, nous reviendrons ci-dessous sur cette définition. Nous allons étudier plus particulièrement 2 types de séquences : les **tuples** et les **tableaux** (*il en existe d'autres que nous n'évoquerons pas ici*).

2 I. Les tuples en Python

Comme déjà dit ci-dessus, un tuple est une séquence. Voici un exemple très simple :

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

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

Dans le code ci-dessus, la variable mon_tuple référence un tuple, ce tuple est constitué des entiers 5, 8, 6 et 9. Comme indiqué dans la définition, chaque élément du tuple est indicé (il possède un indice) :

- **le premier élément** du tuple (l'entier 5) possède l'**indice 0**
- le deuxième élément du tuple (l'entier 8) possède l'indice 1
- le troisième élément du tuple (l'entier 6) possède l'indice 2
- le quatrième élément du tuple (l'entier 9) possède l'indice 3

Comment accéder à l'élément d'indice i dans un tuple qui correspond à mon $(i+1)^{ème}$ élément)?

Simplement en utilisant la «notation entre crochets» :

```
mon_tuple[i]
```

À vous 1

Testez le code suivant (tentez de répondre aux questions qui suivent avant !)

```
mon_tuple = (5, 8, 6, 9)
a = mon_tuple[2]
```

Quelle est la valeur référencée par la variable `a` ?

La variable `mon_tuple` référence le tuple `(5, 8, 6, 9)`, la variable `a` référence l'entier 6 car cet entier 6 est bien le troisième élément du tuple, il possède donc l'indice 2

ATTENTION :

dans les séquences **les indices commencent toujours à 0** (le premier élément de la séquence a pour indice 0), oublier cette particularité est une source d'erreur «classique».

À vous 2

Complétez le code ci-dessous (en remplaçant les ..) afin qu'après l'exécution de ce programme la variable `a` référence l'entier 8.

```
mon_tuple = (5, 8, 6, 9)
a = mon_tuple[..]
a
```

Un tuple ne contient pas forcément des nombres entiers, il peut aussi contenir des nombres décimaux, des chaînes de caractères, des booléens...

À vous 3

Quel est le résultat attendu après l'exécution du programme suivant ?
Vérifiez votre hypothèse en testant ce programme.

```
mon_tuple = ("le", "monde", "bonjour")
print(mon_tuple[2] + " " + mon_tuple[0] + " " + mon_tuple[1] + "!")
```

Grâce au tuple, une fonction peut renvoyer plusieurs valeurs :

À vous 4

Analysez puis testez le code suivant :

```
def add(a, b):
    c = a + b
    return (a, b, c)
mon_tuple = add(5, 8)
print(f"{mon_tuple[0]} + {mon_tuple[1]} = {mon_tuple[2]}")
```

Il faut bien comprendre dans l'exemple ci-dessus que la variable `mon_tuple` référence un tuple (puisque la fonction `add` renvoie un tuple), d'où la «notation entre crochets» utilisée avec `mon_tuple` (`mon_tuple[1]`...)

LORSQU'ON UTILISE UNE FONCTION QUI RENVOIE PLUSIEURS VALEURS :

```
def somme_et_produit(a, b):
    somme = a + b
    produit = a * b
    return somme, produit
```

```
somme_et_produit(5, 3)
```

```
(8, 15)
```

On voit que la fonction renvoie ici un 2-uplet mais qu'on peut récupérer les deux valeurs séparément :

```
s, p = somme_et_produit(5, 3)
print(f's a pour valeur {s} et p a pour valeur {p}')
```

```
s a pour valeur 8 et p a pour valeur 15
```

A retenir

Il n'est **pas possible de modifier un tuple après sa création** (on parle d'**objet immutable**), si vous essayez de modifier un tuple existant, l'interpréteur Python vous renverra une erreur.

```
mon_tuple = (5, 8, 6, 9)
mon_tuple[0]=3
mon_tuple
```

```
Traceback (most recent call last):
File "<input>", line 2, in <module>
TypeError: 'tuple' object does not support item assignment
```

3 II. Les tableaux en Python

ATTENTION :

Dans la suite nous allons employer le terme «**tableau**». Pour parler de ces «tableaux» les concepteurs de Python ont choisi d'utiliser le terme de «**list**» («liste» en français).
Pour éviter toute confusion, notamment par rapport à des notions qui seront abordées en terminale, le choix a été fait d'employer «tableau» à la place de «liste» (dans la documentation vous rencontrerez le terme «list», cela ne devra pas vous pertuber)

Les tableaux sont, comme les tuples, des séquences, mais à la différence des tuples, ils **sont modifiables** (on parle d'**objets «mutables»**).

Pour créer un tableau, il existe différentes méthodes : une de ces méthodes ressemble beaucoup à la création d'un tuple :

```
mon_tab = [5, 8, 6, 9]
mon_tab
```

Notez la présence des **crochets** à la place des parenthèses.

Un tableau est une séquence, il est donc possible de «récupérer» un élément d'un tableau à l'aide de son indice (de la même manière que pour un tuple).

3.1 Accéder à un élément d'un tableau à partir de son indice (l'indexation commence à zéro)

À vous 5

Quelle est la valeur référencée par la variable `ma_variable` après l'exécution du programme ci-dessous ?

```
mon_tab = [5, 8, 6, 9]
ma_variable = mon_tab[2]
ma_variable
```

N.B. Il est possible de saisir directement `mon_tab[2]` dans la cellule sans passer par l'intermédiaire de la variable `ma_variable`.

Il est possible de modifier un tableau à l'aide de la «notation entre crochets» :

À vous 6

Quel est le contenu du tableau référencé par la variable `mon_tab` après l'exécution du programme ci-dessous ? (Vérification en exécutant la cellule comme précédemment)

```
mon_tab = [5, 8, 6, 9]
mon_tab[2] = 15
mon_tab
```

Comme vous pouvez le constater avec l'exemple ci-dessus, l'élément d'indice 2 (le nombre entier 6) a bien été remplacé par le nombre entier 15.

Il est important de savoir qu'on peut numéroter «en partant de la fin». Le dernier élément du tableau correspond ainsi à l'indice -1, l'avant dernier élément à l'indice - 2 etc...

```
mon_tab[-1]
```

```
mon_tab[-3]
```

Ajout d'un élément dans un tableau

Il est aussi possible d'ajouter un élément en fin de tableau à l'aide de la méthode append :

À vous 7

Quel est le contenu du tableau référencé par la variable mon_tab après l'exécution du programme ci-dessous ? (Vérification sur place comme précédemment, on ne le répètera plus...)

```
mon_tab = [5, 8, 6, 9]
mon_tab.append(15)
mon_tab
```

[5, 8, 6, 9, 15]

Suppression d'un élément dans un tablau

L'**instruction** `del` permet de supprimer un élément d'un tableau en utilisant son index :

À vous 8

Quel est le contenu du tableau référencé par la variable `mon_tab` après l'exécution du programme ci-dessous ?

```
del mon_tab[1]
mon_tab
```

Longueur d'un tableau

La **fonction** `len` permet de connaitre le nombre d'éléments présents dans une séquence (tableau et tuple)

À vous 9

Quelle est la valeur référencée par la variable `nb_elements` après l'exécution du programme ci-dessous ?

```
mon_tab = [5, 8, 6, 9]
nb_elements = len(mon_tab)
nb_elements
```

4

Parcourir un tableau :

La boucle `for` permet de parcourir tous les éléments d'une *séquence*. En utilisant `range()` on a vu que l'on peut parcourir une séquence de nombres.

Quelques explications : comme son nom l'indique, la boucle `for` est une boucle ! Nous «sortirons» de la boucle une fois que tous les éléments du tableau mon_tab auront été parcourus. `element` est une variable qui va :

- au premier tour de boucle, référencer le premier élément du tableau (l'entier 5)
- au deuxième tour de boucle, référencer le deuxième élément du tableau (l'entier 8)
- au troisième tour de boucle, référencer le troisième élément du tableau (l'entier 6)
- au quatrième tour de boucle, référencer le quatrième élément de le tableau (l'entier 9)

Remarque : Une chose importante à bien comprendre : le choix du nom de la variable qui va référencer les éléments du tableau les uns après les autres (element) est totalement arbitraire, il est possible de choisir un autre nom sans aucun problème.

Il se trouve que les chaînes de caractères sont également des *séquences* ... de caractères. On peut donc aussi les parcourir très simplement avec une boucle `for`, de deux manières différentes.

3.1.1 Par ses éléments

On peut parcourir une chaîne directement par ses éléments :

```
mon_tab = [5, 8, 6, 9]
for nombre in mon_tab:
```

```
    print (nombre)
```

```
5
8
6
9
```

3.1.2 Par l'indice de ses éléments

On peut aussi utiliser la fonction range() pour parcourir les élémens par leurs indices.

```
for indice in range(len(mon_tab)):
    print(mon_tab[indice])
```

```
5
8
6
9
```

ATTENTION :

si vous avez dans un programme `range(a,b)`, `a` est la borne inférieure et `b` a borne supérieure. Vous ne devez surtout pas perdre de vu que **la borne inférieure est incluse**, mais que **la borne supérieure est exclue**.

Création d'un tableau

Il est possible d'utiliser la méthode `range` pour `remplir` un tableau :

À vous 10

Quel est le contenu du tableau référencé par la variable mon_tab après l'exécution du programme ci-dessous ?

```
mon_tab = []
for element in range(0, 9):
    mon_tab.append(element)
mon_tab
```

[0, 1, 2, 3, 4, 5, 6, 7, 8]

Créer un tableau par compréhension

Nous avons vu qu'il était possible de «remplir» un tableau en renseignant les éléments du tableau les uns après les autres :

```
mon_tab = [5, 8, 6, 9]
```

ou encore à l'aide de la méthode `append` (voir «À vous 10»).

Il est aussi possible d'obtenir exactement le même résultat qu'au «À vous 10» en une seule ligne grâce à la compréhension de tableau :

À vous 11

Quel est le contenu du tableau référencée par la variable mon_tab après l'exécution du programme ci-dessous ?

```
mon_tab = [p for p in range(0, 9)]
mon_tab
```

[0, 1, 2, 3, 4, 5, 6, 7, 8]

Les compréhensions de tableau permettent de rajouter une condition (`if`) :

À vous 12

Quel est le contenu du tableau référencé par la variable mon_tab après l'exécution du programme ci-dessous ?

```
tab1 = [1, 7, 9, 15, 5, 20, 10, 8]
mon_tab = [p for p in tab1 if p > 10]
mon_tab
```

[15, 20]

Autre possibilité, utiliser des composants «arithmétiques» :

À vous 13

Quel est le contenu du tableau référencé par la variable `mon_tab` après l'exécution du programme ci-dessous ?

```
tab1 = [1, 7, 9, 15, 5, 20, 10, 8]
mon_tab = [p**2 for p in tab1 if p < 10]
mon_tab
```

[1, 49, 81, 25, 64]

Rappel : `p**2` permet d'obtenir la valeur de `p` élevée au carrée (p^2).

Comme vous pouvez le remarquer, nous obtenons un tableau `mon_tab` qui contient tous les éléments du tableau `tab1` élevés au carré à condition que ces éléments de `tab1` soient inférieurs à 10. Comme vous pouvez le constater, la compréhension de tableau permet d'obtenir des combinaisons relativement complexes.

4 III. Travailler sur des "tableaux de tableaux"

Chaque élément d'un tableau peut être un tableau, on parle de tableau de tableau.

Voici un exemple de tableau de tableau :

```
m = [[1, 3, 4], [5 ,6 ,8], [2, 1, 3], [7, 8, 15]]
```

Le premier élément du tableau ci-dessus est bien un tableau (`[1, 3, 4]`), le deuxième élément est aussi un tableau (`[5, 6, 8]`)...

Il est souvent plus pratique de présenter ces «tableaux de tableaux» comme suit :

```
m = [[1, 3, 4],
     [5, 6, 8],
     [2, 1, 3],
     [7, 8, 15]]
```

Nous obtenons ainsi quelque chose qui ressemble beaucoup à un "objet mathématique" très utilisé : une **matrice**.

Il est évidemment possible d'utiliser les indices de position avec ces «tableaux de tableaux». Pour cela nous allons considérer notre tableau de tableaux comme une matrice, c'est à dire en utilisant les notions de «ligne» et de «colonne». Dans la matrice ci-dessus :

En ce qui concerne les lignes :

- 1, 3, 4 constituent la première ligne
- 5, 6, 8 constituent la deuxième ligne
- 2, 1, 3 constituent la troisième ligne
- 7, 8, 15 constituent la quatrième ligne

En ce qui concerne les colonnes :

- 1, 5, 2, 7 constituent la première colonne
- 3, 6, 1, 8 constituent la deuxième colonne
- 4, 8, 3, 15 constituent la troisième colonne

Pour cibler un élément particulier de la matrice, on utilise la notation avec «doubles crochets» : `m[ligne][colonne]` (sans perdre de vu que **la première ligne et la première colonne ont pour indice 0**)

À vous 14

Quelle est la valeur référencée par la variable `a` après l'exécution du programme ci-dessous ?

```
m = [[1, 3, 4],
    [5, 6, 8],
    [2, 1, 3],
    [7, 8, 15]]

a = m[1][2]
a
```

8

Comme vous pouvez le constater, la variable a référence bien l'entier situé à la 2e ligne (indice 1) et à la 3e colonne (indice 2), c'est-à-dire 8.

À vous 15

Quel est le contenu du tableau référencé par la variable `mm` après l'exécution du programme ci-dessous (*si nécessaire, affichez d'abord la valeur de `mm` sans exécuter la dernière ligne, pour bien comprendre*) ?

```
m = [1, 2, 3]
mm = [m, m, m]
print(mm)
m[0] = 100
mm
```

```
[[100, 2, 3], [100, 2, 3], [100, 2, 3]]
```

Comme vous pouvez le constater, **la modification du tableau référencé par la variable `m` entraine la modification du tableau référencé par la variable `mm`** (alors que nous n'avons pas directement modifié le tableau référencé par `mm`). Il faut donc être très prudent lors de ce genre de manipulation afin d'éviter des modifications non désirées.

Il est possible de parcourir l'ensemble des éléments d'une matrice à l'aide d'une «double boucle `for`» :

À vous 16

Analysez puis testez le code suivant :

```
m = [[1, 3, 4],
    [5, 6, 8],
    [2, 1, 3],
    [7, 8, 15]]

nb_colonne = 3
nb_ligne = 4

for i in range(0, nb_ligne):
    for j in range(0, nb_colonne):
```

```
        a = m[i][j]
        print(a)
```