Control Flow: Loop Statements

Loops comprise a second major category of control flow statements. Loops implement repetition. You use a loop to make the shell execute a suite of statements, called the loop body, multiple times in a row.

Part (a): [For-loop] You can use a for-loop when you know or can calculate the number of times that the shell should execute the loop body. The general form of a basic for-loop is as follows, where \( v \) denotes a variable, \( \text{collection} \) denotes an expression that produces a sequence of values when executed, and \( \text{suite} \) denotes the loop body:

\[
\text{for } v \text{ in } \text{collection}: \\
\text{suite}
\]

To execute this statement, the shell first executes \( \text{collection} \) to create a sequence of values. It then repeatedly assigns the loop variable, \( v \), to successive values from this sequence and executes the loop body, \( \text{suite} \), after each assignment until all values of the collection have been assigned. Each execution of the loop body is called an iteration of the loop. A string expression is an example of a sequence of values: It is a sequence of character values, or strings of length 1.

To illustrate, we will begin creating an execution trace of the following program. An execution trace shows the statements that are executed, in execution order, and also shows the value each assignment associates with a variable. Output is shown elsewhere—off to the side or below or above the trace.

```python
1 SAYING = "Of course, I talk to myself--sometimes I need expert advice!"
2 VOWELS = 'aeiou'
3 
4 print()
5 
6 count = 0
7 
8 for c in SAYING:
9     if c.lower() in VOWELS:
10         count += 1
11 print("The saying: ", SAYING, ' contains', count, ' vowels. ')```

Part(b): [Iterating over a range] Working with a partner, bring up Spyder and change the working folder to a new folder for today's files (create it in Spyder or create it externally). Download `triangles.py` into this folder and open the file in Spyder (use either **File => Open** or its shortcut, ⌘O).

This program uses the turtle module. If you viewed the video for your second programming project, you already know about this module. You can also refer to the "Turtle Cheat Sheet" as needed.

The program also uses the Python **range** function, which creates a sequence of int values. To learn about this function, enter `help(range)` in the shell. Widen the shell window for easier reading and then scroll up through the window to the start of the documentation output by the shell (to the line **Help on class range in module builtins**:). Below this line, it says **class range**(object). A **class** in Python is a type. This first line therefore indicates that **range** is a built-in Python type.

Like any other type, applying the type name to arguments produces a value of the named type. The line `range(stop) -> range object` indicates that **range** can be applied to one argument, denoted by the name **stop**, and that applying **range** to a legal argument creates an object of type **range**. The square brackets in the next line signify that the enclosed arguments are optional. So this line indicates that **range** can also be applied to either two or three arguments to create an object of type **range** and, like the previous line, it provides names to stand for the arguments. Following this line, you will find a short description of the object created by applying **range** to legal arguments. Read over this description. (There is no need to read past the line **Methods defined here**:.)

Run the program a few times and discuss with your partner the following:

1. What lines make up the body of the for-loop?
2. How many times does the shell execute the body of the for-loop?
3. Why is a 1 added to `i` in the **print** statement (line 18) and what is the effect of executing the statement?
4. On what iteration of the for-loop does the turtle draw the triangle's base? ... its left side? ... its right side?
5. What purpose is served by the last **input** expression (line 22)?

---

1 Classes will be covered in some detail in the second half of the semester.
2 Methods will also be covered in some detail in the second half of the semester.
Part (c): [Placing and filling the triangle] The program currently draws just the outline of an equilateral triangle whose sides are 100 pixels long; left-most corner is at the origin, (0, 0)); and base is along the x-axis. Modify the program so that it draws a filled triangle whose the base is centered with respect to the y-axis, as shown below (the lengths of sides can still be 100 pixels).

![Filled triangle](image)

Part (d): [User input] Modify the program so that it prompts for a length (an int, which must be positive and no more than 600, in pixels); and then draws a filled equilateral triangle with the indicated side-lengths, base along the x-axis and centered around the y-axis. Do not check that the user enters a legal value. For example:

```
Enter the length of a side (0 < length <= 600): 200
Drawing line 1
Drawing line 2
Drawing line 3
Press the "enter" key to quit the program.
```
Part (e): [While-loop] A while-loop is more versatile than a for-loop. You don’t have to calculate the number of times the shell should execute a loop body to use a while-statement. The form of a basic while-loop is as follows, where cond denotes an expression and suite denotes the loop body (a suite of statements):

```python
while cond:
    suite
```

To execute a while-statement, the shell first executes the loop condition, cond. If doing so produces True, the shell executes the loop body, suite, and then returns to the top of the loop and repeats this process; if executing the loop condition ever produces False, the shell exits the while loop. The loop condition typically depends on one or more input values.

Repeatedly prompting a user for an input value that satisfies some input conditions requires a while loop, since the programmer cannot know how many times a user might enter an illegal input.

Add a while-loop to your triangles program so that, as long as the user enters an int not in the specified range, it repeatedly prints an error message and prompts the user for a legal length; and, when the user enters a legal length, the program draws the triangle as in part (d). For example:

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

---

3 In fact, every for-loop can be replaced with an equivalent while-loop. I highly recommend that you figure out how this can be done before your first exam.
4 This explanation is simplified by describing the typical case. In general, the loop condition can have any type. In the general case, 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 (f): [Nested loops, Problem solving] 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 shell output produced by my program, which I used for debugging purposes:

![Shell output and triangle drawing](image)

Before writing any code, you should work out several examples on paper with some easy input lengths, e.g., 150, 200. 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 (g): [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 (not drawn) 600 pixels wide and 600 pixels long 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 (f) to calculate how many rows will fit and where to start drawing each row.

![Shell output and triangle drawing](image)
Part (h): [For gluttons, some exam practice problems] On paper and pencil, 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.

```python
num_str = input("Enter a positive integer: ")  #1
num = int(num_str)  #2

rev = 0  #3
quot, rem = num // 10, num % 10  #4

while (quot + rem) > 0:  #5
    rev = 10 * rev + rem  #6
    quot, rem = quot // 10, quot % 10  #7

print("Input:", num, end="; ")  #8
print("Output:", rev)  #9
```

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

On paper and pencil, 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.

```python
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
```