CSE 1321L: Programming and Problem Solving I Lab

Lab 6

Flow Control (Part 3)

What students will learn:

- Using WHILE loops.
- Using FOR loops.
- Using NESTED FOR loops.

Content

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.

Lastly, in **nested loops**, each iteration of the **outer loop** completes only after all iterations of the **inner loop** are executed. Consider the total number of iterations in a nested loop.

For example:

```
for x in range(3):
    for y in range(5):
        …
```

The total amount of iterations that would have been performed by the end of this nested loop would have been 15, since a single iteration of the outer loop the inner loop performed five. The more loops we nest, we are going to exponentially increase the number of total iterations.

In this example:

```
for x in range(3):
      for y in range(5):
            …
      for y in range(3):
            …
```

The total amount of iteration would be 24, since we only have "2" levels of nesting, a single iteration of the outer loop would be composed of 8 iterations of the inner loop.

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

- Lab6A.py
- Lab6B.py
- Lab6C.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**.

Lab6A: Multiplication and Exponentiation Calculator

This program will calculate results for multiplication or exponentiation using nested loops. We will ask the user what operation the program should perform, and based on the selection we will ask the operands or the base and exponent input. Finally, the program will display the corresponding result.

Requirements:

- You are NOT allowed to use the multiplication * or exponentiation ** operator for this lab. Instead, you must use repeated addition and nested loops.
- Your solution **must use nested FOR loops** to compute multiplication and exponentiation.
- All inputs should be treated as integers.
- The calculator provides three options for users to choose from:
    - Option 1 for Multiplication
    - Option 2 for Exponentiation
    - Option 3 to exit
- Your program should perform the corresponding operation based on the user choice.
- After displaying the result for option 1 or 2, prompt the user to select another operation.
- The program should output "Invalid Choice" if the user enters a choice that is not 1, 2, or 3.

Hints:

- Remember you cannot use the multiplication or exponentiation operator, so you must have a combination of nested loops and the addition operator to calculate the corresponding operation.
- Try breaking down the calculation using just addition.
- You are going to have to work with several loops, one will be the "main" loop of the program, which is going to keep our program working, while the other loops will be used to perform the corresponding calculations.
- As always, you must follow the output format provided in the Sample Outputs.

Sample Output #1

```
Multiplication and Exponent Calculator
Choose option 1 for Multiplication
Choose option 2 for Exponentiation
Choose option 3 to Exit
1

Enter an operand: 4
Enter the other operand: 5
4 x 5 = 20

Multiplication and Exponent Calculator
Choose option 1 for Multiplication
Choose option 2 for Exponentiation
Choose option 3 to Exit
2

Enter the base: 4
Enter the exponent: 3
```

```
4^(3) = 64

Multiplication and Exponent Calculator
Choose option 1 for Multiplication
Choose option 2 for Exponentiation
Choose option 3 to Exit
10

Invalid Choice

Multiplication and Exponent Calculator
Choose option 1 for Multiplication
Choose option 2 for Exponentiation
Choose option 3 to Exit
3

Closing the Calculator...
```

Lab6B: Guess the Number

This program challenges users to guess a randomly generated number. The user will have to guess a number from 1 to 100 and the program should inform the user whether their guess is too low or too high or if they guessed the correct number.

Instead of hardcoding a number, we will use a Random Number Generator (RNG) to generate a random number. The RNG generates a random number within a specified range, in this case between 1 and 100.

This is how you can use an RNG in python:

1. First you need to import the **random** library (this import statement should be at the top of your code):

   ```
   import random
   ```

2. Then, define a variable to store the randomly generated number, and initialize it with the random function:

   ```
   random_number = random.randint(10, 20)
   ```

In the example provided, the computer will generate a number between the range of **inclusive** 10 and **inclusive** 20. Adjust the range to generate a number between 1 and 100.

Requirements:

- Your solution must implement an **RNG**.
- The program should keep asking the user for a guess until the user enters the correct value and guesses the randomly generated number.
- If the user correctly guessed the number, the program should print “Correct! I was thinking of X” where X must be the random number, not the user’s guess.
- If the user did not correctly guess, then the program will notify the user if their guess is too low or too high.
- There is no need for input validation, we will assume the user will always input a number from 1 to 100.
- The RNG will only generate a random integer number, so the user guess should also be read as an integer.
- As always, you must follow the output format provided in the Sample Outputs

Note:
Each time the program runs, the random number generated may differ. Therefore, **the autograder will not work for this specific lab because of the Random Number Generator.**

Sample Output #1:
```
Guess the number I am thinking!
Enter any number between 1 and 100: 50
Too high!
```

```
Enter any number between 1 and 100: 25
Too high!
Enter any number between 1 and 100: 12
Too high!
Enter any number between 1 and 100: 6
Too low!
Enter any number between 1 and 100: 9
Too low!
Enter any number between 1 and 100: 10
Correct! I was thinking of 10
```

Sample Output #2:

```
Guess the number I am thinking!
Enter any number between 1 and 100: 1
Too low!
Enter any number between 1 and 100: 100
Too high!
Enter any number between 1 and 100: 83
Correct! I was thinking of 83
```

Lab6C: Pyramid Pattern Printing

Write a program that prompts the user for a positive integer and prints a pyramid pattern with numbers. The program should prompt the user to enter a positive number or to enter 0 to terminate the program. The input value will be used to determine the number of rows and the numbers inside those rows.

Requirements:

- The pyramid pattern printing must be implemented using a **nested FOR loop**.
- Assume that the user will only enter any positive number and 0.
- After successfully printing the pyramid pattern, the program should re-prompt the user to enter another value or 0 to terminate.
- The row input will determine the number of rows for the pyramid and will also determine the content for each row.
    - Each row should have exactly `` `(row * 2) - 1` `` numbers. For example, row one will have 1 number, while row four will have 7.

Sample Output #1:

```
Enter Number for Rows or 0 to quit: 6
     1
    212
   32123
  4321234
 543212345
65432123456
Enter Number for Rows or 0 to quit: 3
  1
 212
32123
Enter Number for Rows or 0 to quit: 0
```

Sample Output #2:

```
Enter Number for Rows or 0 to quit: 1
1
Enter Number for Rows or 0 to quit: 2
 1
212
Enter Number for Rows or 0 to quit: 3
  1
 212
32123
Enter Number for Rows or 0 to quit: 4
   1
  212
 32123
4321234
Enter Number for Rows or 0 to quit: 0
```

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:
    - Lab6A.py
    - Lab6B.py
    - Lab6C.py
- Programs (source code files) must be uploaded to Gradescope by the due date.