TP1

November 7, 2018

---

1 TP1 : Traitement de fichiers, cryptanalyse, César et de Vigenère

Dans ce TP, on commence par effectuer quelques traitements sur des fichiers et des chaînes de caractères (`string`), puis on reprend la cryptanalyse du chiffrement de César et on la réinvestit dans la cryptanalyse du chiffrement de De Vigenère.

Vous pourrez donc récupérer en partie le travail effectué en TD2 (objectif 20) et en TP1.

1.1 Echauffement

1.1.1 Retour sur le codage ascii

On rappelle l'énoncé de l'exercice 7 du TD2.

L'ASCII est un codage de caractères qui définit 128 codes sur 7 bits. Chaque code correspond à un caractère : chiffres, lettres, symboles mathématiques et de ponctuation.

En Python : - la fonction `ord()` retourne le code ASCII d'un caractère fourni en argument, - la fonction `chr()` retourne le caractère associé à un code donné.

Ecrire un programme qui affiche : - le code ascii des majuscules - la table des 128 codes ASCII.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.2 Traitement

1.2.1 Préparation syntaxique

Dans le but de faire quelques traitements statistiques, et des chiffrements ultérieurs éventuels, écrire la fonction `traitementChaine()` qui : - convertit la chaîne en minuscules (c'est conseillé, ça facilite la suite) - remplace les caractères accentués par les caractères non accentués correspondants ; - élimine les espaces, la ponctuation et les chiffres, sans oublier les retours à la ligne, les guillemets et autres tirets ... ; - enfin, convertit le tout en majuscules et renvoie la chaîne de caractères résultat.

On rappelle ici les méthodes utiles `.lower()`, `.upper()` et `.isalpha()`, pour le traitement des chaînes de caractères (type `string`) et présentées en cours (chapitre 3 : types composés).

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.2.2 Vérification

Vérifier le bon comportement de votre fonction `traitement Chaine()` sur une chaîne de caractères entrée au clavier.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.2.3 Application à un fichier texte

Vous trouverez dans le dossier le fichier d'entrée `93.txt`, le roman bien connu de Victor Hugo (dont on vous conseille vivement la lecture !).

Effectuer le traitement suivant à l'aide de la fonction `traitementChaine()`. - Ouvrir et lire le contenu de ce fichier d'entrée, - effectuer la préparation syntaxique définie précédemment, - sauvegarder le résultat du traitement dans un nouveau fichier de sortie dénommé `93Out.txt`.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.2.4 Autres vérifications

Vérifier l'effet de votre préparation : - en regardant sous l'éditeur le fichier texte généré,
- ou à l'aide de la commande Unix `wc`

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.2.5 Traitements statistiques

Apparition de lettres : dénombrement, fréquences Dans ce qui suit, on considère une chaîne de caractères arbitraire *mais* en majuscules, sans espace ni ponctuction.

Pour un telle chaîne, écrire les fonctions : - `nb_apparitions()` qui compte le nombre d'apparitions de chacune des lettres majuscules (de 'A' à 'Z') dans cette chaîne ; - `frequences()` qui calcule les fréquences de ces apparitions dans cette chaîne ; - cette fonction permettra de sauvegarder *éventuellement* ces fréquences dans un fichier de sortie ; - dans ce cas, le nom du fichier sera défini comme un paramètre de cette fonction. - Chaque ligne du fichier respectera le format suivant : `A 9.180%`

Rmq. Vous pouvez reprendre et adapter les développements du TP1.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Application à 93, roman de V. Hugo On dispose maintenant d'un fichier `93Out.txt` contenant une chaîne de caractères composée uniquement de majuscules, sans espace ni ponctuation. On peut procéder au calcul de quelques statistiques.

- Appliquer les fonctions précédentes au contenu du fichier `93Out.txt` et
- tracer (avec `matplotlib`) les histogrammes de ces apparitions et des fréquences
- sauvegarder les fréquences dans un fichier `freq93.txt`

- sauvegarder les histogrammes dans des fichiers images `hist_app93.pngget` `hist_freq93.png`

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Apparitions de digrammes Un *digramme* (ou *bigramme*) est un motif de deux lettres.
On considère les digrammes suivants : 'ES','DE','LE','EN','RE','NT','ON','ER','TE','EL','AN','SE','ET','LA','AI','I

- Ecrire une fonction `nb_digrammes()` qui compte le nombre d'apparitions d'un bigramme donné dans une chaîne de caractères arbitraire.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

- Appliquer cette fonction au contenu du fichier `93Out.txt`
  - les nombres d'apparitions de digrammes seront stockés avec un dictionnaire adapté
- et tracer un premier histogramme d'apparition des digrammes définis ci-dessus.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Histogramme ordonné Reprendre les valeurs d'apparitions précédentes et tracer un histogramme ordonnés par nombre d'apparitions décroissant.

Indication : Un dictionnaire python est une structure de donnée non ordonnable.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.2.6 Analyse

Comparer vos statistiques avec les fréquences des lettres et bigrammes données pour la langue française (par exemple : http://www.apprendre-en-ligne.net/crypto/stat/francais.html).
Conclure quant au roman de Victor Hugo.

VOTRE REPONSE ICI

1.3 2 César

1.3.1 Chiffrement de César

Écrire un premier programme qui : - lit une chaîne de caractères arbitraire entrée au clavier ; - effectue le traitement de cette chaîne avec la fonction `traitementChaine()` écrite plus haut ; - définit une fonction `cesar()` prenant en entrée la chaîne de caractères et une clé de chiffrement $k \in \{0, \ldots, 25\}$
- la clé est entrée au clavier, - ou choisie aléatoirement (par exemple si la clé entrée au clavier est négative); - puis détermine la chaîne de caractères chiffrée, - et l'affiche à l'écran.

Rmq. La valeur entière k définit le nombre de décalages du chiffrement de César. k est aussi l'indice (compté à partir de 0) d'une lettre de l'alphabet 'A', 'B', ... , 'Z'.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.3.2 Déchiffrer César avec sa clé

Compléter ce code avec une fonction `decesar()` qui détermine le message clair à partir d'un message chiffré et d'une clé de chiffrement donnée. Appliquer cette fonction au traitement précédent et vérifier la correction des deux étapes.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.3.3 Première application

Une fois ce chiffrement validé, modifier les deux traitements précédents pour que la chaîne de caractères à chiffer soit le texte défini dans le fichier d'entrée `In.txt` et la chaîne chiffrée soit sauvegardée dans le fichier de sortie `Out.txt`.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.4 Test de l'indice de coïncidence

Le test de l'indice de coïncidence est présenté dans les documents joints ('Cryptanalyse.pdf' et 'Coincidence.pdf').
Il y a d'autres ressources pertinentes sur le web, en particulier la page wikipedia

1.4.1 Le test

Écrire une fonction `coincidence()` prenant en entrée une chaîne de caractères et qui implante le test de l'indice de coïncidence.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Tester cette fonction sur les versions chiffrée et déchiffrée de la chaîne de caractères précédente (fichiers `In.txt` et `Out.txt`).

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Première analyse Que conclure sur l'intérêt du test de l'indice de coïncidence pour identifier dans une chaîne de caractères arbitraire l'existence d'un message en français sans ou avec chiffrement de César.

VOTRE REPONSE ICI

1.4.2 Déchiffrer César sans sa clé : analyse fréquentielle

Écrire une fonction `cryptanalyseCesar()` qui prend en entrée un texte chiffré par la fonction `cesar()` précédente (en majuscule sans espace ni ponctuation) et qui renvoie *la clé de chiffrement* de César. Vous disposez pour cela des fréquences théoriques d'apparition des lettres que vous avez déterminées à partir du roman de Victor Hugo.

Appliquer la clé de chiffrement ainsi trouvée pour afficher le message en clair.

Méthode : - Comparer la fréquence d'apparition théorique des lettres avec celle du message chiffré, jusqu'à coïncidence, ce qui donne le décalage. - Une méthode est de calculer la somme des différences entre les deux fréquences pour chaque lettre, et de prendre la somme minima.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.5 4 De Vigenère

On va implanter le chiffrement de de Vigenère en s'aidant des documents ressources mentionnés plus haut.

1.5.1 Implantation du chiffrement

Fractionner Commencer par écrire une fonction `fractionne()` qui : - prend en entrée une chaîne de caractères chiffrée (toujours uniquement en majuscules) ; - découpe le texte chiffré en *pas* sous-chaînes en prenant une lettre tous les *pas* lettres en partant de la $i$ème lettre pour $i \in \{0, \cdots, pas - 1\}$; - renvoie la liste des sous-chaînes de caractères ainsi formées.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Tester cette fonction sur la chaîne de caractères composée des 26 lettres de l'alphabet et plusieurs valeurs de *pas* : 1, 2, 3, 4, 5.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Recoller Ecrire une fonction `recolle()` qui effectue le traitement inverse de celui de `fractionne()` : à partir d'une liste de chaînes de caractères obtenue avec `fractionne()` (pour un pas arbitraire), `recolle()` reconstruit la chaîne d'origine.

Indication : définir un traite exception adapté à la lecture de sous-chaînes de taille différente est utile.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Vérifier cette fonction en l'appliquant aux 5 fractionnements obtenus à la question précedente.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Chiffer Écrire une fonction `chiffrement()` - qui prend en entrées : - un message à chiffrer : une chaîne de caractères en majuscules, sans espace ni ponctuation, - et une clé : une chaîne de caractères, également en majuscule sans espace ni ponctuation ; - `fractionne()` cette chaîne en autant de sous-chaînes que de caractères de l'argument clé, - chiffre avec `cesar()` chaque sous-chaîne de caractères (de la liste renvoyée) avec le décalage correspondant à chaque lettre de la clé ; - `recolle()` les sous-chaînes chiffrées pour former le message complet chiffré ; - renvoie ce chiffrement sous la forme d'une chaîne de caractères en majuscules sans espace ni ponctuation.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Applications Chiffrer : - la chaîne de caractères composée des 26 lettres de l'alphabet avec les clés successives : `Z`, `AZ`, `AZE`, `AZER`, `AZERT`; - le texte du fichier `InVigenere.txt` avec la clé `VIVELEMASTER`.
Afficher les textes ainsi chiffrés.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.5.2 Déchiffrer avec clé

De façon similaire, écrire une fonction `dechiffrement()` qui : - prend en entrées : - un texte chiffré à l'aide du cryptosystème de Vigenère, en majuscules sans espace ni ponctuation, - et une clé : une chaine de caractères également en majuscule sans espace ni ponctuation ; - renvoie le déchiffrement sous la forme d'une chaîne de caractères, en majuscules sans espace ni ponctuation.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Tester ce déchiffrement sur les textes chiffrés précédemment.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.5.3 Cryptanalyse

On va implanter le test de l'indice de coïncidence adapté au cryptosystème de de Vigenère.

Indice de coïncidence On commence en écrivant une fonction `calculIC()` qui - prend en entrée une chaîne de caractères (chiffrée) et un pas ; - `fractionne()` cette chaîne en sous-chaînes selon ce pas, - calcule et renvoie la moyenne des indices de coïncidence de toutes ces sous-chaînes.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Calculer les indices de coïncidences des textes chiffrés précédemment.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Identifier la longueur de la clé Écrire une fonction `tailleCle()` qui : - prend en entrée une chaîne de caractère chiffrée (toute en majuscules) ; - effectue le test de l'indice de coïncidence global pour différentes longueurs de clé (en partant de 1, on ne sait jamais si c'était chiffré avec César...) et s'arrête dès que l'on dépasse un paramètre `seuil` par exemple fixé à 0.065 ; - renvoie la meilleure hypothèse sur la longueur de clé correspondante.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Applications Vérifier la bonne estimation de la longueur de clé en appliquant cette identification au texte chiffré issu du fichier `InVigenere.txt` avec la clé `VIVELALICENCE`.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Essayer puis justifier pourquoi appliquer cette identification échoue si on l'applique aux chiffrements du "message-alphabet" `ABC...YZ` (avec les clés précédemment utilisées).

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

VOTRE REPONSE ICI

Déchiffrer Implanter une fonction `decrypter()` qui : - identifie la longueur de la clé du chiffrement de de Vigenère, - l'utilise pour fractionner le message chiffré en sous-chaînes, - utilise la fonction `cryptanalyseCesar()` pour déchiffrer chaque sous chaîne, - et renvoie le texte clair, et la clé.

Vérifier vos développements en appliquant cette fonction au chiffré de `InVigenere.txt`.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

1.5.4 Application

Pour finir, vous pouvez maintenant chiffrer le roman 93 de Victor Hugo avec des clés de votre choix.

Vérifier les statistiques sur votre texte chiffré en traçant les histogrammes de fréquences d'apparition des lettres sur le texte chiffré.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Faire de même avec les histogrammes de fréquences d'apparition des bigrammes sur le texte chiffré.

```
In [ ]: # ENTRER VOTRE CODE ICI
        raise NotImplementedError()
```

Question bonus : Quelle serait, selon vous, la clé la plus robuste ?
VOTRE REPONSE ICI

```
In [ ]:
```