CSCI 1133, Fall 2017
Programming Examination 1

Part 1 Due: 5:00 pm, Friday, Oct 13
Part 2 Due: 5:00 pm, Friday, Oct 20

PLEASE NOTE: This is a graded assessment of individual programming understanding and ability, and is ***not*** a collaborative assignment; you must design, implement and test the solution(s) completely on your own without outside assistance from anyone. You may not consult or discuss the solution with anyone. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class. Note that *providing* a solution or assisting another student in any way on this examination is also considered academic misconduct. Failure to heed these directives will result in a failing grade for the course and an incident report filed with the Office for Student Conduct and Academic Integrity for further sanction.

A. (100 points) **Frequency Analysis**

This examination consists of two parts. The first part involves creating a short text file that must be committed and pushed to your GitHub examination repository (discussed below) prior to the Part 1 deadline on October 13. The second part requires designing and writing a single well-structured Python program that must be committed and pushed to your GitHub examination repository *prior* to the Part 2 deadline on October 20.

Late submissions will not be accepted and will result in a zero score for the exam.

TA help for this examination will not be provided. If you have clarification questions, they must be addressed to the graduate TA for the class.

The total point value will be awarded for solutions that are *complete*, *correct*, and *well structured*. A *"well structured"* program entails good design that employs functional decomposition, appropriate comments and general readability (descriptive names for variables and procedures, appropriate use of blank space, etc.) If you are not sure what this means, review the "well-structured" program requirements provided in Lab2.

Note that your work will be graded using, and must function correctly with, the current version of Python 3 on CSE Labs UNIX machines. If you complete this programming exam using a different system, it is *your* responsibility to ensure it works on CSELabs machines prior to submitting it.

The rubric includes the following specific (accumulative) point deductions:

• Missing academic integrity pledge	-100 points
• Syntax errors	-50 to -100 points
• Misnamed source file or incorrect repository	-25 points
• Use of global variables	-25 points
• Missing main program function	-25 points

Examination Repository

Examination files must be submitted to GitHub using a special "exam repository". Exam repositories have already been created for each registered student and are named using the string `exam-` followed by your X500 userID (e.g., `exam-smit1234`). You must first clone your exam repository in your local home directory before submitting the materials for this exam. If you are having difficulty, consult the second Lab from earlier in the semester or the GitHub tutorial on the class webpage. If your exam repository is missing or something is amiss, please contact the graduate TA. DO NOT SUBMIT YOUR EXAM FILES TO YOUR LAB/EXERCISE REPOSITORY!

Part 1 (*10 points*)

Using a text editor, type the following academic integrity pledge (exactly as it appears), replacing the last line with your full name and X500 ID. Save the text file with the name `academicpledge.txt` and commit/push it to your GitHub examination repository:

> I understand this is a graded, individual examination that may not be discussed with anyone. I also understand that obtaining solutions or partial solutions from outside sources or discussing any aspect of the examination with anyone will result in failing the course.
>
> < replace this line with your name and x500 ID (e.g., John Smith smit1234) >

If you do not commit and push the `academicpledge.txt` file prior to the deadline for Part 2, your entire examination solution will not be graded, resulting in a score of zero. In order to receive the 10 points for Part 1, you must submit the pledge file to your exam repository prior to the deadline for part 1.

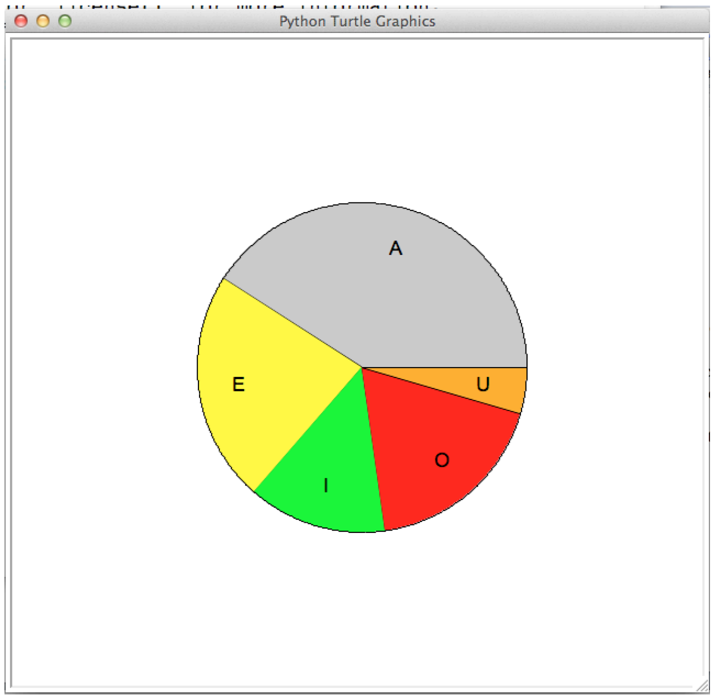
Part 2 (*90 points*)

Using Turtle graphics, write a well-structured Python program that will input a string of any length and count the occurrence of each of the vowels in the string: `a`, `e`, `i`, `o` and `u` (consider `y` a consonant in all cases). For example, in the string:

"My mama's llama pajamas are definitely not cool, especially when worn in public"

the letter `a` occurs 9 times, `e`, 6 times, `i`, 5 times, `o`, 4 times and `u` exactly once. Note that the vowel count is case "insensitive", i.e. the characters `'E'` and `'e'` represent the same vowel.

In order to "visualize" the distribution, your program will construct and display a proportional "pie chart" using Turtle graphics, e.g.,



Requirements:
Your program must do the following:

- Use turtle graphics to implement the graphical interface.
- Solicit an input string using the turtle `.textinput()` function.
- Include a pure function named `vowelCount(astring)` that takes a single string argument, determines the individual vowel frequencies and returns the counts in a Python list, each list element corresponding to the vowels: a, e, i, o and u in order.
- Include another non-pure function named `pieChart(flist)` that will accept a list of frequency counts and draw a colored pie chart as shown above. Note that in a "pie chart", the area of each colored wedge *proportionately* represents the frequency value. You may use any colors you wish.
- You can find descriptions of the `.textinput()` and `.write()` turtle functions (along with all the Turtle graphics module functions) here: *docs.python.org/3/library/turtle.html* .
- Use the turtle.`hideturtle()` method to hide the turtle.

Constraints:

- You may use any *built-in* Python object class methods (string, list, etc.)
- You may use *imported* functions and class methods from the `turtle` and `math` modules only
- You may use any built-in Python functions/operations as appropriate, with the exception of `input()` and `print()` which will not work with a graphical display environment.

Submission Instructions
Store all of your source code in a single module named `piechart.py` Make sure your files are named correctly! Missing or misnamed submissions will not be graded and will result in a grade of zero.
Commit/push your source file to your exam repository *prior* to the deadline for Part 2.

Helpful Hints:

- You will find this project much easier if you employ the principles of "top-down, functional decomposition" as discussed in lecture and implement your program as a collection of short, simple functions. For example, you might consider a separate function for drawing each "wedge" of the pie chart.
- A "proportional" scale implies that individual counts are *normalized* so that they sum to 1. e.g., for the following list of frequency values:

  2 , 4, 6

  the first value represents 2/12 of the *total* count, the second value represents 4/12 of the *total* count and the third value represents 6/12 of the *total* count. Note that this scale is easily determined by dividing each of the counts by the total count.