UMBC
AN HONORS UNIVERSITY IN MARYLAND

CMSC 201 Fall 2018
Homework 3 – While Loops

Assignment: Homework 3 – While Loops
Due Date: Friday, September 28th, 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 https://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.

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 lab, 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 around 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.)

```
linux2[2]% python3 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

linux2[3]% python3 python hw3_part1.py
Please enter the starting height of the hailstone: 0
Hailstone stopped at height 0

linux2[4]% python3 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 an integer division problem **without** using the division, integer division, mod, or multiplication operators.

The program should ask the user for two integers, 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.)

```
linux2[12]% python3 hw3_part2.py
Please enter the first  number: 0
Please enter the second number: 15
0 // 15 = 0

linux2[13]% python3 hw3_part2.py
Please enter the first  number: 15
Please enter the second number: 7
15 // 7 = 2

linux2[14]% python3 hw3_part2.py
Please enter the first  number: 7359
Please enter the second number: 9
7359 // 9 = 817

linux2[15]% python3 hw3_part2.py
Please enter the first  number: 201
Please enter the second number: 42
201 % 42 = 6

linux2[16]% python3 hw3_part2.py
Please enter the first  number: 2329
Please enter the second number: 17
2329 % 17 = 137
```

hw3_part3.py (Worth 9 points)

A popular internet-based DVD rental company is buying up everyone's used DVDs. They want you to create a program that makes sure the DVDs they buy are decent and sold to them at a good price.

Your program should first ask how many times a movie has been watched, then it should ask how much money they want to sell this movie for, and finally it should ask what genre of movie it is.

The user must enter valid information for each input, and the program must re-prompt 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 times watched, price, and genre as entered by the user.

You can assume that the ***type*** of input will be correct for each question asked, but you must validate the value entered.

If the user enters an invalid input, the program must tell the user why it is invalid: for watches, whether it is too high or low; for price, whether it is too high or low or not divisible of 0.25; for genre, that it must be 'romance', or 'comedy' to be accepted.

The input must be validated to these specifications:

- Times watched must be between 0 and 10, inclusive.
- Price must be:
  - Between 0 and 15, inclusive, and
  - Divisible by 0.25 (so 1.25, 2.50, 5.00, 10.75, etc. are all accepted)
- Genre must be 'romance', 'comedy', in all lowercase.

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

HINT: You can use floating point numbers with the modulus operator! Test this out in the interpreter before you use it.

(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.)

```
linux3[151]% python3 hw3_part3.py
Please enter the amount of times watched: 4
Please enter the price you want to sell for: 9.25
Enter the genre of movie (romance, comedy): comedy
You are selling a comedy movie that has been watched 4
times for 9.25 dollars.

linux3[152]% python3 hw3_part3.py
Please enter the amount of times watched: -10
You can't watch a movie a negative amount of times.
Please enter the amount of times watched: -1
You can't watch a movie a negative amount of times.
Please enter the amount of times watched: 20
We don't accept movies watched 20 times
Please enter the amount of times watched: 15
We don't accept movies watched 15 times
Please enter the amount of times watched: 10
Please enter the price you want to sell for: -10
We can't accept a negative amount.
Please enter the price you want to sell for: -10.43
We can't accept a negative amount.
We only accept prices that can be paid in quarters.
Please enter the price you want to sell for: 9.12
We only accept prices that can be paid in quarters.
Please enter the price you want to sell for: 7.75
Enter the genre of movie (romance, comedy): action
action is not a valid type, choose from romance or comedy
Enter the genre of movie (romance, comedy): adventure
adventure is not a valid type, choose from romance or
comedy
Enter the genre of movie (romance, comedy): romance
You are selling a romance movie that has been watched 10
times for 7.75 dollars.

linux3[153]%
```

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

Write a program that is able to act like your personal pedometer, by asking the user how many steps they travelled in one week. The information calculated must be the minimum and maximum steps travelled (and the days that you walked those steps!). *Make sure you use a loop to solve this problem.*

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

1. Get the number of steps travelled each day
2. Calculate and display the minimum & maximum steps walked.
3. Calculate and display the day those steps were taken.

As the program asks the user for the steps walked 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).

You do not have to validate user input for this problem. You can assume that all numbers entered with be bigger than or equal to 0.

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[123]% python3 hw3_part4.py
For day # 1
steps today: 345
For day # 2
steps today: 932
For day # 3
steps today: 1003
For day # 4
steps today: 123
For day # 5
steps today: 1249
For day # 6
steps today: 985
For day # 7
steps today: 0
The min day was day 7 when you walked 0 steps.
The max day was day 5 when you walked 1249 steps.
```

hw3_part5.py (Worth 5 points)

Write a program that prints the numbers from 1 up to 105 (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 3**, print the message:
   The dog of wisdom knows all about this number.
2. If the number you would print is **divisible by 7**, print the message:
   Hold on I have a meme for this.
3. If the number you would print is **divisible by 3 and 7**, instead print out:
   Some infinities are bigger than other infinities!

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

Here is a *partial* sample output, showing from 1 to 13, and from 97 to 105.

```
linux2[169]% python hw3_part5.py
1
2
The dog of wisdom knows all about this number.
4
5
The dog of wisdom knows all about this number.
Hold on I have a meme for this.
8
The dog of wisdom knows all about this number.
10
11
The dog of wisdom knows all about this number.
13                       [...]
97
Hold on I have a meme for this.
The dog of wisdom knows all about this number.
100
101
The dog of wisdom knows all about this number.
103
104
Some infinities are bigger than other infinities!
```

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 down starting from the area of the box **`(width * height)`**, and should continue counting down (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.)

```
linux3[113]% python3 hw3_part6.py
enter a width: 4
enter a height: 2
8 7 6 5
4 3 2 1

linux3[114]% python3 hw3_part6.py
enter a width: 12
enter a height: 7
84 83 82 81 80 79 78 77 76 75 74 73
72 71 70 69 68 67 66 65 64 63 62 61
60 59 58 57 56 55 54 53 52 51 50 49
48 47 46 45 44 43 42 41 40 39 38 37
36 35 34 33 32 31 30 29 28 27 26 25
24 23 22 21 20 19 18 17 16 15 14 13
12 11 10 9 8 7 6 5 4 3 2 1

linux3[115]% python3 hw3_part6.py
enter a width: 11
enter a height: 13
143 142 141 140 139 138 137 136 135 134 133
132 131 130 129 128 127 126 125 124 123 122
121 120 119 118 117 116 115 114 113 112 111
110 109 108 107 106 105 104 103 102 101 100
99 98 97 96 95 94 93 92 91 90 89
88 87 86 85 84 83 82 81 80 79 78
77 76 75 74 73 72 71 70 69 68 67
66 65 64 63 62 61 60 59 58 57 56
55 54 53 52 51 50 49 48 47 46 45
44 43 42 41 40 39 38 37 36 35 34
33 32 31 30 29 28 27 26 25 24 23
22 21 20 19 18 17 16 15 14 13 12
11 10 9 8 7 6 5 4 3 2 1

linux3[116]%
```

(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 correct 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.**