Checkpoint 3
Due October 17

Checkpoint 3 will focus on the following objectives:

1. Cleaning up existing object-oriented structure
2. Adding Canonicizers (Text manipulation)
3. Improving our linked-list
4. Adding a stack implementation based on linked-list
5. Understanding some general approaches to authorship attribution

These objectives are important in preparing for checkpoint 4 as you will receive your first modules from the instructor that contains many of the statistical methods used for your authorship attribution analysis.

In previous checkpoints, we started the framework of our authorship attribution problem. More specify, checkpoint 1 introduced us to file management, file reading, and object-oriented design. Checkpoint 2 introduced us to graphical interfaces using TK and matplotlib as well as cleaning up our commandline plotter and starting our Slink class (singly linked list) that will be used in our project. It is important to complete the steps in the previous two checkpoints.

The below steps must be completed:

Authorship Attribution Background.

1. ~~Please read over chapter 3 and chapter 5 on the introduction of methods used for authorship attribution from the file posted on moodle. (Side note: Prof. Patrick Joula is known for his own program called JGAAP. Look at manual for this program might give you some insight on designing an interface for your own program.) Please start a class called AAMethods that will simply be passed. (We will come back to this in checkpoint 4). For now write a summary of your findings from chapter 3 and chapter 5 in the docstring of this class. The length should not be more than a standard page with 12pt font and no less than one-half a page.~~

Canonicizers. When we first read in our document, we did not care about case or white space. We simply strip our punctuation and placed it into the Sentence class. Now we will give the user options to how we will preprocess the data **AFTER WE HAVE READ IT INTO OUR DOCUMENT CLASS.**

2. ~~In order to apply this preprocessing, we are going to use a factory class named TextFilter. TextFilter takes in a document and a list of strings. Each string in the list states which filter should be applied to the text. Note, the filter will **NOT** apply these filters at init The initialization will only store this information to be used in the "apply" method.~~
3. ~~Create methods in TextFilter that will do the jobs below~~
    a. ~~Normalize whitespace – make all white space, 1 space. This will delete any extra spaces or line returns that exists.~~
    b. ~~Normalize case – will make all letters lower case~~
    c. ~~Strip null characters – will remove characters not in the standard ascii set of letters and numbers~~

Checkpoint 3
Due October 17

   d. Strip numbers – will remove all numbers

4. Create a method in TextFilter called apply. When invoked it will apply all the requested filters in the order in which the user provided them.

LinkedList. You will now upgrade your Slink class.

5. Upgraded your nodes in Slink to have multiple pieces of data, namely data is now a list, and next is also a list. This will be important as the difference of chain method that we introduce in checkpoint 4 will need this feature. For example, the list could contain both word and number of words.
6. In order to make sure the number before work, we will make a new frequency method in basic stats that now uses Slink in place of a dictionary. Call this slinkFreq, and do a run time analysis of this in the docstring of the function

Stack. You will now make a simple stack.

7. Create a stack class called SStack. The extra S stands for using Slink as our data structure in our ADT. The class must have the same required interface of stack ADT plus the same runtime as those we talk about in class. See class notes for the details.
8. Use your SStack to implement the Slink optimization we talked about in class, i.e., premake nodes into a stack, then add and remove in slink uses the premade node in the stack first. If you run out of nodes in the stack you may make new ones.
9. You will now make a new topN that will use the linked-list in number 6 with our new stack. You will walk the linked list and compare values. If larger than the other topN, remove the node from the linked list and place it into the stack. We can improve the performance by manually walking the linked-list and not