

Rappels sur les dictionnaires en Python

Fiche de synthèse

① DÉFINITION

Définition 1 (Dictionnaire) Un **dictionnaire** est un ensemble de couples (clé, valeur). On accède à une valeur par sa **clé**, et non par une position comme dans une liste.

Dans un annuaire, les noms sont les clés et les numéros de téléphone les valeurs. Chaque clé est unique ; une même valeur peut apparaître plusieurs fois. Un dictionnaire est **mutable** : on peut ajouter, modifier et supprimer des couples.

② CRÉER UN DICTIONNAIRE

```
d = {} # dictionnaire vide
films = {'Aladdin': 569439, 'Rocketman': 277017}
couples = [('a', 1), ('b', 2)]
d2 = dict(couples) # {'a': 1, 'b': 2}
doubles = {k: 2 * v for k, v in d2.items()}
```

La dernière ligne construit un dictionnaire **par compréhension** : `{'a': 2, 'b': 4}`.

③ OPÉRATIONS

Instruction	Effet
<code>films['Aladdin']</code>	valeur associée à la clé
<code>films['Up'] = 120</code>	ajoute le couple, ou remplace la valeur si la clé existe déjà
<code>del films['Up']</code>	supprime le couple
<code>'Up' in films</code>	True si la clé est présente
<code>len(films)</code>	nombre de couples

④ PARCOURIR UN DICTIONNAIRE

```
for film in films: # les clés
    print(film)
for n in films.values(): # les valeurs
    print(n)
for film, n in films.items(): # les couples
    print(f"{film} : {n} entrées")
```

Parcourir `films` ou `films.keys()` donne les clés, dans l'ordre où elles ont été ajoutées.

⑤ CLÉ ABSENTE

Lire une clé absente provoque une erreur `KeyError`. Deux façons de l'éviter :

```
try:
    n = films['Mon film']
except KeyError:
    n = 0

n = films.get('Mon film', 0)
```

Python exécute le bloc `try` ; si une erreur `KeyError` s'y produit, il exécute le bloc `except` au lieu d'arrêter le programme. La méthode `get(cle, default)` renvoie la valeur associée à la clé, ou `default` si la clé est absente (`None` si on ne donne pas de second paramètre).

⑥ TYPES DES CLÉS ET DES VALEURS

- Une **clé** doit être d'un type non mutable : entier, chaîne de caractères, tuple. Une liste ne peut pas servir de clé : `{[1, 2]: 'a'}` provoque l'erreur `TypeError: unhashable type: 'list'`.
- Une **valeur** peut être de n'importe quel type, y compris une liste ou un autre dictionnaire.

```
grille = {(0, 0): 'T', (0, 1): 'S'}
thon = {'espece': 'T', 'gestation': 3}
mer = {(0, 0): thon}
print(mer[(0, 0)]['espece']) # affiche T
```

⑦ RECHERCHE : LISTE OU DICTIONNAIRE

- Chercher un élément dans une **liste** demande, dans le pire cas, de parcourir tous ses éléments : la complexité est en $O(n)$.
- Chercher une clé dans un **dictionnaire** est en $O(1)$ en moyenne : une **fonction de hachage** transforme la clé en un indice de tableau, ce qui donne directement l'emplacement de la valeur, sans parcours.

Pour $n = 10\,000\,000$, le test `n - 1 in grande_liste` prend quelques centièmes de seconde, le même test sur un dictionnaire environ une microseconde.

À retenir.

- Un dictionnaire associe des valeurs à des clés uniques et non mutables.
- On ajoute ou remplace par `d[cle] = valeur`, on supprime par `del d[cle]`, on teste la présence par `cle in d`.
- On parcourt les clés, `.values()` ou `.items()`.
- Une clé absente se traite avec `try/except KeyError` ou avec `get`.
- La recherche d'une clé est en $O(1)$ en moyenne, contre $O(n)$ dans une liste.