REPETITION STRUCTURES

Introduction

A repetition structure causes a statement or set of statements to execute repeatedly.

A repetition structure, which is more commonly known as a ***loop*** **causes repeating of a sequence of statements.**

Instead of writing the same sequence of statements over and over we can write the code for the operation once, and then place that code in a structure that makes the computer repeat it as many times as necessary.

There are two types of loops:

- ***A count-controlled loop*** repeats a specific number of times. (for)
- ***A condition-controlled loop*** uses a true/false condition to control the number of times that it repeats. (while)

The *while* Loop: A Condition-Controlled Loop

A condition-controlled loop causes a statement or set of statements to repeat as long as a condition is true. In Python you use the ***while*** statement to write a condition-controlled loop.

The *while* loop work: *while a condition is true, some task is executed.*

The loop has two parts:
(1) a condition that is tested for a true or false value
(2) a statement or set of statements that is repeated as long as the condition is true.

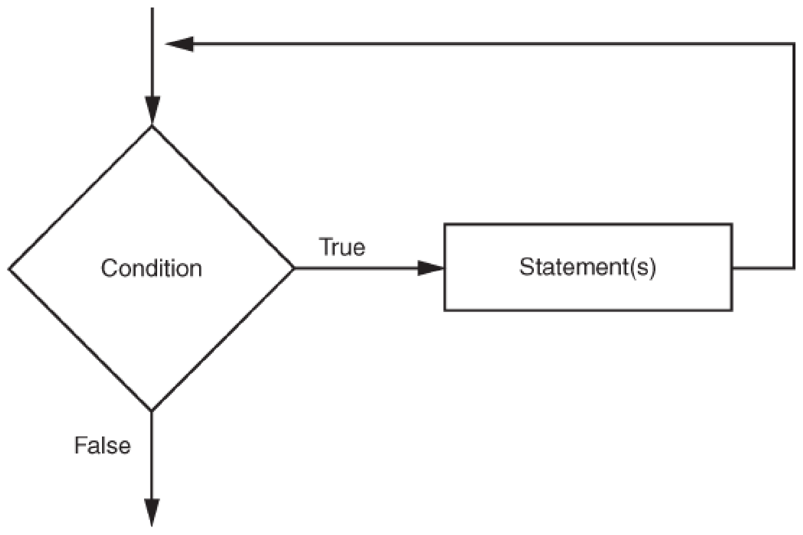

Figure 4-1 The logic of a `while` loop

The general format of the while loop in Python:

```
while condition:
    statement
    statement
   etc.
```

When the while loop executes, the *condition* is tested. If the *condition* is true, the statements that appear in the block following the while clause are executed, and then the loop starts over. If the *condition* is false, the program exits the loop.

The while loop is known as **a *pretest* loop**, which means it tests its condition *before* performing an iteration. Because the test is done at the beginning of the loop, you usually have to perform some steps prior to the loop **to make sure that the loop executes at least once.**

In all but rare cases, loops must contain within themselves a way **to terminate**. This means that something inside the loop must eventually make the test condition false.

If a loop does not have a way of stopping, it is called an **infinite loop**. An *infinite loop* continues to repeat until the program is interrupted.

Sentinels

A sentinel is a special value that marks the end of a sequence of values or items.

When a program reads the sentinel value, it knows it has reached the end of the sequence, so the loop terminates.

A sentinel value must be distinctive enough that it will not be mistaken as a regular value in the sequence.

Input Validation Loops

Input validation is the process of inspecting data that has been input to a program, to make sure it is valid before it is used in a computation. Input validation is commonly done with a loop that iterates as long as an input variable references bad data.

If a user provides bad data as input to a program, the program will process that bad data and, as a result, will produce bad data as output.

Programs should be designed in such a way that bad input is never accepted. When input is given to a program, it should be inspected before it is processed. If the input is invalid, the program should discard it and prompt the user to enter the correct data. This process is known as ***input validation.***

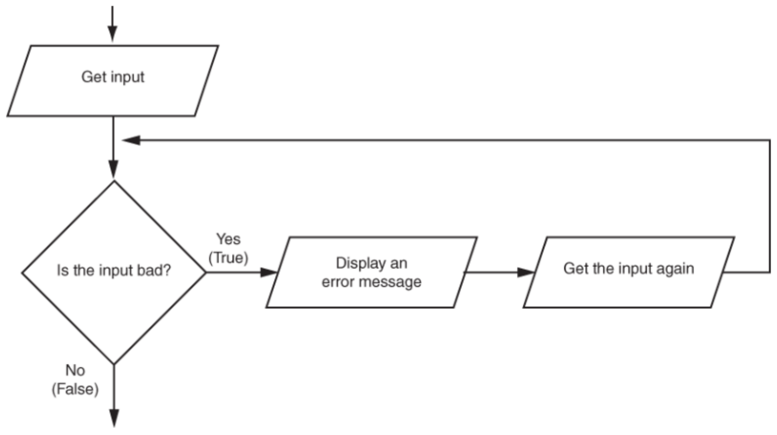

Figure 4-7 Logic containing an input validation loop

The flowchart reads input in two places: first just before the loop and then inside the loop. The first input operation—just before the loop—is called **a *priming read***, and its purpose is to get the first input value that will be tested by the validation loop. If that value is invalid, the loop will perform subsequent input operations.

Nested Loops

A loop that is inside another loop is called a nested loop.