Dictionnaires en Python

Activité – Première NSI

① Pourquoi un nouveau type ?

Exercice 1 *Un répertoire téléphonique avec des listes*

On souhaite stocker un répertoire téléphonique : chaque personne est associée à un numéro. Une première idée consiste à utiliser deux listes parallèles :

```
noms     = ["Alice", "Bob",   "Chloé"]
numeros  = ["06...", "07...", "06..."]
```

1. Pour trouver le numéro d'Alice, écrire en pseudo-code (ou en Python) la suite d'opérations à effectuer. *Indication : utiliser* `noms.index(...)`.
2. Que se passe-t-il si la personne recherchée n'est pas dans la liste `noms` ?
3. Si l'on veut ajouter « David » au répertoire, combien de listes doit-on modifier ? Quel risque cela représente-t-il ?

Remarque 1. Un **dictionnaire** stocke des couples **clé–valeur** : à chaque clé est associée une valeur, accessible directement sans passer par un indice. C'est exactement ce qu'il faut pour un répertoire.

② Créer et accéder

Définition 1 **(Dictionnaire)** Un dictionnaire Python est une collection de couples (clé, valeur), notée entre accolades :

```
repertoire = {"Alice": "06...", "Bob": "07...", "Chloé": "06..."}
```

L'accès à une valeur se fait par sa clé entre crochets :

```
print(repertoire["Alice"])    # 06...
```

Si la clé n'existe pas, Python lève une `KeyError`.

Exercice 2 *Premiers accès*

On donne le dictionnaire suivant, qui associe à chaque pays sa capitale :

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

1. Que vaut `capitales["Italie"]` ? Et `capitales["Allemagne"]` ?
2. Que vaut `len(capitales)` ?
3. Que se passe-t-il si l'on écrit `capitales["Belgique"]` ?
4. Comment ajouter (ou modifier) une entrée ? Donner la syntaxe pour ajouter `"Belgique" : "Bruxelles"`.
5. Comment supprimer une entrée ? Donner la syntaxe pour supprimer le `"Portugal"`.

③ Tester l'appartenance d'une clé

Propriété 1 **(Opérateur `in`)** `cle in dico` renvoie `True` si `cle` apparaît parmi les clés du dictionnaire, `False` sinon. C'est la manière sûre d'éviter une `KeyError`.

Exercice 3 *Accès sans erreur*

On reprend le dictionnaire `capitales` de l'exercice précédent.

1. Que renvoient les expressions suivantes ?
   a) `"France" in capitales`
   b) `"Belgique" in capitales`
   c) `"Paris" in capitales`
2. Compléter la fonction `capitale_de(pays)` qui renvoie la capitale du pays si elle est connue, ou la chaîne `"inconnue"` sinon.

```python
def capitale_de(pays):
    if ............:
        return ............
    else:
        return "inconnue"
```

3. Tester avec les pays `"France"`, `"Belgique"` et `"Italie"`.

④ Parcourir un dictionnaire

Propriété 2 **(Trois façons de parcourir)**

— `for cle in dico:` parcourt les **clés** ;

— `for valeur in dico.values():` parcourt les **valeurs** ;

— `for cle, valeur in dico.items():` parcourt les **couples** (clé, valeur).

Exercice 4 *Parcours en pratique*

On considère le dictionnaire `notes` qui associe à chaque élève sa note sur 20 :

```
notes = {"Alice": 14, "Bob": 9, "Chloé": 17, "David": 11, "Emma": 6}
```

1. Compléter la fonction `moyenne(notes)` qui renvoie la moyenne des notes.

```
def moyenne(notes):
    s = ............
    for ............ in notes.values():
        s = ............
    return s / ............
```

2. Compléter la fonction `eleves_admis(notes)` qui renvoie la *liste* des prénoms ayant une note supérieure ou égale à 10.

```
def eleves_admis(notes):
    admis = ............
    for ............, ............ in notes.items():
        if ............:
            admis.append(............)
    return admis
```

3. Tester les deux fonctions sur le dictionnaire ci-dessus. Les résultats attendus sont 11,4 pour la moyenne et `["Alice", "Chloé", "David"]` pour les admis.

5 Application : compter les occurrences

Exercice 5 *Compter les lettres d'un mot*

On souhaite construire un dictionnaire qui, à chaque lettre d'un mot, associe le nombre de fois où elle apparaît. Par exemple, pour `"abracadabra"`, on attend :

```
{"a": 5, "b": 2, "r": 2, "c": 1, "d": 1}
```

1. Compléter la fonction `compter_lettres(mot)` :

```
def compter_lettres(mot):
    occurrences = ............
    for lettre in mot:
        if lettre in occurrences:
            occurrences[lettre] = ............
        else:
            occurrences[lettre] = ............
```

```
    return occurrences
```

2. Tester avec `"abracadabra"`, puis avec `"banane"`.
3. Comment cette fonction permettrait-elle de répondre rapidement à la question : « quelle est la lettre la plus fréquente dans un texte ? » ?

6 Liste ou dictionnaire ?

Exercice 6 *Choisir la bonne structure*

Pour chaque situation, indiquer si une **liste** ou un **dictionnaire** est la structure la plus adaptée. Justifier en une phrase.

1. Stocker les températures relevées chaque heure pendant 24 h.
2. Stocker, pour chaque ville française, sa population.
3. Stocker la suite des notes obtenues en NSI au cours de l'année (dans l'ordre).
4. Stocker, pour chaque élève, sa note sur 20.
5. Stocker la liste des sommets d'un graphe.
6. Stocker, pour chaque sommet d'un graphe, la liste de ses voisins.

7 Ce qu'il faut retenir

Propriété 3 **(Création et accès)** Un dictionnaire se note entre accolades : `{cle1: val1, cle2: val2, ...}`. L'accès à une valeur se fait par `dico[cle]` ; on ajoute ou modifie une entrée par `dico[cle] = valeur` ; on supprime par `del dico[cle]`.

Propriété 4 **(Test d'appartenance)** `cle in dico` teste la présence d'une clé. C'est la manière sûre d'éviter une `KeyError` avant un accès.

Propriété 5 **(Parcours)** Les trois écritures à connaître :

— `for cle in dico:` parcourt les clés ;

— `for valeur in dico.values():` parcourt les valeurs ;

— `for cle, valeur in dico.items():` parcourt les couples.

Remarque 2. Un dictionnaire se distingue d'une liste par :

— l'**indexation par clé** (et non par position) ;
— l'**absence d'ordre** historique (les éléments ne sont pas « rangés » comme dans une liste, même si Python conserve l'ordre d'insertion à partir de la version 3.7) ;
— un accès en temps quasi-constant à une valeur connaissant sa clé, là où une liste impose un parcours.

Les dictionnaires sont fondamentaux pour représenter des associations : répertoires, comptages, graphes, configurations, etc.