CSE 231 Fall 2010

Programming Project 07

This assignment is worth 50 points (5.0% of the course grade) and must be **completed and turned in before 11:59 on Monday, November 1st , 2010**

Assignment Overview

This assignment will give you more experience on the use of:

1. Dictionaries
2. Functions
3. Lists

The goal of this project is to gain more practice with dictionaries and all the capabilities they present. Dictionaries provide an indispensable structure to efficiently store various types of data, expanding on the capabilities of a list. This project will provide the opportunity to exploit different features of dictionaries and turn a complex problem into a simple task.

Given an input text file, use a Markov chain algorithm to restructure the text. Save the result to a file.

Background

A Markov chain is a way to predict "what to do next" based on "what you did recently" (for the curious, all the gory details are here: http://en.wikipedia.org/wiki/Markov_chain). A Markov chain maintains a fixed series of states, and based on those states determines what to do next. It is something like walking a path. The direction you take on your next step is determined by a fixed number of previous steps. The number of previous steps that are remembered is fixed at the beginning of the process.

Such a method can easily be applied to bodies of text to create, more often than not, humorous constructions of reasonably sensible sentences. This construction is based on the sampling of existing text to create a Markov chain. One can scan the existing document, observing the previous ***n*** words, and remember the word that follows. Using this information, one can create a new document based on the Markov chain created from the previous document. One notable instance of using Markov chains for text generation is the dissociated press command in Emacs.

Project Description / Specification

Upon reading an input text file, construct a dictionary that represents a Markov chain. We will set the number of previous words we observe in this chain to ***two***. Thus, for every word, we record as a key the previous two words in the document, and the value as the next word.

Example:

The quick brown fox jumps over the lazy dog.

```
markovChain['The quick'] = ['brown']
markovChain['quick brown'] = ['fox']
…
```

The created dictionary must contain a ***list*** of words for each value. In our program, most dictionary values will only have a single two-word key, but since it is possible that the same key might occur for two different values, we must maintain a list. (For example, in “to be or not to be that is the question” the key ‘tobe’ has two possible next words: [‘or’, ‘that’])

To create a new text document, we begin by selecting the first two words of the original text file as the first two words of our new document. These first two words are our initial key. The value associated with that key will be the third word in the new text document. If there is more than one word in the dictionary value, one of the words is chosen randomly. A new key is formed from the second and third words, and its value added as the fourth word of the document. The newly chosen word is added to the new text document, the key updated and so on. The algorithm cycles by randomly picking a new word from the dictionary value based on the two-word key, adding the value of the dictionary to the text document, and updating the key.

Replacement of a key word is always in the following form.

```
    Key = Word1 + ' ' + Word2    # put back space that split() removed
    Word3 = markovChain[Key]     # randomly chosen word in this list
    Key = Word2 + ' ' + Word3
```

If a key does not have an entry in the dictionary, the program should end text generation. Of course, based on the input file, we could enter an infinite loop of text generation, so limit output to 500 words.

Once the new text has been generated, display the results, and prompt the user for the name of an output file. The user may choose to discard the text by not entering a file name, at which point the program should end.

Example Output

```
Enter the text file name: TreasureIsland.txt

The results:

-----------------------------------------------------------------------
-
From the side of the others, we should want you to help work the
vessel home." "Ah," said he, "but I had—remarkable pious. And I was
not defenseless, courage glowed again in my heart, and I know it.
But where was you, do you call yourself, mate?" "Jim," I told him
the whole story of our voyage and the ringleader, too." He was
concealed by this time, behind another tree-trunk, but he must have
shown the feeling in my face, for he repeated the statement hotly:
"Rich! rich! I says. And I'll tell you, and no more. I were in
Flint's ship when he buried the treasure; he and six along—six
strong seamen. They was ashore nigh on a week, and us standing off
and on in the most liberal of men. "Ay, but you see," returned Ben
Gunn, I am; and I saw a figure leap with great rapidity behind the
trunk of a fellow-creature. But at my last words he perked up into a
kind of punishment common enough among the buccaneers, in which the
offender is put ashore with a little powder and shot and left behind
```

```
on some desolate and distant island. "Marooned three years agone,"
he continued, "then you'll up, and you'll say
...
--------------------------------------------------------------------
-


Enter file name to write output to <Enter for skip>: lol.txt
```

Your project will do the following:

1) Prompt for the file name containing the text to work on.
2) Create a function that takes as input, the file name of the text, and returns a list containing all the words in the file. Only newline characters are to be removed from the text. Keep all punctuation (they will likely be attached to words).
3) Create a function that takes as input, a list containing all words in the text file, and returns a dictionary formatted as specified above.
4) Create a function that takes as input, a list containing all words in the text file, a dictionary formatted as specified above, and returns a string containing the text created from running the Markov chain.
5) Print the text, and then prompt for a file name to write the new text out to. No file name indicates the text is to be discarded.
6) If a file is selected, write the file to include newlines so that no more than 80 characters are written on a line, and no words are split across two lines.

Deliverables

Turn in proj07.py, using the handin program.
Save a copy to your H: drive.

List of Files to Download

TreasureIsland.txt

Notes and Hints:

When adding a word to the dictionary, you must first check if the key for the word already exists. If so, you will simply be appending the word to the list already created. Otherwise, create a list containing the word, and assign it to the entry. The **`in`** function for dictionaries will prove useful here.

The random module contains various functions for selecting random numbers or randomly selecting an item from a list. See **`random.sample()`** and **`random.randint()`**.

Punctuation should ***not*** be stripped, otherwise punctuation will not be seen in the new document. Because the rules here are a little loose, the punctuation generated may not be completely correct, for example parentheses may not match, quotes may not match, and you may get multiple punctuation marks in a row. Don’t worry about this.