CMSC 201 Fall 2015
Homework 6 – Strings and File I/O

Assignment: Homework 6 – Strings and File I/O
Due Date: Thursday, October 22nd, 2015 by 8:59:59 PM
Value: 4% of final grade

Homework 6 is designed to help you practice using file I/O, including reading to files, writing to files, and making use of string functions like **`split()`** to help parse the input. More importantly, you will be solving problems using algorithms you create and code yourself.

Remember to enable Python 3 before you run your programs:

```
/usr/bin/scl enable python33 bash
```

Instructions

Each one of these exercises should be completed in a **separate python file**. For this assignment, you may assume that all the input you get will be of the correct type (*e.g.,* if you ask the user for a whole number, they will give you an integer).

For this assignment, you'll need to follow the class coding standards, a set of rules designed to make your code clear and readable. The class coding standards are on Blackboard under "Course Documents" in a file titled "CMSC 201 - Python Coding Standards."
You will **lose major points** if you do not following the 201 coding standards.

A very important piece of following the coding standards is writing a complete **file header comment block**. Make sure that each file has a comment block at the top (see the coding standards document for an example).

NOTE: **You must use `main()` in each of your files.**

Details

Homework 6 is broken up into three parts. **Make sure to complete all 3 parts.**

NOTE: Your filenames for this homework must match the given ones exactly.
And remember, filenames are case sensitive.

hw6_part1.py
For this part of the homework you will write code that creates a simple text file, called **`names.txt`**, based on input from the user.

You will prompt the user for a name, and will continue to accept new names until they enter "**`DONE`**". For each of these names, you should print a line saying "**`Hello Name!`**" to the **`names.txt`** file.
You do not need to change anything about the name (*e.g.*, names do not need to be changed to start with a capital letter.) The name may also be more than word, and your program should accept that.

(HINT: Make sure to open the file for writing, and to close it when you're done.)

You can check the contents of the file **`names.txt`** by opening it in **`emacs`**:

```
emacs names.txt
```

Or by using the "**`more`**" command to print the contents to the terminal:

```
more names.txt
```

(If your file is very long, it will display one "screen" worth of text at a time. You can move to the next screen by hitting the space bar, or you can quit immediately by typing "**`q`**" – you don't need to hit enter afterwards.)

Sample output for this problem can be found on the next page.

Here is the **sample output** for hw6_part1.py, with the user input in blue. Your output does not need to be identical, but should be similar.

```
bash-4.1$ python hw6_part1.py
Please enter a name (or "DONE" to stop): Aya
Please enter a name (or "DONE" to stop): Brandon
Please enter a name (or "DONE" to stop): CARLY!!!
Please enter a name (or "DONE" to stop): David
Please enter a name (or "DONE" to stop): e e cummings
Please enter a name (or "DONE" to stop): Francoise
Please enter a name (or "DONE" to stop): General Patton
Please enter a name (or "DONE" to stop): Hrabowski
Please enter a name (or "DONE" to stop): DONE

bash-4.1$ more names.txt
Hello Aya!
Hello Brandon!
Hello CARLY!!!!
Hello David!
Hello e e cummings!
Hello Francoise!
Hello General Patton!
Hello Hrabowski!
```

hw6_part2.py

For this part of the homework, you will calculate a weighted grade from data contained in a file. *(REMINDER: A weighted total is computed by multiplying each grade by the corresponding weight and adding them together.)*

(WARNING! *This part of the homework is the **most challenging**, so budget plenty of time and brain power. And read the instructions carefully!)*

Your program should first prompt the user for the name of the file they want to read the data from. (You can assume that the file the user provides exists in the current directory. In other words, it will open successfully.)
After you open the file, you will use the data inside (weights and scores) to calculate and print out the user's final weighted score.

The file will be formatted as follows (examples below and on next page):

- One or more lines of numbers, separated by spaces
- On each line:
  - First number is a decimal (the weight for that type of assignment)
    - You do <u>not</u> need to check that the weights add up to 1
  - All following numbers are integers (there will be at least 1)
    - (The different scores for that type of assignment)
    - You do <u>not</u> need to check that the scores are valid
- May have different amounts of scores for each assignment type
- May have any number of assignment types (but at least 1)

For example, the line

```
    0.45   72 100   58   44   93   89   78   92
```

means that this type of assignment is worth 45% of the total grade. There are eight total assignments of this type.
We can calculate the average of these eight assignments to be 78.25. If we multiply 78.25 by the weight, we can calculate this part of the weighted grade to be 35.2125 %. (See the sample input below for more examples.)

(HINT: Make sure to open the file for reading, <u>and to close it</u> when you're done.)

HINT: Make sure you use the filename provided by the user when opening the file. We will test your code with files that aren't named **`grades.txt`**.

<u>PROTIP</u>: This would be a good time to use incremental programming!
Incremental development is when you are only working on a small piece of the code at a time, and testing that the piece of code works before moving on to the next piece. This makes it a lot easier to fix any mistakes.
For example, for this problem, you might first write the code to get and use the name of the file, and test that this works before moving on. Next, you might write the code to read each line, and test that this works before moving on. Then, you might write the code to read in and store the weight and the scores on each line, and test that this works before moving on. etc…

Here is some **<u>sample output</u>**, with the user input in blue. (Please note that the line containing the user input has wrapped around from the previous line.)

```
bash-4.1$ python hw6_part2.py
Please enter the name of the file that contains the
grades: grades.txt
Your final weighted score is 79.5
```

Here is the **<u>sample input file</u>**, **`grades.txt`**, that was used to create the sample output above.
In this sample input, the user has 3 assignments that together are worth 70% of the grade; 10 assignments that together are 20% of the grade, and 5 assignments that together are worth 10% of the grade.

```
0.7 70  89  84
0.2 70  52  85  91  77  0  70  88  100  47
0.1 100 96  88  84  92
```

You can **directly download the file** using the "**`cp`**" command. The command below will copy the file "**`grades.txt`**" from my public directory to your current directory. The period at the end ("**`.`**") means that the file will have the same name after you copy it, so **`grades.txt`** will be the copied file's name. <u>Make sure to run the command from the folder you want the file to be copied into!</u>

`cp /afs/umbc.edu/users/k/k/k38/pub/cs201/grades.txt .`

hw6_part3.py
Finally, you will write a program that takes in the name of a file from the user and calculates the total number of words in the file, as well as the average word length.

This time, the filename you get from the user must be **checked for validity**: the filename must end in either "**`.dat`**" or "**`.txt`**" in order to be considered a valid filename for this program. If the user inputs an invalid filename (*e.g.*, "**`book.doc`**") you must continue to prompt them until they give a valid filename.

(HINT: If an invalid filename is given, your program should also tell the user what a valid filename looks like – see the sample output for an example.)

(HINT: Some of the code you wrote for hw4_part3.py *may help you to check if the end of the filename is valid.)*

Once you have a valid filename from the user, you should open the file and count the total number of words in the file, as well as calculating the average word length.

You may assume:

- A filename that ends in "**`.dat`**" or "**`.txt`**" will successfully open a file when used with the function **`open()`**
- <u>A "word" in a file is any set of characters separated by a **space**</u>
    - For example, "**`copy-right 1977 by author of book`**" would be six words ("**`copy-right`**" is a single word and the number "**`1977`**" counts as a word). You don't need to do any checking of word contents.

(HINT: Make sure to open the file for reading, <u>and to close it</u> when you're done.)

Sample output for this problem can be found on the next page.

Here is the **sample output** for hw6_part3.py, with the user input in blue. Your output does not need to be identical, but should be similar.

```
bash-4.1$ python hw6_part3.py
Please enter the name of the file to open: book.text
        The file must end in .txt or .dat to be valid.
Please enter the name of the file to open: myFile.doc
        The file must end in .txt or .dat to be valid.
Please enter the name of the file to open: shelley.txt
The file shelley.txt has 77986 words in it.
On average, each word is 4.60644218193009 characters long.
```

The **sample input file** that was used to create the sample output above, **`shelley.txt`**, is much too long to include in this document. However, you can **directly download the file** using the "**`cp`**" command.

The command below will copy the file "**`shelley.txt`**" from my public directory to your current directory. The period at the end ("**`.`**") means that the file will have the same name after you copy it, so **`shelley.txt`** will be the copied file's name. Make sure to run the command from the folder you want the file to be copied into!

`cp /afs/umbc.edu/users/k/k/k38/pub/cs201/shelley.txt .`

Submitting

Once all three parts of your Homework 6 are complete, it is time to turn them in with the **`submit`** command.

Don't forget to complete the header block comment for each file! Make sure that you updated the header block's file name and description for each file.

You must be logged into your GL account, and you must be in the same directory as the Homework 6 files. To double check this, you can type **`ls`**.

```
linux1[3]% ls
hw6_part1.py   hw6_part2.py   hw6_part3.py
linux1[4]%
```

To submit your files, we use the **`submit`** command, where the class is **`cs201`**, and the assignment is **`HW6`**. Type in (all on one line)
`submit cs201 HW6 hw6_part1.py hw6_part2.py hw6_part3.py`
and press enter.

```
linux1[4]% submit cs201 HW6 hw6_part1.py hw6_part2.py
hw6_part3.py
Submitting hw6_part1.py...OK
Submitting hw6_part2.py...OK
Submitting hw6_part3.py...OK
linux1[5]%
```

If you don't get a confirmation like the one above, check that you have not made any typos or errors in the command.

You can **double-check that all three homework files were submitted** by using the **`submitls`** command. Type in **`submitls cs201 HW6`** and hit enter.

And you're done!