Assignment 10

Objectives: Approach old problems using Object Oriented Programming. Write classes and methods. Use interfaces and protocols.

Note: Include DocStrings in each script you submit analogous to the following:

```
"""Interprets text as paragraphs, sentences and words


Submitted by Mauricio Arias. NetID: ma6918
This script takes a chapter from the book Don Quixote de la Mancha and
analyzes it to separate it in paragraphs and sentences. It provides
some basic functionality to reformat and characterize the structure of
the text.
"""
```

Make single-line DocStrings for all methods and multiline DocStrings for all classes. For the latter use a similar format as for the DocStrings used for the scripts: in the lower part of the DocString include a simple description for each attribute and a simple description for each method: in many cases PyCharm should prompt you.

Note: In all the scripts below, you are to catch the exceptions when opening files and tell the user there was a problem opening the file.

Full assignment. Tilting at windmills
For this assignment, we'll make classes to represent key types of structures for analyses of the basic structure of a novel.

Task 1 (4 points). Make a class named Sentence. This class should allow the analysis and characterization of a sentence in terms of the words it contains. Add the following functionality:

- When given a sentence, its constructor will process the text to generate a string with the original text and a list of words (strings) for analysis.
- Before spliting the text into words, the following characters should be removed completely: "_", "‟" and "‟" (`'_', '\u201c' and '\u201d'` in UTF-8). Each occurrence of the character "–" (`'\u2014'` in UTF-8) should be replaced by a space.
- For recognizing the words spaces are used as markers. However, the words have to be extracted from the parts so generated. Additional punctuation marks are expected to occur only at the end of the word and are not considered part of the word itself. In this way, "dog" is a word, "don't" is another word, but "talked," (note the comma) is not a word; the word "talked" is part of the string though. Notice that characters normally considered punctuation are not considered as such when they occur in the middle of the word. Hence "pre-made" is considered a single word.

The following methods should be implemented:
__init__: Constructor with a string as an argument.
__len__: Number of words in the sentence.
__str__: returns the original string.
__iter__: generates an iterator associated with the sentence that moves through the words. See example in module.
__next__: moves to the next word.

__getitem__: extracts the words by indexing. Accepts negative numbers. Returns None if the index is out of range.
__contains__: returns true if the word is present in the word list independent of case.
reset_iterator: resets the position of the iterator.
split(): splits the sentence at the character ";" generating a list of Sentence objects. If the character is not present the object returns itself as the only object in a list.

Name the module `[NetID]_sentences.py`.

Task 2 (4 points). Make a class named Paragraph. This class should allow the analysis and characterization of a paragraph of text in terms of the sentences it contains. Add the following functionality:

- When given a paragraph, its constructor will process the text to generate a string and a list of non-empty Sentence objects.
- The characters that separate sentences are the following: "`?`", "`!`", "`.`" and tandem combinations of them.
- Before spliting the text into sentences, the following characters should be removed completely: "_", "‟", and "‟" (`'_', '\u201c' and '\u201d'` in UTF-8).

The following methods should be implemented:
__len__: number of sentences in the paragraph.
__str__: returns the original string.
__iter__: generates an iterator associated with the paragraph that moves through the sentences. It has an associated position variable to keep track of which element should be provided next.
__next__: returns to the next sentence and updates the position.
__getitem__: extracts the sentences by indexing. Accepts negative numbers. Returns None if the index is out of range.
reset_iterator: resets the position of the iterator.

Name the module `[NetID]_paragraphs.py`.

Task 3 (3 points). Analyze the first chapter of the book Don Quixote at https://www.gutenberg.org/cache/epub/996/pg996.txt using the classes defined above. (The text is UTF-8.) Import each class from the corresponding module.
The first chapter is between the markers "p007.jpg (150K)" immediately followed by "Full Size" at the beginning and "p007b.jpg (61K)" at the end as full lines.

Notice that, in the chapter, each line of text should be appended to the previous one using a space unless it is empty. An empty line marks the end of the paragraph and the beginning of the next paragraph. Consecutive empty lines should be consolidated into one.

For the chapter, display the following information:

- the number of sentences in total,
- the number of paragraphs,
- the number of words,
- the average number of words per sentence,

- the average number of sentences per paragraph,
- how many sentences have the word “he” in them and
- how many sentences have the word “mancha” in them.

Show also a summary of the chapter by displaying the first and last sentences of each paragraph as stored in the Sentence objects.

Name the script `[NetID]_chapter.py` and the results `[NetID]_analysis.txt`.

All files described above should be submitted.