Semestre 1 / Cours 4 - Dictionnaires

ITC MPSI & PCSI – Année 2025-2026

Sommaire

Définition

Rappel :

Si `L` est une liste, ses éléments sont indicés par des entiers naturels 0,1,2...

Par exemple : si `L = [125,'a','toto',158]`, alors :

```
L[0] = 125
L[1] = 'a'
L[2] = 'toto'
L[3] = 158
```

Avec un dictionnaire, les indices peuvent ne pas être des entiers.

Ces indices sont appelés des **clés**.

La création d'un dictionnaire peut se faire en fournissant entre accolades une liste de termes de la forme `clé : valeur`. Par exemple :

```
>>> D = {'abricot' : 5 , 'ananas' : 2 , 'pomme' : 8}
```

crée un dictionnaire stocké dans la variable `D`.

Avec ce dictionnaire :

```
>>> D = {'abricot' : 5 , 'ananas' : 2 , 'pomme' : 8}
```

on dispose de trois valeurs 5, 2 et 8 indicées respectivement par les clés `'abricot'`, `'ananas'` et `'pomme'`. La valeur correspondant à une clé donnée s'obtient comme pour les listes :

```
>>> D['ananas']
2
```

Avec ce dictionnaire :

```
>>> D = {'abricot' : 5 , 'ananas' : 2 , 'pomme' : 8}
```

on peut modifier la valeur associée à une clé :

```
>>> D['abricot'] = 10
>>> D
{'abricot': 10, 'ananas': 2, 'pomme': 8}
```

Avec ce dictionnaire :

```
>>> D = {'abricot' : 10 , 'ananas' : 2 , 'pomme' : 8}
```

on peut ajouter une nouvelle clé avec la valeur associée :

```
>>> D['cerise'] = 15
>>> D
{'abricot': 10, 'ananas': 2, 'pomme': 8, 'cerise': 15}
```

Exercice 1 :

Créer un dictionnaire dont les clés sont les 26 lettres minuscules de l'alphabet et la valeur associée à chaque clé est sa place dans l'alphabet :

```
D = {'a':1 , 'b':2 , ...}
```

Indication :

```
>>> chr(97)
'a'
>>> chr(98)
'b'
```

Exercice 1 :

Créer un dictionnaire dont les clés sont les 26 lettres minuscules de l'alphabet et la valeur associée à chaque clé est sa place dans l'alphabet :

```
D = {'a':1 , 'b':2 , ...}
```

Indication :

```
>>> chr(97)
'a'
>>> chr(98)
'b'
```

- Un premier dictionnaire

```
D = {}
for k in range(1, 27):
    D[chr(96+k)] = k
```

Parcours d'un dictionnaire

Il est possible de parcourir les données stockées dans un dictionnaire en itérant sur les clés :

```
>>> D = {'abricot': 10, 'ananas': 2, 'pomme': 8, \
↪ 'cerise': 15}
>>> for c in D.keys() :
...     print(c)
...
abricot
ananas
pomme
cerise
```

- On peut remplacer l'instruction `for c in D.keys()` par `for c in D`

Il est possible de parcourir les données stockées dans un dictionnaire en itérant sur les clés :

```
>>> D = {'abricot': 10, 'ananas': 2, 'pomme': 8, \
↪ 'cerise': 15}
>>> for c in D.keys() :
...     print(c)
...
abricot
ananas
pomme
cerise
```

- On peut remplacer l'instruction `for c in D.keys()` par `for c in D`
- Par défaut, ce sont les clés qui sont parcourues.

Il est possible de parcourir les données stockées dans un dictionnaire en itérant sur les valeurs :

```
>>> D = {'abricot': 10, 'ananas': 2, 'pomme': 8, \
↪ 'cerise': 15}
>>> for v in D.values() :
...     print(v)
...
10
2
8
15
```

Il est possible de parcourir les données stockées dans un dictionnaire en itérant sur les couples clé,valeur :

```
>>> D = {'abricot': 10, 'ananas': 2, 'pomme': 8, \
↪ 'cerise': 15}
>>> for c,v in D.items() :
...     print(c)
...     print(v)
...
abricot
10
ananas
2
pomme
8
cerise
15
```

Attention : Il est important de noter que les éléments d'un dictionnaire ne sont pas ordonnés. Donc, pour le parcours ou l'affichage des clés ou des valeurs, il n'y a aucun ordre prévisible !

Exercice 2 :

On considère deux dictionnaires `D1` et `D2`, et on suppose que les clés de `D1` sont différentes des clés de `D2`. Créer un dictionnaire `D` obtenu en concaténant les deux dictionnaires `D1` et `D2`.

Exercice 2 :

On considère deux dictionnaires D1 et D2, et on suppose que les clés de D1 sont différentes des clés de D2. Créer un dictionnaire D obtenu en concaténant les deux dictionnaires D1 et D2.

Concaténation de deux dictionnaires

```
D = {}
for c in D1:
    D[c] = D1[c]
for c in D2:
    D[c] = D2[c]
```

Opérations élémentaires sur un dictionnaire

Dictionnaire vide

```
>>> D = {}
>>> type(D)
<class 'dict'>
```

Longueur

```
>>> D = {'toto':2 , 'tata':3 , 'titi':5}
>>> len(D)
3
```

Suppression

```
>>> D = {'toto':2 , 'tata':3 , 'titi':5}
>>> del(D['tata'])
>>> D
{'toto': 2, 'titi': 5}
```

Suppression et renvoie

```
>>> D = {'toto':2 , 'tata':3 , 'titi':5}
>>> D.pop('titi')
5
>>> D
{'toto': 2, 'tata': 3}
```

Tester la présence d'une clef

```
>>> D = {'toto':2 , 'tata':3 , 'titi':5}
>>> 'tata' in D   # Est-ce que 'tata' est une clé de \
↪ D ?
True
>>> 'tata' in D.keys() # Même question
True
>>> 7 in D.values() # Est-ce que 7 est une valeur de \
↪ D ?
False
```

Exercice 3 :

Créer une fonction `CompteLettres` à qui on fournit une chaine de caractères et qui renvoie un dictionnaire dont les clés sont les lettres présentes dans la chaine, et pour chaque clé, la valeur est le nombre de fois où la lettre apparaît. Par exemple,
`CompteLettres('abracadabra')` renverrait le dictionnaire

```
D = { 'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1 }
```

Exercice 3 :

Créer une fonction `CompteLettres` à qui on fournit une chaine de caractères et qui renvoie un dictionnaire dont les clés sont les lettres présentes dans la chaine, et pour chaque clé, la valeur est le nombre de fois où la lettre apparaît.

Comptage de lettres

```
def CompteLettres(ch:str)->dict:
    D = {}
    for c in ch:
        if c in D:
            D[c] += 1
        else:
            D[c] = 1
    return D
```

Exercice 4 :

On considère dans cet exercice deux dictionnaires D1 et D2 dont les valeurs sont toutes des nombres réels. Améliorer l'exercice 2 pour pouvoir concaténer les deux dictionnaires D1 et D2. Lorsqu'une clé figure dans chacun des deux dictionnaires, on fera la somme des deux valeurs.

Exercice 4 :

On considère dans cet exercice deux dictionnaires D1 et D2 dont les valeurs sont toutes des nombres réels. Améliorer l'exercice 2 pour pouvoir concaténer les deux dictionnaires D1 et D2. Lorsqu'une clé figure dans chacun des deux dictionnaires, on fera la somme des deux valeurs.

Concaténation de deux dictionnaires

```
D={}
for c in D1:
    D[c] = D1[c]
for c in D2:
    if c in D:
        D[c] += D2[c]
    else:
        D[c] = D2[c]
```

Exercice 5 :

On considère un dictionnaire D dont les éléments sont de la forme NOM:NOTE, où NOM est le nom d'un élève et NOTE est sa moyenne générale (sur 20). Créer deux dictionnaires D1 et D2 où D1 (respectivement D2) est la partie de D dont les clés sont les noms des élèves ayant une note <10 (respectivement ≥ 10).

Exercice 5 :

On considère un dictionnaire D dont les éléments sont de la forme NOM:NOTE, où NOM est le nom d'un élève et NOTE est sa moyenne générale (sur 20). Créer deux dictionnaires D1 et D2 où D1 (respectivement D2) est la partie de D dont les clés sont les noms des élèves ayant une note <10 (respectivement ≥ 10).

Partition d'un dictionnaire

```
D1 = {}
D2 = {}
for c in D:
    if D[c] < 10:
        D1[c] = D[c]
    else:
        D2[c] = D[c]
```

Copie d'un dictionnaire

Origine du problème

- Une affectation `<var> = <dict>` ne duplique pas l'objet `<dict>`, comme pour les listes.

Origine du problème

- Une affectation `<var> = <dict>` ne duplique pas l'objet `<dict>`, comme pour les listes.
- Mais elle crée une deuxième étiquette sur cet objet.

Origine du problème

- Une affectation `<var> = <dict>` ne duplique pas l'objet `<dict>`, comme pour les listes.
- Mais elle crée une deuxième étiquette sur cet objet.
- Les dictionnaires étant des objets mutables, ceci peut donner encore des surprises.

Origine du problème

- Une affectation `<var> = <dict>` ne duplique pas l'objet `<dict>`, comme pour les listes.
- Mais elle crée une deuxième étiquette sur cet objet.
- Les dictionnaires étant des objets mutables, ceci peut donner encore des surprises.

```
>>> D1 = {1 : 2, 2: 3, 3:[1, 2, 3]}
>>> D2 = D1
>>> D1[1] = 3
>>> D1[3][0] = -1
>>> D1
{1: 3, 2: 3, 3: [-1, 2, 3]}
>>> D2
{1: 3, 2: 3, 3: [-1, 2, 3]}
```

Comment dupliquer un dictionnaire ?

On utilise **deepcopy** du module **copy**.

Comment dupliquer un dictionnaire ?

On utilise **deepcopy** du module **copy**.

```
>>> D1 = {1 : 2, 2: 3, 3:[1, 2, 3]}
>>> from copy import deepcopy
>>> D2 = deepcopy(D1)
>>> D1[0] = 0
>>> D1[3][0] = 0
>>> D1[1] = 3
>>> D1[3][0] = -1
>>> D1
{1: 3, 2: 3, 3: [-1, 2, 3], 0: 0}
>>> D2
{1: 2, 2: 3, 3: [1, 2, 3]}
```