# Dictionnaires

**Cours**

Un dictionnaire est un type abstrait de données constitué d'éléments sous forme de couples **clé-valeur** (*key-value* en anglais).

Un dictionnaire n'a **pas d'ordre**, on accède à **chaque valeur par sa clé**.

La clé d'un dictionnaire peut être un mot (de type `str`), un nombre (de type `int` ou `float`), un p-uplet, etc., mais pas un tableau (type `list`) ni un autre dictionnaire car ce sont des types muables.

## Interface

Les principales primitives constituant l'interface d'un dictionnaire sont :

- `creer() → dict` : construire d'un dictionnaire vide.
- `est_vide() → bool` : vérifier si un dictionnaire est vide ou non.
- `ajouter(clé, valeur)` : ajouter un élément formé du couple clé-valeur.
- `supprimer(clé) → valeur` : supprimer un élément identifié par sa clé.
- `lire(clé) → valeur` : lire la valeur associée à une clé.
- `taille() → int` : renvoyer le nombre d'éléments dans un dictionnaire.

## Implémentation

Prenons pour exemple de vouloir stocker les capitales de plusieurs pays dans une structure de données. Ni les tableaux, ni les listes chaînées ne semblent offrir une solution efficace :

- Les pays ne sont pas classés dans un ordre particulier, un tableau ne permettrait donc pas d'accéder facilement à la capitale d'un pays.
- Une liste chaînée aurait l'inconvénient de devoir parcourir toute la liste pour accéder à la capitale d'un pays.

Dans ce cas, un dictionnaire est la structure de donnée la mieux adaptée.

Python possède naturellement une structure de dictionnaire (le type `dict`). Mais ce n'est pas le cas dans tous les langages, plusieurs implémentations sont alors possibles, par exemple en utilisant une fonction de hachage.

### Avec le type `dict` de Python

Les dictionnaires Python, de type `dict`, offrent immédiatement toutes les primitives d'un dictionnaire :

```
>>> capitale = {}           # creer()
>>> len(capitale) == 0      # est_vide()
True
>>> capitale["France"] = "Paris"     # ajouter(cle, valeur)
>>> capitale["Allemagne"] = "Berlin"     # ajouter(cle, valeur)
>>> capitale["Italie"] = "Rome"     # ajouter(cle, valeur)
>>> capitale["France"]              # lire(cle)
'Paris'
>>> len(capitale)           # taille()
3
>>> capitale.pop("Italie")          # supprimer(clé)
'Rome'
```

Rappelons quelques unes des autres fonctionalités vues en classe de première :

- Un dictionnaire peut être créé avec plusieurs couples de clés-valeurs séparés par des virgules, le tout encadré par des accolades “{ }” :

  ```
  >>> capitale = {'France': 'Paris', 'Allemagne': 'Berlin', 'Italie': 'Rome'}
  ```

- Les éléments sont affichés sans ordre particulier :

  ```
  >>> capitale
  {'Allemagne': 'Berlin', 'France': 'Paris', 'Italie': 'Rome'}
  ```

- Il est possible d'accèder à une valeur **par sa clé, et uniquement par sa clé, mais pas par sa position**. Les valeurs d'un dictionnaire n'ont pas de position :

  ```
  >>> capitale["France"]
  'Paris'
  >>> capitale[0]
  Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
      capitale[0]
  KeyError: 0
  ```

  Dans ce cas, l'interpréteur renvoie un message d'erreur car la clé `0` n'existe pas dans le dictionnaire `capitale`.

- La méthode `.get()` permet aussi de récupérer une valeur associée à une clé, mais sans lever d'erreur si la clé n'existe pas :

  ```
  >>> capitale.get('France')
  'Paris'
  >>> capitale.get(0)
  >>> capitale.get('Espagne', 'Non défini')
  Non défini
  ```

- Le mot clé `in` permet de vérifier si une clé est présente dans un dictionnaire.

  ```
  >>> "France" in capitale
  True
  >>> "Espagne" in capitale
  False
  ```

  ⚠️ Les mot clé `in` s'applique seulement aux clés d'un dictionnaire, pas à ses valeurs :

```
>>> "Paris" in capitale
False
```

- De la même façon, `for` permet d'itérer sur les clés d'un dictionnaire :

```
>>> for cle in capitale:
...     print(capitale[cle], 'est en', cle)
...
Berlin est en Allemagne
Paris est en France
Rome est en Italie
```

1

- Les méthodes `.keys()` et `.values()` renvoient les clés et les valeurs d'un dictionnaire :

```
>>> capitale.keys()
dict_keys(['Allemagne', 'France', 'Italie'])
>>> capitale.values()
dict_values(['Berlin', 'Paris', 'Rome'])
```

et la méthode `.items()` renvoie tous les couples de clé-valeur d'un dictionnaire :

```
>>> capitale.items()
dict_items([('Allemagne', 'Berlin'), ('France', 'Paris'), ('Italie', 'Rome')])
```

⚠️ Les objets renvoyés par ces trois méthodes sont de types Python un peu particuliers : `dict_keys`, `dict_values` et `dict_items`. Pour les utiliser, il faut souvent les transformer en tableaux avec la fonction `list()` :

```
>>> list(capitale.values())
['Berlin', 'Paris', 'Rome']
>>> list(capitale.items())
[('Allemagne', 'Berlin'), ('France', 'Paris'), ('Italie', 'Rome')]
```

- Il est possible de modifier la valeur associée à une clé existante (rappel : si la clé n'existe pas, un nouvel élément est créé dans le dictionnaire) :

```
capitale["Italie"] = "Roma"
```

- La méthode `.pop(clé)` ou l'instruction `del` permettent de supprimer un élément :

```
>>> capitale
{'Allemagne': 'Berlin', 'France': 'Paris', 'Italie': 'Roma'}
>>> capitale.pop("Italie")
>>> capitale
{'Allemagne': 'Berlin', 'France': 'Paris'}
>>> del capitale["France"]
>>> capitale
{'Allemagne': 'Berlin'}
```

Ou encore le dictionnaire entier avec l'instruction `del capitale`, dans ce cas la variable `capitale` n'existe plus.

- Autre particularité vue en classe de première, un dictionnaire peut être créé par compréhension :

```
>>> carres = {x:x**2 for x in range(10)}
>>> carres
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

- Enfin, les dictionnaires sont de types muables ⚠️, il faut donc faire particulièrment attention pour copier un dictionnaire ou passer un dictionnaire en argument d'une fonction[2].

## Avec un tableau et une fonction de hachage

Commençons par créer une classe `Dico`, stockant les valeurs d'un dictionnaire dans un tableau de 100 valeurs :

```
class Dico:

    def __init__(self):
        self.t = [None] * 100
```

À quelle position du tableau enregistrer les noms de capitale associés à chaque pays ? C'est le rôle d'une fonction de hachage qui permet de transformer la clé, ici le nom du pays, en un indice du tableau.

**Cours**

Une fonction de hachage est un algorithme mathématique qui transforme une valeur donnée (par exemple une chaîne de caractère ou un autre type de donnée) en une chaîne alphanumérique, appelée valeur de hachage ou *hash* en anglais.

L'opération inverse qui permet de retrouver la valeur initiale à partir de la valeur de hachage est en principe difficile à réaliser.

Il existe des méthodes mathématiques complexes[3] pour définir des fonctions de hachage efficaces. Pour notre exemple, nous utilisons une fonction très simple qui additionne les valeurs Unicode de chaque lettre, le tout modulo 100 :

```
def hachage(chaine):
    hash = 0
    for c in chaine:
        hash += ord(c)
    return hash % 100
```

Le nombre renvoyé par cette fonction de hachage sera l'indice dans le tableau. Par exemple, la capitale de la France sera stockée dans le tableau à la position d'indice 91, la capitale de l'Allemagne à l'indice 2, etc. :

```
>>> hachage("France")
91
>>> hachage("Allemagne")
2
```



Note: Il est très difficile de retrouver, à partir d'une valeur de hachage par exemple `2`, quelle était la chaîne d'origine, ici `"Allemagne"`.

Ajoutons les primitives d'un dictionnaire à la classe `Dico` :

Ecole Internationale PACA | CC-BY-NC-SA 4.0

```
class Dico:

    def __init__(self):
        self.t = [None] * 100


    def ajouter(self, cle, valeur):
        self.t[hachage(cle)] = valeur

    def lire(self, cle):
        return self.t[hachage(cle)]

    def supprimer(self, cle):
        self.t[hachage(cle)] == None

    def taille(self):
        l = 0
        for elem in self.t:
            if elem is not None: l += 1
        return l

    def est_vide(self):
        return self.taille() == 0
```

et créons le dictionnaire des capitales :

```
>>> capitale = Dico()
>>> capitale.ajouter('France', 'Paris')
>>> capitale.ajouter('Allemagne', 'Berlin')
>>> capitale.ajouter('Italie', 'Rome')
>>> capitale.lire('France')
Paris
```

Il y a une « collision » quand la fonction de hachage n'est pas assez performante, comme ici, et renvoie la même valeur de hachage pour deux clés différentes :

```
>>> hachage("Danemark")
3
>>> hachage("Irlande")
3
```

Dans ce cas, l'implémentation du dictionnaire doit permettre de gérer la collision, par exemple en faisant pointer l'indice correspondant à plusieurs clés vers une liste chaînée contenant toutes les clés-valeurs partageant ce même indice.

---

1. Il est aussi possible d'utiliser la méthode `items()` et d'écrire : `>>> for cle, val in 'capitale.items(): val, 'est en', cle)` ↩
2. ⚠️ Attention au signe `=` pour copier un dictionnaire :

```
>>> d1 = {'one':1, 'two':2, 'three':3}
>>> d2 = d1
>>> d2['three'] = 4
>>> d1
{'one':1, 'two':2, 'three':4}
```

`d1` a aussi été modifiée quand on a modifié `d2` ! Les deux variables `d1` et `d2` sont en fait deux noms qui font référence vers le même objet. Pour remédier à ce problème, il faut utiliser aussi la fonction `dict()` qui renvoie un nouveau dictionnaire :

```
>>> d1 = {'one':1, 'two':2, 'three':3}
>>> d2 = dict(d1)
>>> d2['three'] = 4
>>> d1
{'one':1, 'two':2, 'three':3}
>>> d2
{'one':1, 'two':2, 'three':4}
```

Ou encore utiliser la méthode `.copy()` :

```
>>> d1 = {'one':1, 'two':2, 'three':3}
>>> d2 = d1.copy()
```

Note: Si le dictionnaire contient des tableaux, les deux méthodes ci-dessus ne fonctionnent plus, il faut utiliser `deepcopy`.

⚠️ Attention aussi aux dictionnaires passés en paramètre de fonction :

```
def test(var):
    print("Adresse de la variable passée en argument: ", hex(id(var))  )
    var['3'] = 3
    print("Adresse de la variable une fois modifiée: ", hex(id(var)))

d = {'1': 1, '2':2}
print("Adresse de la variable dictionnaire: ", hex(id(d)) )
test(d)
print("dictionnaire=", d)
```

↩

3. Il existe de nombreux algorithme de hachage :
   - MD5 : un des premiers algorithmes de hachage, il est aujourd'hui considéré comme peu sûr.
   - SHA-1 (SHA est l'acronyme de Secure Hashing Algorithm) : il n'est plus recommandé pour le stockage sécurisé des mots de passe car il est sujet à des attaques.
   - SHA-2 : une famille d'algorithmes de hachage, comprenant SHA-256 et SHA-512. Plus sécurisé que SHA-1, il est aussi très utilisé en cryptographie, par exemple, le Bitcoin utilise SHA-256.
   - NTLM : utilisé dans les environnements Microsoft Windows.

   ↩