CSE 1321L: Programming and Problem Solving I Lab

Lab 5

Flow Control (Part 2)

What students will learn:

- Using **WHILE** loops.
- Using **FOR** loops.
- Using **Nested FOR** loops.

Content

If there is one thing computers are good at, it is repeating something over and over. The concept of repetition (which some call "iteration" and other "looping") is not terribly difficult since we humans repeat things in our daily lives. Any decent programming language is going to support iteration and usually allows for two different kinds of "looping templates". These templates are exactly what this lab is going to cover.

The two kings of loops we will cover are the **FOR** and **WHILE** loop. You want to memorize the template for these:

```
while <condition>:
      <body>

for <element> in <iterable>:
      <body>
```

In a **WHILE** loop, the <condition> can be anything (variable or expression) that resolves to a **Boolean** value (**True** or **False**). This could mean **Boolean** variables, as well as **comparison** or **logical expressions**.

In a **FOR** loop, the <iterable> can be anything that can be iterated over (we will see more about these at a later module. Right now, the only things that you can iterate over are **strings** and **ranges**). At each iteration of the loop, <element> will hold one element from the <iterable>. These elements are retrieved one by one, in the order that they appear in the <iterable>.

It is important to know **when** to use them. Here is an overall guideline to help you out:

- Use a **FOR** loop when you want to repeat something **a certain number of times**. For example:
  - If you want to repeat something 100 times.
  - If you want to count from 50 to 3000 in increments of 10.
- Use a **WHILE** loop when you **do not know how many times** something will repeat. For example:

- If you ask the user to enter a number between 1 to 10 and they consistently enter a higher number such as 45, then this loop could go on forever. Eventually, the user would enter a valid number.

As with previous weeks, all labs should have the appropriate file names:

- Lab5A.py
- Lab5B.py
- Lab5C.py

Lastly, make sure you review the sample output and make sure the output of your program follows the exact same format including the input statements, print statement, etc. As always, user input is shown in **red** and **bold**.

Lab5A: Largest of 10

For this lab, you will write a program that will ask the user to input **10 positive integer** numbers, **one at a time**. While it does this, the program should also keep track of the largest number the user has entered so far. Once the user enters 10 positive integers, the program should display the largest number entered by the user.

Requirements:

- Your solution must use a **FOR** loop exclusively for this lab.
- The user input must be handled inside the loop, and it must be handled dynamically so **do not** manually write 10 input statements.
- The program assumes the user will only enter valid positive integers, so no input validation is required.
- The program must keep track of the highest number entered **so far** by the user.
- We have not covered collections yet, so your solution **should not** use lists or other data structures.
- Make sure to review the Sample Output as guidelines for the prompts and print statements. Your output **must** match the Sample Output provided.

Hints:

- The user will input **positive integer** numbers, so what do you think is the lowest value the user may input?
- Your solution should not contain a collection such as a list, therefore the program cannot remember all the user input.

Sample Output #1

```
Please enter 10 numbers and this program will display the largest.
Please enter number 1: 50
Please enter number 2: 51
Please enter number 3: 10
Please enter number 4: 1
Please enter number 5: 99
Please enter number 6: 1000
Please enter number 7: 1010
Please enter number 8: 42
Please enter number 9: 86
Please enter number 10: 1000

The largest number was 1010
```

Lab5B: The Box

For this lab we will write a program that will print a box, and right triangles made of asterisks `*`. The size of each shape will be based on a size input by the user.

Requirements:

- Your solution must use **Nested FOR** loops exclusively to print the shapes.
  - You **CANNOT** use the multiplication operator ***** to repeatedly print a string.
- The user will enter a whole number for the size, there is no need to implement input validation.
- After the size value has been read, the program should use the size value to print a box, a right-facing right triangle, and a left-facing right triangle composed of asterisks `*`.
- The size value determines the dimensions of the shapes: length and width for the box, and height for the triangles.

Hint:

- You may require a nested **FOR** loop for each shape: one for the box, another for the left-facing right triangle, and another for the right-facing right triangle.
- Remember that when working with a nested for loop, each iteration of the "**outer**" **FOR** loop must wait for the "**inner**" **FOR** loop to finish executing.
- Each single iteration of the **Outer FOR** loop completes only after the **Inner FOR** loop has finished executing all its iterations.

Sample Output #1

```
Please enter a value for the size: 4
This is the requested 4x4 box:
****
****
****
****
This is the requested right-facing 4x4 right-triangle:
*
**
***
****
This is the requested left-facing 4x4 right-triangle:
   *
  **
 ***
****
```

Sample Output #2

```
Please enter a value for the size: 5
This is the requested 5x5 box:
*****
*****
*****
*****
*****
This is the requested right-facing 5x5 right-triangle:
```

```
*
**
***
****
*****
This is the requested left-facing 5x5 right-triangle:
    *
   **
  ***
 ****
*****
```

Sample Output #3

```
Please enter a value for the size: 8
This is the requested 8x8 box:
********
********
********
********
********
********
********
********
This is the requested right-facing 8x8 right-triangle:
*
**
***
****
*****
******
*******
********
This is the left-facing requested 8x8 right-triangle:
       *
      **
     ***
    ****
   *****
  ******
 *******
********
```

Lab5C: Say “please”

For this lab, we will create a simple program that keeps looping until the user inputs “please”.

Requirements:

- Since we do not know how many times this loop will keep iterating, the most fitting type of loop to be used here is a **WHILE** loop.
- Your solution must implement a **WHILE** loop exclusively.
- Keep asking ‘*If you would like to stop this program, say “please”: ’*
- Keep looping until the user enters “please”.
- The program should terminate if the user enters “please”
    - The user input should be case-sensitive, meaning the input **must** be exactly “please” in all lowercase letters.

Sample Output #1

```
If you would like to stop this program, say "please": please
Program complete
```

Sample Output #2

```
If you would like to stop this program, say "please": pLeAsE
If you would like to stop this program, say "please": Please
If you would like to stop this program, say "please": please
Program complete
```

Sample Output #3

```
If you would like to stop this program, say "please": no
If you would like to stop this program, say "please": nah
If you would like to stop this program, say "please": nuh-uh
If you would like to stop this program, say "please": ok fine
If you would like to stop this program, say "please": please
Program complete
```

Submission Instructions:

- Programs must follow the output format provided. This includes each blank line, colons (:), and other symbols.
- Programs must be working correctly.
- Programs must be written in Python.
- Programs must be submitted with the correct **.py** format.
- Programs must be saved in files with the correct file name:
    - Lab5A.py
    - Lab5B.py
    - Lab5C.py
- Programs (source code files) must be uploaded to Gradescope by the due date.