

## Chapitre 9 : Les tuples

### I. La notion de mutabilité:

Vous avez rencontré jusqu'à présent une séquence Python - la liste. La liste est un exemple classique d'une séquence Python, bien qu'il existe d'autres séquences qui méritent d'être mentionnées, et nous allons vous les présenter maintenant.

Mais avant d'en parler il faut savoir la notion de la **mutabilité**.

la **mutabilité** - est une propriété de n'importe quelle donnée de Python qui décrit sa disponibilité à être librement modifiée pendant l'exécution du programme. Il existe deux types de données Python: **mutables** et **immuables** :

exemple : `list.append(1)`

**Les données immuables ne peuvent pas être modifiées de cette manière .**

Imaginez qu'une liste ne peut être attribuée et relue. Vous ne pourrez ni lui ajouter un élément, ni en supprimer aucun élément. Cela signifie que l'ajout d'un élément à la fin de la liste nécessiterait la recreation de la liste à partir de zéro.

Vous devrez créer une liste complètement nouvelle, composée de tous les éléments de la liste déjà existante, plus le nouvel élément.

Le type de données dont nous voulons vous parler maintenant est un **tuple** . **Un tuple est un type de séquence immuable** . Il peut se comporter comme une liste, mais il ne doit pas être modifié.

### II. Les tuples :

La première et la distinction la plus claire entre les listes et les tuples est la syntaxe utilisée pour les créer - les **tuples préfèrent utiliser des parenthèses** , tandis que les listes aiment voir les crochets, bien qu'il soit également **possible**

de créer un tuple uniquement à partir d'un ensemble de valeurs séparées par des virgules .

```
tuple1 = (1, 2, 4, 8)
tuple2 = 1., .5, .25, .125
```

**Remarque:** chaque élément de tuple peut être d'un type différent (virgule flottante, entier ou tout autre type de données non encore introduit).

```
tuple1 = (1, 'n', 4, 0.3)
tuple2 = 1., .5, .25, .125
print(tuple1)
|
```

```
==== RESTART: C:/Users:
(1, 'n', 4, 0.3)
```

- **Comment créer une tuple :**

Il est possible de créer un tuple vide - des parenthèses sont alors nécessaires:

```
emptyTuple = ()
```



Si vous souhaitez créer un **tuple à un élément** , vous devez prendre en considération le fait que, pour des raisons de syntaxe (un tuple doit être distingué d'une valeur simple ordinaire), vous devez terminer la valeur par une virgule:

```
oneElementTuple1 = (1, )
oneElementTuple2 = 1.,
```

La suppression des virgules ne gâchera pas le programme dans un sens syntaxique, mais vous obtiendrez à la place une variable simple, pas une tuples.

- **Comment utiliser une tuple :**

Si vous souhaitez obtenir les éléments d'un tuple afin de les relire, vous pouvez utiliser les mêmes conventions auxquelles vous êtes habitué lors de l'utilisation des listes.

<pre> myTuple = (1, 10, 100, 1000) print(myTuple[0]) print(myTuple[-1]) print(myTuple[1:]) print(myTuple[:-2]) </pre>		<pre> ==== RESTART: C:/Use 1 1000 (10, 100, 1000) (1, 10) </pre>
<pre> myTuple = (1, 10, 100, 1000)  #Méthode 1 : for ele in myTuple :     print(ele) print("#####") #Méthode 2 : for i in range(len(myTuple)):     print(myTuple[i]) </pre>		<pre> ==== RESTART: C:/Users/TOSI 1 10 100 1000 ##### 1 10 100 1000 </pre>

**Remarque :** Les similitudes peuvent être trompeuses - **n'essayez pas de modifier le contenu d'un tuple !** Ce n'est pas une liste

```

myTuple = (1, 10, 100, 1000)

myTuple[0] = 5

print(myTuple)

```

Résultat :

```

==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/
Traceback (most recent call last):
  File "C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python
in <module>
    myTuple[0] = 5
TypeError: 'tuple' object does not support item assignment

```

Et je peux pas ajouter un élément à la tuple ni supprimer un élément de la tuple

```

myTuple = (1, 10, 100, 1000)

myTuple.append(5 )

del myTuple[1]

print(myTuple)

```

```

Traceback (most recent call last):
  File "C:/Users/TOSHIBA/AppData/Local/Programs/Python/Py
in <module>
    myTuple.append(5 )
AttributeError: 'tuple' object has no attribute 'append'

```

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/
Traceback (most recent call last):
  File "C:/Users/TOSHIBA/AppData/Local/Programs/Python/Pytho
in <module>
    del myTuple[1]
TypeError: 'tuple' object doesn't support item deletion
```

Qu'est-ce que les tuples peuvent faire d'autre pour vous?

- la **len()** fonction accepte les tuples et retourne le nombre d'éléments contenus à l'intérieur;
- l'« + »opérateur peut joindre des tuples ensemble (nous vous l'avons déjà montré)
- l'« \* »opérateur peut multiplier les tuples, tout comme les listes;
- les opérateurs in et not in fonctionnent de la même manière que dans les listes.

```
myTuple = (1, 10, 100)
t1 = myTuple + (1000, 10000)
t2 = myTuple * 3
print(len(t2))
print(t1)
print(t2)
print(10 in myTuple)
print(-10 not in myTuple)
```

```
==== RESTART: C:/Users/TOSHIBA/AppDat
9
(1, 10, 100, 1000, 10000)
(1, 10, 100, 1, 10, 100, 1, 10, 100)
True
True
```

L'une des propriétés de tuple les plus utiles est leur capacité à **apparaître sur le côté gauche de l'opérateur d'affectation** pour échanger les valeurs des tuples (cela ça marche aussi avec les listes)

```
var = 6
t1 = (1, )
t2 = (2, )
t3 = (3, var)
t1, t2, t3 = t2, t3, t1
print(t1, t2, t3)
```

```
==== RESTART: C:/Users/TOSHIBA,
(2,) (3, 6) (1,)
```

- **La fonction tuple() :**

Vous pouvez également créer un tuple à l'aide d'une fonction intégrée Python appelée tuple(). Ceci est particulièrement utile lorsque vous souhaitez convertir

un certain itérable (par exemple, une liste, une plage, une chaîne, etc.) en un tuple :

```
myTup = tuple((1, 2, "string"))
print(myTup)
lst = [2, 4, 6]
print(type(lst))
tup = tuple(lst)
print(tup)
```

```
==== RESTART: C:/Users/TOSI
(1, 2, 'string')
<class 'list'>
(2, 4, 6)
```

De la même manière, lorsque vous souhaitez convertir un itérable en liste, vous pouvez utiliser une fonction intégrée Python appelée `list()` :

### III. Exercices :

#### Exercice 1 :

Quelle est la sortie de l'extrait de code suivant?

```
tup = 1, 2, 3
a, b, c = tup
print(a * b * c)
```

#### Exercice 2 :

Ecrire un programme qui permet de :

- 1- Lire une tuple à 2 dimensions de L lignes et C colonnes
- 2- Afficher ensuite cette matrice ligne par ligne.
- 3- Calculer et afficher la somme des éléments de la diagonale principale
- 4- Afficher un message qui précise si elle est triangulaire supérieure ou pas.  
Une matrice est triangulaire supérieure si toutes les valeurs au-dessous de la première diagonale sont égales à zéro.

**Exemple de matrice triangulaire supérieure 4 x 4 :**

5	3	-61	9
0	2	11	4
0	0	15	-7
0	0	0	3

Dans cet exemple, les éléments au-dessous de la première diagonale sont tous égaux à zéro et donc, c'est une matrice triangulaire supérieure.

### Exercice 3 :

Ecrire un programme en python qui permet d'échanger la dernière occurrence d'une valeur minimale d'une tuple avec le premier élément de la tuple

Exemple : |**5**| 6 | 1 | 6 | 1 | 6 | **1** | => | **1** | 6 | 1 | 6 | 1 | 6 | **5** |