

# Les dictionnaires

## Capacités attendues

- ✓ Construire une entrée de dictionnaire.
- ✓ Itérer sur les éléments d'un dictionnaire.

## 1 Découverte de la structure

Nous allons voir un nouveau type, le **dictionnaire** qui permet de regrouper des données, mais auxquelles on accède grâce à une **clé** plutôt que grâce à un indice, comme pour les tableaux et p-uplets.

### Exercice 1

1. Créer une variable `dico`, initialement vide, avec l'instruction :

```
| dico = {}
```

2. Vérifier son type, en tapant l'instruction suivante dans la console :

```
| type(dico)
```

3. Ajouter un élément au dictionnaire avec la syntaxe suivante :

```
| dico["nom"] = "Lennon"
```

Visualiser le contenu de la variable avec la console et l'écrire ici :

**'nom': 'Lennon'**

→ "nom" est une **clé** du dictionnaire, et "Lennon" une **valeur** .

4. Rajouter d'autres éléments au dictionnaire :

```
| dico["prenom"] = "John"  
| dico["naissance"] = 1940
```

→ On voit que le **type des valeurs** , comme celui des clés, peut varier (caractères, entiers, ou d'autres).

5. Consulter le contenu de `dico` et vérifier qu'il s'affiche sous la forme suivante :

```
{'nom': 'Lennon', 'prenom': 'John', 'naissance': 1940}
```

6. Que renvoie l'instruction `dico[0]` ? Expliquer.

**KeyError: 0**  
**Un dictionnaire n'a pas d'indices.**

7. Que renvoie l'instruction `dico["prenom"]` ?

```
»> dico["prenom"]  
'John'
```

C'est ainsi qu'on accède à une valeur associée à une clé.

8. Que renvoie l'instruction `len(dico)` ?

```
3
```

9. Quelles valeurs renvoient les instructions suivantes ?

- `dico.keys()` : `dict_keys(['nom', 'prenom', 'naissance'])`
- `dico.values()` : `dict_values(['Lennon', 'John', 1940])`
- `dico.items()` : `dict_items([('nom', 'Lennon'), ('prenom', 'John'), ('naissance', 1940)])`

On utilise ces fonctions pour faire des parcours de dictionnaire. Tester les instructions suivantes :

```
| for k in dico.keys():  
|     print(dico[k])
```

Modifier ensuite le code pour afficher les **clés** du dictionnaire.

10. On peut aussi créer un dictionnaire en utilisant la syntaxe suivante :

```
dictionnaire = {clé1: valeur1, clé2: valeur2}.
```

Créer un deuxième dictionnaire `dico_2` avec cette méthode, dans lequel on stocke les informations suivantes : le nom "McCartney", le prénom "Paul", et la date de naissance 1942.

## 2 Exercices

→ Pour chacun des exercices, vérifier la solution sur ordinateur, puis la noter sur votre feuille.

### Exercice 2 – (Les données EXIF d'une image)

Les photos numériques sont accompagnées de métadonnées, consultables dans leurs propriétés.

1. Représenter les données EXIF présentées ci-contre sous la forme d'un dictionnaire nommé `exif`.
2. Quelle instruction permet d'afficher la hauteur de l'image ?

```
»> exif = {"largeur": 4592, "hauteur": 2584}  
»> exif["hauteur"]  
2584
```

Images	
Dimensions	4592 x 2584
Largeur	4592 pixels
Hauteur	2584 pixels

### Exercice 3 – (Un groupe de rock)

On veut stocker la liste des membres d'un groupe de rock dans un dictionnaire rockband dont le contenu sera :

Clé	Valeurs
chanteur	Julien Casablanca
guitariste rythmique	Albert Hammond Jr
guitariste principal	Nick Valensi
bassiste	Nikolai Fraiture
batteur	Fabrizio Moretti

1. Écrire l'instruction permettant de créer le dictionnaire correspondant.

```
rockband = {"chanteur": "Julien Casablanca", "guitariste rythmique": "Albert Hammond Jr", "guitariste principal": "Nick Valensi", "bassiste": "Nikolai Fraiture", "batteur": "Fabrizio Moretti"}
```

2. Écrire l'instruction renvoyant les noms de tous les membres du groupe.

```
rockband.values()
```

3. Écrire une fonction `est_membre(rockband, nom)` renvoyant `True` si `nom` est présent dans le groupe, `False` sinon. On fera un parcours de dictionnaire.

```
def est_membre(rockband, nom):  
    for nom_membre in rockband.values():  
        if nom_membre == nom:  
            return True  
    return False
```

4. Tester si "Nick Valensi" et "Bjork" appartiennent au groupe ou non.

```
>>> est_membre(rockband, "Nick Valensi")  
True  
>>> est_membre(rockband, "Bjork")  
False
```

## 3 Cours

Dans certains cas, on peut ne pas vouloir accéder à des données par leur indice comme avec un tableau ou un p-uplet, car une autre information (souvent de type caractères, mais les autres types sont possibles) aurait plus de sens. On utilise alors un dictionnaire.

### Définition

Un dictionnaire est une **collection non-ordonnée** d'éléments.  
Ces éléments sont constitués d'une **clé** associée à une **valeur**.

## 3.1 Initialisation

- Pour créer un **dictionnaire vide**, on utilise les **accolades** (ou bien le mot-clé `dict`).
- Pour créer un **dictionnaire non-vide**, on utilise les accolades suivies de couples `cle: valeur` séparés par des virgules.

Exemples :

```
d1 = dict()      # d1 est un dictionnaire vide
d2 = {}          # d2 aussi

dico = {"algorithmes": "suite d'instructions",
        "mémoire": "espace de stockage des données",
        "processeur": "unité de calcul"} # dictionnaire à 3 couples clé-valeur
```

## 3.2 Modification

- On ajoute un élément à un dictionnaire avec la syntaxe :

```
| dico[cle] = valeur
```

où `cle` et `valeur` sont des éléments de n'importe quel type de base (entier, flottant, caractères, booléen).

**Attention, une liste ne peut pas être une clé d'un dictionnaire car ce n'est pas un type *hashable*. Une liste peut être modifiée en place et on risque donc de changer la structure du dictionnaire depuis l'extérieur.**

- On peut modifier la valeur associée à une clé déjà existante en utilisant la même syntaxe.

Exemples :

```
# ajoute un nouveau couple à dico~:
dico["programme"] = "algorithmes écrits dans un langage de programmation"

# modifie la valeur associée à la clé processeur
dico["processeur"] = "constitué d'une unité de calcul et une unité de commande"
```

## 3.3 Parcours

On peut itérer sur les éléments d'un dictionnaire, de la même manière qu'on peut le faire sur les éléments d'un tableau ou d'un `p-uplet`.

Cela permet d'appliquer des algorithmes comme ceux vus sur ces autres types construits : recherche d'occurrences, calcul de statistiques, etc.

Prenons un exemple valable :

```
d = {}
d["auteur"] = "J. R. R. Tolkien"
d["titre"] = "The Lord of the Rings"
d["genre"] = "Roman"

for key in d.keys():
    print(key)
```

On peut utiliser différentes syntaxes pour parcourir le dictionnaire d :

### Parcours sur les clés

On utilise la méthode `keys()` qui s'applique aux dictionnaires et permet d'accéder à la liste des clés, ou alors on fait le parcours sans méthode explicite :

```
for key in d.keys():  
    value = d[key]  
    print(value)
```

```
for key in d:  
    value = d[key]  
    print(key)
```

### Parcours sur les valeurs

On utilise la méthode `values()` qui s'applique aux dictionnaires et permet d'accéder à la liste des valeurs :

```
for value in d.values():  
    print(value)
```

**Attention !** On n'a alors pas accès à la clé associée à chaque valeur.

### Parcours sur les couples clé/valeur

On utilise la méthode `items()` qui s'applique aux dictionnaires et permet d'accéder à la liste des valeurs :

```
for key, value in d.items():  
    print(key, value)
```

→ Il est donc possible de n'utiliser que la méthode `keys()`, mais les autres syntaxes permettent d'alléger le code.

### Exercice 4 – (Manipulation de dictionnaires)

1.

```
dico = {"alexandre": 17, "mehdi": 18, "jeanne": 16, "charlotte": 19,  
        "sarah": 18, "noé": 19}  
  
def f(dic):  
    for cle, valeur in dic.items():  
        if valeur > 18:  
            return cle
```

Que renvoie l'appel de `f(dico)` ?

"charlotte"

2. Quelle est la valeur affichée à l'exécution du programme Python suivant ?

```
ports = { 'http': 80, 'imap': 142, 'smtp': 25 }  
ports['ftp'] = 21  
print(ports['ftp'])
```

21

```
3. | panier = [{'fruit': 'banane', 'nombre': 25}, {'fruit': 'orange', 'nombre': 124},  
              {'fruit': 'pomme', 'nombre': 75}, {'fruit': 'kiwi', 'nombre': 51}]
```

Quelle expression utiliser pour afficher le nombre de pommes ?

```
print(len(panier))
```

4. Par quelle expression remplacer les pointillés dans le programme Python suivant, pour que son exécution affiche le numéro de Dupond ?

```
repertoire = [{'nom': 'Dupont', 'tel': '5234'},  
              {'nom': 'Tournesol', 'tel': '5248'}, {'nom': 'Dupond', 'tel': '3452'}]  
  
for i in range(len(repertoire)):  
    if .....~:  
        print(repertoire[i]['tel'])
```

```
if repertoire[i]["nom"] == "Dupond":
```

### Exercice 5 – (Recherche de maximum)

Écrire une fonction `recherche_max` qui prend en paramètre un dictionnaire `dico` dont les valeurs sont des **entiers positifs** et qui renvoie la clé de la première occurrence de la plus grande valeur de `dico`.

```
def recherche_max(dico):  
    max = 0  
    for cle in dico:  
        if dico[cle] > max:  
            max = dico[cle]  
    return max
```

### Exercice 6 – (Comptage d'occurrences)

On veut écrire une fonction `compte_occurrences(chaine)` qui compte les occurrences de toutes les lettres présentes dans une chaîne de caractères `chaine`, et stocke le résultat dans un dictionnaire dont les clés sont ces lettres, et les valeurs leur nombre d'occurrences. Ce dictionnaire est renvoyé par la fonction. Par exemple :

```
>>> compte_occurrences('bonjour')  
{ 'b': 1, 'o': 2, 'n': 1, 'j': 1, 'u': 1, 'r': 1 }
```

1. Écrire l'algorithme (pseudo-code) permettant de résoudre ce problème.

h

2. Traduire cet algorithme en programme Python.

**Note :** Pour tester si une clé `cle` est présente dans un dictionnaire `dico`, on peut utiliser la syntaxe

`cle in dico` qui vaut `True` (est présente) ou `False` (n'est pas présente).

## Exercice 7 – (TP : résultats d'une élection)

Une élection est organisée, pour laquelle les votes sont enregistrés par une machine électronique. L'objectif de ce TP est d'écrire les fonctions qui vont permettre de comptabiliser automatiquement les votes et déterminer le vainqueur de l'élection.

### Représentation des données

Les votes sont enregistrés dans un tableau de chaînes de caractères. Les électeurs peuvent sélectionner parmi une liste de candidats, voter nul en entrant un nom qui ne fait pas partie des candidats ou bien voter blanc.

Par exemple, à la fin du vote, la machine a en mémoire le tableau suivant :

```
votes = ["Alan Turing", "Ada Lovelace", "Ada Lovelace", "",  
         "George Boole", "Mark Zuckerberg", ""]
```

Une chaîne de caractères vide correspond à un vote blanc.

1. Les candidats étant Alan Turing, Ada Lovelace et George Boole, quel est le vainqueur ? Combien y a-t-il de votes pour les autres candidats, de votes nuls et de votes blancs ?

h

Dans la suite, on stocke cette information dans une variable :

```
candidats = ["Alan Turing", "Ada Lovelace", "George Boole"].
```

### Dépouillement

On veut écrire une fonction `depouillement` qui prend en paramètres :

- un tableau de chaînes de caractères `candidats` ;
- un tableau de chaînes de caractères `votes`.

et qui renvoie un dictionnaire dont :

- les **clés** sont les noms des candidats (stockés dans `candidats`) et les chaînes de caractères "Nul" et "Blanc" ;
- les **valeurs** sont le nombre de bulletins associé (stockés dans `votes`).

2. Il faut commencer par créer un dictionnaire nommé `occurrences`, dont les clés sont les noms des candidats, ainsi que "Nul" et "Blanc", et les valeurs sont initialisées à 0.

Écrire le programme correspondant à cette initialisation.

→ On peut aussi utiliser la compréhension sur les dictionnaires, avec la syntaxe suivante :

```
d = {cle:valeur for cle in tableau}
```

3. Écrire une fonction `est_present(tab, v)` prenant en paramètres un tableau `tab` de type `list` Python et une valeur `v`, et renvoyant `True` si `v` est présente dans `tab`, `False` sinon.
4. Écrire la fonction `depouillement`, en se basant sur les questions précédentes et sur l'algorithme écrit ci-dessous en psuedo-code (langage naturel) :

```
def depouillement(candidats, votes):
    # initialisation du dictionnaire occurrences - question 1
    pour chaque element v de votes
        si v est présent dans candidats # fonction écrite dans la question 2
            incrémenter la valeur associée à la clé v dans occurrences
        sinon si v vaut ""
            incrémenter la valeur associée à la clé "Blanc" dans occurrences
        sinon
            incrémenter la valeur associée à la clé "Nul" dans occurrences
    renvoyer occurrences
```

### Note

La syntaxe `if v in tab` pourrait être utilisée ligne 4. Cela revient à écrire une fonction de recherche d'occurrence dans un tableau renvoyant un booléen. Vous pourrez l'utiliser par la suite, mais il faut **savoir écrire la fonction de recherche d'occurrence** !

5. Tester cette fonction avec les variables `candidats` et `votes` définies dans l'introduction. Vérifier qu'on obtient bien le dictionnaire suivant :

```
{'Alan Turing': 1, 'Ada Lovelace': 2, 'George Boole': 1, 'Blanc': 2, 'Nul': 1}
```

### Trouver le vainqueur

On considère qu'il n'y a qu'un seul vainqueur à l'élection, c'est-à-dire qu'un candidat a obtenu plus de votes que tous les autres.

1. Réécrire ci-dessous un programme permettant de trouver le maximum d'un tableau de nombres `t` (vous n'avez pas besoin de l'écrire sur l'ordinateur):

`h`

2. Adapter ce programme pour écrire une fonction `vainqueur` qui recherche le maximum des valeurs stockées cette fois-ci dans un dictionnaire : ce sera la fonction `vainqueur(d)`, avec `d` ayant pour clés les candidats (chaînes de caractères) et pour valeurs le nombre de votes qui leur est associé (obtenu avec la fonction `depouillement` de la partie A).

Cette fonction renvoie la clé associée à la plus grande valeur. On prendra soin d'exclure les votes nuls et blancs.

3. Appliquer la fonction `vainqueur` au dictionnaire obtenu à la question 5, et stocker le résultat du vote dans une variable `resultat`.

Vérifier qu'Ada Lovelace est bien la grande gagnante de ce scrutin.