# Les dictionnaires et les ensembles

## *Les dictionnaires*

On a parfois besoin d'associer une valeur à une autre. Par exemple un âge à un nom. Pour cela, on peut créer 2 listes, une avec les noms et l'autres avec les âges :

```
>>> noms = ["Alice", "Bob", "Charlie"]
>>> ages = [23, 31, 56]
```

Pour obtenir l'âge de Bob, il faut faire `ages[noms.index("Bob")]`, ce qui n'est pas très lisible. De la même manière, mettre à jour son âge est tout aussi lourd. On pourrait aussi utiliser une seule liste de couples :

```
noms_ages = [('Alice', 23), ('Bob', 31), ('Charlie', 56)]
```

La recherche ou la modification des valeurs est encore plus compliquée.
C'est pourquoi de nombreux langages, dont Python, possèdent une structure spéciale appelée **dictionnaire** qui permet de faire tout cela plus simplement. Il y a plusieurs façons de créer un dictionnaire vide, ou non.

```
>>> dico_vide = {}  # Dictionnaire vide
>>> dico_vide2 = dict() # Autre définition d'un dictionnaire vide
>>> dico = {"Alice" : 23, "Bob" : 31, "Charlie" : 56} # Définition directe
>>> dico2 = dict(noms_ages) # Définition à partir d'une liste de couples
>>> dico2
{'Alice': 23, 'Bob': 31, 'Charlie': 56}
```

On peut manipuler les dictionnaires de façon assez similaire aux listes.

```
>>> dico["Alice"]  # Accès a une valeur
23
>>> dico["Bob"] = 32    # Modification d'une valeur
>>> dico["Daniel"] = 10  # Ajout d'une nouvelle valeur
>>> del dico["Charlie"]  # Suppression d'une valeur
>>> dico    # Verification
{'Alice': 23, 'Bob': 32, 'Daniel': 10}
>>> len(dico) # Longueur du dictionnaire
3
>>> "Charlie" in dico # Test d'appartenance
False
>>> "Alice" in dico
True
```

Les dictionnaires stockent deux types de données : les clefs (ici les noms) et les valeurs (ici les âges). Ce sont des types spéciaux correspondant à des listes.

```
>>> dico.keys() # liste des clefs
dict_keys(['Alice', 'Bob', 'Daniel'])
>>> dico.values() # liste des valeurs
dict_values([23, 32, 10])
>>> dico.items() # liste des couples (clef, valeur)
dict_items([('Alice', 23), ('Bob', 32), ('Daniel', 10)])
```

## *Dictionnaires, recherche et parcours*

Contrairement à une liste, tester si une clef se trouve dans le dictionnaire est quasi-immédiat, peu importe la taille du dictionnaire. C'est parce qu'en interne, les clefs sont représentées par des nombres, appelés **hachages** et qu'une table indique les valeurs associées à chaque hachage. On parle de **table de hachage**. Comme dans un vrai dictionnaire, il n'est pas nécessaire de lire toutes les pages une par une pour trouver le mot cherché. On peut très rapidement se déplacer dans les pages et ainsi trouver le mot cherché ou au contraire découvrir qu'il n'y est pas. Lorsqu'on a besoin de faire des boucles où on fait des tests d'appartenance à chaque tour et si le nombre de données à parcourir est assez grand, utiliser un dictionnaire sera bien plus efficace et rapide qu'avec une liste.
Ainsi, la deuxième version de cette fonction est plus rapide que la première.

```
def decimal_vers_bin(v):
    vus = [] # liste des restes
    rep = "" # conv en binaire
    while v not in vus:
        vus.append(v)
        d, v = fois_2(v)
        rep = rep + str(d)
    pos = vus.index(v)
    ...
```

```
def decimal_vers_bin(v):
    vus = {} # dico des restes
    rep = "" # conv en binaire
    i = 0  # indice actuel
    while v not in vus:
        vus[v] = i
        i = i + 1
        d, v = fois_2(v)
        rep = rep + str(d)
    pos = vus[v]
    ...
```

Les clefs des dictionnaires doivent être de type **immuable**. C'est-à-dire que cela doit être des valeurs qui ne peuvent pas être modifiées, comme les nombres (entiers ou réels), les n-uplets et les chaînes de caractères. Au contraire, les listes et les dictionnaires sont **mutables** et ne peuvent donc pas servir de clef.

## *Les ensembles*

Si on a juste besoin de tester l'appartenance d'une valeur à un ensemble, sans avoir besoin de récupérer une valeur associée, on peut utiliser le type **ensemble**, qui est lui aussi mutable. Cela revient essentiellement à utiliser un dictionnaire dans lequel on n'associe aucune valeur aux clefs. Si on ajoute un élément qui est déjà dans l'ensemble, il n'y a pas de doublon.

```
>>> ensemble_vide = set()      # Création d'un ensemble vide
>>> ensemble = {"Alice", "Bob", "Charlie"}    # Création directe
>>> ensemble.add("Daniel")     # Ajout d'un élément
>>> ensemble.add("Daniel")     # Ne fait rien puisque Daniel y est déjà
>>> ensemble.remove("Alice") # On enlève Alice
>>> len(ensemble)              # Nombre d'éléments dans l'ensemble
3
>>> ensemble
{'Bob', 'Daniel', 'Charlie'}
>>> "Marcel" in ensemble       # Test d'appartenance
False
```

Par contre l'ordre d'ajout n'est pas forcément conservé dans l'ensemble. C'est aussi le cas pour les dictionnaires, même si dans les dernières versions de Python, cet ordre est préservé.