Random Sentence Generator

This assignment gives you practice with dictionaries and functions. Your task is to apply a Markov chain algorithm to the text in a given file and then use the results to output random sentences that sort-of, sometimes, make sense.

Deliverables: your program named username1_username2.py

Background

Markov chains refer to a random process constructed by utilizing only the current state to identify possibilities for the next state (For more info see http://en.wikipedia.org/wiki/Markov_chain). For example, if a frog jumping among lily-pads is incapable of remembering which lily-pads it had previously jumped to, it is only limited by its current position. The frog's next jump will not be influenced by where it has previously been. A simple version of this method can be used to make a program that automatically generates reasonably sensible sentences (though often rather humorous). Sentences will be built 1 word at a time, where the current state will be the word last added to the end of the sentence, and so the next word to add will be chosen based only on what words are likely to follow the current word, with no consideration given to any earlier words in the sentence. The likelihood of words following a given word can be automatically determined using a reasonably large body of text, such as a book, by simply looking at the word that follows each time the given word appears in the text.

Your Task

You will write a program that uses the above method to generate random sentences based on a specific file of text. It is up to you to decide how to organize your code into functions, but **the quality of organization will be part of your grade.** There are 2 main parts to your task; getting word connection info from the text file, and actually generating sentences, described in detail below. Creating just 2 functions, one for each of these parts, will not earn full credit for organization, fyi.

1. Building dictionary of word connections from a text file

It would be very inefficient if every time we went to add the next word to a sentence, we had to look through the text file for all occurrences of the previous word to get the likelihoods of following words. Instead your program will read through the entire file exactly once, and build a dictionary to store this information for every word in the file. At the same time you will create a list of all words in the text that appear at the beginning of a sentence. **You must use a dictionary and list in the way described below**, using this particular method is part of the assignment!

The text file to use should be included as a **command line argument,** just like the image file in the previous HW.

The dictionary you build will contain an entry for each unique word in the text file, where the key is the word, and the value is a list of all words that ever follow that word; words should appear in the value list as many times as they appear in the text directly after the key word.

Example:

Given the text:
"The quick brown fox jumps over the brown frog which jumps over the brown log which jumps not."

The resulting dictionary would have the following entries:

key : value

'The' : ['quick']
'quick' : ['brown']
'brown' : ['fox', 'frog', 'log']
'fox' : ['jumps']
'jumps' : ['over', 'over', 'not.']
'over' : ['the', 'the']
'the' : ['brown', 'brown']
'frog' : ['which']
'which' : ['jumps', 'jumps']
'log' : ['which']

Note that the value for each key is always a **list**, even if that list only contains one word. KEEP the punctuation with the words, it adds to the quality of sentences constructed! Note that in the above example "The" and "the" are considered 2 completely different words, and "not." keeps the period with it, this is correct.

The list of start words should simply be a list of all words in the text that begin a sentence. You may have to think a bit though about what criteria to use to automatically determine if a word begins a sentence. You don't have to deal with every possible edge case in this, some combination of looking for words that follow specific punctuation and begin with a capital letter should work well enough. Note a word should not be added to the start word list twice, though the same word with different punctuation is again counted as two separate words.

Example:

Text: "Hello. How are you on this fine day? My name is Sherri. Hello again! How goes it?"

startWords = ['Hello.', 'How', 'My', 'Hello']

2. Generating Sentences

Now we can finally use our dictionary and list to generate sentences. To create a sentence, begin by selecting a random word from the start word list and add it to the sentence as the first word. You may find the choice(sequence) function from the random module useful for this. To get the next word, obtain the dictionary entry with this first word as the key, and choose a random word from the list of words that is the value in that entry. Add the newly chosen word to the sentence and use it as the next key. The algorithm then continues on randomly picking a new word from the list associated with the current key word, adding the chosen word to the sentence, and updating the current key to be this most recently added word.

Generated sentences should be at least 4 words long, but no more than 50 words. You should end your sentence if either of the following conditions occurs:

1. the sentence reaches 50 words in length
2. the last added word itself ends in a period, exclamation point, or question mark (this is why you will keep punctuation with your words in the dictionary)
    a. the one exception to this is that if your current sentence still has fewer than 4 words, go ahead and keep going. All sentences must have at least 4 words.

Final Program

Your final program should get the text file to use as a **command line argument** and output the first randomly generated sentence. It should then continually ask the user if they want to continue or quit, and if they choose continue output another sentence using a new random start word. If they choose quit, the program should, of course, quit. I provide you with 2 text files for testing, the complete text of Treasure Island, and a collection of Mother Goose nursery rhymes. Use your own and see what sort of sentences you can create!

Example of running my program (user input shown in bold):

```
sgoings$  python  sgoings.py  TreasureIsland.txt
Generated Sentence: Well, I turned on my boots, and mortal
white rock.
Enter ‘c’ to continue or ‘q’ to quit: c
Generated Sentence: "Ah, Silver!" says you.
Enter ‘c’ to continue or ‘q’ to quit: q

sgoings$  python  sgoings.py  MotherGoose.txt
Generated Sentence: Here's a nail, and one for thee!
Enter ‘c’ to continue or ‘q’ to quit: c
Generated Sentence: One for my pony now for all the next
day; Lumpety, lumpety, lump!
Enter ‘c’ to continue or ‘q’ to quit: q
```