Python - les conteneurs

I - Definitions

Définition 1

Les **tuples (ou n-uplet)** : un tuple est une collection ordonnée et **non modifiable** d'éléments éventuellement hétérogènes.

```
t = 12345, 54321, 'hello!'
singleton = 'hello',\\
# <-- note trailing comma
vide = ()
```

Définition 2

Les **listes** : une liste est une collection ordonnée et **modifiable** d'éléments hétérogènes. (Tableau indexé)

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple']
vide = []
```

Définition 3

Les **dictionnaires** : un dictionnaire est un tableau associatif **modifiable** il permet de stocker des couples – des paires (clé ,valeur) avec des valeurs de tous types et éventuellement hétérogènes et des clés ayant la contrainte d'être hashable.

```
tel = {'jack': 4098, 'sape': 4139}
vide= {}
```

II - Opérations sur les tuples

1) Création

```
# Tuple vide
my_tuple = ()
print(my_tuple) # Output: ()
# Tuple avec des entiers
my_tuple = (1, 2, 3)
print(my_tuple) # Output: (1, 2, 3)
# Tuple avec differents types
my_tuple = (1, "Hello", 3.4)
print(my_tuple) # Output: (1, "Hello", 3.4)
# sans parenthese
my_tuple = 3, 4.6, "dog"
print(my_tuple)
# Output: 3, 4.6, "dog"
```

2) Accès aux éléments

Les tuples possèdent un index qui commence à 0. On peut accéder à leurs éléments par le numéro d'index.

```
my_tuple = ('p','e','r','m','i','t')
print(my_tuple[0])
# 'p'
print(my_tuple[5])
# 't
```

L'index -1 fait référence au dernier élément, l'index -2 à l'élément avant celui-ci etc . . .

```
print(my_tuple[-1])
# Output: 't'
print(my_tuple[-6])
# Output: 'p'
```

3) Méthodes associées

1. **count(x)** : Compte le nombre d'items x
2. **index(x)** : Retourne l'index du premier item égal à x

```
my_tuple = ('a','p','p','l','e',)
print(my_tuple.count('p')) # Output: 2
print(my_tuple.index('l')) # Output: 3
```

III - Opérations sur les listes

Une liste (souvent appelée tableau dans les autres langages) permet de stocker plusieurs valeurs (chaîne, nombre) dans une structure unique :

```
maListe1 = ["pomme","orange","fraise"]
maListe2=[56,76,45,89]
```

On accède aux éléments de la liste par leur indice de position. Le premier élément de la liste possède l'indice 0.

```
print(maListe1[0]) #Output : pomme
```

On peut aussi créer une liste vide :

```
maListeVide = []
```

1) Parcours de la liste

1. Boucle while

```
i=0
while(i<len(maListe1):
        print(maListe1[i])
        i=i+1
print("Fin de liste")
#Output
pomme
orange
fraise
```

2. Boucle for

```
for fruit in maListe1:
        print(fruit)
print("Fin de liste")
#Output
pomme
orange
fraise
```

2) Ajouter un ou des éléments dans la liste

1. Ajouter un élément en fin de liste : **append()**

```
lst = list(range(3))
print(lst)
# [0, 1, 2]

lst.append(100)
print(lst)
# [0, 1, 2, 100]

lst.append('new')
print(lst)
# [0, 1, 2, 100, 'new']
```

Une liste est également ajoutée en tant qu'élément unique, non combiné.

```
lst.append([3, 4, 5])
print(lst)
# [0, 1, 2, 100, 'new', [3, 4, 5]]
```

2. Ajouter une autre liste à une liste (= combiner des listes) : **extend()**, **+**, **+=** Vous pouvez utiliser la méthode extend() pour ajouter une autre liste à une liste, c'est-à-dire combiner des listes. Tous les éléments sont ajoutés à la fin de la liste d'origine.
Vous pouvez spécifier d'autres objets itérables, tels que tuple.

```
lst = list(range(3))
print(lst)
# [0, 1, 2]

lst.extend([100, 101, 102])
print(l)
# [0, 1, 2, 100, 101, 102]

lst.extend((-1, -2, -3))
print(lst)
# [0, 1, 2, 100, 101, 102, -1, -2, -3]
```

Dans le cas d'une chaîne (str), chaque caractère est ajouté un par un.

```
lst.extend('new')
print(lst)
# [0, 1, 2, 100, 101, 102, -1, -2, -3, 'n', 'e', 'w']
```

Vous pouvez également combiner des listes avec l'opérateur +.
Dans le cas de l'opérateur +, une nouvelle liste est renvoyée.
Vous pouvez également ajouter une autre liste à la liste existante avec l'opérateur +=.

```
lst2 = lst + [5, 6, 7]
print(lst2)
# [0, 1, 2, 100, 101, 102, -1, -2, -3, 'n', 'e', 'w', 5, 6, 7]

lst += [5, 6, 7]
print(lst)
# [0, 1, 2, 100, 101, 102, -1, -2, -3, 'n', 'e', 'w', 5, 6, 7]
```

3. Insérer un élément dans une liste : **insert()** Vous pouvez insérer un élément à n'importe quel index (position) avec la méthode insert().
   Définissez l'index pour le premier paramètre et l'élément à insérer pour le deuxième paramètre. L'index au début est 0 (indexation de base zéro). Pour les valeurs négatives, -1 signifie un avant la fin.

```
lst = list(range(3))
print(lst)
# [0, 1, 2]

lst.insert(0, 100)
print(lst)
# [100, 0, 1, 2]

lst.insert(-1, 200)
print(lst)
# [100, 0, 1, 200, 2]
```

Comme append(), la liste est ajoutée en tant qu'élément unique, non combiné.

```
lst.insert(0, [-1, -2, -3])
print(lst)
# [[-1, -2, -3], 100, 0, 1, 200, 2]
```

4. Insérer une autre liste dans une liste : slice Si vous spécifiez une plage à l'aide de slice et affectez une autre liste ou tuple, tous les éléments sont ajoutés.

```
lst = list(range(3))
print(lst)
# [0, 1, 2]

lst[1:1] = [100, 200, 300]
print(lst)
# [0, 100, 200, 300, 1, 2]
```

Vous pouvez également remplacer l'article d'origine. Tous les éléments de la plage spécifiée sont remplacés.

```
lst = list(range(3))
print(lst)
# [0, 1, 2]

lst[1:2] = [100, 200, 300]
print(lst)
# [0, 100, 200, 300, 2]
```

3) Supprimer un élément dans la liste

1. Supprimer tous les éléments : méthode **clear()**

```
lst = list(range(10))
print(lst)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

lst.clear()
print(lst)
# []
```

2. Supprimer un élément par index et obtenir sa valeur : méthode **pop()**

```
lst = list(range(10))
print(lst)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(lst.pop(0))
# 0

print(lst)
# [1, 2, 3, 4, 5, 6, 7, 8, 9]

print(lst.pop(3))
# 4

print(lst)
# [1, 2, 3, 5, 6, 7, 8, 9]
```

Vous pouvez utiliser des valeurs négatives pour spécifier la position à partir de la fin. L'indice à la fin est -1.

```
print(lst.pop(-2))
# 8

print(lst)
# [1, 2, 3, 5, 6, 7, 9]
```

Si l'argument est omis, le dernier élément est supprimé.

```
print(lst.pop())
# 9

print(lst)
# [1, 2, 3, 5, 6, 7]
```

La spécification d'un index inexistant a généré une erreur.

```
# print(lst.pop(100))
# IndexError: pop index out of range
```

3. Supprimer un élément par valeur : méthode **remove()**

```
lst = ['Alice', 'Bob', 'Charlie', 'Bob', 'Dave']
print(lst)
# ['Alice', 'Bob', 'Charlie', 'Bob', 'Dave']

lst.remove('Alice')
print(lst)
# ['Bob', 'Charlie', 'Bob', 'Dave']
```

Si la liste contient en plusieurs correspondants à la valeur désignée, seul le premier est supprimé.

```
lst.remove('Bob')
print(lst)
# ['Charlie', 'Bob', 'Dave']
```

La spécification d'une valeur inexistante a généré une erreur.

```
lst.remove('xxx')
# ValueError: list.remove(x): x not in list
```

4. Supprimer les éléments par index ou tranche : instruction **del**.
clear(), pop() et remove() sont des méthodes de list. Vous pouvez également supprimer des éléments d'une liste avec des instructions del.
Spécifiez l'élément à supprimer par index. Le premier indice est 0 et le dernier est -1.

```
lst = list(range(10))
print(lst)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

del lst[0]
print(lst)
# [1, 2, 3, 4, 5, 6, 7, 8, 9]

del lst[-1]
print(lst)
# [1, 2, 3, 4, 5, 6, 7, 8]

del lst[6]
print(lst)
# [1, 2, 3, 4, 5, 6, 8]
```

Vous pouvez supprimer plusieurs éléments avec slice.

```
lst = list(range(10))
print(lst)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

del lst[2:5]
print(lst)
# [0, 1, 5, 6, 7, 8, 9]

lst = list(range(10))
del lst[:3]
print(lst)
# [3, 4, 5, 6, 7, 8, 9]

lst = list(range(10))
del lst[4:]
print(lst)
# [0, 1, 2, 3]

lst = list(range(10))
del lst[-3:]
print(lst)
# [0, 1, 2, 3, 4, 5, 6]
```

Il est également possible de supprimer tous les éléments en spécifiant toute la plage.

```
lst = list(range(10))
del lst[:]
print(lst)
# []
```

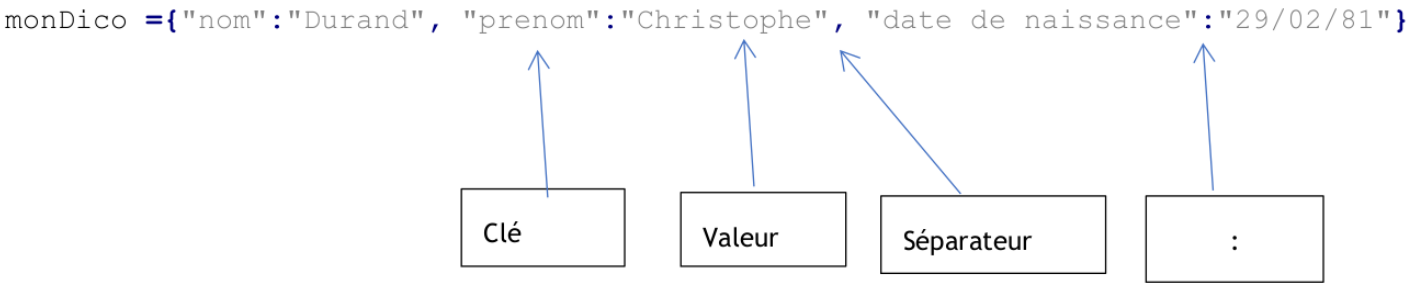
Vous pouvez également spécifier l'étape comme [start :stop :step].

```
lst = list(range(10))
del lst[2:8:2]
print(lst)
# [0, 1, 3, 5, 7, 8, 9]

lst = list(range(10))
del lst[::3]
print(lst)
# [1, 2, 4, 5, 7, 8]
```

IV - Opérations sur les dictionnaires

Les dictionnaires ressemblent aux listes. Chaque élément d'un dictionnaire est composé de 2 parties, on parle de pairs "clé/valeur".



Pour afficher le dictionnaire on utiliser la fonction print(monDico).

1) Construction à partir d'un dictionnaire vide

```
monDico={}   #dictionnaire vide : {}
monDico["nom"]="Durand"
monDico["prenom"]="Christophe"
monDico["date de naissance"]="29/02/1981"
```

2) Afficher la valeur associée à une clé

```
print("Bonjour je m'appelle "+monDico["prenom"]+" "+monDico["nom"]+",
je suis ne le "+monDico["date de naissance"])
#Output : Bonjour je m'appelle Christophe Durand,
# je suis ne le 29/02/1981
```

3) Supprimer des éléments dans un dictionnaire

Le conteneur dictionnaire dispose des mêmes méthodes et instructions que les liste pour supprimer des éléments.

1. Supprimer tous les éléments d'un dictionnaire : **clear()**

```
d = {'k1': 1, 'k2': 2, 'k3': 3}
d.clear()
print(d)
# {}
```

2. Supprime un élément par une clé et renvoie une valeur : **pop()**
   En spécifiant une clé à la méthode pop(), l'élément est supprimé et sa valeur est renvoyée.

```
d = {'k1': 1, 'k2': 2, 'k3': 3}

removed_value = d.pop('k1')
print(d)
# {'k2': 2, 'k3': 3}

print(removed_value)
# 1
```

   Par défaut, la spécification d'une clé inexistante génère KeyError.

```
d = {'k1': 1, 'k2': 2, 'k3': 3}

# removed_value = d.pop('k4')
# print(d)
# KeyError: 'k4'
```

   Si le deuxième argument est spécifié, sa valeur est retournée si la clé n'existe pas. Le dictionnaire lui-même reste inchangé.

```
d = {'k1': 1, 'k2': 2, 'k3': 3}

removed_value = d.pop('k4', None)
print(d)
# {'k1': 1, 'k2': 2, 'k3': 3}

print(removed_value)
# None
```

3. Supprimer un élément et renvoyer une clé et une valeur : **popitem()**
   La méthode popitem() supprime un élément d'un dictionnaire et renvoie un tuple de sa clé et de sa valeur (clé, valeur). Vous ne pouvez pas spécifier l'élément à supprimer.
   Une erreur KeyError est levée pour un dictionnaire vide.

```
d = {'k1': 1, 'k2': 2}

k, v = d.popitem()
print(k)
print(v)
print(d)
# k2
# 2
# {'k1': 1}

k, v = d.popitem()
print(k)
print(v)
print(d)
# k1
# 1
# {}

# k, v = d.popitem()
# KeyError: 'popitem(): dictionary is empty'
```

4. Supprimer un élément par une clé d'un dictionnaire : **del**

```
d = {'k1': 1, 'k2': 2, 'k3': 3}

del d['k2']
print(d)
# {'k1': 1, 'k3': 3}
```

Vous pouvez spécifier et supprimer plusieurs éléments.

```
d = {'k1': 1, 'k2': 2, 'k3': 3}

del d['k1'], d['k3']
print(d)
# {'k2': 2}
```

Si une clé inexistante est spécifiée, l'erreur KeyError est générée.

```
d = {'k1': 1, 'k2': 2, 'k3': 3}

# del d['k4']
# print(d)
# KeyError: 'k4'
```

4) Ajouter des éléments dans un dictionnaire

1. **Ajouter et mettre à jour des éléments au dictionnaire en spécifiant des clés.**
Lorsqu'une clé inexistante est spécifiée, un nouvel élément est ajouté, et lorsqu'une clé existante est spécifiée, la valeur existante est mise à jour (écrasée).

```
d = {'k1': 1, 'k2': 2}

d['k3'] = 3
print(d)
# {'k1': 1, 'k2': 2, 'k3': 3}

d['k1'] = 100
print(d)
# {'k1': 100, 'k2': 2, 'k3': 3}
```

2. **Concaténer (fusionner) plusieurs dictionnaires : update(), | operator, |= operator**
   Si la clé chevauche une clé existante, elle sera écrasée avec la valeur du dictionnaire spécifiée dans l'argument.

```
d1 = {'k1': 1, 'k2': 2}
d2 = {'k1': 100, 'k3': 3, 'k4': 4}

d1.update(d2)
print(d1)
# {'k1': 100, 'k2': 2, 'k3': 3, 'k4': 4}
```

   Depuis Python 3.9, il est possible de fusionner deux dictionnaires en utilisant l'opérateur |. Lorsque deux dictionnaires ont la même clé, la valeur de droite est prioritaire.

```
d1 = {'k1': 1, 'k2': 2}
d2 = {'k1': 100, 'k3': 3, 'k4': 4}

print(d1 | d2)
# {'k1': 100, 'k2': 2, 'k3': 3, 'k4': 4}

print(d2 | d1)
# {'k1': 1, 'k3': 3, 'k4': 4, 'k2': 2}
```

   Avec l'opérateur |=, une liste de (clé, valeur) peut être spécifiée sur le côté droit.

```
d = {'k1': 1, 'k2': 2}

d |= [('k1', 100), ('k3', 3), ('k4', 4)]
print(d)
# {'k1': 100, 'k2': 2, 'k3': 3, 'k4': 4}
```

5) Parcourir un dictionnaire à l'aide d'une boucle "for"

```
myDict = {'pomme':20, 'melon':45,'framboise':456}
#Affiche les cles
for cle in myDict.keys():
        print(cle)
#Output
pomme
melons
franboise

#Affiche les valeurs
for valeur in myDict.values():
        print(valeur)
#Output
20
45
456

#Affiche couple cle /valeur
for cle,valeur in myDict.items():
        print("La cle {} contient la valeur {}.".format(cle, valeur))
#Output
La cle pomme contient la valeur 20.
La cle melons contient la valeur 45.
La cle franboise contient la valeur 456.
```