Dictionnaires : Exercices complémentaires

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'])
```

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}
```

Exercice 3 : Matrices

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

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) : Mij pour chaque couple (i, j) tel que $Mi, j \neq 0$.

2. Définir le dictionnaire qui code la matrice :

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

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( sommeMat(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 ?

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  seco
ndes
```

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)
```

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 de hachage de la forme :

$$\begin{matrix} h : & N \times N & \rightarrow & [0, m-1] \\ & (k, i) & \rightarrow & (h1(k) + ih2(k)) \mod m \end{matrix}$$

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).

Valeur de la clé k	$h_1(k)$	$h_2(k)$	$h(k)$
5			
28			
19			
15			
20			
33			
12			
17			
10			

Représentation de la table :

Indice	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

2. Proposer une **fonction** double_hachage 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
    """
```


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).

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 : $2x^5 - 3x^{34} + 4x^{105}$.

Indice	degré	coefficient
0		
1		
2		
3		
4		
5		

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

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 $4x^3 - 2x^5 + 4x^9$,** 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 2 tableaux correspondant au stockage du polynôme $3x^5 - x^{18} + 7x^{20}$.

Indice	degré	coefficient
0		
1		
2		
3		
4		
5		
6		
7		

Indice	IndiceCouple
0	
1	
2	
3	
4	
5	
6	
7	