Unit 4: Loops | **Skill Builder 1: For Loops**

In this lesson, you will learn about the loop concept in programming and examine the **For…EndFor** loop

Objectives:

- Describe the concept of looping in programming
- Write programs and functions using the **For…EndFor** structure
- Control the amount of output space (scrolling) taken using **DispAt**
- Using **getKey(1)** as a 'pause' command

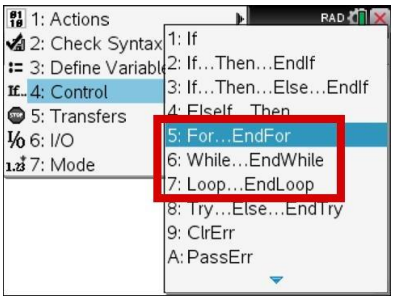
Teacher Tip: There are three *fundamental* loops in TI-Nspire™ CX TI-Basic: **For**, **While**, and **Loop**. A loop structure gives a program the ability to process a set of statements over and over, either iterating over a sequence of values (as in the **For** loop) or until a specific condition is met (or not) as in **While** and **Loop**. Unit 4 lessons introduce each one of these structures individually.

Programs may become complicated because it is sometimes necessary to mix all the control structures, (sequence, **If** statements and loops) into a program to satisfy more complex algorithms. Indenting and comments help to clarify programs.

This is what makes programming fun.

About Loops

The TI-Basic programming language has the ability to process a set of statements over and over. This repetition of statements is called **looping**.

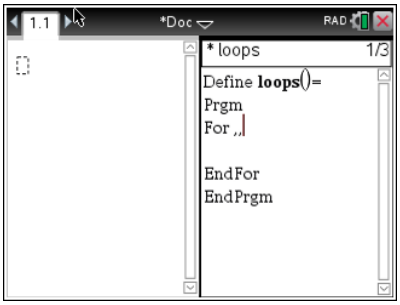


The three loop structures you will learn in this Unit are each accessed by selecting **menu > Control** from the Program Editor menu. (See **For**, **While**, and **Loop** to the right.)

The **While…** and **Loop…** structures will be explored in later lessons in this Unit. For more information about additional Control structures in this menu, see the Reference Guide at the TI Codes website.

For…EndFor

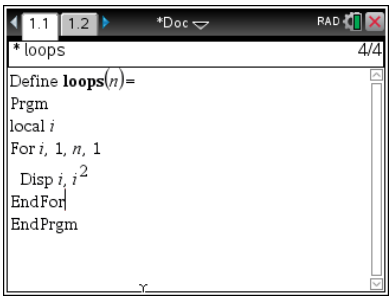
The **For…EndFor** loop is used to process an arithmetic sequence of values. This process is known as 'iteration.'



Selecting the **For…EndFor** statement from the control menu gives you the necessary components for building the rest of the structure:

For , ,

EndFor



The commas after the word **For** indicate that you need to provide at least **three** components. An additional comma and a fourth component can be added for the increment value. (Note: If the increment value is 1, the fourth component can be omitted.)

Here's an example:

For *i*, 1, *n*, 1
Disp *i*, i^2
EndFor

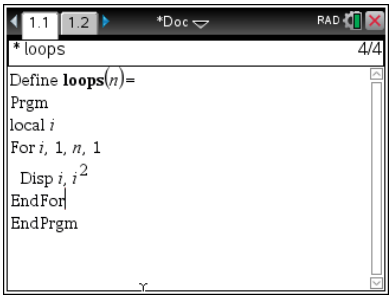
- *i* is the **loop control variable**: The first item must be a variable.
- 1 is the **starting value**: Each time the loop is processed it will take on values, or count, from the specified *start value* (1).
- *n* is the **ending value**: Each time the loop is processed it will take on values, or count, to the specified *end value* (n).
- 1 is the **increment value**: Each time the loop is processed it will take on values, or count, from the start to the end value by the specified *increment value* (1).

You can also read the **For** statement as: "For i going from 1 to n by 1s"

Any or all of the last three components can be numbers, variables, or expressions.

Write the Program

Enter the code into a program using an argument *(n)*, and run it.



Teacher Tips:

The fourth component can be omitted (the default increment is 1):

For *i*, 1, n uses a default value of 1 as the increment

The increment value can be any number or numeric variable or expression, including negative numbers:

For *i*, 5, -5, -1 starts at 5 and goes down to -5

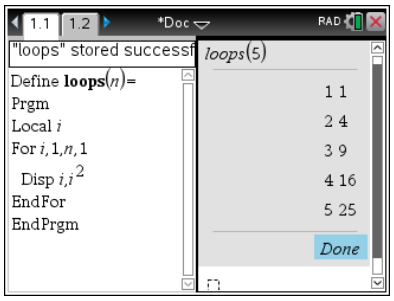
The value of the loop control variable does not have to 'hit' the end value. When the value of i exceeds the end value, the loop will terminate:

For *i*, 1, 10, 6 will process the values 1 and 7 and then end

At the end of the loop, the loop control variable is larger than the end value.

Do not assign any values to the *loop control variable* inside the loop. This will disrupt the loop processing and cause undesirable and unpredictable results. Normally, the loop control variable is a **Local** variable.

Running the Program

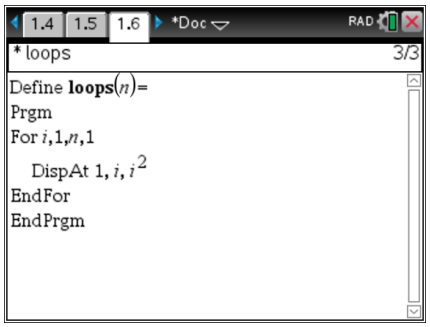


Running the program produces the results shown to the right (in a split screen to see the code and the result).

- The value for n (5) is an argument to the program.
- The loop control variable is a Local variable; it does not affect the rest of the problem.
- The loop begins with i=1 and Displays the values of i and i2.
- After the **Disp** statement, the **EndFor** statement passes control back to the **For** statement where the increment (1) is added to i.
- If the resulting value is less than or equal to the end value, the loop is processed again with the new value for i.
- This process is repeated until the end value is surpassed.

After the loop ends, *i* will be <u>larger</u> than the end value. Add another **Disp** *i* statement after the **EndFor** statement to see for yourself.

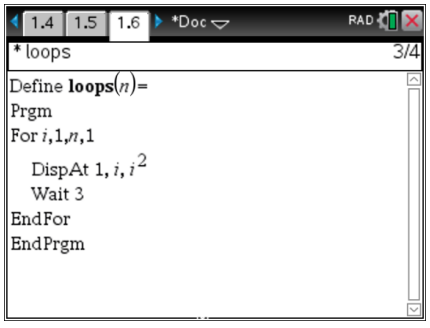
Large Output and DispAt



When you use a 'large' argument in this program, such as **loops**(10) or **loops(**100), the output is a long list and is challenging to inspect by scrolling.

You can use **DispAt 1, *i*, *i*^{2}** to place all output information on the same line of the screen. Try it.

DispAt is located in **menu > I/O**.



The output is displayed too fast to observe each pair of values. There are two ways to fix this problem.

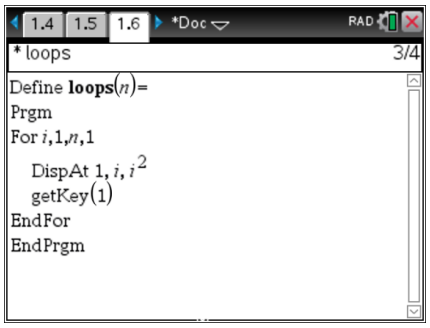
One Fix for DispAt

There are two ways to control the speed of the output. The first is:

Wait <seconds>

Wait is located in **menu > Control**. (Since **Wait** is the last option, use the up arrow.*)*

Adding **Wait 3** after **DispAt** tells the program to wait or pause for three seconds after each new set of values is displayed.



Another Fix for DispAt

The second method for controlling the speed of the output is:

getKey(1)

getKey( ) is located in **menu > I/O**.

Adding **getKey(1)** (*you have to type the number 1*) after **DispAt** pauses the program until a key is pressed.

Consider using a message such as **DispAt 2, "Press any key to continue"** before the **getKey(1)** function. This gives the user greater control than **Wait**.

Teacher Tip: getKey(0) is the other version of **getKey( )**. It looks for a keypress but does not pause the program. This is a useful tool for interactive programs that need to process code *and* look for a keypress. It comes in handy when programming the TI-Innovator™ Hub and the TI-Innovator™ Rover.
getKey(0) is like **getKey** on the TI-84 Plus.
getKey(1) is similar to the **Pause** statement on the TI-84 Plus.
getKey( ) is actually much more intricate. It *returns* the string value of the key pressed.
In a Calculator app, type **getKey(1)**, press **enter**, and then press a key to see the result. This value can then be used by a program to handle special keys in special ways.
For more information about **getKey( )**, see the latest version of the TI-Nspire Reference Guide.