III. Dictionnaires

La manipulation des dictionnaires a été vue en classe de Première. Les notions abordées sont rappelées brièvement à la fin de ce cours : je vous conseille de revoir rapidement ces instructions.

1) Définition du type abstrait

Un dictionnaire est un type de **données linéaires** associant à un ensemble de **clés**, un ensemble correspondant de **valeurs**.

On retrouve une structure qui ressemble beaucoup à une liste. Toutefois, au lieu d'associer chaque élément à un indice de position comme dans une liste, on associe chaque élément (=valeur) à une clé. On dit qu'un dictionnaire contient des couples (clé, valeur).

Rem : Le type abstrait ne spécifie pas si les clés doivent être uniques. Dans le cas où elles le sont, le dictionnaire est appelé "**Map**" ou "**tableau associatif**".

Son interface propose ces primitives :
- ajout d'une nouvelle valeur associée à une nouvelle clé (on parlera de nouveau couple clé-valeur)
- modification d'une valeur associée à une clé existante
- suppression d'un couple clé-valeur
- récupération de la valeur associée à une clé donnée.

Exemple :
Un répertoire téléphonique est un exemple de tableau associatif :
- les clés sont les noms ;
- les valeurs sont les numéros de téléphone.

En Python, il existe nativement une structure de tableau associatif : le **dictionnaire**.

2) Dictionnaire et temps d'accès aux données

Rappelons pour l'instant qu'un dictionnaire est de la forme :

```
contacts = {"Antoine":"0648653125", "Kylian":"0712369503",
"Paul":"0613784596"}
```

et qu'on accède aux valeurs par la syntaxe :
`contacts["Kylian"]` renvoie `'0712369503'`

Intéressons-nous à la problématique du temps d'accès aux données, par une expérience de mesure de temps d'exécution permise par le module timeit de Python.

❐ — Exercice **1** —

Préparation des mesures
Considérons deux fonctions `fabrique_liste(n)` et `fabrique_dict(n)` renvoyant respectivement :
- le tableau des n premiers carrés parfaits
- le dictionnaire dont les clés sont les n premiers carrés parfaits et les valeurs sont 42.

Le contenu de ces listes ou dictionnaires n'a aucune importance. En effet, dans nos mesures, nous cherchons à accéder à la complexité **dans le pire des cas** i.e. le cas où la valeur n'apparait pas dans le tableau.

Afin de faire des mesures de temps d'exécution précises, nous allons utiliser la bibliothèque `timeit` que nous allons importer.

Pour faire des mesures de temps avec timeit, nous allons utiliser la fonction timeit. Cette fonction prend pour paramètre l'instruction que l'on souhaite tester sous forme de **chaine de caractères**. Si cette fonction fait appel à des variables internes à la fonction, on doit utiliser `globals = globals()`. `number` représente le nombre de fois où nous souhaitons relancer la fonction, afin de réaliser une moyenne.

Par exemple :

```
n = 10
a=timeit.timeit('[i for i in range(n)]', globals=globals(), number=100)/100)
```

La variable a est 8.835599999999777e-07 : c'est le temps mis pour créer une liste de 10 éléments, moyenné sur 100 essais.

❐ — Exercice **2** —

Temps de recherche pour le tableau

- Grâce au module timeit, nous allons mesurer le temps nécessaire pour faire une recherche de l'élément "a" pour un tableau lst de 10 éléments, 100 éléments, 1000 éléments. Vous enregistrerez le résultat dans un dictionnaire "taille tableau : temps recherche" qui nous servira à réaliser une représentation graphique.
- Par rapport aux résultats temporels obtenus, quelle est la complexité de la recherche d'une valeur dans un tableau ?

Temps de recherche pour un dictionnaire

- Grâce au module timeit, nous allons mesurer le temps nécessaire pour faire une recherche de la clé "a" pour un dictionnaire lst de 10 éléments, 100 éléments, 10000 éléments. Vous enregistrerez le résultat dans un dictionnaire "taille dictionnaire : temps recherche" qui nous servira à réaliser une représentation graphique.
- Par rapport aux résultats temporels obtenus, quelle est la complexité de la recherche d'une valeur dans un dictionnaire ?

- Si vous avez le temps, réalisez un graphique à l'aide de matplotlib afin de comparer visuellement les temps de recherches pour les tableaux et pour les dictionnaires. On prendra une échelle logarithmique sur l'axe des ordonnées et des abscisses (échelle dite loglog).

Vous avez du voir qu'il y a donc une différence fondamentale à connaître entre les temps de recherche d'un éléments à l'intérieur :
- une liste : temps proportionnel à la taille de la liste.
- un dictionnaire : temps constant, indépendant de la taille de la liste.

Pour aller plus loin… Malheureusement, l'explication de cette différence est complètement hors-programme bien qu'elle soit très intéressante d'un points de vue intellectuelle. Si vous voulez en savoir davantage, je vous invite à regarder la très bonne vidéo disponible à l'adresse ci-dessous :

https://urlz.fr/enl1 (youtube : Wandida — les tables de hachage).

3) Rappel sur les dictionnaires

Tout ce suit provient du cours de Première et doit être su. Si vous n'êtes pas certain de connaitre certaines notions, reprenez-les à l'aide de ce qui suit.

`dressing = {"pantalons" : 3, "pulls":4, "tee-shirts":8}`
`dressing["pulls"]` permet d'accéder à la valeur de la clé "pulls" ie. 4

❐ — Exercice **1** —

- Que va renvoyer `vocabulaire["suivant"]` ?

```
vocabulaire = {"navigateur":"browser", "précédent":"back",
"suivant":"forward"}
```

- Que va renvoyer `AlanTuring["décès"]` ?

```
AlanTuring = {"naissance":(23,6,1912), "décès":(12,6,1954), "lieu
naissance":"Londres",  "lieu décès":"Wilmslow"}
```

Définition :
Un dictionnaire est une donnée composite non ordonnée.
Il fonctionne par un **système d'association clé/valeur**. Les clés, comme les valeurs, peuvent être de types différents. En Python, un dictionnaire est délimité par des accolades.

Rappel :
- crochets [ ] -> listes
- parenthèses ( ) -> tuples
- accolades { } -> dictionnaires

❐ — Exercice **2** —

- À partir de l'exercice 1, quelle instruction permet d'afficher sur la console :

```
{'navigateur': 'browser', 'précédent': 'back', 'suivant': 'forward'}
```

- Que va renvoyer l'instruction `type(vocabulaire)` ?

Il est possible d'obtenir la liste des clés et des valeurs avec la méthode keys() et la méthode values.
- `dressing.keys()` renvoie `dict_keys(['pantalons', 'pulls', 'tee-shirts'])`
- `vocabulaire.values()` renvoie `dict_values(['browser', 'back', 'forward'])`

Création d'un dictionnaire vide :
On crée un dictionnaire vide par l'instruction :
monDico = dict() ou unAutreDico = {}

Ajout / Modification d'un élément dans un dictionnaire :
Il suffit d'ajouter ou d'écraser une paire clé : valeur

`dressing["chaussettes"] = 12` va modifier le dressing :
`{'pantalons': 3, 'pulls': 4, 'tee-shirts': 8, 'chaussettes': 12}`

❐ — Exercice **3** —

❖ Modifiez le dressing en enlevant une paire de chaussettes.

Suppression d'une valeur
On utilise l'instruction `del`, fonctionnant également pour les tableaux.

`del dressing["chaussettes"]` supprime bien l'entrée chaussettes :
`print(dressing)` renvoie : `{'pantalons': 3, 'pulls': 4, 'tee-shirts': 8}`

❐ — Exercice **4** —

Créer une fonction `achat(habit)` qui augmente de 1 le nombre d'habits (pantalon, pull ou tee-shirt) de mon dressing.

Exemple :

```
print(dressing)
achat("pantalons")
print(dressing)

{'pantalons': 3, 'pulls': 4, 'tee-shirts': 8}
{'pantalons': 4, 'pulls': 4, 'tee-shirts': 8}
```

Test d'appartenance à un dictionnaire
Le mot `in` permet de tester l'appartenance d'une clé à un dictionnaire. Un booléen est renvoyé.

`"cravates" in dressing` renvoie `False`
`"pulls" in dressing` renvoie `True`