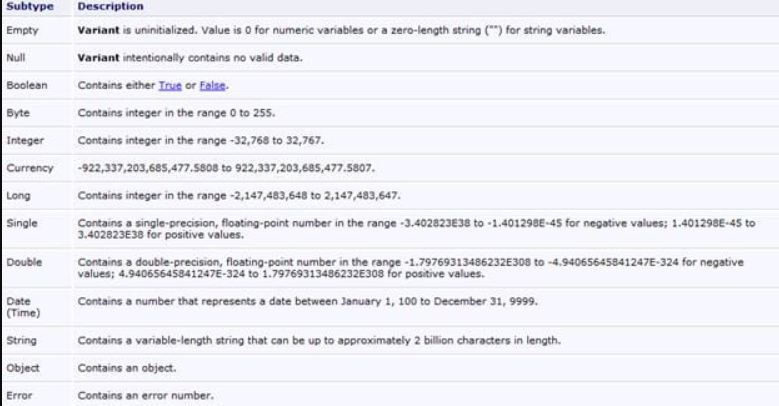
**VB Scripting**

**VBScript** (Visual Basic Script) is developed by Microsoft with the intention of developing dynamic web pages. It is client-side scripting language like JavaScript. VBScript is a light version of Microsoft Visual Basic. The syntax of VBScript is very similar to that of Visual Basic.

VBScript is just a **scripting** language. So, it cannot run its code on its own. It needs a bigger programming language to host it.

**Data types**

1) There is only one data type: **Variant**. It can store different kinds of information based on the context in which it is used.  
2) If used in a numeric context it is a number or a string if used in a string expression.  
3) If a number has to behave as a string we could enclose it within “ “.  
4) There are various subtypes to a variant. You can explicitly specify these subtypes to achieve a clear definition for your data. The below is a screenshot from the VB User guide that shows all the subtypes of data that can be used:



* Declaring a variable before its use is optional, although it’s a good practice to do so.

Unlike other languages, VBScript has only 1 data type called Variant.

As this is the only data type that is used in VBScript, it’s the only data type that is returned by all the functions in the VBScript.

A variant data type can contain different kinds of information, depending on how it is used. For Example, If we use this data type in the String context then this will behave like a String and if we use this in the Numeric context then this will behave like a Number. This is the specialty of a Variant data type.

**A Variant data type can contain several subtypes**. Now, let’s take a look at what all values/data will be returned if a particular subtype is used.

Subtypes include:

#1) Empty: This subtype indicates that the value will be 0 in the case of Numeric Variables and “for String Variables.

#2) Null: This subtype indicates that there is no valid data.

#3) Boolean: This subtype indicates that the resultant value will be either true or false.

#4) Byte: This subtype exhibits that the resultant value will lie in the range between 0 to 255 i.e. the result will be from any value ranging from 0 to 255.

#5) Integer: This subtype shows that the resultant value will lie in the range between -32768 to 32767 i.e. the result will be from any value ranging from -32768 to 32767

#6) Currency: This subtype indicates that the resultant value will lie in the range between -922,337,203,685,477.5808 to 922,337,203,685,477.5807 i.e. the result will be from any value ranging from -327-922,337,203,685,477.5808 to 922,337,203,685,477.5807.

#7) Long: This subtype shows that the resultant value will lie in the range from -2,147,483,648 to 2,147,483,647 i.e. result will be from any value in between -2,147,483,648 to 2,147,483,647.

#8) Single: This subtype exhibits that the resultant value will be from any value between -3.402823E38 to -1.401298E-45 in case of negative values.

And for positive values, the result will be from any value between 1.401298E-45 to 3.402823E38.

#9) Double: This subtype indicates that the resultant value will be from any value between -1.79769313486232E308 to 4.94065645841247E-324 in case of negative values.

And for positive values, the result will be from any value between 4.94065645841247E-324 to 1.79769313486232E308.

#10) Date (Time): This subtype will return a number that will represent a date value between January 1, 100 to December 31, 9999

#11) String: This subtype will return a variable-length string value which can approximately be up to 2 billion characters in length.

#12) Object: This subtype will return an object.

#13) Error: This subtype will return an error number.

**Declaring Variable**

Dim x – This declares  x  
Dim x, y, z – This declares multiple variables  
X=10 – This is how a value is assigned. As a general rule, the variable is the left-hand side component and the right is its value.  
X=”Swati” – this is the way a string value is assigned.

To make declarations mandatory this is how the code has to be written:  
*Option Explicit*  
*Dim x, stri*

* A variable containing a single value is a scalar variable and the one that has more than one is an array.
* A one dimensional Array can be declared as **Dim A(10)**. All the arrays in VB Script are zero-based that means the array index starts from 0 through the number declared. That means, our array A has 11 elements. Starting from 0 to 10.
* To declare **a 2-dimensional array** simply separate the row count and column count by a comma. Eg: **Dim A(5, 3)**. This means it has **6 rows and 4 columns**. The first number is always row and the second a comma.
* There is also a **dynamic array** whose size can change during runtime.  These arrays can be declared using dim or redim statements

If an array is declared as Dim A(10) and during runtime, if we need more space we can do the same by using the statement: redim A(10). There is a “Preserve” statement that can be used in conjunction with the redim statement.

Dim A(10,10)

……

….

Redim preserve A(10,20)

This piece of code shows how we do it. Initially, A is a 11 by 11 array. Then we are resizing it to be an 11 by 21 array and the preserve statement will make sure that the data that is previously contained in the array is not lost.

**Constants**

1. As the name implies a constant is nothing but an unchanging value in a program that is assigned a name.
2. They can be declared by prefixing “Const” to a name.
3. Eg: Const a=”10” or Const Astr=”Swati”.
4. This value cannot be changed accidentally while the script is running

**Operators**

Some of the important operators that are most commonly used are:

1. String concatenation: & (Eg: Dim x=”good”&”day”, so x contains “goodday”
2. Addition (+)
3. Subtraction (-)
4. Multiplication (\*)
5. Division(/)
6. Logical negation (Not)
7. Logical conjunction (And)
8. Logical disjunction ( Or)
9. Equality(=)
10. Inequality (<>)
11. Less than (<)
12. Greater than(>)
13. Less than or equal to(<=)
14. Greater than or equal to (>=)
15. Object equivalence(Is)

**Comments in vbScripts**

Any Statement that starts with a Single Quote (‘) is treated as a comment:

Any Statements that start with the keyword REM are treated as Comments

**Formatting Tips:**

**#1)** No Semicolon is required to end the particular statement in VBScript.

**#2)** If 2 or more lines are written in the same line in VBScript then**Colons (:) act as a line separator**.

**Let’s understand this with the help of an Example:**

<script type=”text/vbscript”></script>

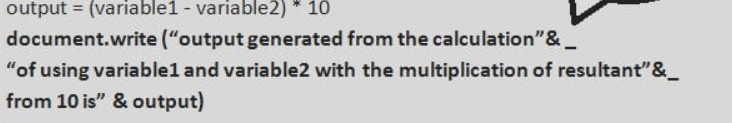
</head>

<body>

**variable1 = 11:variable2 = 21:variable3=34**

</script>

* if a statement is lengthy and required to break into multiple statements then you can use the **underscore** “\_”.



**Types of Variable Declaration**

Implicit and Explicit

**Implicit**

When variables are used directly without declaration, it is termed as Implicit Declaration. Ex: age=10;

**Explicit Declaration**

Declaring variables before using them is called an Explicit Declaration of variables. This is what we have done above.

**Eg**: Dim age

**Option Explicit:**

This provides a mechanism where the user has to declare all the variables using Dim, Public or Private Statements before using them in the Script. In option explicit, if a user tries to use the variables which are not declared in case of Option Explicit then an error occurs.

**Scope of Variable**

A variable can be declared by using any of the 3 keywords – ***Dim, Public or Private***.

***Dim –****If a variable is declared using Dim keyword inside the function then its scope will be limited to the function level only i.e. this variable cannot be accessed once the function ends.*

***Private –****If a variable is declared using Private keyword inside the function then its scope will not be limited to function level alone, but it can be accessed everywhere in that particular script. Here, the scope of a variable is wider than in the case of Dim as it’s not limited to the function level but a particular script.*

***Public –****If a variable is declared using Public keyword inside the function then its scope will not be limited till function level, but it can be accessed everywhere in that particular script and in other scripts too. Here, the scope of a variable is much wider than that in case of Dim and Private.*

**Arrays**

**Dim Arraycount (4)** – This means that this is an array having 5 values starting from index ‘0’.

To assign values to this array, you can give any numeric or string type values for each index. Use double quotes for assigning string values.

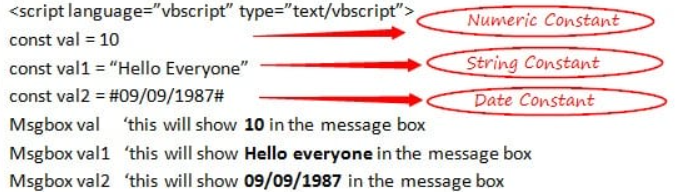
array count (0) = 1  
array count (1) = 2  
array count (2) = 3  
array count (3) = 4  
array count (4) = 5

**Multidimensional Array**: When there is more than one array then it is known as a Multidimensional array. Let’s assume that there is an array of 2 dimensions i.e. there are 6 rows and 6 columns, hence it can be declared as follows:

Dim arrayage (6, 6)

**Constants and Operators**

**‘const’** keyword to declare Constants in your script. Constants can be declared as public or private depending on its usage. If it’s declared **Public** then it can be used everywhere in a particular script and also in all the other scripts and procedures whereas if it’s declared as **Private** then it can be used only in one particular script in which you are working including its procedures and classes.



**VBScript Data Type Constants**

Given below are some of the Data Type Constants in the VBScript.

* **vbEmpty**: This is used for a data type which is not initialized with the value as 0.
* **vbNull**: This is used when there is no valid data with the value as 1.
* **vbBoolean**: This is used for a Boolean data type with the value as 11.
* **vbByte**: This is used for byte data type with the value as 17.
* **vbInteger**: This is used for integer data type with the value as 2.
* **vbLong**: This is used for the long data type with the value as 3.
* **vbSingle**: This is used for a single data type with the value as 4.
* **vbDouble**: This is used for a double data type with the value as 5.
* **vbDate:** This is used for date data type with the value as 7.
* **vbString**: This is used for string data type with the value as 8.
* **vbObject**: This is used for object data type with the value as 9.
* **vbArray**: This is used for array data type with the value as 8192.

**VBScript String Constants**

Given below are some of the String Constants in the VBScript.

* **vbCr**: This is used for print and display functions representing a carriage return character, basically for the purpose of returning to the beginning of the line. Value of this is Chr(13).
* **vbCrLf**: This is also used for print and display functions representing a carriage return with line feed character having values as Chr(13) & Chr(10). This works in the same manner as in the above case of pressing the Enter key.
* **vbformfeed**: This is used for ‘form feed’ which means for advancing downwards to the next page or in more common terms, works as a page separator. This has the value of Chr(12) and it does not work in OS Windows.
* **vbLf**: This is used for advancing downwards to the next line i.e. to go to the next line. This represents a line feed character with the value of Chr(10).
* **vbNewLine**: This is used at times in the combination of vbCrLf and vbLf i.e. Chr(13) & Chr(10) and sometimes only vbLf works i.e. Chr(10). This is a platform-dependent one.
* **vbNullChar**: This represents the character which is having the value as 0. This has its value as ‘0’.
* **vbNullString**: This represents a string which is having value 0. This is not equivalent to zero length string
* **vbTab**: This is used for providing horizontal tab, having its value as Chr(9).

**VBScript Date and Time Constants**

Given below are some of the Date and Time Constants in the VBScript.

* **vbSunday**: This represents the week of the day ‘Sunday’ holding the value as 1. If you want to work in the script having the usage of Sunday then can use this constant.
* **vbMonday**: This represents the week of the day ‘Monday’ holding the value as 2. If you want to work in the script having the usage of Monday then can use this constant.
* **vbTuesday**: This represents the week of the day ‘Tuesday’ holding the value as 3. If you want to work in the script having the usage of Tuesday then can use this constant.
* **vbWednesday**: This represents the week of the day ‘Wednesday’ holding the value as 4. If you want to work in the script having the usage of Wednesday then can use this.
* **vbThursday**: This represents the week of the day ‘Thursday’ holding the value as 5. If you want to work in the script having the usage of Thursday then can use this constant.
* **vbFriday**: This represents the week of the day ‘Friday’ holding the value as 6. If you want to work in the script having the usage of Friday then can use this constant.
* **vbSaturday**: This represents the week of the day ‘Saturday’ holding the value as 7. If you want to work in the script having the usage of Saturday then can use this constant.
* **vbFirstFourDays**: This makes use of the week that has at least 4 days in the new year, holding the value as 2.

**Concatenation Operators**

There are 2 Concatenation operators in the VBScript. As the name itself suggests, concatenation means joining. This works for numeric and string values. Let’s see its operators as follows:

**+:** This is our arithmetic operator ‘+’ only. If we take 2 strings as “Good” and “Bad” then this operator will concatenate this to Good + Bad = GoodBad and if we take 2 numeric values as 1 and 2 so concatenation in this case using ‘+’ will be 1 + 2 = 3.

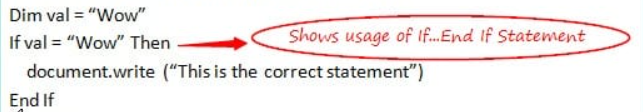
**&:** This is our normal ‘&’ symbol. This concatenates 2 values using this symbol. If we consider the above example itself, then String concatenation will be Good & Bad = GoodBad and Numeric concatenation will be 1 & 2 = 12

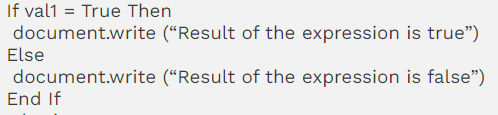
**Conditional Statements**

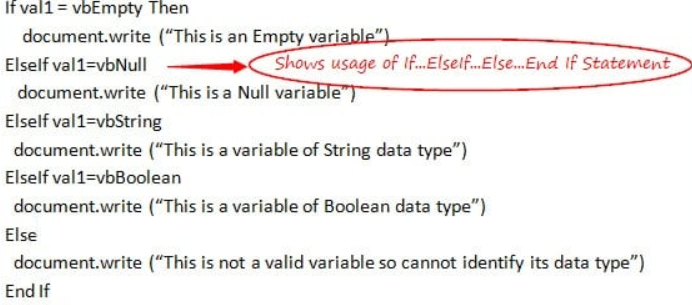
VBScript language supports 4 kinds of Conditional Statements.

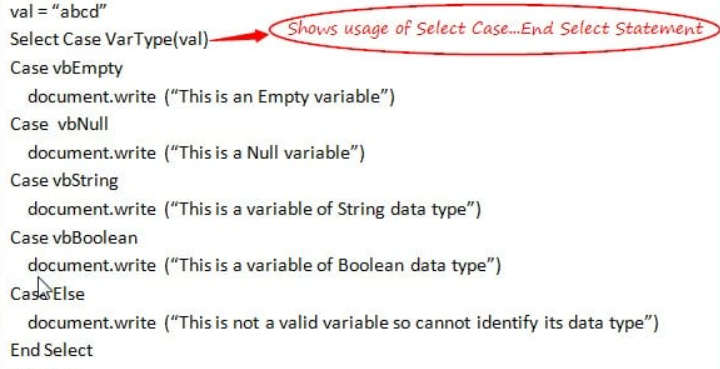
***Following is the list of Conditional Statements:***

* VBScript If…End If Statement
* VBScript If…Else…End If Statement
* VBScript If…ElseIf…Else…End If Statement
* VBScript Select Case…End Select Statement

If End If -> 

If Else End If -> 

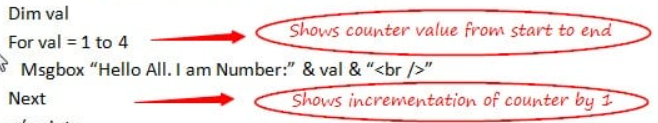
If ElseIf End if -> 

Select Case -> 

**Loops in vbScript**

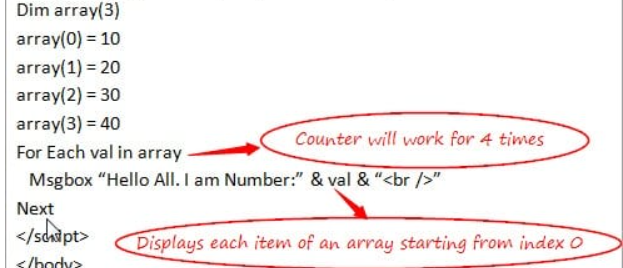
*Broadly, there are 3 types of loops in the VBScript, which are as follows:*

* For Loop
* Do Loop
* While Loop

For loop -> 

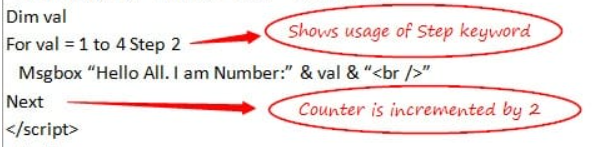
**For Each Loop**

For Each Loop is an extension of For Loop. This is used in case of**‘Arrays’**. When you want to repeat the code for each index value of an array then you can use ‘For Each Loop’. This works in the same manner as the above but the implementation is slightly different.



**For Loop with ‘Step’ keyword and ‘Exit For’ Statement**

In the case of ‘For Loop’, the counter is incremented by 1 when it comes to ‘Next’ keyword. But if you wish to change this value and if you want to specify the counter value by yourself then you can do so with the help of ‘**Step**’ keyword. It can be a **positive** or **negative** value depending on the requirement and accordingly it will increase or decrease the counter value.



**‘Exit For’** is used to exit from the ‘For Loop’ block of the code. If anytime, in between the loop you wish to exit, then you can do so using the ‘Exit For’ Statement. In the above example, ‘For Loop’ is terminated when a value is equal to 3 and hence, the message is displayed only once.

**#2) Do Loop**

Do Loops are used when you are not sure about the number of **repetitions** (unlike in case of For Loop) that can take place in the code on the basis of some conditions. There are 2 types of Do Loops in the VBScript.

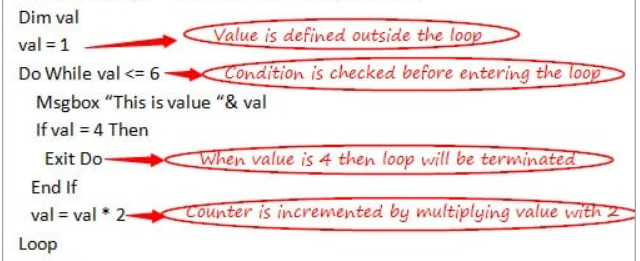
* **Do While Loop**
* **Do Until Loop**

**Do While Loop**

This makes use of keywords ‘Do’ and ‘While’. This can further be divided into **2 cases** depending upon the placement of the ‘Do’ and ‘While’ keywords. In the first case, Do and While are used in the beginning of the loop and in other cases, Do is used in the beginning of the Loop whereas While is used at the end of the loop.

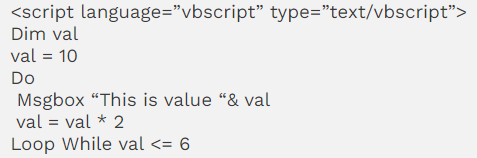
***Let’s see the implementation of both with the help of some Simple Examples:***

**Case 1: Do While….Loop**



**Case 2: Do….Loop While**

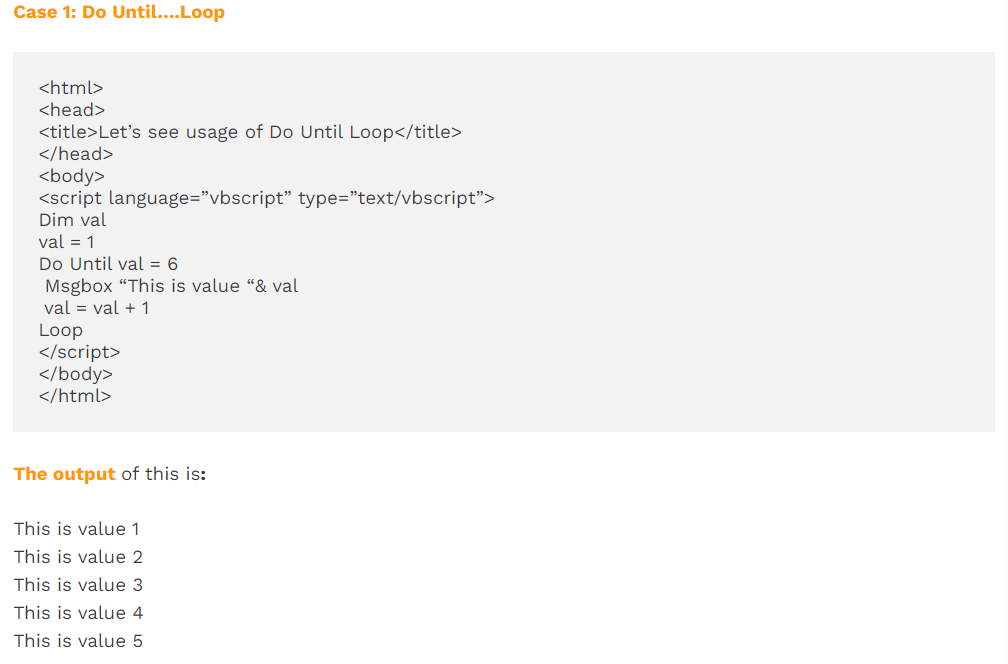
As I mentioned in above note that Do While maybe not able to execute even at once when the condition is not satisfied at all. Do….While solves this issue and in this case even if the condition is not satisfied but at least**one-time** loop can be executed.



**Do Until Loop**

This works in the same manner as ‘Do While’ Loops but with a difference that the **Do While** loop initially checks the condition and if it is**true** only after that the statements are executed and in the case of **Do Until**, the loop will be executed until the condition becomes**false**. This is used when you are not sure about the number of times when the loop can be executed.

* Do Until Loop is also divided into 2 cases like in the case of Do While.



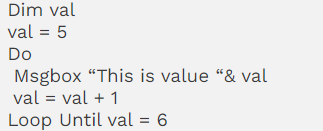
***Let’s understand the working of the code:***

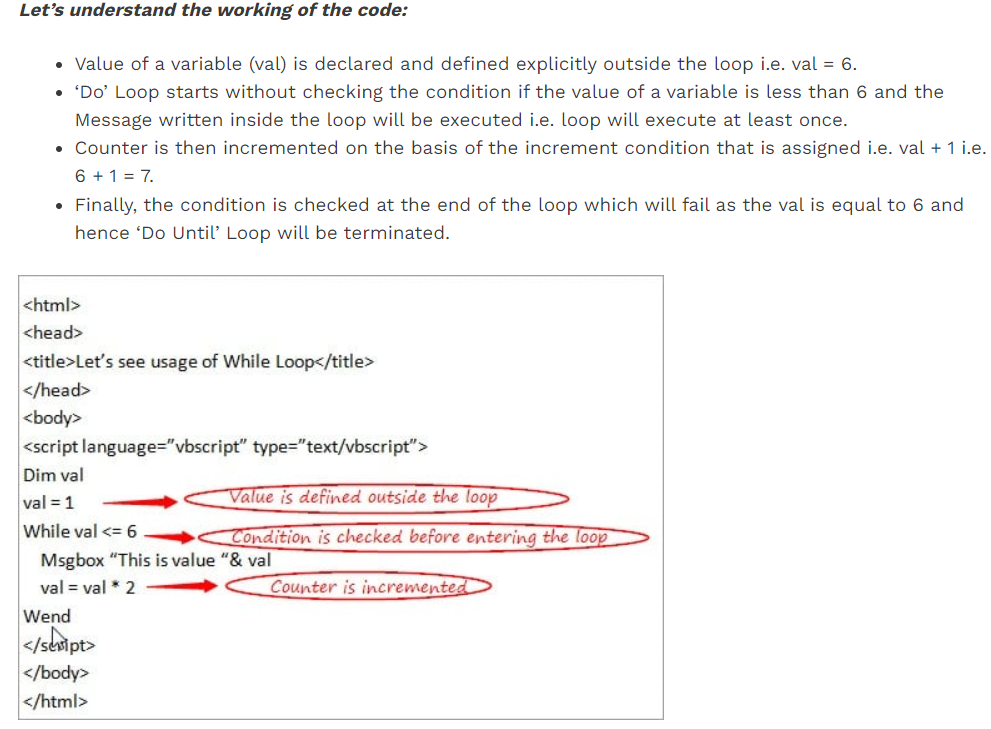
* Value of a variable (val) is declared and defined explicitly outside the loop i.e. val = 1.
* ‘Do Until’ Loop starts with the checking of the condition that the value of a variable should not be equal to 6.
* The message written inside the loop displays when the condition is satisfied.
* Counter is then incremented on the basis of the increment condition that is assigned i.e. here it is incrementing by 1 i.e. val = val + 1
* Loop will work till val = 5 as when the val becomes 6 then the condition becomes false and the loop will come to an end.

**Note**: If the value of a variable is declared as 6 (val = 6) in the above example then ‘Do Until’ Loop cannot be executed even at once as when val =6, the condition becomes false and a loop cannot be executed at all.

**Case 2: Do….Loop Until**

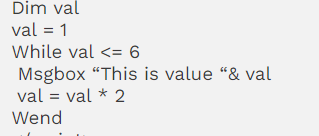
As mentioned in the above note that ‘Do Until’ loop may not be able to execute even at once when the condition is not satisfied at all; Do….Until solves this issue and in this case even if the condition is not satisfied, at least **one-time** loop can be executed.





**#3) While Loop**

However, this is same as the ‘Do While’ loop which we discussed just now but as it’s good to know about all the types of loops, let’s see about this too. This is also used when you are not sure about the number of **repetitions** in a loop. This tests the condition before entering the loop.



**Functions And Procedures**

When a need arises to accomplish a particular task then a piece of code can be written including several statements under a named section inside a program which is known as **Function/Procedure** in terms of programming language.

Functions and Procedures are mainly used to provide assistance for arranging the code in a program in an organizable way. Using Function/Procedure, the same code can be used multiple times by calling them and this reduces the pain of writing the same code again and again at the Script level.

Basically, there are two different types of Procedures in the VBScript.

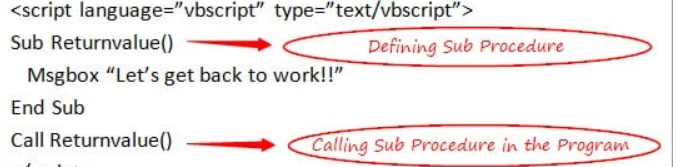
* **Sub Procedures**
* **Function Procedures**

**#1) Sub Procedures**

This is a type of procedure which includes a set of statements inside the block of code and after execution, it does not return any value. This can be defined by making use of keywords like ‘Sub’ and ‘End Sub’ respectively. It may or may not take an input. This can be parameterized i.e. can take arguments inside the brackets if required.

**Calling Sub Procedure**

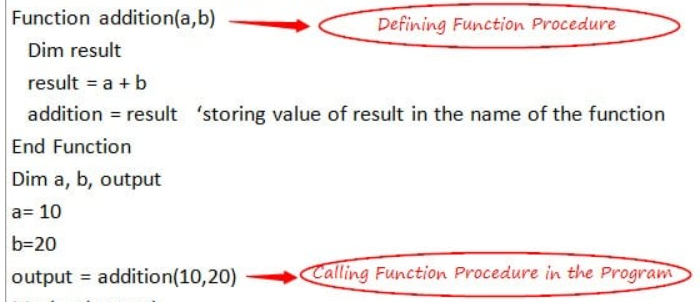
Sub Procedure called in a program using ‘**Call’** keyword following the name of the Sub procedure. If you do not want to use ‘Call’ keyword while calling Sub procedure then you can simply call it by just writing the name of the Sub Procedure without making use of the Call keyword.



**#2) Function Procedures**

This is a type of procedure which includes a set of statements inside the block of the code and after execution **may return** value also. This can take an input if required, depending upon the situations. This can be defined by making use of keywords like ‘Function’ and ‘End Function’ respectively.

Function Procedures when used with arguments i.e. by taking parameters inside the brackets, will **return a value** in such a case.



**Sub Procedure vs Functions**

* Sub Procedure never takes an input while Function Procedure may take an input if required.
* Sub Procedure starts and ends with using Sub and End Sub respectively while Function Procedure starts and ends with Function and End Function respectively.
* The most **important** difference is Sub Procedure never returns a value while the Function Procedure may return a value (as shown above).

**Inbuilt Functions in the VBScript**

Some important inbuilt functions that can be used directly in the script are explained below in detail.

#1) Format Functions

There are various inbuilt functions available for formatting the expression and they can be used directly in the script without any need for writing the code explicitly before using them in the script.

* **FormatCurrency:**This is used to convert a specified expression in the form of a Currency value.
* **FormatDateTime:**This is used to convert a specified expression in the form of a Date and Time value.
* **FormatNumber:**This is the most **important** and widely used Format Function and is used to convert the specified expression in the form of a Number.
* **FormatPercent:**This is used to convert the specified expression in the form of a Percentage value.

#2) Math and Conversion Functions

There are various inbuilt functions for performing mathematical operations and conversion purposes and can be used directly in the script without any need of writing the code explicitly before using them in the script.

* **Abs**: This is the **frequently** used Math function for the purpose of returning the Absolute value for a related number which is specified as a parameter.
* **Int:** This is Math Function and is used in those cases where it is required to fetch an integer part from a given expression/ number.
* **cDate**: This is one of the **frequently** used Conversion Function for converting the expression which includes the Date or Time parameters into Date subtype.
* **cStr:** This is the Conversion function which is used for converting the expression into String subtype.

**Working with Excel File in vbScript(**[**https://www.softwaretestinghelp.com/vbscript-excel-tutorial-11/**](https://www.softwaretestinghelp.com/vbscript-excel-tutorial-11/)**)**

Microsoft Excel needs to be installed on your computer to work with the Excel files. By creating an Excel Object, VBScript provides you the assistance to perform important operations like creating, Open and editing Excel files.

**Creating an Excel File Using Excel Object** : In this section, we will see the various steps involved in creating an excel file using the Excel Object mechanism in the VBScript.

Refer to CreateExcel.vbs for complete code on how to create excel and write data to it.

Set myxl = createobject("excel.application") **' to create excel object**

myxl.Application.Visible = true **'To make Excel visible**

myxl.Workbooks.Add **' to add a workbook**

myxl.ActiveWorkbook.SaveAs "C:\Users\satissingh\Desktop\Test.xlsx" **'Save the Excel file**

myxl.Workbooks.Open "C:\Users\satissingh\Desktop\Test.xlsx" **' Opening the excel file**

myxl.Application.Visible = true **' making it visible**

set mysheet = myxl.ActiveWorkbook.Worksheets("Sheet1") **' setting sheet name**

**'Enter values in Sheet1**.

mysheet.cells(1,1).value ="Name"

mysheet.cells(1,2).value ="Age"

mysheet.cells(2,1).value ="Ram"

mysheet.cells(2,2).value ="20"

mysheet.cells(3,1).value ="Raghu"

mysheet.cells(3,2).value ="15"

myxl.ActiveWorkbook.Save **' saving the changes**

myxl.Application.Quit **' quit the applicaiton**

Set myxl=nothing **' releasing memeory**

**Reading data from excel file**

Set obj = createobject("Excel.Application") **'Creating an Excel Object**

obj.visible=True **'Making an Excel Object visible**

Set obj1 = obj.Workbooks.open("C:\Users\satissingh\Documents\Important\VBScripting\Test.xlsx") **'Opening an Excel file**

Set obj2=obj1.Worksheets("Sheet1") **'Referring Sheet1 of excel file**

Msgbox obj2.Cells(2,2).Value **'Value from the specified cell will be read and shown**

obj1.Close **'Closing a Workbook**

obj.Quit **'Exit from Excel Application**

Set obj1=Nothing **'Releasing Workbook object**

Set obj2 = Nothing **'Releasing Worksheet object**

Set obj=Nothing **'Releasing Excel object**

**Let’s understand how it works:**

* Firstly, an Excel Object with the name **‘obj’** is created using**‘createobject’** keyword and defining Excel application in the parameter as you are creating an Excel Object.
* Then the Excel Object that is created above is made visible to the users of the sheet.
* The next step is to **open** an excel file by specifying the location of the file.
* Then, a **worksheet** of the workbook or an excel file is specified to access the data from a particular sheet of an excel file.
* Finally, the value from the particular cell (2nd column from 2nd row) is **read** and displayed with the help of a message box.
* The workbook object is then **closed** as the task has been completed.
* Excel Object is then **exited** as the task has been finished.
* Finally, all the objects are **released** by using the ‘Nothing’ keyword.

**Deleting from Excel**

set obj = CreateObject("Excel.Application")

obj.visible = True

set obj1 = obj.WorkBooks.open("C:\Users\satissingh\Documents\Important\VBScripting\Test.xlsx")

set obj2 = obj1.WorkSheets("Sheet1")

obj2.Rows("4:4").Delete ' Deleting 4th row in excel

obj1.save()

obj1.close

obj.Quit

set obj1 = Nothing

set obj2 = Nothing

**Addition & Deletion of a Sheet from an Excel File**

In this section, let’s see the different steps of adding and deleting an excel sheet from an excel file using the Excel Object mechanism in VBScript. Here also I will use the same excel file that is created above.

**Following is the Code for this scenario:**

Set obj = createobject(“Excel.Application”)   ‘**Creating an Excel Object**

obj.visible=True                                    ‘**Making an Excel Object visible**

Set obj1 = obj.Workbooks.open(“C:\newexcelfile.xls”)    ‘**Opening an Excel file**

Set obj2=obj1.sheets.Add  **‘Adding a new sheet in the excel file**

obj2.name=”Sheet1”     **‘Assigning a name to the sheet created above**

Set obj3= obj1.Sheets(“Sheet1”)  **‘Accessing Sheet1**

obj3.Delete       **‘Deleting a sheet from an excel file**

obj1.Close**‘Closing a Workbook**

obj.Quit                                                  ‘**Exit from Excel Application**

Set obj1=Nothing                                 ‘**Releasing Workbook object**

Set obj2 = Nothing                               **‘Releasing Worksheet object**

Set obj3 = Nothing**‘Releasing Worksheet object**

Set obj=Nothing                                   ‘**Releasing Excel object**

**Copying & Pasting of Data from one Excel File to Another Excel File**

In this section, we will see the different steps involved in copying/pasting data from one excel file to another excel file using the Excel Object mechanism in the VBScript. I have used the same excel file that was used in the above scenarios.

**Following is the Code for this scenario:**

Set obj = createobject(“Excel.Application”)   ‘**Creating an Excel Object**

obj.visible=True                                    ‘**Making an Excel Object visible**

Set obj1 = obj.Workbooks.open(“C:\newexcelfile.xls”)    ‘**Opening an Excel file1**

Set obj2 = obj.Workbooks.open(“C:\newexcelfile1.xls”)    ‘**Opening an Excel file2**

obj1.Worksheets(“Sheet1”).usedrange.copy  **‘Copying from an Excel File1**

obj2.Worksheets(“Sheet1”).usedrange.pastespecial  **‘Pasting in Excel File2**

obj1.Save**‘ Saving Workbook1**

obj2.Save**‘Saving Workbook2**

obj1.Close**‘Closing a Workbook**

obj.Quit                                                 ‘**Exit from Excel Application**

Set obj1=Nothing                                ‘**Releasing Workbook1 object**

Set obj2 = Nothing                              **‘Releasing Workbook2 object**

Set obj=Nothing                                  ‘**Releasing Excel object**

**ADODB Connection Objects**

Connection Objects are mainly referred to as the objects that provide support to the Coders to work and deal with the **Database Connections**. Any Databases like SQL Server, PL/SQL, etc. need to be **installed** on your computer to work with the Database Connections. **ADO** stands for **ActiveX Data Objects** and this provides the mechanism to act as an intermediary between the QTP and the Database.

**Properties and Methods of Connection Objects**

There are different properties and methods that support extracting data from the database.

**The list of ADODB Connection Object properties is as follows:**

**#1) Connection string**

This is a very useful property that is used for creating a database connection and includes connection details like the details of the **Driver, Database Server Name, Username, and Password**.

**#2) Connection Timeout:** This is used for defining the required time for **waiting** for a connection to get established.

**#3) Provider:** This property provides all provider-related details i.e. the name of the **Connection Provider**.

**#4) State:** This provides information about the state of the connection i.e. if the Connection is **ON or OFF**.

The above-mentioned ones are the properties of a Connection Object. However, there is a **RecordSet Object**also (will we discuss it in a while).

**The list of its properties is as follows:**

**#1) BOF:** This property is used to know the position of the current record. If the position of the current record is present **before the first record** of the recordset, then this property will return true.

**#2) EOF:** This is just the reverse of the above one. If the position of the current record is present **after the last record**of the recordset, then this property will return true.

**Note:** Values of EOF and BOF will be false when there are no records in the recordset. This is useful in the case of validating empty records i.e. when there are no records in the recordset.

**#3) MaxCount:** This is useful in setting the **maximum** values of rows/records that can be returned from the database i.e. if you want to fetch a maximum of 20 rows at a time from the data then you can set this property as 20.

**Let’s now take a look at the Methods:**

**The list of ADODB Connection Object and RecordSet Object methods are as follows:**

* **Open:**This method is used to open a database connection object/recordset object.
* **Execute:**This is used to execute a SQL Query that is provided.
* **Close:**This is used to close a database connection/recordset which is opened.
* **Cancel:**This is a recordset method that is used for canceling the existing execution of the Database Connection.
* **clone:** This is a recordset method and is used for getting the clone of the existing recordset object.
* **move:** This is a recordset method that is used to move the pointer at a specified location inside a recordset on the basis of the count that is mentioned.
* **count:**This is a recordset method and is used to get the total number of fields present in the recordset i.e. total number of columns in a table.
* **item(i):** This is a recordset method. This is used to get the item whose index is specified by the total number of fields present in a recordset.

These are the different properties and methods that you will use while dealing with the Connection related objects.

sdga