Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

VB Script
Basic

People matter, results count.

Contents

VB Script Fundamentals - Variables

VB Script Variables
Definition 1): Variable is a named memory location for storing program information
Definition 2): A variable is a convenient placeholder that refers to a computer memory location where we can store program information that may change during the time our script is running.

Purpose of Variable:

a) Comparing values Example:
Dim x,y,a
x=100
y=100
a=x=y :: Msgbox a 'It returns True

b) Holding Program Result
Example:
Cost=Tickets*Price

c) Passing parameters – can be passed as parameters
Function abc (intLength , intBreadth)

d) To store data that returned by functions
Example:
myDate=Now ' It returns current date & time

e) To hold data
Example:
myName="testing"

Declaring Variables

Variables are explicitly declared using - Dim statement, the Public statement, and the Private statement.
For example:
Dim length
Dim volume
Multiple variables are declared using **comma** as delimiter
Example:
Dim a, b, testing

Variables are implicitly declared just by using them wherever needed in the script. That is not generally a good practice because we could misspell the variable name in one or more places, causing unexpected results when our script is run. For that reason, the Option Explicit statement is available to require explicit declaration of all variables.

Option Explicit statement:
It should be the first statement in script.

Option Explicit Statement:
Forces explicit declaration of all variables in a script.

```
Option Explicit ' Force explicit variable declaration.
Dim MyVar ' Declare variable.
MyInt = 10 ' Undeclared variable generates error.
MyVar = 10 ' Declared variable does not generate
 error.
```

VB Script Fundamentals - Variables

Naming Standards/Conventions/Rules for Variables :

Variable names follow the standard rules for naming anything in VBScript. A variable name:

a) Must begin with an alphabetic character.

```
Dim abc  , ab88  'Right
Dim ~~1ab    'Wrong
```

b) Cannot contain an embedded period. 'Underscore (_) character is allowed'

```
Dim abc     'Right
Dim ab.c , ab-c , ab c    ' wrong declarations
Dim ab_c   'Right
```

c) Must not exceed 255 characters.

d) Must be unique in the scope in which it is declared.

Scope / Lifetime of Variables

A variable's **scope** is determined by where we declare it.
When we declare a variable within a procedure, only code within that procedure can access or change the value of that variable.
If we declare a variable outside a procedure, we make it recognizable to all the procedures in our script. This is a script-level variable, and it has script-level scope.

The **lifetime** of a variable depends on how long it exists. The lifetime of a script-level variable extends from the time it is declared until the time the script is finished running. At procedure level, a variable exists only as long as you are in the procedure.

Example:

```
Dim x,y,z
x=10
y=20
z=x+y
Msgbox z    'Returns 30
Function fun_Addition
  Dim a,b,c
  a=30
  b=40
  c=a+b+y
  Msgbox c   ' Returns 90
End Function
Call fun_Addition
```

VB Script Fundamentals - Variables

Array Variables:

A variable containing a series of values, is called an array variable.
Array variables and scalar variables are declared in the same way, except that the declaration of an array variable uses parentheses () following the variable name.

Example:
Dim A(3)

Although the number shown in the parentheses is 3, all arrays in VBScript are zero-based, so this array actually contains 4 elements.
We assign data to each of the elements of the array using an index into the array.
Beginning at zero and ending at 4, data can be assigned to the elements of an array as follows:

A(0) = 256
A(1) = 324
A(2) = 100
A(3) = 55

Similarly, the data can be retrieved from any element using an index into the particular array element you want.

For example:

retValue= A(4)

Arrays aren't limited to a single dimension. We can have maximum **60 dimensions**

In the following example, the **myTwoDArray** variable is a two-dimensional array consisting of 6 rows and 6 columns:

Dim **myTwoDArray**(**5**, **5**)**.**

In a two-dimensional array, the first number is always the number of rows; the second number is the number of columns.

VB Script Fundamentals – Data Types

VB Script Data Types –

VBScript has only one data type called a Variant. A Variant is a special kind of data type that can contain different kinds of information, depending on how it is used. Because Variant is the only data type in VBScript, it is also the data type returned by all functions in VBScript.

Variant Subtypes –

Beyond the simple numeric or string classifications, a Variant can make further distinctions about the specific nature of numeric information. For example, we can have numeric information that represents a date or a time. When used with other date or time data, the result is always expressed as a date or a time. We can also have a rich variety of numeric information ranging in size from Boolean values to huge floating-point numbers. These different categories of information that can be contained in a Variant are called subtypes. Most of the time, we can just put the kind of data we want in a Variant, and the Variant behaves in a way that is most appropriate for the data it contains.

The following table shows subtypes of data that a Variant can contain.

Sr.No.	Subtype	Description
1	Empty	Variant is uninitialized. Value is 0 for numeric variables or a zero-length string ("") for string variables.
2	Null	Variant intentionally contains no valid data.
3	Boolean	Contains either True or False.
4	Byte	Contains integer in the range 0 to 255.
5	Integer	Contains integer in the range -32,768 to 32,767.
6	Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807.
7	Long	Contains integer in the range -2,147,483,648 to 2,147,483,647.
8	Single	Contains a single-precision, floating-point number in the range -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.
9	Double	Contains a double-precision, floating-point number in the range -1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values.
10	Date (Time)	Contains a number that represents a date between January 1, 100 to December 31, 9999.
11	String	Contains a variable-length string that can be up to approximately 2 billion characters in length.
12	Object	Contains an object.
13	Error	Contains an error number.

VB Script Fundamentals - Operators

VB Script Operators :- Operators are used for performing mathematical, comparison and logical operations.
VB Script has a full range of operators, including arithmetic operators, comparison operators, concatenation operators, and logical operators.

Operator Precedence :- When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence.
We can use parentheses to override the order of precedence and force some parts of an expression to be evaluated before others. Operations within parentheses are always performed before those outside. Within parentheses, however, standard operator precedence is maintained.
When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last.
Comparison operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear. Arithmetic and logical operators are evaluated in the following order of precedence.

1) Arithmetic Operators:

Operator Description
1) Exponentiation Operator (^) Raises a number to the power of an exponent
2) Multiplication Operator (*) Multiplies two numbers.
3) Division Operator (/) Divides two numbers and returns a floating-point result.
4) Integer Division Operator (\) Divides two numbers and returns an integer result.
5) Mod Operator Divides two numbers and returns only the remainder.
6) Addition Operator (+) Sums two numbers.
7) Subtraction Operator (-) Finds the difference between two numbers or indicates the negative value of a numeric expression.
8) Concatenation Operator (&) Forces string concatenation of two expressions.

2) Comparison Operators : Used to compare expressions.

Operator Description
1) = (Equal to) Used to compare expressions.
2) <> (Not equal to) Used to compare expressions.
3) < Less than 4) > Grater than
5) <= Less than or equal to 6) >= Greater than or equal to
7) Is Object equivalence

3) Concatenation Operators :

Operator Description - Addition Operator (+) :- Sums two numbers If Then
1) Both expressions are numeric Add.
2) Both expressions are strings Concatenate.
3) One expression is numeric and the other is a string Add.

Concatenation Operator (&) Forces string concatenation of two expressions

4) Logical Operators:

Operator Description

1) Not Performs logical negation on an expression result= Not expression
2) And Performs a logical conjunction on two expressions. result= expression1 And expression2
3) Or Performs a logical disjunction on two expressions. result= expression1 Or expression2
4) Xor Performs a logical exclusion on two expressions. result= expression1 Xor expression2
5) Eqv Performs a logical equivalence on two expressions. result= expression1 Eqv expression2
6) Imp Performs a logical implication on two expressions. result= expression1 Imp expression2

VB Script Fundamentals - Operators

Arithmetic Operators	Addition (+) operator	Concatenation Operator	Comparison Operators	Logical Operators

Dim a,b,c
a=10
b=3
c=a^b
Msgbox c '1000
 	
c=a*b
Msgbox c '30
 	
c=a/b
Msgbox c '3.33333333
 	
c=a\b
Msgbox c '3
 	
c=a mod b
Msgbox c '1
 	
c=a-b
Msgbox c '7
 	
Dim a,b,c
a=10
b=2
c=3
d=c*a^b
'c=a+b
Msgbox d '1000
	Dim a,b,c
	a=10
	b=2
	c=a+b
	Msgbox c '12
	(if both are numeric, then it adds)
	
	a="10"
	b=2
	c=a+b
	Msgbox c '12
	(one is string another numeric, then it adds)
	
	a="10"
	b="2"
	c=a+b
	Msgbox c '102
	(if both are strings, then it concatenates)
	
	a="hydera"
	b="bad"
	c=a+b
	Msgbox c 'hyderabad
	
	a="gagan"
	b=2
	c=a+b
	Msgbox c 'error
		Dim a,b,c
		a=10
		b=2
		c=a&b
		Msgbox c '102
		
		a="10"
		b=2
		c=a&b
		Msgbox c '102
		
		a="10"
		b="2"
		c=a&b
		Msgbox c '102
		
		a="hydera"
		b="bad"
		c=a&b
		Msgbox c '102
			Dim x,y,z
			x=10
			y=20
			z=x=y
			Msgbox z 'False
			
			x=10
			y=20
			z=x>y
			Msgbox z 'False
			
			x=10
			y=20
			z=x>=y
			Msgbox z 'False
			
			x=10
			y=20
			z=x<>y
			Msgbox z 'True
			
			x=10
			y=20
			z=x<y
			Msgbox z 'True
			
			x=10
			y=20
			z=x<=y
			Msgbox z 'True
				AND , OR , NOT
				
				Dim shoeSize
				shoeSize = 10
				If shoeSize = 10 **OR** shoeSize = 12 Then
				'Some code
				EndIf
				
				Dim shoeSize
				shoeSize = 10
				If shoeSize >= 10 **AND** shoeSize <= 12 Then
				'Some code
				Msgbox "Shoesize within limit"
				EndIf

VB Script Fundamentals - – Constants & I/O Operations

VB Script Constants

A constant is a meaningful name that takes the place of a number or string and never changes.

Creating Constants

We create user-defined constants in VBScript using the Const statement. Using the Const statement, we can create string or numeric constants with meaningful names and assign them literal values.
Const statement - Declares constants for use in place of literal values.

Example:

```
Const MyString = "This is my string."
Const MyAge = 49
Const CutoffDate = #6-1-97#
```

Note that String literal is enclosed in quotation marks (" ").
Represent Date literals and time literals by enclosing them in number signs (#).

We declare multiple constants by separating each constant name and value with a comma. For example:
Const price= 100, state = “Maharashtra” , x= 27

Input and Output Operations

InputBox Function - Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns the contents of the text box.

Example:

```
Dim Input
Input = InputBox("Enter your name")
Msgbox ("You entered: " & Input)
```

Msgbox Function - Displays a message in a dialog box, waits for the user to click a button, and returns a value indicating which button the user clicked.

Example:

```
Dim MyVar
MyVar = Msgbox ("Hello World!", 65, "Msgbox Example")
```

MyVar holds the value either 1 or 2, depending on which button is clicked

VB Script Fundamentals – Conditional Statements

Flow Control (Conditional Statements) - We can control the flow of our script with conditional statements and looping statements.
Using conditional statements, we can write VBScript code that makes decisions and repeats actions. The following conditional statements are available in VBScript:

1) If…Then…Else Statement
2) Select Case Statement

If...Then...Else:

The If...Then...Else statement is used to evaluate whether a condition is True or False and, depending on the result, to specify one or more statements to run.
Usually the condition is an expression that uses a comparison operator to compare one value or variable with another.
If...Then...Else statements can be nested to as many levels as you need.

Running Statements if a Condition is True

To run more than one line of code, we must use the multiple-line (or block) syntax. This syntax includes the End If statement.

```
Dim x
x= 20
If x>10 Then
    Msgbox "x value is: "&x
   Msgbox "Bye Bye"
End If
```

Running Certain Statements if a Condition is True and Running Others if a Condition is False

We can use an If...Then...Else statement to define two blocks of executable statements: one block to run if the condition is True, the other block to run if the condition is False.
Example:

```
Dim x
x= Inputbox (" Enter a value")
      If x>100 Then
                  Msgbox “Condition  Satisfied “
                  Msgbox "X value is: "&X
       Else
                  Msgbox " Condition  Not Satisfied“
                  Msgbox "X value is: "&X
        End If
```

Deciding Between Several Alternatives

A variation on the If...Then...Else statement allows us to choose from several alternatives. Adding ElseIf clauses expands the functionality of the If...Then...Else statement so we can control program flow based on different possibilities.

Example:

```
Dim x
x= Inputbox (" Enter a value")

If x>0 and x<=100 Then
  Msgbox "Hello Test Msgbox"
  Msgbox "X is a Small Number"
  Msgbox "X value is "&x

Else IF x>100 and x<=500 Then
  Msgbox "Hello Test Msgbox"
  Msgbox "X is a Medium Number"

Else
  Msgbox "Hello Sir"
  Msgbox "X is a Grand Number“

End If
End If
```

VB Script Fundamentals – Conditional Statements

Select Case :

Case The Select Case structure provides an alternative to If...Then...ElseIf for selectively executing one block of statements from among multiple blocks of statements. A Select Case statement provides capability similar to the If...Then...Else statement, but it makes code more efficient and readable

Example:

```
Option explicit
Dim x,y, Operation, Result
x= Inputbox (" Enter x value")
y= Inputbox ("Enter y value")
Operation= Inputbox ("Enter an Operation")

Select Case Operation
                Case "add"
                        Result= cdbl (x)+cdbl (y)
                        Msgbox "Addition of x,y values is "&Result
                Case "sub"
                        Result= x-y
                        Msgbox "Substraction of x,y values is "&Result
                Case "mul"
                        Result= x*y
                        Msgbox "Multiplication of x,y values is "&Result
                Case Else
                        Msgbox "Wrong Operation“
End Select
```

VB Script Fundamentals – Iterating in Loops

Flow Control (Loop Statements)

- Looping allows us to run a group of statements repeatedly.
- Some loops repeat statements until a condition is False
- Others repeat statements until a condition is True.
- There are also loops that repeat statements a specific number of times.
- The following looping statements are available in VBScript:
- Do...Loop: Loops while or until a condition is True.
- While...Wend: Loops while a condition is True.
- For...Next: Uses a counter to run statements a specified number of times.
- For Each...Next: Repeats a group of statements for each item in a collection or each element of an array.

Do Loops

Do While condition

Example:

```
Dim x
Do While x<5 x=x+1
Msgbox "Hello QTP"
Loop
```

Do Until condition

Example 1:

```
Dim x
Do Until x=5 x=x+1
Msgbox "Test"
Msgbox "Hello QTP"
Loop
```

Example 2:

```
Dim x
x=1
Do
Msgbox “Hello Test”
x=x+1
Loop Until x=5
```

Do

Example:

```
Dim x
x=1
Do
Msgbox "Hello QTP"
x=x+1
Loop While x<5
```

VB Script Fundamentals - Iterating in Loops

Flow Control (Loop Statements)

- Looping allows us to run a group of statements repeatedly.
- Some loops repeat statements until a condition is False
- Others repeat statements until a condition is True.
- There are also loops that repeat statements a specific number of times.
- The following looping statements are available in VBScript:
- Do...Loop: Loops while or until a condition is True.
- While...Wend: Loops while a condition is True.
- For...Next: Uses a counter to run statements a specified number of times.
- For Each...Next: Repeats a group of statements for each item in a collection or each element of an array.

1) **Using Do Loops** - We can use Do...Loop statements to run a block of statements an indefinite number of times. The statements are repeated either while a condition is True or until a condition becomes True.

a) Repeating Statements While a Condition is True :- Repeats a block of statements while a condition is True or until a condition becomes True

i) **Do While condition**

```
Statements
----------
----------
Loop
```

Or, we can use this below syntax:

Example:

```
i) Dim x
Do While x<5 x=x+1
Msgbox "Hello QTP"
Loop
```

ii) **Do**

```
Statements
----------
----------
Loop While condition
```

Example:

```
Dim x
x=1
Do
Msgbox "Hello QTP"
x=x+1
Loop While x<5
```

b) Repeating a Statement Until a Condition Becomes True

VB Script Fundamentals – Iterating in Loops

iii) **Do Until condition**
Statements

Loop
Or, we can use this below syntax:
Example:
Dim x
Do Until x=5 x=x+1
Msgbox "Test"
Msgbox "Hello QTP"
Loop

Or, we can use this below syntax:
iv) **Do**
Statements

Loop Until condition
Or, we can use this below syntax:

Example:
Dim x
x=1
Do
Msgbox “Hello Test”
Msgbox "Hello QTP"
x=x+1
Loop Until x=5

2) **While...Wend** Statement - Executes a series of statements as long as a given condition is True.
Syntax:
While condition
Statements

Wend
Example:
Dim x
x=0
While x<5 x=x+1
  Msgbox "Hello QTP"
Wend

3**) For...Next** Statement - Repeats a group of statements a specified number of times.
Syntax:
For counter = start to end [Step step]
statements
Next
Example:
Dim x
For x= 1 to 5 step 1
Next

4) **For Each...Next** Statement - Repeats a group of statements for each element in an array or collection.
Syntax:
For Each item In array
Statements
Next
Example: (1)
Dim a,b,x (3)
a=20
b=30
x(0)= "Addition is "& a+b
x(1)="Substraction is " & a-b
x(2)= "Multiplication is " & a*b
x(3)= "Division is " & a/b

For Each element In x
Msgbox element
Next

VB Script Fundamentals - User Defined Functions

In VBScript, there are two kinds of procedures available; the Sub procedure and the Function procedure.

Sub Procedures - A Sub procedure is a series of VBScript statements (enclosed by Sub and End Sub statements) that perform actions but don't return a value.
A Sub procedure can take arguments (constants, variables, or expressions that are passed by a calling procedure).
If a Sub procedure has no arguments, its Sub statement must include an empty set of parentheses ().

PROCEDURE W/O ARGUMENT
Syntax:
Sub Procedure name ()
Statements

End Sub

```
Procedure Celsius()

fDegrees = 30
Celsius = (fDegrees - 32) * 5 / 9
(a+b+c)

End Procedure
```

PROCEDURE WITH ARGUMENT
Sub Procedure name (argument1, argument2)
Statements

End Sub

```
Procedure cal(a,b,c)

 cal = (a+b+c)

End Procedure
```

Function Procedures
A Function procedure is a series of VBScript statements enclosed by the Function and End Function statements.
A Function procedure is similar to a Sub procedure, but can also return a value.
A Function procedure can take arguments (constants, variables, or expressions that are passed to it by a calling procedure).

PROCEDURE W/O ARGUMENT
Sub Procedure name ()
Statements

End Sub

```
Function Celsius()

fDegrees = 30
Celsius = (fDegrees - 32) * 5 / 9
(a+b+c)

End Function
```

PROCEDURE WITH ARGUMENT
Sub Procedure name (argument1, argument2)
Statements

End Sub

```
Function cal(a,b,c)

 cal = (a+b+c)

End Function
```

VB Script Fundamentals - User Defined Functions

❖ **Getting Data into and out of Procedures –**

- Each piece of data is passed into our procedures using an argument.
- Arguments serve as placeholders for the data we want to pass into our procedure. We can name our arguments any valid variable name.
- When we create a procedure using either the Sub statement or the Function statement, parentheses must be included after the name of the procedure.
- Any arguments are placed inside these parentheses, separated by commas.

❖ **Using Sub and Function Procedures in Code**

A Function in our code must always be used on the right side of a variable assignment or in an expression.

For example:

```
intValue = Round( intTempVal)

                OR

Msgbox "The Rounded Value is " & Round( intTempVal)
```

To call a Sub procedure from another procedure, type the name of the procedure along with values for any required arguments, each separated by a comma.

The Call statement is not required, but if you do use it, you must enclose any arguments in parentheses.

The following example shows two calls to the MyProc procedure. One uses the Call statement in the code; the other doesn't. Both do exactly the same thing.

Examples -

```
Call MyProc(firstarg, secondarg)

MyProc firstarg, secondarg  ' Parenthesis to be omitted  when Call statement is not used
```

VB Script Fundamentals - User Defined Functions Examples

```
'*********************************************
'       Login Operation
'*********************************************
Function Login(Agent,Pwd)
   SystemUtil.Run "C:\Program Files\HP\QuickTest
Professional\samples\flight\app\flight4a.exe"
    Dialog("Login").Activate
    Dialog("Login").WinEdit("Agent Name:").Set Agent
    Dialog("Login").WinEdit("Password:").Set Pwd
    Dialog("Login").WinButton("OK").Click

    If Window("Flight Reservation").Exist(10) Then
        Login="Login Operation Sucessful"
        'Msgbox Login
        else
Login="Login Operation Unsuccessful"
        Dialog("Login").Dialog("Flight Reservations").WinButton("OK").Click
        Dialog("Login").WinButton("Cancel").Click
        'Msgbox Login
End If
End Function

'*****************************************
```

```
'       Closing Application
'***********************************
Function Close_App()
if Window("Flight Reservation").Exist(3) Then
Window("Flight Reservation").Close
End If
End Function

'***********************************
```

```
'       Update Order
'*****************************************
Function Update_Order(Tickets)
Window("Flight Reservation").Activate
Window("Flight Reservation").WinEdit("Tickets:").Set Tickets
Window("Flight Reservation").WinButton("Update Order").Click
wait(10)
update=Window("Flight Reservation").ActiveX("Threed Panel
Control").GetROProperty ("text")
If update="Update Done..." Then
Update_Order= "Order Updated Successfully"
'Msgbox Update_Order
Else
Window("Flight Reservation").Dialog("Flight
Reservations").WinButton("OK").Click
Update_Order= "Order Not Updated"
'Msgbox Update_Order
End If
End Function
```

```
‘ Function to send a mail
'*********************************************
Function SendMail(SendTo, Subject, Body, Attachment)
Set otl=CreateObject("Outlook.Application")
Set m=otl.CreateItem(0)
m.to=SendTo
m.Subject=Subject
m.Body=Body
If (Attachment <> "") Then
Mail.Attachments.Add(Attachment)
End If
m.Send
otl.Quit
Set m = Nothing
Set otl = Nothing
End Function
Call SendMail("testing@testing.com","hi","This is test mail for
testing","")
```

VB Script Fundamentals – Built in Functions

Built-In Functions of VB Script – Below are the types and count of functions associated with each type - Conversions (25) , Dates/Times (19) , Formatting Strings (4) , Input/Output (3) , Math (9) , Miscellaneous (3) , Rounding (5) , Strings (30), Variants (8)

Math Functions:

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

```
Dim max,min
max=100
min=1
Randomize
Msgbox Int((max-min+1)*Rnd+min)
```

Exp function - returns *e* raised to a power

Array Functions:

IsArray function - returns a Boolean value that indicates whether a specified variable is an array

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

Example –

```
days=Array("Sun","Mon","Tue","Wed","Thu","Fri","Sat")
msgbox LBound(days)  ‘ Returns 0
msgboxUBound(days)  ‘ Returns 6
```

String Functions:

Left Function - Returns a specified number of characters of a given string from left side

Example:

```
Dim val,x
val="Hyderabad"
x=Left(val,3)
Msgbox x ' Output: Hyd
```

Mid function -Returns a specified number of characters of a given string

Example:

```
Dim val,x
val="Hyderabad"
x=Mid(Val,5,3)
Msgbox x ' Output: rab
```

StrReverse - retuns reverse value of a string

Example:

```
Dim val,x
val="Hyderabad"
x=StrReverse(val)
Msgbox x 'Output dabaredyH
```

StrComp Function - It compares two string (Binary and textual)
if a) Both are equal, returns 0(zero)
b) String 1 greater than string 2, returns 1(one)
c) String 2 greater than string 1, returns -1
Example:

```
Dim str1,str2,x
str1="India"
str2="India"
x=StrComp(str1,str2,1)
Msgbox x 'Output 0
```

VB Script Fundamentals – Built in Functions

String Functions:

Len Function - Returns the number of characters in a string or the number of bytes required to store a variable.

Example-
```
Dim Mystring
mystring=Len("Test Msgbox")
Msgbox mystring
```

Replace - It replace a sub string with given value (another sub string)

Example-
```
mystring=Replace("kb script", "k","v")
Msgbox mystring
```

Conversion Functions:

CDate Function - Returns an expression that has been converted to a Variant of subtype Date.

Example-
```
MyDate = "October 19, 1962"   ' Define date.
MyShortDate = CDate(MyDate)   ' Convert to Date data type.
MyTime = "4:35:47 PM"         ' Define time.
MyShortTime = CDate(MyTime)   ' Convert to Date data type.
```

CInt Function - Returns an expression that has been converted to a Variant of subtype Integer.

Example-
```
Dim MyDouble, MyInt
MyDouble = 2345.5678     ' MyDouble is a Double
MyInt = CInt(MyDouble)   ' MyInt contains 2346
```

DateFunctions:

Date Function - Returns the Current System Date.

Example
```
Dim mydate
mydate=Date
Msgbox mydate
```

Day Function -

Example -
```
Dim myday
myday=Day("17,December,2009")
Msgbox myday
```

DateDiff Function - Returns the number of intervals between two dates.

Example
```
Dim Date1, Date2, x
Date1=#10-10-09#
Date2=#10-10-11#
x=DateDiff("yyyy", Date1, Date2)
Msgbox x 'Differnce in Years
```

DatePart Functions- Returns the specified part of a given date

Example –
```
d=CDate("2010-02-16")
NewDate  = (DatePart("m",d))
```

DateAdd function - returns a date to which a specified time interval has been added

Example -
```
Msgbox DateAdd("yyyy",1,"31-Jan-10")  'Returns 1/31/2011
```

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

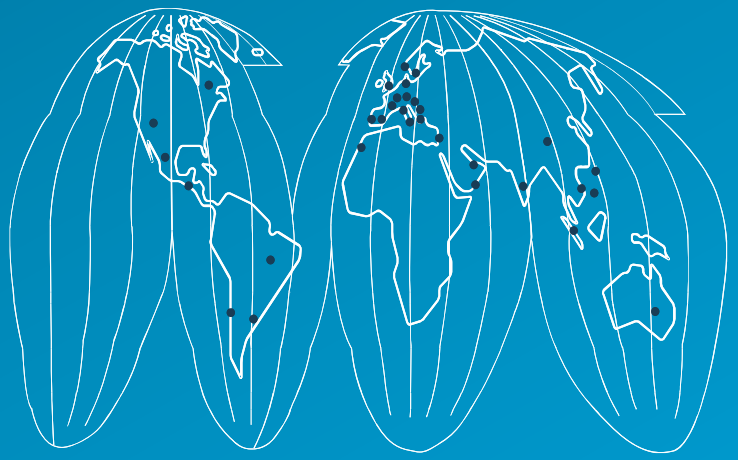
People matter, results count.

About Capgemini

With more than 120,000 people in 40 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2011 global revenues of EUR 9.7 billion.

Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Rightshore® is a trademark belonging to Capgemini



www.capgemini.com