Chapter 12: Looping

Section 12.1: For...Next

`For`...`Next` loop is used for repeating the same action for a finite number of times. The statements inside the following loop will be executed 11 times. The first time, `i` will have the value 0, the second time it will have the value 1, the last time it will have the value 10.

```
For i As Integer = 0 To 10
    'Execute the action
    Console.Writeline(i.ToString)
Next
```

Any integer expression can be used to parameterize the loop. It is permitted, but not required, for the control variable (in this case `i`) to also be stated after the `Next`. It is permitted for the control variable to be declared in advance, rather than within the `For` statement.

```
Dim StartIndex As Integer = 3
Dim EndIndex As Integer = 7
Dim i As Integer

For i = StartIndex To EndIndex - 1
    'Execute the action
    Console.Writeline(i.ToString)
Next i
```

Being able to define the Start and End integers allows loops to be created that directly reference other objects, such as:

```
For i = 0 to DataGridView1.Rows.Count - 1
    Console.Writeline(DataGridView1.Rows(i).Cells(0).Value.ToString)
Next
```

This would then loop through every row in `DataGridView1` and perform the action of writing the value of Column 1 to the Console. *(The -1 is because the first row of the counted rows would be 1, not 0)*

It is also possible to define how the control variable must increment.

```
For i As Integer = 1 To 10 Step 2
    Console.Writeline(i.ToString)
Next
```

This outputs:

> 1 3 5 7 9

It is also possible to decrement the control variable (count down).

```
For i As Integer = 10 To 1 Step -1
    Console.Writeline(i.ToString)
Next
```

This outputs:

> 10 9 8 7 6 5 4 3 2 1

You should not attempt to use (read or update) the control variable outside the loop.

Section 12.2: For Each...Next loop for looping through collection of items

You can use a `For Each...Next` loop to iterate through any `IEnumerable` type. This includes arrays, lists, and anything else that may be of type IEnumerable or returns an IEnumerable.

An example of looping through a DataTable's Rows property would look like this:

```
For Each row As DataRow In DataTable1.Rows
    'Each time this loops, row will be the next item out of Rows
    'Here we print the first column's value from the row variable.
    Debug.Print(Row.Item(0))
Next
```

An important thing to note is that the collection must not be modified while in a `For Each` loop. Doing so will cause a `System.InvalidOperationException` with the message:

> Collection was modified; enumeration operation may not execute.

Section 12.3: Short Circuiting

Any loop may be terminated or continued early at any point by using the `Exit` or `Continue` statements.

Exiting

You can stop any loop by exiting early. To do this, you can use the keyword `Exit` along with the name of the loop.

Loop	Exit Statement
For	`Exit For`
For Each	`Exit For`
Do While	`Exit Do`
While	`Exit While`

Exiting a loop early is a great way to boost performance by only looping the necessary number of times to satisfy the application's needs. Below is example where the loop will exit once it finds the number 2.

```
Dim Numbers As Integer() = {1,2,3,4,5}
Dim SoughtValue As Integer = 2
Dim SoughtIndex
For Each i In Numbers
    If i = 2 Then
        SoughtIndex = i
        Exit For
    End If
Next
Debug.Print(SoughtIndex)
```

Continuing

Along with exiting early, you can also decide that you need to just move on to the next loop iteration. This is easily done by using the `Continue` statement. Just like `Exit`, it is proceeded by the loop name.

Loop	Continue Statement
For	`Continue For`
For Each	`Continue For`
Do While	`Continue Do`
While	`Continue While`

Here's an example of preventing even numbers from being added to the sum.

```
Dim Numbers As Integer() = {1,2,3,4,5}
Dim SumOdd As Integer = 0
For Each i In Numbers
    If Numbers(i) \ 2 = 0 Then Continue For
    SumOdd += Numbers(i)
Next
```

Usage Advice

There are two alternative techniques that can be used instead of using `Exit` or `Continue`.

You can declare a new Boolean variable, initializing it to one value and conditionally setting it to the other value inside the loop; you then use a conditional statement (e.g. `If`) based on that variable to avoid execution of the statements inside the loop in subsequent iterations.

```
Dim Found As Boolean = False
Dim FoundIndex As Integer
For i As Integer = 0 To N - 1
    If Not Found AndAlso A(i) = SoughtValue Then
        FoundIndex = i
        Found = True
    End If
Next
```

One of the objections to this technique is that it may be inefficient. For example, if in the above example `N` is 1000000 and the first element of the array `A` is equal to `SoughtValue`, the loop will iterate a further 999999 times without doing anything useful. However, this technique can have the advantage of greater clarity in some cases.

You can use the `GoTo` statement to jump out of the loop. Note that you cannot use `GoTo` to jump *into* a loop.

```
    Dim FoundIndex As Integer
    For i As Integer = 0 To N - 1
        If A(i) = SoughtValue Then
            FoundIndex = i
            GoTo Found
        End If
    Next
    Debug.Print("Not found")
Found:
    Debug.Print(FoundIndex)
```

This technique can sometimes be the neatest way to jump out of the loop and avoid one or more statements that are executed just after the natural end of the loop.

You should consider all of the alternatives, and use whichever one best fits your requirements, considering such

things as efficiency, speed of writing the code, and readability (thus maintainability).

Do not be put off using `GoTo` on those occasions when it is the best alternative.

Section 12.4: While loop to iterate while some condition is true

A `While` loop starts by evaluating a condition. If it is true, the body of the loop is executed. After the body of the loop is executed, the `While` condition is evaluated again to determine whether to re-execute the body.

```
Dim iteration As Integer = 1
While iteration <= 10
  Console.Writeline(iteration.ToString() & " ")

  iteration += 1
End While
```

This outputs:

> 1 2 3 4 5 6 7 8 9 10

Warning: A `While` loop can lead to an *infinite loop*. Consider what would happen if the line of code that increments `iteration` were removed. In such a case the condition would never be True and the loop would continue indefinitely.

Section 12.5: Nested Loop

> A nested loop is a loop within a loop, an inner loop within the body of an outer one. How this works is that the first pass of the outer loop triggers the inner loop, which executes to completion. Then the second pass of the outer loop triggers the inner loop again. This repeats until the outer loop finishes. a break within either the inner or outer loop would interrupt this process.

The Structure of a For Next nested loop is :

```
For counter1=startNumber to endNumber (Step increment)

    For counter2=startNumber to endNumber (Step increment)

        One or more VB statements

    Next counter2

Next  counter1
```

Example :

```
    For  firstCounter = 1 to 5

      Print “First Loop of ” + firstCounter

    For   secondCounter= 1 to 4

      Print “Second Loop of ” + secondCounter

    Next secondCounter
```

```
  Next firstCounter
```

Section 12.6: Do...Loop

Use `Do...Loop` to repeat a block of statements `While` or `Until` a condition is true, checking the condition either at the beginning or at the end of the loop.

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
Loop While x < 10
```

or

```
Dim x As Integer = 0
Do While x < 10
    Console.Write(x & " ")
    x += 1
Loop
```

> 0 1 2 3 4 5 6 7 8 9

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
Loop Until x = 10
```

or

```
Dim x As Integer = 0
Do Until x = 10
    Console.Write(x & " ")
    x += 1
Loop
```

> 0 1 2 3 4 5 6 7 8 9

`Continue Do` can be used to skip to the next iteration of the loop:

```
Dim x As Integer = 0
Do While x < 10
    x += 1
    If x Mod 2 = 0 Then
        Continue Do
    End If
    Console.Write(x & " ")
Loop
```

> 1 3 5 7 9

You can terminate the loop with `Exit Do` - note that in this example, the lack of any condition would otherwise cause an infinite loop:

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
    If x = 10 Then
        Exit Do
    End If
Loop
```

> 0 1 2 3 4 5 6 7 8 9