VB SCRIPT

BE VII – Computer & Electronics

What is VBScript?

- VBScript is a scripting language
- A scripting language is a lightweight programming language
- VBScript is a light version of Microsoft's programming language Visual Basic
- **VBScript is only supported by Microsoft's browsers (Internet Explorer)**

How does it Work?

When a VBScript is inserted into an HTML document, Internet Explorer browser will read the HTML and interpret the VBScript. The VBScript can be executed immediately, or at a later event.

VBScript only works in Microsoft browsers (Internet Explorer).

The HTML <script> tag is used to insert a VBScript into an HTML page.

Put a VBScript into an HTML Page

The example below shows how to use VBSript to write text on a web page:

Example (IE Only)

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

The example below shows how to add HTML tags to the VBScript:

Example (IE Only)

```
<html>
<body>
<script type="text/vbscript">
document.write("<h1>Hello World!</h1>")
</script>
</body>
</html>
```

Example Explained

To insert a VBScript into an HTML page, we use the <script> tag. Inside the <script> tag we use the type attribute to define the scripting language.

So, the <script type="text/vbscript"> and </script> tells where the VBScript starts and ends:

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

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

The **document.write** command is a standard VBScript command for writing output to a page.

By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a VBScript command and execute the code line. In this case the browser will write Hello World! to the page:

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

How to Handle Simple Browsers

Browsers that do not support scripting, will display VBScript as page content.

To prevent them from doing this, the HTML comment tag should be used to "hide" the VBScript.

Just add an HTML comment tag <!-- before the first VBScript statement, and a --> (end of comment) after the last VBScript statement, like this:

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

VBScripts can be placed in the body and in the head section of an HTML document.

Where to Put the VBScript

VBScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, or at a later event, such as when a user clicks a button. When this is the case we put the script inside a function or a sub procedure, you will learn about procedures in a later chapter.

Scripts in <head>

Put your functions and sub procedures in the head section, this way they are all in one place, and they do not interfere with page content.

Example (IE Only)

```
<html>
<head>
```

```
<script type="text/vbscript">
function myFunction()
alert("Hello World!")
end function
</script>
</head>

<body onload="myFunction()">
</body>
</html>
```

Scripts in <body>

If you don't want your script to be placed inside a function, and especially if your script should write page content, it should be placed in the body section.

Example (IE Only)

```
<html>
<head>
</head>

<body>
<script type="text/vbscript">
document.write("This message is written by VBScript")
</script>
</body>

</html>
```

Scripts in <head> and <body>

You can place an unlimited number of scripts in your document, and you can have scripts in both the body and the head section.

Example (IE Only)

```
<html>
<head>
<script type="text/vbscript">
function myFunction()
alert("Hello World!")
end function
</script>
</head>

<body>
<button onclick="myFunction()">Click me</button>
<script type="text/vbscript">
document.write("This message is written by VBScript")
</script>
</body>
</html>
```

Using an External VBScript

If you want to run the same VBScript on several pages, without having to write the same script on every page, you can write a VBScript in an external file.

Save the external VBScript file with a .vbs file extension.

Note: The external script cannot contain the <script> tag!

To use the external script, point to the .vbs file in the "src" attribute of the <script> tag:

Example

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

Simple Algebra

x=5, y=6, z=x+y

A letter (like x) could be used to hold a value (like 5), and that you could use the information above to calculate the value of z to be 11.

These letters are called **variables**, and variables can be used to hold values (x=5) or expressions (z=x+y).

VBScript Variables

As with algebra, VBScript variables are used to hold values or expressions.

A variable can have a short name, like x, or a more descriptive name, like carname.

Rules for VBScript variable names:

- Must begin with a letter
- Cannot contain a period (.)
- Cannot exceed 255 characters

In VBScript, all variables are of type *variant*, which can store different types of data.

Declaring (Creating) VBScript Variables

Creating variables in VBScript is most often referred to as "declaring" variables.

You can declare VBScript variables with the Dim, Public or the Private statement. Like this:

```
Dim x
Dim carname
```

Now you have created two variables. The name of the variables are "x" and "carname".

You can also declare variables by using its name in a script. Like this:

```
carname="Volvo"
```

Now you have also created a variable. The name of the variable is "carname". However, this method is not a good practice, because you can misspell the variable name later in your script, and that can cause strange results when your script is running.

If you misspell for example the "carname" variable to "carnime", the script will automatically create a new variable called "carnime". To prevent your script from doing this, you can use the Option Explicit statement. This statement forces you to declare all your variables with the dim, public or private statement.

Put the Option Explicit statement on the top of your script. Like this:

```
Option Explicit
Dim carname
carname=some value
```

Assigning Values to Variables

You assign a value to a variable like this:

```
carname="Volvo"
x=10
```

The variable name is on the left side of the expression and the value you want to assign to the variable is on the right. Now the variable "carname" has the value of "Volvo", and the variable "x" has the value of "10".

Lifetime of Variables

When you declare a variable within a procedure, the variable can only be accessed within that procedure. When the procedure exits, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different procedures, because each is recognized only by the procedure in which it is declared.

If you declare a variable outside a procedure, all the procedures on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

VBScript Array Variables

An array variable is used to store multiple values in a single variable.

In the following example, an array containing 3 elements is declared:

```
Dim names(2)
```

The number shown in the parentheses is 2. We start at zero so this array contains 3 elements. This is a fixed-size array. You assign data to each of the elements of the array like this:

```
names(0)="Tove"
names(1)="Jani"
names(2)="Stale"
```

```
mother=names(0)
```

You can have up to 60 dimensions in an array. Multiple dimensions are declared by separating the numbers in the parentheses with commas. Here we have a two-dimensional array consisting of 5 rows and 7 columns:

```
Dim table(4,6)
```

Assign data to a two-dimensional array:

Example (IE Only)

```
<html>
<body>

<script type="text/vbscript">
Dim x(2,2)
x(0,0)="Volvo"
x(0,1)="BMW"
x(0,2)="Ford"
x(1,0)="Apple"
x(1,1)="Orange"
x(1,2)="Banana"
x(2,0)="Coke"
x(2,1)="Pepsi"
x(2,2)="Sprite"
for i=0 to 2
   document.write("<p>")
   for j=0 to 2
      document.write(x(i,j) & "<br />")
   next
   document.write("</p>")
next
</script>

</body>
</html>
```

VBScript has a full range of operators, including arithmetic operators, comparison operators, concatenation operators, and logical operators.

Arithmetic		Comparison		Logical	
Description	**Symbol**	**Description**	**Symbol**	**Description**	**Symbol**
Exponentiation	^	Equality	=	Logical negation	Not
Unary negation	-	Inequality	<>	Logical conjunction	And
Multiplication	*	Less than	<	Logical disjunction	Or
Division	/	Greater than	>	Logical exclusion	Xor
Integer division	\	Less than or equal to	<=		

Modulus arithmetic	Mod	Greater than or equal to	>=		
Addition	+				
Subtraction	-				
String concatenation	&				

When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. Comparison operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear. Arithmetic and logical operators are evaluated in the following order of precedence.

When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. Likewise, when addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right.

The string concatenation (&) operator is not an arithmetic operator, but in precedence it does fall after all arithmetic operators and before all comparison operators.

VBScript has two kinds of procedures:

- Sub procedure
- Function procedure

VBScript Sub Procedures

A Sub procedure:

- is a series of statements, enclosed by the Sub and End Sub statements
- can perform actions, but **does not return** a value
- can take arguments
- without arguments, it must include an empty set of parentheses ()

```
Sub mysub()
  some statements
End Sub
```

or

```
Sub mysub(argument1,argument2)
  some statements
End Sub
```

Example (IE Only)

```
Sub mysub()
  alert("Hello World")
End Sub
```

VBScript Function Procedures

A Function procedure:

- is a series of statements, enclosed by the Function and End Function statements
- can perform actions and **can return** a value
- can take arguments that are passed to it by a calling procedure
- without arguments, must include an empty set of parentheses ()
- returns a value by assigning a value to its name

```
Function myfunction()
 some statements
 myfunction=some value
End Function
```

or

```
Function myfunction(argument1,argument2)
 some statements
 myfunction=some value
End Function
```

Example (IE Only)

```
function myfunction()
  myfunction=Date()
end function
```

How to Call a Procedure

There are different ways to call a procedure. You can call it from within another procedure, on an event, or call it within a script.

Example (IE Only)

Call a procedure when the user clicks on a button:

```
<body>
<button onclick="myfunction()">Click me</button>
</body>
```

Procedures can be used to get a variable value:

```
carname=findname()
```

Here you call a Function called "findname", the Function returns a value that will be stored in the variable "carname".

Function procedures can calculate the sum of two arguments:

Example (IE Only)

```
Function myfunction(a,b)
myfunction=a+b
End Function

document.write(myfunction(5,9))
```

The function "myfunction" will return the sum of argument "a" and argument "b". In this case 14.

When you call a procedure you can use the Call statement, like this:

```
Call MyProc(argument)
```

Or, you can omit the Call statement, like this:

```
MyProc argument
```

Conditional Statements

Conditional statements are used to perform different actions for different decisions.

In VBScript we have four conditional statements:

- **If statement** - executes a set of code when a condition is true
- **If...Then...Else statement** - select one of two sets of lines to execute
- **If...Then...ElseIf statement** - select one of many sets of lines to execute
- **Select Case statement** - select one of many sets of lines to execute

---

If...Then...Else

Use the If...Then...Else statement if you want to

- execute some code if a condition is true
- select one of two blocks of code to execute

If you want to execute only **one** statement when a condition is true, you can write the code on one line:

```
If i=10 Then alert("Hello")
```

There is no ..Else.. in this syntax. You just tell the code to perform **one action** if a condition is true (in this case If i=10).

If you want to execute **more than one** statement when a condition is true, you must put each statement on separate lines, and end the statement with the keyword "End If":

```
If i=10 Then
alert("Hello")
i = i+1
End If
```

There is no ..Else.. in the example above either. You just tell the code to perform **multiple actions** if the condition is true.

If you want to execute a statement if a condition is true and execute another statement if the condition is not true, you must add the "Else" keyword:

Example (IE Only)

```
<html>
<body>
<head>
<script type="text/vbscript">
Function greeting()
i=hour(time)
If i < 10 Then
  document.write("Good morning!")
Else
  document.write("Have a nice day!")
End If
End Function
</script>
</head>

<body onload="greeting()">
</body>

</html>
```

In the example above, the first block of code will be executed if the condition is true, and the other block will be executed otherwise (if i is greater than 10).

If...Then...ElseIf

You can use the If...Then...ElseIf statement if you want to select one of many blocks of code to execute:

Example (IE Only)

```
<html>
<body>
<head>
<script type="text/vbscript">
Function greeting()
i=hour(time)
If i = 10 Then
  document.write("Just started...!")
ElseIf i = 11 then
  document.write("Hungry!")
ElseIf i = 12 then
  document.write("Ah, lunch-time!")
ElseIf i = 16 then
  document.write("Time to go home!")
Else
  document.write("Unknown")
End If
End Function
</script>
</head>

<body onload="greeting()">
</body>
```

```
</html>
```

Select Case

You can also use the "Select Case" statement if you want to select one of many blocks of code to execute:

Example (IE Only)

```
<html>
<body>
<script type="text/vbscript">
d=weekday(date)
Select Case d
  Case 1
    document.write("Sleepy Sunday")
  Case 2
    document.write("Monday again!")
  Case 3
    document.write("Just Tuesday!")
  Case 4
    document.write("Wednesday!")
  Case 5
    document.write("Thursday...")
  Case 6
    document.write("Finally Friday!")
  Case else
    document.write("Super Saturday!!!!")
End Select
</script>

</body>
</html>
```

This is how it works: First we have a single expression (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each Case in the structure. If there is a match, the block of code associated with that Case is executed.

Looping Statements

Looping statements are used to run the same block of code a specified number of times.

In VBScript we have four looping statements:

- **For...Next statement** - runs code a specified number of times
- **For Each...Next statement** - runs code for each item in a collection or each element of an array
- **Do...Loop statement** - loops while or until a condition is true
- **While...Wend statement** - Do not use it - use the Do...Loop statement instead

---

For...Next Loop

Use the **For...Next** statement to run a block of code a specified number of times.

The **For** statement specifies the counter variable (**i**), and its start and end values. The **Next** statement increases the counter variable (**i**) by one.

```
<html>
<body>

<script type="text/vbscript">
For i = 0 To 5
  document.write("The number is " & i & "<br />")
Next
</script>

</body>
</html>
```

The Step Keyword

With the **Step** keyword, you can increase or decrease the counter variable by the value you specify.

In the example below, the counter variable (**i**) is INCREASED by two, each time the loop repeats.

```
For i=2 To 10 Step 2
  some code
Next
```

To decrease the counter variable, you must use a negative **Step** value. You must specify an end value that is less than the start value. In the example below, the counter variable (**i**) is DECREASED by two, each time the loop repeats.

```
For i=10 To 2 Step -2
  some code
Next
```

Exit a For...Next

You can exit a For...Next statement with the Exit For keyword.

```
For i=1 To 10
  If i=5 Then Exit For
  some code
Next
```

For Each...Next Loop

A **For Each...Next** loop repeats a block of code for each item in a collection, or for each element of an array.

```
<html>
<body>
```

```
<script type="text/vbscript">
Dim cars(2)
cars(0)="Volvo"
cars(1)="Saab"
cars(2)="BMW"

For Each x In cars
  document.write(x & "<br />")
Next
</script>

</body>
</html>
```

Do...Loop

If you don't know how many repetitions you want, use a Do...Loop statement.

The Do...Loop statement repeats a block of code while a condition is true, or until a condition becomes true.

Repeat Code While a Condition is True. While keyword is used to check a condition in a Do...Loop statement.

```
Do While i>10
  some code
Loop
```

If **i** equals 9, the code inside the loop above will never be executed.

```
Do
  some code
Loop While i>10
```

The code inside this loop will be executed at least one time, even if **i** is less than 10.

Exit a Do...Loop

You can exit a Do...Loop statement with the Exit Do keyword.

```
Do Until i=10
  i=i-1
  If i<10 Then Exit Do
Loop
```

The code inside this loop will be executed as long as **i** is different from 10, and as long as **i** is greater than 10.

While … Wend Loop
Executes a series of statements as long as a given condition is **True**.

Syntax
While *condition*
  [*statements*]
Wend

Part	Description
condition	Numeric or string expression that evaluates to **True** or **False**. If *condition* is **Null**, *condition* is treated as **False**.
statements	One or more statements executed while condition is **True**.

Remarks
If *condition* is **True**, all statements in *statements* are executed until the **Wend** statement is encountered. Control then returns to the **While** statement and *condition* is again checked. If *condition* is still **True**, the process is repeated. If it is not **True**, execution resumes with the statement following the **Wend** statement.

While...Wend loops can be nested to any level. Each **Wend** matches the most recent **While**.

*The **Do...Loop** statement provides a more structured and flexible way to perform looping.*

- **After Studying JavaScript** -

VBScript & JavaScript

	VBScript	**JavaScript**
Arrays	Declaration (e.g., Dim, Static) Lbound, Ubound ReDim, Erase	new Array() new Object()
Assignment	=	=
Comments	 Single Quote (')	// /* comment */

Control Flow	Do ....Loop
	For ...Next, For Each ...Next
	While ...Wend
	If ... Then ...Else
		do { statements }
		while (condition)
		for (expression; condition; [expression])
		{
		statements
		
		}
		
		while (condition) { statements }
		if (condition) { statements1 } else { statements2 }
		switch (expression)
		{
		case label :
		statement;
		break;
		
		case label :
		statement;
		break; ...
		
		default : statement;
		}

Literals	Empty Null True, False User Defined Literals(e.g., 123.456, "test")	NaN null true, false User Defined Literals(e.g., 123.456, "test")
Operators	Arithmetic +, -, *, /, \, ^, Mod, Negation (-) Strings concatenation (&) Comparison =, <,>, <>, <=,>=, Is Logical Not, And, Or, Xor, Eqv, Imp	Arithmetic +, ++, -, --, *, / % String + += Logical && || ! Bitwise & ^ | ~ <<>> >>>

		Comparison== |= > >= < <= Special: , delete new this
Options	Option Explicit	N/A

Declaring Procedures	Functions
	Sub
		function function-name(arg1, arg2, ... argN)
		{
		}

Calling Procedures:	With Call Keyword or Directly.	
Exiting Procedures	Exit Function Exit Sub	
Procedure-level Variables	Dim Static	var variable-name

When one uses Option Explicit it forces the user to declare all variables. If one tries to use a variable without declaring it an error is generated. If one doesn't use Option Explicit one is free to use variables without declaring them, however, this is bad programming practice as it is easy to misspell a variable and produce faulty code.

VB SCRIPT Practical

Display Date and Time

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

document.write("Today's date is " & Date())

document.write("<br />")

document.write("The time is " & Time())

</script>
```

Display the days

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

document.write("<p>")

document.write(WeekDayName(1))

document.write("<br />")

document.write(WeekDayName(2))

document.write("</p><p>")



document.write("Get the abbreviated name of a weekday:")

document.write("<br />")

document.write(WeekDayName(1,True))

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

```
document.write(WeekDayName(2,True))

document.write("</p><p>")


document.write("Get the current weekday:")

document.write("<br />")

document.write(WeekdayName(weekday(Date)))

document.write("<br />")

document.write(WeekdayName(weekday(Date), True))

document.write("</p>")

</script>
```

Display the months

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

document.write("<p>")

document.write(MonthName(1))

document.write("<br />")

document.write(MonthName(2))

document.write("</p><p>")


document.write("Here is how you get the abbreviated name of a month:")

document.write("<br />")

document.write(MonthName(1,True))

document.write("<br />")

document.write(MonthName(2,True))

document.write("</p><p>")


document.write("Here is how you get the current month:")

document.write("<br />")

document.write(MonthName(Month(Date)))
```

```
document.write(MonthName(Month(Date),True))

document.write("</p>")

</script>
```

Display the current month and day

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

document.write("Today's day is " & WeekdayName(Weekday(Date)))

document.write("<br />")

document.write("The month is " & MonthName(Month(Date)))

</script>
```

Format Date and Time

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

document.write(FormatDateTime(Date(),vbGeneralDate))

document.write("<br />")

document.write(FormatDateTime(Date(),vbLongDate))

document.write("<br />")

document.write(FormatDateTime(Date(),vbShortDate))

document.write("<br />")

document.write(FormatDateTime(Now(),vbLongTime))

document.write("<br />")

document.write(FormatDateTime(Now(),vbShortTime))

</script>
```

Uppercase or lowercase characters?

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

txt="Have a nice day!"

document.write(UCase(txt))

document.write("<br />")

document.write(LCase(txt))

</script>
```

Reverse a string

```
<script type="text/vbscript">
sometext = "Hello Everyone!"
document.write(StrReverse(sometext))
</script>
```

Round a number

```
<script type="text/vbscript">
i = 48.66776677
j = 48.3333333
document.write(Round(i))
document.write("<br />")
document.write(Round(j))
</script>
```

Return a random number

```
<script type="text/vbscript">
Randomize()
document.write(Rnd())
</script>
```

Return a random number between 0-99

```
<script type="text/vbscript">
Randomize()
randomNumber=Int(100 * Rnd())
document.write("A random number: <b>" & randomNumber & "</b>")
</script>
```