CMSC 201 Fall 2017
Homework 3 – While Loops

Assignment: Homework 3 – While Loops
Due Date: Friday, September 29th, 2017 by 8:59:59 PM
Value: 40 points

Collaboration: For Homework 3, 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.

You should already be familiar with variables, expressions, **`input()`**, and **`print()`**. You should also be familiar with one-way, two-way, and multi-way decision structures.

This assignment will focus on implementing algorithms using **`while`** loops, including any Boolean logic needed.

At the end, your Homework 3 files must run without any errors.

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

Additional Instructions – Creating the hw3 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 Homework 1 and Homework 2, you should create a directory to store your Homework 3 files. We recommend calling it **`hw3`**, 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 3 files in the same **`hw3`** 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:

- Naming Conventions
- Use of Whitespace
- Comments (specifically, File Header Comments)
    - ***For Homework 3, you should start using In-Line Comments where appropriate***
- Line Length

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 do need to worry about "input validation" on a number of the problems. Many of the parts of this assignment center on validating input from the user. 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.

For example, if your program asks the user to enter a whole number, it is acceptable if your program crashes if they enter something else like "dog" or "twenty" or "88.2" instead.

Here is what that might look like:

```
Please enter a number: twenty
Traceback (most recent call last):
  File "test_file.py", line 10, in <module>
    num = int(input("Please enter a number: "))
ValueError: invalid literal for int() with base 10: 'twenty'
```

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.

hw3_part1.py **(Worth 6 points)**

This program simulates the up and down movement of a hailstone in a storm.

The program should ask the user for an integer, which will be the starting height of the hailstone. Based on the current value of the height, the program will repeatedly do the following:

- If the current height is 1 (or 0), quit the program
- If the current height is even, cut it in half (divide by 2)
- If the current height is odd, multiply it by 3, then add 1

The program will keep updating the number, following the above rules, until the number is 1. It should print out the height of the hailstone at each step, including at the end. Once the hailstone is at height 1 (or 0), the program should end, and print out that the hailstone stopped.

(HINT: Think carefully about the order in which the program checks each of the conditions, or it won't perform correctly.)

For example, given a starting value of 24, here are the numbers to output:

```
24 -> 12 -> 6 -> 3 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
```

For this part of the homework, you can assume the following:

- The number will be positive (zero or greater than zero)

(See the next page for sample output.)

Here is some sample output for **hw3_part1.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 hw3_part1.py
Please enter the starting height of the hailstone: 36
Hail is currently at height 36
Hail is currently at height 18
Hail is currently at height 9
Hail is currently at height 28
Hail is currently at height 14
Hail is currently at height 7
Hail is currently at height 22
Hail is currently at height 11
Hail is currently at height 34
Hail is currently at height 17
Hail is currently at height 52
Hail is currently at height 26
Hail is currently at height 13
Hail is currently at height 40
Hail is currently at height 20
Hail is currently at height 10
Hail is currently at height 5
Hail is currently at height 16
Hail is currently at height 8
Hail is currently at height 4
Hail is currently at height 2
Hail stopped at height 1

bash-4.1$ python hw3_part1.py
Please enter the starting height of the hailstone: 0
Hail stopped at height 0

bash-4.1$ python hw3_part1.py
Please enter the starting height of the hailstone: 16
Hail is currently at height 16
Hail is currently at height 8
Hail is currently at height 4
Hail is currently at height 2
Hail stopped at height 1
```

(HINT: If you want to prevent the program from outputting decimal numbers like 6.0 and 3.0, you will need to use integer division and/or casting.)

hw3_part2.py **(Worth 4 points)**

Write a program that is able to calculate the answer to an exponentiation problem **without** using exponentiation.

The program should ask the user for two numbers, and should compute the answer to **`firstNum ** secondNum`**. The program should then output the full equation, including the answer, to the user.

For these inputs, you can assume the following:

- The first number may be any integer, including negative values
- The second number will be positive (zero or greater than zero)

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 hw3_part2.py
Please enter the first  number: 0
Please enter the second number: 7
0 ** 7 = 0

bash-4.1$ python hw3_part2.py
Please enter the first  number: 5
Please enter the second number: 0
5 ** 0 = 1

bash-4.1$ python hw3_part2.py
Please enter the first  number: 201
Please enter the second number: 37
201 ** 37 = 16529287156691474278000699829162234449038360515657327520483688508302167368774186647401

bash-4.1$ python hw3_part2.py
Please enter the first  number: -7
Please enter the second number: 3
-7 ** 3 = -343

bash-4.1$ python hw3_part2.py
Please enter the first  number: -7
Please enter the second number: 6
-7 ** 6 = 117649
```

hw3_part3.py **(Worth 4 points)**

Create a program that will have the user enter a time, putting in the hours, minutes, and time period (AM or PM) individually, and in that order.

The user must enter valid information for each input, and the program must reprompt the user as many times as needed until they enter valid input for each question. Once they enter valid values for all three questions, the program should display the time as entered by the user.

If the user enters an invalid input, the program must tell the user why it is invalid: for hours and minutes, that whether it is too high or low; for time period, that it must be 'AM' or 'PM' to be correct

The input must be validated to these specifications:

- Hours must be between 1 and 12, inclusive.
- Minutes must be between 0 and 59, inclusive.
- Time period must be 'AM' or 'PM', in all caps.

NOTE: It is *perfectly acceptable* for the program to display `3 : 7 PM` when it is 3:07 PM. (And similar displays whenever minutes is less than 10.) You do not have to worry about fixing this in your program.

(See the next page for sample output.)

Here is some sample output for **hw3_part3.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 hw3_part3.py
Please enter the time: hours, then minutes, then AM/PM
Please enter the hour: 7
Please enter the minute: 14
Please enter the time period: AM
It is 7 : 14 AM

bash-4.1$ python hw3_part3.py
Please enter the time: hours, then minutes, then AM/PM
Please enter the hour: 1
Please enter the minute: 7
Please enter the time period: PM
It is 1 : 7 PM

bash-4.1$ python hw3_part3.py
Please enter the time: hours, then minutes, then AM/PM
Please enter the hour: 0
The hours value is too low.
Please enter the hour: 13
The hours value is too high.
Please enter the hour: 8
Please enter the minute: -1
The minutes value is too low.
Please enter the minute: 60
The minutes value is too high.
Please enter the minute: 59
Please enter the time period: dogs
Time period must be 'AM' or 'PM'.
Please enter the time period: am
Time period must be 'AM' or 'PM'.
Please enter the time period: PM
It is 8 : 59 PM
```

hw3_part4.py **(Worth 4 points)**

Write a program that is able to calculate how much money will be raised during a charity run, by asking the user how many pledges they secured, how much money each of the pledges was for, and how many miles they covered. Donations work on a per mile basis: someone pledges to pay a specific amount for each mile covered. The more miles, the more money in donations.

Your need to create a program that does the following, in this exact order:

1. Get the number of donations
2. Get the value of each donation
3. Get the number of miles
4. Calculate and display the total amount donated to charity

As the program asks the user for the value of each pledge, it must print out the number of the pledge, so they don't lose track of which one they are on (see the sample output).

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 hw3_part4.py
How many pledges did you secure? 5
For pledge # 1
How much money was pledged? 10
For pledge # 2
How much money was pledged? .25
For pledge # 3
How much money was pledged? 9.99
For pledge # 4
How much money was pledged? 8
For pledge # 5
How much money was pledged? 20
How many miles did you run for charity? 201
Based on your 201 miles you earned $ 9696.24 for the
charity.
```

hw3_part5.py **(Worth 5 points)**

Write a program that prints the numbers from 1 up to 77 (inclusive), one per line. However, there are three special cases where instead of printing the number, you print a message instead:

1. If the number you would print is **divisible by 4**, print the message:
   Four leaf clovers are lucky!
2. If the number you would print is **divisible by 5**, print the message:
   Where do you see yourself in five years?
3. If the number you would print is **divisible by 4 and 5**, instead print out:
   The year 2020 is coming soon!

Print the exact strings given! Failing to do so will lose you points.

Here is some partial sample output, showing from 1 to 20, and from 73 to 77.

```
bash-4.1$ python hw3_part5.py
1
2
3
Four leaf clovers are lucky!
Where do you see yourself in five years?
6
7
Four leaf clovers are lucky!
9
Where do you see yourself in five years?
11
Four leaf clovers are lucky!
13
14
Where do you see yourself in five years?
Four leaf clovers are lucky!
17
18
19
The year 2020 is coming soon!
                        [...]
73
74
Where do you see yourself in five years?
Four leaf clovers are lucky!
77
```

hw3_part6.py **(Worth 7 points)**

Create a program that will output a "counting" box.

The program should prompt the user for these inputs, **in exactly this order:**

1. The width of the box
2. The height of the box

For these inputs, you can assume the following:

- The height and width will be integers greater than zero

Using this width and height, the program will print out a box where there are **`width`** numbers on each line and **`height`** rows. The numbers must count up starting from 1, and should continue counting up (do not restart the numbering).

HINT: 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()`**.
You can put anything you want inside the quotation marks – above, we have used a single space, to separate the numbers in the counting box. You can also use an empty string, a comma, or even whole words!

(See the next page for sample output.)

Here is some sample output for **hw3_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 hw3_part6.py
Please enter a width:  4
Please enter a height: 2
1 2 3 4
5 6 7 8

bash-4.1$ python hw3_part6.py
Please enter a width:  12
Please enter a height: 7
1 2 3 4 5 6 7 8 9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84

bash-4.1$ python hw3_part6.py
Please enter a width:  11
Please enter a height: 13
1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63 64 65 66
67 68 69 70 71 72 73 74 75 76 77
78 79 80 81 82 83 84 85 86 87 88
89 90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109 110
111 112 113 114 115 116 117 118 119 120 121
122 123 124 125 126 127 128 129 130 131 132
133 134 135 136 137 138 139 140 141 142 143
```

(NOTE: The "box" might not actually be a box. The number of digits increases as the value gets larger, and so the box gets wider.)

hw3_part7.py **(Worth 6 points)**

Write a program that draws a right triangle (a triangle with equal width and height). Your program should prompt the user for the height of the triangle, and the symbol it should be made of.

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

You should then use the symbol they chose to print a right triangle of the height they indicated.

HINT: 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.

```
bash-4.1$ python hw3_part7.py
Please enter the height of your triangle: 3
Please enter a single character for the symbol: $
$
$$
$$$

bash-4.1$ python hw3_part7.py
Please enter the height of your triangle: 0
Please enter a single character for the symbol: g

bash-4.1$ python hw3_part7.py
Please enter the height of your triangle: 7
Please enter a single character for the symbol: W
W
WW
WWW
WWWW
WWWWW
WWWWWW
WWWWWWW
```

Submitting

Once your **hw3_part1.py**, **hw3_part2.py**, **hw3_part3.py**, **hw3_part4.py**, **hw3_part5.py**, **hw3_part6.py**, and **hw3_part7.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 3 Python files. To double-check you are in the directory with the correct files, you can type **ls**.

```
linux1[3]% ls
hw3_part1.py  hw3_part3.py  hw3_part5.py  hw3_part7.py
hw3_part2.py  hw3_part4.py  hw3_part6.py
linux1[4]%
```

To submit your Homework 3 Python files, we use the **submit** command, where the class is **cs201**, and the assignment is **HW3**. Type in (all on one line) **submit cs201 HW3 hw3_part1.py hw3_part2.py hw3_part3.py hw3_part4.py hw3_part5.py hw3_part6.py hw3_part7.py** and press enter.

```
linux1[4]% submit cs201 HW3 hw3_part1.py hw3_part2.py
hw3_part3.py hw3_part4.py hw3_part5.py hw3_part6.py
hw3_part7.py
Submitting hw3_part1.py...OK
Submitting hw3_part2.py...OK
Submitting hw3_part3.py...OK
Submitting hw3_part4.py...OK
Submitting hw3_part5.py...OK
Submitting hw3_part6.py...OK
Submitting hw3_part7.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.**