VBScript

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

How Does it Work?

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

How to Put VBScript Code in an HTML Document

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

And it produces this output:

```
Hello from VBScript!
```

To insert a script in an HTML document, use the <script> tag. Use the type attribute to define the scripting language.

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

Then comes the VBScript: The command for writing some text on a page is **document.write**:

```
document.write("Hello from VBScript!")
```

The script ends:

```
</script>
```

How to Handle Older Browsers

Older browsers that do not support scripts will display the script as page content. To prevent them from doing this, you can use the HTML comment tag:

```
<script type="text/vbscript">
<!--
  some statements
-->
</script>
```

Examples

Head section
Scripts can be placed in the head section. Usually we put all the "functions" in the head section. The reason for this is to be sure that the script is loaded before the function is called.

Body section
Execute a script that is placed in the body section. Scripts in the body section are executed when the page is loading.

Where to Put the VBScript

Scripts 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, other times when a user triggers an event.

Scripts in the head section: Scripts to be executed when they are called or when an event is triggered go in the head section. When you place a script in the head section you will assure that the script is loaded before anyone uses it:

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

Scripts in the body section: Scripts to be executed when the page loads go in the body section. When you place a script in the body section it generates the content of the page:

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

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

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

VBScript Variables

A variable is a "container" for information you want to store. A variable's value can change during the script. You can refer to a variable by name to see its value or to change its value. In VBScript, all variables are of type *variant*, that can store different types of data.

Rules for Variable Names:

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

Declaring Variables

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

```
dim name
name=some value
```

Now you have created a variable. The name of the variable is "name".

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

```
name=some value
```

Now you have also created a variable. The name of the variable is "name".

However, the last 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. This is because when you misspell for example the "name" variable to "nime" the script will automatically create a new variable called "nime". To prevent your script from doing this you can use the Option Explicit statement. When you use this statement you will have 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 name
name=some value
```

Assigning Values to Variables

You assign a value to a variable like this:

```
name="Madhav"
i=200
```

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 "name" has the value "Madhav".

Lifetime of Variables

How long a variable exists is its lifetime.

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.

Array Variables

Sometimes you want to assign more than one value to a single variable. Then you can create a variable that can contain a series of values. This is called an array variable. The declaration of an array variable uses parentheses ( ) following the variable name. 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)="Madhav"
names(1)="Dutt"
names(2)="Kumar"
```

Similarly, the data can be retrieved from any element using the index of the particular array element you want. Like this:

```
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)
```

VBScript Procedures

We have two kinds of procedures: The Sub procedure and the Function procedure.

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 that are passed to it by a calling procedure
- without arguments, must include an empty set of parentheses ()

```
Sub mysub()
 some statements
End Sub


or

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

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
```

Call a Sub or Function Procedure

When you call a Function in your code, you do like this:

```
name = findname()
```

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

Or, you can do like this:

```
msgbox "Your name is " & findname()
```

Here you also call a Function called "findname", the Function returns a value that will be displayed in the message box.

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

```
Call MyProc(argument)
```

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

```
MyProc argument
```

VBScript Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In VBScript we have four conditional statements:

- **if statement** - use this statement if you want to execute a set of code when a condition is true
- **if...then...else statement** - use this statement if you want to select one of two sets of lines to execute
- **if...then...elseif statement** - use this statement if you want to select one of many sets of lines to execute

- **select case statement** - use this statement if you want to select one of many sets of lines to execute

---

If....Then.....Else

You should 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 msgbox "Hello"
```

There is no ..else.. in this syntax. You just tell the code to perform **one action** if the 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
   msgbox "Hello"
   i = i+1
end If
```

There is no ..else.. in this syntax 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:

```
if i=10 then
   msgbox "Hello"
else
   msgbox "Goodbye"
end If
```

The first block of code will be executed if the condition is true, and the other block will be executed otherwise (if i is not equal to 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:

```
if payment="Cash" then
   msgbox "You are going to pay cash!"
 elseif payment="Visa" then
   msgbox "You are going to pay with visa."
 elseif payment="AmEx" then
   msgbox "You are going to pay with American Express."
 else
   msgbox "Unknown method of payment."
end If
```

Select Case

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

```
select case payment
 case "Cash"
   msgbox "You are going to pay cash"
 case "Visa"
   msgbox "You are going to pay with visa"
 case "AmEx"
   msgbox "You are going to pay with American Express"
 case Else
   msgbox "Unknown method of payment"
end select
```

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.

VBScript Looping Statements

Very often when you write code, you want to allow the same block of code to run a number of times. You can use looping statements in your code to do this.

In VBScript we have four looping statements:

- **For...Next statement** - runs statements a specified number of times.
- **For Each...Next statement** - runs statements 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

You can use a **For...Next** statement to run a block of code, when you know how many repetitions you want.

You can use a counter variable that increases or decreases with each repetition of the loop, like this:

```
For i=1 to 10
  some code
Next
```

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

Step Keyword

Using 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 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.

```
dim cars(2)
cars(0)="Volvo"
cars(1)="Saab"
cars(2)="BMW"

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

Do...Loop

You can use Do...Loop statements to run a block of code when you do not know how many repetitions you want. The block of code is repeated while a condition is true or until a condition becomes true.

Repeating Code While a Condition is True

You use the While keyword 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.

Repeating Code Until a Condition Becomes True

You use the Until keyword to check a condition in a Do...Loop statement.

```
Do Until i=10
  some code
Loop
```

If **i** equals 10, the code inside the loop will never be executed.

```
Do
  some code
Loop Until i=10
```

The code inside this loop will be executed at least one time, even if **i** is equal to 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.

VBScript Summary

This tutorial has taught you how to add VBScript to your HTML pages, to make your web site more dynamic and interactive.

You have learned how to create variables and functions, and how to make different scripts run in response to different scenarios.

Now You Know VBScript, What's Next?

The next step is to learn ASP.

While scripts in an HTML file are executed on the client (in the browser), scripts in an ASP file are executed on the server.

With ASP you can dynamically edit, change or add any content of a Web page, respond to data submitted from HTML forms, access any data or databases and return the results to a browser, customize a Web page to make it more useful for individual users.

Since ASP files are returned as plain HTML, they can be viewed in any browser.

VBScript Functions

- Date/Time functions
- Conversion functions
- Format functions
- Math functions
- Array functions
- String functions
- Other functions

Date/Time Functions

Function	Description
CDate	Converts a valid date and time expression to the variant of subtype Date
Date	Returns the current system date
DateAdd	Returns a date to which a specified time interval has been added
DateDiff	Returns the number of intervals between two dates
DatePart	Returns the specified part of a given date
DateSerial	Returns the date for a specified year, month, and day
DateValue	Returns a date
Day	Returns a number that represents the day of the month (between 1 and 31, inclusive)
FormatDateTime	Returns an expression formatted as a date or time
Hour	Returns a number that represents the hour of the day (between 0 and 23, inclusive)
IsDate	Returns a Boolean value that indicates if the evaluated expression can be converted to a date
Minute	Returns a number that represents the minute of the hour (between 0 and 59, inclusive)
Month	Returns a number that represents the month of the year (between 1 and 12, inclusive)
MonthName	Returns the name of a specified month
Now	Returns the current system date and time
Second	Returns a number that represents the second of the minute (between 0 and 59, inclusive)
Time	Returns the current system time
Timer	Returns the number of seconds since 12:00 AM
TimeSerial	Returns the time for a specific hour, minute, and second
TimeValue	Returns a time
Weekday	Returns a number that represents the day of the week (between 1 and 7, inclusive)
WeekdayName	Returns the weekday name of a specified day of the week
Year	Returns a number that represents the year

Conversion Functions

Function	Description
Asc	Converts the first letter in a string to ANSI code
CBool	Converts an expression to a variant of subtype Boolean
CByte	Converts an expression to a variant of subtype Byte
CCur	Converts an expression to a variant of subtype Currency
CDate	Converts a valid date and time expression to the variant of subtype Date
CDbl	Converts an expression to a variant of subtype Double
Chr	Converts the specified ANSI code to a character
CInt	Converts an expression to a variant of subtype Integer
CLng	Converts an expression to a variant of subtype Long
CSng	Converts an expression to a variant of subtype Single

CStr	Converts an expression to a variant of subtype String
Hex	Returns the hexadecimal value of a specified number
Oct	Returns the octal value of a specified number

Format Functions

Function	Description
FormatCurrency	Returns an expression formatted as a currency value
FormatDateTime	Returns an expression formatted as a date or time
FormatNumber	Returns an expression formatted as a number
FormatPercent	Returns an expression formatted as a percentage

Math Functions

Function	Description
Abs	Returns the absolute value of a specified number
Atn	Returns the arctangent of a specified number
Cos	Returns the cosine of a specified number (angle)
Exp	Returns *e* raised to a power
Hex	Returns the hexadecimal value of a specified number
Int	Returns the integer part of a specified number
Fix	Returns the integer part of a specified number
Log	Returns the natural logarithm of a specified number
Oct	Returns the octal value of a specified number
Rnd	Returns a random number less than 1 but greater or equal to 0
Sgn	Returns an integer that indicates the sign of a specified number
Sin	Returns the sine of a specified number (angle)
Sqr	Returns the square root of a specified number
Tan	Returns the tangent of a specified number (angle)

Array Functions

Function	Description
Array	Returns a variant containing an array
Filter	Returns a zero-based array that contains a subset of a string array based on a filter criteria
IsArray	Returns a Boolean value that indicates whether a specified variable is an array
Join	Returns a string that consists of a number of substrings in an array
LBound	Returns the smallest subscript for the indicated dimension of an array
Split	Returns a zero-based, one-dimensional array that contains a specified number of substrings
UBound	Returns the largest subscript for the indicated dimension of an array

String Functions

Function	Description
InStr	Returns the position of the first occurrence of one string within another. The search begins at the first character of the string
InStrRev	Returns the position of the first occurrence of one string within another. The search begins at the last character of the string
LCase	Converts a specified string to lowercase

Left	Returns a specified number of characters from the left side of a string
Len	Returns the number of characters in a string
LTrim	Removes spaces on the left side of a string
RTrim	Removes spaces on the right side of a string
Trim	Removes spaces on both the left and the right side of a string
Mid	Returns a specified number of characters from a string
Replace	Replaces a specified part of a string with another string a specified number of times
Right	Returns a specified number of characters from the right side of a string
Space	Returns a string that consists of a specified number of spaces
StrComp	Compares two strings and returns a value that represents the result of the comparison
String	Returns a string that contains a repeating character of a specified length
StrReverse	Reverses a string
UCase	Converts a specified string to uppercase

Other Functions

Function	Description
CreateObject	Creates an object of a specified type
Eva	Evaluates an expression and returns the result
GetLocale	Returns the current locale ID
GetObject	Returns a reference to an automation object from a file
GetRef	Allows you to connect a VBScript procedure to a DHTML event on your pages
InputBox	Displays a dialog box, where the user can write some input and/or click on a button, and returns the contents
IsEmpty	Returns a Boolean value that indicates whether a specified variable has been initialized or not
IsNull	Returns a Boolean value that indicates whether a specified expression contains no valid data (Null)
IsNumeric	Returns a Boolean value that indicates whether a specified expression can be evaluated as a number
IsObject	Returns a Boolean value that indicates whether the specified expression is an automation object
LoadPicture	Returns a picture object. Available only on 32-bit platforms
MsgBox	Displays a message box, waits for the user to click a button, and returns a value that indicates which button the user clicked
RGB	Returns a number that represents an RGB color value
Round	Rounds a number
ScriptEngine	Returns the scripting language in use
ScriptEngineBuildVersion	Returns the build version number of the scripting engine in use
ScriptEngineMajorVersion	Returns the major version number of the scripting engine in use
ScriptEngineMinorVersion	Returns the minor version number of the scripting engine in use
SetLocale	Sets the locale ID and returns the previous locale ID
TypeName	Returns the subtype of a specified variable
VarType	Returns a value that indicates the subtype of a specified variable

VBScript Keywords

Keyword	Description

empty	Used to indicate an uninitialized variable value. A variable value is uninitialized when it is first created and no value is assigned to it, or when a variable value is explicitly set to empty. Example: dim x 'the variable x is uninitialized! x="ff" 'the variable x is NOT uninitialized anymore x=empty 'the variable x is uninitialized! **Note:** This is not the same as Null!!
isEmpty	Used to test if a variable is uninitialized. Example: If (isEmpty(x)) 'is x uninitialized?
nothing	Used to indicate an uninitialized object value, or to disassociate an object variable from an object to release system resources. Example: set myObject=nothing
is nothing	Used to test if a value is an initialized object. Example: If (myObject Is Nothing) 'is it unset? **Note:** If you compare a value to Nothing, you will not get the right result! Example: If (myObject = Nothing) 'always false!
null	Used to indicate that a variable contains no valid data. One way to think of Null is that someone has explicitly set the value to "invalid", unlike Empty where the value is "not set". **Note:** This is not the same as Empty or Nothing!! Example: x=Null 'x contains no valid data
isNull	Used to test if a value contains invalid data. Example: if (isNull(x)) 'is x invalid?
true	Used to indicate a Boolean condition that is correct (true has a value of -1)
false	Used to indicate a Boolean condition that is not correct (false has a value of 0)

VBScript Examples

Basic

Write text using VBScript

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

Format text with HTML tags

```
<html>
<body>
<script type="text/vbscript">
document.write("<h1>Hello World!</h1>")
document.write("<h2>Hello World!</h2>")
</script>
</body>
</html>
```

A function in the head section

```
<html>
<head>
<script type="text/vbscript">
alert("Hello")
</script>
</head>
<body>
<p>
We usually use the head section for "functions".
The reason for this is to be sure that the script is loaded before the function is called.
</p>
</body>
</html>
```

A script in the body section

```
<html>
<body>
<script type="text/vbscript">
document.write("Scripts in the body section are executed when the page is loading")
</script>
</body>
</html>
```

Variables

Create a variable

```
<html>
<body>
<script type="text/vbscript">
dim name
name="Jan Egil"
document.write(name)
</script>
</body>
</html>
```

Insert a variable value in a text

```
<html>
<body>
<script type="text/vbscript">
dim name
name="Jan Egil"
document.write("My name is: " & name)
</script>
</body>
</html>
```

Create an array

```
<html>
<body>
<script type="text/vbscript">
dim famname(5)
famname(0)="Jan Egil"
famname(1)="Tove"
famname(2)="Hege"
famname(3)="Stale"
famname(4)="Kai Jim"
famname(5)="Borge"
for i=0 to 5
  document.write(famname(i) & "<br />")
next
</script>
</body>
</html>
```

Procedures

Sub procedure

```
<html>
<head>
<script type="text/vbscript">
sub mySub()
  msgbox("This is a sub procedure")
end sub
</script>
</head>
<body>
<script type="text/vbscript">
call mySub()
</script>
<p>A sub procedure does not return a result.</p>
</body>
</html>
```

Function procedure

```
<html>
<head>
<script type="text/vbscript">
function myFunction()
 myFunction = "BLUE"
end function
</script>
</head>
<body>
<script type="text/vbscript">
document.write("My favorite color is " & myFunction())
</script>
<p>A function procedure CAN return a result.</p>
</body>
</html>
```

Conditional Statements

If...then..else statement

```
<html>
<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>
```

If...then..elseif statement

```
<html>
<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 statement

```
<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>
<p>This example demonstrates the "select case" statement.<br />
You will receive a different greeting based on what day it is.<br />
Note that Sunday=1, Monday=2, Tuesday=3, etc.</p>
</body>
</html>
```

Random link

```
<html>
<body>
<script type="text/vbscript">
randomize()
r=rnd()
if r>0.5 then
 document.write("<a href='http://www.madhavendra.com'>Learn Web Development!</a>")
else
 document.write("<a href='http://www.refsnesdata.no'>Visit Refsnes Data!</a>")
end if
</script>
<p>
This example demonstrates a link, when you click on the link it will take you to madhavendra.com
OR to
RefsnesData.no. There is a 50% chance for each of them.
</p>
</body>
</html>
```

Looping

For..next loop

```
<html>
<body>
<script type="text/vbscript">
for i = 0 to 5
 document.write("The number is " & i & "<br />")
next
</script>
</body>
</html>
```

Looping through the HTML headers

```
<html>
<body>
<script type="text/vbscript">
for i=1 to 6
 document.write("<h" & i & ">This is header " & i & "</h" & i & ">")
next
</script>
</body>
</html>
```

For..each loop

```
<html>
<body>
<script type="text/vbscript">
dim names(2)
names(0) = "Madhav"
names(1) = "Kumar"
names(2) = "Dutt"

for each x in names
  document.write(x & "<br />")
next
</script>
</body>
</html>
```

Do...While loop

```
<html>
<body>
<script type="text/vbscript">
i=0
do while i < 10
  document.write(i & "<br />")
  i=i+1
loop
</script>
</body>
</html>
```

Date and Time Functions

Display date and time

```
<html>
<body>
<script type="text/vbscript">
document.write("Today's date is " & date())
document.write("<br />")
document.write("The time is " & time())
</script>
</body>
</html>
```

Display the days

```
<html>
<body>
<p>VBScripts' function <b>WeekdayName</b> is used to get a weekday:</p>
<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>
</body>
</html>
```

Display the months

```
<html>
<body>
<p>VBScripts' function <b>MonthName</b> is used to get a month:</p>
```

```
<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("<br />")
document.write(MonthName(month(date), true))
document.write("</p>")
</script>
</body>
</html>
```

Display the current month and day

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

Countdown to year 3000

```
<html>
<body>
<p>Countdown to year 3000:</p>
<p>
<script type="text/vbscript">
millennium=cdate("1/1/3000 00:00:00")
document.write("It is " & DateDiff("m", Now(), millennium) & " months to year 3000!<br />")
document.write("It is " & DateDiff("d", Now(), millennium) & " days to year 3000!<br />")
document.write("It is " & DateDiff("h", Now(), millennium) & " hours to year 3000!<br />")
document.write("It is " & DateDiff("n", Now(), millennium) & " minutes to year 3000!<br />")
document.write("It is " & DateDiff("s", Now(), millennium) & " seconds to year 3000!<br />")
</script>
</p>
</body>
</html>
```

Add a time interval to a date

```
<html>
<body>
<script type="text/vbscript">
document.write(DateAdd("d",30,date()))
</script>
<p>
This example uses <b>DateAdd</b> to calculate a date 30 days from today.
</p>
<p>
Syntax for DateAdd: DateAdd(interval,number,date).
</p>
```

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

Format date and time

```
<html>
<body>
<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>
<p>Syntax for FormatDateTime: FormatDateTime(date,namedformat).</p>
</body>
</html>
```

Is this a date?

```
<html>
<body>
<script type="text/vbscript">
somedate="10/30/99"
document.write(IsDate(somedate))
</script>
</body>
</html>
```

Other Built-in Functions

Uppercase or lowercase characters?

```
<html>
<body>
<script type="text/vbscript">
txt="Have a nice day!"
document.write(ucase(txt))
document.write("<br />")
document.write(lcase(txt))
</script>
</body>
</html>
```

Remove leading or trailing spaces from a string

```
<html>
<body>
<script type="text/vbscript">
fname=" Bill "
document.write("Hello" & trim(fname) & "Gates<br />")
document.write("Hello" & rtrim(fname) & "Gates<br />")
document.write("Hello" & ltrim(fname) & "Gates<br />")
</script>
</body>
</html>
```

Reverse a string

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

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

Round a number

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

Return a random number

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

Return a random number between 0-99

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

Return a specified number of characters from the left or right side of a string

```
<html>
<body>
<script type="text/vbscript">
sometext="Welcome to our Web Site!!"
document.write(Left(sometext,5))
document.write("<br />")
document.write(Right(sometext,5))
</script>
</body>
</html>
```

Replace some characters in a string

```
<html>
<body>
<script type="text/vbscript">
sometext="Welcome to this Web!!"
document.write(Replace(sometext, "Web", "Page"))
</script>
```

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

Return a specified number of characters from a string

```
<html>
<body>
<script type="text/vbscript">
sometext="Welcome to our Web Site!!"
document.write(Mid(sometext, 9, 2))
</script>
</body>
</html>
```

Date/Time Functions

The **CDate** function converts a valid date and time expression to type Date, and returns the result.

Tip: Use the IsDate function to determine if date can be converted to a date or time.

Note: The IsDate function uses local setting to determine if a string can be converted to a date ("January" is not a month in all languages.)

Syntax

```
CDate(date)
```

Parameter	Description
date	Required. Any valid date expression (like Date() or Now())

Example 1

```
d="April 22, 2001"
if IsDate(d) then
  document.write(CDate(d))
end if
Output:
4/22/01
```

Example 2

```
d=#2/22/01#
if IsDate(d) then
  document.write(CDate(d))
end if
Output:
2/22/01
```

Example 3

```
d="3:18:40 AM"
if IsDate(d) then
  document.write(CDate(d))
end if
Output:
3:18:40 AM
```

The **Date** function returns the current system date.

Syntax

```
Date
```

Example 1

```
document.write("The current system date is: ")
document.write(Date)
Output:
The current system date is: 1/14/2002
```

The **DateAdd** function returns a date to which a specified time interval has been added.

Syntax

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

Parameter	Description
interval	Required. The interval you want to add Can take the following values: • yyyy - Year • q - Quarter • m - Month • y - Day of year • d - Day • w - Weekday • ww - Week of year • h - Hour • n - Minute • s - Second
number	Required. The number of interval you want to add. Can either be positive, for dates in the future, or negative, for dates in the past
date	Required. Variant or literal representing the date to which interval is added

Example 1

```
'Add one month to January 31, 2000
document.write(DateAdd("m",1,"31-Jan-00"))
Output:
2/29/2000
```

Example 2

```
'Add one month to January 31, 2001
document.write(DateAdd("m",1,"31-Jan-01"))
Output:
2/28/2001
```

Example 3

```
'Subtract one month from January 31, 2001
document.write(DateAdd("m",-1,"31-Jan-01"))
Output:
12/31/2000
```

The **DateDiff** function returns the number of intervals between two dates.

Syntax

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

Parameter	Description
interval	Required. The interval you want to use to calculate the differences between

	date1 and date2 Can take the following values: • yyyy - Year • q - Quarter • m - Month • y - Day of year • d - Day • w - Weekday • ww - Week of year • h - Hour • n - Minute • s - Second
date1,date2	Required. Date expressions. Two dates you want to use in the calculation
firstdayofweek	Optional. Specifies the day of the week. Can take the following values: • 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting • 1 = vbSunday - Sunday (default) • 2 = vbMonday - Monday • 3 = vbTuesday - Tuesday • 4 = vbWednesday - Wednesday • 5 = vbThursday - Thursday • 6 = vbFriday - Friday • 7 = vbSaturday - Saturday
firstweekofyear	Optional. Specifies the first week of the year. 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 1

```
document.write(Date & "<br />")
document.write(DateDiff("m",Date,"12/31/2002") & "<br />")
document.write(DateDiff("d",Date,"12/31/2002") & "<br />")
document.write(DateDiff("n",Date,"12/31/2002"))
Output:
1/14/2002
11
351
505440
```

Example 2

```
document.write(Date & "<br />")
'Note that in the code below
'is date1>date2
document.write(DateDiff("d","12/31/2002",Date))
Output:
1/14/2002
-351
```

Example 3

```
'How many weeks (start on Monday),
'are left between the current date and 10/10/2002
document.write(Date & "<br />")
document.write(DateDiff("w",Date,"10/10/2002",vbMonday))
Output:
1/14/2002
38
```

The **DatePart** function returns the specified part of a given date.

Syntax

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

Parameter	Description
interval	Required. The interval of time to return. Can take the following values: • yyyy - Year • q - Quarter • m - Month • y - Day of year • d - Day • w - Weekday • ww - Week of year • h - Hour • n - Minute • s - Second
date	Required. Date expression to evaluate
firstdayofweek	Optional. Specifies the day of the week. Can take the following values: • 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting • 1 = vbSunday - Sunday (default) • 2 = vbMonday - Monday • 3 = vbTuesday - Tuesday • 4 = vbWednesday - Wednesday • 5 = vbThursday - Thursday • 6 = vbFriday - Friday • 7 = vbSaturday - Saturday
firstweekofyear	Optional. Specifies the first week of the year.

	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 1

```
document.write(Date & "<br />")
document.write(DatePart("d",Date))
Output:
1/14/2002
14
```

Example 2

```
document.write(Date & "<br />")
document.write(DatePart("w",Date))
Output:
1/14/2002
2
```

The **DateSerial** function returns a Variant of subtype Date for a specified year, month, and day.

Syntax

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

Parameter	Description
year	Required. A number between 100 and 9999, or a numeric expression. Values between 0 and 99 are interpreted as the years 1900–1999. For all other year arguments, use a complete four-digit year
month	Required. Any numeric expression
day	Required. Any numeric expression

Example 1

```
document.write(DateSerial(1996,2,3) & "<br />")
document.write(DateSerial(1990-20,9-2,1-1))
Output:
2/3/1996
6/30/1970
```

The **DateValue** function returns a type Date.

Note: If the year part of date is omitted this function will use the current year from the computer's system date.

Note: If the date parameter includes time information it will not be returned. However, if date includes invalid time information, a run-time error will occur.

Syntax

```
DateValue(date)
```

Parameter	Description
date	Required. A date from January 1, 100 through December 31, 9999 or any expression that can represent a date, a time, or both a date and time

Example 1

```
document.write(DateValue("31-Jan-02") & "<br />")
document.write(DateValue("31-Jan") & "<br />")
document.write(DateValue("31-Jan-02 2:39:49 AM"))
Output:
1/31/2002
1/31/2002
1/31/2002
```

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

Syntax

```
Day(date)
```

Parameter	Description
date	Required. Any expression that can represent a date

Example 1

```
document.write(Date & "<br />")
document.write(Day(Date))
Output:
1/14/2002
14
```

The **FormatDateTime** function formats and returns a valid date or time expression.

Syntax

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

Parameter	Description
date	Required. Any valid date expression (like Date() or Now())
format	Optional. A Format value that specifies the date/time format to use

Example 1

```
document.write("The current date is: ")
document.write(FormatDateTime(Date()))
Output:
The current date is: 2/22/2001
```

Example 2

```
document.write("The current date is: ")
```

```
document.write(FormatDateTime(Date(),1))
Output:
The current date is: Thursday, February 22, 2001
```

Example 3

```
document.write("The current date is: ")
document.write(FormatDateTime(Date(),2))
Output:
The current date is: 2/22/2001
```

Format Values

Constant	Value	Description
vbGeneralDate	0	Display a date in format mm/dd/yy. If the date parameter is Now(), it will also return the time, after the date
vbLongDate	1	Display a date using the long date format: weekday, month day, year
vbShortDate	2	Display a date using the short date format: like the default (mm/dd/yy)
vbLongTime	3	Display a time using the time format: hh:mm:ss PM/AM
vbShortTime	4	Display a time using the 24-hour format: hh:mm

The **Hour** function returns a number between 0 and 23 that represents the hour of the day.

Syntax

```
Hour(time)
```

Parameter	Description
time	Required. Any expression that can represent a time

Example 1

```
document.write(Now & "<br />")
document.write(Hour(Now))
Output:
1/15/2002 10:07:47 AM
10
```

Example 2

```
document.write(Hour(Time))
Output:
10
```

The **IsDate** function returns a Boolean value that indicates if the evaluated expression can be converted to a date. It returns True if the expression is a date or can be converted to a date; otherwise, it returns False.

Note: The IsDate function uses local setting to determine if a string can be converted to a date ("January" is not a month in all languages.)

Syntax

```
IsDate(expression)
```

Parameter	Description
expression	Required. The expression to be evaluated

Example 1

```
document.write(IsDate("April 22, 1947") & "<br />")
document.write(IsDate(#11/11/01#) & "<br />")
document.write(IsDate("#11/11/01#") & "<br />")
document.write(IsDate("Hello World!"))
Output:
True
True
False
False
```

The **Minute** function returns a number between 0 and 59 that represents the minute of the hour.

Syntax

```
Minute(time)
```

Parameter	Description
time	Required. Any expression that can represent a time

Example 1

```
document.write(Now & "<br />")
document.write(Minute(Now))
Output:
1/15/2002 10:34:39 AM
34
```

Example 2

```
document.write(Minute(Time))
Output:
34
```

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

Syntax

```
Month(date)
```

Parameter	Description
date	Required. Any expression that can represent a date

Example 1

```
document.write(Date & "<br />")
document.write(Month(Date))
Output:
1/15/2002
1
```

The **MonthName** function returns the name of the specified month.

Syntax

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

Parameter	Description
month	Required. Specifies the number of the month (January is 1, February is 2, etc.)
abbreviate	Optional. A Boolean value that indicates if the month name is to be abbreviated. Default is False

Example 1

```
document.write(MonthName(8))
Output:
August
```

Example 2

```
document.write(MonthName(8,true))
Output:
Aug
```

The **Now** function returns the current date and time according to the setting of your computer's system date and time.

Syntax

```
Now
```

Example 1

```
document.write(Now)
Output:
1/15/2002 10:52:15 AM
```

The **Second** function returns a number between 0 and 59 that represents the second of the minute.

Syntax

```
Second(time)
```

Parameter	Description
time	Required. Any expression that can represent a time

Example 1

```
document.write(Now & "<br />")
document.write(Second(Now))
Output:
1/15/2002 10:55:51 AM
51
```

Example 2

```
document.write(Second(Time))
Output:
51
```

The **Time** function returns the current system time.

Syntax

```
Time
```

Example 1

```
document.write(Time)
Output:
11:07:27 AM
```

The **Timer** function returns the number of seconds since 12:00 AM.

Syntax

```
Timer
```

Example 1

```
document.write(Time & "<br />")
document.write(Timer)
Output:
11:11:13 AM
40273.2
```

The **TimeSerial** function returns the time for a specific hour, minute, and second.

Syntax

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

Parameter	Description
hour	Required. A number between 0 and 23, or a numeric expression
minute	Required. Any numeric expression
second	Required. Any numeric expression

Example 1

```
document.write(TimeSerial(23,2,3) & "<br />")
document.write(TimeSerial(0,9,11) & "<br />")
document.write(TimeSerial(14+2,9-2,1-1))
Output:
11:02:03 PM
12:09:11 AM
4:07:00 PM
```

The **TimeValue** function returns a Variant of subtype Date that contains the time.

Syntax

```
TimeValue(time)
```

Parameter	Description
time	Required. A time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.) or any expression that represents a time in that range

Example 1

```
document.write(TimeValue("5:55:59 PM") & "<br />")
document.write(TimeValue(#5:55:59 PM#) & "<br />")
document.write(TimeValue("15:34"))
Output:
5:55:59 PM
5:55:59 PM
3:34:00 PM
```

The **Weekday** function returns a number between 1 and 7, that represents the day of the week.

Syntax

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

Parameter	Description
date	Required. The date expression to evaluate
firstdayofweek	Optional. Specifies the first day of the week. Can take the following values: • 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting • 1 = vbSunday - Sunday (default) • 2 = vbMonday - Monday • 3 = vbTuesday - Tuesday • 4 = vbWednesday - Wednesday • 5 = vbThursday - Thursday • 6 = vbFriday - Friday • 7 = vbSaturday - Saturday

Example 1

```
document.write(Date & "<br />")
document.write(Weekday(Date))
Output:
1/15/2002
3
```

The **WeekdayName** function returns the weekday name of a specified day of the week.

Syntax

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

Parameter	Description
weekday	Required. The number of the weekday
abbreviate	Optional. A Boolean value that indicates if the weekday name is to be abbreviated
firstdayofweek	Optional. Specifies the first day of the week. Can take the following values: • 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting • 1 = vbSunday - Sunday (default) • 2 = vbMonday - Monday • 3 = vbTuesday - Tuesday • 4 = vbWednesday - Wednesday • 5 = vbThursday - Thursday • 6 = vbFriday - Friday • 7 = vbSaturday - Saturday

Example 1

```
document.write(WeekdayName(3))
Output:
Tuesday
```

Example 2

```
document.write(Date & "<br />")
document.write(Weekday(Date) & "<br />")
document.write(WeekdayName(Weekday(Date)))
Output:
1/15/2002
3
Tuesday
```

Example 3

```
document.write(Date & "<br />")
document.write(Weekday(Date) & "<br />")
document.write(WeekdayName(Weekday(Date),true))
Output:
1/15/2002
3
Tue
```

The **Year** function returns a number that represents the year.

Syntax

```
Year(date)
```

Parameter	Description
date	Required. Any expression that can represent a date

Example 1

```
document.write(Date & "<br />")
document.write(Year(Date))
Output:
1/15/2002
2002
```

Conversion Functions

The **Asc** function converts the first letter in a string to ANSI code, and returns the result.

Syntax

```
Asc(string)
```

Parameter	Description
string	Required. A string expression. Cannot be an empty string!

Example 1

```
document.write(Asc("A") & "<br />")
document.write(Asc("F"))
Output:
65
70
```

Example 2

```
document.write(Asc("a") & "<br />")
document.write(Asc("f"))
Output:
97
102
```

Example 3

```
document.write(Asc("M") & "<br />")
document.write(Asc("Madhavendra"))
Output:
77
77
```

Example 4

```
document.write(Asc("2") & "<br />")
document.write(Asc("#"))
Output:
50
35
```

The **CBool** function converts an expression to type Boolean.

Syntax

```
CBool(expression)
```

Parameter	Description
expression	Required. Any valid expression. A nonzero value returns True, zero returns False. A run-time error occurs if the expression can not be interpreted as a numeric value

Example 1

```
dim a,b
a=5
b=10
document.write(CBool(a) & "<br />")
document.write(CBool(b))
Output:
True
True
```

The **CByte** function converts an expression to type Byte.

Syntax

```
CByte(expression)
```

Parameter	Description
expression	Required. Any valid expression

Example 1

```
dim a
a=134.345
document.write(CByte(a))
Output:
134
```

Example 2

```
dim a
a=14.345455
document.write(CByte(a))
Output:
14
```

The **CCur** function converts an expression to type Currency.

Syntax

```
CCur(expression)
```

Parameter	Description
expression	Required. Any valid expression

Example 1

```
dim a
a=134.345
document.write(CCur(a))
```

```
Output:
134.345
```

Example 2

```
dim a
a=1411111111.345455
'NB! This function rounds off to 4 decimal places
document.write(CCur(a))
Output:
1411111111.3455
```

The **CDate** function converts a valid date and time expression to type Date, and returns the result.

Tip: Use the IsDate function to determine if date can be converted to a date or time.

Note: The IsDate function uses local setting to determine if a string can be converted to a date ("January" is not a month in all languages.)

Syntax

```
CDate(date)
```

Parameter	Description
date	Required. Any valid date expression (like Date() or Now())

Example 1

```
d="April 22, 2001"
if IsDate(d) then
  document.write(CDate(d))
end if
Output:
2/22/01
```

Example 2

```
d=#2/22/01#
if IsDate(d) then
  document.write(CDate(d))
end if
Output:
2/22/01
```

Example 3

```
d="3:18:40 AM"
if IsDate(d) then
  document.write(CDate(d))
end if
Output:
3:18:40 AM
```

The **CDbl** function converts an expression to type Double.

Syntax

```
CDbl(expression)
```

Parameter	Description
expression	Required. Any valid expression

Example 1

```
dim a
a=134.345
document.write(CDbl(a))
Output:
134.345
```

Example 2

```
dim a
a=14111111113353355.345455
document.write(CDbl(a))
Output:
1.41111111133534E+16
```

The **Chr** function converts the specified ANSI character code to a character.

Note: The numbers from 0 to 31 represents nonprintable ASCII codes, i.e. Chr(10) will return a linefeed character.

Syntax

```
Chr(charcode)
```

Parameter	Description
charcode	Required. A number that identifies a character

Example 1

```
document.write(Chr(65) & "<br />")
document.write(Chr(97))
Output:
A
a
```

Example 2

```
document.write(Chr(37) & "<br />")
document.write(Chr(45))
Output:
%
-
```

Example 3

```
document.write(Chr(50) & "<br />")
document.write(Chr(35))
Output:
2
#
```

The **CInt** function converts an expression to type Integer.

Note: The value must be a number between -32768 and 32767.

Syntax

```
CInt(expression)
```

Parameter	Description
expression	Required. Any valid expression

Example 1

```
dim a
a=134.345
document.write(CInt(a))
Output:
134
```

Example 2

```
dim a
a=-30000.24
document.write(CInt(a))
Output:
-30000
```

The **CLng** function converts an expression to type Long.

Note: The value must be a number between -2147483648 and 2147483647.

Syntax

```
CLng(expression)
```

Parameter	Description
expression	Required. Any valid expression

Example 1

```
dim a,b
a=23524.45
b=23525.55
document.write(CLng(a) & "<br />")
document.write(CLng(b))
Output:
23524
23526
```

The **CSng** function converts an expression to type Single.

Syntax

```
CSng(expression)
```

Parameter	Description
expression	Required. Any valid expression

Example 1

```
dim a,b
a=23524.4522
b=23525.5533
document.write(CSng(a) & "<br />")
document.write(CSng(b))
Output:
23524.45
23525.55
```

The **CStr** function converts an expression to type String.

Syntax

```
CStr(expression)
```

Parameter	Description
expression	Required. Any valid expression If expression is: • Boolean - then the CStr function will return a string containing true or false. • Date - then the CStr function will return a string that contains a date in the short-date format. • Null - then a run-time error will occur. • Empty - then the CStr function will return an empty string (""). • Error - then the CStr function will return a string that contains the word "Error" followed by an error number. • Other numeric - then the CStr function will return a string that contains the number.

Example 1

```
dim a
a=false
document.write(CStr(a))
Output:
false
```

Example 2

```
dim a
a=#01/01/01#
document.write(CStr(a))
Output:
1/1/2001
```

The **Hex** function returns a string that represents the hexadecimal value of a specified number.

Note: If number is not a whole number, it is rounded to the nearest whole number before being evaluated.

Syntax

```
Hex(number)
```

Parameter	Description
number	Required. Any valid expression If number is: • Null - then the Hex function returns Null. • Empty - then the Hex function returns zero (0). • Any other number - then the Hex function returns up to eight hexadecimal characters.

Example 1

```
document.write(Hex(3) & "<br />")
document.write(Hex(5) & "<br />")
document.write(Hex(9) & "<br />")
document.write(Hex(10) & "<br />")
document.write(Hex(11) & "<br />")
document.write(Hex(12) & "<br />")
document.write(Hex(400) & "<br />")
document.write(Hex(459) & "<br />")
document.write(Hex(460))
Output:
3
5
9
A
B
C
190
1CB
1CC
```

The **Oct** function returns a string that represents the octal value of a specified number.

Note: If number is not already a whole number, it is rounded to the nearest whole number before being evaluated.

Syntax

```
Oct(number)
```

Parameter	Description
number	Required. Any valid expression If number is: • Null - then the Oct function returns Null. • Empty - then the Oct function returns zero (0). • Any other number - then the Oct function returns up to 11 octal

	characters.

Example 1

```
document.write(Oct(3) & "<br />")
document.write(Oct(5) & "<br />")
document.write(Oct(9) & "<br />")
document.write(Oct(10) & "<br />")
document.write(Oct(11) & "<br />")
document.write(Oct(12) & "<br />")
document.write(Oct(400) & "<br />")
document.write(Oct(459) & "<br />")
document.write(Oct(460))
Output:
3
5
11
12
13
14
620
713
714
```

Format Functions

The **FormatCurrency** function returns an expression formatted as a currency value using the currency symbol defined in the computer's control panel.

Syntax

```
FormatCurrency(Expression[,NumDigAfterDec[,
IncLeadingDig[,UseParForNegNum[,GroupDig]]]])
```

Parameter	Description
expression	Required. The expression to be formatted
NumDigAfterDec	Optional. Indicates how many places to the right of the decimal are displayed. Default is -1 (the computer's regional settings are used)
IncLeadingDig	Optional. Indicates whether or not a leading zero is displayed for fractional values: • -2 = TristateUseDefault - Use the computer's regional settings • -1 = TristateTrue - True • 0 = TristateFalse - False
UseParForNegNum	Optional. Indicates whether or not to place negative values within parentheses: • -2 = TristateUseDefault - Use the computer's regional settings • -1 = TristateTrue - True • 0 = TristateFalse - False
GroupDig	Optional. Indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings: • -2 = TristateUseDefault - Use the computer's regional settings • -1 = TristateTrue - True

	• 0 = TristateFalse - False

Example 1

```
document.write(FormatCurrency(20000))
Output:
$20,000.00
```

Example 2

```
document.write(FormatCurrency(20000.578,2))
Output:
$20,000.58
```

Example 3

```
document.write(FormatCurrency(20000.578,2,,,0))
Output:
$20000.58
```

The **FormatDateTime** function formats and returns a valid date or time expression.

Syntax

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

Parameter	Description
date	Required. Any valid date expression (like Date() or Now())
format	Optional. A Format value that specifies the date/time format to use

Example 1

```
document.write("The current date is: ")
document.write(FormatDateTime(Date()))
Output:
The current date is: 2/22/2001
```

Example 2

```
document.write("The current date is: ")
document.write(FormatDateTime(Date(),1))
Output:
The current date is: Thursday, February 22, 2001
```

Example 3

```
document.write("The current date is: ")
document.write(FormatDateTime(Date(),2))
Output:
The current date is: 2/22/2001
```

Format Values

Constant	Value	Description
vbGeneralDate	0	Display a date in format mm/dd/yy. If the date parameter is Now(), it will also return the time, after the date
vbLongDate	1	Display a date using the long date format: weekday, month day, year
vbShortDate	2	Display a date using the short date format: like the default (mm/dd/yy)
vbLongTime	3	Display a time using the time format: hh:mm:ss PM/AM
vbShortTime	4	Display a time using the 24-hour format: hh:mm

The **FormatNumber** function returns an expression formatted as a number.

Syntax

```
FormatNumber(Expression[,NumDigAfterDec[,
IncLeadingDig[,UseParForNegNum[,GroupDig]]]])
```

Parameter	Description
expression	Required. The expression to be formatted
NumDigAfterDec	Optional. Indicates how many places to the right of the decimal are displayed. Default is -1 (the computer's regional settings are used)
IncLeadingDig	Optional. Indicates whether or not a leading zero is displayed for fractional values: • -2 = TristateUseDefault - Use the computer's regional settings • -1 = TristateTrue - True • 0 = TristateFalse - False
UseParForNegNum	Optional. Indicates whether or not to place negative values within parentheses: • -2 = TristateUseDefault - Use the computer's regional settings • -1 = TristateTrue - True • 0 = TristateFalse - False
GroupDig	Optional. Indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings: • -2 = TristateUseDefault - Use the computer's regional settings • -1 = TristateTrue - True • 0 = TristateFalse - False

Example 1

```
document.write(FormatNumber(20000))
Output:
20,000.00
```

Example 2

```
document.write(FormatNumber(20000.578,2))
Output:
20,000.58
```

Example 3

```
document.write(FormatNumber(20000.578,2,,,0))
Output:
20000.58
```

The **FormatPercent** function returns an expression formatted as a percentage (multiplied by 100) with a trailing % character.

Syntax

```
FormatPercent(Expression[,NumDigAfterDec[,
IncLeadingDig[,UseParForNegNum[,GroupDig]]]])
```

Parameter	Description
expression	Required. The expression to be formatted
NumDigAfterDec	Optional. Indicates how many places to the right of the decimal are displayed. Default is -1 (the computer's regional settings are used)
IncLeadingDig	Optional. Indicates whether or not a leading zero is displayed for fractional values: • -2 = TristateUseDefault - Use the computer's regional settings • -1 = TristateTrue - True • 0 = TristateFalse - False
UseParForNegNum	Optional. Indicates whether or not to place negative values within parentheses: • -2 = TristateUseDefault - Use the computer's regional settings • -1 = TristateTrue - True • 0 = TristateFalse - False
GroupDig	Optional. Indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings: • -2 = TristateUseDefault - Use the computer's regional settings • -1 = TristateTrue - True • 0 = TristateFalse - False

Example 1

```
'How many percent is 6 of 345?
document.write(FormatPercent(6/345))
Output:
1.74%
```

Example 2

```
'How many percent is 6 of 345?
document.write(FormatPercent(6/345,1))
Output:
1.7%
```

Math Functions

The **Abs** function returns the absolute value of a specified number.

Note: If the number parameter contains Null, Null will be returned

Note: If the number parameter is an uninitialized variable, zero will be returned.

Syntax

```
Abs(number)
```

Parameter	Description
number	Required. A numeric expression

Example 1

```
document.write(Abs(1) & "<br />")
document.write(Abs(-1))
Output:
1
1
```

Example 2

```
document.write(Abs(48.4) & "<br />")
document.write(Abs(-48.4))
Output:
48.4
48.4
```

The **Atn** function returns the arctangent of a specified number.

Syntax

```
Atn(number)
```

Parameter	Description
number	Required. A numeric expression

Example 1

```
document.write(Atn(89))
Output:
1.55956084453693
```

Example 2

```
document.write(Atn(8.9))
Output:
1.45890606062322
```

Example 3

```
'calculate the value of pi
dim pi
pi=4*Atn(1)
document.write(pi)
Output:
3.14159265358979
```

The **Cos** function returns the cosine of a specified number (angle).

Syntax

```
Cos(number)
```

Parameter	Description
number	Required. A numeric expression that expresses an angle in radians

Example 1

```
document.write(Cos(50.0))
Output:
0.964966028492113
```

Example 2

```
document.write(Cos(-50.0))
Output:
0.964966028492113
```

The **Exp** function returns *e* raised to a power.

Note: The value of number cannot exceed 709.782712893.

Tip: Also look at the Log function.

Syntax

```
Exp(number)
```

Parameter	Description
number	Required. A valid numeric expression

Example 1

```
document.write(Exp(6.7))
Output:
812.405825167543
```

Example 2

```
document.write(Exp(-6.7))
Output:
1.23091190267348E-03
```

The **Hex** function returns a string that represents the hexadecimal value of a specified number.

Note: If number is not a whole number, it is rounded to the nearest whole number before being evaluated.

Syntax

```
Hex(number)
```

Parameter	Description
number	Required. Any valid expression If number is: • Null - then the Hex function returns Null. • Empty - then the Hex function returns zero (0). • Any other number - then the Hex function returns up to eight hexadecimal characters.

Example 1

```
document.write(Hex(3) & "<br />")
document.write(Hex(5) & "<br />")
document.write(Hex(9) & "<br />")
document.write(Hex(10) & "<br />")
document.write(Hex(11) & "<br />")
document.write(Hex(12) & "<br />")
document.write(Hex(400) & "<br />")
document.write(Hex(459) & "<br />")
document.write(Hex(460))
Output:
3
5
9
A
B
C
190
1CB
1CC
```

The **Int** function returns the integer part of a specified number.

Note: If the number parameter contains Null, Null will be returned.

Tip: Also look at the Fix function.

Syntax

```
Int(number)
```

Parameter	Description
number	Required. A valid numeric expression

Example 1

```
document.write(Int(6.83227))
Output:
6
```

Example 2

```
document.write(Int(6.23443))
Output:
6
```

Example 3

```
document.write(Int(-6.13443))
Output:
-7
```

Example 4

```
document.write(Int(-6.93443))
Output:
-7
```

The **Fix** function returns the integer part of a specified number.

Note: If the number parameter contains Null, Null will be returned.

Tip: Also look at the Int function.

Syntax

```
Fix(number)
```

Parameter	Description
number	Required. A valid numeric expression

Example 1

```
document.write(Fix(6.83227))
Output:
6
```

Example 2

```
document.write(Fix(6.23443))
Output:
6
```

Example 3

```
document.write(Fix(-6.13443))
Output:
-6
```

Example 4

```
document.write(Fix(-6.93443))
Output:
-6
```

The **Log** function returns the natural logarithm of a specified number. The natural logarithm is the logarithm to the base e.

Note: Negative values are not allowed.

Tip: Also look at the Exp function.

Syntax

```
Log(number)
```

Parameter	Description
number	Required. A valid numeric expression > 0

Example 1

```
document.write(Log(38.256783227))
Output:
3.64432088381777
```

The **Oct** function returns a string that represents the octal value of a specified number.

Note: If number is not already a whole number, it is rounded to the nearest whole number before being evaluated.

Syntax

```
Oct(number)
```

Parameter	Description
number	Required. Any valid expression If number is: • Null - then the Oct function returns Null. • Empty - then the Oct function returns zero (0). • Any other number - then the Oct function returns up to 11 octal characters.

Example 1

```
document.write(Oct(3) & "<br />")
document.write(Oct(5) & "<br />")
document.write(Oct(9) & "<br />")
document.write(Oct(10) & "<br />")
document.write(Oct(11) & "<br />")
document.write(Oct(12) & "<br />")
document.write(Oct(400) & "<br />")
document.write(Oct(459) & "<br />")
document.write(Oct(460))
Output:
3
5
11
12
13
14
620
713
714
```

The **Rnd** function returns a random number. The number is always less than 1 but greater or equal to 0.

Syntax

```
Rnd[(number)]
```

Parameter	Description
number	Optional. A valid numeric expression If number is: • <0 - Rnd returns the same number every time • >0 - Rnd returns the next random number in the sequence • =0 - Rnd returns the most recently generated number • Not supplied - Rnd returns the next random number in the sequence

Example 1

```
document.write(Rnd)
Output:
0.7055475
```

Example 2

```
'If you refresh the page,
'using the code in example 1,
'the SAME random number will show over and over.
'Use the Randomize statement generate a new random number
'each time the page is reloaded!
Randomize
document.write(Rnd)
Output:
0.4758112
```

Example 3

```
'Here is how to produce random integers in a
'given range:
dim max,min
max=100
min=1
document.write(Int((max-min+1)*Rnd+min))
Output:
71
```

The **Sgn** function returns an integer that indicates the sign of a specified number.

Syntax

```
Sgn(number)
```

Parameter	Description
number	Required. A valid numeric expression If number is: • >0 - Sgn returns 1 • =0 - Sgn returns 0

	• <0 - Sgn returns -1

Example 1

```
document.write(Sgn(15))
Output:
1
```

Example 2

```
document.write(Sgn(-5.67))
Output:
-1
```

Example 3

```
document.write(Sgn(0))
Output:
0
```

The **Sin** function returns the sine of a specified number (angle).

Syntax

```
Sin(number)
```

Parameter	Description
number	Required. A valid numeric expression that expresses an angle in radians

Example 1

```
document.write(Sin(47))
Output:
0.123573122745224
```

Example 2

```
document.write(Sin(-47))
Output:
-0.123573122745224
```

The **Sqr** function returns the square root of a number.

Note: The number parameter cannot be a negative value.

Syntax

```
Sqr(number)
```

Parameter	Description
number	Required. A valid numeric expression >= 0

Example 1

```
document.write(Sqr(9))
Output:
3
```

Example 2

```
document.write(Sqr(0))
Output:
0
```

Example 3

```
document.write(Sqr(47))
Output:
6.85565460040104
```

The **Tan** function returns the tangent of a specified number (angle).

Syntax

```
Tan(number)
```

Parameter	Description
number	Required. A valid numeric expression that expresses an angle in radians

Example 1

```
document.write(Tan(40))
Output:
-1.1172149309239
```

Example 2

```
document.write(Tan(40))
Output:
1.1172149309239
```

Array Functions

The **Array** function returns a variant containing an array.

Note: The first element in the array is zero.

Syntax

```
Array(arglist)
```

Parameter	Description
arglist	Required. A list (separated by commas) of values that is the elements in the array

Example 1

```
dim a
```

```
a=Array(5,10,15,20)
document.write(a(3))
Output:
20
```

Example 2

```
dim a
a=Array(5,10,15,20)
document.write(a(0))
Output:
5
```

The **Filter** function returns a zero-based array that contains a subset of a string array based on a filter criteria.

Note: If no matches of the value parameter are found, the Filter function will return an empty array.

Note: If the parameter inputstrings is Null or is NOT a one-dimensional array, an error will occur.

Syntax

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

Parameter	Description
inputstrings	Required. A one-dimensional array of strings to be searched
value	Required. The string to search for
include	Optional. A Boolean value that indicates whether to return the substrings that include or exclude value. True returns the subset of the array that contains value as a substring. False returns the subset of the array that does not contain value as a substring. Default is True.
compare	Optional. Specifies the string comparison to use. Can have one of the following values: • 0 = vbBinaryCompare - Perform a binary comparison • 1 = vbTextCompare - Perform a textual comparison

Example 1

```
dim a(5),b
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
b=Filter(a,"n")
document.write(b(0) & "<br />")
document.write(b(1) & "<br />")
document.write(b(2))
Output:
Sunday
Monday
Wednesday
```

Example 2

```
dim a(5),b
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
b=Filter(a,"n",false)
document.write(b(0) & "<br />")
document.write(b(1) & "<br />")
document.write(b(2))
Output:
Saturday
Tuesday
```

The **IsArray** function returns a Boolean value that indicates whether a specified variable is an array. If the variable is an array, it returns True, otherwise, it returns False.

Syntax

```
IsArray(variable)
```

Parameter	Description
variable	Required. Any variable

Example 1

```
dim a(5)
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
document.write(IsArray(a))
Output:
True
```

Example 2

```
dim a
a="Saturday"
document.write(IsArray(a))
Output:
False
```

The **Join** function returns a string that consists of a number of substrings in an array.

Syntax

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

Parameter	Description
list	Required. A one-dimensional array that contains the substrings to be joined
delimiter	Optional. The character(s) used to separate the substrings in the returned string. Default is the space character

Example 1

```
dim a(5),b
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
b=Filter(a,"n")
document.write(join(b))
Output:
Sunday Monday Wednesday
```

Example 2

```
dim a(5),b
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
b=Filter(a,"n")
document.write(join(b,", "))
Output:
Sunday, Monday, Wednesday
```

The **LBound** function returns the smallest subscript for the indicated dimension of an array.

Note: The LBound for any dimension is ALWAYS 0.

Tip: Use the LBound function with the UBound function to determine the size of an array.

Syntax

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

Parameter	Description
arrayname	Required. The name of the array variable
dimension	Optional. Which dimension's lower bound to return. 1 = first dimension, 2 = second dimension, and so on. Default is 1

Example 1

```
dim a(10)
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
a(5)="Thursday"
document.write(UBound(a))
document.write("<br />")
document.write(LBound(a))
Output:
10
0
```

The **Split** function returns a zero-based, one-dimensional array that contains a specified number of substrings.

Syntax

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

Parameter	Description
expression	Required. A string expression that contains substrings and delimiters
delimiter	Optional. A string character used to identify substring limits. Default is the space character
count	Optional. The number of substrings to be returned. -1 indicates that all substrings are returned
compare	Optional. Specifies the string comparison to use. Can have one of the following values: • 0 = vbBinaryCompare - Perform a binary comparison • 1 = vbTextCompare - Perform a textual comparison

Example 1

```
dim txt,a
txt="Hello World!"
a=Split(txt)
document.write(a(0) & "<br />")
document.write(a(1))
Output:
Hello
World!
```

The **UBound** function returns the largest subscript for the indicated dimension of an array.

Tip: Use the UBound function with the LBound function to determine the size of an array.

Syntax

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

Parameter	Description
arrayname	Required. The name of the array variable
dimension	Optional. Which dimension's upper bound to return. 1 = first dimension, 2 = second dimension, and so on. Default is 1

Example 1

```
dim a(10)
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
a(5)="Thursday"
document.write(UBound(a))
document.write("<br />")
document.write(LBound(a))
Output:
10
0
```

String Functions

The **InStr** function returns the position of the first occurrence of one string within another.

The InStr function can return the following values:

- If string1 is "" - InStr returns 0
- If string1 is Null - InStr returns Null
- If string2 is "" - InStr returns start
- If string2 is Null - InStr returns Null
- If string2 is not found - InStr returns 0
- If string2 is found within string1 - InStr returns the position at which match is found
- If start > Len(string1) - InStr returns 0

Tip: Also look at the InStrRev function

Syntax

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

Parameter	Description
start	Optional. Specifies the starting position for each search. The search begins at the first character position by default. This parameter is required if compare is specified
string1	Required. The string to be searched
string2	Required. The string expression to search for
compare	Optional. Specifies the string comparison to use. Default is 0 Can have one of the following values: • 0 = vbBinaryCompare - Perform a binary comparison • 1 = vbTextCompare - Perform a textual comparison

Example 1

```
dim txt,pos
txt="This is a beautiful day!"
pos=InStr(txt,"his")
document.write(pos)
Output:
2
```

Example 2

```
dim txt,pos
txt="This is a beautiful day!"
'A textual comparison starting at position 4
pos=InStr(4,txt,"is",1)
document.write(pos)
Output:
6
```

Example 3

```
dim txt,pos
txt="This is a beautiful day!"
```

```
'A binary comparison starting at position 1
pos=InStr(1,txt,"B",0)
document.write(pos)
Output:
0
```

The **InStrRev** function returns the position of the first occurrence of one string within another. The search begins from the end of string, but the position returned counts from the beginning of the string.

The InStrRev function can return the following values:

- If string1 is "" - InStrRev returns 0
- If string1 is Null - InStrRev returns Null
- If string2 is "" - InStrRev returns start
- If string2 is Null - InStrRev returns Null
- If string2 is not found - InStrRev returns 0
- If string2 is found within string1 - InStrRev returns the position at which match is found
- If start > Len(string1) - InStrRev returns 0

Tip: Also look at the InStr function

Syntax

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

Parameter	Description
string1	Required. The string to be searched
string2	Required. The string expression to search for
start	Optional. Specifies the starting position for each search. The search begins at the last character position by default (-1)
compare	Optional. Specifies the string comparison to use. Default is 0 Can have one of the following values: • 0 = vbBinaryCompare - Perform a binary comparison • 1 = vbTextCompare - Perform a textual comparison

Example 1

```
dim txt,pos
txt="This is a beautiful day!"
pos=InStrRev(txt,"his")
document.write(pos)
Output:
2
```

Example 2

```
dim txt,pos
txt="This is a beautiful day!"
'textual comparison
pos=InStrRev(txt,"B",-1,1)
document.write(pos)
Output:
11
```

Example 3

```
dim txt,pos
txt="This is a beautiful day!"
'binary comparison
pos=InStrRev(txt,"T")
document.write(pos)
Output:
1
```

Example 4

```
dim txt,pos
txt="This is a beautiful day!"
'binary comparison
pos=InStrRev(txt,"t")
document.write(pos)
Output:
15
```

The **Left** function returns a specified number of characters from the left side of a string.

Tip: Use the Len function to find the number of characters in a string.

Tip: Also look at the Right function.

Syntax

```
Left(string,length)
```

Parameter	Description
string	Required. The string to return characters from
length	Required. Specifies how many characters to return. If set to 0, an empty string ("") is returned. If set to greater than or equal to the length of the string, the entire string is returned

Example 1

```
dim txt
txt="This is a beautiful day!"
document.write(Left(txt,11))
Output:
This is a b
```

Example 2

```
dim txt
txt="This is a beautiful day!"
document.write(Left(txt,100))
Output:
This is a beautiful day!
```

Example 3

```
dim txt,x
txt="This is a beautiful day!"
x=Len(txt)
```

```
document.write(Left(txt,x))
Output:
This is a beautiful day!
```

The **Len** function returns the number of characters in a string.

Syntax

```
Len(string|varname)
```

Parameter	Description
string	A string expression
varname	A variable name

Example 1

```
dim txt
txt="This is a beautiful day!"
document.write(Len(txt))
Output:
24
```

Example 2

```
document.write(Len("This is a beautiful day!"))
Output:
24
```

The **LTrim** function removes spaces on the left side of a string.

Syntax

```
LTrim(string)
```

Parameter	Description
string	Required. A string expression

Example 1

```
dim txt
txt="  This is a beautiful day!  "
document.write(LTrim(txt))
Output:
"This is a beautiful day!  "
```

The **RTrim** function removes spaces on the right side of a string.

Syntax

```
RTrim(string)
```

Parameter	Description
string	Required. A string expression

Example 1

```
dim txt
txt="  This is a beautiful day!  "
document.write(RTrim(txt))
Output:
"  This is a beautiful day!"
```

The **Trim** function removes spaces on both sides of a string.

Syntax

```
Trim(string)
```

Parameter	Description
string	Required. A string expression

Example 1

```
dim txt
txt="  This is a beautiful day!  "
document.write(Trim(txt))
Output:
"This is a beautiful day!"
```

The **Mid** function returns a specified number of characters from a string.

Tip: Use the Len function to determine the number of characters in a string.

Syntax

```
Mid(string,start[,length])
```

Parameter	Description
string	Required. The string expression from which characters are returned
start	Required. Specifies the starting position. If set to greater than the number of characters in string, it returns an empty string ("")
length	Optional. The number of characters to return

Example 1

```
dim txt
txt="This is a beautiful day!"
document.write(Mid(txt,1,1))
Output:
T
```

Example 2

```
dim txt
txt="This is a beautiful day!"
document.write(Mid(txt,1,11))
Output:
This is a b
```

Example 3

```
dim txt
txt="This is a beautiful day!"
document.write(Mid(txt,1))
Output:
This is a beautiful day!
```

Example 4

```
dim txt
txt="This is a beautiful day!"
document.write(Mid(txt,10))
Output:
beautiful day!
```

The **Replace** function replaces a specified part of a string with another string a specified number of times.

Syntax

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

Parameter	Description
string	Required. The string to be searched
find	Required. The part of the string that will be replaced
replacewith	Required. The replacement substring
start	Optional. Specifies the start position. Default is 1
count	Optional. Specifies the number of substitutions to perform. Default value is -1, which means make all possible substitutions
compare	Optional. Specifies the string comparison to use. Default is 0 Can have one of the following values: • 0 = vbBinaryCompare - Perform a binary comparison • 1 = vbTextCompare - Perform a textual comparison

Example 1

```
dim txt
txt="This is a beautiful day!"
document.write(Replace(txt,"beautiful","horrible"))
Output:
This is a horrible day!
```

The **Right** function returns a specified number of characters from the right side of a string.

Tip: Use the Len function to find the number of characters in a string.

Tip: Also look at the Left function.

Syntax

```
Right(string,length)
```

Parameter	Description

string	Required. The string to return characters from
length	Required. Specifies how many characters to return. If set to 0, an empty string ("") is returned. If set to greater than or equal to the length of the string, the entire string is returned

Example 1

```
dim txt
txt="This is a beautiful day!"
document.write(Right(txt,11))
Output:
utiful day!
```

Example 2

```
dim txt
txt="This is a beautiful day!"
document.write(Right(txt,100))
Output:
This is a beautiful day!
```

Example 3

```
dim txt,x
txt="This is a beautiful day!"
x=Len(txt)
document.write(Right(txt,x))
Output:
This is a beautiful day!
```

The **Space** function returns a string that consists of a specified number of spaces.

Syntax

```
Space(number)
```

Parameter	Description
number	Required. The number of spaces you want in the string

Example 1

```
dim txt
txt=Space(10)
document.write(txt)
Output:
"          "
```

The **StrComp** function compares two strings and returns a value that represents the result of the comparison.

The StrComp function can return one of the following values:

- -1 (if string1 < string2)
- 0 (if string1 = string2)
- 1 (if string1 > string2)
- Null (if string1 or string2 is Null)

Syntax

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

Parameter	Description
string1	Required. A string expression
string2	Required. A string expression
compare	Optional. Specifies the string comparison to use. Default is 0 Can have one of the following values: • 0 = vbBinaryCompare - Perform a binary comparison • 1 = vbTextCompare - Perform a textual comparison

Example 1

```
document.write(StrComp("VBScript","VBScript"))
Output:
0
```

Example 2

```
document.write(StrComp("VBScript","vbscript"))
Output:
-1
```

Example 3

```
document.write(StrComp("VBScript","vbscript",1))
Output:
0
```

The **String** function returns a string that contains a repeating character of a specified length.

Syntax

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

Parameter	Description
number	Required. The length of the returned string
character	Required. The character that will be repeated

Example 1

```
document.write(String(10,"#"))
Output:
##########
```

Example 2

```
document.write(String(4,"*"))
Output:
****
```

Example 3

```
document.write(String(4,42))
Output:
****
```

Example 4

```
document.write(String(4,"XYZ"))
Output:
XXXX
```

The **StrReverse** function reverses a string.

Syntax

```
StrReverse(string)
```

Parameter	Description
string	Required. The string to be reversed

Example 1

```
dim txt
txt="This is a beautiful day!"
document.write(StrReverse(txt))
Output:
!yad lufituaeb a si sihT
```

Other Functions

The **CreateObject** function creates an object of a specified type.

Syntax

```
CreateObject(servername.typename[,location])
```

Parameter	Description
servername	Required. The name of the application that provides the object
typename	Required. The type/class of the object
location	Optional. Where to create the object

Example 1

```
dim myexcel
Set myexcel=CreateObject("Excel.Sheet")
myexcel.Application.Visible=True
...code...
myexcel.Application.Quit
Set myexcel=Nothing
```

The **GetLocale** function returns the current locale ID.

A locale contains a set of user preference information: like language, country, region, and cultural conventions. The locale determines such things as keyboard layout, sort order, date, time, number, and currency formats.

The return value can be one of the 32-bit values shown in the Locale ID chart.

Syntax

```
GetLocale()
```

Example 1

```
dim c
c=GetLocale
document.write(c)
Output:
1033
```

Locale ID Chart

Locale Description	Short String	Hex Value	Decimal Value
Afrikaans	af	0x0436	1078
Albanian	sq	0x041C	1052
Arabic – United Arab Emirates	ar-ae	0x3801	14337
Arabic - Bahrain	ar-bh	0x3C01	15361
Arabic - Algeria	ar-dz	0x1401	5121
Arabic - Egypt	ar-eg	0x0C01	3073
Arabic - Iraq	ar-iq	0x0801	2049
Arabic - Jordan	ar-jo	0x2C01	11265
Arabic - Kuwait	ar-kw	0x3401	13313
Arabic - Lebanon	ar-lb	0x3001	12289
Arabic - Libya	ar-ly	0x1001	4097
Arabic - Morocco	ar-ma	0x1801	6145
Arabic - Oman	ar-om	0x2001	8193
Arabic - Qatar	ar-qa	0x4001	16385
Arabic - Saudi Arabia	ar-sa	0x0401	1025
Arabic - Syria	ar-sy	0x2801	10241
Arabic - Tunisia	ar-tn	0x1C01	7169
Arabic - Yemen	ar-ye	0x2401	9217
Armenian	hy	0x042B	1067
Azeri – Latin	az-az	0x042C	1068
Azeri – Cyrillic	az-az	0x082C	2092
Basque	eu	0x042D	1069
Belarusian	be	0x0423	1059
Bulgarian	bg	0x0402	1026
Catalan	ca	0x0403	1027
Chinese - China	zh-cn	0x0804	2052
Chinese - Hong Kong S.A.R.	zh-hk	0x0C04	3076
Chinese – Macau S.A.R	zh-mo	0x1404	5124
Chinese - Singapore	zh-sg	0x1004	4100
Chinese - Taiwan	zh-tw	0x0404	1028
Croatian	hr	0x041A	1050

Czech	cs	0x0405	1029
Danish	da	0x0406	1030
Dutch – The Netherlands	nl-nl	0x0413	1043
Dutch - Belgium	nl-be	0x0813	2067
English - Australia	en-au	0x0C09	3081
English - Belize	en-bz	0x2809	10249
English - Canada	en-ca	0x1009	4105
English – Carribbean	en-cb	0x2409	9225
English - Ireland	en-ie	0x1809	6153
English - Jamaica	en-jm	0x2009	8201
English - New Zealand	en-nz	0x1409	5129
English – Phillippines	en-ph	0x3409	13321
English - South Africa	en-za	0x1C09	7177
English - Trinidad	en-tt	0x2C09	11273
English - United Kingdom	en-gb	0x0809	2057
English - United States	en-us	0x0409	1033
Estonian	et	0x0425	1061
Farsi	fa	0x0429	1065
Finnish	fi	0x040B	1035
Faroese	fo	0x0438	1080
French - France	fr-fr	0x040C	1036
French - Belgium	fr-be	0x080C	2060
French - Canada	fr-ca	0x0C0C	3084
French - Luxembourg	fr-lu	0x140C	5132
French - Switzerland	fr-ch	0x100C	4108
Gaelic – Ireland	gd-ie	0x083C	2108
Gaelic - Scotland	gd	0x043C	1084
German - Germany	de-de	0x0407	1031
German - Austria	de-at	0x0C07	3079
German - Liechtenstein	de-li	0x1407	5127
German - Luxembourg	de-lu	0x1007	4103
German - Switzerland	de-ch	0x0807	2055
Greek	el	0x0408	1032
Hebrew	he	0x040D	1037
Hindi	hi	0x0439	1081
Hungarian	hu	0x040E	1038
Icelandic	is	0x040F	1039
Indonesian	id	0x0421	1057
Italian - Italy	it-it	0x0410	1040
Italian - Switzerland	it-ch	0x0810	2064
Japanese	ja	0x0411	1041
Korean	ko	0x0412	1042
Latvian	lv	0x0426	1062
Lithuanian	lt	0x0427	1063
FYRO Macedonian	mk	0x042F	1071
Malay - Malaysia	ms-my	0x043E	1086
Malay – Brunei	ms-bn	0x083E	2110
Maltese	mt	0x043A	1082
Marathi	mr	0x044E	1102
Norwegian - Bokmål	no-no	0x0414	1044
Norwegian – Nynorsk	no-no	0x0814	2068
Polish	pl	0x0415	1045

Portuguese - Portugal	pt-pt	0x0816	2070
Portuguese - Brazil	pt-br	0x0416	1046
Raeto-Romance	rm	0x0417	1047
Romanian - Romania	ro	0x0418	1048
Romanian - Moldova	ro-mo	0x0818	2072
Russian	ru	0x0419	1049
Russian - Moldova	ru-mo	0x0819	2073
Sanskrit	sa	0x044F	1103
Serbian - Cyrillic	sr-sp	0x0C1A	3098
Serbian – Latin	sr-sp	0x081A	2074
Setsuana	tn	0x0432	1074
Slovenian	sl	0x0424	1060
Slovak	sk	0x041B	1051
Sorbian	sb	0x042E	1070
Spanish - Spain	es-es	0x0C0A	1034
Spanish - Argentina	es-ar	0x2C0A	11274
Spanish - Bolivia	es-bo	0x400A	16394
Spanish - Chile	es-cl	0x340A	13322
Spanish - Colombia	es-co	0x240A	9226
Spanish - Costa Rica	es-cr	0x140A	5130
Spanish - Dominican Republic	es-do	0x1C0A	7178
Spanish - Ecuador	es-ec	0x300A	12298
Spanish - Guatemala	es-gt	0x100A	4106
Spanish - Honduras	es-hn	0x480A	18442
Spanish - Mexico	es-mx	0x080A	2058
Spanish - Nicaragua	es-ni	0x4C0A	19466
Spanish - Panama	es-pa	0x180A	6154
Spanish - Peru	es-pe	0x280A	10250
Spanish - Puerto Rico	es-pr	0x500A	20490
Spanish - Paraguay	es-py	0x3C0A	15370
Spanish - El Salvador	es-sv	0x440A	17418
Spanish - Uruguay	es-uy	0x380A	14346
Spanish - Venezuela	es-ve	0x200A	8202
Sutu	sx	0x0430	1072
Swahili	sw	0x0441	1089
Swedish - Sweden	sv-se	0x041D	1053
Swedish - Finland	sv-fi	0x081D	2077
Tamil	ta	0x0449	1097
Tatar	tt	0X0444	1092
Thai	th	0x041E	1054
Turkish	tr	0x041F	1055
Tsonga	ts	0x0431	1073
Ukrainian	uk	0x0422	1058
Urdu	ur	0x0420	1056
Uzbek – Cyrillic	uz-uz	0x0843	2115
Uzbek – Latin	uz-uz	0x0443	1091
Vietnamese	vi	0x042A	1066
Xhosa	xh	0x0434	1076
Yiddish	yi	0x043D	1085
Zulu	zu	0x0435	1077

The **GetObject** function returns a reference to an automation object from a file.

Syntax

```
GetObject([pathname][,class])
```

Parameter	Description
pathname	Optional. The full path and name of the file that contains the automation object. If this parameter is omitted, the class parameter is required
class	Optional. The class of the automation object. This parameter uses this syntax: appname.objectype

The GetRef function allows you to connect a VBScript procedure to a DHTML event on your pages.

Syntax

```
Set object.event=GetRef(procname)
```

Parameter	Description
object	Required. The name of a DHTML object with which DHTML event is associated
event	Required. The name of a DHTML event to which the function is to be bound
procname	Required. The name of a Sub or Function procedure to be associated with the DHTML event

Example 1

```
Function test()
dim txt
txt="GetRef Test" & vbCrLf
txt=txt & "Hello World!"
MsgBox txt
End Function
Set Window.Onload=GetRef("test")
```

The **InputBox** function displays a dialog box, where the user can write some input and/or click on a button. 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 ("").

Note: A Help button is added to the dialog box when both the helpfile and the context parameter are specified.

Tip: Also look at the MsgBox function.

Syntax

```
InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,context])
```

Parameter	Description
prompt	Required. The message to show in the dialog box. Maximum length is 1024 characters. You can separate the lines using a carriage return character (Chr(13)), a linefeed character (Chr(10)), or carriage return–linefeed character combination (Chr(13) & Chr(10)) between each line
title	Optional. The title of the dialog box. Default is the application name
default	Optional. A default text in the text box
xpos	Optional. The horizontal distance of the left edge of the dialog box from the left edge of the screen. If omitted, the dialog box is horizontally centered

ypos	Optional. The vertical distance of the upper edge of the dialog box from the top of the screen. If omitted, the dialog box is vertically positioned one-third of the way down the screen
helpfile	Optional. The name of a Help file to use. Must be used with the context parameter
context	Optional. The Help context number to the Help topic. Must be used with the helpfile parameter

Example 1

```
dim fname
fname=InputBox("Enter your name:")
MsgBox("Your name is " & fname)
```

The **IsEmpty** function returns a Boolean value that indicates whether a specified variable has been initialized or not. It returns true if the variable is uninitialized; otherwise, it returns False.

Syntax

```
IsEmpty(expression)
```

Parameter	Description
expression	Required. An expression (most often a variable name)

Example 1

```
dim x
document.write(IsEmpty(x) & "<br />")
x=10
document.write(IsEmpty(x) & "<br />")
x=Empty
document.write(IsEmpty(x) & "<br />")
x=Null
document.write(IsEmpty(x))
Output:
True
False
True
False
```

The **IsNull** function returns a Boolean value that indicates whether a specified expression contains no valid data (Null). It returns True if expression is Null; otherwise, it returns False.

Syntax

```
IsNull(expression)
```

Parameter	Description
expression	Required. An expression

Example 1

```
dim x
document.write(IsNull(x) & "<br />")
x=10
document.write(IsNull(x) & "<br />")
x=Empty
```

```
document.write(IsNull(x) & "<br />")
x=Null
document.write(IsNull(x))
Output:
False
False
False
True
```

The **IsNumeric** function returns a Boolean value that indicates whether a specified expression can be evaluated as a number. It returns True if the expression is recognized as a number; otherwise, it returns False.

Note: If expression is a date the IsNumeric function will return False.

Syntax

```
IsNumeric(expression)
```

Parameter	Description
expression	Required. An expression

Example 1

```
dim x
x=10
document.write(IsNumeric(x) & "<br />")
x=Empty
document.write(IsNumeric(x) & "<br />")
x=Null
document.write(IsNumeric(x) & "<br />")
x="10"
document.write(IsNumeric(x) & "<br />")
x="911 Help"
document.write(IsNumeric(x))
Output:
True
True
False
True
False
```

The **IsObject** function returns a Boolean value that indicates whether the specified expression is an automation object. It returns True if expression is an automation object; otherwise, it returns False.

Syntax

```
IsObject(expression)
```

Parameter	Description
expression	Required. An expression

Example 1

```
dim x
set x=me
document.write(IsObject(x))
Output:
True
```

Example 2

```
dim x
x="me"
document.write(IsObject(x))
Output:
False
```

The **LoadPicture** function returns a picture object.

Graphics formats that is recognized by the LoadPicture function:

- bitmap files (.bmp)
- icon files (.ico)
- run-length encoded files (.rle)
- metafile files (.wmf)
- enhanced metafiles (.emf)
- GIF files (.gif)
- JPEG files (.jpg)

Note: This function is available only on 32-bit platforms.

Syntax

```
LoadPicture(picturename)
```

Parameter	Description
picturename	Required. The name of the picture file to be loaded

The **MsgBox** function displays a message box, waits for the user to click a button, and returns a value that indicates which button the user clicked.

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

Note: The user can press F1 to view the Help topic when both the helpfile and the context parameter are specified.

Tip: Also look at the InputBox function.

Syntax

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

Parameter	Description
prompt	Required. The message to show in the message box. Maximum length is 1024 characters. You can separate the lines using a carriage return character

	(Chr(13)), a linefeed character (Chr(10)), or carriage return–linefeed character combination (Chr(13) & Chr(10)) between each line
buttons	Optional. A value or a sum of values that specifies the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. Default value is 0 • 0 = vbOKOnly - OK button only • 1 = vbOKCancel - OK and Cancel buttons • 2 = vbAbortRetryIgnore - Abort, Retry, and Ignore buttons • 3 = vbYesNoCancel - Yes, No, and Cancel buttons • 4 = vbYesNo - Yes and No buttons • 5 = vbRetryCancel - Retry and Cancel buttons • 16 = vbCritical - Critical Message icon • 32 = vbQuestion - Warning Query icon • 48 = vbExclamation - Warning Message icon • 64 = vbInformation - 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 wont work until the user responds to the message box) We can divide the buttons values into four groups: The first group (0–5) describes the buttons to be displayed in the message box, the second group (16, 32, 48, 64) describes the icon style, the third group (0, 256, 512, 768) indicates which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the buttons parameter, use only one number from each group
title	Optional. The title of the message box. Default is the application name
helpfile	Optional. The name of a Help file to use. Must be used with the context parameter
context	Optional. The Help context number to the Help topic. Must be used with the helpfile parameter

Example 1

```
dim answer
answer=MsgBox("Hello everyone!",65,"Example")
document.write(answer)
```

The **RGB** function returns a number that represents an RGB color value.

Syntax

```
RGB(red,green,blue)
```

Parameter	Description
red	Required. A number from 0 to 255, inclusive, representing the red component of the color
green	Required. A number from 0 to 255, inclusive, representing the green component of the color
blue	Required. A number from 0 to 255, inclusive, representing the blue component of the color

Example 1

```
document.write(rgb(255,0,0))
Output:
255
```

Example 2

```
document.write(rgb(255,30,30))
Output:
1974015
```

The **Round** function rounds a number.

Syntax

```
Round(expression[,numdecimalplaces])
```

Parameter	Description
expression	Required. The numeric expression to be rounded
numdecimalplaces	Optional. Specifies how many places to the right of the decimal are included in the rounding. Default is 0

Example 1

```
dim x
x=24.13278
document.write(Round(x))
Output:
24
```

Example 2

```
dim x
x=24.13278
document.write(Round(x,2))
Output:
24.13
```

ScriptEngine Function

The ScriptEngine function returns the scripting language in use.

This function can return one of the following strings:

- VBScript - Indicates that Microsoft Visual Basic Scripting Edition is the current scripting engine
- JScript - Indicates that Microsoft JScript is the current scripting engine
- VBA - Indicates that Microsoft Visual Basic for Applications is the current scripting engine

ScriptEngineBuildVersion Function

The ScriptEngineBuildVersion function returns the build version number of the scripting engine in use.

ScriptEngineMajorVersion Function

The ScriptEngineMajorVersion function returns the major version number of the scripting engine in use.

ScriptEngineMinorVersion Function

The ScriptEngineMinorVersion function returns the minor version number of the scripting engine in use.

Syntax

```
ScriptEngine
ScriptEngineBuildVersion
ScriptEngineMajorVersion
ScriptEngineMinorVersion
```

Example 1

```
document.write(ScriptEngine & "<br />")
document.write(ScriptEngineBuildVersion & "<br />")
document.write(ScriptEngineMajorVersion & "<br />")
document.write(ScriptEngineMinorVersion)
Output:
VBScript
6330
5
5
```

The **SetLocale** function sets the locale ID and returns the previous locale ID.

A locale contains a set of user preferences, like language, country, region, and cultural conventions. The locale also determines such things as keyboard layout, sort order, date, time, number, and currency formats.

Syntax

```
SetLocale(lcid)
```

Parameter	Description
lcid	Required. A short string, hex value, or decimal value in the Locale ID chart, that identifies a geographic locale. If the lcid parameter is set to 0, the locale will be set by the system

Example 1

```
document.write(SetLocale(2057))
document.write(SetLocale(2058))
Output:
1033
2057
```

Locale ID Chart (As above chart…)

The **TypeName** function returns the subtype of a specified variable.

The TypeName function can return one of the following values:

- Byte - Indicates a byte value
- Integer - Indicates an integer value
- Long - Indicates a long integer value
- Single - Indicates a single-precision floating-point value
- Double - Indicates a double-precision floating-point value
- Currency - Indicates a currency value
- Decimal - Indicates a decimal value
- Date - Indicates a date or time value
- String - Indicates a character string value
- Boolean - Indicates a boolean value; True or False
- Empty - Indicates an unitialized variable
- Null - Indicates no valid data
- <object type> - Indicates the actual type name of an object
- Object - Indicates a generic object
- Unknown - Indicates an unknown object type
- Nothing - Indicates an object variable that doesn't yet refer to an object instance
- Error - Indicates an error

Syntax

```
TypeName(varname)
```

Parameter	Description
varname	Required. A variable name

Example 1

```
dim x
x="Hello World!"
document.write(TypeName(x) & "<br />")
x=4
document.write(TypeName(x) & "<br />")
x=4.675
document.write(TypeName(x) & "<br />")
x=Null
document.write(TypeName(x) & "<br />")
x=Empty
document.write(TypeName(x) & "<br />")
x=True
document.write(TypeName(x))
Output:
String
Integer
Double
Null
Empty
Boolean
```

The **VarType** function returns a value that indicates the subtype of a specified variable.

The VarType function can return one of the following values:

- 0 = vbEmpty - Indicates Empty (uninitialized)
- 1 = vbNull - Indicates Null (no valid data)
- 2 = vbInteger - Indicates an integer
- 3 = vbLong - Indicates a long integer

- 4 = vbSingle - Indicates a single-precision floating-point number
- 5 = vbDouble - Indicates a double-precision floating-point number
- 6 = vbCurrency - Indicates a currency
- 7 = vbDate - Indicates a date
- 8 = vbString - Indicates a string
- 9 = vbObject - Indicates an automation object
- 10 = vbError - Indicates an error
- 11 = vbBoolean - Indicates a boolean
- 12 = vbVariant - Indicates a variant (used only with arrays of Variants)
- 13 = vbDataObject - Indicates a data-access object
- 17 = vbByte - Indicates a byte
- 8192 = vbArray - Indicates an array

Note: If the variable is an array VarType() returns 8192 + VarType(array_element). Example: for an array of integer VarType() will return 8192 + 2 = 8194.

Syntax

```
VarType(varname)
```

Parameter	Description
varname	Required. A variable name

Example 1

```
dim x
x="Hello World!"
document.write(VarType(x) & "<br />")
x=4
document.write(VarType(x) & "<br />")
x=4.675
document.write(VarType(x) & "<br />")
x=Null
document.write(VarType(x) & "<br />")
x=Empty
document.write(VarType(x) & "<br />")
x=True
document.write(VarType(x))
Output:
8
2
5
1
0
11
```