Traitement de texte

Objectif : rendre l'élève capable de :

- lire et écrire dans un fichier de type `.txt` ou `.csv`
- rechercher un mot dans une chaine de caractères
- rechercher le plus long mot commun à deux chaines de caractères

1 Lire le contenu d'un fichier texte

1.1 Ouvrir et fermer un fichier de type `.txt`

Il est possible de lire et d'écrire dans un fichier texte. Pour cela, il faut créer un pointeur vers le fichier lors de l'appel du fichier en déclarant ce qu'on veut en faire :

- `'r'` ouvrir le fichier uniquement pour le lire
- `'w'` ouvrir le fichier pour y écrire en effaçant le contenu du fichier s'il y en avait. Si le fichier n'existe pas, il sera créé.
- `'r+'` lire dans le fichier et y écrire.
- `'a'` permet d'écrire dans le fichier en plaçant le pointeur à la fin du texte.

Pour plus de modes lecture/écriture, prendre, par exemple, ce tuto.

On ferme le fichier en ajoutant `.close()` au pointeur.

1.2 Lire et écrire dans un fichier de type `.txt`

On considère un fichier ouvert avec le pointeur `fic` (`fic = open(...)`).

Pour lire une ligne, on utilise `fic.readline()` et pour tout lire `fic.read()`. Pour lire 3 caractères `fic.read(3)`.

Il est possible de ranger toutes les lignes d'un fichier dans une liste en tapant `fic.readlines()`. Si on veut n'en lire que 3, on écrit plutôt `fic.readlines(3)`.

Pour écrire dans un fichier qu'on souhaite initialement vide, on utilise la méthode write() associée au pointeur :

```
fic =  open('fichier.txt', 'w')
fic.write('Voici comment mettre du texte dans le fichier.')
fic.close()
```

1.3 Quelques méthodes pratiques

Si `fich` est un pointeur vers un fichier ouvert à la lecture, taper `donnees = fich.readlines()` permet d'obtenir une liste dont les éléments sont les lignes du fichier sous forme de chaines de caractères. Les chaines se terminent par `\n` (sauf parfois la dernière).

Pour enlever les terminaisons en `\n` d'une chaines de caractères `chaine`, on peut utiliser la méthode `rstrip()`. Cela évide de recopier en mémoire toute la chaine. En effet, on pouvait aussi écrire `chaine = chaine[:-1]`.

L'autre méthode utile est `split(sep)` où `sep` est une chaine de caractères permettant de couper la chaine `chaine` à chaque occurrence de `sep` dans `chaine`, lorsqu'on écrit `chaine.split(sep)`. On obtient une liste des sous chaines de `chaine`.

1.4 Préparation du fichier .py

Q - 1 : *A l'aide de la console, se placer dans le répertoire* `~/INFO/1-INTRO/INTRO-3`.

Rappel si le dossier existe, vous pouvez directement taper `cd ~//NFO/1-INTRO/INTRO-3` pour vous y rendre.

Q - 2 : *Créer (et ouvrir par la même occasion) le fichier* `nomp-nomTP.py` *en tapant :*

```
idle3 nomp-INTRO-Tp-3.py
```

2 Mais t'es pas là, mais t'es où ?

Vos pseudos de connexion contenus dans le fichier `pnom.txt` sont cachés dans le texte du fichier `grille.txt`. Cependant, il en manque un. Lequel ?

2.1 Chargement des données

Q - 3 : *Écrire les lignes de codes permettant d'obtenir le contenu du fichier* `grille.txt` *(resp.* `pnom.txt`*) dans une variable* `data_grille` *(resp.* `data_pseudos`*).*

Q - 4 : *A partir de la variable* `data_grille` *écrire un programme permettant d'obtenir :*

- *le nombre de lignes* `nbl` *et le nombre de colonnes* `nbc` *du fichier* `grille.txt`
- *une variable* `texte` *contenant tous les caractères du fichier* `grille.txt` *mis bout à bout sans retour à la ligne*

Q - 5 : *A partir de la variable* `data_pseudos` *écrire un programme permettant d'obtenir dans une liste* `pseudos` *les pseudos de la classe (sans les retours à la ligne).*

2.2 Recherche d'un mot dans une chaine de caractères

Algorithm 1 Teste la présence d'un mot à partir d'une position dans une chaîne de caractères.

entrée: *mot* et *chaine* deux chaînes de caractères, *i*, un indice
résultat: *vrai* si *mot* correspond aux premiers caractères de *chaine* à partir de l'indice *i* ; *faux* sinon
teste_mot(mot, chaine, i)

```
j, m ← 0, taille(mot )
tant que  j < m et mot [j]=chaine[i+j] faire
    j← j + 1
fin tant que
renvoi:  j == m
```

Q - 6 : *Écrire la fonction* `teste_mot` *conformément à l'algorithme* Algo *1.*

Algorithm 2 Recherche d'un mot dans une chaîne de caractères.

entrée: *mot* et *chaine* deux chaînes de caractères
résultat: l'indice de la première occurrence de *mot* dans *chaine* ; sinon -1
recherche_mot(mot, chaine)
i, n, m, trouve← 0, taille(chaine), taille(mot), faux
tant que i≤n-m et **non** trouve **faire**
    trouve← non(teste_mot(mot, chaine, i))
    i←i + 1
fin tant que
si non trouve **alors**
    i ← -1
fin si
renvoi: i

Q - 7 : *Écrire la fonction* `recherche_mot` *conformément à l'algorithme* Algo *2.*

Q - 8 : *Tester la présence de votre pseudo.*

2.3 Mais où sont-ils ?

Q - 9 : *Placer, pour chaque pseudo, dans une liste* `ouestce`*, un tuple contenant le numéro de ligne et le numéro de colonne du premier caractère du pseudo dans le fichier* `grille.txt`*. Afficher le pseudo absent du fichier* `grille.txt`*.*

Q - 10 : *Ouvrir en écriture un fichier* `pseudo.csv` *(et le refermer).*

Q - 11 : *Écrire sur chaque ligne du fichier* `pseudo.csv`*, le pseudo, le numéro de ligne et le numéro de colonne correspondants, séparés par des* `;`*.*

Q - 12 : *Ouvrir le fichier avec un tableur et vérifier quelques résultats.*

3 Génétique

Le but de cette partie est d'utiliser deux fichiers (ici uniquement en lecture) contenant des chaînes de caractères représentant du code génétique et d'effectuer différentes opérations sur ces chaînes pour trouver les séquences d'ADN les plus longues, communes aux souris et aux hommes.
Remarque: sujet initialement proposé par M. Sérier.

3.1 Of Mice and Men

L'ADN est composé de séquences de nucléotides. Chaque nucléotide est constitué de trois éléments liés entre eux : un groupe phosphate lié à un ose, le désoxyribose, lui- même lié à une base azotée.

Il existe quatre bases azotées différentes dans l'ADN : l'adénine (notée A), la thymine (notée T), guanine (notée G) et la cytosine (notée C). Chaque base est fixée à un désoxyribose pour former un nucléoside. Lorsqu'un nucléoside est lié à un ou plusieurs phosphate, on dit qu'il s'agit d'un nucléotide. Dans l'ADN, les nucléotides sont reliés

entre eux selon une certaine séquence grâce à des liaisons impliquant un groupe phosphate, qu'on appelle des liaisons 5'-3' phosphodiester. Pour fabriquer un brin d'ADN, il suffit donc d'enchaîner des nucléotides en les reliant par ce type de liaisons, appelées liaisons fortes. Exemple de brin d'ADN :

ATTGCCGTATGTATTGCGCT

On utilisera les données contenues dans les fichiers `adn_souris.txt` et `adn_humain.txt`.

3.2 Chargement des séquences d'ADN

Q - 13 : *Écrire les lignes de codes permettant d'obtenir le contenu du fichier `adn-humain.txt` (resp. `adn-souris.txt`) dans une variable `donnees_hum` (resp. `donnees_sou`).*

Q - 14 : *A partir des variables `donnees_hum` et `donnees_sou`, écrire un programme créant deux chaînes de caractères (`adn_hum` et `adn_sou`) contenant les deux brins d'ADN à étudier.*

3.3 Fabrications des outils

Q - 15 : *Écrire une fonction `freq_lettre` prenant en arguments un chaine `ch` et une lettre `e` et renvoyant la fréquence d'apparition de `e` dans `ch`, c'est à dire le rapport entre le nombre d'occurrences de la lettre dans la chaîne et le nombre de lettres de la chaîne.*

Q - 16 : *En déduire une fonction `frequence` prenant en arguments une chaine `ch` et une liste `bases` et qui renvoie la liste des fréquences d'apparitions des éléments de `bases` dans `ch`.*

Q - 17 : *Écrire une fonction `prefixes` prenant en arguments un entier `n` et une liste `bases` dont chacun des éléments est un caractère. La fonction `prefixes` renvoie une liste contenant tous les mots de longueur `n` formés avec les éléments de `bases`.*

Remarque: un mot de longueur $n+1$ est formé d'une lettre de `base` suivi d'un mot de longueur n...

Q - 18 : *Écrire une fonction `present` prenant en arguments une liste `motifs` et une chaine de caractère `ch` et renvoie une liste avec les occurrences des éléments de `motifs` dans `ch`.*

Q - 19 : *En déduire une fonction `reccurent` prenant en arguments une liste `bases`, un entier `n` et une chaine de caractère `ch` et renvoie un motif de longueur `n` le plus fréquent dans `ch`.*

Q - 20 : *Déterminer un motif de longueur 6 commun au deux brins d'ADN.*

3.4 Mot le plus long

La méthode développée est assez mauvaise si l'on veut déterminer le plus grand motif commun à deux chaînes.

Une méthode simple est de construire les motifs communs de longueur n + 1 à l'aide des motifs communs aux deux chaînes de longueur n.

Remarque: si le mot AGGATTA::::GGAT est commun aux deux chaînes, GGATTA::::GGAT aussi !

Q - 21 : *Implémenter cette méthode et déterminer un motif de longueur 10 commun aux deux chaînes. Puis un motif de longueur 20.*