

## PARTIE 1 : Types construits

Nous avons manipulé des variables avec des types simples : int, bool, float, str. Ce sont des conteneurs ne contenant qu'une valeur.

Nous avons manipulé aussi, sans vraiment l'étudier, des types construits (ou types abstraits) : liste, tuple. Ces types de données s'appellent des séquences.

Dans ce cours nous allons étudier trois types de séquence : les p-uplets, les tableaux et les dictionnaires. En python, les tableaux s'appellent les listes et les p-uplets s'appellent les tuples.



### DÉFINITION 1 : Séquence

Famille indexée d'éléments par les entiers strictement positifs inférieurs ou égaux à un certain entier, ce dernier étant appelé « longueur » de la séquence.

En informatique, une liste est une structure de données représentant une séquence finie d'éléments auxquels on peut accéder efficacement par leur position, ou indice, dans la séquence.

En Python, une séquence est une collection ordonnée d'objets qui permet d'accéder à ses éléments par leur numéro de position dans la séquence. Les listes et les tuples sont des séquences.

## 1. Les types construits : les p-uplets – tuples en Python



### DÉFINITION 2 : p-uplet

Un p-uplet est une séquence **immutable**. C'est à dire une suite indexée de valeurs (de n'importe quel type) que l'on ne peut pas modifier.

### 1.1. Construire les objets tuples



#### PROPRIÉTÉ 1 :

Un tuple est une séquence d'éléments entre parenthèses déparés par des virgules.

✓ EXEMPLE 1 :

```
tuple1 = ()      # Tuple vide
tuple2 = (3,)    # Tuple avec un seul element
tuple3 = (1,4,5,6,2,7)  # Tuple de 6 elements numerotes de 0 a 5
tuple4 = (1,'a','voiture')  # Tuple d'elements de differents types
type(tuple1)    # permet d'accéder au type de l'objet
```

*.\scripts\part1\exemple1.py*

```
*** Console de processus distant Réinitialisée ***
>>> type(tuple4)
<class 'tuple'>
>>> type(tuple3)
<class 'tuple'>
```

*Remarque* : Le mot tuple étant le nom d'une fonction interne, vous devez éviter de l'utiliser comme nom de variable.

**PROPRIÉTÉ 2 : Opérations simples sur les tuples**

Différentes opérations de base sont disponibles pour les tuples :

- La **concaténation**. Deux tuples peuvent être collés l'un à l'autre avec la commande +
- La **reproduction** de tuples. Un tuple peut être "multiplié" avec la commande \*



EXEMPLE 2 :

```
>>> tuple1 = (1,2,3)
>>> tuple2 = (5,6)
>>> tuple1 + tuple2          # concatenation de deux tuples
(1, 2, 3, 5, 6)
>>>
>>> tuple4 = tuple1 * 3      # "multiplication d'un tuple"
>>> tuple4
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>>
>>> print(* tuple4)         # Dispersion d'un tuple
1 2 3 1 2 3 1 2 3
```

*.\scripts\part1\exemple2.png*

**PROPRIÉTÉ 3 : Indices des tuples**

Soit *i* un entier et **monTuple** un tuple. On accède à un élément du tuple en écrivant : **monTuple[i]**

```
>>> monTuple = (5,1,3,0,2)
>>> monTuple [0]          # Acces au premier element du tuple
5
>>> monTuple [3]          # Acces au 4eme element du tuple (indice 3)
0
>>> monTuple [-1]         # Acces au dernier element du tuple
2
>>> monTuple [1:4]        # slicing: Affiche de monTuple [1] à monTuple [3]
(1, 3, 0)
>>> monTuple [2:]         # slicing: Affiche de monTuple [2] à la fin
(3, 0, 2)
>>> monTuple [:2]         # slicing: Affiche du debut a monTuple [1]
(5, 1)
```



**Exercice 1 :** Créer une fonction nommée `coorVecteur()` prenant 4 paramètres *xA*, *yA*, *xB*, *yB* et renvoyant sous forme de tuple les coordonnées du vecteur  $\overrightarrow{AB}$

```
>>> coorVecteur(2,1,5,8)
(3, 7)          # En effet x_AB=5 -2=3 et y_AB=8 -1=7
```

*.\scripts\part1\exo1.py*



**Exercice 2 :** En utilisant la fonction, `coorVecteur()` créer une fonction `multVecteurk()` créant les coordonnées du vecteur *k*

```
>>> multVecteurk(5,2,1,5,8)
(15, 35)        # car x_AB=3 et y_AB=7
```

*.\scripts\part1\exo2.py*

## 1.2. Itérer un tuple



**PROPRIÉTÉ 4 :** Un tuple est itérable. Cela signifie que l'on peut organiser une itération (boucle for) sur cette structure.



EXEMPLE

3 :

```
monTuple = (2,4,5,8,6)      # Initialisation du tuple
n = len(monTuple)           # n est la longueur du tuple, ici n=5
for i in range(n):          # Balayage du tuple de i=0 à i=4
    print(monTuple[i], end=" ") # Affichage des elements du tuple
print()                     # Retour à la ligne

for elt in monTuple:        # Balayage du tuple par des elements
    print(elt, end=" ")     # Affichage des elements du tuple9 >>>
```

\*\*\* Console de processus distant Réinitialisée \*\*\*

```
2 4 5 8 6
2 4 5 8 6
```

Dans les deux cas, le résultat est le même

🔥 **Exercice 3 :** Créer une fonction `affichageTuple()` prenant comme paramètre un tuple et affiche sous forme de tuple chaque indice et sa valeur correspondante. On utilisera des boucles pour balayer le tuple.

```
>>> monTuple = (4,3,6)
>>> affichageTuple(monTuple)
0 4
1 3
2 6
```

.\scripts\part1\exo3A.py

Il est possible de récupérer la longueur d'un tuple avec l'instruction suivante :

```
>>> monTuple = (1,3,4,2,8)
>>> len(monTuple)
5
```

.\scripts\part1\exo3B.png

Dans cet exemple, le tuple a une longueur de 5 avec des indices allant de 0 à 4.

🔥 **Exercice 4 :** Sans utiliser l'instruction `len()`, créer une fonction `longueurTuple()` qui renvoie la longueur d'un tuple.


```
>>> monTuple = (1,3,4,2,8)
>>> longueurTuple(monTuple)
5
```

.\scripts\part1\exo4.py

🔥 **Exercice 5 :** Créer une fonction `affichagePair()` qui affiche uniquement les éléments pairs du tuple.

```
>>> monTuple = (1,3,4,2,8,7,1,6)
>>> affichagePair(monTuple)
4 2 8 6
```

.\scripts\part1\exo5.py


 **Exercice 6 : Créer une fonction `verifElement()` qui vérifiera si une valeur appartient ou non à un tuple**

```
>>> monTuple = (2,4,1,8,7)
>>> verifElement(4,monTuple)
1                                     # 4 est bien dans monTuple. On affiche l'indice trouve
>>> verifElement(5,monTuple)
5 False                             # 5 n'appartient pas à monTuple
```

.\scripts\part1\exo6.py

### 1.3. Opération et méthodes communes aux listes et aux tuples

Opérations ou méthodes	Description
<code>x in L</code>	Renvoie True si un élément de L est égale à x, False sinon
<code>x not in L</code>	Renvoie True si aucun un élément de L n'est égale à x, False sinon
<code>len(L)</code>	Renvoie le nombre d'éléments de L
<code>L == L1</code>	Renvoie True si L et L1 sont de même type, ont la même longueur, et ont des éléments égaux deux à deux.
<code>L[i]</code>	Renvoie l'élément d'indice i de L. Le premier élément a pour indice 0.
<code>L[i:j]</code>	Renvoie une partie de l'indice i à j non inclus
<code>L.index(x)</code>	Renvoie l'indice de la première apparition de x dans L
<code>L.count(x)</code>	Renvoie le nombre d'apparitions de x dans L
<code>L+L1</code>	Renvoie une nouvelle séquence concaténation de L et L1.
<code>L*n</code>	Renvoie une nouvelle séquence composée de la concaténation de L avec lui-même n fois.

 **Exercice 7 : Tester toutes ces opérations et méthodes avec :**

```
1 >>> L = (5,4,8,9,1,4)
2 >>> L1 = (4,3,1)
3 >>> x = 9
4 >>> n = 3
```

.\scripts\part1\exo7.py

🔥 Exercice 8 : Créer une fonction `maxiTuple()` qui prendra comme paramètre un p-uplet et qui renverra la valeur de l'élément le plus grand.

```
>>> monTuple = (1,0,-15,8,-6,1,6,4,-2,3,1)
>>> maxiTuple(monTuple)
8
```

.\\scripts\\part1\\exo8.py

🔥 Exercice 9 : Créer une fonction `comptage()` qui prendra comme paramètre un p-uplet et une valeur et qui comptera le nombre d'éléments de cette valeur dans le p-uplet.

```
>>> monTuple = (1,0,-15,8,-6,1,6,4,-2,3,1)
>>> comptage(monTuple,1)
3
```

.\\scripts\\part1\\exo9.py

🔥 Exercice 10 : Créer une fonction `sommePlus()` qui prendra comme paramètre un p-uplet et qui renverra la somme des valeurs strictement positives.

```
>>> monTuple = (1,0,-15,8,-6,1,6,4,-2,3,1)
>>> sommePlus(monTuple)
24
```

.\\scripts\\part1\\exo10.py

🔥 Exercice 11 : Modifier le programme de l'exercice précédent afin de ne renvoyer que la somme des valeurs paires positives. Nommer cette fonction `sommePlus2()`

```
>>> monTuple = (1,0,-15,8,-6,1,6,4,-2,3,1)
>>> sommePlus2(monTuple)
18
```

.\\scripts\\part1\\exo11.py