Usage des dictionnaires

1 Fonction de hachage

On considère des clés représentées par des chaînes de caractères codées sur un ensemble de 256 caractères (l'alphabet ASCII 8 bits par exemple) et l'on associe à chaque clé l'entier qu'elle représente en base 256. Ainsi, par exemple, puisque les caractères B, l, o et p correspondent aux valeurs 66, 108, 111 et 112 respectivement, la clé « Blop » est associée à l'entier :

$$66 \times 256^3 + 108 \times 256^2 + 111 \times 256 + 112 = 1114402672$$

Q1 : Écrivez une fonction python qui prend en entrée une chaîne de caractères en ASCII 8 bits et renvoie l'entier associé.

Indication : On pourra utiliser la fonction `ord(c)` qui renvoie la valeur ASCII du caractères `c`.

Notons h la fonction précédente qui associe à une chaine S un entier i. Introduisant la fonction g qui compose la fonction h avec une congruence modulo 255.

$$h : S \to i = h(S)$$

$$g : S \to j = h(S) \bmod 255$$

Pour tout mot S, si un mot T est obtenu à partir de S par permutation de ses lettres (mêmes lettres, même nombre d'occurrences, mais l'ordre est quelconque) alors $g(S) = g(T)$.

Q2 : En utilisant pyhton, vérifier cette assertion avec les chaînes `"chien"` et `"niche"`.

Q3 : Expliquer l'origine phénomène. Quelle précaution faut-il prendre lors de son usage dans un dictionnaire ?

2 Comptage rapide par dictionnaire ou liste de compteurs

Le fichier fourni `ladisparition.txt` contient un extrait d'un roman de George Perec. On cherche à compter le nombre d'apparitions de chaque lettre utilisée.

Q1 : Dans un premier temps, on cherche à compter le nombre d'apparitiosn des cinq premières lettres de l'alphabet, en minuscule. Pour cela, on initialise un dictionnaire : `d={'a':0,'b':0,'c':0,'d':0,'e':0}`

En n'utilisant qu'une boucle, rédiger un programme qui compte le nombre des apparitions de ces lettres pour le texte fourni (on peut incrémenter un compteur en utilisant la syntaxe : `d['a'] = d['a']+1`).

Q2 : On cherche maintenant à déterminer le nombre d'apparitions de chaque caractère du texte (majuscule et minuscule séparées) à l'aide d'un dictionnaire de compteurs.

Proposer un programme qui réalise cet objectif. Le dictionnaire sera initialement vide (`d={}`) ; les compteurs seront initialisés, si nécessaire, dans la boucle, en utilisant la syntaxe : `d['a'] = 0`.

Q3 : On cherche enfin le nombres d'apparitions de chaque lettre minuscule sous la forme d'une liste (de 26 compteurs). On pourra commencer par initialiser une liste contenant 26 zéros que l'on créera par compréhension. Et, en utilisant la convention ASCII, on pourra associer à la lettre `'a'` le compteur d'index 0, et à la lettre `'z'` le compteur d'index 25. La résolution ne doit utiliser qu'une seule boucle.

3 Manipulation de dictionnaire

3.1 Présentation

Les étudiants de PSI* ont pris l'habitude de se rendre en salle d'études pour travailler. Chaque étudiant ne peut aller dans cette salle qu'une seule fois par jour, mais peut y rester autant de temps qu'il le désire. La période de travail d'un étudiant est donc caractérisée par deux quantités : son instant d'arrivée et son instant de départ. Dans ce sujet un instant de la journée est représenté par un unique entier égal au nombre de secondes s'étant écoulées depuis 00h00. Par exemple, si un étudiant arrive en salle d'études à l'instant 45296, cela signifie qu'il est arrivé à 12h34m56s. En effet :

$$45296 = 12 \times 3600 + 34 \times 60 + 56.$$

Les instants d'arrivées et de départs des différents étudiants sont stockés dans un dictionnaire dont les clés sont des chaînes de caractères (les noms des étudiants) et dont les valeurs sont des couples d'entiers (`deb`, `fin`) où `deb` est l'instant d'arrivée de l'étudiant et `fin` est son instant de départ. Un tel dictionnaire sera appelé un planigramme. Par exemple le dictionnaire `plani0` est un planigramme :

```
plani0 = {
'Aymane' : (47704 , 51650) , # Arrivee : 13 h15m04s --- Depart : 14 h20m50s
'Sophie' : (50400 , 54000) , # Arrivee : 14 h00m00s --- Depart : 15 h00m00s
'Emile' : (48104 , 49545) , # Arrivee : 13 h21m44s --- Depart : 13 h45m45s
'Garance': (48104 , 52050) , # Arrivee : 13 h21m44s --- Depart : 14 h27m30s
'Adrien' : (48104 , 49500) , # Arrivee : 13 h21m44s --- Depart : 13 h45m00s
'Thomas' : (49200 , 50400) # Arrivee : 13 h40m00s --- Depart : 14 h00m00s
}
```

Q1 : Donnez et justifier la seconde ligne affichée par le programme suivant :

```
for c in plani0 :
    deb , fin = plani0[c]
    print (c, "est arrivé(e) àl'instant ",deb , "et parti(e) àl'instant ",fin )
```

Q2 : Qu'affiche le programme suivant ? (justifier)

```
print ('Adrien' in plani0 )
print (48104 in plani0 )
print ((48104 , 49500) in plani0 )
```

Q3 : Qu'affiche le programme suivant ? (justifier)

```
print ([c for c in plani0 if plani0 [c][0] == plani0 ['Emile'][0]])
```

Les horaires présents dans le planigramme étant renseignés manuellement, on souhaite vérifier la cohérence des heures d'arrivées et de départs. Pour que le planigramme soit cohérent, il faut que :

- les instants d'arrivées et de départs soient positifs ou nuls ;
- l'instant d'arrivée de chaque étudiant soit inférieur ou égal à son instant de départ ;
- chaque étudiant soit parti de la salle d'études au plus tard à 23h59m59s.

Q4 : Écrire une fonction `est_coherent(plani: dict) -> bool` qui renvoie si le planigramme donné en entrée est cohérent.

Q5 : Quelle est la complexité de votre fonction `est_coherent` ? Justifier.

Q6 : On souhaite tester la fonction `est_coherent`. Expliquer pourquoi un jeu de tests doit contenir au moins 5 tests.

La manière dont sont représentées les instants dans un planigramme n'est pas très lisible pour un humain. On souhaite donc construire un dictionnaire dans lequel chaque instant inst est donné par un triplet d'entiers `(h,m,s)` où `h` est le nombre d'heures, `m` le nombre de minutes et `s` le nombre de secondes qui se sont écoulées depuis minuit. Par exemple, à partir du planigramme `plani0`, on obtient le dictionnaire :

```
{"Aymane": ((13 ,15 , 4) ,(14 ,20 ,50)), "Sophie" : ((14 , 0, 0) ,(15, 0, 0)),
 "Emile" : ((13 ,21 ,44) ,(13 ,45 ,45)), "Garance": ((13 ,21 ,44) ,(14 ,27 ,30)),
 "Adrien": ((13 ,21 ,44) ,(13 ,45 , 0)), "Thomas" : ((13 ,40 , 0) ,(14, 0, 0))}
```

Q7 : Écrire une fonction `convertir(plani: dict) -> dict` qui renvoie un dictionnaire dans lequel les instants sont donnés sous le format heures-minutes-secondes, comme dans l'exemple ci-dessus. Votre fonction ne doit pas modifier le dictionnaire `plani` donné en entrée. La fonction devra être de complexité linéaire en `len(plani)`.

3.2 Étudiant ayant le plus travaillé

On souhaite déterminer le ou les étudiants ayant passé le plus de temps dans la salle d'études. Par exemple, pour le planigramme plani0, les étudiants ayant étudié le plus longtemps sont Aymane et Garance puisqu'ils sont restés 3946 secondes en salle d'études, contre 3600 secondes pour Sophie, 1441 secondes pour Émile, 1396 secondes pour Adrien et 1200 secondes pour Thomas.

Q1 : Écrire une fonction `plus_long(plani:dict) -> list` qui prend en entrée un planigramme et renvoie la liste des étudiants qui ont passé le plus de temps en salle d'études. Par exemple, à partir du dictionnaire `plani0`, votre fonction doit renvoyer `['Aymane', 'Garance']` (l'ordre des éléments de la liste n'a pas d'importance). Si le dictionnaire est vide, votre fonction renverra la liste vide.

Les étudiants se sont rendus compte que le temps passé en salle d'études n'est pas le seul critère pour déterminer si le travail a été fructueux. Catherine est persuadée que l'entraide entre étudiants est un facteur prédominant dans la réussite en PSI. Deux étudiants ont pu s'entraider s'ils ont passé au moins une seconde ensemble dans la salle d'études. On appelle **score d'entraide** d'un étudiant le nombre d'autres étudiants avec lesquels il a pu s'entraider. Par exemple, avec le planigramme `plani0`, le score d'entraide de Sophie est 2 car elle était dans la salle d'études en même temps que Aymane et Garance.

Q2 : Écrire une fonction `score_entraide(plani: dict) -> dict` qui renvoie un dictionnaire `d_SE` ayant les mêmes clés que la planigramme `plani`, et tel que pour toute

clé `c`, l'entier `d_SE[c]` est le score de l'étudiant `c`. Le planigramme donné en entrée ne doit pas être modifié par votre fonction. Par exemple pour le planigramme `plani0`, le dictionnaire `d_SE` vaut :

```
{"Aymane": 5, "Sophie": 2, "Emile": 4, "Garance": 5, "Adrien": 4, "Thomas": 4}
```

3.3 Occupation de la salle d'études

Afin d'optimiser l'utilisation des salles du lycée, Sébastien a besoin de savoir à quels moments la salle d'études est occupée. Il désire donc avoir à sa disposition la liste de tous les intervalles de temps pendant lesquels au moins un étudiant se trouve dans cette salle. Pour pouvoir lire cette liste facilement, il souhaite que :

- Les intervalles de la liste soient tous disjoints deux à deux.
- Les intervalles soient triées par ordre chronologique.

Par exemple, la liste d'intervalles pour le planigramme `plani0` est `[(47704,54000)]`. Voici deux autres exemples de planigrammes et les listes d'intervalles associées :

```
plani1={'E1':(7,8),'E2':(1,4),'E3':(10,13),'E4':(2,3),'E5':(11,12),'E6':(3,5)}
L1=[(1,5),(7,8),(10,13)]

plani2={'E1':(5,6),'E2':(1,2),'E3':(3,4),'E4':(2,3)}
L2=[(1,4),(5,6)]
```

Q1 : Écrire une fonction `occupation(plani: dict[str, int*int]) -> list[int*int]` qui prend en entrée un planigramme et renvoie la liste d'intervalles qui lui correspond.