# TP : Premiers pas avec les dictionnaires

L'objectif est d'apprendre à créer, consulter, modifier et parcourir un dictionnaire Python, puis de l'utiliser pour compter des occurrences et repérer des doublons. Les listes manipulées contiennent des entiers ou des chaînes de caractères. Les fonctions ne doivent pas modifier leurs arguments, sauf indication contraire.

## 1 Découvrir les dictionnaires

Un dictionnaire associe une **valeur** à chaque **clé**. Dans l'exemple suivant, les clés sont les prénoms des élèves et les valeurs sont leurs notes.

```
notes = {"Alice": 12, "Basile": 15, "Chloe": 9}
```

Contrairement à une liste, on consulte un dictionnaire à l'aide d'une clé, qui n'est pas nécessairement un entier :

```
>>> notes["Alice"]
12
```

Chaque clé apparaît une seule fois dans un dictionnaire ; plusieurs clés peuvent être associées à une même valeur.

**Question 1.** Créer le dictionnaire `notes` ci-dessus. Écrire une instruction permettant de consulter la note de Basile.

Une affectation de la forme `d[cle] = valeur` ajoute la clé si elle est absente, ou remplace la valeur associée si elle est déjà présente.

**Question 2.** Remplacer la note de Chloe par 11, puis ajouter Diane, dont la note est 15. Afficher le dictionnaire obtenu.

**Question 3.** La fonction `len` donne le nombre de clés d'un dictionnaire. Déterminer le nombre d'élèves enregistrés après la question précédente.

L'expression `cle in d` teste la présence d'une **clé** dans le dictionnaire et renvoie un booléen.

**Question 4.** Prévoir, puis vérifier les résultats des trois expressions suivantes : `"Alice"in notes`, `"Emma"in notes` et `15 in notes`.

**Question 5.** Exécuter `notes["Emma"]`. Que se passe-t-il ?

## 2 Parcourir un dictionnaire

La boucle suivante parcourt les **clés** du dictionnaire :

```
for prenom in notes:
    print(prenom)
```

Dans le corps de cette boucle, `notes[prenom]` donne la note de l'élève courant. Pour les exemples de cette section, on repart du dictionnaire initial :

```
notes = {"Alice": 12, "Basile": 15, "Chloe": 9}
```

**Question 6.** Écrire une boucle qui affiche chaque prénom suivi de la note correspondante.

**Question 7.** Écrire une fonction `moyenne(notes)` qui renvoie la moyenne des notes. Le dictionnaire est supposé non vide.

```
>>> moyenne(notes)
12.0
```

**Question 8.** Écrire une fonction `admis(notes)` qui renvoie la liste des prénoms des élèves dont la note est supérieure ou égale à 10. Conserver l'ordre du parcours du dictionnaire.

```
>>> admis(notes)
['Alice', 'Basile']
>>> admis({})
[]
```

## 3 Construire un dictionnaire

L'instruction `d = {}` crée un dictionnaire vide. On peut ensuite y ajouter les associations une à une.

**Question 9.** Écrire une fonction `construire_notes(prenoms, valeurs)` qui renvoie un dictionnaire associant chaque prénom à la note située au même indice. Les deux listes ont la même longueur et les prénoms sont distincts.

```
>>> construire_notes(["Alice", "Basile", "Chloe"], [12, 15, 9])
{'Alice': 12, 'Basile': 15, 'Chloe': 9}
>>> construire_notes([], [])
{}
```

**Question 10.** Écrire une fonction `relever(notes, seuil)` qui renvoie un nouveau dictionnaire contenant seulement les élèves dont la note est supérieure ou égale au seuil. Le dictionnaire reçu en argument ne doit pas être modifié.

```
>>> relever(notes, 12)
{'Alice': 12, 'Basile': 15}
>>> relever(notes, 18)
{}
```

## 4 Compter des occurrences et repérer des doublons

On veut maintenant mémoriser des informations sur les éléments d'une liste. Un dictionnaire permet de retrouver directement l'information associée à un élément déjà rencontré.

**Question 11.** Écrire une fonction `occurrences(L)` qui renvoie un dictionnaire associant à chaque élément de `L` son nombre d'apparitions. Partir d'un dictionnaire vide et parcourir la liste. Si l'élément courant est déjà une clé, augmenter son compteur ; sinon, créer une entrée de valeur 1.

```
>>> occurrences(["chat", "chien", "chat", "chat", "chien"])
{'chat': 3, 'chien': 2}
>>> occurrences([4, 7, 4, 2, 7, 4])
{4: 3, 7: 2, 2: 1}
>>> occurrences([])
{}
```

**Question 12.** Écrire une fonction `contient_doublon(L)` qui renvoie `True` si la liste contient au moins deux éléments égaux, et `False` sinon. Utiliser un dictionnaire pour mémoriser les éléments déjà rencontrés et arrêter le parcours dès qu'un doublon est détecté. On pourra associer la valeur `True` à chaque élément mémorisé : seule la présence de la clé nous intéresse ici.

```
>>> contient_doublon([4, 7, 2, 4])
True
>>> contient_doublon([4, 7, 2])
False
>>> contient_doublon([])
```

```
False
```

**Question 13.** Écrire une fonction `sans_doublon(L)` qui renvoie une nouvelle liste contenant chaque élément de `L` une seule fois, dans son ordre de première apparition. Utiliser un dictionnaire pour mémoriser les éléments déjà ajoutés.

```
>>> sans_doublon([4, 7, 2, 4, 7, 3])
[4, 7, 2, 3]
>>> sans_doublon(["chat", "chien", "chat"])
['chat', 'chien']
>>> sans_doublon([])
[]
```