CSCI 1100 — Computer Science 1 Homework 3
Lists, While Loops and If statements

Overview

This homework is worth **90 points** toward your overall homework grade, and is due Thursday, October 12, 2017 at 11:59:59 pm. In the zip folder associated with this homework you will find `syllables.py`. The module `syllables.py` is written to help you compute the number of syllables in a word and is to be used in Part 1.

The goal of this assignment is to work with lists, use if statements and loops. As your programs get longer, you will need to develop some strategies for testing your program. Here are a few simple ones: start testing early, and test small parts of your program by writing a little bit and testing. We will walk you through program construction in the homework description and provide some ideas for testing.

As always, make sure you follow the program structure guidelines. You will be graded on program correctness as well as good program structure.

Fair Warning About Excess Collaboration

For HW 3 and for the remaining homework assignments this semester we will be using software that compares **all** submitted programs, looking for inappropriate similarities. This handles a wide variety of differences between programs, so that if you either (a) took someone else's program, modified it (or not), and submitted it as your own, (b) wrote a single program with one or more colleagues and submitted modified versions separately as your own work, or (c) submitted (perhaps slightly modified) software submitted in a previous year as your software, this software will mark these submissions as very similar. All of (a), (b), and (c) are beyond what is acceptable in this course — they are violations of the academic integrity policy. Furthermore, this type of copying will prevent you from learning how to solve problems and will hurt you in the long run. The more you write your own code, the more you learn.

Please read the **Collaboration Policy** document for acceptable levels of collaboration and how you can protect yourself. The document was/will be distributed during class and can also be found on the course resources page on **Piazza** `https://piazza.com/rpi/fall2017/csci1100/resources`. Penalties for excess collaboration can be as high as:

- 0 on the homework, and
- an additional overall 5% reduction on the semester grade.

Penalized students will also be prevented from dropping the course. More severe violations, such as stealing someone else's code, will lead to an automatic F in the course. A student caught in a second academic integrity violation will receive an automatic F.

By submitting your homework you are asserting that you both (a) understand the academic integrity policy and (b) have not violated it.

Finally, please note that this policy is in place for the small percentage of problems that will arise in this course. Students who follow the strategies outlined above and use common sense in doing so will not have any trouble with academic integrity.

Part 1: How complex is the language used in the text?

Create a folder for HW 3. Download the zip file `hw3Files.zip` from Piazza. Put it in this folder and unzip it. You should see a file named `syllables.py` which will be a helper module for this homework. Write your program in the same folder as this file and name it `hw3Part1.py`.

A few things to get familiar with before solving this part.

In this part, you must get familiar with a function called `.split()` that takes a piece of text, and converts it to a list of strings. Here is an example run:

```
>>> line = "Citadel Morning News. News about the Citadel in the morning, pretty self
    explanatory."
>>> m = line.split()
>>> m
['Citadel', 'Morning', 'News.', 'News', 'about', 'the', 'Citadel', 'in', 'the',
    'morning,', 'pretty', 'self', 'explanatory.']
```

You will also need to use the function `find_num_syllables()` from the file `syllabus.py` which takes as input an English word as a string and that returns the total number of syllables in that word as an integer. The module works even if the word has punctuation symbols, so you do not need to remove those explicitly. Make sure you import this module appropriately into your program.

```
>>> find_num_syllables("computer")
3
>>> find_num_syllables("science")
1
>>> find_num_syllables("introduction")
4
```

Clearly, the second result is incorrect. The module we provided is not a perfect implementation of syllabus count, so you may find errors. It is not your job to fix them, use the module as it is, with errors and all. Do not worry about the mistakes it makes. To properly compute this, we would need to use a Natural Language Processing (NLP) module like NLTK, which we have not installed in this course.

Problem specification.

In this part, you will read a paragraph as text from the user containing multiple English sentences. Assume a period marks the end of a sentence. Read the paragraph as a single (long) line of text. Compute and print the following measures corresponding to the overall readability of this text.

- `ASL` (average sentence length) is given by the number of words per sentence. Print `ASL`.
- `PHW` (percent hard words): To compute this first count the number of words of three or more syllables that do not contain a hyphen (`-`) and three-syllable words that do not end with `'es'` or `ed`. Divide this count by the total number of words in the text. Print `PHW`.
- Collect all words that are used in the `PHW` computation in a list exactly as they appear in the input, and print this list.

- `ASYL` (average number of syllables) is given by the total number of syllables divided by the total number of words. Print `ASYL`.
- `GFRI` is given by the formula `0.4*(ASL` PHW)+. Print `GFRI`.
- `FKRI` is given by the formula `206.835-1.015*ASL-86.4*ASYL`. Print `FKRI`.

Note that the measures `GFRI` and `FKRI` are slightly modified versions of well-known readability measures named Gunning-Fog and Flesch Kincaid. The higher the value is in each measure, the more difficult it is to read a text.

The following shows example runs of the program.

First example

```
Enter a paragraph => There is a theory which states that if ever anyone discovers
    exactly what the Universe is for and why it is here, it will instantly disappear and
    be replaced by something even more bizarre and inexplicable. There is another theory
    which states that this has already happened.
There is a theory which states that if ever anyone discovers exactly what the Universe
    is for and why it is here, it will instantly disappear and be replaced by something
    even more bizarre and inexplicable. There is another theory which states that this
    has already happened.

Here are the hard words in this paragraph:
['discovers', 'exactly', 'Universe', 'instantly', 'disappear', 'something',
    'inexplicable.', 'another', 'already']
Statistics: ASL:23.50 PHW:0.19 ASYL:1.64
Readability index (GFRI): 9.48
Readability index (FKRI): 41.43
```

Second example

```
Enter a paragraph => We hold these truths to be self-evident, that all men are created
    equal, that they are endowed by their Creator with certain unalienable Rights, that
    among these are Life, Liberty and the pursuit of Happiness.
We hold these truths to be self-evident, that all men are created equal, that they are
    endowed by their Creator with certain unalienable Rights, that among these are Life,
    Liberty and the pursuit of Happiness.

Here are the hard words in this paragraph:
['unalienable', 'Liberty', 'Happiness.']
Statistics: ASL:35.00 PHW:0.09 ASYL:1.51
Readability index (GFRI): 14.03
Readability index (FKRI): 40.48
```

When you are finished, submit your program to Submitty as `hw3Part1.py`. You must use this filename, or your submission will not work in Submitty. You do **not** have to submit any of the files we have provided.

Part 2: Population change now with bears

We are going to write a program to compute a type of population balance problem similar to the bunnies and foxes you computed in Lab 3. This problem will have bears, berry fields and the

tourists. We will just use the word berries to mean the area of the berry fields. We will count the numbers of bears and tourists as well.

Bears need a lot of berries to survive and get ready for winter. So the area of berry fields is very important part to their population. Berry fields in general spread over time, but if they are trampled too heavily by bears, then they may stop growing and may reduce in size. Tourists are the worst enemy of bears, often habituating them to humans and causing aggressive behavior. Sadly, this can lead to them being killed to avoid risk to human life.

Here is how the population of each group is linked to each other from one year to the next. Suppose the variable `bears` stores the number of bears in a given year and `berries` stores the area of the berry fields.

- The number of `tourists` in a given year is determined as follows. If there are less than 4 or more than 15 bears, there are no tourists. It is either not interesting enough or too dangerous for them.

  In other cases, there are about 10,000 tourists for each bear up to 10 and then 20,000 tourists for each additional bear. It is a great idea to write a function for computing tourists and test it separately.

- The number of `bears` and `berries` in the next year is determined by the following formulas given the population of bears, berries and tourists in the given year:

  ```
  bears_next = berries/(50*(bears+1))) + bears*0.60 - (log(1+tourists,10)*0.1)
  berries_next = (berries*1.5) - (bears+1)*(berries/14) - (math.log(1+tourists,10)*0.05)
  ```

  Remember none of these values can end up being negative. Negative values should be clipped to zero. Also, berries and tourists are integers. The log function is in the math module.

  You must write a function that takes as input bears, berries and tourists, and returns the next year's bear and berry populations as a tuple.

```
>>> find_next(5,1000,40000)
(6, 1071.1984678861438)
```

Problem Specification.

Write a program that reads two values, the current population of bears and the area of berry fields. Your program then finds and prints the population of all three groups (bears, berries, tourists) for this year 1 and the next 9 years and prints this information. You must use a loop to do this.

Once completed, your program should output: the smallest and largest values of the population of bears, berries and tourists reached in your computation.

The following is an example output of your program (values separated by a single TAB):

```
Number of bears => 5
5
Size of berry area => 200
200
Year     Bears  Berries   Tourists
1        5      200.0     50000
```

```
2       3       214.1     0
3       2       259.9     0
4       2       334.2     0
5       3       429.7     0
6       3       521.7     0
7       4       633.5     40000
8       4       723.8     40000
9       4       827.0     40000

Min:    2       200.0     0
Max:    5       827.0     50000
```

When you are finished, submit your program to Submitty as `hw3Part2.py`.

Notes: Yes, bears may go down to zero and come back up. Why? Bears from neighboring areas move in. The min values for each population may come from a different year.

Part 3: Real Steel

In this section, you will write a program that asks for the user for the name and location of a robot. The location of the robot will be given by an X and a Y coordinate (integer values). The robot initially has energy of 10.

Assume the robot moves within a grid, between points `(0,0)` (top left corner like in images) and `(100,100)` (bottom right corner). You can assume the initial location is a valid point in the grid. The robot cannot move outside of this grid, so if it comes to an edge, it will remain there. The energy of the robot is between 0 and 10, cannot be negative or higher than 10.

Once you read the robot, you will ask for an action for the robot from the user repeatedly until the user enters `end` (upper or lower case does not matter).

The following are the possible commands:

- `up,left,right,down` move the robot by 10 units in the given direction. To move, the energy of the robot should be at least 1. After the move, the robot's energy increases by one (maximum value is 10).
- `attack` does not move the robot, but the energy goes down by three. To attack, the energy must be at least three.

After each move, display the location of the robot and its energy level.

If the move entered by the user is not one of the valid moves and it is not `end`, then you will do nothing but still display the location of the robot. The commands should be entered correctly with no additional spaces. However, they can be upper or lower case.

Here is a potential run of this program:

```
Name of robot => Adam
Adam
X location => 20
20
Y location => 20
20
Robot Adam is at (20,20) with energy: 10
```

```
Enter a command (up/left/right/down/attack/end) => up
Robot Adam is at (20,10) with energy: 10
Enter a command (up/left/right/down/attack/end) => attack
Robot Adam is at (20,10) with energy: 7
Enter a command (up/left/right/down/attack/end) => attack
Robot Adam is at (20,10) with energy: 4
Enter a command (up/left/right/down/attack/end) => dosomething
Robot Adam is at (20,10) with energy: 4
Enter a command (up/left/right/down/attack/end) => up
Robot Adam is at (20,0) with energy: 5
Enter a command (up/left/right/down/attack/end) => left
Robot Adam is at (10,0) with energy: 6
Enter a command (up/left/right/down/attack/end) => up
Robot Adam is at (10,0) with energy: 7
Enter a command (up/left/right/down/attack/end) => down
Robot Adam is at (10,10) with energy: 8
Enter a command (up/left/right/down/attack/end) => right
Robot Adam is at (20,10) with energy: 9
Enter a command (up/left/right/down/attack/end) => end
```

When you are finished, submit your program to Submitty as `hw3Part3.py`.