Structures de données en Python

1) Les chaînes de caractères

Une chaîne de caractères est une suite de caractères (voir le cours « premiers pas en Python » pour la manipulation des chaines de caractères).

On peut ajouter ou supprimer des caractères à une chaîne de caractères mais les caractères ne sont pas modifiables. Autrement dit, on peut supprimer un « A » dans une chaîne puis insérer un « B » à la même position, mais on ne peut pas transformer un « A » en « B ».

2) Les listes

Une liste est une **collection ordonnée d'objets** (de préférence de même nature).

Syntaxes possibles pour **déclarer une liste** :

```
maListeVide = []
maListeAvecUnElement = [element]
maListe = [element1,element2,element3]
```

On peut ajouter des éléments après la création de la liste avec la méthode **append** :

```
maListe=["de Gaulle","Pompidou","Giscard","Mitterand"]
>>> maListe.append("Chirac")
>>> print(maListe)
["de Gaulle","Pompidou","Giscard","Mitterand","Chirac"]
```

On peut ajouter une liste à une liste avec la méthode **extend** :

```
>>> liste1=[1,2]
>>> liste2=[3,4]
>>> liste1.extend(liste2)
>>> print(liste1)
[1, 2, 3, 4]
```

On peut insérer un élément à une position donnée avec la méthode **insert** :

```
maListe.insert(2,x)
```

Ici on insert l'élément x en 3e position (la liste démarre en index 0)

La fonction **len** permet d'obtenir le nombre d'élément de la liste

```
longueur = len(maListe)
```

On peut accéder à un élément par son index :

```
maListe=["a","b","c"]
>>> liste[1]
'b'
```

On peut accéder au nombre d'élément ayant la même valeur dans une liste avec la méthode **count** :

```
maListe=[1,5,4,6,8,4,9,7,6,4,3,1,0,5,9]
>>> maListe.count(4)
3
```

On peut accéder à l'index du premier élément donné trouvé dans une liste avec la méthode **index** :

```
maListe=[1,5,4,6,8,4,9,7,6,4,3,1,0,5,9]
>>> maListe.index(4)
2
```

On peut effacer une liste avec **clear** :

```
>>> maListe=[1,5,4,6,8,4,9,7,6,4,3,1,0,5,9]
>>> maListe.clear()
>>> print(maListe)
[]
>>>
```

On peut supprimer un élément d'une liste à partir de son index avec la méthode **pop** :

```
>>> maListe=[1,5,4,6,8,4,9,7,6,4,3,1,0,5,9]
>>> element=maListe.pop(1)
>>> print(maListe)
[1,4,6,8,4,9,7,6,4,3,1,0,5,9]
>>> print(element)
5
>>>
```

Important : la méthode pop supprime l'élément de la liste et le renvoie en valeur de retour.

On peut supprimer un élément d'une liste à partir de sa valeur avec la méthode **remove** :

```
>>> maListe=[1,5,4,6,8,4,9,7,6,4,3,1,0,5,9]
>>> maListe.remove(6)
>>> print(maListe)
[1,5,4,8,4,9,7,6,4,3,1,0,5,9]
>>>
```

Supprime le premier élément de la liste dont la valeur est passée en argument (ici le 6)

Générateur d'une liste composée d'une suite arithmétique avec **range** :

```
>>> list(range(10))
[0,1,2,3,4,5,6,7,8,9]
```

Trier une liste avec la méthode **sort** , inverser une liste avec la méthode **reverse** :

```
>>> maListe=[1,5,4,6,8,4,9,7,6,4,3,1,0,5,9]
>>> maListe.sort()
>>> print(maListe)
[0, 1, 1, 3, 4, 4, 4, 5, 5, 6, 6, 7, 8, 9, 9]
>>> maListe.reverse()
>>> print(maListe)
[9, 9, 8, 7, 6, 6, 5, 5, 4, 4, 4, 3, 1, 1, 0]
```

Avec sort ou reverse, la liste est modifiée.

Trier une liste avec la fonction **sorted** :

```
>>> maListe=[1,5,4,6,8,4,9,7,6,4,3,1,0,5,9]
>>> print(sorted(maListe))
[0, 1, 1, 3, 4, 4, 4, 5, 5, 6, 6, 7, 8, 9, 9]
>>> print(maListe)
[1, 5, 4, 6, 8, 4, 9, 7, 6, 4, 3, 1, 0, 5, 9]
>>>
```

Avec sorted, la liste est copiée mais n'est pas modifiée : sorted(maListe) est une nouvelle liste.

Remarque importante sur les listes Python

Python autorise les listes d'objets de types différents et hétéroclites. Un objet d'une liste peut lui-même être une liste. Dans la pratique, une liste (ou un tableau) est généralement constitué d'objets de même type : liste de nombres, liste de noms, liste de tuples etc…

3) La copie de listes

En Python, la copie de listes avec la syntaxe naturelle pose problème :

```
maListe2 = maListe1
```

Les deux noms pointent vers le même emplacement mémoire et il n'y aura donc en réalité qu'une seule liste existante. Toute modification sur une des deux listes entraînera la même modification sur l'autre liste :

```
>>> liste1=[1,2]
>>> liste2=liste1
>>> liste2[0]=3
>>> print(liste1)
[3, 2]
```

Il existe plusieurs méthodes pour copier réellement des listes :

Méthode 1 - Copier les éléments individuellement de la liste sources à la liste cible.

```
list2 = [x for x in list1]
```

```
>>> liste1=[1,2]
>>> liste2 = [x for x in liste1]
>>> liste2[0]=3
>>> print(liste1)
[1, 2]
```

ou

```
list2 = [] + list1
```

```
>>> liste1=[1,2]
>>> liste2 = [] + liste1
>>> liste2[0]=3
>>> print(liste1)
[1, 2]
```

Méthode 2 – Utiliser la fonction deepcopy du module copy :

```
import copy
List2 = copy.deepcopy(List1)
```

```
>>> import copy
>>> liste2 = copy.deepcopy(liste1)
>>> liste2[0]=3
>>> print(liste1)
[1, 2]
```

4) Les tuples

Un tuple (ou p-uplet) est, comme une liste, une **séquence d'éléments** qui peuvent être de type différents mais qui, comme les chaînes de caractères, ne sont **pas modifiables**.

Syntaxes possibles pour déclarer un t-uple :

```
monTupleVide = ()
monTupleAvecUnElement = (element,)
monTupleAvecUnElement = element,
monTuple = (element1,element2,element3)
monTuple = monTuple + (element4,)
```

Pour déclarer un tuple, les parenthèses ne sont pas obligatoires mais les virgules le sont.

Exemple :

```
>>> monTuple=(1,2,3)
>>> print(monTuple)
(1, 2, 3)
>>> monTuple=monTuple+(1,)
>>> print(monTuple)
(1, 2, 3, 1)
>>> print(monTuple[2])
3
>>>
```

Les opérations sur les tuples sont similaires à celles que l'on peut effectuer sur les listes, à l'exclusion des opérations de modification, puisque les tuples ne sont pas modifiables (append, sort, remove, clear, pop, insert, reverse ne s'appliquent pas aux tuples).

L'utilisation de listes constituées de tuples est très intéressante.
Par exemple, une image peut être structurée comme une liste de lignes, chaque ligne étant elle-même constituée d'une liste de tuples (R,V,B) représentant les couleurs de chaque pixel.
Par exemple, une image de 2x2 pixels sera représentée par la liste multidimensionnelle suivante :

[[(200,100,125),(140,5,127)],[(125,255,100),(112,110,95)]]

5) Les dictionnaires

Un dictionnaire est une structure de donnée similaire à une liste pour le contenu, mais dont l'accès aux valeurs se fait non pas par un index mais par une clé.

Les clés sont nécessairement être d'un type immuable (int, str, tuple) et être toutes différentes.
Les valeurs sont de n'importe quel type.

syntaxes pour déclarer un dictionnaire :

```
monDicoVide = dict()
monDicoVide = {}
monDico = {cle1:val,cle2:val,...,cleN,val}
monDico = dict([(cle1,val),...,(cleN,val)])
monDico = dict(((cle1,val),...,(cleN,val)))
monDico = dict(zip([cle1,...,cleN],[val,...,val]))
```

Pour ajouter un élément (clé : valeur) ou modifier une valeur :

```
monDico[maCle] = maValeur
```