



VBA

visual basic for application

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

VBA stands for **V**isual **B**asic for **A**pplications, an event-driven programming language from Microsoft. It is now predominantly used with Microsoft Office applications such as MS-Excel, MS-Word and MS-Access.

This tutorial teaches the basics of VBA. Each of the sections contain related topics with simple and useful examples.

Audience

This reference has been prepared for the beginners to help them understand the basics of VBA. This tutorial will provide enough understanding on VBA from where you can take yourself to a higher level of expertise.

Prerequisites

Before proceeding with this tutorial, you should install MS Office, particularly MS-Excel.

Disclaimer & Copyright

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com.

Table of Contents

About the Tutorial.....	i
Audience	i
Prerequisites	i
Disclaimer & Copyright.....	i
Table of Contents	ii
1. VBA – OVERVIEW.....	1
2. VBA – EXCEL MACROS	3
3. VBA – EXCEL TERMS	7
Modules	7
Procedure	8
4. VBA – MACRO COMMENTS	9
5. VBA – MESSAGE BOX.....	10
6. VBA – INPUTBOX	14
7. VBA – VARIABLES.....	17
Data Types	17
8. VBA – CONSTANTS.....	20
9. VBA – OPERATORS.....	22
The Arithmetic Operators.....	22
The Comparison Operators	24
The Logical Operators	26
The Concatenation Operators	28

10. VBA – DECISIONS.....	30
If Statement	31
If Else Statement	32
If Elseif - Else statement.....	34
Nested If Statement	36
Switch Statement.....	38
11. VBA – LOOPS	40
For Loop	41
For Each Loops	43
While Wend Loops	45
Do While Loops	46
Do Until Loops.....	49
Loop Control Statements.....	53
Exit For	53
Exit Do.....	54
12. VBA – STRINGS	56
Instr	57
InString Reverse	58
LCase	60
UCase	60
Left.....	61
Right	62
Mid	63
Ltrim	64
Rtrim	64
Trim	65
Len	65

Replace	66
Space.....	67
StrComp	68
String Function	69
String Reverse Function.....	70
13. VBA – DATE-TIME FUNCTION	71
Date Functions	71
Date Function.....	72
CDate Function.....	72
DateAdd Function	73
DateDiff Function	75
DatePart Function	77
DateSerial Function.....	79
Format DateTime Function.....	80
IsDate Function	81
Day Function	81
Month Function	82
Year Function	82
Month Name.....	83
WeekDay.....	84
WeekDay Name	85
Time Functions.....	86
Now Function.....	87
Hour Function	87
Minute Function.....	88
Second Function.....	88
Time Function	89

Timer Function	89
Time Serial Function	90
TimeValue Function	91
14. VBA – ARRAYS.....	92
Array Declaration	92
Assigning Values to an Array	92
Multi-Dimensional Arrays	93
ReDim Statement	94
Array Methods	96
LBound Function	96
UBound Function	97
Split Function	99
Join Function	100
Filter Function	101
IsArray Function	102
Erase Function.....	103
15. VBA – USER-DEFINED FUNCTIONS.....	105
Function Definition	105
Calling a Function	106
16. VBA – SUB PROCEDURE	108
Calling Procedures.....	108
17. VBA – EVENTS.....	110
Worksheet Events	110
Workbook Events	111

18. VBA – ERROR HANDLING	114
Syntax Errors	114
Runtime Errors	114
Logical Errors.....	115
Err Object	115
Error Handling	115
19. VBA – EXCEL OBJECTS.....	117
Application Objects	117
Workbook Objects	117
Worksheet Objects.....	118
Range Objects	118
20. VBA – TEXT FILES	119
File System Object (FSO)	119
Write Command.....	124
21. VBA – PROGRAMMING CHARTS	126
22. VBA – USER FORMS	129

1. VBA – Overview

VBA stands for **V**isual **B**asic for **A**pplications an event-driven programming language from Microsoft that is now predominantly used with Microsoft office applications such as MS-Excel, MS-Word, and MS-Access.

It helps techies to build customized applications and solutions to enhance the capabilities of those applications. The advantage of this facility is that you NEED NOT have visual basic installed on our PC, however, installing Office will implicitly help in achieving the purpose.

You can use VBA in all office versions, right from MS-Office 97 to MS-Office 2013 and also with any of the latest versions available. Among VBA, Excel VBA is the most popular. The advantage of using VBA is that you can build very powerful tools in MS Excel using linear programming.

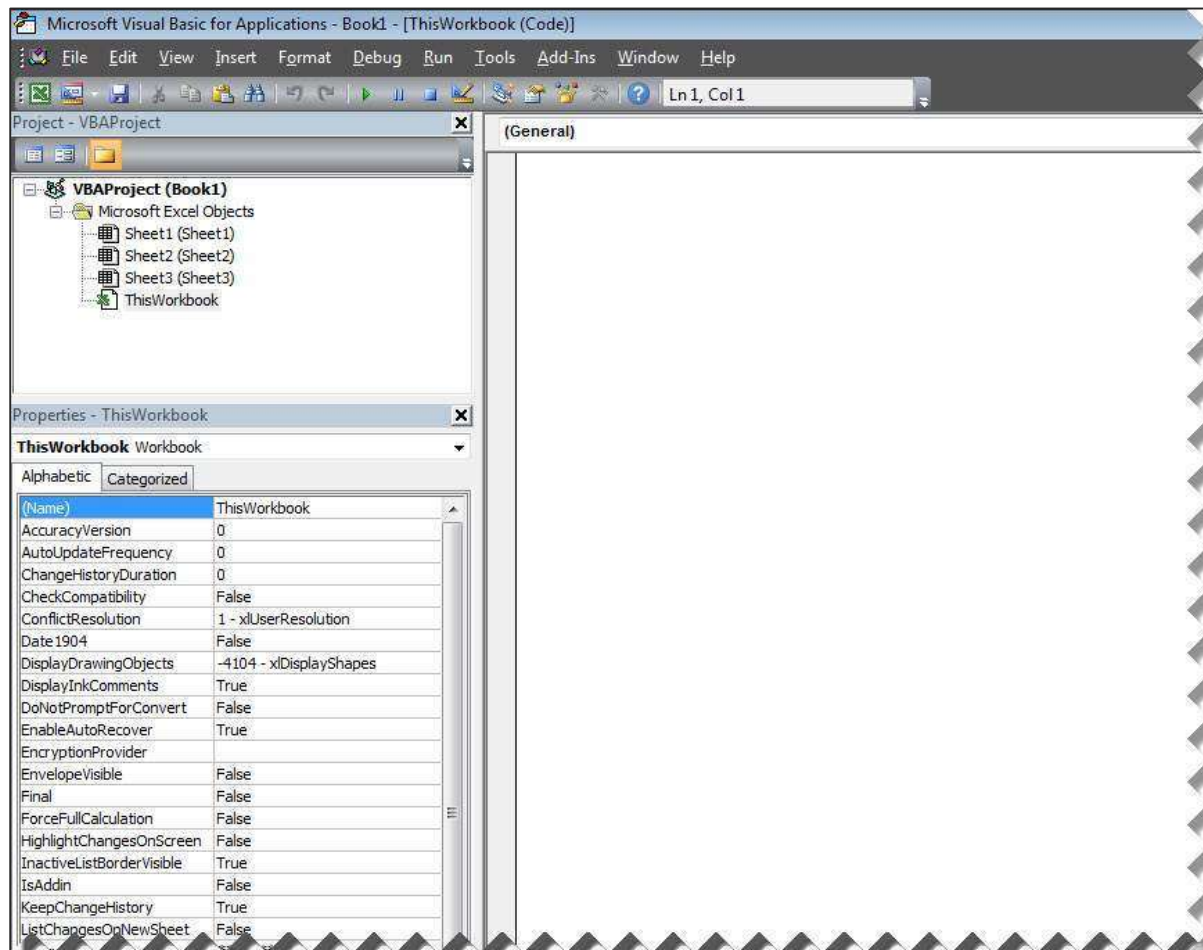
Application of VBA

You might wonder why to use VBA in Excel as MS-Excel itself provides loads of inbuilt functions. MS-Excel provides only basic inbuilt functions which might not be sufficient to perform complex calculations. Under such circumstances, VBA becomes the most obvious solution.

For example, it is very hard to calculate the monthly repayment of a loan using Excel's built-in formulas. Rather, it is easy to program a VBA for such a calculation.

Accessing VBA Editor

In Excel window, press "ALT+F11". A VBA window opens up as shown in the following screenshot.

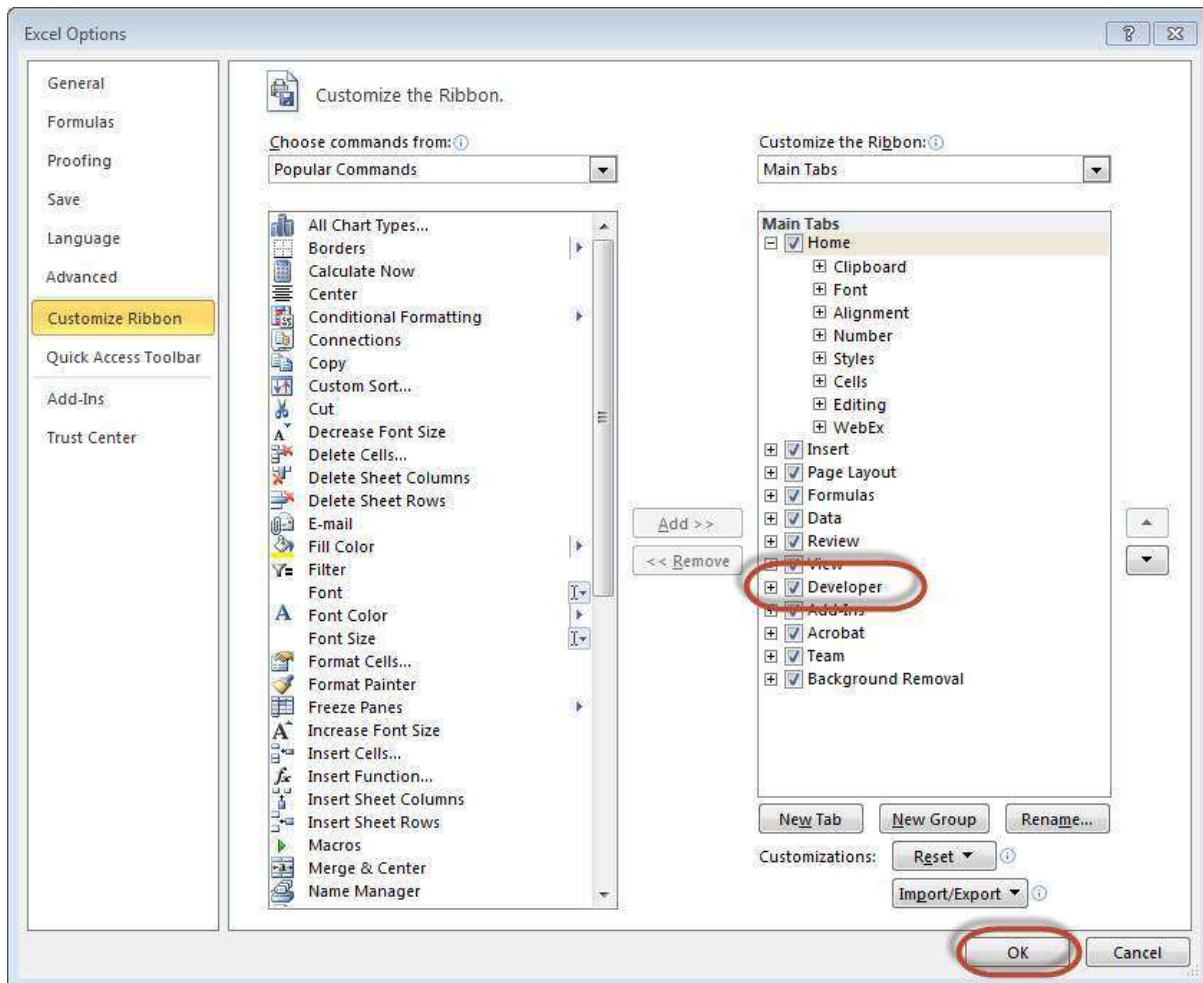


2. VBA – Excel Macros

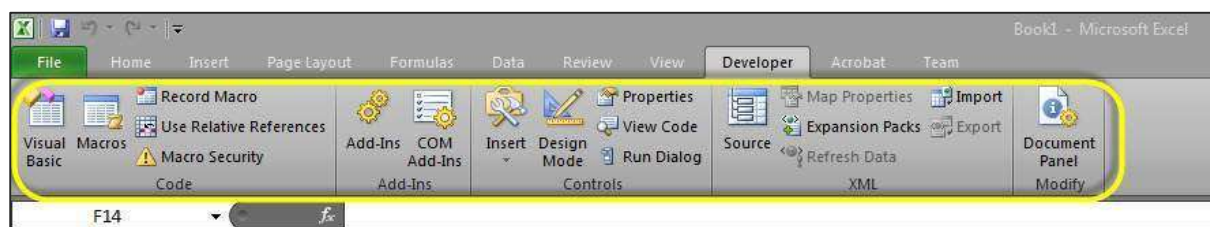
In this chapter, you will learn how to write a simple macro in a step by step manner.

Step 1: First, enable 'Developer' menu in Excel 20XX. To do the same, click File -> Options.

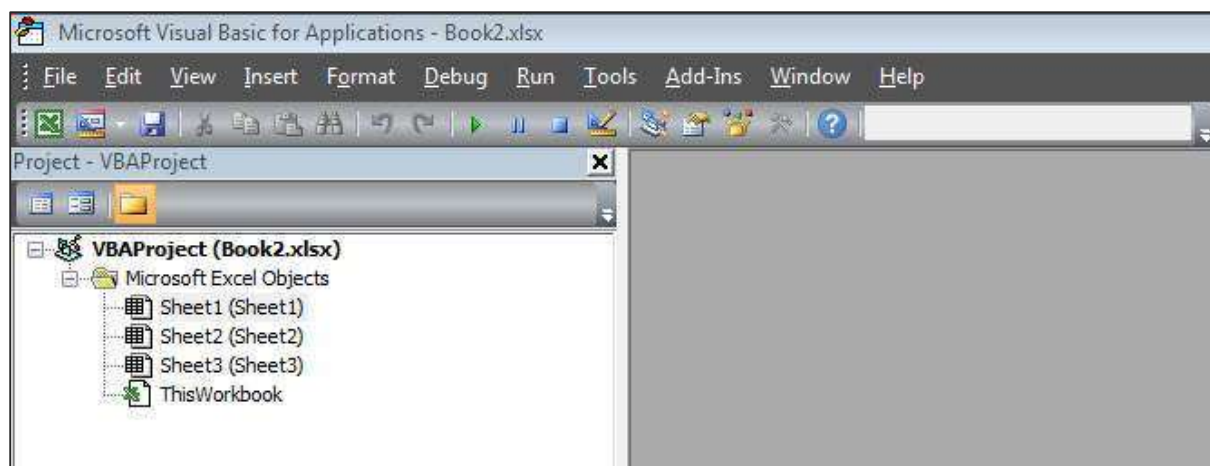
Step 2: Click 'Customize the Ribbon' tab and check 'Developer'. Click 'OK'.



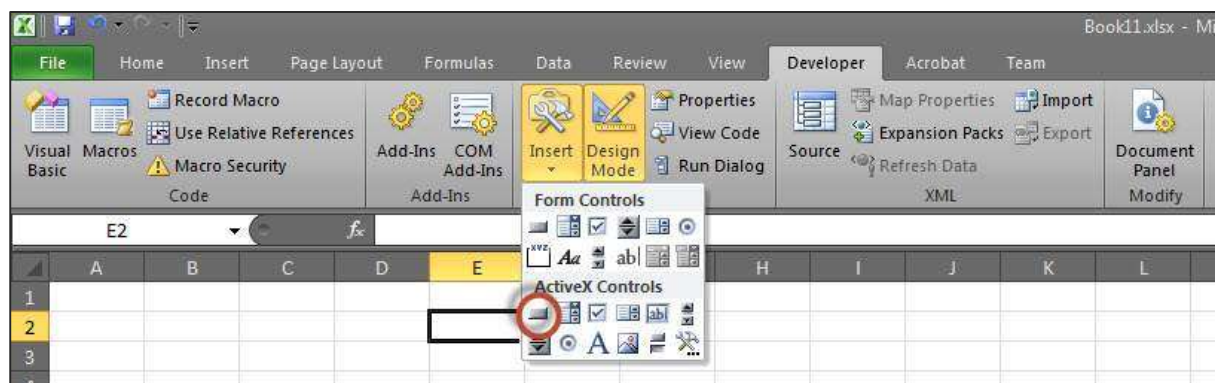
Step 3: The 'Developer' ribbon appears in the menu bar.



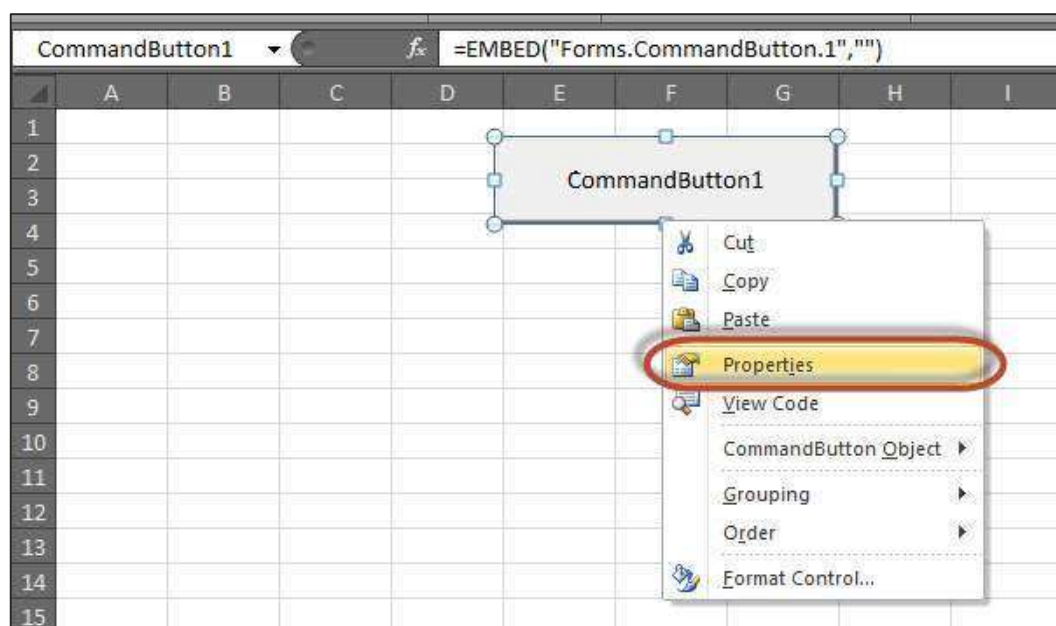
Step 4: Click the 'Visual Basic' button to open the VBA Editor.



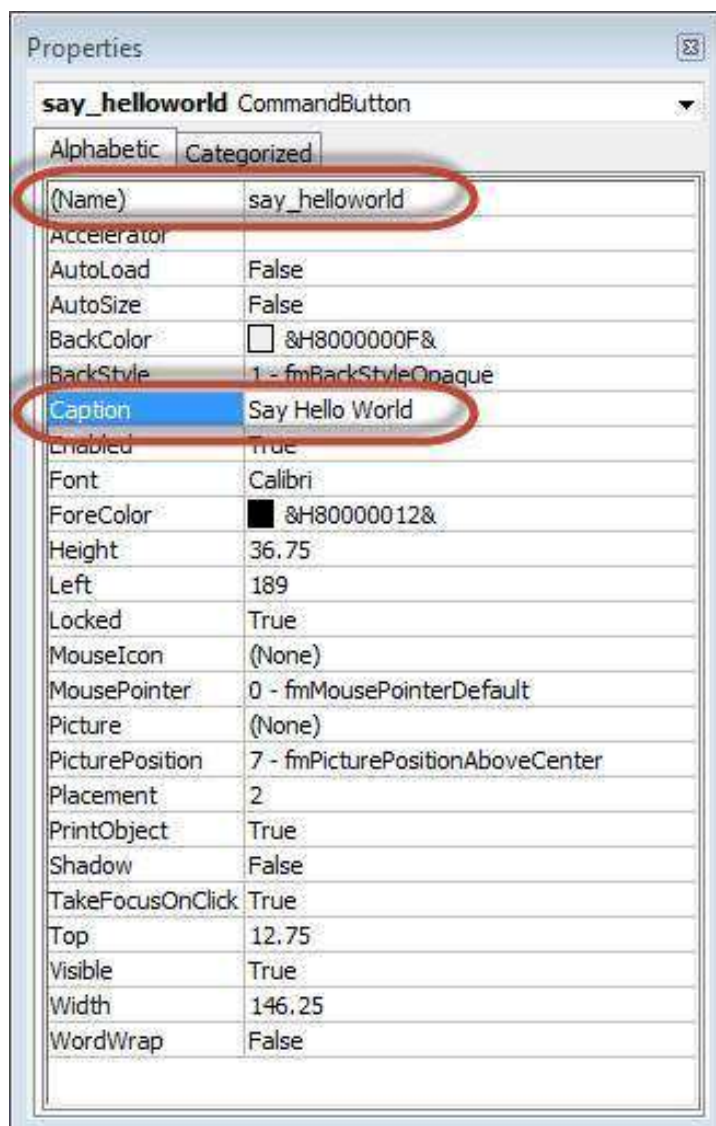
Step 5: Start scripting by adding a button. Click Insert -> Select the button.



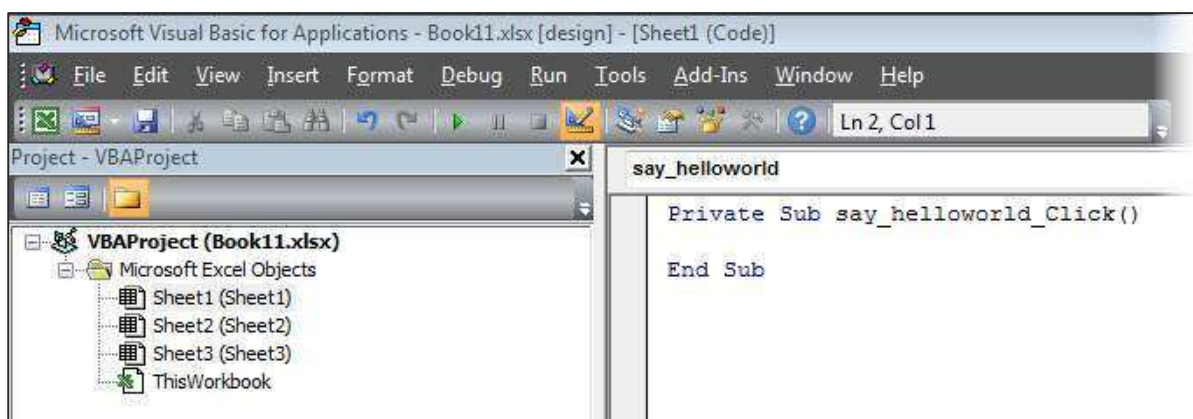
Step 6: Perform a right-click and choose 'properties'.



Step 7: Edit the name and caption as shown in the following screenshot.



Step 8: Now double-click the button and the sub-procedure outline will be displayed as shown in the following screenshot.



Step 9: Start coding by simply adding a message.

```
Private Sub say_helloworld_Click()  
    MsgBox "Hi"  
End Sub
```

Step 10: Click the button to execute the sub-procedure. The output of the sub-procedure is shown in the following screenshot.



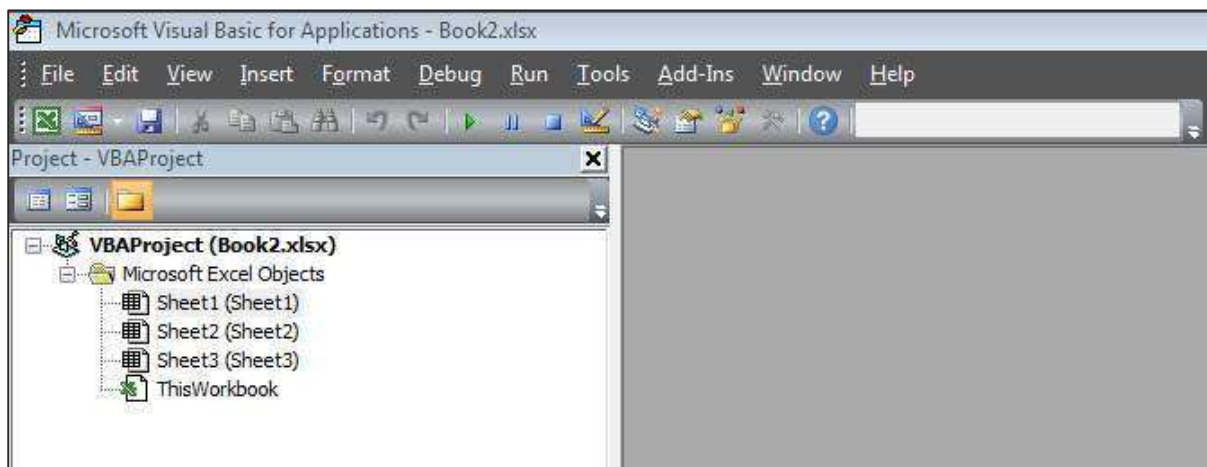
Note: In further chapters, we will demonstrate using a simple button, as explained from step#1 to 10. Hence , it is important to understand this chapter thoroughly.

3. VBA – Excel Terms

In this chapter, you will acquaint yourself with the commonly used excel VBA terminologies. These terminologies will be used in further modules, hence understanding each one of these is important.

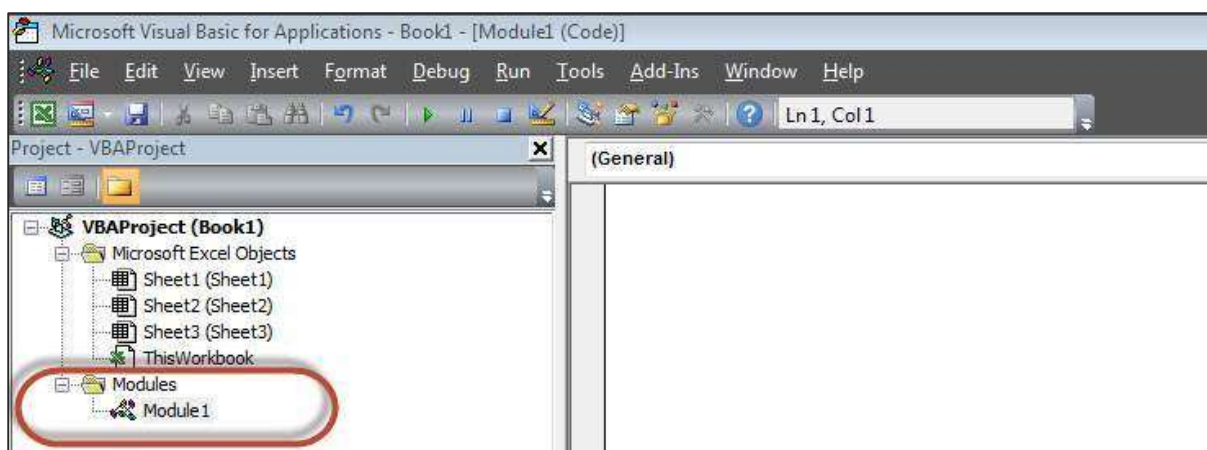
Modules

Modules is the area where the code is written. This is a new Workbook, hence there aren't any Modules.



To insert a Module, navigate to Insert -> Module. Once a module is inserted 'module1' is created.

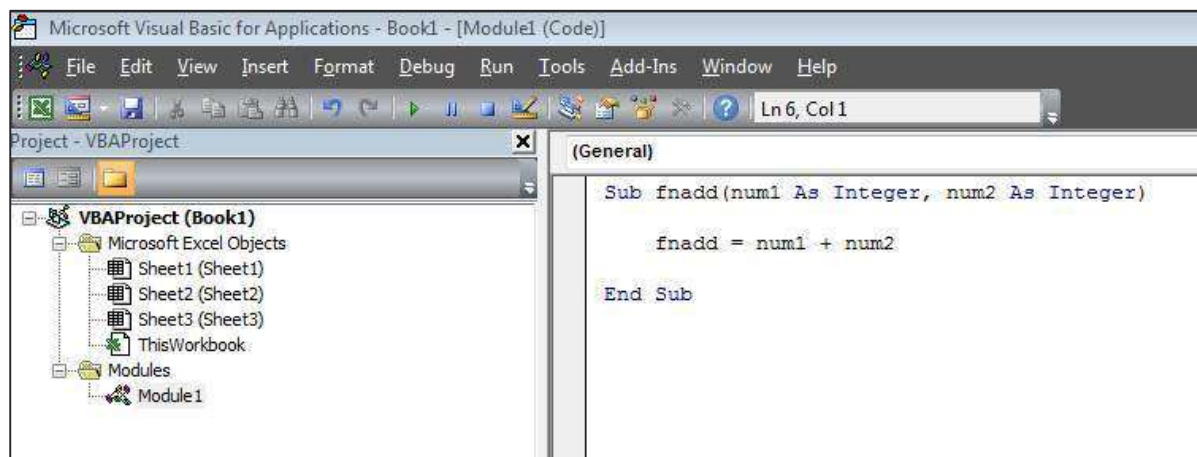
Within the modules, we can write VBA code and the code is written within a Procedure. A Procedure/Sub Procedure is a series of VBA statements instructing what to do.



Procedure

Procedures are a group of statements executed as a whole, which instructs Excel how to perform a specific task. The task performed can be a very simple or a very complicated task. However, it is a good practice to break down complicated procedures into smaller ones.

The two main types of Procedures are Sub and Function.



Function

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

Apart from inbuilt Functions, VBA allows to write user-defined functions as well and statements are written between **Function** and **End Function**.

Sub-procedures

Sub-procedures work similar to functions. While sub procedures DO NOT Return a value, functions may or may not return a value. Sub procedures CAN be called without call keyword. Sub procedures are always enclosed within **Sub** and **End Sub** statements.

4. VBA – Macro Comments

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

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

Comments in VBA are denoted by two methods.

- Any statement that starts with a Single Quote (') is treated as comment. Following is an example.

```
' This Script is invoked after successful login  
' Written by : TutorialPoint  
' Return Value : True / False
```

- Any statement that starts with the keyword "REM". Following is an example.

```
REM This Script is written to Validate the Entered Input  
REM Modified by : Tutorial point/user2
```


5. VBA – Message Box

The **MsgBox function** displays a message box and waits for the user to click a button and then an action is performed based on the button clicked by the user.

Syntax

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

Parameter Description

- **Prompt** - A Required Parameter. A String that is displayed as a message in the dialog box. The maximum length of prompt is approximately 1024 characters. If the message extends to more than a line, then the lines can be separated using a carriage return character (Chr(13)) or a linefeed character (Chr(10)) between each line.
- **Buttons** - An Optional Parameter. A Numeric expression that specifies the type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If left blank, the default value for buttons is 0.
- **Title** - An Optional Parameter. A String expression displayed in the title bar of the dialog box. If the title is left blank, the application name is placed in the title bar.
- **Helpfile** - An Optional Parameter. A String expression that identifies the Help file to use for providing context-sensitive help for the dialog box.
- **Context** - An Optional Parameter. A Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If context is provided, helpfile must also be provided.

The **Buttons** parameter can take any of the following values:

- 0 vbOKOnly - Displays OK button only.
- 1 vbOKCancel - Displays OK and Cancel buttons.
- 2 vbAbortRetryIgnore - Displays Abort, Retry, and Ignore buttons.
- 3 vbYesNoCancel - Displays Yes, No, and Cancel buttons.
- 4 vbYesNo - Displays Yes and No buttons.
- 5 vbRetryCancel - Displays Retry and Cancel buttons.
- 16 vbCritical - Displays Critical Message icon.
- 32 vbQuestion - Displays Warning Query icon.
- 48 vbExclamation - Displays Warning Message icon.
- 64 vbInformation - Displays Information Message icon.

- 0 vbDefaultButton1 - First button is default.
- 256 vbDefaultButton2 - Second button is default.
- 512 vbDefaultButton3 - Third button is default.
- 768 vbDefaultButton4 - Fourth button is default.
- 0 vbApplicationModal Application modal - The current application will not work until the user responds to the message box.
- 4096 vbSystemModal System modal - All applications will not work until the user responds to the message box.

The above values are logically divided into four groups: The **first group** (0 to 5) indicates the buttons to be displayed in the message box. The **second group** (16, 32, 48, 64) describes the style of the icon to be displayed, the **third group** (0, 256, 512, 768) indicates which button must be the default, and the **fourth group** (0, 4096) determines the modality of the message box.

Return Values

The MsgBox function can return one of the following values which can be used to identify the button the user has clicked in the message box.

- 1 - vbOK - OK was clicked
- 2 - vbCancel - Cancel was clicked
- 3 - vbAbort - Abort was clicked
- 4 - vbRetry - Retry was clicked
- 5 - vbIgnore - Ignore was clicked
- 6 - vbYes - Yes was clicked
- 7 - vbNo - No was clicked

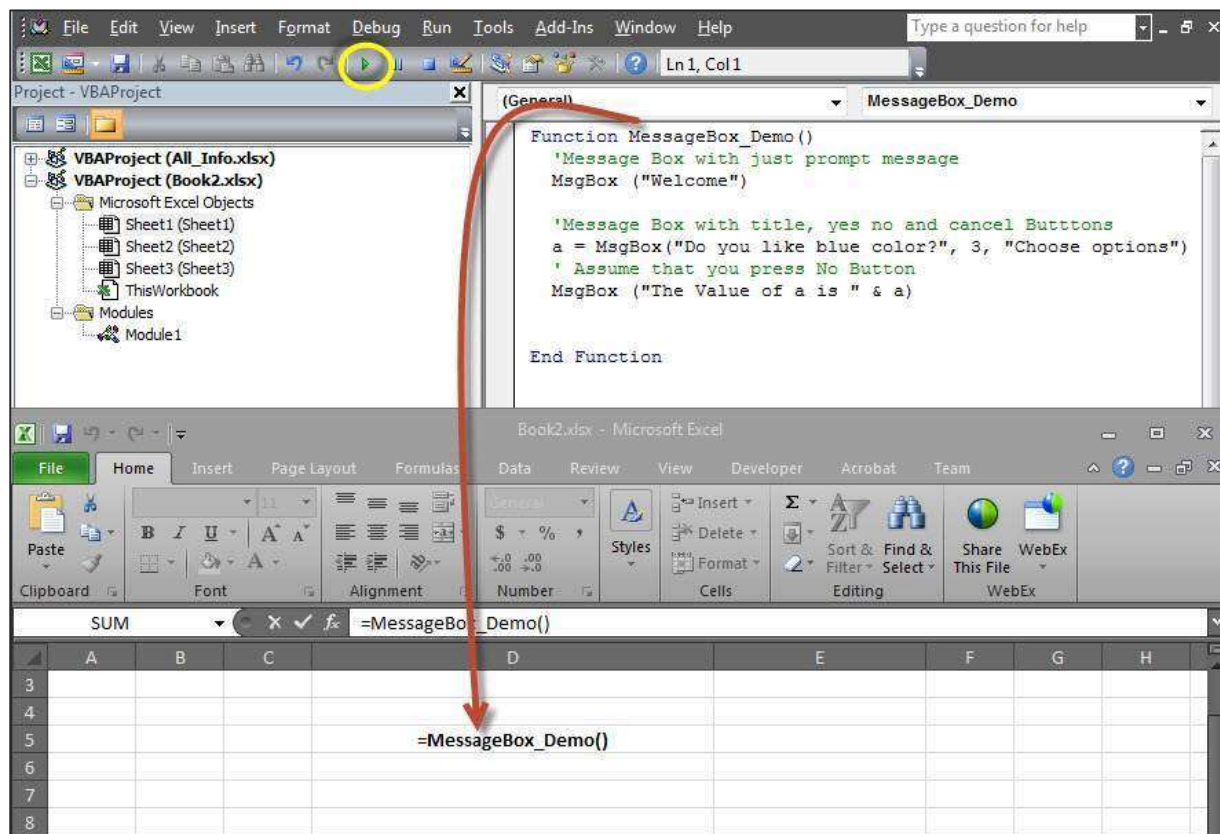
Example

```
Function MsgBox_Demo()
'Message Box with just prompt message
MsgBox("Welcome")

'Message Box with title, yes no and cancel Buttons
a = MsgBox("Do you like blue color?",3,"Choose options")
' Assume that you press No Button
msgbox ("The Value of a is " & a)
End Function
```

Output

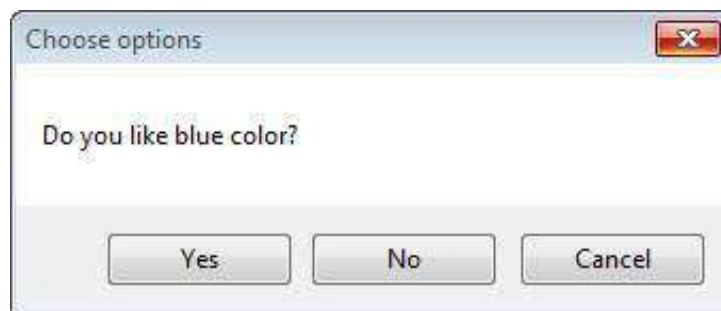
Step 1: The above Function can be executed either by clicking the "Run" button on VBA Window or by calling the function from Excel Worksheet as shown in the following screenshot.



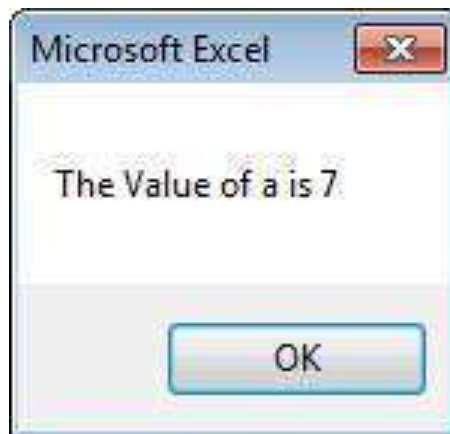
Step 2: A Simple Message box is displayed with a message "Welcome" and an "OK" Button



Step 3: After Clicking OK, yet another dialog box is displayed with a message along with "yes, no, and cancel" buttons.



Step 4: After clicking the 'No' button, the value of that button (7) is stored as an integer and displayed as a message box to the user as shown in the following screenshot. Using this value, it can be understood which button the user has clicked.



6. VBA – InputBox

The **InputBox function** prompts the users to enter values. After entering the values, if the user clicks the OK button or presses ENTER on the keyboard, the InputBox function will return the text in the text box. If the user clicks the Cancel button, the function will return an empty string ("").

Syntax

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

Parameter Description

- **Prompt** - A required parameter. A String that is displayed as a message in the dialog box. The maximum length of prompt is approximately 1024 characters. If the message extends to more than a line, then the lines can be separated using a carriage return character (Chr(13)) or a linefeed character (Chr(10)) between each line.
- **Title** - An optional parameter. A String expression displayed in the title bar of the dialog box. If the title is left blank, the application name is placed in the title bar.
- **Default** - An optional parameter. A default text in the text box that the user would like to be displayed.
- **XPos** - An optional parameter. The position of **X** axis represents the prompt distance from the left side of the screen horizontally. If left blank, the input box is horizontally centered.
- **YPos** - An optional parameter. The position of **Y** axis represents the prompt distance from the left side of the screen vertically. If left blank, the input box is vertically centered.
- **Helpfile** - An optional parameter. A String expression that identifies the helpfile to be used to provide context-sensitive Help for the dialog box.
- **Context** - An optional parameter. A Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If context is provided, helpfile must also be provided.

Example

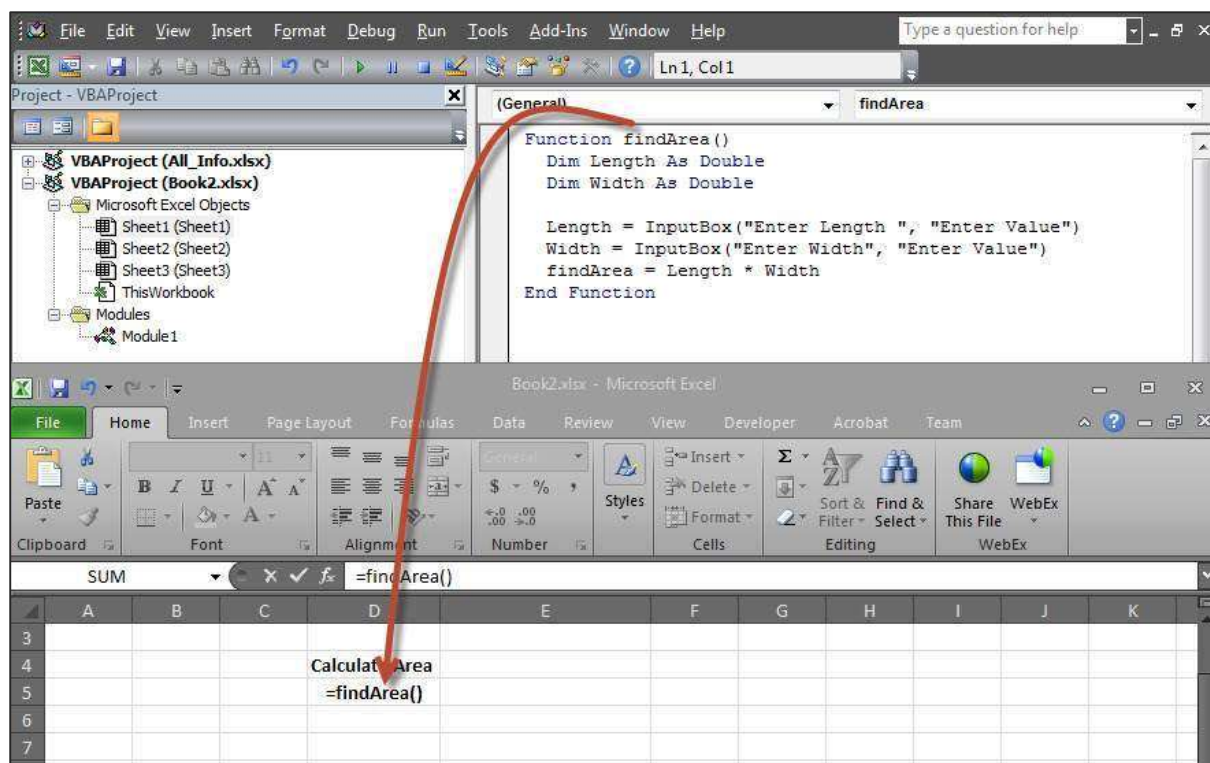
Let us calculate the area of a rectangle by getting values from the user at run time with the help of two input boxes (one for length and one for width).

```
Function findArea()
    Dim Length As Double
    Dim Width As Double

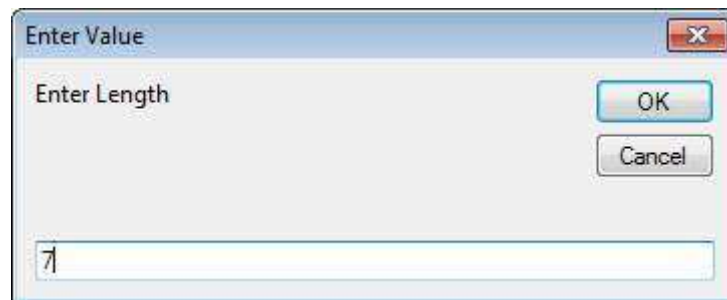
    Length = InputBox("Enter Length ", "Enter a Number")
    Width = InputBox("Enter Width", "Enter a Number")
    findArea = Length * Width
End Function
```

Output

Step 1: To execute the same, call using the function name and press Enter as shown in the following screenshot.




Step 2: Upon execution, the First input box (length) is displayed. Enter a value into the input box.



The screenshot shows a VBA 'Enter Value' dialog box. The title bar is 'Enter Value' with a close button. The text inside says 'Enter Length'. There are 'OK' and 'Cancel' buttons on the right. At the bottom, there is a text input field containing the number '7'.

Step 3: After entering the first value, the second input box (width) is displayed.



The screenshot shows a VBA 'Enter Value' dialog box. The title bar is 'Enter Value' with a close button. The text inside says 'Enter Width'. There are 'OK' and 'Cancel' buttons on the right. At the bottom, there is a text input field containing the number '4'.

Step 4: Upon entering the second number, click the OK button. The area is displayed as shown in the following screenshot.

B	C	D	E
		Calculate Area	
		28	

7. VBA – Variables

Variable is a named memory location used to hold a value that can be changed during the script execution. Following are the basic rules for naming a variable.

- You must use a letter as the first character.
- You can't use a space, period (.), exclamation mark (!), or the characters @, &, \$, # in the name.
- Name can't exceed 255 characters in length.
- You cannot use Visual Basic reserved keywords as variable name.

Syntax

In VBA, you need to declare the variables before using them.

```
Dim <<variable_name>> As <<variable_type>>
```

Data Types

There are many VBA data types, which can be divided into two main categories, namely numeric and non-numeric data types.

Numeric Data Types

Following table displays the numeric data types and the allowed range of values.

Type	Range of Values
Byte	0 to 255
Integer	-32,768 to 32,767
Long	-2,147,483,648 to 2,147,483,648
Single	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values
Double	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807

Decimal	+/- 79,228,162,514,264,337,593,543,950,335, if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places)
---------	--

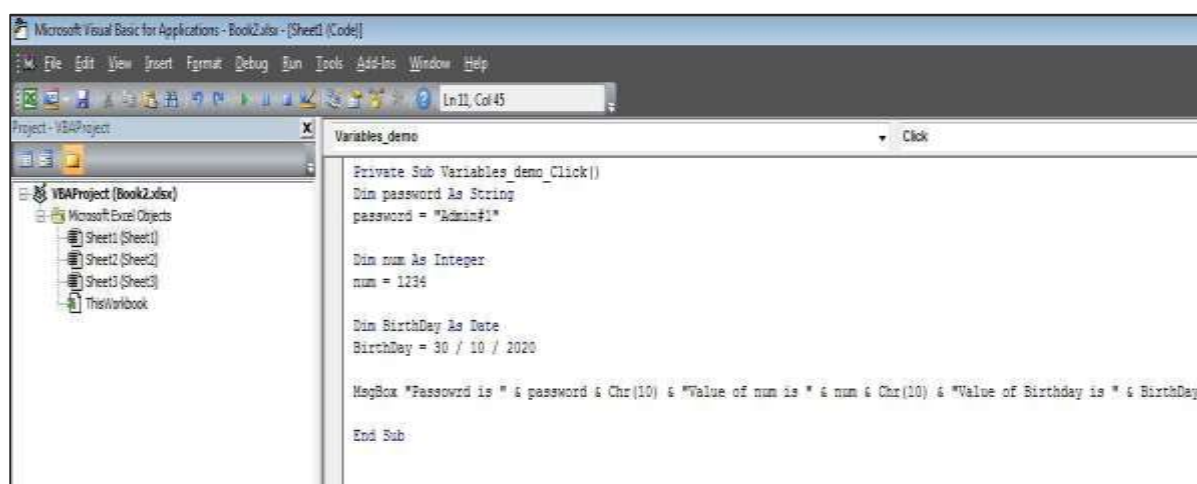
Non-Numeric Data Types

Following table displays the non-numeric data types and the allowed range of values.

Type	Range of Values
String (fixed length)	1 to 65,400 characters
String (variable length)	0 to 2 billion characters
Date	January 1, 100 to December 31, 9999
Boolean	True or False
Object	Any embedded object
Variant (numeric)	Any value as large as double
Variant (text)	Same as variable-length string

Example

Let us create a button and name it as 'Variables_demo' to demonstrate the use of variables.



```

Private Sub Variables_demo_Click()

    Dim password As String

    password = "Admin#1"

    Dim num As Integer

```

```
num = 1234

Dim BirthDay As Date
BirthDay = 30 / 10 / 2020

MsgBox "Passowrd is " & password & Chr(10) & "Value of num is " & num &
Chr(10) & "Value of Birthday is " & BirthDay

End Sub
```

Output

Upon executing the script, the output will be as shown in the following screenshot.



8. VBA – Constants

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

Following are the rules for naming a constant.

- You must use a letter as the first character.
- You can't use a space, period (.), exclamation mark (!), or the characters @, &, \$, # in the name.
- Name can't exceed 255 characters in length.
- You cannot use Visual Basic reserved keywords as variable name.

Syntax

In VBA, we need to assign a value to the declared Constants. An error is thrown, if we try to change the value of the constant.

```
Const <<constant_name>> As <<constant_type>> = <<constant_value>>
```

Example

Let us create a button "Constant_demo" to demonstrate how to work with constants.

```
Private Sub Constant_demo_Click()  
    Const MyInteger As Integer = 42  
    Const myDate As Date = #2/2/2020#  
    Const myDay As String = "Sunday"  
  
    MsgBox "Integer is " & MyInteger & Chr(10) & "myDate is " & myDate & Chr(10)  
    & "myDay is " & myDay  
  
End Sub
```

Output

Upon executing the script, the output will be displayed as shown in the following screenshot.



9. VBA – Operators

An **Operator** can be defined using a simple expression - $4 + 5$ is equal to 9. Here, 4 and 5 are called **operands** and $+$ is called **operator**. VBA supports following types of operators:

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

The Arithmetic Operators

Following arithmetic operators are supported by VBA:

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

Operator	Description	Example
+	Adds the two operands	$A + B$ will give 15
-	Subtracts the second operand from the first	$A - B$ will give -5
*	Multiplies both the operands	$A * B$ will give 50
/	Divides the numerator by the denominator	B / A will give 2
%	Modulus operator and the remainder after an integer division	$B \text{ MOD } A$ will give 0
^	Exponentiation operator	$B ^ A$ will give 100000

Arithmetic Operators — Example

Add a button and try the following example to understand all the arithmetic operators available in VBA.

```
Private Sub Constant_demo_Click()  
    Dim a As Integer  
    a = 5  
  
    Dim b As Integer  
    b = 10  
  
    Dim c As Double  
  
    c = a + b  
    MsgBox ("Addition Result is " & c)  
  
    c = a - b  
    MsgBox ("Subtraction Result is " & c)  
  
    c = a * b  
    MsgBox ("Multiplication Result is " & c)  
  
    c = b / a  
    MsgBox ("Division Result is " & c)  
  
    c = b Mod a  
    MsgBox ("Modulus Result is " & c)  
  
    c = b ^ a  
    MsgBox ("Exponentiation Result is " & c)  
End Sub
```

When you click the button or execute the above script, it will produce the following result.

```
Addition Result is 15  
  
Subtraction Result is -5  
  
Multiplication Result is 50
```

Division Result is 2

Modulus Result is 0

Exponentiation Result is 100000

The Comparison Operators

There are following comparison operators supported by VBA.

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

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

Comparison Operators – Example

Try the following example to understand all the Comparison operators available in VBA.

```
Private Sub Constant_demo_Click()

    Dim a: a = 10
    Dim b: b = 20
    Dim c

    If a = b Then
        MsgBox ("Operator Line 1 : True")
    End If
End Sub
```

```
Else
    MsgBox ("Operator Line 1 : False")
End If

If a<>b Then
    MsgBox ("Operator Line 2 : True")
Else
    MsgBox ("Operator Line 2 : False")
End If

If a>b Then
    MsgBox ("Operator Line 3 : True")
Else
    MsgBox ("Operator Line 3 : False")
End If

If a<b Then
    MsgBox ("Operator Line 4 : True")
Else
    MsgBox ("Operator Line 4 : False")
End If

If a>=b Then
    MsgBox ("Operator Line 5 : True")
Else
    MsgBox ("Operator Line 5 : False")
End If

If a<=b Then
    MsgBox ("Operator Line 6 : True")
Else
    MsgBox ("Operator Line 6 : False")
End If
End Sub
```


When you execute the above script, it will produce the following result.

```
Operator Line 1 : False

Operator Line 2 : True

Operator Line 3 : False

Operator Line 4 : True

Operator Line 5 : False

Operator Line 6 : True
```

The Logical Operators

Following logical operators are supported by VBA.

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

Operator	Description	Example
AND	Called Logical AND operator. If both the conditions are True, then the Expression is true.	a<>0 AND b<>0 is False.
OR	Called Logical OR Operator. If any of the two conditions are True, then the condition is true.	a<>0 OR b<>0 is true.
NOT	Called Logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	NOT(a<>0 OR b<>0) is false.
XOR	Called Logical Exclusion. It is the combination of NOT and OR Operator. If one, and only one, of the expressions evaluates to be True, the result is True.	(a<>0 XOR b<>0) is false.

Logical Operators – Example

Try the following example to understand all the Logical operators available in VBA by creating a button and adding the following function.

```
Private Sub Constant_demo_Click()

    Dim a As Integer

    a = 10

    Dim b As Integer
```

```
b = 0

If a <> 0 And b <> 0 Then
    MsgBox ("AND Operator Result is : True")

Else
    MsgBox ("AND Operator Result is : False")
End If

If a <> 0 Or b <> 0 Then
    MsgBox ("OR Operator Result is : True")
Else
    MsgBox ("OR Operator Result is : False")
End If

If Not (a <> 0 Or b <> 0) Then
    MsgBox ("NOT Operator Result is : True")
Else
    MsgBox ("NOT Operator Result is : False")
End If

If (a <> 0 Xor b <> 0) Then
    MsgBox ("XOR Operator Result is : True")
Else
    MsgBox ("XOR Operator Result is : False")
End If

End Sub
```

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

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

The Concatenation Operators

Following Concatenation operators are supported by VBA.

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

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

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

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

Note: Concatenation Operators can be used for both numbers and strings. The output depends on the context, if the variables hold numeric value or string value.

Concatenation Operators

Following table shows all the Concatenation operators supported by VBScript language. Assume variable A holds 5 and variable B holds 10, then -

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

Example

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

```
Private Sub Constant_demo_Click()
    Dim a as Integer : a = 5
    Dim b as Integer : b = 10
    Dim c as Integer

    c=a+b
    msgbox ("Concatenated value:1 is " &c) 'Numeric addition
    c=a&b
```

```

    msgbox ("Concatenated value:2 is " &c) 'Concatenate two numbers
End Sub

```

Try the following example to understand all the Logical operators available in VBA by creating a button and adding the following function.

```

Concatenated value:1 is 15

Concatenated value:2 is 510

```

Concatenation can also be used for concatenating two strings. Assume variable A = "Microsoft" and variable B = "VBScript" then -

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

Example

Try the following example to understand all the Logical operators available in VBA by creating a button and adding the following function.

```

Private Sub Constant_demo_Click()
    Dim a as String : a = "Microsoft"
    Dim b as String : b = "VBScript"
    Dim c as String

    c=a+b
    msgbox("Concatenated value:1 is " &c) 'addition of two Strings
    c=a&b
    msgbox("Concatenated value:2 is " &c) 'Concatenate two String
End Sub

```

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

```

Concatenated value:1 is MicrosoftVBScript

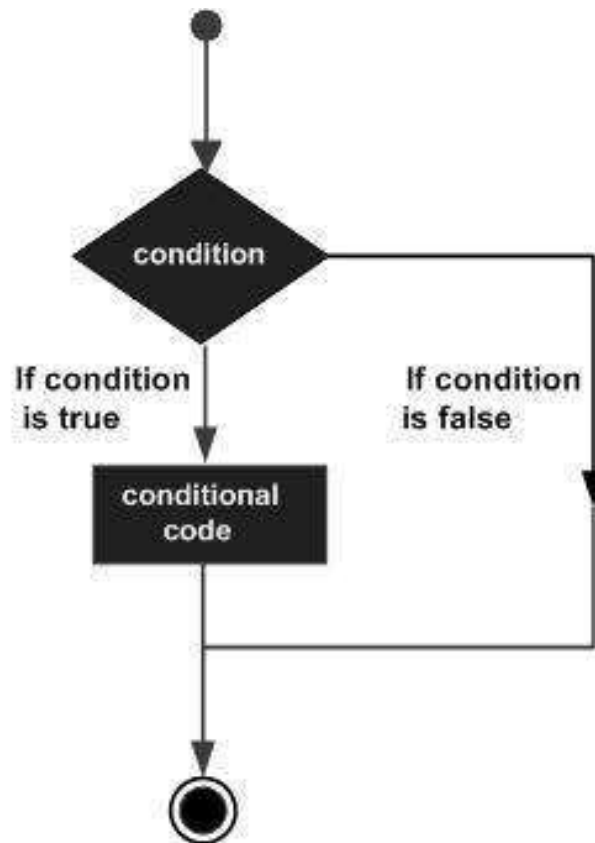
Concatenated value:2 is MicrosoftVBScript

```

10. VBA – Decisions

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

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



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

Statement	Description
<u>if statement</u>	An if statement consists of a Boolean expression followed by one or more statements.
<u>if..else statement</u>	An if else statement consists of a Boolean expression followed by one or more statements. If the condition is True, the statements under If statements are executed. If the condition is false, the Else part of the script is executed.

<u>if...elseif..else statement</u>	An if statement followed by one or more ElseIf statements, that consists of Boolean expressions and then followed by an optional else statement , which executes when all the condition become false.
<u>nested if statements</u>	An if or elseif statement inside another if or elseif statement(s).
<u>switch statement</u>	A switch statement allows a variable to be tested for equality against a list of values.

If Statement

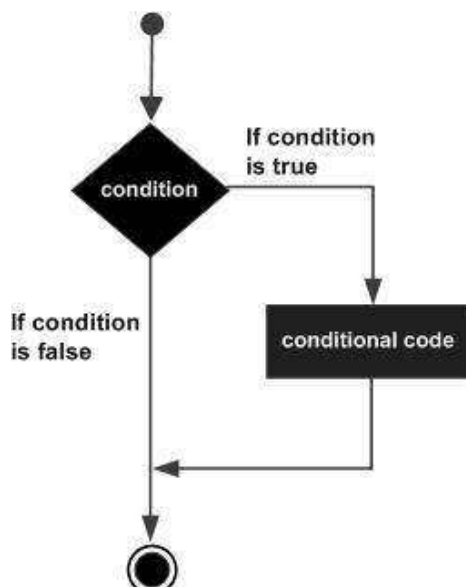
An If statement consists of a Boolean expression followed by one or more statements. If the condition is said to be True, the statements under If condition(s) are executed. If the condition is said to be False, the statements after the If loop are executed.

Syntax

Following is the syntax of an **If** statement in VBScript.

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

Flow Diagram



Example

For demo purpose, let us find the biggest between the two numbers of an Excel with the help of a function.

```
Private Sub if_demo_Click()  
    Dim x As Integer  
    Dim y As Integer  
  
    x = 234  
    y = 32  
  
    If x > y Then  
        MsgBox "X is Greater than Y"  
    End If  
End Sub
```

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

```
X is Greater than Y
```

If Else Statement

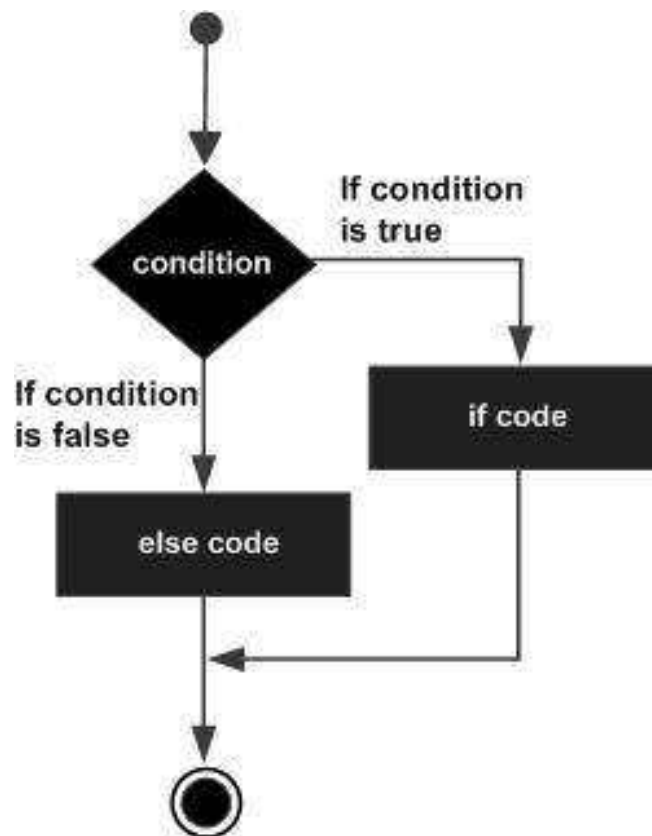
An If statement consists of a Boolean expression followed by one or more statements. If the condition is said to be True, the statements under If condition(s) are executed. If the condition is said to be False, the statements under Else Part is executed.

Syntax

Following is the syntax of an **If** Else statement in VBScript.

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

Flow Diagram



Example

For demo purpose, let us find the biggest between the two numbers of an Excel with the help of a function.

```
Private Sub if_demo_Click()  
    Dim x As Integer  
    Dim y As Integer  
  
    x = 234  
    y = 324  
  
    If x > y Then  
        MsgBox "X is Greater than Y"  
    Else  
        MsgBox "Y is Greater than X"  
    End If  
End Sub
```


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

Y is Greater than X

If Elseif - Else statement

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

Syntax

Following is the syntax of an If Elseif - Else statement in VBScript.

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

ElseIf (boolean_expression) Then
    Statement 1
    .....
    ....
    Statement n

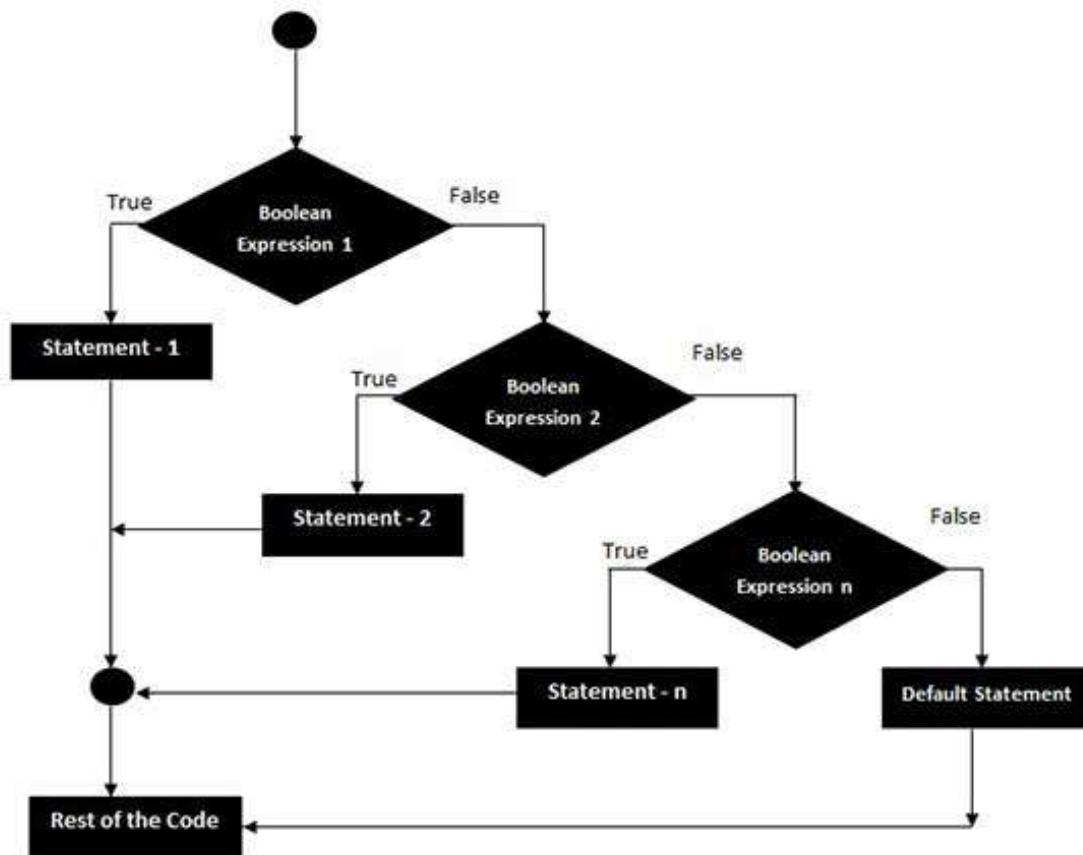
ElseIf (boolean_expression) Then
    Statement 1
    .....
    ....

    Statement n

Else
    Statement 1
    .....
    ....
    Statement n

End If
```

Flow Diagram



Example

For demo purpose, let us find the biggest between the two numbers of an Excel with the help of a function.

```

Private Sub if_demo_Click()
    Dim x As Integer
    Dim y As Integer

    x = 234
    y = 234

    If x > y Then
        MsgBox "X is Greater than Y"
    ElseIf y > x Then
        MsgBox "Y is Greater than X"
    Else

```

```

        MsgBox "X and Y are EQUAL"
    End If
End Sub

```

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

```
X and Y are EQUAL
```

Nested If Statement

An If or ElseIf statement inside another If or ElseIf statement(s). The inner If statements are executed based on the outermost If statements. This enables VBScript to handle complex conditions with ease.

Syntax

Following is the syntax of an Nested **If** statement in VBScript.

```

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

```

```

Else
    Statement 1
    .....
    ....
    Statement n
End If

```

Example

For demo purpose, let us find the type of a positive number with the help of a function.

```

Private Sub nested_if_demo_Click()
    Dim a As Integer
    a = 23

    If a > 0 Then
        MsgBox "The Number is a POSITIVE Number"
        If a = 1 Then
            MsgBox "The Number is Neither Prime NOR Composite"
        ElseIf a = 2 Then
            MsgBox "The Number is the Only Even Prime Number"
        ElseIf a = 3 Then
            MsgBox "The Number is the Least Odd Prime Number"
        Else
            MsgBox "The Number is NOT 0,1,2 or 3"
        End If
    ElseIf a < 0 Then
        MsgBox "The Number is a NEGATIVE Number"
    Else
        MsgBox "The Number is ZERO"
    End If
End Sub

```

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

```

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

```

Switch Statement

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

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

Syntax

Following is the syntax of a Switch statement in VBScript.

```
Select Case expression
    Case expressionlist1
        statement1
        statement2
        ....
        ....
        statement1n
    Case expressionlist2
        statement1
        statement2
        ....
        ....
    Case expressionlistn
        statement1
        statement2
        ....
        ....
    Case Else
        elsestatement1
        elsestatement2
        ....
        ....
End Select
```

Example

For demo purpose, let us find the type of integer with the help of a function.

```
Private Sub switch_demo_Click()  
    Dim MyVar As Integer  
    MyVar = 1  
  
    Select Case MyVar  
        Case 1  
            MsgBox "The Number is the Least Composite Number"  
        Case 2  
            MsgBox "The Number is the only Even Prime Number"  
        Case 3  
            MsgBox "The Number is the Least Odd Prime Number"  
        Case Else  
            MsgBox "Unknown Number"  
    End Select  
End Sub
```

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

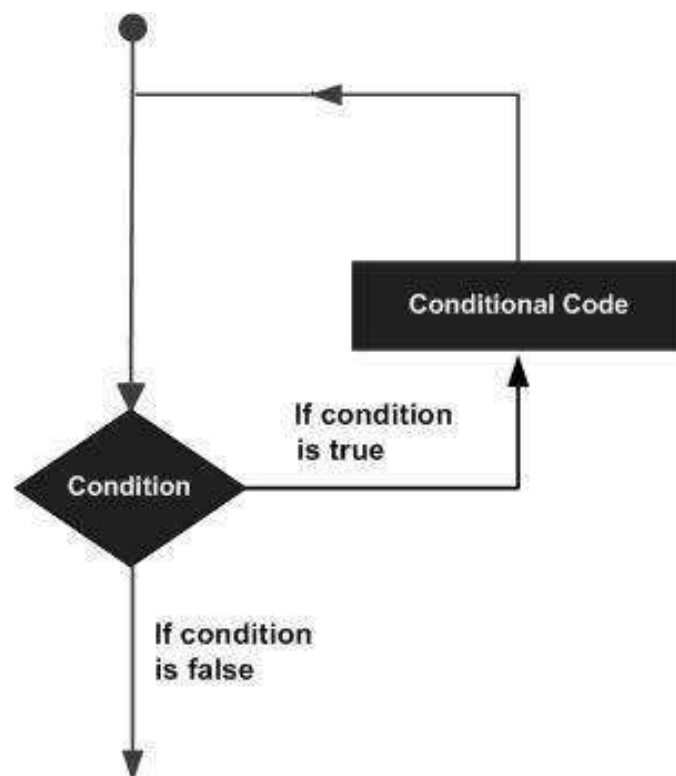
The Number is the Least Composite Number

11. VBA – Loops

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

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

A loop statement allows us to execute a statement or group of statements multiple times. Following is the general form of a loop statement in VBA.



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

Loop Type	Description
<u>for loop</u>	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
<u>for ..each loop</u>	This is executed if there is at least one element in the group and reiterated for each element in a group.
<u>while..wend loop</u>	This tests the condition before executing the loop body.

<u>do..while loops</u>	The do..While statements will be executed as long as the condition is True.(i.e.,) The Loop should be repeated till the condition is False.
<u>do..until loops</u>	The do..Until statements will be executed as long as the condition is False.(i.e.,) The Loop should be repeated till the condition is True.

For Loop

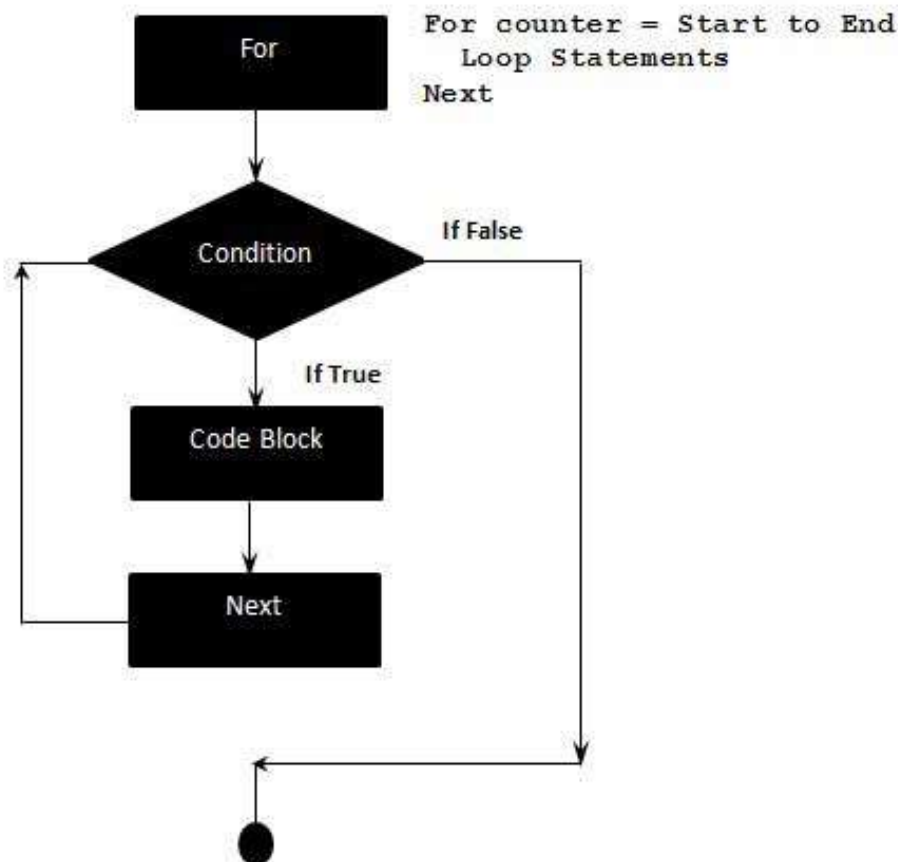
A **for** loop is a repetition control structure that allows a developer to efficiently write a loop that needs to be executed a specific number of times.

Syntax

Following is the syntax of a **for** loop in VBA.

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


Flow Diagram



Following is the flow of control in a For Loop:

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

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    Dim a As Integer  
    a = 10  
    For i = 0 To a Step 2  
        MsgBox "The value is i is : " & i  
    Next  
End Sub
```

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

```
The value is i is : 0  
  
The value is i is : 2  
  
The value is i is : 4  
  
The value is i is : 6  
  
The value is i is : 8  
  
The value is i is : 10
```

For Each Loops

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

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

Syntax

Following is the syntax of a **For Each** loop in VBA.

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

Example

```
Private Sub Constant_demo_Click()
    'fruits is an array
    fruits = Array("apple", "orange", "cherries")
    Dim fruitnames As Variant

    'iterating using For each loop.
    For Each Item In fruits
        fruitnames = fruitnames & Item & Chr(10)
    Next

    MsgBox fruitnames
End Sub
```

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

```
apple
orange
cherries
```

While Wend Loops

In a **While...Wend** loop, if the condition is True, all the statements are executed until the **Wend** keyword is encountered.

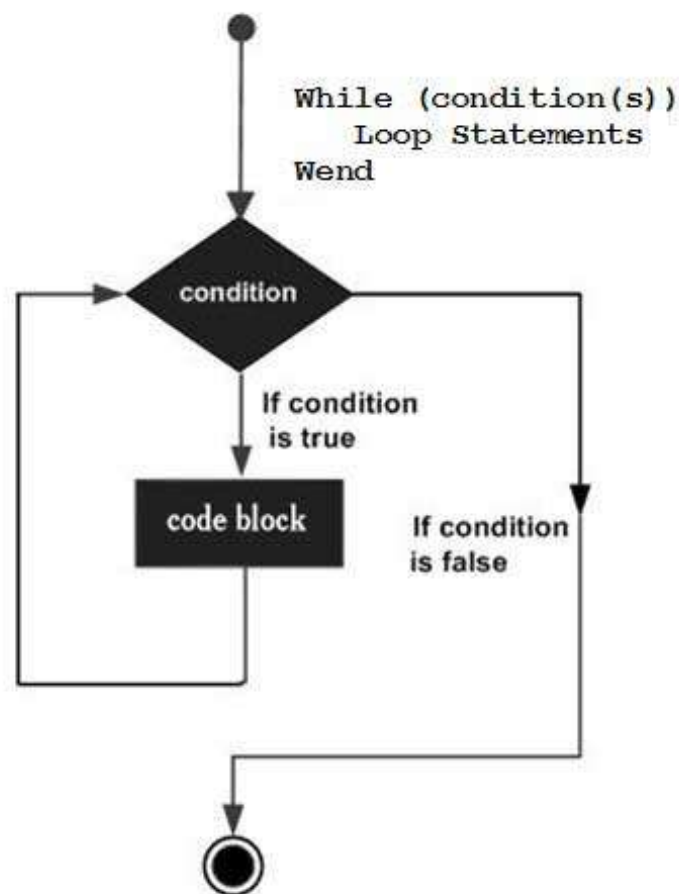
If the condition is false, the loop is exited and the control jumps to the very next statement after the **Wend** keyword.

Syntax

Following is the syntax of a **While..Wend** loop in VBA.

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

Flow Diagram



Example

```
Private Sub Constant_demo_Click()  
    Dim Counter : Counter = 10  
    While Counter < 15      ' Test value of Counter.  
        Counter = Counter + 1  ' Increment Counter.  
        msgbox "The Current Value of the Counter is : " & Counter  
    Wend  ' While loop exits if Counter Value becomes 15.  
End Sub
```

When the above code is executed, it prints the following in a message box.

```
The Current Value of the Counter is : 11  
  
The Current Value of the Counter is : 12  
  
The Current Value of the Counter is : 13  
  
The Current Value of the Counter is : 14  
  
The Current Value of the Counter is : 15
```

Do While Loops

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

Syntax

Following is the syntax of a **Do...While** loop in VBA.

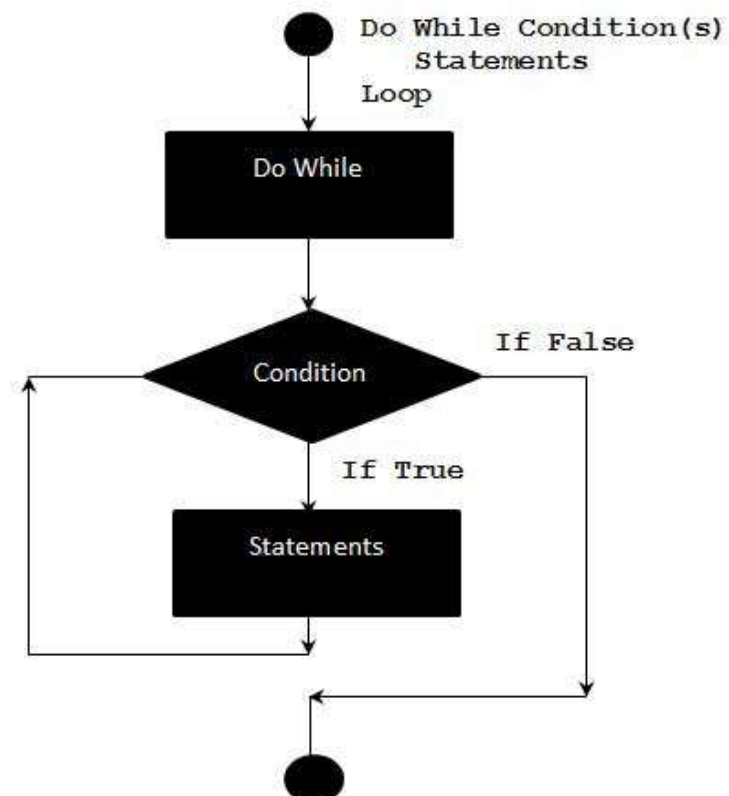
```
Do While condition  
    [statement 1]  
    [statement 2]  
    ...  
    [statement n]  
  
[Exit Do]
```

```

[statement 1]
[statement 2]
...
[statement n]
Loop

```

Flow Diagram



Example

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

```

Private Sub Constant_demo_Click()
    Do While i < 5
        i = i + 1
        msgbox "The value of i is : " & i
    Loop
End Sub

```

When the above code is executed, it prints the following output in a message box.

```
The value of i is : 1

The value of i is : 2

The value of i is : 3

The value of i is : 4

The value of i is : 5
```

Alternate Syntax

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

```
Do
    [statement 1]
    [statement 2]
    ...
    [statement n]
    [Exit Do]
    [statement 1]
    [statement 2]
    ...

    [statement n]
Loop While condition
```

Example

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

```
Private Sub Constant_demo_Click()
    i = 10
    Do
        i = i + 1
```

```
MsgBox "The value of i is : " & i  
Loop While i < 3 'Condition is false.Hence loop is executed once.  
End Sub
```

When the above code is executed, it prints the following output in a message box.

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

Do Until Loops

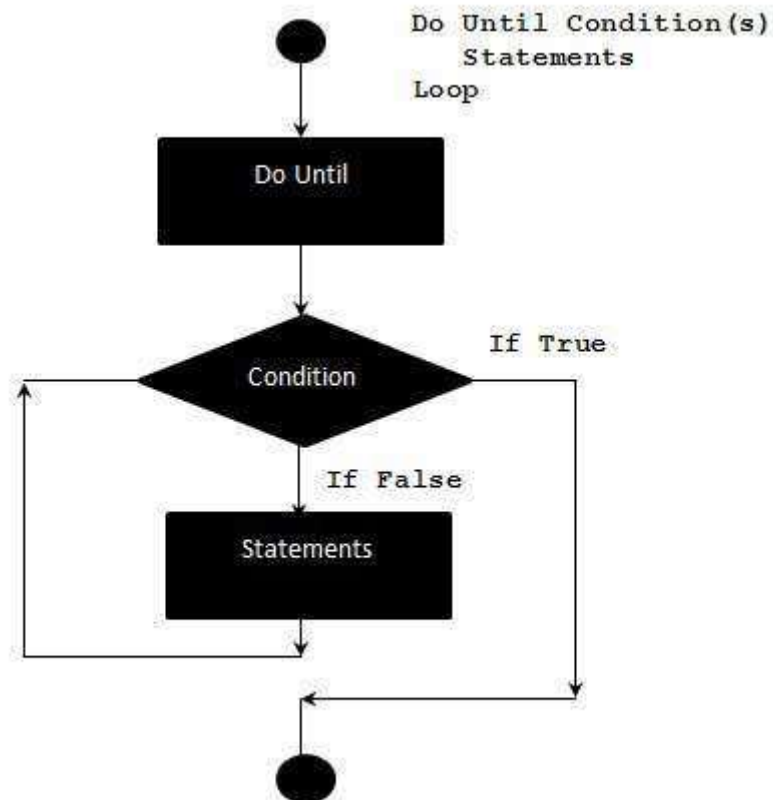
A **Do...Until** loop is used when we want to repeat a set of statements as long as the condition is false. The condition may be checked at the beginning of the loop or at the end of loop.

Syntax

Following is the syntax of a **Do..Until** loop in VBA.

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

Flow Diagram



Example

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

```

Private Sub Constant_demo_Click()
    i=10
    Do Until i>15 'Condition is False.Hence loop will be executed
        i = i + 1

        msgbox ("The value of i is : " & i)
    Loop
End Sub
  
```

When the above code is executed, it prints the following output in a message box.

The value of i is : 11

The value of i is : 12

The value of i is : 13

The value of i is : 14

The value of i is : 15

The value of i is : 16

Alternate Syntax

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

Do

[statement 1]

[statement 2]

...

[statement n]

[Exit Do]

[statement 1]

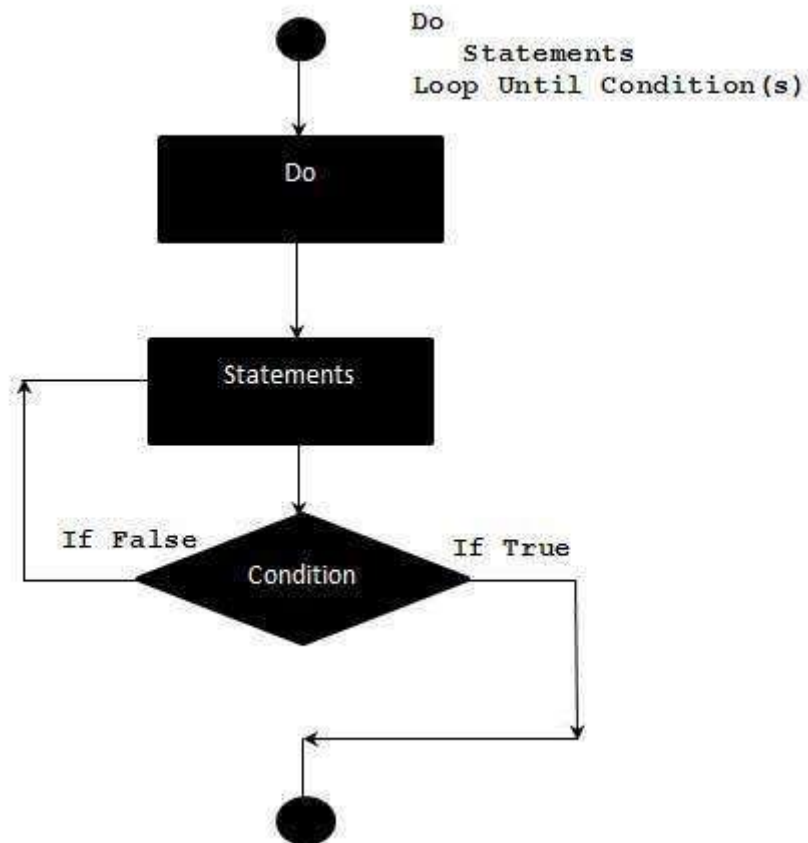
[statement 2]

...

[statement n]

Loop Until condition

Flow Diagram



Example

The following example uses **Do...Until** loop to check the condition at the end of the loop. The statements inside the loop are executed at least once, even if the condition is True.

```

Private Sub Constant_demo_Click()
    i=10
    Do
        i = i + 1
        msgbox "The value of i is : " & i
    Loop Until i<15 'Condition is True.Hence loop is executed once.
End Sub
  
```

When the above code is executed, it prints the following output in a message box.

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

Loop Control Statements

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

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

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

Exit For

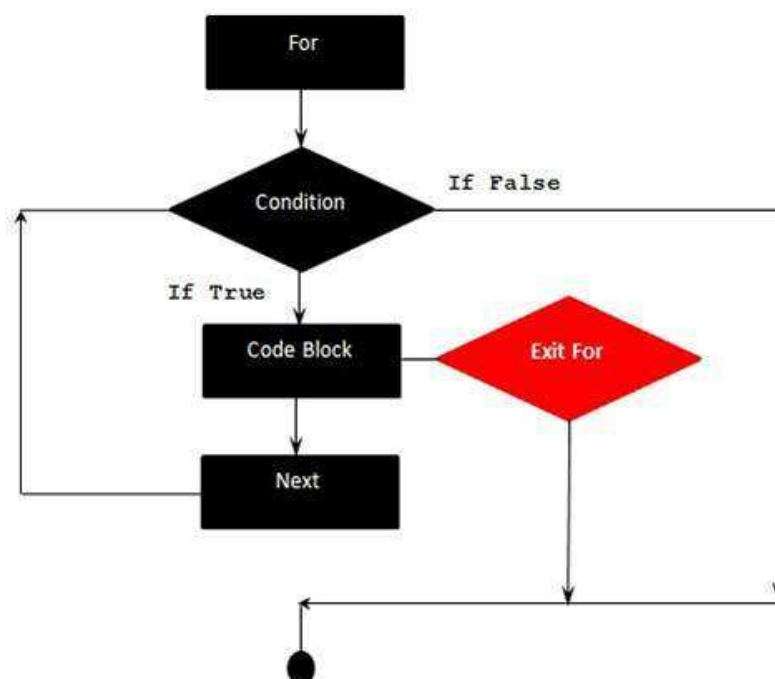
A **Exit For** statement is used when we want to exit the **For** Loop based on certain criteria. When **Exit For** is executed, the control jumps to the next statement immediately after the **For** Loop.

Syntax

Following is the syntax for **Exit For** Statement in VBA.

```
Exit For
```

Flow Diagram



Example

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

```
Private Sub Constant_demo_Click()  
    Dim a As Integer  
    a = 10  
    For i = 0 To a Step 2 'i is the counter variable and it is incremented by 2  
        MsgBox ("The value is i is : " & i)  
        If i = 4 Then  
            i = i * 10 'This is executed only if i=4  
            MsgBox ("The value is i is : " & i)  
            Exit For 'Exited when i=4  
        End If  
    Next  
End Sub
```

When the above code is executed, it prints the following output in a message Box.

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

Exit Do

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

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

Syntax

Following is the syntax for **Exit Do** Statement in VBA.

```
Exit Do
```

Example

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

```
Private Sub Constant_demo_Click()  
  
i = 0  
Do While i <= 100  
    If i > 10 Then  
        Exit Do    ' Loop Exits if i>10  
    End If  
    MsgBox ("The Value of i is : " & i)  
    i = i + 2  
Loop  
End Sub
```

When the above code is executed, it prints the following output in a message box.

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

12. VBA – Strings

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

Syntax

```
variablename = "string"
```

Examples

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

String Functions

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

Function Name	Description
<u>InStr</u>	Returns the first occurrence of the specified substring. Search happens from the left to the right.
<u>InstrRev</u>	Returns the first occurrence of the specified substring. Search happens from the right to the left.
<u>LCase</u>	Returns the lower case of the specified string.
<u>UCase</u>	Returns the upper case of the specified string.
<u>Left</u>	Returns a specific number of characters from the left side of the string.
<u>Right</u>	Returns a specific number of characters from the right side of the string.
<u>Mid</u>	Returns a specific number of characters from a string based on the specified parameters.
<u>Ltrim</u>	Returns a string after removing the spaces on the left side of the specified string.

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

Instr

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

Syntax

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

Parameter Description

- **Start** - An optional parameter. Specifies the starting position for the search. The search begins at the first position from the left to the right.
- **String1** - A required parameter. String to be searched.
- **String2** - A required parameter. String against which String1 is searched.
- **Compare** - An optional parameter. Specifies the string comparison to be used. It can take the following mentioned values:
 - 0 = vbBinaryCompare - Performs Binary Comparison (Default)
 - 1 = vbTextCompare - Performs Text Comparison

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    Dim Var As Variant  
    Var = "Microsoft VBScript"  
    MsgBox ("Line 1 : " & InStr(1, Var, "s"))  
    MsgBox ("Line 2 : " & InStr(7, Var, "s"))  
    MsgBox ("Line 3 : " & InStr(1, Var, "f", 1))  
    MsgBox ("Line 4 : " & InStr(1, Var, "t", 0))  
    MsgBox ("Line 5 : " & InStr(1, Var, "i"))  
    MsgBox ("Line 6 : " & InStr(7, Var, "i"))  
    MsgBox ("Line 7 : " & InStr(Var, "VB"))  
End Sub
```

When you execute the above function, it produces the following output.

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

InString Reverse

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

Syntax

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

Parameter Description

- **String1** - A required parameter. String to be searched.
- **String2** - A required parameter. String against which String1 is searched.
- **Start** - An optional parameter. Specifies the starting position for the search. The search begins at the first position from the right to the left.
- **Compare** - An optional parameter. Specifies the string comparison to be used. It can take the following mentioned values:
 - 0 = vbBinaryCompare - Performs Binary Comparison (Default)
 - 1 = vbTextCompare - Performs Text Comparison

Example

Add a button and place the following function.

```
Private Sub Constant_demo_Click()
    var="Microsoft VBScript"
    msgbox("Line 1 : " & InStrRev(var,"s",10))
    msgbox("Line 2 : " & InStrRev(var,"s",7))
    msgbox("Line 3 : " & InStrRev(var,"f",-1,1))
    msgbox("Line 4 : " & InStrRev(var,"t",5))
    msgbox("Line 5 : " & InStrRev(var,"i",7))
    msgbox("Line 6 : " & InStrRev(var,"i",7))
    msgbox("Line 7 : " & InStrRev(var,"VB",1))
End Sub
```

Upon executing the above script, it produces the following result.

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

LCase

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

Syntax

```
LCase(String)
```

Example

Add a button and place the following function inside the same.

```
Private Sub Constant_demo_Click()  
  
    var="Microsoft VBScript"  
    msgbox("Line 1 : " & LCase(var))  
    var="MS VBSCRIPT"  
    msgbox("Line 2 : " & LCase(var))  
    var="microsoft"  
    msgbox("Line 3 : " & LCase(var))  
  
End Sub
```

Upon executing the above script, it produces the following output.

```
Line 1 : microsoft vbscript  
Line 2 : ms vbscript  
Line 3 : microsoft
```

UCase

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

Syntax

```
UCase(String)
```

Example

Add a button and place the following function inside the same.

```
Private Sub Constant_demo_Click()
    var="Microsoft VBScript"
    msgbox("Line 1 : " & UCase(var))
    var="MS VBSCRIPT"
    msgbox("Line 2 : " & UCase(var))
    var="microsoft"
    msgbox("Line 3 : " & UCase(var))
End Sub
```

Upon executing the above script, it produces the following output.

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

Left

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

Syntax

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

Parameter Description

- **String** - A required parameter. Input String from which the specified number of characters to be returned from the left side.
- **Length** - A required parameter. An Integer, which specifies the number of characters to be returned.

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    Dim var as Variant
    var="Microsoft VBScript"
    msgbox("Line 1 : " & Left(var,2))
```

```

var="MS VBSCRIPT"
msgbox("Line 2 : " & Left(var,5))
var="microsoft"
msgbox("Line 3 : " & Left(var,9))
End Sub

```

When you execute the above function, it produces the following output.

```

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

```

Right

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

Syntax

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

Parameter Description

- **String** - A required parameter. Input String from which the specified number of characters to be returned from the right side.
- **Length** - A required parameter. An Integer, which Specifies the number of characters to be returned.

Example

Add a button and add the following function.

```

Private Sub Constant_demo_Click()
var="Microsoft VBScript"
msgbox("Line 1 : " & Right(var,2))
var="MS VBSCRIPT"
msgbox("Line 2 : " & Right(var,5))
var="microsoft"
msgbox("Line 3 : " & Right(var,9))
End Sub

```

When you execute the above function, it produces the following output.

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

Mid

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

Syntax

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

Parameter Description

- **String** - A required parameter. Input String from which the specified number of characters to be returned.
- **Start** - A required parameter. An Integer, which specifies the starting position of the string.
- **Length** - An optional parameter. An Integer, which specifies the number of characters to be returned.

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    Dim var as Variant
    var="Microsoft VBScript"
    msgbox("Line 1 : " & Mid(var,2))
    msgbox("Line 2 : " & Mid(var,2,5))
    msgbox("Line 3 : " & Mid(var,5,7))
End Sub
```

When you execute the above function, it produces the following output.

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

Ltrim

The Ltrim function removes the blank spaces from the left side of the string.

Syntax

```
LTrim(String)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    Dim var as Variant  
    var = "      Microsoft VBScript"  
    msgbox "After Ltrim : " & LTrim(var)  
End Sub
```

When you execute the function, it produces the following output.

```
After Ltrim : Microsoft VBScript
```

Rtrim

The Rtrim function removes the blank spaces from the right side of the string.

Syntax

```
RTrim(String)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    Dim var as Variant  
    var = "Microsoft VBScript      "  
    msgbox("After Rtrim : " & RTrim(var))  
End Sub
```

When you execute the above function, it produces the following output.

```
After Rtrim : Microsoft VBScript
```

Trim

The Trim function removes both the leading and the trailing blank spaces of the given input string.

Syntax

```
Trim(String)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    var="Microsoft VBScript"  
    var =      "      Microsoft VBScript      "  
    msgbox ("After Trim : " & Trim(var))  
End Sub
```

When you execute the above function, it produces the following output.

```
After trim : Microsoft VBScript
```

Len

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

Syntax

```
Len(String)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    Dim var1 as Variant  
    Dim var2 as Variant  
    var1 ="Microsoft VBScript"  
    msgbox("Length of var1 is : " & Len(var1))  
    var2 =      "      Microsoft VBScript      "  
    msgbox ("Length of var2 is : " & Len(var2))  
End Sub
```


When you execute the above function, it produces the following output.

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

Replace

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

Syntax

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

Parameter Description

- **String** - A required parameter. The Input String which is to be searched for replacing.
- **Find** - A required parameter. The part of the string that will be replaced.
- **Replacewith** - A required parameter. The replacement string, which would be replaced against the find parameter.
- **Start** - An optional parameter. Specifies the start position from where the string has to be searched and replaced. Default value is 1.
- **Count** - An optional parameter. Specifies the number of times the replacement has to be performed.
- **Compare** - An optional parameter. Specifies the comparison method to be used. Default value is 0.
 - 0 = vbBinaryCompare - Performs a binary comparison
 - 1 = vbTextCompare - Performs a Textual comparison

Example

```
Private Sub Constant_demo_Click()
    Dim var as Variant
    var="This is VBScript Programming"

    'VBScript to be replaced by MS VBScript
    msgbox("Line 1: " & Replace(var,"VBScript","MS VBScript"))
```

```

'VB to be replaced by vb
msgbox("Line 2: " & Replace(var,"VB","vb"))

''is' replaced by ##
msgbox("Line 3: " & Replace(var,"is","##"))

''is' replaced by ## ignores the characters before the first occurrence
msgbox("Line 4: " & Replace(var,"is","##",5))

''s' is replaced by ## for the next 2 occurrences.
msgbox("Line 5: " & Replace(var,"s","##",1,2))

''r' is replaced by ## for all occurrences textual comparison.
msgbox("Line 6: " & Replace(var,"r","##",1,-1,1))

''t' is replaced by ## for all occurrences Binary comparison
msgbox("Line 7: " & Replace(var,"t","##",1,-1,0))

End Sub

```

When you execute the above function, it produces the following output.

```

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

```

Space

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

Syntax

```
space(number)
```

Parameter Description

- **Number** - A required parameter. The number of spaces that we want to add to the given string.

Example

```
Private Sub Constant_demo_Click()
    Dim var1 as Variant
    var1="Microsoft"

    Dim var2 as Variant
    var2="VBScript"
    msgbox(var1 & Space(2)& var2)
End Sub
```

When you execute the above function, it produces the following output.

```
Microsoft VBScript
```

StrComp

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

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

Syntax

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

Parameter Description

- **String1** – A required parameter. The first string expression.
- **String2** - A required parameter. The second string expression.
- **Compare** - An optional parameter. Specifies the string comparison to be used. It can take the following values:

- 0 = vbBinaryCompare - Performs Binary Comparison(Default)
- 1 = vbTextCompare - Performs Text Comparison

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    Dim var1 as Variant
    MsgBox("Line 1 :" & StrComp("Microsoft","Microsoft"))
    MsgBox("Line 2 :" & StrComp("Microsoft","MICROSOFT"))
    MsgBox("Line 3 :" & StrComp("Microsoft","MiCrOsOfT"))
    MsgBox("Line 4 :" & StrComp("Microsoft","MiCrOsOfT",1))
    MsgBox("Line 5 :" & StrComp("Microsoft","MiCrOsOfT",0))
End Sub
```

When you execute the above function, it produces the following output.

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

String Function

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

Syntax

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

Parameter Description

- **Number** - A required parameter. An integer value, which would be repeated for a specified number of times against the character parameter.
- **Character** - A required parameter. Character value, which has to be repeated for a specified number of times.

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    msgbox("Line 1 :" & String(3,"$"))
    msgbox("Line 2 :" & String(4,"*"))
    msgbox("Line 3 :" & String(5,100))
    msgbox("Line 4 :" & String(6,"ABCDE"))
End Sub
```

When you execute the above function, it produces the following output.

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

String Reverse Function

The StrReverse function reverses the specified string.

Syntax

```
StrReverse(string)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    msgbox("Line 1 : " & StrReverse("VBSCRIPT"))
    msgbox("Line 2 : " & StrReverse("My First VBScript"))
    msgbox("Line 3 : " & StrReverse("123.45"))
End Sub
```

When you execute the above function, it produces the following output.

```
Line 1 : TPIRCBV
Line 2 : tpircSBV tsriF yM
Line 3 : 54.321
```

13. VBA – Date-Time Function

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

Date Functions

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

Date Function

Date

The Function returns the current system date.

Syntax

```
date()
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    Dim a as Variant  
    a = date()  
    msgbox "The Value of a : " & a  
End Sub
```

When you execute the function, it produces the following output.

```
The Value of a : 19/07/2014
```

CDate Function

CDate

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

Syntax

```
cdate(date)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    Dim a as Variant  
    Dim b as Variant  
    a = cdate("Jan 01 2020")  
    msgbox("The Value of a : " & a)  
    b = cdate("31 Dec 2050")
```

```
msgbox("The Value of b : " & b)  
End Sub
```

When you execute the function, it produces the following output.

```
The Value of a : 1/01/2020  
The Value of b : 31/12/2050
```

DateAdd Function

DateAdd

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

Syntax

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

Parameter Description

- **Interval** - A required parameter. It can take the following values:
 - d - day of the year
 - m - month of the year
 - y - year of the year
 - yyyy - year
 - w - weekday
 - ww - week
 - q - quarter
 - h - hour
 - m - minute
 - s - second
- **Number** - A required parameter. It can take both positive and negative parameters.
- **Date** - A required parameter. A variant or literal representing the date to which an interval is added.

Example


```

Private Sub Constant_demo_Click()
    ' Positive Interval
    date1=01-Jan-2013
    msgbox("Line 1 : " &DateAdd("yyyy",1,date1))
    msgbox("Line 2 : " &DateAdd("q",1,date1))
    msgbox("Line 3 : " &DateAdd("m",1,date1))
    msgbox("Line 4 : " &DateAdd("y",1,date1))
    msgbox("Line 5 : " &DateAdd("d",1,date1))
    msgbox("Line 6 : " &DateAdd("w",1,date1))
    msgbox("Line 7 : " &DateAdd("ww",1,date1))
    msgbox("Line 8 : " &DateAdd("h",1,"01-Jan-2013 12:00:00"))
    msgbox("Line 9 : " &DateAdd("n",1,"01-Jan-2013 12:00:00"))
    msgbox("Line 10 : " &DateAdd("s",1,"01-Jan-2013 12:00:00"))

    ' Negative Interval
    msgbox("Line 11 : " &DateAdd("yyyy",-1,date1))
    msgbox("Line 12 : " &DateAdd("q",-1,date1))
    msgbox("Line 13 : " &DateAdd("m",-1,date1))
    msgbox("Line 14 : " &DateAdd("y",-1,date1))
    msgbox("Line 15 : " &DateAdd("d",-1,date1))
    msgbox("Line 16 : " &DateAdd("w",-1,date1))
    msgbox("Line 17 : " &DateAdd("ww",-1,date1))
    msgbox("Line 18 : " &DateAdd("h",-1,"01-Jan-2013 12:00:00"))
    msgbox("Line 19 : " &DateAdd("n",-1,"01-Jan-2013 12:00:00"))
    msgbox("Line 20 : " &DateAdd("s",-1,"01-Jan-2013 12:00:00"))
End Sub

```

When you execute the above function, it produces the following output.

```

Line 1 : 27/06/1895
Line 2 : 27/09/1894
Line 3 : 27/07/1894
Line 4 : 28/06/1894
Line 5 : 28/06/1894
Line 6 : 28/06/1894
Line 7 : 4/07/1894

```

```

Line 8 : 1/01/2013 1:00:00 PM
Line 9 : 1/01/2013 12:01:00 PM
Line 10 : 1/01/2013 12:00:01 PM
Line 11 : 27/06/1893
Line 12 : 27/03/1894
Line 13 : 27/05/1894
Line 14 : 26/06/1894
Line 15 : 26/06/1894
Line 16 : 26/06/1894
Line 17 : 20/06/1894
Line 18 : 1/01/2013 11:00:00 AM
Line 19 : 1/01/2013 11:59:00 AM
Line 20 : 1/01/2013 11:59:59 AM

```

DateDiff Function

DateDiff

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

Syntax

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

Parameter Description

- **Interval** - A required parameter. It can take the following values:
 - d - day of the year
 - m - month of the year
 - y - year of the year
 - yyyy - year
 - w - weekday
 - ww - week
 - q - quarter
 - h - hour
 - m - minute
 - s - second

- **Date1** and **Date2** - Required parameters.
- **Firstdayofweek** – An optional parameter. Specifies the first day of the week. It can take the following values:
 - 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting
 - 1 = vbSunday - Sunday
 - 2 = vbMonday - Monday
 - 3 = vbTuesday - Tuesday
 - 4 = vbWednesday - Wednesday
 - 5 = vbThursday - Thursday
 - 6 = vbFriday - Friday
 - 7 = vbSaturday - Saturday
- **Firstdayofyear** – An optional parameter. Specifies the first day of the year. It can take the following values:
 - 0 = vbUseSystem - Use National Language Support (NLS) API setting
 - 1 = vbFirstJan1 - Start with the week in which January 1 occurs (default)
 - 2 = vbFirstFourDays - Start with the week that has at least four days in the new year
 - 3 = vbFirstFullWeek - Start with the first full week of the new year

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    Dim fromDate as Variant
    fromDate="01-Jan-09 00:00:00"
    Dim toDate as Variant
    toDate="01-Jan-10 23:59:00"
    msgbox("Line 1 : " & DateDiff("yyyy",fromDate,toDate))
    msgbox("Line 2 : " & DateDiff("q",fromDate,toDate))
    msgbox("Line 3 : " & DateDiff("m",fromDate,toDate))
    msgbox("Line 4 : " & DateDiff("y",fromDate,toDate))
    msgbox("Line 5 : " & DateDiff("d",fromDate,toDate))
    msgbox("Line 6 : " & DateDiff("w",fromDate,toDate))
End Sub
```

```
msgbox("Line 7 : " & DateDiff("ww", fromDate, toDate))  
msgbox("Line 8 : " & DateDiff("h", fromDate, toDate))  
msgbox("Line 9 : " & DateDiff("n", fromDate, toDate))  
msgbox("Line 10 : " & DateDiff("s", fromDate, toDate))  
End Sub
```

When you execute the above function, it produces the following output.

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

DatePart Function

DatePart

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

Syntax

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

Parameter Description

- **Interval** - A required parameter. It can take the following values:
 - d - day of the year
 - m - month of the year
 - y - year of the year
 - yyyy - year
 - w - weekday
 - ww - week
 - q - quarter

- h - hour
- m - minute
- s - second
- **Date1** - A required parameter.
- **Firstdayofweek** – An optional parameter. Specifies the first day of the week. It can take the following values:
 - 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting
 - 1 = vbSunday - Sunday
 - 2 = vbMonday - Monday
 - 3 = vbTuesday - Tuesday
 - 4 = vbWednesday - Wednesday
 - 5 = vbThursday - Thursday
 - 6 = vbFriday - Friday
 - 7 = vbSaturday - Saturday
- **Firstdayofyear** – An optional parameter. Specifies the first day of the year. It can take the following values:
 - 0 = vbUseSystem - Use National Language Support (NLS) API setting
 - 1 = vbFirstJan1 - Start with the week in which January 1 occurs (default)
 - 2 = vbFirstFourDays - Start with the week that has at least four days in the new year
 - 3 = vbFirstFullWeek - Start with the first full week of the new year

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    Dim Quarter as Variant
    Dim DayOfYear as Variant
    Dim WeekOfYear as Variant

    Date1 = "2013-01-15"
    Quarter = DatePart("q", Date1)
    msgbox("Line 1 : " & Quarter)
```

```

DayOfYear = DatePart("y", Date1)
msgbox("Line 2 : " & DayOfYear)
WeekOfYear = DatePart("ww", Date1)
msgbox("Line 3 : " & WeekOfYear)
msgbox("Line 4 : " & DatePart("m",Date1))
End Sub

```

When you execute the above function, it produces the following output.

```

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

```

DateSerial Function

DateSerial

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

Syntax

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

Parameter Description

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

Example

Add a button and add the following function.

```

Private Sub Constant_demo_Click()
    msgbox(DateSerial(2013,5,10))
End Sub

```

When you execute the above function, it produces the following output.

10/05/2014

Format DateTime Function

Format DateTime

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

Syntax

FormatDateTime(date,format)

Parameter Description

- **Date** – A required parameter.
- **Format** – An optional parameter. The Value that specifies the date or time format to be used. It can take the following values:
 - 0 = vbGeneralDate - Default
 - 1 = vbLongDate - Returns date
 - 2 = vbShortDate - Returns date
 - 3 = vbLongTime - Returns time
 - 4 = vbShortTime - Returns time

Example

Add a button and add the following function.

<pre>Private Sub Constant_demo_Click() d="2013-08-15 20:25" msgbox("Line 1 : " & FormatDateTime(d)) msgbox("Line 2 : " & FormatDateTime(d,1)) msgbox("Line 3 : " & FormatDateTime(d,2)) msgbox("Line 4 : " & FormatDateTime(d,3)) msgbox("Line 5 : " & FormatDateTime(d,4)) End Sub</pre>

When you execute the above function, it produces the following output.

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

IsDate Function

IsDate

A Function, which returns a Boolean value whether or not the given input is a date.

Syntax

```
IsDate(expression)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    msgbox("Line 1 : " & IsDate("Nov 03, 1950"))
    msgbox("Line 2 : " & IsDate(#01/31/20#))
    msgbox("Line 3 : " & IsDate(#05/31/20 10:30 PM#))
End Sub
```

When you execute the above function, it produces the following output.

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

Day Function

Day Function

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

Syntax

```
Day(date)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox(Day("2013-06-30"))  
End Sub
```

When you execute the above function, it produces the following output.

```
30
```

Month Function

Month Function

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

Syntax

```
Month(date)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox(Month("2013-06-30"))  
End Sub
```

When you execute the above function, it produces the following output.

```
6
```

Year Function

Year

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

Syntax

```
Year(date)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox(Year("2013-06-30"))  
End sub
```

When you execute the above function, it produces the following output.

```
2013
```

Month Name

MonthName

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

Syntax

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

Parameter Description

- **Month** – A required parameter. It specifies the number of the month.
- **Toabbreviate** - An optional parameter. A Boolean value that indicates if the month name is to be abbreviated. If left blank, the default value would be taken as False.

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox("Line 1 : " & MonthName(01,True))  
    msgbox("Line 2 : " & MonthName(01,false))  
    msgbox("Line 3 : " & MonthName(07,True))  
    msgbox("Line 4 : " & MonthName(07,false))  
End Sub
```

When you execute the above function, it produces the following output.

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

WeekDay

WeekDay

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

Syntax

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

Parameter Description

- **Date** – A required parameter. The weekday will return a specified date.
- **Firstdayofweek** – An optional parameter. Specifies the first day of the week. It can take the following values.
 - 0 = vbUseSystemDayOfWeek - Use National Language Support (NLS) API setting
 - 1 = vbSunday - Sunday
 - 2 = vbMonday - Monday
 - 3 = vbTuesday - Tuesday
 - 4 = vbWednesday - Wednesday
 - 5 = vbThursday - Thursday
 - 6 = vbFriday - Friday
 - 7 = vbSaturday - Saturday

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    msgbox("Line 1: " & Weekday("2013-05-16",1))
    msgbox("Line 2: " & Weekday("2013-05-16",2))
    msgbox("Line 3: " & Weekday("2013-05-16",2))
End Sub
```

```

msgbox("Line 4: " & Weekday("2010-02-16"))
msgbox("Line 5: " & Weekday("2010-02-17"))
msgbox("Line 6: " & Weekday("2010-02-18"))
End Sub

```

When you execute the above function, it produces the following output.

```

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

```

WeekDay Name

WeekDay

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

Syntax

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

Parameter Description

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

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    MsgBox("Line 1 : " & WeekdayName(3))
    MsgBox("Line 2 : " & WeekdayName(2,True))
    MsgBox("Line 3 : " & WeekdayName(1,False))
    MsgBox("Line 4 : " & WeekdayName(2,True,0))
    MsgBox("Line 5 : " & WeekdayName(1,False,1))
End Sub
```

When you execute the above function, it produces the following output.

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

Time Functions

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

Now Function

Now Function

The Function Now returns the current system date and time.

Syntax

```
Now()
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    Dim a as Variant  
    a = Now()  
    msgbox("The Value of a : " & a)  
End Sub
```

When you execute the above function, it produces the following output.

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

Hour Function

Hour Function

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

Syntax

```
Hour(time)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox("Line 1: " & Hour("3:13:45 PM"))  
    msgbox("Line 2: " & Hour("23:13:45"))  
    msgbox("Line 3: " & Hour("2:20 PM"))  
End Sub
```

When you execute the above function, it produces the following output.

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

Minute Function

Minute Function

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

Syntax

```
Minute(time)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox("Line 1: " & Minute("3:13:45 PM"))  
    msgbox("Line 2: " & Minute("23:43:45"))  
    msgbox("Line 3: " & Minute("2:20 PM"))  
End Sub
```

When you execute the above function, it produces the following output.

```
Line 1: 13  
Line 2: 43  
Line 3: 20
```

Second Function

Second Function

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

Syntax

```
Second(time)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox("Line 1: " & Second("3:13:25 PM"))  
    msgbox("Line 2: " & Second("23:13:45"))  
    msgbox("Line 3: " & Second("2:20 PM"))  
End Sub
```

When you execute the above function, it produces the following output.

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

Time Function

Time Function

The Time Function returns the current system time.

Syntax

```
Time()
```

Example

```
Private Sub Constant_demo_Click()  
    msgbox("Line 1: " & Time())  
End Sub
```

When you execute the above function, it produces the following output.

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

Timer Function

Timer Function

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

Syntax

```
Timer()
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox("Time is : " & Now())  
    msgbox("Timer is: " & Timer())  
End Sub
```

When you execute the above function, it produces the following output.

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

Time Serial Function

Time Serial Function

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

Syntax

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

Parameter Description

- **Hour** - A required parameter, which is an integer between 0 and 23 or any numeric expression.
- **Minute** - A required parameter, which is an integer between 0 and 59 or any numeric expression.
- **Second** - A required parameter, which is an integer between 0 and 59 or any numeric expression.

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox(TimeSerial(20,1,2))
```

```
msgbox(TimeSerial(0,59,59))  
msgbox(TimeSerial(7*2,60/3,15+3))  
End Sub
```

When you execute the above function, it produces the following output.

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

TimeValue Function

TimeValue Function

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

Syntax

```
TimeValue(StringTime)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    msgbox(TimeValue("20:30"))  
    msgbox(TimeValue("5:15"))  
    msgbox(TimeValue("2:30:58"))  
End Sub
```

When you execute the above function, it produces the following output.

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

14. VBA – Arrays

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

Array Declaration

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

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

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

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

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

Assigning Values to an Array

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

Example

Add a button and add the following function.

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

    MsgBox("Value stored in Array index 0 : " & arr(0))
    MsgBox("Value stored in Array index 1 : " & arr(1))
    MsgBox("Value stored in Array index 2 : " & arr(2))
    MsgBox("Value stored in Array index 3 : " & arr(3))
    MsgBox("Value stored in Array index 4 : " & arr(4))
    MsgBox("Value stored in Array index 5 : " & arr(5))
End Sub
```

When you execute the above function, it produces the following output.

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

Multi-Dimensional Arrays

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

Example

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

```
Private Sub Constant_demo_Click()
```

```

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

msgbox("Value in Array index 0,1 : " & arr(0,1))
msgbox("Value in Array index 2,2 : " & arr(2,2))

End Sub

```

When you execute the above function, it produces the following output.

```

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

```

ReDim Statement

ReDim statement is used to declare dynamic-array variables and allocate or reallocate storage space.

Syntax

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

Parameter Description

- **Preserve** - An optional parameter used to preserve the data in an existing array when you change the size of the last dimension.
- **Varname** - A required parameter, which denotes the name of the variable, which should follow the standard variable naming conventions.

- **Subscripts** - A required parameter, which indicates the size of the array.

Example

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

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

```
Private Sub Constant_demo_Click()  
    Dim a() as variant  
    i=0  
    redim a(5)  
    a(0)="XYZ"  
    a(1)=41.25  
    a(2)=22  
  
    REDIM PRESERVE a(7)  
    For i=3 to 7  
        a(i)= i  
    Next  
  
    'to Fetch the output  
    For i=0 to ubound(a)  
        MsgBox a(i)  
    Next  
End Sub
```

When you execute the above function, it produces the following output.

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

Array Methods

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

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

LBound Function

LBound Function

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

Syntax

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

Parameter Description

- **ArrayName** – A required parameter. This parameter corresponds to the name of the array.
- **Dimension** - An optional parameter. This takes an integer value that corresponds to the dimension of the array. If it is '1', then it returns the lower bound of the first dimension; if it is '2', then it returns the lower bound of the second dimension and so on.

Example

Add a button and add the following function.

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

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

    ' For MultiDimension Arrays :
    Dim arr2(3,2) as Variant
    MsgBox("The smallest Subscript of the first dimension of arr2 is : " &
    LBound(arr2,1))

    MsgBox("The smallest Subscript of the Second dimension of arr2 is : " &
    LBound(arr2,2))

End Sub
```

When you execute the above function, it produces the following output.

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

UBound Function

UBound Function

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

Syntax

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


Parameter Description

- **ArrayName** - A required parameter. This parameter corresponds to the name of the array.
- **Dimension** - An optional parameter. This takes an integer value that corresponds to the dimension of the array. If it is '1', then it returns the lower bound of the first dimension; if it is '2', then it returns the lower bound of the second dimension, and so on.

Example

Add a button and add the following function.

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

    msgbox("The smallest Subscript value of the given array is : " &
    UBound(arr))

    ' For MultiDimension Arrays :
    Dim arr2(3,2) as Variant
    msgbox("The smallest Subscript of the first dimension of arr2 is : " &
    UBound(arr2,1))
    msgbox("The smallest Subscript of the Second dimension of arr2 is : " &
    UBound(arr2,2))

End Sub
```

When you execute the above function, it produces the following output.

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

Split Function

Split Function

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

Syntax

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

Parameter Description

- **Expression** - A required parameter. The string expression that can contain strings with delimiters.
- **Delimiter** - An optional parameter. The parameter, which is used to convert into arrays based on a delimiter.
- **Count** -An optional parameter. The number of substrings to be returned, and if specified as -1, then all the substrings are returned.
- **Compare** - An optional parameter. This parameter specifies which comparison method is to be used.
 - 0 = vbBinaryCompare - Performs a binary comparison
 - 1 = vbTextCompare - Performs a textual comparison

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    ' Splitting based on delimiter comma '$'
    Dim a as Variant
    Dim b as Variant
    a=Split("Red $ Blue $ Yellow","$")
    b=ubound(a)
    For i=0 to b
        msgbox("The value of array in " & i & " is :" & a(i))
    Next
End Sub
```

When you execute the above function, it produces the following output.

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

Join Function

Join Function

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

Syntax

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

Parameter Description

- **List** - A required parameter. An array that contains the substrings that are to be joined.
- **Delimiter** - An optional parameter. The character, which used as a delimiter while returning the string. The default delimiter is Space.

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()  
    ' Join using spaces  
    a = array("Red","Blue","Yellow")  
    b = join(a)  
    msgbox("The value of b " & " is : " & b)  
  
    ' Join using $  
    b = join(a,"$")  
    msgbox("The Join result after using delimiter is : " & b)  
End Sub
```

When you execute the above function, it produces the following output.

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

Filter Function

Filter

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

Syntax

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

Parameter Description

- **Inputstrings** - A required parameter. This parameter corresponds to the array of strings to be searched.
- **Value** - A required parameter. This parameter corresponds to the string to search for against the inputstrings parameter.
- **Include** - An optional parameter. This is a Boolean value, which indicates whether or not to return the substrings that include or exclude.
- **Compare** - An optional parameter. This parameter describes which string comparison method is to be used.
 - 0 = vbBinaryCompare - Performs a binary comparison
 - 1 = vbTextCompare - Performs a textual comparison

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    Dim a,b,c,d as Variant
    a = array("Red","Blue","Yellow")
    b = Filter(a,"B")
    c = Filter(a,"e")
    d = Filter(a,"Y")

    For each x in b
```

```
        msgbox("The Filter result 1: " & x)
    Next

    For each y in c
        msgbox("The Filter result 2: " & y)
    Next

    For each z in d
        msgbox("The Filter result 3: " & z)
    Next
End Sub
```

When you execute the above function, it produces the following output.

```
The Filter result 1: Blue
The Filter result 2: Red
The Filter result 2: Blue
The Filter result 2: Yellow
The Filter result 3: Yellow
```

IsArray Function

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

Syntax

```
IsArray(variablename)
```

Example

Add a button and add the following function.

```
Private Sub Constant_demo_Click()
    Dim a,b as Variant
    a = array("Red","Blue","Yellow")
    b = "12345"

    msgbox("The IsArray result 1 : " & IsArray(a))
    msgbox("The IsArray result 2 : " & IsArray(b))
End Sub
```

```
End Sub
```

When you execute the above function, it produces the following output.

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

Erase Function

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

Syntax

```
Erase ArrayName
```

- Fixed numeric array, each element in an array is reset to Zero.
- Fixed string array, each element in an array is reset to Zero length " ".
- Array of objects, each element in an array is reset to special value Nothing.

Example

Add a button and add the following function.

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

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

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

    ' All values would be erased.
    msgbox("The value at Zeroth index of NumArray is " & NumArray(0))
    msgbox("The value at First index of NumArray is " & NumArray(1))
End Sub
```

```
msgbox("The value at Second index of NumArray is " & NumArray(2))  
msgbox("The value at Third index of NumArray is " & NumArray(3))  
End Sub
```

When you execute the above function, it produces the following output.

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

15. VBA – User-Defined Functions

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

Apart from inbuilt functions, VBA allows to write user-defined functions as well. In this chapter, you will learn how to write your own functions in VBA.

Function Definition

A VBA function can have an optional return statement. This is required if you want to return a value from a function.

For example, you can pass two numbers in a function and then you can expect from the function to return their multiplication in your calling program.

Note: A function can return multiple values separated by a comma as an array assigned to the function name itself.

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

Syntax

Add a button and add the following function.

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

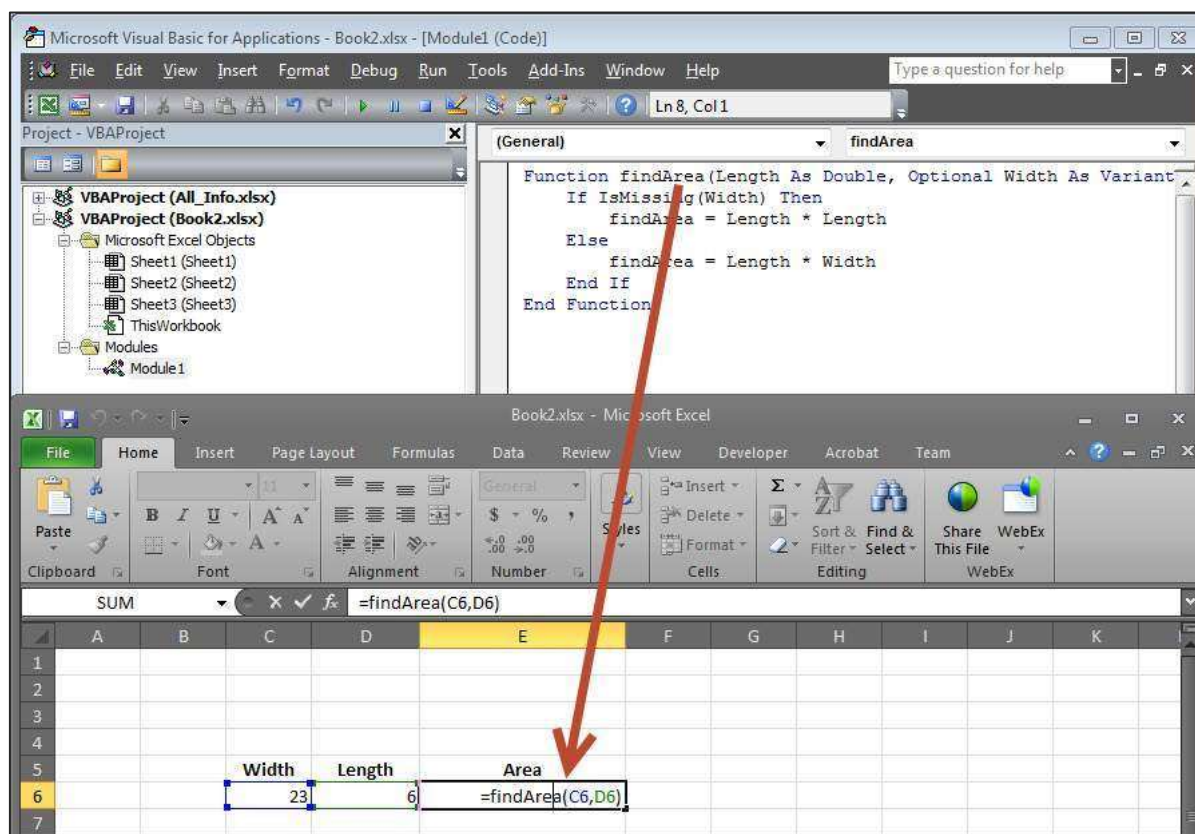

Example

Add the following function which returns the area. Note that a value/values can be returned with the function name itself.

```
Function findArea(Length As Double, Optional Width As Variant)
    If IsMissing(Width) Then
        findArea = Length * Length
    Else
        findArea = Length * Width
    End If
End Function
```

Calling a Function

To invoke a function, call the function using the function name as shown in the following screenshot.



The output of the area as shown below will be displayed to the user.

Width	Length	Area
23	6	138

16. VBA – Sub Procedure

Sub Procedures are similar to functions, however there are a few differences.

- Sub procedures DO NOT Return a value while functions may or may not return a value.
- Sub procedures CAN be called without a call keyword.
- Sub procedures are always enclosed within Sub and End Sub statements.

Example

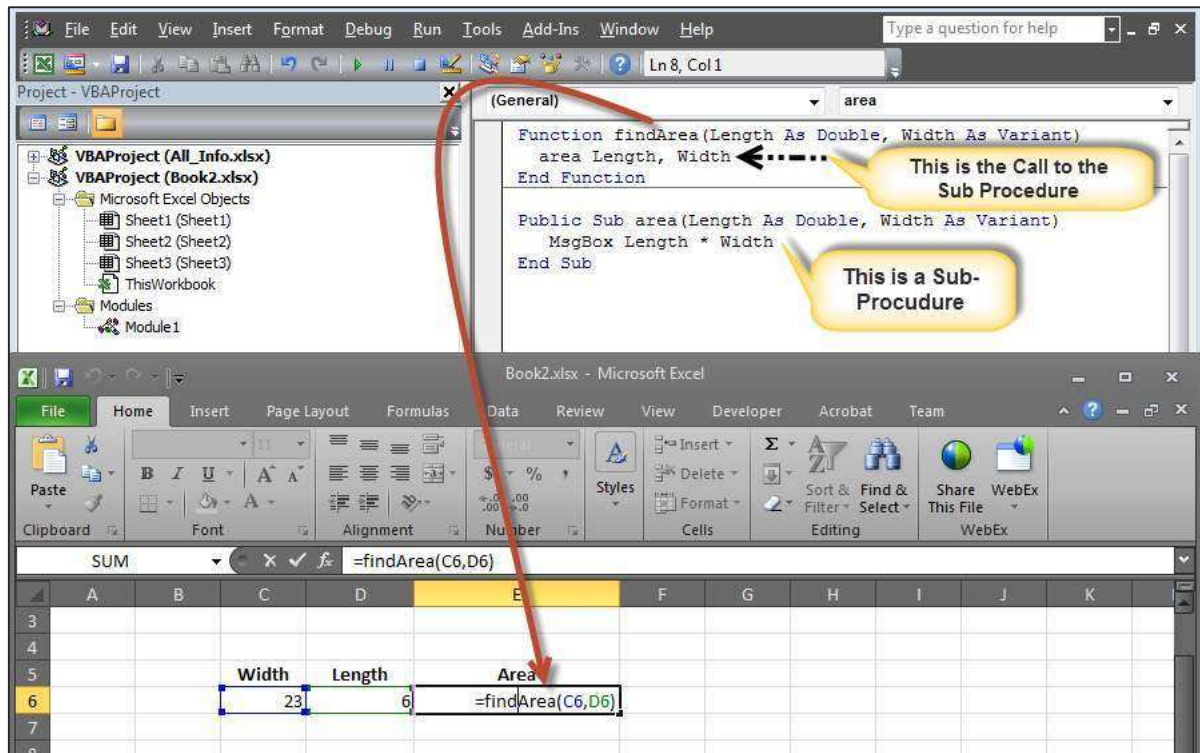
```
Sub Area(x As Double, y As Double)
    MsgBox x * y
End Sub
```

Calling Procedures

To invoke a Procedure somewhere in the script, you can make a call from a function. We will not be able to use the same way as that of a function as sub procedure WILL NOT return a value.

```
Function findArea(Length As Double, Width As Variant)
    area Length, Width    ' To Calculate Area 'area' sub proc is called
End Function
```

Now you will be able to call the function only but not the sub procedure as shown in the following screenshot.



The area is calculated and shown only in the Message box.



The result cell displays ZERO as the area value is NOT returned from the function. In short, you cannot make a direct call to a sub procedure from the excel worksheet.

	Width	Length	Area
	23	6	0

The Output is shown as ZERO as the sub procedure displays the message box and no value is returned from the function.

17. VBA – Events

VBA, an event-driven programming can be triggered when you change a cell or range of cell values manually. Change event may make things easier, but you can very quickly end a page full of formatting. There are two kinds of events.

- Worksheet Events
- Workbook Events

Worksheet Events

Worksheet Events are triggered when there is a change in the worksheet. It is created by performing a right-click on the sheet tab and choosing 'view code', and later pasting the code.

The user can select each one of those worksheets and choose "WorkSheet" from the drop down to get the list of all supported Worksheet events.



Following are the supported worksheet events that can be added by the user.

```
Private Sub Worksheet_Activate()
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
Private Sub Worksheet_Calculate()
Private Sub Worksheet_Change(ByVal Target As Range)
Private Sub Worksheet_Deactivate()
Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
```

Example

Let us say, we just need to display a message before double click.

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
    MsgBox "Before Double Click"
End Sub
```

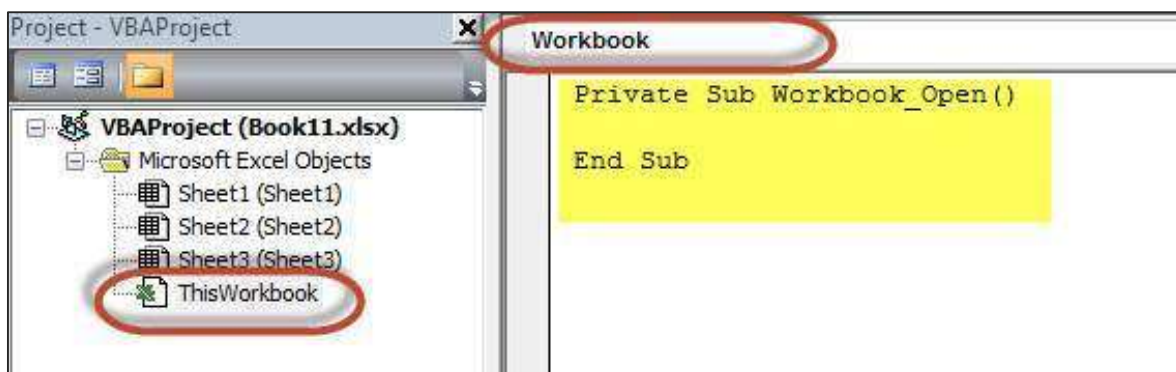
Output

Upon double-clicking on any cell, the message box is displayed to the user as shown in the following screenshot.



Workbook Events

Workbook events are triggered when there is a change in the workbook on the whole. We can add the code for workbook events by selecting the 'ThisWorkbook' and selecting 'workbook' from the dropdown as shown in the following screenshot. Immediately Workbook_open sub procedure is displayed to the user as seen in the following screenshot.



Following are the supported Workbook events that can be added by the user.

```
Private Sub Workbook_AddinUninstall()
Private Sub Workbook_BeforeClose(Cancel As Boolean)
Private Sub Workbook_BeforePrint(Cancel As Boolean)
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)
Private Sub Workbook_Deactivate()
Private Sub Workbook_NewSheet(ByVal Sh As Object)
Private Sub Workbook_Open()
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
Private Sub Workbook_SheetBeforeDoubleClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)
Private Sub Workbook_SheetCalculate(ByVal Sh As Object)
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)
Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
Private Sub Workbook_SheetFollowHyperlink(ByVal Sh As Object, ByVal Target As Hyperlink)
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, ByVal Target As Range)
Private Sub Workbook_WindowActivate(ByVal Wn As Window)
Private Sub Workbook_WindowDeactivate(ByVal Wn As Window)
Private Sub Workbook_WindowResize(ByVal Wn As Window)
```

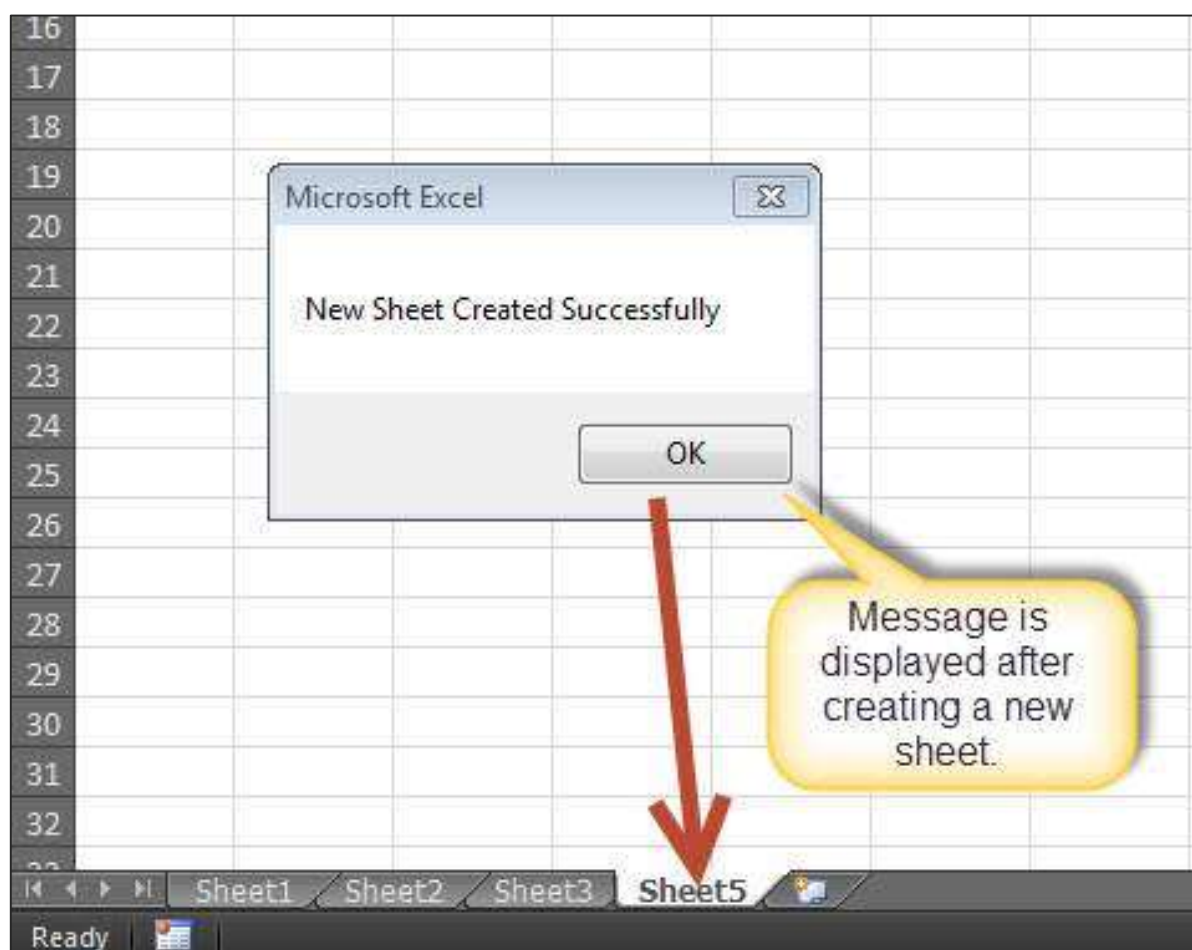

Example

Let us say, we just need to display a message to the user that a new sheet is created successfully, whenever a new sheet is created.

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    MsgBox "New Sheet Created Successfully"
End Sub
```

Output

Upon creating a new excel sheet, a message is displayed to the user as shown in the following screenshot.



18. VBA – Error Handling

There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

Syntax Errors

Syntax errors, also called as parsing errors, occur at the interpretation time for VBScript. For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
Function ErrorHanlding_Demo()  
dim x,y  
x = "Tutorialspoint"  
y = Ucase(x  
End Function
```

Runtime Errors

Runtime errors, also called exceptions, occur during execution, after interpretation.

For example, the following line causes a runtime error because here the syntax is correct but at runtime it is trying to call fnmultiply, which is a non-existing function.

```
Function ErrorHanlding_Demo1()  
    Dim x,y  
    x = 10  
    y = 20  
    z = fnadd(x,y)  
    a = fnmultiply(x,y)  
End Function  
  
Function fnadd(x,y)  
    fnadd = x+y  
End Function
```

Logical Errors

Logical errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

For example, dividing a number by zero or a script that is written which enters into infinite loop.

Err Object

Assume if we have a runtime error, then the execution stops by displaying the error message. As a developer, if we want to capture the error, then **ErrorObject** is used.

Example

In the following example, **Err.Number** gives the error number and **Err.Description** gives the error description.

```
Err.Raise 6    ' Raise an overflow error.
MsgBox "Error # " & CStr(Err.Number) & " " & Err.Description
Err.Clear     ' Clear the error.
```

Error Handling

VBA enables an error-handling routine and can also be used to disable an error-handling routine. Without an On Error statement, any run-time error that occurs is fatal: an error message is displayed, and the execution stops abruptly.

```
On Error { GoTo [ line | 0 | -1 ] | Resume Next }
```

Keyword	Description
GoTo line	Enables the error-handling routine that starts at the line specified in the required line argument. The specified line must be in the same procedure as the On Error statement, or a compile-time error will occur.
GoTo 0	Disables the enabled error handler in the current procedure and resets it to Nothing.
GoTo -1	Disables the enabled exception in the current procedure and resets it to Nothing.
Resume Next	Specifies that when a run-time error occurs, the control goes to the statement immediately following the statement where the error occurred, and the execution continues from that point

Example

```
Public Sub OnErrorDemo()  
    On Error GoTo ErrorHandler ' Enable error-handling routine.  
    Dim x, y, z As Integer  
    x = 50  
    y = 0  
    z = x / y ' Divide by ZERO Error Raises  
  
    ErrorHandler: ' Error-handling routine.  
    Select Case Err.Number ' Evaluate error number.  
        Case 10 ' Divide by zero error  
            MsgBox ("You attempted to divide by zero!")  
        Case Else  
            MsgBox "UNKNOWN ERROR - Error# " & Err.Number & " : " &  
Err.Description  
    End Select  
    Resume Next  
End Sub
```

19. VBA – Excel Objects

When programming using VBA, there are few important objects that a user would be dealing with.

- Application Objects
- Workbook Objects
- Worksheet Objects
- Range Objects

Application Objects

The Application object consists of the following:

- Application-wide settings and options.
- Methods that return top-level objects, such as ActiveCell, ActiveSheet, and so on.

Example

```
'Example 1 :  
Set xlapp = CreateObject("Excel.Sheet")  
xlapp.Application.Workbooks.Open "C:\test.xls"  
  
'Example 2 :  
Application.Windows("test.xls").Activate  
  
'Example 3:  
Application.ActiveCell.Font.Bold = True
```

Workbook Objects

The Workbook object is a member of the Workbooks collection and contains all the Workbook objects currently open in Microsoft Excel.

Example

```
'Ex 1 : To close Workbooks
Workbooks.Close

'Ex 2 : To Add an Empty Work Book
Workbooks.Add

'Ex 3: To Open a Workbook
Workbooks.Open FileName:="Test.xls", ReadOnly:=True

'Ex : 4 - To Activate WorkBooks
Workbooks("Test.xls").Worksheets("Sheet1").Activate
```

Worksheet Objects

The Worksheet object is a member of the Worksheets collection and contains all the Worksheet objects in a workbook.

Example

```
'Ex 1 : To make it Invisible
Worksheets(1).Visible = False

'Ex 2 : To protect an WorkSheet
Worksheets("Sheet1").Protect password:=strPassword, scenarios:=True
```

Range Objects

Range Objects represent a cell, a row, a column, or a selection of cells containing one or more continuous blocks of cells.

```
'Ex 1 : To Put a value in the cell A5
Worksheets("Sheet1").Range("A5").Value = "5235"

'Ex 2 : To put a value in range of Cells
Worksheets("Sheet1").Range("A1:A4").Value = 5
```

20. VBA – Text Files

You can also read Excel File and write the contents of the cell into a Text File using VBA. VBA allows the users to work with text files using two methods:

- File System Object
- Write Command

File System Object (FSO)

As the name suggests, FSOs help the developers to work with drives, folders, and files. In this section, we will discuss how to use a FSO.

Object Type	Description
Drive	Drive is an Object. Contains methods and properties that allow you to gather information about a drive attached to the system.
Drives	Drives is a Collection. It provides a list of the drives attached to the system, either physically or logically.
File	File is an Object. It contains methods and properties that allow developers to create, delete, or move a file.
Files	Files is a Collection. It provides a list of all the files contained within a folder.
Folder	Folder is an Object. It provides methods and properties that allow the developers to create, delete, or move folders.
Folders	Folders is a Collection. It provides a list of all the folders within a folder.
TextStream	TextStream is an Object. It enables the developers to read and write text files.

Drive

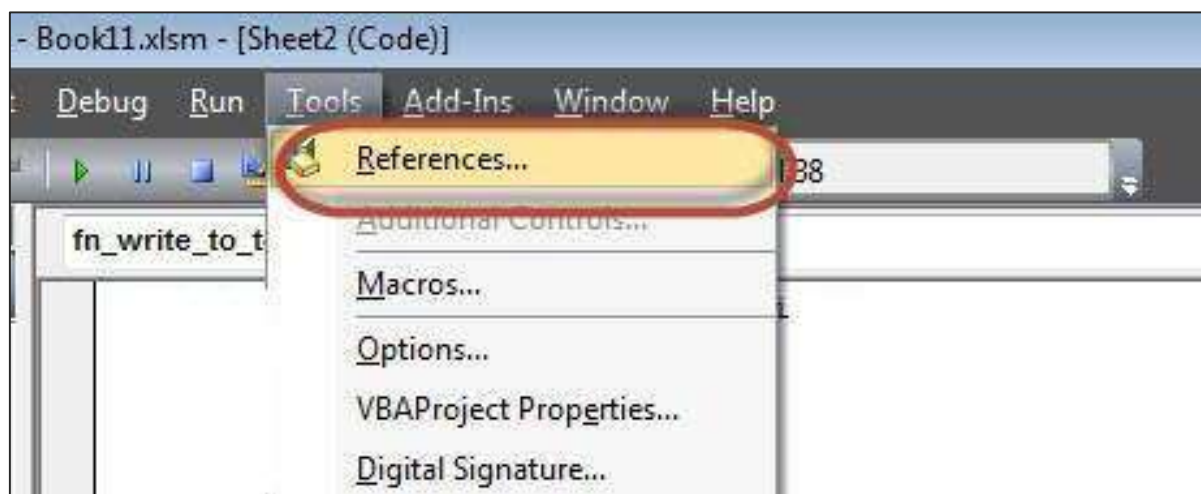
Drive is an object, which provides access to the properties of a particular disk drive or network share. Following properties are supported by **Drive** object:

- AvailableSpace
- DriveLetter
- DriveType
- FileSystem
- FreeSpace
- IsReady

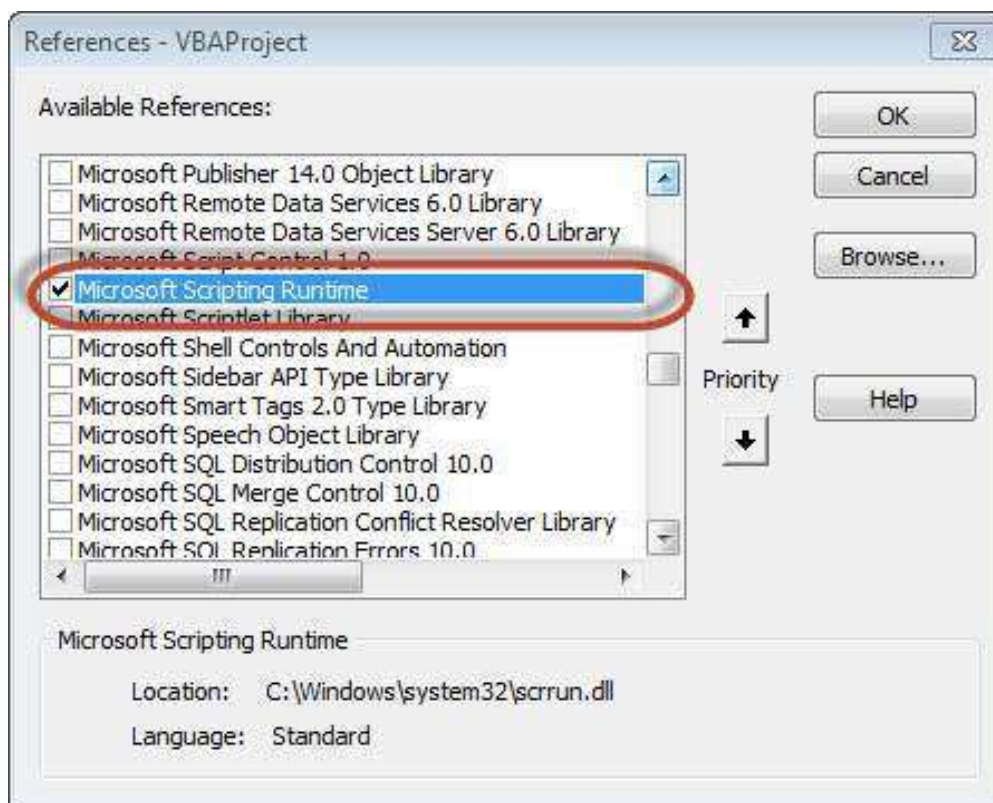
- Path
- RootFolder
- SerialNumber
- ShareName
- TotalSize
- VolumeName

Example

Step 1: Before proceeding to scripting using FSO, we should enable Microsoft Scripting Runtime. To do the same, navigate to Tools -> References as shown in the following screenshot.



Step 2: Add "Microsoft Scripting RunTime" and Click OK.



Step 3 : Add Data that you would like to write in a Text File and add a Command Button.

A1		fx State							
	A	B	C	D	E	F	G	H	I
1	State	Prevalence	95% Confidence Interval						
2	Alabama	33	(31.5, 34.4)						
3	Alaska	25.7	(23.9, 27.5)						
4	Arizona	26	(24.3, 27.8)						
5	Arkansas	34.5	(32.7, 36.4)						
6	California	25	(23.9, 26.0)						
7	Colorado	20.5	(19.5, 21.4)						
8	Connecticut	25.6	(24.3, 26.9)						
9	Delaware	26.9	(25.2, 28.6)						
10	District of Columbia	21.9	(19.8, 24.0)						
11									
12									

Step 4: Now it is time to Script.

```
Private Sub fn_write_to_text_Click()
    Dim FilePath As String
    Dim CellData As String
    Dim LastCol As Long
    Dim LastRow As Long
    Dim fso As FileSystemObject
    Set fso = New FileSystemObject
    Dim stream As TextStream
```



```

LastCol = ActiveSheet.UsedRange.Columns.Count
LastRow = ActiveSheet.UsedRange.Rows.Count

' Create a TextStream.
Set stream = fso.OpenTextFile("D:\Try\Support.log", ForWriting, True)

CellData = ""

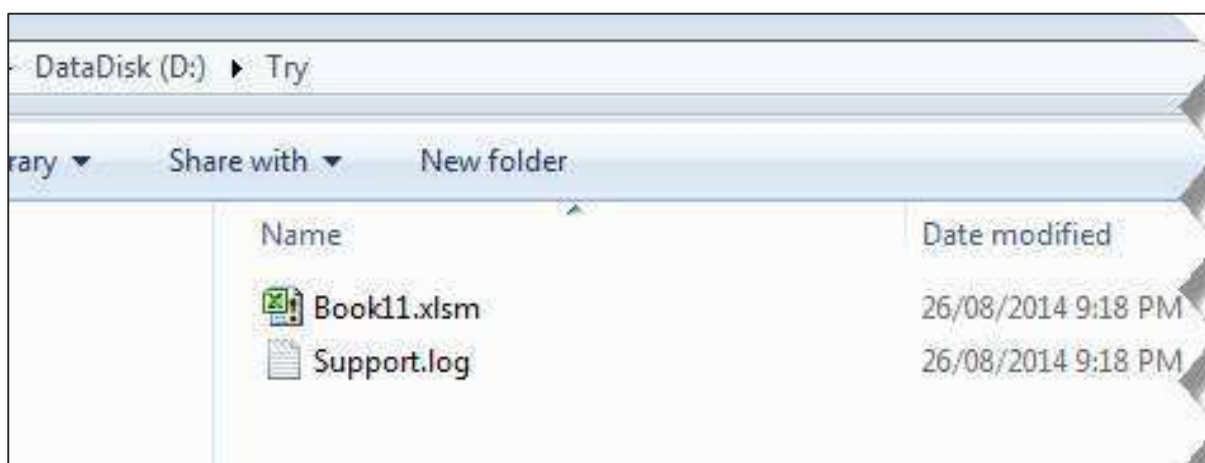
For i = 1 To LastRow
    For j = 1 To LastCol
        CellData = Trim(ActiveCell(i, j).Value)
        stream.WriteLine "The Value at location (" & i & "," & j & ")" & CellData
    Next j
Next i

stream.Close
MsgBox ("Job Done")
End Sub

```

Output

When executing the script, ensure that you place the cursor in the first cell of the worksheet. The Support.log file is created as shown in the following screenshot under "D:\Try".



The Contents of the file are shown in the following screenshot.

1	The Value at location (1,1)State
2	The Value at location (1,2)Prevalence
3	The Value at location (1,3)95% Confidence Interval
4	The Value at location (2,1)Alabama
5	The Value at location (2,2)33
6	The Value at location (2,3) (31.5, 34.4)
7	The Value at location (3,1)Alaska
8	The Value at location (3,2)25.7
9	The Value at location (3,3) (23.9, 27.5)
10	The Value at location (4,1)Arizona
11	The Value at location (4,2)26
12	The Value at location (4,3) (24.3, 27.8)
13	The Value at location (5,1)Arkansas
14	The Value at location (5,2)34.5
15	The Value at location (5,3) (32.7, 36.4)
16	The Value at location (6,1)California
17	The Value at location (6,2)25
18	The Value at location (6,3) (23.9, 26.0)
19	The Value at location (7,1)Colorado
20	The Value at location (7,2)20.5
21	The Value at location (7,3) (19.5, 21.4)
22	The Value at location (8,1)Connecticut
23	The Value at location (8,2)25.6
24	The Value at location (8,3) (24.3, 26.9)
25	The Value at location (9,1)Delaware
26	The Value at location (9,2)26.9
27	The Value at location (9,3) (25.2, 28.6)
28	The Value at location (10,1)District of Columbia
29	The Value at location (10,2)21.9
30	The Value at location (10,3) (19.8, 24.0)
31	

Write Command

Unlike FSO, we need NOT add any references, however, we will NOT be able to work with drives, files and folders. We will be able to just add the stream to the text file.

Example

```
Private Sub fn_write_to_text_Click()
    Dim FilePath As String
    Dim CellData As String
    Dim LastCol As Long
    Dim LastRow As Long

    LastCol = ActiveSheet.UsedRange.Columns.Count
    LastRow = ActiveSheet.UsedRange.Rows.Count

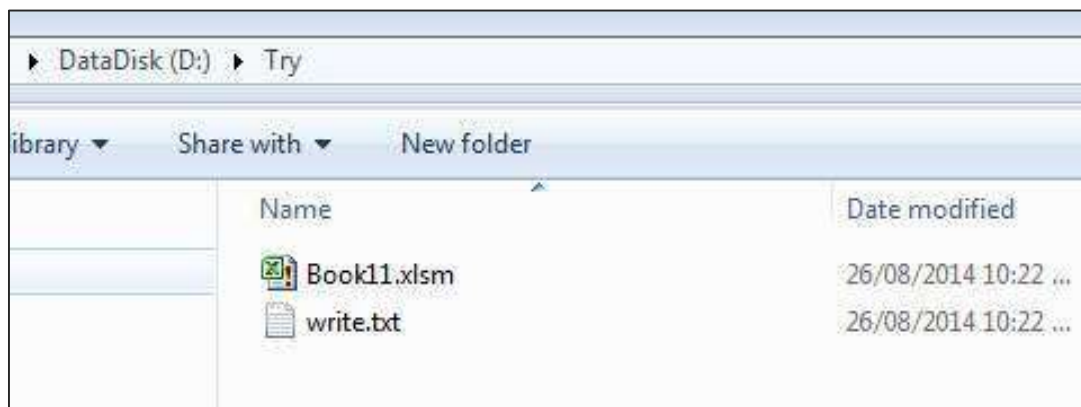
    FilePath = "D:\Try\write.txt"
    Open FilePath For Output As #2

    CellData = ""
    For i = 1 To LastRow
        For j = 1 To LastCol
            CellData = "The Value at location (" & i & ", " & j & ")" &
Trim(ActiveCell(i, j).Value)
            Write #2, CellData
        Next j
    Next i

    Close #2
    MsgBox ("Job Done")
End Sub
```

Output

Upon executing the script, the "write.txt" file is created in the location "D:\Try" as shown in the following screenshot.



The contents of the file are shown in the following screenshot.

```

1 The Value at location (1,1)State
2 The Value at location (1,2)Prevalence
3 The Value at location (1,3)95% Confidence Interval
4 The Value at location (2,1)Alabama
5 The Value at location (2,2)33
6 The Value at location (2,3)(31.5, 34.4)
7 The Value at location (3,1)Alaska
8 The Value at location (3,2)25.7
9 The Value at location (3,3)(23.9, 27.5)
10 The Value at location (4,1)Arizona
11 The Value at location (4,2)26
12 The Value at location (4,3)(24.3, 27.8)
13 The Value at location (5,1)Arkansas
14 The Value at location (5,2)34.5
15 The Value at location (5,3)(32.7, 36.4)
16 The Value at location (6,1)California
17 The Value at location (6,2)25
18 The Value at location (6,3)(23.9, 26.0)
19 The Value at location (7,1)Colorado
20 The Value at location (7,2)20.5
21 The Value at location (7,3)(19.5, 21.4)
22 The Value at location (8,1)Connecticut
23 The Value at location (8,2)25.6
24 The Value at location (8,3)(24.3, 26.9)
25 The Value at location (9,1)Delaware
26 The Value at location (9,2)26.9
27 The Value at location (9,3)(25.2, 28.6)
28 The Value at location (10,1)District of Columbia
29 The Value at location (10,2)21.9
30 The Value at location (10,3)(19.8, 24.0)

```

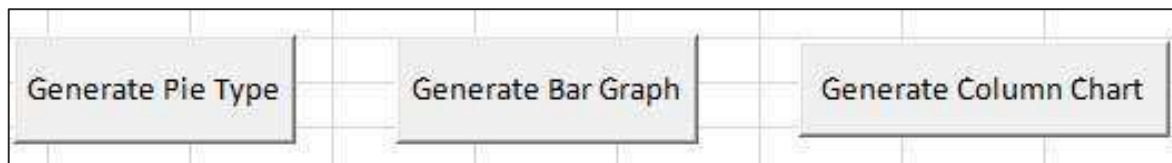
21. VBA – Programming Charts

Using VBA, you can generate charts based on certain criteria. Let us take a look at it using an example.

Step 1: Enter the data against which the graph has to be generated.

Q23		f _x
	A	B
1	Year	Fuel Usage in Million Cubic Meters
2	1980	185.5
3	1990	214.1
4	2000	467.34
5	2010	1023.77
6		

Step 2: Create 3 buttons - one to generate a bar graph, another to generate a pie chart, and another to generate a column chart.



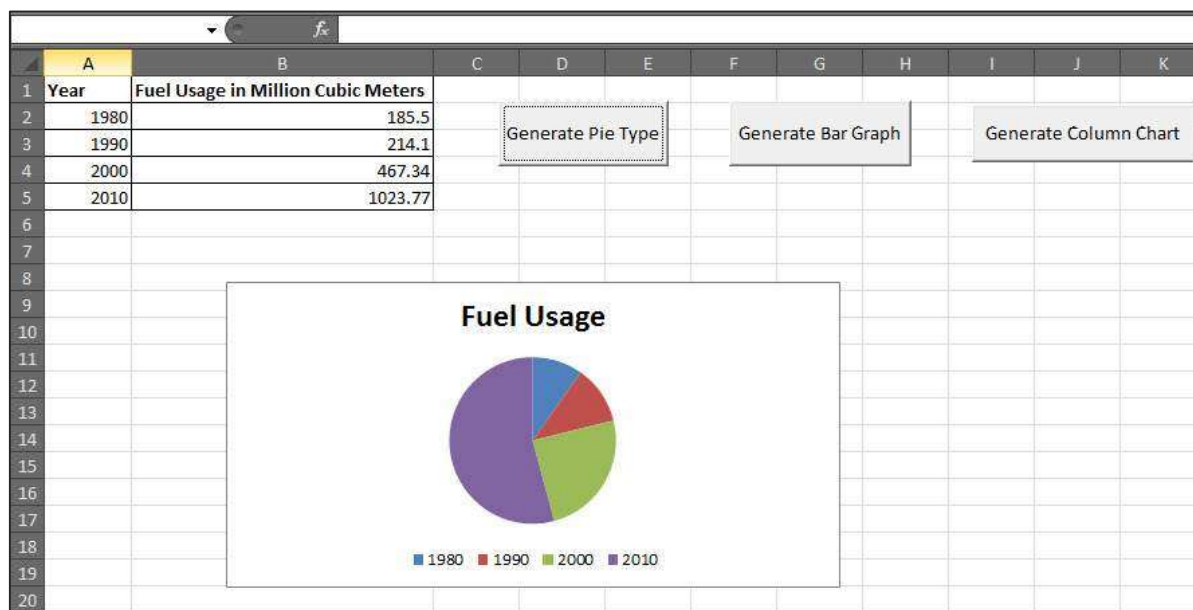
Step 3: Develop a Macro to generate each one of these type of charts.

```
' Procedure to Generate Pie Chart
Private Sub fn_generate_pie_graph_Click()
    Dim cht As ChartObject
    For Each cht In Worksheets(1).ChartObjects
        cht.Chart.Type = xlPie
    Next cht
End Sub

' Procedure to Generate Bar Graph
Private Sub fn_Generate_Bar_Graph_Click()
    Dim cht As ChartObject
    For Each cht In Worksheets(1).ChartObjects
        cht.Chart.Type = xlBar
    Next cht
End Sub

' Procedure to Generate Column Graph
Private Sub fn_generate_column_graph_Click()
    Dim cht As ChartObject
    For Each cht In Worksheets(1).ChartObjects
        cht.Chart.Type = xlColumn
    Next cht
End Sub
```

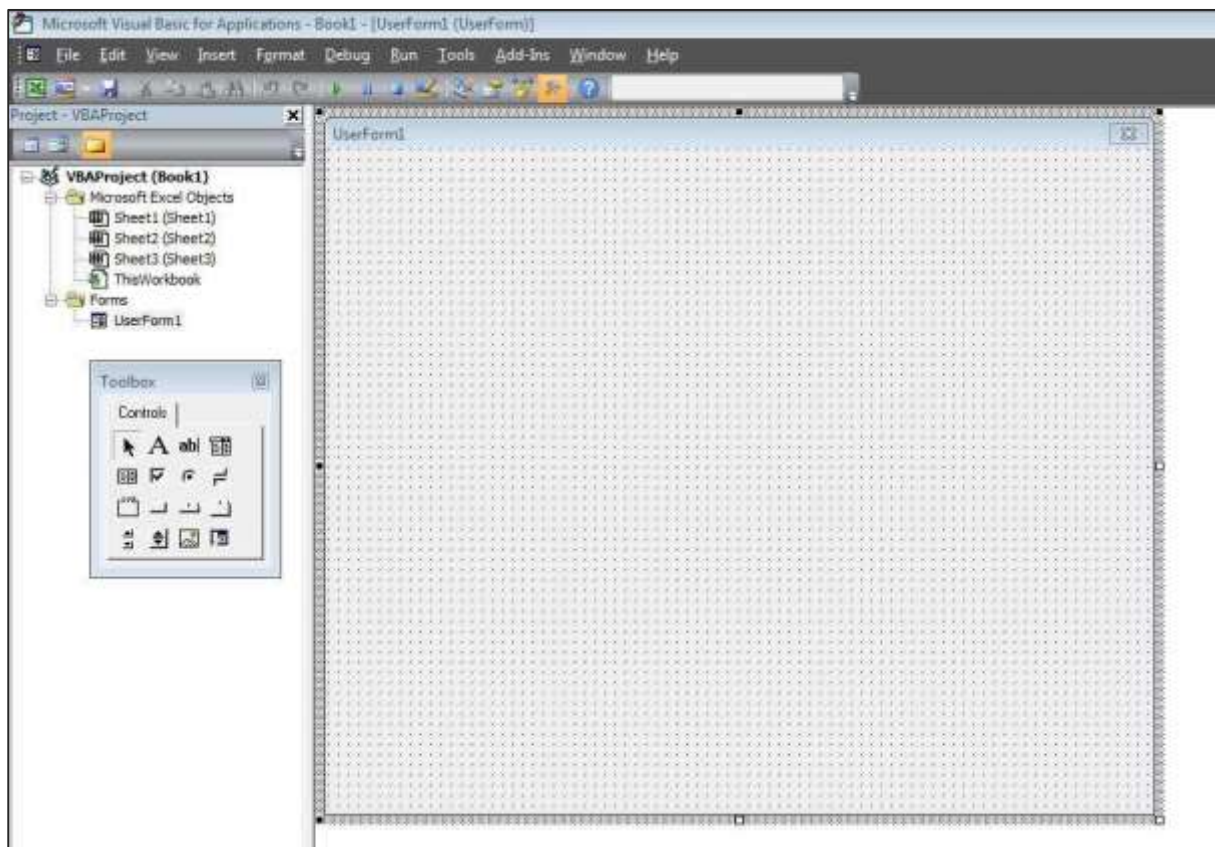
Step 4: Upon clicking the corresponding button, the chart is created. In the following output, click on generate Pie Chart button.



22. VBA – User Forms

A **User Form** is a custom-built dialog box that makes a user data entry more controllable and easier to use for the user. In this chapter, you will learn to design a simple form and add data into excel.

Step 1: Navigate to VBA Window by pressing Alt+F11 and Navigate to "Insert" Menu and select "User Form". Upon selecting, the user form is displayed as shown in the following screenshot.



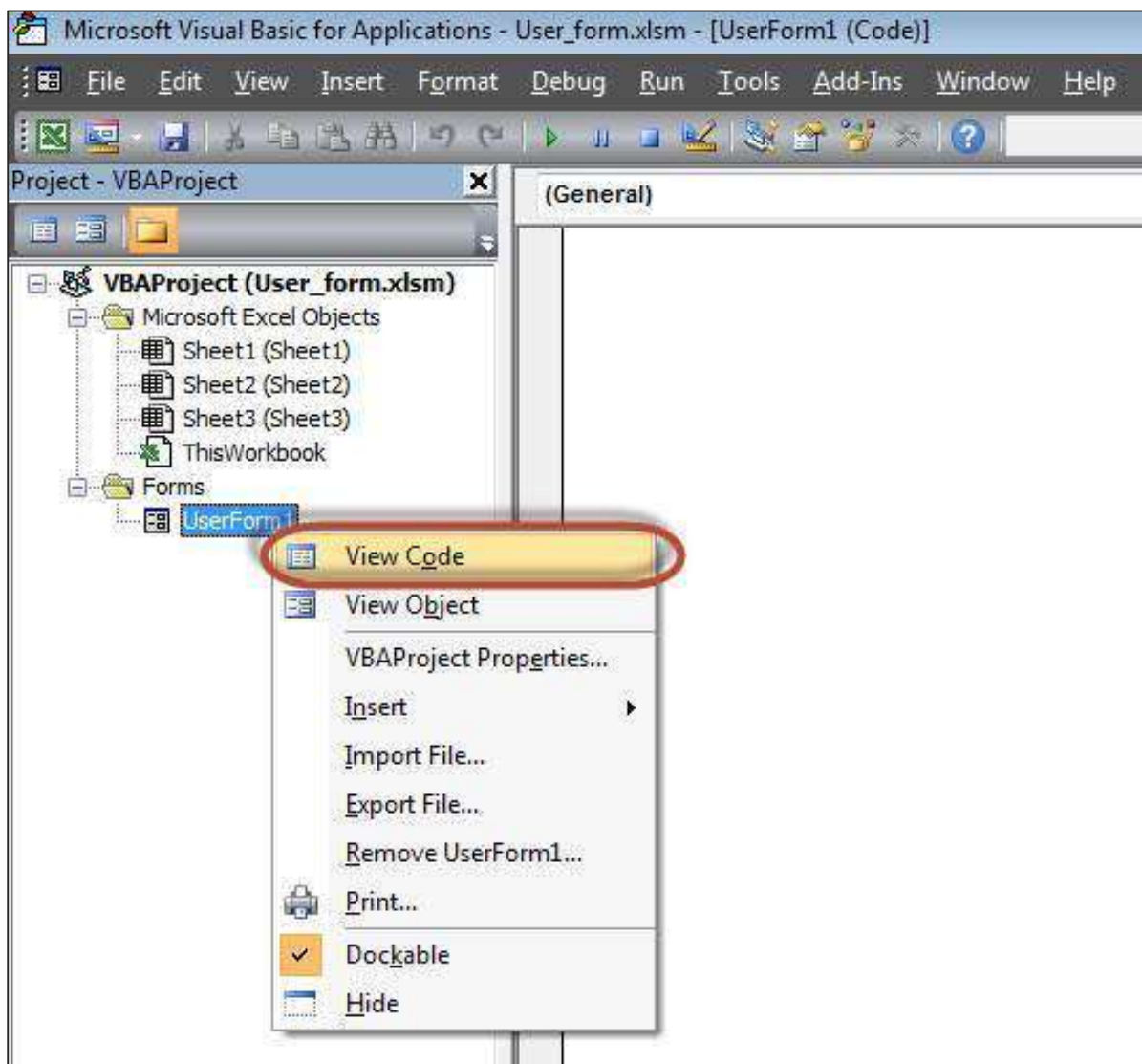
Step 2: Design the forms using the given controls.

Step 3: After adding each control, the controls have to be named. Caption corresponds to what appears on the form and name corresponds to the logical name that will be appearing when you write VBA code for that element.

Step 4: Following are the names against each one of the added controls.

Control	Logical Name	Caption
From	frmempform	Employee Form
Employee ID Label Box	empid	Employee ID
firstname Label Box	firstname	First Name
lastname Label Box	lastname	Last Name
dob Label Box	dob	Date of Birth
mailid Label Box	mailid	Email ID
Passportholder Label Box	Passportholder	Passport Holder
Emp ID Text Box	txtempid	NOT Applicable
First Name Text Box	txtfirstname	NOT Applicable
Last Name Text Box	txtlastname	NOT Applicable
Email ID Text Box	txtemailid	NOT Applicable
Date Combo Box	cmbdate	NOT Applicable
Month Combo Box	cmbmonth	NOT Applicable
Year Combo Box	cmbyear	NOT Applicable
Yes Radio Button	radioyes	Yes
No Radio Button	radiono	No
Submit Button	btnsubmit	Submit
Cancel Button	btncancel	Cancel

Step 5: Add the code for the form load event by performing a right-click on the form and selecting 'View Code'.



Step 6: Select 'Userform' from the objects drop-down and select 'Initialize' method as shown in the following screenshot.



Step 7: Upon Loading the form, ensure that the text boxes are cleared, drop-down boxes are filled and Radio buttons are reset.

```
Private Sub UserForm_Initialize()

    'Empty Emp ID Text box and Set the Cursor
    txttempid.Value = ""
    txttempid.SetFocus

    'Empty all other text box fields
    txtfirstname.Value = ""
    txtlastname.Value = ""
    txtemailid.Value = ""

    'Clear All Date of Birth Related Fields
    cmbdate.Clear
    cmbmonth.Clear
    cmbyear.Clear

    'Fill Date Drop Down box - Takes 1 to 31
    With cmbdate
        .AddItem "1"
        .AddItem "2"
        .AddItem "3"
        .AddItem "4"
        .AddItem "5"
        .AddItem "6"
        .AddItem "7"
        .AddItem "8"
        .AddItem "9"
        .AddItem "10"
        .AddItem "11"
        .AddItem "12"
        .AddItem "13"
        .AddItem "14"
```

```
.AddItem "15"  
.AddItem "16"  
.AddItem "17"  
.AddItem "18"  
.AddItem "19"  
.AddItem "20"  
.AddItem "21"  
.AddItem "22"  
.AddItem "23"  
.AddItem "24"  
.AddItem "25"  
.AddItem "26"  
.AddItem "27"  
.AddItem "28"  
.AddItem "29"  
.AddItem "30"  
.AddItem "31"  
End With  
  
'Fill Month Drop Down box - Takes Jan to Dec  
With cmbmonth  
.AddItem "JAN"  
.AddItem "FEB"  
.AddItem "MAR"  
.AddItem "APR"  
.AddItem "MAY"  
.AddItem "JUN"  
.AddItem "JUL"  
.AddItem "AUG"  
.AddItem "SEP"  
.AddItem "OCT"  
.AddItem "NOV"  
.AddItem "DEC"  
End With
```

```
'Fill Year Drop Down box - Takes 1980 to 2014
```

```
With cmbyear
```

```
.AddItem "1980"  
.AddItem "1981"  
.AddItem "1982"  
.AddItem "1983"  
.AddItem "1984"  
.AddItem "1985"  
.AddItem "1986"  
.AddItem "1987"  
.AddItem "1988"  
.AddItem "1989"  
.AddItem "1990"  
.AddItem "1991"  
.AddItem "1992"  
.AddItem "1993"  
.AddItem "1994"  
.AddItem "1995"  
.AddItem "1996"  
.AddItem "1997"  
.AddItem "1998"  
.AddItem "1999"  
.AddItem "2000"  
.AddItem "2001"  
.AddItem "2002"  
.AddItem "2003"  
.AddItem "2004"  
.AddItem "2005"  
.AddItem "2006"  
.AddItem "2007"  
.AddItem "2008"  
.AddItem "2009"  
.AddItem "2010"  
.AddItem "2011"
```

```

.AddItem "2012"
.AddItem "2013"
.AddItem "2014"
End With

'Reset Radio Button. Set it to False when form loads.
radioyes.Value = False
radiono.Value = False
End Sub

```

Step 8: Now add the code to the Submit button. Upon clicking the submit button, the user should be able to add the values into the worksheet.

```

Private Sub btnsubmit_Click()
    Dim emptyRow As Long

    'Make Sheet1 active
    Sheet1.Activate

    'Determine emptyRow
    emptyRow = WorksheetFunction.CountA(Range("A:A")) + 1

    'Transfer information
    Cells(emptyRow, 1).Value = txttempid.Value
    Cells(emptyRow, 2).Value = txtfirstname.Value
    Cells(emptyRow, 3).Value = txtlastname.Value
    Cells(emptyRow, 4).Value = cmbdate.Value & "/" & cmbmonth.Value & "/" &
cmbyear.Value
    Cells(emptyRow, 5).Value = txtemailid.Value

    If radioyes.Value = True Then
        Cells(emptyRow, 6).Value = "Yes"
    Else
        Cells(emptyRow, 6).Value = "No"
    End If
End Sub

```

Step 9: Add a method to close the form when the user clicks the Cancel button.

```

Private Sub btncancel_Click()
    Unload Me
End Sub

```

Step 10: Execute the form by clicking the "Run" button. Enter the values into the form and click the 'Submit' button. Automatically the values will flow into the worksheet as shown in the following screenshot.

Employee ID	First Name	Last Name	Date of Birth	Email ID	Passport Holder
100	Jim	Smith	1-Feb-82	jim@Tutorialspoint.com	Yes
101	Jack	Rose	19-Aug-88	Jack.Rose@Tutorialspoint.com	No
102	Cathy	Nelson	27-Jul-86	cathy@tutorialspoint.com	No

Employee Form	
Employee ID	102
First Name	Cathy
Last Name	Nelson
Date of Birth	27 JUL 1986
Email ID	cathy@tutorialspoint.com
Passport Holder	<input type="radio"/> Yes <input checked="" type="radio"/> No
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	