ICS3U: Nested Loops

Much like nested `if` blocks, Python allows a programmer to nest one loop inside of another. Each loop is indented one additional level. Nested for loops, for example, would look something like this.

```
for variable in sequence:
   for variable2 in sequence2:
      for variable3 in sequence3:
         …
```

Nested while loops would have a similar structure.

```
while condition1:
   while condition2:
      while condition3:
     …
```

Nested loops are used when an iterative process must itself be repeated several times. For instance, the process of reading values and finding their average is often best done as a loop. If this averaging must be done multiple times, then the averaging loop could be nested inside of another loop that would determine the number of times it runs.

The code below prints multiplication facts from 1×1 to 10×10. The **outer loop** iterates 10 times. For each iteration of the outer loop, the **inner loop** will iterate 10 times. This results in $10 \times 10 = 100$ iterations in total. Try it and see the output for yourself.

```
for a in range(1, 11):
   for b in range(1, 11):
      print(a, "x", b, "=", a*b)
```

The program begins by initiating a loop that assigns a value of 1 to `a`. Inside of this loop, a second loop is started, assigning a value of 1 to `b`. The value of `b` is incremented by 1 during each iteration of the inner loop until it reaches a maximum value of 10. Thus, at this point the program has generated all facts from $1 \times 1 = 1$ to $1 \times 10 = 10$. Since `b` has reached the final value in the `range` of the inner loop, the program returns to the outer loop, which increments the value of `a` by 1, giving it a value of 2. The inner loop is started again, assigning the values 1 – 10 to `b`. This produces the facts from $2 \times 1 = 2$ to $2 \times 10 = 20$. This process continues until the outer loop assigns the final value of 10 to `a`, resulting in the facts from $10 \times 1 = 10$ to $10 \times 10 = 100$.

A common application of nested loops is to “wrap” an existing program with a loop that allows for multiple runs. Below is an example.

```
user_action = "Y":
while user_action == "Y" or user_action == "y":
   num = int(input("Enter a positive integer: "))
   for count in range(1, num):
      print(count)
   user_action = input("Run again? Enter 'Y' to run, anything else to stop: ")
```

In this example, the main programs itself asks the user to enter a number, then prints the values from 1 up until that number. When the main program finishes, the user is asked if they want to run it again. As long as the user chooses to do so and either of the conditions `user_action == "Y"` or `user_action == "y"` is `True`, then the program will repeat itself.

As more and more loops are nested, the program will complete more and more iterations each time. A program that contains multiple nested loops may take a long time to complete. Consider a program that contains three nested loops, each of which iterates 100 times each. It might look something like the code below.

```
for x in range(100):
   for y in range(100):
      for z in range(100):
         print("hello")
```

This will result in the `print` statement being executed $100 \times 100 \times 100 = 1\,000\,000$ times. For larger numbers of iterations, this will be an even greater number. Due to the rapid growth factor, many programmers tend to avoid nesting loops unless they are either necessary, or when they greatly simplify the implementation of an algorithm.