COSC 101 Homework 7: Fall 2023

The due date for this homework is **Friday, 12 April 2024, 11:00pm**.

The topics covered in this assignment include:

- Program design
- Dictionaries
- File I/O
- Exceptions
- Loops
- Functions
- Lists
- Strings

Instructions

Download `hw7.zip` and unzip the compressed file to reveal seven files included with this assignment:

- `hw07_haiku.py`: file in which all your code will be written
- `hw07_syllables.csv`: a CSV file containing words and the number of syllables in those words
- `hw07_anthology.txt`: an relatively small input text file containing haikus for testing
- `hw07_novel.txt`: a large input text file that can be used for testing
- `hw07_novel_haikus_10500.txt`: a text file containing all the haikus that can be found in `hw07_novel.txt`

All of your work for this assignment will be completed in the file `hw07_haiku.py`.

Background: Haiku

Haiku is a form of Japanese poetry. One of the key characteristics of a haiku is its syllable pattern: a haiku is composed of three short phrases consisting of 5, 7, and 5 syllables.

Your goal is to write a program that finds haikus that unexpectedly exist within a corpus of text.

Program Specification

The structure for creating a program that searches for haikus that happen to exist in a body of text has *mostly* been created for you. In `hw07_haiku.py`, the functions described below are started, including a few test examples for each function.

Functions to write

1. `normalize_word`

This function accepts a string (a word) and returns a *normalized* version of the word in which any trailing `.,?!:;` characters are removed, and the word is lower-cased. Hint: use the `rstrip` and `lower` string methods.

2. `load_input_text`

This function accepts a string which is the name of a file from which to load a body of text within which to search for haikus. The function should return a list of all words found in the file, where each word has been *normalized* (use the `normalize_word` function created in step 1, above). The function should use

`try/except` to check whether the filename given as the parameter actually exists; if the file doesn't exist the function should return an empty list.

3. `load_syllables`

This function accepts a string which is the name of a `.csv` (comma-separated) file containing information about how many syllables different words have. Each line of the input file can be assumed to have a word followed by a comma, followed by an integer number of syllables. You *do not need to check whether the input file exists* for this function (unlike for the function written for step 2, above). This function must return a Python dictionary where the key is a word and the value is the number of syllables for that word. You should *not* "normalize" any of the words in the input file; just use them as-is.

4. `words_to_syllables`

This function accepts two parameters: a list of normalized words (i.e., the kind of list produced by `load_input_text`), and a syllable dictionary (like one created by `load_syllables`). The function should return a list of integers, where each integer is a number of syllables corresponding to the words given to the function as the first parameter. Moreover, _any words that do not appear in the syllable dictionary should be removed from the word list, **in place**. Put slightly differently, you should modify the word list in place to remove any words that don't appear in the dictionary. For example, if the list of words is `['a', 'apple', 'notaword']` and the syllable dictionary is `{'a': 1, 'apple':2}`, the function should return the list `[1, 2]` (there is one syllable in 'a' and 2 syllables in 'apple') and should modify the word list in place so that it contains exactly `['a', 'apple']`. To modify the list in place, you'll need to use either the `remove` or `pop` method (or the `del` operator).

5. `check_haiku`

This function accepts a list of integers representing syllables, i.e., a list produced by `words_to_syllables`. The function should check whether a haiku can be formed from the given syllables **starting at the beginning of the list**. If a haiku *can* be formed, the function should return the index in the list *after* the end of the haiku. If no haiku can be formed the function should return -1.

For example, if the list `[1, 2, 2, 1, 3, 2, 1, 2, 1, 1, 1, 3, 1]` is given to the function, it should return 10 (the index of the value `3`). Recall that a Haiku is formed 3 verses containing 5 syllables, 7 syllables, and 5 syllables. The first three elements in the list can form the first verse, the next 4 elements in the list can form the 2nd verse, and the next 4 elements can form the 3rd verse. The function should return the index of the *next* element in the list. You can also think of the function as returning the number of values in the list (starting at the beginning) that are needed to form a haiku.

The algorithm that you can use to determine whether a haiku can be formed is as follows:

1. Create an empty haiku
2. Add the next word of the text to the haiku
3. If the number of syllables in the haiku is less than 5, go to step 2
4. If the number of syllables in the haiku is greater than 5, go to step 1
5. Add the next word of the text to the haiku
6. If the number of syllables in the haiku is less than 12, go to step 5
7. If the number of syllables in the haiku is greater than 12, go to step 1
8. Add the next word of the text to the haiku
9. If the number of syllables in the haiku is less than 17, go to step 8
10. If the number of syllables in the haiku is greater than 17, go to step 1
11. Output the haiku, then go to step 1

Hint: use `while` loops and a nested structure (a new level of nesting for each verse) to implement this algorithm.

6. `find_haikus`

This function accepts a list of integers representing syllables. It should return a nested list in which each sublist contains exactly 2 items: the index in the list at which a haiku starts, and the *stopping index* of the haiku, which should be the index just *after* the end of the haiku. Your function should check, for every index in the list, whether a haiku can be formed from the syllables *starting at that index*; use the `check_haiku` function and pass in a slice of the list starting at a given index. Note that one haiku found in a given body of text can overlap another haiku found in the text.

7. `print_haiku`

This function accepts the full list of words in the input text, the list of syllables corresponding to the full list of words, the starting index of a haiku found in the text, and the stopping index of the haiku (one index beyond the end of the haiku). The function should print a nicely formatted haiku in which each verse is separated by forward slashes (`/`). For example:

```
an old silent pond / a frog jumps into the pond / splash silence again
```

Hints: use list slicing to extract the words in the haiku as well as the exact syllables in the haiku. Use an accumulator pattern or modify the list of words in the haiku in place to insert the forward slash in the correct position in the haiku word list. Use the string `join` method to join all the words (including the slashes) of the haiku together.

8. `haiku_finder`

There is nothing to do for this step: this function has already been written for you. It uses the functions written in previous parts to accomplish this task. This function accepts two file names (strings); the syllable CSV file name, and the name of the input text. It loads the syllable data, loads and normalizes the input text, creates a list of syllables corresponding to the input text, then finds and prints all the haikus found in the text.

Uncomment each line of code as you implement the different required methods it uses.

9. `main`

There is nothing to do for this step: a `main` function has already been written for you. Uncomment the two line of code provided in `main` when you are done implementing the rest of the program and are ready to test the program.

Testing

Incremental testing for each function

As you implement the required functions make sure you test them to tease out any potential bugs. Skipping this step may make debugging your program exponentially harder when you only test the entire program.

We have provided you with some tests in the `docstring` of each function. Make sure to first run the functions with these values inside `main`. You **must** run each function with at least one test case that you come up with. Once you are done testing a function, comment out the tests you wrote for it in `main`. **DO NOT** delete the tests as they count towards 10% of your grade.

Program Testing (all together)

When you are done implementing all the required functions, test your program to check that it's working properly. We provide here a few additional notes on the input files and text processing.

The file `hw07_syllables.csv` contains a list of English words and the number of syllables in each word. The file contains one word per line; the word and syllable count are separated by a comma.

We have provided three sample input files: * `hw07_anthology.txt` contains two haikus by famous Japanese poets, but because your program searches for *overlapping* haikus you should find 6 haikus in this file. * `hw07_novel.txt` is the complete text of *Flatland* by Edwin A. Abbott, which contains 10500 unexpected haikus * `hw07_novel_haikus_10500.txt` is the complete set of haikus found in the above text file, for reference.

Example

A sample run of the program is shown below. Make sure your program **exactly** matches this format.

Program output:

```
File in which to find haikus? hw07_anthology.txt
Found 6 haikus.
an old silent pond / a frog jumps into the pond / splash silence again
old silent pond a / frog jumps into the pond splash / silence again a
a frog jumps into / the pond splash silence again / a summer river
into the pond splash / silence again a summer / river being crossed
splash silence again / a summer river being / crossed how pleasing with
a summer river / being crossed how pleasing with / sandals in my hands
```

Submission

Submit your completed assignment on Moodle under your course section's Homework 7. You should upload the following file:

- `hw07_haiku.py`

Remember to complete the questions at the top of the provided files and that the files you submit need to have these exact filenames.

Grading

Your assignment will be graded on two criteria:

1. Correctness: this document contains details for how you must complete each function, including examples. Be sure that you run your functions once for each example and make sure they work correctly according to your tests for each one. [90%]

   The correctness part of your grade is broken down as follows:

Category	Portion of grade
`normalize_word`	10%
`load_input_text`	10%
`load_syllables`	10%
`words_to_syllables`	15%
`check_haiku`	15%
`find_haikus`	10%
`print_haiku`	10%

   Testing [10%] - refer to the section on Testing.

2. Program design and style [10%]: style and program design become increasingly important the more complex your program becomes. For these programs, adhere to the following guidelines:
    - Variable names should be meaningful
    - Programs should contain at least a few descriptive comments. Do *not* comment every line of code with low level explanations of what each line does. Focus on high level ideas.