IAE 101 – Introduction to Digital Intelligence
Fall, 2019
Programming Project 04 – Poetry Generator

Assigned: Friday, November 15th, 2019
Due: **Wednesday, December 4th, 2019, at 11:55 PM**

Description:

Poetry is one of the most intricate activities that humans perform. It relies heavily on our facility with both syntax (the form) and semantics (the content or meaning) of language in order to work. Think about how a poem is constructed. You must choose each word with attention both to its form--how it looks, how it sounds--and to its meaning, as well as its relation to the other words in your poem.

Is it possible for a computer to perform this same activity and produce human-looking results? In this project, you will implement a poetry generator that uses the language analysis and selection tools we have discussed in class to try to mimic human construction of poems. This will require the random generation of words as well as the use of algorithms to analyze the structure of those words to detect and use features such as rhymes, syllables, and stresses. Your implementation should use these features of words to construct rules that will be applied to word choice. Words will be randomly selected, but those selections will be constrained by the rules you impose regarding the structure of your poem. This is a fun combination of creativity and rigor as you attempt to capture and express the ineffable-seeming methods that poets use to create their works.

Setup:

1. Install the `NLTK` Python library:

Windows: `py -3 -m pip install nltk --user`

Mac: `python3 -m pip install nltk --user`

2. Download `NLTK` corpora (plural of corpus) and other files.

a. Open the Python interpreter (e.g., start Idle in interactive (shell) mode).

Then, from inside the interpreter execute the following Python statements.

b. `>>> import nltk`

c. `>>> nltk.download('all')`

The total size of the download is about 3.2 GB (gigabytes).

3. Install the pronouncing Python library:

```
Windows:    py -3 -m pip install pronouncing --user

Mac:        python3 -m pip install pronouncing --user
```

Please do the installation as soon as possible (this weekend) so that any installation problems can be resolved in a timely fashion.

Directions:

Inside the file poetry_generator.py, you have been provided with several functions that you will use in defining the rules for your poetry generator.

The first function, random_word_generator(), will generate a random word or a random sequence of words using the *conditional frequency distribution* derived from the bigrams in your selected corpus. You pass in a source word and an integer and the function will return a list of words selected in sequence, such that each word is one that commonly follows the word before it in the corpus.

The second function, count_syllables(), will return an integer representing the number of syllables in the word passed in as argument. For words that have multiple pronunciations with differing numbers of syllables, the function returns the max number of syllables for any pronunciation of that word.

The third function, get_rhymes(), takes a word as input and returns a list of words that rhyme with the input word.

The fourth function, get_stresses(), takes a word as input and returns a list of the patterns of stresses that be used to pronounce the word. Some words can be pronounced in multiple ways, and each way will have a different patter of stresses. Each element of the list is a string of numbers, one number per syllable in a pronunciation of the input word. Number '1' indicates that a syllable has primary stress. Number '2' indicates that a syllable has secondary stress. Number '0' indicates that the syllable is unstressed.

Example: The word permit has two pronunciations. Their stresses are represented by the strings "01" and "12". The first pronunciation places all the stress on the second syllable. The second places primary stress on the first syllable and only secondary stress on the second. The result returned by calling get_stresses('permit') is the list: ['01', '12'].

You are responsible for implementing two functions: generate_line() and generate_poem().

Your implementation of these two functions will determine the structure and composition of the poems generated by your program.

The function generate_line() will determine, for each line, how many words or syllables it is composed of, what is the relation between words--is there a meter (a pattern of stresses) the line has to obey--whether the line has to rhyme with an earlier line. You impose these restrictions as you generate words for the line reject words that don't obey the rules. The output of generate_line() should be a string of randomly generated words chosen according to the criteria you decided upon for lines of your poem.

The same is true for generate_poem(). Here you'll implement the rules about how lines are related to each other (enforced by your implementation of generate_line(), and how many lines your poem will contain. The output of generate_poem() should be a string of your randomly created poem.

Print the string returned by generate_poem() to display your poem to the user.

Submission Instructions:

1. Please add your name, netid, and student ID to the top of poetry_generator.py as comments.
2. Upload your poetry_generator.py file to Blackboard.
3. Upload two poems created by your poetry generator.