Informatique

TP n°4 - Dictionnaires

PSI - Lycée Rabelais

1 Manipulation de dictionnaires

1.1 Instructions élémentaires

On considère le dictionnaire `carre` regroupant les carrés des nombres entiers allant de 1 à 5. Chaque clé sera un entier, la valeur associée étant le caractère de ce nombre.

Q1. Écrire l'ensemble d'instruction nécessaire à la création du dictionnaire `carre`.

```
carre={1:1,2:4,3:9,4:16,5:25}
```

On peut également envisagé une création par compréhension :

```
carre = {x:x**2 for x in range(1,6)}
```

ou par insertion :

```
carre = {}
for x in range(1,6) :
carre[x]=x**2
```

Q2. Écrire sur sa feuille les résultats issus des instructions suivantes :

- **a)** `carre[2]` : *renvoie 4, valeur associée à la clé 2*
- **b)** `carre[7]` : *Key error car 7 n'est pas une clé de carre*
- **c)** `carre[6]=36` : *insertion de `(6:36)` en fin de dictionnaire*
- **d)** `carre[0]=0` : *insertion de `0:0` en fin de dictionnaire. On remarquera que le tri des clés ne se fait pas automatiquement.*
- **e)** `carre[2]=5` : *modification de l'élément de clé `2` : sa valeur est désormais `(2:5)`*
- **f)** `del(carre[6])` : *suppression de l'élément de clé 6, i.e. `(6:36)`*

Q3. Quelle instruction écrire pour vérifier que la clé `1` est présente dans le dictionnaire `carre` ? Tester cette instruction et écrire sur sa feuille ce que renvoie `Python`.

Il suffit d'écrire `carre[1]` dans la fenêtre de commande. Le résultat renvoyé est `True`. En tapant par exemple `carre[25]`, le résultat renvoyé est `False` car 25 n'est pas une clé de `carre`.

On rappelle que `d.keys()` permet de visualiser l'ensemble des clés d'un dictionnaire `d`. `d.items()` permet de visualiser l'ensemble des éléments (clé,valeur) du dictionnaire `d`.

Q4. Écrire l'ensemble des instructions nécessaires pour récupérer dans une liste `cle` l'ensemble des clés présentent dans `carre`. Faire de même avec une liste `valeurs` contenant l'ensemble des valeurs de `carre`.

```
cle=[]
# on recupere chaque cle du dictionnaire d par d.keys()
for c in carre.keys() :
    cle.append(c)

 # l'instruction carre.items() renvoie l'ensemble des tuples (cle,valeur)
   du dictionnaire carre. Il faut selectionner le deuxieme element de
   chaque tuple pour obtenir seulement les valeurs

valeurs=[]
for v in carre.items():
    valeurs.append(v[1])
```

Q5. Comment créer un dictionnaire `carre2` contenant cette fois les n premiers entiers ? Tester votre code avec $n = 5$, puis $n = 100$.

```
n=100
# par comprehension :
carre2 = {x:x**2 for x in range(1,n+1)}
```

```
n=100
# ou par insertion :
carre2 = {}
for x in range(1,n+1) :
carre2[x]=x**2
```

1.2 Fonction `min_max`

Écrire une fonction `min_max` qui prend en paramètre 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])` renvoie `{"min":1, "max":9}`. On utilisera autant que possible le formalisme des dictionnaires.

```
def min_max2(L):
    d={"min":L[0],"max":L[0]}
    for i in range(0,len(L)):
        if(L[i]<d["min"]):
            d["min"]=L[i]
        elif(L[i]>d["max"]):
            d["max"]=L[i]
    return d
```

2 Caractère équiprobable de `random.shuffle`

La fonction `random.shuffle` de la bibliothèque `random` est utilisée pour mélanger "aléatoirement" les élements d'une liste `L` passée en argument. `random.shuffle` modifie sur place la liste `L` : pour observer son effet, il faut demander à afficher la liste `L` après modification.

Q1. Tester plusieurs fois `random.shuffle` sur la liste `L=[1,2,3,4,5]`. On importera au préalable la bibliothèque `random`.

Réponse :

```
import random
L = [1,2,3,4,5]
random.shuffle(L)
print(L)
```

On cherche à vérifier que les permutations issues de l'utilisation de `random.shuffle` sont équiprobables. Nous allons pour cela considérer une liste de longueur p, contenant les entiers compris entre 0 et $p-1$, et appliquer n fois la fonction `random.shuffle`. À chaque application, la nouvelle permutation obtenue est comptée.

On définit pour cela la fonction `test(n,p)` qui réalise n permutations de la liste L de longueur p. Afin de compter le nombre d'occurence de chaque permutation, la fonction `test(n,p)` doit renvoyer un dictionnaire contenant :

- pour clés, les permutations de L converties en tuples ;
- pour valeurs, le nombre de fois que chaque permutation apparaît.

Q2. Pourquoi ne peut-on pas utiliser les listes permutées comme clés du dictionnaire renvoyé par `test(n,p)` ?

Réponse : Les clés d'un dictionnaire doivent être immuables : c'est le cas du type `tuple`, pas du type `list`.

Q3. Que devrait renvoyer `test(10000,2)` si la fonction `random.shuffle` mélange de manière équiprobable les éléments du tuple ?

Réponse : `(0,1):5000, (1,0):5000`

Q4. Écrire la fonction `test(n,p)`. On pourra utiliser l'instruction `tuple(L)` pour convertir la liste `L` en tuple.

```
def test(n, p):
    dico = {}
    L=[] # on peut aussi creer L par l'instruction list(range(p))
    # on cree la liste des entiers de 0 a p - 1
    for j in range(0,p):
        L.append(j)
    for i in range(n):
        #on melange la liste (qui est mutable)
        random.shuffle(L)
        # on transforme la liste melangee en p-uplet, immuable
        t=tuple(L)
```

```
        # on utilise t comme cle dans le dictionnaire
        if t in dico: #
            dico[t] = dico[t] + 1
        else:
            dico[t] = 1
    return dico
```

3 Comparaison de tailles de liste

On utilise un dictionnaire pour comparer des listes de nombres qui sont tous du même type, soit du type `int` soit du type `float`.

Q1. Écrire une fonction `occurences` qui prend en paramètre une liste de nombres et renvoie un dictionnaire dont les clés sont les différents nombres de la liste avec pour valeur le nombre d'occurences de chaque nombre. Par exemple, `occurences([3,5,-2,3,3,-2]` vaut `{-2:2,3:3,5:1}`

Q2. Écrire une fonction `taille` qui prend en paramètre un dictionnaire obtenu comme ci-dessus et renvoie la longueur de la liste qui a été considérée.

Q3. Écrire une fonction `compare` qui prend en paramètres deux listes de nombres de même longueur et renvoie `True` si les deux listes contiennent les mêmes nombres, pas nécessairement dans le même ordre, et `False` sinon, en utilisant la fonction `occurences`. Quelle est la complexité en temps de la fonction `compare` ?

```
#### Question 1 ####

def occurences(liste):
    d={}
    for element in liste:
        if element in d:
            d[element]=d[element]+1
        else:
            d[element]=1
    return d

%#### Question 2 ####

def taille(dico):
    somme=0
    for cle in dico:
        somme=somme+dico[cle]
    return somme

assert(taille(occurences(L))==len(L)) # si la condition est verifiee,
   aucune erreur n'est renvoyee.


#### Question 3 ####
```

```
24
25 def compare(liste1,liste2):
26     assert(len(liste1)==len(liste2)) # verifie que les deux listes ont
   meme longueur
27     dico1=occurences(liste1)
28     dico2=occurences(liste2)
29     for c in dico1:
30         if (c not in dico2 or dico1[c]!=dico2[c]):
31             return False
32         else:
33             return True
```

4 Techniques de stockage des coefficients d'un 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 `hash`, qui s'apparente à la fonction identité ici : pour tout $n \in \mathbb{N}$, $h(n) = n$. 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
...	...	...

Q1. Donner le tableau correspondant au stockage du polynôme $2x^5 - 3x^{34} + 4x^{105}$.

indice 0	-1	-1
indice 1	105	4
indice 2	34	-3
indice 3	-1	-1
indice 4	-1	-1
indice 5	5	2
indice 6	-1	-1
indice 7	-1	-1

Q2. Quel est le problème avec, par exemple, le polynôme $8 - 5x^2 + 3x^{10}$?

Réponse : Le hachage donne la même valeur d'indice pour l'ordre 2 et l'ordre 10 : h(2)%8 = h(10)%8 = 2. L'élément (2,-5) est donc écrasé du tableau, remplacé par (10,3) : il y a collision.

Q3. 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. Que devient le nouveau tableau pour le polynôme $8-5x^2+3x^{10}$? Donner un exemple de polynôme de degré minimum qui génère une collision pour chaque monôme excepté le premier.

Réponse : Le hachage donne la même valeur d'indice pour l'ordre 2 et l'ordre 10 : h(2)%8 = h(10)%8 = 2. L'élément (2,-5) est stocké à l'indice 2, et l'élément (10,3) est stocké à l'indice 3, qui correspond à une cellule vide.

On prend un polynôme de la forme $ax^p + bx^{p+8} + cx^{p+1} + dx^{p+2} + ex^{p+3}$. *Le hachage pour le premier monôme donne l'indice i=p%8 (libre). Pour le deuxième monôme, on obtient (p+8)%8 qui donne aussi i, donc l'indice d'une case occupée. L'élément sera donc stocké en* $i+1$. *Le troisième monôme est stocké en (p+2)%8 qui donne* $i+1$, *case désormais occupée. Il faudra donc le stocker en* $i+2$, *etc. Pour illustrer, on peut par exemple poser* $p=0$ *et a, b, c, d et e égaux à* 1 *:* $1+x^8+x+x^2+x^3$.

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. 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 l'on trouve le couple (degré, coefficient) dans le premier tableau. On complète le tableau avec des -1.

Des extraits des tableaux pour le polynôme $4x^3-2x^5+4x^9$ sont donnés ci-dessous.

indice 0	3	4
indice 1	5	−2
indice 2	9	4
indice 3	0	0
indice 4	0	0
...	...	...

indice 0	−1
indice 1	2
indice 2	−1
indice 3	0
indice 4	−1
indice 5	1
...	...

Q4. Donner les deux tableaux correspondant au stockage du polynôme $3x^5-x^{18}+7x^{20}$.

On remarque que $5\%8=5$, $18\%8=2$ et $20\%8=4$: aucune collision n'est générée. On obtient donc les tableaux suivants :

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	−1
indice 4	2
indice 5	0
indice 6	−1
indice 7	−1

Q5. Que devient le stockage du polynôme $8-5x^2+3x^{10}$?

On remarque que $0\%8 = 0$, $2\%8 = 2$ et $10\%8 = 2$: une collision est générée. Ce nouveau stockage n'empêche pas les problèmes de collision : il faudra donc décaler l'indice de l'élément $(10, 3)$ de $+1$. On obtient donc les tableaux suivants :

indice 0	0	8
indice 1	2	−5
indice 2	10	3
indice 3	0	0
indice 4	0	0
indice 5	0	0
indice 6	0	0
indice 7	0	0

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

5 Création d'un dictionnaire à partir d'un fichier .csv

On considère le tableau donné par le fichier `pays_langues.csv` présenté dans le cours sur les dictionnaires, et dont les premières lignes sont rappelées ci-dessous :

ISO	Country	Capital	Area(in sq km)	Population	Continent	Currency	Languages	Neighbours
AD	Andorra	Andorra la Vella	468	77006	EU	Euro	ca	ES,FR
AE	United Arab Emirates	Abu Dhabi	82880	9630959	AS	Dirham	ar-AE, fa,en,hi,ur	SA,OM
...	...	...	...	...	...	...	...	...

Compléter la fonction `enregistrer` suivante pour créer l'ensemble du dictionnaire associé à ce fichier. On testera le résultat en affichant les informations relatives aux pays suivants : `Andorra`, `United Arab Emirates` et `France`.

```
# ===================================
# Fonction creation d'un dictionnaire
# ===================================

def enregistrer(data):
# Creation d'un dictionnaire d vide
    d={}
# Ouverture en lecture uniquement du fichier source nomme "data"
    fic=open(data,'r')
    titres=fic.readline().rstrip('\n').split('\t') # on extrait la
   premiere ligne du tableau (titres)
    donnees=fic.readlines() # on extrait les lignes suivantes relatives a
   chaque pays
    for ligne in donnees:
        iso,name,capital,area,pop,continent,currency,language,voisin=ligne
   .rstrip('\n').split('\t')
```

```
        d[name]={titres[0]:iso,titres[2]:capital,titres[3]:float(area),
    titres[4]:int(pop),titres[5]:continent,titres[6]:currency,titres[7]:
    language,titres[8]:voisin}
     fic.close()
     return d

# Instruction pour creer le dictionnaire :
d=enregistrer('pays_langues.csv')
```

6 Compression de texte : LZ78

Cet algorithme de compression de texte, dû à Abraham Lempel et Jacob Ziv (1978), construit et utilise un dictionnaire pour compresser des données. Il les décompresse à l'aide du dictionnaire inversé. L'algorithme est le suivant :

- On se donne un texte, `texte`, à compresser
- On initialise le code avec `code=" "` (où `" "` désigne la chaîne de caractères vide)
- On initialise un dictionnaire `dc={" ":0}` ; les clés en seront des chaînes, les valeurs leurs numéros d'insertion
- On place une fenêtre de longueur un en position `i=0` au dessus de `texte`
- On étend la fenêtre d'observation tant que la chaîne `w` qui y figure est dans le dictionnaire (et tant que l'on n'atteint pas la fin du texte)
- On ajoute au code la chaîne `"p*s|"` avec `p` tel que `dc[w]=p` et `s` le caractère suivant.
- On insère la nouvelle clé `w + s` dans le dictionnaire
- On réitère le procédé à partir de la position i qui suit celle de `s`, tant que...

À l'issue du procédé, `code` et `dc` permettent de reconstituer le texte. En pratique, pour des données d'une certaine taille la place prise par `code` et `dc` est nettement inférieure à celle de `texte`.

```
def compressionLZ78(texte) :
    '''
    Parametres
    ----------
    texte : str, une chaine de caracteres a compresser

    Returns
    -------
    code et dc, le texte compresse et le dictionnaire associe

    '''
    assert "*" not in texte and "|" not in texte
    n=len(texte)
    i,code,dc=0,"",{"":0}

```

```
    def compresser(code,i):
        assert i < len(texte)
        w=""
        while (i<n and (w+texte[i] in dc)):
            w=w+texte[i]
            i=i+1
        if i<n:
            ... # a completer
        return code,i+1

    while i<n:
        code, i = compresser(code,i)
    return code, dc
```

Q1. Avec `texte = "Les tontons flingueurs ont flingué mes moutons."`, l'algorithme construit le code : `"0*L|0*e|0*s|0* |0*t|0*o|0*n|5*o|7*s|4*f|0*l|0*i|7*g|0*u|2*u|0*r|3* |6*n|5* |...  "`

Pour chaque ajout d'un segment `"p*s|"` à `code`, il y a un ajout simultané dans le dictionnaire. Préciser les 12 premiers ajouts.

code	ajout	code	ajout	code	ajout
0*L|	dc['L']=1	0*o|	dc['o']=6	0*l|	dc['l']=11
0*e|	dc['e']=2	0*n|	dc['n']=7	0*i|	dc['i']=12
0*s|	dc['s']=3	5*o|	dc['to']=8	7*g|	dc['ng']=13
0* |	dc[' ']= 4	7*s|	dc['ns']=9	0*u|	dc['u']=14
0*t|	dc['t']=5	4*f|	dc[' f']= 10	...	...

Attention ! Dans `dc[' f']= 10`, il faut bien lire "espace + f"

Q2. Dans le schéma d'implantation de l'algorithme qui est proposé, compléter la sous-fonction `compresser(code,i)` qui gère la fenêtre glissante.

```
def compressionLZ78(texte) :
    '''
    Parametres :    texte : str, une chaine de caracteres a compresser
    Returns :    code et dc, le texte compresse et le dictionnaire associe
    '''
    assert "*" not in texte and "|" not in texte
    n=len(texte)
    i,code,dc=0,"",{"":0}
    def compresser(code,i):
        assert i < len(texte)
        w=""
        while (i<n and (w+texte[i] in dc)):
            w=w+texte[i]
            i=i+1
        if i<n:
```

```
            # w est dans le dictionnaire
            # w1 = w+texte[i] est ajoute
            p = len(dc)
            c=texte[i]
            w1=w+c
            dc[w1]=p
            code=code+"%s*%s|"%(dc[w],c)
        return code,i+1
    while i<n:
        code, i = compresser(code,i)
    return code, dc

texte1="Les tontons flingueurs ont flingue mes moutons."
res=compressionLZ78(texte1)
```

Q3. Écrire une fonction `decompressionLZ78(code,dc)` qui prend en arguments un code de compression, `code`, et un dictionnaire `dc` renvoyés par `compressionLZ78(texte)` et reconstitue et renvoie le texte d'origine.

```
def decompressionLZ78(code,dc) :
    '''
    Parametres
    ----------
    code : str, code obtenu par LZ78
    dc : dictionnaire associe

    Returns
    -------
    Le texte decompresse
    '''
    ### inversion du dictionnaire
    dci={v:k for k,v in dc.items()}
    texte =""
    if code[-1]=='|':
        code=code[0:-1]
    C=code.split('|')
    for element in C:
        E=element.split('*')
        w=dci[int(E[0].strip())]
        s=E[1]
        texte=texte+w+s
    return texte

string=decompressionLZ78(res[0],res[1])
```