UMBC
AN HONORS UNIVERSITY IN MARYLAND

CMSC 201 Spring 2019
Homework 3 – While Loops

Assignment: Homework 3 – While Loops
Due Date: Friday, March 1st, 2019 by 11:59:59 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/spring19-201-collab

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.)

```
linux[0]$ 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

linux[0]$ python3 hw3_part1.py
Please enter the starting height of the hailstone: 0
Hailstone stopped at height 0

linux[0]$ python3 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 multiplication 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 or 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.)

```
linux[0]$ python3 hw3_part2.py
Please enter the first  number: 0
Please enter the second number: 15
0 * 15 = 0

linux[0]$ python3 hw3_part2.py
Please enter the first  number: 15
Please enter the second number: 7
15 * 7 = 105

linux[0]$ python3 hw3_part2.py
Please enter the first  number: 9
Please enter the second number: 7359
9 * 7359 = 66231

linux[0]$ python3 hw3_part2.py
Please enter the first  number: 201
Please enter the second number: 1
201 * 1 = 201
```

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

You are a space explorer.

On an alien planet, you are storing information about plants, animals and minerals you encounter. Your software is going to ask you for whether the observed item is “plant,” “animal,” or “mineral”. You will enter a positive float for its weight in grams, and a threat level which is a float between 0 and 1, inclusively.

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 information entered by the user.

If the user enters an invalid input, the program must tell the user why it is invalid: for weight, that it must be a positive float; for threat level, that it is too high or too low; for fauna type, that it must be ‘plant’, ‘animal’, or ‘mineral’ to be accepted.

The input must be validated to these specifications:

- Type must be ‘plant’, ‘animal’, or ‘mineral’.
- Weight must be a positive float
- Threat level must be between 0 and 1, inclusively.

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

(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.)

```
linux[0]$ python3 hw3_part3.py
Hello explorer!  Please enter the type of object you are
observing: dog
Invalid type.  Must be "plant", "animal", or "mineral":
animal
Excellent.  Please enter the weight in grams: 10000
Thank you.  Please enter the threat level of the object:
0.1
Data on animal weighing 10000.0 grams with a threat level
of 0.1 has been recorded.  Explore responsibly.


linux[0]$ python3 hw3_part3.py
Hello explorer!  Please enter the type of object you are
observing: PLANT
Invalid type.  Must be "plant", "animal", or "mineral":
Plant
Invalid type.  Must be "plant", "animal", or "mineral":
plant
Excellent.  Please enter the weight in grams: 0
Invalid weight.  Weight must be greater than zero: -15
Invalid weight.  Weight must be greater than zero: 50000
Thank you.  Please enter the threat level of the object:
-1000
INVALID INPUT!  Threat too low!  Threat level must be
between 0 and 1, inclusively: 9000
INVALID INPUT!  Threat too high!  Threat level must be
between 0 and 1, inclusively: 0
Data on plant weighing 50000.0 grams with a threat level
of 0.0 has been recorded.  Explore responsibly.
```

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

You are a pokemon trainer. (No, I'm not putting the accent on the e, which means that they are totally novel and not trademarked creatures.)

Write a program that is able to calculate information on the pokemon a trainer has caught. You will ask the user how many pokemon they caught and then the battle rating (BR) of each. Finally, you will print the average BR of the pokemon caught.

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

1. Get the number of pokemon caught
2. Get the battle rating of each pokemon caught
3. Calculate and display the average pokemon BR.

As the program asks the user for the pokemon, it must print out which number the user is on. **BR can be any integer! You don't need to check for validity!**

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.)

```
linux[0]$ python3 hw3_part4.py
Professor Johnson here!  How many pokemon did you catch
today, young trainer? 0
Hohoho!  Surely you caught more than that!  How many was
it really? 3
Pokemon # 1
What is the BR of this pokemon? 521
Pokemon # 2
What is the BR of this pokemon? 201
Pokemon # 3
What is the BR of this pokemon? 9000
Wicked!  The 3 pokemon you caught had an average battle
rating of:  3240.6666666666665
```

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 3**, print the message: *Octopodes have three hearts.*
2. If the number you would print is **divisible by 5**, print the message: *Why were there only five golden rings?*
3. If the number you would print is **divisible by 3 and 5**, instead print out: *Threeve. A combination of three and five. Simply stunning.*

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.

```
linux[0]$ python3 hw3_part5.py
1
2
Octopodes have three hearts.
4
Why were there only five golden rings?
Octopodes have three hearts.
7
8
Octopodes have three hearts.
Why were there only five golden rings?
11
Octopodes have three hearts.
13
14
Threeve.  A combination of three and five.  Simply
stunning.
16
17
   [... oodles of numbers ... (don't print this)]
103
104
Threeve.  A combination of three and five.  Simply
stunning.
106
```

```
107
Octopodes have three hearts.
109
Why were there only five golden rings?
```

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 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.)

```
linux[0]$ python3 hw3_part6.py
Please enter a width:  4
Please enter a height: 2
1 2 3 4
5 6 7 8

linux[0]$ python3 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

linux[0]$ python3 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.)

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.**