

In this second lesson for Unit 4 you will learn about the **While...End** loop. We'll compare it to the **For...End** loop and even show why it is more powerful and versatile than the **For...End** loop.

**Objectives:**

- Learn the structure of the **While...End** loop.
- Compare it to the **For...End** loop.
- See how it is used to ensure valid input values.

**The While... End Loop**

The **While...End** loop will continue looping as long as its <condition> is True. It looks like this:

```

While <condition>
  <loop body>
End

```

**Notes**

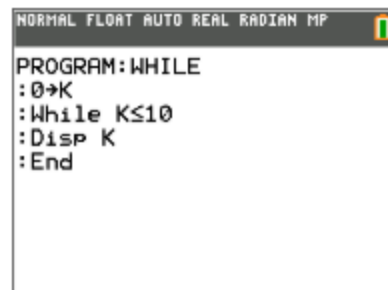
The <condition> is a logical expression such as **X>0**.

The <loop body> is any set of statements, including other loops and **If** structures. It is processed whenever the <condition> is True.

The keyword **End** is used to indicate the bottom of the <loop body>. At the **End** statement the program loops back to the **While** statement and tests the <condition> again.

'Initialize' the **While**'s condition: establish a value so that the condition is properly established as True or False. If the initial condition is False the loop is completely skipped. If the condition is True then the loop body is processed. The **0→K** at the top of this program sets the initial condition to False. Without it there's no way of knowing what will happen because any value could have been stored in the variable **K** before the program runs.

Somewhere in the <loop body> there should be a statement that will have an effect on the <condition> so that the loop will eventually end and statements after the loop will be processed. Usually this statement is near the bottom of the <loop body>. **K+1→K** ensures that eventually **K** will be greater than 10.



```

NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:WHILE
:0→K
:While K≤10
:Disp K
:End

```

*An 'infinite' loop! Why?*



```

NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:WHILE
:0→K
:While K≤10
:Disp K
:K+1→K
:End

```

*What will this program do?*



## TI-84 PLUS FAMILY

We'll write a section of a program (a 'code segment') that makes sure that the user enters a positive number, tells her when that entry is invalid and to enter another value in its place.

See if you can write this without peeking at the next page!

We start with an **Input** statement with a message to get a value from the user...

The loop body will be entered only when  $N$  is negative, otherwise the entire loop body is skipped.

Finally, use the **Input** statement again at the bottom of the <loop body> to request another value from the user and then **End** the <loop body>.

*This code segment will ask for a value from the user and make sure that the entry is a positive number. Below this code segment will be more statements to process the number entered.*

### STUDENT ACTIVITY

```
NORMAL FLOAT AUTO REAL RADIAN MP
PR9MYHLIU
ENTER A POSITIVE NUMBER-3
NOT POSITIVE
ENTER A POSITIVE NUMBER-8
NOT POSITIVE
ENTER A POSITIVE NUMBER0
NOT POSITIVE
ENTER A POSITIVE NUMBER3
Done
```

```
NORMAL FLOAT AUTO REAL RADI AN MP
PROGRAM:VALID
:Input "ENTER A POSITIVE N
UMBER",N
:
:
:
:
:
:
:
```

```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:VALID
:Input "ENTER A POSITIVE N
UMBER",N
:
:While N≤0
:Disp "NOT POSITIVE"
:
:
```

```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:VALID
:Input "ENTER A POSITIVE N
UMBER",N
:
:While N≤0
:Disp "NOT POSITIVE"
:
:Input "ENTER A POSITIVE N
UMBER",N
:End
```