Table des matières

1. Définition

En plus des types simples le Python offre des types de données plus évolués : les conteneurs.

Un conteneur est un objet composite destiné à contenir d'autres objets.

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

```
t = 12345, 54321, 'hello!'
#création d'un tuple vide
mon_nuplet = tuple()
#équivalent
mon_tuple=()
#création d'un n-uplet avec un seul élément
t1 ='a', #  la parenthèse indique que c'est un tuple
```

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

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
#création d'une liste vide
ma_liste = []
#équivalent
ma_liste = list()
```

1.3. **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}
#création d'un dictionnaire vide
mon_dictionnaire = {}
#équivalent
mon_dictionnaire = dict()
```

2. Opérations sur les tuples

2.1. 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 différents types
my_tuple = (1, "Hello", 3.4)
print(my_tuple)  # Output: (1, "Hello", 3.4)
# sans parenthèse
my_tuple = 3, 4.6, "dog"
print(my_tuple)   # Output: 3, 4.6, "dog"
```

2.2. 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 -2 à l »élément avant celui-ci etc …

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

2.3. Méthodes associées

Method	Description
count(x)	Retourne le nombre d'items *x*
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
```

2.4. Modification d'un tuple

```
51  my_tuple = (4, 2, 3, [6, 5])
52  my_tuple[1] = 9
```

Une exception s'est produite : TypeError
'tuple' object does not support item assignment

3. 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"]  ou   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])` affiche pomme

On peut aussi créer une liste vide

```
maListeVide = []
```

4.1. Parcours de la liste

Boucle while	Boucle for

`while(i<3):`
`  print(maListe1[i])`
`  i=i+1`
`print("Fin de liste")`
	`for fruit in maListe1:`
	`  print(fruit)`
	`print("Fin de liste")`

pomme orange fraise Fin de liste	pomme orange fraise Fin de liste

4.2. Utilisations de fonctions

Suppression avec le numéro de l'index	Taille	
`del(maListe1[1])`	`len(maListe1)`	
Surprime le deuxième élément de la liste	Donne la taille de la liste	

4.3. En utilisant des méthodes

On utilise des fonctions particulières appelées méthodes associées à l'objet liste. Ces fonctions leurs sont propres et sont écrites avec une syntaxe avec un point sur l'objet :

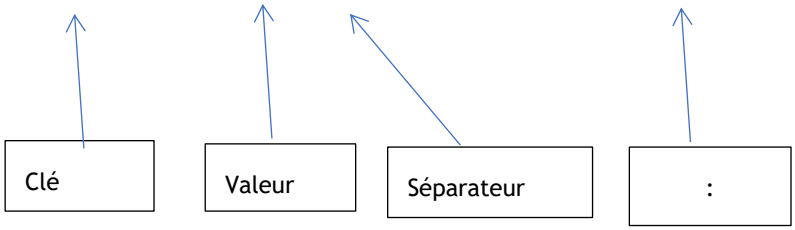
```
mon-objet.nom-de-la-méthode(attributs)
```

Ajout	Suppression avec le nom de la valeur	Connaitre le nombre d'occurrence d'une valeur
`maListe1.append("raisin")`	`maListe1.remove("raisin")`	`maListe1.count("fraise")`
Ajoute « raisin » en fin de liste	Supprime « raisin »	Donne 1

4. 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".

```
monDico ={"nom":"Durand", "prenom":"Christophe", "date de naissance":"29/02/81"}
```



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

5.1. Construction à partir d'un dictionnaire vide qui se note {}:

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

5.2. Afficher la valeur associée à une clé :

```
print("Bonjour je m'appelle "+monDico["prenom"]+" "+monDico["nom"]+", je suis né le "+monDico["date de naissance"])
```

Affiche : Bonjour je m'appelle Christophe Durand, je suis né le 29/02/1981

- La fonction **`del()`** permet, comme pour les listes, de supprimer une paire "clé/valeur" d'un dictionnaire.

5.3. Parcourir un dictionnaire à l'aide d'une boucle for

Vous pouvez utiliser une boucle **for** pour traiter successivement tous les éléments d'un dictionnaire mais attention :

- Au cours de l'itération, ce sont les **clés** utilisées qui seront successivement affectées à la variable de travail et non les **valeurs** ;
- L'ordre dans lequel les éléments sont extraits est **imprévisible** (il n'y a pas d'ordre dans un dictionnaire comme c'est le cas avec une liste).

Parcours d'un dictionnaire : `myDict={'pomme':20,'melons':45,'franboise':456}`

```
#Affiche les clés
for cle in myDict.keys():
    print(cle)
#Affiche les valeurs
for valeur in myDict.values():
    print(valeur)
#Affiche couple clé /valeur
for cle,valeur in myDict.items():
    print("La clé {} contient la valeur {}.".format(cle, valeur))
```

Affiche

pomme
melons
franboise
20
45
456
La clé pomme contient la valeur 20.
La clé melons contient la valeur 45.
La clé franboise contient la valeur 456.