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:

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

where `v` is a **loop variable**; `collection` is a **collection** (sequence or set of values); and `suite` is a **loop body** (sequence of statements indented below the loop header).

To execute it, `v` is assigned successive values from `collection` and `suite` is executed 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.

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

```python
SAYING = "Seek and ye shall find."
VOWELS = 'aeiou'
CONSONANTS = 'bcdfghjklmnpqrstvwxyz'
v_num, c_num = 0, 0

for c in SAYING:
    if c.lower() in VOWELS:
        v_num += 1
    elif c.lower() in CONSONANTS:
        c_num += 1

print("\nThe saying is " + SAYING +"")
print("It contains:"
print("\t" + str(v_num) + " vowels")
print("\t" + str(c_num) + " consonants")
print("\t" + str(len(SAYING) - v_num - c_num) + " other characters")
```
**Part (b) [Iterating over a range]:** Python provides many functions that return collections. One of the most useful is the `range` function. It can be called with one, two, or three arguments:

- **`range(stop)`**: returns 0, 1, 2, ..., stop - 1
- **`range(start, stop)`**: returns start, start + 1, start + 2, ..., stop - 1
- **`range(start, stop, step)`**: returns start, start + step, start + 2*step, ..., N, where N is either the largest integer such that N + step >= stop, if step is positive; or the smallest integer such that N + step <= stop, if step is negative; and step is not 0.

Working with a partner, download `triangles.py` into the folder you created for this week and open the file (use either 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.

With your partner, discuss the code and run it a few times. Check your understanding:

- What lines of code make up the for-loop?
- How many times does the shell execute the body of the for-loop?
- Why is a 1 added to `i` in the `print` statement (line 18) and what is the effect of executing the statement?
- On what iteration of the for-loop does the turtle draw the triangle’s base? ... its left side? ... its right side?
- What purpose do you think the last `input` expression (line 22) serves? (If you are uncertain, try running the program with the line commented out.)

Modify the program so it asks the user for a length in pixels (a positive `int`), and then draws a filled equilateral triangle with the indicated side-length. For example:
Part (c) [While-loop]: A while-loop is more versatile than a for-loop. The form of a basic while-loop is as follows:

```python
while condition:
    suite
```

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

To execute it, `condition` is evaluated and, if `condition` is True, `suite` is executed, repeatedly, until `condition` becomes False, terminating the loop. The flow diagram to the right depicts the control flow.

A common input pattern is to request input that satisfies some condition, repeatedly, until the user enters a legal input. This pattern is implemented using a while loop, as shown by the pseudo-code to the right.

Add a while-loop to your triangles program so that, as long as the user enters an `int` that is 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 (b). For example:

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

---

1 Every for-loop can be replaced with a while-loop; but some problems, e.g., the common input problem solved by the pseudo-code in this part, require a while-loop—they cannot be solved using just a for-loop.

2 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) [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 output produced by my program, which I used for debugging purposes:

Before writing any code, work out several examples on paper using 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 (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 (not drawn) 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 (e) to calculate how many rows will fit and where to start drawing each row.
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.

```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?

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
```