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 sequence or set of values, called a **collection**; and **suite** is an indented sequence of statements, called the **loop body**.

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

To illustrate, we will assume the user enters $1.20 at the prompt in the program below and create an **execution trace**, which lists the statements executed, in order, and shows the affects of assignments; output is shown off to the side or below or above the trace.

```python
P = "Enter a price: 
D = 0.15
ps = input(P)
am = ''
df = False
ff = True
for n in ps:
    if n in '0123456789':
        am += n
    elif n == '.' and not df:
        am += n
        df = True
    elif n == '$' and ff:
        ff = False
    elif n != '!':
        print("Error in input:", ps)
a = float(am)
d = a * D
print("Cost: $" + str(round(a - d, 2)))
print("You saved: $" + str(round(d, 2)))
```
**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. It returns either a collection of integers or, if there are no integers in the range, an empty collection:

- **range(stop):**
  begins at 0, counts up by 1, goes up to but does not include `stop`

- **range(start, stop):**
  begins at `start`, counts up by 1, goes up to but does not include `stop`

- **range(start, stop, step):**
  begins at `start`, “counts” by `step`, goes up/down to but does not include `stop`

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, run it a few times. Discuss it, referring to the cheat sheet as necessary. Check your understanding:

- What lines of code make up the body of the for-loop?
- How many times does the shell execute the body of the for-loop? Why?
- 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 whose base is centered on the origin. For example:
**Part (c) [While-loop]**: A while-loop is more versatile than a for-loop. The form of a basic while-loop is:

```python
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 part (e) of this exercise.)

---

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

![Image of triangles](image)

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 (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.

![Image of stacked triangles](image)
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, '*\n', end=' ') #5
    fac *= i #6

fac *= num #7
print(num, '='\n, fac) #8
```