

Fiche méthode : dictionnaires

1. L'essentiel de la syntaxe

Un dictionnaire associe des **clés** à des **valeurs**. On accède à une valeur par sa clé, comme on accède à un élément d'une liste par son indice, mais les clés peuvent être des chaînes, des entiers, des tuples...

Opération	Écriture	Coût
dictionnaire vide	<code>d =</code>	$O(1)$
ajouter ou modifier une valeur	<code>d[cle] = valeur</code>	$O(1)$
lire une valeur (la clé doit exister)	<code>d[cle]</code>	$O(1)$
tester si une clé est présente	<code>cle in d</code>	$O(1)$
supprimer une clé	<code>del d[cle]</code>	$O(1)$
nombre de clés	<code>len(d)</code>	$O(1)$
parcourir les clés	<code>for cle in d:</code>	$O(n)$
parcourir les couples	<code>for cle, v in d.items():</code>	$O(n)$
parcourir les valeurs	<code>for v in d.values():</code>	$O(n)$

● Méthode

- `cle in d` est en temps constant, alors que `x in L` est en $O(n)$: c'est souvent la raison d'utiliser un dictionnaire.
- Les clés doivent être non modifiables : `str`, `int`, `float`, `tuple`. Une liste ne peut pas être une clé.
- `for cle in d:` parcourt les **clés**, pas les valeurs.

2. Schéma « compter »

```
def occurrences(L: list) -> dict:
    '''Renvoie le dictionnaire élément -> nombre d'apparitions dans L.'''
    d = {}
    for x in L:
        if x in d:
            d[x] = d[x] + 1      # déjà vu : on incrémente
        else:
            d[x] = 1             # première apparition : on crée la clé
    return d
```

3. Schéma « regrouper »

Chaque clé est associée à une **liste** que l'on complète.

```
def par_longueur(mots: list) -> dict:
    '''Renvoie le dictionnaire longueur -> liste des mots de cette longueur.'''
    d = {}
    for mot in mots:
        k = len(mot)            # la clé du groupe
        if k not in d:
            d[k] = []           # premier mot du groupe : liste vide
        d[k].append(mot)
    return d
```

4. Schéma « clé de valeur maximale »

C'est le schéma « meilleur élément » de la fiche méthode sur le maximum, en parcourant les clés.

```
def cle_max(d: dict) -> object:
    '''d non vide. Renvoie une clé de d de valeur maximale.'''
    meilleure = None
    for cle in d:
        if meilleure is None or d[cle] > d[meilleure]:
            meilleure = cle
    return meilleure
```

5. Les pièges

● Attention !

1. `d[x] = d[x] + 1` sans tester `x in d` : `KeyError` à la première apparition.
2. Confondre la clé et la valeur : dans `for cle in d:`, la valeur est `d[cle]`.
3. Supprimer des clés pendant un parcours du dictionnaire : `RuntimeError`. Construire plutôt un nouveau dictionnaire.
4. Écrire `d =` dans la boucle : le dictionnaire est recréé vide à chaque tour.
5. Utiliser une liste comme clé : `TypeError`. Utiliser un tuple.

6. Exemple rédigé

Écrire une fonction `notes_par_eleve(notes: list) -> dict` qui, à partir d'une liste de couples (`nom`, `note`), renvoie le dictionnaire associant à chaque élève la liste de ses notes.

1. **Exemple** : `[('Léa', 12), ('Tom', 9), ('Léa', 15)]` → `Léa' [12, 15], 'Tom' [9]` `'Léa': [12, 15], 'Tom': [9]`.
2. **Entrée / sortie** : une liste de couples ; un dictionnaire dont les clés sont des chaînes et les valeurs des listes.
3. **Schéma** : regrouper ; la clé du groupe est le nom.
4. **Code et trace** :

```
def notes_par_eleve(notes: list) -> dict:
    '''notes : liste de couples (nom, note).
    Renvoie le dictionnaire nom -> liste
    des notes de cet élève.'''
    d = {}
    for (nom, note) in notes:
        if nom not in d:
            d[nom] = []
        d[nom].append(note)
    return d
```

couple lu	d après le tour
-	{}
('Léa', 12)	Léa' [12] 'Léa': [12]
('Tom', 9)	Léa' [12], 'Tom' [9] 'Léa': [12], 'Tom': [9]
('Léa', 15)	Léa' [12, 15], 'Tom' [9] 'Léa': [12, 15], 'Tom': [9]