TP n° 12 – Dictionnaires

On donne en Annexe quelques fonctions et méthodes liées aux dictionnaires python.

I Décompte des occurrences de lettres dans un mot ou un texte

D'après une idée de Serge Bays.

Exercice 1. On considère un mot, représenté en python par une chaîne de caractères (objet python de type `str`). Consulter l'Annexe du TP 03 pour un rappel sur les fonctions et méthodes sur les chaînes de caractères.
Dans un script python, écrire une fonction `comptage(mot)` qui prend comme argument une chaîne de caractères et renvoie un dictionnaire dont chaque clé est un caractère différent du mot et l'élément associé à la clé est le nombre de fois que ce caractère apparaît dans le mot.

Solution 1.

```
def comptage(mot):
    '''renvoie un dictionnaire contenant le nombre
    d'occurences de chaque caractère du mot'''
    occurrences = {}
    for caractere in mot:
        if caractere in occurrences:
            occurrences[caractere] = occurrences[caractere] + 1
        else:
            occurrences[caractere]=1
    return occurrences
```

Exercice 2. Tester la fonction précédente sur le mot de votre choix et en afficher le dictionnaire des occurrences des caractères. Puis la tester sur un deuxième mot contenant exactement les mêmes caractères que le premier mot, avec le même nombre d'occurrences pour chaque caractère, mais avec les caractères dans un ordre différent.
Tester si les dictionnaires des occurrences des caractères des deux des mots sont identiques. Conclusion ?

Solution 2.

```
print(comptage('tatayoyo'))
print(comptage('yoayttoa'))
print(comptage('tatayoyo') == comptage('yyaottoa'))
```

À l'affichage d'un dictionnaire, python affiche les couples clé/élément dans l'ordre dans lequel ils ont été ajoutés au dictionnaire. Mais pour python deux dictionnaires sont identiques s'ils sont constitués exactement des mêmes couples clé/élément, quel que soit l'ordre dans lequel on les a ajoutés. C'est une différence avec les listes, qui ne sont égales que si tous les éléments sont dans le même ordre.

Exercice 3. On considère maintenant un texte, qui est aussi représenté en python par une chaîne de caractères. La différence entre un mot et un texte provient du fait que dans un texte il y a des caractères supplémentaires : la ponctuation (point, virgule, point-virgule, deux points, point d'exclamation, point d'interrogation), des espaces et des caractères spéciaux dits d'échappement dont le plus courant est celui qui signifie le passage à la ligne (`\n`).
Définir une fonction `statistiques(texte)` qui prend comme argument une chaîne de caractères et renvoie un dictionnaire dont chaque clé est un caractère différent du texte, à l'exclusion des ponctuations, espaces et caractère d'échappement, et l'élément associé à la clé est le nombre de fois que ce caractère apparaît dans le texte.

Solution 3.

```
def statistiques(texte):
    '''renvoie un dictionnaire contenant le nombre
```

```
    d'occurences de chaque caractère du texte
    à l'exclusion des ponctuations, espaces et saut de ligne'''
    occurrences = {}
    for caractere in texte:
        if caractere not in " ,;.:!?\n":
            if caractere in occurrences:
                occurrences[caractere] = occurrences[caractere] + 1
            else:
                occurrences[caractere]=1
    return occurrences
```

Exercice 4. Copier dans votre répertoire personnel les fichiers hugo.txt et perec.txt présents dans le répertoire Ressources. Avec python : écrire les instructions pour ouvrir hugo.txt, récupérer le contenu du fichier sous la forme d'une chaîne de caractères unique, fermer le fichier. Appliquer la fonction `statistiques` au contenu de ce fichier et afficher à l'écran les statistiques des occurrences des caractères dans le texte.

Solution 4.

```
file=open('hugo.txt','r')
hugo=file.read()
file.close()
print(statistiques(hugo))
```

Exercice 5. On souhaite trier les données pour lister les caractères dans l'ordre décroissant de leurs occurrences dans un texte. La méthode `.items()` permet d'accéder à l'ensemble des couples clé/élément sous la forme d'un objet python dédié de type `dict_items`. Afficher l'ensemble de ces couples pour le contenu du fichier hugo.txt.
Dans le but de trier les données, il peut être intéressant de les manipuler sous forme de liste, ce qui permet de réviser les algorithmes de tri vus au TP 07. Pour transformer l'objet précédent en liste, il suffit d'utiliser la fonction `list`. Exemple :

```
dico = {'a' : 5, 'b' : 12} # un dictionnaire
couples = dico.items()     # objet dict_items contenant les couples clé/élément
liste = list(dico.items()) # une liste des couples clé/élément
```

Écrire une fonction `tri_occurrences` qui prend en argument un dictionnaire des nombres d'occurrences des caractères d'un texte et renvoie une liste des couples clé/élément triés dans l'ordre décroissant de leurs occurrences dans un texte.
L'afficher pour le contenu du fichier hugo.txt puis pour celui du fichier perec.txt.
Quelle est la particularité du second texte ?

Solution 5.

```
print(statistiques(hugo).items())

#Tri par insertion
def tri_occurrences(dictionnaire):
    '''Tri par insertion des données du dictionnaire
    après obtention de la liste des couples clé/élément'''
    # création de la liste des couples clé/élément
    liste=list(dictionnaire.items())
    # Parcour de 1 à la taille de la liste
    for i in range(1, len(liste)):
        # On mémorise la position initiale de l'élément
        k = liste[i]
        # On décale cet élément vers la gauche tant que
        # l'élément précédent est plus grand que lui
        j = i-1
```

```
        while j >= 0 and k[1] > liste[j][1] :
                liste[j + 1] = liste[j]
                j -= 1
        liste[j + 1] = k
    return liste

def tri_occurrences2(dictionnaire):
    '''Tri à bulles des données du dictionnaire
    après obtention de la liste des couples clé/élément'''
    # création de la liste des couples clé/élément
    liste=list(dictionnaire.items())
    # Parcours de 1 à la taille de la liste
    for i in range(1, len(liste)):
        # Parcours des éléments précédents
        for j in range(0, len(liste)-i):
            # On permutte les deux éléments successifs
            if liste[j][1] < liste[j+1][1] :
                liste[j], liste[j+1] = liste[j+1], liste[j]
    return(liste)

print(tri_occurrences(statistiques(hugo)))
file=open('perec.txt','r')
perec=file.read()
file.close()
print(tri_occurrences(statistiques(perec)))
```

Le texte contenu dans perec.txt ne contient aucun « e » qui est pourtant la lettre la plus utilisée dans la langue française. Il s'agit d'un extrait de *La Disparition* de Georges Pérec, roman intégralement écrit sans la lettre e.

II Anagrammes

D'après une idée de Mickaël Péchaud.

Une anagramme d'un mot est obtenue par permutation des lettres de ce mot. Dans une anagramme, on ignore la casse (majuscule/minuscule), les symboles diacritiques (accents et cédilles), tirets et autres apostrophes.
Ainsi, les mots suivants sont des anagrammes :

- « nectar » et « carnet » ;
- « sortie » et « rôties » ;
- « niche » et « Chine ».

Exercice 6. Pour éliminer tous les symboles diacritiques, tirets, apostrophes et passer toutes les lettres en minuscules, on fournit le code d'une fonction :

```
import unidecode
def enleve_diacritiques_majuscules_tirets(mot):
    '''renvoie le mot en minuscules, sans les diacritiques,
    tirets, apostrophes, etc.'''
    return unidecode.unidecode(mot).lower().replace('-','')
```

Écrire une fonction `sont_anagrammes` qui prend en argument deux mots, teste si les deux mots sont des anagrammes l'un de l'autre et renvoie le résultat du test, c'est-à-dire un booléen (`True` ou `False`).

Solution 6.

```
import unidecode
def enleve_diacritiques_majuscules_tirets(mot):
    '''renvoie le mot s en minuscule, et sans les accents et cedilles'''
    return unidecode.unidecode(mot).lower().replace('-','')

def sont_anagrammes(mot1, mot2):
    '''teste si deux mots sont anagrammes l'un de l'autre après
    avoir éliminé les diacritiques, accents, etc.'''
    return comptage(enleve_diacritiques_majuscules_tirets(mot1)) == comptage(enleve_diacritiques_ma

print(sont_anagrammes('tatayoyo', 'yôyötâtà'))
print(sont_anagrammes('tatayoyo', 'chaudchocolat'))
```

Exercice 7. On dispose dans le répertoire Ressources d'une liste de tous les mots français (selon GNU aspell [1]), contenu dans un fichier nommé listemotsfrancais.txt. Ce fichier contient 629 570 mots : **ne pas tenter de lire directement ce fichier, sa taille est trop importante.** La structure du contenu de ce fichier est très simple : un mot par ligne et chaque ligne est terminée par `\n`.
Copier dans votre répertoire personnel le fichier listemotsfrancais.txt présent dans le répertoire Ressources.
Avec python : écrire les instructions pour ouvrir ce fichier, récupérer le contenu du fichier, fermer le fichier.
En vous appuyant sur le TP 04, créer une liste contenant la liste des mots en français. **Ne pas tenter d'afficher directement cette liste en entier, sa taille est trop importante.**
On pourra vérifier que le code fait ce qui est attendu en affichant les 10 premiers éléments de la liste, ce qui devrait donner :
`['ah', 'an', 'ans', 'as', 'en', 'ha', 'AAAI', 'enharnacher', 'enhardi', 'enhardis']`

Solution 7.

```
fichier=open('listemotsfrancais.txt','r')
contenu=fichier.read()
fichier.close()
listemotsfrancais = contenu.split('\n')
print(listemotsfrancais[0:10])
```

1. `http://aspell.net/`.

Exercice 8. Écrire une fonction `anagrammes` qui prend en argument un mot et renvoie un dictionnaire dont la clé est le mot et l'élément est la liste de toutes les anagrammes françaises de ce mot.
Attention, la liste de tous les mots français contient 629 570 mots : sur les ordinateurs de la salle de travaux pratiques, une recherche prend plus de trente secondes. Pour tester le code, il est donc recommandé d'utiliser une beaucoup plus petite liste, par exemple une liste des 50 000 premiers mots français selon GNU aspell. Avec une telle liste, l'instruction `print(anagrammes(chien))` devrait renvoyer :
`{'chien': ['chiné']}`
Une fois que le code est correct, afficher toutes les anagrammes des mots de votre choix en utilisant la liste complète des mots français selon GNU aspell.

Solution 8.

```
def anagrammes(mot):
    '''renvoie un dictionnaire contenant un unique couple
    dont la clé est mot et l'élément est la listes des anagrammes
    de mot'''
    anagrammes = {}
    liste_anagrammes = []
    motbis=enleve_diacritiques_majuscules_tirets(mot)
    for motcandidat in listemotsfrancais:
        motcandidatbis = enleve_diacritiques_majuscules_tirets(motcandidat)
        if mot != motcandidat:
            if sont_anagrammes(motbis,motcandidatbis):
                liste_anagrammes.append(motcandidat)
    anagrammes[mot] = liste_anagrammes
    return anagrammes
print(anagrammes('chien'))
```

Annexe

Fonctions et méthodes sur les dictionnaires

Opération	Exemple
Création d'un dictionnaire vide	`d={}`
Création d'un dictionnaire avec un couple `cle/element`	`d={cle : element}`
Test d'appartenance d'une clé	`cle in d`
Ajout d'un couple `cle/element`	`d[cle]=element`
Élément correspondant à une clé	`d[cle]`
Test d'égalité de deux dictionnaires	`d1 == d2`
Ensemble des clés	`d.keys()`
Ensemble des éléments	`d.items()`