Control Flow: Loop Statements

A loop repeatedly executes a suite of sub-statements, called the loop body. Python provides two kinds of loop statements: a for-loop and a while-loop. This exercise gives you practice with loops and with the Python Turtle module.

Part (a) [For-loop]: A for-loop has the form:

```
for v in collection:
    suite
```

where v is a loop variable; collection is a sequence or set of items, called a collection; and suite is an indented sequence of statements, called the loop body.

To execute a for-loop, the console assigns successive values from collection to the loop variable and executes suite following each assignment, as shown in the flow diagram on the right. Each execution of the loop body is called an iteration of the loop.

Consider the program and the flow diagram for it below. Discuss with your partner how the flow diagram below was produced from the program and the flow diagram above.

```python
1  w = input("Enter a word: ")
2  r = ""
3  for c in w:
4      r = c + r
5  print(w, "=>", r)
```

In Python, a str is a collection of characters (str's of length 1). So line 4 will assign each character from w to the variable c, in order, and execute line 5 (the loop body) on successive iterations.
Assume that the user enters **CSE 231** at the prompt (i.e., the user touches the C-key, then the S-key, then the E-key, then the space-bar, then the 2-key, then the 3-key, then the 1-key, and finally the enter-key). With your partner, ‘execute’ the program on paper: One of you keeps track of the statement that will be executed next, while the other keeps track of the current value of each variable.

What output will the program produce for this user input?

AFTER you have completed the above, check your understanding by stepping through the visualization under the Visualization1 link in the Artifact section of today’s website.

**Part(b) [Iterating over a range]**: Python provides many functions that return collections for a program to iterate over. One of the most useful is the range function, which returns either a collection of integers or, if there are no integers in the range, an empty collection. The simplest form of the range function takes a single argument:

\[
\text{range}(\text{stop}) : \text{returns the range of integers that begins with 0 and counts up to, but does not include, stop}
\]

Working with a partner, download either trianglesPC.py, if you have a PC, or trianglesPC.py, if you have a MAC, into the folder you created for this week and open the downloaded file in Spyder (use File => Open or its shortcut, ⌘O).

This program uses the turtle module. If you performed the Hour of Code exercise recommended in the CSE 231 Welcome message, you already know about this module. If you did not, refer to the “Turtle Cheat Sheet” as needed.

Run the program. It should create a Python Turtle Graphics window and draw a (randomly) colored triangle in the window. If you don’t see the window, look for it under other windows on your display. (Don’t take too long to find it – it will disappear in 5 seconds.)

With a partner, discuss how the program works, referring to the cheat sheet as necessary. Check your understanding by answering the following:

- What lines of code make up the body of the for-loop?

---

1 The other two forms of the range function take either two or three arguments:

- **range**(start, stop): returns the range that begins with **start** and counts up to, but does not include, **stop**
- **range**(start, stop, step): returns the range that begins with **start**, counts (up or down) by **step**, and goes to, but does not include, **stop**
• How many times does the shell execute the body of the for-loop?

• On what iteration of the for-loop does the turtle draw the triangle’s base? ... its left side? ... its right side?

• What are the coordinates of the bottom left and bottom right corners of the triangle?

• Why is 120 used for the argument in the `turtle.left(120)` statement?

• What is the effect of the `time.sleep(5)` statement (line 23)? (Try running the program with that line commented out to check.)

Modify the program to prompt the user for a length in pixels (an `int` between 0 and 600) and then:

1. draw a filled equilateral triangle with side-length equal to the user-provided length and with the base centered on the origin, and
2. display what side is being drawn in the console just before drawing each side.

For example (here, the user entered the ‘200’ after the first prompt):
Part (c) [While-loop]: A while-loop is more versatile than a for-loop. The form of a basic while-loop is:

```
while cond:
    suite
```

where `cond` is a loop condition (Boolean expression) and `suite` is a loop body (indented sequence of statements).

To execute a while loop, the console evaluates `cond` and, if `cond` is `True`, it repeatedly executes `suite` until `cond` becomes `False`, when it terminates the loop. The flow diagram to the right depicts the control flow.

A common input pattern is to repeatedly request a “legal” input, until the user enters one. You can implement this pattern using a while loop, as shown by the pseudo-code to the right.

Modify your triangles program so that it prompts the user for a positive length that is no more than 600, repeatedly, until the user enters a legal length. It should then draw the triangle as in part (b), with the base on the x-axis and centered on the y-axis. Also, it should print an error message if the user enters an illegal integer. For example:

(You will fix the problem of the triangle being “cut off” in a later part of this exercise.)

---

2 Every for-loop can be replaced with a while-loop; but some problems—e.g., the input problem in this part—cannot be solved using a for-loop.

3 For simplicity, we describe the typical case. In general, the loop condition can have any type; Python treats a loop condition that is non-zero or non-empty as `True`, and a loop condition that is zero or empty as `False`. 
Part (d) [Problem solving, nested loops]: Modify your program so that it draws a “row” of as many triangles as can fit, side-by-side, within the coordinates (-300, 0) to (300, 0). Center the row of triangles with respect to the y-axis. Select a random color for each triangle. Use the length entered by a user. In the following example, you also see some output produced by my program, which I used for debugging purposes:

![Image of triangle drawing](image1.png)

Before writing any code, work out several examples on paper using some easy input lengths, e.g., 150, 175. How can you calculate the maximum number of triangles that will fit? Knowing the number of triangles, how can you calculate where to start drawing each triangle in its turn?

Part (e) [Extra problem solving for experts]: Modify your program so that it draws as many rows of triangles as can fit, stacked one on top of the other, within a square 600 pixels wide and 600 pixels tall and centered at the origin. Hint: You can use some trigonometry to calculate the height of the triangles (a row). Then, you can use essentially the same logic as in part (d) to calculate how many rows will fit and where to start drawing each row.

![Image of triangle drawing](image2.png)
Part (f) [For gluttons, some exam practice problems]: Trace the execution of the following program assuming the user enters a 319 at the prompt. Trace as long as needed to figure out what it displays for this input.

```
num = int(input("Enter a positive integer: ")) #1
rev = 0 #3
rem = num % 10 #4
quot = num // 10 #5
while (quot + rem) > 0: #6
    rev = 10 * rev + rem #7
    rem = quot % 10 #8
    quot = quot // 10 #9
print("Input: ", num, end="; ") #10
print("Output: ", rev) #11
```

What will it display if the user enters 5003012 at the prompt?

Trace the execution of the following program assuming the user enters a 5 at the prompt. Trace as long as needed to figure out what it displays for this input.

```
num_str = input("Enter a positive integer: ") #1
num = int(num_str) #2
fac = 1 #3
for i in range(1, num): #4
    print(i, '* ', end=' ') #5
    fac *= i #6
fac *= num #7
print(num, '=', fac) #8
```