In this lab, we're going to explore more about string manipulation, using string methods, slicing and using data files and web pages. Do ALL printing nicely. DON'T forget to add comments at the top of your file.

(1) Let's write a file. You're going to CREATE a file called labWk7.txt. You're going to WRITE into that file. Open the file, ready for writing.

Then ask the user to enter 10 numbers, one at a time. Write each number, followed by a newline, into the file. Don't forget to close it at the end.

(2) Ok. Now. Let's READ the file labWk7.txt. Instead of what we did in class, let's do things a different way. We'll use the .read() file method to read all the contents in at once. THEN, we'll split the string, so we SHOULD end up with a list, where the elements of that list are strings (we'll refer to this as list A). That sucks, because we still probably want to do math with the numbers.

So, let's create a NEW list. We'll call this list B. Iterate through list A, turning each element into a number, and then append that number to list B. Print list B at the end - it should be a list of numbers, and NOT strings.

(3) Print out the min, the max and the average (the mean) value in the list. NICELY.

(4) Download the file parasites.txt from nexus. You MUST put this file in the SAME folder as your python file for this lab.

parasites.txt is a text file containing a list of nucleotides associated with common parasites. Each parasite has a name, like Schisto unique AA825099.

There are three example parasites in this file (*but there could be thousands, so keep that in mind. DO NOT do anything that relies on you KNOWING how many*

parasites are in this text!)

Let's imagine we want to find the NAME of a parasite containing a particular nucleotide subsequence (such as 'ttgtgta'). The structure of this file is that there is a name (such as *Schisto unique AA825099*), followed by a new line, followed by the complete sequence of nucleotides that make up that parasite. We often want to find names of parasites that are associated with subsequences of nucleotides.

First, write code that reads in the text of this file as a string. Now, in the command window, once you have read in that information, try the following two things.

First just type the name of the variable. You should see a bunch of letters. NOTICE that there is some thing that occurs again and again - it looks like \n. \n is the newline character, and you see it every time the enter key is pressed. Meaning that each one of these \n characters is a new line. When we ask python to show us the contents of the variable, it shows us the newline

characters. But if we ask Python the PRINT the contents of the variable, it should look different, because it takes each of those \n characters in to account. In your interpreter, now type print(data), and see how it looks different.

Look at the file (open it in text edit, or some other editor), and see what the structure is.

(a) Write a python program that asks for a single string (a small subsequence, such as 'ttgtgta'), and which finds this subsequence of nucleotides in the parasites.txt file. This program will open the parasites.txt file, read the file as a single string, and search that string for ONLY the FIRST instance of the subsequence parameter.

Ask the user to enter a subsequence (like 'ttgtgta'). PRINT OUT the index of where you find the subsequence. If the sequence is NOT found, print "SUBSEQUENCE not found"

HINT: Remember what the find method returns if something is not found? If it IS found it returns the INDEX where the sequence starts.

(b) Print out the NAME of the parasite in which you find the subseq. The name occurs at the beginning of each sequence, following a '>' symbol, and ending with a newline character '\n'. Remember, you CANNOT see this newline character, but it's there and python can see it, every time you press the enter key.

In this file, there is a '\n' at the end of EVERY line.

These \n characters act as if you had pressed the return key. That means that whenever there is a new line, there is a hidden \n. We can use this information to help us find the name of the parasite (because we can 'look' for them).

So, HOW are you going to find the name of the parasite that contains a particular subsequence? READ AND UNDERSTAND ALL the following hints.

HINT: This is a HARD question. TAKE YOUR TIME. Figure out the problem.

HINT: WORK out how you will solve this on a piece of paper. If you KNOW where a subsequence is, HOW do you find the name of a parasite associated with that subsequence. You can use string methods to achieve this.

HINT: You need ONLY find the first instance of a parasite that contains a particular subsequence.

HINT: find and rfind can really help you here. How? You know about find. It FINDS the FIRST index where the sequence is found. rfind finds the LAST index where information is found. How can that help you? Honestly - unless you figure this out on paper, you're going to struggle.

HINT: find and rfind also take optional parameters. WHAT are they, and HOW can these help you find what you want? Look up the string method documentation. Again, make sure you figure

out on paper, remembering that you can 'find' different points (index values) in your string. How can you use these index values to help you?

HINT: You HAVE to do this step by step. Find the first thing that helps you? You get an index number. You should ALSO remember that ALL strings START at zero, and end at len(string).

HINT: You do NOT know the names of the parasites in advance, so you cannot look for them directly

HINT: There could be MANY more than three parasites in this file, so do not use anything that relies on the fact that you know that there are three examples in this file

HINT: You can extract the name of the parasite using slicing. What does slicing NEED to work?

(5) Python is really cool, because it includes libraries that you can use to do all kinds of stuff for you. There is a library you can use to help you access web pages. That library is called urllib.

To use it, at the top of your python code (right at the top of the .py file), you write:

import urllib.request

urllib.request includes methods that mean you can grab stuff from real web pages. One method is called urlopen. It acts JUST like the open function for normal text - opening the webpage so you can read the text.

So to open a webpage from the internet (rather than opening the file from your computer), read the contents, and then close that page, we do:

```
connection = urllib.request.urlopen(filename)
data = str(connection.read())
connection.close()
```

Then the variable data contains very similar information information as if you had read the information from a file.

So, lets do a little webscraping!

Create a filename variable, that contains the string: 'http://cs.union.edu/CSDEPT/staff/'

You should check out this webpage - it's the directory of the computer science department staff.

Read this page as a string. You can print it out. It'll look like nonsense. In fact what your looking at is HTML - HyperText Markup Language. ALL of the internet looks like this.

HINT: It is possible you'll get an error. If you see an SSL error of some kind, replace the code above with:

```
 import ssl
context = ssl.create_unverified_context()
connection = urllib.request.urlopen(filename, context=context)
data = str(connection.read())
connection.close()
```

In this HTML, email addresses are often highlighted as links, using the phrase 'mailto:' in HTML.

Here's what we want to do.

- Find the first instance of mailto:,

- then use this information to zero in on email addresses. You might start by finding the first one, printing out a slice of text 30 characters either side of what you find. Then identify where the email address is. HOW would you recognize a union email address? What symbol is a reliable indicator of an email address?

- You need to FIND things in RELATION to other things. That is...knowing an index value - say, a mailto:, you need to find the first important symbol that occurs after it. If ONLY find let you specify where you start searching. Wait. It does!

- Once you've found the important symbol, can you USE it's index position to slice out an email address.

HINT: We can assume that the email address is NO LONGER than 6 characters to the left of the symbol, and not more than 10 characters to the right.

After you've printed an email address....FIND the next mailto: statement.

Eventually, I want you to find EVERY mailto: statement (i.e. keep finding mailto: until there are no more. How can we tell?) and slice out each email address that appears in this page. Print each one nicely.

Congratulations. You're now ready for your first spam email campaign!

As usual, turn in code to nexus.