

## LESSON 26

### INTRODUCTION TO VBSCRIPT

A scripting language is a scaled down version of a programming language that is used within another application called a host application. A key component of Microsoft ActiveX technology is called ActiveX Scripting. ActiveX Scripting provides the support for scripting languages such as VBScript to be integrated into host application such as Web browsers. For example, ActiveX Scripting provides support for JavaScript and VBScript, which can be used with the host application, Internet Explorer. ActiveX Scripting provides the opportunity for vendors to create other scripting languages to use with ActiveX technology. Although VBScript is designed to work with ActiveX controls and other objects embedded in Web pages, it can also be used as a general scripting language in other host application besides Web browsers. For example, VBScript can be the scripting language for a Web server.

VBScript is an adaptation and extension of Microsoft's popular family of Visual Basic languages that include Visual Basic for Applications and Visual Basic, version 5.0. Visual Basic for Applications is the scripting language for the Microsoft Office family of products, while Visual Basic 5.0 is a superset of Visual Basic for Applications and comes in three editions: the Standard Edition for students or hobbyists, the Professional Edition for individual developers, and the Enterprise Edition for teams of developers.

#### VBScript Basics

VBScript is easy to learn if you already know Visual Basic for Applications or Visual Basic 5.0. VBScript enables Visual Basic programmers to leverage their knowledge of the language in developing highly interactive Web pages. This chapter covers VBScript, version 1.0. To use VBScript version 1.0, you need Internet Explorer, version 3.0 and later or Netscape Navigator 3.0 with the Ncompass ScriptActive plug-in. At the time of this writing, Netscape Communications' browser, Netscape Communicator, does not support VBScript or ActiveX controls. VBScript, version 2.0, is only supported by Internet Explorer 4.0, which is only available in beta at the time of this writing.

A computer program is a set of instructions that a computer can execute. In an HTML document, the computer program written in VBScript goes inside the pair of `<SCRIPT>` and `</SCRIPT>` tags. For example:

```
<SCRIPT LANGUAGE="VBScript">
<!--
Total Results = NewResult + OldResults
-->
</SCRIPT>
```

The `<SCRIPT>` tag precedes and follows the VBScript code. In the first `<SCRIPT>` tag, you must use the LANGUAGE attribute to specify which scripting language you will use. This is because ActiveX Scripting technology is capable of supporting more than one kind of scripting language. The name of the

scripting language that you assign to the LANGUAGE attribute should be quotes.

The `<SCRIPT>` tag is followed by `<!--` and the `</SCRIPT>` tag is preceded by `-->`. The `<!--` and `-->` combination is used to hide the code from browsers that do not support scripting so that the code is not displayed. The best place for scripting code that is not connected to a particular control is in `<HEAD>` section of HTML document. For example:

```
<HTML>
<HEAD>
<TITLE>Compute Total Results</TITLE>
<SCRIPT LANGUAGE="VBScript">
<!--
Total Results = NewResult + OldResults
-->
</SCRIPT>
<BODY>
More codes go here
</BODY>
</HTML>
```

As a useful convention, put all of the procedures and functions that you will use in the HTML document in the `<HEAD>` section. This practice insures that all of the procedures and functions will be loaded before the user has an opportunity to activate a control that calls them. If the user activates a control before a function or procedure that it calls is loaded, the control will malfunction.

#### VBScript Data Types

A data type defines the kinds of values that a variable can contain. VBScript has only one data type called a Variant, which can contain either numeric or string information. A Variant acts like a number when you assign a numeric value to it and a string when assign a string value to it. Variants can also contain subtypes of numeric and string data. Refer the following table for example, a subtype of the numeric type is the date or time subtype when the numeric data is expressed as a date or a time. You can also have floating point numbers and other subtypes of numeric data. The variant behaves in the way appropriate to the kind of data contained in the variable.

Subtype	Content
Empty	Either 0 or ""
Null	No valid data
Boolean	True(-1) or False(0)
Byte	Integer from 0 to 35
Integer	Integer from -32,768 to 32,767
Long	Real from -2,147,483,648 to 2,147,483,647

Single ----- Real number from -3.402823E38 to -1.401298E-45 for negative values  
Real number from 1.401298E-45 to 3.402823E38 for positive values

Double ----- Real number from -1.79769313486232E308 to -4.94065645841247E-324 for negative values  
Real number from 4.94065645841247E-324 to 1.79769313486232E308 for positive values

Date(Time) ----- Date between January 1, 100 to December 31, 9999

String ----- Up to about 2 million characters

Object ----- OLE Automation Object

Error ----- Error number

The VarType function can be used to determine the data type of the information stored in a Variant. In addition, there are conversion functions available to convert from one subtype to another.

### Vbscript Data Types - Variables

A variable is the name of a storage location in the memory of the computer that contains data that can be modified during the execution of a computer program. For example, you could have a variable called StudentAge in which you could store the age of a student for use in computing the average age of students in a school.

**Notice:** All variables in VBScript are of the Variant data type. You can declare variables explicitly or implicitly. The following is an example of a variable declared explicitly:

```
<SCRIPT LANGUAGE="VBScript">
<!--
```

```
Dim StudentAge
```

```
-->
</SCRIPT>
```

You can explicitly declare several variables at the same time by separating each variable by a comma. For example:

```
<SCRIPT LANGUAGE="VBScript">
<!--
```

```
Dim StudentAge, StudentName, StudentNumber
```

```
-->
</SCRIPT>
```

To declare a variable implicitly, you simply name it somewhere in your script without using Dim. For example:

```
<SCRIPT LANGUAGE="VBScript">
<!--
```

```
StudentAge = 18
```

```
-->
</SCRIPT>
```

**Notice:** You should avoid declaring variables implicitly in your program. You could introduce a bug by unintentionally misspelling the same variable name in different places in your program.

It is better programming practice to use the Option Explicit statement that requires the explicit declaration of all variables.

The Option Explicit statement must be the first statement after the <SCRIPT> tag. For example:

```
<SCRIPT LANGUAGE="VBScript">
<!--
```

```
Option Explicit
```

```
-->
</SCRIPT>
```

You have wide latitude in naming your variables as long as the name meets the following restrictions: the variable name must begin with a letter of the alphabet, it cannot contain a period, it cannot exceed 255 characters, and it must be unique within the scope in which it is declared. For example, you cannot declare two variables with the name StudentAge within the same procedure or function.

**Notice:** Subprocedures and functions are discussed in more detail later in this tutorial. Basically, subprocedures and functions allow you to divide your script into manageable modules or pieces of code in which each module performs a specific task.

Two important concepts involving variables are scope and lifetime. Scope refers to the extent to which a given variable can be referenced in a VBScript program. For example, if a variable is declared within a subprocedure or function, then this variable can be referenced only within that subprocedure and function. Such a variable cannot be referenced outside of the particular subprocedure or function in which it is declared. A variable that is declared locally within a subprocedure or function is called a local variable. For example:

```
<SCRIPT LANGUAGE="VBScript">
<!--
```

```
Option Explicit
```

```
Sub cmdButton_Onclick()
```

```
Dim StudentAge
```

```
' The rest of your code follows here.
```

```
End Sub
```

```
-->
</SCRIPT>
```

In this example, the variable StudentAge is declared locally within the subprocedure cmdButton\_OnClick denoted by the Sub prefix. It cannot be referenced outside of this subprocedure. The apostrophe (') is used to create a comment. If a variable is declared within a set of <SCRIPT> tags but outside of any subprocedure or function, then the variable is global in scope and can be accessed from anywhere within a set of <SCRIPT> tags on the same Web page. For example

```
<SCRIPT LANGUAGE="VBScript">
<!--
```

```
Option Explicit
```

```
Dim StudentNumber
```

```
Sub cmdButton_OnClick()
```

```
' The rest of your code follows here.
```

```
End Sub
```

```
-->
</SCRIPT>
```

In this example, the variable `StudentNumber` is global in scope and can be referenced from anywhere within the program. In other words, the variable can be referenced from anywhere within the `<SCRIPT>` tags in which the variable is declared. So any subprocedure or function within a set of `<SCRIPT>` tags can reference a global variable. However, in the preceding example, the variable `StudentAge` is local in scope in that it can be referenced only within the procedure `cmdButton_OnClick()`

**Notice:** If a global variable and a local variable have the same name, then the local variable will take precedence over the global variable within the subprocedure or function in which the local variable occurs.

The lifetime of a variable is the length of time that a variable exists. The lifetime of a global variable starts from the time it is declared and ends when the `</SCRIPT>` tag is reached. The lifetime of a local variable starts from the time it is declared within a subprocedure or function and ends when the subprocedure or function ends. Since a local variable's lifetime is only as long as the time the procedure or function in which it occurs is executing, there can be several local variables with the same name. This is because subprocedures or functions are executed one at a time, so two local variables with the same name will not exist at the same time. You can assign a value to a variable by using the equal sign (`=`). In an assignment statement the variable name goes on the left side of the expression, the equal sign goes to the right of variable name and the value assigned to the variable name goes to the right of the equal sign. For example:

```
variable_name = value
```

Here is an example of an assignment statement in VBScript code:

```
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
Dim StudentAge
StudentAge = 19
-->
</SCRIPT>
```

A variable can contain single values or multiple values. If a variable contains a single value, it is called a scalar variable. If the variable contains multiple values, it is called an array. An array variable, like a scalar variable, is also declared using `Dim`. However, in declaring an array variable you must also specify the number of elements and dimensions in the array. This is done by using a parentheses after the array name. For example:

```
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
Dim Students(9)
-->
</SCRIPT>
```

In this example, the name of the array is called `Students`, which has only one dimension because there is only one number between the parentheses. Since the number of elements is 9, there are 10 elements in the `Students` array. This is because the count for the number of elements in any array starts with the number 0 instead of the number 1. So 0 through 9 elements is

a total of 10 rather than 9 elements. Since the number of elements in the array is specified, this is called a fixed size array. You can assign values to the elements of an array by using the assignment operator (`=`) and the index or number of the element in the array. For example, in the `Students(9)` array, you can assign a student name to each element of the array, as shown here:

```
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
Students(0) = "Mary"
Students(1) = "Elizabeth"
Students(2) = "Joe"
. . . . .
Students(9) = "Susan"
-->
</SCRIPT>
```

You can also retrieve the value of an element in an array by using the assignment operator (`=`) and the number or index of the element in the array. For example:

```
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
StudentName = Students(6)
-->
</SCRIPT>
```

In this example, the name of the student that is the seventh element in the array will be assigned to the variable `StudentName`. An array can have more than one dimension. If an array has more than one dimension, then the dimensions between the parentheses are separated by commas. For example: `Students(2, 2)`

In this example, the `Students` array has two dimensions. In effect, a multidimensional array is an array of arrays. This two-dimensional array is an array of three elements, each of which is itself an array of three elements. In a two-dimensional array, the first dimension represents the rows and the second dimension represents the columns, as shown here:

	<b>Column0</b>	<b>Column1</b>	<b>Column2</b>
<b>Row0</b>	Students(0,0)	Students(0,1)	Students(0,2)
<b>Row1</b>	Students(1,0)	Students(1,1)	Students(1,2)
<b>Row2</b>	Students(2,0)	Students(2,1)	Students(2,2)

The zero column of the array could be used to store the name of the student in the form of a string, the one column could be used to store the age of the student in the form of a number, and the two column could be used to store the identification number of the student in the form of a string by simply placing quotation marks around the identification number. For example:

```
StudentNumber = "26939"
```

In this example, the identification number of the student is stored as a string. If there were no quotation marks around the number 26939, then it would be stored as a number in the variable `StudentNumber`. You can have up to 60 dimensions in an array.

**Notice:** Although different columns can contain different datatypes, you should make sure that each column contains only one datatype. In other words, do not put a string and a number in the same column.

The arrays we have described so far are fixed size arrays. In a fixed size array, the size of the dimension or dimensions are explicitly stated. For example, Students(9) is a fixed size array whose size is explicitly stated as 9; that is, it contains 10 elements, Students(0) through Students(9). You can also have an array whose size changes dynamically as your VBScript code is executing. This kind of array is called a dynamic array. A dynamic array is declared by using a Dim statement or an ReDim statement. However, with a dynamic array, the size is not explicitly stated between the parentheses. For example:

```
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
Dim FirstYearStudents()
ReDim SecondYearStudents()
-->
</SCRIPT>
```

The advantage of a dynamic array is that if you do not know how many elements will be in the array ahead of time, you can declare the array without specifying its size. However, at sometime in your program you will have to specify the size of the array. To specify the size of a previously declared dynamic array, you use ReDim. For example:

```
ReDim FirstYearStudents(9)
```

This example sizes the dynamic array FirstYearStudents to a fixed size array containing 10 elements. So dynamic arrays enable the VBScript programmer to declare an array without specifying its size. This allows the end-user to provide information as the VBScript code is running and the re-sizing the array using ReDim based upon the user's input. A re-dimensioned array can be re-dimensioned multiple times. For example, you can use ReDim on the array FirstYearStudents(15) to transform it into any array of 16 elements:

```
ReDim FirstYearStudents(15)
```

However, this re-dimensioning of the array FirstYearStudents will erase the prior contents of the array. In other words, the contents of elements 0 through 9, resulting from the first use of ReDim, will be erased. To re-dimension the array while at the same time preserving its contents, you must use the keyword Preserve. For example:

```
<SCRIPT LANGUAGE="VBScript">
<!-- ReDim FirstYearStudents(9)
ReDim Preserve FirstYearStudents(15)
-->
</SCRIPT>
```

In this example, the 10 elements of the array sized with the first use of the ReDim statement will be preserved after the array FirstYearStudents is resized a second time.

**Notice: You can use ReDim as many times as you like in a program. Remember that if you use ReDim without the keyword Preserve, the prior contents of the array will be lost.**

If you use Preserve with ReDim, the prior contents of the array will be preserved as long as the re-dimensioned array is larger than the original array. For example, if you re-dimension the array FirstYearStudents as follows:

```
ReDim Preserve FirstYearStudents(5)
```

the contents of the first 6 elements of the array will be retained and the last 9 elements of the array will be lost because the size of this array is smaller than the size of the original array.

When you re-dimension arrays it is sometimes easy to lose track of the size of the array. It is important to know the size of the area you are working with so that you do not attempt to access an array element outside of the array. If your script attempts to access an array element outside of the array, your program will generate a Subscript out of range error message. You can use the UBound function to keep track of the size of an array. The UBound function takes the name of the arrays as an argument and returns the highest subscript for the array:

```
intArraySize = UBound(FirstYearStudents)
```

After the execution of this assignment statement, the variable intArraySize contains the highest subscript in the array. If the array has 10 elements, the intArraySize will contain the number 9, since 9 is the highest subscript in an array containing 10 elements. The UBound function is also useful in determining how many times a loop must repeat. You can use the UBound function to set the number of times the code in a loop will execute based upon the size of the array. You will learn more about loops termed for loops later in this tutorial. Here is an initial example of how to use UBound function to set the number of iterations of a for loop:

```
For Index = 1 to UBound(FirstYearStudents)
MsgBox "The current Student number is " & Index
Next
```

### Vbscript Data Types-Comments

A comment is text that you add to your program to explain what the program does and how it does it. The comment itself is not an executable part of the program. Comments in VBScript begin with an apostrophe ('), which indicates that every character from the apostrophe to the end of the line is a comment. However, the next line must also have an apostrophe to be a comment.

**Notice:** Apostrophes are VBScript comments while <!-- and --> are HTML comments.

You can place a comment in the middle of a line or on a line by itself:

```
<SCRIPT LANGUAGE="VBScript">
<!--
Option Explicit
```

```
Dim StudentNumber
Sub cmdButton_OnClick()
```

```
Dim Age ' This variable contains the student's age
' The rest of your code follows here
```

```
-->
</SCRIPT>
```

You can also present comments in a block. For example:

```

*****
' This subprocedure computes how much a student owes in
' tuition when the cmdButton is clicked by the user
*****

```

You create a comment block placing an apostrophe in front of each line of the comment. Comment blocks are useful when it is necessary to make extended comments on a script, subprocedure, or function.

## VBSRIPT OPERATORS

V BScript has arithmetic operators, comparison operators, concatenation operators, and logical operators. If there are several operators in an expression, they are evaluated in a specified order known as the order of operator precedence. You can override operator precedence by using parentheses. This forces the operators within the parentheses to be evaluated before the operators outside of the parentheses. The operators within the parentheses are evaluated according to the usual order of precedence. If there are a number of different kinds of operators within an expression, then arithmetic operators are evaluated first, comparison operators are evaluated second, and logical operators are evaluated last. The following table sets forth the order of precedence for arithmetic operators.

Operator	Symbol
Exponentiation	^
Negation	-
Multiplication	*
Division /	
Integer Division \	
Modulo Arithmetic	Mod
Addition	+
Subtraction	-
String Concatentation	&

Multiplication and Division are evaluated as they occur from left to right in the same expression. Addition and Subtraction are also evaluated from left to right as they occur in the same expression. String concatenation is not an Arithmetic operator, but in the order of precedence it occurs after the Arithmetic operators and before all of the comparison operators. All comparison operators have equal precedence, so they are simply evaluated in the order in which they appear from left to right. The next table sets forth the comparison operators.

Operator	Symbol
Equality	=
Inequality	<>
Less Than	<
Greater Than	>
Less Than or Equal to	<=
Greater Than or Equal to	>=
Object Equivalence	Is

The Is operator of Object Equivalence indicates that two objects refer to the same object. Is does not compare the objects

or their values. The next table sets forth the order of precedence for the logical operators.

Operator	Symbol
Negation	Not
Conjunction	And
Disjunction	Or
Exclusion	Xor
Equivalence	Eqv
Implication	Imp

## Controlling The Flow Of Vbscript

This section covers the use of control structures. Control structures enable you to direct the flow and direction of your VBScript program based upon testing whether a condition or conditions are True or False.

1. If . . . Then
2. If . . . Then . . . Else
3. Select Case
4. Loops
5. Do Loops
6. For Next

### Controlling The Flow Of Vbscript (If . . . Then)

The most basic control structure is the If . . . Then statement. Here is the syntax:

```
If (condition) Then
```

```
Statements following Then are executed
```

```
End If
```

The If . . . Then statement tests to determine whether a condition is True or False. If the test determines that the condition is True then the code in the If . . . Then statement is executed. If the condition is False then the code in the If . . . Then statement is not executed as shown in the list below:

```

<HTML>
<HEAD>
<TITLE>If Then Control Structure</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
Dim TestCondition
TestCondition = True
If TestCondition = True Then
MsgBox "Condition Tested True."
End If
-->
</SCRIPT>
</BODY>
</HTML>

```

The MsgBox function in this example displays a message box. This function is particularly useful for displaying results. The MsgBox function has the following syntax:

```
MsgBox (prompt [, buttons] [, title] [, helpfile, context])
```

Since the variable TestCondition is specifically assigned the value of True, the condition of the If . . . Then statement will test True and the code within the If . . . Then statement is executed. In this case, the code in the If . . . Then statement is a message box function with the message “Condition Tested True.” If TestCondition is assigned the value of False, then the message box is not displayed. Instead of stating the test condition as “TestCondition = True”, you could simply use the form:

```
If TestCondition Then
MsgBox “Condition Tested True.”
End If
```

### Controlling The Flow Of Vbscript (If . . . Then . . . Else)

Another useful structure is the If . . . Then . . . Else statement. This control structure allows you to do something in your code whether the tested condition is True or False. Here is the syntax:

```
If (condition = True) Then
executed code
Else
executed code
End If

<HTML>
<HEAD>
<TITLE>If Then Else Control Structure</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=“VBScript”>
<!-- Option Explicit
Dim TestCondition
TestCondition = False
If TestCondition = True Then
MsgBox “Condition Tested True.”
Else
MsgBox “Condition Tested False.”
End If
-->
</SCRIPT>
</BODY>
</HTML>
```

Here, the condition TestCondition is set to False. As a result, the condition of the If . . . Then statement tests False and the message “Condition Tested True.” is not displayed. Instead, the code in the False portion of the statement is executed and the “Condition Tested False” message is displayed. If you have more than one condition to test for, you can still use the If . . . Then . . . Else statement with one change: replace the Else with ElseIf. Here is the syntax:

```
If (condition1 = True) Then
‘ executed code
ElseIf (condition2 = True) Then
‘ executed code
```

```
End If
<HTML>
<HEAD>
<TITLE>If Then Else Control Structure</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=“VBScript”>
<!-- Option Explicit
Dim TestCondition
TestConditionOne = False
TestConditionTwo = True
If TestConditionOne = True Then
MsgBox “TestConditionOne Tested True.”
ElseIf TestConditionTwo = True Then
MsgBox “TestConditionTwo Tested True.”
Else
MsgBox “No Condition Tested True.”
End If
-->
</SCRIPT>
</BODY>
</HTML>
```

In this example, you can test whether TestConditionOne or TestConditionTwo is true. Here, TestConditionOne tests to false and TestConditionTwo tests to true so the message “TestConditionTwo Tested True.” is displayed. The previous example tests for condition where no conditions are true. Any number of ElseIf statements can be added to test for as many conditions as you wish.

### Controlling The Flow Of vbscript (Select Case)

Another control structure to test multiple conditions is the Select Case structure. Here is the syntax:

```
Select Case Expression
Case Expression1
‘ executed code
Case Expression2
‘ executed code
Case Else
‘ executed code
End Select
```

Select Case tests Expression against Expression1 and Expression2. If there is a match the code under the matched expression gets executed. There is only one match per Select Case structure. If there is no match, the code following the Case Else is executed at the end of the Select Case structure, as shown in the example below:

```
<HTML>
<HEAD>
<TITLE> Example of Select Case Control </TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=“VBScript”>
<!-- Option Explicit
```

```

Dim Age
Age = 25
Select Case Age
Case "10"
MsgBox "The Selected Age is 10."
Case "15"
MsgBox "The Selected Age is 15."
Case "20"
MsgBox "The Selected Age is 20."
Case "25"
MsgBox "The Selected Age is 25."
Case Else
MsgBox "No Age is selected."
End Select
—>
</SCRIPT>
</BODY>
</HTML>

```

The example above is good one of how it displays in Internet Explorer. In this example, the program tests the variable Age against each of the expressions following the word Case. There is no match testing the variable Age against 10. Age is then tested against 15 and 20, and there is still no match. Finally, the program tests Age against 25; since the value of Age is equal to 25, there is a match and the code following the line Case "25" executes, displaying the message, "The Selected Age is 25." If there is no match, then the code following the line Case Else executes by default, displaying the message "No Age is selected."

### Controlling The Flow Of Vbscript (Loops)

Loops are particularly valuable control structures that allow you to repeatedly execute code until a particular condition is satisfied. There are several different kinds of loops: Do . . . Loops repeatedly execute code as long as or until a specified condition is true; While . . . Wend loops continue while a condition is true; For . . . Next loops repeat for a specific number of times

1. **Do . . . Loops**
2. **For . . . Next**

### Controlling The Flow Of Vbscript (Do . . . Loops)

A Do . . . Loops is a control structure that repeats either as long as a condition is true or until a condition true. Here is the syntax of a Do . . . Loop (called a Do While . . . Loop) that repeats as long as a condition is true:

Do While condition

‘ execute code

Loop

The Do . . . Loop checks the condition, and if is True, then the program enters the loop and the code inside the loop executes. When the end of the code in the loop is reached, then the program tests the condition again. If the condition is True, the program enters the loops again and the code inside the loop executes again. This looping continues until the test condition if False; at that time the next statement after the Do While . . .

Loop is executed. An important characteristic of the Do While . . . Loop shown in the following example, is that the code in the loop never executes unless the condition is True at least one time.

```

<HTML>
<HEAD>
<TITLE> Do While Loop Control Structure </TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
Dim Index
Index = 0
Do While Index <= 2
Index = Index + 1
If Index > 2 Then
MsgBox "Index is Greater Than 2."
End If
Loop
—>
</SCRIPT>
</BODY>
</HTML>

```

In this example, the program goes through the following steps, which are further illustrated in the following analysis:

1. The initial value of the variable Index is 0
2. The program tests the value of Index 0 in the condition portion of the loop, Do While Index <= 2. Since 0 is less than 2, the program enters the loop and the code in the loop executes
3. The value of Index increments by 1
4. The program then tests the condition of the If . . . Then statement, Index > 2. Since Index is less than 2 the code in the If . . . Then statement does not execute.
5. The program then goes back to the start of the loop. The program tests the condition Index <= 2 again. Since Index is now equal to 1, and is less than 2, the program enters the loop and the code inside the loop executes
6. Index increments and now has the value of 2
7. The condition of the If . . . Then statement is tested. Since Index is still less than 2 the code in the If . . . Then statement does not execute
8. The program then goes back to the start of the loop. The condition Index <= 2 is tested again. Since Index is equal 2, the program enters the loop and the code inside the loop executes.
9. Index increments and now has the value of 3.
10. Since Index is now greater than 2, the code in the If . . . Then statement executes and the message box displays.
11. After the user clicks the OK button of the message box, the program goes back to the start of the Do While . . . Loop and tests the condition Index < 2 again. This time Index is

equal to 3 so the condition of the Do While . . . loop is not satisfied and the loop is not entered.

12. The program then executes the next line of code following the Do While . . . loop.

A nother kind of loop is the Do . . . Until loop. In this kind of control structure, the looping continues until the condition tests true. Here is the syntax

Do Until condition

' execute code

### Loop

In this kind of loop, the code within the loop executes only if the test condition is False. If the test condition is True, then the code in the loop never executes. For example, the following example modifies the previous example to use the Do . . . Until loop structure.

```
<HTML>
<HEAD>
<TITLE> Do Until Loop </TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit

Dim Index
Index = 0
Do Until > 2

Index = Index + 1
If Index > 2 Then
MsgBox "Index is Greater Than 2."
End If
Loop
-->
</SCRIPT>
</BODY>
</HTML>
```

In this example, the Do . . . Until loop executes until the value of the variable Index increments to the value of 3. At that point, the condition of the If . . . Then statement is True and the program does not enter the Do . . . Until loop.

Now let's look at control structures that allow the code in the loop to execute at least once before, rather than after, testing the condition. In these kinds of loops, the code always executes at least one time even if the condition tests False. For example, the Do loop, While structure allows the code in the loop to execute at least one time and then loops as long as a certain condition is True. Here is the syntax:

Do

' execute code

Loop While condition

The next example provides an example of the Do . . . loop, While control structure

```
<HTML>
<HEAD>
<TITLE> Do While Loop Control Structure </TITLE>
```

```
</HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit

Dim Index
Index = 6
Do

Index = Index + 1
If Index > 2 Then
MsgBox "Index is Greater Than 2."
End If
Loop While Index <= 2
-->
</SCRIPT>
</BODY>
</HTML>
```

In this example, even though Index has an initial value greater than 2, and therefore the test condition is never satisfied, the code in the loop still executes one time and the message box appears.

The Do Loop Until is similar control structure, except that the code in the loop executes at least once and the loop repeats only if the condition at the end of the loop is False. Here is the syntax:

Do

' execute code

Loop Until condition

The next example of the Do Loop Until control structure.

```
<HTML>
<HEAD>
<TITLE> Do Loop Until Control Structure </TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit

Dim Index
Index = 0
Do

Index = Index + 1
If Index > 2 Then
MsgBox "Index is Greater Than 2."
End If
Loop Until Index > 2
-->
</SCRIPT>
</BODY>
</HTML>
```

In this example, the code inside the loop executes until the condition Index > 2 is True. In other words, the loop keeps executing as long as the test condition at the end of the loop Index > 2 is False.

### Controlling The Flow of Vbscript (For . . . Next)

The For . . . Next loop shown in the following example is a good control structure to use when you know in advance how many times you want the code in the loop to execute. Here is the syntax:

```
For Counter = Start to End
' execute code
Next
<HTML>
<HEAD>
<TITLE> For Loop </TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
For Index = 1 to 6
MsgBox "The current Index is " & Index
Next
-->
</SCRIPT>
</BODY>
</HTML>
```

In this example, the message box displays six times with the value of Index for each repetition of For . . . Loop. When Index reaches the value of 6, the For . . . Loop stops. You can increase the counter of the For . . . Loop in increments greater than 1, using the following syntax:

```
For Counter = Start to End Step Increment
' executed code
Next
For example, in the following codes, you can increment the counter to 6 by a value of 2 with each repetition through the loop.
<HTML>
<HEAD>
<TITLE> For Loop Incrementing By Two </TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
For Index = 1 to 6 Step 2
MsgBox "The current Index is " & Index
Next
-->
</SCRIPT>
</BODY>
</HTML>
```

With this example, the For . . . Loop executes the message box statement 4 times for Index values of 0, 2, 4, and 6, increasing Index by a value of 2 with each repetition through the loop. You can also decrease the counter using a negative Step value, as shown in the following example. Be sure to make the ending value less than the starting value.

```
<HTML>
<HEAD>
<TITLE> For Loop Incrementing By Two </TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
For Index = 6 to 0 Step -2
MsgBox "The current Index is " & Index
Next
-->
</SCRIPT>
</BODY>
</HTML>
```

Here the For . . . Loop executes four times for the Index values of 6, 4, 2, and 0, decreasing Index by a value of 2 with each repetition through the loop.

### Procedures and Functions

There are two kinds of procedures in VBScript: subprocedures and functions. You must define subprocedures and functions before you use them. They should be placed in the <HEAD> section of your HTML document.

1. **Subprocedures**
2. **Functions**

#### Procedures and Functions - Subprocedures

A subprocedure is a procedure in VBScript that has the following syntax:

```
Sub Name(argument_one, argument_two, argument_n)
' execute code
End Sub
```

A subprocedure (see the example below) can take arguments in the form of constants, variables, or expressions that are passed to the subprocedure when it is called. A subprocedure can also take multiple arguments. If a subprocedure does not have any arguments, the set of empty parentheses follows the name of the subprocedure.

```
<HTML>
<HEAD>
<TITLE> Greeting Subprocedure</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!-- Option Explicit
Sub Greeting()
MsgBox "Hello World."
Call Greeting
-->
</SCRIPT>
</BODY>
</HTML>
```

The Greeting subprocedures begins with the Sub keyword and ends with End Sub. The subprocedure displays the message "Hello World" when the keyword Call invokes the

