Dictionnaires

Contexte

Contrairement aux listes, tuples, chaines de caractère et tableaux numpy que vous avez vu en SUp et 3/2, la notion de dictionnaire est nouvelle. Un cours–TD de révision (3/2)

1 Rappels

Un dictionnaire est une structure de données d qui associe des **clés** et des **valeurs**. Comme pour une fonction en maths, une valeur peut être associée à plusieurs clés, mais à toute clé est associée une unique valeur.

On dispose des opérations suivantes :

1. créer un dictionnaire vide

```
d={}
OU
d=dict()
```

2. tester si c est bien une clé

```
c in d
```

3. ajouter une association (clé, valeur)

```
d[nouvellecle]=v
```

4. modifier une association (clé, valeur)

```
d[anciennecle]=NouvelleValeur
```

5. Supprimer une clé

```
del(d[c])
OU
d.pop(c)
```

6. parcourir toutes les clés

```
for c in d:
```

7. créer un petit dictionnaire. Par exemple le code :

```
d={1:3, 'a': [1,2,3], '12':'douze', (4,2,1.1):{1.1:'un dico dans un dico'}}
```

a le même effet que le code :

```
d=dict()
d2=dict()
d2[1.1]='un dico dans un dico'
d[1]=3
d['a']=[1,2,3]
d['12']='douze'
d[(4,2,1.1)]=d2
```

8. accéder à la longueur du dictionnaire (nombre de clés)

Remarques

1. La syntaxe ressemble un peu à celle des listes
2. Pour un dictionnaire et contrairement aux listes, de nombreux objets peuvent être utilisés comme **clé** : int, float, str, tuple[a].
3. N'importe quel objet python peut être une valeur.
4. La complexité temporelle de toutes ces opérations est en $O(1)$, sauf le parcours d'un dictionnaire
5. si A est un structure attention :

```
for x in A:
```

permet de parcourir les **clés** si A est un dictionnaire, et les **valeurs** si A est une liste, une chaine de caractère ou un tuple.

6. Les dictionnaires de sont pas ordonnés : le dictionnaire `'h':False, 2:True, 'b':12` et le dictionnaire `'b':12, 2:True, 'h':False` sont identiques.

a. Le critère précis est que l'objet soit **hashable** et c'est le cas en pratique s'il est **non mutable**

2 Exercices

2.1 De base

Exercice 1 Ecrire une boucle qui parcourt toutes les clés d'un dictionnaire et écrit les couples (clé, valeur).

Exercice 2 Ecrire une fonction qui prend en entrée un dictionnaire et un paramètre c et renvoie en sortie True si c est une clé de d et False sinon.

Exercice 3

1. Ecrire une fonction qui prend en entrée un dictionnaire d et renvoie son « dictionnaire inverse » : un dictionnaire d', telles que les clé de d' sont les valeurs de d, et pour d'[c]=[liste des clés de d qui pointent vers c]
   Exemple : `inverse({1:'a', 2:'b', 3:'a'})` est `{'a':[1,3], 'b':[2]}`
2. Ecrire une fonction qui teste si un dictionnaire est « injectif ».
3. Ecrire une fonction qui teste si deux dictionnaires ont les mêmes clés
4. Ecrire une fonction qui reçoit deux dictionnaires d1, d2 et retourne leur intersection, c'est-à-dire le dictionnaire dont les clés sont communes à d1 et d3 et la valeur associée est le couple des deux valeurs initialement présentes dans d1 et d2.
5. Ecrire une fonction qui reçoit deux dictionnaires d1, d2 et retourne leur union, c'est-à-dire le dictionnaire dont les clés sont ... et la valeur associée est ...
6. Ecrire une fonction qui `compose` reçoit deux dictionnaires d1, d2 et retourne le dictionnaire d3 ainsi défini : c est une clé de d3 si c est un clé de d1 et d1[c] un clé de d2. On a alors d3[c]=d2[d1[c]].

Exercice 4 Ecrire une fonction `comptage` qui prend en entrée une chaine de caractère, et renvoie en sortie un dictionnaire dont les clés sont les caractères de cette chaine, et les valeurs le nombre de fois où le caractère apparait. Par exemple `comptage('ananas')` renverra le dictionnaire `{'a':3, 'n':2, 's':1}`.

2.2 Intermédiaire

Exercice 5 Ecrire une fonction qui prend en entrée deux chaines de caractères et teste si elles sont anagrammes l'une de l'autre. Complexité requise linéaire en la taille des chaines.

Exercice 6 Soit $N \in \mathbb{N}^*$. Ecrire une fonction qui reçoit une liste d'entiers compris entre 0 et N et retourne la liste triée, avec un coût **linéaire en (N+la taille de la liste)**.
Et si les entiers ne sont plus compris entre 0 et N ? Complexité ?

2.3 Vers la programmation dynamique

Exercice 7 Soit $a \in \mathbb{N}^*$ La suite de Syracuse partant de n est la suite définie par $u_0 = a$, $u_{n+1} = u_n/2$ si u_n est pair et $u_{n+1} = 3u_n + 1$ si u_n est impair.
Que se passe-t-il si on choisit $a = 1$?
Lorsque u_n vaut 1, on décide dont arrêter les calculs.

1. Programmer la suite et éxécuter pour plusieurs valeurs de a. Conjecture ?
2. On appelle **temps de vol de** a le premier entier n tel que $u_n = 1$. Programmer une fonction qui reçoit a et retourne son temps de vol.
3. Programmer une fonction qui reçoit n et retourne tous les temps de vol de 1 à n. Pour minimiser la complexité temporelle, utiliser un dictionnaire.

Exercice 8

1. Ecrire une fonction récursive naïve permettant de calculer les coefficients binomiaux grâce à la formule de Pascal. Montrer que sa complexité est exponentielle.
2. Expliquer le fonctionnement du code suivant :

```
1
2 def binom(n,p):
```

```
    if p>n:
        return 0
    M=-np.ones((n+1,p+1))
    for i in range(n+1):
        for j in range(p+1):
            if j==0 or j==i:
                M[i,j]=1
    def aux_rec(i,j):

        if M[i,j]==-1:
            b=aux_rec(i-1, j)+aux_rec(i-1, j-1)
            M[i,j]=b
            return b
        else:
            return M[i,j]

    return aux_rec(n,p)
```

3. Proposer un code plus court et n'employant pas un tableau numpy mais un dictionnaire. **Ce principe s'appelle la mémoïsation**.

3 Solutions