

## Loops

A loop is a group of statements that is repeated. Visual Basic has three types of loops: **For** loops, **While** loops, and **Do Until** loops.

### For Loops

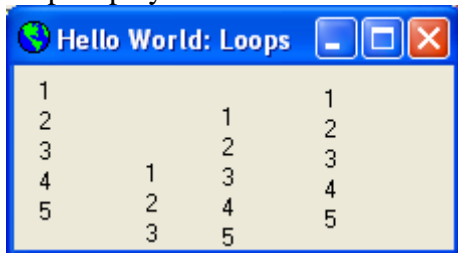
A **FOR** loop allows you to specify a starting value for a variable, an end value, and the amount you want to change, or increment. The first format does not specify an increment: it will increment by +1:

```
For <variable> = <start value> to <end value>
    <statements>
Next <variable>
```

The examples below can be entered into a new project with no controls, no property changes. In order to draw on the form we create an instance of the Graphics class and get the forms Graphics to draw on.

```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
    System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
    Dim G As Graphics
    G = Me.CreateGraphics
    Dim Num As Integer
    Dim Y As Integer = e.Y
    For Num = 1 To 5
        G.DrawString(Num, Me.Font, Brushes.Blue, e.X, Y)
        Y = Y + 15
    Next Num
End Sub 'Form1_MouseDown
```

Each time you click the form the loop is executed to print the values 1, 2, 3, 4, and 5. Y is given an initial value of the position clicked, then incremented by 15 each time through the loop display the numbers in a column. What happens if you don't increment Y?



**Experiment:** Set the font property of the form before you run the program.

The second format of a For loop specifies an increment:

```
For <variable> = <start value> to <end value> Step <increment>
    <statements>
Next <variable>
```

Change the for loop statement as shown below, leave the rest the same.

```
For Num = 5 To 25 Step 5
```

What do you expect to print?

When the **For** loop is executed the variable is assigned the starting value. Then the value is compared to the end value. If the increment is a positive number and the value is less than or equal to the end value, the statements in the body of the loop are executed, otherwise the next line after the loop is executed. If the increment is a negative number, then the value must be greater than or equal to the end value. After the statements in the body of the loop are executed, the next statement adds the increment to the variable and goes back to the top to check the value. The statements in a **For** loop may execute 0 or more times. If the starting value is past the end value it does not execute at all.

The example below does not print anything because x has a starting value that is greater than 1 and the increment is +1. (If you don't specify step size, the default is +1):

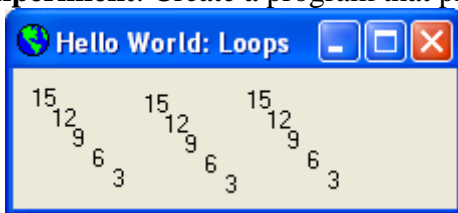
```
For Num = 10 To 1
```

You must specify a negative step size if the starting value is greater than the end value:

```
For Num = 10 To 1 Step -1
```

**Experiment:** Make the program print 10 9 8 7 6 5 4 3 2 1 Blast Off!  
Instead of incrementing Y, increment a variable X to display the values horizontally.

**Experiment:** Create a program that prints the values as shown below:



## For Loop Fun

These programs use loops to create interesting designs. No controls are placed on the form and no properties are changed. Start a new project or delete the code from the previous exercise.

Look at each example, run it, then make some slight modifications. Experiment!

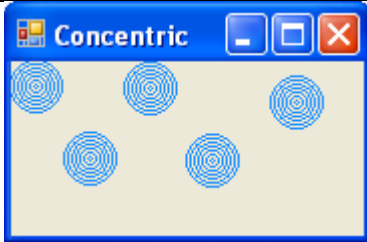
### Sea Shells:

```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
Dim G As Graphics
G = Me.CreateGraphics 'So that we can draw on the form
Dim Num As Integer
For Num = 1 To 25 Step 5
G.DrawEllipse(Pens.Crimson, e.X, e.Y, Num, Num)
Next Num
End Sub 'Form1_MouseDown
```

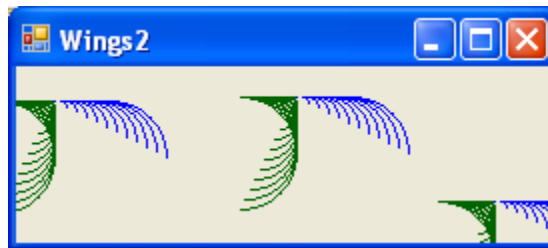
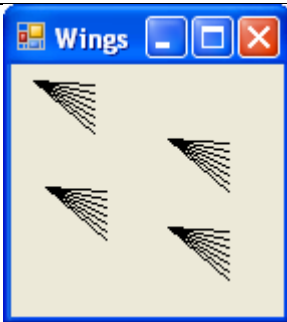


**Concentric:**

```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
    System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
    Dim G As Graphics
    G = Me.CreateGraphics 'So that we can draw on the form
    Dim Num As Integer
    Dim Hlf As Integer
    For Num = 2 To 26 Step 4
        Hlf = Num / 2
        G.DrawEllipse(Pens.DodgerBlue, e.X - Hlf, e.Y - Hlf, Num, Num)
    Next Num
End Sub 'Form1_MouseDown
```



```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
    System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
    Dim G As Graphics = Me.CreateGraphics 'So that we can draw on the form
    Dim Num As Integer
    For Num = 2 To 26 Step 4
        G.DrawLine(Pens.Black, e.X, e.Y, e.X + 30, e.Y + Num)
    Next Num
End Sub 'Form1_MouseDown
```



```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
    System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
    Dim G As Graphics = Me.CreateGraphics 'So that we can draw on the form
    Dim Num As Integer
    For Num = 1 To 60 Step 5
        G.DrawArc(Pens.Blue, e.X, e.Y, Num, Num, 270, 90)
        G.DrawArc(Pens.DarkGreen, e.X - Num, e.Y, Num, Num, 270, 180)
    Next Num
End Sub 'Form1_MouseDown
```

## While Loops

The while loop uses a Boolean expression to determine when to end the loop:  
The format of a while loop is:

```
While <Boolean expression>
    <statements>
End While
```

The examples below compares a For and a While Loop. Both examples generate the values 1, 2, 3, 4, 5.

For Loop:

```
For N = 1 To 5
    ...
Next N
```

While Loop:

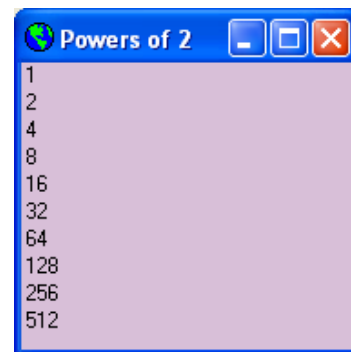
```
N = 1 'Initial value in line 1 of For loop
While N<=5 'Test is in line 1 of For loop
    ...
    N = N + 1 'Increment is in line 3 in For loop
End While
```

(One advantage of the while loop is that the Boolean expression can include AND and OR.) The example below can be entered into a new project with no controls, no property changes. This program generates powers of 2.

```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
    System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
    Dim G As Graphics = Me.CreateGraphics 'So that we can draw on the form
    Dim Num As Integer
    Dim Y As Integer = 0
    Num = 1
    While Num < 1000
        G.DrawString(Num, Me.Font, Brushes.Black, 0, Y)
        Num = Num * 2
        Y = Y + 15
    End While
End Sub 'Form1_MouseDown
```

Notice that the test value of 1000 is never hit exactly. When the while loop is executed, the Boolean expression is tested: if it is true the body of the loop is executed. When it gets to the wend statement it goes back to the top and tests again. The loop is repeated until the test fails, then the next statement after the wend is executed. A While loop can execute zero or more times.

After the value 512 is printed, num is assigned a new value of 1024. When the control goes to the while statement, the expression Num<1000 is false and the loop ends. When the loop ends the next statement after the loop is executed.



**Note:** If the Boolean expression is true, all of the statements in the loop execute, even if the expression becomes false during the loop. Look at the difference in the output when the position of the increment is changed: Everything is exactly the same except for the increment:

```
Num = 1
While Num < 1000
  Num = Num * 2
  Me.CreateGraphics.DrawString(Num,...
  Y = Y + 15
End While
```

A common mistake with loops is to omit the first value and include an extra value past the *intended* end point.

### Endless Loops

Occasionally, you create a loop that will never end. In the example above, if you leave out the statement `Num=Num*2`, `Num` starts out as 1 and stays 1. The statement `Num<1000` will always be true. Depending on what else the loop is doing, the system may eventually give you an error message, or you will realize the program is “frozen” or “hung-up.” If you are working in VB, you can click the black square on the toolbar to end the program.

### Do Loop

The Do Loop is similar to the While loop except that it does the test at the end of the loop. A do loop executes one or more times. The format of a Do Until loop is:

**Do**

<statements>

**Loop Until** <Boolean expression>

```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
  System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
  Dim G As Graphics = Me.CreateGraphics 'So that we can draw on the form
  Dim Num As Integer
  Dim Y As Integer = 0
  Num = 1
  Num = 10
  Do
    G.DrawString(Num, Me.Font, Brushes.Red, 0, Y)
    Y = Y + 15
    Num = Num - 1
  Loop Until Num = 0
  G.DrawString("Blast Off!", Me.Font, Brushes.Fuchsia, 0,
  Y)
End Sub 'Form1_MouseDown
```

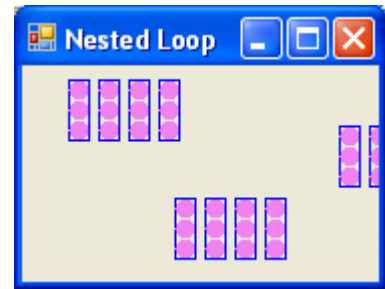


## Nested Loops

A *nested* loop is a loop inside a loop. The loops do not have to be the same type. There can be a **For** loop inside a **While** loop; a **For** loop inside a **For** loop, etc. The inner loop is executed in its entirety for each execution of the outer loop.

### Nested Loop Programs

The illustration shows a design created using nested loops. The outer loop draws 4 rectangles. The inner loop draws 3 circles inside each rectangle. The statement to draw the rectangle (in the outer loop) is executed 4 times, but the statement to draw the circle (in the inner loop) is executed 12 times!



```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
Dim G As Graphics = Me.CreateGraphics 'So that we can draw on the form
Dim Rec As Integer
Dim Circle As Integer
For Rec = 1 To 4
    G.DrawRectangle(Pens.Blue, e.X + Rec * 15, e.Y, 10, 30)
    For Circle = 0 To 2
        G.FillEllipse(Brushes.Violet, e.X + Rec * 15, e.Y + Circle * 10, 10, 10)
    Next Circle
Next Rec
End Sub 'Form1_MouseDown
```

**Experiment:** Create a similar design with squares inside circles.

**Experiment:** Try to figure out what the following code does before you run it:

```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
Dim G As Graphics = Me.CreateGraphics 'So that we can draw on the form
Dim Y As Integer
Dim Mth As Integer
Dim D As Date
For Mth = 1 To 12
    D = Mth & "/1/2002"
    G.DrawString(Format(D, "MMMM"), Me.Font, Brushes.Aqua, 0, Y)
    Y = Y + 15
Next Mth
End Sub 'Form1_MouseDown
```