Les dictionnaires en Python

Résumé

Dans une liste, on accède à un élément par son *numéro* (son indice). Mais parfois, on veut retrouver une information par son *nom* : le numéro de téléphone de « Alice », la note de l'élève « Dupont », le prix de l'article « stylo ». Le **dictionnaire** est la structure qui permet cela : il associe des **clés** (les noms) à des **valeurs** (les informations). C'est l'équivalent informatique d'un annuaire ou d'un répertoire : on cherche par nom, pas par position.

Prérequis : Les listes.

① Mémo

Définition 1 (Structure clé-valeur)

Un dictionnaire associe des **clés** (uniques, immuables) à des **valeurs** (quelconques).

```
d = {"nom": "Euler", "prenom": "Leonhard", "annee": 1707}
d = {}   # dictionnaire vide
d = dict()
```

Les clés peuvent être des `str`, `int`, `float`, `tuple`, mais pas des `list` ni des `dict` (types non hachables).

Opérations courantes

`d[cle]` Accès à la valeur associée à `cle` (erreur `KeyError` si absente).
`d.get(cle, defaut)` Renvoie la valeur ou `defaut` si la clé est absente.
`d[cle] = val` Ajoute ou modifie l'association.
`del d[cle]` Supprime l'entrée.
`cle in d` Teste l'existence d'une clé (`True`/`False`).
`len(d)` Nombre d'entrées.

Parcours d'un dictionnaire

```
for cle in d:                  # parcours des clés
for cle in d.keys():           # idem, explicite
for valeur in d.values():      # parcours des valeurs
for cle, valeur in d.items():  # parcours clé + valeur
```

⚠ Pièges fréquents

- Confondre `d[cle]` (accès) et `d[cle] = val` (affectation).
- Accéder à une clé inexistante sans `get` provoque un `KeyError`.
- Un dictionnaire n'est pas ordonné par construction (l'ordre d'insertion est conservé depuis Python 3.7, mais ce n'est pas une propriété à exploiter algorithmiquement).
- Utiliser une liste comme clé provoque un `TypeError`.

⚠ Erreurs classiques

Code erroné	Code correct	Explication
`d["age"]` quand la clé n'existe pas	`d.get("age", 0)`	L'accès direct lève un `KeyError` ; `.get()` renvoie une valeur par défaut.
`d[[1, 2]] = "v"`	`d[(1, 2)] = "v"`	Une liste est mutable et ne peut pas servir de clé : utiliser un tuple (immutable).
`d2 = d` (alias)	`d2 = dict(d)` ou `d2 = d.copy()`	Comme pour les listes, l'affectation crée un alias : modifier `d2` modifie `d`.
`d["nom"] = "A"` puis `d["nom"] = "B"`	Vérifier avec `if "nom" not in d:`	Ajouter une clé déjà existante écrase silencieusement l'ancienne valeur.

② Exemples

Programme 1 · Compteur d'occurrences

```python
def compter_lettres(texte):
    freq = {}
    for lettre in texte:
        if lettre in freq:
            freq[lettre] += 1
        else:
            freq[lettre] = 1
    return freq

# Version avec get
def compter_lettres_v2(texte):
    freq = {}
    for lettre in texte:
        freq[lettre] = freq.get(lettre, 0) + 1
    return freq
```

Programme 2 · Répertoire téléphonique

```python
repertoire = {}
repertoire["Alice"] = "06 12 34 56 78"
repertoire["Bob"]   = "07 98 76 54 32"

print(repertoire.get("Charlie", "Inconnu"))  # "Inconnu"
```

Programme 3 · Parcours avec items()

```python
notes = {"Alice": 15, "Bob": 12, "Charlie": 18}
for eleve, note in notes.items():
    if note >= 16:
        print(f"{eleve} a obtenu la mention Très bien")
```

Programme 4 · Dictionnaire en compréhension

```python
carres = {x: x**2 for x in range(1, 6)}
# {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Programme 5 · Table de correspondance (p-uplet comme clé)

```
distances = {
    ("Paris", "Lyon"): 465,
    ("Paris", "Marseille"): 775,
    ("Lyon", "Marseille"): 315
}
print(distances[("Paris", "Lyon")])  # 465
```

③ Exercices

Exercice 1 *Fréquence des mots* Écrire une fonction `frequence_mots(texte)` qui prend un texte (chaîne de caractères) et renvoie un dictionnaire donnant le nombre d'occurrences de chaque mot. La fonction doit ignorer la casse (tout est mis en minuscules) et la ponctuation (remplacer `.`, `,`, `!` et `?` par des espaces avant de découper).

Exemple. `frequence_mots("Le chat dort. Le chien aboie !")` renvoie `{"le": 2, "chat": 1, "dort": 1, "chien": 1, "aboie": 1}`.

Exercice 2 *Inverser un dictionnaire* Écrire une fonction `inverser(d)` qui échange clés et valeurs d'un dictionnaire. On suppose que les valeurs sont toutes distinctes et immuables.

Exemple. `inverser({"a": 1, "b": 2})` renvoie `{1: "a", 2: "b"}`.

Exercice 3 *Carnet de notes* On représente un carnet de notes par un dictionnaire dont les clés sont les noms des élèves et les valeurs sont des listes de notes.

```
carnet = {"Alice": [15, 12, 18], "Bob": [8, 11, 14], "Charlie": [16, 17]}
```

1. Écrire une fonction `moyenne_eleve(carnet, nom)` qui renvoie la moyenne d'un élève.
2. Écrire une fonction `meilleur_eleve(carnet)` qui renvoie le nom de l'élève ayant la meilleure moyenne.
3. Écrire une fonction `au_dessus(carnet, seuil)` qui renvoie la liste des noms des élèves dont la moyenne est supérieure ou égale au seuil.

Exercice 4 *Fusion de dictionnaires* Écrire une fonction `fusionner(d1, d2)` qui fusionne deux dictionnaires de compteurs en additionnant les valeurs pour les clés communes.

Exemple. `fusionner({"a": 3, "b": 1}, {"b": 2, "c": 4})` renvoie `{"a": 3, "b": 3, "c": 4}`.

Solutions des exercices

Corrigé de l'exercice 1.

```
def frequence_mots(texte):
    texte = texte.lower()
    for signe in ".,!?":
        texte = texte.replace(signe, " ")
    mots = texte.split()
    freq = {}
    for mot in mots:
        freq[mot] = freq.get(mot, 0) + 1
    return freq
```

Principe : `lower()` met tout le texte en minuscules, puis chaque signe de ponctuation est remplacé par un espace avec `replace()`. `split()` découpe ensuite le texte en une liste de mots (séparateur par défaut : les espaces). On utilise le schéma du compteur avec `get` pour éviter le `KeyError`.

Corrigé de l'exercice 2.

```
def inverser(d):
    return {v: k for k, v in d.items()}
```

Attention : si deux clés ont la même valeur, l'une sera écrasée. L'hypothèse de valeurs distinctes est essentielle.

Corrigé de l'exercice 3.

```
def moyenne_eleve(carnet, nom):
    notes = carnet[nom]
    return sum(notes) / len(notes)

def meilleur_eleve(carnet):
    meilleur = None
    meilleure_moy = -1
    for nom in carnet:
        moy = moyenne_eleve(carnet, nom)
        if moy > meilleure_moy:
            meilleure_moy = moy
            meilleur = nom
    return meilleur

def au_dessus(carnet, seuil):
    return [nom for nom in carnet
            if moyenne_eleve(carnet, nom) >= seuil]
```

Vérification : moyennes : Alice = 15, Bob = 11, Charlie = 16,5. Le meilleur est Charlie. Au-dessus de 14 : `["Alice", "Charlie"]`.

Corrigé de l'exercice 4.

```python
def fusionner(d1, d2):
    resultat = dict(d1)   # copie de d1
    for cle, val in d2.items():
        resultat[cle] = resultat.get(cle, 0) + val
    return resultat
```

Principe : on copie `d1`, puis pour chaque entrée de `d2`, on ajoute la valeur à celle existante (ou 0 si la clé est absente).