CMSC 201 Fall 2017

Homework 5 – Functions and Strings

Assignment: Homework 5 – Functions and Strings
Due Date: Friday, October 13th, 2017 by 8:59:59 PM
Value: 40 points

Collaboration: For Homework 5, 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-Fa17.

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
```

Instructions

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

This assignment will focus on using functions to make code that is more modular. Some of these functions may be useful in later assignments!

At the end, your Homework 5 files must run without any errors.

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

Part 6 of this homework requires that you use `return`, which will not be covered until Lecture 11. (The slides will be on the website by Monday at noon if you don't want to wait until your class meets.)
None of the other parts of the homework should use `return`!

Additional Instructions – Creating the hw5 Directory

During the semester, you'll want to keep your different Python programs organized, organizing them in appropriately named folders (also known as directories).

Just as you did for previous homeworks, you should create a directory to store your Homework 5 files. We recommend calling it **`hw5`**, and creating it inside the **`Homeworks`** directory inside the **`201`** directory.

If you need help on how to do this, refer back to the detailed instructions in Homework 1. (You don't need to make a separate folder for each file. You should store all of the Homework 5 files in the same **`hw5`** folder.)

Coding Standards

Prior to this assignment, you should re-read the Coding Standards, available on Blackboard under "Assignments" and linked on the course website at the top of the "Assignments" page.

For now, you should pay special attention to the sections about:

- Comments
  - ***Function header comments***
  - We've given a few of the function headers as examples, but you will need to create the rest on your own
    - The first five problems will have an output of "None"
- Constants
  - For Homework 5, you must use constants instead of magic numbers!!! Magic strings are also forbidden!!!!!!
- Make sure to **read the last page of the Coding Standards document**, which prohibits the use of certain tools and Python keywords
  - Also, this section states that **you may not use built-in functions** or concepts we haven't covered in class!
    If you use a built-in function to solve a problem, you will earn **zero points** for that entire problem.

Additional Specifications

For this assignment, **you must use** `main()` as seen in your `lab2.py` file, and as discussed in class.

For this assignment, you should pay attention to each problem's instructions on using "input validation." For example, the user may enter a negative value, but your program may require a positive value. **Make sure to follow each part's instructions about input validation.**

If the user enters a different type of data than what you asked for, your program may crash. This is acceptable.

You must create the functions specified in each problem, and they must be named ***exactly*** as shown.

Questions

Each question is worth the indicated number of points. Following the coding standards is worth 4 points. If you do not have complete file headers and correctly named files, you will lose points.

hw5_part1.py **(Worth 4 points)**

For this part of the homework you will create a program that asks the user for a string and a letter, and then counts how many times that letter appears in the string. You must create a function called **`numLetter()`** that handles counting the number of times the letter appears, and prints out the results. It must be case *in*sensitive.

The program must contain a **`main()`** and a function called **`numLetter()`**, implemented as described in the function header comment given below. (You should include this function header comment in your own code.)

```
############################################################
# numLetter() counts the instances of a letter in a string
# Input:      aString; a string of the phrase to search in
#             letter;  a single character to search for
# Output:     None
```

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

```
bash-4.1$ python hw5_part1.py
Enter a string: Dogs are good, dogs are great.
Enter a letter to search for: d
There are 3 instances of d in the string

bash-4.1$ python hw5_part1.py
Enter a string: Peter Piper picked a peck of pickled
peppers!
Enter a letter to search for: P
There are 9 instances of P in the string

bash-4.1$ python hw5_part1.py
Enter a string: act tta aga agg aga tat acc atg ggt tct
Enter a letter to search for: G
There are 7 instances of G in the string
```

hw5_part2.py **(Worth 5 points)**

For this part of the homework, you will write code to search through a phrase, looking for occurrences of a word. Each time an occurrence is found, the program must report the index at which the word starts as well. After searching, it must print out the total number of times word was found.

The program must be case *in*sensitive, and must use a function called **`inPhrase()`** that takes in the phrase and word as its two formal parameters. All of the searching and printing of results must happen inside the **`inPhrase()`** function.

Inside **`main()`**, the program must ensure that the word is shorter than the phrase, re-prompting the user as necessary. You may assume that both inputs will be at least 1 character.

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

```
bash-4.1$ python hw5_part2.py
Please enter a phrase: Dogs are good, dogs dogs dogs
Please enter a word to search for: DOGS
Found DOGS at index 0
Found DOGS at index 15
Found DOGS at index 20
Found DOGS at index 25
Found DOGS a total of 4 times

bash-4.1$ python hw5_part2.py
Please enter a phrase: Hello!
Please enter a word to search for: Good morning
The word cannot be longer than the phrase.
Please enter a shorter word to search for: Goodbye
The word cannot be longer than the phrase.
Please enter a shorter word to search for: ello
Found ello at index 1
Found ello a total of 1 times
```

hw5_part3.py **(Worth 6 points)**

Create a program that checks the grammar of a sentence: it must start with a capital letter, and it must end with `?`, `!`, or `.` (a question mark, exclamation mark, or period).

The program must use two functions for this: **`checkCapital()`** and **`checkPunctuation()`**, both of which take in the complete sentence as their only formal parameter. Each function must print out whether the sentence passes or fails their requirement.

Within **`main()`**, the user can continue entering sentences to check until they want to quit, which they do by entering an empty string (hitting Enter without typing anything).

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 hw5_part3.py
Enter a sentence (enter nothing to quit): dogs are good
WRONG - Sentences start with a capital letter.
WRONG - Sentences use punctuation.
Enter a sentence (enter nothing to quit): dogs are good.
WRONG - Sentences start with a capital letter.
Correct punctuation!
Enter a sentence (enter nothing to quit): Dogs are good
Correct capitalization!
WRONG - Sentences use punctuation.
Enter a sentence (enter nothing to quit): Dogs are good!
Correct capitalization!
Correct punctuation!
Enter a sentence (enter nothing to quit): Dogs are good?
Correct capitalization!
Correct punctuation!
Enter a sentence (enter nothing to quit): I!
Correct capitalization!
Correct punctuation!
Enter a sentence (enter nothing to quit):
```

hw5_part4.py **(Worth 9 points)**

Write a program that asks the user to enter a number, and then checks different properties of the number using functions.

The program must contain a **`main()`**, as well as the following 3 functions:

- **`checkOddOrEven()`**
  - Task: check if number is odd or even
  - Input: userNum, an integer
  - Output: None (prints out results to the screen)

- **`checkPositive()`**
  - Task: check if number is negative, positive, or zero
  - Input: userNum, an integer
  - Output: None (prints out results to the screen)

- **`checkDivisible()`**
  - Task: check if the given number is divisible by a second number
  - Input: userNum, an integer
    - The second number should be asked for within the function itself, and is not passed in as a second formal parameter.
  - Output: None (prints out results to the screen)

Within **`main()`**, the user is asked for a number, and then each of the three functions above are called, in the order they are shown.

Below is an example function header comment for **`checkPositive()`**.

```
############################################################
# checkPositive() prints if the given number
#                 is positive, negative, or zero
# Input:          userNum; an integer chosen by the user
# Output:         None
```

(See the next page for sample output.)

Here is some sample output for **hw5_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 hw5_part4.py
Enter the number you would like to check: 18
18 is even
18 is positive
Enter number to divide by: 2
18 is divisible by 2

bash-4.1$ python hw5_part4.py
Enter the number you would like to check: -77
-77 is odd
-77 is negative
Enter number to divide by: -18
-77 is not divisible by -18

bash-4.1$ python hw5_part4.py
Enter the number you would like to check: 0
0 is even
0 is zero
Enter number to divide by: 5
0 is divisible by 5

bash-4.1$ python hw5_part4.py
Enter the number you would like to check: 35
35 is odd
35 is positive
Enter number to divide by: 4
35 is not divisible by 4

bash-4.1$ python hw5_part4.py
Enter the number you would like to check: -123456789
-123456789 is odd
-123456789 is negative
Enter number to divide by: 3
-123456789 is divisible by 3
```

hw5_part5.py **(Worth 6 points)**

Write a program that, within **`main()`**,

1. Asks for the number of words the user will enter into the program
2. Then prompts for each of the words (and stores them in a list)

Once all of the words have been entered, the program must print them out in reverse order from how they were entered, and must also show what the word would be backwards. (See the sample output for an example.)

The program must use a function called **`backwards()`**, that works as specified in the function header provided below.

```
##################################################
# backwards() reverses a string and prints the result
# Input:      forString, a string to reverse
# Output:     None
```

(Again, do not use any built-in function or "trick" that circumvents the point of this assignment, or you will earn zero points. If you're not using a loop to create the backwards string, you're doing it wrong.)

(See the next page for sample output.)

Here is some sample output for **hw5_part5.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 hw5_part5.py
How many words would you like to turn backwards: 5
Please enter string #1: dog
Please enter string #2: bird
Please enter string #3: horse
Please enter string #4: fish
Please enter string #5: llama
The string 'llama' reversed is 'amall'.
The string 'fish' reversed is 'hsif'.
The string 'horse' reversed is 'esroh'.
The string 'bird' reversed is 'drib'.
The string 'dog' reversed is 'god'.

bash-4.1$ python hw5_part5.py
How many words would you like to turn backwards: 3
Please enter string #1: Kayak
Please enter string #2: Racecar
Please enter string #3: Stats
The string 'Stats' reversed is 'statS'.
The string 'Racecar' reversed is 'racecaR'.
The string 'Kayak' reversed is 'kayaK'.

bash-4.1$ python hw5_part5.py
How many words would you like to turn backwards: 0

bash-4.1$ python hw5_part5.py
How many words would you like to turn backwards: 1
Please enter string #1: step on NO pets
The string 'step on NO pets' reversed is 'step ON no
pets'.
```

(Some of these examples are palindromes, which means they are the same backwards and forwards. If you pay attention to the punctuation, the reversal becomes a bit clearer.)

hw5_part6.py **(Worth 6 points)**

NOTE: You will not be able to complete this part of the homework until we go over how to return from functions in Lecture 11. The slides will be on the website by Monday at noon if you don't want to wait until your class meets.

This final program will allow the user to enter a list of temperatures. The user can continue entering items indefinitely, stopping only when they enter the sentinel value "STOP". (We guarantee that the user will enter at least one temperature.)

Once all of the temperatures have been entered and saved in a list, the program will call the average function, which works as specified in the function header provided below.

```
##########################################################
# average() calculates and returns the average
# Input:    numList; a list of floats
# Output:   avgNum;  a float, average of list's numbers
```

The **`average()`** function does not print out any information – instead, it returns the value of the average to the **`main()`** function that called it. The printing out of the average will need to occur in **`main()`**.

Although this program is calculating average temperature, the **`average()`** function can be used to calculate the average of any list of numbers.

HINT: Think carefully about how to handle the sentinel value and the casting of the numbers as they are received from the user.

(See the next page for sample output.)

Here is some sample output for **hw5_part6.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 hw5_part6.py
Enter a temperature, 'STOP' to exit: 17.2
Enter a temperature, 'STOP' to exit: 55.6
Enter a temperature, 'STOP' to exit: 80.2
Enter a temperature, 'STOP' to exit: 89.3
Enter a temperature, 'STOP' to exit: 100
Enter a temperature, 'STOP' to exit: 0
Enter a temperature, 'STOP' to exit: -5.3
Enter a temperature, 'STOP' to exit: 66.2
Enter a temperature, 'STOP' to exit: 91.5
Enter a temperature, 'STOP' to exit: STOP
The average is 54.96666666666667

bash-4.1$ python hw5_part6.py
Enter a temperature, 'STOP' to exit: 0
Enter a temperature, 'STOP' to exit: 98
Enter a temperature, 'STOP' to exit: 76
Enter a temperature, 'STOP' to exit: -12
Enter a temperature, 'STOP' to exit: 55
Enter a temperature, 'STOP' to exit: 43
Enter a temperature, 'STOP' to exit: 82
Enter a temperature, 'STOP' to exit: 70
Enter a temperature, 'STOP' to exit: 74
Enter a temperature, 'STOP' to exit: -5
Enter a temperature, 'STOP' to exit: 15
Enter a temperature, 'STOP' to exit: STOP
The average is 45.09090909090909
```

Submitting

Once your **hw5_part1.py**, **hw5_part2.py**, **hw5_part3.py**, **hw5_part4.py**, **hw5_part5.py**, and **hw5_part6.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 5 Python files. To double-check you are in the directory with the correct files, you can type **ls**.

```
linux1[3]% ls
hw5_part1.py  hw5_part3.py  hw5_part5.py
hw5_part2.py  hw5_part4.py  hw5_part6.py
linux1[4]%
```

To submit your Homework 5 Python files, we use the **submit** command, where the class is **cs201**, and the assignment is **HW5**. Type in (all on one line) **submit cs201 HW5 hw5_part1.py hw5_part2.py hw5_part3.py hw5_part4.py hw5_part5.py hw5_part6.py** and press enter.

```
linux1[4]% submit cs201 HW5 hw5_part1.py hw5_part2.py
hw5_part3.py hw5_part4.py hw5_part5.py hw5_part6.py
Submitting hw5_part1.py...OK
Submitting hw5_part2.py...OK
Submitting hw5_part3.py...OK
Submitting hw5_part4.py...OK
Submitting hw5_part5.py...OK
Submitting hw5_part6.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.**