IV. Les collections

1. List

Les listes (ou **list** / **array**) en python sont une variable dans laquelle on peut mettre plusieurs variables.

1.1. Créer une liste en python

Pour créer une liste, rien de plus simple :

```
>>> liste = []
```

Vous pouvez voir le contenu de la liste en l'appelant comme ceci:

```
>>> liste
<type 'list'>
```

1.2. Ajouter une valeur à une liste python

Vous pouvez ajouter les valeurs que vous voulez lors de la création de la liste python :

```
>>> liste = [1,2,3]
>>> liste
[1, 2, 3]
```

Ou les ajouter après la création de la liste avec la méthode append (qui signifie "ajouter" en anglais):

```
>>> liste = []
>>> liste
[]
>>> liste.append(1)
>>> liste
[1]
>>> liste.append("ok")
>>> liste
[1, 'ok']
```

On voit qu'il est possible de mélanger dans une même liste des variables de type différent. On peut d'ailleurs mettre une liste dans une liste.

1.3. Afficher un item d'une liste

Pour lire une liste, on peut demander à voir l'index de la valeur qui nous intéresse:

```
>>> liste = ["a","d","m"]
>>> liste[0]
'a'
>>> liste[2]
'm'
```

Le premier item commence toujours avec l'index 0. Pour lire le premier item on utilise la valeur **0**, le deuxième on utilise la valeur **1**, etc.

Il est d'ailleurs possible de modifier une valeur avec son index

```
>>> liste = ["a","d","m"]
>>> liste[0]
'a'
>>> liste[2]
'm'
>>> liste[2] = "z"
>>> liste
['a', 'd', 'z']
```

1.4. Supprimer une entrée avec un index

Il est parfois nécessaire de supprimer une entrée de la liste. Pour cela vous pouvez utiliser la fonction `del`.

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

1.5. Supprimer une entrée avec sa valeur

Il est possible de supprimer une entrée d'une liste avec sa valeur avec la méthode `remove`.

```
>>> liste = ["a", "b", "c"]
>>> liste.remove("a")
>>> liste
['b', 'c']
```

1.6. Inverser les valeurs d'une liste

Vous pouvez inverser les items d'une liste avec la méthode `reverse`.

```
>>> liste = ["a", "b", "c"]
>>> liste.reverse()
>>> liste
['c', 'b', 'a']
```

1.7. Compter le nombre d'items d'une liste

Il est possible de compter le nombre d'items d'une liste avec la fonction `len`.

```
>>> liste = [1,2,3,5,10]
>>> len(liste)
5
```

1.8. Compter le nombre d'occurrences d'une valeur

Pour connaitre le nombre d'occurrences d'une valeur dans une liste, vous pouvez utiliser la méthode **count**.

```
>>> liste = ["a","a","a","b","c","c"]
>>> liste.count("a")
3
>>> liste.count("c")
2
```

1.9. Trouver l'index d'une valeur

La méthode **index** vous permet de connaitre la position de l'item cherché.

```
>>> liste = ["a","a","a","b","c","c"]
>>> liste.index("b")
3
```

1.10. Manipuler une liste

Voici quelques astuces pour manipuler des listes:

```
>>> liste = [1, 10, 100, 250, 500]
>>> liste[0]
1
>>> liste[-1] # Cherche la dernière occurence
500
>>> liste[-4:] # Affiche les 4 dernières occurrences
[500, 250, 100, 10]
>>> liste[:] # Affiche toutes les occurences
[1, 10, 100, 250, 500]
>>> liste[2:4] = [69, 70]
[1, 10, 69, 70, 500]
>>> liste[:] = [] # vide la liste
[]
```

1.11. Boucler sur une liste

Pour afficher les valeurs d'une liste, on peut utiliser une boucle:

```
>>> liste = ["a","d","m"]
>>> for lettre in liste:
...     print lettre
...
a
d
m
```

Si vous voulez en plus récupérer l'index, vous pouvez utiliser la fonction **enumerate**.

```
>>> for lettre in enumerate(liste):
...     print lettre
...
(0, 'a')
(1, 'd')
(2, 'm')
```

Les valeurs retournées par la boucle sont des tuples.

1.12. Copier une liste

Beaucoup de débutants font l'erreur de copier une liste de cette manière

```
>>> x = [1,2,3]
>>> y = x
```

Or si vous changez une valeur de la liste y, la liste x sera elle aussi affectée par cette modification :

```
>>> x = [1,2,3]
>>> y = x
>>> y[0] = 4
>>> x
[4, 2, 3]
```

En fait cette syntaxe permet de travailler sur un même élément nommé différemment. Alors comment copier une liste qui sera indépendante ?

```
>>> x = [1,2,3]
>>> y = x[:]
>>> y[0] = 9
>>> x
[1, 2, 3]
>>> y
[9, 2, 3]
```

Pour des données plus complexes, vous pouvez utiliser la fonction **deepcopy** du module **copy**

```
>>> import copy
>>> x = [[1,2], 2]
>>> y = copy.deepcopy(x)
>>> y[1] = [1,2,3]
>>> x
[[1, 2], 2]
>>> y
[[1, 2], [1, 2, 3]]
```

1.13. Transformer une chaine en liste

Parfois il peut être utile de transformer une chaine de caractère en liste. Cela est possible avec la méthode **split**.

```
>>> ma_chaine = "ESSADDOUKI:Mostafa:Meknes"
>>> ma_chaine.split(":")
['ESSADDOUKI', 'Mostafa', 'Meknes']
```

1.14. Transformer une liste en chaine

L'inverse est possible avec la méthode "**join**".

```
>>> liste =['ESSADDOUKI', 'Mostafa', 'Meknes']
>>> "*".join(liste)
' ESSADDOUKI*Mostafa*Meknes '
```

1.15. Trouver un item dans une liste

Pour savoir si un élément est dans une liste, vous pouvez utiliser le mot clé in de cette manière :

```
>>> liste = [1,2,3,5,10]
>>> 3 in liste
True
>>> 11 in liste
False
```

1.16. La fonction range

La fonction **range** génère une liste composée d'une simple suite arithmétique.

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

1.17. Agrandir une liste par une liste

Pour mettre bout à bout deux listes, vous pouvez utiliser la méthode **extend**

```
>>> x = [1, 2, 3, 4]
>>> y = [4, 5, 1, 0]
>>> x.extend(y)
>>> print x
[1, 2, 3, 4, 4, 5, 1, 0]
```

1.18. Astuces

Afficher les 2 premiers éléments d'une liste

```
>>> liste = [1,2,3,4,5]
>>> liste[:2]
[1, 2]
```

Afficher le dernier item d'une liste :

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[-1]
6
```

Afficher le 3ème élément en partant de la fin :

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[-3]
4
```

Afficher les 3 derniers éléments d'une liste :

```
>>> liste = [1, 2, 3, 4, 5, 6]
>>> liste[-3:]
[4, 5, 6]
```

Vous pouvez additionner deux listes pour les combiner ensemble en utilisant l'opérateur **+:**

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> x + y
[1, 2, 3, 4, 5, 6]
```

Vous pouvez même multiplier une liste :

```
>>> x = [1, 2]
>>> x*5
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

Ce qui peut être utile pour initialiser une liste :

```
>>> [0] * 5
[0, 0, 0, 0, 0]
```

2. Tuple

Un tuple est une liste qui ne peut plus être modifiée.

2.1. Créer un tuple

Pour créer un tuple, vous pouvez utiliser la syntaxe suivante:

```
>>> mon_tuple = ()
```

2.2. Ajouter une valeur à un tuple

Pour créer un tuple avec des valeurs, vous pouvez le faire de cette façon:

```
>>> mon_tuple = (1, "ok", "cpge")
```

Les parenthèses ne sont pas obligatoires mais facilite la lisibilité du code (rappelons que la force de python est sa simplicité de lecture):

```
>>> mon_tuple = 1, 2, 3
>>> type(mon_tuple)
<type 'tuple'>
```

Lorsque vous créez un tuple avec une seule valeur, n'oubliez pas d'y ajouter une virgule, sinon ce n'est pas un tuple.

```
>>> mon_tuple = ("ok")
>>> type(mon_tuple)
<type 'str'>
>>> mon_tuple = ("ok",)
>>> type(mon_tuple)
<type 'tuple'>
```

2.3. Afficher une valeur d'un tuple

Le tuple est une sorte de liste, on peut donc utiliser la même syntaxe pour lire les données du tuple.

```
>>> mon_tuple[0]
1
```

Et évidemment si on essaie de changer la valeur d'un index, l'interpreteur nous insulte copieusement:

```
>>> mon_tuple[1] = "ok"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

2.4. A quoi sert un tuple alors?

Le tuple permet une affectation multiple :

```
>>> v1, v2 = 11, 22
>>> v1
11
>>> v2
22
```

Il permet également de renvoyer plusieurs valeurs lors d'un appel d'une fonction :

```
>>> def donne_moi_ton_nom():
...                 return ("CPGE", "BAB ESSAHRA")
...
>>> donne_moi_ton_nom()
(CPGE, BAB ESSAHRA)
```

On utilisera un Tuple pour définir des sortes de constantes qui n'ont donc pas vocation à changer.

3. Dictionnaire

Un dictionnaire en python est une sorte de liste mais au lieu d'utiliser des index, on utilise des clés, c'est à dire des valeurs autres que numériques.

3.1. Comment créer un dictionnaire ?

Pour initialiser un dictionnaire, on utile la syntaxe suivante :

```
>>> a = {}
```

3.2. Comment ajouter des valeurs dans un dictionnaire ?

Pour ajouter des valeurs à un dictionnaire il faut indiquer une clé ainsi qu'une valeur :

```
>>> a = {}
>>> a["nom"] = "essaddouki"
>>> a["prenom"] = "mostafa"
>>> a
{'nom': 'essaddouki', 'prenom': 'mostafa'}
```

Vous pouvez utiliser des clés numériques comme dans la logique des listes.

3.3. Récupérer une valeur dans un dictionnaire

La méthode **get** vous permet de récupérer une valeur dans un dictionnaire et si la clé est introuvable, vous pouvez donner une valeur à retourner par défaut :

```
>>> data = {"name": "mostafa", "age": 30}
>>> data.get("name")
'mostafa'
>>> data.get("adresse", "Adresse inconnue")
'Adresse inconnue'
```

3.4. Supprimer une entrée de dictionnaire

Il est possible de supprimer une entrée en indiquant sa clé, comme pour les listes:

```
>>> del a["nom"]
>>> a
{'prenom': 'mostafa'}
```

3.5. Récupérer les clés par une boucle

Pour récupérer les clés on utilise la méthode **keys**

```
>>> fiche = {"nom":"essaddouki","prenom":"mostafa"}
>>> for cle in fiche.keys():
...             print(cle)
nom
prenom
```

3.6. Récupérer les valeurs par une boucle

Pour cela on utilise la méthode **values**

```
>>> fiche = {"nom":"essaddouki","prenom":"mostafa"}
>>> for valeur in fiche.values():
...             print(valeur)
...
essaddouki
mostafa
```

3.7. Récupérer les clés et les valeurs par une boucle

Pour récupérer les clés et les valeurs en même temps, on utilise la méthode **items** qui retourne un tuple.

```
>>> fiche = {"nom":"essaddouki","prenom":"mostafa"}
>>> for cle,valeur in fiche.items():
...             print(cle, valeur)
...
nom essaddouki
prenom mostafa
```

3.8. Utiliser des tuples comme clé

Une des forces de python est la combinaison tuple/dictionnaire qui fait des merveilles dans certains cas comme lors de l'utilisation de coordonnées.

```
>>> b = {}
>>> b[(3,2)]=12
>>> b[(4,5)]=13
>>> b
{(4, 5): 13, (3, 2): 12}
```

3.9. Créer une copie indépendante d'un dictionnaire

Comme pour toute variable, vous ne pouvez pas copier un dictionnaire en faisant dic1 = dic2 :

```
>>> d = {"k1":"mostafa", "k2":"sara"}
>>> e = d
>>> d["k1"] = "XXX"
>>> e
{'k2': 'sara', 'k1': 'XXX'}
```

Pour créer une copie indépendante vous pouvez utiliser la méthode `copy`:

```
>>> d = {"k1":"mostafa", "k2":"sara"}
>>> e = d.copy()
>>> d["k1"] = "XXX"
>>> e
{"k1":"mostafa", "k2":"sara"}
```