CS1110 Lab 9: More Practice for A6 (Apr 19-20, 2016)

First Name: _____ **Last Name**: _____ **NetID:** _____

The lab assignments are very important. Remember this: *The lab problems feed into the assignments and the assignments define what the exams are all about.*

Start *before* your lab meets.

We recommend spending an hour or two on the lab *before* coming to your section, so you can use your in-person time to ask questions most efficiently.

Also, this strategy of starting beforehand increases your chances of checking in at your lab section, which will probably take less time than waiting in line at consulting hours!

Getting credit

Complete all required blank boxes and lines on this handout. When you are finished, show your written answers to one of the CS 1110 lab staff in your section on April 19-20 or in any consulting hours up to and including April 25 (earlier days have shorter lines). The staff member will ask you a few questions to make sure you understand the material, and then swipe your Cornell ID card or directly make a notation in CMS to record your success. This physical piece of paper is yours to keep.

Getting set up

To maximize how this lab resonates with Assignment 6, you should browse through the A6 handout, especially §2 and §3. Do the necessary A6 downloading, run `ShowPlayTools.py` and look at the definition of the class `Speech` that is part of `PlayTools.py`.

From the Lab webpage, download and unzip `Lab_9.zip` into a folder named (for example) `Lab_9`. In the command shell, navigate the file system so that this folder is THE CURRENT WORKING DIRECTORY. Review the slides from the April 11 and April 13 lectures.

1 The Class `Speech`

The module `PlayToolsLab.py` is just like `PlayTools.py` except that it has three additional methods that you are to design and check out.

1.1 The Method `Condense`

Here is a speech

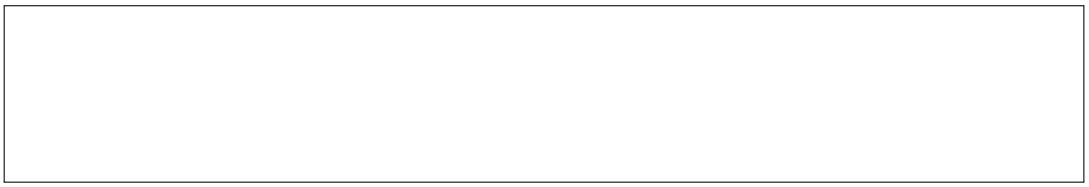
```
Messenger
      Much deserved on his part and equally remembered by
      Don Pedro: he hath borne himself beyond the
      promise of his age, doing, in the figure of a lamb,
      the feats of a lion: he hath indeed better
      bettered expectation than you must expect of me to
      tell you how.
```

and here is its "condensed" version

```
Messenger
      Much deserved on his part and equally remembered by
```

To condense a speech we throw away all but the first line. Add the following method to the class `Speech` that is defined in `PlayToolsLab.py`:

```
def Condense(self):
    """ Modifies self.lines so that it includes just the
    first line in the spoken text.
    """
```



To test, complete the implementation of `ShowCondense.py` so that it applies your `Condense` method to the first 10 speeches in *Much Ado About Nothing.* and then prints them out. *In the interests of time, you don't have to show your* `ShowCondense.py` *to a staff member, but we highly recommend that you do.*

1.2 The Method `NumQuestions`

Add the following method to the class `Speech` that is defined in `PlayToolsLab.py`:

```
def NumQuestions(self):
    """ Returns the total number of question marks that
    appear throughout self.lines.
    """
```

Complete the implementation of `ShowNumQuestions.py` so that it uses your `NumQuestions` method to compute and print the total number of question marks in *Hamlet*. The correct number is 373. *In the interests of time, you don't have to show your* `ShowNumQuestions.py` *to a staff member, but we highly recommend that you do.*

1.3 Tragedy Facts

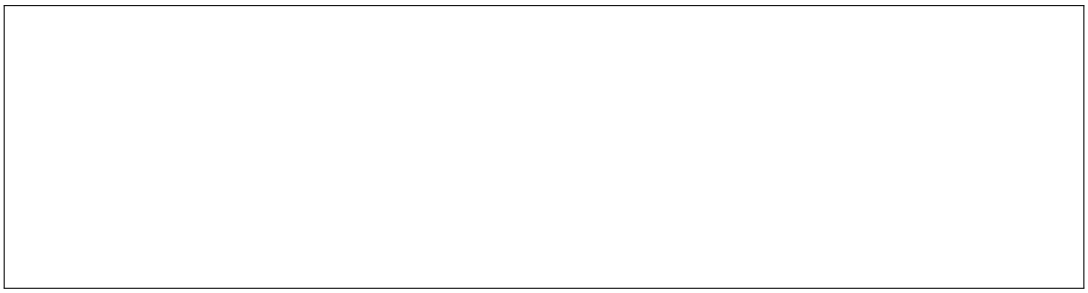
Take a look at the module `TragedyFacts.py` and complete its implementation in stages.

(a) Complete the first loop body so that it prints the names of the tragedies and the number of speeches in each. Testing hint: ANTONY AND CLEOPATRA has 1174; CORIOLANUS, 1105.

```
for p in theTragedies():
```

(b) Complete the second loop body so that it assigns to `N` the number of lines in the play `p`. Testing hint: ANTONY AND CLEOPATRA has 3347; CORIOLANUS, 3585.

```
for p in theTragedies():
    L = ListOfSpeeches(p)
```



```
    print p,N
```

2 The Class `Fraction`

Study the definition of the `Fraction` class that is given in `TheFractionClass.py`. Review how it can be used by browsing through and running `ShowFractionClass.py`.

Recall that we can approximate square roots by repeated averaging:

```
def sqrtF(a):
    """ Returns a float that approximates the square root
    of a obtained by doing 5 "rectangle averagings."

    PreC: a is a float with positive value
    """
    x = a
    half = 1./2.
    for k in range(6):
        x = (x + (a/x))*half
    return x
```

In `ShowFractionSqrt.py`, implement an analog of this that works when `a` and the returned approximation are `Fraction` objects. The intent is just for you to "translate" the mentions of floats into mentions of objects and mentions of operations on floats into method calls on objects.

```
def sqrtF(a):
    """ Returns a Fraction that approximates the square root
    of a obtained by doing 5 "rectangle averagings."

    PreC: a is a Fraction with positive value
    """
```

The module `ShowFractionSqrt.py` is set up for you to do this.

3 The Class `SimpleDate`

Study the definition of the `SimpleDate` class that is given in `TheSimpleDateClass.py`. Review how it can be used by browsing through and running `ShowSimpleDateClass.py`.

Run the module `OrderDates.py` and observe that it generates a list of 5 randomly generated `SimpleDates`. Modify the module so that it prints the list three times, once with it sorted by year, once with it sorted by month index, and once with it sorted by day. You will have to write three getter functions and apply the `sort` method three times.

Write down the code for one of the functions you wrote below (Could be as short as one header line and one other line of code, if you skip the docstring):

Write down your code that sorts the the list by year here. (Could be as short as one line.)

You don't have to show use your entire module `OrderDates.py`; filling in the boxes above suffices.