Informatique	Programmation

Langage Python
Listes et Tuples

En langage Python, les chaînes de caractères, les listes et les tuples sont **trois types de *séquence*.**

1. Les listes

Une liste est une collection ordonnée et modifiable d'éléments

Ces éléments peuvent êtres tous de même type : liste d'entiers, liste de réels, liste de chaîne,...
Mais la liste peut aussi être hétérogène et contenir des éléments de différents types.
Il est possible aussi de construire des listes de listes (chaque élément est une liste).

1.1. *Déclarer une liste :*

Les éléments sont séparée par des virgules et entourés par des crochets.

```
>>> ville = ['Belfort', 'Besancon', 'Dole', 'Vesoul']      # liste de str
>>> population = [50e3, 117e3, 24e3, 15e3]                 # liste de float
>>> superficie = [1710, 6505, 3838, 907]                   # liste de int
```

Il est possible de déclarer une liste vide pour la remplir ultérieurement.

```
>>> resultats = []          # pas encore de résultats
```

La fonction *len()* permet de connaître la longueur d'une liste (nombre d'éléments).

```
>>> len(villes)
4
>>> len(resultats)
0
```

1.2. *Accéder aux éléments de la liste*

➢ Un indice (ou index) permet d'accéder individuellement à chaque élément de la liste.

L'indice est un nombre entier placé entre crochets
Le premier élément de la liste possède l'indice zéro

```
>>> print(ville[0])          # premier élément de la liste
Belfort
```

➢ **Opérations de lecture ou d'écriture :**

```
>>> uneVille = ville[1]      # lecture dans la liste (copie de l'élément)
>>> ville[1] = 'BESANCON'    # écriture dans la liste (remplacement de l'élément)
```

➢ **Accéder au dernier élément de la liste :**

```
>>> print(ville[-1])          # indice -1
Vesoul
```

➢ **Accès à une tranche de la liste :**

[n:p+1] représente une tranche de liste

de l'élément de rang n inclus à l'élément de rang p inclus

```
>>> 2villes = ville[1:3]      # tranche du 2ème au 3ème élément
>>> print(2villes)
['Besancon', 'Dole']
```

La notation [n:] permet d'extraire une tranche de liste de l'élément de rang n inclus jusqu'à la fin.

```
>>> 3villes = ville[1:]       # tranche du 2ème au dernier élément
>>> print(3villes)
['Besancon', 'Dole', 'Vesoul']
```

La notation [:p+1] permet d'extraire une tranche de liste du début à l'élément de rang p inclus.

```
>>> 3villes = ville[:3]       # tranche du 1er au 3ème élément
>>> print(3villes)
['Belfort', 'Besancon', 'Dole']
```

1.3. *Concaténation de listes ou ajout d'un élément :*

```
>>> autresVilles = ['Morteau', 'Montbeliard']
>>> ville = ville + autresVilles
>>> len(villes)
6
```

+ est l'opérateur de concaténation

➢ **Autre manière d'ajouter un élément dans la liste :**

Une liste est un *objet* (au sens logiciel du terme) possédant des *méthodes* (fonctions permettant d'agir sur cet objet). L'objet *list* possède une méthode *append()* permettant d'ajouter un élément.

```
>>> ville.append('Pontarlier')
```

Attention, ne mettez pas une liste dans les parenthèses de append() !

```
>>> autresVilles = ['Lure', 'Luxeuil']
>>> ville.append(autresVilles)
>>> print(ville)
['Belfort', 'Besancon', 'Dole', 'Vesoul', ['Lure', 'Luxeuil']]
>>> len(ville)
5
```

La liste contient 5 éléments et non 6 ! Le dernier élément de la liste est une liste de deux éléments.

La solution consiste à utiliser la méthode *extend().*

```
>>> ville.extend(autresVilles)          # extend() étend la liste
>>> print(ville)
['Belfort', 'Besancon', 'Dole', 'Vesoul', 'Lure', 'Luxeuil']
```

➢ **Insérer un élément dans la liste :**

Utiliser la fonction *insert() en indiquant en premier argument le rang où l'élément doit être inséré.*

```
>>> ville.insert(0, 'Morteau')          # 0 : insérer au début
['Morteau', 'Belfort', 'Besancon', 'Dole', 'Vesoul']
```

1.4. Supprimer un élément ou vider la liste :

La méthode *pop()* permet de supprimer le dernier élément :

```
>>> derniereVille = ville.pop()          #derniereVille contient l'élément supprimé
```

L'instruction del permet du supprimer un élément quelconque ou une tranche de la liste.

Supprimer l'élément de rang n :

```
>>> del(ville[n])
```

Supprimer une tranche de la liste :

```
>>> del(ville[1:3])    # supprime les éléments de rang 1 et 2
>>> ville
['Belfort', 'Vesoul']
```

Supprimer une tranche de la liste à la fin :

```
>>> del(ville[2:])     # supprime les éléments à partir du rang 2
>>> ville
['Belfort', 'Besancon']
```

Supprimer une tranche de la liste au début :

```
>>> del(ville[:2])     # supprime les éléments avant le rang 2
>>> ville
['Dole', 'Vesoul']
```

Vider la liste :

```
>>> del(ville[:])
```

[:] représente tous les éléments, du premier au dernier.

Supprimer l'élément dont on connaît la valeur (première occurrence) :

```
>>> ville.remove('Dole')
```

1.5. Copier une liste :

Deux manières de réaliser la copie d'une liste :

```
>>> copie = ville[:]
>>> copie = list(ville)
```

Attention : la ligne suivante ne crée pas une copie de la liste mais une référence.

```
>>> laliste = ville    # modification de ville => modification de laliste
```

1.6. Modifier l'ordre des éléments dans la liste :

Trier la liste :

```
>>> ville.sort()
```

Inverser l'ordre des éléments de la liste :

```
>>> ville.reverse()
```

1.7. Rechercher un élément dans la liste :

Identifier l'indice de la première occurrence d'un élément présent dans la liste, et de valeur connue :

```
>>> ville.index('Dole')
2
>>> ville.index('Dijon')
Traceback (innermost last):
  File "<stdin>", line 1, in <module>
ValueError: 'Dijon' is not in list
```

Si l'objet recherché n'est pas présent dans la liste, la méthode *index()* provoque une erreur !

➢ **Deux solutions pour vérifier si un objet est présent dans la liste :**

L'instruction in renvoie True si l'élément est présent dans la liste, sinon elle renvoie False.

```
>>> 'Dole' in ville
True
>>> 'Dijon' in ville
False
```

La méthode count() renvoie le nombre d'occurences d'un élément dans la liste.

Elle renvoie 0 si l'élément n'est pas présent.

```
>>> ville.count('Dole')
1
>>> ville.count('Dijon')
0
```

2. Les tuples

Un tuple est une collection ordonnée et non modifiable d'éléments
Utilisé pour contenir des constantes

Un tuple se déclare comme une liste, avec des parenthèses à la place des crochets :

```
>>> unTuple = (2, 4, 8, 16)
```

Un tuple s'utilise comme une liste, opérations de modification exclues.

```
>>> premierElement = unTuple[0]
>>> unTuple.index(4)
1
```

➢ **Avantages des tuples sur les listes :**

Le parcours d'un tuple est plus rapide	**Un tuple consomme moins de mémoire**