CMSC 201 Fall 2017
Homework 6 – Recursion

Assignment: Homework 6 – Recursion
Due Date: ***Tuesday***, November 21st, 2017 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-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.

Objective

Homework 6 is designed to give you lots of practice with recursion, recursion, and recursion. You should think carefully about the base case(s), recursive case(s), and recursive call(s) that each problem will need.

If your solution to a question does not use recursion, you will earn zero points for that question, even if the non-recursive code solves the problem. (Part 5 is not recursive, but all of the other parts are.)

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)**

Create a program that calculates the summation of a number, stopping at a second number (specified by the user) rather than at 1. The program must contain a **`main()`** and a recursive function, the name of which is up to you.

The numbers entered by the user may be positive or negative. However, your program may assume that the second number entered is always lower than the first number entered.

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
Please input the number you want to sum from: 10
Please input the number you want to sum down to: 1
The summation from 10 to 1 is 55

bash-4.1$ python hw6_part1.py
Please input the number you want to sum from: 999
Please input the number you want to sum down to: 56
The summation from 999 to 56 is 497960

bash-4.1$ python hw6_part1.py
Please input the number you want to sum from: 15
Please input the number you want to sum down to: -80
The summation from 15 to -80 is -3120

bash-4.1$ python hw6_part1.py
Please input the number you want to sum from: -1
Please input the number you want to sum down to: -10
The summation from -1 to -10 is -55
```

hw6_part2.py **(Worth 4 points)**

Next, you will create a program that is the recursive version of HW3's part 7: drawing a right triangle, using a height and symbol 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 symbol will be a single character

REMINDER: You can keep the **`print()`** *function from printing on a new line by using* **`end=""`** *at the end:* **`print("Hello", end="")`**. *If you do want to print a new line, you can call print without an argument:* **`print()`**.

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: 8
Please enter the symbol to use: o
o
oo
ooo
oooo
ooooo
oooooo
ooooooo
oooooooo

bash-4.1$ python hw6_part2.py
Please enter the height of the triangle: 1
Please enter the symbol to use: W
W

bash-4.1$ python hw6_part2.py
Please enter the height of the triangle: 0
Please enter the symbol to use: d
```

hw6_part3.py **(Worth 4 points)**

For this part of the homework, you will write a recursive function that counts the number of ears and horns in a line of horses and unicorns. (Horses have 2 ears, while unicorns have 2 ears and a horn, for a total of 3.) The unicorns and horses alternate in the line, with a unicorn in the first position, a horse in the second, etc. The user will enter the length of the line of equines; it is guaranteed to be a positive number (zero or higher).

The program must contain a **`main()`** and a recursive function called **`count()`**. The **`count()`** function will take in an integer (the line length) and, through recursion, return the total number of ears and horns. 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_part3.py
How long is the line of equines? 0
In a line of 0 equines, there are 0 ears and horns.

bash-4.1$ python hw6_part3.py
How long is the line of equines? 10
In a line of 10 equines, there are 25 ears and horns.

bash-4.1$ python hw6_part3.py
How long is the line of equines? 2
In a line of 2 equines, there are 5 ears and horns.

bash-4.1$ python hw6_part3.py
How long is the line of equines? 3
In a line of 3 equines, there are 8 ears and horns.

bash-4.1$ python hw6_part3.py
How long is the line of equines? 88
In a line of 88 equines, there are 220 ears and horns.

bash-4.1$ python hw6_part3.py
How long is the line of equines? 980
In a line of 980 equines, there are 2450 ears and horns.
```

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

Create a program that asks the user for a word, and then checks to see if that word is a palindrome. (Palindromes are words that are the same backwards and forwards – see the sample output for some examples.) When checking for a palindrome, the program should be **case insensitive**. If the word is not a palindrome, the program must print out the reversed word in the message.

The program must contain a **`main()`** and a recursive function called **`reverseStr()`**. The **`reverseStr()`** function will take in a string and, through recursion, return that string in reverse. 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
Please enter a word to check for palindrome-ness: UMBC
Sorry, the word UMBC is NOT a palindrome.
Backwards, it becomes CBMU.

bash-4.1$ python hw6_part1.py
Please enter a word to check for palindrome-ness: racecar
The word racecar IS a palindrome.

bash-4.1$ python hw6_part1.py
Please enter a word to check for palindrome-ness: TacoCat
The word TacoCat IS a palindrome.

bash-4.1$ python hw6_part1.py
Please enter a word to check for palindrome-ness: taco cat
Sorry, the word taco cat is NOT a palindrome.
Backwards, it becomes tac ocat.
```

Part 5 and Part 6 will both tackle the problem of generating the levels of a Pascal's triangle. Part 5 must contain a NON-recursive solution to the problem, while Part 6 must contain a RECURSIVE solution. You may tackle them in either order, but we recommend Part 5 first.

hw6_part5.py **(Worth 8 points)**

This part of the homework assignment is NOT RECURSIVE.

The last program will generate the levels of Pascal's triangle. You should read the Wikipedia page for information about Pascal's triangle, paying special attention to how the triangle is constructed, as seen here.

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

```
############################################################
# pascal() creates each level of Pascal's
#          triangle, reaching the requested height
# Input:   levelsToMake; an int, the number of levels requested
# Output:  None (the levels are printed from the function)
```

Your code must perform basic **input validation** to ensure that numbers are greater than or equal to 1, and tell the user what it will accept as valid input.

Note that the code should follow the conventional numbering of a Pascal's triangle: the first row (the single 1 at the top) is **row zero**, meaning that a request for a triangle of height 5 will result in six rows being printed out.

It is also **highly recommended** that you create a Pascal's triangle by hand (to a height of at least 8), paying attention to the steps taken as you go, prior to beginning your code.

(You do not need to worry about printing out a "centered" triangle – the numbers being left-aligned (as seen in the sample output) is acceptable.)

Here is the sample output for **hw6_part5.py** and **hw6_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 hw6_part5.py
Welcome to the Pascal's triangle generator.
Please enter the number of levels to generate: -1
Your number must be positive (greater than zero).
Please enter the number of levels to generate: 0
Your number must be positive (greater than zero).
Please enter the number of levels to generate: 1
1
1 1

bash-4.1$ python hw6_part6.py
Welcome to the Pascal's triangle generator.
Please enter the number of levels to generate: 6
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1

bash-4.1$ python hw6_part5.py
Welcome to the Pascal's triangle generator.
Please enter the number of levels to generate: 13
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
```

hw6_part6.py **(Worth 10 points)**

The last program will use recursion to generate the levels of Pascal's triangle. You should read the Wikipedia page for information about Pascal's triangle, paying special attention to how the triangle is constructed, as seen here.

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

```
############################################################
# pascal() uses recursion to create each level of Pascal's
#          triangle, reaching the requested height
# Input:   currLevel;    an int, the current level being created
#          levelsToMake; an int, the number of levels requested
#          levelList;    a 2D list of ints, containing the levels
#                                           as they are created
# Output:  None (levelList is changed in place, and the updated
#                levelList will be the same in main() )
```

Your code must perform basic **input validation** to ensure that numbers are greater than or equal to 1, and tell the user what it will accept as valid input.

Think carefully about what your base cases should be! Also, note that the code should follow the conventional numbering of a Pascal's triangle: the first row (the single 1 at the top) is **row zero**, meaning that a request for a triangle of height 5 will result in six rows being generated in total.

It is also **highly recommended** that you create a Pascal's triangle by hand (to a height of at least 8), paying attention to the steps taken as you go, prior to beginning your code.

(You do not need to worry about printing out a "centered" triangle – the numbers being left-aligned (as seen in the sample output) is acceptable.)

(See the *previous* page for sample output.)

Submitting

Once your **hw6_part1.py**, **hw6_part2.py**, **hw6_part3.py**, **hw6_part4.py**, **hw6_part5.py**, and **hw6_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 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  hw6_part6.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 hw6_part6.py** and press enter.

```
linux1[4]% submit cs201 HW6 hw6_part1.py hw6_part2.py
hw6_part3.py hw6_part4.py hw6_part5.py hw6_part6.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
Submitting hw6_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.**