Chapitre 4
Dictionnaires

Ce chapitre est le quatrième chapitre de l'année de Première NSI. Après les tuples et les listes, on découvre un nouveau type construit de Python : le **dictionnaire**. Alors qu'une liste associe une valeur à un **indice** (un entier), un dictionnaire associe une valeur à une **clé**, qui peut être un texte, un entier, etc. Cette structure est omniprésente en informatique : un annuaire téléphonique, un carnet de notes, un inventaire de stock sont naturellement des dictionnaires.

I. Définition et création

Les types construits vus jusqu'ici (tuples, listes) rangent des valeurs dans un certain **ordre** : on accède à un élément par son indice. Un dictionnaire fonctionne différemment : il ne range pas les valeurs dans l'ordre, il les **associe** à des clés. Pour retrouver une valeur, on donne sa clé, pas sa position.

Définition 4.1.
Un **dictionnaire** en Python est un objet contenant des paires **clé-valeur**.
- Les **clés** sont des éléments non modifiables et doivent être **uniques**. Elles ont pour types des objets immuables comme les entiers, les chaînes de caractères, les tuples.
- Les **valeurs** associées aux clés sont, elles, modifiables et peuvent être de n'importe quel type.

R Un dictionnaire est **non ordonné** : il n'y a pas de premier élément, de deuxième élément, etc. Seule compte l'association clé-valeur.

R Un dictionnaire est **modifiable** : on peut modifier ses valeurs, ajouter ou supprimer des éléments après sa création.

Exemple 4.1.
On crée un dictionnaire vide soit en utilisant la commande `dict()`, soit en utilisant des accolades {}. ■

```
dico_1 = dict() # On cree un dico vide
dico_2 = {} # On cree un autre dico vide
```

Exemple 4.2.
Pour créer un dictionnaire directement avec des paires clé-valeur, on utilise exclusivement les accolades. Une valeur est associée à une clé selon la syntaxe `clé : valeur` et les différentes paires sont séparées par des virgules. ■

```
pokemon_1 = {"nom": "bulbizarre", "type": "plante"}
pokemon_2 = {"nom": "herbizarre", "type": "plante"}
pokedex = {1: pokemon_1, 2: pokemon_2} # dictionnaire de dictionnaires !
```

On remarquera ici que les clés sont bien uniques au sein de leurs dictionnaires, même si des dictionnaires différents les partagent : la clé `"nom"` existe dans `pokemon_1` et dans `pokemon_2`, ce qui ne pose aucun problème.

Exemple 4.3.
On accède à la valeur associée à une clé avec `dictionnaire[clé]`. Contrairement aux listes, ce n'est pas un indice qui est utilisé, mais la clé elle-même. ■

```
pokemon_1 = {"nom": "bulbizarre", "type": "plante"}
print(pokemon_1["nom"])
print(pokemon_1["type"])
```

Ce programme affiche `bulbizarre` puis `plante`.

Exemple 4.4.
Les valeurs d'un dictionnaire peuvent être de n'importe quel type : entiers, chaînes, listes, voire d'autres dictionnaires. Un même dictionnaire peut donc regrouper des informations très variées sur un même objet. ■

```
pokemon_8 = {"nom": "pikachu", "numero": 25, "stats": [55, 40, 50]}
print(pokemon_8["stats"][0])
```

Ce programme affiche `55` : on lit d'abord la liste associée à la clé `"stats"`, puis son premier élément.

Exemple 4.5.
Une clé peut être un entier, une chaîne de caractères ou un tuple. Les clés doivent être immuables : on peut donc utiliser un tuple pour représenter des coordonnées. ■

```
positions = {(0, 0): "origine", (1, 0): "droite", (0, 1): "haut"}
print(positions[(0, 0)])
```

Ce programme affiche `origine`.

■ **Exercice 4.1.**
L'objectif de cet exercice et des suivants est de compléter notre pokédex.
1. Créer un dictionnaire vide `pokedex_3`.
2. Créer un dictionnaire `pokemon_4` pour Salamèche avec la clé `"nom"` (on s'occupera du type plus tard).

■

II. Ajout d'un élément

On peut ajouter un couple clé-valeur à un dictionnaire à condition que la clé soit bien unique. Pour cela, il suffit de faire `dictionnaire[clé] = valeur`.

■ **Exemple 4.6.**
On ajoute au dictionnaire `pokemon_3` le couple clé-valeur `"nom"-"florizarre"` en faisant : ■

```
pokemon_3 = {"type": "plante"}
pokemon_3["nom"] = "florizarre"
print(pokemon_3)
```

Ce programme affiche `{'type': 'plante', 'nom': 'florizarre'}` : la clé `"nom"` n'existait pas encore, elle a été ajoutée avec sa valeur.

■ **Exercice 4.2.**

1. Ajouter au dictionnaire `pokemon_3` le type de Florizarre (c'est un type `"plante"`).
2. Ajouter le dictionnaire `pokemon_3` au pokédex `pokedex_3` avec la clé 3.

■

III. Modification d'un élément

On peut modifier la valeur associée à une clé, mais pas la clé elle-même. Pour cela, on utilise à nouveau la commande `dictionnaire[clé] = valeur` : si la clé existe déjà, sa valeur est simplement écrasée.

■ **Exemple 4.7.**
On a créé un dictionnaire pour Carapuce, mais celui-ci n'a pas le bon type ; on le modifie donc pour corriger cela. ■

```
pokemon_7 = {"nom": "carapuce", "type": "foudre"}
pokemon_7["type"] = "eau"
print(pokemon_7)
```

Ce programme affiche {'nom': 'carapuce', 'type': 'eau'} : la clé "type" existait déjà, sa valeur a été remplacée.

Exercice 4.3.
Modifier le dictionnaire suivant afin de corriger l'erreur sur le type de Reptincel : Reptincel est un pokémon de type "feu", pas de type "psy". ■

```
pokemon_5 = {"nom": "Reptincel", "type": "psy", "exp": 9999}
```

IV. Suppression d'un élément

On peut supprimer un couple clé-valeur d'un dictionnaire en indiquant la clé du couple à supprimer grâce à la commande `dico.pop(clé)`. La méthode `pop` supprime le couple et renvoie la valeur qui lui était associée.

Exemple 4.8.
On a récupéré un dictionnaire pour Dracaufeu, mais celui-ci comporte l'élément "plat favori" dont on ne veut pas ; on le supprime donc. ■

```
pokemon_6 = {"nom": "dracaufeu", "type": "feu", "plat favori": "risotto"}
pokemon_6.pop("plat favori")
print(pokemon_6)
```

Ce programme affiche {'nom': 'dracaufeu', 'type': 'feu'} : le couple clé-valeur "plat favori" a été retiré.

Exemple 4.9.
La méthode `pop` renvoie la valeur du couple supprimé. On peut réutiliser cette valeur : c'est ainsi que l'on « renomme » une clé, en réaffectant la valeur à une nouvelle clé. ■

```
d = {"a": 1, "b": 2}
d["z"] = d.pop("a")
print(d)
```

Ce programme affiche {'b': 2, 'z': 1} : la clé "a" a été supprimée, et sa valeur 1 a été réaffectée à la nouvelle clé "z".

Exercice 4.4.
Supprimer l'élément "exp" du dictionnaire de Reptincel `pokemon_5` (défini à la section précédente). ■

V. Fusion de deux dictionnaires

On peut fusionner deux dictionnaires grâce à la commande `dico_1.update(dico_2)` : les couples clé-valeur de `dico_2` sont ajoutés à `dico_1` (et remplacent ceux de même clé, s'il y en a).

Exemple 4.10.
On crée le dictionnaire `type_plante` et on le fusionne avec `pokemon_3`. ■

```
type_plante = {"type": "plante"}
pokemon_3.update(type_plante)
print(pokemon_3)
```

Exemple 4.11.
Si un couple clé-valeur de `dico_2` possède une clé déjà présente dans `dico_1`, la valeur de `dico_1` est remplacée par celle de `dico_2` : c'est une modification, pas un doublon. ■

```
d1 = {"a": 1, "b": 2}
d2 = {"b": 3, "c": 4}
d1.update(d2)
print(d1)
```

Ce programme affiche `{'a': 1, 'b': 3, 'c': 4}` : la clé `"b"` existait déjà, sa valeur 2 a été remplacée par 3.

Exercice 4.5.

1. Créer un dictionnaire `type_feu` sur le modèle de `type_plante` (avec la valeur `"feu"`).
2. Le fusionner avec le dictionnaire de Salamèche `pokemon_4` créé dans la première section.

■

VI. Parcours d'un dictionnaire

On peut parcourir un dictionnaire selon ses clés, ses valeurs ou ses couples clé-valeur grâce aux commandes suivantes :

Parcours selon	Clés	Valeurs	Couples clé-valeur
Commande pour `dico`	`dico.keys()`	`dico.values()`	`dico.items()`

■ **Exemple 4.12.**
L'algorithme suivant permet de parcourir le dictionnaire `pokemon_1` selon ses clés. Affichez-les ! ■

```
for cle in pokemon_1.keys():
    print(cle)
```

■ **Exemple 4.13.**
L'algorithme suivant permet de parcourir le dictionnaire `pokemon_1` selon ses valeurs. Affichez-les ! ■

```
for valeur in pokemon_1.values():
    print(valeur)
```

■ **Exemple 4.14.**
L'algorithme suivant permet de parcourir le dictionnaire `pokemon_1` selon les couples clé-valeur. Affichez-les ! ■

```
for cle, valeur in pokemon_1.items():
    print(cle, valeur)
```

■ **Exemple 4.15.**
Itérer directement sur un dictionnaire (`for cle in dico:`) parcourt ses clés, exactement comme `dico.keys()`. C'est l'écriture la plus courante. ■

```
for cle in pokemon_1:
    print(cle)
```

> **R** Un dictionnaire étant non ordonné, l'ordre d'affichage des clés lors d'un parcours n'est pas garanti : il ne faut jamais écrire de programme qui suppose un « premier » ou un « dernier » élément d'un dictionnaire.

La présence d'une clé dans un dictionnaire se teste avec l'opérateur `in`, et le nombre de couples clé-valeur s'obtient avec `len`. Accéder à une clé absente provoque une erreur `KeyError` : il vaut mieux tester la clé avant d'y accéder, ou utiliser la méthode `get`.

■ **Exemple 4.16.**
L'opérateur `in` teste la présence d'une **clé** (pas d'une valeur) dans un dictionnaire. La fonction `len` compte le nombre de couples clé-valeur. ■

```
pokemon_1 = {"nom": "bulbizarre", "type": "plante"}
print("nom" in pokemon_1)
```

```
print("attaque" in pokemon_1)
print(len(pokemon_1))
```

Ce programme affiche `True`, `False` puis 2.

Exemple 4.17.
Accéder à une clé qui n'existe pas lève une erreur `KeyError`. La méthode `dico.get(clé, valeur_par_défaut)` renvoie la valeur associée à la clé si elle existe, et la valeur par défaut sinon, sans lever d'erreur. ■

```
pokemon_1 = {"nom": "bulbizarre", "type": "plante"}
print(pokemon_1["attaque"])
```

Ce programme lève une erreur `KeyError: 'attaque'`.

```
pokemon_1 = {"nom": "bulbizarre", "type": "plante"}
print(pokemon_1.get("attaque", "inconnue"))
```

Ce programme affiche `inconnue` : la clé `"attaque"` est absente, la valeur par défaut est renvoyée.

Exemple 4.18.
Les valeurs d'un dictionnaire peuvent être des listes ou des dictionnaires : on parle alors de dictionnaire imbriqué. On y accède en enchaînant les clés (et éventuellement les indices). ■

```
pokedex = {1: {"nom": "bulbizarre", "type": "plante"},
           2: {"nom": "herbizarre", "type": "plante"}}
print(pokedex[1]["nom"])
```

Ce programme affiche `bulbizarre` : `pokedex[1]` renvoie le dictionnaire du pokémon numéro 1, dont on lit ensuite la clé `"nom"`.

Exemple 4.19.
On peut construire un dictionnaire au fil d'une boucle : c'est le motif central de l'accumulation par clé, que l'on retrouvera dans les exercices de comptage. ■

```
phrase = "ananas"
occurrences = {}
for lettre in phrase:
    occurrences[lettre] = occurrences.get(lettre, 0) + 1
print(occurrences)
```

Ce programme affiche `{'a': 3, 'n': 2, 's': 1}` : chaque lettre de la chaîne a été comptée dans un dictionnaire.

■ **Exercice 4.6.**

1. Que va renvoyer le dictionnaire `pokedex` selon qu'il est parcouru par ses :
   — clés ?
   — valeurs ?
   — couples ?
2. Programmer en Python le parcours du pokédex selon les couples clé-valeur, de façon à afficher, pour chaque pokémon, son numéro et son nom.

■

VII. Exercices

Les exercices suivants reprennent et complètent les notions du chapitre. Pensez à bien lire les énoncés, à tester vos programmes, et à interpréter les messages d'erreur éventuels.

Exercices d'automatismes

Les exercices d'automatismes exercent les réflexes fondamentaux : calcul mental d'expressions Python, lecture de code court, prédiction d'affichage, complétion. Chaque question doit pouvoir être traitée en moins de trente secondes.

■ Exercice 4.7.

Types et valeurs : répondre aux questions suivantes sans exécuter les programmes.

1. Quel est le type de la valeur renvoyée par l'expression suivante ?

```
10 / 4
```

2. Quel est le type de la valeur renvoyée par l'expression suivante ?

```
10 // 4
```

3. Quel est le type de la variable `a` après l'affectation suivante ?

```
a = "42"
```

4. Quel est le type de la variable `a` après l'affectation suivante ?

```
a = [1, 2]
```

5. Quel est le type de la variable `a` après l'affectation suivante ?

```
a = {}
```

■

■ Exercice 4.8.

Dictionnaires : clé et valeur. On considère le dictionnaire suivant.

```
eleve = {"prenom": "Sami", "age": 17, "classe": "1G1"}
```

1. Que renvoie `eleve["age"]` ?
2. Quelle instruction permet d'ajouter la clé `"nsi"` avec la valeur `17` à ce dictionnaire ?
3. Quelle instruction permet de modifier la classe de Sami pour qu'elle devienne `"1G2"` ?
4. Que renvoie `len(eleve)` après l'ajout de la question 2 ?
5. Que renvoie `"classe" in eleve` ?

■

■ **Exercice 4.9.**

Accès, `in` et erreurs : répondre aux questions suivantes.

1. Que renvoie l'expression `d["b"]` pour `d = {"a": 1, "b": 2, "c": 3}` ?
2. Que renvoie l'expression `1 in d` pour le même dictionnaire `d` ? Justifier.
3. Que se passe-t-il à l'exécution du programme suivant ?

```
eleve = {"prenom": "Sami", "age": 17}
print(eleve["classe"])
```

4. Que renvoie l'expression `eleve.get("classe", "inconnue")` pour le dictionnaire `eleve` précédent ?
5. Que renvoie l'expression `eleve.get("age", 0)` pour le même dictionnaire ?

■

■ **Exercice 4.10.**

Parcours d'un dictionnaire : déterminer ce qui est affiché par chacun des programmes suivants.

1. Qu'affiche le programme suivant ?

```
d = {"a": 1, "b": 2}
for v in d.values():
    print(v)
```

2. Qu'affiche le programme suivant ?

```
d = {"a": 1, "b": 2}
for cle, valeur in d.items():
    print(cle, valeur)
```

3. Quelle méthode permet d'obtenir uniquement les clés d'un dictionnaire ?
4. Qu'affiche le programme suivant ?

```
d = {"a": 1, "b": 2, "c": 3}
total = 0
for cle in d:
    total = total + d[cle]
print(total)
```

5. Qu'affiche le programme suivant ?

```
d = {"a": 1, "b": 2}
print(list(d.keys()))
```

■

■ **Exercice 4.11.**
Rappels : boucles, fonctions, listes. Répondre aux questions suivantes.

1. Combien de fois le corps de la boucle suivante est-il exécuté ?

```
for i in range(3, 8):
    pass
```

2. Que renvoie `len("bonjour")` ?
3. Qu'affiche le programme suivant ?

```
def double(x):
    return 2 * x

print(double(21))
```

4. Qu'affiche le programme suivant ?

```
L = [1, 2, 3]
L.append(4)
print(len(L))
```

5. Quelle erreur lève l'instruction `int("abc")` ?

■

Exercices d'application

Les exercices d'application reprennent les notions du chapitre, du plus simple au plus difficile. Ils demandent d'écrire de petites fonctions ou de courts programmes Python.

■ **Exercice 4.12.**
On considère les données suivantes, enregistrées dans le dictionnaire `repertoire` :

Nom	Téléphone
Aya	0612345678
Rayan	0687654321
Hanae	0765432198
Piotr	0777666555

```
repertoire = {"Aya": "0612345678",
              "Rayan": "0687654321",
              "Hanae": "0765432198",
              "Piotr": "0777666555"}
```

1. Comment accéder au téléphone de Piotr ?

2. Comment savoir si `"Fanny"` est enregistrée dans le répertoire ?
3. Modifier le numéro de Aya : il se termine par un 9 et non un 8.
4. Ajouter `"Raoul"` dont le numéro est `"0789898989"`.
5. Supprimer `"Piotr"` du répertoire.

■

■ **Exercice 4.13.**

1. Créer un dictionnaire `notes` contenant les notes suivantes : `"maths"` : 15, `"nsi"` : 18, `"anglais"` : 12.
2. Afficher la note de NSI.
3. Ajouter la note `"histoire"` : 14 au dictionnaire.
4. Modifier la note d'anglais : elle passe à 13.
5. Afficher le nombre de matières enregistrées.

■

■ **Exercice 4.14.**

1. Créer à la main, en une seule instruction, le dictionnaire correspondant au tableau suivant :

Nombre	Carré
1	1
2	4
3	9
4	16
5	25
6	36
7	49

2. Créer le même dictionnaire en partant d'un dictionnaire vide et en ajoutant les nombres et leur carré instruction par instruction.
3. Recommencer en utilisant une boucle.

■

■ **Exercice 4.15.**
On veut construire un petit dictionnaire de traduction français-anglais.

1. Créer un dictionnaire `traduction` avec les couples suivants : `"chat"` : `"cat"`, `"chien"` : `"dog"`, `"oiseau"` : `"bird"`.
2. Afficher la traduction anglaise de `"chat"`.
3. Ajouter la traduction `"poisson"` : `"fish"`.
4. Tester si le mot `"souris"` possède une traduction dans le dictionnaire, et l'afficher si c'est le cas.

■

■ **Exercice 4.16.**
Convertir les deux listes suivantes en un dictionnaire en utilisant une seule boucle.

```
keys = ["Ten", "Twenty", "Thirty"]
values = [10, 20, 30]
```

On doit obtenir le dictionnaire {'Ten': 10, 'Twenty': 20, 'Thirty': 30}.
■

■ **Exercice 4.17.**
On reprend le dictionnaire `repertoire` de l'exercice sur l'annuaire. Écrire un programme qui parcourt le répertoire selon les couples clé-valeur et affiche, pour chaque contact, une ligne de la forme :

```
Aya : 0612345678
```

■

■ **Exercice 4.18.**
On considère la liste de fruits suivante, relevée dans un panier :

```
fruits = ["pomme", "poire", "pomme", "kiwi", "pomme", "poire"]
```

Écrire un programme qui construit un dictionnaire `occurrences` associant à chaque fruit le nombre de fois où il apparaît dans la liste. On doit obtenir {'pomme': 3, 'poire': 2, 'kiwi': 1}. ■

■ **Exercice 4.19.**
On considère le dictionnaire imbriqué suivant, dont les données sont enfouies dans plusieurs niveaux.

```
sample_dict = {
   "class": {
      "student": {
         "name": "Mike",
         "marks": {
            "physics": 70,
            "history": 80
         }
      }
   }
}
```

1. Comment accéder à la valeur correspondant à `"history"` ?
2. On pourra commencer par examiner `sample_dict["class"]` : quel est son type ?
3. Comment accéder à la valeur correspondant à `"name"` ?

■

■ **Exercice 4.20.**
Changer la clé `"city"` en `"location"` dans le dictionnaire suivant :

```
sample_dict = {"name": "Kelly",
               "age": 25,
               "salary": 8000,
               "city": "New york"}
```

On doit obtenir le dictionnaire `{'name': 'Kelly', 'age': 25, 'salary': 8000, 'location': 'New york'}`. On pourra lire la valeur associée à `"city"`, l'affecter à la nouvelle clé, puis supprimer l'ancienne. ■

■ **Exercice 4.21.**
On considère le dictionnaire des salaires suivant :

```
sample_dict = {"emp1": {"name": "Jhon", "salary": 7500},
               "emp2": {"name": "Emma", "salary": 8000},
               "emp3": {"name": "Brad", "salary": 6500}}
```

1. Changer le salaire de Brad en 8500.
2. À l'aide d'une boucle, calculer le cumul des salaires des trois employés.
3. À l'aide d'une boucle, créer la liste des noms des employés.

■

■ Exercice 4.22.
On considère le dictionnaire suivant, qui associe à chaque élève sa liste de notes :

```
releve = {"Lina": [15, 12, 18],
          "Sami": [10, 14],
          "Zoe": [17, 13, 16, 11]}
```

Écrire un programme qui affiche, pour chaque élève, son nom et sa moyenne. On pourra utiliser la fonction `sum` pour additionner les notes d'une liste, et `len` pour les compter. ■

■ Exercice 4.23.
Nous allons apprendre à compter les lettres d'une chaîne de caractères. Voici la chaîne en question, utilisée fréquemment pour remplir des zones de texte lorsqu'on développe une interface :

```
"lorem ipsum dolor sit amet consectetur adipiscing elit"
```

Nous allons produire le dictionnaire suivant, qui contient le nombre d'apparitions de chaque caractère dans la chaîne :

```
{' ': 7, 'i': 6, 'e': 5, 't': 5, 'o': 4, 's': 4, 'l': 3, 'r': 3, 'm': 3,
 'c': 3, 'p': 2, 'u': 2, 'd': 2, 'a': 2, 'n': 2, 'g': 1}
```

Voici l'algorithme :

```
creer un dictionnaire vide
Pour chaque lettre de la chaine :
    si la lettre n'est pas presente dans le dictionnaire :
        l'ajouter avec la valeur 1
    sinon :
        augmenter sa valeur de 1
```

1. Implémenter cet algorithme sur la chaîne donnée.
2. Reprendre cet algorithme et l'implémenter dans une fonction `compteur(chaine)` qui renvoie le dictionnaire des occurrences.

■

■ Exercice 4.24.
On reprend le pokédex créé dans le cours, contenant des dictionnaires de pokémons.

Écrire un programme qui parcourt le pokédex selon les couples clé-valeur et affiche, pour chaque pokémon, son numéro, son nom et son type. ■

Exercices de réflexion

Les exercices de réflexion demandent plus d'analyse : déroulement de programmes, repérage d'erreurs, manipulation des subtilités des dictionnaires, écriture de petits algorithmes. Ils sont classés du plus abordable au plus exigeant.

■ **Exercice 4.25.**
On reprend le dictionnaire des carrés construit dans les exercices d'application : `carres = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}`.

1. Écrire un programme qui construit, à l'aide d'une boucle, la liste des clés de ce dictionnaire.
2. Écrire un programme qui construit, à l'aide d'une boucle, la liste des valeurs de ce dictionnaire.
3. Écrire un programme qui construit deux listes simultanément, l'une des clés et l'autre des valeurs, en une seule boucle.

■

■ **Exercice 4.26.**
Le programme suivant est censé afficher la note de NSI d'un élève. Il contient une erreur.

```
releve = {"maths": 15, "nsi": 18, "anglais": 12}
matiere = "physique"
print(releve[matiere])
```

1. Sans l'exécuter, expliquer ce qui va se passer : quelle erreur Python lève-t-il, et pourquoi ?
2. Proposer une correction du programme qui affiche la note de la matière demandée si elle existe, et le message `"matiere inconnue"` sinon, sans jamais lever d'erreur. On pourra utiliser `in` ou la méthode `get`.

■

■ **Exercice 4.27.**
Accumulation par clé. On considère la liste de matières suivante, relevée dans le carnet de notes d'une classe :

```
matieres = ["maths", "nsi", "maths", "anglais", "nsi", "nsi"]
```

Écrire un programme qui construit, **sans utiliser** la méthode `get`, un dictionnaire `comptes` associant à chaque matière le nombre de fois où elle apparaît. On utilisera un test `if matiere in comptes:` pour décider s'il faut incrémenter ou initialiser la valeur. ■

■ **Exercice 4.28.**
On considère le programme suivant.

```
d = {"a": 1, "b": 2, "c": 3}
print(d[0])
```

1. Prédire ce qui est affiché à l'écran, ou l'erreur levée, en justifiant.
2. Expliquer pourquoi il n'existe pas de « premier élément » d'un dictionnaire : que faudrait-il utiliser à la place d'un indice pour accéder à une valeur ?

■

■ **Exercice 4.29.**
On reprend le comptage de lettres de l'exercice d'application. Python propose un outil tout prêt pour ce type de comptage : la fonction `Counter` du module `collections`.

1. Tester le programme suivant sur la chaîne `"lorem ipsum dolor sit amet consectetur adipiscing elit"`.

```
from collections import Counter

chaine = "lorem ipsum"
comptage = Counter(chaine)
print(comptage)
```

2. Expliquer en quoi le résultat ressemble à un dictionnaire, et comment on peut récupérer le nombre d'apparitions du caractère `"m"` avec la notation `comptage["m"]`.

■

■ **Exercice 4.30.**
Inversion d'un dictionnaire. On considère le dictionnaire suivant, qui associe à chaque nom de matière sa note :

```
notes = {"maths": 15, "nsi": 18, "anglais": 12}
```

Écrire un programme qui construit un dictionnaire `inverse` associant à chaque

note la matière correspondante. On doit obtenir {`15: 'maths', 18: 'nsi', 12: 'anglais'`}. Que se passerait-il si deux matières avaient la même note ? ■