# CMSC 201 Fall 2018
## Homework 6 – Recursion

**Assignment:** Homework 6 – Recursion
**Due Date:** Friday, November 16 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-fa18.

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.

## 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.

## Questions

Each question is worth the indicated number of points. Following the coding standards is worth 6 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 8 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 `reverse()`. The `reverse()` 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.)

```
linux2[5]% python3 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.

linux2[6]% python3 hw6_part1.py
Please enter a word to check for palindrome-ness: racecar
The word racecar IS a palindrome.

linux2[7]% python3 hw6_part1.py
Please enter a word to check for palindrome-ness: TacoCat
The word TacoCat IS a palindrome.

linux2[8]% python3 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.
```

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

Write a program that uses a recursive function to find the greatest common divisor (GCD) of two integers. (The GCD is the largest number that divides evenly into both of the integers – see the sample output for some examples.)

The program must contain a **`main()`** and a recursive function called **`gcd()`**. The **`gcd()`** function should take in three arguments: the first number, the second number, and an integer to help the function "keep track" of its current attempt at finding a GCD value. The program may also contain any other functions you deem necessary.

The program should prompt the user for two integers to find the GCD for, but it cannot assume that the provided number will be a valid value! It must perform basic **input validation** to ensure that numbers are greater than or equal to 1, and should tell the user what it will accept as valid input.

*(HINT: Before you start coding, come up with an algorithm to find the GCD!)*

Here is some sample output for **hw6_part2.py**, with the user input in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
linux2[5]% python3 hw6_part2.py
Please enter the first integer: 775
Please enter the second integer: 90
The GCD for 775 and 90 is 5

linux2[6]% python3 hw6_part2.py
Please enter the first integer: 6
Please enter the second integer: 6
The GCD for 6 and 6 is 6

linux2[7]% python3 hw6_part2.py
Please enter the first integer: 99
Please enter the second integer: 9
The GCD for 99 and 9 is 9

linux2[8]% python3 hw6_part2.py
Please enter the first integer: 37
Please enter the second integer: 562
The GCD for 37 and 562 is 1
```

**hw6_part3.py** **(Worth 16 points)**

*(WARNING: This part of the homework is the most challenging, so budget plenty of time and brain power.)*

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.

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 next page for sample output.)

Here is the sample output for **hw6_part3.py**, with the user input in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
linux2[5]% python3 hw6_part3.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

linux2[6]% python3 hw6_part3.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

linux2[7]% python3 hw6_part3.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
```

## Submitting

Once your **`hw6_part1.py`**, **`hw6_part2.py`**, and **`hw6_part3.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_part2.py  hw6_part3.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`** 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 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.**