CS1110 Spring 2016

Assignment 6: Due Friday April 22 at 6pm

The figure in 4.1 was corrected on Sat April 16th at 4pm.

You must work either on your own or with one partner. If you work with a partner, you and your partner must first register as a group in CMS (this requires an invitation issued by one of you on CMS and the other of you accepting it on CMS) and then submit your work as a group. As mentioned in class, we strongly recommend against a "divide-and-conquer"/Henry-Ford-assembly-line approach: each member of a group should work on each part of each problem to get the full educational value out of the assignment.

You may discuss background issues and general solution strategies with others outside your CMS group, but the program(s) you submit must be the work of just you (and your partner). We assume that you are thoroughly familiar with the discussion of academic integrity that is on the course website. Any doubts that you have about "crossing the line" should be discussed with a member of the teaching staff before the deadline, *and you should also document such situations as comments in the header of your submission file(s).*

Warning: submit an early version, and be clear on the slip day policies. Recent experiences suggest that we:

1. . . .remind you that the specific slip day policies are posted on the Assignments page of the course website. *You can spend at most two slip days on an assignment; slip days are counted on the 24-hour marks.* See `http://www.cs.cornell.edu/courses/cs1110/2016sp/assignments/index.php` for more details.
2. . . .advise you to *submit something to CMS well in advance of the deadline*, since you can always overwrite earlier submissions. For instance, even if you are planning to use two slip days, it would be wise to still submit what you have before the usual Friday 6pm deadline or on Saturday, to avoid the possibility that you somehow accidentally miss the (slip-day) deadline and, with nothing on CMS, get no credit for the assignment.

Topics. Classes, objects, constructors, methods, lists of Objects, sorting, dictionaries.

1 Working with Long Files and Multiple Files in Komodo Edit

You are now starting to work with longer programs and multiple files. Some tips:

- The "minus" and "plus" signs on the left "margin" of Komodo Edit allow you to temporarily hide and re-show logical portions of code, ranging from the small scale, like the bodies of `if`-statements, to entire function or class definitions.
- If you'd like to view two files at the same time, go to Komodo Edit's "View" menu[1] and select "Split View". You can toggle whether the two views are arranged vertically (which most people seem to prefer) or horizontally (Komodo Edit's default) by using the "Rotate Split View" option in the same menu, or, less obviously, by double-clicking on the white "margin" between the two views.
- You can comment or un-comment whole regions using Komodo Edit's "Code" menu options "Comment Region" and "Un-Comment Region". Similarly options exist for indenting and un-indenting selected portions of code.

2 The Data

Because Shakespeare's work is in the public domain, online versions are readily available, e.g.,

`http://shakespeare.mit.edu`

We have downloaded thirty-seven `.txt` files from this site (one for each play) and placed them in a folder called `ThePlays`. You will see this folder (and other things) when you download and unzip `A6.zip`. The files have been tweaked a little bit to make the assignment more straightforward[2]. Here is an excerpt from `ROMEO AND JULIET.txt`:

[1]Mac: in the menu bar at the top of the screen; Windows: click on the three-stripe "hamburger" button (`https://en.wikipedia.org/wiki/Hamburger_button`).

[2]Do not be alarmed or concerned that (for example) `HAMLET.txt` differs from your private copy of the play.

```
FRIAR LAURENCE

    Come, come with me, and we will make short work;
    For, by your leaves, you shall not stay alone
    Till holy church incorporate two in one.

    Exeunt

ACT III
SCENE I. A public place.

    Enter MERCUTIO, BENVOLIO, Page, and Servants

BENVOLIO

    I pray thee, good Mercutio, let's retire:
    The day is hot, the Capulets abroad,
    And, if we meet, we shall not scape a brawl;
    For now, these hot days, is the mad blood stirring.

MERCUTIO

    Thou art like one of those fellows that when he
    enters the confines of a tavern claps me his sword
    upon the table and says 'God send me no need of
    thee!' and by the operation of the second cup draws
    it on the drawer, when indeed there is no need.


BENVOLIO

    Am I like such a fellow?

MERCUTIO

    Come, come, thou art as hot a Jack in thy mood as
    any in Italy, and as soon moved to be moody, and as
    soon moody to be moved.
```

The layout of the data is structured. Think of each line in each file as a separate string. If the string has a non-blank at the start, then it signals either a new act, a new scene, or the name of a "speaker". Using this fact it is possible to "repackage" the data in a Shakespearean-text file like `HAMLET.txt` so that the data sits in a more computationally-amenable "list of speeches" and a "list of scenes." To proceed further we need to define some handy classes.

3 The Classes `Speech` and `Scene`

Take a look at the module `PlayTools.py`. In it you will see that there are two class definitions and several functions. The class `Speech` is used to encode an uninterrupted utterance by someone, such as

```
MERCUTIO

    Thou art like one of those fellows that when he
    enters the confines of a tavern claps me his sword
    upon the table and says 'God send me no need of
    thee!' and by the operation of the second cup draws
    it on the drawer, when indeed there is no need.
```

(so in the excerpt given above, there are two speeches by Mercutio). A `Speech` object packages the speaker's name as a string and the spoken text as a list of strings. The class `Scene` is used to encode scene information from line pairs like

```
ACT III
SCENE I. A public place.
```

A `Scene` object packages the act number as an `int`, the scene number as an `int`, and the location as a string.

Continue your browse through `PlayTools.py`. The functions `theComedies()`, `theTragedies()`, and `theHistories()` return lists of strings that are the names of the plays. The functions `ListOfSpeeches` and `ListOfScenes` can be used to assemble all the speech information and act/scene information from a given play file.[3]

You should run the demo script `ShowPlayTools.py` to solidify your understanding of these important tools because you will be using them throughout the assignment.

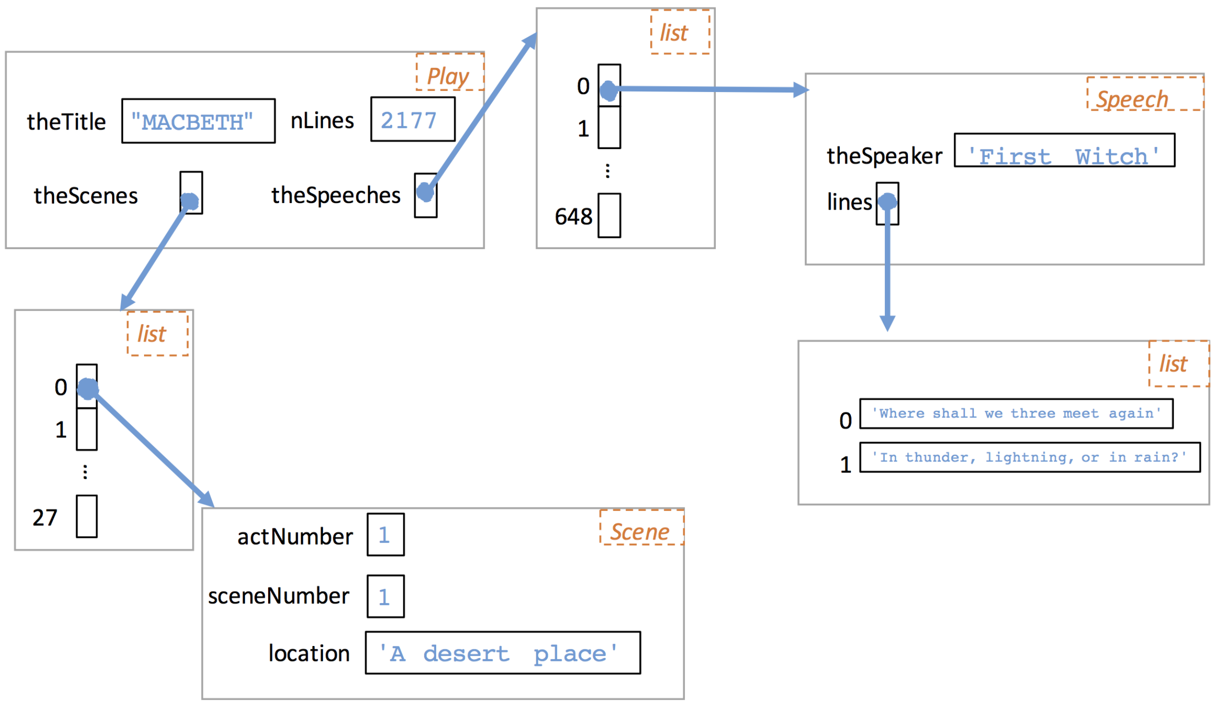
4 The Class `Play`

The module `ThePlayClass.py` contains a number of skeleton definitions that need completion. In the end you will submit `ThePlayClass.py` to CMS.

4.1 Implement the constructor

Start by completing the constructor (the function `__init__`) for the class `Play`. Make use of the functions `ListOfSpeeches` and `ListOfScenes` that are available for use because of the importing of `PlayTools.py`.

If you have your constructor working, your code will create `Play` objects that look like the following schematic. (We haven't drawn in all the arrows, but you should get the idea.)



As part of completing the constructor, note that you will need to compute the total number of lines in the play named by the string parameter `p`. To do this you will have to add up the number of lines in each speech.

4.2 Testing

We've provided a rough and partial test script for you called `CheckConstructor.py`. You won't be submitting it; it's just for your convenience.

[3] Actually, there are some formatting special cases that our code doesn't catch, since we opted for brevity over complete coverage. Don't spend effort on "fixing" these functions, just use them as is; but do be aware that some passages might get omitted by these functions. More information is given in the docstring code.
Note that real-life data is generally like this — kinda messa, but hopefully still useful.

If you run CheckConstructor.py , you should get a printout of (1) a report "Constructor test test_macbeth assertions passed", meaning that a preliminary rough check of your constructor run for the play MACBETH passed, and (2) a table like this that reports the number of lines in each of the tragedies:

```
3347 ANTONY AND CLEOPATRA
3585 CORIOLANUS
3607 HAMLET
2436 JULIUS CAESAR
3267 KING LEAR
2177 MACBETH
3388 OTHELLO
2847 ROMEO AND JULIET
2305 TIMON OF ATHENS
2344 TITUS ANDRONICUS
```

If our preliminary checks of your constructor fail, you will get some error messages printed out; consult CheckConstructor.py for more details.

Of course, the correctness of your constructor implementation requires testing in additional directions. Make sure that all the attributes are properly set up before proceeding.

4.3 If you have trouble debugging

One suggestion is to create a fake small play file — call it something like 'TEST.txt' — and place it in the directory/folder ThePlays. Perhaps the file could contain just the segment of ROMEO AND JULIET Act III Scene I that has been excerpted in Section 2; for that excerpt, the first speech by Mercutio is 5 lines long, and second one is three lines long.

With this small fake play around, you can alter CheckConstructor.py to use it. For instance, you could write a test function similar to the pre-existing function test_macbeth that creates a Play for TEST, e.g., with a line like my_play = Play("Test"). Then use print statements to check what the lines in each speech in my_play look like.

5 Sorting a List of Plays

Complete the module ShowSort.py so that it produces a table that reports the number of lines in each of the 37 plays. It should be sorted so that the play with the fewest number of lines is at the top of the table and the play with the most lines is at the bottom, e.g.,

```
1765 COMEDY OF ERRORS
2067 MIDSUMMERS NIGHTS DREAM
2077 TEMPEST
2147 TWO GENTLEMAN OF VERONA
2177 MACBETH

       :

3388 OTHELLO
3396 RICHARD III
3402 CYMBELINE
3585 CORIOLANUS
3607 HAMLET
```

You will have to define an appropriate getter function that is needed for the sort. (Make sure to give it a suitable docstring.) And you will have write a loop that iterates over a list of play names. To create this list make effective use of the functions theComedies(), theTragedies() and theHistories() that are available if you import everything from PlayTools.py. Submit ShowSort.py to CMS.

6 The Method `MajorParts`

If a speaker's name is all uppercase letters, like `'JULIET'`, then the speaker is a *major character*. Otherwise, the speaker is a *minor character* like `'Nurse'`. Revisit the module `ThePlayClass.py` and take a look at the class definition for `Play`. Complete the implementation of the method `MajorParts`:

```
def MajorParts(self):
    """ Returns an alphabetized list of strings that
    name all the major characters in this Play.
    """
```

As an example, the code

```
P = Play('ROMEO AND JULIET')
mParts = P.MajorParts()
for s in mParts:
    print s
```

produces

```
ABRAHAM
BALTHASAR
BENVOLIO
CAPULET
FRIAR JOHN
FRIAR LAURENCE
GREGORY
JULIET
LADY CAPULET
LADY MONTAGUE
MERCUTIO
MONTAGUE
PAGE
PARIS
PETER
PRINCE
ROMEO
SAMPSON
TYBALT
```

Here are some hints that you may find useful as you proceed to develop `MajorParts`. Build up a list (say `L`) of major characters through repeated appending. This will involve looking at the speaker associated with every speech in the play. If the speaker is *not* already in the "current" version of `L`, then append if they are a major character. Otherwise, there is nothing to do because the speaker is in `L` already. Note, the list returned by `MajorParts` should have no repeated entries. The script `CheckMajorParts.py` can be used to help check your implementation.

7 The Method `Freq`

Continue browsing through the `ThePlayClass.py` and turn your attention to the method `Freq` that is part of the class `Play`:

```
def Freq(self,w):
    """ Returns an int that is the number of times that w
    (as a word on its own) occurs in this Play. (If w is "th",
    "the" doesn't add to its count.)


    PreC: w is a string.
    """
```

By implementing this method you will be able to examine how many times Shakespeare penned a given word over the course of his playwriting career. Your code will have to look at every line in every speech. To process a particular line, use the function `stringToWordlist` that is part of `GetData.py`. It can be used to break a string into a list of words. Once that is done you can count how many times `w` is in that list. The module `CheckFreq.py` can be used to help check your implementation.

8 The Method `SpeakersAndLines`

Continue browsing through `ThePlayClass.py` and turn your attention to the method `SpeakersAndLines` that is part of the class `Play`:

```
    def SpeakersAndLines(self):
        """ Returns a dictionary whose keys are speakers and whose
        values are the total number of lines that the speaker has
        in this Play."""
```

Implement this method and complete the module `ShowSpeakersAndLines.py` so that it reports the number of speakers in each tragedy who have at least 200 lines. The output should look something like this:

```
ANTONY AND CLEOPATRA
     407   OCTAVIUS CAESAR
     344   DOMITIUS ENOBARBUS
     744   MARK ANTONY
     616   CLEOPATRA

CORIOLANUS
     285   COMINIUS
     237   BRUTUS
     299   SICINIUS
     263   AUFIDIUS
     203   MARCIUS
     566   MENENIUS
     308   VOLUMNIA
     611   CORIOLANUS

HAMLET
     273   HORATIO
    1308   HAMLET
     322   LORD POLONIUS
     437   KING CLAUDIUS

       etc
```

Submit `ShowSpeakersAndLines.py` to CMS.

Warning: make sure that your method `SpeakersAndLines` actually returns a dictionary; and that your `main` function in `ShowSpeakersAndLines.py` loops through a dictionary produced by `SpeakersAndLines` — and *not* through the list of speeches in a `Play` — in order to print its output. The point of this part of the exercise is to demonstrate your ability to work with dictionaries.

9 The Function `FullScene`

Run the module `ShowFullScene.py` and observe that it displays all the act/scene information for *Two Gentleman of Verona*:

```
TWO GENTLEMAN OF VERONA

Act 1  Scene 1   Verona
Act 1  Scene 2   The same
Act 1  Scene 3   The same
Act 2  Scene 1   Milan
Act 2  Scene 2   Verona
Act 2  Scene 3   The same
Act 2  Scene 4   Milan
Act 2  Scene 5   The same
Act 2  Scene 6   The same
Act 2  Scene 7   Verona
Act 3  Scene 1   Milan
Act 3  Scene 2   The same
Act 4  Scene 1   The frontiers of Mantua
Act 4  Scene 2   Milan
Act 4  Scene 3   The same
Act 4  Scene 4   The same
Act 5  Scene 1   Milan
Act 5  Scene 2   The same
Act 5  Scene 3   The frontiers of Mantua
Act 5  Scene 4   Another part of the forest
```

By saying "The same" Shakespeare is telling us that the location of a scene is the same as the location of the previous scene. We would like to display the same information with the locations in *full format*:

```
Act 1  Scene 1   Verona
Act 1  Scene 2   Verona
Act 1  Scene 3   Verona
Act 2  Scene 1   Milan
Act 2  Scene 2   Verona
Act 2  Scene 3   Verona
Act 2  Scene 4   Milan
Act 2  Scene 5   Milan
Act 2  Scene 6   Milan
Act 2  Scene 7   Verona
Act 3  Scene 1   Milan
Act 3  Scene 2   Milan
Act 4  Scene 1   The frontiers of Mantua
Act 4  Scene 2   Milan
Act 4  Scene 3   Milan
Act 4  Scene 4   Milan
Act 5  Scene 1   Milan
Act 5  Scene 2   Milan
Act 5  Scene 3   The frontiers of Mantua
Act 5  Scene 4   Another part of the forest
```

Complete `ShowFullScene.py` so that it displays both the original act/scene information and the full format version. You are required to implement the function `FullScene` and to make effective use of it in `main()`. Your implementation of `FullScene` should be able to work for any play–not just *Two Gentlemen of Verona*. Submit `ShowFullScene.py` to CMS.

10 CMS

Here is a summary of what you must upload to CMS:

```
ThePlayClass.py          This will have your implementation of the constructor
                         and the methods MajorParts, Freq, SpeakersAndLines

ShowSort.py

ShowSpeakersAndLines.py

ShowFullScene.py         This will include your implementation of the function
                         FullScene
```

All functions to be graded must be named *exactly* as mentioned above/defined in the skeleton code you have been provided, and taking exactly the number of parameters with the same meanings specified above/in the skeleton code. Any changes in capitalization, spelling, number of parameters, etc. will cause an error in the auto-testing code and probably result in point deductions for you.

Reminders about submission and CMS:

1. If you meant to be grouped, verify ahead of time that CMS group invitations were issued and accepted before submitting.
2. Make sure your submitted `.py` files begin with header comments listing:
   (a) The name of the file
   (b) The name(s) and netid(s) of the person or group submitting the file
   (c) The date the file was finished

   If there were other people who contributed to your thought processes in developing your code, mention them by name in header comments. (You don't have to mention course-staff members, although it makes sense to do so.)
3. Make sure all your functions have appropriate docstrings. These should include explanations of what the parameter variables "mean" and what preconditions (constraints) are assumed on their values, and what the user can expect as a result of calling your functions.
4. If you submit earlier than two hours before the submission deadline, CMS allows you to download your files to check that they are what you think you submitted. We thus strongly suggest that you submit a version early and do this check. (If you want to make changes, no problem: CMS allows you to overwrite an older version by uploading a new version.)
5. (We have heard that the following issue has been fixed, but better to be safe than sorry...) Do not "reload"/"refresh" the CMS assignment submission page after the submission deadline passes: this will trigger another upload of your files, which can result in your submission being counted as late. The safest policy is to close your browser tab/window after upload.