***Path218 Post-Discussion 3 and Homework 3***

**A quick review of Dictionaries, Functions, Modules, and some VSG commands:**

A **Dictionary** is similar to a list, but where a list is indexed, or “keyed” by an integer, a dictionary can be keyed by other things, such as strings.

Dictionaries are not ordered like lists, but you can always generate an ordered list of keys (e.g., for looping through) with **sorted(D.keys())**

• **D={}** creates an empty dictionary.

• **D={‘key1’: 1, ‘key2’:2, ‘key3’:3}** creates a populated dictionary. D[‘key1’] will return 1.

• D[‘name’] = ‘entry’ adds the value ‘entry’ to the dictionary, under the key ‘name’.

• del D[‘name’] will delete the value ‘name’ from the dictionary.

• D.keys() returns a list of all the keys in D

• D.values() returns a list of all the values in D.

• D.has\_key(‘key1’) or key1 in D checks whether key1 is a key in D.

• for key in D: loops through all the keys in D

Functions:

• def foo(input1, input2): defines a function called foo that takes input1 and input2

• a return output command in the function specifies the output

• multiple outputs can be returned as a sequence (e.g. return output1, output2)

• from goo import foo imports the function foo from the module or file goo

• **import goo** imports all of the functions in a module or file goo

**The VSG Module:**

**(1) Import VSG\_Module: from VSG\_Module import \*** <VSG\_Module.py needs to be in same folder as your script>

**(2) Use** vline(), vrect(), vcircle(), vtext() with inputs to specify position, size, and color of shapes to draw:

Inputs that can be used to specify position: x1=*left edge*, x2=*right edge*, y1=*bottom edge*, y2=*top edge*, xc=*h-center*, yc=*v-center*

Inputs that can be used to specify size: r=*radius*, height=*height*, width=*width*, strokewidth=*line/border width*

Inputs that can be used to specify color:fill=*Color*, stroke=*Color***.** Some *Color* choices: black,blue,green,red,orange,gray,yellow

Example of creating a line: vline(x1=0, y1=0, x2=80, y2=80, stroke=red, strokewidth=10)

Example of creating a circle: vcircle(xc=20, yc=20, r=9, stroke=red, fill=black, strokewidth=3)

Example of creating a text: vtext(text='foo', xc=50, yc=15, font='times 12 Bold', stroke=blue)

**(3) Show drawing:** vdisplay()

**HW3 consists of Q1cd, Q2bcd, and Q3 below, and a request for a brief project description (Q4).**

**You're welcome to skip one of the first three problems (Q1,Q2,Q3) if time is limiting**

**But please provide at least a basic (non-binding) idea of what you might do as a project (Q4).**

**Due Monday 07/21/15@9:54pm. Rename as YourName\_PS3.doc and send to path218homework@gmail.com**

(Problems 1a, 1b and 2a were covered in section with some [slightly cleaned up] solutions provided below as templates for the remaining problems.)

**Question 1. Making a faster antisense routine**

**Start: an antisense program similar to last week's, but designed to work on long sequences**

from time import clock

F=open('ColiDH1.fa',mode='rU')

S=''

for L in F:

if L[0]!='>':

S+=L.strip()

F.close()

aS=''

counter=0

for c in S:

if c=='A':

aS='T'+aS

if c=='T':

aS='A'+aS

if c=='C':

aS='G'+aS

if c=='G':

aS='C'+aS

counter+=1

if counter % 100000==0:

print('base='+str(counter)+'\t'+'time='+str(clock()))

**1a1.** Examine time per operation as the loop progresses... how are we doing? **Slow**

**1a2.** A lot of typing involved with the "if" statements. Can we make this more concise with a dictionary?

from time import clock

F=open('ColiDH1.fa',mode='rU')

S=''

for L in F:

if L[0]!='>':

S+=L.strip()

F.close()

aS=''

counter=0

d={'A':'T','T':'A','G':'C','C':'G'}

for c in S:

if c in d:

aS=aS+d[c]

counter+=1

if counter % 50000==0:

print('base='+str(counter)+'\t'+'time='+str(clock()))

**1b.** Maybe the program is so slow because of the need to add each base to position zero of the aS string. Can we make this work by adding each base at the end, then reverse the string once this is complete?

from time import clock

F=open('ColiDH1.fa',mode='rU')

S=''

for L in F:

if L[0]!='>':

S+=L.strip()

F.close()

aS=''

counter=0

d={'A':'T','T':'A','G':'C','C':'G'}

for c in S:

if c in d:

aS=aS+d[c]

counter+=1

if counter % 50000==0:

print('base='+str(counter)+'\t'+'time='+str(clock()))

aS=aS[::-1] ## [::-1] returns every element in a string/list jumping by -1, quickly reversing the order

print('finished'+'\t'+'time='+str(clock()))

**1c.**  Another approach to a quick antisense transformation is to use the built in "replace" method that is an option for any given string. As we've done before, **S.replace('b','f')** returns a derivative of string S in which all "b" characters have been replaced by "f". You can make the Antisense operation even faster by using "Replace" methods.

Paste Your Code Here

Paste Your Output Here

**1d (optional).**  Anyone wanting to try for an even faster Antisense routine (in pure python) is welcome.

Paste Your Code Here

Paste Your Output Here

**Question 2. Looking for genes in E. coli**

**2a.** Of the 64 possible base triplets, three triplets (ATG, GTG, and TTG) predominantly specify initiation and three (TAA,TGA, and TAG) specify chain termination. We'll start by making a list of ORFS for the sense strand

**2ai.** Generate a list of all open reading frames (ORFs) in the first 100000 of E. coli.

F=open('ColiDH1.fa',mode='rU') ## Our Now-Standard code to read the Coli sequence into an array **S**

S=''

for L in F:

if L[0]!='>':

S+=L.strip()

F.close()

starts=['ATG','GTG','TTG'] ## The three common start triplets

stops=['TAA','TGA','TAG'] ## The three stop triplets

for i in range(100000): ## Iterate through the first 100000 bases of the sequence

if S[i:i+3] in starts: ## S[i:i+3] is the triplet starting at base i, if this triplet is in **starts**, then an ORF is (potentially) starting

for j in range(i,100000,3): ## This range() statement generates a list, starting at i, running to 100000, and counting by 3s

if S[j:j+3] in stops: ## The first j in that list where S[j:j+3] gives a stop codon will end the open reading frame

print(i,j) ## Print the starts and stops

break ## Also we need to break out of the "for j" loop, since the first stop will end the ORF

**2aii.** A problem is that some "apparent" ORFs may share the same stop codon (with the shorter ORFs being fragments of the larger). Make a list where only the largest ORF for each stop codon is listed. We'll also limit ourseleves to ORFs above 500base pairs.

• The need here is a way to keep track of what stop sites have already been used. One way to do this is with a dictionary, where stop sites are the keys and start sites the values. If a given stop site has already been "taken" with an earlier start site, it will turn up as "in" the dictionary. Here is some code

F=open('ColiDH1.fa',mode='rU')

S=''

for L in F:

if L[0]!='>':

S+=L.strip()

F.close()

starts=['ATG','GTG','TTG']

stops=['TAA','TGA','TAG']

ORFs={} ## Sets up a dictionary to hold pairs of stop codons (keys) and start sites (values) for sense open reading frame

## We could also use a pair of lists for this: one list being all (start,stop) pairs and one being all stop sites. But the Dictionary is faster to use.

for i in range(100000):

if S[i:i+3] in starts:

for j in range(i,100000,3):

if S[j:j+3] in stops:

if not(j in ORFs) and j-i>500: ##Only record the earliest start site 'i' for each stop site j, also require j-i>500

ORFs[j]=i ##If these conditions are met, enter key,value pair {j:i} into dictionary

break

for j in sorted(ORFs.keys()): ## Loop to print results: First, Iterate through every key in the dictionary

i=ORFs[j] ## Recover the start site, Given the key (stop site)

f=i%3 ## A "division by 3 remainder" operation with i gives the reading frame

print('Start='+str(i)+' Stop='+str(j)+' Frame=+'+str(f))

**Result**

Start=244 Stop=2797 Frame=+1

Start=2801 Stop=3731 Frame=+2

Start=3734 Stop=5018 Frame=+2

Start=7264 Stop=7771 Frame=+1

Start=8175 Stop=9189 Frame=+0

**etc...**

**2aiii.**Another use of the dictionary is to draw a map. We'll use the "community" VSG Module (introduced on Tuesday) to make the drawing happen.

F=open('ColiDH1.fa',mode='rU')

S=''

for L in F:

if L[0]!='>':

S+=L.strip()

F.close()

starts=['ATG','GTG','TTG']

stops=['TAA','TGA','TAG']

ORFs={}

for i in range(100000):

if S[i:i+3] in starts:

for j in range(i,100000,3):

if S[j:j+3] in stops:

if not(j in ORFs) and j-i>500:

ORFs[j]=i

break

from VSG\_Module import \* ## Import the **VSG** module (which we'll use to make a graphic of ORFs in each frame)

for j in sorted(ORFs.keys()):

i=ORFs[j]

frame=i%3

vline(x1=i/100,x2=j/100,y1=frame\*10,y2=frame\*10,strokewidth=10,stroke=blue)

## Horizontal line from x=i to x=j (scale 1:100, so all x values divided by 100). Vertical position based on reading frame, with y=10\*frame

vdisplay() ## Display the drawing in the browser

Result: (Pasted in as a screenshot):



We can also add a bit to the code to draw a simple scale with numbers and a "main" genome line

(see Tuesday's lecture notes for more info on VSG's vline() and vtext() methods)

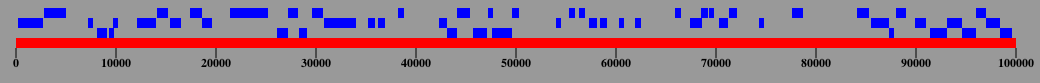
vline(x1=0,x2=1000,y1=-10,y2=-10,strokewidth=10,stroke=red)##Simple red line representing 0-100kb of DH1 (1:100 scale)

for j in range(0,100001,10000): ## Loop through multiples of 10000 to make a very primitive scale bar

vtext(text=str(j),xc=j/100,yc=-30,font="times 12 bold",stroke=black) ## Text element (sequence numbers)

vline(x1=j/100,x2=j/100,y1=-25,y2=-15,stroke=black,strokewidth=1) ## A little tick-line above the sequence number

vdisplay() ## Display the drawing in the browser



**2b.** Make an equivalent lists of **antisense** ORFs >500b in the first 100000 bp of the E. coli genome. As with 2aii above, output the **start** position of the ORF, the **end** of the ORF (a smaller number than the start) and the **frame** (which will be -0, -1, or -2 depending on whether the starting base is a multiple of 3 (-0 frame), 3n-1 (-1 frame), or 3n-2 (-2 frame). Format your result as in **2aii** above.

Paste Your Code Here

Paste Your Output Here

**2c.** Generate a graphic map similar to that above, but adding lines that represent the coding regions in the three antisense reading frames below the red line and numbers (you can use green for the antisense ORFs).

Paste Your Code Here

Paste a screen-grab of your output here:

**2d.** Make a list of codons used in the open reading frames identified in 2a+2b with the number of times each codon is used. Format your list as a series of lines with first the codon (e.g., 'TTG') followed by a tab ('\t') followed by the total number of instances of that codon.

Paste Your Code Here

Paste Your Output Here

**Question 3.** **Find the most important things that Shakespeare ever said**

Great writers use several techniques (sometimes called "weapons") for extreme emphasis:

(1) A short message.

(2) A first word in all capitals,

(3) An exclamation point at the end of the phrase, and

(4) Repeating the phrase more than once.

Using the Shakespeare.txt file distributed last week, print all groups of five consecutive words that (i) start with a word in all capitals, (ii) end with an exclamation point, and (iii) appear at least twice in Shakespeare.txt.

Here are some hints:

• Convert the file into a list of words

• Make an empty dictionary D where each key (a phrase of five consecutive words) will be associated with a values (the number of times that phrase appears in Shakespeare.txt).

• Go through the WordList starting from each position x (need to stop five before the end)

• Check condition that WordList[x] be in upper case: WordList[x]== WordList[x].upper()

• Check condition that the fifth word end with '!': WordList[x+4][-1]=='!'

• Make a phrase string ( w5=str(word[x:x+5]) )

• If w5 is not already in D, add it and set the initial value to zero

• Increment the value D[w5]

• Once the dictionary is finished, go through it key by key [for w5 in D:], printing all w5 for which D[w5]>1.

Paste Your Code Here

Paste Your Output Here

**(Pseudo)Question 4.** Tell us a bit about what you propose as a potential class project. Propose something that you think would be useful and interesting. Although you needn't be constrained by any specific project design, we'd suggest that the proposal involve the following four elements:

(i) Quickly describe the dataset you'll be working with (can be public, part of your current work or your lab's work, or anything else which you can get access to). Maximum three sentences here.

(ii) A sample (if available) {Sequences: provide a few lines of DNA sequence from whatever source you'll be working with, Images: paste a screenshot of an example image that you might use below, other data types, just paste a few lines of whatever the data is).

(iii) A parameter that you hope to measure from the dataset, or question that you propose to address about (or from) the dataset.

(iv) A preview (not binding) on how what the final result/product/visualization from the proposed project might look like. Can be a quick sketch, drawing, bit of descriptive text, pseudodata, etc.

Note that the pre-proposal isn't binding (indeed we'll chime in with any suggestions if it seems overly ambitious, etc), but should be a useful starting point for thinking about this.