CSE101 – Spring 2020
Programming Assignment #7
(13 points, Submission due date: 27 May 2020)

Instructions

For each of the following problems, create an error-free Python program.

- Each program should be submitted in a separate Python file that follows a particular naming convention: Submit the answer for problem 1 as “Assign7Answer1.py” and for problem 2 as “Assign7Answer2.py”.
- These programs should execute properly in PyCharm using the setup we created in lab.
- At the top of every file add your name and Stony Brook email address in a comment.
- Write 2 test cases to test your code for all problems that do not take user input.
- Download these supplemental files – unzip the files and add them to your project directory. The problems will refer to specific files that you will load data from.

Problems

Problem 1: Text analysis (5 points)

Automated authorship detection is the process of using a computer program to analyze a large collection of texts, one of which has an unknown author, and making guesses about the author of that text. The basic idea is to use different statistics from the text – called ”features” in the machine learning community – to form a linguistic ”signature” for each text. For example, one basic linguistic feature utilizes the number of unique words. Once a signature is obtained, comparisons can be made to known authors and the text of their works.

In this problem, you will write a program to analyze some text files to determine three base-level linguistic features:

1. the average word length,

2. the Type-Token Ratio which is the number of different words divided by the total number of words (this is an indicator of how repetitive the text is), and

3. the Hapax Legomena Ratio which is the number of unique words (words only used once) divided by the total number of words.

We also want to see the most frequent words for the text to see how they differ by text. However, if we don’t do any additional text processing, this list will mainly be commonly used words across all texts such as “the”, “to”, “a”, and “I”. One standard approach to this issue is to keep a list of the common words that you want to exclude and then ignore them.

We will take a simpler approach and just look at the 10 most frequent words that are at least 5 characters long. This will exclude a lot of the common connecting words and be easier to implement.

Write a function called `text_features` that takes a file name as its parameter and output those base-level linguistic features and 10 most frequent words (and the count for how many times each word was used) as follows. With '`file.txt`' as an input file your program should produce the following output. (Actual numbers may vary slightly depending on how you handled your data – I reduced all the strings to lowercase and removed punctuation from any words, similar to the spam filtering problem. I also made sure to ignore any blank strings.)

```
Total Words 33
Average Word Length 4.15
Number of Different Words 25
Number of Words Used Once 20
Type-Token Ratio 0.7576
Hapax Legomena Ratio 0.6061

Most Frequent Words (word, count):
confidence 2, cicero 1, tullius 1, marcus 1, started 1, before 1,
defeated 1, twice 1
```

Assume that words in an input file are separated by the usual white space characters.

You will want to write some helper functions that will be useful to write `text_features`. Some possible helper functions would be:

- `get_average_word_length` which returns the average length of the words contained in the file as a real number (float).
- `get_number_of_different_words` which returns the number of different words (where duplicate words are counted as one word).
- `get_number_of_unique_words` which returns the number of words that appear exactly once (duplicate words are not counted).
- `get_most_frequent_words` which returns the most frequent words

Note that I did not specify what these functions take as their input parameter(s). It depends on how you design your program. There are several possible designs that you could consider. One possibility would be to create a dictionary and pass it to these functions. Another possibility would be to create a list and use it rather than a dictionary. With a list it might be a little easier to write the first function (`get_average_word_length`), it will be a lot easier to write the other functions with a dictionary.

You are provided with a test file, file.txt, and two "mystery" files, mystery1.txt and mystery2.txt to analyze. The mystery files were obtained from Project Gutenberg, a website with text files of free public-domain books. Below is sample output from my solution for the two mystery files.

mystery1.txt Output:

```
Total Words 163401
Average Word Length 4.05
Number of Different Words 10896
Number of Words Used Once 5597
Type-Token Ratio 0.0667
Hapax Legomena Ratio 0.0343

Most Frequent Words (word, count):
there 767, which 656, could 493, would 428, shall 427, helsing 299,
before 277, again 246, seemed 243, about 239,
```

mystery2.txt Output:

```
Total Words 78012
Average Word Length 4.39
Number of Different Words 7693
Number of Words Used Once 3972
Type-Token Ratio 0.0986
Hapax Legomena Ratio 0.0509

Most Frequent Words (word, count):
tuppence 579, tommy 533, there 326, julius 295, would 236, about 213,
don't 189, james 155, think 154, right 142,
```

Problem 2: Python class for conference attendees (8 points)

Write a Python program using classes that will keep track of people attending an international conference.

Write an Attendee class that will be used to keep track of an attendee's information as follows:

- Name
- Company
- Country
- Email address

The Attendee class should provide a constructor method that takes as input the above arguments and saves the information for an Attendee. It should be possible to create an object of an Attendee class as follows:

```
Attendee1 = Attendee("John", "ABC Pharma", "Barbados", "john@abcpharma.com")
```

The Attendee class should have a method `getInfo()` that returns the attendee's information as a dictionary. For example:

```
>>>Attendee1.getInfo()

{"Name":"John", "Company":"ABC Pharma", "Country":"Barbados",
"Email":"john@abcpharma.com"}
```

Additionally, you need to write a Conference class. This class will have methods to support several features, listed below. Read all of the directions first, but I would suggest doing these tasks one at a time and testing it before implementing the next task:

1. The conference class constructor takes as arguments the conference name, location, and dates. All of these should be strings, as shown below:

```
>>>conference1 = Conference("National conference on computing", "Seoul", "22-
25 December, 2020")

Conference "National conference on computing" is created.
```

2. Add an attendee to the conference. For Example:

```
>>> conference1.addAttendee(Attendee1)
Attendee John is added to the list of conference participants.
```

3. Remove an existing attendee from the conference. For example:

```
>>> conference1.removeAttendee(Attendee1)
Attendee John is removed from the list of conference participants.
```

4. Display all of the attendees (displaying the name, company, country, and email address for each attendee). For example:

```
>>> conference1.displayAttendees()
-----------------------------------------------------------------------
Name        Company           Country                 Email
-----------------------------------------------------------------------
John        ABC Pharma        Barbados          john@abcpharma.com
Lisa        HealthTech Inc.   Malaysia          lisa@healthtech.inc
```

5. Check if a specific person is attending the conference (matching on their name) and return True if that person is attending.

```
>>> conference1.checkAttendee("John")
John is attending the conference.
>>> conference1.checkAttendee("Seung")
Seung is not attending the conference.
```

6. Display all of the attendees from a specified country.

```
>>> conference1.displayAttendeesFromCountry("Barbados")
-----------------------------------------------------------------------
Name        Company           Country                 Email
-----------------------------------------------------------------------
John        ABC Pharma        Barbados          john@abcpharma.com
```

7. Create random groups for the attendees: make a method take takes as an argument the number of groups to create. This method randomly places all of the attendees into the provided number of groups with approximately equal number of attendees per group. The method should print each group and the list of attendees in that group, as shown below:

```
>>> conference1.groupAttendees(2)

Group 1:

-----------------------------------------------------------------------
Name        Company           Country                 Email
-----------------------------------------------------------------------
John        ABC Pharma        Barbados          john@abcpharma.com

Group 2:

-----------------------------------------------------------------------
Name        Company           Country                 Email
-----------------------------------------------------------------------
Lisa        HealthTech Inc.   Malaysia          lisa@healthtech.inc
```

Your program should load the attendees from the provided `attendees.txt` file – there is one attendee per line in the file and the attendees are listed in the format: "name, company, country, email"

Make sure to write at least 2 test cases to test each method listed above.