Homework 4 – More Loops

Due Date: Monday, October 6, 2020 by 11:59:59 PM
Value: 50 points

This assignment falls under the standard cmsc201 academic integrity policy. This means you should not discuss/show/copy/distribute your solutions, your code or your main ideas for the solutions to any other student.

Make sure that you have a **complete file header comment at the top of <u>each</u> 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 **`for i`** and **`while`** loops, including any Boolean logic needed.

At the end, your Homework 4 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 hw4 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 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 the **`cmsc201`** 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 4 files in the same **`hw4`** 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)

Additional Specifications

For this assignment, **you must use** `if __name__ == "__main__":` 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'
```

Allowed Keywords/Built-Ins

For this assignment you should use:

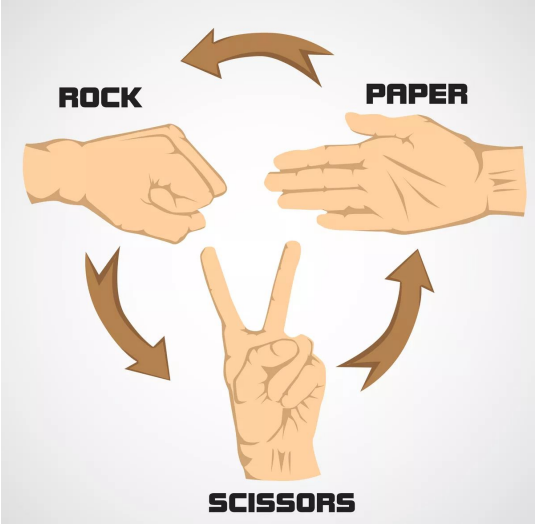
1. For loops, with either range, len
2. Lists, declaration, in keyword, append and remove functions, len
3. if/elif/else
4. print, string formatting is up to you
5. input
6. casting
7. variable declaration and assignment, algebra
8. String operations split(), strip(), concatenation (+)
9. while loops

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.

hw4_part1.py - Rock, Paper, Scissors (10 points)

Implement the game of rock paper scissors. Recall that the game works as such where the arrows direct victory:



This means that paper beats rock, rock beats scissors, and scissors beats paper.

You should loop while the user doesn't enter "stop".

Use `choice(['rock', 'paper', 'scissors'])` to get a random computer choice.

You will need to import the following modules under your header comment:

```
import sys
from random import choice, seed
```

For testing purposes, you must have this bit of code above your if __name__ == __main__ block:

```
if len(sys.argv) >= 2:
  seed(sys.argv[1])
```

Here is some sample output for **hw4_part1.py**, with the user input in **blue**.

```
linux[0]$ python3 hw4_part1.py
Enter rock, paper, or scissors to play, stop to end.
rock
Rock crushes scissors, you win.
Enter rock, paper, or scissors to play, stop to end.
paper
Both paper, there is a tie.
Enter rock, paper, or scissors to play, stop to end.
scissors
Rock crushes scissors, you lose.
Enter rock, paper, or scissors to play, stop to end.
blah
You need to select rock, paper or scissors.
Enter rock, paper, or scissors to play, stop to end.
rock
Paper covers rock, you lose.
Enter rock, paper, or scissors to play, stop to end.
rock
Both rock, there is a tie.
Enter rock, paper, or scissors to play, stop to end.
scissors
Both scissors, there is a tie.
Enter rock, paper, or scissors to play, stop to end.
paper
Both paper, there is a tie.
Enter rock, paper, or scissors to play, stop to end.
paper
Paper covers rock, you win.
Enter rock, paper, or scissors to play, stop to end.
stop
```

hw4_part2.py (10 points) - Caesar Salad Cipher

We're going to implement a Caesar Salad Cipher here. The standard Caesar Cipher works this way:

You pick an offset. Then for each letter, you'll move it up by that offset. So for instance:
'abc' with offset 3 will become 'def'
'Hello' with offset 7 will become 'Olssv'
'Robot' with offset 0 will become 'Robot'

We are not going to change anything which is not an ASCII uppercase or lowercase letter.

The equation is generally:

new_char = old_char + offset

But because 'z' + offset won't be in range you'll want to use mod.

new_char = (old_char + offset) mod 26

But what you ask, is a **Caesar Salad Cipher**?
Well in this case, what it does is not just add an offset, but it also will add the index squared to each position (starting at zero):

(Assuming that characters are between 0 and 25 instead of their ASCII values.)

new_char (at index i) = (old_char(at index i) + i^2 + offset) mod 26

This means that:

'Hello' encrypts to 'Omwbl' (with offset 7)

'Robot' encrypts to 'Usiam' (with offset 3)

What you will need for this problem:

- Uppercase ASCII letters start at ord(c) == 65 and go up to 90 (inclusively)
- Lowercase ASCII letters start at ord(c) == 97 and go up to 122 (inclusively).
- chr(my_int) will convert my_int into a character. If your character is not a potential ASCII value, you'll get an error.
- ord(my_char) will convert a character to its ASCII/unicode value.
- You should get the value of a letter like this:
    - uppercase: ord(char) - 65
    - lowercase: ord(char) - 97
- When you convert back to a character you need to add that value back:
    - uppercase: chr(value + 65)
    - lowercase: chr(value + 97)
- In order to keep it in range, you'll probably want to use mod.
- If it is not an alphabetical character, do nothing, even to numbers. This prevents us from accidentally mapping into ASCII characters which don't render on the screen properly.

```
linux[0]$ python3 hw4_part2.py
What is the string to encrypt? (or stop)Hello
What is the offset? 7
The encrypted string is: Omwbl
What is the string to encrypt? (or stop)Robot
What is the offset? 3
The encrypted string is: Usiam
What is the string to encrypt? (or stop)bucket of goo
What is the offset? 50
The encrypted string is: ztersq jp ada
What is the string to encrypt? (or stop)gaul is divided
into three parts
What is the offset? 6
The encrypted string is: mhea ni vrxfxxx kwlr ydgol wkgpx
What is the string to encrypt? (or stop)stop
```

hw4_part3.py - (10 points) - Draw a Circle, But Don't Stand in It

Let's draw a circle. Remember the circle equation with center (a,b) and radius r is:

$$(x-a)^2 + (y-b)^2 = r^2$$

Pretend that (a, b) = (0,0) and we'll restrict the radius between 0 and 20.

So the new equation is just:

$$x^2 + y^2 = r^2$$

In order to draw a circle you remember that the possible values for x and y are between:

$$-r \leq x, y \leq r$$

So you're going to need to modify your range to make sure you get the negative sides.

The circle will be oblong, really kind of an ellipse, mainly because you see that the characters are higher than they are wide. But don't worry about that too much/at all. Don't try to fix it by drawing an ellipse with major and minor axes in inverse proportion to character height/width.

```
linux[0]$ python3 hw4_part3.py
What is the radius? (between 0 and 20) 3
   *
 *****
 *****
*******
 *****
 *****
   *

linux[1]$ python3 hw4_part3.py
What is the radius? (between 0 and 20) 9
         *
      *********
     ***********
    *************
   ***************
  *****************
  *****************
  *****************
  *****************
*********************
  *****************
  *****************
  *****************
  *****************
   ***************
    *************
     ***********
      *********
         *
```

hw4_part4.py (10 points) - Checkerboard of Cards

Take in the size of a game-board and then draw an alternating pattern.

The symbols are '\u2660', '\u2665', '\u2666', '\u2663' they are unicode symbols using the escape sequence \u followed by 4 hexadecimal digits (keep them in this order).

Each row should start at the next element in the list to produce somewhat of a cycling pattern. Within each row, you should simply cycle through the elements in order.

So for row 0 you start with offset 0, and go through the list until you're at whatever length the user inputs.

For row 0, you start on \u2660 and cycle through.
For row 1, you start on \u2665 and cycle through.
For row 2, you start on \u2666 and cycle through.
For row 3, you start on \u2663 and cycle through.
For row 4, you start on \u2660 again and cycle through.
…

In order to be able to print single characters, you should use end='' optional keyword argument in print like so:

```
print(something, end='')
```

That will ensure that there is no newline, tab, or space after each printed character. (Default is end='\n')

```
linux[0]$ python3 hw4_part4.py
What size of board do you want? (between 1 and 50) 8
♠♥♦♣♠♥♦♣
♥♦♣♠♥♦♣♠
♦♣♠♥♦♣♠♥
♣♠♥♦♣♠♥♦
♠♥♦♣♠♥♦♣
♥♦♣♠♥♦♣♠
♦♣♠♥♦♣♠♥
♣♠♥♦♣♠♥♦

linux[1]$ python3 hw4_part4.py
What size of board do you want? (between 1 and 50) 4
♠♥♦♣
♥♦♣♠
♦♣♠♥
♣♠♥♦

linux[2]$ python3 hw4_part4.py
What size of board do you want? (between 1 and 50) 2
♠♥
♥♦

linux[3]$ python3 hw4_part4.py
What size of board do you want? (between 1 and 50) 6
♠♥♦♣♠♥
♥♦♣♠♥♦
♦♣♠♥♦♣
♣♠♥♦♣♠
♠♥♦♣♠♥
♥♦♣♠♥♦
```

hw4_part5.py - (10 points) - Number Guesser

You must implement a number guesser. Then you should ask the user for guesses until they guess the number.

- If the number is too big, tell them that.
- If the number is too small, tell them that.
- You should count the number of steps it took, and tell the user at the end.

You will need to import the following modules under your header comment:

```
import sys
from random import randint, seed
```

For testing purposes, you must have this bit of code above your if __name__ == __main__ block:

```
if len(sys.argv) >= 2:
  seed(sys.argv[1])
```

To use randint, you should specify the lower and upper bound (inclusive) in this case:

```
randint(1, 100)
```

```
linux[0]$ python3 hw4_part5.py
Guess a number between 1 and 100: 50
Your guess is too low.
Guess a number between 1 and 100: 72
Your guess is too high.
Guess a number between 1 and 100: 65
Your guess is too high.
Guess a number between 1 and 100: 64
Your guess is too high.
Guess a number between 1 and 100: 62
Your guess is too high.
Guess a number between 1 and 100: 57
You guessed the value!  It took you 6 steps.

linux[1]$ python3 hw4_part5.py
Guess a number between 1 and 100: 75
Your guess is too low.
Guess a number between 1 and 100: 80
Your guess is too low.
Guess a number between 1 and 100: 85
Your guess is too low.
Guess a number between 1 and 100: 90
You guessed the value!  It took you 4 steps.

linux[2]$ python3 hw4_part5.py
Guess a number between 1 and 100: 30
Your guess is too low.
Guess a number between 1 and 100: 70
Your guess is too high.
Guess a number between 1 and 100: 50
Your guess is too low.
Guess a number between 1 and 100: 60
Your guess is too high.
Guess a number between 1 and 100: 55
You guessed the value!  It took you 5 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 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 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 3 Python files, we use the **`submit`** command, where the class is **`cmsc201`**, and the assignment is **`HW4`**. Type in (all on one line) **`submit cmsc201 HW4 hw4_part1.py hw4_part2.py hw4_part3.py hw4_part4.py hw4_part5.py hw4_part6py`** and press enter.

```
linux1[4]% submit cmsc201 HW4 hw4_part1.py hw4_part2.py
hw4_part3.py hw4_part4.py hw4_part5.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.**