# TD 4 : ENSEMBLES ET DICTIONNAIRES

## Exercice 1. Compter les lettres d'un texte

(1) Ecrivez une fonction `compterCaracteres(s)` qui reçoit une chaîne de caractères `s` et détermine et renvoie le nombre d'apparitions dans s de chaque caractère possible. On supposera que les caractères sont codés sur un octet (il y en a donc 256 possibles).

Pour commencer il faut trouver une bonne représentation de cette réponse. Afin de ne faire qu'une seule exploration du texte (qui peut être très long) utilisez une liste formée de 256 compteurs, un pour chaque caractère possible, avec cette convention : le $i$-eme élément de la liste exprime le nombre d'apparitions du caractère dont le code interne est $i$.

Exemple de programme utilisant votre fonction :

```
cpt = compterCaracteres('''maitre corbeau sur un arbre perche
tenait en son bec un fromage''')
print 'tous les caracteres:', cpt
print 'lettres minuscules uniquement:'
for i in range(ord('a'), ord('z') + 1):
print chr(i), '-->', cpt[i]
Affichage obtenu :
tous les caracteres: [0, 0, ... quelques valeurs sont non nulles , 0, 0]
lettres minuscules uniquement:
a -> 5
b -> 3
...
z -> 0
```

A savoir :

— La fonction `ord` prend un caractère et renvoie l'entier qui est sont codage interne. Par exemple, `ord('A')` vaut `65`.

— La fonction `chr` fait le contraire : elle prend un entier et renvoie le caractère dont il est le code. Par exemple, `chr(65)` vaut `'A'`.

— On peut créer une liste comportant `n` éléments dont la valeur est `v` de la manière suivante : `l = [ v ] * n`. Par exemple `l = [0] * 10`

*Correction:*

```
def compterCaracteres(s):
    result = [ 0 ] * 256
    for x in s:
        result[ord(x)] = result[ord(x)] + 1
    return result
```

## Exercice 2. Nombre d'occurrences de chaque mot d'un texte

(1) Ecrivez une fonction `quelsMots(texte)` qui prend pour argument une chaîne de caractères et rend l'ensemble des mots du texte qu'elle représente.

A savoir :

— La méthode `split` permet de décomposer une chaîne de caractères en une liste de mots. Si `s` est une chaîne de caractères, `s.split()` retourne la liste des mots qui composent `s`. Par défaut, `split` considère que les mots sont séparés par des espaces ou des retours à la ligne. Vous pouvez aussi spécifier vos propres séparateurs à l'aide du paramètre `sep`.

Exemple d'utilisation :

```
print quelsMots('bbb ddd ccc aaa bbb ddd ccc bbb ddd bbb')
Affichage obtenu : ['aaa', 'bbb', 'ccc', 'ddd']
```

*Correction:*

```
def quelsMots(s):
    return list(set(s.split()))
```

(2) Ecrivez une fonction `compterMots(texte)` qui prend pour argument une chaîne de caractère et rend les mots du texte, chacun associé au nombre de fois qu'il apparaît dans le texte.

Indication. Utilisez un dictionnaire.

Exemple d'utilisation :

```
print compterMots('bbb ddd ccc aaa bbb ddd ccc bbb ddd bbb')
Affichage obtenu : {'aaa': 1, 'bbb': 4, 'ccc': 2, 'ddd': 3}
```

*Correction:*

```
def compterMots(s):
    listeMots = s.split()
    cpt = { }
    for mot in listeMots:
        if mot in cpt:
            cpt[mot] = cpt[mot] + 1
        else:
            cpt[mot] = 1
    return cpt
```

(3) Ecrivez une fonction `afficherMots(texte)` qui prend pour argument une chaîne de caractère et affiche les mots du texte, chacun associé au nombre de fois qu'il apparaît dans le texte, rangés par ordre décroissant du nombre d'apparitions.

Exemple d'utilisation :

```
afficherMots('bbb ddd ccc aaa bbb ddd ccc bbb ddd bbb')
Affichage obtenu :
bbb --> 4
ddd --> 3
ccc --> 2
aaa --> 1
```

*Rappels.* Si `dico` est un dictionnaire, alors `dico.items()` renvoie la liste des paires qui le constituent.

D'autre part, la fonction `sorted(l)`, où `l` est une liste, construit une nouvelle liste triée dont les éléments sont ceux de `l`. Les éléments de `l` peuvent eux-même être des objets complexes (liste, tuples, ...), auquel cas, la manière de les comparer n'est pas très claire. Il est possible, dans ce cas, de spécifier quel élément de l'objet complexe servira à effectuer le tri, à l'aide du paramètre de la fonction `sorted` nommé `key`. Ce dernier est une fonction qui prend un élément de `liste` et retourne un élément de ce dernier.

Si, par exemple, `liste` est une liste de tuples comportant une chaîne de caractères et un nombre et que l'on désire la trier selon la chaîne de caractères, on pourra utiliser la fonction suivante :

```
def clef(t):
    return t[0]
```

La fonction `sorted` est alors appelée de la manière suivante : `sorted(l, key=clef)`

*Note :* Une autre expression de la fonction `clef` est `lambda t: t[0]`.

*Correction:*

```
def clef(t):
    return t[0]

def afficherMots(s):
    cpt = compterMots(s)
    paires = sorted(cpt.items(), key=clef)
    for x in paires:
        print(x)

def afficherMots2(s):
    cpt = compterMots(s)
    paires = sorted(cpt.items(), key=lambda t : t[0])
    for x in paires:
        print(x)
```

**Exercice 3. Carnet d'adresses**

(1) Ecrivez une application pour gérer un carnet d'adresses. Celui-ci se compose de fiches, chacune formée d'un certain nombre de champs, comme ceci :

| nom du champ | valeur du champ |
|---|---|
| nom | Gaston Lagaffe |
| âge | 28 |
| fonction | responsable du rangement |
| telephone | 01 02 03 04 |
| e-mail | glagaffe@gmail.com |
| adresse | Avenue Aimé de Mesmaeker s/n - Bruxelles (Be) |

Pour la création des fiches, on se donnera la fonction très élémentaire suivante :

```
def nouvelleFiche(vNom, vAge, vFonction, vTel, vMail, vAdresse):
    return { 'nom': vNom, 'age': vAge, 'fonction': vFonction, 'tel': vTel, \
        'mail': vMail, 'adresse': vAdresse }
```

Ecrivez

— une fonction qui saisit au clavier les composants d'une fiche, crée la fiche et l'insère dans le carnet d'adresses,
— une fonction qui retrouve et affiche la fiche d'une personne à partir du nom de celle-ci,
— une fonction pour retrouver et afficher les fiches – il peut y en avoir plusieurs – qui contiennent un couple (nom du champ, valeur du champ) donné,
— une fonction pour retrouver et afficher les fiches qui satisfont à un critère quelconque donné par l'utilisateur,
— un « programme principal » avec un menu pour choisir une opération parmi les précédentes, genre :

```
Menu:
 C          Créer une nouvelle fiche
 A   nom    Afficher la fiche correspondant à un nom donné
 V   champ  valeur fiches ayant une Valeur donnée pour un champ donné
 S   critère fiches qui Satisfont un critère donné
 Q          Quitter ce programme
Votre commande ?
```

En définitive, dans cet exercice on utilise les dictionnaires de deux manières : d'une part, chaque fiche est un dictionnaire associant les noms des champs à leurs valeurs. D'autre part, le carnet d'adresse sera lui-même un dictionnaire associant chaque nom à une fiche.

*Correction:*

```
carnetAdresses = {}
```

```
def nouvelleFiche(vNom, vAge, vFonction, vTel, vMail, vAdresse):
    return { 'nom': vNom, 'age': vAge, 'fonction': vFonction, 'tel': vTel, \
            'mail': vMail, 'adresse': vAdresse }

def saisir():
    nom = input('Nom ? ')
    if nom in carnetAdresses:
        print('Désolé, il y a déjà une fiche pour %s' % nom)
        return
    age = int(input('Age ? '))
    fon = input('Fonction ? ')
    tel = input('Téléphone ? ')
    mel = input('Adresse électronique ? ')
    adr = input('Adresse postale ? ')
    carnetAdresses[nom] = nouvelleFiche(nom, age, fon, tel, mel, adr)

def afficher(fiche):
    print("nom       %s" % fiche['nom'])
    print("age       %d" % fiche['age'])
    print("fonction  %s" % fiche['fonction'])
    print("téléphone %s" % fiche['tel'])
    print("e-mail    %s" % fiche['mail'])
    print("adresse   %s" % fiche['adresse'])
    print("-----------------------------------------------------")

def chercherParNom(nom):
    if nom not in carnetAdresses:
        print("Désolé, il n'y a pas de fiche pour %s" % nom)
        return
    afficher(carnetAdresses[nom])

def chercherParChampValeur(champ, valeur):
    for fiche in carnetAdresses.values():
        if champ in fiche and str(fiche[champ]) == valeur:
            afficher(fiche)

def chercherParCritere(critere):
    for fiche in carnetAdresses.values():
        if critere(fiche):
            afficher(fiche)

if __name__ == '__main__':
    while True:

        print('Menu')
        print('    C                 créer une nouvelle fiche' )
        print('    A nom             fiche correspondant au nom donné')
        print('    V champ valeur    fiches où le champ donné a la valeur donnée')
        print('    S critère         fiches satisfaisant le critère')
        print('    Q                 quitter ce programme')
        reponse = input('Votre commande? ').strip()
        if reponse == '':
            continue
        arguments = reponse[1:].strip()
        commande = reponse[0]
        if commande == 'C':
            saisir()
```

```
        elif commande == 'A':
            chercherParNom(arguments)
        elif commande == 'V':
            morceaux = arguments.split()
            chercherParChampValeur(morceaux[0].strip(), morceaux[1].strip())
        elif commande == 'S':
            critere = eval(arguments)
            chercherParCritere(critere)
        elif commande == 'Q':
            break
        else:
            print('Commande incomprise: %s' % commande)

    print('Au revoir!')
```

**Exercice 4. Enregistrer une structure dans un fichier**

(1) Reprenez le programme de l'exercice précédent et ajoutez-lui le nécessaire pour que
- en quittant le programme, le carnet d'adresses soit automatiquement enregistré dans un fichier,
- au lancement du programme, le carnet d'adresses soit automatiquement chargé depuis ce fichier.

*Correction:*

```
def sauvegarder(nomFichier):
    fichier = file(nomFichier, 'w')
    for fiche in carnetAdresses.values():
        print >> fichier, fiche['nom']
        print >> fichier, fiche['age']
        print >> fichier, fiche['fonction']
        print >> fichier, fiche['tel']
        print >> fichier, fiche['mail']
        print >> fichier, fiche['adresse']
    print >> fichier, '%FIN%'
    fichier.close()

def restaurer(nomFichier):
    fichier = file(nomFichier, 'r')
    while True:
        nom = fichier.readline()[:-1]
        if nom == '%FIN%':
            break
        age = int(fichier.readline()[:-1])
        fonction = fichier.readline()[:-1]
        tel = fichier.readline()[:-1]
        mail = fichier.readline()[:-1]
        adresse = fichier.readline()[:-1]
        carnetAdresses[nom] = nouvelleFiche(nom, age, fonction, tel, mail, adresse)
    fichier.close()
```

**Exercice 5. Analyse syntaxique**

(1) Une pile est une structure de données possédant une opération empiler pour insérer un élément, une opération dépiler pour extraire un élément et obéissant à la convention suivante : le dépilement fournit l'élément le plus récemment empilé (un peu comme dans une

pile d'assiette où il vaut mieux prendre en priorité l'assiette qui se trouve au sommet de la pile).

Pour rappeler cela on dit qu'une pile est une structure LIFO (pour last int first out) ou, en français, « dernier entré premier sorti ».

En Python on peut implémenter une pile en se donnant une liste `p` et en s'astreignant à ne la modifier qu'à l'aide des fonctions `p.append(x)` (ajout de `x` à la fin de `p`) pour empiler et `x = p.pop()` (extraction du dernier élément de `p`) pour dépiler.

Application. On souhaite écrire une fonction `bienForme(s)` prenant pour argument un texte représentant une sorte de formule qui contient un certain nombre de caractères `(, ), [, ], <, >,{, },}` etc., comme :

`(exp0230a<a1[0], a2min(-25C), max(60C), a3>, exp0230b<>)`

et répondant à la question : ce texte est-il bien parenthésé ? Ou encore : les caractères « ouvrants » `(, [, {, <` s'apparient-ils bien avec les « fermants » correspondants, respectivement `), ], }, >`.

*Indication.* La méthode suggérée consiste à se donner une pile et à examiner chaque caractère du texte en parcourant ce dernier de la gauche vers la droite. Chaque fois qu'on rencontre un caractère ouvrant, on l'empile. Lorsqu'on rencontre un caractère fermant, on vérifie qu'il correspond bien au caractère ouvrant le plus récemment rencontré. Si ce n'est pas le cas, on signale une erreur et on interrompt l'exécution du programme.

En fait il faut détecter et signaler trois sortes d'erreur :

— un caractère fermant ne correspond pas au dernier caractère ouvrant rencontré,
— il y a trop caractères fermants,
— il y a trop de caractères ouvrants.

*Truc.* Vous aurez besoin de la correspondance entre les caractères ouvrants `(, [, {, <` et les caractères fermants correspondants `), ], }, >`. Pour cela, employez un dictionnaire, initialisée au début du programme.

*Correction:*

```
def bienForme(texte):
    oppose = { '(': ')', '[': ']', '<': '>', '{': '}' }
    ouvrants = oppose.keys()
    fermants = oppose.values()
    pile = []

    for car in texte:
        if car in ouvrants:
            pile.append(car)
        elif car in fermants:
            if len(pile) == 0:
                return 'trop de caractères fermants'
            else:
                cbis = oppose[pile.pop()]
                if cbis != car:
                    return 'caractères dépareillés'
    if len(pile) > 0:
        return 'trop de caractères ouvrants'
    return 'ok'
```