

## L10 : Types construits

Nous avons vu les types de base (nombre entier ou flottant, caractères, booléens).

Il existe des types construits, qui sont des éléments plus élaborés.

- **les séquences** : ce sont des ensembles finis et ordonnés d'éléments (indexés de 0 à n-1 pour un ensemble constitué de n éléments). Parmi ces types construits, nous avons déjà vu les listes (ou tableaux). Dans ce chapitre, nous allons découvrir les tuples.
- **les dictionnaires** : contrairement aux listes, les dictionnaires n'ont pas de structure ordonnée.

### 1. Les tuples

La notion de *n-uplet* est la notion **mathématique** et **informatique** qui étend la notion de paire (x,y) ou couple (x, y), de triplet (x,y,z) à un ensemble ordonné de n nombres. Ainsi, on parle de :

- un 1-uplet est un élément unique ;
- un 2-uplet est un couple (ou doublet) ;
- un 3-uplet est un triplet ;
- un 4-uplet est un quadruplet, ..., etc.

En anglais, et donc en informatique, on parle de *tuple*.

#### a. Définition

**Un *tuple* est une séquence d'éléments finie et ordonnée de type quelconque, séparés par des virgules. Le tout est « encapsulé » dans des parenthèses.**

Un *tuple* peut aussi bien contenir des nombres entiers, des nombres décimaux, des chaînes de caractères que des booléens, ..., etc.

“Ordonné” signifie que les éléments sont indexés. Le numéro de l'élément s'appelle **l'indice** et commence par convention à 0.

#### b. Utilisation des tuples

On pourra manipuler les tuples avec les mêmes syntaxes et méthodes que les listes :

- $a = ()$  initialise un tuple vide
- $len(a)$  renvoie le nombre d'éléments du tuple  $a$
- concaténation :  $a = a + (2, )$  ajoute l'élément 2 au tuple  $a$
- comme pour les listes ou chaînes de caractères, on utilise des crochets pour accéder à des éléments du tuple :
  - $a[0]$  renvoie le premier élément du tuple,
  - $a[len(a)-1]$  renvoie le dernier (on pourra aussi utiliser  $a[-1]$ ).

### c. Exemples

Pour **définir un tuple**, on utilise des parenthèses (et non des crochets comme pour une liste).

*Exemple :*

```
# On crée un tuple
tuple1 = (0, 'coucou', 3, 'Paul')
```

Pour **accéder à un élément d'un tuple**, on utilise une notation avec des crochets en indiquant l'indice de cet élément à l'intérieur.

*Exemple :*

```
# On veut afficher l'élément 'coucou'
>>> print(tuple1[1])
coucou
```

**Remarque :** Contrairement aux listes, **les tuples ne sont pas modifiables (ou mutables)**

*Exemple :*


```
# On veut modifier l'élément 3 en 4
>>> tuple1[2] = 4
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    tuple1[2] = 4
TypeError: 'tuple' object does not support item assignment
```

Comme pour les listes, on peut effectuer des opérations.

*Exemple :*

```
# On définit deux tuples :
>>> tuple1 = (0, 'coucou', 3, 'Paul')
>>> tuple2 = (7, 8)

# On concatène ces deux tuples :
>>> tuple3 = tuple1 + tuple2
>>> print(tuple3)
(0, 'coucou', 3, 'Paul', 7, 8)
```

 Attention, pour créer un tuple avec un seul élément, il faut penser à écrire une virgule !

*Exemple :*

```
# On définit un tuple à un seul élément
>>> tuple1 = (0,)
```

## 2. Le dictionnaire

Le dictionnaire est un type de données qui n'a pas une structure ordonnée. Il est composé de **clés** auxquelles sont associées une **valeur**. Aussi bien les clés que les valeurs peuvent être de tous les types.

**Syntaxe pour définir un dictionnaire :** on utilise des **accolades**. On crée ensuite une clé en la nommant entre guillemets, à laquelle on affecte une valeur à l'aide du signe **:**.

Les éléments sont séparés par une virgule.

```
# On créé un dictionnaire
>>> dictionnaire1 = {"NOM" : "Dupont", "Prénom" : "Pierre", "Age" : "16"}
```

On peut modifier un dictionnaire en effectuant quelques opérations :

- **ajouter une clé** : on utilise des crochets

*Exemple :*

```
# On créé la clé "classe" au dictionnaire précédent
>>> dictionnaire1["Classe"] = "1F"
>>> print(dictionnaire1)
# On obtient l'affichage suivant :
{'NOM': 'Dupont', 'Prénom': 'Paul', 'Age': '16', 'classe': '1F'}
```

- **parcourir la liste des clés** : on utilise la méthode **keys**

*Exemple :*

```
# On créé la fonction suivante :
for cle in dictionnaire1.keys():
    print(cle)
# On obtient l'affichage suivant :
NOM
Prénom
Age
classe
```

- **parcourir la liste des valeurs** : on utilise la méthode **values**

*Exemple :*

```
# On créé la fonction suivante :
for valeur in dictionnaire1.values():
    print(valeur)
# On obtient l'affichage suivant :
Dupont
Paul
16
1F
```

- **parcourir la liste des clés et des valeurs** : on utilise la méthode *items*

Exemple :

```
# On créé la fonction suivante :
for cle,valeur in dictionnaire1.items():
    print(cle,valeur)
# On obtient l'affichage suivant :
NOM Dupont
Prénom Paul
Age 16
classe 1F
```

- **mettre en forme l'affichage** : on utilise la méthode *format*

Exemple :

```
# On créé la fonction suivante :
for cle,valeur in dictionnaire1.items():
    print('le {} est {}'.format(cle,valeur))
# On obtient l'affichage suivant :
le NOM est Dupont
le Prénom est Paul
le Age est 16
le classe est 1F
```

- **demander la valeur d'une clé** : on utilise la méthode *get*

Exemple :

```
# On demande la valeur associée au NOM :
print(dictionnaire1.get("NOM"))
# On obtient l'affichage suivant :
Dupont
```

- **supprimer une clé** : on utilise la méthode *del* (attention : on utilise des crochets)

Exemple :

```
# On demande la valeur associée au NOM :
del(dictionnaire1["Prénom"])
print(dictionnaire1)
# On obtient l'affichage suivant :
{'NOM': 'Dupont', 'Age': '16', 'classe': '1F'}
```

- **copier un dictionnaire de façon indépendante** : on utilise la méthode *copy* ou *deepcopy*

Exemple :

```
# On demande la valeur associée au NOM :
dictionnaire2 = dictionnaire1.copy()
dictionnaire1["Prénom"] = "Paul"
print(dictionnaire1)
print(dictionnaire2)
# On obtient l'affichage suivant :
{'NOM': 'Dupont', 'Age': '16', 'classe': '1F', 'prénom': 'Paul'}
{'NOM': 'Dupont', 'Age': '16', 'classe': '1F'}
```