The Mad Hatter Family
Programming Project 1
50 points

This is the story of the Sorting Hat's younger brother the Tay Sway Beret, which began in 1989 in the streets of New York. Nobody really talks much about Tay. . . It's kind of a blank space in the history of Wizarding. And as a result, no one really knows much about what Tay actually *does*. The prevailing theory is that, when placed on a head, he does nothing. And to be entirely honest, this makes sense. After all, if Tay did something useful, he probably would have had some sort of cameo in the series, don't you think?

Our goal in this project is to do different sorts of wizardly things, in the hopes that TSB will stop being a lazy beret and actually earn his keep. Yeah, we said it. He's a burden to the wizarding society, and frankly, who the hell wears a beret?

Job Details. This project tests your basic understanding of key concepts in Python, particularly functions, lists, and strings. Each question should be written in a separate `.py` file. (You can submit all of question 1 in a single file.) In addition, you will need to provide the output of the programs that you run. For that, you will need to copy and paste the thomas output to a notepad++ file and print that as well.

Question 1 – The Opposite Test

All parts of question 1 will require an input string that will contain only spaces, uppercase, and lowercase letters. You may not use any built-in string methods on any part of this question, with the exception of either `lower()`, or in its true non-Muggles name *reducio* or its counterpart `upper()` (aka *engorgio*). You know, because size matters.

You may find it necessary to cast your own spells (build your own functions) as an aid to the bigger effects needed for the project. Feel free to do so as you need.

Question 1a

[**5 points**] It is well known that Harry Potter was a crappy wizard and was hard carried through the entire plot by Hermione. One spell that Harry surely did not know was Priori Incantantem, which roughly speaking, is the "Reverse Spell." Since we are want to mimic the uselessness of the great HP, we shall consider input strings where reversing it does literally nothing. Just like Harry.

You must write a function called `priori` that takes an input string as defined above, strip its blank space characters in deference to the Tay Sway Beret, and determine whether or not it is a palindrome. (A palindrome is a string that reads the same both forwards and backwards.) Using this definition, an empty string is considered a palindrome. Your function should return `True` if it is a palindrome and `False` otherwise. For example, `m nm nm` is a palindrome.

Question 1b

[**10 points**] Write a function called `prioriincantantum` that determines whether or not the input string (after stripping spaces) is a case-insensitive palindrome. In other words, upper and lowercase versions of the same letter are considered the same. For example, `A bB a` is a case-insensitive palindrome.

Question 2 – The Measure of a Man

[**15 points**] It is likely that the Tay Sway Beret had applied for the position of the Sorting Hat, but much like Harry, was utterly useless and a waste of (blank) space. What he probably tried to do is figure out how many letters were in a person's name. And, although such knowledge is important, you don't need a damned magical hat to tell you how many letters you have in your name. But here we are. Working with a beret. Damn the French.

Given a list that consists of strings, write a function called `beret` that returns the average length of all the strings in the list. For this portion of the project, you are not allowed to use any built-in method from Python to determine the length of a string (for instance, `len()`). To test your program, you will need to create a list in the main program and send it in as a parameter to the function.

Question 3 – A Tale of Two Merges

[**20 points**] What is most likely about the Tay Sway Beret that instead of sorting, it knew how to merge. I probably don't need to tell you at this point how useless and pointless the task of merging two lists is. And in particular, it probably doesn't have any useful connection to sorting at all. So it's extra worthless. Really, in a sense, we're just taking the dregs of the wizarding society and just hoping that we end up with something or someone who is at least average.

Given two lists consisting of integers that are already in sorted (non-decreasing) order, write a function called `merge` that returns a new list that combines both lists such that they are in sorted order, and the resulting output has no duplicates. You cannot use the sort function from Python in any part of this question. Here are a couple of examples so that you can understand what the function is conceptually supposed to do:

```
[1, 32] + [2, 3, 12] -> [1, 2, 3, 12, 32]
[1, 4, 6, 6] + [1, 4, 4, 14] -> [1, 4, 6, 14]
```

We are going to write a function called `merge` that can deal with *really large* lists. And, since typing really large Python lists directly into our program is irritating, we're going to build those lists using an external file. Formally, your `merge` function must take as input two filenames, each of which contains a *representation* of a Python list (which we describe below). You will need to read the information from these files and build the Python list using the tools we learned in class. Your function will either output the merged list if its length is less than 25 items, otherwise it will output the length of the merged list (after it performs the merge). The format of these files is as follows:

```
<size of list>
<item 1 of list>
<item 2 of list>
...
```

For example, the first Python list from the second example above would have a text file that looks like:

```
4
1
4
6
6
```

How do you read in files? Here's how. Let's suppose you have a file called `list1.txt`. To read it in, you'll need to have a data type that represents that file. Here's how you would open the file:

```
filehandle = open("list1.txt")
```

To get the number from each line, you'll use a loop to iterate through the lines of the filehandle. You can think of filehandle as a "list" of lines if you prefer, but unlike a list, you can't ever go back to a previous line. So, as you iterate over the lines in the file, you'll need to save those values to an actual Python list if you wish to use them in the future. Here is some sample template code to set that up:

```
for line in filehandle:
   # Do the stuff you want to each line
```

Remember that the line data type is `str` just like it is when you use the `input()` command, so you will likely need to convert `line` to an integer the way we did in class before you can use it. You will need to test your merge program on two very large lists to make sure that it is done efficiently. (We'll talk about code efficiency more as the semester goes on.) I have created the files for you, called `list1.txt` and `list2.txt` and you can download them from `blue.butler.edu/~agupta/DS210/Projects/Project1SupplementaryCode/`. If you want to test your code on other large lists, you may create them by playing around with the `generatelists.py` script that is available at the same link.

1 Turn In Materials

For each question, you must submit a separate Python script (in Canvas) with the following things:

- The script itself
- A block comment at the end of the script that contains the output of the script on sample test runs of your script that showcase how it works. For question 3, this must include the output of your code for the provided `list1.txt` and `list2.txt` files.