In this module we will be covering repetition structures - *while and for* loops. We will also look at counting and summing. We will examine how we nest loops.

Repetition Structures

When we write computer programs we usually have the need to write a set of statements that we need to perform over and over again - in other words we need to repeat the same statements in a program many times. We do this through repetition structures. The two repetition structures we will be looking at are *while* and *for* loops. A loop is where you repeat statements over and over until you are through with what you want to accomplish.

There are two broad categories of loops - condition-controlled and count-controlled. A condition-controlled loop uses a true/false condition to control the number of times to do the loop. A count-controlled loop only performs the loop a specified number of times.

***While* Loop**

This type of loop causes a set of statements to repeat as long as a condition is true. So While a condition is true - execute the statements in the loop. When the condition is false - skip to the statement following the last statement in the While loop. The SYNTAX is:

```
while condition:

    statement

    statement

    etc.
```

The computer will check to see if the condition is true - if so, it will then execute the statements in the block sequentially. If the condition is false, the computer will skip the statements in the block and jump to the statement following the last statement in the while block. We call this a "pre-test" loop.

Problem - Infinite Loops

Every once in a while a loop NEVER stops executing - this is called an Infinite Loop condition. There must be something in the loop statements that will cause the loop to stop (cause the condition to be false). This is where we must have the correct logic in the loop to make this happen.

The *For* Loop

The *for* loop is a count-controlled loop. The SYNTAX is:

```
for variable in [value1, value2, etc.]:

    statement

    statement

    etc.
```

The first time through the loop, the *variable (which you make up the name)* is given the value1; the second time through the loop, the variable is given the value2; etc. until the last time through the loop the variable is given the valuex (last value).

For loop using the Range Function

PYTHON has a built in function called *range* that simplifies the count-controlled loop.

Counting - calculating a running total

A running total is a sum of numbers that accumulates with each time through the loop (iteration). The variable (we name it) that is used to keep the running total is called an Accumulator.

example: total = total + 1 (the "new" total value - on the left side - is equal to the old total value + 1)

Sums

A sum uses the old total, adds another variable to the old total to create a new total that is used in the next expression:

total = total + number

Sentinel

This is a special value that marks the end of a sequence of values.

Input Validation Loops

One way of inspecting the data being used as input to a program to be sure that it is valid data (valid values), is to create an input validation loop that will check each data value you input, before allowing that data to move on through your program statements.

This would mean that our large picture Logic is: Input - Check - Process - Output

Nested Loops

A loop that is put inside another loop. This logic can get VERY complex so be sure you understand the logic before using this. On page 153 you will find a flowchart that explains the nested loop logic.