Dictionnaires

1. Exercices

1.1. Utilisation des dictionnaires

1.1.1. Températures

On dispose des températures à Montélimar à 8h00 dans un dictionnaire :

```
temp={'J1':-10, 'J2':-9, 'J3':-4, 'J4':0, 'J5':-1, 'J6':4, 'J7':-5, 'J8':1, 'J9'
    :-2}
```

1. Écrire une fonction `moyenne` qui prend en argument un dictionnaire `d` du type de celui défini ci-dessus et qui renvoie la valeur moyenne des températures.

```
def moyenne3(d):
        S=0
        for x in d: #parcours des cles de d
                S=S+d[x] #on ajoute la temperature de la cle x
        return S/len(d)
```

2. Écrire une fonction `froid(d,T0)` qui prend en argument un dictionnaire d du type de celui défini ci-dessus et qui renvoie la liste des jours et le nombre de jours où la température a été inférieure à une certaine température `TO`.

```
def froid(d,T0):
        N=0 #initialisation du nbre de jours
        liste_jour=[] #initialisation de la liste des jours ou T<T0
        for x in d: #parcours des cles de d
                if d[x]<T0: #on a trouve un jour ou il faisait froid
                        N=N+1 # on ajoute un jour
                        liste_jour=liste_jour+[x] #on ajoute le jour
        return N,liste_jour
```

1.1.2. Moyennes

Étant donné un dictionnaire python dont les clés sont les noms des élèves et les valeurs sont les listes des notes et qui renvoie un autre dictionnaire dont les clés sont le nom des élèves et les valeurs la moyenne de leur note. Par exemple : `nom_moy({'Louise':[12,15,12], 'Gabriel':[15,17,16], 'Léon':[8,18,7]})` renverra `{'Louise': 13, 'Gabriel': 16, 'Léon': 11$}`.

Indication : On pourra commencer par programmer une fonction intermédiaire.

```
def moyenne(L):
        M=0
        for x in L:
                M=M+x
        return M/len(L)
def nom_moy(d):
        d_moy ={} # dictionnaire des moyennes
        for c in d: # parcours des cles
        d_moy[c]=moyenne(d[c]) # calcul de la moyenne des valeurs associees a c
return d_moy
```

1.1.3. Occurrences dans une chaîne de caractère

Écrire une fonction `occurrences(texte)` qui prend en argument une chaîne de caractère texte et renvoie un dictionnaire dont les clés sont les lettres qui apparaissent dans le texte et les valeurs le nombre d'occurrences de ces lettres.

Par exemple : `occurrences('ACCTAGCCCTA')` renverra `'A':3, 'C':5, 'T':2, 'G':1}`.

```
def occurrences(texte):
        occ={x:0 for x in texte} # dictionnaire des occurrences
        for x in texte: # x parcourt la chaine de caracteres
                occ[x]+=1
        return occ
```

1.1.4. Min-Max dans un dictionnaire

Écrire une fonction `min_max` qui prend en argument une liste de nombres non vide et renvoie un dictionnaire dont les clés sont les chaînes `"min"` et `"max"` avec pour valeurs respectives le minimum et le maximum des nombres de la liste.

Par exemple : `min_max([8, 5, 9, 3, 1, 7]` renverra `{"min":1, "max": 9}`.

```
def min_max(L):
        m,M=L[0],L[0]
        for i in range(1,len(L)):
                if m<L[i]:
                        m=L[i]
                if M>L[i]:
                        M=L[i]
        return {"min":m,"max":M}
```

1.1.5. Matrice

1. Comment peut-on manipuler des matrices en Python en utilisant un dictionnaire?

clé : 2-uplet qui indique la position du coefficient indiqué comme valeur de la clé.

On s'intéresse ici aux matrices parcimonieuses, c'est-à-dire dont la plupart des coefficients sont nuls. Une telle matrice M de dimensions (n,p) pourra être codée par un dictionnaire ayant pour couples clefs/valeurs :

→ `'dim'` : (n,p)

→ (i,j) : M_{ij} pour chaque couple (i,j) tel que $M_{i,j} \neq 0$.

2. Donner le dictionnaire qui code la matrice : $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 \end{pmatrix}$

```
M={'dim':(2, 4), (1, 2):4}
```

3. Proposer une fonction d'addition de deux matrices (de même dimension).

```
def somme_mat(M1,M2):
        assert M1['dim']==M2['dim'] , "les deux matrices doivent etre de meme
            dimension"
        M={'dim':M1['dim']}
        for c1 in M1:
                if c1!="dim":
                        if c1 in M2:
                                M[c1]=M1[c1]+M2[c1]
                        else:
                                M[c1]=M1[c1]
        for c2 in M2:
                if c2!="dim" :
                        if c2 not in M:
                                M[c2]=M2[c2]
        return M
```

4. Si la première matrice contient c coefficients non nuls, et la seconde c', quelle est la complexité temporelle de cet algorithme?

En $O(c,c')$. Sinon, pour une matrice écrite avec une liste de liste c'est en $O(np)$ (deux boucles `for` imbriquées.

1.1.6. Quelle heure est-il?

Un site de voyage permet de calculer les temps de parcours entre deux villes sous forme d'un dictionnaire qui contient 4 clefs (`'jours'`,`'heures'`,`'minutes'` et `'secondes'`) dont les valeurs associées représentent respectivement les durées en jours, heures, minutes et secondes pour le voyage, le tout de manière unique,

c'est-à-dire que 27 heures vaut en fait 1 jour et 3 heures. Néanmoins, si vous faites plusieurs escales, vous voudriez bien connaître la durée totale que vous aurez passée dans les transports.

On peut décomposer ce problème en plusieurs sous-problèmes plus simples à résoudre.

1. Écrire une fonction `decomposition(duree)` qui prend en argument une durée exprimée en secondes et renvoie un dictionnaire à valeurs entières dont les clefs sont `'jours'`, `'heures'`, `'minutes'` et `'secondes'`.

```
def decomposition(duree):
        d={} # dictionnaire de retour
        N=24*3600 # nbre de secondes sur une journee
        j=duree//N # Nbre de jours entiers
        d['jours']=j
        h=(duree%N)//3600 # nbre d heures (3600 s) dans le nbre de secondes
            restantes
        d['heures']=h
        m=((duree%N)%3600)//60 # nbre de minutes (60s) dans le nbre de
            secondes restanteqs
        d['minutes']=m
        s=((duree%N)%3600)%60 # nbre de secondes restantes d['secondes']=s
        return d
```

2. Écrire la fonction inverse `secondes(dico)` qui prend en argument un dictionnaire du type précédent pour renvoyer la valeur correspondante en secondes.

```
def secondes(dico):
        N=dico['secondes']+dico['minutes']*60+dico['heures']*3600+dico['jours'
            ]*3600*24
        return N
```

3. En utilisant les deux fonctions précédentes, écrire la fonction `addition(dico1, dico2)` qui va additionner correctement deux dictionnaire `dico1` et `dico2` correspondant à deux voyages successifs et renvoyer le dictionnaire du même type correspondant à la durée totale du voyage.

```
def addition(dico1,dico2):
        # convertir les deux dictionnaires en une duree en secondes
        duree1=secondes(dico1)
        duree2=secondes(dico2)
        # duree totale en secondes
        duree_tot=duree1+duree2
        return decomposition(duree_tot) # on renvoie le dico correspondant a
            la duree totale en secondes
```

4. Écrire une fonction `affichage(dico)` qui affiche (à l'aide de `print`) le temps total passé en transport de manière un peu plus lisible pour le commun des mortels.

Par exemple, l'appel à la fonction renvoie :

```
>>> affichage({'jours': 3, 'heures': 22, 'minutes': 10, 'secondes': 54})
Vous allez voyager un total de 3 jours, 22 heures, 10 minutes et 54 secondes
```

```
def affichage(dico):
        print('Vous allez voyager un total de', dico['jours'],' jours, ', dico
            ['heures'] , ' heures, ' , dico['minutes'], ' minutes et ' , dico[
            'secondes'], ' secondes')
```

5. Enfin, écrire une fonction `temps_total(liste_de_durees)` qui prend en argument une liste (de taille arbitraire) de durées sous la forme des dictionnaires précédents et renvoie le temps total de parcours à l'aide de la fonction `affichage(dico)` précédente.

```
def temps_total(liste_de_durees):
        duree_tot=0
        for d in liste_de_durees:
                duree=secondes(d)
                duree_tot=duree_tot+duree
        return affichage(decomposition(duree_tot))
```

1.2. Dictionnaire et table de hachage

1.2.1. Double hachage

Le double hachage est l'une des meilleurs méthodes connues pour l'adressage ouvert. Il utilise une fonction de hachage de la forme :

$$\begin{array}{rccc} h: & N\times N & \longrightarrow & [\![0, m-1]\!] \\ & (k,i) & \mapsto & (h_1(k)+ih_2(k)) \bmod m \end{array}$$

où h_1 et h_2 sont des fonctions de hachages. i prend les valeurs suivantes :

→ Par défaut, $i = 0$.

→ S'il y a collision, on incrémente i de 1 , jusqu'à ne plus avoir de collision pour la clé considérée.

→ Il reprend la valeur 0 pour la clé suivantes ...

1. Insérer les clés : 5,28,19,15,20,33,12,17,10 dans une table de taille $m = 13$ avec $h_1(k) = k \bmod 13$ et $h_2(k) = 1 + (k \bmod 12)$.

→ $5 : h_1(5) = 5\%13 = 5; h_2(5) = 1 + (5\%12) = 6; h(5) = (5 + 0 \times 6)\%13 = 5$

→ $28 : h_1(28) = 28\%13 = 2; h_2(28) = 1 + (28\%12) = 5; h(28) = (2 + 0 \times 5)\%13 = 2$

→ $19 : h_1(19) = 19\%13 = 6; h_2(19) = 1 + (19\%12) = 8; h(19) = (6 + 0 \times 8)\%13 = 6$

→ $15 : h_1(15) = 15\%13 = 2; h_2(15) = 1 + (15\%12) = 4; h(15) = (2 + 0 \times 4)\%13 = 2.$

Il y a collision, on incrémente i de 1, $i = 1$.

$h(15) = (2 + 1 \times 4)\%13 = 6$, il y a collision, on incrémente i de 1 : $i = 2$

$h(15) = (2 + 2 \times 4)\%13 = 10$, ok

→ $20 : h_1(20) = 20\%13 = 7; h_2(20) = 1 + (20\%12) = 9; h(20) = (7 + 0 \times 9)\%13 = 7$

→ $33 : h_1(33) = 33\%13 = 7; h_2(33) = 1 + (33\%12) = 10; h(33) = (7 + 0 \times 10)\%13 = 7$

Il y a collision, on incrémente i de 1, $i = 1$.

$h(33) = (7 + 1 \times 10)\%13 = 4$.

→ $12 : h_1(12) = 12\%13 = 12; h_2(12) = 1 + (12\%12) = 1; h(12) = (12 + 0 \times 1)\%13 = 12$

→ $17 : h_1(17) = 17\%13 = 4; h_2(17) = 1 + (17\%12) = 6; h(17) = (4 + 0 \times 6)\%13 = 4$

Il y a collision, on incrémente i de 1, $i = 1$.

$h(17) = (4 + 1 \times 6)\%13 = 10$, il y a collision, on incrémente de 1 : $i = 2$

$h(17) = (4 + 2 \times 6)\%13 = 3$

→ $10 : h_1(10) = 10\%13 = 10; h_2(10) = 1 + (10\%12) = 11; h(10) = (10 + 0 \times 11)\%13 = 10$

Il y a collision, on incrémente i de 1, $i = 1$.

$h(10) = (10 + 1 \times 11)\%13 = 8$.

indice 0	
indice 1	
indice 2	28
indice 3	17
indice 4	33
indice 5	5
indice 6	19
indice 7	20
indice 8	10
indice 9	
indice 10	15
indice 11	
indice 12	12

2. Proposer une fonction en Python qui prend en argument une clé `c` (entier) et la taille `m` de la table, et renvoie la valeur de `h(c)`.

```
import numpy as np
m=13
tab_h=np.zeros(m) # table de hachage initialisee
def double_hachage(c,m):
        if c==0:
                tab_h[0]=c
                return 0
        i=0
```

```
        h1=c%m
        h2=1+(c%(m-1))
        h=(h1+i*h2)%m
        while tab_h[h]!=0: # si coefficient non nul dans le tableau, il y a
            collision
                i=i+1 # incremente i de un
                h=(h1+i*h2)%m # on calcule la nouvelle valeur de hachage
        tab_h[h]=c # ajout de c dans la case h
        return h


L=[5,28,19,15,20,33,12,17,10]
for x in L:
        double_hachage(x,13)
>>> tab_h
array([ 0., 0., 28., 17., 33., 5., 19., 20., 10., 0., 15., 0., 12.])
```

1.2.2. Polynôme

On considère des polynômes non nuls à coefficients entiers de degré quelconque mais qui ne contiennent pas plus de cinq monômes. On utilise un tableau de longueur $16 = 8 \times 2$ pour stocker les couples (degré, coefficient) dans lequel on pourrait stocker au maximum huit couples. Les places non occupées contiennent la valeur -1 .

La fonction de hachage h est la fonction identité : pour tout $n \in N, h(n) = n$. Donc à un degré qui vaut 10 , on associe le nombre 10, soit $h(10) = 10$. Ensuite, avec `10%8`, on obtient l'indice 2 et à cet indice on écrit le degré, (la clé), suivi du coefficient, (la valeur).

Par exemple, le polynôme $8 + 3x^{10} - 5x^{12}$ est stocké dans un tableau de la forme :

indice 0	0	8
indice 1	-1	-1
indice 2	10	3
indice 3	-1	-1
indice 4	12	-5
indice …	…	…

1. Donner le tableau correspondant au stockage du polynôme $2x^5 - 3x^{34} + 4x^{105}$.

degré 5 : $h(5) = 5$, puis `5%8=5`
degré 34 : $h(34) = 34$, puis `34%8=2`
degré 105 : $h(105) = 105$, puis `105%8=1`

indice 0	-1	-1
indice 1	105	4
indice 2	34	-3
indice 3	-1	-1
indice 4	-1	-1
indice 5	5	2

2. Quel est le problème avec par exemple le polynôme $8 - 5x^2 + 3x^{10}$?

degré 0 : `0%8=0`
degré 2 : `2%8=2`
degré 10 : `10%8=2` => collision !

3. En cas de collision, on décide d'utiliser la première place libre suivante. Les monômes sont entrés dans le tableau suivant l'ordre de lecture. Donner un exemple de polynôme de degré minimum qui génère une collision pour chaque monôme excepté le premier.

Il y a collision, si tous les restes des divisions euclidiennes des degrés sont égaux. Par exemple : $x + x^2 + x^{10} + x^{26} + x^{34}$

4. On envisage une autre possibilité de stockage avec deux tableaux, un tableau pour les couples (degré, coefficient) et un tableau pour les indices, les deux tableaux ayant pour capacité 8 . Avec le polynôme $4x^3 - 2x^5 + 4x^9$, on obtient les deux tableaux de la manière suivante :

→ dans le premier tableau, on écrit chaque degré avec le coefficient correspondant suivant l'ordre des degrés et on complète le tableau avec des 0 ;

→ dans le second tableau, on calcule $d\%8$ où d est un degré et on place à l'indice trouvé l'indice où on trouve le couple (degré,coefficient) dans le premier tableau. On complète le tableau avec des -1 .

Extraits des tableaux :

indice 0	3	4
indice 1	5	-2
indice 2	9	4
indice 3	0	0
indice …	…	…

indice 0	-1
indice 1	2
indice 2	-1
indice 3	0
indice 4	-1
indice 5	1

Donner les deux tableaux correspondant au stockage du polynôme $3x^5 - x^{18} + 7x^{20}$.

indice 0	5	3
indice 1	18	-1
indice 2	20	7
indice 3	0	0
indice 4	0	0
indice 5	0	0
indice 6	0	0
indice 7	0	0

indice 0	-1
indice 1	-1
indice 2	1
indice 3	0
indice 4	2
indice 5	-1
indice 6	-1
indice 7	-1

`5%8=3  18%8=2  20%8=4`