TP d'Informatique n°9

Lecture et écriture de fichiers

Python permet de lire et d'écrire dans des fichiers (texte, par exemple). Cette fonctionnalité peut être utilisée pour séparer le code en lui-même des données qu'il utilise (par exemple, un fichier de configuration) ou qu'il crée (par exemple, on peut ainsi sauvegarder un résultat de simulation).

Important : dans tout ce TP, les fichiers texte que l'on souhaitera lire et/ou écrire devront être placés dans le même dossier que votre fichier python. Pour indiquer à Python que c'est dans ce dossier qu'il faut aller regarder, vous devez effectuer la manipulation suivante à chaque fois que vous lancez Pyzo : faites un clique droit dans la console, puis sélectionnez

"change current directory to editor file path".

I Lecture dans un fichier

Ouverture et fermeture d'un fichier

Tout programme destiné à lire (ou écrire) un fichier texte doit toujours commencer par **ouvrir** le fichier texte en question, et se terminer en le **refermant**.

- *fichier* = `open("nomdufichier.txt", mode)` ouvre le fichier texte *nomdufichier*, et le stocke comme objet dans la variable *fichier*. L'argument optionnel *mode* peut valoir `"r"`, `"w"` ou `"a"`, et sert à indiquer si l'on veut ouvrir le fichier en lecture seule (`"r"`) ou si l'on veut écrire dessus (`"w"` ou `"a"` – voir section **II**).
- *fichier*`.close()` referme le fichier ouvert précédemment.

Lorsque vous ouvrez un fichier, n'oubliez jamais de le fermer après utilisation. Si ce n'est pas fait, le fichier resté ouvert continue à occuper de la mémoire – et cela peut causer des erreurs si d'autres applications essayent d'accéder au fichier en question.

Lecture d'un fichier

On suppose que l'on dispose d'une variable `f` contenant un fichier précédemment ouvert en lecture seule, via `f = open("nomdufichier.txt", "r")`.

- `f.read()` lit l'intégralité du contenu du fichier `f`, et le renvoie sous forme d'une seule longue chaîne de caractères.
- `f.readline()` lit la première ligne du fichier `f`, et le renvoie sous forme d'une chaîne de caractères. Un second appel à la méthode `f.readline()` renvoie alors la seconde ligne, et ainsi de suite.
- `f.readlines()` lit l'intégralité du contenu du fichier `f`, et le renvoie sous forme d'une liste dont chaque élément est une ligne de `f` (donnée sous forme d'une chaîne de caractères).

Remarque : Dans la suite de ce TP, on appellera généralement `f` la variable recevant le contenu du fichier à manipuler. Mais bien sûr, on peut lui donner n'importe quel autre nom !

Remarque : La méthode `.readlines()` présente un inconvénient : si le fichier est très gros, elle peut causer un ralentissement du système puisque python mettra d'un coup la totalité du contenu du fichier en mémoire vive. Pour pallier cela, on peut utiliser à la place la méthode `.readline()` : même si elle est moins pratique d'usage, elle ne présente pas ce problème puisqu'elle lit seulement les lignes du fichier une par une.

Exercice 1 *Familiarisation*
Créez un fichier texte `zoo.txt`, en l'enregistrant dans le même dossier que votre fichier Python. Remplissez ce fichier en y écrivant quelques noms d'animaux (un par ligne), par exemple :

```
vache
lapin
pangolin
tardigrade
```

La suite de cet exercice peut être faite directement dans la console.

1. Ouvrez le fichier `zoo` en lecture seule, et appelez-le `f`, en tapant `f = open("zoo.txt","r")`.
2. Tapez (dans la console) `f.read()`, et observez le résultat. Puis tapez à nouveau `f.read()`.

Explications :

- Le retour à la ligne est représenté en Python par un caractère spécial : `\n`. Par exemple, essayez de taper `print("abc\ndef")`. Attention : `\n` compte comme un seul caractère (peut créer des erreurs d'indice si l'on cherche à accéder à un caractère spécifique d'une chaîne de caractères).
- Python ne peut lire qu'**une seule fois** de suite un fichier donné. Le fait d'ouvrir un fichier crée un "curseur de lecture" au début du fichier ; lorsqu'on utilise la commande `read()`, ce curseur balaye le fichier jusqu'à atteindre la fin de celui-ci, mais ne "repart" pas au début. Un second appel à `read()` ne renvoie donc rien ! Si l'on veut recommencer la lecture, il faut fermer le fichier puis le ré-ouvrir [1].

3. "Rembobinez" le fichier `zoo`, par exemple en le fermant puis en le ré-ouvrant. Puis, affichez (joliment, cette fois) son contenu, en tapant `print(f.read())` au lieu de juste `f.read()`.
4. Rembobinez le fichier `zoo`, puis tapez `f.readlines()`, et observez la différence avec le résultat renvoyé par `f.read()`.
5. Rembobinez le fichier `zoo`, puis exécutez plusieurs fois de suite la commande `f.readline()` (sans "s" cette fois !). Observez la différence avec les deux commandes précédentes.
6. Fermez le fichier en tapant `f.close()`.
   Essayez à présent de taper `f.read()`, et observez le résultat.

> Les exercices qui suivent demandent des réponses sous forme de scripts ou de fonctions : n'oubliez pas que tous vos programmes devront toujours commencer par un appel à `open()` et se terminer par un appel à `close()` !

Exercice 2 *Itération sur les lignes*
On veut écrire un script qui affiche successivement chacune des lignes du fichier `zoo`, et ce sans utiliser la commande `read()` (sinon ce serait trop facile).

1. Écrire un script qui accomplit cet objectif en utilisant `readlines()` et une boucle for.
2. Écrire un script qui accomplit cet objectif en utilisant `readline()` et une boucle while.

Remarque : il est aussi possible d'itérer directement sur la variable qui a reçu le contenu du fichier. Par exemple, on peut écrire :

```
f = open("zoo.txt", "r")
for ligne in f:
    print(ligne)
f.close()
```

1. Une autre possibilité est d'utiliser `f.seek(0)`, qui replace le curseur de lecture au début du fichier.

Exercice 3 *Lecture*

1. **a.** Écrire une fonction `lire(nom_fichier)` qui prend en entrée un nom de fichier (une chaîne de caractères au format `"nomdufichier.txt"`) et qui renvoie, en utilisant `readline()` (sans "s"), la liste des lignes du fichier. La tester sur un fichier que vous aurez créé à partir du bloc-notes.
   b. Ajouter un deuxième argument à la fonction, indiquant le nombre maximal de lignes à lire dans le fichier.
   c. Modifier votre fonction pour qu'elle ignore les lignes commençant par `"#"`, et répertorie seulement les autres. La tester après avoir ajouté quelques `"#"` dans votre fichier texte.
2. **a.** Écrire une fonction `compte_lettre(nom_fichier)` qui renvoie le nombre total de `"e"` dans un fichier.
   b. Écrire une fonction `maxE(nom_fichier)` qui renvoie le numéro de la ligne contenant le plus de `"e"` (ou le numéro d'une des lignes en contenant le plus, s'il y en a plusieurs).
   c. Écrire une fonction `rechercheE(nom_fichier)` qui cherche le premier `"e"` apparaissant dans le fichier, et renvoie sa position (on renverra une liste à deux éléments formée du numéro de la ligne où apparaît le premier `"e"`, et de la position du `"e"` dans cette ligne). Si aucun `"e"` n'est présent, la fonction doit renvoyer "Aucune occurrence trouvée".

Quant on lit ou que l'on écrit des informations dans un fichier, tout se fait *via* des **chaînes de caractères**. On peut donc avoir besoin de faire des conversions de type :

- Pour écrire un nombre dans un fichier texte, il faut commencer par le convertir en chaîne de caractères à l'aide de la commande `str()`.
- Au contraire, si l'on lit un nombre dans un fichier texte, le nombre est renvoyé sous forme de chaîne de caractères ; il faut alors lui appliquer `int()` ou `float()` pour le récupérer sous forme numérique.

Exercice 4 *Lecture de nombres*

Écrire une fonction `somme(nom_fichier)` qui prend en entrée un fichier contenant une liste de nombres, à raison d'un nombre par ligne, et qui renvoie la somme de ces nombres.

Exercice 5 *BONUS - si vous avez du temps. Les exercices 6 et 7 sont à faire en priorité.*

Créer un fichier texte dont chaque ligne consiste d'un nombre suivi d'un mot, les deux étant séparés par un seul espace. Par exemple :

```
17 Chlore
6 Carbone
19 Potassium
112 Copernicium
```

1. Écrire une fonction `parseur(ch)` prenant en entrée une chaîne de caractères au format d'une des lignes ci-dessus - c'est-à-dire formée d'un nombre, puis d'un espace `" "`, puis d'un mot - et qui renvoie la liste `[nombre,mot]` contenant ces deux données. Par exemple, `parseur("12 abc")` doit renvoyer `[12,"abc"]`.
   Interdiction d'utiliser la méthode `.split()` !
2. Écrire une fonction `recherche(nom_fichier,mot)` qui prend en entrée un fichier de cette forme, recherche si le mot donné en entrée est présent dans le fichier, et si cela est le cas renvoie le nombre qui l'accompagne.
3. Écrire une fonction `mini(nom_fichier)` qui prend en entrée un fichier de cette forme, détermine le plus petit des nombres présents dans le fichier, et renvoie le mot qui l'accompagne.
4. Écrire une fonction `dico(nom_fichier)` qui prend en entrée un fichier de cette forme, et renvoie le dictionnaire tel que chaque ligne du fichier correspond à une entrée du dictionnaire.

II Écriture dans un fichier

Écriture dans un fichier

On suppose que l'on dispose d'une variable `f` contenant un fichier ouvert précédemment, en mode écriture (*mode* = `"w"` ou `"a"`).
Alors `f.write(`*texte*`)` écrit dans `f` le texte (chaîne de caractères) donné en argument.

- **Mode "w" (pour "write") :** si `f` a été ouvert par `f = open("nomdufichier.txt", "w")`, alors le contenu précédent du fichier est effacé, et remplacé par le texte spécifié.
- **Mode "a" (pour "append") :** si `f` a été ouvert par `f = open("nomdufichier.txt", "a")`, alors le texte spécifié est ajouté en fin de fichier.

Exercice 6 *Familiarisation*

1. Recopiez et exécutez le script suivant :

```
f = open("flore.txt", "w")
f.write("pissenlit")
f.write("coquelicot")
f.write("renoncule")
f.close()
```

Remarque : Si le fichier `flore` n'existait pas au préalable, alors l'appel à `open()` le crée !
Ouvrez le fichier `flore` ainsi créé, et contemplez l'étendue du problème (puis refermez-le).

2. Améliorez le script précédent en y ajoutant les retours à la ligne qui lui font défaut !
3. À la suite du script précédent, ajoutez :

```
f = open("flore.txt", "w")
f.write("tulipe")
f.close()
```

Exécutez-le puis regardez le contenu du fichier `flore`.

4. Remplacez le "w" par "a" (pour "append") dans le second appel de `open()`. Relancez votre programme, puis regardez ce que contient cette fois le fichier `flore`.

Exercice 7

1. a. Écrire une fonction `ecrire(nom_fichier, L)` qui prend en entrée un nom de fichier (une chaîne de caractères au format `"nomdufichier.txt"`) et une liste de nombres `L`, et qui écrit chacun de ces nombres dans le fichier spécifié, à raison d'un nombre par ligne.
   N'oubliez pas que, pour écrire un nombre dans un fichier, il faut commencer par le convertir en chaîne de caractères à l'aide de `str()` !
   b. Tester cette fonction avec la liste de votre choix.
2. Écrire une fonction `ajoute_num(nouveau_fichier, vieux_fichier)` qui copie le contenu d'un fichier existant dans un nouveau fichier en ajoutant au début de chaque ligne : 'ligne' et le numéro de la ligne. Le nouveau fichier devra donc être de la forme :

```
ligne 1 : ...
ligne 2 : ...
ligne 3 : ...
...
```

3. Écrire une fonction `separe(texte)` qui prend en entrée une chaîne de caractères `texte`, et qui crée un fichier `mots.txt` contenant la liste des mots présents dans `texte`. Par exemple, si `texte = "abricot pomme orange"`, alors le fichier créé doit être :

```
abricot
pomme
orange
```

III Compléments

1) Contexte – Commande `with`

On a vu que la commande **`open()`** doit toujours être associée à la commande **`close()`** correspondante, sinon le fichier restera ouvert en prenant une part de mémoire vive de l'ordinateur et ne pourra pas être modifié par d'autres programmes. Par conséquent, on devrait toujours faire suivre **`open`** d'un **`close`** associé. Mais si une erreur se produit entre les deux commandes, le programme s'arrêtera sans fermer le fichier. Pour éviter cela, on peut utiliser un **contexte** avec la commande `with`.
Par exemple, considérons le script suivant, qui affiche successivement toutes les lignes du fichier texte `zoo` :

```
f = open("zoo.txt", "r")
listeLignes = f.readlines()
for ligne in listeLignes:
    print(ligne)
f.close()
```

En utilisant la commande `with`, ce script ce ré-écrit ainsi :

```
with open("zoo.txt", "r") as f:
    listeLignes = f.readlines()
    for ligne in listeLignes:
        print(ligne)
```

Remarques :

- L'instruction `with` introduit un bloc d'indentation. C'est à l'intérieur de ce bloc que nous effectuons toutes les opérations sur le fichier.
- Une fois sorti du bloc d'indentation, Python fermera automatiquement le fichier. Il n'est donc pas nécessaire de faire appel à la méthode `.close()`.
- Si une erreur se produit lors de l'exécution du programme, la commande `with` referme automatiquement le fichier avant de renvoyer le message d'erreur : cela garantit ainsi que le fichier est toujours convenablement fermé, même en cas de dysfonctionnement du programme.

2) Contrôle du curseur de lecture

- `curseur.seek(n)` place le curseur au n^{me} caractère du fichier ;
- `curseur.tell()` renvoie la position du curseur dans le fichier.

Exercices bonus

Splitting de chaines de caractères

Exercice 8

1. Écrire une fonction `sommeTexte(ch)` prenant en entrée une chaîne de caractères `ch` formée de nombres séparés par des espaces, par exemple `"10.1 5.21 12"`, et qui renvoie la somme de ces nombres.
2. Écrire une fonction `sommeComplete(nom_fichier)` prenant en entrée un fichier contenant des nombres séparés par des espaces (éventuellement répartis sur plusieurs lignes), et renvoie la somme de tous ces nombres.

Lecture de fichiers FASTA

FASTA est un format de fichier permettant de stocker des séquences nucléiques ou protéiques sous forme textuelle. A l'origine, il s'agit du format utilisé par le programme informatique d'alignement de séquences également appelé FASTA, mais il est maintenant possible de récupérer des séquences en format FASTA sur quasiment toutes les bases de données de séquences protéiques et nucléiques (SwissProt, UniProt, *nr*, etc.).

Un fichier FASTA contient une ou plusieurs séquences (auquel cas on parle de fichier *multi-fasta*). Une séquence dans un fichier s'organise ainsi : une ligne avec un identifiant, commençant toujours par ">", suivie de une ou plusieurs lignes contenant la séquence. Par exemple, voici un fichier FASTA contenant 3 séquences :

```
>lcl|NC_003977.2_cds_YP_009173867.1_5 [gene=X] [locus_tag=HBVgp3]
ATGGCTGCTAGGCTGTGCTGCCAACTGGATCCTGCGCGGGACGTCCTTTGTTTACGTCCCGTCGGCGCTG
AATCCTGCGGACGACCCTTCTCGGGGTCGCTTGGGACTCTCTCGTCCCCTTCTCCGTCTGCCGTTCCGAC
CGACCACGGGGCGCACCTCTCTTTACGCGGACTCCCCGTCTGTGCCTTCTCATCTGCCGGACCGTGTGCA
CTTCGCTTCACCTCTGCACGTCGCATGGAGACCACCGTGAACGCCCACCAAATATTGCCCAAGGTCTTAC
ATAAGAGGACTCTTGGACTCTCAGCAATGTCAACGACCGACCTTGAGGCATACTTCAAAGACTGTTTGTT
TAAAGACTGGGAGGAGTTGGGGGAGGAGATTAGGTTAAAGGTCTTTGTACTAGGAGGCTGTAGGCATAAA
TTGGTCTGCGCACCAGCACCATGCAACTTTTTCACCTCTGCCTAA
>lcl|NC_003977.2_cds_YP_009173857.1_6 [gene=C] [locus_tag=HBVgp4]
ATGCAACTTTTTCACCTCTGCCTAATCATCTCTTGTTCATGTCCTACTGTTCAAGCCTCCAAGCTGTGCC
TTGGGTGGCTTTGGGGCATGGACATCGACCCTTATAAAGAATTTGGAGCTACTGTGGAGTTACTCTCGTT
TTTGCCTTCTGACTTCTTTCCTTCAGTACGAGATCTTCTAGATACCGCCTCAGCTCTGTATCGGGAAGCC
TTAGAGTCTCCTGAGCATTGTTCACCTCACCATACTGCACTCAGGCAAGCAATTCTTTGCTGGGGGGAAC
TAATGACTCTAGCTACCTGGGTGGGTGTTAATTTGGAAGATCCAGCGTCTAGAGACCTAGTAGTCAGTTA
TGTCAACACTAATATGGGCCTAAAGTTCAGGCAACTCTTGTGGTTTCACATTTCTTGTCTCACTTTTGGA
AGAGAAACAGTTATAGAGTATTTGGTGTCTTTCGGAGTGTGGATTCGCACTCCTCCAGCTTATAGACCAC
CAAATGCCCCTATCCTATCAACACTTCCGGAGACTACTGTTGTTAGACGACGAGGCAGGTCCCCTAGAAG
AAGAACTCCCTCGCCTCGCAGACGAAGGTCTCAATCGCCGCGTCGCAGAAGATCTCAATCTCGGGAATCT
CAATGTTAG
>lcl|NC_003977.2_cds_YP_009173868.3_7 [gene=C] [locus_tag=HBVgp4]
ATGGACATCGACCCTTATAAAGAATTTGGAGCTACTGTGGAGTTACTCTCGTTTTTGCCTTCTGACTTCT
TTCCTTCAGTACGAGATCTTCTAGATACCGCCTCAGCTCTGTATCGGGAAGCCTTAGAGTCTCCTGAGCA
TTGTTCACCTCACCATACTGCACTCAGGCAAGCAATTCTTTGCTGGGGGGAACTAATGACTCTAGCTACC
TGGGTGGGTGTTAATTTGGAAGATCCAGCGTCTAGAGACCTAGTAGTCAGTTATGTCAACACTAATATGG
GCCTAAAGTTCAGGCAACTCTTGTGGTTTCACATTTCTTGTCTCACTTTTGGAAGAGAAACAGTTATAGA
GTATTTGGTGTCTTTCGGAGTGTGGATTCGCACTCCTCCAGCTTATAGACCACCAAATGCCCCTATCCTA
TCAACACTTCCGGAGACTACTGTTGTTAGACGACGAGGCAGGTCCCCTAGAAGAAGAACTCCCTCGCCTC
GCAGACGAAGGTCTCAATCGCCGCGTCGCAGAAGATCTCAATCTCGGGAATCTCAATGTTAG
```

Pour avoir un fichier FASTA réel sur lequel tester vos fonctions, vous pouvez télécharger des séquences depuis la base de données du NCBI (National Center for Biotechnology Information).
Pour la suite, on utilisera l'exemple des séquences formant le génome du virus de l'Hépatite B. Vous pouvez télécharger ces séquences à l'adresse suivante :

`https://www.ncbi.nlm.nih.gov/nuccore/NC_003977.2`

- Cliquez sur **Send To** en haut à droite de la page
- Sélectionnez "Coding Sequences"
- Vérifiez que le format est bien "FASTA Nucleotide"
- Cliquez sur **Create File** pour télécharger le fichier.

Une fois téléchargé, mettez ce fichier dans le même répertoire que vos scripts, sous un nom court pour pouvoir le taper facilement (par exemple : `sequences.txt`).

Exercice 9 *Parsing*

1. Écrire une fonction Python prenant en entrée le nom d'un fichier FASTA, et renvoyant le nombre de séquences contenues dans le fichier. *Rappelez-vous qu'il y a autant de séquences que d'identifiants, et que les identifiants ont un format particulier...*
2. Écrire une fonction Python prenant en entrée le nom d'un fichier FASTA, et renvoyant un couple `[id, seq]` où `id` est l'identifiant de la première séquence du fichier, et `seq` la chaîne de caractères contenant la séquence elle-même. Attention, si la séquence s'étend sur plusieurs lignes, il faut toutes les concaténer pour les renvoyer sous forme d'une seule longue chaîne de caractères ! (Et notamment, il faut supprimer les "\n" qui concluent chaque ligne ...)
   Pour vous aider, vous pouvez vous inspirer si besoin du programme suggéré dans la question suivante.
3. Compléter la fonction suivante, prenant en entrée le nom d'un fichier FASTA, pour qu'elle renvoie une liste contenant le couple `[id, sequence]` pour chaque séquence dans le fichier source.

```
def lire_sequences(nomfichier):

    seq = ""
    listeSequences = []

    f = open(...)           # j'ouvre mon fichier
    for ligne in f:
        if ...:             # si la ligne est un identifiant
            if seq!="":
                listeSequences.append([id, seq])
            id=ligne[1:-1] # la ligne sans le ">" au début ni le '\n' à la fin
            seq = ""
        else:
            seq += ...      # on ajoute la ligne sans le '\n' à la fin
    listeSequences.append([id, seq])
    f.close()
    return listeSequences
```

4. Écrire une fonction Python prenant en entrée le nom d'un fichier FASTA, et affichant la taille de la plus grande séquence, la taille de la plus petite séquence, et la taille moyenne des séquences dans le fichier. *D'un point de vue "performance", est-il judicieux de réutiliser la fonction codée dans la question précédente ?*

Exercice 10 *Recherche de motifs*

On veut écrire un moteur de recherche simpliste pour trouver des motifs dans des séquences d'ADN.

1. Écrire une fonction `est_polynucleique(sequence)`, prenant en entrée une chaîne de caractères `sequence`, et renvoyant `True` si la séquence est valide, et `False` sinon (une séquence est dite valide si elle ne contient que les lettres A, T, G ou C).
2. Écrire une fonction prenant en entrée le nom d'un fichier FASTA, demandant à l'utilisateur de rentrer un motif nucléique *valide* (utilisant la fonction `est_polynucleique` précédemment codée pour valider le motif), et affichant les identifiants des séquences contenant ce motif.

Exercice 11 *Lettres les plus fréquentes*

On s'intéresse à l'identification des lettres les plus fréquentes au sein de plusieurs séquences.

1. Écrire une fonction `compte_lettre(sequences, i)` prenant en entrée une liste de n séquences `sequences`, et un entier i, et renvoyant le couple $(l, freq(i,l))$ où l est la lettre apparaissant le plus de fois à la position i dans les séquences, et $freq(i,l)$ la fréquence d'apparition de la lettre l à la position i.

   Exemple : Avec `sequences=["ATTAGC","ATCCGA","TTGACC"]`, on aura

   ```
   >>> compte_lettre(sequences, 0)
   ("A", 0.6666666666666666)

   >>> compte_lettre(sequences, 1)
   ("T", 1)

   >>> compte_lettre(sequences, 2)
   ("C", 0.3333333333333333) # ou n'importe quelle lettre ici
   ```

2. Écrire une fonction `consensus(nomfichier, i)`, prenant en entrée le nom d'un fichier FASTA et un entier i, et renvoyant la lettre apparaissant le plus de fois à la position i parmi toutes les séquences.

Exercice 12 *Enzyme de restriction*

Une enzyme de restriction est une enzyme capable de couper une molécule d'ADN au niveau d'un motif particulier dans la séquence. On s'intéresse ici à la réalisation d'un script capable de prédire les séquences obtenues après l'action de la **TaqI** sur un génome donné (séquence target : $T|CGA$).

1. Écrire une fonction `trypsine(sequence)` prenant en entrée une séquence d'ADN, et renvoyant la liste des séquences obtenues après digestion. L'effet de la digestion est le suivant : à chaque fois que la séquence "TCGA" apparaît, on coupe la séquence entre "T" et "CGA". Par exemple :

   ```
   >>> trypsine("ATTCGATCG")
   ["ATT", "CGATCG"]

   >>> trypsine("ATTCGATCGTCGAAAATA")
   ["ATT", "CGATCGT", "CGAAAATA"]
   ```

2. Catastrophe ! Au cours de vos recherches, vous avez séquencé le génome d'une souche inconnue de *Thermophilus aquaticus*. Cependant, un bug informatique a supprimé le fichier FASTA contenant le génome complet d'origine, obtenu par séquençage après des mois à isoler la souche. Heureusement, vous avez retrouvé le fichier FASTA contenant les séquences résultant de la **TaqI**. Écrivez un script capable de reconstituer l'ensemble des génomes d'origine possibles à partir de votre fichier. *NB : cela revient à générer l'ensemble des permutations des k séquences du fichier. On pourra procéder récursivement, ou chercher sur Google la fonction de la librairie standard permettant de le faire à notre place...*
3. À votre avis, comment pourrait on savoir lequel de ces génomes est le bon parmi tous les génomes regénérés ? (proposez une approche bioinformatique, pas un programme Python !)