CMSC 201, Section 40
Spring 2020
Homework 4 – While Loops

Assignment: Homework 4 – While Loops
Due Date: Monday, March 9th before midnight
Value: 40 points

Collaboration: For Homework 4, 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 in Section 60.

If you collaborate with another student, you must both email your instructor and make a note in your code. When emailing your instructor, include the other student's name and UMBC email address, the date that the collaboration took place, and a brief description of what the collaboration was about. Place this same information in the header comment of your code (see below). Even if someone helped you, but you didn't get a chance to help them (or vice versa), you both still need to follow this procedure. You must email your instructor within 24 hours of the collaboration occurring (we recommend emailing them immediately, so you don't forget any details).

Make sure that you have a **complete file header comment at the top of <u>each</u> file**, and that all the information is correctly filled in.

```
"""
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
Collaboration:
   DESCRIPTION OF HOW YOU COLLABORATED
   (Include this section only if you collaborated.)
"""
```

Instructions

For each of the questions below, you are given a problem that you must solve or a task you must complete. This homework will deal with the concepts of **`while`** loops, which were covered last week and in the first lecture this week.

At the end, your Homework 4 files must run without any errors.

Additional Instructions – Creating the hw4 Directory

Just as you did for your previous homeworks, you should create a directory to store your Homework 4 files. We recommend calling it **`hw4`** and creating it inside the **`Homeworks`** directory inside your **`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 Homework 4 files in the same **`hw4`** folder.)

Coding Standards

Prior to this assignment, you should re-read the Coding Standards, available 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
- In-line Comments
- Line Length

Additional Specifications

You must use `if __name__ == '__main__':` as discussed in class.

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 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 header comments and correctly named files, you will lose points

hw4_part1.py - Guessing Game **(Worth 8 points)**

Write a program that allows the user to play a guessing game for a number between 1 and 10, inclusive. The user is given 5 chances to guess the correct number. The loop exits if the user guesses the correct number or if they run out of chances.

If the user guesses a number less than 1 or greater than 10, they should be re-prompted to enter a new number. This does not count as one of their guesses.

Create a constant at the start of your program named CORRECT_GUESS and set it equal to the value 8**.** You should also have constants for the minimum number, maximum number, and maximum number of guesses.

Make sure to use your constants in your user prompts (i.e. do not use literal integers for the minimum and maximum values).

(Hint: A Boolean flag-controlled loop works well for this problem!)

See the next page for sample output. The user input is in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
linux6[268]% python3 hw4_part1.py
Guess a number between 1 and 10, inclusive: 8
Correct!
linux6[269]% python3 hw4_part1.py
Guess a number between 1 and 10, inclusive: 3
Incorrect. Try again.
Guess a number between 1 and 10, inclusive: 5
Incorrect. Try again.
Guess a number between 1 and 10, inclusive: 0
Invalid guess. Number must be between 1 and 10, inclusive.
Guess a number between 1 and 10, inclusive: 1
Incorrect. Try again.
Guess a number between 1 and 10, inclusive: 9
Incorrect. Try again.
Guess a number between 1 and 10, inclusive: 8
Correct!
linux6[270]% python3 hw4_part1.py
Guess a number between 1 and 10, inclusive: 1
Incorrect. Try again.
Guess a number between 1 and 10, inclusive: 2
Incorrect. Try again.
Guess a number between 1 and 10, inclusive: 3
Incorrect. Try again.
Guess a number between 1 and 10, inclusive: 4
Incorrect. Try again.
Guess a number between 1 and 10, inclusive: 5
You've used all your guesses.
```

hw4_part2.py – Grocery List (Worth 7 points)

Write a program that allows the user to enter grocery items necessary to make a particular dish. If the item is already on the list, print out a message and do not add it again.

Your program needs to use a sentinel while loop to get the items and store them in a list. The loop will quit when the user enters ‘q’ to signify they are done entering items.

You do not have to validate user input for this problem. You can assume that all things entered are valid items.

You may also assume that the user will enter at least one item.

Here is some sample output, with the user input in **blue**.

For this problem, your output should look EXACTLY like that shown below.

```
linux6[272]% python3 hw4_part2.py
What dish are shopping for? burritos

Items for the list (enter q to quit): cheese
Items for the list (enter q to quit): lettuce
Items for the list (enter q to quit): salsa
Items for the list (enter q to quit): beef
Items for the list (enter q to quit): q
Your burritos has 4 ingredients. They are cheese, lettuce,
salsa, beef.
linux6[273]% python3 hw4_part2.py
What dish are shopping for? apple pie

Items for the list (enter q to quit): apple pie
Items for the list (enter q to quit): q
Your apple pie has 1 ingredients. They are apple pie.
linux6[274]%
```

hw4_part3.py – Bank Account (Worth 6 points)

Write a program that manages the user's bank account. The user starts with $100 in their account. The user can choose among 4 options:

1. Deposit: asks the user to enter an amount and adds it to the balance
2. Withdraw: asks the user to enter an amount and subtracts it from the balance
3. Show balance: prints out how much money is in the account
4. Quit: prints out a thank you message and ends the program

You may assume that all menu input values are integers, but you must validate the value (i.e. it must be between 1 and 4, inclusive). You may assume that all deposits and withdrawals are valid (i.e. either integer or float).

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.)

```
linux6[252]% python3 hw4_part3.py
Welcome to your bank account. You currently have $100.

1) Deposit
2) Withdraw
3) Show balance
4) Quit
What would you like to do? 0
Invalid option
What would you like to do? 3
Your balance is $100

1) Deposit
2) Withdraw
3) Show balance
4) Quit
What would you like to do? 1
How much do you want to deposit? 50

1) Deposit
2) Withdraw
```

```
3) Show balance
4) Quit
What would you like to do? 3
Your balance is $150.0

1) Deposit
2) Withdraw
3) Show balance
4) Quit
What would you like to do? 2
How much do you want to withdraw? 20.50

1) Deposit
2) Withdraw
3) Show balance
4) Quit
What would you like to do? 3
Your balance is $129.5

1) Deposit
2) Withdraw
3) Show balance
4) Quit
What would you like to do? 4
Thank you for using our online banking!
```

hw4_part4.py – Blue-rays (Worth 9 points)

A popular internet-based Blu-ray rental company is buying up everyone's used Blu-rays. They want you to create a program that makes sure the Blue-rays they buy are in good shape and sold to them at a fair 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 the movie for, and finally it should ask the movie genre.

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 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 by 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 acceptable)
- Genre must be 'romance' or '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 Python interactive mode before you use it.

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[174]% python3 hw4_part4.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.

linux6[279]% python3 hw4_part4.py
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.
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: 50
We can't accept a price greater than $15
Please enter the price you want to sell for: 7.75
Enter the genre of the movie (romance, comedy): action
action is not a valid type, choose from romance or comedy
Enter the genre of the movie (romance, comedy): comedy
You are selling a comedy that has been watched 10 times
for $7.75.
```

hw4_part5.py – Pedometer (Worth 6 points)

Write a program that acts 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 while 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 and 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 will be greater than or equal to 0. You can also assume that you will walk a different number of steps each day.

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[174]% python3 hw4_part5.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.
```

Submitting

Once your **`hw4_part1.py`**, **`hw4_part2.py`**, **`hw4_part3.py`**, **`hw4_part4.py`**, and **`hw4_part5.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 may resubmit any or all your files as many times as you like up until the due date/time. Be aware that when you resubmit a file, you are overwriting the last version that you submitted!

You must be logged into your account on GL, and you must be in the same directory as your Homework 4 Python files. To double-check that you are in the directory with the correct files, you can type **`ls`**.

```
linux1[3]% ls
hw4_part1.py  hw4_part3.py  hw4_part5.py
hw4_part2.py  hw4_part4.py
linux1[4]% █
```

To submit your Homework 4 Python files, we use the **`submit`** command, where the class is **`cmsc201s`**, and the assignment is **`HW4`**. Type in (all on one line, even if it wraps around the screen) **`submit cmsc201s HW4 hw4_part1.py hw4_part2.py hw4_part3.py hw4_part4.py hw4_part5.py`** and press Enter.

```
linux1[4]% submit cmsc201s HW4 hw4_part1.py hw4_part2.py
hw4_part3.py hw4_part4.py hw4_part5.py hw4_part6.py
Submitting hw4_part1.py...OK
Submitting hw4_part2.py...OK
Submitting hw4_part3.py...OK
Submitting hw4_part4.py...OK
Submitting hw4_part5.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.**