CMSC 201 Fall 2017
Lab 10 – For Loops

Assignment: Lab 10 – For Loops
Due Date: **During discussion**, November 6th through November 9th
Value: 10 points (8 points during lab, 2 points for Pre Lab quiz)

This week's lab will give you practice with debugging more complex problems, such as logic errors, that you may start encountering.

(Having concepts explained in a new and different way can often lead to a better understanding, so make sure to pay attention as your TA explains.)

Part 1A: Review – For Loops

We can use `for` loops to perform two different actions: *iterating* over a list, or performing an action a certain number of times. We will see both below. *Iterating* over a list means moving through a list, one element at a time.

For example:

```
fruitList = ["kiwi", "banana", "peach"]
for f in range(len(fruitList)):
    print("I ate a", fruitList[f])

Output:
I ate a kiwi
I ate a banana
I ate a peach
```

As another example:

```
greetings = ["Hello", "Hola", "Ciao", "Salut"]
for i in range(len(greetings)):
    greetings[i] = "Goodbye"
print(greetings)

Output:
['Goodbye', 'Goodbye', 'Goodbye', 'Goodbye']
```

We can also run a `for` loop a specific number of times. For example:

```
numToRun = 5
for n in range(numToRun):
    print("Executing for time #", n)

Output:
Executing for time # 0
Executing for time # 1
Executing for time # 2
Executing for time # 3
Executing for time # 4
```

You can either define the number of times to run prior to using it in the **`for`** loop (like we did above with **`numToRun`**), with an expression in its place (like we did with **`len(greetings)`**), or with just a number (as we will see below).

The loop variable in a **`for`** loop behaves in a specific way. The loop variable will first be set to the first value generated by range (often 0). On the next iteration of the loop, it will change to the next value (often 1, 2, 3, etc. – all the way up to **one less** than the number specified by **`range()`**.)

For example:

```
for i in range(4):
    print("The value of i is:", i)

Output:
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
```

You can also be much more specific in choosing the exact range that your **`for`** loop variable will iterate your code over.

For example:

```
for number in range(5, 8):
    print("This will execute 3 times:", number)

Output:
This will execute 3 times: 5
This will execute 3 times: 6
This will execute 3 times: 7
```

Notice how the previous loop will only execute 3 times, starting at the number 5 and ending at the number 7. **Specifying the starting number for the `range()` function is optional!** If you don't include it then 0 is chosen as the default start for the **`range()`** function. You can also optionally specify the increment that the **`range()`** function increases by for each iteration.

For example:

```
for num in range(1, 10, 2):
    print("The value increases by 2:", num)

Output:
The value of num is: 1
The value of num is: 3
The value of num is: 5
The value of num is: 7
The value of num is: 9
```

See how the above loop only executes 5 times? This time, the loop variable **`number`** will increment by 2 each iteration, instead of 1. **The third number, which specifies the increment of the range is also optional!** If you don't include it, an increment of 1 is chosen by default. For the **`range()`** function, the only hard requirement is a number to end the range on.

You can also use a negative increment in your **`range()`** function. This can be used to count down, or to iterate backwards through a list. If you use this, you need to **make sure that your starting number is higher than your ending number**, and make sure that the end of the range you specify is one less than where you actually want to stop.

For example:

```
for i in range(5, -1, -1):
    print("The value of i is:", i)

Output:
The value of i is 5
The value of i is 4
The value of i is 3
The value of i is 2
The value of i is 1
The value of i is 0
```

Part 2: Exercise

In this lab, you'll be creating one file, **`factors.py`**. You'll also be putting your skills at creating algorithms from scratch to the test, so think carefully about what your code needs to do and why.

You are also ***REQUIRED TO USE FOR LOOPS*** for this lab.

The program you'll be coding asks the user for a number (an integer), and then prints out all of the factors of that number. For example, if given 6, the factors are 1, 2, 3, and 6. The factors of 8 are 1, 2, 4, and 8. (See the sample output for more examples.)

Your program ***must adhere*** to the following requirements, but the details (whether you create a function, or write everything up in **`main()`**) is up to you.

- Must verify that the number entered is higher than zero
- Must store the factors of the number in a list as they are found
- Must iterate over that list to print out the final response
- *(HINT: You will need at least two for loops to complete this lab.)*

Tasks

- ☐ Create a **`factors.py`** file
- ☐ Carefully consider how to find a number's factors
- ☐ Write the code to determine the factors of the number
  - ☐ (A separate function or within main() are both acceptable)
- ☐ Write the code to iterate over the list of factors
- ☐ Show your work to your TA

Part 3A: Creating Your File

First, create the **`lab10`** folder using the **`mkdir`** command -- the folder needs to be inside your **`Labs`** folder as well.

Next, create a Python file called **`factors.py`** using the "**`touch`**" command in GL.

The "`touch`" command creates a new blank file, but doesn't open it.

Once a file has been "touched", you can open and edit it using emacs.

```
touch factors.py
emacs factors.py
```

The first thing you should do with any new Python file is create and fill out the comment header block at the top of your file. Here is a template:

```
# File:          FILENAME.py
# Author:        YOUR NAME
# Date:          11/TODAY/2017
# Section:       YOUR SECTION NUMBER
# E-mail:        USERNAME@umbc.edu
# Description:   YOUR DESCRIPTION GOES HERE AND HERE
#                YOUR DESCRIPTION CONTINUED SOME MORE
```

Part 3B: Sample Output

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.)

```
bash-4.1$ python factors.py
Enter a (positive) number to find the factors of: -7
Sorry, enter a number greater than zero: 0
Sorry, enter a number greater than zero: -9
Sorry, enter a number greater than zero: 6
1 is a factor of 6
2 is a factor of 6
3 is a factor of 6
6 is a factor of 6

bash-4.1$ python factors.py
Enter a (positive) number to find the factors of: 30
1 is a factor of 30
2 is a factor of 30
3 is a factor of 30
5 is a factor of 30
6 is a factor of 30
10 is a factor of 30
15 is a factor of 30
30 is a factor of 30

Enter a (positive) number to find the factors of: 123
1 is a factor of 123
3 is a factor of 123
41 is a factor of 123
123 is a factor of 123

bash-4.1$ python factors.py
Enter a (positive) number to find the factors of: 137
1 is a factor of 137
137 is a factor of 137

bash-4.1$ python factors.py
Enter a (positive) number to find the factors of: 1
1 is a factor of 1
```

Part 4: Completing Your Lab

Since this is an in-person lab, you do not need to use the **`submit`** command to complete your lab. Instead, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

Tasks

- ☐ Create a **`factors.py`** file
- ☐ Carefully consider how to find a number's factors
- ☐ Write the code to determine the factors of the number
  - ☐ (A separate function or within main() are both acceptable)
- ☐ Write the code to iterate over the list of factors
- ☐ Show your work to your TA

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave!