# Chapitre 9 : Dictionnaires

### Définition d'un dictionnaire

Il existe deux façons de définir un dictionnaire.

La première consiste à donner littéralement l'ensemble des couples `clef:valeur` lors de sa déclaration. La collection de couples `clef:valeur` est donnée entre accolades, et chaque couple est séparé par une virgule. Un couple `clef:valeur` est spécifié littéralement en donnant la clef suivie de la valeur qui lui est associée, les deux étant séparés par deux points, soit

`dico={clef_1:valeur_1 , clef_2:valeur_2 , ...}`

Par exemple,

```
In [ ]: dico_astr={'Planete':'Terre', 'Satellite':'Lune', 'Etoile':'Soleil'}
```

La deuxième façon de définir un dictionnaire, consiste à déclarer un dictionnaire vide, puis de le remplir. La déclaration d'un dictionnaire vide est `{}`. `dico_astro2=`

L'ajout de nouveaux couples `clef:valeur` au dictionnaire `dico_astro2` ainsi initialisé se fait au moyen de la syntaxe suivante: `dico_astro2[clef] = valeur`

```
In [ ]: dico_astro2 = {}
        dico_astro2['Planete']= 'Terre'
        dico_astro2['Satellite']= 'Lune'
        dico_astro2['Etoile']= 'Soleil'
```

### Accéder à une valeur

- Dans un tableau, la valeur correspondant à la position `i` est accessible en faisant suivre le nom du tableau de l'indice entre crochets : `print(str(tab_astro[i]))`
- Dans un dictionnaire, la position est remplacé par la clef :
  `print(str(dico_astro['Satellite']))`

**Remarque** : Lorsque l'on instancie une case d'un tableau avec un indice qui dépasse la taille su tableau, l'interpréteur affiche un message d'erreur. Il en est de même si on utilise une clef non contenue dans le dictionnaire lors d'une instanciation.

Pour modifier une valeur, il suffit d'affecter une nouvelle valeur à la clef correspondante.

Pour ajouter une valeur dans un dictionnaire, il suffit d'affecter une valeur à une clef non encore utilisée.

**Accéder à la liste des clefs et valeurs**

- **keys()** : la méthode `keys()` renvoie un tableau dont les éléments sont les clefs du dictionnaire auquel la méthode s'applique.
- **list()** : on utilise la fonction `list()`pour extraire les éléments et en faire un tableau exploitable.

```
In [ ]: dico_astr={'Planete':'Terre', 'Satellite':'Lune', 'Etoile':'Soleil'}
        print(list(dico_astr.keys()))
```

- **values()** : on peut utiliser la méthode `values()` qui renvoie un tableau dont les éléments sont les valeurs du dictionnaire à laquelle la méthode s'applique :

```
In [ ]: dico_astr={'Planete':'Terre', 'Satellite':'Lune', 'Etoile':'Soleil'}
        print(list(dico_astr.values()))
```

**Supprimer une entrée (clef:valeur)**

L'instruction `del(dico[cl])` supprime du dictionnaire `dico` la clef `cl` et la valeur associée.

```
In [ ]: dico_astr={'Planete':'Terre', 'Satellite':'Lune', 'Etoile':'Soleil'}
        del(dico_astr['Etoile'])
```

**Fusionner de deux dictionnaires**

La méthode `copy()` permet de copier un dictionnaire et la méthode `dico1.update(dico2)` permet de concaténer le dictionnaire `dico2` passé en paramètre au dictionnaire `dico1`.

```
In [ ]: dic1 = {'a': 100, 'b': 200}
        dic2 = {'x': 300, 'y': 200}
        dic = dic1.copy()
        dic.update(dic2)
```

**Construire un dictionnaire à partir de deux tableaux**

Pour construire un dictionnaire il est possible de fusionner deux tableaux en utilisant la fonction `zip(clefs,valeurs)`

```
In [ ]: keys = ['red', 'green', 'blue']
        values = ['#FF0000','#008000', '#0000FF']
        color_dictionary = dict(zip(keys, values))
```

**Récapitulatif et comparaison Dictionnaire/Tableau**

| Opération | Tableau | Dictionnaire |
|---|---|---|
| Définition d'un conteneur vide | t=[] | d={} |
| Définition d'un littéral | t=[val_1, val_2, ...] | d={clef_1:val_1,clef_2:val_2, ...} |
| Appel d'une valeur | t[indice] | d[clef] |
| Ajout d'une valeur | t.append(nlle_val) | d[clef]=nlle_val |
| Modification d'une valeur | t[indice]=nlle_val | d[clef]=nlle_val |
| Nombre d'éléments | len(t) | len(d) |
| Suppression d'un élément | del(t[indice]) | del(d[clef]) |

# Le format JSON

Le format JavaScript Object Notation (JSON) est un format de fichier très répandu permettant de stocker des données sous une forme structurée. Il ne comporte que des associations clés→valeurs (à l'instar des dictionnaires), ainsi que des listes ordonnées de valeurs.

Pour manipuler les fichiers JSON, on utilise le module `json` de Python grâce à l'instruction `import json`.

La fonction `loads(texteJSON)` permet de décoder le fichier `texteJSON` passé en argument et de le transformer en dictionnaire ou en liste.

```
In [ ]: import  json
        fichier = open ("./files/bdp.json","rt")
        strjson=fichier.read()
        fichier.close()
        cours = json.loads(strjson)
        print (cours)
```

La fonction `dumps(dictionnaire,sort_keys=False)` transforme un dictionnaire ou une liste passé en paramètre en texte au format JSON. La variable `sort_keys` permet de trier (`True`) ou non (`False`) les clés du dictionnaire dans l'ordre alphabétique mise au format JSON avec une éventuelle indentation de n espaces (`indent=n`).

La fonction `dump(dictionnaire, fichier, indent =n)` permet d'enregistrer en texte au format JSON le dictionnaire directement dans le fichier précédemment ouvert, avec une éventuelle indentation.

```
In [ ]: import  json
        cours2 = {'nom cours': 'Java',
         'theme': 'Algorithmique',
         'etudiants': [{'nom': 'Martin', 'prenom': 'Jean', 'Dept': '95'},
          {'nom': 'Mohamed', 'prenom': 'Ali', 'Dept': '93'},
          {'nom': 'dupond', 'prenom': 'Bertrand', 'Dept': '95'}]}

In [ ]: fichier2 = open ("./files/coursjava.json","w")
        cours2_json = json.dumps(cours2, sort_keys=False, indent=4)
        print(cours2_json)
        json.dump(cours2, fichier2, indent=4)
        fichier2.close()
```