IAE 101 – Introduction to Digital Intelligence
Fall, 2024
Programming Project 03 – Poetry Generator

Assigned: Friday, November 8th, 2024
Due: **Tuesday, November 26th, 2024, at 11:59 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()`

    This will launch a new dialog window listing all the NLTK related packages that can be downloaded. For this project you should select "all-corpora", so you can have access to all the word sources (but, downloading just the 'gutenberg' and 'shakespeare' corpora would be enough). We won't be using any of the other packages available, but you may want to explore them on your own.

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.

**Submission Instructions:**

1. Please add your name, netid, and student ID, etc. to the top of `poetry_generator.py` as comments.
2. Upload your `poetry_generator.py` file to Blackboard.
3. Upload a brief document describing the kind of poem you program writes.
4. Upload two poems created by your poetry generator.

**Resources:**

Inside the file `poetry_generator.py`, you have been provided with several functions that you will use to implement your poetry generator.

1. `next_word_generator()`, will generate a random word using the *conditional frequency distribution* derived from the bigrams in your selected corpus. You pass in a source word and the function will return a new word, such that the new word is one that frequently follows the source word in the corpus according to the relations in the CFD.
    - For example, if you called the function like this: `next_word_generator('to')` then, it would return a new word.
    - The selection is made proportional to the conditional frequency of all words in the corpus, conditioned on the source word 'to'.
    - If the source word argument is left out, then its value defaults to None. If the parameter has value None, then the function returns a new word randomly selected from the entire corpus.

2. `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.

3. `get_rhymes()`, takes a word as input and returns a list of words that rhyme with the input word. The list has been filtered so that it only includes words from the corpus.

4. `get_stresses()`, takes a word as input and returns a list of the patterns of stresses that can be used to pronounce the word. Some words can be pronounced in multiple ways, and each way will have a different pattern 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']`.

**Requirements:**

You are responsible for implementing several functions:

1. `generate_rhyming_lines()`
2. `generate_10_syllable_lines()`
3. `generate_metered_line()` (**Bonus: not required**)
4. `generate_line()`
5. `generate_poem()`

The first 3 functions in this list are meant to test your ability to construct lines that have a specific property:

- `generate_rhyming_lines()` – For this function you must use the helper functions described above to generate 2 lines of randomly selected words. Each line must be composed of 5 words. The two lines must rhyme with each other. There are no other requirements. The function should return a list whose elements are the two random rhyming lines constructed by your code.

- `generate_10_syllable_lines()` – For this function you must the use the helper functions described above to generate 2 lines of randomly selected words. Each line must be composed of 10 syllables total. There are no other requirements. In particular, there is no requirement regarding the number of words; however many words there are, the sum of their syllable counts must be 10 per line. The function should return a list whose elements are the two random 10 syllable lines constructed by your code.

- `generate_metered_line()` – (**Bonus: not required**) For this function you must use the helper functions described above to generate 1 line of randomly selected words. The line must be composed of 10 syllables, and the rhythm of the line must match the following pattern of stress: "0101010101". This is the meter of Iambic Pentameter. The function should return the random line constructed by your code as a string.

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

- `generate_line()` - This function 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.

- `generate_poem()` - This function implements 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.

Advice: If you are looking to keep things simple, I would suggest you choose a style of poetry that is simple with few requirements. For example, one form of haiku are composed of three lines: the first line has five syllables, the second has seven syllables, and the third line has 5 syllables. There are no requirements concerning either rhyming or meter for this style. There may be other, similarly simple styles to choose from as well.

**Submission Instructions** (repeating this because so many of you ignore it)**:**

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 a brief document containing the following:
    a. A brief description of the kind of poem you program writes.
    b. Two poems created by your poetry generator.