TP-Cours d'Informatique 04

Dictionnaires sous Python

ITC – PCSI^2 2024-2025 – Lycée Fabert (Metz) – D.Mengel

Arès avoir découvert les listes, nous nous intéresserons lors de cette séance à une autre structure de données : **les dictionnaires**.

Sans entrer dans les détails, sachez que les dictionnaires ont un temps d'accès, d'effacement, un test d'appartenance ...très rapide car indépendant du nombre d'éléments. Cela les rend très utiles pour certaines applications, notamment lorsqu'on doit manipuler un nombre important de données.

I. Dictionnaires : opérations courantes et parcours

I.1. Différences et similitudes avec les listes

Différences

- Une liste L est indexée par les entiers 0, 1, ..., **len** (L)−1 et si $i \in [\![0;$**len**(L)-1$]\!]$, alors L[i] permet d'accéder à l'élément d'indice i de la liste L.
- Un dictionnaire d est indexé par des clés (**keys**) et si cle est une clé de d, alors d[cle] permet d'accéder à la valeur (**value**) associée à la clé cle.

Un peu de vocabulaire : Les couples **(keys,values)** constituent les objets **items** du dictionnaire.

- Les clés n'apparaissent qu'une fois dans un dictionnaire et sont de type non mutable : entier, réel, chaine de caractères, booléen mais ni des listes ou des dictionnaires.
- Les valeurs sont de tout type et peuvent apparaitre plusieurs fois.

Similitudes

- longueur : **len**(d) donne le nombre d'objets dans un dictionnaire d,
- test d'appartenance : cle **in** d indique si la clé cle est présente dans le dictionnaire d,
- pop(cle) permet de retirer la clé cle (et sa valeur) du dictionnaire d.

Remarque : appliquée à une liste L, L.pop() retire le dernier élément de L mais pour un dictionnaire d on doit obligatoirement préciser la clé de l'objet qu'on souhaite retirer d.pop(cle)

I.2. Opérations courantes sur un dictionnaire

Assez de théorie, lancez Pyzo et testez les opérations suivantes dans la console.

```
1 fruits={} # ou fruits=dict()
2 type(fruits)
3 len(fruits)
4
5 fruits['pommes']=23
6 fruits['prunes']=27
7 fruits
8 len(fruits)
```

Listing 1 – Construction d'un dictionnaire vide puis remplissage

```
fruits={'bananes':6,'poires':18,'pommes':23,'prunes':27,'abricots':7}
'bananes' in fruits
'citrons' in fruits
```

Listing 2 – Création d'un dictionnaire rempli et test d'appartenance

```
listeFruits=[('bananes',6),('poires',18),('pommes',23),('prunes',27),('
    abricots',7)]
fruits2=dict(listeFruits)
fruits2
```

Listing 3 – Création à partir d'une liste de tuples

```
fruits['abricots']
```

Listing 4 – Consulter un élément particulier

```
fruits['poires']=fruits['poires']+1
fruits
```

Listing 5 – Mise à jour d'une value

```
fruits.pop('bananes')
fruits
```

Listing 6 – Retirer un élément

I.3. Parcourir un dictionnaire

Cela revient à dresser la liste des éléments présents dans un dictionnaire. On peut le parcourir par clés, valeurs ou objets en utilisant les techniques suivantes. Testons les codes suivants dans la console.

```
for fruit in fruits.keys(): # le .keys() est facultatif
    print(fruit)

for fruit in fruits:
    print(fruit)
```

Listing 7 – Parcours par clés

```
for nbre in fruits.values():
    print(nbre)
```

Listing 8 – Parcours par valeurs

```python
for objet in fruits.items():
    print(objet)

for fruit,nbre in fruits.items():
    print("il y a ",nbre, fruit," dans mon panier.")
```

Listing 9 – Parcours par objet

```python
list(fruits)
```

Listing 10 – Obtention de la liste des clés présentes dans un dictionnaire par la méthode list()

II. Applications

Passons maintenant dans l'éditeur de Pyzo pour résoudre les exercices suivants.

Exercice 1 (Création et manipulation d'un dictionnaire) :

1. Créez un dictionnaire de nom 'moi' dans lequel vous renseignerez votre Nom, Prénom, et Age.
2. C'est votre anniversaire, ajoutez-vous un an.
3. Ajoutez la couleur de vos yeux dans le dictionnaire (clé "Couleur des yeux").
4. Parcourez et affichez le contenu du dictionnaire.
5. Vous ne souhaitez plus indiquer votre âge modifiez le dictionnaire.

Exercice 2 (comptage des éléments d'un « tableau » à l'aide d'un dictionnaire) :

On souhaite compter le nombre de fois où on rencontre les différentes caractères dans une chaine de caractères.

1. Créez une fonction utilisant un dictionnaire qui permette de remplir cette tâche.
2. Testez le programme avec la chaîne de caractère suivante :
   "J'adore programmer en Python, c'est si intuitif! Surtout en TP avec mes professeurs de PCSI2!!".
3. Modifiez la fonction pour qu'elle ne compte que les lettres de l'alphabet et leurs versions accentuées (et non les espaces, ponctuations ...). On pourra utiliser la méthode lower() sur la chaine pour la passer en minuscule.
4. On souhaite utiliser le programme pour déterminer le nombre de fois où on rencontre chaque lettre dans la liste des mots français.

   - Téléchargez la liste en suivant ce lien :
     https://www.pcsi2.net/fabert/wp-content/uploads/physique/mots-francais.txt
   - On rappelle comment récupérer le contenu du fichier (Cf TP-Cours 01, exercice 3).

   ```python
   file = open("mots-francais.txt",'r',encoding="utf-8")
   listeMots = file.read().split()
   file.close()
   ```

   - En déduire le nombre de fois où on rencontre la lettre a. Même question pour la lettre ù.
   - Faire trouver et afficher la lettre que l'on rencontre le plus de fois et combien de fois ?
     Réponse : La lettre e est celle qui apparait le plus : 22171 fois.

III. Utilisation et intérêt des dictionnaires

III.1 Tests d'appartenance

Nous allons tenter de comparer, sur un exemple, les coûts d'un test d'appartenance (c.-à-d. d'une recherche) d'un élément dans une liste ou un dictionnaire.

Analysez le programme Python ci-dessous puis décrivez les tâches réalisées par les différentes parties/-zones de code. Enfin, interprétez le graphique obtenu :

```
1  import time
2  import random as rd
3  import matplotlib.pyplot as plt
4
5  tempsListe = []
6  tempsDictionnaire = []
7  nbMesures = 0
8
9  for n in range(100,1000,20):
10
11     D = {}
12     L = []
13     nbMesures = nbMesures + 1
14     print(nbMesures)
15
16     while len(D)<n:
17         hasard = rd.randint(0,100000)
18         if hasard not in D:
19             D[hasard] = True
20             L.append(hasard)
21
22     tdebut = time.time()
23     for k in range(100000):
24         k in D
25     tfin = time.time()
26     tempsDictionnaire.append(tfin-tdebut)
27
28     tdebut = time.time()
29     for k in range(100000):
30         k in L
31     tfin = time.time()
32     tempsListe.append(tfin-tdebut)
33
34 plt.figure()
35 plt.plot([20*i+100 for i in range(nbMesures)],tempsListe, color = "blue"
       , label="100000 tests effectués sur une liste")
36 plt.plot([20*i+100 for i in range(nbMesures)],tempsDictionnaire, color =
       "pink", linestyle="--",label="100000 tests effectués sur un
      dictionnaire")
37 plt.legend()
38 plt.xlabel('Taille de la liste/du dict.')
39 plt.ylabel('Durée en s.')
40 plt.show()
```

Listing 11 – Durée de recherche de 100 000 valeurs dans une liste ou un dictionnaire

Zone 1 :

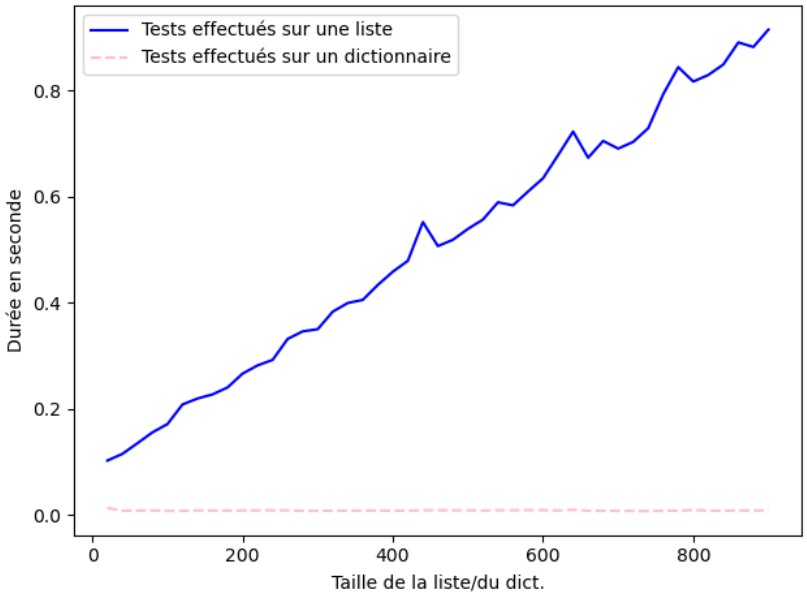
Zone 2 :

Zone 3 :

Zone 4 :

Zone 5 :

Zone 6 :



III.2 Utilisation de dictionnaires en boite noire

Dans ce paragraphe, nous allons voir comment utiliser un dictionnaire pour créer une liste d'entiers deux à deux distincts. Nous comparerons le temps d'exécution de cette fonction avec une fonction réalisant la même tâche mais n'utilisant que des listes.

1. Écrire la fonction LAleatoire(nbValeurs,a,b) → **list** qui crée une liste aléatoire de nbValeurs entiers deux à deux distincts tous compris entre a:**int** et b:**int** . Cette fonction utilisera le test d'appartenance d'un entier à une liste avant d'y ajouter cet entier.
2. Écrire la fonction LAleatoireBN(nbValeurs,a,b) → **list** qui crée une liste aléatoire de nbValeurs entiers deux à deux distincts tous compris entre a:**int** et b:**int** . Cette fonction utilisera le test d'appartenance d'un entier à un dictionnaire avant d' ajouter cet entier à une liste.

III.3 Tri par comptage

Le tri par comptage prend comme argument une liste L d'entiers. L'idée est de former dans un premier temps et par comptage un dictionnaire D donnant le nombre d'occurences de chaque entier de la liste L dans L.

1. Écrire la fonction comptage(L) → **dict** qui compte et range dans un dictionnaire les éléments de L auxquels on associe le nombre d'occurences de ces éléments dans L.
2. Écrire la fonction triComptage(L) → **list** de paramètre la liste d'entiers L et qui renvoie une liste triée formée des éléments de L.

   Pour ce faire, on détermine le minimum et le maximum de L et on parcourt tous les entiers entre ces deux valeurs en vérifiant s'ils appartiennent ou non au dictionnaire D obtenu en utilisant la fonction précédente à partir de L. À chaque fois qu'on qu'un tel entier est dans D, on sait donc combien de fois celui-ci apparait dans L. En partant d'une liste de 0 de même longueur que celle de L, on peut alors construire, à la volée, la liste triée formée des éléments de L