1. Take previous code you’ve written and convert it into a function-based program. The user decides at the beginning of the program which of the following three options they want, and then the appropriate function is called:

(a) Function 1 takes a sentence and splits it into vowels and consonants and outputs how many of each it found. For example, if the sentence is “the cat sleeps”, the output should be “It has 4 vowels and 8 consonants.”

(b) Function 2 does the same on the word level: it splits the sentence into words and reports the number of vowels and consonants per word. (A word here is defined as anything between whitespace.)

(c) Function 3 checks how many words of length \( n \) the sentence has, where \( n \) is specified by the user.

2. Write a program with three functions:

(a) The first function reads in the POS tagged text from file \texttt{vm.pos} and creates a lexicon that contains the ambiguity class for each word.
   - Ambiguity class = the set of all tags for a word. You may represent this as a string as in, e.g., \texttt{’JJ/NN’}

(b) The second function is a look-up function that returns the ambiguity class for a given word.

(c) The third function finds all the words for a given POS tag (not ambiguity class!).
   - You will have to think about what dictionaries you want to maintain.

3. (a) Take a list of strings and, using \texttt{pop()}, define a function that \texttt{recursively} pulls the first letter off each list item and creates a string of “first letters.” For example, if the list is \texttt{[’gamma’,’alpha’,’nu’,’delta’,’alpha’,’lambda’,’frodo’]}, the function would return the string \texttt{’gandalf’}.

(b) Now, do the same thing, but accumulating the letters in reverse order. In this same example, you would return \texttt{’fladnag’}. Do not use python slicing to reverse the string, but rather accumulate the letters in reverse order as you go.