Les questions sont plus ou moins indépendantes. Toutes les fonctions écrites (ou mentionnées) dans les questions précédentes peuvent être utilisées a posteriori.

Il est devenu habituel de dire que l'ADN se présente comme un texte composé à l'aide de quatre lettres A,C,G,T, qui s'enchaînent sans interruption et qui est orienté avec un début et une fin.
On souhaite faire une étude statistique des lettres présentes dans une séquence d'ADN.

1. (a) Ecrire une fonction `frequence_A` qui prend en argument une chaine de caractères `ADN` et qui retourne la fréquence de la lettre A dans cette chaîne.

   (b) On souhaite comparer la fréquence d'apparition de la lettre 'A' entre deux séquences d'ADN. Ecrire une fonction Python `compare` qui prend en argument deux chaines de caractères `ADN1`, `ADN2` et retourne 'elles sont proches' si la fréquence de la lettre A dans `ADN1` et dans `ADN2` différe de moins de 0.01. La fonction retournera 'elles ne sont pas proches' dans le cas contraire.

On s'intéresse maintenant aux acides aminés, il faut alors regarder les codons qui se lisent par trois (cf le tableau de la dernière page), on doit donc diviser la chaîne de caractères par codon.

2. (a) Completer (sur votre copie) la fonction `liste_codon` python qui prend en argument une chaine de caractère `ADN` et qui retourne une liste dont les éléments sont les codons de la chaîne. (On supposera que la longueur de la chaine de caractères est bien divisible par 3 sans le vérifier dans la fonction)

```
1 def liste_codon(ADN):
2     L=[]
3     for i in range(0,   ,   ):
4         L=L+[ADN[   :   ]]
5     return(L)
```

   Exemple : si `ADN='GCAGAGTTTTGGTGC'`, la liste retournée sera : `['GCA','GAG', 'TTT','TGG','TGC']`.

   (b) On suppose que l'on a créé une liste `code_genetique` qui contient tous les codons possibles. `code_genetique=['GCA','GCC', 'GCG', ...., 'TAG', 'TGA']`

   Quelle est la longueur de la liste `code_genetique` ? Comment obtenir cette longueur avec une commande Python ?

   (c) On suppose que l'on a une chaine de caractères `ADN` à notre disposition. Ecrire une fonction python `test` qui vérifie si chaque codons de la liste `L=liste_codon(ADN)` est bien un codon du code génétique.

   (d) Compléter (sur votre copie) la fonction `start` qui prend en argument une liste de codons et qui retourne l'indice de la première fois où l'on trouve le codon START ('ATG'). Si jamais il n'y en a pas, elle devra retourner un message d'erreur.

```
1 def start(L):
2     i=0
3     while L[i]          :
4         if i<len(L)-1:
5
6         else:
7             return('pas de codon START')
8     return(   )
```

   (Vous pouvez aussi proposer une fonction différente si vous ne comprenez pas la logique de celle-ci, mais attention aux problèmes d'indices.)

   (e) Ecrire une fonction `stop` qui prend en argument une liste de codons et qui retourne l'indice de la première fois où l'on trouve un codon STOP. Si jamais il n'y en a pas, elle devra retourner un message d'erreur.

   (f) Ecrire une fonction `proteine` qui prend en argument une liste de codons et retourne la sous-liste des codons entre le premier codon START et le premier codon STOP après ce codon START. (Cette sous-liste contiendra les deux codons START et STOP. On ne se penchera pas sur le problème d'erreurs, et on supposera que notre liste contient bien un codon START et un codon STOP dans le bon ordre)

A une séquence d'ADN correspond une unique séquence d'ARN grâce aux règles de complémentarité : G et C sont inversé, A devient U et T devient A. Par exemple, la séquence d'ADN 'AATCGA' est transcrite en 'UUAGCU.'

3. (a) Compléter (sur votre copie) la fonction python `transcription_lettre` qui prend en argument une lettre correspondant à de l'ADN et qui retourne la lettre d'ARN correspondante.

```
1 def transcription_lettre(lettre):
2         if lettre==    :
3                 lettre='G'
4         elif           :
5                 lettre='C'
6         elif lettre=='A':
7
8         elif           :
9                 lettre=
10    return(         )
```

   (b) Ecrire une fonction python `transcription` qui prend en argument une chaine de caractères correspondant à de l'ADN et qui retourne la chaine de caractères d'ARN correspondante.

   Parfois il y a des erreurs dans la transcription et une lettre est mal transmise.

   (c) Ecrire une fonction python `mutation` qui prend en argument une chaine de caractères (correspondant à de l'ADN) et qui retourne une chaine de caractères où les lettres 'C' et 'G' sont inversées avec probabilité 99% et non inversée avec probabilité 1%, la lettre 'A' est bien changée en 'U' avec probabilité 99% et changée en 'C' avec probabilité 1% et la lettre 'T' devient la lettre 'A' avec proba 99% et changée en 'U' avec proba 1% . Après l'avoir importée, on pourra utiliser la fonction random() qui retourne un réel aléatoire entre 0 et 1.