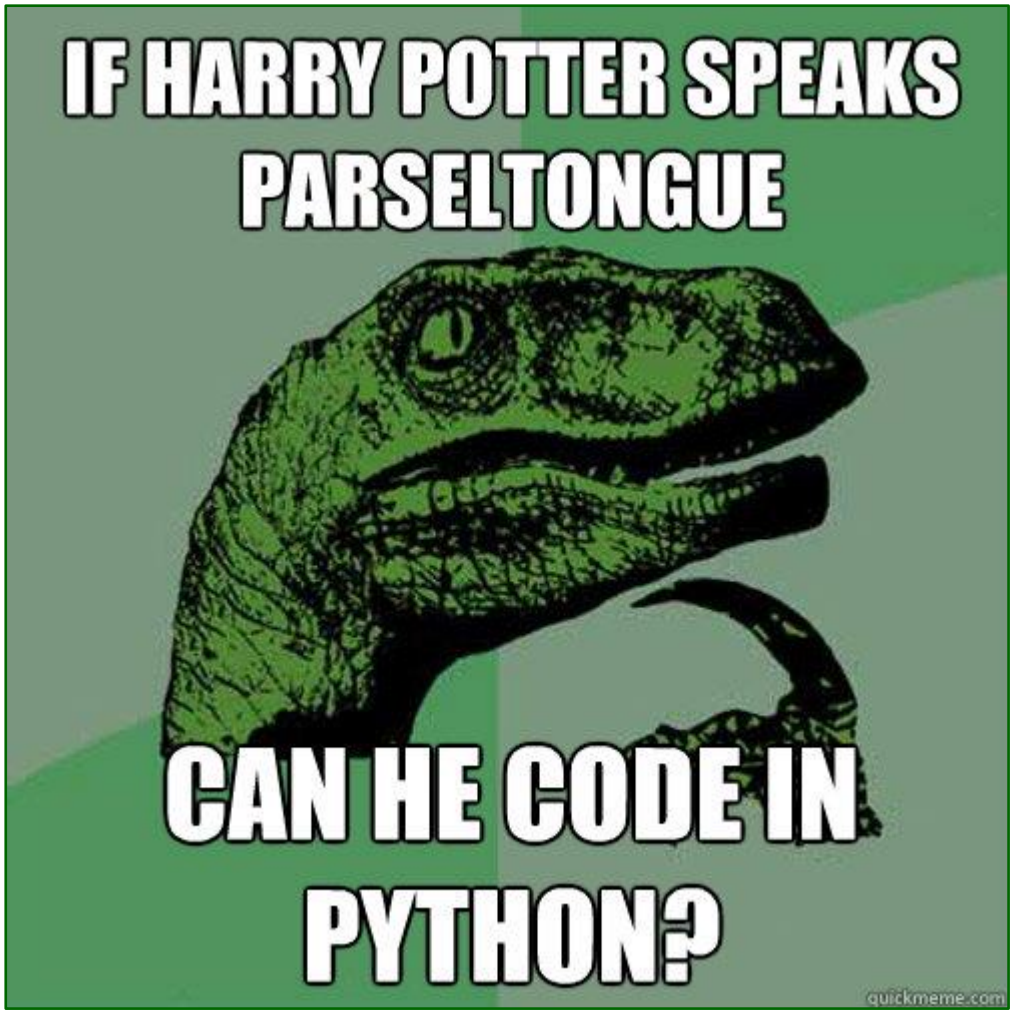
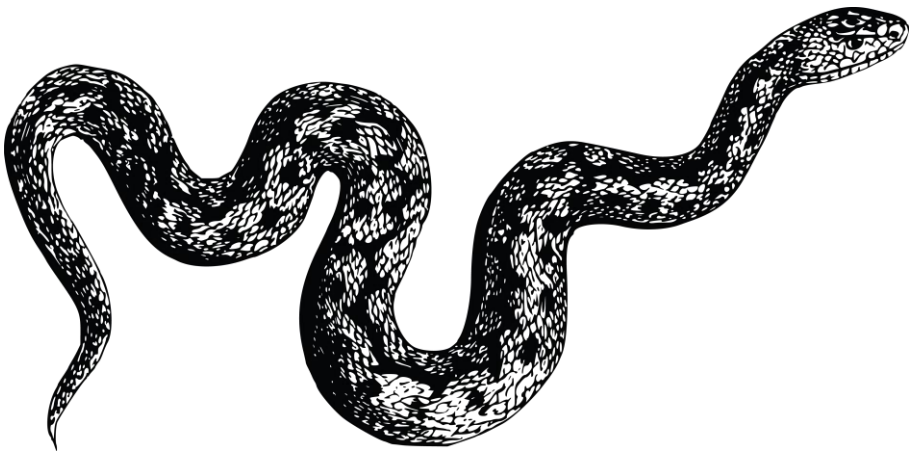


DEVOIR MAISON 3 PYTHON

Dossier explicatif du programme

GERBER Zoé 20183203
TIJANI Lobna 20181449
MARTIN Fanny 20181851
LEMERCIER Camille 20180595

Table des matières



I. Consigne

A partir d'une séquence nucléotidique, vous recherchez tous les ORFs dans les 3 phases directes, ainsi que celles sur le brin reverse complémentaire. Vous donnerez les coordonnées de tous les ORFs trouvés ainsi que leur séquence, leur longueur et leur traduction.

Notre séquence doit donc commencer avec ATG (codon START, c'est une méthionine AUG), se terminer par TAG ou TAA ou TGA (codon STOP) et doit tenir compte de la séquence à partir du premier caractère, puis la deuxième, puis de troisième.

Pour expliquer notre programme, nous ferons des captures d'écran du programme, encadrés en vert, et des captures d'écran de ce que renvoi le programme, encadré en orange.

II. Etapes

1) ADN

La première étape de notre programme est de demander à l'utilisateur d'entrer la séquence d'ADN, qu'on va appeler *seqADN,* qu'il va falloir transcrire puis traduire.

```
#On rentre tout d'abord une séquence nucleotidique(ADN);

seqADN = input("Entrez une séquence nucléotidique : ")
print("\n")
```

Pour la suite du déroulement de ce programme nous utiliserons la séquence d'ADN « CCCATGCAGTGCCATGGATGCTAACCAAATGAGATCGAATCGTATTAACGTAT GCGTTGA » comme séquence entré par l'utilisateur, pour les prochaines captures d'écran, dans le but de montrer le fonctionnement du programme. Il est évidemment possible de choisir n'importe quelle autre séquence nucléotidique, du moment qu'elle soit valide.

2) Validité

Comme dit précédemment on doit vérifier que notre séquence tapée est valide, c'est-à-dire qu'elle ne contient que des bases azotées de l'ADN (A pour Adénine, T pour Thymine, C pour Cytosine ou G pour Guanine) et non d'autres caractères. De plus, il est possible d'entrer la séquence en minuscule ou majuscule, ça sera pris en compte grâce à la méthode « .upper() ».

```
#On vérifie si la séquence donnée est valide :

seqADN = seqADN.upper()
def verif (adn):
    i=0
    erreur = 0
    for i in adn :
        if not (( i == 'A' ) or ( i == 'T' ) or ( i == 'C' ) or ( i == 'G' )):
            erreur = erreur + 1
    return (erreur)

erreur = verif(seqADN)
if (verif(seqADN) == 0):#il n'y a donc aucune erreur dans la sequence donnée.
     print (seqADN, "est valide \n")
else:
    print (seqADN , "n'est pas une sequence valide")
    print ("il y a ", erreur , "erreur(s) dans cette séquence \n")
```

Une fonction verif contenant un compteur d'erreur qui s'incrémente à chaque caractère non valide entré par l'utilisateur affichera le nombre d'erreurs exact que contient la séquence.

Si tout est ok, le programme est déroulé.

Par exemple, si j'entre un caractère non valable, voilà ce que le programme affiche :

```
Entrez une séquence nucléotidique : monty python


MONTY PYTHON n'est pas une sequence valide
il y a  10 erreur(s) dans cette séquence
```

3) ADN Complémentaire

Une fois la séquence d'ADN fixée par l'utilisateur, nous pouvons commencer à la manipuler.

Nous débutons donc par mettre en double brin notre simple brin, c'est-à-dire de lui donner sa séquence complémentaire, appelée *seqADNcompl*.

```
#Creation de la séquence de l'ADN complémentaire :

def seqADNc (adn):
    i=0
    seqADNcompl = ""
    for i in adn :
        if (i == "A"):
            seqADNcompl = seqADNcompl + "T"
        elif (i == "T" ):
            seqADNcompl = seqADNcompl + "A"
        elif (i == "C"):
            seqADNcompl = seqADNcompl + "G"
        else :
            seqADNcompl = seqADNcompl + "C"
    return (seqADNcompl)

#Séquence du brin complémentaire

seqADNcompl=seqADNc(seqADN)
if (verif(seqADN) == 0):#affiche seulement si la séquence d'ADN donnée est valide.
    print ("la sequence d'ADN complémentaire à  notre sequence de base est : ", seqADNcompl,"\n")
```

Une fonction seqADNc va parcourir, base par base, la séquence d'ADN et ajouter sa base complémentaire (A→T, C→G, T→A, G→C). Elle affiche ensuite la séquence complémentaire.

Séquence entrée par l'utilisateur : 5' ——————— 3'

Séquence complémentaire : 3' ——————— 5'

4) Reverse complémentaire

De la même manière, une fonction reverseC va parcourir, base par base, la séquence d'ADN complémentaire et ajouter les bases identiques, en ajoutant après la séquence, pour la « retourner » et revenir en 5' vers 3' : c'est le reverse complémentaire, appelée *seqADNrc*.

Séquence complémentaire : 3' ——————— 5'

Séquence reverse : 5' ——————— 3'

Elle affiche ensuite la séquence reverse complémentaire.

```
#Création de la séquence reverse complémentaire :

def reverseC(adn):
    i=0
    seqADNrc = ""
    for i in adn :
        if (i == "A"):
            seqADNrc = "A" + seqADNrc
        elif (i == "T" ):
            seqADNrc = "T" + seqADNrc
        elif (i == "C"):
            seqADNrc  = "C" + seqADNrc
        else :
            seqADNrc  = "G" + seqADNrc
    return (seqADNrc)

#séquence du brin reverse complémentaire:
seqADNrc = reverseC(seqADNcompl)
if (verif(seqADN) == 0):#affiche seulement si la séquence d'ADN donnée est valide.
    print ("la sequence d'ADN reverse complémentaire est : ", seqADNrc,"\n")
```

Nous avons donc d'affiché, à ce stade du programme :

```
Entrez une séquence nucléotidique : CCCATGCAGTGCCATGGATGCTAACCAAATGAGATCGAATCGTAT
TAACGTATGCGTTGA


CCCATGCAGTGCCATGGATGCTAACCAAATGAGATCGAATCGTATTAACGTATGCGTTGA est valide

la sequence d'ADN complémentaire à  notre sequence de base est :  GGGTACGTCACGGTA
CCTACGATTGGTTTACTCTAGCTTAGCATAATTGCATACGCAACT

la sequence d'ADN reverse complémentaire est :  TCAACGCATACGTTAATACGATTCGATCTCATT
TGGTTAGCATCCATGGCACTGCATGGG
```

5) Transcription

Une fois nos deux brins obtenus, nous allons les transcrire en ARN messager, c'est-à-dire, d'un point de vue écriture de la séquence, nous allons remplacer les Thymine (T) par des Uracile (U). On passe de la variable *seqADN* à *seqARN* / *seqARNrc*

Nous effectuons ce travail sur les deux brins avec une seule fonction transcription, que l'on applique ensuite sur chaque brin pour les afficher :

```
#Transcription de la sequence d'ADN en ARNmessager;

"""changer les T en U"""

def transcription (adn):
    i=0
    seqARN = ""
    for i in adn :
        if (i == "A"):
            seqARN = seqARN + "A"
        elif (i == "T" ):
            seqARN = seqARN + "U"
        elif (i == "C"):
            seqARN = seqARN + "C"
        else :
            seqARN = seqARN + "G"
    return (seqARN)

#séquence ARN du brin direct:

seqARN = transcription(seqADN)
if (verif(seqADN) == 0):#affiche seulement si la séquence d'ADN donnée au debut est valide.
    print ("la sequence d'ARN transcrite est : ", seqARN ,"\n")


#sequence ARN du brin reverse complémentaire:

SeqARNrc = transcription(seqADNrc)
if (verif(seqADN) == 0):#affiche seulement si la séquence d'ADN donnée au debut est valide.
    print("la sequence ARN a partir du reverse complementaire:",SeqARNrc,"\n \n\n")
```

6) Code génétique

L'objectif est de traduire l'ARN messager transcrit en une protéine (différente selon les cadres de lecture).

Pour cela on crée un dictionnaire, avec les acides aminés correspondants aux codons d'ARN.

On sait que 1 codon = 3 nucléotides

Donc pour faire par rang de 3 nucléotides par 3 nucléotides on utilise : range (...,...,3), car par défaut, range fait par pas de 1.

```
#Création du code génétique
#afin de permettre la traduction de l'ARNm

codeGene = {    "UUU" : "F",
                "UCU" : "S",
                "UAU" : "Y",
                "UGU" : "C",
                "UUC" : "F",
                "UCC" : "S",
                "UAC" : "Y",
                "UGC" : "C",
                "UUA" : "L",
                "UCA" : "S",
                "UAA" : "stop",
                "UGA":  "stop",
                "UUG":  "L",
                "UCG":  "S",
                "UAG":  "stop",
                "UGG":  "W",
                "CUU":  "L",
                "CCU":  "P",
                "CAU":  "H",
                "CGU":  "R",
                "CUC":  "L",
                "CCC":  "P",
                "CAC":  "H",
                "CGC":  "R",
                "CUA":  "L",
                "CCA":  "P",
                "CAA":  "Q",
                "CGA":  "R",
                "CUG":  "L",
                "CCG":  "P",
                "CAG":  "Q",
                "CGG":  "R",
                "AUU":  "I",
                "ACU":  "T",
                "AAU":  "N",
                "AGU":  "S",
```

```
"AUC":  "I",
"ACC":  "T",
"AAC":  "N",
"AGC":  "S",
"AUA":  "I",
"ACA":  "T",
"AAA":  "K",
"AGA":  "R",
"AUG":  "M",
"ACG":  "T",
"AAG":  "K",
"AGG":  "R",
"GUU":  "V",
"GCU":  "A",
"GAU":  "D",
"GGU":  "G",
"GUC":  "V",
"GCC":  "A",
"GAC":  "D",
"GGC":  "G",
"GUA":  "V",
"GCA":  "A",
"GAA":  "E",
"GGA":  "G",
"GUG":  "V",
"GCG":  "A",
"GAG":  "E",
"GGG":  "G",
```

Une fois le code génétique inséré, on peut commencer à traduire l'ARN en acides aminés assemblés en protéines

.

7) Traduction

La traduction se déroule selon 3 phases, donc les protéines seront différentes selon le cadre de lecture.

Avant de traduire il faut une méthionine, donc AUG, qui est le codon initiateur de la traduction, pour cela, on crée une fonction init qui permet de repérer le codon start selon le bon cadre de lecture, et ainsi lancer la traduction.

Cadre de lecture 1 : NNNNNNNNNNNNNNNNNNN

Cadre de lecture 2 : NNNNNNNNNNNNNNNNNNN

Cadre de lecture 3 : NNNNNNNNNNNNNNNNNNN

```
    #Phase1#

def init1(arn):
    i=0
    seqARNtrad1=""
    for i in range(0,len(arn),3):
        codon=arn[i:i+3]
        if (codon=="AUG"):
            seqARNtrad1= arn[i:]
            break
    return(seqARNtrad1)


    #Phase 2#

def init2(arn):
    i=0
    seqARNtrad2=""
    for i in range(1,len(arn),3):
        codon=arn[i:i+3]
        if (codon=="AUG"):
            seqARNtrad2= arn[i:]
            break
    return (seqARNtrad2)


    #Phase 3#

def init3(arn):
    i=0
    seqARNtrad3=""
    for i in range(2,len(arn),3):
        codon=arn[i:i+3]
        if (codon=="AUG"):
            seqARNtrad3= arn[i:]
            break
    return (seqARNtrad3)
```

Une fois la traduction initiée, elle peut se dérouler grâce au code génétique, et s'interrompre lors d'une rencontre avec le codon stop, toujours dans le bon cadre de lecture.

```
#création fonctions fin ORF, dans la phase 1(a partir de la sequence qui commence par un ATG):

def FinSeq(arn):
    i=0
    SeqStop1=""
    for i in range(0,len(arn),3):
        codon=arn[i:i+3]
        if ((codon=="UAG")or (codon=="UGA")or(codon=="UAA")):
            SeqStop1=arn[:i]
            break
        elif ((codon!="UAG")or (codon!="UGA")or(codon!="UAA")):
            SeqStop1=arn
    return (SeqStop1)
```

La fonction tradC se charge de l'élongation de la traduction. La variable *prot* est donc complétée au fil de la traduction.

De plus on a defini plus la variable *codon* de taille 3.

```
#Création de la fonction qui permet la traduction des séquences qui commence par un AUG :

def tradC(arn):
    i=0
    prot = ""
    for i in range(0,len(arn)-taille+1,3) :
        codon = arn[i:i+taille]
        prot = prot + codeGene[codon]
        if ("stop" in prot):
            break
    return (prot)
```

Une fois la protéine entière, on s'interresse au coordonnées des ORFs. Pour cela, on ajoute debutORF1,2 et 3 , qui sont 3 fonctions qui donnent les coordonnées des méthionines, donc du début des ORFs.

Après ça ; on réalise un appel global de nos fonctions pour que l'affichage final soit bien organisé.

```
def debutORF1(arn):
    i=0
    debut1=0
    for i in range(0,len(arn),3):
        codon=arn[i:i+3]
        if (codon=="AUG"):
            debut1= i+1#car le i correspond a un chiffre en dessous de celui voulu.
            break
        elif(codon!="AUG"):
            debut1=0
    return (debut1)


    #Phase 2


def debutORF2(arn):
    i=0
    debut2=0
    for i in range(1,len(arn),3):
        codon=arn[i:i+3]
        if (codon=="AUG"):
            debut2= i+1#car le i correspond a un chiffre en dessous de celui voulu.
            break
        elif(codon!="AUG"):
            debut2=0
    return (debut2)

    #Phase3

def debutORF3(arn):
    i=0
    debut3=0
    for i in range(2,len(arn),3):
        codon=arn[i:i+3]
        if (codon=="AUG"):
            debut3= i+1#car le i correspond a un chiffre en dessous de celui voulu.
            break
        elif(codon!="AUG"):
            debut3=0
    return (debut3)
```

seqARN1/2/3 correspondent aux séquences qui commencent par le codon start, et sont créées grâce à la fonction init1/2/3, comme vu précédemment selon les trois phases.

Ces séquences créées sont ensuite utilisées dans la fonction FinSeq, qui permet de créer les séquences *seqARNtrad1/2/3*, correspondants aux séquences à traduire (elles commencent par un codon start et finissent avant le codon stop).

Les *debut1/2/3* quant à elles, correspondent au numéro du A dans le codon AUG, ce qui donne le début de l'ORF. La fonction debutORF, est la même que la fonction init à la différence qu'elle retourne le i (coordonnée de la lettre A) à la place de la séquence qui commence après le AUG.

Les *fin1/2/3*, correspondent à la taille des séquences à traduire, ce qui renvoie le numéro du dernier nucléotide de la séquence, et donc les coordonnées de la fin de l'ORF (attention il faut additionner au résultat le début, car la séquence utilisée ne possède pas ce qui précède le codon start, on ajoute donc *debut1/2/3* au résultat final)

C'est le même principe pour les séquences sur l'ADN reverse complémentaire.

```
#appel des fonctions pour sequences a traduire
""" création des sequences qui commence par un AUG dans les trois phases du brin direct"""

seqARN1 = init1(seqARN)#phase1
seqARN2 = init2(seqARN)#phase2
seqARN3 = init3(seqARN)#phase3

""" création des séquences  a traduire, qui commence a partir du codon start et qui finisse avant le codon stop dans les trois phases """

seqARNtrad1=FinSeq(seqARN1)#phase1
seqARNtrad2=FinSeq(seqARN2)#phase2
seqARNtrad3=FinSeq(seqARN3)#phase3
```

```
#appel des fonctions pour debut ORF

"""permet de ressortir les coordonnées du A du codon start """

debut1=0
debut1=debutORF1(seqARN)#phase1
debut2=0
debut2=debutORF2(seqARN)#phase2
debut3=0
debut3=debutORF3(seqARN)#phase3

#appel des fonctions pour fin ORF:
""" permet de recupérer les coordonnées du dernier nucleotides avant le codon stop trouvé dans les trois phases différentes
calcul de la taille de la sequence a traduire, on y ajoutera le nombre i, coordonnée du A afin de rajouter le debut de la séquence que nous avons precedement enlevé.""

fin1=0
fin1=len(seqARNtrad1)#phase1
fin2=0
fin2=len(seqARNtrad2)#phase2
fin3=0
fin3=len(seqARNtrad3)#phase3
```

On peut donc commencer à traduire et afficher les coordonnées et les protéines (selon les phases).

Pour la traduction de la première protéine, par exemple :

```
if (verif(seqADN) == 0):#affiche seulement si la sequence d'ADN donnée est valide.
    print("-->les proteines sur l'ARNm\n \n")

    #Proteine1

    prot1 = tradC(seqARNtrad1)

    print("-Proteine en phase 1:\n")
    if len(prot1)==0:
        print("il n'y a pas de codon start dans la phase 1 du brin direct donc, pas de proteine dans cette phase.")

    elif len(prot1)!=0:
        print("sequence à traduire en phase 1 :",(seqARNtrad1))
        print ("La protéine 1 traduite selon le 1er ORF est : " , prot1)
        print ("La longueur de la proteine 1 est de ",len(prot1), "acide(s) aminé(s)")
        if ((fin1+debut1-1)!=len(seqARN)):
            print(" le debut de cet ORF est au ",debut1,"ème nucléotide, et la fin au ",(fin1+debut1-1), "ème nucléotide")
        else:
            print(" le debut de cet ORF est au ",debut1,"ème nucléotide, et la fin au ",len(seqADN), "ème nucléotide, (la fin de la sequence, il n'y a pas de codon sto

    print("\n")
```

Ce qui affichera :

```
-Proteine en phase 1:

sequence à traduire en phase 1 : AUGCAGUGCCAUGGAUGC
La protéine 1 traduite selon le 1er ORF est :  MQCHGC
La longueur de la proteine 1 est de  6 acide(s) aminé(s)
 le debut de cet ORF est au  4 ème nucléotide, et la fin au  21 ème nucléotide
```

Comme dit plus haut, on réalise le même principe pour le reverse complémentaire, et donc 3 autres protéines différentes seront obtenues.

```
-->les proteines sur l'ARNm du reverse complémentaire:


-Proteine en phase 1:

sequence à traduire en phase 1 : AUGGCACUGCAUGGG
La protéine 1 traduite selon le 1er ORF est :  MALHG
La longueur de la proteine 1 est de  5 acide(s) aminé(s)
 le debut de cet ORF est au  46 ème nucléotide, et la fin au  60 ème nucléotide, (la fin de la sequence, il n'y a pas de codon stop)


-Proteine en phase 2:

sequence à traduire en phase 2 : AUGGG
La protéine 2 traduite selon le 2eme ORF est :  M
La longueur de la proteine 2 est de  1 acide(s) aminé(s)
 le debut de cet ORF est au  56 ème nucléotide, et la fin au  60 ème nucléotide, (la fin de la sequence, il n'y a pas de codon stop)


-Proteine en phase 3:

il n'y a pas de codon start dans la phase 3 du brin reverse complémentaire donc, pas de proteine dans cette phase.
```

III. Conclusion

L'objectif de ce programme était donc, à partir d'une séquence simple brin quelconque d'ADN, produire des séquences protéiques associées, en passant d'un simple brin à un double brin, puis en parcourant ces séquences selon les 3 cadres de lectures définis en biologie.