Dictionnaires : Exercices complémentaires – Eléments de correction

Partie 1 : Manipulation des dictionnaires

Exercice 1 : Températures

On dispose des températures à Bordeaux 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 ddu type de celui défini ci-dessus etqui renvoie la valeur moyenne des températures.

```
temp={'J1':-10,'J2':-9,'J3':-4, 'J4':0, 'J5':-1, 'J6':4, 'J7':-5, 'J8':1, 'J9':-2}
print(moyenne(temp))#-2.888888888888889
```

2. Écrire une **fonction froid(d,T0)** qui prend en paramètre un dictionnaire d du type de temp et une temperature t0. La fonction renvoie la liste des jours et le nombre de jours pour lesquels la température a été inférieure à t0.

```
print( froid(temp, -1) ) #(5, ['J1', 'J2', 'J3', 'J7', 'J9'])
```

```
def moyenne( d ):
    """ entrée : d, dictionnaire des températures
        sortie : flottant, moyenne des températures                """
    S=0
    for val in d.values(): #parcours des cles de d
        S = S + val
    return S / len( d )
temp={'J1':-10, 'J2':-9, 'J3':-4, 'J4':0, 'J5':-1, 'J6':4, 'J7':-5, 'J8':1,'J9':-2}
print(moyenne(temp))#-2.888888888888889


def froid( d , t0 ):
    """ entrées : d, dictionnaire des températures
                : t0, flottant, temperature plafond
        sorties : entier, correspondant au nombre de jours
                : liste, correspondant aux noms des jours           """
    nbJ=0 #initialisation du nbre de jours
    listeJours = [] #initialisation de la liste des jours ou T<T0
    for k in d: #parcours des cles de d
        if d[k]<t0: #on a trouve un jour ou il faisait froid
            nbJ=nbJ+1 # on ajoute un jour
            listeJours.append(k) #on ajoute le jour
    return nbJ,listeJours
print( froid(temp, -1) ) #(5, ['J1', 'J2', 'J3', 'J7', 'J9'])
```


Exercice 2 : Moyennes

Ecrire une fonction nomMoy qui prend en paramètre un dictionnaire 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.

Exemple :

```
print( nomMoy({'Lina':[12,15,12], 'Gabin':[15,17,16], 'Léa':[8,18,7]}))
#{'Lina': 13.0, 'Gabin': 16.0, 'Léa': 11.0}
```

```
def moyenne(L):
    moy=0
    for val in L:
        moy=moy + val
    return moy / len(L)

def nomMoy(d):
    d_moy ={} # dictionnaire des moyennes
    for k,v in d.items() : # parcours des cles
            d_moy[k]=moyenne(v) # calcul de la moyenne des valeurs associees a k
    return d_moy
print( nomMoy({'Lina':[12,15,12], 'Gabin':[15,17,16], 'Léa':[8,18,7]}))
#{'Lina': 13.0, 'Gabin': 16.0, 'Léa': 11.0}
```

Exercice 3 : Matrices

1. Proposer une structure de dictionnaire permettant de manipuler des matrices en Python

clé : 2-uplet (i,j) qui indique la position
valeur, : coefficient à la position (i ,j)

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

2. Définir le dictionnaire qui code la matrice :

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 \end{matrix}$$

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

3. Proposer une **fonction sommeMat** de 2 matrices (de même dimension).

```
mat = {'dim':(2, 4), (1, 2):4}
mat2= {'dim':(2, 4), (0,0):3, (1, 2):4}
print( somme_mat(mat,mat2) ) #{'dim': (2, 4), (1, 2): 8, (0, 0): 3}
```

```
def somme_mat(M1,M2):
    assert M1['dim']==M2['dim'] ,"les 2 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

mat = {'dim':(2, 4), (1, 2):4}
mat2= {'dim':(2, 4), (0,0):3, (1, 2):4}
print( somme_mat(mat,mat2) ) #{'dim': (2, 4), (1, 2): 8, (0, 0): 3}
```

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

Quelle serait la complexité dans le cas d'une matrice représentée sous forme de liste de listes.

En $O(cc')$.

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

Exercice 4 : Temps de parcours

Un site de voyages permet de calculer les temps de parcours entre 2 villes sous forme d'un dictionnaire qui comporte les 4 clefs suivantes : 'jours', 'heures', 'minutes' et 'secondes' dont les valeurs associées représentent respectivement les durées en jours, heures, minutes et secondes du voyage, le tout de manière unique.

Exemple : 27 heures se traduit par 1 jour et 3 heures.

Si le voyage comporte plusieurs escales, on souhaite connaître la durée totale que vous aurez passée dans les transports.

Pour résoudre ce problème, on le décompose en plusieurs sous-problèmes plus simples à résoudre.

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

```
dRes = decomposition( 100124 )
print(dRes) #{'jours': 1, 'heures': 3, 'minutes': 48, 'secondes': 44}
dRes2 = decomposition( 1000 )
print(dRes2)#{'jours': 0, 'heures': 0, 'minutes': 16, 'secondes': 40}
```

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

```
nbs = secondes(dRes)
print(nbs) #100124
```

3. A l'aide des 2 fonctions précédentes, écrire **la fonction addition(dico1, dico2)**qui qui va additionner correctement 2 dictionnaires d1 et d2 correspondant à deux voyages successifs et renvoyer un dictionnaire du même type correspondant à la durée totale du voyage.

```
dTotal = addition(dRes,dRes2)
print(dTotal) #{'jours': 1, 'heures': 4, 'minutes': 5, 'secondes': 24}
```

4. Écrire une fonction **affichage(dico)** qui affiche le temps total passé en transport de manière un peu plus lisible au format ci-dessous.

```
affichage(dTotal) #Le voyage dure au total 1 jours, 4 heures, 5 minutes et 24 secondes
```

5. Ecrire une **fonction tempsTotal(liste_de_durees)** qui prend en paramètre une liste (de taille arbitraire) de durées sous la forme des dictionnaires précédents et renvoie le temps total de parcours sous forme de dictionnaire.

```
L= [dRes, dRes2]
dtot = tempsTotal(L)
affichage(dtot)
```

```
def decomposition( duree ):
    """ entrées : duree, entier, represente la durée en secondes
        sortie : d, dictionnaire donc les clés sont jours, heures, minutes    """
    d = {} # dictionnaire de retour
    nbSecJours = 24*3600 # nbre de secondes sur une journee
    j=duree//nbSecJours
    # Nbre de jours entiers
    d['jours']=j
    h = (duree%nbSecJours)//3600 # nb d heures (3600 s) dans le nb de sec restantes
    d['heures'] = h
    m = ((duree%nbSecJours)%3600)//60 # nb de min (60s) dans le nb de sec restantes
```

```
    d['minutes']=m
    s=((duree%nbSecJours)%3600)%60 # nb de secondes restantes
    d['secondes']=s
    return d
# tests
dRes = decomposition( 100124 )
print(dRes) #{'jours': 1, 'heures': 3, 'minutes': 48, 'secondes': 44}
dRes2 = decomposition( 1000 )
print(dRes2)#{'jours': 0, 'heures': 0, 'minutes': 16, 'secondes': 40}

def secondes( d ):
    """ entrée : d, dictionnaire donc les clés sont jours, heures, minutes
        sortie : duree, entier, represente la durée en secondes            """
    duree =d['secondes']+d['minutes']*60+d['heures']*3600+d['jours']*3600*24
    return duree
#tests
nbs = secondes(dRes)
print(nbs) #100124

def addition( d1, d2):
    """ entrée : d1,d2: dictionnaires
        sortie : dictionnaire qui correspond à la somme de d1 et d2        """
    # conversion des 2 dictionnaires en une duree en secondes
    duree1 = secondes(d1)
    duree2 = secondes(d2)
    # duree totale en secondes
    duree_tot = duree1 + duree2
    return decomposition(duree_tot) # on renvoie le dictionnaire correspondant a la
 duree totale en secondes

dTotal = addition(dRes,dRes2)
print(dTotal) #{'jours': 1, 'heures': 4, 'minutes': 5, 'secondes': 24}

def affichage(d):
    """ entrée : d: dictionnaire     """
    print('Le voyage dure au total ', d['jours'],' jours, ', d ['heures'] , ' heure
s, ' , d['minutes'], ' minutes et ' , d[ 'secondes'], ' secondes')
affichage(dTotal) #Le voyage dure au total  1  jours,  4  heures,  5  minutes et  2
4  secondes

def tempsTotal( listeDurees ):
    """ entrée : listeDurees, liste de dictionnaires
        sortie : dictionnaire qui correspond à la somme des dictionnaires contenus
dans listeDurees         """
    duree_tot=0
    dr={'jours': 0, 'heures': 0, 'minutes': 0, 'secondes': 0}
    for d in listeDurees:
        dr=addition(dr,d)
    return  dr
L= [dRes, dRes2]
dtot = tempsTotal(L)
affichage(dtot)
```


Partie 2 : Dictionnaire et table de hachage

Exercice 5 : Double hachage

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

$$h : N \times N \rightarrow [0, m-1]$$
$$(k, i) \rightarrow (h1(k) + ih2(k)) \mod m$$

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

k	$h_1(k)$	$h_2(k)$	$h(k)$
5	$h_1(5) = 5\%13 = \mathbf{5}$	$h_2(5) = 1 + (5\%12) = \mathbf{6}$	$h(5) = (5 + 0 \times 6)\ \%\ 13 = \mathbf{5}$
28	$h_1(28) = 28\%13 = \mathbf{2}$	$h_2(28) = 1 + (28\%12) = \mathbf{5}$	$h(28) = (2 + 0 \times 5)\%13 = \mathbf{2}$
19	$h_1(19) = 19\%13 = 6$	$h_2(19) = 1 + (19\%12) = \mathbf{8}$	$h(19) = (6 + 0 \times 8)\%13 = \mathbf{6}$
15	$h_1(15) = 15\%13 = \mathbf{2}$	$h_2(15) = 1 + (15\%12) = \mathbf{4}$	$h(15) = (2 + 0 \times 4)\%13 = \mathbf{2}$. **Collision** on incrémente i de 1, $i = 1$. $h(15) = (2 + 1 \times 4)\%13 = 6$, **Collision** on incrémente i de 1 : $i = 2$ $h(15) = (2 + 2 \times 4)\%13 = \mathbf{10}$, ok
20	$h_1(20) = 20\%13 = \mathbf{7}$	$h_2(20) = 1 + (20\%12) = \mathbf{9}$	$h(20) = (7 + 0 \times 9)\%13 = \mathbf{7}$
33	$h_1(33) = 33\%13 = 7$	$h_2(33) = 1 + (33\%12) = \mathbf{10}$	$h(33) = (7 + 0 \times 10)\%13 = \mathbf{7}$ **Collision,** on incrémente i de 1, $i = 1$. $h(33) = (7 + 1 \times 10)\%13 = \mathbf{4}$.
12	$h_1(12) = 12\%13 = \mathbf{12}$	$h_2(12) = 1 + (12\%12) = \mathbf{1}$	$h(12) = (12 + 0 \times 1)\%13 = \mathbf{12}$
17	$h_1(17) = 17\%13 = \mathbf{4}$	$h_2(17) = 1 + (17\%12) = \mathbf{6}$	$h(17) = (4 + 0 \times 6)\%13 = 4$ **Collision,** on incrémente i de 1, $i = 1$. $h(17) = (4 + 1 \times 6)\%13 = 10$, **Collision,** on incrémente de 1 : $i = 2$ $h(17) = (4 + 2 \times 6)\%13 = \mathbf{3}$
10	$h_1(10) = 10\%13 = \mathbf{10}$	$h_2(10) = 1 + (10\%12) = \mathbf{11}$	$h(10) = (10 + 0 \times 11)\%13 = 10$ **Collision**, on incrémente de 1, $i = 1$. $h(10) = (10 + 1 \times 11)\%13 = \mathbf{8}$.

Représentation de la table :

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

2. Proposer une **fonction** en Python qui prend en argument une table de hachage tab_h de type array et une clé c(entier) qui renvoie la valeur de h(c) et agit par effet de bord sur tab_h.

```
import numpy as np

def double_hachage( tab_h , k   ):
    """ entrées : tab_h : array, table de hachage
                : k, entier qui correspond à la valeur à hacher
        sortie : h, la valeur hachée de k
        tab_h est modifié par effet de bord
    """
    m = np.size(tab_h)

    i=0
    h1 = k % m
    h2 = 1 + ( k % ( m - 1 ))
    h = ( h1 + i * h2 ) % m

    while tab_h[h]!=-1: #si coefficient different de - 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] = k # ajout de c dans la case h
    return h

print("Exercice sur le double hachage")
m = 13  #taille de la table
#tab_h = np.zeros(m)
tab_h = np.full( m, -1 ) # table de hachage initialisee avec des -1
""" ou
tab_h = np.empty(m, dtype=int) # table de hachage
tab_h.fill(-1)                 #initialisee avec des -1
"""
L = [5,28,19,15,20,33,12,17,10]
for val in L:
    print( double_hachage(tab_h, val) )
print( tab_h ) # [-1 -1 28 17 33  5 19 20 10 -1 15 -1 12]
```


Exercice 6 : Polynômes

On considère des polynômes non nuls à coefficients entiers de degré quelconque mais qui ne contiennent pas plus de 5 monômes. On utilise un tableau de longueur 16 = 8×2 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	degré	coefficient
0	0	8
1	-1	-1
2	10	3
3	-1	-1
4	12	-5
…	…	…

1. Donner le tableau correspondant au 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	degré	coefficient
0	-1	-1
1	105	4
2	34	-3
3	-1	-1
4	-1	-1
5	5	2

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

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 : $x + x^2 + x^{10} + {}_x26 + {}_x34$

4. On envisage une autre possibilité de stockage avec 2 tableaux :
   - un tableau pour les couples (degré, coefficient)
   - un tableau pour les indices,

les 2 tableaux ayant pour capacité 8 .

Avec le polynôme $4x^3 - 2x^5 + 4x$, on obtient les 2 tableaux de la manière suivante :

- Dans le 1er tableau, on écrit chaque degré avec le coefficient correspondant suivant l'ordre des degrés et on complète le tableau avec des 0.
- Dans le 2ème tableau, on calcule d%8 où d est un degré. 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	degré	coefficient
0	3	4
1	5	-2
2	9	4
3	0	0
…	…	…

Indice	IndiceCouple
0	-1
1	2
2	-1
3	0
4	-1
5	1

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

Indice	degré	coefficient
0	5	3
1	18	-1
2	20	7
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0

Indice	IndiceCouple
0	-1
1	-1
2	1
3	0
4	2
5	-1
6	-1
7	-1

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