

Activité 6

De GGAAATT... à Met-Asp-Gly...

Autrement dit « Quelques éléments de manipulation de texte... »

Je vous propose une version adaptée d'un cours-TD que je propose au lycée¹.

Otto MATO et Kurt KETCHUP, généticiens et cultivateurs de tomates, ont vu leur récolte décimée par un virus. Ils font appel à vous pour synthétiser la protéine responsable afin de mettre au point un antivirus. Afin de remplir votre mission, ils vous envoient :

- le fichier *GenomeTomatoBushy.txt* qui contient le génome de l'ADN du virus tomato bushy stunt virus mis en cause ;
- afin de vérifier l'intégrité du fichier, son code de contrôle est : 0xD009² ;
- une mini base de donnée *Genome.sqlite3* qui contient le codage des acides aminés
- le protocole à suivre... c'est à dire ce cours-TD !

On donne quelques éléments de syntaxe pour le traitement des chaînes de caractères.

ord	retourne le code ASCII (en base 10) du caractère : <code>ord('a') = 97</code>
hex	retourne un nombre en hexadécimal : <code>hex(124) = '0x7c'</code> (Ox pour hexadécimal)
float	convertit, si possible, la chaîne en réel : <code>x = float('3.4')</code>
str	écrit un nombre sous forme de chaîne de caractères : <code>s = str(3.4)</code>
replace	remplace une chaîne de caractères 'x' par une autre 'y' : <code>s = s.Replace('x','y')</code>
split	découpe la chaîne de caractères au caractère spécifié : <code>t = s.split(',')</code>

1 Lecture - écriture d'un fichier texte

1.1 Lecture d'un fichier texte

a Séquence de l'ADN

Listing 6.1 – Lecture d'un fichier texte

1. Comme j'avais quelques doutes sur les résultats, avant de vous l'envoyer, j'ai fait relire la trame du sujet à une collègue de bio. Le principe de la méthode est bon ; c'est déjà ça... mais le fichier proposé ne correspond pas à un bout d'ADN ! Damned, afin d'être exploitable, j'ai pris je ne sais où le plus petit fichier de la base de données et, manifestement... il y a bug ! Le problème est que vous allez trouver des « protéines » qui contiennent, sauf erreur : 4, 34, 14, 28, 28 et 23 acides aminés ! On dira que ce sont des microprotéines ; je vous enverrai un bon fichier un jour. Désolé !

2. Ox devant une chaîne de caractères signifie qu'il s'agit d'un nombre en hexadécimal

```

1 fichier = open("GenomeTomatoBushy.txt", "r")
2 seq = fichier.read()
4 fichier.close()

```

Supposons une organisation des fichiers :

C :

Python

Test

fichier1.txt

Travail

fichier2.txt

script.py

Par défaut, le gestionnaire de fichier, recherche le fichier souhaité dans le dossier contenant le script python. Sur notre exemple, on peut accéder à *fichier2.txt* sans spécifier de chemin d'accès. Si ce n'est pas le cas, il faut le faire :

- soit en relatif par rapport au dossier en cours : pour accéder à *fichier1.txt*, il faudra écrire `open("../Test/fichier1.txt", "r")` ;
- soit en absolu par rapport à la racine du système de fichier (la structure est alors liée au système d'exploitation d'où la nécessité, parfois d'utiliser des slash ou antislash). Il faudra écrire `open("C:/Python/Test/fichier1.txt", "r")`.

Un fichier doit être ouvert `fichier = open(chemin d'accès, "r")` puis... fermé `fichier.close()`. Pour lire le contenu du fichier on peut :

- lire l'intégralité du fichier et le stocker dans une chaîne de caractères `s = fichier.read()` ;
- lire le fichier ligne après ligne et le stocker dans une chaîne de caractères `fichier.readline()`. On peut ainsi faire un traitement ligne après ligne ; il faudra s'assurer de parcourir l'intégralité du fichier ;
- lire toutes les lignes d'un fichier `s = fichier.readlines()` et le stocker dans une liste de chaînes de caractères.

Ici, la séquence n'est constituée que d'une seule chaîne de caractère, la première solution est préférable.

b Exploitation d'un fichier .csv

L'exploitation d'un tel fichier a déjà été abordé dans les activités précédentes !

Si le fichier contient, par exemple, des données expérimentales séparées par des virgules ou point virgules (fichier .csv par exemple) ; la dernière solution est fort intéressante.

Par exemple, lors de l'acquisition d'un signal sinusoïdal, on peut stocker dans un fichier pour chaque point le temps et la valeur de la ddp mesurée (les valeurs sont séparées par une virgule et les points sont séparés par un retour à la ligne) : 0,-3.807332002

0.0009,1.99171806

0.0018,7.155799425

0.0027,10.10245104

0.0036,9.9189054

... `signal = fichier.readlines()` contient alors la liste `['0,-3.807332002', '0.0009,1.99171806', '0.0018,7.155799425', '0.0027,10.10245104', '0.0036,9.9189054'...]`

`s1 = signal[1]` contient la chaîne de caractère `'0.0009,1.99171806'`³. On peut utiliser l'instruction `split` pour découper une chaîne de caractères : `r1 = s1.split(',')` contient alors une liste de deux chaînes de caractères : `['0.0009', '1.99171806']`. Il reste enfin à convertir les chaînes de caractères en nombres réels : `t1 = float(r1[0])`.

1.2 Écriture d'un fichier texte

Lorsque l'on aura terminé notre travail, on enregistrera dans le fichier *GenomeProteine.txt* la liste des acides aminés constituant la protéine du virus.

Listing 6.2 – Ecriture d'un fichier texte

```
1 res = open('GenomeProteine.txt', 'w', encoding='utf8')
2 for p in proteine:
3     print (p)
4     res.write(p)
5 res.close()
```

Il n'y a pas de soucis ici puisque l'on cherche à écrire des chaînes de caractères dans un fichier texte (excepté la précision `encoding = 'utf8'` pour gérer les caractères accentués). Pour écrire des nombres, il faudrait les convertir : la syntaxe la plus simple est `str` mais on peut formater la chaîne de caractère affichée (en particulier, pour préciser le nombre de chiffres significatifs à afficher).

2 Code de contrôle d'un texte

Afin de valider le transfert de données (ou d'un mot de passe!), on peut associer à un texte un code de contrôle (ou signature ou checksum). Une chaîne de caractères, ou un nombre est calculé grâce à un algorithme idoine à partir du texte d'origine et du texte transmis; les deux informations doivent, bien sûr, correspondre.

Question 1 Que fait cet algorithme? Vérifier l'intégrité du fichier transmis.

```
1 def checksum(txt):
2     s1 = 0
3     s2 = 0
4     y = 1
5     for i in range (len(txt)):
6         s1+=ord(txt[i])
7         s2+=y*ord(txt[i])
8         y += 1
9         if y==17:
10            y=1
11     s1 = s1 %256
12     s2 = s2 %256
13     return hex (256*s1+s2)
```

Il y a, bien sûr, plus compliqué!!!

3. avec certains types de fichiers (ou de système d'exploitation) on obtiendra plutôt `'0.0009,1.99171806\ n'`; pour supprimer une partie de la chaîne de caractère, on utilise l'instruction `replace` : `s1 = s1.replace('\ n', '')`

3 Du génome de l'ADN à la séquence de l'ARN

La variable *seq* (cf listing 6.1) contient la séquences des différentes bases azotées A (adénine), T (thymine), C (cytosine), G (guanine) constitutive d'un des deux brins d'ADN du virus.

Afin de reconstituer le brin d'ADN, il faut compléter la séquence initiale par la séquence, lue en ordre inverse, en permutant A et T d'une part, C et G d'autre part. Par exemple si l'on part de ATTC...TCGA ou obtiendra ATTC...TCGATCGA...GAAT.

Question 2 Ecrire une routine reconstituant le brin d'ADN complet à partir de la séquence *seq* passée en paramètre.

L'ADN, confinée dans le noyau de la cellule, transmet alors son code à l'ARN messager. L'ADN des chromosomes est traduite en ARN; la séquence reste inchangée à ceci près que l'on remplace la base T par U (uracile).

Question 3 Ecrire une routine retournant la séquence de l'ARN.

On suppose que la variable *seq_arn* contient cette séquence.
L'ARN messager est ensuite traduit en protéine.

4 A la recherche du codon « start »

4.1 Recherche « naïve » d'une chaîne de caractères

L'algorithme naïf de recherche de caractère consiste, pour chaque position du texte, à comparer caractère par caractère de gauche à droite le texte et la chaîne recherchée et de continuer ainsi tant que les lettres sont les mêmes. Si on arrive à la fin de la chaîne recherchée, une occurrence de cette chaîne est trouvée et on continue. Tout se passe comme si on déplaçait une « fenêtre » sur le texte... et on regarde si le contenu de la fenêtre correspond à la chaîne recherchée.

Question 4 Ecrire le code python d'une routine `recherche(M,T)` implémentant l'algorithme de recherche naïf d'une chaîne de caractères *M* dans un texte *T*. On attend, en retour, un tableau de tous les indices de début de la chaîne *M* dans le texte *T*.
On testera cette fonction sur un exemple simple.

4.2 Recherche du codon start

On appelle codon la succession de 3 bases azotées (nucléotides). Le codage des diverses protéines dans l'ARN est compris entre un codon start (constitué de 'AUG') et un codon stop (constitué de 'UAA', 'UAG' ou 'UGA').

- entre ce codon start et un codon stop, il doit y avoir un multiple de 3 nucléotides (si on a AUGUGGAFUAA, UAA ne peut pas être un codon stop puisqu'il y a 5 nucléotides après le codon stop).
- le nombre de nucléotides doit être grand... au moins 3×190 dans le vrai virus de la tomate, ici on a des microprotéines!!!
- la séquence ainsi repérée doit posséder en amont et en aval des séquences qui ne sont pas traduites. Ces séquences ne sont pas forcément des multiples de 3. On peut donc avoir un codon start d'indice 100 par exemple, et un autre d'indice 200.

Question 5 Ecrire le code permettant de stocker dans la variable *starts* les indices des différentes occurrences de la chaîne 'AUG' dans *seq_arn*.

5 Structure des protéines

5.1 Recherche du premier enchaînement

Intéressons nous au premier codon start trouvé dans *seq_arn*. Il faut maintenant isoler la chaîne de caractère (chaque caractère correspond à une base azotée) comprise entre ce codon start (inclus) et un codon stop (exclus).

Question 6 Ecrire le code permettant d'extraire cette chaîne de caractères et de stocker dans une variable *sequence0* la liste des différents codons constitutifs. Le contenu de cette variable *sequence0* devrait correspondre à ['AUG', 'GAC', 'GGU', 'UUG']. Attention, on veut une liste de chaînes de 3 caractères !

5.2 Et ainsi de suite...

On complique un peu...

Question 7 Ecrire le code permettant de stocker dans une variable *sequences* la liste des différentes séquences rencontrées. Chacune de ces séquences est, elle-même, une liste de chaînes de caractères. Attention, une séquence 'AUG' comprise entre un codon start et un codon stop n'est pas un codon start !⁴.

5.3 Structure des protéines

On y est presque, un codon (3 bases azotées) codant un acide aminé, il reste à identifier les enchaînements d'acides aminés.

On verra en fin d'année, pour les survivants, quelques notions sur la gestion de bases de données. On l'utilise, ici comme une boîte noire. Je vous donne le code permettant, par exemple, de trouver l'acide aminé correspondant au premier codon de chaque enchaînement. Le codon AUG correspond bien à la méthionine.

```
1 import sqlite3
2 conn = sqlite3.connect('Genome.sqlite3')
3 cur = conn.cursor()
4
5 param = ("AUG",) # il faut la virgule, !!!
6 cur.execute('SELECT AcideAmine FROM GENOME WHERE Codon = ?',
7             param)
8 AA = cur.fetchone()
9 print("Acide aminé trouvé : ", AA)
10
11 cur.close()
12 conn.close()
```

4. *sequences* est donc une liste de listes ; on abordera bientôt cette structure de donnée dans le traitement d'image. En fait, ça marche comme une liste... de listes. Par exemple *sequences*[1] donne accès à la première sous-liste (d'indice 1) et *sequences*[1][3] donnera accès au quatrième élément de la seconde liste !

a Première microprotéine

Question 8 Modifier ce code afin de stocker dans la variable *proteine0* l'enchainement des acides aminés correspondant à *sequence0*

b Pour conclure...

Question 9 Ecrire le code permettant de stocker dans une variable *proteines* la liste des différentes protéines; chacune des protéines étant une liste d'acides aminés.

Question 10 Ecrire la structure de ces différentes protéines dans un fichier texte et... envoyez le moi!

Protéine n°1 - 4 acides aminés : méthionine-acide aspartique-glycine-leucine

Protéine n°2 - 34 acides aminés : ...

6 Pour le fun... à vous de jouer les Merrifield !

C'est juste pour vous montrer que l'on peut alors faire des choses « originales » avec la programmation objet. Mais, ce n'est plus très blond tout ça!

Je vous donne à la fin de la trame de cette activité, le code d'une classe *Robot* qui va simuler les différentes étapes de l'enchainement des acides aminés et le code qui permet de l'exploiter.

6.1 Classe Robot

Listing 6.3 – Déclaration d'une classe

```

1 class Robot():
2     def __init__(self):
3         self.Resine = ""
4         self.Ajout = ""
5         self.AA = []
6
7     def imprime(self):
8         s = self.Resine
9         if len(self.AA) == 0 and self.Resine != "":
10            s = s + "C1"
11        else :
12            for i in range(len(self.AA)):
13                if i == 0 and self.Resine == "":
14                    s = "HOOC-"
15                else :
16                    s = s + "CO-"
17                s = s + "[" + self.AA[i] + "]-NH"
18                if i == len(self.AA)-1:
19                    s = s + "2"
20                else:
21                    s = s + "-"
22            if self.Ajout != "":
23                s = s + " + " + "HOOC-["+self.Ajout + "]-NH2"
24            print(s)

```

```
25
26     def preparer(self):
27         self.Resine = "Polystère-CH2-"
28         self.imprime()
29
30     def prelever(self, aa):
31         self.Ajout = aa
32         self.imprime()
33
34     def ajouter(self):
35         self.AA.append(self.Ajout)
36         self.Ajout = ""
37         self.imprime()
38
39     def decrocher(self):
40         self.Resine = ""
41         self.imprime()
```

6.2 Instanciation et appel de fonctions

Après, cette classe s'utilise comme n'importe quoi d'autre en python. Ce n'importe quoi d'autre est en fait, la plupart du temps, une classe!

Listing 6.4 – Instanciation et utilisation d'une classe

```
1 r = Robot()
2 r.preparer()
3 for i in range(len(proteine0)):
4     r.prelever(proteine[len(proteine0)-1-i])
5     r.ajouter()
6 r.decrocher()
```

Question 11 Tous les BCPSTistes vont s'empresse d'ajouter des fonctions protection, déprotection à la classe. Chiche :).