Pour ce TP, veuillez télécharger le fichier `TP1ListeMotsFrancais.txt` qui se trouve sur le CDP de la classe. Mettez le dans le même dossier que votre fichier Python actuel. Ce fichier contient tous les mots français [1]. Les lignes suivantes permettent d'ouvrir le fichier et de créer une variable `Liste` qui va contenir tous les mots.

```
fichier=open("TP1ListeMotsFrancais.txt", "r")
Liste=fichier.readlines()
```

Attention sous Pyzo, il faudra faire `run file as script` dans `run` ou `Démarrer le script` au lieu de F5 la première fois. Sinon Pyzo ne cherche pas dans le bon dossier le fichier .txt.

Meta-dictionnaire

Exercice 1 ($\star$). 1. Combien y a-t-il de mots dans la liste ? Attribuer à une variable, notée `mot`, le 50-ième mot de la liste. Taper mot dans la console pour afficher chacun des caractères, à la fin, vous devriez voir un \n (caractère spécial pour le retour à la ligne).

2. Modifier la liste `Liste` pour enlever le [2] caractère spécial `"\n"` à la fin de chaque mot de la liste `Liste`. On utilisera le slicing/trançage.
3. Redéfinir la variable mot comme le 50-ième mot de la liste `Liste` (qui a changé à cause de la question précédente).
4. Tester les commandes `sorted(mot))` puis `"".join(sorted(mot))`.

On rappelle qu'un mot est une anagramme d'un autre mot si on peut passer de l'un à l'autre en échangeant les lettres. Par exemple *chien* et *niche*. Remarquez alors que `"".join(sorted("chien"))` est égale à `"".join(sorted("niche"))`

5. 𝄞Trouver les mots qui ont le plus d'anagrammes. Pour cela, en parcourant toute la liste, créer un dictionnaire dont les clés seront les chaînes de caratèreres de mots triés par ordre alphabétique et dont les valeurs seront le nombre de mots qui contiennent exactement les mêmes lettres mais dans un ordre différents. Par exemple, `D["cehin"]` vaudra 3, en effet, `chien`, `niche` et `chine` sont les seuls trois mots du dictionnaire qui contiennent une et une seule fois les lettres c, e, h, i et n.
6. Trouver la valeur maximale dans le dictionnaire précédemment crée ainsi que la ou les clés qui ont cette valeur maximale. De plus, afficher tous les mots de `Liste` qui ont le nombre maximal d'anagrammes.

Créer des fonctions de hachage

On souhaite maintenant créer une fonction de hachage très simple [3].

Exercice 2 ($\star$). 1. La commande `ord("z")` renvoie un entier naturel correspondant à un numéro pour la lettre `"z"` et de même pour toutes les lettres. Remarquez que `ord("x")-97` donnera un numéro de la lettre `x` entre 0 et 25. Créer une fonction `f(mot)` qui à un `mot` va renvoyer :

$$\sum_{i=0}^{n-1} (\text{ord}(\text{mot}\,[i]) - 97) \times 26^i$$

où n est la longueur du mot. Vérifier que `f("cpge")` vaut 74752 et que `f("anticonstitutionnellement")` vaut 17764056427617870564620886194998959 8 La fonction `f` ferait-elle une bonne fonction de hachage ?

2. Créer une fonction de hachage `h1(mot)` qui a un mot renvoie le reste de la division euclidienne de `f(mot)` par $n = 10^3$.
3. Ainsi, si on créait des dictionnaires avec cette fonction de hachage, quelle serait la taille de la table de hachage ? Si on hache tous les mots du dictionnaire, y aurait-il à votre avis des collisions ? Combien y aurait-il en moyenne de mots dans chaque alvéole ?
4. Créer une fonction `Bilan` qui va construire une liste `L` à n éléments, tels que `L[i]` compte le nombre de mots de la liste `Liste` tels que `h1(mot)`=i. C'est-à-dire que `L[i]` va compter combien d'éléments serait dans l'alvéole `i` si on utiliser la fonction `h1` comme fonction de hachage et les mots de la langue française comme clé.
5. Utiliser cette fonction bilan pour vérifier s'il y a eu des collisions, trouver le nombre maximum de mots qui seraient dans la même alvéole, le nombre moyen d'occupation de chaque alvéole, ainsi que l'écart-type.

1. Avec toutes les variantes : tous les accords possibles des adjectifs et des noms, toutes les conjugaisons des verbes. Les accents et les majuscules ont été retirés.
2. Le et non pas les, car contrairement aux apparences, `"\n"` compte pour un caractère et non deux caractères.
3. Bien plus simple que celle utilisé par Python.

On rappelle que l'écart-type des nombres x_1, x_2, …, x_n est donné par :

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-m)^2} \quad \text{où} \quad m=\frac{1}{n}\sum_{i=1}^{n}x_i$$

6. On rappelle le principe du paradoxe des anniversaires : si on a un groupe de p personnes, la probabilité qu'au moins deux d'entre-elles aient le même jour d'anniversaire (en négligeant le cas du 29 février) vaut : [4]

$$1-\prod_{i=0}^{p-1}\frac{365-i}{365}$$

En particulier, pour 23 personnes, il y a une chance sur deux pour que deux d'entre elles soient nées le même jour. Ici, on peut voir les collisions comme des gens qui sont nés le même jour, ainsi si on a k clés différentes et prises aléatoirement (ce qui n'est pas le cas en pratique) et que l'on a une table de hachage avec n alvéoles, alors la probabilité qu'il y ait (au moins) une collision est :

$$p(n,k)=1-\prod_{i=0}^{k-1}\frac{n-i}{n}$$

Que vaut $p(n,k)$ lorsque $n=10^3$ (correspondant à la fonction de hachage `h1`) et avec `k=len(Liste)`) ? Représenter sur un graphique `p(n,len(Liste))` en fonction de n. Quel n faut-il prendre pour que la probabilité qu'il y ait une collision soit inférieure à $1/2$. Est-ce réaliste de prendre une table de hachage de cette taille ?

7. On a donc une fonction de hachage `h1` qui renvoie un nombre positif pour une chaîne de caractère constitué de lettres minuscules et sans accent. Adaptez `h1` en une fonction `h2` qui puisse contenir les majuscales, les accents et toutes sortes de caractères usuelles.

8. Créer une fonction de hachage `h3` qui hache les nombres et les chaînes de caractères, pour les chaînes de caractères, il suffit de reprendre ce que fait `h2`, pour les entiers relatifs, il suffit de renvoyer le résultat modulo n. Pour un flottant x, on pourra les multiplier par une puissance de 10 de façon à avoir un entier $p=10^m x$ et renvoyer le résultat de p modulo n.
Pour les tuples constitué de nombres ou de chaînes de caractères, on pourra renvoyer la somme des hachage de chacun des éléments du tuple. Mais attention, un tuple peut être constitué lui même de tuples, qui sont eux-même des tuples etc. Il faudra donc procéder par récursivité dans ce cas.
Si `x` est une variable, on pourra utiliser `isinstance(x,tuple)` qui renvoie `True` si `x` est un tuple. Idem avec `int, float, dict, str` etc.

9. Compter le nombre de multiplications nécessaires pour la fonction `f(mot)`.

10. Récrire la fonction `h1` qui va d'abord calculer `f(mot)` sans utiliser la fonction `f` mais en utilisant l'algorithme de Horner dans lequel on calcule directement modulo n. Puis compter le nombre de multiplications nécessaires.

Exercice 3 (𝄞⋆⋆)**.** Dans l'exercice 1, on a utilisé la commande `"".join(sorted(mot))` pour trier les lettres de `mot` en boîte noire. Programmer une fonction qui étant donnée une chaîne de caractères la trie également par ordre alphabétique. On pourra adapter le tri Fusion ou le tri rapide déjà vu, et placer la lettre `x` avant la lettre `y` si `ord(x)` est strictement plus petit que `ord(y)`.

4. En effet, calculons la probabilité qu'elles soient tous nées à des jours différents. Alors, chaque personne a 365 possibilités, ainsi le groupe a 365^p possibilités. Si on cherche à compter les possibilités où les gens naissent à des jours différents, alors la première a 365 possibilités, la deuxième 364, la troisième 363 etc. Au final cela fera $\prod_{i=0}^{p-1}(365-i)$. Comme ici, ce sont des probabilités uniformes $\frac{\prod_{i=0}^{p-1}(365-i)}{365^i}$ représente la probabilité que les personnes soient tous nées des jours différents.