Chapitre 2: Les collections

Les listes en Python

Comment créer une liste en Python?

En programmation Python, une liste est créée en plaçant tous les éléments entre crochets [], séparés par des virgules.
Il peut avoir n'importe quel nombre d'éléments et ils peuvent être de différents types (entier, flottant, chaîne, etc.).

```
# liste vide
liste = []
# liste d'entiers
liste = [1, 2, 3]
# liste avec des types de données mixtes
liste = [10, "IWA", 2.5]
```

Une liste peut également avoir une autre liste en tant qu'élément. Cela s'appelle une liste imbriquée.

```
# liste imbriquée
my_list = ["IWA", [1, 2, 3], 2.8, ['x','y']]
```

Les listes en Python

➜Comment accéder aux éléments d'une liste?

Vous accédez aux éléments de la liste en vous référant au numéro d'index. L'exemple suivant affiche le deuxième élément de la liste:

```
liste = ["Blue", "Red", "Green"]
print(liste[1])
```

Indexation négative

L'indexation négative signifie à partir de la fin, -1 se réfère au dernier élément, -2 se réfère à l'avant-dernier élément, etc. L'exemple suivant affiche le dernier élément de la liste:

```
liste = ["Blue", "Red", "Green"]
print(liste[-1])
```

Les listes en Python

➔Comment accéder aux éléments d'une liste?

Plage d'index

Vous pouvez spécifier une plage d'index en spécifiant par où commencer et où terminer la plage. Lors de la spécification d'une plage, la valeur de retour sera une nouvelle liste avec les éléments spécifiés.
L'exemple suivant renvoie le troisième et quatrième éléments:

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]
print(liste[2:4])
```

La recherche commencera à l'index 2 (inclus) et se terminera à l'index 4 (non inclus).
En ignorant la valeur de départ, la plage commencera au premier élément.
L'exemple suivant renvoie les éléments du début jusqu'à la chaîne « orange »:

Les listes en Python

Modifier la valeur d'un élément

Pour modifier la valeur d'un élément spécifique, referez-vous au numéro d'index. L'exemple suivant change le deuxième élément:

```
liste = ["Blue", "Red", "Green"]
liste[1] = "Black"
print(liste)
```

Parcourir une liste en Python

Vous pouvez parcourir les éléments de la liste en utilisant la boucle for. L'exemple suivant affiche tous les éléments de la liste, un par un:

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]

for i in liste:
  print(i)
```

Les listes en Python

Vérifiez si un élément existe dans la liste

Pour déterminer si un élément spécifié est présent dans une liste, utilisez le mot clé in. L'exemple suivant vérifie si la couleur « Red » est présent dans la liste:

```
liste = ["Blue", "Red", "Green"]
if "Red" in liste:
  print("'Red' existe dans la liste")
```

Longueur d'une liste

Pour déterminer le nombre d'éléments d'une liste, utilisez la fonction len(). L'exemple suivant affiche le nombre d'éléments dans la liste:

```
liste = ["Blue", "Red", "Green"]

print(len(liste))
```

Les listes en Python

Ajouter des éléments à la liste

Pour ajouter un élément à la fin de la liste, utilisez la méthode append(). L'exemple suivant ajoute l'élément « Black » en utilisant la méthode append():

```
liste = ["Blue", "Red", "Green"]
liste.append("Black")
print(liste)
```

Pour ajouter un élément à l'index spécifié, utilisez la méthode insert(). L'exemple suivant insère l'élément dans la troisième position:

```
liste = ["Blue", "Red", "Green"]
liste.insert(2, "Black")
print(liste)
```

Les listes en Python

Supprimer un élément de la liste

Il existe plusieurs méthodes pour supprimer des éléments d'une liste:

remove():

La méthode remove() supprime l'élément spécifié:

```
liste = ["Blue", "Red", "Green"]
liste.remove("Red")
print(liste)
```

pop():

La méthode pop() supprime l'index spécifié, (ou le dernier élément si l'index n'est pas spécifié):

```
liste = ["Blue", "Red", "Green"]
liste.pop()
print(liste)
```

Les listes en Python

Supprimer un élément de la liste

Il existe plusieurs méthodes pour supprimer des éléments d'une liste:

del:

Le mot clé del supprime l'index spécifié:

```
liste = ["Blue", "Red", "Green"]
del liste[1]
print(liste)
```

clear():

La méthode clear() vide la liste:

```
liste = ["Blue", "Red", "Green"]
liste.clear()
print(liste)
```

Les listes en Python

Concaténer deux listes

Il existe plusieurs façons de concaténer deux ou plusieurs listes en Python. L'un des moyens les plus simples consiste à utiliser l'opérateur +.

Le mot clé del supprime l'index spécifié:

```
liste1 = ["A", "B" , "C"]
liste2 = [1, 2, 3]
liste3 = liste1 + liste2
print(liste3)
```

Les listes en Python

Copier une liste

Vous ne pouvez pas copier une liste simplement en tapant liste2 = liste1, car liste2 ne sera qu'une référence à liste1, et les modifications apportées dans liste1 seront automatiquement apportées dans liste2.
Il existe un moyen de faire une copie, une façon consiste à utiliser la méthode de liste intégrée copy().

```
liste1 = ["Blue", "Red", "Green"]
liste2 = liste1.copy()
print(liste2)
```

Les listes en Python

Fonctions

- **`l.sort()`** : Trie une liste (si les objets ne sont pas des types standards il faut rajouter une fonction qu'on verra plus tard)
- **`l.reverse()`** : retourne une liste dont les éléments sont ceux de l mais d'ordre inversé
- **`l.count(v)`** : retourne le nombre d'occurrence de v dans l
- **`l.expend(l2)`** : insère tous les éléments de l2 à la fin de l1 (concaténation)
- On peut créer des matrices sous forme de liste de listes

```
M=[[1, 2, 3], [4, 5, 6]]
```

Les tuples en Python

Définition

Un tuple est une séquence immuable d'objets. Les tuples sont des séquences, tout comme les listes. La différence entre les tuples et les listes est, les tuples ne peuvent pas être modifiés contrairement aux listes et les tuples utilisent des parenthèses (), tandis que les listes utilisent des crochets [].

Créer un tuple est aussi simple, il suffit de mettre les valeurs séparées par des virgules. Vous pouvez également mettre ces valeurs séparées par des virgules entre parenthèses. Par exemple :

```
t1 = ('A', 'B', 'C', 1, 2, 3)
print(t1)
t2 = 'A', 'B', 'C', 1, 2, 3
print(t2)
```

Un tuple vide est écrit comme deux parenthèses ne contenant rien:

```
tuple = ()
```

Les tuples en Python

Accéder aux éléments de tuple

Vous pouvez accéder aux éléments de tuple en vous référant au numéro d'index, entre crochets. L'exemple suivant affiche le deuxième élément du tuple:

```
tupl = ("PHP", "Pyton", "Java")
print(tupl[1])
```

L'indexation négative et la plage d'index comme les listes, il n'y a aucune différence

Les tuples en Python

Modifier la valeur d'un tuple en Python

Une fois un tuple créé, vous ne pouvez pas modifier ses valeurs. Les tuples sont immuables. Mais il y'a une astuce. Vous pouvez convertir le tuple en liste, modifier la liste et reconvertir la liste en tuple.

```
#créer un tuple
tupl = ("A", "B", "C")
#convertir tuple en liste
liste = list(tupl)
#modifier le 1er élément du liste
liste[0] = "X"
#convertir liste en tuple
tupl = tuple(liste)
#afficher le tuple
print(tupl)
```

Les dictionnaires en Python

Définition

Un dictionnaire est une collection non ordonnée, modifiable et indexée. En Python, les dictionnaires sont écrits avec des accolades {}, et ils ont des clés et des valeurs. L'exemple suivant crée et affiche un dictionnaire:

```
dictionnaire = {
  1 : "Python",
  2 : "PHP",
  3 : "Java"
}
print(dictionnaire)
```

Les dictionnaires en Python

Accès aux éléments

Vous pouvez accéder aux éléments d'un dictionnaire en vous référant à son clé, entre crochets []. L'exemple suivant récupère la valeur de la clé 1:

```
dictionnaire = {
   1 : "Python",
   2 : "PHP",
   3 : "Java"
}
print(dictionnaire)#Ou  print(dictionnaire.get(1))
```

Les dictionnaires en Python

Changer les valeurs d'un dictionnaire

Vous pouvez modifier la valeur d'un élément spécifique en vous référant à son clé.
L'exemple suivant change la valeur du clé 1:

```
dictionnaire = {
  1 : "Python",
  2 : "PHP",
  3 : "Java"
}

dictionnaire[1] = "Django"

print(dictionnaire)
```

Les dictionnaires en Python

Parcourir un dictionnaire en Python

Vous pouvez parcourir les éléments d'un dictionnaire en utilisant la boucle for. Lorsque vous parcourez un dictionnaire, les valeurs de retour sont les clés du dictionnaire, mais il existe également des méthodes pour renvoyer les valeurs. L'exemple suivant affiche tous les clés du dictionnaire :

```
dictionnaire = {
  1 : "Python",
  2 : "PHP",
  3 : "Java"
}

for key in dictionnaire:
  print(key)
```

Les dictionnaires en Python

Parcourir un dictionnaire en Python

Vous pouvez parcourir les éléments d'un dictionnaire en utilisant la boucle for. Lorsque vous parcourez un dictionnaire, les valeurs de retour sont les clés du dictionnaire, mais il existe également des méthodes pour renvoyer les valeurs.

Récupérer les clés

```
dictionnaire = {
  1 : "Python",
  2 : "PHP",
  3 : "Java"
}
for key in dictionnaire:
  print(key)
```

Récupérer les valeurs

```
dictionnaire = {
  1 : "Python",
  2 : "PHP",
  3 : "Java"
}
for key in dictionnaire:
  print(dictionnaire[key])
```

Vous pouvez également utiliser la méthode values() pour renvoyer les valeurs d'un dictionnaire:

```
for val in dictionnaire.values():
  print(val)
```

Les dictionnaires en Python

Parcourir un dictionnaire en Python

Vous pouvez parcourir les clés et les valeurs à l'aide de la méthode items():

```
dictionnaire = {
   1 : "Python",
   2 : "PHP",
   3 : "Java"
}
for key, value in dictionnaire.items():
   print(key, value)
```

Vérifiez si une clé existe dans un dictionnaire

```
if 2 in dictionnaire:
   print("2 existe dans le dictionnaire")
```

Les dictionnaires en Python

Ajouter des éléments au dictionnaire

L'ajout d'un élément au dictionnaire se fait en utilisant une nouvelle clé et en lui affectant une valeur

```
dictionnaire = {
   1 : "Python",
   2 : "PHP",
   3 : "Java"
}
dictionnaire[4] = "C++"
print(dictionnaire)
```

Les dictionnaires en Python

Supprimer un élément du dictionnaire

On utilise les fonctions pop, delete, popitem et clear

```
dictionnaire = {
   1 : "Python",
   2 : "PHP",
   3 : "Java"
}
dictionnaire.pop(2)
print(dictionnaire)
```

La méthode popitem() supprime le dernier élément inséré (dans les versions antérieures de Python 3.7, la méthode popitem() supprime un élément aléatoire):

Les dictionnaires en Python

Dictionnaires imbriqués

Un dictionnaire peut également contenir des dictionnaires, c'est ce qu'on appelle des dictionnaires imbriqués.

```
persone = {
  "p1" : {
    "nom" : "Alex",
    "age" : 18
  },
  "p2" : {
    "nom" : "Thomas",
    "age" : 25
  },
  "p3" : {
    "nom" : "Yohan",
    "age" : 44
  }
}
```

Les Sets en Python

Définition

Set(ou ensemble en français) est une collection d'éléments non ordonnée. Chaque élément du Set est unique (pas de doublons) et doit être immuable (ne peut pas être modifié). Pourtant, un Set lui-même est modifiable. Nous pouvons y ajouter ou en supprimer des éléments.

En Python, les Sets sont écrits avec des accolades {}. L'exemple suivant crée un Set:

```
mySet = {"A", "B", "C"}
print(mySet)
```

Les Sets en Python

Ajouter des éléments

Pour ajouter un élément à un Set, utilisez la méthode add(). Pour ajouter plusieurs éléments à un Set, utilisez la méthode update().

L'exemple suivant ajoute un élément à un Set, en utilisant la méthode add():

```
mySet = {"A", "B", "C"}
mySet.add("D")
print(mySet)
```

L'exemple suivant ajoute plusieurs éléments à un Set, en utilisant la méthode update():

```
mySet = {"A", "B", "C"}
mySet.update(["D", "E", "F"])
print(mySet)
```

Les Sets en Python

Supprimer un élément du Set

Pour supprimer un élément d'un Set, utilisez la méthode remove() ou discard().

L'exemple suivant supprime l'élément « B » en utilisant la méthode remove():

```
mySet = {"A", "B", "C"}
mySet.remove("B")  # mySet.discard("B")
print(mySet)
```

La différence entre la méthode remove() et discard() est la suivante: Si l'élément à supprimer n'existe pas, remove() déclenchera une erreur. Tandis que discard() ne déclenchera pas une erreur si l'élément à supprimer n'existe pas.

Vous pouvez également utiliser la méthode pop() pour supprimer un élément, mais cette méthode supprimera le dernier élément. N'oubliez pas que les Sets ne sont pas ordonnés, vous ne saurez donc pas quel élément sera supprimé.

Les tableaux en Python

Définition

En programmation, un tableau est une collection d'éléments du même type.
Les tableaux sont pris en charge en Python grâce au module « array ».

Différence entre Liste et Tableau en Python

Nous pouvons traiter les listes comme des tableaux. Cependant, le type d'éléments stockés est complètement différent. Par exemple:

```
#créer une liste avec des éléments de différents types
liste = ["A", 5, 2.2]
```

Si vous créez des tableaux à l'aide du module « array », tous les éléments du tableau doivent être du même type.

```
import array as arr
tableau = arr.array('d', ["A", 5, 2.2])
```

Le code ci-dessus affiche une erreur, car on attend des éléments de type float

Les tableaux en Python

Comment créer un tableau en Python

D'abord, nous devons importer le module « array » pour créer des tableaux. Par exemple:

```
import array as arr
tableau = arr.array('d', [1.0, 1.1, 1.2, 1.3])
print(tableau)
```

La lettre ‘d’ est un code de type. Cela détermine le type du tableau lors de la création.
Les codes de type couramment utilisés sont listés comme suit:

Code	Type
b	signed char
B	unsigned char
h	signed short
H	unsigned short
l	signed long
L	unsigned long
i	int
f	float
d	double

Les tableaux en Python

Manipulation d'un tableau

D'abord, nous devons importer le module « array » pour créer des tableaux. Par exemple:

```
import array as arr
tab = arr.array('i', [1, 2, 3, 4, 5, 6])
print(tab[1])
print(tab[-1])
print(tab[2:4])
print(tab[:3])
tab[1] = 100
for i in tab:
  print(i)
if 5 in tab:
  print("5 existe dans le tableau")
print(len(tab))
tab.append(7)
tab.remove(4)
```

Les Namedtuple en Python

Définition

Python prend en charge un type de conteneur comme les dictionnaires appelé « namedtuple() » présent dans le module « collections ». Comme les dictionnaires, ils contiennent des clés qui sont hachées à une valeur particulière.

```
from collections import namedtuple
# Declaring namedtuple()
Student = namedtuple('Student', ['name', 'age'])
# Adding values
S = Student('Mohamed', '35')
# Access using index
print("The Student age using index is : "+ str(S[1]))
# Access using name
print("The Student name using keyname is : "+ S.name)
```