- Enchaînemots

Pour aider le rédacteur d'un texte, il est possible de lui proposer pour chaque mot qu'il écrit un candidat pour le mot suivant. Avec cet objectif, une structure de données peut associer à chaque mot les mots qui sont susceptibles de le suivre, avec une « note » permettant de déterminer le plus vraisemblable.

Pour simplifier, dans ce travail, un « mot » désigne une succession contiguë de caractères alphabétiques — il est clair que cette définition est insuffisante dans le cas général, par exemple en français *aujourd'hui* et *tire-laine* devraient être comptés pour un seul mot, mais *jusqu'alors* et *était-on* pour deux.

L'utilisation de la structure de données en question passe par deux étapes :

— une phase d'apprentissage consistant à parcourir un ou des textes et compter les occurrences de chaque couple de mots,

— une phase d'utilisation proposant pour un mot donné le mot qui le suit le plus souvent.

1 - Structure

a) Définir une fonction `estlettreFr(cara)` déterminant si `cara` est un caractère alphabétique français :

une lettre de l'alphabet ou un symbole *e dans l'o* ou *e dans l'a*, avec éventuellement un signe diacritique (cédille, accent), majuscule ou minuscule.

b) Définir une classe Python `couplesmots` permettant de mémoriser pour chaque couple de mots une valeur entière déterminant combien de fois ce couple a déjà été rencontré. Discuter des structures utilisables, de leurs avantages.

c) Définir pour cette classe (en les testant au fur et à mesure) les méthodes suivantes :

un constructeur prenant en paramètre une fonction (comme `estlettreFr`) déterminant si un caractère peut appartenir à un mot ;

`afficher()` permettant d'afficher tous les couples stockés et les nombres d'occurrences correspondant ;

`ajoutercouple(prem,sec)` permettant d'ajouter une occurrence d'un couple de mots, retournant le nombre d'occurences obtenu (au moins 1) ;

`ajouterchaine(texte)` permettant d'ajouter tous les couples de mots de la chaîne de caractères `texte` et retournant le dernier mot rencontré dans `texte` ;

`ajoutersuitechaine(motprec,texte)` effectuant le même travail en supposant que `motprec` précède le premier mot et forme donc le premier couple (il est peut-être possible de la programmer avant la précédente :-) ;

`ajouterfichier(nomfic)` effectuant le même travail que `ajouterchaine` mais avec le texte contenu dans le fichier texte dont le nom est fourni (à ne pas charger intégralement en mémoire!).

d) Définir (dans n'importe quel ordre) pour cette classe (en les testant au fur et à mesure) les méthodes suivantes :

`meilleursuivant(prem)` donnant le mot qui suit le plus souvent `prem` ;

`unsuivant(prem)` donnant aléatoirement un des mots qui suivent `prem` ;

`lessuivants(prem)` donnant une liste, triée par ordre décroissant de nombres d'occurrence, des mots pouvant suivre `prem` ;

`lessuivocc(prem)` donnant une liste, triée de même, des couples (mot, nombre d'occurrences) ;

`phrase(prem,n)` fournissant une chaîne d'au maximum `n` mots (séparés par un espace) commençant par le mot `prem` et continuant avec à chaque étape le mot le plus vraisemblable après le précédent.

e) Écrire une nouvelle version (`phraseunique`) de la méthode `phrase` pour qu'elle n'utilise pas deux fois le même mot en utilisant si nécessaire un suivant moins fréquent.

f) Écrire une nouvelle version (`phraseproba`) de la méthode `phrase` pour qu'elle choisisse chaque suivant aléatoirement mais en privilégiant les plus fréquents.

2 - Application

Tester la structure avec les fichiers `cyrano.txt, salammbo.txt ou notredame.txt` (en français) et aussi avec les fichier `doriangray.txt ou littlewomen.txt` (en anglais, changer la fonction à la création), .

3 - Étude

Comparer le temps d'exécution des méthodes pour des programmations différentes (pas forcément dues à la même personne :-).