**Outline of your GeneFinder file:**

(Order of how your functions are placed matters)

from dna import \*

from load import \*

import random

restOfORF(DNA)

oneFrameV2(DNA)

longestORFV2(DNA)

longestORFBothStrands( DNA )

collapse(L)

longestORFNoncoding(DNA, numReps)

findORFs(DNA)

indORFsBothStrands(DNA)

getCoordinates(gene,DNA)

geneFinder(DNA,minLen)

printGenes(geneL)

seq=loadSeq("X73525.fa")

#seq=loadSeq("salDNA.fa")

#print(test(seq))

#gene = geneFinder(seq,1000)

#printGenes(gene)

**Summary of your midterm instructions, color coding the functions, hope it helps connect things better!**

X73525 = loadSeq("X73525.fa")

geneList = geneFinder(X73525, minLen)

geneList = geneFinder(X73525)

printGenes(geneList)

print the results

Before running geneFinder.py you need to:

* determine you minLen value
* find the longest ORF we see in the non-coding sequence
* define ORF’s longer that this longest ORF as potential genes

Run, longestORFNoncoding( X73525,1500)

1500 is the number of random DNA sequences that the code will test

After finding potential genes we blast them to find their purpose

**What are all the functions needed in geneFinder?**

**Functions required for geneFinder.py**

geneFinder function should:

* identify ORF’s longer than the minLen
* return a list with information about each

geneFinder should do this,

1. first call findORFsBothStrands to obtain a list of ORFs in the input DNA
2. Then, it should run through this list, keeping only those ORFs which are longer than minLen
3. For each ORF that is long enough, geneFinder should calculate

* The start and end positions of the ORF in DNA using getCoordinates
* The protein sequence of the ORF using codingStrandToAA

These calculations should then be placed in a list:

[beginningCoord, endCoord, proteinSequence] # geneFinder output

(There will be a list like this for every ORF that is long enough)

4.We then have to collect these lists in another list (a list of lists)

5.Final step, sort this list of lists before returning it.

The name of our final list of lists is finalOutputList.

We can sort this list by starting coordinate and return like:

finalOutputList.sort()

return finalOutputList

**Details on getCoordinates**

getCoordinates(orf,DNA) returns the start and end coordinates of an ORF in DNA.

first consider the case where the ORF is in the forward strand.

In this case we get the start coordinates as follows:

>>> testDNA="ACGTTCGA"

>>> testORF="GTT"

>>> testDNA.find(testORF)

2

Now we get the end coordinates by adding the length of the ORF to the start coordinate

When we search and a sequence is not present, we search by using .find.

>>> testDNA="ACGTTCGA"

>>> testORF="GAA"

>>> testDNA.find(testORF)

-1

Why the negative one? The sequence is not present. We need to search the reverse complement of orf.

Kinda looks like this:

>>> testDNA="ACGTTCGA"

>>> revCompTestORF=reverseComplement("GAA")

>>> testDNA.find(revCompTestORF)

3

Here are some examples of getCoordinates:

>>> getCoordinates("GTT", "ACGTTCGA")

[2, 5]

>>> getCoordinates("CGAA", "ACGTTCGA")

[3, 7]

**Details on findORFs and findORFsBothStrands**

findORFs(DNA) identifies the ORFs in the unshuffled DNA and returns them as a list.

If it found 0 ORFs it should return an empty list.

Easy task, why?

.findORFs calls oneFrameV2 in each of the three possible reading frames of the sequence

It’s then supposed to combine all the ORFs found in each frame and return them.

>>> findORFs("ATGGGATGAATTAACCATGCCCTAA")

['ATGGGA', 'ATGCCC', 'ATGAAT']

>>> findORFs("GGAGTAAGGGGG")

[]

Write a function called findORFsBothStrands( DNA ) that:

* searches both the forward and reverse complement strands for ORFs
* returns a list with all the ORFs found.

For example:

>>> findORFsBothStrands('ATGAAACAT')

['ATGAAACAT', 'ATGTTTCAT']

OneFrameV2

>>> oneFrameV2("ATGCCCATGGGGAAATTTTGACCC")

['ATGCCCATGGGGAAATTT']

For our gene-finder application we want it to only return the largest ORF from a set of nested ORFs like this.

How can we write oneFrameV2?

* search through the string until you find the first ATG
* then we call restOfORF to get the ORF.
* When you search for the next ATG, skip and look after the end of the ORF just found. While loop is convenient here.

Finally,

create a longestORFV2(DNA) fxn that calls oneFrameV2.

**Details on longest ORFNoncoding**

Look for larger open reading frames

ORF: a stretch of sequence between a start codon and the next in frame stop codon.

What are we doing, looking for genes:

identify it’s longestORFs

to see if these are genes, we can

look at many non-coding regions

find a distribution of open reading frame lengths for non-coding DNA:

find lengths of ORF’s in non-coding DNA

compare the longest ORF’s in our test sequence, the lengths in ours.

If the length of the ORF’s in our test sequence are longer than these are likely to be genes

**Approach :**

Look at Salmonella sequence and identify ORFs with known non-coding sequences

To get non-coding sequences, we make them randomly by shuffling our sequences.

To assess weather long ORFs are genes

generate re-shuffled sequences, same length as our test DNA

measure the max ORF length in each

Then ask,

From our reshuffles, Is the very longest ORF among these still shorter than some ORFs we observe in our real DNA?

If our real DNA ORFs are longer then the ORFs in the reshuffles, then that suggest those ORF’s are genes in the real DNA

longestORFNoncoding: makes reshuffles, finds the very longest ORF among them and returns the length, a number.

Generate reshuffles

How do we do this?

1. Take DNA String and turn it into a list, using a built in function list.

list (string) and returns list of symbols