CMSC 201 Spring 2018

Homework 6 – Recursion & File I/O

Assignment: Homework 6 – Recursion
Due Date: Friday, April 27th, 2018 by 8:59:59 PM
Value: 40 points

Collaboration: For Homework 6, collaboration is allowed. Make sure to consult the syllabus about the details of what is and is not allowed when collaborating. You may not work with any students who are not taking CMSC 201 this semester.

If you work with someone, remember to note their name, email address, and what you collaborated on by filling out the Collaboration Log.

You can find the Collaboration Log at http://tinyurl.com/collab201-Sp18.

Remember that all collaborators need to fill out the log each time; even if the help was only "one way" help.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
# File:    FILENAME.py
# Author:  YOUR NAME
# Date:    THE DATE
# Section: YOUR DISCUSSION SECTION NUMBER
# E-mail:  YOUR_EMAIL@umbc.edu
# Description:
#    DESCRIPTION OF WHAT THE PROGRAM DOES
```

You can get all of the files used in the sample output for parts 4 and 5 using the following command from within your hw6 folder:

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/hw6_* .
```

Instructions

For each of the questions below, you are given a problem that you must solve or a task you must complete.

Objective

Homework 6 is designed to give you lots of practice with recursion and file input/output. For parts 1, 2, and 3, you should think carefully about the base case(s), recursive case(s), and recursive call(s) that each problem will need. For parts 4, 5, and 6, remember to close the file when you're done!

If your solution to part 1, 2, or 3 does not use recursion, you will earn zero points for that question, even if your code solves the problem.

Remember to enable Python 3 before running and testing your code:

```
scl enable python33 bash
```

Coding Standards

Prior to this assignment, **you should be familiar with the entirety of the Coding Standards**, available on Blackboard under "Assignments" and linked on the course website at the top of the "Assignments" page.

You should be commenting your code, and using constants in your code (not magic numbers or strings).
Any numbers other than 0 or 1 are magic numbers!

You will **lose major points** if you do not follow the 201 coding standards.

If you have questions about commenting, whitespace, or any other coding standards, please come to office hours.

Additional Instructions – Creating the hw6 Directory

Just as you did for previous homeworks, you should create a directory to store your Homework 6 files. We recommend calling it **`hw6`**, and creating it inside the **`Homeworks`** directory inside the **`201`** directory.

Questions

Each question is worth the indicated number of points. Following the coding standards is worth 4 points. Failing to have complete file headers or failing to have correctly named files will lead to a deduction of points.

hw6_part1.py **(Worth 4 points)**

For this part of the homework, you will write a recursive function that counts the number of tails in a line of Pembroke and Cardigan Welsh Corgis. (Cardigan corgis have a tail, while Pembroke corgis do not.) The dogs alternate in the line, with a Cardigan in the first position, a Pembroke in the second, etc. The user will enter the length of the line of canines; it is guaranteed to be a positive number (zero or higher).

The program must contain a **`main()`** and a recursive function called **`countTails()`**. The **`countTails()`** function must take in a single integer (the line length) and, through recursion, return the total number of tails. The program may also contain any other functions you deem necessary.

Here is some sample output, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
bash-4.1$ python hw6_part1.py
How long is the line of corgis? 1
In a line of 1 corgis, there are 1 tails.

bash-4.1$ python hw6_part1.py
How long is the line of corgis? 2
In a line of 2 corgis, there are 1 tails.

bash-4.1$ python hw6_part1.py
How long is the line of corgis? 37
In a line of 37 corgis, there are 19 tails.

bash-4.1$ python hw6_part1.py
How long is the line of corgis? 987
In a line of 987 corgis, there are 494 tails.
```

hw6_part2.py **(Worth 10 points)**

Next, you will create a program that is the recursive version of HW4's part 1: drawing a right triangle, using a height and symbols provided by the user.

The program must contain a **`main()`** and a recursive function called **`recurTri()`**. The program may also contain any other functions you deem necessary.

For these inputs, you can assume the following:

- The height will be a positive integer (zero or greater)
- The symbols will each be a single character

Here is some sample output, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
bash-4.1$ python hw6_part2.py
Please enter the height of the triangle: 3
Please enter the outline symbol to use: X
Please enter the fill    symbol to use: o
X
XX
XXX

bash-4.1$ python hw6_part2.py
Please enter the height of the triangle: 10
Please enter the outline symbol to use: .
Please enter the fill    symbol to use: Q
.
..
.Q.
.QQ.
.QQQ.
.QQQQ.
.QQQQQ.
.QQQQQQ.
.QQQQQQQ.
..........
```

hw6_part3.py **(Worth 6 points)**

Create a program that determines all of the factors of a positive number. The program must contain a **`main()`** and a recursive function, the name of which is up to you.

The number entered by the user is guaranteed to be positive (greater than zero). The factors must be printed out from smallest to largest.

Here is some sample output, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
What number do you wish to know the factors for? 1
1

bash-4.1$ python hw6_part3.py
What number do you wish to know the factors for? 15
1
3
5
15

bash-4.1$ python hw6_part3.py
What number do you wish to know the factors for? 176
1
2
4
8
11
16
22
44
88
176

bash-4.1$ python hw6_part3.py
What number do you wish to know the factors for? 201
1
3
67
201
```

hw6_part4.py **(Worth 6 points)**

Create a program that takes in the name of a file from the user and calculates the total number of words in the file, the average word length, and the total number of sentences.

For this part of the homework, 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 reprompt 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.)

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

You may assume:

- A filename that ends in "**`.dat`**" or "**`.txt`**" will exist in the directory
- A "word" in a file is any set of characters separated by **whitespace**
  - 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). Tabs and newlines count as whitespace. You do not need to do any checking of a word's content being valid.
- A sentence in a file is any set of characters separated by a **period**.

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

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

Here is the sample output for **hw6_part4.py** with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
bash-4.1$ python hw6_part4.py
Please enter the name of the file to open: hw6_directs.txt
The file hw6_directs.txt has 59 words in it.
On average, each word is 4.6440677966101696 characters
long.
There are 8 sentences in the file.

bash-4.1$ python hw6_part4.py
Please enter the name of the file to open: hw6_shelley.txt
The file hw6_shelley.txt has 77986 words in it.
On average, each word is 4.60644218193009 characters long.
There are 3125 sentences in the file.

bash-4.1$ python hw6_part4.py
Please enter the name of the file to open: hw6_zimmer.doc
        The file must end in .txt or .dat to be valid.
Please enter the name of the file to open: hw6_zimmer.text
        The file must end in .txt or .dat to be valid.
Please enter the name of the file to open: hw6_zimmer_txt
        The file must end in .txt or .dat to be valid.
Please enter the name of the file to open: hw6_zimmer.txt
The file hw6_zimmer.txt has 48925 words in it.
On average, each word is 4.679979560551865 characters
long.
There are 3347 sentences in the file.
```

hw6_part5.py **(Worth 10 points)**

The last program will take in the name of a file from the user and calculate some basic information about it, before printing out all of the information and calculated details to the user.

For this part of the homework, you can assume that the filename provided will be valid, and that the data inside will be well-formatted (*i.e.*, free of errors). Each line of the file will use the following format:

```
Dog Breed Name,numBorn,numAdopted
```

The dog breed name is guaranteed to be no more than 22 letters, and the numbers are guaranteed to be no higher than 9999 (four digits).

The information printed out to the screen must be the dog breed name, the number of dogs adopted, the number of dogs born, and the percentage of dogs adopted from those born. There must also be column headings printed at the top of the list. (See the sample output for an example.)

To make the task a bit simpler, you are allowed to use magic numbers, but ***only*** when formatting a string. Other numbers (like which position the breed name is stored in, for example), must be made into constants.

PROTIP: **This would be a good time to use incremental development!**
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 open and read in the file, and test that this works before moving on. Next, you might write the code to calculate the percentages, and test that this works before moving on.

Here is the sample output for **hw6_part5.py** with the user input in **blue**.
(Your output should look *exactly* the same as the output below.)

```
bash-4.1$ python hw6_part5.py
Please enter the name of the file to open: hw6_dogs.txt
Breed Name..... -- Adopted / Born -- Percentage
Airedale Terrier..... --     305 /  372 --  81.99%
Briard............... --     138 /  438 --  31.51%
Ches. Bay Retriever... --    1918 / 1918 -- 100.00%
Dachshund............ --     784 /  815 --  96.20%
Eurasier............. --     127 /  324 --  39.20%
French Brittany...... --      76 /  264 --  28.79%
Great Dane........... --     178 /  200 --  89.00%
Huntaway............. --      47 /   74 --  63.51%
Irish Setter......... --      41 /  297 --  13.80%
Japanese Spitz....... --     355 /  685 --  51.82%
Kangal Dog........... --     148 /  438 --  33.79%
Leonberger........... --     155 /  224 --  69.20%
Maremma Sheepdog..... --     183 /  228 --  80.26%
Newfoundland......... --      96 /  204 --  47.06%
Otterhound........... --      58 /   68 --  85.29%
Papillon............. --     447 /  623 --  71.75%
Queensland Heeler.... --     197 /  371 --  53.10%
Rottweiler........... --      85 /  314 --  27.07%
Saluki............... --      74 /  192 --  38.54%
Tosa................. --      76 /   85 --  89.41%
Utonagan............. --     112 /  118 --  94.92%
Vizsla............... --      50 /  174 --  28.74%
White Shepherd....... --     303 /  646 --  46.90%
Xoloitzcuintli....... --     227 /  536 --  42.35%
Yorkshire Terrier.... --     670 /  920 --  72.83%
Zuchon............... --       0 /   35 --   0.00%
```

Submitting

Once your **hw6_part1.py**, **hw6_part2.py**, **hw6_part3.py**, **hw6_part4.py**, and **hw6_part5.py** files are complete, it is time to turn them in with the **submit** command. (You may also turn in individual files as you complete them. To do so, only **submit** those files that are complete.)

You must be logged into your account on GL, and you must be in the same directory as your Homework 6 Python files. To double-check you are in the directory with the correct files, you can type **ls**.

```
linux1[3]% ls
hw6_part1.py  hw6_part3.py  hw6_part5.py
hw6_part2.py  hw6_part4.py
linux1[4]%
```

To submit your Homework 6 Python 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 hw6_part4.py hw6_part5.py** and press enter.

```
linux1[4]% submit cs201 HW6 hw6_part1.py hw6_part2.py
hw6_part3.py hw6_part4.py hw6_part5.py
Submitting hw6_part1.py...OK
Submitting hw6_part2.py...OK
Submitting hw6_part3.py...OK
Submitting hw6_part4.py...OK
Submitting hw6_part5.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 check that your homework was submitted by following the directions in Homework 0. Double-check that you submitted your homework correctly, since **an empty file will result in a grade of zero for this assignment.**