Entrées, sorties

I. Lecture/écriture dans un fichier texte

I.1. `with` et `open`

a) Exercice d'introduction

Le fichier « fr_FR.dic » distribué avec OpenOffice contient un dictionnaire de la langue française.
Le code suivant ouvre le fichier en lecture,

```
with open('dictionnaire.dic',encoding='utf-8') as f:
    lignes=f.readlines()
mots=[ligne.split('/')[0] for ligne in lignes]
mots=[mot.split('\n')[0] for mot in mots][1:]
```

Que contient la variable `mots` ? Pourquoi ?
Créer une fonction `anagramme(m)` qui trouve tous les anagrammes d'un mot `m` donné.
Variation : modifier votre fonction `anagramme(m)` de sorte à ce que les caractères accentués soient considérés comme identiques.
Plus compliqué : obtenir la liste de toutes les listes d'anagrammes du dictionnaire.

b) Ouverture/fermeture d'un fichier : syntaxe

Syntaxe : open

Pour ouvrir un fichier ***texte***, on utilise la fonction `open` du noyau Python. La fonction renvoie un ***objet*** qui représente le fichier au sein du code Python : ***c'est cet objet qui permettra de lire, d'écrire ou d'ajouter*** du texte dans le fichier. La syntaxe générale de la fonction `open` est la suivante :
`objet=open(nom,mode)`
où `nom` désigne le nom du fichier (éventuellement précédé de son chemin d'accès si le fichier n'est pas dans le même répertoire que le code Python exécuté)
et où `mode` est un caractère désignant la façon dont le fichier est ouvert :

- si `mode` vaut `'r'` (pour ***read***), on pourra lire dans le fichier ;
- si `mode` vaut `'w'` (pour ***write***), le fichier est effacé. On pourra ensuite écrire dans le fichier.
- Si `mode` vaut `'a'` (pour ***append***), le fichier est ouvert. On pourra ajouter du texte à la fin du le fichier.
- Si `mode` vaut `'+'`, le fichier est mis-à-jour s'il existait, créé sinon. Souvent associé à `'w'` et `'a'`.

Important ! Blocage de l'accès à un fichier

Nous avons vu au chapitre 1 section I.3. que le système d'exploitation assure entre autre la gestion des fichiers. Python fait directement appel au système d'exploitation pour l'ouverture d'un fichier et la fonction `open` n'est en fait qu'une interface permettant au programmeur de ne pas avoir à se soucier de la façon dont les divers systèmes d'exploitation gèrent les accès aux fichiers. Par exemple, la façon dont un texte est encodé sous forme binaire dépend du système d'exploitation : `utf-8` pour Linux et MacOS, `latin1` pour les versions de Windows antérieures à la version

7 mais `utf-8` à nouveau pour Windows 8...
Tout ceci est géré par la fonction `open` et le programmeur n'a donc pas à s'en préoccuper, ce qui permet aux codes écrits d'être indépendants du système d'exploitation sur lequel ils sont exécutés : on dit que le code est ***indépendant de la plateforme***.
Cependant, certains problèmes doivent absolument être pris en considération lors de l'accès à un fichier. Essentiellement, il faut comprendre que :

- pour ouvrir un fichier, encore faut-il avoir les ***droits d'accès requis par le système d'exploitation*** ;
- lorsqu'un fichier est ouvert par un programme, ***il ne peut plus l'être par un autre pour éviter que deux programmes apportent des modifications différentes et simultanées à un même fichier*** ce qui provoquerait ou bien des plantages machines, ou bien des erreurs sur les modifications attendues dans le fichier.

Si un fichier est ouvert par Python, il ne peut plus l'être par un autre programme (y compris un second code Python qu'on tenterait d'exécuter simultanément avec le premier).
C'est la raison pour laquelle ***un fichier ouvert doit absolument être fermé après utilisation !***
Deux alternatives sont possibles :

Syntaxe : close

C'est ***la méthode*** (au sens informatique du terme) permettant de fermer un fichier, c'est-à-dire de prévenir le système d'exploitation que le fichier ne sera plus utilisé par le code Python. Si `objet` désigne un fichier ouvert, la fermeture du fichier s'accomplit à l'aide de la syntaxe :
`objet.close()`

Une alternative (préférable) est l'emploi de l'instruction `with` :

Syntaxe : with

L'instruction `with` permet de fermer ***automatiquement*** le fichier dès que les tâches de lecture/écriture/modification sont accomplies. Parce que la fermeture s'accomplit automatiquement, cette instruction évite beaucoup d'erreurs qui sont - qui plus est - sous irrattrapables : un fichier ouvert qui n'aurait pas été refermé ne pourra plus être ni lu ni modifié jusqu'au redémarrage de la machine.
Pour ouvrir un fichier on utilisera donc de préférence la syntaxe :

```
with open(nom,mode) as objet:
    BlocDeCodeIndente # le fichier est ouvert
PoursuiteDuCodeNonIndente # le fichier a été automatiquement ferme
                          # objet est inaccessible
```

I.2. Lecture/écriture

Une fois le fichier ouvert, on dispose essentiellement de deux méthodes pour lire/écrire dans le fichiers :

Syntaxe : Méthode `readlines`

`fichier.readlines()` permet de lire l'intégralité du fichier et renvoie son contenu dans une chaîne de caractères.
Les lignes sont séparées par le caractère spécial `'\n '`. Exemple :

```
with open("monfichier.txt",'r') as f:
    contenu = f.readlines()
```

Syntaxe : Méthode `writelines`

`fichier.writelines(chaine)` permet d'écrire le contenu de `chaine` dans le fichier. Pour passer à la ligne, on utilise le caractère spécial `'\n '`. Exemple :

```
chaine = 'un fichier sur\ndeux lignes'
with open("monfichier.txt",'w+') as f:
    f.readlines(chaine)
```

I.3. Le format `csv` et l'écriture d'un fichier texte

Syntaxe : Format `csv`

Le format `csv` est un format générique pour l'écriture de listes ou de tableaux dans un fichier texte.
On utilise pour cela deux séparateurs :
`'\n '` est utilisé comme séparateur des lignes du tableau tandis que
`','` est utilisé comme séparateur pour les colonnes à l'intérieur d'une même ligne.
La première ligne du fichier peut - éventuellement - contenir des informations générales sur le fichier, comme par exemple le titre des colonnes présentes dans le fichier.

Calculer, pour chaque lettre de l'alphabet, sa fréquence dans le dictionnaire Openoffice.
Écrire un fichier texte, nommé `"frequences.csv"` dont la première ligne est constituée des lettres de l'alphabet, séparées par des virgules, et dont la seconde ligne donne les fréquences d'apparition de chacune de ces lettres (séparées par des virgules).

II. Représentations graphiques

Comme nous l'avons vu au chapitre 4, aux TDs ?? et ?? et dans le devoir à la maison, les modules `numpy` et `matplotlib.pyplot` peuvent être utilisés pour faire divers types de représentations graphiques.

II.1. `numpy`

Ce module apporte essentiellement deux grandes améliorations au noyau Python :

1) un grand nombre de fonctions et constantes mathématiques ;
2) des objets de type `numpy.ndarray` permettant de créer des tableaux de nombres. Ces tableaux ont un fonctionnement similaire quoique différent des listes avec quelques avantages et quelques inconvénients :
   - les tableaux `numpy` doivent posséder des éléments de type identique ;
   - ce sont des tableaux d'ordre n ou $n\times p$ ou $n\times p\times q$ (où $n,p,q\in\mathbb{N}^*$) etc... Notamment, il est impossible de créer des tableaux triangulaires comme par exemple un triangle de Pascal ;