CMSC 201 Section 60

Homework 4 - Let's Loop Again

Due Date: Monday, October 9th, 2023 by 11:59:59 PM
Value: 40 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. Also, you should not post these problems on any forum, internet solutions website, etc.

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
"""
```

Creating Your HW4 Directory

```
linux3[1]% cd cmsc201/Homeworks
linux3[2]% mkdir hw4
linux3[3]% cd hw4
linux3[4]%
```

Submission Details

Submit the files under the following titles:
(These are case sensitive as usual.)
submit cmsc201 HW4 {files go here}

Problem 1 - Exceed the Gauss Sum	exceed_gauss.py
Problem 2 - Burger Builder	burger_builder.py
Problem 3 - Rock Paper Scissors	rock_paper_scissors.py
Problem 4 - Red Rover	red_rover.py
Problem 5 - Factor Me	factor_me.py

For example you would do:

```
linux1[4]% submit cmsc201 HW4 exceed_gauss.py
burger_builder.py rock_paper_scissors.py red_rover.py
factor_me.py
Submitting exceed_gauss.py...OK
Submitting burger_builder.py...OK
Submitting rock_paper_scissors.py...OK
Submitting red_rover.py...OK
Submitting factor_me.py...OK
linux1[5]%
```

From here, you can use **emacs** to start creating and writing your different Homework 4 Python programs.

You don't need to and should not make a separate folder for each file. You should store all of the Homework 4 files in the same **hw4** folder.

Coding Standards

Coding standards for CMSC 201 can be found here.

For now, you should pay special attention to the sections about:

- Naming Conventions
- Use of Whitespace
- Comments (specifically, File Header Comments)
- Variable names.

Use if __name__ == '__main__':
Make sure to include **if __name__ == '__main__':** under your header comment of each file and indent your code inside it.

Input Validation

For this assignment, you do **not** need to worry about **SOME** “input validation.”

If the user enters a different type of data than what you asked for, your program may crash. This is acceptable.

Unlike in HW1 you didn't have to worry about any input validation since you didn't have if statements, but now you do, so you can worry a little about it. You can try to prevent invalid input, but only when that invalid input is of the correct type.

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.

But it's a good idea to try to catch a zero division error for instance, or entering negative numbers when only positive numbers are allowed.

Here is what that error 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 Built-ins/Methods/etc

- Declaring and assigning variables, ints, floats, bools, strings.
- Casting int(x), str(x), float(x), (technically bool(x))
- Using +, -, *, /, //, %, **; +=, -=, *=, /=, //=, %=, **= where appropriate
- Print, with string formatting, with end= or sep=:
    - '{}'.format(var), '%d' % some_int, f-strings
    - Really the point is that we don't care how you format strings in Python
    - Ord, chr, but you won't need them this time.
- Input, again with string formatting in the prompt, casting the returned value.
- Using the functions provided to you in the starter code.
- Comparisons ==, <=, >=, >, <, !=, in
- Logical and, or, not
- if/elif/else, nested if statements
- Using import with libraries and specific functions **as allowed** by the project/homework.
- String Operations:
    - upper(), lower(), join(), strip(), split()
    - concatenation +, +=
- For loops, both *for i* and *for each* type.
- Lists, list(), indexing, i.e. my_list[i] or my_list[3]
    - 2d-lists if you want them/need them my_2d[i][j]
    - Append, remove
- len()
- While loops
    - sentinel values, boolean flags to terminate while loops

Forbidden Built-ins/Methods/etc

This is not a complete listing, but it includes:

- list slicing
- Declaring your own Functions
- Dictionaries
    - creation using dict(), or {}, copying using dict(other_dict)
    - .get(value, not_found_value) method
    - accessing, inserting elements, removing elements.
- break, continue
- methods outside those permitted within allowed types
    - for instance str.endswith
    - list.index, list.count, etc.
- Keywords you definitely don't need: await, as, assert, async, class, except, finally, global, lambda, nonlocal, raise, try, yield
- The *is* keyword is forbidden, not because it's necessarily bad, but because it doesn't behave as you might expect (it's not the same as ==).
- built in functions: any, all, breakpoint, callable, classmethod, compile, exec, delattr, divmod, enumerate, filter, map, max, min, isinstance, issubclass, iter, locals, oct, next, memoryview, property, repr, reversed, round, set, setattr, sorted, staticmethod, sum, super, type, vars, zip
- If you have read this section, then you know the secret word is: argumentative.
- exit() or quit()
- If something is not on the allowed list, not on this list, then it is probably forbidden.
- The forbidden list can always be overridden by a particular problem, so if a problem allows something on this list, then it is allowed for that problem.

Problem 1 - Exceed the Gauss Sum

Call your file exceed_gauss.py

The Gauss sum involves adding numbers from 1 to N given some final N.

Our goal in this program is to exceed the number that you input by calculating the Gauss Sum until the total value exceeds the inputted number.

For instance if we put in number = 5, then 1 + 2 + 3 = 6 which is greater than it, so it took 3 iterations to get bigger than 5.

For another example, if you choose the number to be 214, then as it turns out:
1 + 2 + 3 + … + 18 + 19 + 20 + 21 = 231 which is greater than 214. The number one less would be 210 which means that 231 is the smallest Gauss sum that will be bigger than the given number.

Notice here that we don't know the number of iterations we need. You must use a loop, even if you have another formula for the summation.

Sample Output:

```
linux[34]$ python exceed_gauss.py
What number do you want to test? 214
After 21 iterations, the gauss sum is 231 which exceeds
(or is equal to) the number 214

linux[35]$ python exceed_gauss.py
What number do you want to test? 28813119
After 7591 iterations, the gauss sum is 28815436 which
exceeds (or is equal to) the number 28813119

linux[36]$ python exceed_gauss.py
What number do you want to test? -4
After 0 iterations, the gauss sum is 0 which exceeds (or
is equal to) the number -4

linux[37]$ python exceed_gauss.py
What number do you want to test? 28
After 7 iterations, the gauss sum is 28 which exceeds (or
is equal to) the number 28

linux[38]$ python exceed_gauss.py
What number do you want to test? 729
After 38 iterations, the gauss sum is 741 which exceeds
(or is equal to) the number 729

linux[38]$ python exceed_gauss.py
What number do you want to test? 321123321123
After 801403 iterations, the gauss sum is 321123784906
which exceeds (or is equal to) the number 321123321123
```

Problem 2 - Burger Builder

You are tasked with building a hamburger or cheeseburger.

There of course will be a top and bottom bun, along with either hamburger, cheese, tomato, onions, jalapeno peppers, or mushrooms, or any other condiment you can think of.

You should always ask for what they want to put next onto the stack, they will start with the "bottom bun", as gravity dictates. Move up the stack of the burger adding layers until the user reaches the "top bun".

Then you should give results counting the number of burger layers, and then the condiment layers (not counting cheeses), but only output any given condiment or vegetable once.

Then finally, output whether it's a hamburger or cheeseburger whether it has cheese on it or not (just one layer of cheese makes it a cheeseburger).

Sample output:

User input is generally colored **blue** to help distinguish it from the rest of the text.

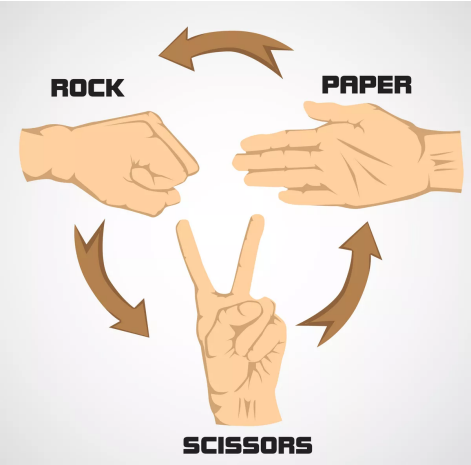
```
linux4[116]% python3 burger.py
What do you want to add? bottom bun
What do you want to add? ketchup
What do you want to add? burger
What do you want to add? cheese
What do you want to add? tomato
What do you want to add? jalapenos
What do you want to add? tomato
What do you want to add? mushrooms
What do you want to add? burger
What do you want to add? cheese
What do you want to add? onions
What do you want to add? top bun
You have created a 2-cheeseburger with the condiments:
ketchup, tomato, jalapenos, mushrooms, onions

linux4[117]% python3 burger.py
What do you want to add? burger
You must start with the bottom bun!
What do you want to add? ketchup
You must start with the bottom bun!
What do you want to add? bottom bun
What do you want to add? cheese
What do you want to add? burger
What do you want to add? cheese
What do you want to add? burger
What do you want to add? cheese
What do you want to add? burger
What do you want to add? cheese
What do you want to add? top bun
You have created a 3-cheeseburger with the condiments:
No Condiments
```

```
linux4[117]% python3 burger.py
What do you want to add? bottom bun
What do you want to add? burger
What do you want to add? ketchup
What do you want to add? onion
What do you want to add? tomato
What do you want to add? top bun
You have created a 1-hamburger with the condiments:
ketchup, onion, tomato
```

Problem 3 - Rock, Paper, Scissors

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 `the_choice = 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 rock_paper.py, with the user input in **blue**.

```
linux[0]$ python3 rock_paper.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
```

Problem 4 - Red Rover

In the game Red Rover, you have two teams of people. You can add people to either team, and then you say "send Rachel on over" which means that you will transfer Rachel from the current team to the other team. Let's say that the two teams are called "Red" and "Blue".

The person sent over has to try to break through the line. If they do, then they go back to their own team. If they don't they join the opposing team.

1) Add a person to the red team and then one person to the blue team. Make sure that you keep the teams the same size by alternating where people are added. Add people in pairs.
2) "begin the game" stop allowing people to be added and begin the game
3) display

   Display all the people on the current team. Then ask who to send over again.
4) [person]

   Send the person from the current team up to the other team. Ask if they broke through the line, if they did, then they stay on their own team, otherwise they must join the opposite team.

Once one of the teams is down to one person left, that team loses and the other team wins, having acquired everyone.

Whenever I write [A or B] I mean the string "A" or "B" but no brackets and no word or. Look at the sample output for clarity on that.

Sample output:

User input is generally colored **blue** to help distinguish it from the rest of the text.

```
linux4[116]% python3 red_rover.py
Who should we add to the Red team? Jim
Who should we add to the Blue team? Sally
Who should we add to the Red team? Archie
Who should we add to the Blue team? Saagar
Who should we add to the Red team? Jill
Who should we add to the Blue team? Jack
Who should we add to the Red team? Tim
Who should we add to the Blue team? Lex
Who should we add to the Red team? begin the game
Who should we add to the Blue team?
Who should Red team send over? display
The Red Team is composed of:
Jim, Archie, Jill, Tim,
Who should Red team send over? Archie
Did they make it through the line? yes
Archie stays on the Red team
Who should Blue team send over? display
The Blue Team is composed of:
Sally, Saagar, Jack, Lex,
Who should Blue team send over? Lex
Did they make it through the line? no
Lex changes to the Red team
Who should Red team send over? Jim
Did they make it through the line? yes
Jim stays on the Red team
```

Continued on the next page

```
Who should Blue team send over? Saagar
Did they make it through the line? no
Saagar changes to the Red team
Who should Red team send over? display
The Red Team is composed of:
Jim, Archie, Jill, Tim, Lex, Saagar,
Who should Red team send over? Jim
Did they make it through the line? no
Jim changes to the Blue team
Who should Blue team send over? display
The Blue Team is composed of:
Sally, Jack, Jim,
Who should Blue team send over? Sally
Did they make it through the line? no
Sally changes to the Red team
Who should Red team send over? Jim
Jim is not on the Red team
Who should Red team send over? display

The Red Team is composed of:
Archie, Jill, Tim, Lex, Saagar, Sally,
Who should Red team send over? Tim
Did they make it through the line? yes
Tim stays on the Red team
Who should Blue team send over? Jack
Did they make it through the line? no
Jack changes to the Red team
Who should Red team send over? display
The Red Team has won
```

It's ok if you ask the question one last time before checking about victory.

Problem 5 - Factor Me

Write this program in a file called factor_me.py

For this program you will factor a number into its prime factors less than 50. If the number has factors greater than 50, then we'll just leave them unfactored.

The list of primes less than 50 is:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

You don't need to get your own primes or write any is_prime type function.

Now, let's discuss prime factorization. Every positive integer can be factored into a unique product of primes. For instance $25 = 5^2$, and $24 = 2^3*3$, and perhaps something like $8085 = 3*5*7^2*11$.

We are going to determine the factorization of numbers that are input and display it in this form: 3*5*7*7*11 Repeated factors can just be repeated since we don't really have superscripts in a terminal.

Here's an example-hint for how to think about this process.

$128 = 2^7$, so what we would do is divide 128 / 2 and you see that it's still divisible by 2. So do it again, 64, again, 32, all the way down to something where you can't evenly divide out another 2.

Now, as you know we don't have every possible prime in our list, so it's possible that after dividing out all of the primes we have there's something left over. If so, just output that number at the end as "unfactorable." Of course in reality it is factorable as long as you have sufficiently many primes in your list.

```
linux3[14]% python3 factor_me.py
Enter a number to factor: 8085
The factors are: 3*5*7*7*11

linux3[15]% python3 factor_me.py
Enter a number to factor: 227056896
The factors are: 2*2*2*2*2*2*2*2*3*3*11*17*17*31

linux3[16]% python3 factor_me.py
Enter a number to factor: 128
The factors are: 2*2*2*2*2*2*2

linux3[17]% python3 factor_me.py
Enter a number to factor: 128
The factors are: 2*2*2*2*2*2*2

linux3[17]% python3 factor_me.py
Enter a number to factor: 7469
The factors are: 7*11
This part of the number couldn't be factored with
primes less than 50:  97

linux3[17]% python3 factor_me.py
Enter a number to factor: 97
We didn't find any factors
This part of the number couldn't be factored with
primes less than 50:  97
```