

VBScript Tutorial



VBSCRIPT TUTORIAL

Simply Easy Learning by tutorialspoint.com

tutorialspoint.com

ABOUT THE TUTORIAL

VBScript Tutorial

Microsoft VBScript (Visual Basic Script) is a general-purpose, lightweight and active scripting language developed by Microsoft that is modeled on Visual Basic. Nowadays, VBScript is the primary scripting language for Quick Test Professional (QTP), which is a test automation tool.

This tutorial will teach you how to use VbScript scripting language in your day-2-day life of any Web based or Automation project development.

Audience

This tutorial has been prepared for the beginners to help them understand basic-to-advanced functionality of VBScript. After completing, this tutorial you will find yourself at a moderate level of expertise in using Microsoft VBScript from where you can take yourself to next levels.

Prerequisites

Before you start doing practice with various types of examples given in this reference, I'm making an assumption that you are already aware about what is a computer program and what is a computer programming language.

Copyright & Disclaimer Notice

©All the content and graphics on this tutorial are the property of tutorialspoint.com. Any content from tutorialspoint.com or this tutorial may not be redistributed or reproduced in any way, shape, or form without the written permission of tutorialspoint.com. Failure to do so is a violation of copyright laws.

This tutorial may contain inaccuracies or errors and tutorialspoint provides no guarantee regarding the accuracy of the site or its contents including this tutorial. If you discover that the tutorialspoint.com site or this tutorial content contains some errors, please contact us at webmaster@tutorialspoint.com

Table of Content

VBScript Tutorial	2
Audience	2
Prerequisites	2
Copyright & Disclaimer Notice.....	2
VBScript - Overview	11
Features of VBScript	11
VBscript – Version History and Uses	11
Disadvantages	12
Where VBScript is Today ?	12
VBScript – Syntax	13
Whitespace and Line Breaks.....	13
Formatting	13
SINGLE LINE SYNTAX.....	13
MULTIPLE LINE SYNTAX	14
Reserved Words	14
Case Sensitivity.....	15
Comments in VBScript	15
VBScript - Enabling.....	17
VBScript in Internet Explorer	17
VBScript – Placement	18
VBScript in <head>...</head> section:	18
VBScript in <body>...</body> section:	19
VBScript in <body> and <head> sections:	19
VBScript in External File :.....	20
VBScript Placement in QTP	20
VBScript – Variables	21
Declaring Variables	21
Assigning Values to the Variables	21
RULES :	22
EXAMPLES :	22
Scope of the Variables	22
1. DIM	22
2. PUBLIC	23
3. PRIVATE	23
VBScript – Constants	25
Declaring Constants	25
SYNTAX:	25

EXAMPLE 1:	25
EXAMPLE 2:	26
EXAMPLE 3:	26
VBScript – Operators	27
The Arithmetic Operators	27
Example	28
The Comparison Operators	28
Example	29
The Logical Operators:	30
Example	31
The Concatenation Operators	32
Example	33
VBScript - Decisions	35
Example	37
Flow Diagram	38
Example :	38
Flow Diagram	39
Example :	39
Example :	41
Example :	42
VBScript - Loops	43
for loop	44
Syntax:	44
Flow Diagram:	45
Example:	45
For...Each loop	46
Syntax:	46
Example:	46
While...Wend Loop	47
Syntax:	47
Flow Diagram:	48
Example :	48
Do..while loops	49
Syntax:	49
Flow Diagram	50
Example :	50
Alternate Syntax :	51
Flow Diagram	52
Example :	52

Do..until loops	53
Syntax:	53
Flow Diagram	53
Example :	54
Alternate Syntax :	54
Flow Diagram	55
Example :	55
Loop Control Statements:.....	56
Flow Diagram	57
Example :	57
VBScript – Events	60
onclick Event Type:	60
Example:	60
Example:	61
onmouseover and onmouseout:	61
Example:	61
HTML 4 Standard Events	62
VBScript – Cookies	63
How It Works?	63
Storing Cookies:	64
SYNTAX:	64
EXAMPLE:	64
Reading Cookies:	64
EXAMPLE:	65
Setting the Cookies Expiration Date:	65
Example:	65
Deleting a Cookie:	66
Example:	66
VBScript – Numbers	67
NUMBER CONVERSION FUNCTIONS	67
Syntax :	67
Example	68
NUMBER FORMATING FUNCTIONS	68
Syntax :	69
Description	69
Settings	69
Example :	69
MATHEMATICAL FUNCTIONS	70
Syntax :	71

Description	71
Example	71
VBScript - Strings.....	73
Syntax :	73
Examples :	73
String Functions :	73
VBScript – Arrays.....	86
Array Declaration:	86
Assigning Values to an Array :	86
Example :	86
Multi Dimension Arrays	87
Example :	87
Redim Statement	88
Example :	88
Array Methods :	89
VBScript – Date and Time.....	96
Date Functions	96
Time Functions.....	110
VBScript – Procedures.....	115
Function Definition:	115
Example :	115
Calling a Function :	116
Function Parameters:	116
Returning a Value from a Function :	116
Example :	117
Sub Procedures :	117
Example :	118
Calling Procedures :	118
Advanced Concepts for Functions	118
ByVal-Pass the parameters by value	118
What are ByVal Parameters?	118
Example :	118
BRef-Pass the parameters by the reference	119
What are ByRef Parameters?	119
Example :	119
VBScript – Dialog Boxes	121
VBScript MsgBox Function.....	121
Syntax :	121
Parameter Description :	121

RETURN VALUES :	122
Example :	123
VBScript InputBox Function	123
Syntax :	123
Parameter Description :	123
Example :	124
VBScript – Object Oriented	125
What is an Object.....	125
SYNTAX :	125
EXAMPLE :	125
Destroying the Objects.....	125
SYNTAX :	125
EXAMPLE :	125
Object Usage :	126
CLASS.....	126
What is a Class?	126
SYNTAX :	126
CLASS VARIABLES :	126
Class Properties :	126
EXAMPLE :	127
Class Methods :	127
EXAMPLE :	127
Class Events :	127
EXAMPLE :	127
SCRIPTING.FILESYSTEMOBJECT	128
What are FSO Objects?	128
Objects and Collections :	128
Drive.....	128
EXAMPLE :	129
Drives	129
EXAMPLE :	129
File	130
METHODS :	130
PROPERTIES :	130
EXAMPLE :	131
Files.....	132
PROPERTIES :	132
EXAMPLE :	132
Folder	132

METHODS :	132
PROPERTIES :	133
EXAMPLE :	133
Folders	134
METHODS :	135
PROPERTIES :	135
EXAMPLE :	135
TextStream.....	135
SYNTAX :	135
EXAMPLE :	135
SCRIPTING.DICTIONARY	136
What are Dictionary Objects?.....	136
Syntax:	136
Example :	136
Exists Method :	136
Parameter Description :.....	137
Example :	137
Items Method :	137
Parameter Description :.....	137
Example :	137
Keys Method :	138
Parameter Description :.....	138
Example :	138
Remove Method :	138
Parameter Description :.....	138
Example :	139
Remove All Method :	139
Parameter Description :.....	139
Example :	139
DEBUG.....	140
What are Debug Objects?.....	140
Write.....	140
EXAMPLE :	140
WriteLine	140
EXAMPLE :	140
Enabling Debug Mode :	141
VBScript – Reg Expressions	142
RegExp Object.....	142
Properties :	142

Methods :	142
Matches Collection Object.....	142
Match Object.....	143
All about Pattern Parameter:	143
POSITION MATCHING:	143
LITERALS MATCHING :	143
CHARACTER CLASSES MATCHING :	144
REPETITION MATCHING :	145
ALTERNATION & GROUPING :	145
Building Regular Expressions	145
Example :	145
VBScript – Error Handling	147
Syntax errors :	147
Runtime errors:	147
Logical errors:	148
Err Object	148
EXAMPLE :	148
VBScript – Misc Statements.....	149
Option Explicit	149
SYNTAX :	149
EXAMPLE :	149
ScriptEngine.....	150
SYNTAX :	150
EXAMPLE :	150
IsEmpty	151
SYNTAX :	151
EXAMPLE :	151
IsNull	151
SYNTAX :	151
EXAMPLE :	151
IsObject.....	152
SYNTAX:.....	152
EXAMPLE :	152
IsNumeric.....	153
SYNTAX :	153
EXAMPLE :	153
TypeName.....	153
SYNTAX :	153
EXAMPLE :	154

Eval	155
SYNTAX :	155
EXAMPLE :	155
Execute	155
SYNTAX :	155
EXAMPLE :	155
With..End With	156
SYNTAX :	156
EXAMPLE :	156
Randomize	157
SYNTAX :	157
EXAMPLE :	157

VBScript - Overview



BScript stands for **V**isual **B**asic Scripting that forms a subset of Visual Basic for Applications (VBA).

VBA is a product of Microsoft which is included NOT only in other Microsoft products such as MS Project and MS Office but also in Third Party tools such as AUTO CAD.

Features of VBScript

- VBScript is a lightweight scripting language, which has a lightning fast interpreter.
- VBScript, for the most part, is case insensitive. It has a very simple syntax, easy to learn and to implement.
- Unlike C++ or Java, VBScript is an object-based scripting language and NOT an Object-Oriented Programming language.
- It uses Component Object Model (**COM**) in order to access the elements of the environment in which it is executing.
- Successful execution of VBScript can happen only if it is executed in Host Environment such as Internet Explorer (**IE**), Internet Information Services (**IIS**) and Windows Scripting Host (**WSH**)

VBscript – Version History and Uses

VBScript was introduced by Microsoft way back in 1996 and the first version was 1.0. The Current Stable version of VBScript is 5.8, which is available as part of IE8 or Windows 7.

The VBScript usage areas are aplenty and not restricted to the below list.

- VBScript is used as a scripting language in one of the popular Automation testing tools – Quick Test Professional abbreviated as **QTP**
- Windows Scripting Host, which is used mostly by Windows System administrators for automating the Windows Desktop.
- Active Server Pages (**ASP**), a server side scripting environment for creating dynamic webpages which uses VBScript or Java Script.
- VBScript is used for Client side scripting in Microsoft Internet Explorer.

- Microsoft Outlook Forms usually runs on VBScript; however, the application level programming relies on VBA (Outlook 2000 onwards).

Disadvantages

- VBScript is used only by IE Browsers. Other browsers such as Chrome, Firefox DONOT Support VBScript. Hence, JavaScript is preferred over VBScript.
- VBScript has a Limited command line support.
- Since there is no development environment available by default, debugging is difficult.

Where VBScript is Today ?

The current version of VBScript is 5.8, and with the recent development of .NET framework, Microsoft has decided to provide future support of VBScript within ASP.NET for web development.

Hence, there will NOT be any more new versions of VBScript engine but the entire defect fixes and security issues are being addressed by the Microsoft sustaining Engineering Team.

However, VBScript engine would be shipped as part of all Microsoft Windows and IIS by default.

VBScript – Syntax

Your First VBScript

Let us write a VBScript to print out "Hello World".

```
<html>
<body>
<script language="vbscript" type="text/vbscript">
    document.write("Hello World!")
</script>
</body>
</html>
```

In the above example, we called a function *document.write*, which writes a string into the HTML document. This function can be used to write text, HTML or both. So, above code will display following result:

```
Hello World!
```

Whitespace and Line Breaks

VBScript ignores spaces, tabs and newlines that appear within VBScript programs.

Because one can use spaces, tabs and newlines freely within the program so you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Formatting

VBScript is based on Microsoft's Visual Basic. Unlike JavaScript, no statement terminators such as semicolon is used to terminate a particular statement.

SINGLE LINE SYNTAX

Colons are used when two or more lines of VBScript ought to be written in a single line. Hence, in VBScript, Colons act as a line separator.

```
<script language="vbscript" type="text/vbscript">
var1 = 10 : var2 = 20
</script>
```

MULTIPLE LINE SYNTAX

When a statement in VBScript is lengthy and if user wishes to break it into multiple lines, then the user has to use underscore "_".

This improves the readability of the code. The following example illustrates how to work with multiple lines.

```
<script language="vbscript" type="text/vbscript">
var1 = 10
var2 = 20
Sum = var1 + var2
document.write("The Sum of two numbers"&_
"var1 and var2 is " & Sum)
</script>
```

Reserved Words

The following list shows the reserved words in VBScripting. These reserved words SHOULD NOT be used as a constant or variable or any other identifier names.

Loop	Lset	Me
Mod	New	Next
Not	Nothing	Null
On	Option	Optional
Or	ParamArray	Preserve
Private	Public	RaiseEvent
ReDim	Rem	Resume
Rset	Select	Set
Shared	Single	Static
Stop	Sub	Then
To	True	Type
And	As	Boolean
ByRef	Byte	ByVal
Call	Case	Class
Const	Currency	Debug
Dim	Do	Double

Each	Else	Elseif
Empty	End	EndIf
Enum	Eqv	Event
Exit	False	For
Function	Get	GoTo
If	Imp	Implements
In	Integer	Is
Let	Like	Long
TypeOf	Until	Variant
Wend	While	With
Xor	Eval	Execute
Msgbox	Erase	ExecuteGlobal
Option Explicit	Randomize	SendKeys

Case Sensitivity

VBScript is a case-insensitive language. This means that language keywords, variables, function names and any other identifiers need NOT be typed with a consistent capitalization of letters.

So identifiers `int_counter`, `INT_Counter` and `INT_COUNTER` have the same meaning within VBScript.

Comments in VBScript

Comments are used to document the program logic and the user information with which other programmers can seamlessly work on the same code in future.

It can include information such as developed by, modified by and it can also include incorporated logic. Comments are ignored by the interpreter while execution.

Comments in VBScript are denoted by two methods.

1. Any statement that starts with a Single Quote (') is treated as comment. Following is the example:

```
<script language="vbscript" type="text/vbscript">
<!--
' This Script is invoked after successful login
' Written by : TutorialsPoint
```



```
' Return Value : True / False  
//- >  
</script>
```

2. Any statement that starts with the keyword “REM”. Following is the example:

```
<script language="vbscript" type="text/vbscript">  
<!--  
    REM This Script is written to Validate the Entered Input  
    REM Modified by : Tutorials point/user2  
//- >  
</script>
```

VBScript - Enabling

Not All the modern browsers support VBScript. VBScript is supported just by Microsoft's Internet Explorer while other browsers(Firefox and Chrome) just support JavaScript. Hence, the developers prefer JavaScript over VBScript.

Though Internet Explorer (IE) supports VBScript, many a times you may need to enable or disable this feature manually. This tutorial will make you aware of the procedure of enabling and disabling VBScript support in Internet Explorer.

VBScript in Internet Explorer

Here are simple steps to turn on or turn off VBScript in your Internet Explorer:

1. Follow **Tools-> Internet Options** from the menu
2. Select **Security** tab from the dialog box
3. Click the **Custom Level** button
4. Scroll down till you find **Scripting option**
5. Select *Enable* radio button under **Active scripting**
6. Finally click OK and come out

To disable VBScript support in your Internet Explorer, you need to select *Disable* radio button under **Active scripting**.

VBScript – Placement

VBScript Placement in HTML File

There is a flexibility given to include VBScript code anywhere in an HTML document. But the most preferred way to include VBScript in your HTML file is as follows:

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

In the following section, we will see how we can put VBScript in different ways:

VBScript in <head>...</head> section:

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows:

```
<html>
<head>
<script type="text/Vbscript">
<!--
Function sayHello()
    MsgBox("Hello World")
End Function
//-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

This will produce the following result: A button with the name SayHello. Upon clicking on the Button, the message box is displayed to the user with the message "Hello World".

VBScript in <body>...</body> section:

If you need a script to run as the page loads so that the script generates content in the page, the script goes in the <body> portion of the document. In this case, you would not have any function defined using VBScript:

```
<html>
<head>
</head>
<body>
<script type="text/vbscript">
<!--
    document.write("Hello World")
//-->
</script>
<p>This is web page body </p>
</body>
</html>
```

This will produce the following result:

```
Hello World
This is web page body
```

VBScript in <body> and <head> sections:

You can put your VBScript code in <head> and <body> section altogether as follows:

```
<html>
<head>
<script type="text/vbscript">
<!--
Function sayHello()
    msgbox("Hello World")
End Function
//-->
</script>
</head>
<body>
<script type="text/vbscript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

This will produce the following result: Hello World message with a 'Say Hello' button. Upon Clicking on the button a message box with a message "Hello World" is displayed to the user.

```
Hello World
```

VBScript in External File :

As you begin to work more extensively with VBScript, you will likely find that there are cases, where you are reusing identical VBScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The *script* tag provides a mechanism to allow you to store VBScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external VBScript file in your HTML code using *script* tag and its *src* attribute:

```
<html>
<head>
<script type="text/vbscript" src="filename.vbs" ></script>
</head>
<body>
.....
</body>
</html>
```

To use VBScript from an external file source, you need to write your all VBScript source code in a simple text file with extension ".vbs" and then include that file as shown above.

For example, you can keep following content in filename.vbs file and then you can use *sayHello* function in your HTML file after including filename.vbs file.

```
Function sayHello()
    MsgBox "Hello World"
End Function
```

VBScript Placement in QTP

VBScript is placed in QTP(Quick Test Professional) tool but it is NOT enclosed within HTML Tags. The Script File is saved with the extension .vbs and it is executed by Quick Test Professional execution engine.

VBScript – Variables

VBScript Variables

V

ariable is a named memory location used to hold a value that can be changed during the script

execution. VBScript has only **ONE** fundamental data type, Variant.

Rules for Declaring Variables:

- Variable Name must begin with an alphabet.
- Variable names cannot exceed 255 characters.
- Variables Should NOT contain a period(.)
- Variable Names should be unique in the declared context.

Declaring Variables

Variables are declared using “dim” keyword. Since there is only ONE fundamental data type, all the declared variables are variant by default. Hence, a user **NEED NOT** mention the type of data during declaration.

Example 1: In this Example, IntValue can be used as a String, Integer or even arrays.

```
Dim Var
```

Example 2: Two or more declarations are separated by comma(,)

```
Dim Variable1,Variable2
```

Assigning Values to the Variables

Values are assigned similar to an algebraic expression. The variable name on the left hand side followed by an equal to (=) symbol and then its value on the right hand side.

RULES :

- The numeric values should be declared without double quotes.
- The String values should be enclosed within doublequotes("")
- Date and Time variables should be enclosed within hash symbol(#)

EXAMPLES :

```
' Below Example, The value 25 is assigned to the variable.
Value1 = 25

' A String Value 'VBScript' is assigned to the variable StrValue.
StrValue = "VBScript"

' The date 01/01/2020 is assigned to the variable DToday.
Date1 = #01/01/2020#

' A Specific Time Stamp is assigned to a variable in the below example.
Time1 = #12:30:44 PM#
```

Scope of the Variables

Variables can be declared using the following statements that determines the scope of the variable. The scope of the variable plays a crucial role when used within a procedure or classes.

- Dim
- Public
- Private

1. DIM

Variables declared using "Dim" keyword at a Procedure level are available only within the same procedure. Variables declared using "Dim" Keyword at script level are available to all the procedures within the same script.

Example : In the below example, the value of Var1 and Var2 are declared at script level while Var3 is declared at procedure level.

Note : The scope of this chapter is to understand Variables. Functions would be dealt in detail in the upcoming chapters.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

Dim Var1
Dim Var2

Call add()
Function add()
    Var1 = 10
```

```

    Var2 = 15
    Dim Var3
    Var3 = Var1+Var2
    MsgBox Var3 'Displays 25, the sum of two values.
End Function

Msgbox Var1    ' Displays 10 as Var1 is declared at Script level
Msgbox Var2    ' Displays 15 as Var2 is declared at Script level
Msgbox Var3    ' Var3 has No Scope outside the procedure. Prints Empty

</script>
</body>
</html>

```

2. PUBLIC

Variables declared using "Public" Keyword are available to all the procedures across all the associated scripts. When declaring a variable of type "public", Dim keyword is replaced by "Public".

Example : In the below example, Var1 and Var2 are available at script level while Var3 is available across the associated scripts and procedures as it is declared as Public.

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
Dim Var1
Dim Var2
Public Var3

Call add()

Function add()
    Var1 = 10
    Var2 = 15
    Var3 = Var1+Var2
    MsgBox Var3 'Displays 25, the sum of two values.
End Function

Msgbox Var1    ' Displays 10 as Var1 is declared at Script level
Msgbox Var2    ' Displays 15 as Var2 is declared at Script level
Msgbox Var3    ' Displays 25 as Var3 is declared as Public

</script>
</body>
</html>

```

3. PRIVATE

Variables that are declared as "Private" have scope only within that script in which they are declared. When declaring a variable of type "Private", Dim keyword is replaced by "Private".

Example : In the below example, Var1 and Var2 are available at Script Level. Var3 is declared as Private and it is available only for this particular script. Use of "Private" Variables is more pronounced within the Class.

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
Dim Var1

```



```
Dim Var2
Private Var3

Call add()
Function add()
    Var1 = 10
    Var2 = 15
    Var3 = Var1+Var2
    MsgBox Var3 'Displays the sum of two values.
End Function

Msgbox Var1 ' Displays 10 as Var1 is declared at Script level
Msgbox Var2 ' Displays 15 as Var2 is declared at Script level
Msgbox Var3 ' Displays 25 but Var3 is available only for this script.

</script>
</body>
</html>
```

VBScript – Constants

VBScript constants

Constant is a named memory location used to hold a value that CANNOT be changed during the script execution. If a user tries to change a Constant Value, the Script execution ends up with an error. Constants are declared the same way the variables are declared.

Declaring Constants

SYNTAX:

```
[Public | Private] Const Constant_Name = Value
```

The Constant can be of type Public or Private. The Use of Public or Private is Optional. The Public constants are available for all the scripts and procedures while the Private Constants are available within the procedure or Class. One can assign any value such as number, String or Date to the declared Constant.

EXAMPLE 1:

In this example, the value of pi is 3.4 and it displays the area of the circle in a message box.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim intRadius
    intRadius = 20
    const pi=3.14
    Area = pi*intRadius*intRadius
    MsgBox Area

</script>
</body>
</html>
```

EXAMPLE 2:

The below example illustrates how to assign a String and Date Value to a Constant.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Const myString = "VBScript"
    Const myDate = #01/01/2050#
    MsgBox myString
    MsgBox myDate

</script>
</body>
</html>
```

EXAMPLE 3:

In the below example, the user tries to change the Constant Value; hence, it will end up with an **Execution Error**.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim intRadius
    intRadius = 20
    const pi=3.14
    pi = pi*pi      'pi VALUE CANNOT BE CHANGED.THROWS ERROR'
    Area = pi*intRadius*intRadius
    MsgBox Area

</script>
</body>
</html>
```

VBScript – Operators

What is an Operator?

Simple answer can be given using expression *4 + 5 is equal to 9*. Here, 4 and 5 are called operands and + is called operator. VBScript language supports following types of operators:

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Concatenation Operators

The Arithmetic Operators

There are following arithmetic operators supported by VBScript language:

Assume variable A holds 5 and variable B holds 10, then:

Operator	Description	Example
+	Adds two operands	A + B will give 15
-	Subtracts second operand from the first	A - B will give -5
*	Multiply both operands	A * B will give 50
/	Divide numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B MOD A will give 0
^	Exponentiation Operator	B ^ A will give 100000

Example

Try the following example to understand all the arithmetic operators available in VBScript:

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
    Dim a : a = 5
    Dim b : b = 10
    Dim c

    c = a+b
    Document.write ("Addition Result is " &c)
    Document.write ("<br></br>")    'Inserting a Line Break for readability
    c = a-b
    Document.write ("Subtraction Result is " &c)
    Document.write ("<br></br>")    'Inserting a Line Break for readability
    c = a*b
    Document.write ("Multiplication Result is " &c)
    Document.write ("<br></br>")
    c = b/a
    Document.write ("Division Result is " &c)
    Document.write ("<br></br>")
    c = b MOD a
    Document.write ("Modulus Result is " &c)
    Document.write ("<br></br>")
    c = b^a
    Document.write ("Exponentiation Result is " &c)
    Document.write ("<br></br>")
</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Addition Result is 15

Subtraction Result is -5

Multiplication Result is 50

Division Result is 2

Modulus Result is 0

Exponentiation Result is 100000
```

To understand these operators in a better way, you can [Try it yourself](#).

The Comparison Operators

There are following comparison operators supported by VBScript language:

Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is False.
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A <> B) is True.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is False.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is True.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is False.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is True.

Example

Try the following example to understand all the Comparison operators available in VBScript:

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
    Dim a : a = 10
    Dim b : b = 20
    Dim c

    If a=b Then
        Document.write ("Operator Line 1 : True")
        Document.write ("<br></br>") 'Inserting a Line Break for readability
    Else
        Document.write ("Operator Line 1 : False")
        Document.write ("<br></br>") 'Inserting a Line Break for readability
    End If

    If a<>b Then
        Document.write ("Operator Line 2 : True")
        Document.write ("<br></br>")
    Else
        Document.write ("Operator Line 2 : False")
        Document.write ("<br></br>")
    End If

    If a>b Then
        Document.write ("Operator Line 3 : True")
        Document.write ("<br></br>")
    Else
        Document.write ("Operator Line 3 : False")
        Document.write ("<br></br>")
    End If

    If a<b Then
        Document.write ("Operator Line 4 : True")
        Document.write ("<br></br>")
    End If

```

```

Else
    Document.write ("Operator Line 4 : False")
    Document.write ("<br></br>")
End If

If a>=b Then
    Document.write ("Operator Line 5 : True")
    Document.write ("<br></br>")
Else
    Document.write ("Operator Line 5 : False")
    Document.write ("<br></br>")
End If

If a<=b Then
    Document.write ("Operator Line 6 : True")
    Document.write ("<br></br>")
Else
    Document.write ("Operator Line 6 : False")
    Document.write ("<br></br>")
End If

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```

Operator Line 1 : False
Operator Line 2 : True
Operator Line 3 : False
Operator Line 4 : True
Operator Line 5 : False
Operator Line 6 : True

```

To understand these operators in a better way, you can [Try it yourself](#).

The Logical Operators:

There are following logical operators supported by VBScript language:

Assume variable A holds 10 and variable B holds 0, then:

Operator	Description	Example
AND	Called Logical AND operator. If both the conditions are True then Expression becomes true.	a<>0 AND b<>0 is False.
OR	Called Logical OR Operator. If any of the two conditions are True then condition becomes true.	a<>0 OR b<>0 is true.

NOT	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	NOT(a<>0 OR b<>0) is false.
XOR	Called Logical Exclusion. It is the combination of NOT and OR Operator. If one, and only one, of the expressions evaluates to True, result is True.	(a<>0 XOR b<>0) is false.

Example

Try the following example to understand all the Logical operators available in VBScript:

```
<!DOCTYPE html>

<html>

<body>

<script language="vbscript" type="text/vbscript">

    Dim a : a = 10

    Dim b : b = 0

    Dim c

    If a<>0 AND b<>0 Then

        Document.write ("AND Operator Result is : True")

        Document.write ("<br></br>")  'Inserting a Line Break for readability

    Else

        Document.write ("AND Operator Result is : False")

        Document.write ("<br></br>")  'Inserting a Line Break for readability

    End If

    If a<>0 OR b<>0 Then

        Document.write ("OR Operator Result is : True")

        Document.write ("<br></br>")

    Else

        Document.write ("OR Operator Result is : False")

        Document.write ("<br></br>")

    End If

    If NOT(a<>0 OR b<>0) Then

        Document.write ("NOT Operator Result is : True")
```



```

        Document.write ("<br></br>")
    Else
        Document.write ("NOT Operator Result is : False")
        Document.write ("<br></br>")
    End If

    If (a<>0 XOR b<>0) Then
        Document.write ("XOR Operator Result is : True")
        Document.write ("<br></br>")
    Else
        Document.write ("XOR Operator Result is : False")
        Document.write ("<br></br>")
    End If
</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```

AND Operator Result is : False
OR Operator Result is : True
NOT Operator Result is : False
XOR Operator Result is : True

```

To understand these operators in a better way, you can [Try it yourself](#).

The Concatenation Operators

There are following Concatenation operators supported by VBScript language:

Assume variable A holds 5 and variable B holds 10 then:

Operator	Description	Example
+	Adds two Values as Variable Values are Numeric	A + B will give 15
&	Concatenates two Values	A & B will give 510

Example

Try the following example to understand the Concatenation operator available in VBScript:

```
<!DOCTYPE html>

<html>

<body>

<script language="vbscript" type="text/vbscript">

    Dim a : a = 5

    Dim b : b = 10

    Dim c

    c=a+b

    Document.write ("Concatenated value:1 is " &c) 'Numeric addition

    Document.write ("<br></br>") 'Inserting a Line Break for readability

    c=a&b

    Document.write ("Concatenated value:2 is " &c) 'Concatenate two numbers

    Document.write ("<br></br>") 'Inserting a Line Break for readability

</script>

</body>

</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Concatenated value:1 is 15

Concatenated value:2 is 510
```

Assume variable A="Microsoft" and variable B="VBScript", then:

Operator	Description	Example
+	Concatenates two Values	A + B will give MicrosoftVBScript
&	Concatenates two Values	A & B will give MicrosoftVBScript

Example

Try the following example to understand the Concatenation operator available in VBScript:

```
<!DOCTYPE html>

<html>
```

```
<body>

<script language="vbscript" type="text/vbscript">

    Dim a : a = "Microsoft"

    Dim b : b = "VBScript"

    Dim c


    c=a+b

    Document.write ("Concatenated value:1 is " &c) 'Numeric addition

    Document.write ("<br></br>") 'Inserting a Line Break for readability

    c=a&b

    Document.write ("Concatenated value:2 is " &c) 'Concatenate two numbers

    Document.write ("<br></br>") 'Inserting a Line Break for readability

</script>

</body>

</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Concatenated value:1 is MicrosoftVBScript

Concatenated value:2 is MicrosoftVBScript
```

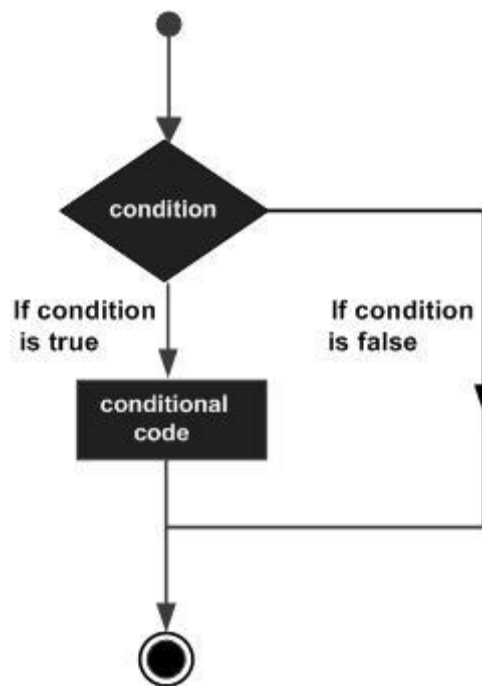
Note : Concatenation Operators can be used for numbers and strings. The Output depends on the context if the variables hold numeric value or String Value.

To understand these Operators in a better way, you can [Try it yourself](#).

VBScript - Decisions

Decision making allows programmers to control the execution flow of a script or one of its sections. The execution is governed by one or more conditional statements.

Following is the general form of a typical decision making structure found in most of the programming languages:



VBScript provides following types of decision making statements. Click the following links to check their details.

Statement	Description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if..else statement	An if else statement consists of a boolean expression followed by one or more statements. If the condition is True, the statements

	under If statements are executed. If the condition is false, Else part of the script is Executed
<u>if...elseif..else statement</u>	An if statement followed by one or more Elseif Statements, that consists of boolean expressions and then followed by an optional else statement , which executes when all the condition becomes false.
<u>nested if statements</u>	An if or elseif statement inside another if or elseif statement(s).
<u>switch statement</u>	A switch statement allows a variable to be tested for equality against a list of values.

if statement

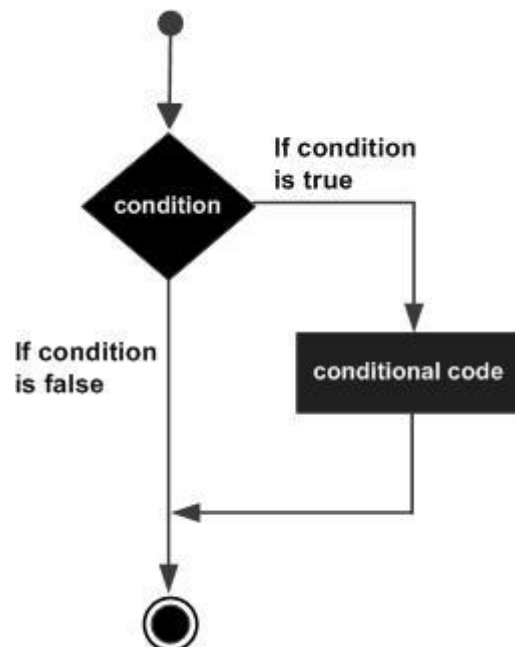
An **If** statement consists of a boolean expression followed by one or more statements. If the condition is said to be True, the statements under **If** condition(s) are Executed. If the Condition is said to be False, the statements after the **If** loop are executed.

Syntax :

The syntax of an **If** statement in VBScript is:

```
If(boolean_expression) Then
    Statement 1
    .....
    Statement n
End If
```

Flow Diagram



Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
Dim a : a = 20
Dim b : b = 10

If a > b Then
    Document.write "a is Greater than b"
End If

</script>
</body>
</html>
```

When the above code is executed, it produces the following result:

```
a is Greater than b
```

if...else statement

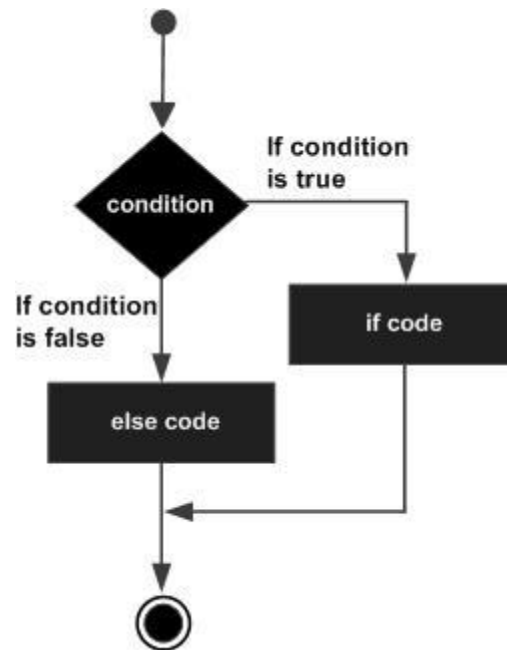
An **If** statement consists of a boolean expression followed by one or more statements. If the condition is said to be True, the statements under **If** condition(s) are Executed. If the Condition is said to be False, the statements under **Else** Part would be executed.

Syntax

The syntax of an if statement in VBScript is:

```
If (boolean_expression) Then
    Statement 1
    .....
    Statement n
Else
    Statement 1
    .....
    Statement n
End If
```

Flow Diagram



Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
Dim a : a = 5
Dim b : b = 25

If a > b Then
    Document.write "a is Greater"
Else
    Document.write "b is Greater"
End If

</script>
</body>
</html>
```

When the above code is executed, it produces the following result:

```
b is Greater
```

if...elseif...else statement

An **If** statement followed by one or more **Elseif** Statements that consists of boolean expressions and then followed by a default **else** statement, which executes when all the condition becomes false.

Syntax

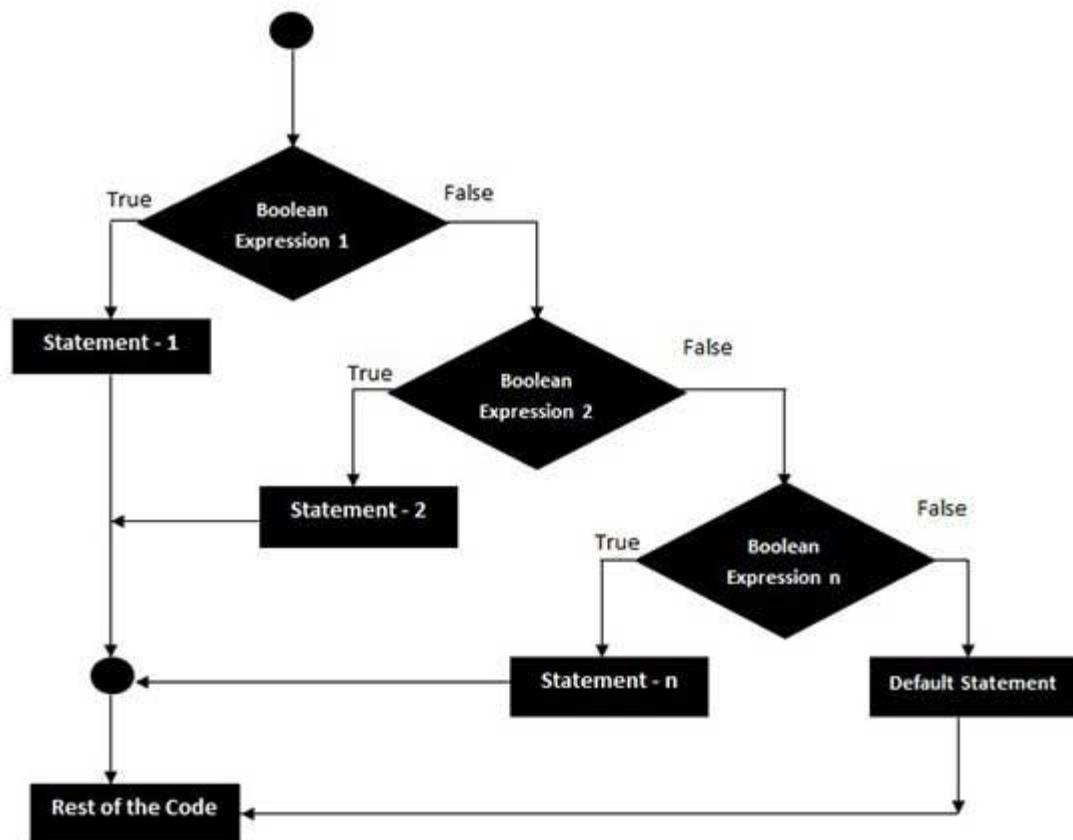
The syntax of an If-Elseif-Else statement in VBScript is:

```

If (boolean_expression) Then
    Statement 1
    .....
    Statement n
ElseIf (boolean_expression) Then
    Statement 1
    .....
    Statement n
ElseIf (boolean_expression) Then
    Statement 1
    .....
    Statement n
Else
    Statement 1
    .....
    Statement n
End If

```

Flow Diagram



Example :

```
<!DOCTYPE html>
```



```

<html>
<body>
<script language="vbscript" type="text/vbscript">
Dim a
a = -5

If a > 0 Then
    Document.write "a is a POSITIVE Number"
ElseIf a < 0 Then
    Document.write "a is a NEGATIVE Number"
Else
    Document.write "a is EQUAL than ZERO"
End If
</script>
</body>
</html>

```

When the above code is executed, it produces the following result:

```
a is a NEGATIVE Number
```

nested if statements

An **If** or **Elseif** statement inside another **If** or **Elseif** statement(s). The Inner **If** statements are executed based on the Outermost **If** statements. This enables VBScript to handle complex conditions with ease.

Syntax

The syntax of a Nested if statement in VBScript is:

```

If (boolean_expression) Then
    Statement 1
    .....
    .....
    Statement n
    If (boolean_expression) Then
        Statement 1
        .....
        .....
        Statement n
    ElseIf (boolean_expression) Then
        Statement 1
        .....
        .....
        Statement n
    Else
        Statement 1
        .....
        .....
        Statement n
    End If
Else
    Statement 1
    .....
    .....
    Statement n
End If

```

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
Dim a
a = 23

If a > 0 Then
    Document.write "The Number is a POSITIVE Number"
    If a = 1 Then
        Document.write "The Number is Neither Prime NOR Composite"
    ElseIf a = 2 Then
        Document.write "The Number is the Only Even Prime Number"
    ElseIf a = 3 Then
        Document.write "The Number is the Least Odd Prime Number"
    Else
        Document.write "The Number is NOT 0,1,2 or 3"
    End If
ElseIf a < 0 Then
    Document.write "The Number is a NEGATIVE Number"
Else
    Document.write "The Number is ZERO"
End If
</script>
</body>
</html>
```

When the above code is executed, it produces the following results:

```
The Number is a POSITIVE Number
The Number is NOT 0,1,2 or 3
```

switch statement

When a User want to execute a group of statements depending upon a value of an Expression, then Switch Case is used. Each value is called a **Case**, and the variable being switched **ON** based on each case. **Case Else** statement is executed if test expression doesn't match any of the Case specified by the user.

Case Else is an optional statement within Select Case, however, it is a good programming practice to always have a Case Else statement.

Syntax :

The syntax of a Switch Statement in VBScript is:

```
Select Case expression
    Case expressionlist1
        statement1
        statement2
        ....
        ....
        statement1n
    Case expressionlist2
        statement1
        statement2
        ....
        ....
    Case expressionlistn
```

```
statement1
statement2
....
....
Case Else
    elstatement1
    elstatement2
    ....
    ....
End Select
```

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
Dim MyVar
MyVar = 1

Select case MyVar
    case 1
        Document.write "The Number is the Least Composite Number"
    case 2
        Document.write "The Number is the only Even Prime Number"
    case 3
        Document.write "The Number is the Least Odd Prime Number"
    case else
        Document.write "Unknown Number"
End select
</script>
</body>
</html>
```

In the above example, the value of MyVar is 1. Hence, Case 1 would be executed.

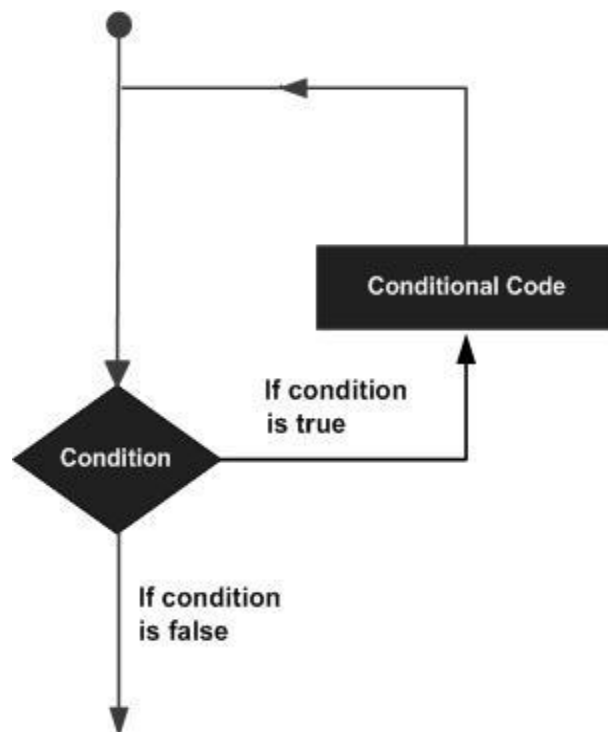
```
The Number is the Least Composite Number
```

VBScript - Loops

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general from of a loop statement in VBScript.



VBScript provides the following types of loops to handle looping requirements. Click the following links to check their detail.

Loop Type	Description
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
for ..each loop	This is executed if there is at least one element in group and reiterated for each element in a group.
while..wend loop	This tests the condition before executing the loop body.
do..while loops	The do..While statements will be executed as long as condition is True.(i.e.,) The Loop should be repeated till the condition is False.
do..until loops	The do..Until statements will be executed as long as condition is False.(i.e.,) The Loop should be repeated till the condition is True.

for loop

A **for** loop is a repetition control structure that allows a developer to efficiently write a loop that needs to execute a specific number of times.

Syntax:

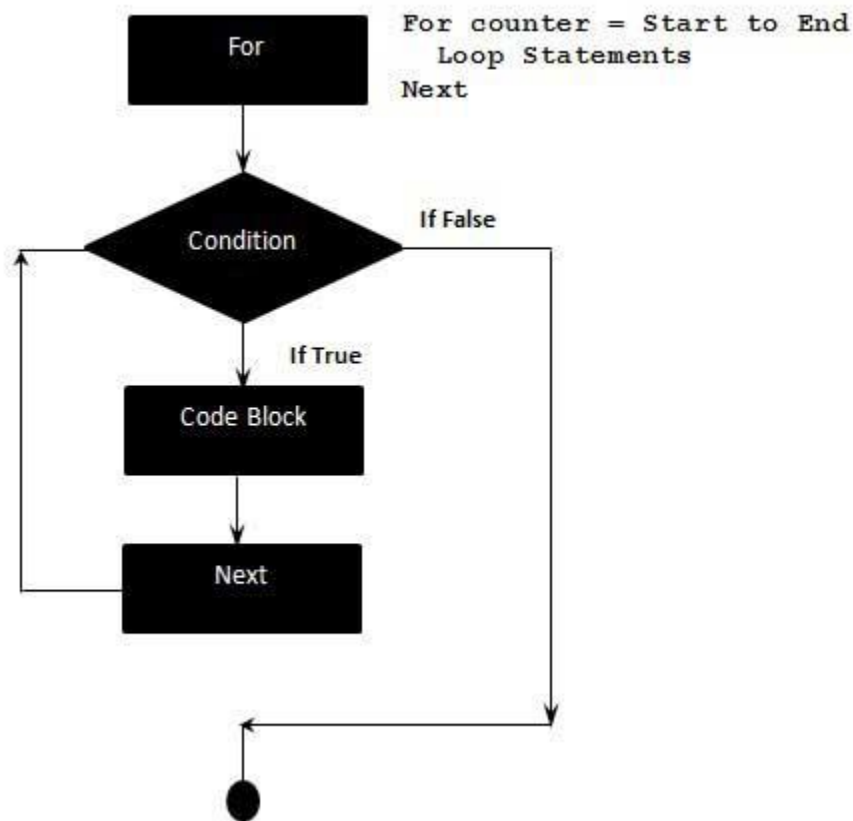
The syntax of a **for** loop in VBScript is:

```

For counter = start To end [Step stepcount]
    [statement 1]
    [statement 2]
    ....
    [statement n]
    [Exit For]
    [statement 11]
    [statement 22]
    ....
    [statement n]
Next

```

Flow Diagram:



Here is the flow of control in a For Loop:

1. The **For** step is executed first. This step allows you to initialize any loop control variables and increment the step counter variable.
2. Secondly, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the For Loop.
3. After the body of the for loop executes, the flow of control jumps to the **Next** statement. This statement allows you to update any loop control variables. It is updated based on the step counter value.
4. The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the For Loop terminates.

Example:

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim a : a=10
    For i=0 to a Step 2 'i is the counter variable and it is incremented by 2
        document.write("The value is i is : " & i)
```

```

        document.write("<br></br>")
    Next

</script>
</body>
</html>

```

When the above code is compiled and executed, it produces the following result:

```

The value is i is : 0

The value is i is : 2

The value is i is : 4

The value is i is : 6

The value is i is : 8

The value is i is : 10

```

For...Each loop

A **For Each** loop is used when we want to execute a statement or a group of statements for each element in an array or collection.

A **For Each** loop is similar to For Loop; however, the loop is executed for each element in an array or group. Hence, the step counter won't exist in this type of loop and it is mostly used with arrays or used in context of File system objects in order to operate recursively.

Syntax:

The syntax of a **For Each** loop in VBScript is:

```

For Each element In Group
    [statement 1]
    [statement 2]
    ....
    [statement n]
    [Exit For]
    [statement 11]
    [statement 22]
Next

```

Example:

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

'fruits is an array
fruits=Array("apple","orange","cherries")
Dim fruitnames

'iterating using For each loop.
For each item in fruits
    fruitnames=fruitnames&item&vbnewline
Next

```

```
msgbox fruitnames  
  
</script>  
</body>  
</html>
```

When the above code is executed, it prints all the fruitnames with one item in each line.

```
apple  
orange  
cherries
```

While...Wend Loop

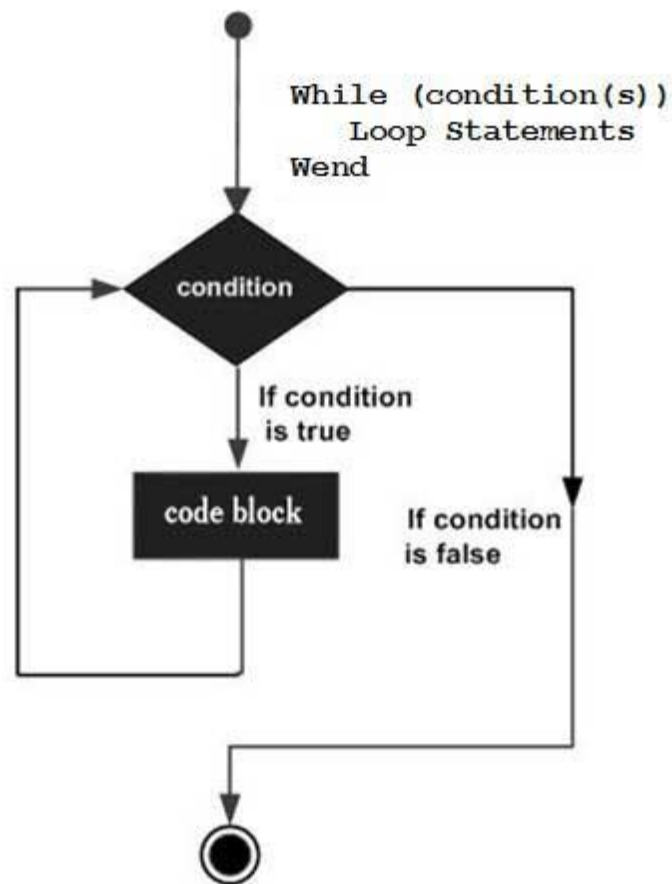
In a **While..Wend** loop, if the condition is True, all statements are executed until **Wend** keyword is encountered. If the condition is false, the loop is exited and the control jumps to very next statement after **Wend** keyword.

Syntax:

The syntax of a **While..Wend** loop in VBScript is:

```
While condition(s)  
    [statements 1]  
    [statements 2]  
    ...  
    [statements n]  
Wend
```


Flow Diagram:



Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim Counter : Counter = 10
    While Counter < 15      ' Test value of Counter.
        Counter = Counter + 1  ' Increment Counter.
        document.write("The Current Value of the Counter is : " & Counter)
        document.write("<br><br>")
    Wend      ' While loop exits if Counter Value becomes 15.

</script>
</body>
</html>
```

When the above code is executed, it prints the following output in the console.

```
The Current Value of the Counter is : 11
The Current Value of the Counter is : 12
```

The Current Value of the Counter is : 13

The Current Value of the Counter is : 14

The Current Value of the Counter is : 15

Do..while loops

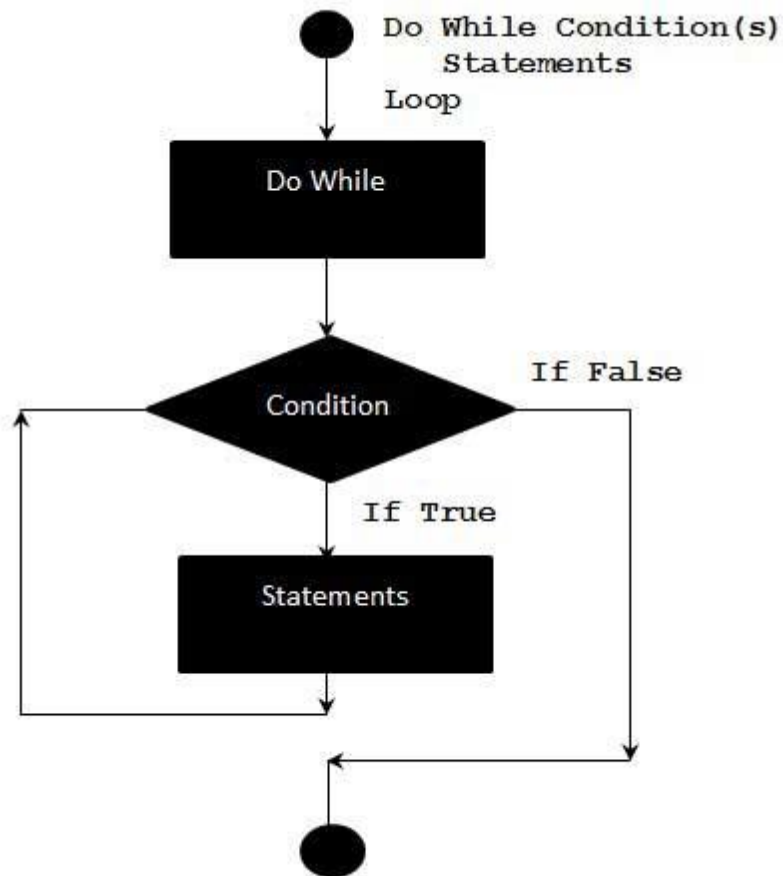
A **Do..While** loop is used when we want to repeat a set of statements as long as the condition is true. The Condition may be checked at the beginning of the loop or at the end of the loop.

Syntax:

The syntax of a **Do..While** loop in VBScript is:

```
Do While condition
    [statement 1]
    [statement 2]
    ...
    [statement n]
    [Exit Do]
    [statement 1]
    [statement 2]
    ...
    [statement n]
Loop
```

Flow Diagram



Example :

The below example uses **Do..while** loop to check the condition at the beginning of the loop. The statements inside the loop are executed only if the condition becomes True.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Do While i < 5
        i = i + 1
        Document.write("The value of i is : " & i)
        Document.write("<br></br>")
    Loop

</script>
</body>
</html>
```

When the above code is executed, it prints the following output in the console.

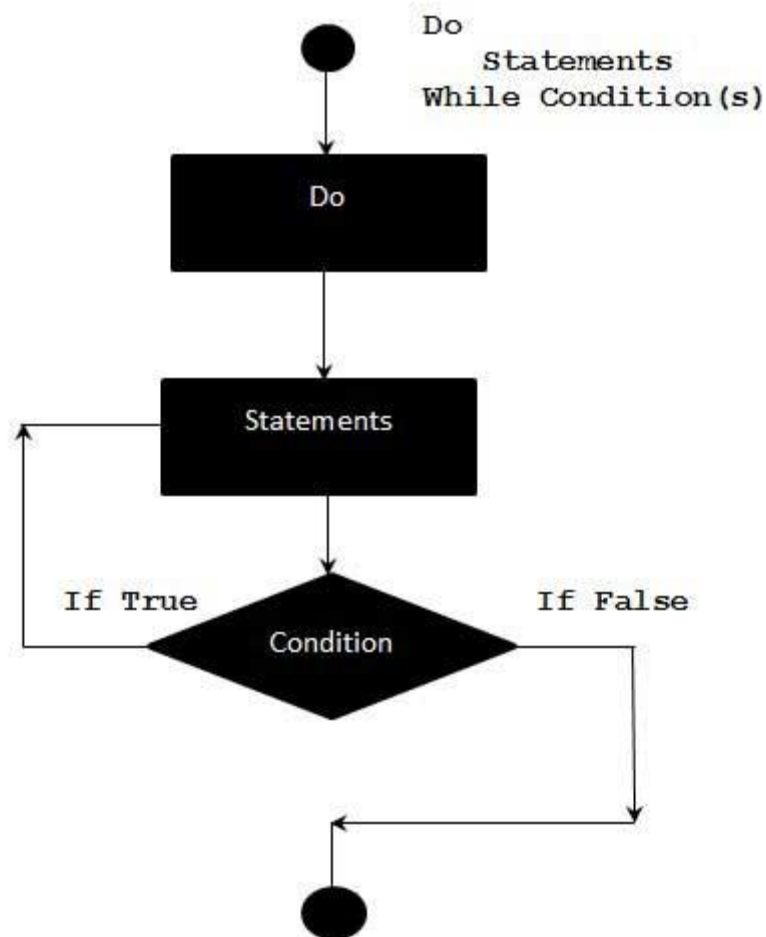
```
The value of i is : 1  
The value of i is : 2  
The value of i is : 3  
The value of i is : 4  
The value of i is : 5
```

Alternate Syntax :

There is also an alternate Syntax for **Do..while** loop which checks the condition at the end of the loop. The Major difference between these two syntax is explained below with an example.

```
Do  
    [statement 1]  
    [statement 2]  
    ...  
    [statement n]  
    [Exit Do]  
    [statement 1]  
    [statement 2]  
    ...  
    [statement n]  
Loop While condition
```

Flow Diagram



Example :

The below example uses **Do..while** loop to check the condition at the end of the loop. The Statements inside the loop are executed atleast once even if the condition is False.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    i=10
    Do
        i = i + 1
        Document.write("The value of i is : " & i)
        Document.write("<br></br>")
    Loop While i<3 'Condition is false.Hence loop is executed once.

</script>
</body>
</html>
```

When the above code is executed, it prints the following output in the console.

```
The value of i is : 11
```

Do..until loops

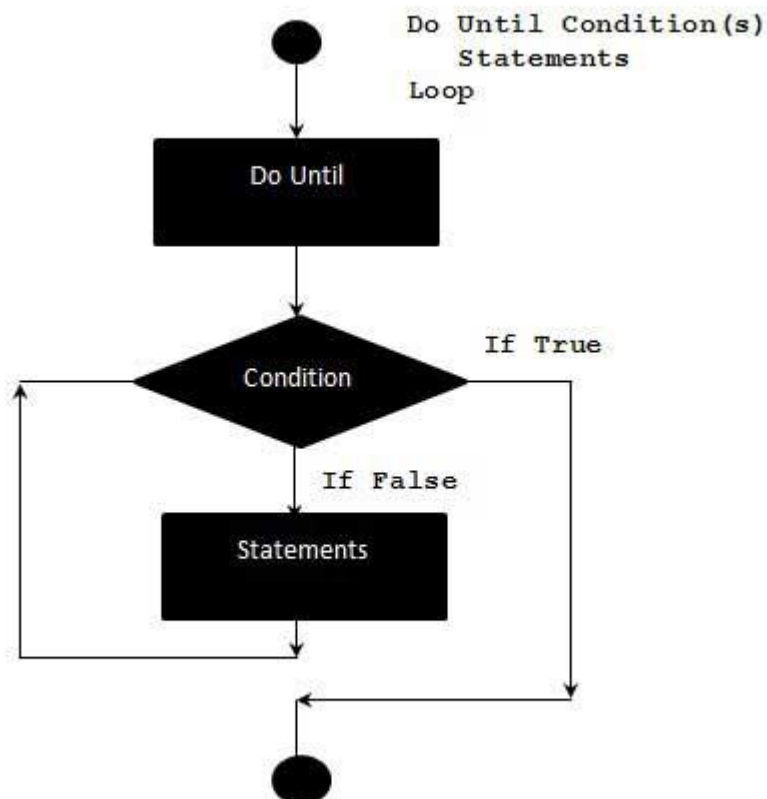
A **Do..Until** loop is used when we want to repeat a set of statements as long as the condition is false. The Condition may be checked at the beginning of the loop or at the end of loop.

Syntax:

The syntax of a **Do..Until** loop in VBScript is:

```
Do Until condition
    [statement 1]
    [statement 2]
    ...
    [statement n]
[Exit Do]
[statement 1]
[statement 2]
...
[statement n]
Loop
```

Flow Diagram



Example :

The below example uses **Do..Until** loop to check the condition at the beginning of the loop. The Statements inside the loop are executed only if the condition is false. It exits out of the loop when the condition becomes true.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    i=10
    Do Until i>15 'Condition is False.Hence loop will be executed
        i = i + 1
        Document.write("The value of i is : " & i)
        Document.write("<br></br>")
    Loop

</script>
</body>
</html>
```

When the above code is executed, it prints the following output in the console.

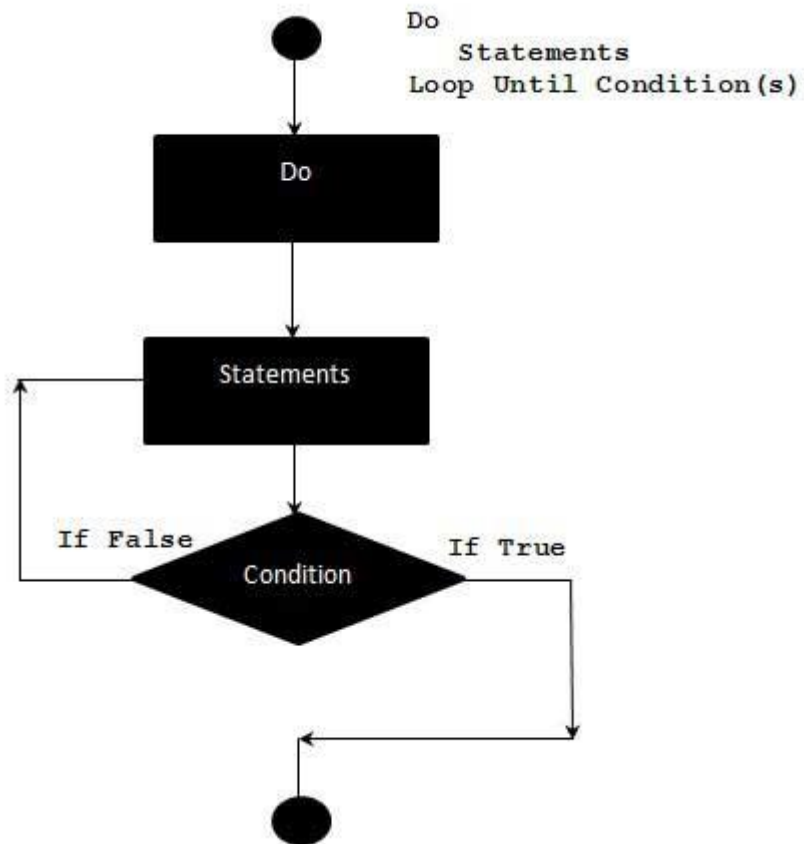
```
The value of i is : 11
The value of i is : 12
The value of i is : 13
The value of i is : 14
The value of i is : 15
The value of i is : 16
```

Alternate Syntax :

There is also an alternate Syntax for **Do..Until** loop which checks the condition at the end of the loop. The Major difference between these two syntax is explained below with an example.

```
Do
    [statement 1]
    [statement 2]
    ...
    [statement n]
    [Exit Do]
    [statement 1]
    [statement 2]
    ...
    [statement n]
Loop Until condition
```

Flow Diagram



Example :

The below example uses **Do..Until** loop to check the condition at the end of the loop. The Statements inside the loop are executed atleast once even if the condition is True.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    i=10
    Do
        i = i + 1
        Document.write("The value of i is : " & i)
        Document.write("<br></br>")
    Loop Until i<15 'Condition is True.Hence loop is executed once.

</script>
</body>
</html>
```

When the above code is executed, it prints the following output in the console.

```
The value of i is : 11
```


Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all the remaining statements in the loop are NOT executed.

VBScript supports the following control statements. Click the following links to check their detail.

Control Statement	Description
Exit For statement	Terminates the For loop statement and transfers execution to the statement immediately following the loop
Exit Do statement	Terminates the Do While statement and transfers execution to the statement immediately following the loop

Exit For Statement

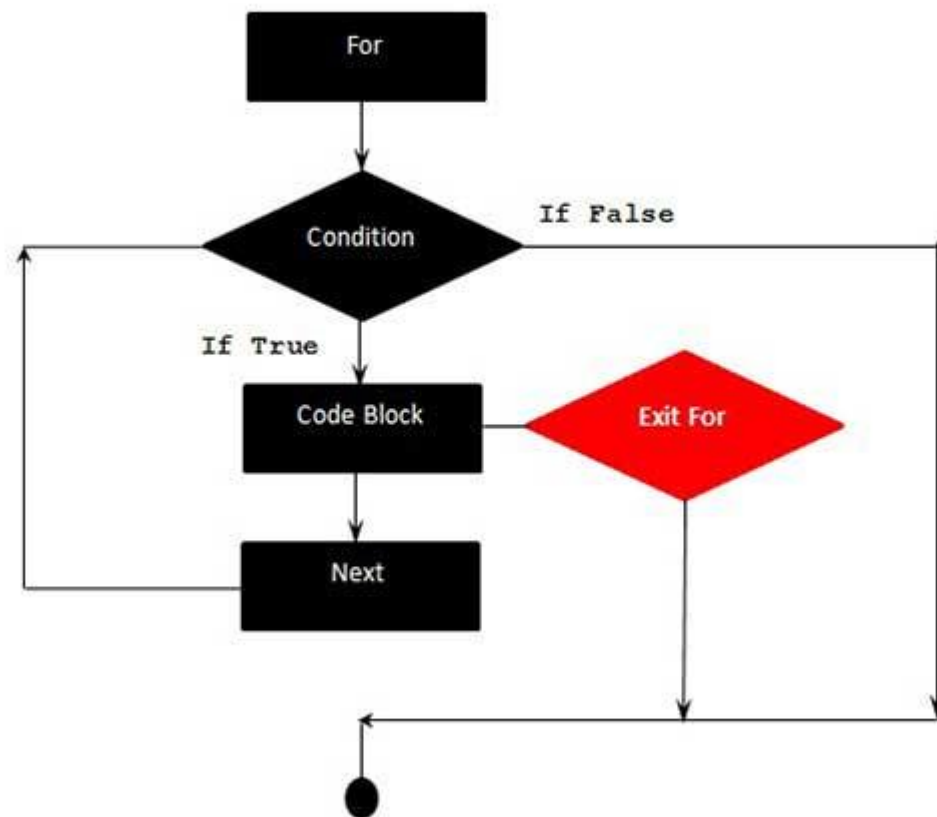
A **Exit For** Statement is used when we want to Exit the **For** Loop based on certain criteria. When **Exit For** is executed, the control jumps to next statement immediately after the **For** Loop.

Syntax:

The syntax for **Exit For** Statement in VBScript is:

```
Exit For
```

Flow Diagram



Example :

The below example uses **Exit For**. If the value of the Counter reaches 4, the For Loop is Exited and control jumps to the next statement immediately after the For Loop.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim a : a=10
    For i=0 to a Step 2 'i is the counter variable and it is incremented by 2
        document.write("The value is i is : " & i)
        document.write("<br></br>")
        If i=4 Then
            i=i*10 'This is executed only if i=4
            document.write("The value is i is : " & i)
            Exit For 'Exited when i=4
        End If
    Next

</script>
</body>
</html>
```

When the above code is executed, it prints the following output in the console.

```
The value is i is : 0  
The value is i is : 2  
The value is i is : 4  
The value is i is : 40
```

Exit Do Statement

An **Exit Do** Statement is used when we want to Exit the **Do** Loops based on certain criteria. It can be used within both **Do..While** and **Do..Until** Loops.

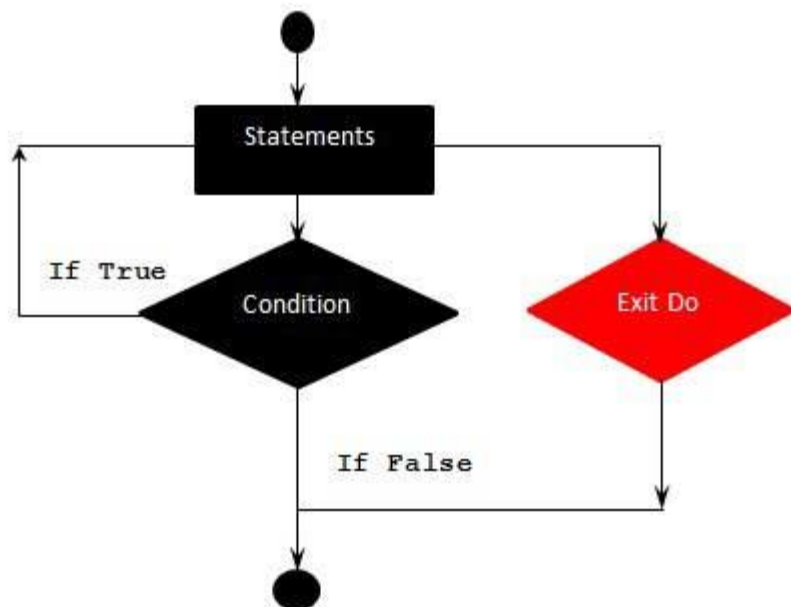
When **Exit Do** is executed, the control jumps to next statement immediately after the **Do** Loop.

Syntax:

The syntax for **Exit Do** Statement in VBScript is:

```
Exit Do
```

Flow Diagram



Example :

The below example uses **Exit Do**. If the value of the Counter reaches 10, the Do Loop is Exited and control jumps to the next statement immediately after the For Loop.

```
<!DOCTYPE html>  
<html>  
<body>
```

```
<script language="vbscript" type="text/vbscript">

i = 0
Do While i <= 100
    If i > 10 Then
        Exit Do      ' Loop Exits if i>10
    End If
    document.write("The Value of i is : " &i)
    document.write("<br></br>")
    i = i + 2
Loop

</script>
</body>
</html>
```

When the above code is executed, it prints the following output in the console.

```
The Value of i is : 0
The Value of i is : 2
The Value of i is : 4
The Value of i is : 6
The Value of i is : 8
The Value of i is : 10
```

VBScript – Events

What is an event?

VBScript's interaction with HTML is handled through events that occur when the user or browser manipulates a page.

When the page loads, that is an event. When the user clicks a button, that click too is an event. Another example of events are like pressing any key, closing window, resizing window, etc.

Developers can use these events to execute VBScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated and virtually any other type of response imaginable to occur.

Events are a part of the Document Object Model (DOM) and every HTML element has a certain set of events, which can trigger VBScript Code. Please go through this small tutorial for a better understanding [HTML Event Reference](#). Here, we will see few examples to understand a relation between Event and VBScript.

onclick Event Type:

This is the most frequently used event type, which occurs when a user clicks mouse's left button. You can put your validation, warning, etc., against this event type.

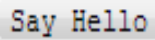
Example:

```
<html>
<head>
<script language="vbscript" type="text/vbscript">

Function sayHello()
    msgbox "Hello World"
End Function

</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello"/>
</body>
</html>
```

This will produce the following result, and when you click Hello button, then onclick event will occur which will trigger sayHello() function.



onsubmit event type:

Another most important event type is *onsubmit*. This event occurs when you try to submit a form. So you can put your form validation against this event type.

The Form is submitted by clicking on Submit button, the message box appears.

Example:

```
<html>
<head>
</head>
<body>
<script language="VBScript">

Function fnSubmit()
    MsgBox("Hello Tutorialspoint.Com")
End Function

</script>
<form action="/cgi-bin/test.cgi" method="post" name="form1" onSubmit="fnSubmit()">
<input name="txt1" type="text"><br>
<input name="btnButton1" type="submit" value="Submit">
</form>
</script>
</body>
</html>
```

onmouseover and onmouseout:

These two event types will help you to create nice effects with images or even with text as well. The *onmouseover* event occurs when you bring your mouse over any element and the *onmouseout* occurs when you take your mouse out from that element.

Example:

```
<html>
<head>
</head>
<body>
<script language="VBScript">

Function AlertMsg
    MsgBox("ALERT !")
End Function

Function onmourse_over()
    MsgBox("Onmouse Over")
End Function

Sub txt2_OnMouseOut()
    MsgBox("Onmouse Out !!!")
End Sub
```

```

End Sub

Sub btnButton_OnMouseOut()
    MsgBox("onmouse out on Button !")
End Sub

</script>
<form action="page.cgi" method="post" name="form1">
<input name="txt1" type="text" OnMouseOut="AlertMsg()" "><br>
<input name="txt2" type="text" OnMouseOver="onmouse_over()" ">
<br><input name="btnButton" type="button" value="Submit">
</form>
</body>
</html>

```

This will produce a result when you hover the mouse over the text box and also when you move the focus away from the text box and the button.

HTML 4 Standard Events

The standard HTML 4 events are listed here for your reference. Here, script indicates a VBscript function to be executed against that event.

Event	Value	Description
onchange	script	Script runs when the element changes
onsubmit	script	Script runs when the form is submitted
onreset	script	Script runs when the form is reset
onblur	script	Script runs when the element loses focus
onfocus	script	Script runs when the element gets focus
onkeydown	script	Script runs when key is pressed
onkeypress	script	Script runs when key is pressed and released
onkeyup	script	Script runs when key is released
onclick	script	Script runs when a mouse click
ondblclick	script	Script runs when a mouse double-click
onmousedown	script	Script runs when mouse button is pressed
onmousemove	script	Script runs when mouse pointer moves
onmouseout	script	Script runs when mouse pointer moves out of an element
onmouseover	script	Script runs when mouse pointer moves over an element
onmouseup	script	Script runs when mouse button is released

VBScript – Cookies

What are Cookies?

W

eb Browser and Server use HTTP protocol to communicate and HTTP is a stateless protocol. But

for a commercial website, it is required to maintain session information among different pages. For example, one user registration ends after completing many pages. But how to maintain user's session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions and other information required for better visitor experience or site statistics.

How It Works?

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields:

- **Expires** : The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain** : The domain name of your site.
- **Path** : The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure** : If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value** : Cookies are set and retrieved in the form of key and value pairs.

Cookies were originally designed for CGI programming and cookies' data is automatically transmitted between the web browser and web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

VBScript can also manipulate cookies using the *cookie* property of the *Document* object. VBScript can read, create, modify and delete the cookie or cookies that apply to the current web page.

Storing Cookies:

The simplest way to create a cookie is to assign a string value to the `document.cookie` object, which looks like this:

SYNTAX:

```
document.cookie = "key1=value1;key2=value2;expires=date"
```

Here *expires* attribute is optional. If you provide this attribute with a valid date or time, then cookie will expire at the given date or time and after that cookies' value will not be accessible.

EXAMPLE:

Following is the example to set a customer name in *input* cookie.

```
<html>
<head>
<script type="text/vbscript">

Function WriteCookie
    If document.myform.customer.value="" Then
        msgbox "Enter some value!"
    Else
        cookievalue=(document.myform.customer.value)
        document.cookie="name=" + cookievalue
        msgbox "Setting Cookies : " & "name=" & cookievalue
    End If
End Function

</script>
</head>
<body>
<form name="myform" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()" />
</form>
</body>
</html>
```

This will produce the following result. Now enter something in the text box and press the button "Set Cookie" to set the cookies

A screenshot of a web form. On the left, the text "Enter name:" is followed by a text input field. To the right of the input field is a button labeled "Set Cookie". The entire form is contained within a light gray rectangular box.

Now, your system has a cookie called *name*. You can set multiple cookies using multiple *key=value* pairs separated by comma.

You will learn how to read this cookie in next section.

Reading Cookies:

Reading a cookie is just as simple as writing one, because the value of the `document.cookie` object is the cookie. So, you can use this string whenever you want to access the cookie.

The `document.cookie` string will keep a list of *name=value* pairs separated by semicolons where *name* is the *name* of a cookie and value is its string value.

You can use strings' *split()* function to break the string into key and values as follows:

EXAMPLE:

Following is the example to get the cookies set in previous section:

```
<html>
<head>
<script type="text/vbscript">

Function ReadCookie
    allcookies = document.cookie
    msgbox "All Cookies : " + allcookies

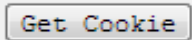
    cookiearray = split(allcookies,";")

    For i=0 to ubound(cookiearray)
        Name = Split(cookiearray(i),"=")
        MsgBox "Key is : " + Name(0) + " and Value is : " + Name(1)
    Next
End Function

</script>
</head>
<body>
<form name="myform" action="">
<input type="button" value="Get Cookie" onclick="ReadCookie()" />
</form>
</body>
</html>
```

Note: Here, *UBound* is a method of *Array* class, which returns the length of an array. We will discuss Arrays in a separate chapter; until that time, please try to digest it.

This will produce the following result. Now, press the button "Get Cookie" to see the cookies, which you have set in previous section.



Note: There may be some other cookies already set on your machine. So, above code will show you all the cookies set at your machine.

Setting the Cookies Expiration Date:

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiration date within the cookie. This can be done by setting the *expires* attribute to a date and time.

Example:

The following example illustrates how to set cookie expiration date after 1 Month :

```
<html>
<head>
<script type="text/vbscript">
Function WriteCookie()
    x = now()
    y = dateadd("m",1,now()) ' Making it to expire next
    cookievalue = document.myform.customer.value
    document.cookie = "name = " & cookievalue
```

```

        document.cookie = "expires = " & y
        msgbox("Setting Cookies : " & "name=" & cookievalue )
    End Function
</script>
</head>
<body>
<form name="myform" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()" />
</form>
</body>
</html>

```

Deleting a Cookie:

Sometimes, you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiration date to a time in the past.

Example:

The following example illustrates how to delete a cookie by setting expiration date 1 Month in the past:

```

<html>
<head>
<script type="text/vbscript">
Function WriteCookie()
    x = now()
    x = now()
    a = Month(x) - 1
    b = day(x)
    c = year(x)
    d = DateSerial(c, a, b)
    e = hour(x)
    msgbox e
    f = minute(x)
    msgbox f
    d = cdate(d & " " & e & ":" & f)
    msgbox d
    cookievalue = document.myform.customer.value
    document.cookie = "name = " & cookievalue
    document.cookie = "expires = " & d
    msgbox("Setting Cookies : " & "name=" & cookievalue )
End Function
</script>
</head>
<body>
<form name="myform" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()" />
</form>
</body>
</html>

```

VBScript – Numbers

Description

Number functions help the developers to handle numbers in an efficient way and also helps them to convert their subtypes. It also helps them to make use of the inbuilt mathematical functions associated with VBScript.

NUMBER CONVERSION FUNCTIONS

Number functions help us to convert a given number from one data subtype to another data subtype.

Function	Description
CDbl	A Function, which converts a given number of any variant subtype to double
CInt	A Function, which converts a given number of any variant subtype to Integer
CLng	A Function, which converts a given number of any variant subtype to Long
CSng	A Function, which converts a given number of any variant subtype to Single
Hex	A Function, which converts a given number of any variant subtype to Hexadecimal

Syntax :

```
Variable_name = Conversion_function_name(expression)
```

The following table shows the possible number conversion functions in VBScript. Number Functions help the developers to handle numbers in an efficient way and also helps them to convert the given input to their sub types.

Function	Description
CDbl	A Function, which converts a given number of any variant subtype to double
CInt	A Function, which converts a given number of any variant subtype to Integer

CLng	A Function, which converts a given number of any variant subtype to Long
CSng	A Function, which converts a given number of any variant subtype to Single
Hex	A Function, which converts a given number of any variant subtype to Hexadecimal

Example

Try the following example to understand all the Number Conversion Functions available in VBScript.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

x = 123
y = 123.882

document.write("x value after converting to double - " & CDb1(x) & "<br />")
document.write("y value after converting to double - " & CDb1(y) & "<br />")
document.write("x value after converting to Int -" & CInt(x) & "<br />")
document.write("y value after converting to Int -" & CInt(y) & "<br />")
document.write("x value after converting to Long -" & CLng(x) & "<br />")
document.write("y value after converting to Long -" & CLng(y) & "<br />")
document.write("x value after converting to Single -" & CSng(x) & "<br />")
document.write("y value after converting to Single -" & CSng(y) & "<br />")
document.write("x value after converting to Hex -" & Hex(x) & "<br />")
document.write("y value after converting to Hex -" & Hex(y) & "<br />")
</script>
</body>
</html>
```

When executed the above script, following is the output:

```
x value after converting to double - 123
y value after converting to double - 123.882
x value after converting to Int -123
y value after converting to Int -124
x value after converting to Long -123
y value after converting to Long -124
x value after converting to Single -123
y value after converting to Single -123.882
x value after converting to Hex -7B
y value after converting to Hex -7C
```

NUMBER FORMATING FUNCTIONS

The Number formatting functions help the developers to express the given number in a format that they wish to.

Function	Description
FormatNumber	A Function, which would return an expression formatted as a number
FormatPercent	A Function, which would return an expression formatted as a percentage

Syntax :

```
variablename = Format_function_Name(Expression[,NumberDigAfterDec[,LeadingDig[,  
UseParForNegNum[,GroupDigits]]]])
```

Description

1. The Required parameter **Format_function_Name** corresponds to any of the below listed number formatting functions.
2. The Optional parameter **Expression** corresponds to any numerical expression, which would result in a number.
3. The Optional parameter **NumberDigAfterDec** corresponds to the number of digits after the decimal place.
4. The Optional parameter **LeadingDig** corresponds to whether or not a leading zero is displayed for fractional values. It takes one of the three values based on the below settings parameter.
5. The Optional parameter **UseParForNegNum** corresponds to whether or not to place negative values within parentheses. It takes one of the three values based on the below settings parameter.
6. The Optional parameter **GroupDigits** corresponds to whether or not numbers are grouped using the group delimiter. It takes one of the three values based on the below settings parameter.

Settings

The above parameters LeadingDig, UseParForNegNum and GroupDigits arguments can have any of the following settings:

- -2 = vbUseDefault - Use the computer's regional settings
- -1 = vbTrue - True
- 0 = vbFalse - False

The following table shows the possible number formatting functions in VBScript. Number Formatting Functions help the developers to handle numbers in an efficient way and express the numbers in the desired format.

Function	Description
FormatNumber	A Function, which would return an expression formatted as a number
FormatPercent	A Function, which would return an expression formatted as a percentage

Example :

Try the following example to understand all the Number Formatting Functions available in VBScript.

```
<!DOCTYPE html>  
<html>  
<body>  
<script language="vbscript" type="text/vbscript">  
  
Dim num : num = -645.998651  
  
document.write("Line 1 : " & FormatNumber(num, 3))& "<br/>"  
  
' The UseParensForNegativeNumbers parameter is set to true.  
document.write("Line 2 : " & FormatNumber (num, 3, , vbTrue))&" <br/> "  
  
' The GroupDigits parameter is set to false.  
document.write("Line 3 : " & FormatNumber (num, 3, , , vbFalse)) & "<br/>"
```

```

document.write("Line 4 : " & FormatPercent(num, 3))& "<br/>"

' The UseParensForNegativeNumbers parameter is set to true.
document.write("Line 5 : " & FormatPercent (num, 3, , vbTrue))&" <br/> "

' The GroupDigits parameter is set to false.
document.write("Line 6 : " & FormatPercent (num, 3, , , vbFalse)) & "<br/>"

</script>
</body>
</html>

```

When executed the above script, following is the output:

```

Line 1 : -645.999
Line 2 : (645.999)
Line 3 : -645.999
Line 4 : -64,599.865%
Line 5 : (64,599.865%)
Line 6 : -64599.865%

```

MATHEMATICAL FUNCTIONS

Mathematical Functions help us to evaluate the mathematical and trigonometrical functions of a given input number.

Function	Description
Int	A Function, which returns the integer part of the given number
Fix	A Function, which returns the integer part of the given number
Log	A Function, which returns the natural logarithm of the given number. Negative numbers disallowed
Oct	A Function, which returns the Octal value of the given percentage
Hex	A Function, which returns the Hexadecimal value of the given number
Rnd	A Function, which returns a random number between 0 and 1
Sgn	A Function, which returns a number corresponding to the sign of the specified number
Sqr	A Function, which returns the square root of the given number. Negative numbers disallowed
Abs	A Function, which returns the absolute value of the given number
Exp	A Function, which returns the value of e raised to the specified number
Sin	A Function, which returns sine value of the given number
Cos	A Function, which returns cosine value of the given number
Tan	A Function, which returns tan value of the given number

Syntax :

```
variablename = Mathematical_function_Name(Expression)
```

Description

Mathematical Functions help us to evaluate the mathematical and trigonometrical functions of a given input number.

Function	Description
Int	A Function, which returns the integer part of the given number
Fix	A Function, which returns the integer part of the given number
Log	A Function, which returns the natural logarithm of the given number. Negative numbers disallowed
Oct	A Function, which returns the Octal value of the given percentage
Hex	A Function, which returns the Hexadecimal value of the given number
Rnd	A Function, which returns a random number between 0 and 1
Sgn	A Function, which returns a number corresponding to the sign of the specified number
Sqr	A Function, which returns the square root of the given number. Negative numbers disallowed
Abs	A Function, which returns the absolute value of the given number
Exp	A Function, which returns the value of e raised to the specified number
Sin	A Function, which returns sine value of the given number
Cos	A Function, which returns cosine value of the given number
Tan	A Function, which returns tan value of the given number

Example

Try the following example to understand all the inbuilt Mathematical Functions available in VBScript.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

Dim num1 : num1 = -645.998651
Dim num2 : num2 = 210

document.write("int Result of num1 is : " & int(num1)) & "<br/>"
document.write("int Result of num2 is : " & int(num2)) & "<br/>"

document.write("Fix Result of num1 is : " & Fix(num1)) & "<br/>"
document.write("Fix Result of num2 is : " & Fix(num2)) & "<br/>"

document.write("Log Result of num2 is : " & Log(num2)) & "<br/>"

document.write("Oct Result of num1 is : " & Oct(num1)) & "<br/>"
document.write("Oct Result of num2 is : " & Oct(num2)) & "<br/>"
```



```

document.write("Hex Result of num1 is : " & Hex(num1)) & "<br/>"
document.write("Hex Result of num2 is : " & Hex(num2)) & "<br/>"

document.write("Rnd Result of num1 is : " & Rnd(num1)) & "<br/>"
document.write("Rnd Result of num2 is : " & Rnd(num2)) & "<br/>"

document.write("Sgn Result of num1 is : " & Sgn(num1)) & "<br/>"
document.write("Sgn Result of num2 is : " & Sgn(num2)) & "<br/>"

document.write("Sqr Result of num2 is : " & Sqr(num2)) & "<br/>"

document.write("Abs Result of num1 is : " & Abs(num1)) & "<br/>"
document.write("Abs Result of num2 is : " & Abs(num2)) & "<br/>"

document.write("Exp Result of num1 is : " & Exp(num1)) & "<br/>"
document.write("Exp Result of num2 is : " & Exp(num2)) & "<br/>"

document.write("Sin Result of num1 is : " & Sin(num1)) & "<br/>"
document.write("Sin Result of num2 is : " & Sin(num2)) & "<br/>"

document.write("Cos Result of num1 is : " & Cos(num1)) & "<br/>"
document.write("Cos Result of num2 is : " & Cos(num2)) & "<br/>"

document.write("Tan Result of num1 is : " & Tan(num1)) & "<br/>"
document.write("Tan Result of num2 is : " & Tan(num2)) & "<br/>"
</script>
</body>
</html>

```

When executed the above script, following is the output:

```

int Result of num1 is : -646
int Result of num2 is : 210
Fix Result of num1 is : -645
Fix Result of num2 is : 210
Log Result of num2 is : 5.34710753071747
Oct Result of num1 is : 37777776572
Oct Result of num2 is : 322
Hex Result of num1 is : FFFFFD7A
Hex Result of num2 is : D2
Rnd Result of num1 is : 0.5130115
Rnd Result of num2 is : 0.5615935
Sgn Result of num1 is : -1
Sgn Result of num2 is : 1
Sqr Result of num2 is : 14.4913767461894
Abs Result of num1 is : 645.998651
Abs Result of num2 is : 210
Exp Result of num1 is : 2.79479883633128E-281
Exp Result of num2 is : 1.59162664037792E+91
Sin Result of num1 is : 0.920530264916375
Sin Result of num2 is : 0.467718518342759
Cos Result of num1 is : 0.390671257418547
Cos Result of num2 is : -0.883877473182372
Tan Result of num1 is : 2.35627845006822
Tan Result of num2 is : -0.529166691689464

```

VBScript - Strings

Strings are a sequence of characters, which can consist of alphabets or numbers or special characters or all of them. A variable is said to be a string if it is enclosed within double quotes " ".

Syntax :

```
variablename = "string"
```

Examples :

```
str1 = "string"    ' Only Alphabets
str2 = "132.45"    ' Only Numbers
str3 = "!@#;$,*"   ' Only Special Characters
Str4 = "Asc23@#"   ' Has all the above
```

String Functions :

There are predefined VBScript String functions, which help the developers to work with the strings very effectively. Below are String methods that are supported in VBScript. Please click on each one of the methods to know in detail.

Function Name	Description
InStr	Returns the first occurrence of the specified substring. Search happens from left to right.
InStrRev	Returns the first occurrence of the specified substring. Search happens from Right to Left.
Lcase	Returns the lower case of the specified string.
Ucase	Returns the Upper case of the specified string.
Left	Returns a specific number of characters from the left side of the string.
Right	Returns a specific number of characters from the Right side of the string.
Mid	Returns a specific number of characters from a string based on the specified parameters.

<u>Ltrim</u>	Returns a string after removing the spaces on the left side of the specified string.
<u>Rtrim</u>	Returns a string after removing the spaces on the right side of the specified string.
<u>Trim</u>	Returns a string value after removing both leading and trailing blank spaces.
<u>Len</u>	Returns the length of the given string.
<u>Replace</u>	Returns a string after replacing a string with another string.
<u>Space</u>	Fills a string with the specified number of spaces.
<u>StrComp</u>	Returns an integer value after comparing the two specified strings.
<u>String</u>	Returns a String with a specified character the specified number of times.
<u>StrReverse</u>	Returns a String after reversing the sequence of the characters of the given string.

InStr

The InStr Function returns the first occurrence of one string within another string. The search happens from left to right.

Syntax :

```
InStr([start,]string1,string2[,compare])
```

Description :

- Start, an Optional Parameter. Specifies the starting position for the search. The search begins at the first position from left to right.
- String1, a Required Parameter. String to be searched.
- String2, a Required Parameter. String against which String1 is searched.
- Compare, an Optional Parameter. Specifies the String Comparison to be used. It can take the below mentioned values:
 - 0 = vbBinaryCompare - Performs Binary Comparison(Default)
 - 1 = vbTextCompare - Performs Text Comparison

Example:

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var="Microsoft VBScript"
document.write("Line 1 : " & InStr(1,var,"s") & "<br />")
```

```

document.write("Line 2 : " & InStr(7,var,"s") & "<br />")
document.write("Line 3 : " & InStr(1,var,"f",1) & "<br />")
document.write("Line 4 : " & InStr(1,var,"t",0) & "<br />")
document.write("Line 5 : " & InStr(1,var,"i") & "<br />")
document.write("Line 6 : " & InStr(7,var,"i") & "<br />")
document.write("Line 7 : " & InStr(var,"VB"))

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```

Line 1 : 6
Line 2 : 0
Line 3 : 8
Line 4 : 9
Line 5 : 2
Line 6 : 16
Line 7 : 11

```

InstrRev

The InStrRev Function returns the first occurrence of one string within another string. The Search happens from right to Left.

Syntax:

```
InStrRev(string1,string2[,start,[compare]])
```

Description :

- String1, a Required Parameter. String to be searched.
- String2, a Required Parameter. String against which String1 is searched.
- Start, an Optional Parameter. Specifies the Starting position for the search. The Search begins at the first position from right to left.
- Compare, an Optional Parameter. Specifies the String Comparison to be used. It can take the below mentioned values:
 - 0 = vbBinaryCompare - Performs Binary Comparison(Default)
 - 1 = vbTextCompare - Performs Text Comparison

Example:

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

```

```

var="Microsoft VBScript"
document.write("Line 1 : " & InStrRev(var,"s",10) & "<br />")
document.write("Line 2 : " & InStrRev(var,"s",7) & "<br />")
document.write("Line 3 : " & InStrRev(var,"f",-1,1) & "<br />")
document.write("Line 4 : " & InStrRev(var,"t",5) & "<br />")
document.write("Line 5 : " & InStrRev(var,"i",7) & "<br />")
document.write("Line 6 : " & InStrRev(var,"i",7) & "<br />")
document.write("Line 7 : " & InStrRev(var,"VB",1))

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```

Line 1 : 6
Line 2 : 6
Line 3 : 8
Line 4 : 0
Line 5 : 2
Line 6 : 2
Line 7 : 0

```

Lcase

The LCase Function returns the string after converting the entered string into lower case letters.

Syntax:

```
Lcase(String)
```

Example:

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var="Microsoft VBScript"
document.write("Line 1 : " & LCase(var) & "<br />")
var="MS VBSCRIPT"
document.write("Line 2 : " & LCase(var) & "<br />")
var="microsoft"
document.write("Line 3 : " & LCase(var) & "<br />")

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```

Line 1 : microsoft vbscript
Line 2 : ms vbscript

```

```
Line 3 : microsoft
```

Ucase

The UCase Function returns the string after converting the entered string into UPPER case letters.

Syntax:

```
UCase(String)
```

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var="Microsoft VBScript"
document.write("Line 1 : " & UCase(var) & "<br />")
var="MS VBSCRIPT"
document.write("Line 2 : " & UCase(var) & "<br />")
var="microsoft"
document.write("Line 3 : " & UCase(var) & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : MICROSOFT VBSCRIPT
Line 2 : MS VBSCRIPT
Line 3 : MICROSOFT
```

Left

The Left Function returns a specified number of characters from the left side of the given input string.

Syntax:

```
Left(String, Length)
```

- String, a Required Parameter. Input String from which the specified number of characters to be returned from left side.
- Length, a Required Parameter. An Integer, which specifies the number of characters to be returned.

Example:

```
<!DOCTYPE html>
```

```

<html>
<body>
<script language="vbscript" type="text/vbscript">

var="Microsoft VBScript"
document.write("Line 1 : " & Left(var,2) & "<br />")
var="MS VBSCRIPT"
document.write("Line 2 : " & Left(var,5) & "<br />")
var="microsoft"
document.write("Line 3 : " & Left(var,9) & "<br />")

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```

Line 1 : Mi
Line 2 : MS VB
Line 3 : microsoft

```

Right

The Right Function returns a specified number of characters from the right side of the given input string.

Syntax :

```
Right(String, Length)
```

- String, a Required Parameter. Input String from which the specified number of characters to be returned from Right side.
- Length, a Required Parameter. An Integer, which Specifies the number of characters to be returned.

Example:

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var="Microsoft VBScript"
document.write("Line 1 : " & Right(var,2) & "<br />")
var="MS VBSCRIPT"
document.write("Line 2 : " & Right(var,5) & "<br />")
var="microsoft"
document.write("Line 3 : " & Right(var,9) & "<br />")

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : pt
Line 2 : CRIPT
Line 3 : microsoft
```

Mid

The Mid Function returns a specified number of characters from a given input string.

Syntax:

```
Mid(String, start[, Length])
```

- String, a Required Parameter. Input String from which the specified number of characters to be returned.
- Start, a Required Parameter. An Integer, which Specifies starting position of the string .
- Length, an Optional Parameter. An Integer, which specifies the number of characters to be returned.

Example:

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var="Microsoft VBScript"
document.write("Line 1 : " & Mid(var,2) & "<br />")
document.write("Line 2 : " & Mid(var,2,5) & "<br />")
document.write("Line 3 : " & Mid(var,5,7) & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : icrosoft VBScript
Line 2 : icros
Line 3 : osoft V
```

Ltrim

The Ltrim Function removes the blank spaces that are there on the left side of the string.

Syntax:

```
LTrim(String)
```

Example:

```
<!DOCTYPE html>
```



```

<html>
<body>
<script language="vbscript" type="text/vbscript">

var =          "          Microsoft VBScript"
document.write("After Ltrim : " & LTrim(var) & "<br />")

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
After Ltrim : Microsoft VBScript
```

Rtrim

The Rtrim Function removes the blank spaces that are there on the Right side of the string.

Syntax:

```
RTrim(String)
```

Example:

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var =          "Microsoft VBScript          "
document.write("After Rtrim : " & RTrim(var) & "<br />")

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
After Rtrim : Microsoft VBScript
```

Trim

The Trim Function removes both the Leading and Trailing blank spaces of the given input string.

Syntax :

```
Trim(String)
```

Example :

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var =          "          Microsoft VBScript          "
document.write("After Trim : " & Trim(var) & "<br />")

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
After trim : Microsoft VBScript
```

Len

The Len Function returns the length of the given input string including the blank spaces.

Syntax :

```
Len(String)
```

Example :

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var1 =          "Microsoft VBScript"
document.write("Length of var1 is : " & Len(var1) & "<br />")

var2 =          "          Microsoft VBScript          "
document.write("Length of var2 is : " & Len(var2) & "<br />")

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Length of var1 is : 18
Length of var2 is : 36
```

Replace

The Replace Function replaces a specified part of a string with a specific string a specified number of times.

Syntax :

```
Replace(string, find, replacewith[, start[, count[, compare]])
```

- string, a Required Parameter. The Input String from to be searched for replacing.
- find, a Required Parameter. The Part of the String that will be replaced.
- replacewith, a Required Parameter. The replacement string, which would be replaced against the find parameter.
- start, an Optional Parameter. Specifies the start position from where the string has to be searched and replaced. Default value is 1.
- count, an Optional Parameter. Specifies the number of times the replacement has to be performed.
- compare, an Optional Parameter. Specifies the comparison method to be used. Default value is 0.
 - 0 = vbBinaryCompare - Performs a binary comparison
 - 1 = vbTextCompare - Performs a Textual comparison

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var="This is VBScript Programming"

'VBScript to be replaced by MS VBScript
document.write("Line 1: " & Replace(var, "VBScript", "MS VBScript") & "<br />")

'VB to be replaced by vb
document.write("Line 2: " & Replace(var, "VB", "vb") & "<br />")

'is' replaced by ##
document.write("Line 3: " & Replace(var, "is", "##") & "<br />")

'is' replaced by ## ignores the characters before the first occurrence
document.write("Line 4: " & Replace(var, "is", "##", 5) & "<br />")

's' is replaced by ## for the next 2 occurrences.
document.write("Line 5: " & Replace(var, "s", "##", 1, 2) & "<br />")

'r' is replaced by ## for all occurrences textual comparison.
document.write("Line 6: " & Replace(var, "r", "##", 1, -1, 1) & "<br />")

't' is replaced by ## for all occurrences Binary comparison
document.write("Line 7: " & Replace(var, "t", "##", 1, -1, 0) & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1: This is MS VBScript Programming
Line 2: This is vbScript Programming
Line 3: Th## ## VBScript Programming
Line 4: ## VBScript Programming
Line 5: Thi## i## VBScript Programming
Line 6: This is VBSc##ipt P##og##amming
Line 7: This is VBScrip## Programming
```

Space

The Space Function fills a string with a specific number of spaces.

Syntax :

```
space(number)
```

- number, a Required Parameter. The number of spaces that we want to add to the given string.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

var1="Microsoft"
var2="VBScript"
document.write(var1 & Space(2) & var2)

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Microsoft VBScript
```

StrComp

The StrComp Function returns an integer value after comparing the two given strings. It can return any of the three values -1, 0 or 1 based on the input strings to be compared.

- If String 1 < String 2 then StrComp returns -1
- If String 1 = String 2 then StrComp returns 0
- If String 1 > String 2 then StrComp returns 1

Syntax :

```
StrComp(string1,string2[,compare])
```

Description :

- String1, a Required Parameter. The first String expression.
- String2, a Required Parameter. The second String expression.
- Compare, an Optional Parameter. Specifies the String Comparison to be used. It can take the below-mentioned values:
 - 0 = vbBinaryCompare - Performs Binary Comparison(Default)
 - 1 = vbTextCompare - Performs Text Comparison

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write("Line 1 :" & StrComp("Microsoft","Microsoft") & "<br />")
document.write("Line 2 :" & StrComp("Microsoft","MICROSOFT") & "<br />")
document.write("Line 3 :" & StrComp("Microsoft","MiCrOsOfT") & "<br />")
document.write("Line 4 :" & StrComp("Microsoft","MiCrOsOfT",1) & "<br />")
document.write("Line 5 :" & StrComp("Microsoft","MiCrOsOfT",0) & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 :0
Line 2 :1
Line 3 :1
Line 4 :0
Line 5 :1
```

String

The String Function fills a string with the specified character the specified number of times.

Syntax :

```
String(number,character)
```

- Number, a Required Parameter. An integer value, which would be repeated for the specified number of times against the character parameter.

- Character, a Required Parameter. Character value, which has to be repeated for the specified number of times.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write("Line 1 : " & String(3,"$") & "<br />")
document.write("Line 2 : " & String(4,"*") & "<br />")
document.write("Line 3 : " & String(5,100) & "<br />")
document.write("Line 4 : " & String(6,"ABCDE") & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : $$$
Line 2 : ****
Line 3 : dddddd
Line 4 : AAAAAA
```

StrReverse

The StrReverse Function reverses the specified string.

Syntax :

```
StrReverse(string)
```

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write("Line 1 : " & StrReverse("VBSCRIPT") & "<br />")
document.write("Line 2 : " & StrReverse("My First VBScript") & "<br />")
document.write("Line 3 : " & StrReverse("123.45") & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : TPIRCSBV
Line 2 : tpirCSBV tsriF yM
Line 3 : 54.321
```

VBScript – Arrays

What is an Array?

We know very well that a variable is a container to store a value. Sometimes, developers are in a position to hold more than one value in a single variable at a time. When a series of values are stored in a single variable, then it is known as array variable.

Array Declaration:

Arrays are declared the same way a variable has been declared except that the declaration of an array variable uses paranthesis. In the below example, the size of the array is mentioned in the brackets.

```
'Method 1 : Using Dim
Dim arr1()           'Without Size

'Method 2 : Mentioning the Size
Dim arr2(5)         'Declared with size of 5

'Method 3 : using 'Array' Parameter
Dim arr3
arr3 = Array("apple","Orange","Grapes")
```

1. Although, the Array size is indicated as 5, it can hold 6 values as array index starts from ZERO.
2. Array Index Cannot be Negative.
3. VBScript Arrays can store any type of variable in an array. Hence, an array can store an integer, string or characters in a single array variable.

Assigning Values to an Array :

The values are assigned to the array by specifying array index value against each one of the values to be assigned. It can be a string.

Example :

```
<!DOCTYPE html>
```

```

<html>
<body>
<script language="vbscript" type="text/vbscript">

Dim arr(5)
arr(0) = "1"           'Number as String
arr(1) = "VBScript"    'String
arr(2) = 100            'Number
arr(3) = 2.45           'Decimal Number
arr(4) = #10/07/2013#   'Date
arr(5) = #12.45 PM#     'Time

document.write("Value stored in Array index 0 : " & arr(0) & "<br />")
document.write("Value stored in Array index 1 : " & arr(1) & "<br />")
document.write("Value stored in Array index 2 : " & arr(2) & "<br />")
document.write("Value stored in Array index 3 : " & arr(3) & "<br />")
document.write("Value stored in Array index 4 : " & arr(4) & "<br />")
document.write("Value stored in Array index 5 : " & arr(5) & "<br />")

</script>
</body>
</html>

```

When the above code is saved as .HTML and executed in Internet Explorer, it produces the following result:

```

Value stored in Array index 0 : 1
Value stored in Array index 1 : VBScript
Value stored in Array index 2 : 100
Value stored in Array index 3 : 2.45
Value stored in Array index 4 : 7/10/2013
Value stored in Array index 5 : 12:45:00 PM

```

Multi Dimension Arrays

Arrays are not just limited to single dimension and can have a maximum of 60 dimensions. Two-dimension arrays are the most commonly used ones.

Example :

In the below example, a multi-dimension array is declared with 3 rows and 4 columns.

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

Dim arr(2,3)          ' Which has 3 rows and 4 columns
arr(0,0) = "Apple"
arr(0,1) = "Orange"
arr(0,2) = "Grapes"
arr(0,3) = "pineapple"
arr(1,0) = "cucumber"
arr(1,1) = "beans"
arr(1,2) = "carrot"
arr(1,3) = "tomato"
arr(2,0) = "potato"
arr(2,1) = "sandwitch"
arr(2,2) = "coffee"

```



```
arr(2,3) = "nuts"

document.write("Value in Array index 0,1 : " & arr(0,1) & "<br />")
document.write("Value in Array index 2,2 : " & arr(2,2) & "<br />")

</script>
</body>
</html>
```

When the above code is saved as .HTML and executed in Internet Explorer, it produces the following result:

```
Value stored in Array index : 0 , 1 : Orange
Value stored in Array index : 2 , 2 : coffee
```

Redim Statement

ReDim Statement is used to Declare dynamic-array variables and allocate or reallocate storage space.

```
Redim [Preserve] varname(subscripts) [, varname(subscripts)]
```

- Preserve - An Optional parameter used to preserve the data in an existing array when you change the size of the last dimension.
- varname - A Required parameter, which denotes Name of the variable, which should follow the standard variable naming conventions.
- subscripts - A Required parameter, which indicates the size of the array.

Example :

In the below example, an array has been redefined and then preserved the values when the existing size of the array is changed.

Note : Upon resizing an array smaller than it was originally, the data in the eliminated elements will be lost.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim a()
    i=0
    redim a(5)
    a(0)="XYZ"
    a(1)=41.25
    a(2)=22

    REDIM PRESERVE a(7)
    For i=3 to 7
        a(i)= i
    Next

    'to Fetch the output
    For i=0 to ubound(a)
        MsgBox a(i)
    Next
```

```
</script>
</body>
</html>
```

When we save the above script as HTML and execute it in Internet Explorer, it produces the following result.

```
XYZ
41.25
22
3
4
5
6
7
```

Array Methods :

There are various inbuilt functions within VBScript which help the developers to handle arrays effectively. All the methods that are used in conjunction with arrays are listed below. Please click on the method name to know in detail.

Function	Description
LBound	A Function, which returns an integer that corresponds to the smallest subscript of the given arrays.
UBound	A Function, which returns an integer that corresponds to the Largest subscript of the given arrays.
Split	A Function, which returns an array that contains a specified number of values. Splitted based on a Delimiter.
Join	A Function, which returns a String that contains a specified number of substrings in an array. This is an exact opposite function of Split Method.
Filter	A Function, which returns a zero based array that contains a subset of a string array based on a specific filter criteria.
IsArray	A Function, which returns a boolean value that indicates whether or not the input variable is an array.
Erase	A Function, which recovers the allocated memory for the array variables.

LBound

The LBound Function returns the smallest subscript of the specified array. Hence, LBound of an array is ZERO.

Syntax :

```
LBound(ArrayName[,dimension])
```

1. ArrayName, a Required parameter. This parameter corresponds to the name of the array.
2. dimension, an Optional Parameter. This takes an integer value that corresponds to the dimension of the array. If it is '1', then it returns the lower bound of the first dimension; if it is '2', then it returns the lower bound of the second dimension and so on.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

Dim arr(5)
arr(0) = "1"           'Number as String
arr(1) = "VBScript"    'String
arr(2) = 100            'Number
arr(3) = 2.45           'Decimal Number
arr(4) = #10/07/2013#   'Date
arr(5) = #12.45 PM#     'Time

document.write("The smallest Subscript value of the given array is : " &
LBound(arr))

' For MultiDimension Arrays :
Dim arr2(3,2)
document.write("The smallest Subscript of the first dimension of arr2 is : " &
LBound(arr2,1) & "<br />")
document.write("The smallest Subscript of the Second dimension of arr2 is : " &
LBound(arr2,2) & "<br />")

</script>
</body>
</html>
```

When the above code is saved as .html and executed in Internet Explorer, it produces the following result:

```
The smallest Subscript value of the given array is : 0
The smallest Subscript of the first dimension of arr2 is : 0
The smallest Subscript of the Second dimension of arr2 is : 0
```

UBound

The UBound Function returns the Largest subscript of the specified array. Hence, this value corresponds to the size of the array.

Syntax :

```
UBound(ArrayName[,dimension])
```

1. ArrayName, a Required parameter. This parameter corresponds to the name of the array.
2. dimension, an Optional Parameter. This takes an integer value that corresponds to dimension of the array. If it is '1', then it returns the lower bound of the first dimension; if it is '2', then it returns the lower bound of the second dimension, and so on.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
```

```

Dim arr(5)
arr(0) = "1"           'Number as String
arr(1) = "VBScript"    'String
arr(2) = 100           'Number
arr(3) = 2.45          'Decimal Number
arr(4) = #10/07/2013#  'Date
arr(5) = #12.45 PM#    'Time

document.write("The Largest Subscript value of the given array is : " &
UBound(arr))

' For MultiDimension Arrays :
Dim arr2(3,2)
document.write("The Largest Subscript of the first dimension of arr2 is : " &
UBound(arr2,1) & "<br />")
document.write("The Largest Subscript of the Second dimension of arr2 is : " &
UBound(arr2,2) & "<br />")

</script>
</body>
</html>

```

When the above code is saved as .HTML and executed in Internet Explorer, then it produces the following result:

```

The Largest Subscript value of the given array is : 5
The Largest Subscript of the first dimension of arr2 is : 3
The Largest Subscript of the Second dimension of arr2 is : 2

```

Split

A Split Function returns an array that contains a specific number of values splitted based on a Delimiter.

Syntax :

```
Split(expression[,delimiter[,count[,compare]]])
```

1. expression, a Required parameter. The String Expression that can contain strings with delimiters.
2. delimiter, an Optional Parameter. The Parameter, which is used to convert into arrays based on a delimiter.
3. count, an Optional Parameter. The number of substrings to be returned, and if specified as -1, then all the substrings are returned.
4. compare, an Optional Parameter. This parameter specifies which comparison method to be used.
 - o 0 = vbBinaryCompare - Performs a binary comparison
 - o 1 = vbTextCompare - Performs a textual comparison

Example :

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

```

```

' Splitting based on delimiter comma '$'
a=Split("Red $ Blue $ Yellow","$")
b=ubound(a)
For i=0 to b
    document.write("The value of array in " & i & " is :" & a(i)& "<br />")
Next

</script>
</body>
</html>

```

When the above code is saved as .HTML and executed in Internet Explorer, it produces the following result:

```

The value of array in 0 is :Red
The value of array in 1 is : Blue
The value of array in 2 is : Yellow

```

Join

A Function, which returns a String that contains a specified number of substrings in an array. This is an exact opposite function of Split Method.

Syntax :

```
Join(List[,delimiter])
```

1. List, a Required parameter. An Array that contains the substrings that are to be joined.
2. delimiter, an Optional Parameter. The Character, which used as a delimiter while returning the string. The Default delimiter is Space.

Example :

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

' Join using spaces
a = array("Red","Blue","Yellow")
b = join(a)
document.write("The value of b " & " is :" & b & "<br />")

' Join using $
b = join(a,"$")
document.write("The Join result after using delimiter is : " & b & "<br />")

</script>
</body>
</html>

```

When the above code is saved as .html and executed in Internet Explorer, it produces the following result:

```

The value of b is :Red Blue Yellow
The Join result after using delimiter is : Red$Blue$Yellow

```

Filter

A Filter Function, which returns a zero-based array that contains a subset of a string array based on a specific filter criteria.

Syntax :

```
Filter(inputstrings,value[,include[,compare]])
```

1. inputstrings, a Required parameter. This parameter corresponds to the array of strings to be searched.
2. value, a Required Parameter. This parameter corresponds to the string to search for against the inputstrings parameter.
3. include, an Optional Parameter. This is a Boolean value, which indicates whether or not to return the substrings that include or exclude.
4. compare, an Optional Parameter. This Parameter describes what string comparison method to be used.
 - o 0 = vbBinaryCompare - Performs a binary comparison
 - o 1 = vbTextCompare - Performs a textual comparison

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

a= array("Red","Blue","Yellow")
b = Filter(a,"B")
c = Filter(a,"e")
d = Filter(a,"Y")

For each x in b
    Document.write("The Filter result 1: " & x & "<br />")
Next

For each y in c
    Document.write("The Filter result 2: " & y & "<br />")
Next

For each z in d
    Document.write("The Filter result 3: " & z & "<br />")
Next

</script>
</body>
</html>
```

When the above code is saved as .HTML and executed in Internet Explorer, it produces the following result:

```
The Filter result 1: Blue
The Filter result 2: Red
The Filter result 2: Blue
```

```
The Filter result 2: Yellow  
The Filter result 3: Yellow
```

IsArray

The IsArray Function returns a boolean value that indicates whether or NOT the specified input variable is an array variable.

Syntax :

```
IsArray(variablename)
```

Example :

```
<!DOCTYPE html>  
<html>  
<body>  
<script language="vbscript" type="text/vbscript">  
  
a = array("Red", "Blue", "Yellow")  
b = "12345"  
  
Document.write("The IsArray result 1 : " & IsArray(a) & "<br />")  
Document.write("The IsArray result 2 : " & IsArray(b) & "<br />")  
  
</script>  
</body>  
</html>
```

When the above code is saved as .HTML and executed in Internet Explorer, it produces the following result:

```
The IsArray result 1 : True  
The IsArray result 2 : False
```

Erase

The Erase Function is used to reset the values of fixed size arrays and free the memory of the dynamic arrays. It behaves depending upon the type of the arrays.

Syntax:

```
Erase ArrayName
```

1. Fixed numeric array, each element in an array is resetted to Zero.
2. Fixed String array, each element in an array is resetted to Zero length " ".
3. Array of Objects, each element in an array is resetted to special value Nothing.

Example :

```
<!DOCTYPE html>  
<html>  
<body>
```

```

<script language="vbscript" type="text/vbscript">

Dim NumArray(3)
NumArray(0) = "VBScript"
NumArray(1) = 1.05
NumArray(2) = 25
NumArray(3) = #23/04/2013#

Dim DynamicArray()
ReDim DynamicArray(9)    ' Allocate storage space.

Erase NumArray           ' Each element is reinitialized.
Erase DynamicArray       ' Free memory used by array.

' All values would be erased.
Document.write("The value at Zeroth index of NumArray is " & NumArray(0) & "<br />")
Document.write("The value at First index of NumArray is " & NumArray(1) & "<br />")
Document.write("The value at Second index of NumArray is " & NumArray(2) & "<br />")
Document.write("The value at Third index of NumArray is " & NumArray(3) & "<br />")

</script>
</body>
</html>

```

When the above code is saved as .HTML and executed in Internet Explorer, it produces the following result:

```

The value at Zeroth index of NumArray is
The value at First index of NumArray is
The value at Second index of NumArray is
The value at Third index of NumArray is

```


VBScript – Date and Time

V

BScript Date and Time Functions help the developers to convert date and time from one format to another or to express the date or time value in the format that suits a specific condition.

Date Functions

Function	Description
<u>Date</u>	A Function, which returns the current system date
<u>CDate</u>	A Function, which converts a given input to Date
<u>DateAdd</u>	A Function, which returns a date to which a specified time interval has been added
<u>DateDiff</u>	A Function, which returns the difference between two time period
<u>DatePart</u>	A Function, which returns a specified part of the given input date value
<u>DateSerial</u>	A Function, which returns a valid date for the given year,month and date
<u>FormatDateTime</u>	A Function, which formats the date based on the supplied parameters
<u>IsDate</u>	A Function, which returns a Boolean Value whether or not the supplied parameter is a date
<u>Day</u>	A Function, which returns an integer between 1 and 31 that represents the day of the specified Date
<u>Month</u>	A Function, which returns an integer between 1 and 12 that represents the month of the specified Date
<u>Year</u>	A Function, which returns an integer that represents the year of the specified Date
<u>MonthName</u>	A Function, which returns Name of the particular month for the specified date
<u>WeekDay</u>	A Function, which returns an integer(1 to 7) that represents the day of the week for the specified day.
<u>WeekDayName</u>	A Function, which returns the weekday name for the specified day.

Date

The Function returns the current system Date.

Syntax :

```
date()
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

  a = date()
  document.write("The Value of a : " & a)

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
The Value of a : 19/07/2013
```

CDate

The Function converts a valid date and time expression to type date.

Syntax :

```
cdate(date)
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

  a = cdate("Jan 01 2020")
  document.write("The Value of a : " & a)
  document.write("<br />")
  b = cdate("31 Dec 2050")
  document.write("The Value of b : " & b)

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
The Value of a : 1/01/2012  
The Value of b : 31/12/2050
```

DateAdd

A Function, which returns a date to which a specified time interval has been added.

Syntax :

```
DateAdd(interval,number,date)
```

Parameter Description :

1. Interval, a Required Parameter. It can take the following values:
 - o d - day of the year.
 - o m - month of the year
 - o y - year of the year
 - o yyyy - year
 - o w - weekday
 - o ww - week
 - o q - quarter
 - o h - hour
 - o m - minute
 - o s - second
2. Number, a Required parameter. It can take both positive and negative parameters.
3. Date, a Required parameter. A Variant or literal representing the date to which interval is added.

Example

```
<!DOCTYPE html>  
<html>  
<body>  
<script language="vbscript" type="text/vbscript">  
  
' Positive Interlal  
date1=01-Jan-2013  
document.write("Line 1 : " &DateAdd("yyyy",1,date1) & "<br />")  
document.write("Line 2 : " &DateAdd("q",1,date1) & "<br />")  
document.write("Line 3 : " &DateAdd("m",1,date1) & "<br />")  
document.write("Line 4 : " &DateAdd("y",1,date1) & "<br />")  
document.write("Line 5 : " &DateAdd("d",1,date1) & "<br />")  
document.write("Line 6 : " &DateAdd("w",1,date1) & "<br />")
```

```

document.write("Line 7 : " & DateAdd("ww",1,date1) & "<br />")
document.write("Line 8 : " & DateAdd("h",1,"01-Jan-2013 12:00:00") & "<br />")
document.write("Line 9 : " & DateAdd("n",1,"01-Jan-2013 12:00:00") & "<br />")
document.write("Line 10 : " & DateAdd("s",1,"01-Jan-2013 12:00:00") & "<br />")

' Negative Interval
document.write("Line 11 : " & DateAdd("yyyy",-1,date1) & "<br />")
document.write("Line 12 : " & DateAdd("q",-1,date1) & "<br />")
document.write("Line 13 : " & DateAdd("m",-1,date1) & "<br />")
document.write("Line 14 : " & DateAdd("y",-1,date1) & "<br />")
document.write("Line 15 : " & DateAdd("d",-1,date1) & "<br />")
document.write("Line 16 : " & DateAdd("w",-1,date1) & "<br />")
document.write("Line 17 : " & DateAdd("ww",-1,date1) & "<br />")
document.write("Line 18 : " & DateAdd("h",-1,"01-Jan-2013 12:00:00") & "<br />")
document.write("Line 19 : " & DateAdd("n",-1,"01-Jan-2013 12:00:00") & "<br />")
document.write("Line 20 : " & DateAdd("s",-1,"01-Jan-2013 12:00:00") & "<br />")
</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```

Line 1 : 27/06/1895
Line 2 : 27/09/1894
Line 3 : 27/07/1894
Line 4 : 28/06/1894
Line 5 : 28/06/1894
Line 6 : 28/06/1894
Line 7 : 4/07/1894
Line 8 : 1/01/2013 1:00:00 PM
Line 9 : 1/01/2013 12:01:00 PM
Line 10 : 1/01/2013 12:00:01 PM
Line 11 : 27/06/1893
Line 12 : 27/03/1894
Line 13 : 27/05/1894
Line 14 : 26/06/1894
Line 15 : 26/06/1894
Line 16 : 26/06/1894
Line 17 : 20/06/1894
Line 18 : 1/01/2013 11:00:00 AM
Line 19 : 1/01/2013 11:59:00 AM
Line 20 : 1/01/2013 11:59:59 AM

```

DateDiff

A Function, which returns the difference between two specified time intervals.

Syntax :

```
DateDiff(interval, date1, date2 [,firstdayofweek[, firstweekofyear]])
```

Parameter Description :

- Interval, a Required Parameter. It can take the following values:

- d - day of the year.
- m - month of the year
- y - year of the year
- yyyy - year
- w - weekday
- ww - week
- q - quarter
- h - hour
- m - minute
- s - second
- date1 and date2 are Required parameters.
- firstdayofweek is Optional. Specifies the first day of the week. It can take the following values:
 - 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting
 - 1 = vbSunday - Sunday
 - 2 = vbMonday - Monday
 - 3 = vbTuesday - Tuesday
 - 4 = vbWednesday - Wednesday
 - 5 = vbThursday - Thursday
 - 6 = vbFriday - Friday
 - 7 = vbSaturday - Saturday
- firstdayofyear is Optional. Specifies the first day of the year. It can take the following values:
 - 0 = vbUseSystem - Use National Language Support (NLS) API setting
 - 1 = vbFirstJan1 - Start with the week in which January 1 occurs (default)
 - 2 = vbFirstFourDays - Start with the week that has at least four days in the new year
 - 3 = vbFirstFullWeek - Start with the first full week of the new year

Example :

```
<!DOCTYPE html>
<html>
```

```

<body>
<script language="vbscript" type="text/vbscript">

fromDate="01-Jan-09 00:00:00"
toDate="01-Jan-10 23:59:00"
document.write("Line 1 : " & DateDiff("yyyy",fromDate,toDate) & "<br />")
document.write("Line 2 : " & DateDiff("q",fromDate,toDate) & "<br />")
document.write("Line 3 : " & DateDiff("m",fromDate,toDate) & "<br />")
document.write("Line 4 : " & DateDiff("y",fromDate,toDate) & "<br />")
document.write("Line 5 : " & DateDiff("d",fromDate,toDate) & "<br />")
document.write("Line 6 : " & DateDiff("w",fromDate,toDate) & "<br />")
document.write("Line 7 : " & DateDiff("ww",fromDate,toDate) & "<br />")
document.write("Line 8 : " & DateDiff("h",fromDate,toDate) & "<br />")
document.write("Line 9 : " & DateDiff("n",fromDate,toDate) & "<br />")
document.write("Line 10 : " & DateDiff("s",fromDate,toDate) & "<br />")

</script>
</body>
</html>

```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```

Line 1 : 1
Line 2 : 4
Line 3 : 12
Line 4 : 365
Line 5 : 365
Line 6 : 52
Line 7 : 52
Line 8 : 8783
Line 9 : 527039
Line 10 : 31622340

```

DatePart

A Function, which returns the specific part of the given date.

Syntax :

```
DatePart(interval,date[,firstdayofweek[,firstweekofyear]])
```

Parameter Description :

- Interval, a Required Parameter. It can take the following values:
 - d - day of the year.
 - m - month of the year
 - y - year of the year
 - yyyy - year
 - w - weekday

- ww - week
- q - quarter
- h - hour
- m - minute
- s - second
- date1 is a required parameter.
- firstdayofweek is Optional. Specifies the first day of the week. It can take the following values:
 - 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting
 - 1 = vbSunday - Sunday
 - 2 = vbMonday - Monday
 - 3 = vbTuesday - Tuesday
 - 4 = vbWednesday - Wednesday
 - 5 = vbThursday - Thursday
 - 6 = vbFriday - Friday
 - 7 = vbSaturday - Saturday
- firstdayofyear is Optional. Specifies the first day of the year. It can take the following values:
 - 0 = vbUseSystem - Use National Language Support (NLS) API setting
 - 1 = vbFirstJan1 - Start with the week in which January 1 occurs (default)
 - 2 = vbFirstFourDays - Start with the week that has at least four days in the new year
 - 3 = vbFirstFullWeek - Start with the first full week of the new year

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

Dim Quarter, DayOfYear, WeekOfYear

Date1 = "2013-01-15"
Quarter = DatePart("q", Date1)
document.write("Line 1 : " & Quarter & "<br />")
DayOfYear = DatePart("y", Date1)
document.write("Line 2 : " & DayOfYear & "<br />")
WeekOfYear = DatePart("ww", Date1)
```

```
document.write("Line 3 : " & WeekOfYear("<br />")
document.write("Line 4 : " & DatePart("m",Date1))

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : 1
Line 2 : 15
Line 3 : 3
Line 4 : 1
```

DateSerial

A Function, which returns a date for the specified day, month and year parameters.

Syntax :

```
DateSerial(year,month,day)
```

Parameter Description :

- year, a Required Parameter. A number between 100 and 9999 or a numeric expression. Values between 0 and 99 are interpreted as the years 1900 to 1999. For all other year arguments, use a complete four-digit year.
- month, a Required Parameter. It can also be in the form of an expression, which should range from 1 to 12.
- day, a Required Parameter. It can also be in the form of an expression, which should range from 1 to 31.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    document.write(DateSerial(2013,5,10))

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Fri May 10 00:00:00 UTC+0530 2013
```


FormatDateTime

A Function, which helps the developers to format and return a valid date and time expression.

Syntax :

```
FormatDateTime (date, format)
```

Parameter Description :

- date, a Required Parameter.
- format, an Optional Parameter. The Value that specifies the date or time format to be used. It can take the following values:
 - 0 = vbGeneralDate - Default.
 - 1 = vbLongDate - Returns date.
 - 2 = vbShortDate - Returns date.
 - 3 = vbLongTime - Returns time.
 - 4 = vbShortTime - Returns time.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

d=("2013-08-15 20:25")
document.write("Line 1 : " & FormatDateTime(d) & " <br />")
document.write("Line 2 : " & FormatDateTime(d,1) & "<br />")
document.write("Line 3 : " & FormatDateTime(d,2) & "<br />")
document.write("Line 4 : " & FormatDateTime(d,3) & "<br />")
document.write("Line 5 : " & FormatDateTime(d,4) & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : 15/08/2013 8:25:00 PM
Line 2 : Thursday, 15 August 2013
Line 3 : 15/08/2013
Line 4 : 8:25:00 PM
Line 5 : 20:25
```

IsDate

A Function, which returns a Boolean Value whether or Not the given input is a date.

Syntax :

```
IsDate(expression)
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write("Line 1 : " & IsDate("Nov 03, 1950") & "<br />")
document.write("Line 2 : " & IsDate("#01/31/20#") & "<br />")
document.write("Line 3 : " & IsDate("#05/31/20 10:30 PM#") & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : True
Line 2 : True
Line 3 : True
```

Day

The Day function returns a number between 1 and 31 that represents the day of the specified date.

Syntax :

```
Day(date)
```

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write(Day("2013-06-30"))

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

Month

The Month function returns a number between 1 and 12 that represents the month of the specified date.

Syntax :

```
Month(date)
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    document.write(Month("2013-06-30"))

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

6

Year

The Year function returns an integer that represents an year of the specified date.

Syntax :

```
Year(date)
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    document.write(Year("2013-06-30"))

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

MonthName

The MonthName function returns the name of the month for the specified date.

Syntax :

```
MonthName(month[,toabbreviate])
```

Parameter Description :

- Month, a Required Parameter. It specifies the number of the month.
- toabbreviate, an Optional Parameter. A Boolean value that indicates if the month name is to be abbreviated. If left blank, the default value would be taken as False.

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write("Line 1 : " & MonthName(01,True) & "<br />")
document.write("Line 2 : " & MonthName(01,false) & "<br />")

document.write("Line 3 : " & MonthName(07,True) & "<br />")
document.write("Line 4 : " & MonthName(07,false) & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : Jan
Line 2 : January
Line 3 : Jul
Line 4 : July
```

WeekDay

The WeekDay function returns an integer from 1 to 7 that represents the day of the week for the specified date.

Syntax :

```
Weekday(date[,firstdayofweek])
```

Parameter Description :

- Date, a Required Parameter. The Week day would be returns for this specified date.
- firstdayofweek, an Optional Parameter. Specifies the first day of the week..
 - 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting
 - 1 = vbSunday - Sunday
 - 2 = vbMonday - Monday
 - 3 = vbTuesday - Tuesday
 - 4 = vbWednesday - Wednesday
 - 5 = vbThursday - Thursday
 - 6 = vbFriday - Friday
 - 7 = vbSaturday - Saturday

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write("Line 1: " & Weekday("2013-05-16",1) & "<br />")
document.write("Line 2: " & Weekday("2013-05-16",2) & "<br />")
document.write("Line 3: " & Weekday("2013-05-16",2) & "<br />")
document.write("Line 4: " & Weekday("2010-02-16") & "<br />")
document.write("Line 5: " & Weekday("2010-02-17") & "<br />")
document.write("Line 6: " & Weekday("2010-02-18") & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1: 5
Line 2: 4
Line 3: 4
Line 4: 3
Line 5: 4
Line 6: 5
```

WeekDayName

The WeekDayName function returns the name of the Weekday for the specified day.

Syntax :

```
WeekdayName(weekday[,abbreviate[,firstdayofweek]])
```

Parameter Description :

- weekday, a Required Parameter. The number of the weekday.
- toabbreviate, an Optional Parameter. A Boolean value Boolean value that indicates if the month name is to be abbreviated. If left blank, the default value would be taken as False.
- firstdayofweek, an Optional Parameter. Specifies the first day of the week.
 - 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting
 - 1 = vbSunday - Sunday
 - 2 = vbMonday - Monday
 - 3 = vbTuesday - Tuesday
 - 4 = vbWednesday - Wednesday
 - 5 = vbThursday - Thursday
 - 6 = vbFriday - Friday
 - 7 = vbSaturday - Saturday

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write("Line 1 : " & WeekdayName(3) & "<br />")
document.write("Line 2 : " & WeekdayName(2,True) & "<br />")
document.write("Line 3 : " & WeekdayName(1,False) & "<br />")
document.write("Line 4 : " & WeekdayName(2,True,0) & "<br />")
document.write("Line 5 : " & WeekdayName(1,False,1) & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1 : Tuesday
Line 2 : Mon
Line 3 : Sunday
Line 4 : Tue
Line 5 : Sunday
```

Time Functions

Function	Description
Now	A Function, which returns the current system date and Time
Hour	A Function, which returns and integer between 0 and 23 that represents the Hour part of the the given time
Minute	A Function, which returns and integer between 0 and 59 that represents the Minutes part of the the given time
Second	A Function, which returns and integer between 0 and 59 that represents the Seconds part of the the given time
Time	A Function, which returns the current system time
Timer	A Function, which returns the number of seconds and milliseconds since 12:00 AM
TimeSerial	A Function, which returns the time for the specific input of hour,minute and second
TimeValue	A Function, which converts the input string to a time format

Now

The Function Now returns the current system date and time.

Syntax :

```
Now()
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

  a = Now()
  document.write("The Value of a : " & a)

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
The Value of a : 19/07/2013 3:04:09 PM
```

Hour

The Hour Function returns a number between 0 and 23 that represents the hour of the day for the specified time stamp.

Syntax :

```
Hour(time)
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    document.write("Line 1: " & Hour("3:13:45 PM") & "<br />")
    document.write("Line 2: " & Hour("23:13:45") & "<br />")
    document.write("Line 3: " & Hour("2:20 PM") & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1: 15
Line 2: 23
Line 3: 14
```

Minute

The Minute Function returns a number between 0 and 59 that represents the Minute of the hour for the specified time stamp.

Syntax :

```
Minute(time)
```

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    document.write("Line 1: " & Minute("3:13:45 PM") & "<br />")
    document.write("Line 2: " & Minute("23:43:45") & "<br />")
    document.write("Line 3: " & Minute("2:20 PM") & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1: 13
```



```
Line 2: 43  
Line 3: 20
```

Second

The Second Function returns a number between 0 and 59 that represents the Second of the hour for the specified time stamp.

Syntax :

```
Second(time)
```

Example

```
<!DOCTYPE html>  
<html>  
<body>  
<script language="vbscript" type="text/vbscript">  
  
    document.write("Line 1: " & Second("3:13:25 PM") & "<br />")  
    document.write("Line 2: " & Second("23:13:45") & "<br />")  
    document.write("Line 3: " & Second("2:20 PM") & "<br />")  
  
</script>  
</body>  
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1: 25  
Line 2: 45  
Line 3: 0
```

Time

The Time Function returns the current system time.

Syntax :

```
Time()
```

Example

```
<!DOCTYPE html>  
<html>  
<body>  
<script language="vbscript" type="text/vbscript">  
  
    document.write("Line 1: " & Time() & "<br />")  
  
</script>  
</body>  
</html>
```

```
</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Line 1: 3:29:15 PM
```

Timer

The Timer Function returns the number of seconds and milliseconds since 12:00 AM.

Syntax :

```
Timer()
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    document.write("Time is : " & Now() & "<br />")
    document.write("Timer is: " & Timer())

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
Time is : 19/07/2013 3:45:53 PM
Timer is: 56753.4
```

TimeSerial

The TimeSerial function returns the time for the specified hour, minute and second values.

Syntax :

```
TimeSerial(hour,minute,second)
```

Parameter Description :

- Hour, a Required parameter, which is an integer between 0 and 23 or any numeric expression.
- Minute, a Required parameter, which is an integer between 0 and 59 or any numeric expression.

- Second, a Required parameter, which is an integer between 0 and 59 or any numeric expression.

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write(TimeSerial(20,1,2) & "<br />")
document.write(TimeSerial(0,59,59) & "<br />")
document.write(TimeSerial(7*2,60/3,15+3) & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
8:01:02 PM
12:59:59 AM
2:20:18 PM
```

TimeValue

The TimeValue Function converts the given input string to a valid time.

Syntax :

```
TimeValue(StringTime)
```

Example

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

document.write(TimeValue("20:30") & "<br />")
document.write(TimeValue("5:15") & "<br />")
document.write(TimeValue("2:30:58") & "<br />")

</script>
</body>
</html>
```

When you save it as .html and execute it in Internet Explorer, then the above script will produce the following result:

```
8:30:00 PM
5:15:00 AM
2:30:58 AM
```

VBScript – Procedures

What is a Function?

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing same code over and over again. This will enable programmers to divide a big program into a number of small and manageable functions.

Apart from inbuilt Functions, VBScript allows us to write user-defined functions as well. This section will explain you how to write your own functions in VBScript.

Function Definition:

Before we use a function, we need to define that particular function. The most common way to define a function in VBScript is by using the **Function** keyword, followed by a unique function name and it may or may not carry a list of parameters and a statement with a **End Function** keyword, which indicates the end of the function. The basic syntax is shown below:

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

Function Functionname(parameter-list)
    statement 1
    statement 2
    statement 3
    .....
    statement n
End Function

</script>
</body>
</html>
```

Example :

```
<!DOCTYPE html>
<html>
<body>
```

```

<script language="vbscript" type="text/vbscript">

    Function sayHello()
        msgbox("Hello there")
    End Function

</script>
</body>
</html>

```

Calling a Function :

To invoke a function somewhere later in the script, you would simple need to write the name of that function with the **Call** keyword.

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Function sayHello()
        msgbox("Hello there")
    End Function

    Call sayHello()

</script>
</body>
</html>

```

Function Parameters:

Till now, we have seen function without a parameter, but there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. The Functions are called using the **Call** Keyword.

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Function sayHello(name, age)
        msgbox( name & " is " & age & " years old.")
    End Function

    Call sayHello("Tutorials point", 7)

</script>
</body>
</html>

```

Returning a Value from a Function :

A VBScript function can have an optional return statement. This is required if you want to return a value from a function.

For example, you can pass two numbers in a function and then you can expect from the function to return their multiplication in your calling program.

NOTE : A function can return multiple values separated by comma as an array assigned to the function name itself.

Example :

This function takes two parameters and concatenates them and returns result in the calling program. In VBScript, the values are returned from a function using function name. In case if you want to return two or more values, then the function name is returned with an array of values. In the calling program, the result is stored in the result variable.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Function concatenate(first, last)
        Dim full
        full = first & last
        concatenate = full 'Returning the result to the function name itself
    End Function

</script>
</body>
</html>
```

Now, we can call this function as follows:

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
    Function concatenate(first, last)
        Dim full
        full = first & last
        concatenate = full 'Returning the result to the function name itself
    End Function
    ' Here is the usage of returning value from function.
    dim result
    result = concatenate("Zara", "Ali")
    msgbox(result)
</script>
</body>
</html>
```

Sub Procedures :

Sub Procedures are similar to functions but there are few differences.

- Sub procedures DONOT Return a value while functions may or may not return a value.
- Sub procedures Can be called without call keyword.
- Sub procedures are always enclosed within **Sub** and **End Sub** statements.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Sub sayHello()
        msgbox("Hello there")
    End Sub

</script>
</body>
</html>
```

Calling Procedures :

To invoke a Procedure somewhere later in the script, you would simply need to write the name of that procedure with or without the **Call** keyword.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Sub sayHello()
        msgbox("Hello there")
    End Sub
    sayHello()

</script>
</body>
</html>
```

Advanced Concepts for Functions

There is lot to learn about VBScript functions. We can pass the parameter byvalue or byreference. Please click on each one of them to know more.

- [ByVal- Pass the parameters by value](#)
- [ByRef- Pass the parameters by the reference](#)

ByVal-Pass the parameters by value

What are ByVal Parameters?

If ByVal is specified, then the arguments are sent as byvalue when the function or procedure is called.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
```

```

Function fnadd(Byval num1, Byval num2)
    num1 = 4
    num2 = 5
End Function

Dim x,y
x=6
y=4
res= fnadd(x,y)
document.write("The value of x is " & x & "<br />")
document.write("The value of y is " & y & "<br />")

</script>
</body>
</html>

```

The above function takes the parameter x and y as by values. Hence, after executing the function, the values are unchanged.

If the above function is saved as .html and executed in IE, the output would be as follows:

```

The value of x is 6
The value of y is 4

```

BRefl-Pass the parameters by the reference

What are ByRef Parameters?

If **ByRef** is specified, then the arguments are sent as a reference when the function or procedure is called.

Example :

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

Function fnadd(ByRef num1, ByRef num2)
    num1 = 4
    num2 = 5
End Function

Dim x,y
x=6
y=4
res= fnadd(x,y)
document.write("The value of x is " & x & "<br />")
document.write("The value of y is " & y & "<br />")

</script>
</body>
</html>

```

The above function takes the parameter x and y as by reference. Hence, after executing the function, the values are changed.

If the above function is saved as .html and executed in IE, the output would be as follows:


```
The value of x is 4  
The value of y is 5
```

VBScript – Dialog Boxes

What is a Dialog Box?

V

BScript allows the developers to interact with the user effectively. It can be a message box to display a

message to a user or an input box with which user can enter the values.

VBScript MsgBox Function

The MsgBox function displays a message box and waits for the user to click a button and then an action is performed based on the button clicked by the user.

Syntax :

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

Parameter Description :

- Prompt - A Required Parameter. A String that is displayed as a message in the dialog box. The maximum length of prompt is approximately 1024 characters. If the message extends to more than a line, then we can separate the lines using a carriage return character (Chr(13)) or a linefeed character (Chr(10)) between each line.
- buttons - An Optional Parameter. A Numeric expression that specifies the type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If left blank, the default value for buttons is 0.
- Title - An Optional Parameter. A String expression displayed in the title bar of the dialog box. If the title is left blank, the application name is placed in the title bar.
- helpfile - An Optional Parameter. A String expression that identifies the Help file to use to provide context-sensitive help for the dialog box.
- context - An Optional Parameter. A Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If context is provided, helpfile must also be provided.

The **Buttons** parameter can take any of the following values:

- 0 vbOKOnly Displays OK button only.
- 1 vbOKCancel Displays OK and Cancel buttons.
- 2 vbAbortRetryIgnore Displays Abort, Retry, and Ignore buttons.
- 3 vbYesNoCancel Displays Yes, No, and Cancel buttons.
- 4 vbYesNo Displays Yes and No buttons.
- 5 vbRetryCancel Displays Retry and Cancel buttons.
- 16 vbCritical Displays Critical Message icon.
- 32 vbQuestion Displays Warning Query icon.
- 48 vbExclamation Displays Warning Message icon.
- 64 vbInformation Displays Information Message icon.
- 0 vbDefaultButton1 First button is default.
- 256 vbDefaultButton2 Second button is default.
- 512 vbDefaultButton3 Third button is default.
- 768 vbDefaultButton4 Fourth button is default.
- 0 vbApplicationModal Application modal. The current application will not work until the user responds to the message box.
- 4096 vbSystemModal System modal. All applications will not work until the user responds to the message box.

The above values are logically divided into four groups: The first group(0 to 5) indicates the buttons to be displayed in the message box. The second group (16, 32, 48, 64) describes the style of the icon to be displayed, the third group (0, 256, 512, 768) indicates which button must be the default, and the fourth group (0, 4096) determines the modality of the message box.

RETURN VALUES :

The MsgBox function can return one of the following values:

- 1 - vbOK - OK was clicked
- 2 - vbCancel - Cancel was clicked
- 3 - vbAbort - Abort was clicked
- 4 - vbRetry - Retry was clicked
- 5 - vbIgnore - Ignore was clicked

- 6 - vbYes - Yes was clicked
- 7 - vbNo - No was clicked

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    'Message Box with just prompt message
    MsgBox("Welcome")

    'Message Box with title, yes no and cancel Buttons
    a = MsgBox("Do you like blue color?",3,"Choose options")
    ' Assume that you press No Button
    document.write("The Value of a is " & a)

</script>
</body>
</html>
```

When the above script is executed, the message box is displayed, and if you press No Button, then the value of a is 7.

The Value of a is 7

VBScript InputBox Function

The InputBox function helps the user to get the values from the user. After entering the values, if the user clicks the OK button or presses ENTER on the keyboard, the InputBox function will return the text in the text box. If the user clicks on the Cancel button, the function will return an empty string ("").

Syntax :

```
InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,context])
```

Parameter Description :

- Prompt - A Required Parameter. A String that is displayed as a message in the dialog box. The maximum length of prompt is approximately 1024 characters. If the message extends to more than a line, then we can separate the lines using a carriage return character (Chr(13)) or a linefeed character (Chr(10)) between each line.
- Title - An Optional Parameter. A String expression displayed in the title bar of the dialog box. If the title is left blank, the application name is placed in the title bar.
- Default - An Optional Parameter. A default text in the text box that the user would like to be displayed.
- XPos - An Optional Parameter. The Position of X axis which represents the prompt distance from left side of the screen horizontally. If left blank, the input box is horizontally centered.
- YPos - An Optional Parameter. The Position of Y axis which represents the prompt distance from left side of the screen Vertically. If left blank, the input box is Vertically centered.

- **helpfile** - An Optional Parameter. A String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box.
- **context** - An Optional Parameter. A Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If context is provided, helpfile must also be provided.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    ' Input Box with only Prompt
    InputBox("Enter a number")

    ' Input Box with a Title
    a=InputBox("Enter a Number","Enter Value")
    msgbox a

    ' Input Box with a Prompt, Title and Default value
    a=InputBox("Enter a Number","Enter Value",123)
    msgbox a

    ' Input Box with a Prompt, Title, Default and XPos
    a=InputBox("Enter your name","Enter Value",123,700)
    msgbox a

    ' Input Box with a Prompt, Title and Default and YPos
    a=InputBox("Enter your name","Enter Value",123,,500)
    msgbox a

</script>
</body>
</html>
```

When the above script is executed, the input box is displayed and displays the entered value by the user.

VBScript – Object Oriented

What is an Object

V

BScript runtime objects help us to accomplish various tasks. This section will help you understand how to instantiate an object and work with it.

SYNTAX :

Inorder to work with objects seamlessly, we need to declare the object and instantiate it using **Set** keyword.

```
Dim objectname      'Declare the object name
Set objectname = CreateObject(object_type)
```

EXAMPLE :

In the below example, we are creating an object of type **Scripting.Dictionary**.

```
Dim obj
Set obj = CreateObject("Scripting.Dictionary")
```

Destroying the Objects

The significance of destroying the Object is to free the memory and reset the object variable.

SYNTAX :

Inorder to destroy the objects, we need to use **Set** Keyword followed by the object name and point it to **Nothing**.

```
Set objectname = Nothing 'Destroy the object.
```

EXAMPLE :

In the below example, we are creating an object of type **Scripting.Dictionary**.

```
Dim obj
Set obj = CreateObject("Scripting.Dictionary")
Set obj = Nothing.
```

Object Usage :

Please click on each one of the given object types to know more.

Object Type	Description
Class	Class is a container, which holds methods and variables associated with it and accessed by creating an object of Type Class.
Scripting.FileSystemObject	It is the group of objects with which we can work with file system.
Scripting.Dictionary	A Group of objects, which are used for creating the dictionary objects.
Debug	A Global Object with which we can send output to the Microsoft script debugger.

CLASS

What is a Class?

Class is a construct that is used to define a unique type. Like Object Oriented Programming, VbScript 5.0 supports the creation of classes and it is very similar to writing COM objects with VB.

Class is simply the template for an object and we instantiate an object to access the properties and methods of it. Classes can contain variables, properties, methods or events.

SYNTAX :

VBScript classes are enclosed within **Class End Class**

```
'Defining the Class
Class classname      'Declare the object name
...
End Class

' Instantiation of the Class
Set objectname = new classname
```

CLASS VARIABLES :

Classes can contain variables, which can be of private or public. Variables within classes should follow VBScript naming conventions. By default, the variables in class are **Public**. That is why they can be accessed outside the class.

```
Dim var1 , var2.
Private var1 , var2.
Public var1 , var2.
```

Class Properties :

Class properties, such as Property Let, which handles the process of data validation and assigning the new value to the private variable. Property set, which assigns the new property value to the private object variable.

Read-only properties have only a Property Get procedure while write-only properties (which are rare) have only a Property Let or a Property Set procedure

EXAMPLE :

In the below example, we are using Properties to wrap private variables.

```
Class Comp

    Private modStrType
    Private OS

    Public Property Let ComputerType(strType)
        modStrType = strType
    End Property

    Public Property Get ComputerType()
        ComputerType = modStrType
    End Property

    Public Property Set OperatingSystem(oObj)
        Set OS = oObj
    End Property

    Public Property Get OperatingSystem()
        Set OperatingSystem = OS
    End Property

End Class
```

Class Methods :

Methods allow the class to perform the operation that the developer wants. The Methods are nothing but Functions or Subroutines.

EXAMPLE :

In the below example, we are using Properties to wrap private variables.

```
Class Car

    Private Model
    Private Year

    Public Start()
        Fuel = 2.45
        Pressure = 4.15
    End Function

End Class
```

Class Events :

There are two events that are automatically associated with every class by default. **Class_Initialize** and **Class_Terminate**.

Class_Initialize is triggered whenever you instantiate an object based on the class. **Class_Terminate** event is fired when the object goes out of scope or when the object is set to Nothing.

EXAMPLE :

In the below example, we will make you understand how the events work in VBScript.


```

'Instantation of the Object
Set objectname = New classname

Private Sub Class_Initialize( )
    Initialization code goes here
End Sub

'When Object is Set to Nothing
Private Sub Class_Terminate( )
    Termination code goes here
End Sub

```

SCRIPTING.FILESYSTEMOBJECT

What are FSO Objects?

As the name suggests, FSO Objects help the developers to work with drives, folders and files. In this section, we will discuss:

Objects and Collections :

Object Type	Description
Drive	Drive is an Object. Contains methods and properties that allow you to gather information about a drive attached to the system
Drives	Drives is a Collection. It Provides a list of the drives attached to the system, either physically or logically.
File	File is an Object. It Contains methods and properties that allow developers to create, delete or move a file.
Files	Files is a Collection. It Provides a list of all files contained within a folder.
Folder	Folder is an Object. It Provides methods and properties that allow developers to create, delete or move folders.
Folders	Folders is a Collection. It Provides a list of all the folders within a Folder.
TextStream	TextStream is an Object. It enables developers to read and write text files.

Drive

Drive is an object, which provides access to the properties of a particular disk drive or network share. The Following properties are supported by **Drive** object:

- AvailableSpace
- DriveLetter
- DriveType
- FileSystem
- FreeSpace
- IsReady

- Path
- RootFolder
- SerialNumber
- ShareName
- TotalSize
- VolumeName

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim oFS, drive, space
    Set oFS = CreateObject("Scripting.FileSystemObject")
    Set drive = oFS.GetDrive(oFS.GetDriveName("C:\"))
    space = "Drive " & UCase(drivePath) & " - "
    space = space & drive.VolumeName & " "
    space = space & "Free Space: " & FormatNumber(drive.FreeSpace/1024, 0)
    space = space & " Kbytes"
    Document.write space

</script>
</body>
</html>
```

If the above script is saved as HTML and executed in IE, we would get the following output in the console.

```
Drive - Win 7 Free Space:20,154,059 Kbytes
```

Drives

Drives is a collection, which provides details of all the drives attached to the system, either physically or logically. It carries two properties:

- Count Property
- Item Property

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim oFS, d, dc, s, n
    Set oFS = CreateObject("Scripting.FileSystemObject")
    Set dc = oFS.Drives
    For Each d in dc
```

```

        n = ""
        s = s & d.DriveLetter & " - "
        If d.DriveType = 3 Then
            n = d.ShareName
        ElseIf d.IsReady Then
            n = d.VolumeName
        Else
            n = "Drive not available"
        End If
        s = s & n & " :: "
    Next

    document.write s
    document.write dc.count

</script>
</body>
</html>

```

If the above script is saved as HTML and executed in IE, we would get the following output in the console.

```

C- Win 7 :: D- Personal :: E- Common :: F- Songs ::
4

```

File

File is an Object, which contains both properties and methods that allow the developers to create, delete or move a file.

METHODS :

- Copy
- Delete
- Move
- openasTextStream

PROPERTIES :

- Attributes
- DateCreated
- DateLastAccessed
- DateLastModified
- Drive
- Name
- ParentFolder

- Path
- ShortName
- ShortPath
- Size
- Type

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim fso, f
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFile("C:\user.js")
    document.write "Line 1: "& f.DateCreated & "<br />"
    document.write "Line 2: "& f.Attributes & "<br />"
    document.write "Line 3: "& f.DateLastAccessed & "<br />"
    document.write "Line 4: "& f.DateLastModified & "<br />"
    document.write "Line 5: "& f.Drive & "<br />"
    document.write "Line 6: "& f.Name & "<br />"
    document.write "Line 7: "& f.ParentFolder & "<br />"
    document.write "Line 8: "& f.Path & "<br />"
    document.write "Line 9: "& f.ShortName & "<br />"
    document.write "Line 10: "& f.ShortPath & "<br />"
    document.write "Line 11: "& f.Size & "<br />"
    document.write "Line 12: "& f.Type & "<br />"
    f.copy ("D:\") & "<br />"      'copying to file to another location'
    f.Move ("E:\") & "<br />"      'Move the file to another location'
    f.Delete ("D:\") & "<br />"    'Delete to file from one location'

</script>
</body>
</html>
```

If the above script is saved as HTML and executed in IE, we would get the following output in the console.

```
Line 1: 08/02/13 06:57:34
Line 2: 32
Line 3: 08/02/13 06:57:34
Line 4: 04/18/12 22:23:37
Line 5: C:
Line 6: user.js
Line 7: C:\
Line 8: C:\user.js
Line 9: user.js
Line 10: C:\user.js
Line 11: 474
Line 12: JScript Script File
```

Files

Files is a collection, which provides a list of all files contained within a folder.

PROPERTIES :

- Count
- Item

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim fso, f, fl, fc, s
    Set oFS = CreateObject("Scripting.FileSystemObject")

    'get the folder by giving its path
    Set f = oFS.GetFolder("D:\PROJECT\")
    Set fc = f.Files

    'Get Item
    Set s = fc.Item("sendmail.vbs")

    'Get Count
    x = fc.Count

    Document.write s
    Document.write x

</script>
</body>
</html>
```

If the above script is saved as HTML and executed in IE, we would get the following output in the console.

```
D:\PROJECT\sendmail.vbs
6
```

Folder

Folder is an Object, which contains both properties and methods that allow the developers to create, delete or move a folder.

METHODS :

- Copy
- Delete
- Move
- CreateTextFile

PROPERTIES :

- Attributes
- DateCreated
- DateLastAccessed
- DateLastModified
- Drive
- Files
- IsRootFolder
- Name
- ParentFolder
- Path
- ShortName
- ShortPath
- Size
- SubFolders
- Type

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim fso, f
    Set fso = CreateObject("Scripting.FileSystemObject")

    ' Enter a Folder Name that exists on your System'
    Set f = fso.GetFolder("D:\PROJECT\")

    ShowFileInfo = "Created: " & f.DateCreated & "<br / >"
    document.write ShowFileInfo

    ShowFileInfo = "attributes " & f.attributes & "<br / >"
    document.write ShowFileInfo

    ShowFileInfo = "Last Accessed : " & f.DateLastAccessed & "<br / >"
    document.write ShowFileInfo

    ShowFileInfo = "DateLastModified : " & f.DateLastModified & "<br / >"
```

```

document.write ShowFileInfo

ShowFileInfo = "Drive : " & f.Drive & "<br / >"
document.write ShowFileInfo

ShowFileInfo = "count : " & f.Files.count & "<br / >"
document.write ShowFileInfo

ShowFileInfo = "IsRoot folder : " & f.IsRootFolder & "<br / >"
document.write ShowFileInfo

ShowFileInfo = "Name : " & f.Name & "<br / >"
document.write ShowFileInfo

ShowFileInfo = "parent folder : " & f.ParentFolder & "<br / >"
document.write ShowFileInfo

ShowFileInfo = "Path : " & f.Path & "<br / >"
document.write ShowFileInfo

ShowFileInfo = "shortname : " & f.ShortName & "<br / >"
document.write ShowFileInfo

ShowFileInfo = "ShortPath : " & f.ShortPath & "<br / >"
document.write ShowFileInfo

ShowFileInfo = "File Size : " & f.Size & "<br / >"
document.write ShowFileInfo

ShowFileInfo = "Type : " & f.Type & "<br / >"
document.write ShowFileInfo

</script>
</body>
</html>

```

If the above script is saved as HTML and executed in IE, we would get the following output in the console.

```

Created: 22/02/2012 8:24:57 PM
attributes 16
Last Accessed : 1/08/2013 12:48:36 PM
DateLastModified : 1/08/2013 12:48:36 PM
Drive : D:
count : 6
IsRoot folder : False
Name : PROJECT
parent folder : D:\
Path : D:\PROJECT
shortname : PROJECT
ShortPath : D:\PROJECT
File Size : 8655239975
Type : File folder

```

Folders

Folders is an collection of all Folder Objects within a Folder object.

METHODS :

- Add

PROPERTIES :

- Count
- Item

EXAMPLE :

If the above script is saved as HTML and executed in IE, we would create a folder with name "Test_Folder".

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim fso, f, fc, nf
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set f = fso.GetFolder("D:\PROJECT")
    Set fc = f.SubFolders
    folderName = "Test_Folder"
    If folderName <> "" Then
        Set nf = fc.Add(folderName)
    Else
        Set nf = fc.Add("New Folder")
    End If

</script>
</body>
</html>
```

TextStream

TextStream object helps the developers to work with text files seamlessly. Developers can read, write or append the contents to the text file using the text stream object.

SYNTAX :

```
TextStream.{property | method( )}
```

EXAMPLE :

If the above script is saved as HTML and executed in IE, we would create a folder with name "Test_Folder".

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim objFSO
    Set objFSO = CreateObject("Scripting.FileSystemObject")
    Dim objTextFile
    Set objTextFile = objFSO.CreateTextFile("D:\Testfile.txt")
    objTextFile.Close
```

TUTORIALS POINT

Simply Easy Learning


```

Const ForAppending = 8
Set objTextFile = objFSO.OpenTextFile("D:\Testfile.txt",ForAppending,True)
objTextFile.WriteLine "Welcome to VBScript Programming"
objTextFile.Close
Set objTextFile = Nothing
Set objFSO = Nothing

</script>
</body>
</html>

```

If the above script is saved as HTML and executed in IE, it will create a text file in D:\ Drive and append the string specified in the WriteLine Method.

```
Welcome to VBScript Programming
```

SCRIPTING.DICTIONARY

What are Dictionary Objects?

A Dictionary object can be compared to a PERL associative array. Any Values can be stored in the array and each item is associated with a unique key. The key is used to retrieve an individual element and it is usually an integer or a string, but can be anything except an array.

Syntax:

VBScript classes are enclosed within **Class End Class**.

```

Dim variablename
Set variablename = CreateObject("Scripting.Dictionary")
variablename.Add (key, item)

```

Example :

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim obj_datadict    ' Create a variable.
    Set obj_datadict = CreateObject("Scripting.Dictionary")
    obj_datadict.Add "a", "Apple"    ' Add some keys and items.
    obj_datadict.Add "b", "Bluetooth"
    obj_datadict.Add "c", "Clear"

</script>
</body>
</html>

```

There are various methods associated with DataDictionary Objects which enable the developers to work with dictionary objects seamlessly.

Exists Method :

Exist Method helps the user to check whether or not the Key Value pair exists.

```
object.Exists(key)
```

Parameter Description :

- **Object**, a Mandatory Parameter. This represents the name of the Dictionary Object.
- **Key**, a Mandatory Parameter. This represents the value of the Dictionary Object.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim d, msg    ' Create some variables.
    Set d = CreateObject("Scripting.Dictionary")
    d.Add "a", "Apple"    ' Add some keys and items.
    d.Add "b", "BlueTooth"
    d.Add "c", "C++"
    If d.Exists("c") Then
        msgbox "Specified key exists."
    Else
        msgbox "Specified key doesn't exist."
    End If

</script>
</body>
</html>
```

Save the file as .HTML, and upon executing the above script in IE, it displays the following message in a message box.

```
Specified key exists.
```

Items Method :

Items Method helps us to get the values stored in the key value pair of the data dictionary object.

```
object.Items( )
```

Parameter Description :

- **Object**, a Mandatory Parameter. This represents the name of the Dictionary Object.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim obj_datadict    ' Create a variable.
    Set obj_datadict = CreateObject("Scripting.Dictionary")
    obj_datadict.Add "a", "Apple"    ' Add some keys and items.
    obj_datadict.Add "b", "Bluetooth"
    obj_datadict.Add "c", "C++"
    a=obj_datadict.items

</script>
</body>
</html>
```

```
msgbox a(0)
msgbox a(2)

</script>
</body>
</html>
```

Save the file as .HTML, and upon executing the above script in IE, it displays the following message in a message box.

```
Apple
C++
```

Keys Method :

```
object.Keys ( )
```

Parameter Description :

- **Object**, a Mandatory Parameter. This represents the name of the Dictionary Object.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim obj_datadict    ' Create a variable.
    Set obj_datadict = CreateObject("Scripting.Dictionary")
    obj_datadict.Add "a", "Apple"    ' Add some keys and items.
    obj_datadict.Add "b", "Bluetooth"
    obj_datadict.Add "c", "C++"
    a=obj_datadict.Keys

    msgbox a(0)
    msgbox a(2)

</script>
</body>
</html>
```

Save the file as .HTML, and upon executing the above script in IE, it displays the following message in a message box.

```
a
c
```

Remove Method :

```
object.Remove (key)
```

Parameter Description :

- **Object** , a Mandatory Parameter. This represents the name of the Dictionary Object.
- **Key** , a Mandatory Parameter. This represents the key value pair that needs to be removed from the Dictionary Object.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim obj_datadict    ' Create a variable.
    Set obj_datadict = CreateObject("Scripting.Dictionary")
    obj_datadict.Add "a", "Apple"    ' Add some keys and items.
    obj_datadict.Add "b", "Bluetooth"
    obj_datadict.Add "c", "C++"
    a=obj_datadict.Keys

    msgbox a(0)
    msgbox a(2)

    obj_datadict.remove("b")    'The key value pair of "b" is removed'

</script>
</body>
</html>
```

Save the file as .HTML, and upon executing the above script in IE, it displays the following message in a message box.

```
a
c
```

Remove All Method :

```
object.RemoveAll()
```

Parameter Description :

- **Object**, a Mandatory Parameter. This represents the name of the Dictionary Object.

Example :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim obj_datadict    ' Create a variable.
    Set obj_datadict = CreateObject("Scripting.Dictionary")
    obj_datadict.Add "a", "Apple"    ' Add some keys and items.
    obj_datadict.Add "b", "Bluetooth"
    obj_datadict.Add "c", "C++"
    a=obj_datadict.Keys

    msgbox a(0)
    msgbox a(2)

    obj_datadict.removeall

</script>
</body>
```

```
</html>
```

DEBUG

What are Debug Objects?

The Debug Objects are global objects that can send output to a script debugger. Here, the debugger what we refer to is Microsoft Script Debugger.

The **Debug** objects cannot be created like other objects but can be used when we are debugging. The following methods are supported by **Debug** Objects. These methods or objects have no effect if the script is NOT executed in debug mode. The Methods supported by Debug Objects are discussed in detail.

Write

The Write method sends strings to the immediate window of the Microsoft Script Debugger at run-time. If the script is not executed in debug mode, then the Write method has no effect.

```
Write Debug.Write([str1 [, str2 [, ... [, strN]]]])
```

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim counter
    counter = 42
    Debug.Write "The value of counter is " & counter

</script>
</body>
</html>
```

WriteLine

The WriteLine method is very similar to Write method. The WriteLine method sends strings, followed by a newline character, to the immediate window of the Microsoft Script Debugger at run time. If the script is not executed in debug mode, then the WriteLine method has no effect.

```
Debug.WriteLine([str1 [, str2 [, ... [, strN]]]])
```

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim counter
    counter = 42
    Debug.WriteLine "The value of counter is " & counter

</script>
</body>
</html>
```

Enabling Debug Mode :

To enable script in debug mode, following actions to be performed by the user:

1. On the **Tools** menu, click Internet Options.
2. In the Internet Options dialog box, click the Advanced tab.
3. In the Browsing category, clear the Disable script debugging check box.
4. Click OK.
5. Exit and restart Internet Explorer.

VBScript – Reg Expressions

What are Regular Expressions?

Regular Expressions is a sequence of characters that forms a pattern, which is mainly used for search and replace. The purpose of creating a pattern is to match specific strings, so that the developer can extract characters based on conditions and replace certain characters.

RegExp Object

RegExp object helps the developers to match the pattern of strings and the properties and methods help us to work with Regular Expressions easily. It is similar to RegExp in JavaScript

Properties :

- **Pattern** - The Pattern property represents a string that is used to define the regular expression and it should be set before using the regular expression object.
- **IgnoreCase** - A Boolean property that represents if the regular expression should be tested against all possible matches in a string if true or false. If not specified explicitly, IgnoreCase value is set to False.
- **Global** - A Boolean property that represents if the regular expression should be tested against all possible matches in a string. If not specified explicitly, Global value is set to False.

Methods :

- **Test** (search-string) - The Test method takes a string as its argument and returns True if the regular expression can successfully be matched against the string, otherwise False is returned.
- **Replace** (search-string, replace-string) - The Replace method takes 2 parameters. If the search is successful then it replaces that match with the replace-string, and the new string is returned. If there are no matches then the original search-string is returned.
- **Execute** (search-string) - The Execute method works like Replace, except that it returns a Matches collection object, containing a Match object for each successful match. It doesn't modify the original string.

Matches Collection Object

The Matches collection object is returned as a result of the Execute method. This collection object can contain zero or more Match objects and the properties of this object are read-only.

- **Count** - The Count property represents the number of match objects in the collection.

- **Item** - The Item method enables the match objects to be accessed from matches collections object.

Match Object

The Match object is contained within the matches collection object. These objects represent the successful match after the search for a string.

- **FirstIndex** - It represents the position within the original string where the match occurred. This index are zero-based which means that the first position in a string is 0.
- **Length** - A value that represents the total length of the matched string.
- **Value** - A value that represents the matched value or text. It is also the default value when accessing the Match object.

All about Pattern Parameter:

The pattern building is similar to PERL. Pattern building is the most important thing while working with Regular Expressions. In this section, we will deal with how to create a pattern based on various factors.

POSITION MATCHING:

The significance of position matching is to ensure that we place the regular expressions at the correct places.

Symbol	Description
^	Matches only the beginning of a string.
\$	Match only the end of a string.
\b	Matches any word boundary
\B	Matches any non-word boundary

LITERALS MATCHING :

Any form of characters such as alphabet, number or special character or even decimal, hexadecimal can be treated as a Literal. Since few of the characters have already got a special meaning within the context of Regular Expression, we need to escape them using escape sequences.

Symbol	Description
Alphanumeric	Matches alphabetical and numerical characters only.
\n	Matches a new line.
\[Matches [literal only
\]	Matches] literal only
\(Matches (literal only
\)	Matches) literal only
\t	Matches horizontal tab
\v	Matches vertical tab

\	Matches literal only
\{	Matches { literal only
\}	Matches } literal only
\\	Matches \ literal only
\?	Matches ? literal only
*	Matches * literal only
\+	Matches + literal only
\.	Matches . literal only
\b	Matches any word boundary
\B	Matches any non-word boundary
\f	Matches a form feed
\r	Matches carriage return
\xxx	Matches the ASCII character of an octal number xxx.
\xdd	Matches the ASCII character of an hexadecimal number dd.
\uxxxx	Matches the ASCII character of an UNICODE literal xxxx.

CHARACTER CLASSES MATCHING :

The character classes are the Pattern formed by customized grouping and enclosed within [] braces. If we are expecting a character class that should not be in the list, then we should ignore that particular character class using the negative symbol, which is a cap ^.

Symbol	Description
[xyz]	Match any of the character class enclosed within the character set.
[^xyz]	Matches any of the character class that are NOT enclosed within the character set.
.	Matches any character class except \n
\w	Match any word character class. Equivalent to [a-zA-Z_0-9]
\W	Match any non-word character class. Equivalent to [^a-zA-Z_0-9]
\d	Match any digit class. Equivalent to [0-9].
\D	Match any non-digit character class. Equivalent to [^0-9].
\s	Match any space character class. Equivalent to [\t\r\n\v\f]
\S	Match any space character class. Equivalent to [^\t\r\n\v\f]

REPETITION MATCHING :

Repetition matching allows multiple searches within the regular expression. It also specifies the number of times an element is repeated in a Regular Expression.

Symbol	Description
*	Matches zero or more occurrences of the given regular Expression. Equivalent to {0,}.
+	Matches one or more occurrences of the given regular Expression. Equivalent to {1,}.
?	Matches zero or one occurrences of the given regular Expression. Equivalent to {0,1}.
{x}	Matches exactly x number of occurrences of the given regular expression.
{x,}	Match atleast x or more occurrences of the given regular expression.
{x,y}	Matches x to y number of occurrences of the given regular expression.

ALTERNATION & GROUPING :

Alternation and grouping helps developers to create more complex Regular Expressions in particularly handling intricate clauses within a Regular Expression which gives a great flexibility and control.

Symbol	Description
()	Grouping a clause to create a clause. "(xy)?(z)" matches "xyz" or "z".
	Alternation combines one regular expression clause and then matches any of the individual clauses. "(ij) (23) (pq)" matches "ij" or "23" or "pq".

Building Regular Expressions

Below are few examples, which clearly explain on how to build a Regular Expression.

Regular Expression	Description
"^\\s*.." and "..\\s*\$"	Represents that there can be any number of leading and trailing space characters in a single line.
"((\\\$ s?))(#s?)?"	Represents an optional \$ or # sign followed by an optional space.
"((\\d+(\\.\\d\\d)?))"	Represents that at least one digit is present followed by an optional decimals and two digits after decimals.

Example :

The below example checks whether or not the user entered an email id whose format should match such that there is an email id followed by '@' and then followed by domain name.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
    strid = "welcome.user@tutorialspoint.co.us"
    Set re = New RegExp
    With re
        .Pattern = "^\\w-\\.]{1,}\\@([\\da-zA-Z-]{1,}\\.){1,}[\\da-zA-Z-]{2,3}$"
```

```
        .IgnoreCase = False
        .Global       = False
    End With

    ' Test method returns TRUE if a match is found
    If re.Test( strid ) Then
        Document.write(strid & " is a valid e-mail address")
    Else
        Document.write(strid & " is NOT a valid e-mail address")
    End If

    Set re = Nothing
</script>
</body>
</html>
```

VBScript – Error Handling

There are three types of errors in programming: (a) Syntax Errors and (b) Runtime Errors (c) Logical Errors.

Syntax errors :

Syntax errors, also called parsing errors, occur at interpretation time for VBScript. For example, the following line causes a syntax error because it is missing a closing parenthesis:

```
<script type="text/vbscript">

dim x,y
x = "Tutorialspoint"
y = Ucase(x

</script>
```

Runtime errors:

Runtime errors, also called exceptions, occur during execution, after interpretation.

For example, the following line causes a runtime error because here syntax is correct but at runtime it is trying to call `fnmultiply`, which is a non-existing function:

```
<script type="text/vbscript">

Dim x,y
x = 10
y = 20
z = fnadd(x,y)
a = fnmultiply(x,y)
Function fnadd(x,y)
    fnadd = x+y
End Function

</script>
```

Logical errors:

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You can not catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

For example, dividing a number by zero or a script that is written which enters into infinite loop.

Err Object

Assume if we have a runtime error, then the execution stops by displaying the error message. As a developer, if we want to capture the error, then **Error** Object is used.

EXAMPLE :

In the below example, **Err.Number** gives the error number and **Err.Description** gives error description.

```
<script type="text/vbscript">

    Err.Raise 6      ' Raise an overflow error.
    MsgBox "Error # " & CStr(Err.Number) & " " & Err.Description
    Err.Clear        ' Clear the error.

</script>
```

VBScript – Misc Statements

There are few other important statements which helps the developers to develop an efficient script. Below are the list of statements tabulated and explained in detail with examples.

Category	Function Name/Statement Name
Options	Option Explicit
Script Engine ID	ScriptEngine
Variants	IsArray, IsEmpty, IsNull, IsNumeric, IsObject, TypeName
Expression	Eval, Execute
Control Statement	With...End With
Math Function	Randomize

Option Explicit

Option Explicit forces the developer to declare the variables using **Dim** statement before they are used in some part of the code.

SYNTAX :

```
Option Explicit
```

EXAMPLE :

If we use **Option Explicit** and if we don't declare the variables then the interpreter will throw an error.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Option Explicit
    Dim x,y,z,a
```

```

x = 10
y = 20
z = fnadd(x,y)
a = fnmultiply(x,y)

Function fnadd(x,y)
    fnadd = x+y
End Function

</script>
</body>
</html>

```

ScriptEngine

ScriptEngine represents the details of the scripting language in use. It is also used in combination with **ScriptEngineMajorVersion**, **ScriptEngineMinorVersion**, **ScriptEngineBuildVersion** which gives the major version of the vbscript engine, minor version the vbscript engine and the build version of vbscript respectively.

SYNTAX :

```
ScriptEngine
```

EXAMPLE :

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim scriptdetails
    scriptdetails = " Version " & ScriptEngine & " - "
    'For getting Major version, use ScriptEngineMajorVersion'

    scriptdetails = scriptdetails & ScriptEngineMajorVersion & "."

    'For getting Minor version, use ScriptEngineMinorVersion'
    scriptdetails = scriptdetails & ScriptEngineMinorVersion & "."

    'For getting Build version, use ScriptEngineBuildVersion'
    scriptdetails = scriptdetails & ScriptEngineBuildVersion

    Document.write scriptdetails

</script>
</body>
</html>

```

Save the file with .html extension upon executing the script in IE , the following result is displayed on the screen.

```
Version VBScript - 5.8.16996
```

IsEmpty

The Function IsEmpty is used to check whether or not the expression is empty. It returns a boolean value. **IsEmpty** returns True if the variable is uninitialized or explicitly set to Empty. Otherwise the expression returns False.

SYNTAX :

```
IsEmpty(expression)
```

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim var, MyCheck
    MyCheck = IsEmpty(var)
    Document.write "Line 1 : " & MyCheck & "<br />"

    var = Null    ' Assign Null.
    MyCheck = IsEmpty(var)
    Document.write "Line 2 : " & MyCheck & "<br />"

    var = Empty   ' Assign Empty.
    MyCheck = IsEmpty(var)
    Document.write "Line 3 : " & MyCheck & "<br />"

</script>
</body>
</html>
```

Save the file with .html extension upon executing the script in IE , the following result is displayed on the screen.

```
Line 1 : True
Line 2 : False
Line 3 : True
```

IsNull

The Function IsNull is used to check whether or not the expression has a valid data. It returns a boolean value. **IsNull** returns True if the variable is Null otherwise the expression returns False.

SYNTAX :

```
IsNull(expression)
```

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim var, res
```



```

    res = IsNull(var)
    document.write "Line 1 : " & res & "<br />"

    var = Null
    res = IsNull(var)
    document.write "Line 2 : " & res & "<br />"

    var = Empty
    res = IsNull(var)
    document.write "Line 3 : " & res & "<br />"

</script>
</body>
</html>

```

Save the file with .html extension upon executing the script in IE , the following result is displayed on the screen.

```

Line 1 : False
Line 2 : True
Line 3 : False

```

IsObject

The IsObject Function is used to check whether or not the expression has a valid Object. It returns a boolean value. **IsObject** returns True if the expression contains an object subtype otherwise the expression returns False.

SYNTAX:

```
IsObject(expression)
```

EXAMPLE :

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
    Dim fso,b
    b = 10
    set fso = createobject("Scripting.FileSystemObject")
    x = isobject(fso)
    Document.write "Line 1 : " & x & "<br />"
    y = isobject(b)
    Document.write "Line 2 : " & y & "<br />"

</script>
</body>
</html>

```

Save the file with .html extension upon executing the script in IE , the following result is displayed on the screen.

```

Line 1 : True
Line 2 : False

```

IsNumeric

The IsNumeric Function is used to check whether or not the expression has a number subtype. It returns a boolean value. **IsObject** returns True if the expression contains an number subtype otherwise the expression returns False.

SYNTAX :

```
IsNumeric(expression)
```

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim var, chk
    var = 20
    chk = IsNumeric(var)
    Document.write "Line 1 : " & chk & "<br />"

    var = "3.1415935745"
    chk = IsNumeric(var)
    Document.write "Line 2 : " & chk & "<br />"

    var = "20 Chapter 23.123 VBScript"
    chk = IsNumeric(var)
    Document.write "Line 3 : " & chk & "<br />"

</script>
</body>
</html>
```

Save the file with .html extension upon executing the script in IE , the following result is displayed on the screen.

```
Line 1 : True
Line 2 : True
Line 3 : False
```

TypeName

The TypeName Function is used to return the variant subtype information of the variable.

SYNTAX :

```
TypeName(varname)
```

The Typename function can return any of the following values.

- Byte - Byte Value
- Integer - Integer Value
- Long - Long Integer Value

- Single - Single-precision floating-point Value
- Double - Double-precision floating-point Value
- Currency - Currency Value
- Decimal - Decimal Value
- Date - Date or Time Value
- String - Character string Value
- Boolean - Boolean Value
- Empty - Uninitialized Value
- Null - No Valid Data
- Object - typename of Object
- Nothing - Object variable that doesn't yet refer to an object instance
- Error

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim ArrVar(2), vartype
    NullVar = Null    ' Assign Null value.

    vartype = TypeName(3.1450)
    Document.write "Line 1 : " & vartype & "<br />"
    vartype = TypeName(432)
    Document.write "Line 2 : " & vartype & "<br />"
    vartype = TypeName("Microsoft")
    Document.write "Line 3 : " & vartype & "<br />"
    vartype = TypeName(NullVar)
    Document.write "Line 4 : " & vartype & "<br />"
    vartype = TypeName(ArrVar)
    Document.write "Line 5 : " & vartype & "<br />"

</script>
</body>
</html>
```

Save the file with .html extension upon executing the script in IE , the following result is displayed on the screen.

```
Line 1 : Double
Line 2 : Integer
Line 3 : String
Line 4 : Null
```

```
Line 5 : Variant()
```

Eval

The Eval Function executes an expression and returns the result either as a string or a number.

SYNTAX :

```
Eval(expression)
```

The argument Expression can be a string expression or a number. If you pass to the Eval function a string that doesn't contain a numeric expression or a function name but only a simple text string, a run-time error occurs. For example, Eval("VBScript") results in an error.

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Document.write Eval("10 + 10") & "<br />"
    Document.write Eval("101 = 200") & "<br />"
    Document.write Eval("5 * 3") & "<br />"

</script>
</body>
</html>
```

Save the file with .html extension upon executing the script in IE , the following result is displayed on the screen.

```
20
false
15
```

Execute

The Execute statement accepts argument that is a string expression containing one or more statements for execution.

SYNTAX :

```
Execute(expression)
```

In VBScript, a = b can be interpreted two ways. It can be treated as an assignment statement where the value of x is assigned to y. It can also be interpreted as an expression that tests if a and b have the same value. If they do, result is True; if they are not, result is False. The Execute statement always uses the first interpretation while the Eval statement always uses the second.

EXAMPLE :

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
```

```

    Dim x
    x = "Global"
    y = "VBScript"
    Execute("x=y")
    msgbox x
    msgbox y

</script>
</body>
</html>

```

Save the file with .html extension upon executing the script in IE , the following result is displayed on the screen.

```

VBScript
VBScript

```

With..End With

The With statement allows us to perform a series of operation on a specified object without explicitly mentioning the object name over again and again.

SYNTAX :

```

With (objectname)
    statement 1
    statement 2
    statement 3
    ...
    ...
    statement n
End With

```

EXAMPLE :

Upon Executing the below script, Winword is opened and the specified text is entered.

```

<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Msg = "Vbscript" & vbCrLf & "Programming"
    Set objWord = CreateObject("Word.Application")
    objWord.Visible = True

    ' Objects methods are accessed without requaliyfying the objects again.'
    With objWord
        .Documents.Add
        .Selection.TypeText Msg
        .Selection.WholeStory
    End With

</script>
</body>
</html>

```

Randomize

The Randomize statement initializes the random number generator which is helpful for the developers to generate a random number.

SYNTAX :

```
Randomize [number]
```

EXAMPLE :

Upon Executing the below script, Winword is opened and the specified text is entered.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">

    Dim MyValue
    Randomize
    MyValue = Int((100 * Rnd) + 1)    ' Generate random value between 1 and 100.
    MsgBox MyValue

</script>
</body>
</html>
```

Save the above script as HTML and upon executing the script in IE, the following output is shown.

42