Dictionnaires

Bases de Python

1 Pourquoi des dictionnaires ?

Dans une liste, on accède aux éléments par leur **indice** (un numéro). Mais ce n'est pas toujours la façon la plus naturelle de retrouver une information. Quand on cherche un numéro dans un annuaire, on ne dit pas « je veux l'élément n°37 » mais « je veux le numéro d'Alice ».

Avec une liste, pour afficher le numéro de Bob, il faut savoir que ce numéro se trouve à l'indice 1 :

```
numeros = ["06 12 34 56 78", "07 98 76 54 32"]
print(numeros[1])
```

```
07 98 76 54 32
```

Avec un dictionnaire, à chaque prénom correspond un numéro, et le prénom suffit pour retrouver ce numéro :

```
annuaire = {"Alice": "06 12 34 56 78", "Bob": "07 98 76 54 32"}
print(annuaire["Bob"])
```

```
07 98 76 54 32
```

Dans les deux cas, on obtient le même numéro ; mais avec le dictionnaire, inutile de connaître sa position.

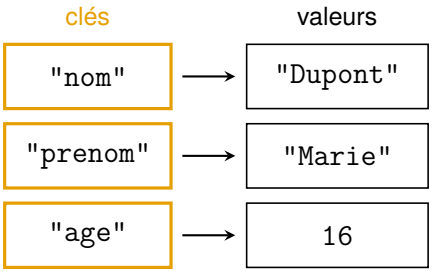
Un **dictionnaire** fonctionne sur ce principe : au lieu d'un indice numérique, on utilise une **clé** (par exemple un nom, un mot, un code) pour accéder à une **valeur** (un numéro de téléphone, une définition, une note).

2 Qu'est-ce qu'un dictionnaire ?

Un dictionnaire s'écrit entre **accolades** `{}`. Il est formé d'**entrées**, séparées par des virgules ; chaque entrée associe une **clé** et sa **valeur**, séparées par deux-points : `clé: valeur`.

```
eleve = {"nom": "Dupont", "prenom": "Marie", "age": 16}
print(eleve)
```

```
{'nom': 'Dupont', 'prenom': 'Marie', 'age': 16}
```



Ici, la clé `"nom"` est associée à la valeur `"Dupont"`, la clé `"prenom"` à la valeur `"Marie"`, et la clé `"age"` à la valeur `16`. Les clés sont des chaînes de caractères ; les valeurs sont des chaînes de caractères et un entier.

Pour la lisibilité, on peut écrire une entrée par ligne :

```
annuaire = {"Alice": "06 12 34 56 78",
            "Bob": "07 98 76 54 32",
            "Charlie": "06 11 22 33 44"}
print(annuaire)
```

```
{'Alice': '06 12 34 56 78', 'Bob': '07 98 76 54 32', 'Charlie': '06 11 22 33 44'}
```

Un **dictionnaire vide** s'écrit avec deux accolades, sans rien entre elles :

```
d = {}
# ou
d = dict()
```

③ Lire une valeur avec sa clé

On accède à une valeur en écrivant le nom du dictionnaire, puis la **clé** entre crochets :

```
annuaire = {"Alice": "06 12 34 56 78",
            "Bob": "07 98 76 54 32",
            "Charlie": "06 11 22 33 44"}

print(annuaire["Alice"])    # 06 12 34 56 78
print(annuaire["Charlie"])  # 06 11 22 33 44
```

De même, avec la fiche de l'élève :

```
eleve = {"nom": "Dupont", "prenom": "Marie", "age": 16}
print(eleve["prenom"])
print(eleve["age"])
```

```
Marie
16
```

Question.
Les entrées de `eleve` sont écrites dans un certain ordre. Que se passe-t-il si l'on écrit `eleve[0]` pour lire la première ?

Réponse.
Python s'arrête avec une erreur de clé, `KeyError` : `0` n'est pas une clé de ce dictionnaire.

```
print(eleve[0])
```

```
KeyError: 0
```

Un dictionnaire n'a pas d'indices : on lit une valeur uniquement grâce à la clé qui lui correspond.

4 Une clé absente

Attention. si la clé n'existe pas, Python lève une erreur `KeyError` :

```
print(annuaire["Diana"])  # KeyError: 'Diana'
```

Pour éviter cette erreur, on peut utiliser la méthode `get()`, qui renvoie `None` (c'est-à-dire aucune valeur) ou une valeur par défaut si la clé est absente :

```
print(annuaire.get("Diana"))         # None
print(annuaire.get("Diana", "???"))  # ???
print(annuaire.get("Alice", "???"))  # 06 12 34 56 78
```

La valeur par défaut est souvent `0`, par exemple pour la note d'un élève absent du dictionnaire :

```
notes = {"Lina": 15, "Tom": 12}
print(notes.get("Sami"))
print(notes.get("Sami", 0))
```

```
None
0
```

Question.
Qu'affiche `print(notes.get("Lina", 0))` ?

Réponse.

```
15
```

La clé `"Lina"` existe : `get()` renvoie simplement sa valeur. La valeur par défaut ne sert que pour une clé absente.

5 Ajouter, modifier ou supprimer une entrée

Un dictionnaire se modifie, comme une liste. Pour **ajouter** un couple clé-valeur ou **modifier** une valeur existante, on utilise la même syntaxe :

```
eleve = {"nom": "Dupont", "prenom": "Marie"}

# Ajouter une nouvelle clé
eleve["classe"] = "1NSI"
print(eleve)

# Modifier une valeur existante
eleve["classe"] = "TNSI"
print(eleve)
```

```
{'nom': 'Dupont', 'prenom': 'Marie', 'classe': '1NSI'}
{'nom': 'Dupont', 'prenom': 'Marie', 'classe': 'TNSI'}
```

Si la clé existe déjà, la valeur est **remplacée** (un dictionnaire ne peut pas contenir deux fois la même clé).

On supprime un couple clé-valeur avec le mot-clé `del` :

```
eleve = {"nom": "Dupont", "prenom": "Marie", "classe": "1NSI"}
del eleve["classe"]
print(eleve)  # {'nom': 'Dupont', 'prenom': 'Marie'}
```

Instruction	Effet sur le dictionnaire
`eleve["classe"] = "1NSI"`	ajout d'une entrée : la clé `"classe"` est nouvelle
`eleve["classe"] = "TNSI"`	modification : la clé `"classe"` existe déjà
`del eleve["classe"]`	suppression de l'entrée, clé et valeur

À retenir. Une clé nouvelle provoque un ajout, une clé existante provoque une modification.

6 Longueur et appartenance

La fonction `len()` renvoie le nombre de couples clé-valeur, c'est-à-dire le nombre d'entrées :

```
eleve = {"nom": "Dupont", "prenom": "Marie", "classe": "1NSI"}
print(len(eleve))  # 3
```

L'opérateur `in` teste si une **clé** est présente dans le dictionnaire :

```python
print("nom" in eleve)     # True
print("age" in eleve)     # False
print("Dupont" in eleve)  # False (in teste les CLÉS, pas les valeurs)
```

`"Dupont"` est une valeur, et non une clé : sur un dictionnaire, `in` cherche seulement parmi les clés.

7 Parcourir un dictionnaire par ses clés

Par défaut, une boucle `for` sur un dictionnaire parcourt ses **clés**, dans l'ordre où les entrées ont été ajoutées :

```python
eleve = {"nom": "Dupont", "prenom": "Marie", "classe": "1NSI"}

for cle in eleve:
    print(f"{cle} : {eleve[cle]}")
```

```
nom : Dupont
prenom : Marie
classe : 1NSI
```

À chaque tour, la variable `cle` reçoit une clé, et `eleve[cle]` donne la valeur qui lui est associée :

Tour	Valeur de cle	Valeur associée	Affichage
1	`"nom"`	`"Dupont"`	`nom : Dupont`
2	`"prenom"`	`"Marie"`	`prenom : Marie`
3	`"classe"`	`"1NSI"`	`classe : 1NSI`

On peut aussi utiliser la méthode `keys()` pour être plus explicite ; elle donne les mêmes clés :

```python
for cle in eleve.keys():
    print(cle)
```

```
nom
prenom
classe
```

8 Parcourir les valeurs et les couples

Parcours des valeurs

La méthode `values()` renvoie les valeurs, sans les clés :

```python
for valeur in eleve.values():
    print(valeur)
```

```
Dupont
Marie
1NSI
```

Question.
`"Marie" in eleve` donne `False`, car `"Marie"` n'est pas une clé. Comment tester si `"Marie"` fait partie des valeurs du dictionnaire ?

Réponse.
Avec la méthode `values()`, le mot-clé `in` cherche parmi les valeurs :

```
print("Marie" in eleve.values())
```

```
True
```

Parcours des couples (clé, valeur)

La méthode `items()` renvoie les paires sous forme de tuples. Le déballage sépare chaque couple en deux variables, la clé et la valeur :

```
for cle, valeur in eleve.items():
    print(f"{cle} → {valeur}")
```

```
nom → Dupont
prenom → Marie
classe → 1NSI
```

Exercice 1 *Opérations de base* **(3 points)** Soit le dictionnaire suivant :

```
pays = {"France": "Paris", "Allemagne": "Berlin", "Espagne": "Madrid"}
```

1. Comment accéder à la capitale de l'Allemagne ?
2. Comment ajouter l'Italie (capitale : Rome) ?
3. Que renvoie `"Paris" in pays` ? Pourquoi ?
4. Écrire une boucle qui affiche chaque pays et sa capitale au format `France -> Paris`.

9 Une liste de dictionnaires

En pratique, on combine souvent les dictionnaires et les listes. Par exemple, une classe d'élèves peut être représentée comme une **liste de dictionnaires** ; à chaque tour, la variable `eleve` contient un dictionnaire, et on lit ses valeurs avec les clés `"prenom"`, `"nom"` et `"moyenne"` :

```
classe = [
    {"nom": "Dupont", "prenom": "Marie", "moyenne": 14.5},
    {"nom": "Martin", "prenom": "Alice", "moyenne": 16.0},
    {"nom": "Durand", "prenom": "Bob", "moyenne": 12.5}
]

# Afficher les noms et moyennes
for eleve in classe:
    print(f"{eleve['prenom']} {eleve['nom']} : {eleve['moyenne']}")
```

```
Marie Dupont : 14.5
Alice Martin : 16.0
Bob Durand : 12.5
```

```
# Calculer la moyenne de la classe
total = sum([e["moyenne"] for e in classe])
print(f"Moyenne de la classe : {total / len(classe)}")
```

```
Moyenne de la classe : 14.333333333333334
```

Question.
Que vaut `classe[1]["prenom"]` ?

Réponse.

```
print(classe[1]["prenom"])
```

```
Alice
```

`classe[1]` est l'élément d'indice 1 de la liste, c'est-à-dire le deuxième dictionnaire ; la clé `"prenom"` y donne la valeur `"Alice"`. On écrit d'abord l'indice dans la liste, puis la clé dans le dictionnaire.

Avec une liste de tuples, `("Dupont", "Marie", 14.5)`, il faudrait se souvenir que la moyenne se trouve à l'indice 2 du tuple ; avec un dictionnaire, la clé `"moyenne"` dit ce que contient la valeur : le programme est plus lisible.

Le programme de NSI appelle **p-uplet nommé** un tel ensemble de valeurs repérées par des noms (les clés) plutôt que par des positions : c'est ce qu'un dictionnaire Python représente, par opposition au tuple, ou p-uplet, dont les éléments sont repérés par leur indice.

10 Des clés immuables

Les clés d'un dictionnaire doivent être de type **immuable** : entiers, chaînes de caractères ou tuples. Les listes et les dictionnaires, qui se modifient, ne peuvent pas servir de clés.

```
# Clés de types variés
d = {42: "un entier", "nom": "une chaîne", (0, 1): "un tuple"}
print(d[42])       # un entier
print(d[(0, 1)])   # un tuple
```

```
# Une liste comme clé → erreur
d = {[1, 2]: "interdit"}  # TypeError: unhashable type: 'list'
```

Exemple concret : pour stocker les cases d'un plateau de jeu, on utilise les coordonnées `(ligne, colonne)` comme clés :

```
plateau = {}
plateau[(0, 0)] = "tour"
plateau[(0, 1)] = "cavalier"
plateau[(3, 4)] = "pion"
print(plateau[(0, 0)])   # tour
```

Ce ne serait pas possible avec des listes `[0, 0]`, car une liste ne peut pas être une clé de dictionnaire.

11 Des valeurs de tout type

Les valeurs n'ont aucune contrainte : elles peuvent être des entiers, des flottants, des chaînes de caractères, des listes, d'autres dictionnaires, etc.

```
fiche = {
    "nom": "Dupont",
    "notes": [15, 12, 18, 14],
    "adresse": {"rue": "10 rue des Lilas", "ville": "Lyon"}
}
print(fiche["notes"][2])            # 18
print(fiche["adresse"]["ville"])    # Lyon
```

On enchaîne deux paires de crochets : `fiche["notes"]` donne la liste des notes, puis `[2]` donne sa note d'indice 2 ; de même, `fiche["adresse"]` donne un dictionnaire, dans lequel la clé `"ville"` donne `"Lyon"`.

12 Compter les apparitions

Une utilisation classique des dictionnaires consiste à compter combien de fois chaque élément apparaît, par exemple chaque lettre d'un mot. On part d'un dictionnaire vide ; pour chaque lettre, si elle est déjà une clé, on ajoute 1 à sa valeur, sinon on crée son entrée, avec la valeur 1.

```
mot = "radar"
compte = {}
for lettre in mot:
    if lettre in compte:
        compte[lettre] += 1
    else:
        compte[lettre] = 1
print(compte)
```

Question.
Qu'affiche ce programme ?

Réponse.

```
{'r': 2, 'a': 2, 'd': 1}
```

Tour	Valeur de lettre	Déjà une clé ?	Contenu de compte après le tour
1	`'r'`	non	`{'r': 1}`
2	`'a'`	non	`{'r': 1, 'a': 1}`
3	`'d'`	non	`{'r': 1, 'a': 1, 'd': 1}`
4	`'a'`	oui	`{'r': 1, 'a': 2, 'd': 1}`
5	`'r'`	oui	`{'r': 2, 'a': 2, 'd': 1}`

La méthode `get()` permet d'écrire ce test en une seule ligne : `frequences.get(lettre, 0)` renvoie 0 pour une lettre absente, et le calcul donne 1, comme avec le `else`.

```
# Compter les lettres d'un mot
mot = "abracadabra"
frequences = {}
for lettre in mot:
    frequences[lettre] = frequences.get(lettre, 0) + 1
print(frequences)  # {'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}
```

13 Construire un dictionnaire en compréhension

Comme les listes, les dictionnaires peuvent être créés par **compréhension**. Entre accolades, on écrit la clé, deux-points, la valeur, puis la boucle qui fournit les valeurs de la variable :

```
carres = {n: n**2 for n in range(6)}
```

Question.
Que contient le dictionnaire `carres` ?

Réponse.

```
print(carres)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Chaque entier de 0 à 5 devient une clé, associée à son carré.

Une compréhension transforme aussi une liste de tuples en dictionnaire, grâce au déballage :

```
eleves = [("Lina", 15), ("Tom", 12), ("Sami", 17)]
notes = {prenom: note for prenom, note in eleves}
print(notes)
print(notes["Tom"])
```

```
{'Lina': 15, 'Tom': 12, 'Sami': 17}
12
```

14 À RETENIR

1. Un dictionnaire associe des clés à des valeurs, entre accolades : `{"nom": "Dupont"}`.
2. On lit une valeur par sa clé, `eleve["nom"]` ; un dictionnaire n'a pas d'indices.
3. `notes.get("Sami", 0)` évite l'erreur `KeyError` : `get()` renvoie `None`, ou la valeur par défaut donnée.
4. `eleve[cle] = valeur` ajoute une entrée si la clé est nouvelle et modifie sa valeur si la clé existe ; `del eleve[cle]` supprime une entrée.
5. `"nom" in eleve` cherche parmi les clés ; `"Marie" in eleve.values()`, parmi les valeurs.
6. On parcourt les clés avec `for cle in eleve`, les valeurs avec `values()`, les couples avec `items()`.
7. Une clé est immuable (entier, chaîne de caractères, tuple) ; une valeur peut être de n'importe quel type.
8. `compte[lettre] = compte.get(lettre, 0) + 1` compte les apparitions de chaque élément.

15 Résumé

Opération	Syntaxe	Exemple
Création	`{clé: valeur, ...}`	`{"nom": "Alice"}`
Accès	`d[clé]`	`d["nom"]` → `"Alice"`
Accès sûr	`d.get(clé, défaut)`	`d.get("age", 0)`
Ajout / modification	`d[clé] = valeur`	`d["age"] = 17`
Suppression	`del d[clé]`	`del d["age"]`
Longueur	`len(d)`	`len(d)` → `1`
Appartenance	`clé in d`	`"nom" in d` → `True`
Parcours clés	`for k in d:`	
Parcours valeurs	`for v in d.values():`	
Parcours couples	`for k, v in d.items():`	

Exercices

Exercice 2 *Carnet de notes* **(4 points)** On représente les notes d'un élève par un dictionnaire :

```
notes = {"maths": [15, 12, 18], "NSI": [16, 14], "anglais": [13, 11, 17, 15]}
```

1. Écrire un programme qui calcule et affiche la moyenne de chaque matière.

2. Écrire un programme qui affiche la matière ayant la meilleure moyenne.

Exercice 3 *Comptage de mots* **(4 points)** Écrire un programme qui demande une phrase à l'utilisateur et affiche le nombre d'occurrences de chaque mot.

Exemple : pour « le chat et le chien », le programme affiche `{"le": 2, "chat": 1, "et": 1, "chien": 1}`.

Indication : utiliser `.split()` pour découper la phrase en mots, puis le schéma avec `.get()`.

Exercice 4 *Dictionnaire inverse* **(4 points)** Soit un dictionnaire associant des pays à leurs capitales :

```
capitales = {"France": "Paris", "Allemagne": "Berlin",
             "Espagne": "Madrid", "Italie": "Rome"}
```

Écrire un programme qui crée le dictionnaire **inverse** : les capitales deviennent les clés, et les pays deviennent les valeurs. Afficher le résultat.

Exercice 5 *Synthèse — répertoire téléphonique* **(5 points)** Écrire un programme qui simule un répertoire téléphonique. Le programme affiche un menu en boucle :

- 1 : ajouter un contact (nom + numéro) ;
- 2 : rechercher un contact (par nom) ;
- 3 : afficher tous les contacts ;
- 4 : quitter.

Le programme continue tant que l'utilisateur ne choisit pas 4.