1 Définitions : p-uplet nommés

> Un **p-uplet nommé** est un p-uplet pour lequel **chaque élément est identifié par un nom** plutôt que par son indice.

Exemple : Les métadonnées EXIF d'une photographie numérique pourraient par exemple être représentées par un p-uplet nommé (présentées ici au format `JSON` - JavaScript Object Notation) :

```
exif_data = {
"Version Exif" : "Exif version 2.1"
"Date" : "2003:08:11"
"Compression" : "JPEG"
"Espace colorimétrique" : "sRGB"
"Dimension X" : 2240
"Dimension Y" : 1680
}
```

> On accède alors à la **valeur** d'une donnée directement en utilisant une **clef** plutôt qu'un indice.

La "dimension X" de la photo précédente s'obtient ainsi : `exif_data["Dimension X"]`.

On aurait pu représenter les données EXIF par un simple p-uplet mais son utilisation aurait été moins agréable.

Exemple en Python :

```
exif_data = ("Exif version 2.1",  "2003:08:11", "JPEG", "sRGB", 2240, 1680)
```

Avec cette représentation, la "dimension X" de la photo précédente s'obtient ainsi : `exif_data[4]`. Ça fonctionne, mais c'est moins lisible ! Et cela nécessite de toujours se rappeler dans quel ordre les différentes données sont enregistrées.

2 Dictionnaires

> En Python, un p-uplet nommé est implémenté par un **dictionnaire**.
> Un dictionnaire est donc une **collection d'items identifiés par une clef à laquelle correspond une valeur**.

Une différence importante avec les tuples, est la possibilité de les modifier après leur création : ce sont des objets **mutables**.

Voici à travers une exemple la syntaxe pour manipuler les dictionnaires Python :

```
########## Création d'un dictionnaire #################
>>> dico  = {'Henri':(4,'roi'), 'Louis':(16, 'roi')}
>>> dico
{'Henri': (4, 'roi'), 'Louis': (16, 'roi')}

########## Accès à une valeur avec sa clef ############
>>> dico['Henri']
(4, 'roi')
```

```
########## Ajout d'un élément ##########################
>>> dico['Élisabeth'] = (1,'reine')
>>> dico
{'Henri': (4, 'roi'), 'Louis': (16, 'roi'), 'Élisabeth': (1, 'reine')}

########## Modification d'un élément ##################
>>> dico['Élisabeth'] = (2,'reine')
>>> dico
{'Henri': (4, 'roi'), 'Louis': (16, 'roi'), 'Élisabeth': (2, 'reine')}
```

Remarque importante : les **clefs** des dictionnaires doivent être **uniques** et être des objets **non mutables** (ex : tuples, chaînes de caractères, entiers).
Contre-exemple : une liste ne peut pas être une clef.

3 Itérations sur les dictionnaires

Python propose des méthodes pour accéder *directement* à l'ensemble des clefs d'un dictionnaire (**keys**), ou bien l'ensemble des valeurs (**values**), ou encore à l'ensemble des couples clefs/valeurs (**items**).

Ces méthodes spéciales sont utiles en particulier pour parcourir les éléments d'un dictionnaire.
Pour itérer sur les items complets (couples clef/valeur) :

```
>>> for (clef, valeur) in dico.items():
        print(f"La valeur {valeur} est associée à la clef {clef}")

La valeur (4, 'roi') est associée à la clef Henri
La valeur (16, 'roi') est associée à la clef Louis
La valeur (2, 'reine') est associée à la clef Élisabeth
```

Pour itérer sur les clefs :

```
>>> for clef in dico.keys():
        print(clef)

Henri
Louis
Élisabeth
```

Pour itérer sur les valeurs :

```
>>> for valeur in dico.values():
        print(valeur)

(4, 'roi')
(16, 'roi')
(2, 'reine')
```

- L'instruction `for k in dico` itère par défaut sur les clefs du dictionnaire.
- On peut créer une liste des clefs ou des valeurs avec les instructions `list(dico.keys())` et `list(dico.values())` .