TP d'Informatique

28 Mars 2018
BT1
Durée : 2 heures

Nom :
Prénom:
Groupe:

Correcteur:

Kévin GAST, Alexandre MANUEL et Zinedine REBIAI
v. 1.0
9 pages
BT1_2022_TP_2018-03-28

Consignes pour ce TP

- Lire la totalité du sujet avant de commencer.
- Ne pas savoir expliquer votre code si on vous le demande représente un cas de triche.
- Posez vos questions aux assistants pendant le TP, et envoyez un mail à supbiotech-bioinfo-bt1@googlegroups.com en dehors.
- Si vous avez un message à nous faire passer concernant votre TP, merci d'ajouter cela dans un fichier README.txt dans votre rendu. Les retours sur ce TP sont les bienvenus également.

Introduction

Le principe du TP

Vous pouvez dans votre vie, en laboratoire, devoir traiter des données informatiques. Ces données seront quasi systématiquement dans des fichiers, ceci dû aux avantages que représentent les fichiers par rapport à du texte copié/collé (taille maximale bien plus importante, persistance des données, etc). Nous allons à travers ce TP traiter des fichiers contenant de l'ADN et de l'ARNm.

Le TP est divisé en **exercices** qui peuvent être réalisés **indépendamment** les uns des autres. Seul le dernier exercice a comme prérequis d'avoir les autres exercices qui fonctionnent.

Lisez attentivement le sujet afin de bien comprendre ce que l'on attend de vous, normalement il suffit à lui seul pour réaliser le TP grâce aux indications que l'on vous fournit. Vous pouvez également poser des questions mais nous ne vous donnerons bien évidement pas la solution au problème.

Comment réaliser ce TP

Vous devrez écrire le corps des fonctions dans les fichiers python donnés. Afin de tester votre TP, vous pouvez lancer *check.py*.

Veillez à ne pas casser les signatures des fonctions dans les fichiers donnés.

Rappel :

Si vous voulez ajouter un paramètre à une des fonctions sans casser sa signature, deux possibilités s'offrent à vous. Admettons que vous vouliez ajouter un paramètre `param` à la fonction `grid_is_complete(grid)`.

Vous pouvez soit ajouter le paramètre avec une valeur par défaut définie :

```
def is_finished(grid, param=42):
    # ...
```

Soit transformer `is_finished` en « fonction chapeau » :

```
def is_finished_sub(grid, param):
    # ...

def is_finished(grid):
    return is_finished_sub(grid, 42)
```

Avoir la signature suivant ne fonctionnera pas et occasionnera des points en moins sur votre note :

```
def is_finished(grid, param):
    # ...
```

La triche

La triche sera fortement sanctionnée. Tricher veut dire ne pas savoir expliquer votre code si on vous le demande. Vous avez en revanche le droit, et êtes même invités à collaborer avec vos camarades. Ne prenez pas le risque de rendre un code que vous ne comprenez pas. Nous ferons des interviews aléatoires pour vous demander d'expliquer votre TP.

Le rendu du TP

Votre rendu doit être sous la forme d'une archive (fichier compressé). Cette dernière devra se nommer de la manière suivante :

```
tp9_nom_prenom.zip
```

Remplacez `nom` et `prenom` par votre nom et prénom sans espace et sans accent.

Cette archive **doit** contenir un unique dossier portant le même nom que l'archive (sans le .zip à la fin). Ce dossier devra contenir au moins les fichiers suivants :

```
filter.py
split.py
transcription.py
translation.py
do_all.py
```

Vous pouvez ajouter un fichier README.txt dans ce même dossier si vous avez un message à nous faire passer concernant votre rendu ou un retour à faire sur le sujet du TP.

/!\ ATTENTION /!\ : Un rendu non conforme entrainera un retrait de deux points sur votre note finale.

Tester votre TP

Vous pouvez à tout moment tester votre TP pour avoir une idée de si celui-ci est correct ou non. Pour cela une possibilité s'offre à vous :

La test suite

En exécutant le fichier *check.py*, vous aurez un retour sur vos paliers. Ce retour est à titre indicatif. Les tests que nous utiliserons pour vous noter seront similaires mais différents.

Rappels

Fonctions liées aux fichiers :

```
f = open(nom_du_fichier, mode_d_ouverture) # f est de type file
lines = f.readlines() # lines est la liste des lignes du fichier
content = f.read() # content contient tout le contenu du fichier
f.write('hello') # écrit hello dans le fichier f
f.close() # ferme le fichier f
```

Modes d'ouverture :

- r : Read. Permet de lire dans le fichier
- w : Write. Permet d'écrire dans le fichier. Avant ouverture, le fichier est vidé. S'il n'existe pas il est créé.
- a : Append. Permet d'écrire dans le fichier à partir de la fin du dit fichier. Sert à ajouter du texte à un fichier. Si le fichier n'existe pas, il est créé.
- r+ : Read + Write. Permet de lire et écrire dans le fichier. Ne détruit pas le contenu du fichier comme w. Le curseur est placé au début du fichier

Indication pratique

Pour ne pas avoir les \n à la fin des lignes lues par la fonction readlines(), vous pouvez utiliser ce code :

```
lines = [line.strip('\r\n') for line in f.readlines()]
```

au lieu de celui-ci :

```
lines = f.readlines()
```

1. Exercice 1 : Filter

Ecrire la fonction `seq_filter_file(input, output)` qui prend en argument deux chemins de fichiers. Cette fonction ouvre le fichier `input` puis pour chaque ligne représentant de l'ADN ou de l'ARNm, écrit cette ligne dans `output`. La fonction retourne également un dictionnaire qui contient le nombre de lignes représentant de l'ADN et de l'ARNm.

Exemple :

Le fichier output_ex1.txt n'existe pas initialement.

Contenu d'input_ex1.txt :

```
Chaines d'ADN :
ATGCAGATC
CCGATGAGG
ATTCAG

Chaines d'ARNm :
UACGUCUAG
GGCUACUCC
```

Code exécuté :

```
print(seq_filter_file('input_ex1.txt', 'output_ex1.txt'))
```

Affiche :

```
{'dna':3, 'mrna':2}
```

Contenu d'output_ex1.txt après appel de la fonction :

```
ATGCAGATC
CCGATGAGG
ATTCAG
UACGUCUAG
GGCUACUCC
```

2. Exercice 2 : Split

Ecrire la fonction `seq_split_file(input)` qui prend en argument un chemin de fichier. Le fichier contient exclusivement des lignes d 'ADN et d'ARNm. Cette fonction ouvre le fichier `input` et inscrit les lignes représentant de l'ADN et les lignes représentant de l'ARNm dans deux fichiers distincts. Les noms de ces deux fichiers se construisent en utilisant le nom du fichier `input` préfixé de **`dna_`** ou **`mrna_`**.

Exemple :

Les fichiers dna_input_ex2.txt et mrna_input_ex2.txt n'existent pas initialement.

Contenu d'input_ex2.txt :

```
ATGCAGATC
CCGATGAGG
ATTCAG
UACGUCUAG
GGCUACUCC
```

Code exécuté :

```
seq_split_file('input_ex2.txt')
```

Contenu de dna_input_ex2.txt après appel de la fonction :

```
ATGCAGATC
CCGATGAGG
ATTCAG
```

Contenu de mrna_input_ex2.txt après appel de la fonction :

```
UACGUCUAG
GGCUACUCC
```

3. Exercice 3 : Transcription

Ecrire la fonction `dna_to_mrna_file(input)` qui prend en argument un chemin de fichier. Le fichier ne contient que des lignes représentant de l'ADN. Cette fonction écrit la transcription de chaque ligne dans un nouveau fichier. Le nom de ce nouveau fichier se construit en ajoutant le préfixe **mrna_** au paramètre.

Exemple :

Le fichier mrna_dna_input_ex3.txt n'existe pas initialement.

Contenu de dna_input_ex3.txt :

```
ATGCAGATC
CCGATGAGG
ATTCAG
```

Code exécuté :

```
dna_to_mrna_file('dna_input_ex3.txt')
```

Contenu de mrna_dna_input_ex3.txt après appel de la fonction :

```
UACGUCUAG
GGCUACUCC
UAAGUC
```

4. Exercice 4 : Translation

Ecrire la fonction `mrna_to_aa_files(flist, output)` qui prend en argument une liste de chemins de fichiers ne contenant que des lignes d'ARNm, et prend également en paramètre un autre chemin de fichier. Cette fonction ouvre tous les fichiers de `flist` et ajoute la traduction de chacune des lignes dans le fichier `output`.

Exemple :

Le fichier aa_ex4.txt n'existe pas initialement.

Contenu de mrna_1_ex4.txt :

```
UACGUCUAG
GGCUACUCC
```

Contenu de mrna_2_ex4.txt :

```
UACGUCUAG
GGCUACUCC
UAAGUC
```

Code exécuté :

```
mrna_to_aa_files(['mrna_1_ex4.txt', 'mrna_2_ex4.txt'], 'aa_ex4.txt')
```

Contenu de aa_ex4.txt après appel de la fonction :

```
TyrVal
GlyTyrSer
TyrVal
GlyTyrSer
```

5. Exercice 5 : Do all

Rappel : Pour faire cet exercice, vous devez d'abord avoir fait les quatre premiers exercices.

Ecrire la fonction `check_for_prots(input, output)` qui prend en paramètre deux chemins de fichiers. Le fichier `input` peut contenir des lignes dont certaines représentent de l'ADN ou de l'ARNm (comme dans l'exercice 1). Cette fonction ouvre `input` puis appelle toutes les fonctions créées précédemment afin d'ajouter, à la fin de output, la traduction en acides aminés des séquences d'ADN ou d'ARNm présentes dans le premier fichier.

Exemple :

Le fichier aa_ex5.txt peut déjà contenir des choses. Celles-ci ne doivent pas être effacées.

Contenu du fichier input_ex1.txt :

```
Chaines d'ADN :
ATGCAGATC
CCGATGAGG
ATTCAG

Chaines d'ARNm :
UACGUCUAG
GGCUACUCC
```

Code exécuté :

```
check_for_prots('input_ex1.txt', 'aa_ex5.txt')
```

Contenu du fichier aa_ex5.txt après appel de la fonction :

```
... contenu initialement dans aa_ex5.txt ...
TyrVal
GlyTyrSer
TyrVal
GlyTyrSer
```

Félicitations ! Vous êtes venus à bout de ce TP. N'oubliez pas de le rendre. Si vous avez eu des difficultés, pensez à les indiquer dans un fichier README.txt que nous lirons attentivement. Un code qui ne marche pas mais qui montre vos essais vaut plus que pas de code du tout.

N'hésitez pas à appeler un assistant lors de la séance de TP, ou à envoyer un mail à `supbiotech-bioinfo-bt1@googlegroups.com` si celle-ci est terminée.