CMSC 201 Spring 2018
Homework 3 – While Loops

Assignment: Homework 3 – While Loops
Due Date: Friday, March 2nd, 2018 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-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
```

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
- Constants
- 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
Hailstone is currently at height 36
Hailstone is currently at height 18
Hailstone is currently at height 9
Hailstone is currently at height 28
Hailstone is currently at height 14
Hailstone is currently at height 7
Hailstone is currently at height 22
Hailstone is currently at height 11
Hailstone is currently at height 34
Hailstone is currently at height 17
Hailstone is currently at height 52
Hailstone is currently at height 26
Hailstone is currently at height 13
Hailstone is currently at height 40
Hailstone is currently at height 20
Hailstone is currently at height 10
Hailstone is currently at height 5
Hailstone is currently at height 16
Hailstone is currently at height 8
Hailstone is currently at height 4
Hailstone is currently at height 2
Hailstone stopped at height 1

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

bash-4.1$ python hw3_part1.py
Please enter the starting height of the hailstone: 16
Hailstone is currently at height 16
Hailstone is currently at height 8
Hailstone is currently at height 4
Hailstone is currently at height 2
Hailstone 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 a modulus problem **without** using the division, integer division, mod, or multiplication operators.

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 positive integer, or zero
- The second number may be any positive integer (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: 15
0 % 15 = 0

bash-4.1$ python hw3_part2.py
Please enter the first  number: 15
Please enter the second number: 7
15 % 7 = 1

bash-4.1$ python hw3_part2.py
Please enter the first  number: 9
Please enter the second number: 7359
9 % 7359 = 9

bash-4.1$ python hw3_part2.py
Please enter the first  number: 201
Please enter the second number: 42
201 % 42 = 33

bash-4.1$ python hw3_part2.py
Please enter the first  number: 2329
Please enter the second number: 17
2329 % 17 = 0
```

hw3_part3.py **(Worth 9 points)**

Create a currency exchange program that will have the user enter a monetary amount, putting in the dollars, cents, and currency type (CAD, USD, or AUD) 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 amount and currency as entered by the user.

If the user enters an invalid input, the program must tell the user why it is invalid: for dollars, whether it is too high or low; for cents; whether it is too high or low, and that it must be divisible by 10; for currency type, that it must be 'CAD', 'USD', or 'AUD' to be accepted.

The input must be validated to these specifications:

- Dollars must be between 0 and 1000, inclusive.
- Cents must be
  - Between 0 and 99, inclusive, and
  - Divisible by 10 (so 0, 10, 20, 30, etc… are all valid)
- Currency type must be 'CAD', 'USD', or 'AUD' in all caps.

HINT: You should be using **constants** for at least some of these integer and string values!

NOTE: It is *perfectly acceptable* for the program to display `16.0` when the user is exchanging $16.00. 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 amount: dollars, then cents, then
currency type.
Please enter the dollar amount: 20
Please enter the cent amount: 50
Select a currency type (CAD, USD, AUD)
What currency type are you exchanging? USD
You are exchanging 20.50 in USD

bash-4.1$ python hw3_part3.py
Please enter the amount: dollars, then cents, then
currency type.
Please enter the dollar amount: -1
You cannot exchange negative amounts.
Please enter the dollar amount: 1001
The exchange does not carry more than 1000.
Please enter the dollar amount: 750
Please enter the cent amount: -1
Cents must be multiples of 10.
Cents must be positive.
Please enter the cent amount: 100
Cents must be less than 100.
Please enter the cent amount: 27
Cents must be multiples of 10.
Please enter the cent amount: 444
Cents must be multiples of 10.
Cents must be less than 100.
Please enter the cent amount: 30
Select a currency type (CAD, USD, AUD)
What currency type are you exchanging? EUR
That is not a valid currency, choose from CAD, USD, AUD
What currency type are you exchanging? JPY
That is not a valid currency, choose from CAD, USD, AUD
What currency type are you exchanging? cad
That is not a valid currency, choose from CAD, USD, AUD
What currency type are you exchanging? Aud
That is not a valid currency, choose from CAD, USD, AUD
What currency type are you exchanging? AUD
You are exchanging 750.30 in AUD
```

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

Write a program that is able to calculate information about a road trip, by asking the user how many days they travelled and how many miles they travelled for each of those days. The information calculated must be the total number of miles travelled, as well as the average per day.

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

1. Get the number of days travelled
2. Get the number of miles travelled each day
3. Calculate and display the total miles driven
4. Calculate and display the average miles driven per day

As the program asks the user for the miles driven each day, it must print out the number of the day, 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 long was the road trip in days? 5
For day # 1
How many miles were driven? 380
For day # 2
How many miles were driven? 200.6
For day # 3
How many miles were driven? 0
For day # 4
How many miles were driven? 86.5
For day # 5
How many miles were driven? 906
You drove a total of 1573.1 miles.
You drove an average of 314.62 miles per day.
```

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

Write a program that prints the numbers from 1 up to 110 (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 5**, print the message:
   Touch is one of the five senses.
2. If the number you would print is **divisible by 7**, print the message:
   Four score and seven years ago.
3. If the number you would print is **divisible by 5 and 7**, instead print out:
   We are closer to 2035 than we are to 2000!

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

Here is a *partial* sample output, showing from 1 to 16, and from 104 to 110.

```
bash-4.1$ python hw3_part5.py
1
2
3
4
Touch is one of the five senses.
6
Four score and seven years ago.
8
9
Touch is one of the five senses.
11
12
13
Four score and seven years ago.
Touch is one of the five senses.
16
                        [...]
104
We are closer to 2035 than we are to 2000!
106
107
108
109
Touch is one of the five senses.
```

hw3_part6.py **(Worth 8 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 0, 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
0 1 2 3
4 5 6 7

bash-4.1$ python hw3_part6.py
Please enter a width:  12
Please enter a height: 7
0 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

bash-4.1$ python hw3_part6.py
Please enter a width:  11
Please enter a height: 13
0 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
```

(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.)

Submitting

Once your **hw3_part1.py**, **hw3_part2.py**, **hw3_part3.py**, **hw3_part4.py**, **hw3_part5.py**, and **hw3_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 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_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_part6py** 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
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
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.**