



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	8
Modules	8
Procedure	9
4. VBA – MACRO COMMENTS	10
5. VBA – MESSAGE BOX.....	11
6. VBA – INPUTBOX	16
7. VBA – VARIABLES.....	19
Data Types	19
8. VBA – CONSTANTS.....	22
9. VBA – OPERATORS.....	24
The Arithmetic Operators.....	24
The Comparison Operators	26
The Logical Operators	29
The Concatenation Operators	32

10. VBA – DECISIONS.....	35
If Statement	36
If Else Statement	37
If Elseif - Else statement.....	40
Nested If Statement	42
Switch Statement.....	44
11. VBA – LOOPS	47
For Loop	48
For Each Loops	50
While Wend Loops	52
Do While Loops	54
Do Until Loops.....	57
Loop Control Statements.....	61
Exit For	61
Exit Do.....	63
12. VBA – STRINGS	65
Instr	66
InString Reverse	67
LCase	69
UCase	70
Left.....	70
Right	72
Mid	73
Ltrim	74
Rtrim	74
Trim	75
Len	75

Replace	76
Space.....	78
StrComp	79
String Function	80
String Reverse Function.....	81
13. VBA – DATE-TIME FUNCTION	83
Date Functions	83
Date Function.....	84
CDate Function.....	84
DateAdd Function	85
DateDiff Function	87
DatePart Function	90
DateSerial Function.....	91
Format DateTime Function.....	92
IsDate Function	93
Day Function	94
Month Function	95
Year Function	95
Month Name.....	96
WeekDay.....	97
WeekDay Name	98
Time Functions.....	99
Now Function.....	100
Hour Function	100
Minute Function.....	101
Second Function.....	102
Time Function	102

Timer Function	103
Time Serial Function	103
TimeValue Function	104
14. VBA – ARRAYS.....	106
Array Declaration	106
Assigning Values to an Array	106
Multi-Dimensional Arrays	107
ReDim Statement	109
Array Methods	110
LBound Function	111
UBound Function	112
Split Function	113
Join Function	115
Filter Function	116
IsArray Function	117
Erase Function.....	118
15. VBA – USER-DEFINED FUNCTIONS.....	120
Function Definition	120
Calling a Function	121
16. VBA – SUB PROCEDURE	123
Calling Procedures.....	123
17. VBA – EVENTS.....	125
Worksheet Events	125
Workbook Events	126

18. VBA – ERROR HANDLING	129
Syntax Errors	129
Runtime Errors	129
Logical Errors.....	130
Err Object	130
Error Handling	130
19. VBA – EXCEL OBJECTS.....	132
Application Objects	132
Workbook Objects	132
Worksheet Objects.....	133
Range Objects	133
20. VBA – TEXT FILES	134
File System Object (FSO)	134
Write Command.....	139
21. VBA – PROGRAMMING CHARTS	141
22. VBA – USER FORMS	144

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.

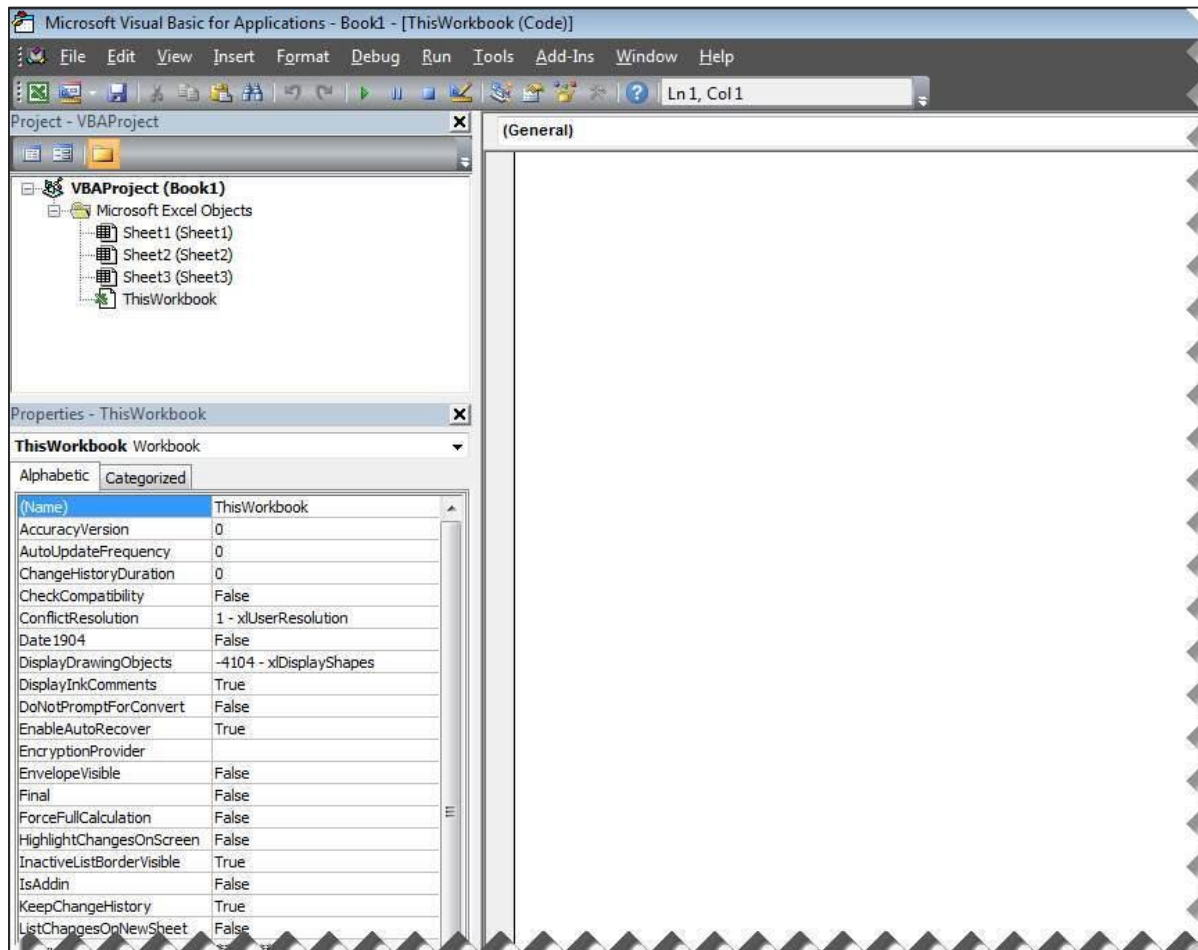
Application of VBA

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.

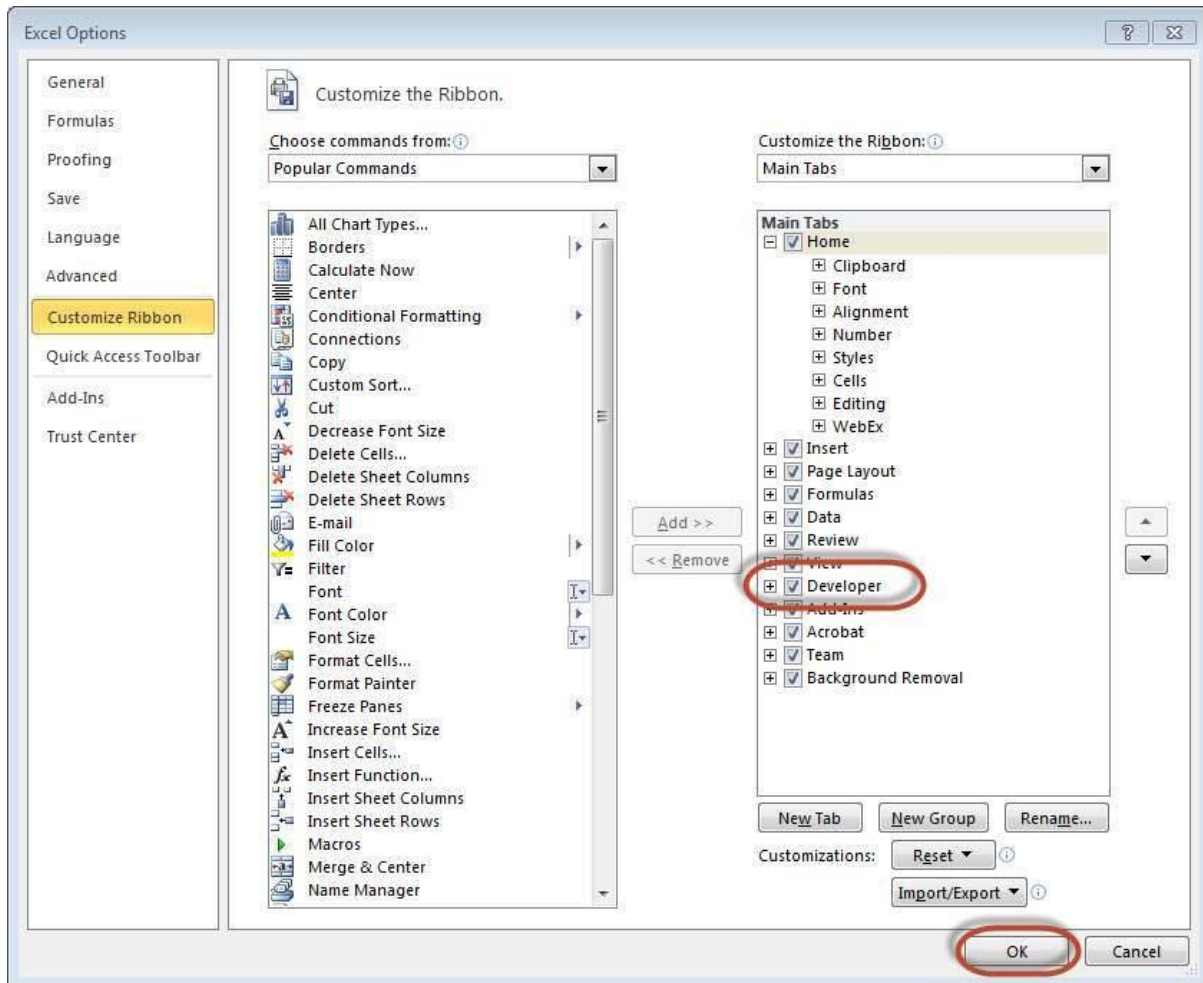


2. VBA – Excel Macros

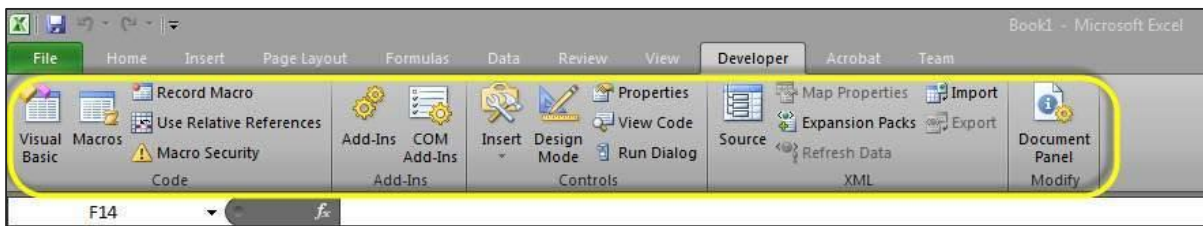
In this chapter, you will learn how to write a simple macro in a step by step manner.

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

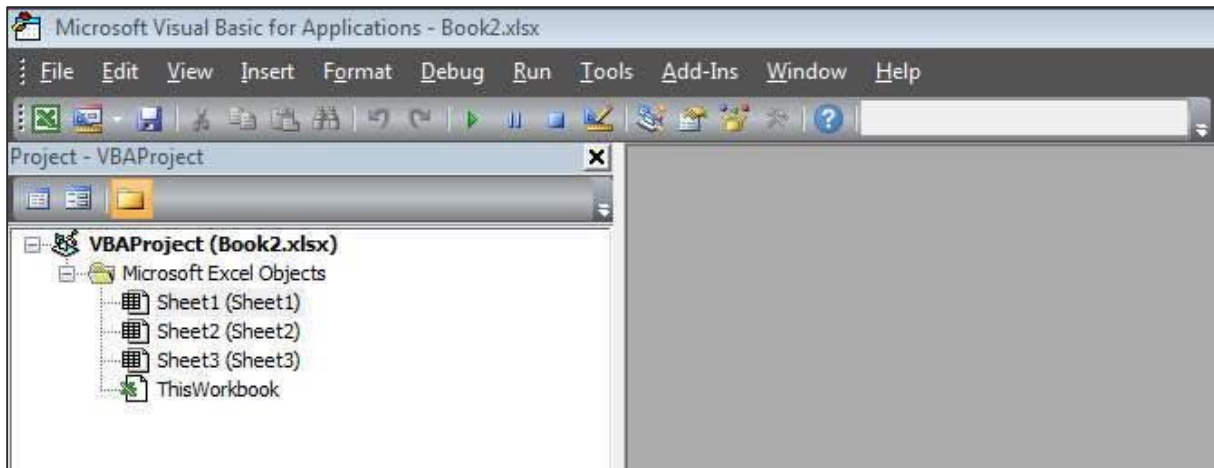
Step 2: Click 'Customize the Ribbon' tab and check 'Developer'. Click 'OK'.



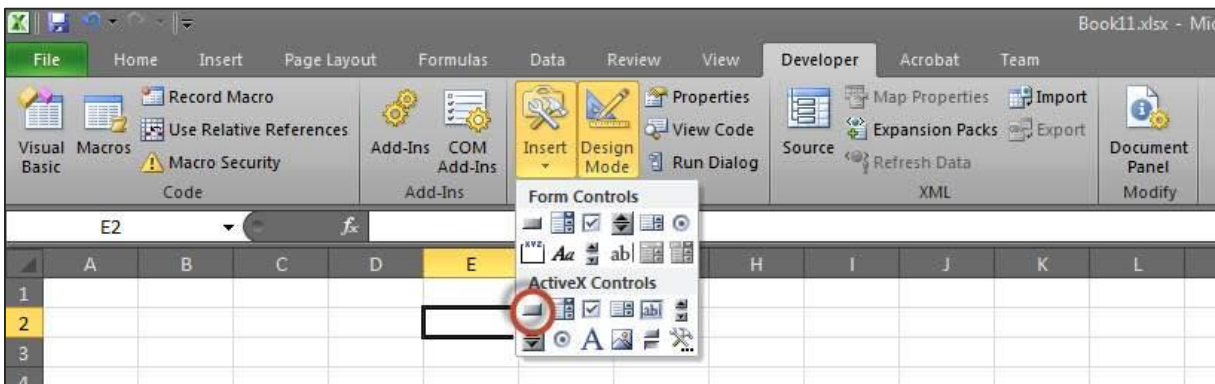
Step 3: The 'Developer' ribbon appears in the menu bar.



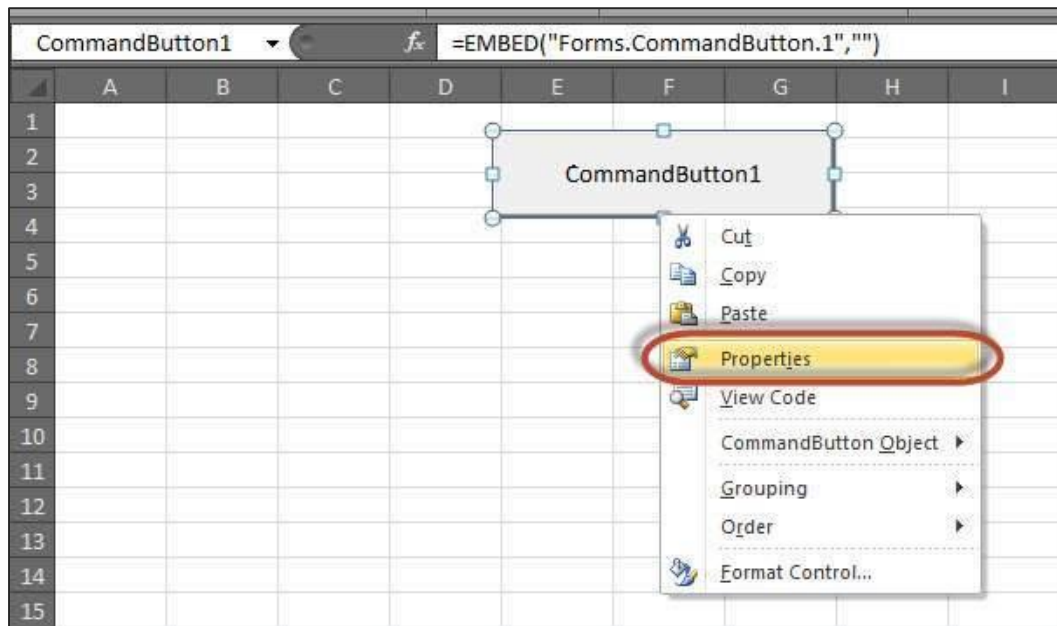
Step 4: Click the 'Visual Basic' button to open the VBA Editor.



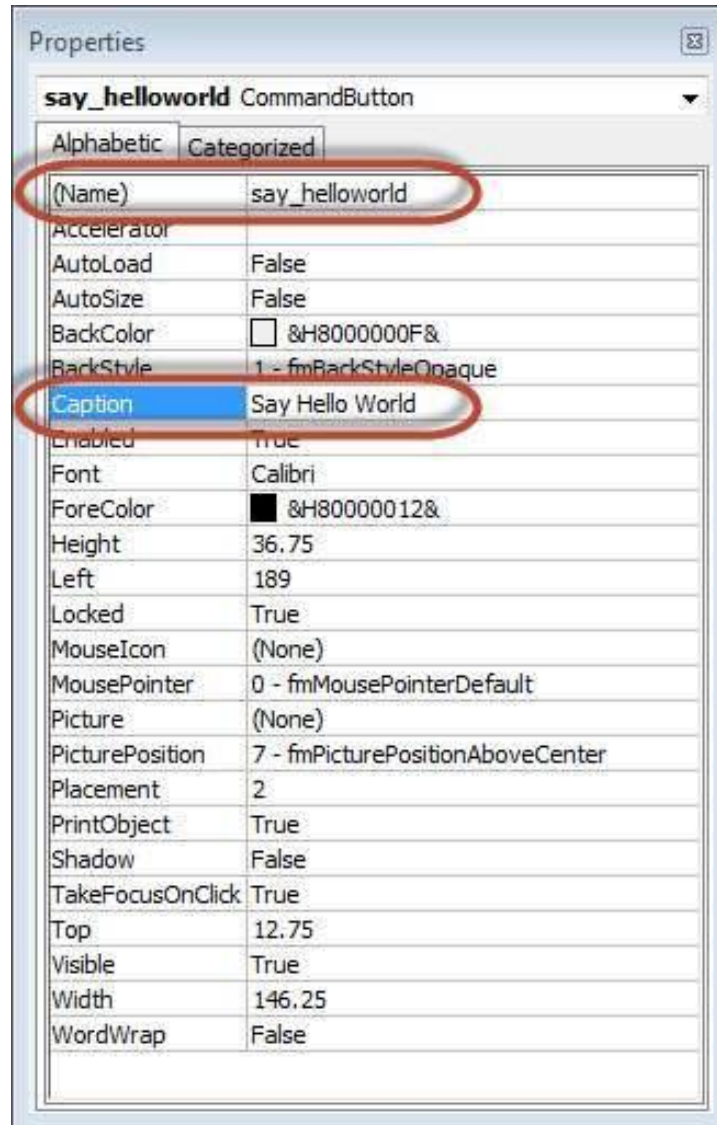
Step 5: Start scripting by adding a button. Click Insert -> Select the button.



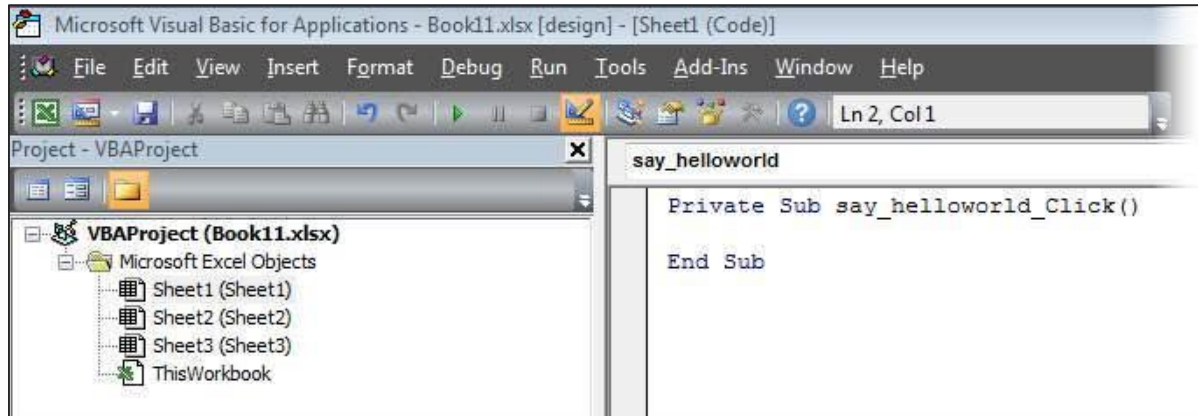
Step 6: Perform a right-click and choose 'properties'.



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.



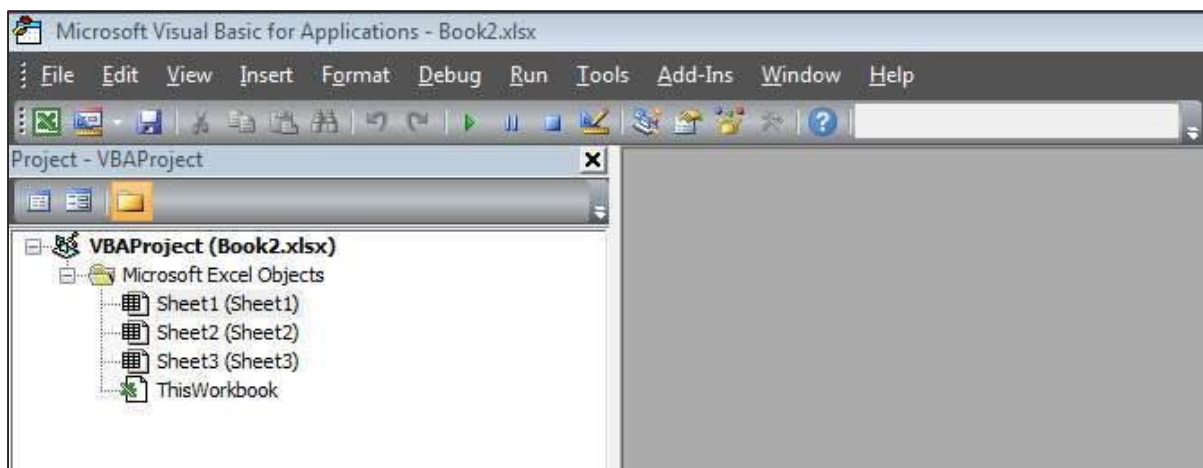
Note: In further chapters, we will demonstrate using a simple button, as explained from step#1 to 10. Hence , it is important to understand this chapter thoroughly.

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.

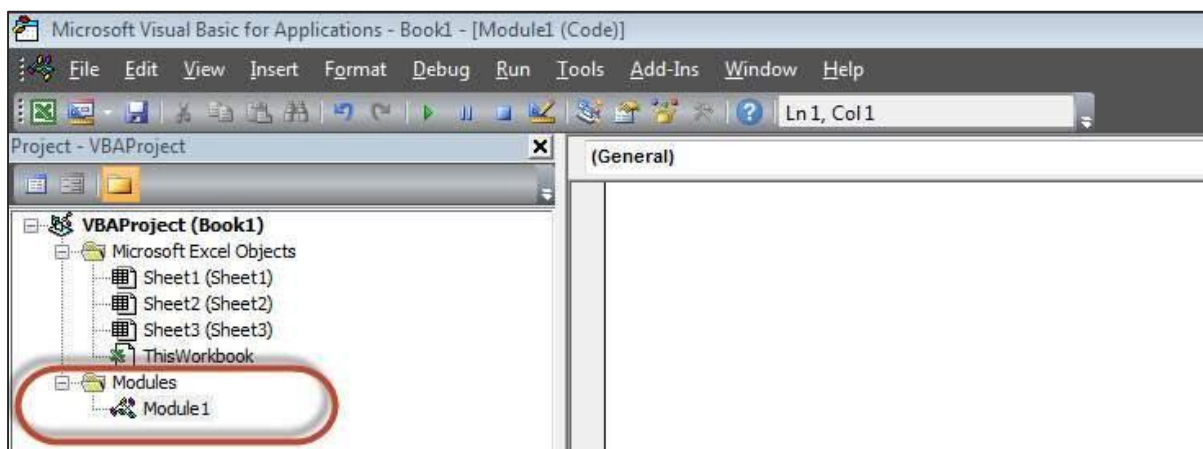
Modules

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.

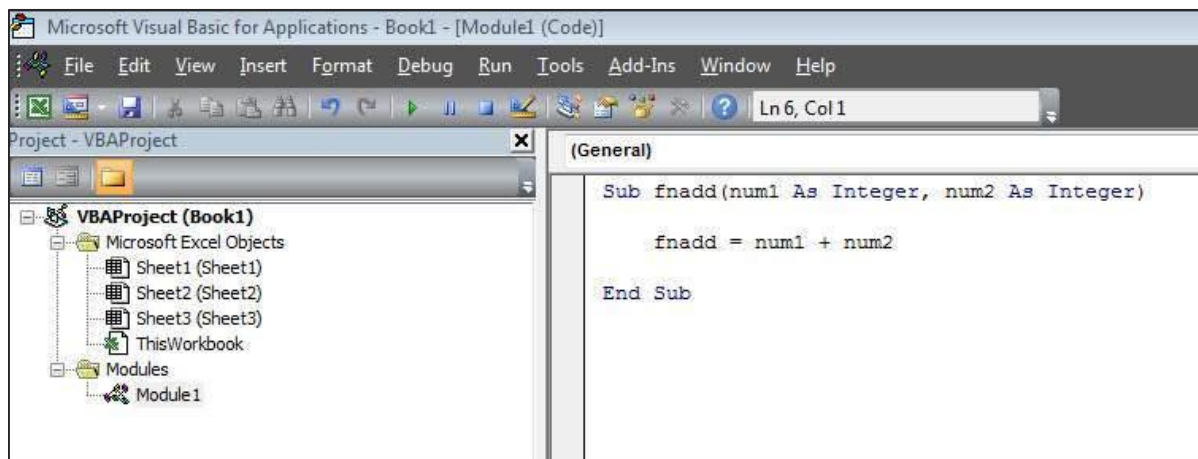
Within the modules, we can write VBA code and the code is written within a Procedure. A Procedure/Sub Procedure is a series of VBA statements instructing what to do.



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 : Tutorialspoint  
' 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 : Tutorialspoint/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

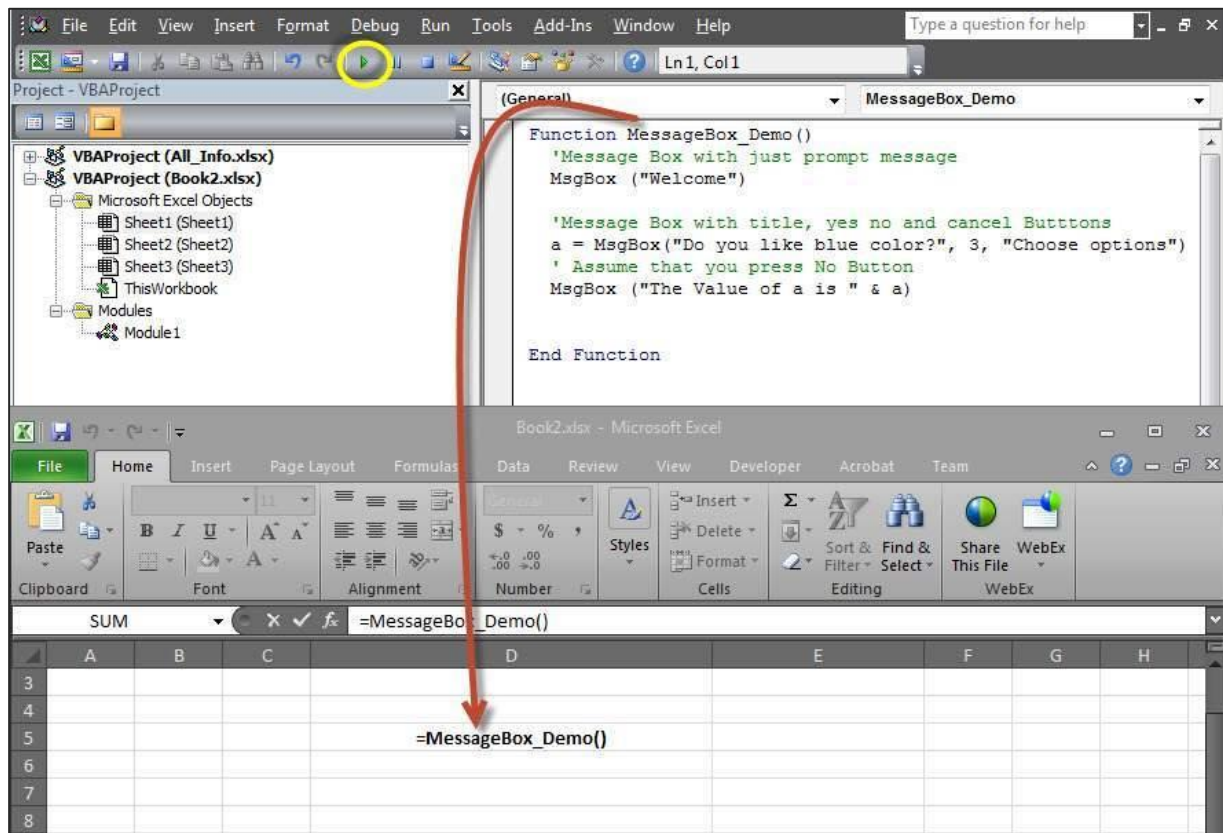
```
Function MsgBox_Demo()
'Message Box with just prompt message
MsgBox("Welcome")

'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

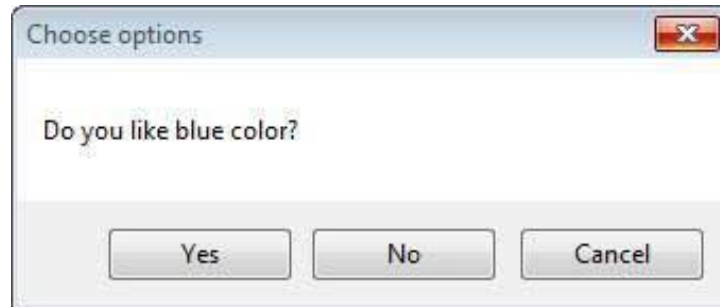
Step 1: The above Function can be executed either by clicking the "Run" button on VBA Window or by calling the function from Excel Worksheet as shown in the following screenshot.



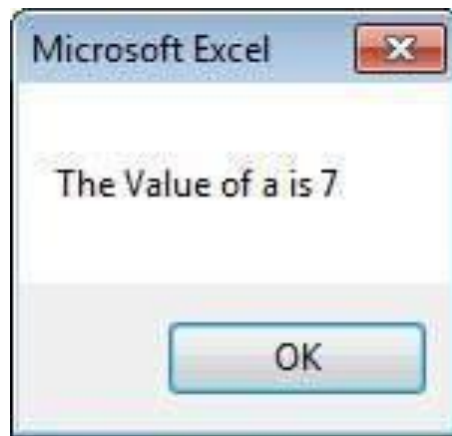
Step 2: A Simple Message box is displayed with a message "Welcome" and an "OK" Button



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

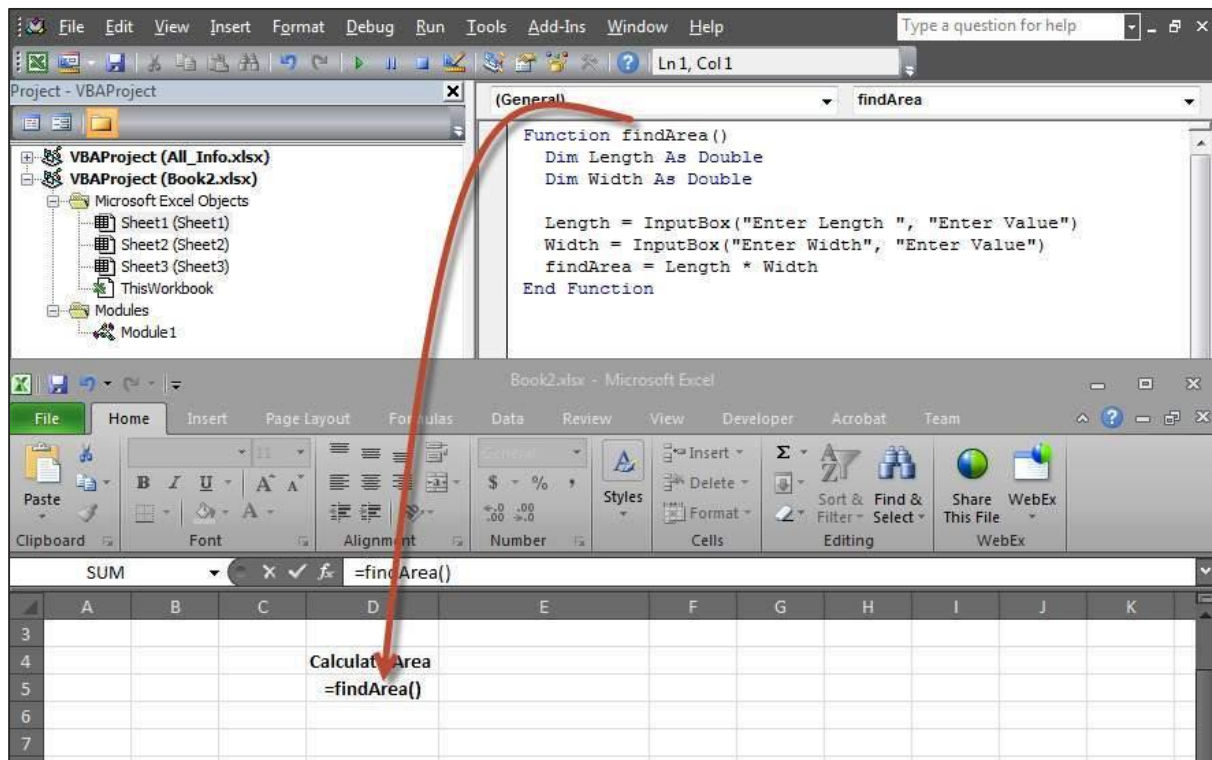
Let us calculate the area of a rectangle by getting values from the user at run time with the help of two input boxes (one for length and one for width).

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

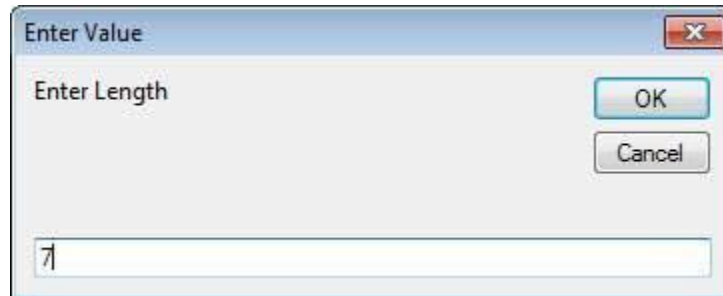
    Length = InputBox("Enter Length ", "Enter a Number")
    Width = InputBox("Enter Width", "Enter a Number")
    findArea = Length * Width
End Function
```

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.



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



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)

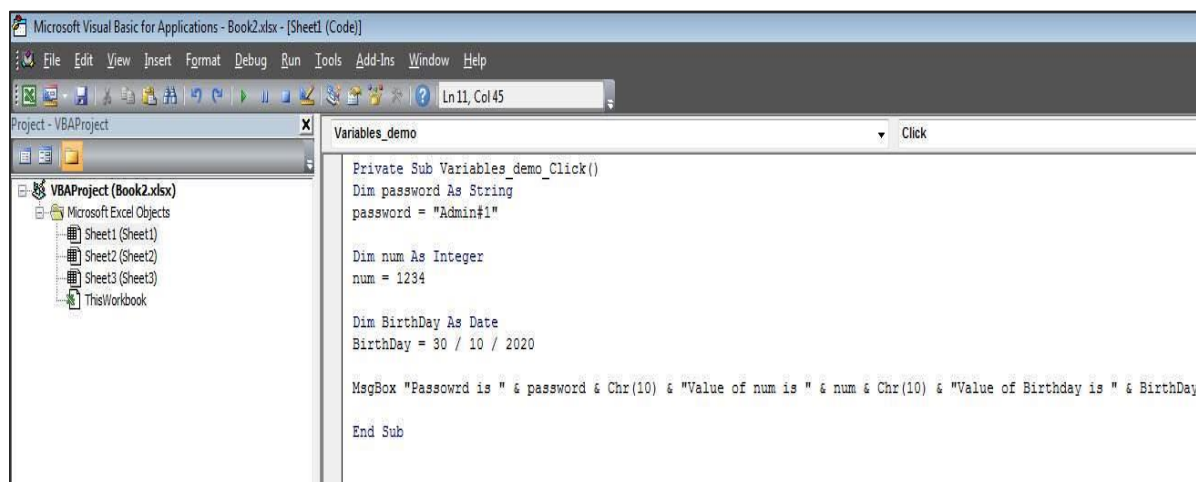
Non-Numeric Data Types

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")
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")
    End If
End Sub
```

```

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

End of ebook preview
If you liked what you saw...
Buy it from our store @ **<https://store.tutorialspoint.com>**