**Problem set philosophy for the Path 218.**

0. Don't panic *(citation: Douglas Adams... "A Hitchhikers Guide to the Galaxy)*

1. The goal of this problem set is to provide practice in running and modification of programs in the IDLE environment, with an intro to some string manipulations useful for sequence and other data.

2. If you fully succeed in whatever time you allot to this, wonderful (and tell us the answers)

3. If you don't fully succeed in whatever time you allot, send us what you have, pasting answers and code into a copy of this document as far as you get. If you're code isn't working, add a few comment lines explaining what you think is going wrong (you can also paste any 'error' statement after the code.

4. The problems are designed to be addressed using coding tools from the section on June 23th and the subsequent handout. But if you are familiar or want to use other *Pythonic* tools not covered in the lectures/handouts/sections, please feel free to use these (there are many solutions to any problem with Python... we're happy with any of them).

5. This homework will be due at 8:14pm on Monday, July 6th. Email your homework (as a modified version of this file) to **path218homework@gmail.com. Please title the file as follows: "YourName\_PS#.docx"**

6. **You don't need** to code alone (unless you want to code alone):  We'll be reserving the Genetics Library (M315) Friday afternoons 1-2PM as an arbitrary assembly point for shared efforts in Pythonic coding. An instructor will be around for most of the first hour. This is experimental-- we'll keep it going if there's continued interest.

**Problem Set 1 starts with fibonacci and "fastQ" programs from 6/23 sections, adding "twists".**

**Problem 1:**

**Initial code:**

## Fibonacci.py Start Code

## Two variables, Last is the Last number on the list, SecondToLast is the Second-To-Last number

Last = 1

SecondToLast = 1

while True: ## keep repeating forever until user stops program

newLast = Last + SecondToLast ## New number to add to list: sum of previous "last two" numbers

SecondToLast = Last ## Previous Last number on list becomes second to last

Last = newLast ## Now you can set the Last number on list to newLast

print(Last) ## Print the number

## stop program by typing control-c or pulling "Restart Shell" menu item down from "Shell" menu

**Questions:**

**1a.** The Fibonacci series involves addition of the most recent two numbers to obtain a new number to append to the series. Modify the code: instead of adding the last two numbers to get the next number, subtract the last two numbers (subtracting the second-to-last integer from the last integer) to get a new value. Check your code (by eye) to see if it is really doing what you expect to get each successive number.

Your code:

Last = int(raw\_input('Please input the last number:'))

SecondtoLast = int(raw\_input('please input another number:'))

Repeats = int(raw\_input('Please input how many numbers you want to appear in the Finonpnacci:'))

i = 0

for i in range(Repeats):

NewLast = Last - SecondtoLast

SecondtoLast = Last

Last = NewLast

print Last

i += 1

Do you note anything repetitive about the result? \_\_\_\_\_\_Repeat every six numbers; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Goals 1a: (i) Make sure everyone can copy original program from Word document, paste into IDLE, and run the code. (ii) Practice in reading existing code: figure out what statement tells Python to add the previous two numbers to generate the new list element (iii) Very simple coding change: switch this statement to instruct python to do a subtraction instead, (iv) Observe the result.

**1b.** Keeping the subtraction, change the initial values (Last and SecondToLast) [choose any integers for these]

Initial Value of variable "**Last**": \_\_1\_\_\_

Initial Value of variable "**SecondToLast**": \_\_2\_\_\_

Your code:

If the same kind of repetitive pattern present, what is the period? \_\_6\_\_\_\_\_\_\_\_

Goals 1b: (i) Practice in reading code to figure out where the initial values of "Last" and "SecondToLast" are set. (ii) Some fun math in figuring out what happens if these values are changed

**1c.** Keeping the subtraction, Rewrite the code again to calculate the 60th and 600'th element in the series

Use whatever values you wish as the initial values for **Last** and **SecondToLast**

Your code:

Element 60 in series:

Element 600 in series:

Goals 1c: (i) Practice in initializing a new variable (in this case a variable that will start at zero before the "While" loop and count the number of times a new value is added to the list.) (ii) Practice in modifying the new variable every time the loop is executed (by adding 1), (iii) Practice in adding an "if" statement to only print the result if the counting variable is equal to a preset value.

**Problem 2**. Recalling the structure of the FastQ file we discussed on Tuesday, every fourth line is a string that represents a small RNA read from a biological sample. The small RNAs were captured by ligation of linkers to the 5' and 3' ends of the RNAs. Once sequenced, the prediction is that all reads will start with the barcode GCAG, followed by a small RNA sequence of generally 15-35bases, followed by sequences from a standard linker (CACTCGGGCACCAAGGAC). The sequences in between these two are derived from a variety of small RNAs captured from the sample. The reads are 36 bases, so often the standard linker is cut off before finishing.

Our Code from this week (cleaned up):

## LineStarts.py: Return counts of first base frequencies in a sequencer output file.

File1=open('fastQfile.txt',mode='rU') ## A fastQ file with a sequence reads and some 'junk' lines

Astarts=0 ## Number of sequences starting with "A" initial value zero

Gstarts=0

Cstarts=0

Tstarts=0

Counter=0

**for** line1 **in** File1: ## Go through each line in the file, with that line put into variable 'line1'

**if** Counter==1:

**if** line1[4]=='A': ## Execute the following statement only if fifth character is "A"

Astarts=Astarts+1 ## We check the fifth character, since the first four were very uniform

**if** line1[4]=='C': ## And we realized (after a bit of excitement) that these are from a linker

Cstarts=Cstarts+1

**if** line1[4]=='G':

Gstarts=Gstarts+1

**if** line1[4]=='T':

Tstarts=Tstarts+1

Counter=Counter+1

if Counter==4:

Counter=0

**print**("Astarts="+str(Astarts)) ##Print "AStarts=" + string from variable AStarts

**print**("Cstarts="+str(Cstarts))

**print**("Gstarts="+str(Gstarts))

**print**("Tstarts="+str(Tstarts))

File1.close() ## close the file

We are interested in just one micro RNA, mir-81, with a published consensus sequence ugagaucaucgugaaagcuagu.

**Problem 2a.**  Modifying our code above, count the lines that have a complete and correct copy of mir-81 situated as expected in a read that contains (a) the barcode (GCAG), immediately followed by (ii) mir-81 sequence, immediately followed by (iii) as much sequence of the linker (CACTCGGGCACCAAGGAC) as will fit in a 36 base read.

Your code:

Number of perfect mir-81, perfect barcode, perfect linker matches:

Goals 2a: (i) Make sure everyone can copy original program from Word document, paste into IDLE, and run the code, this will also require putting fastQfile.txt data file in the same folder/directory as the python script. (ii) A bit of a puzzle, based on the Barcode sequence (GCAG), the mir-81 sequence, and the 3' linker, in figuring out exactly what the first 36 bases of a "perfectly barcoded and linkered" capture of mir-81 would look like. (iii) Figuring out where in the code to define the new integer variable that will hold the count of perfect matches, (iv) Figuring out where in the code you can check for equality between the current data line and the predicted sequence (v) Using the .strip() function to remove any trailing returns or space from the data line, (vi) Adding an "if" statement to test for equality between the data line and your prediction, (vii) if equal, incrementing the count of perfect matches, (viii) Figuring out where in the code you can print the number of perfect matches and how to ensure the it's clear in the output that this is what you are looking for

**2b.**  Barcode and linker primers are not always perfect. Print a list of lines that contain mir-81 but don't start with the expected 5' linker GCAG.

Your code:

Number of perfect mir-81 matches that don't have the expected 5' linker 'GCAG' :

A list of the linker-broken mir-81 isolates (paste all here):

Goals 2b: (i) Going back to the original script. (ii) figuring out where in the code you can check for a barcode on the current line, (ii) Using a slice (e.g., string1[0:4]) to extract a string with the first four bases of the read. (iii) setting up an "if" statement that only continues processing for lines where the first four bases are not the expected barcode (e.g., (v) Testing for the presence of mir-81 in the current line using the "in" operator (e.g., "if string1 in string2:") or the "count" operator (e.g., "if string1.count(string2)>0"), and (vii) Printing the relevant lines.

**2c.**  5' and 3' ends of miRNAs are not always precise. Make a list of sequence reads with a good 5' barcode and a full copy of mir-81 but with additional bases upstream of mir-81.

Your code:

Number of such reads:

A list of the linker-extended mir-81 reads:

Goals 2c: (i) Modifying the code in problem 2a with a bit of additional sophistication. Now instead of checking for a perfect match to the predicted sequence, you want to check for a perfect match at the beginning of the sequence to the barcode and a full copy of mir-81 but not a perfect copy of barcode+mir81.

**2d.**  (Optional Sub-problem) Even when a miRNA is the right length, has perfect barcodes and linkers, there can be internal mismatch variation (due to genetic variation, sequencing errors, RNA editing and transcription errors, and other factors). Modify the code above to return a list (and count) of lines comprised of a perfect barcode followed by the complete mir-81 sequence with a single mismatch (but the correct length). Note.. this is somewhat challenging with tools provided so far, but should be do-able and will gives folks who need an additional challenge something a bit more "crunchy" to work with.

Your code:

Number of such reads:

A list of the "mutated" mir-81 reads:

Goals 2d: Have a problem that is a bit of a challenge but do-able with current tools in the course. (i) For each read set up an "if" statement that makes sure the initial barcode is present and correct, and that the expected portion of the 3' linker is present, (ii) Setting up a variable that will hold (for each time through the loop) the count of matches between the potential mir-81-matching sequence and the expected mir-81 sequence. (iii) setting up a series of "if" statements (one for each base in mir-81) that will increment this "match" variable every time there is a match between mir-81 and the current line, (iv) printing the line if at the end of the loop there are exactly the number of matches desired for the single-mismatch class (v) thinking about how much easier this would be if lists of objects could be operated on as individual entities ("list" objects), which we'll talk about this week.

-------------------------------------------------------------------------------------------------------------

Any comments on problem set 1?

Enter your comments here