EXCEL VBA

FOR THOSE WHO NEVER DEALT WITH VBA

VIKAL JAIN

ABOUT THE AUTHOR

VIKAL JAIN

He is the founder of **VIKOM INSTITUTE**. He is an associate member of the Institute of Chartered Accountants of India. He is pursuing CFA level 1 from USA and holds a diploma in IFRS (India) from ICAI. He has also done B.Com (H).

EXCEL SKILLS AND COMPUTER EFFICIENCY

- Writing a book on "Excel Finance" and Excel Basic to advances
- **More than 20,50,000 likes on "EXCEL TRICKS" page on Facebook.**
- **More than 31,000 group members on Facebook**
- Knowledge of More than 250 formula's in Excel and Macro's in Excel
- VAST experience of MS Excel Open Office, 2003, 2007, 2010, 2013 and 2016
- Teaching Excel at **VIKOM INSTITUTE** and responding Excel queries across the world.
- Well Versed with MS – Word, PowerPoint, Microsoft Publishers, other Microsoft utilities, ERP Packages (Tally ERP9, Tally 9, 7.2 ,6.3), Web Surfing .
- Working experience of SAP

ABOUT THE VIKOM INSTITUTE

VIKOM INSTITUTEis **one stop solution of Excel Training**. It offers you standardized and customized templates, Online Excel training, Corporate Seminars in Excel, Excel E-books, free excel shortcuts files, daily tricks and so on.It is joint venture of various professionals who are experts in Finance, International Taxation, Software Development, Business Modeling, Financial Modeling, costing and so on.

MISSION

VIKOM has only one mission that is to increase effectiveness and efficiency of doing work. "**Removal of Redundancy"**, "**Doing smart work"**, "**Presenting better"**, "**Beyond copy & paste"** are the four main keys of the **VIKOM.**

DEDICATION

This short booklet is dedicated to my mother, my father and my sister who have motivated me, time to time and my FACEBOOK PAGE likers who have increased my excel experience.

ABOUT THE BOOKLET

This booklet is the summary of the **EXCEL VBA - Who Never Dealt With Vba Before** . We have used simple English language to make Excel simpler & easier for you. This book is open to all readers interested in learning BAISC EXCEL VBA, however, the book will produce more advantage for readers who have attended the seminars related to it.

Important Information

- ➔ You CANNOT take the print of booklet
- ➔ You CANNOT copy the contents of the booklet
- ➔ This book considers"**TREES ARE THE GREEN GOLD"**. So keep E-Reading.

FEEDBACK

For any suggested improvements in course content and booklet email us at info@vikominstitute.com or on vikominstiutute@gmail.com

Table of Contents

Chapter 1

UNDERSTANDING MACRO'S

MACROs

If you have a simple set of actions that you need to repeat several times over, you can automate these actions by excel and produce a macro, containing the code to repeat them. It is same as video recording, but instead of recording your activities it record your actions in EXCEL.

Benefits of MACRO recording

- Save time
- Perform complex task
- Automate repetitive task

Limitations of MACRO recording

Even macro has some good advantages it can't allow you to do these type of things:-

- Defined Constants, Variables and Arrays;
- If Statements;
- Loops;
- Calls to Built-In Functions or Other Procedures.

Record your First Macro

To record your macro you must follow these steps

Understand what do you want to record ?

Let suppose my boss have given me this data and said copy this data to every new sheet till 100 times. Now what I will do?

If MACRO will not be there, I will do this

I will copy my data and insert new sheet then paste it to the new sheet till 100 sheets will not over. This is so bulky task

Let's try Macro now to work with this task and build a automation

Open your Excel now

	A	B	C	D	E
1	535	785	618	632	592
2	589	795	892	624	501
3	912	704	827	804	915
4	543	683	583	822	624
5	679	641	845	579	663
6	762	598	808	812	810
7	908	526	583	724	685
8	551	912	942	510	510
9	720	582	592	724	550
10	692	738	654	973	500
11	551	833	864	732	718
12	992	820	945	836	810
13	610	664	824	845	961
14	806	856	925	955	698
15	663	523	680	884	788

Sheet2 Sheet1

Figure 1

See I have to copy this data till 100 times

Go to Macro recording now, in VIEW tab or in DEVELOPER tab

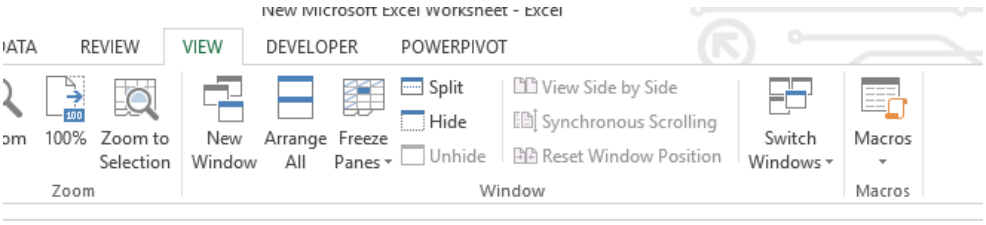


Figure 2

If your developer tab is not enabled then enable it first with these steps, in the Figure 2 I have already enable it.

Enable developer tab

Alternative # 1

Let's enable developer tab now.

Go to file menu then go to options

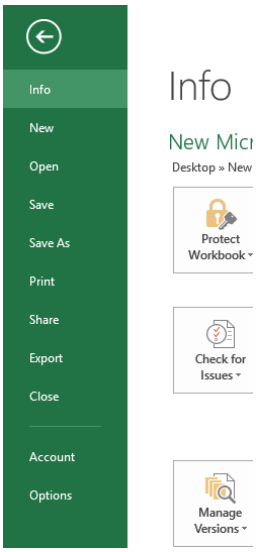


Figure 3

Then go to customize ribbon

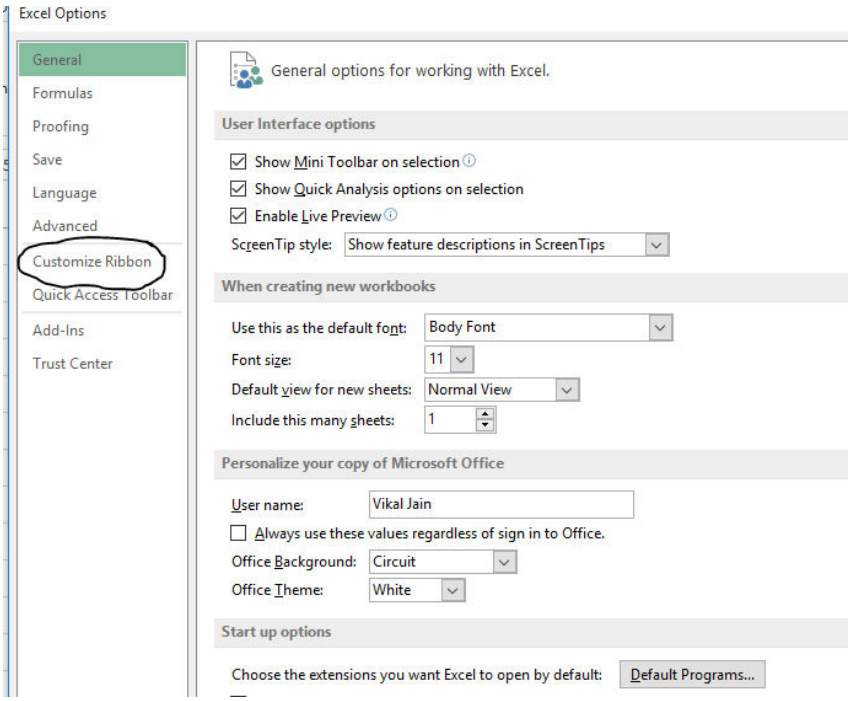


Figure 4

Now check your checkbox of the developer tab

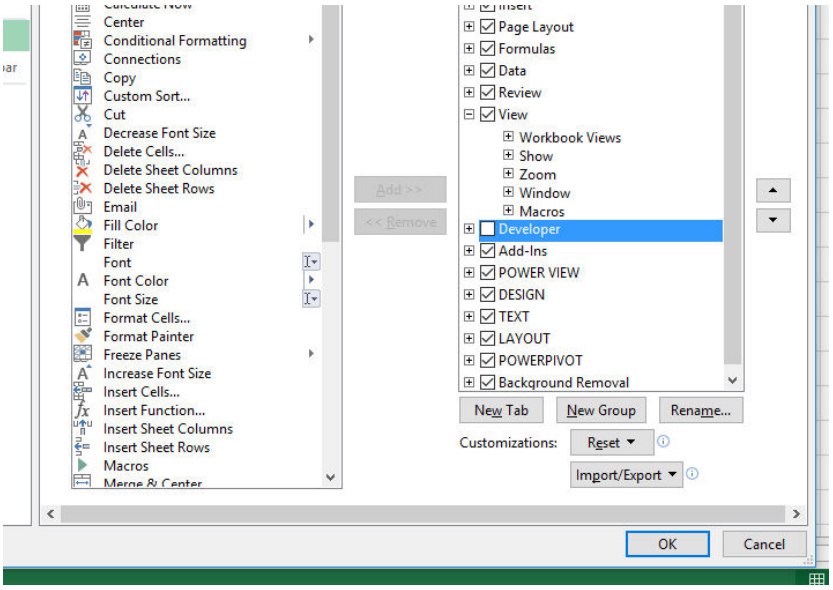


Figure 5

Now your developer tab is enabled

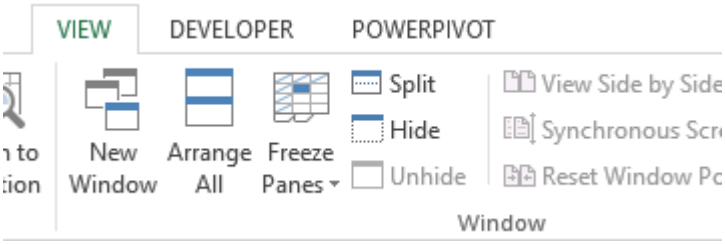


Figure 6

Alternative # 2

Click on any tab then press right click,

Then go to customize the ribbon

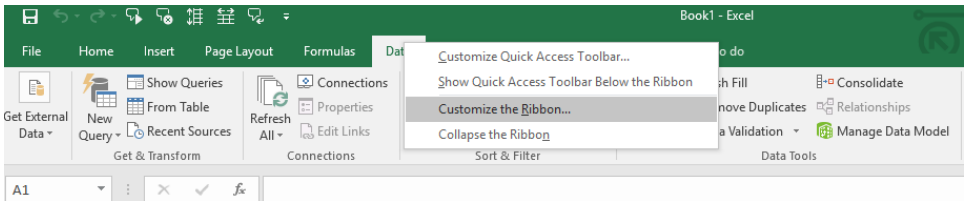


Figure 7

Remaining steps are same after the Figure 5

You can go to macro recording from developer tab also

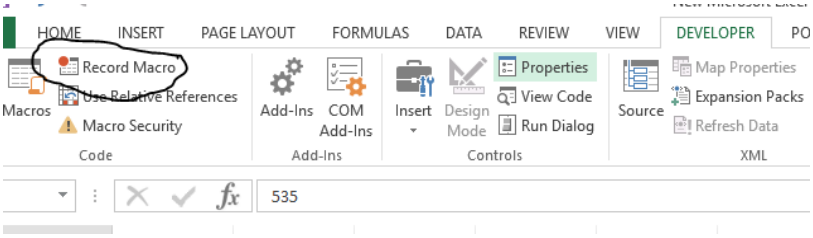


Figure 8

Now go to MACRO RECORDING and select record Macro, when you will select record macro this dialogue box will open

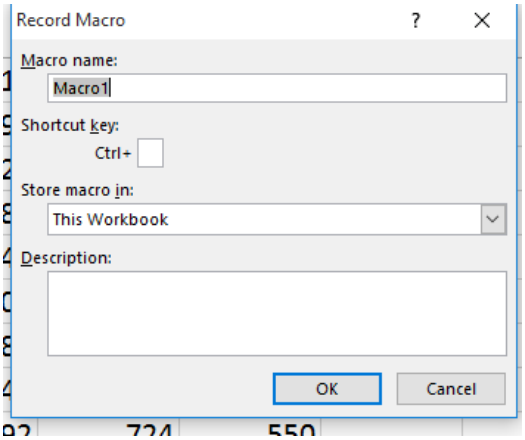


Figure 9

This dialogue box will ask your macro name, shortcut key for the Macro and description, without chaging our macro name and Let's assign a shortcut key "Ctrl + Q"

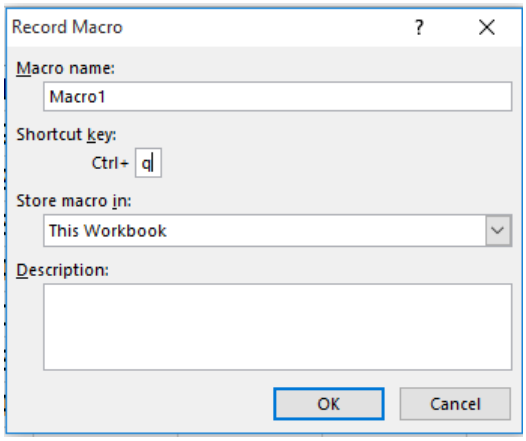
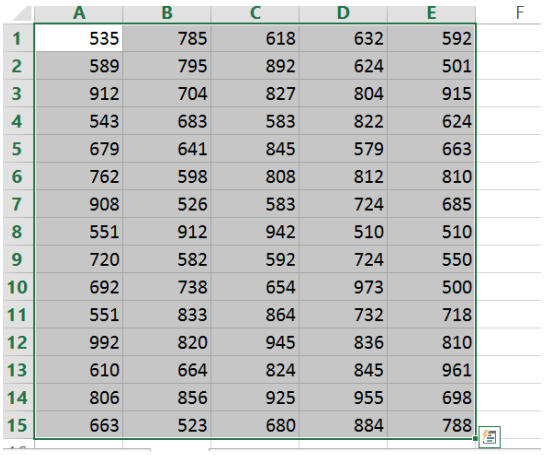


Figure 10

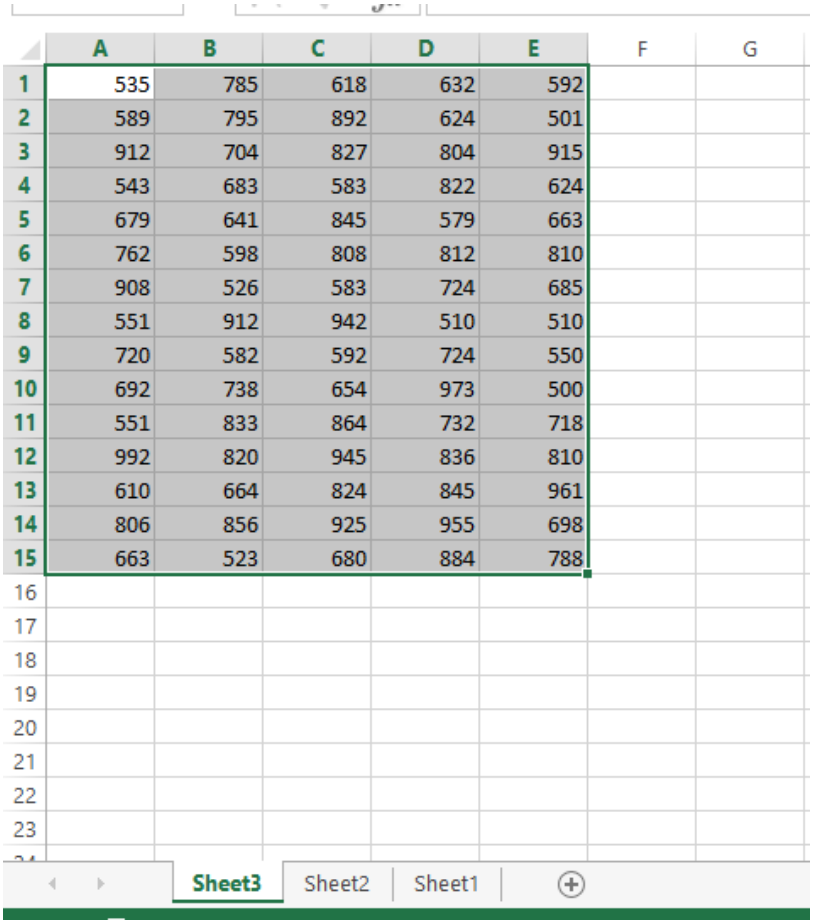
Record your task now, I am copying my data, inserting a new sheet and paste it in this task



	A	B	C	D	E	F
1	535	785	618	632	592	
2	589	795	892	624	501	
3	912	704	827	804	915	
4	543	683	583	822	624	
5	679	641	845	579	663	
6	762	598	808	812	810	
7	908	526	583	724	685	
8	551	912	942	510	510	
9	720	582	592	724	550	
10	692	738	654	973	500	
11	551	833	864	732	718	
12	992	820	945	836	810	
13	610	664	824	845	961	
14	806	856	925	955	698	
15	663	523	680	884	788	

Figure 11

Now when you have completed your all tasks then go to stop recording and press your assigned shortcuts my shortcut is CTRL + Q



	A	B	C	D	E	F	G
1	535	785	618	632	592		
2	589	795	892	624	501		
3	912	704	827	804	915		
4	543	683	583	822	624		
5	679	641	845	579	663		
6	762	598	808	812	810		
7	908	526	583	724	685		
8	551	912	942	510	510		
9	720	582	592	724	550		
10	692	738	654	973	500		
11	551	833	864	732	718		
12	992	820	945	836	810		
13	610	664	824	845	961		
14	806	856	925	955	698		
15	663	523	680	884	788		
16							
17							
18							
19							
20							
21							
22							
23							

Sheet3 Sheet2 Sheet1

Figure 12

See in the above picture my task has been performed by the macro and a new sheet is inserted.

Now press 100 times Ctrl + Q your boss will impress.

Save your Macro Workbook

After recording, whenever you will save or simply press Ctrl + S from your keyboard this dialogue box will open

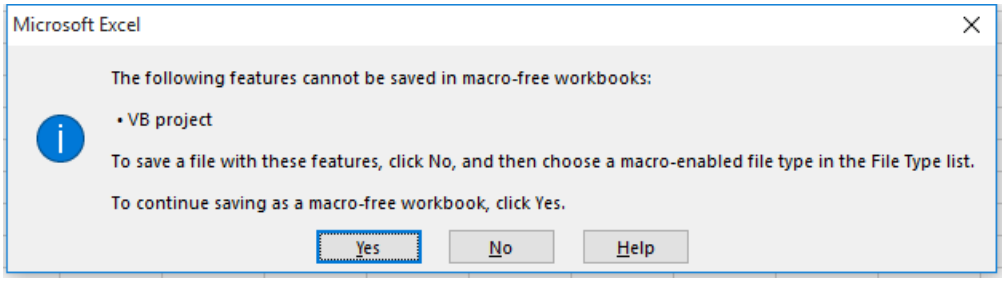


Figure 13

This dialogue box is saying that your workbook is not macro enabled workbook do you want to enable your macro, to save with current features press YES otherwise press no.

To save we will press No since the workbook don't have MACRO

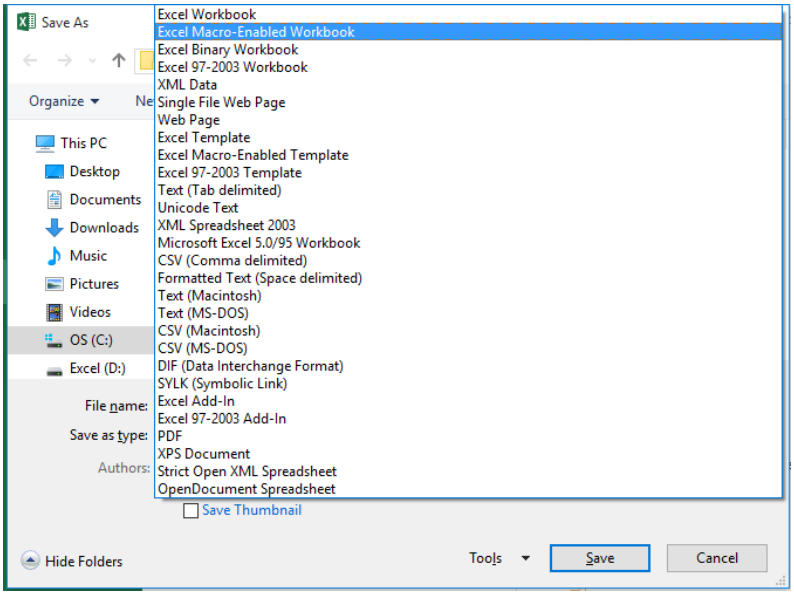


Figure 14

With the help of the above image you can see that there are lots of type to save your excel, Save your workbook as ***MACRO ENABLED WORKBOOK***.

Now when your workbook will save the icon of the workbook will also change.

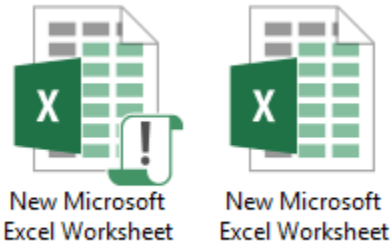


Figure 15

See the left side workbook is the ***MACRO ENABLED WORKBOOK*** and the right side workbook is ***NORMAL WORKBOOK***. You can observe the difference now.

Open your Macro Workbook

To open your macro workbook you can double click on your workbook and simply open it. But you have to enable first before starting the work on the MACRO ENABLED WORKBOOK

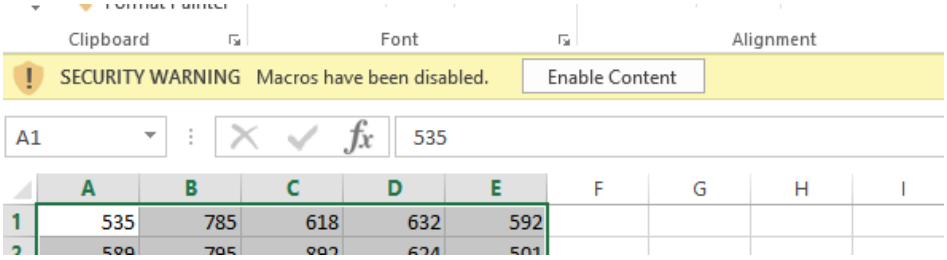


Figure 16

Analysis: - Now question arise is that why MACRO ENABLED WORKBOOK? Why not NORMAL WORKBOOK for MACRO?

When you will not save your workbook as MACRO ENABLED WORKBOOK then your macro will be gone after closing the workbook. You have to work again on your MACRO with every opening of the workbook.

Author Note: in the above case we have pressed CTRL + Q 100 times, since macro cannot handled looping, if we can loop this code then we have to press CTRL + Q just once.

View Your Recorded Macro Code

To view macro code you will have to open VBA editor first. You can open VBA editor by going to developer menu and go to view code. Otherwise simply press ALT + F11

When you will open your VBA editor this window will open.

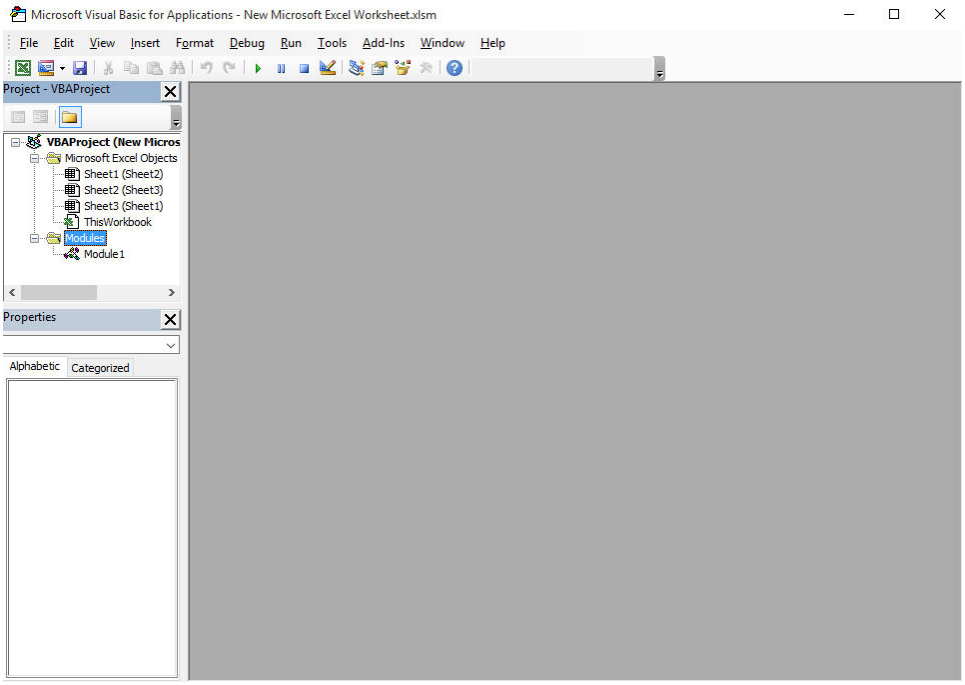


Figure 17

This window is also called as VBA editor. This window helps you to do write yourprogramming and functions in Excel or build an automation.

Now to view your code you have to go to MODULE then go to Module 1, which can be seen in the VBA projects option which is on top left upper side of your screen.

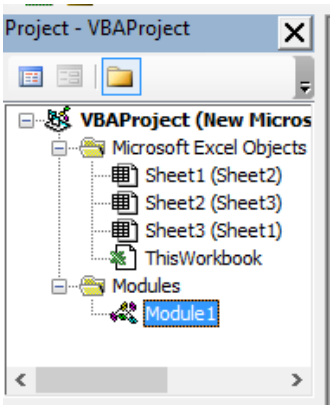


Figure 18

As and when you will click on module 1 then your code will look like this.

```
Sub Macro1()
'
' Macro1 Macro
'
' Keyboard Shortcut: Ctrl+q
'
    Range("A1:E15").Select
    Selection.Copy
    Sheets("Sheet1").Select
    Sheets.Add
    Range("A1").Select
    ActiveSheet.Paste
End Sub
```

Figure 19

See this code is also showing the shortcuts of the MACRO given by us in the process ofrecording of MACRO.

The understanding of this code is discussed in the later chapters.

View Your All Macros

You can also view your all macro after writing or recording. There are alternative methods to open your MACROs list

First go to VIEW OPTION and select VIEW MACRO

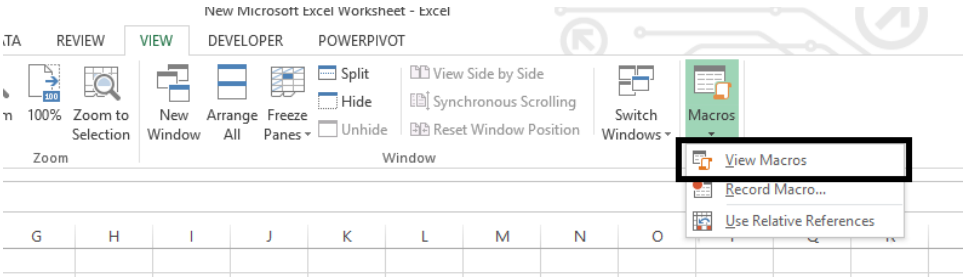


Figure 20

Second go to DEVELOPER OPTION then go to MACROs

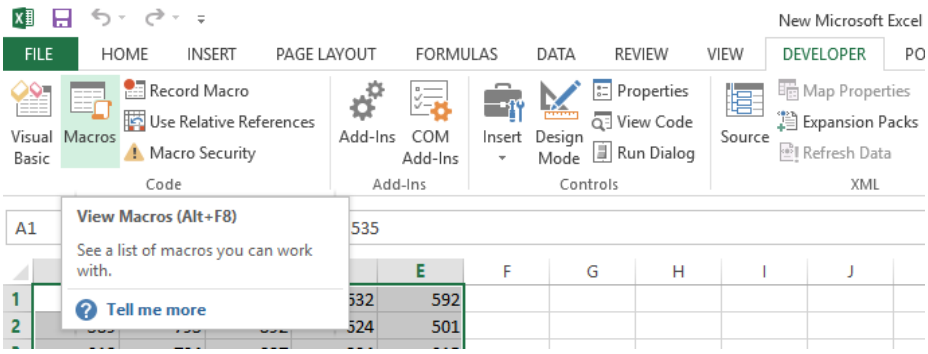


Figure 21

And the third just simply press ALT + F8 now your MACRO list will open like this

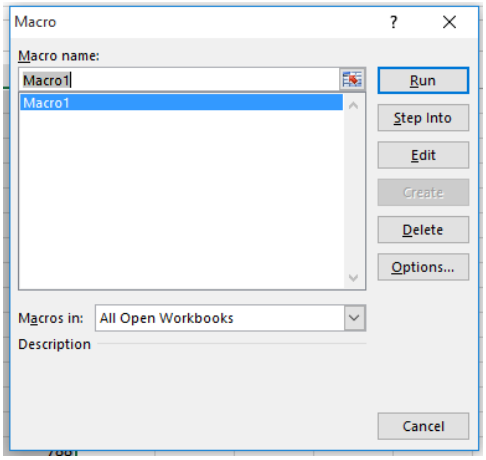


Figure 22

Run, Edit, Delete your MACROs

To run your MACRO you can simply click your shortcut which you have assigned in the process of MACRO recording. My shortcut was CTRL + Q

Or you can press ALT + F8 this will open the complete list of your MACROs then press RUN option.

In this dialogue box you have a option to EDIT your macros, RUN your macros, DELETE your macros, STEP INFO which will open your code directly, or OPTION menu which is for your general information of your MACRO like the shortcut assigned, description given and so on.

Chapter 2

GETTING STARTED

What is excel VBA

VBA stands for Visual Basic for Applications. It's a programming language that enables you to control just about everything in Excel. You'll learn how to create Macros that can be run from things like a button on a spreadsheet, the Excel Ribbon - in fact, lots of places. Learning Excel VBA will enable you to do a lot more with the software than you can via the normal spreadsheet view.

There are lots of advantages of using EXCEL VBA like

- Making Templates
- Building your own functions
- Control other utilities like MS- Word, Power Point and so on
- Customize the whole Excel
- Control fonts, working with hidden sheets and
- So on…

Author Note: before learning of VBA you are totally dependent on EXCEL to do your work, but after learning this VBA, EXCEL will depend upon you. ☺☺

VBA and VB is different

VB abbreviated from Visual Basic and VBA known as Visual Basic for Applications, have both originated from Basic. In this way, they share fundamental similarity. Visual Basic got popularity as a third generation that supports an integrated development environment introduced by the leading software company, Microsoft. From user point of view, Visual Basic is considered one of the relatively 'easy to learn and use' languages for beginners, because it enables the RAD of graphical user interface applications, provides access to databases using data access objects and supports the creation of ActiveX controls and objects.

VBA is a subset of VB which runs inside one of the office applications. As a result VBA inherits the current Office object library and application instance by default and any references that are included. However in VB you have to create the application instances if you need to manipulate one or more of the Office application objects.

IN SIMPLE TERMS the difference between VBA and VB is

There is not much difference between VBA and VB as such. VBA may be thought of cut down version of VB and is primarily used for programming in Office Application such as Word, Excel, Outlook etc. However, VB has wide scope for programming.

Open VBA in Excel

To open your VBA you can simply press ALT + F11 or just go to DEVELOPER TAB and then go to VISUAL BASIC.

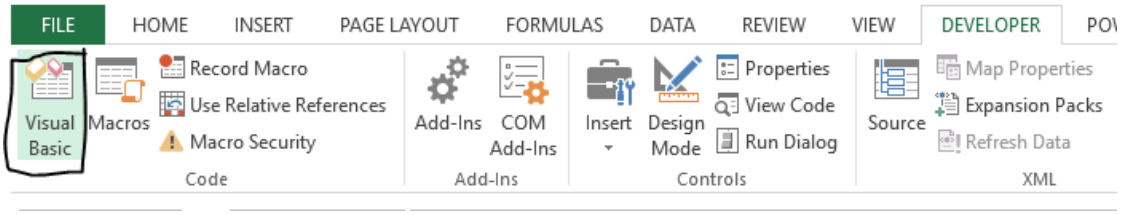


Figure 23

After pressing the Visual Basic or Alt + F11 this window will open

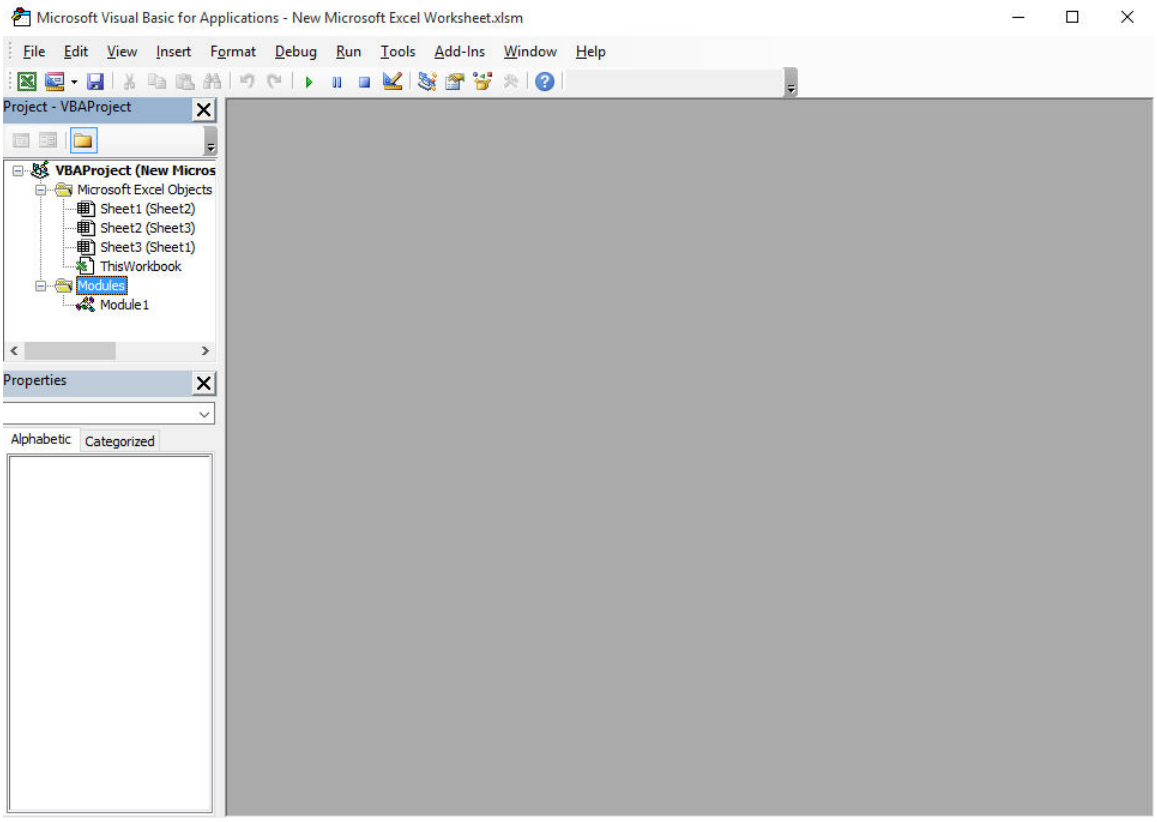


Figure 24

Understanding the VBE

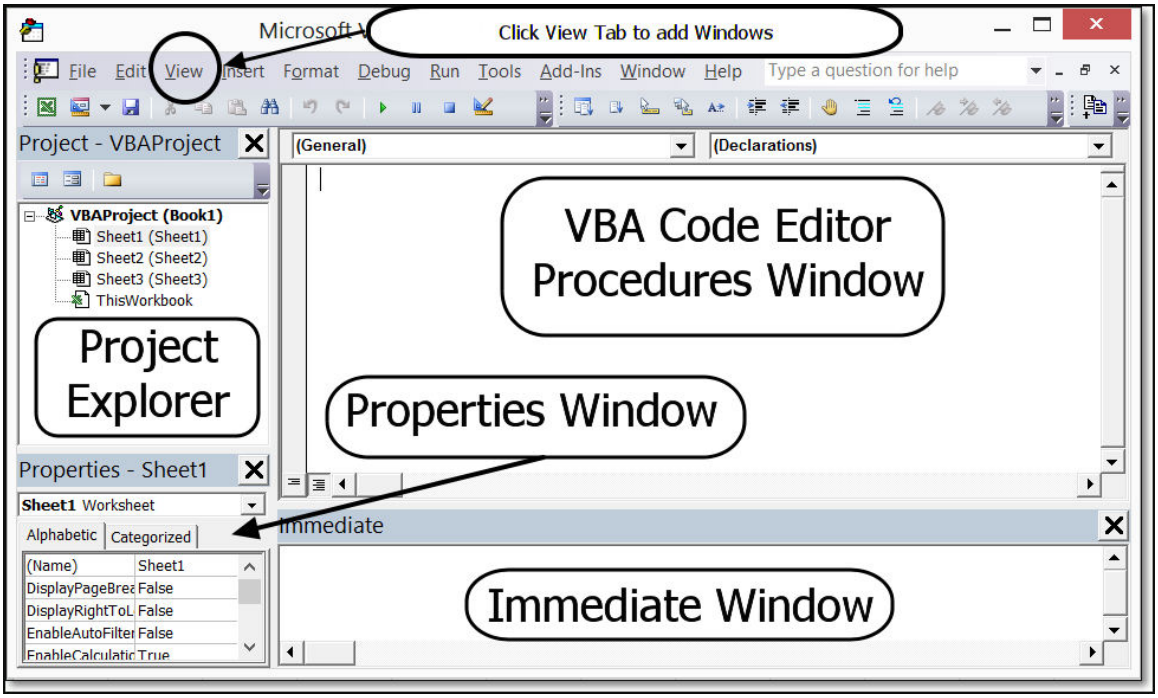


Figure 25

The Project Explorer

The Project Explorer lists any open workbooks and add-ins that are loaded. If you click the + icon next to the VBA Project, you will see that there is a folder with Microsoft Excel objects. There can also be folders for forms, class modules, and standard modules. Each folder includes one or more individual components.

Right-clicking a component and selecting View Code or just double-clicking the components brings up any code in the Programming window. The exception is userforms, where double-clicking displays the userform in Design view.

To display the Project Explorer window, select View, Project Explorer from the menu, and then press Ctrl+R or click the Project Explorer icon on the toolbar.

This pane can show Microsoft Excel objects, userforms, modules, and class modules.

To insert a module, right-click your project, select Insert, and then choose the type of module you want. The available modules are as follows:

- **Microsoft Excel objects**—by default, a project consists of sheet modules for each sheet in the workbook and a single ThisWorkbook module. Code specific to a sheet such as controls or sheet events is placed on the corresponding sheet. Workbook events are placed in the ThisWorkbook module.
- **Forms**—Excel allows you to design your own forms to interact with the user.

- **Modules**—when you record a macro, Excel automatically creates a module in which to place the code. Most of your code will reside in these types of modules.
- **Class modules**—Class modules are Excel's way of letting you create your own objects. They also allow pieces of code to be shared among programmers without the programmer needing to understand how it works.

The Properties Window

The Properties window enables you to edit the properties of various components such as sheets, workbooks, modules, and form controls. The Property list varies according to what component is selected. To display this window, select View, Properties Window from the menu, press F4, or click the Project Properties icon on the toolbar.

The Immediate Window

The VBA Immediate Window is an awesome tool that allows you to get immediate answers about your Excel files, and quickly execute code. It is built into the Visual Basic Editor, and has many different uses that can be very helpful when writing macros, debugging code, and displaying the results of your code. To open immediate window press CTRL + G or click on VIEW option then go to Immediate Window.

The VB editor

VBA editor is the main part of your VBA since its only allow to write your program. Note that the drop down list in the upper right hand corner shows your macro name.

Way to express

Before starting VBA this is to tell you that the reading of this book is so easy since we have express Excel VBA and Excel simultaneously. The upper section of the image will represent the EXCEL and lower side of the image will represent EXCEL VBA.

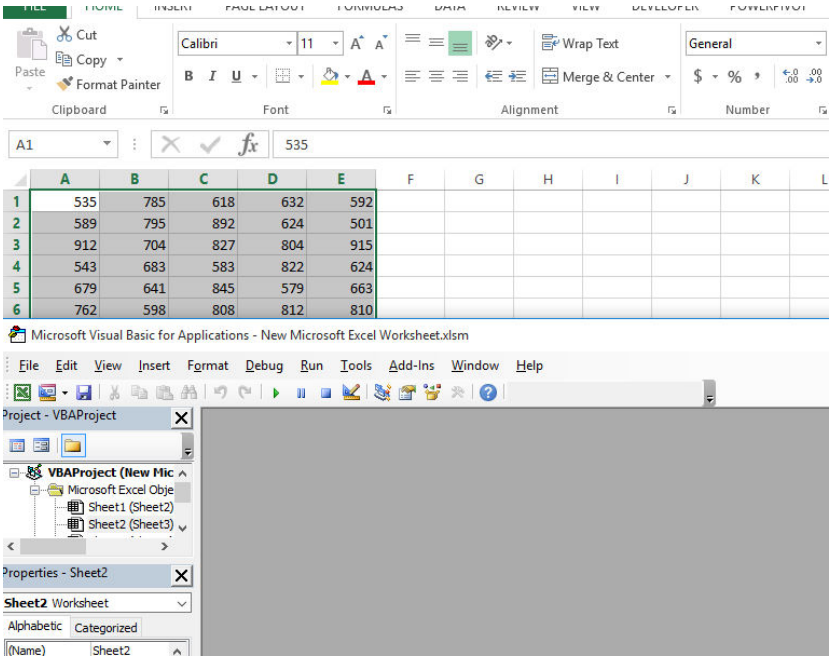


Figure 26

VBA is purely based on Objects oriented programming

Reading is Optional

VBA is based on Objected oriented language which is a computer programming, however we will not discuss the computer programming in the book, and we will try to correlate the Excel functionalities with Excel VBA. If you want to know more the object oriented language then this is the text.

Source http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html

Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure.

In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects.

The Basics: Object Oriented Programming Concepts

If you are new to object-oriented programming languages, you will need to know a few basics before you can get started with code. The following Webopedia definitions will help you better understand object-oriented programming:

Abstraction: The process of picking out (abstracting) common features of objects and procedures.

Class: A category of objects. The class defines all the common properties of the different objects that belong to it.

Encapsulation: The process of combining elements to create a new entity. A procedure is a type of encapsulation because it combines a series of computer instructions.

Information hiding: The process of hiding details of an object or function. Information hiding is a powerful programming technique because it reduces complexity.

Inheritance: a feature that represents the "is a" relationship between different classes.

Interface: the languages and codes that the applications use to communicate with each other and with the hardware.

Messaging: Message passing is a form of communication used in parallel programming and object-oriented programming.

Object: a self-contained entity that consists of both data and procedures to manipulate the data.

Polymorphism: A programming language's ability to process objects differently depending on their data type or class.

Procedure: a section of a program that performs a specific task.

Advantages of Object Oriented Programming

One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

OOPL - Object Oriented Programming Languages

An object-oriented programming language (OOPL) is a high-level programming language based on the object-oriented model. To perform object-oriented programming, one needs an object-oriented programming language. Many modern programming languages are object-oriented, however some older programming languages, such as Pascal, do offer object-oriented versions. Examples of object-oriented programming languages include Java, C++ and Smalltalk.

The First OOPL

Simula, developed in the 1960s at the Norwegian Computing Center in Oslo, is considered to be the first object-oriented programming language. Despite being first, Smalltalk is considered to be the only true object-oriented programming environment and the one against which all others must be compared. It was first developed for educational use at Xerox Corporation's Palo Alto Research Center in the late 1960s and released in 1972.

Methods Vs Properties

METHODS

Methods are the actions that can be performed by an Objects or on an Object. In the above Hose example, paintaing is a Method, building a new room is a method.

```
Sub sbExampleRangeMethods()

Range("A1").Select

Selection.Copy

Range("B5").Select

ActiveSheet.Paste

End Sub
```

Author Note: In short, method is the function or verb that you assign to your code, like Range(“A1”).select here selecting is the method, ActiveSheet.Paste here pasting is the method.

PROPERTY

Properties are the characteristics of an Objects which can be measured and quantified, in the above example House is having properties like Width, Height, Color, etc...

```
Sub sbExampleRangeProperties()

Range("A1").Value = 25

Range("A1").Interior.ColorIndex = 5

End Sub
```

Author Note: In short, properties is the condition that you give to your code like Range("A1").Value = 25 here value is the property. Since condition is basically assigned by equal to sign "=" therefore property is also assign by the same sign.

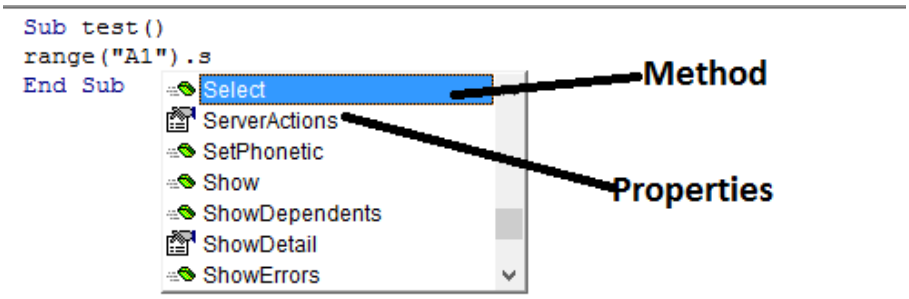


Figure 27

Author Note: in the above image method is showing by box sign and property is showing by hand sign.

OBJECT BROWSER will help you for looking the properties and methods (F2 is the shortcut)

Object browser

OBJECT BROWSER shows the list of all methods and properties, like the function list in excel. To open object browser you must press F2 from your keyboard.

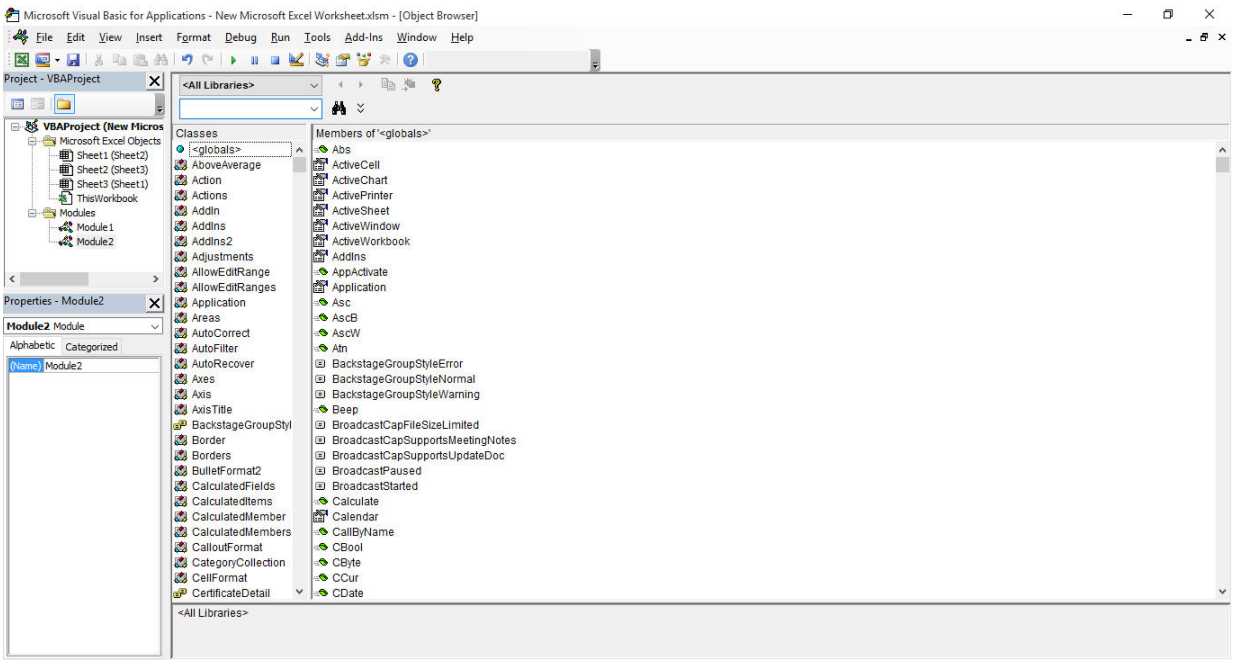


Figure 28

Excel Dot Notation

Excel VBA uses dot notation to separate the various things you can access and manipulate with the programming language. Dot notation is hierarchical, and usually starts with an object. (In Excel, an object is the thing you're trying to manipulate, such as a worksheet.) After the object, you type a dot. You then specify what you want to do with this object, or what you want to manipulate. The doing is called a method. The manipulating is done via properties or parameters.

If all this is confusing, let's try and clear it up.

Think of a Girl. This is an object. We can notate it like this:

Girl

OK, all very simple so far. But you'll need some more information if you were going to buy a television. One thing you may want to know, what is the height of that girl,. To add a size property, you'd do this:

Girl.height

You'd want this to equal something, though, so add an equal sign and a size:

Girl.height = "5.5 feet"

We now have an object (Girl) and a property (the Height). We also have a value for the size (5.5 feet).

If we wanted to check the ring of this girl then we'd be doing something (ring). We can call this "doing" a method. It is a method of the Girl:

Girl.Ring

Methods can come with extra settings, called parameters. A parameter of the offering a ring could be RINGTYPE. The Ringtype would then come with its own values (Gold, Silver, diamond etc). We could represent all this as follows:

Girl.Ring RingType:=Silver

We have a space between the method (Ring) and the parameter (RingType). The value for the parameter comes after a colon and equal sign (:=), with no spaces in between.

We could add more parameters for Ring. For example, we could have a shape parameter, and a extra pebbles:

Girl.Ring RingType:=Silver Shape:=round Extrapebbles:= No

Notice that we've used a space to separate the three parameters and their values.

Shortcuts summary

Table 1

Shortcuts	Particulars
Alt + F11	Open VBA
Alt + F8	Open MACROlist
Ctrl + R	Open Project Explorer
F4	Properties Window
Ctrl + G	Immediate Window
F2	Object Browser

Chapter 4

VBA Operators (Use of Symbols in VBA)

The Line continuation operator "_" (underscore)

If you are writing long piece of code, to make it easier to read you can use "_" (underscore).

For example

```
Sub Myvalue
Range("A1").Value > 300 AND _
Range("B1").Value > 900
End SUB
```

The Parentheses: "()"

The Parentheses is used in various ways in EXCEL and VBA, however it is generally used to define the arguments of the functions or mathematics calculations

For example

```
Range("A1:B10").offset(1,0).select
10 + (8*9)
```

The Comma " ,"

The comma is used to separate the variables used in the group

For example

```
Dim myValue1 as Integer, myValue2 as Integer, myValue3 as Integer
```

Colon and Equal to ":="

Colon and equal to is used to represent the arguments of the Function

For example

The syntax of Add property is expression .Add(Before, After, Count, Type)

So you can represent your arguments as follows

```
Worksheets.add after:=worksheets("Sheet1") Count:=2
```

It means we are saying to VBA, that please insert 2 sheets after the Sheet1 in EXCEL.

The Double Quotes: ""

The double quotes is used to represent any text in EXCEL and VBA.

For Example EXCEL and VBA cannot understand this

Range("A1:B10").value = Vikal

VBA will treat VIKAL as variable if already defined otherwise it will produce some sort or errors.

But in Range("A1:B10").value = "Vikal" Vikal will be treated as text.

The Colon Operator ":"

The Visual Basic language allows you to write as many statements as necessary on the same line. When doing this, the statements must be separated by a colon. Here is an example:

SUB Values()

Firstname = "Vikal" : Lastname = "Jain"

End Sub

Concatenate Operator "&"

It is used to concatenate the text in EXCEL and VBA

For Example

Cell A1 Value is Vikal and Cell B1 value is Jain then you can get the whole name in cell C1 in following manner

=A1&" "&B1 (here" " added by us to add the space between the first name and the last name)

Carriage Return-Line Feed (vbCrlf)

It react as Alt + Enter in VBA as we do in EXCEL.

Author Note:

vb: Visual Basic

Cr: Carriage Return

Lf: LineFeed

Arithmetical operators "+, -, /, ^, *'

They are used in the same manner as; they are used in EXCEL to perform the mathematical operations.

Chapter 5

RANGE & CELLS

What is Range Object

The RANGE object is basically used for selecting the Range in Excel. Whenever you select the range in excel you use ctrl, shift or arrow keys combinations.

Author Note: Range means more than one cell or group of cells in excel either considered vertically or either horizontally. (In excel single cell is also a part of the range but it is generally known as cell.)

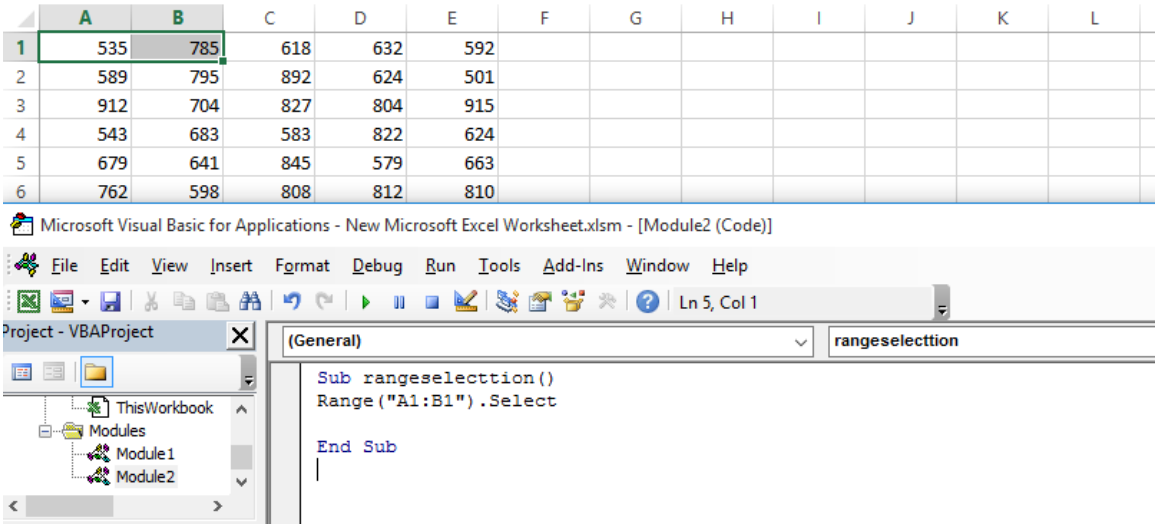


Figure 29

Let's understand the code now

Sub rangeselecttion()

Range("A1:B1").Select

End Sub

Before understanding the code, Let's the code fragmented into parts.

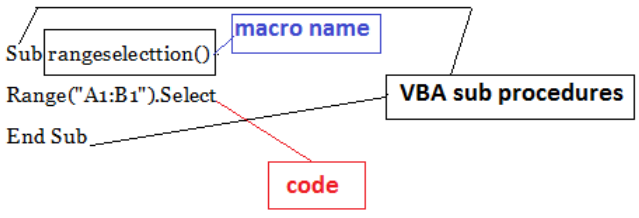


Figure 30

VBA sub Procedures

The VBA editor recognizes a Sub procedure, because the commands are positioned between the following start and end commands:

SUB

.

.

.

END SUB

Whenever you want to write your code in VBA , then you must write your code between the SUB and the END SUB. Only then your code will get executed.

SUBROUTINE will help your VBA to understand your program and it will differentiate the simple English with the program.

MACRO NAME

rangeselecttion()

Macro Name can't contain spaces, though. But you can type an underscore. So this is OK:

SubRange_Selection()

But this is not:

Sub Range Selection()

Take note of the following when coming up with a name for your Subroutines:

They can't start with a number, only alphabetical characters (you can have numbers elsewhere in your names, though)

You can't have full stops/periods in them

You can't use any of the following characters anywhere in your names: #, $, %, &,!.

Your Code

Range("A1:B1").Select

Your program always consist between your subroutine. We have started the Range object which is for selection for range. the syntax of the range object is as follows

Range(Cell1,[Cell2]) as range

Cell1 is the starting position of the range and cell2 is the ending position

“As Range” is denoting that this function will react as range not as a cell or formatting or something else

Let’s try to understand our code

Range(“A1:B1”).select

To denote a range we have range function and A1 to B1 is your range but we have put our range in “” (double quotes) or express the gap between the range with : (colon). But why so ?

A1 and B1 is the Name or the text of your cells and “A1:B1” is the name of your range so the starting position of your range is A1 and ending position of the cell is B1.

You can also express your range like “A1”,”B1”in the range object. Since it require starting position and the ending position only.

So, back to Excel. In the VBA programming language, you'll uses these object, methods, properties and parameters a lot.

In our example we have used Range(“A1:B1”).select, it means we are saying to VBA that please select our range which is “A1:B1”

Execute your Code

To execute your code in VBA you have to simply press F5 or go to RUN menu in you VBA then select RUN SUB/USERFORM F5

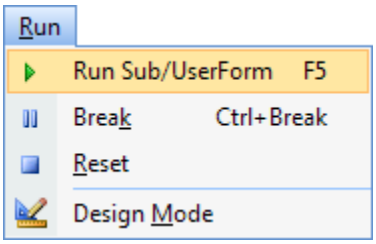


Figure 31

Or simply click play button which is in your VBA toolbar

Figure 32

When you will run your code then till will happen

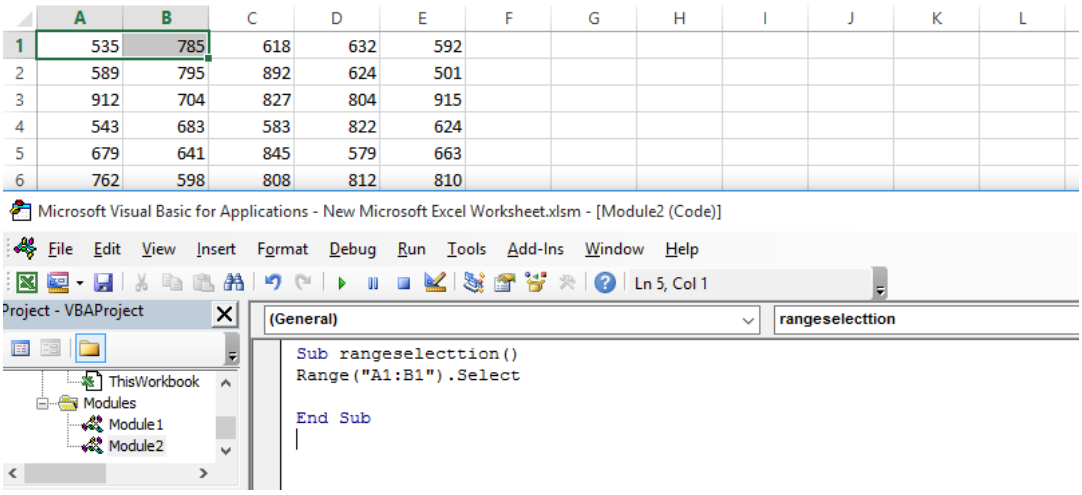


Figure 33

See in excel Range A1:B1 is selected

Name your ranges

Author Note: How do you name your range in Excel Name the ranges means. There are two ways to name your ranges first is go to Name Box and type your name or

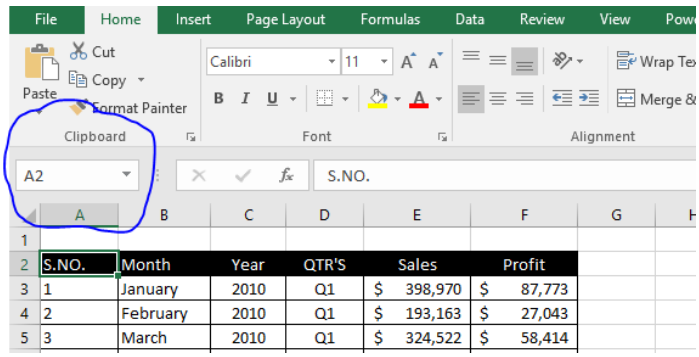


Figure 34

Go to DEFINE NAME in FORMULAS tab

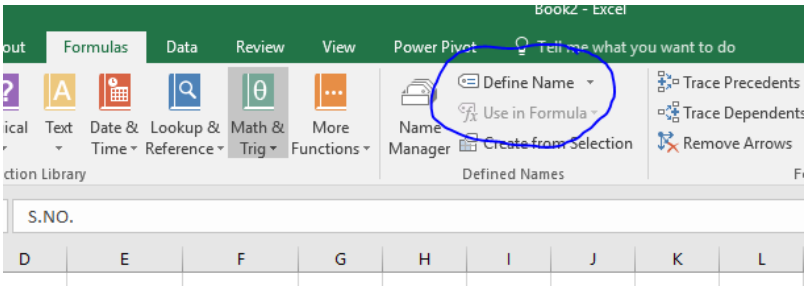


Figure 35

Here is my data in excel which I have name it as “Myrange”

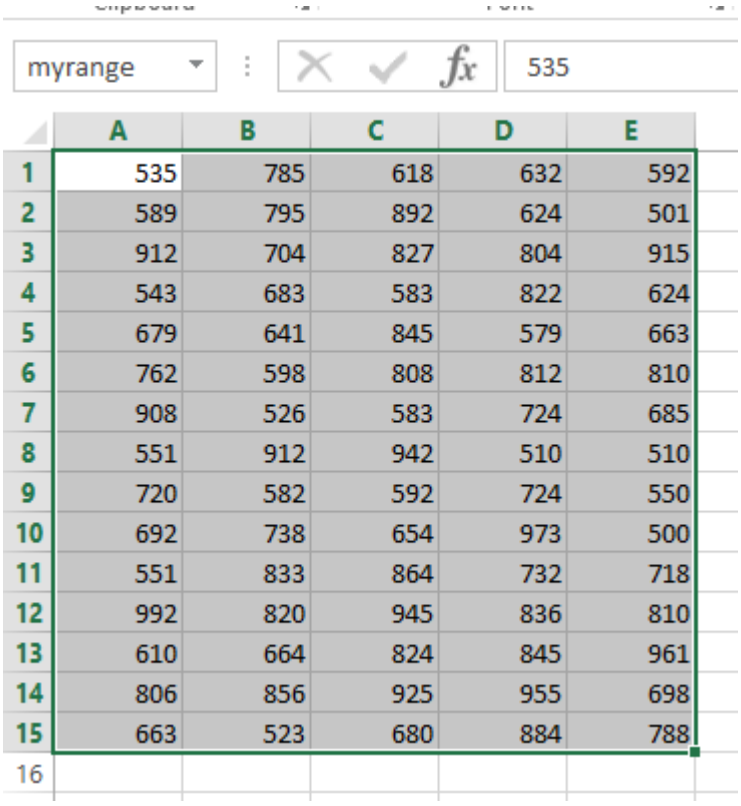


Figure 36

Let's select this range through EXCEL VBA now

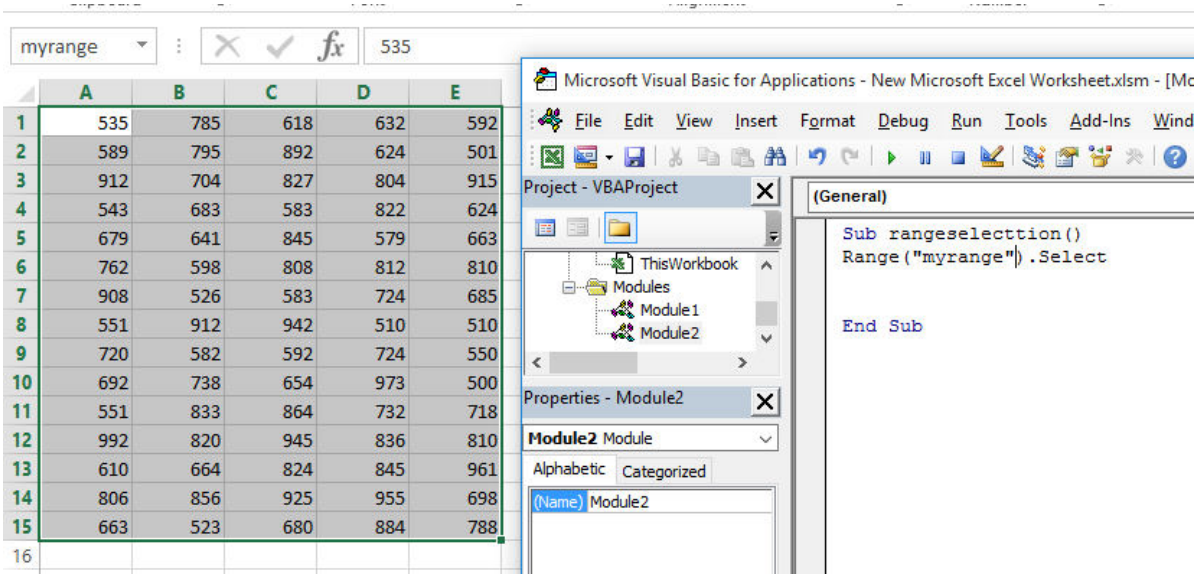


Figure 37

See this time I have typed "Myrange" in the range object in "" double quotes. This is working properly.

Let's forget selection let's enter some data now

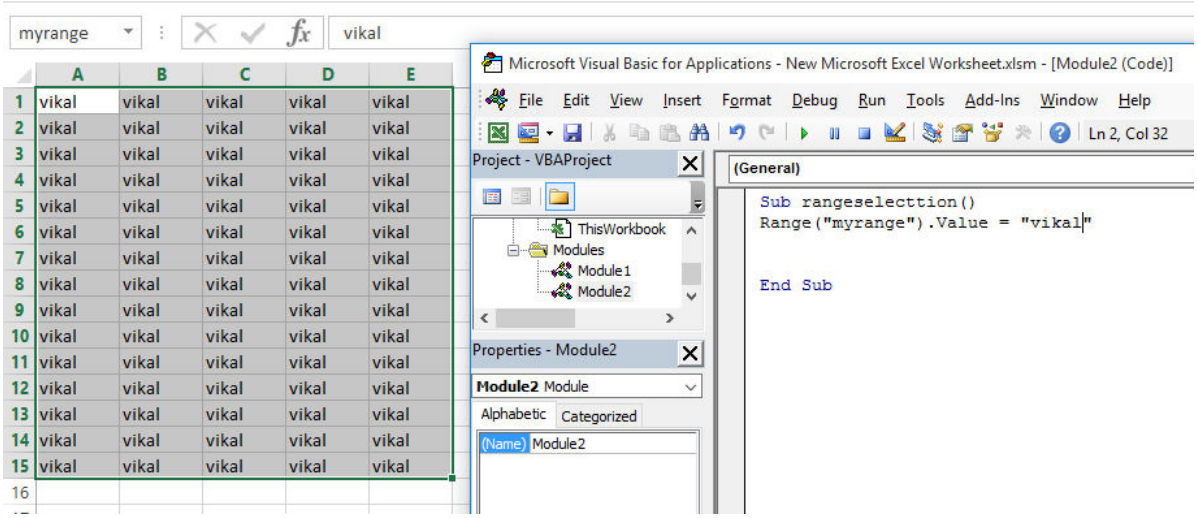


Figure 38

In the above pictures we have little bit change our code instead of typing Range("myrange").select (METHOD)we have typedRange("myrange").value = "vikal" (PROPERTY)

And all cells values from "A1:E15" i.e. my range values has been changed to "vikal". This is the difference between the method and property. Property allow you to change the contents of cells since you have to type = into every properties but method does not allow you to change your contents in cells.

In the code we have said the VBA that VBA please change my range value to "vikal" and it did. Yohoooo!!!!!

You can select single cell with range object also like in the following example we are doing.

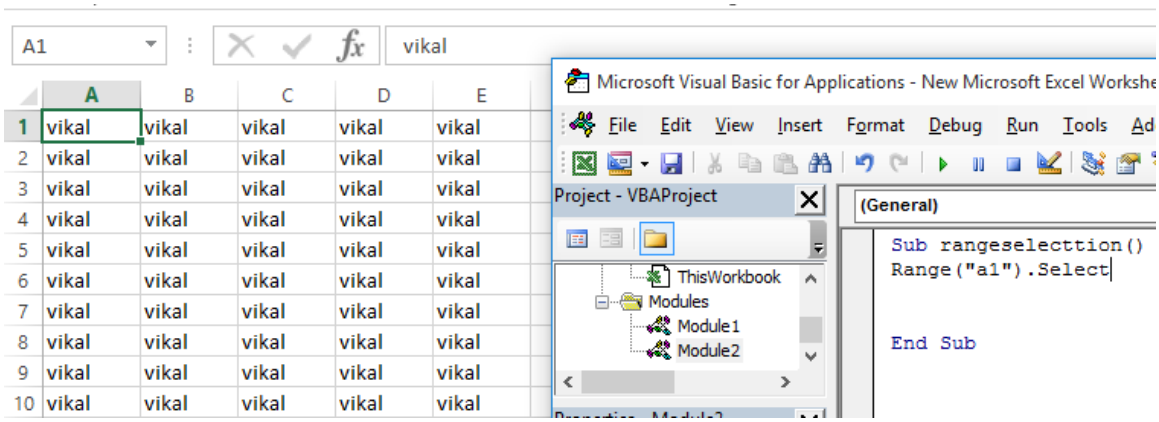


Figure 39

It is not usable to select a single cell with RANGE OBJECT, for this you must use CELLS OBJECT.

Activecell property

This is the awesome property of the VBA that it can allow you to select your range from the Active cell and where you want to be. Let's suppose our active cell is C3 which means our cursor is on cell C3, now select active cell to cell A1.

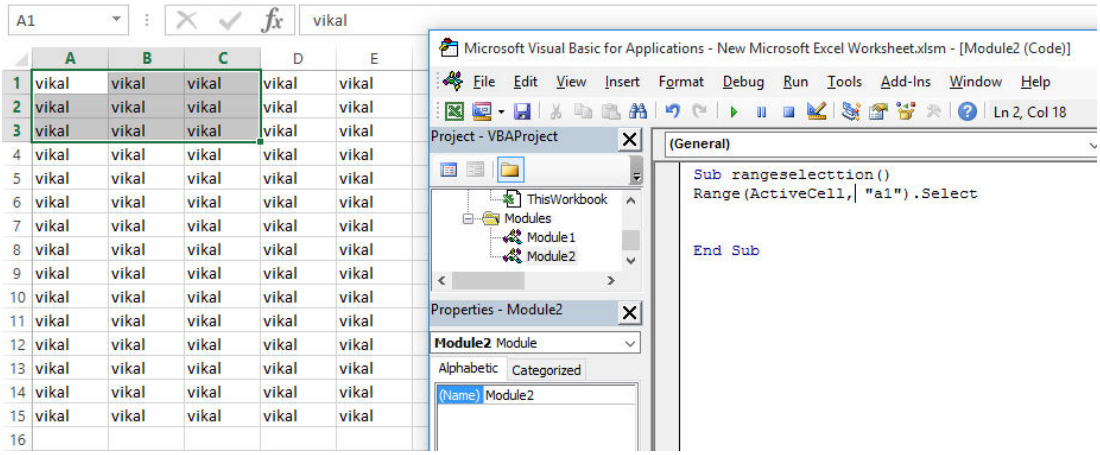


Figure 40

See this time we have select cell1 as active cell and cell2 as A1 and range object selected the range from A1:C3 since our active cell was C3

Cells object

Cell object represents single cell in excel. Use Cell(row, column), where row is the row number and column is the column number, or Cells(index), where index is the index number, to return a Cell object. The following example applies shading to the first cell in the second row.

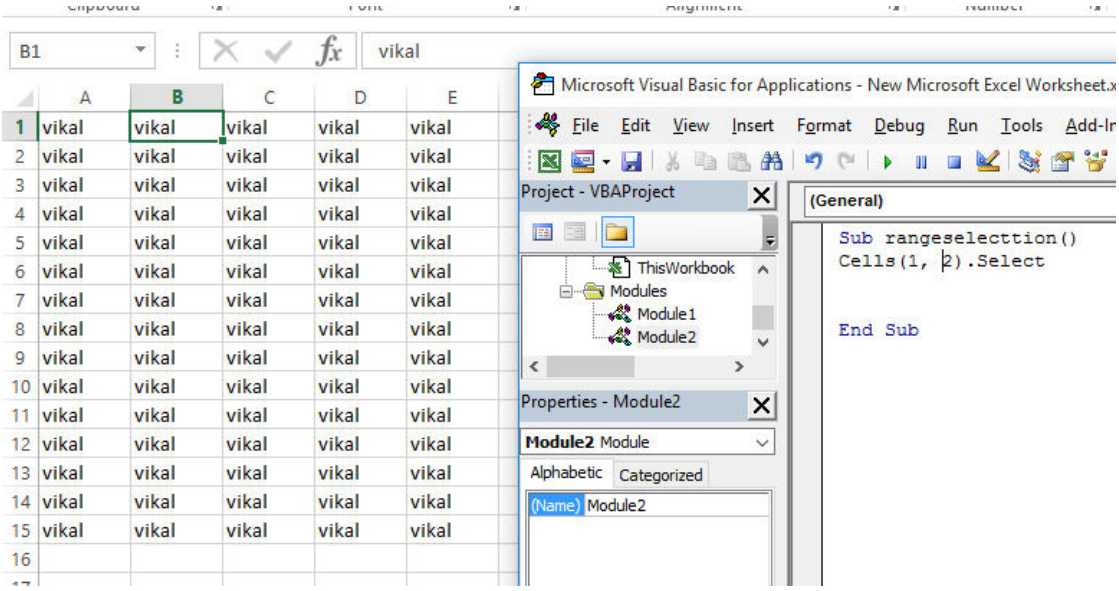


Figure 41

In the image our row number is 1 and column is 2, so our result is selection of cell B2 i.e. intersection of first row and second column

You can simply skip an arguments in your cell property like this if you don't want to select row and column together

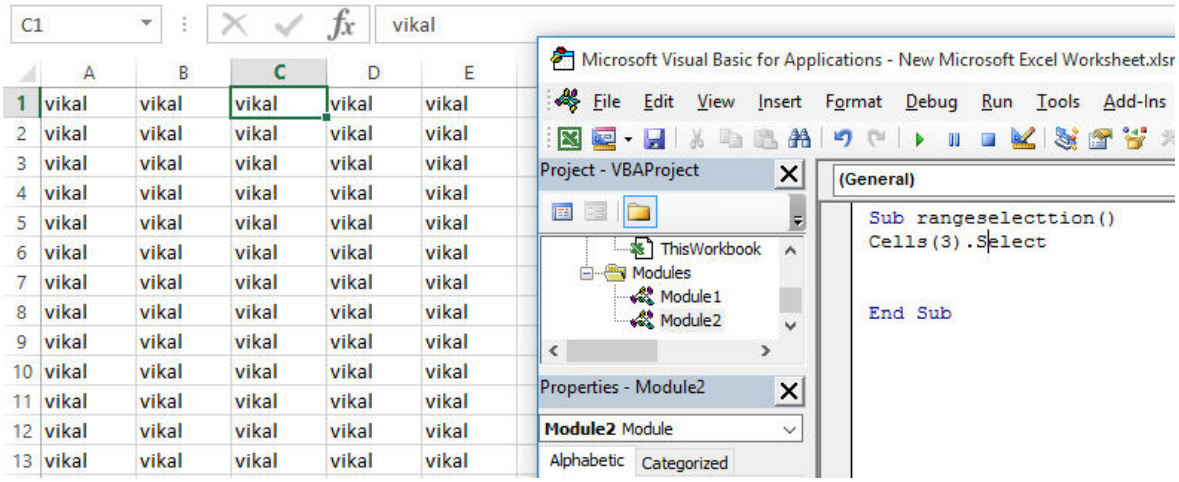


Figure 42

In the above image we have not used row number or column numberinstead we have used index number i.e. 3.

In EXCEL VBA indexing is done on basis on left to right then up to bottom so if we want to select cell A1 then we have to write 1, if we want c1 then 3 or If we want to work with A2 then you have to wait for completion of row number 1 indexing then only your selection will be done.i.e you have write cells(16385) for selecting the A2 cell since there is 16384 column in the excel. In other words wait to finish your row first then start your second row.

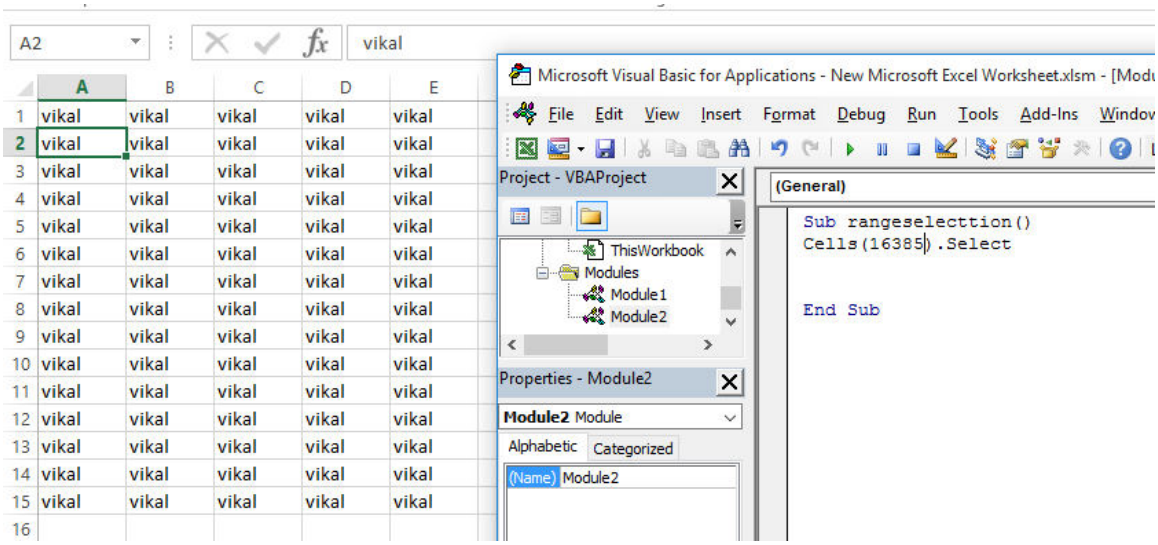


Figure 43

CELLS and RANGE object together

What does cell object do? It's select your cell right and what does range object do? Its select your range but it require your first cell and last cell. So if we express our first cell and last cell through cell object then, will this work. Let's check

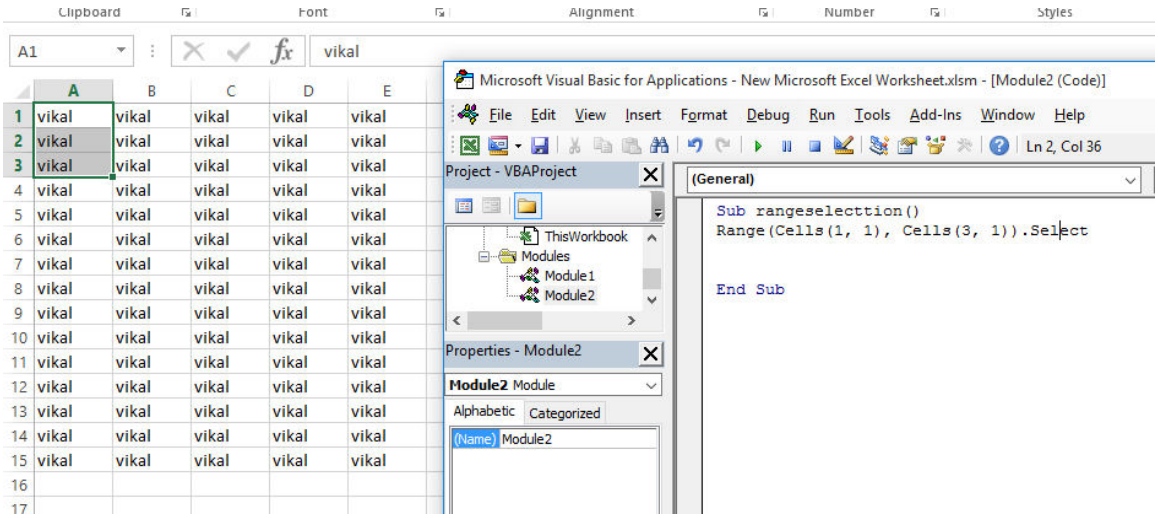


Figure 44

In the above image starting position of your range is Cell(1,1) i.e. cell A1 and ending position is Cell(3,1) i.e. A3. It is working wow.

Offset Property

Offset property in VBA allows you to relocate the position of the range or cell(s).

Syntax = Offset([rowoffset],[columnoffset]) As Range

Let's understand the offset property with the some examples. Let's suppose you have selected the range "E4 : H4" as shown in the Figure 45

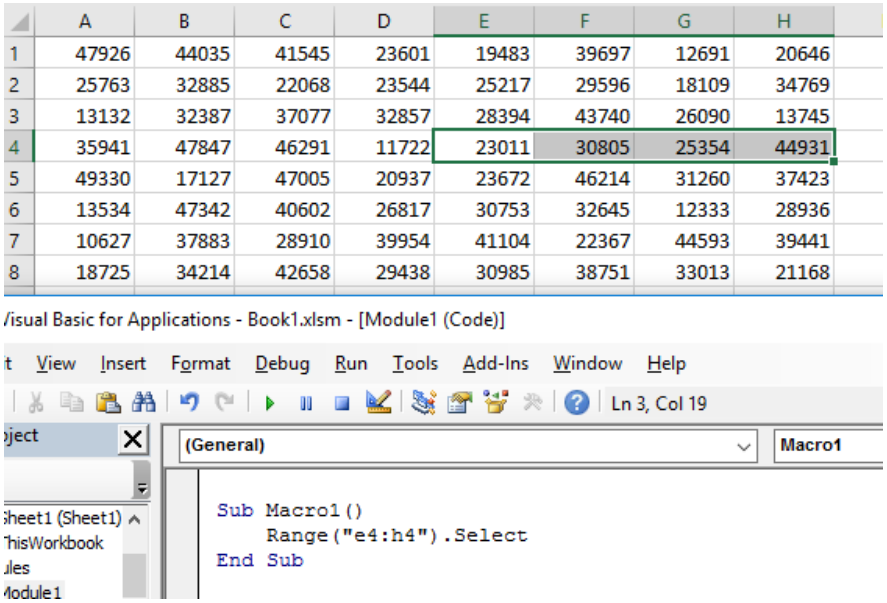


Figure 45

Now you want to change the your range position one row down i.e “E5:H5” you can simply type use this property to relocate your cell(s) or range.

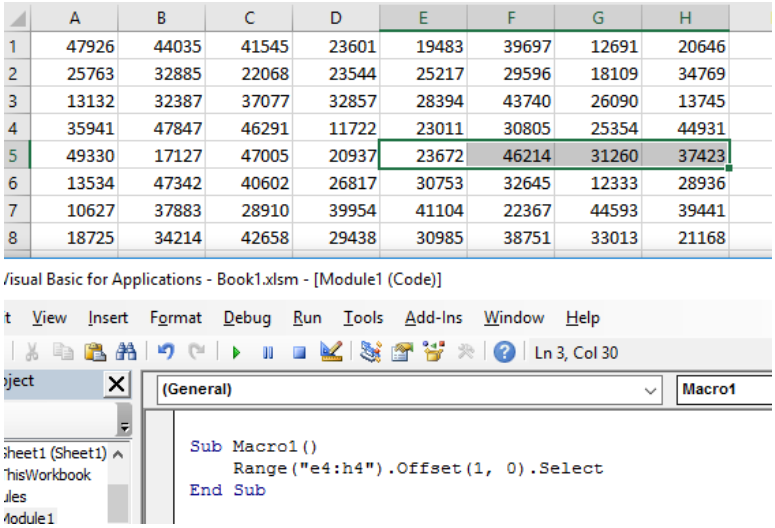


Figure 46

See I have used the Range(“E4:h4”).offset(1,0).select to relocate the location of the range. Range(“E4:h4”) is used for selecting the range, offset (1,0) means we want to switch one row down. See when you will type any number in rowoffset then it will switch the row and if you type any number in column offset then it will switch the column.

You can type 0, +ve , -ve number to relocate the position of the range.

Let’s suppose now we want to select range from C2 to F2 then you have to switch 2 rows left and two columns above so you have to type offset(-2,-2) as shown in Figure 47

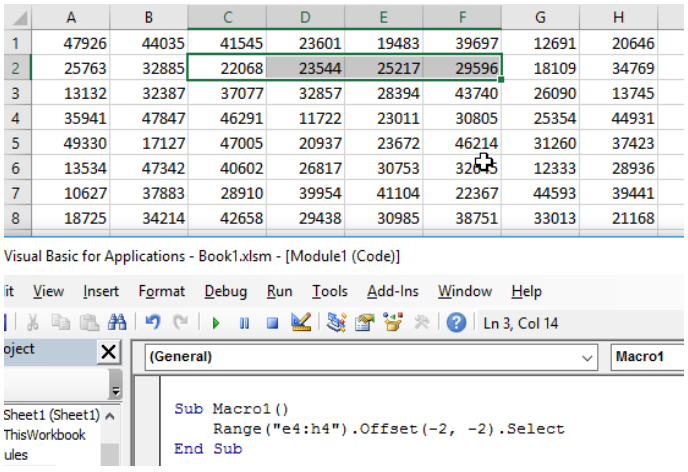


Figure 47

Table 2

Reference	Results
0	No switching
+ve numbers	Switch Right
-ve numbers	Switch left
Skip the arguments	No switching

Resize Property

Resize property is same as Offset property, but instead of relocating, it does resizing.

For example if we re-consider our Figure 45. Now this time we will resize our cell(s) or range. So, let's suppose we want our range 4 rows wider and 2 columns narrower so our revise range would be E4:F4 (4 +4 rows i.e. 7 including row number 4 & F – 2 columns i.e. column E so our revise range would be E7:F7)

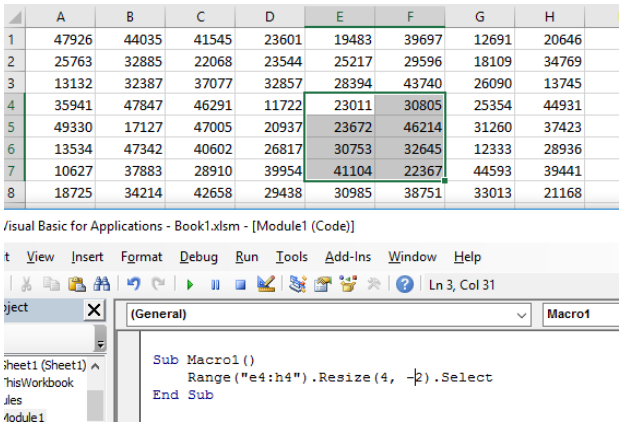


Figure 48

Chapter 6

ERROR & PRECAUTIONS

Errors in VBA

There are three types of error generally occur in VBA.

Syntax Error – This error will occur when you wrongly type the formula name or code name. Like you have typed Rang instead of Range, activcell instead of ActiveCell.

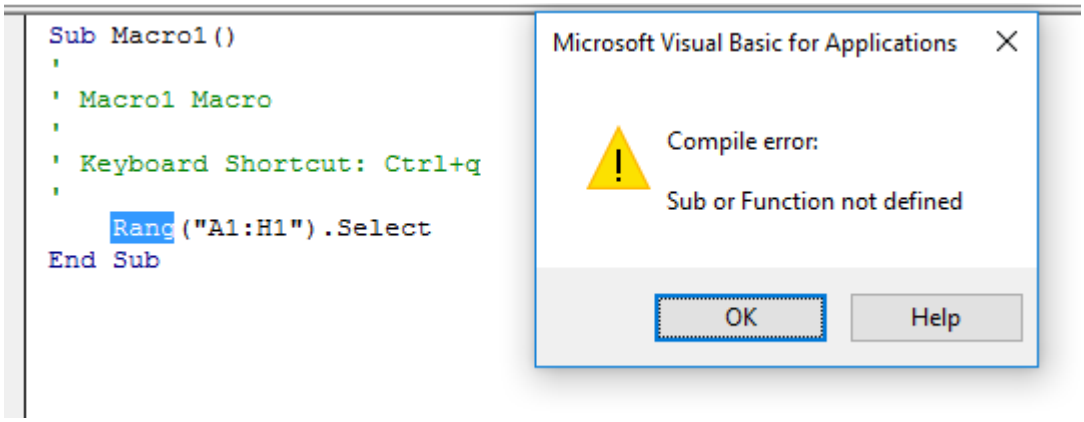


Figure 49

Precautions: when you are typing code in new line press CTRL + Space then select your desired cod, only then this error can be reduced.

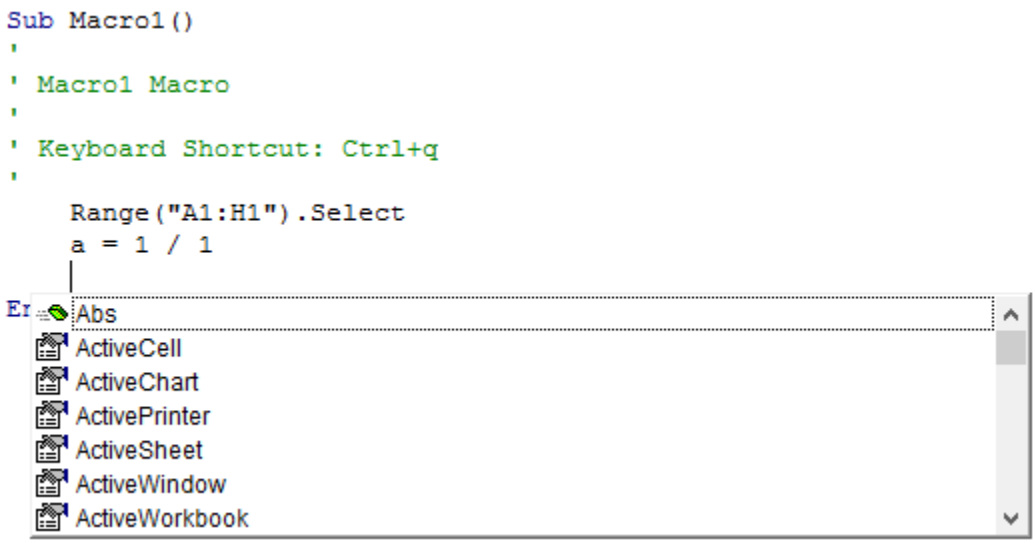


Figure 50

Compile errors - A compile error is generally easy to fix, as the VBA compiler pops up a message box, which provides information on the nature of the error.

Suppose you have forgotten to define the variable in VBA and you are using Option explicit then VBA will immediately show that you are forgot to define the variable.

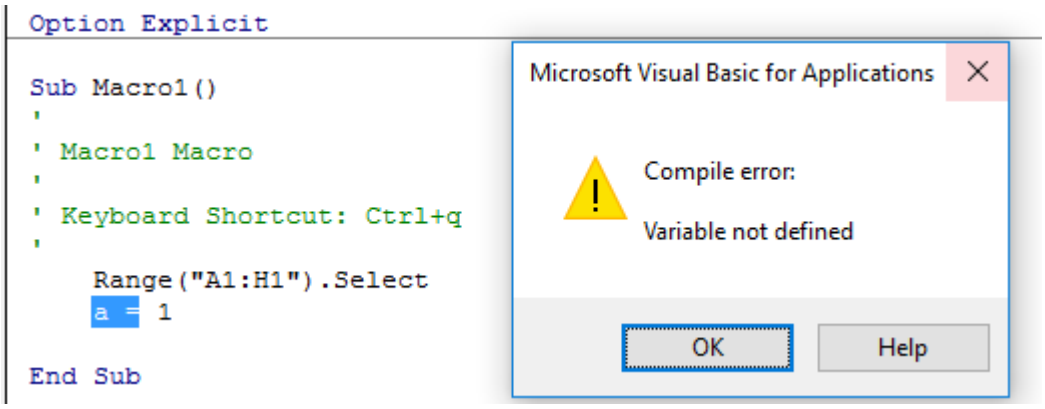


Figure 51

Run time errors - Runtime errors occur during the execution of your code, and cause the code to stop running

For example, if your code attempts to divide by zero, you will be presented with a message box, which states "Run-time error '11': Division by zero".

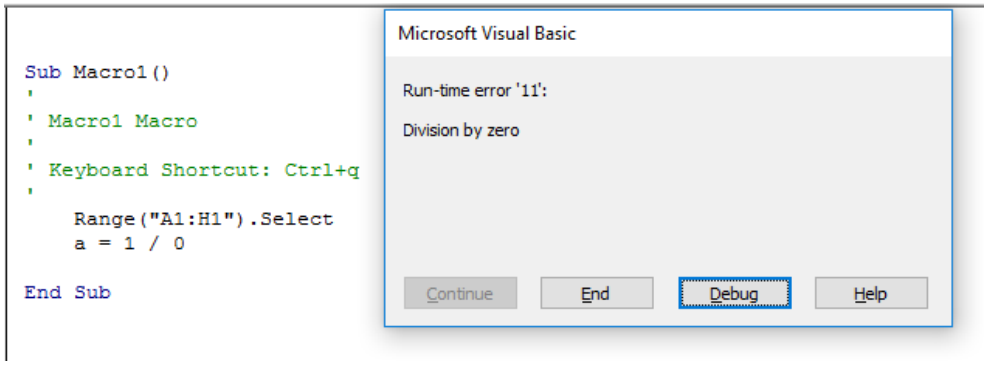


Figure 52

There are some run time errors provided by the Microsoft, which are as follows

Table 3

Error Numbers	Meaning
5	Invalid procedure call
7	Out of memory
9	Subscript out of range (this error arises if you attempt to access elements of an array outside of the defined array size - e.g. if you define an array indexed from 1 to 10, then attempt to access entry no. 11)
11	Division by zero
13	Type mismatch (this error arises when you attempt to assign

	the wrong type of value to a variable - e.g. define i as an integer, then attempt to assign the string "text" to i)
53	File not found (occurs when attempting to open a file)

Auto capitalized features of the VBA

If you want to check that whether you are writing your code properly or not then you must type your code with small alphabetic letters. Since if your code is correct then VBA will automatically capitalize it.

See in the Figure 53, I have typed my code in small letters when I will press enter, it will automatically capitalized as shown in figure 49.

```
Sub Macro1()
'
' Macro1 Macro
'
' Keyboard Shortcut: Ctrl+q
'
    range("A1:H1").select
    a = 1 / 1

End Sub
```

Figure 53

```
Sub Macro1()
'
' Macro1 Macro
'
' Keyboard Shortcut: Ctrl+q
'
    Range("A1:H1").Select
    a = 1 / 1

End Sub
```

Figure 54

Step by step running code

To review the proper working of code you may also run your code by line by line by pressing F8, this allow you to debug the code in the fast and efficient manner

```
Sub Macro1()
'
' Macro1 Macro
'
' Keyboard Shortcut: Ctrl+q
'
    Range("A1:H1").Select
    a = 1 / 1

End Sub
```

Figure 55

Shortcuts summary

Table 4

Shortcuts	Particulars
F8	Run line by line of your code
Ctrl + Space	View all properties and method list

Chapter 7

LAST ROW AND LAST COLUMN

Our data length can be variable every time. We have to always determine what is the last row and what is the last column of our data, so that we can execute our automate operation on every data set.

Finding Last Row

Let's consider excel first

So here is our data shown in the Figure 56. Now, we have to select last row of the our data set. What is the last row number? It is 19 but it can be variable with new data. So for this we have to go to last row of the EXCEL i.e. 1048576 as shown in Figure 57.

	A	B	C	D	E	F	G	H	I
1	47926	44035	41545	23601	19483	39697	12691	20646	
2	25763	32885	22068	23544	25217	29596	18109	34769	
3	13132	32387	37077	32857	28394	43740	26090	13745	
4	35941	47847	46291	11722	23011	30805	25354	44931	
5	49330	17127	47005	20937	23672	46214	31260	37423	
6	13534	47342	40602	26817	30753	32645	12333	28936	
7	10627	37883	28910	39954	41104	22367	44593	39441	
8	18725	34214	42658	29438	30985	38751	33013	21168	
9	44752	48463	35340	46822	17990	24938	35709	45251	
10	22109	36191	29819	45781	27029	22660	25115	33535	
11	22776	34906	25362	20199	18774	20179	11900	32805	
12	13589	49536	25293	49038	24875	22888	17943	18380	
13	34149	12383	34071	32833	49671	13662	38795	28974	
14	48742	43012	39044	26593	13135	17427	25774	49309	
15	45652	27257	35916	20238	41514	41388	21473	26444	
16	44478	10515	49826	26667	49957	21871	36565	22196	
17	39445	28107	32449	38044	21935	30693	35174	25194	
18	43763	28496	44480	29133	29524	16224	34071	39813	
19	22892	47215	20165	42786	44354	35663	48016	27016	
20									

Figure 56

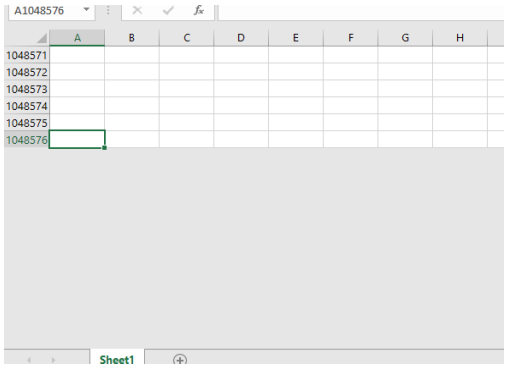
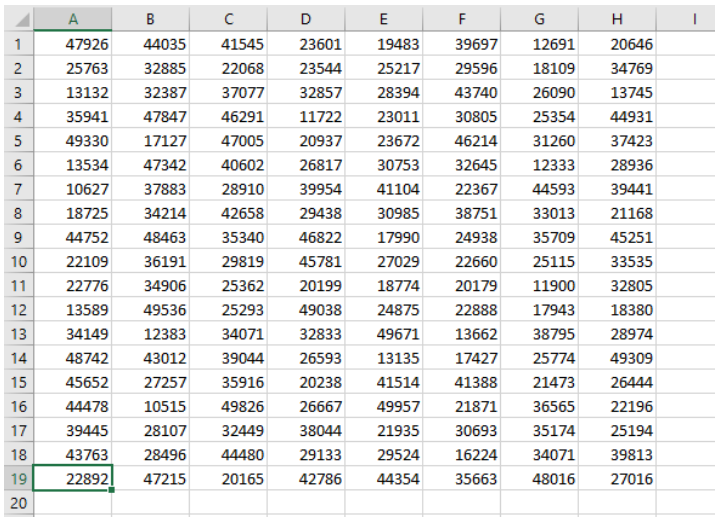


Figure 57

When we are on the last row we have to simply press Ctrl + Up Arrow. So we will on the last row of our data as shown in Figure 58 .



	A	B	C	D	E	F	G	H	I
1	47926	44035	41545	23601	19483	39697	12691	20646	
2	25763	32885	22068	23544	25217	29596	18109	34769	
3	13132	32387	37077	32857	28394	43740	26090	13745	
4	35941	47847	46291	11722	23011	30805	25354	44931	
5	49330	17127	47005	20937	23672	46214	31260	37423	
6	13534	47342	40602	26817	30753	32645	12333	28936	
7	10627	37883	28910	39954	41104	22367	44593	39441	
8	18725	34214	42658	29438	30985	38751	33013	21168	
9	44752	48463	35340	46822	17990	24938	35709	45251	
10	22109	36191	29819	45781	27029	22660	25115	33535	
11	22776	34906	25362	20199	18774	20179	11900	32805	
12	13589	49536	25293	49038	24875	22888	17943	18380	
13	34149	12383	34071	32833	49671	13662	38795	28974	
14	48742	43012	39044	26593	13135	17427	25774	49309	
15	45652	27257	35916	20238	41514	41388	21473	26444	
16	44478	10515	49826	26667	49957	21871	36565	22196	
17	39445	28107	32449	38044	21935	30693	35174	25194	
18	43763	28496	44480	29133	29524	16224	34071	39813	
19	22892	47215	20165	42786	44354	35663	48016	27016	
20									

Figure 58

Now these steps we have to do in EXCEL VBA also

Go to last row of the EXCEL

Press Ctrl + Up arrow once so that we will be on our last row of the data.

This is the secret for last row

Cells(Rows.Count, 1).End(xlUp)

Cells(Rows.Count, 1) Explanation -- Here Row.count is counting all rows in the EXCEL and we have taken 1st column in the cell property so our last row value will be empty since our last row of the excel is empty Cells(Rows.Count, 1) as shown in Figure 57. When we add End(xlup) to this code this will work like as a CTRL + UPARROW i.e. the last row value will be 22892 as shown in Figure 58.

One thing we have to keep in mind that this code is providing the last row value not the number of the last row. If we want to find the last row number then we have to add row corresponding by dot. (Cells(Rows.Count, 1).End(xlUp).row). Now it will show the last row number of our data set.

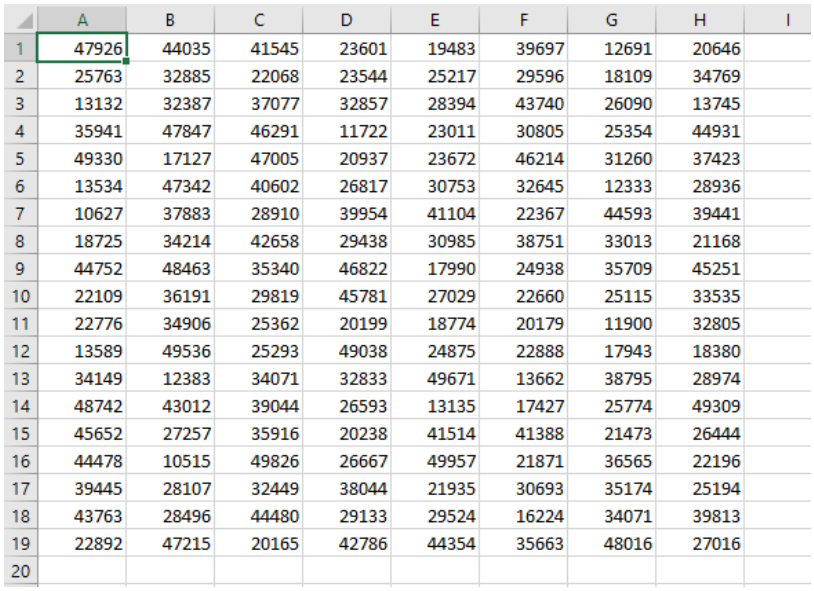
Author Note:

Table 5

XLUP	Work as Up arrow
XLDOWN	Work as down arrow
XLTOLEFT	Work as left arrow
XLTORIGHT	Work as right arrow

Finding Last column

After understanding last row, finding last column is very easy. So here is the our data set in figure



	A	B	C	D	E	F	G	H	I
1	47926	44035	41545	23601	19483	39697	12691	20646	
2	25763	32885	22068	23544	25217	29596	18109	34769	
3	13132	32387	37077	32857	28394	43740	26090	13745	
4	35941	47847	46291	11722	23011	30805	25354	44931	
5	49330	17127	47005	20937	23672	46214	31260	37423	
6	13534	47342	40602	26817	30753	32645	12333	28936	
7	10627	37883	28910	39954	41104	22367	44593	39441	
8	18725	34214	42658	29438	30985	38751	33013	21168	
9	44752	48463	35340	46822	17990	24938	35709	45251	
10	22109	36191	29819	45781	27029	22660	25115	33535	
11	22776	34906	25362	20199	18774	20179	11900	32805	
12	13589	49536	25293	49038	24875	22888	17943	18380	
13	34149	12383	34071	32833	49671	13662	38795	28974	
14	48742	43012	39044	26593	13135	17427	25774	49309	
15	45652	27257	35916	20238	41514	41388	21473	26444	
16	44478	10515	49826	26667	49957	21871	36565	22196	
17	39445	28107	32449	38044	21935	30693	35174	25194	
18	43763	28496	44480	29133	29524	16224	34071	39813	
19	22892	47215	20165	42786	44354	35663	48016	27016	
20									

Figure 59

Can you determine what will be the steps for finding last column

Find last column of the EXCEL

Press CTRL + LEFTARROW

So our code will be for the last column value

Cells(1,Columns.Count).End(xltoleft)

And for the last column no. the code will be

Cells(1,Columns.Count).End(xltoleft).column

Summary

Table 6

Code	Particulars
Cells(Rows.Count, 1).End(xlUp)	Finding the value of the last row
Cells(Rows.Count, 1).End(xlUp).row	Finding the number of the last row
Cells(1,Columns.Count).End(xltoleft)	Finding the value of the last column
Cells(1,Columns.Count).End(xltoleft).column	Finding the number of the last column

Chapter 8

WORKSHEETS

Selecting the sheets

Let's understand moving the worksheets with the following example

Here we have two sheets as given in the Figure 60

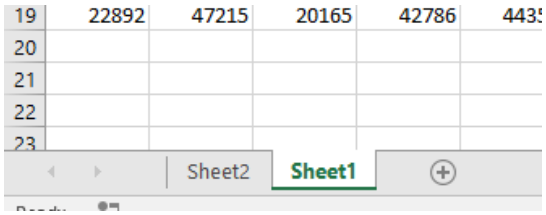


Figure 60

Sheet1 has some data and Sheet2 is the blank sheets as shown in Figure 61 and Figure 62

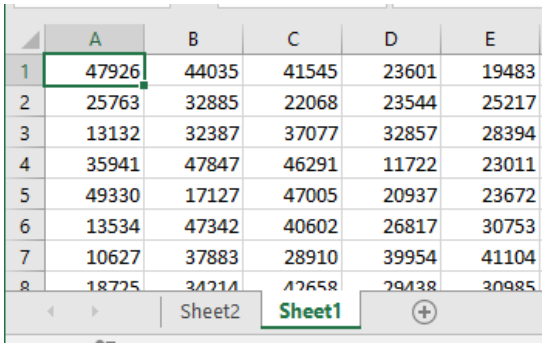


Figure 61

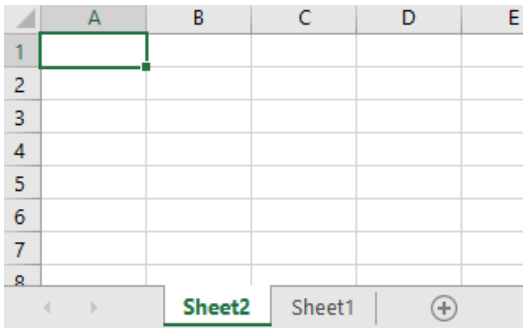


Figure 62

Let's move between the sheets with the following code

Worksheets("Sheet1").Activate

Here sheet1 is the Sheet name therefore we have written the sheet1 in double quotes ("").

Worksheets is the function to refer the sheet.

Activate means Activating the desired sheet.

Whenever you have more than 1 sheet in EXCEL then you can simple refer to the sheet with the Worksheets("Sheet1").Activate. Let's Execute this code by pressing F5.

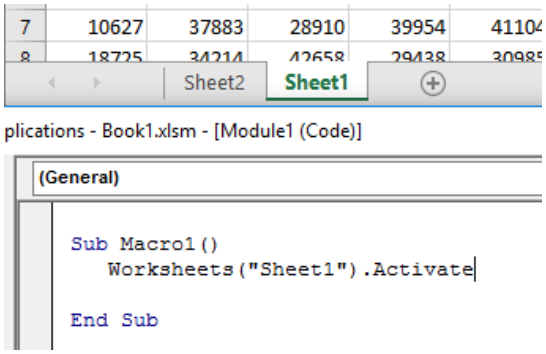


Figure 63

In Figure 63 we have executed the code.

Selecting the Charts

Let's make the chart of our datashown in Figure 64

	A	B	C	D	E	F	G	H
1	Sales made by							
2	Year	Vikal Jain	Akshay Jain	Rahul verma	Mohit Sharma	Alex	Heena Sharma	Fatima
3	2009	44035	41545	23601	19483	39697	12691	20646
4	2010	32885	22068	23544	25217	29596	18109	34769
5	2011	32387	37077	32857	28394	43740	26090	13745
6	2012	47847	46291	11722	23011	30805	25354	44931
7	2013	17127	47005	20937	23672	46214	31260	37423
8	2014	47342	40602	26817	30753	32645	12333	28936
9	2015	37883	28910	39954	41104	22367	44593	39441
10	2016	34214	42658	29438	30985	38751	33013	21168
11	2017	48463	35340	46822	17990	24938	35709	45251
12	2018	36191	29819	45781	27029	22660	25115	33535
13	2019	34906	25362	20199	18774	20179	11900	32805
14	2020	49536	25293	49038	24875	22888	17943	18380
15	2021	12383	34071	32833	49671	13662	38795	28974
16	2022	43012	39044	26593	13135	17427	25774	49309
17	2023	27257	35916	20238	41514	41388	21473	26444
18	2024	10515	49826	26667	49957	21871	36565	22196
19	2025	28107	32449	38044	21935	30693	35174	25194
20	2026	28496	44480	29133	29524	16224	34071	39813
21	2027	47215	20165	42786	44354	35663	48016	27016

Figure 64

To insert a chart of our data select your data first i.e. A2:H21. Then press F1. Now a new Chart1 sheet will be inserted by EXCEL.

Figure 65

Figure 65 is showing the CHART1 sheet and Figure 66 showing the our chart.

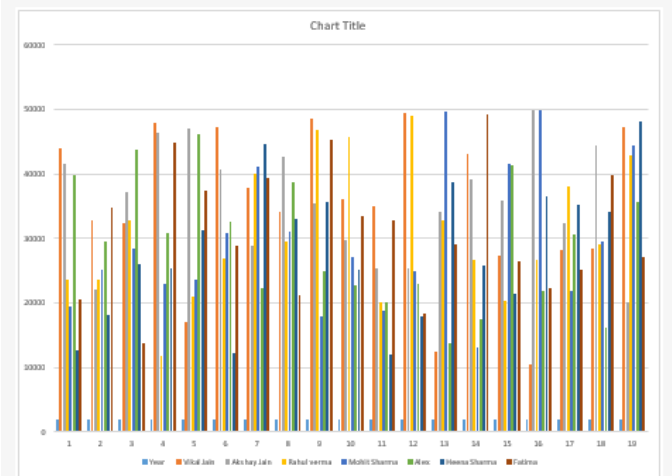


Figure 66

Let's move to Chart1 sheets by VBA now. If we will write this code Worksheets("chart1").Activate then VBA will show error as shown in Figure 67

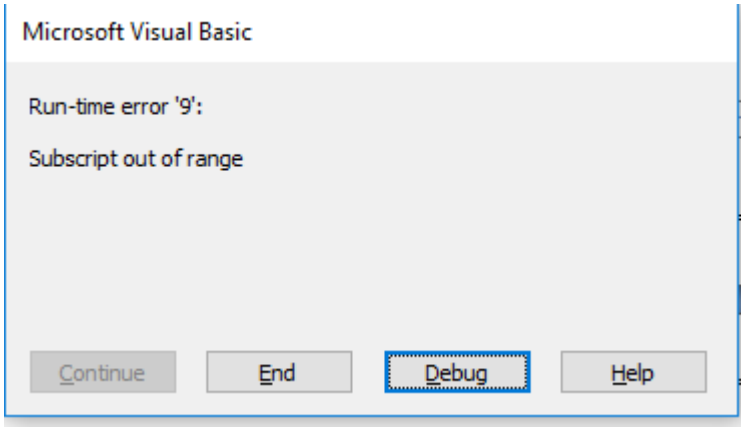


Figure 67

The reason of the error is Worksheets function can't handle the chart sheets it is only made for selecting the sheets

Let's try this code now

Charts("Chart1").Activate

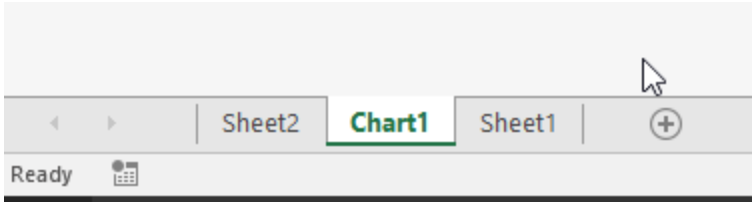


Figure 68

Now our code is running properly.

Selecting the Sheets through Sheet function

Now after Worksheets function and Chart function the question come in the mind that whether we have to remember the every function for the Sheets to refer the sheets.

Answer is No! You can simply use sheet function also to refer the sheets and charts or any other type of the sheets.

Sheets("Sheet1").Activate

Sheets("Chart1").Activate

Here both codes will work in the proper way, without showing the error.

Author Note: Primary difference between these two is Worksheets property identifies only the type "Worksheets" in excel but Sheets is more general and identifies all the types of sheets (Worksheets, Charts, Modules / Macro, Dialog sheets). Therefore it is pure redundancy in EXCEL VBA.

Selecting multiple sheets

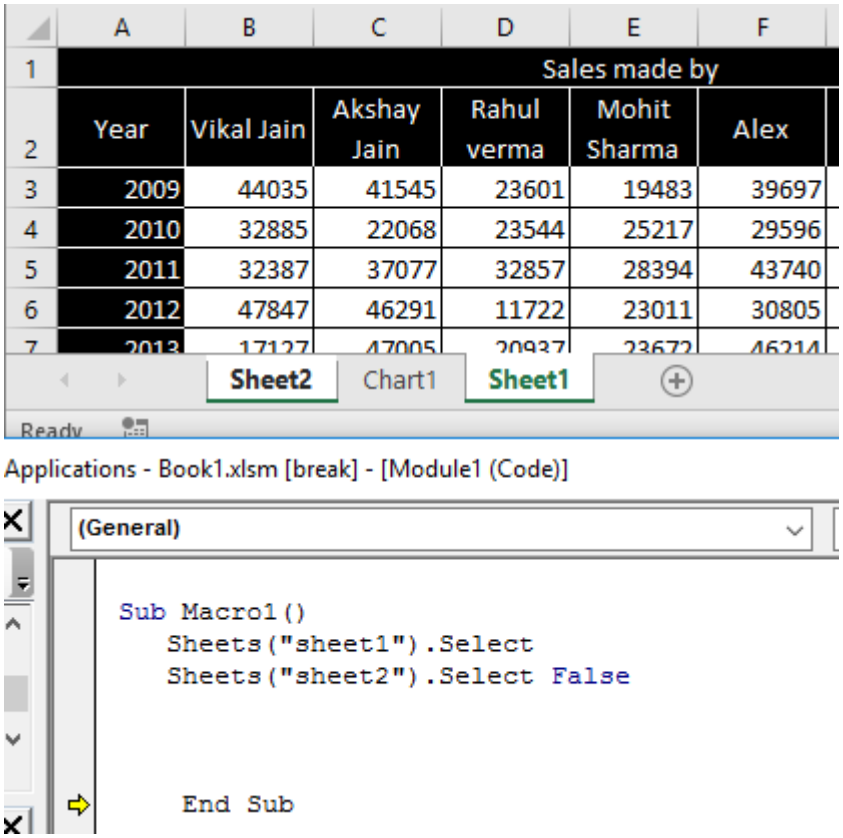
Selecting Multiple sheets is very easy, Let's understand

Sheets("Sheet1").Select

Sheets("Sheet2").Select

If we try these codes then EXCEL first select the Sheet1 then it will move to Sheet2. Rather selecting both sheets simultaneously, it will select only one sheet at a single point of time.

Here is the solution



	A	B	C	D	E	F
1				Sales made by		
2	Year	Vikal Jain	Akshay Jain	Rahul verma	Mohit Sharma	Alex
3	2009	44035	41545	23601	19483	39697
4	2010	32885	22068	23544	25217	29596
5	2011	32387	37077	32857	28394	43740
6	2012	47847	46291	11722	23011	30805
7	2013	17127	47005	20937	23672	46214

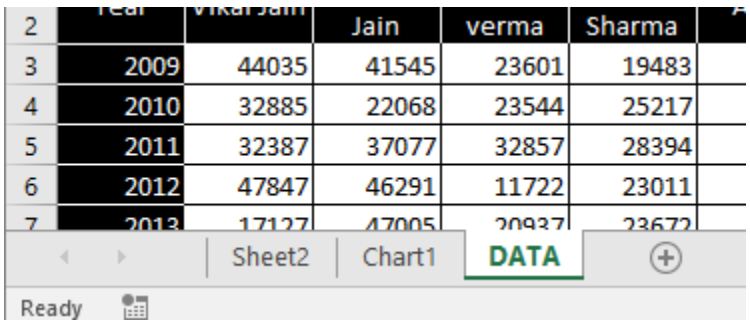
Figure 69

Sheets("Sheet1").Select

Sheets("Sheet2").Select False

Indexation of the Worksheets

Till now our code are totally dependent on the EXCEL, for example excel provides Sheet1, Sheet2 .....by default so we are using that sheets name in our code. Let's change the sheets name now as shown in Figure 70



2	Year	Vikal Jain	Jain	verma	Sharma
3	2009	44035	41545	23601	19483
4	2010	32885	22068	23544	25217
5	2011	32387	37077	32857	28394
6	2012	47847	46291	11722	23011
7	2013	17127	47005	20937	23672

Figure 70

Now we have changed the Sheet name from Sheet1 to DATA. Let's run the same code now.

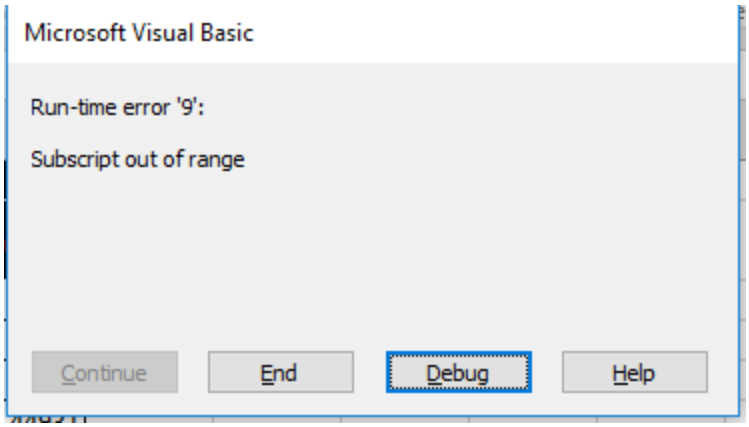


Figure 71

VBA is showing error. So we can avoid this type of errors by indexing the Worksheets.

Indexing through worksheet number

This is so simple method to index your worksheets, instead of giving the whole name to the worksheet, just give the number of the sheet

Sheets("Sheet1").Select

Sheets(1).Select

But we careful EXCEL consider the sheets number as per the location of the sheet, for example if sheet2 is on the left of the all sheets then it will be considered as sheet 1.

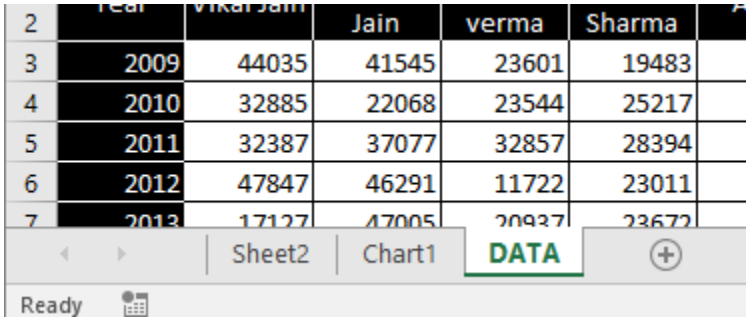


Figure 72

In Figure 72 Sheet2 is the number 1 sheet, chart 1 is the number 2 and so on.

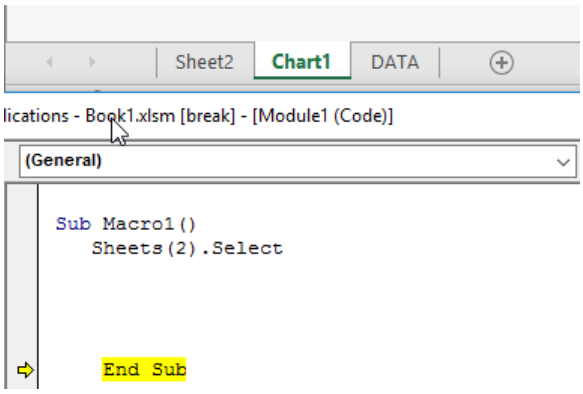


Figure 73

Indexing through VBA Project number

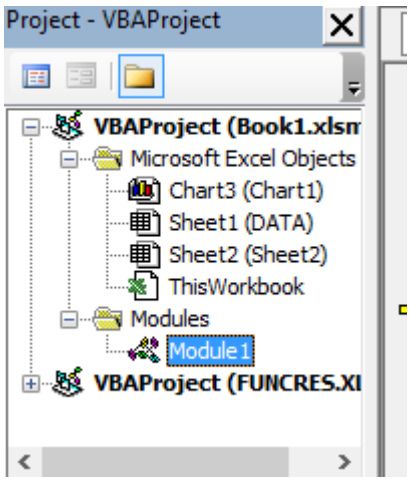


Figure 74

VBA has this window on the TOP LEFT side of the your screen this is called as project window. This window is showing actual sheet name of your sheets like Chart 1 is the chart 3, Data is the sheet 1. You can index through these names also.

If you are unable to see this window then you can press through your key board Ctrl + R or go to view menu then click on Project explorer.

Now you can select your sheets by these names like

Sheets("Sheet1").Select

Sheet1.Select

See this time we did not use the sheet property we simply used the sheet name.

When you will run this code then DATA sheet will be selected since it,s actual name is sheet 1.

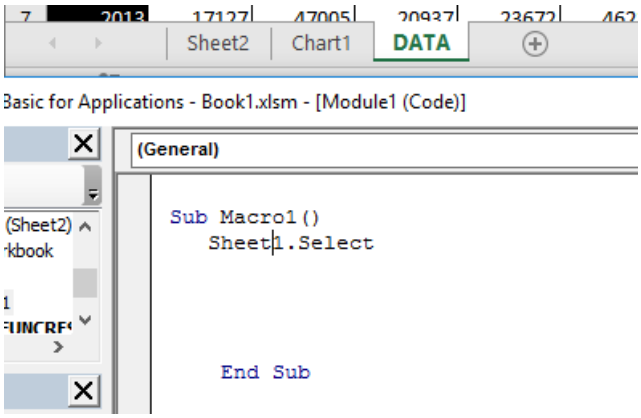


Figure 75

Insert the worksheet

To insert the worksheets we have to first understand the Add property in EXCEL VBA.

Add Property is used to add the worksheets in EXCEL through VBA. Here is the syntax.

Syntax

expression .Add(Before, After, Count, Type)

Parameters

Table 7

Name	Required/ Optional	Data Type	Description
Before	Optional	Variant	An object that specifies the sheet before which the new sheet is added.
After	Optional	Variant	An object that specifies the sheet after which the new sheet is added.
Count	Optional	Variant	The number of sheets to be added. The default value is one.
Type	Optional	Variant	Specifies the sheet type. Can be one of the following XlSheetType constants: xlWorksheet, xlChart,xlExcel4MacroSheet, or xlExcel4IntlMacroSheet. If you are inserting a sheet based on an existing template, specify the path to the template. The default value

			is xlWorksheet.

Source https://msdn.microsoft.com/en-us/library/office/ff839847(v=office.15).aspx

Let's understand

Here is the our worksheets

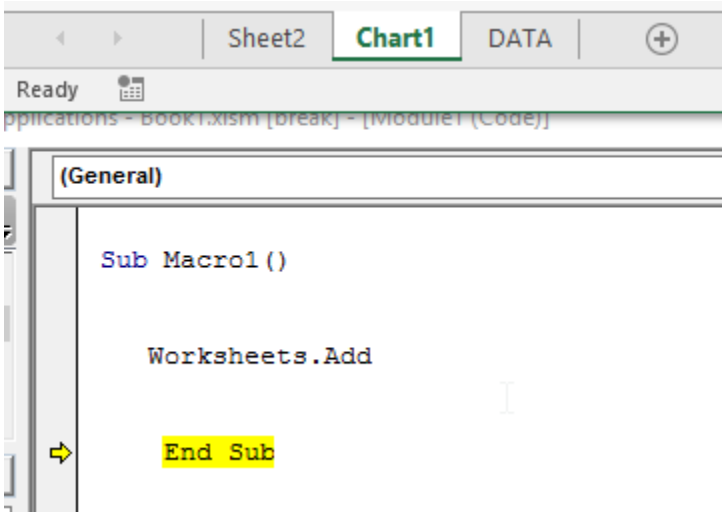


Figure 76

In the Figure 76 we have selected the CHART1 sheet and we have written " Worksheets.Add" in the code. As and when we will perform this code the sheet will be inserted in the left side of the selected sheet as shown in Figure 77.

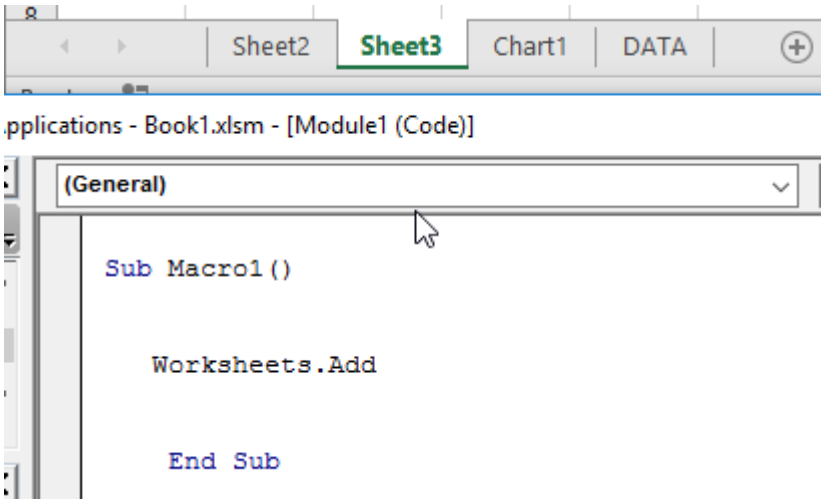


Figure 77

In this case you have to select the sheet first then you have to perform the code. For example let's suppose you always want to insert the sheet on the left side of the DATA sheet then you have to select data sheet first then add a new sheet as shown in Figure 78.

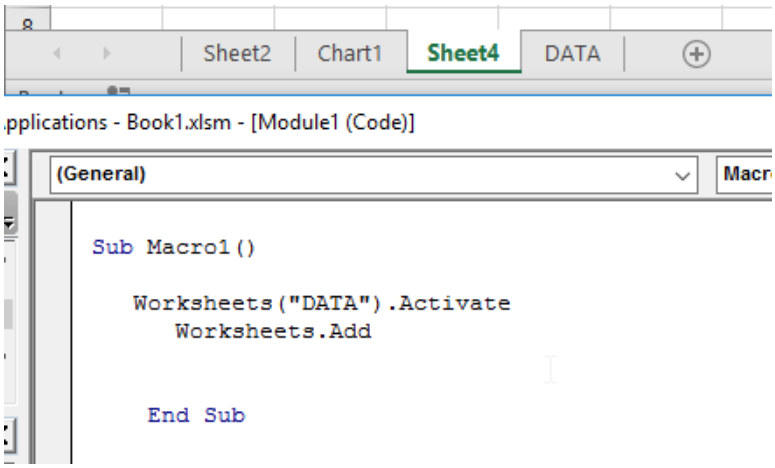


Figure 78

But if you have seen the above table then we can do this without the select the sheets also.

Let's do this we will first insert the sheet before DATA sheet with this code

Worksheets.Add Worksheet("Data")

In this code we have filled out the "before" arguments of the Add syntax i.e. expression .Add(Before, After, Count, Type)

If you want to use the "After arguments then you must use the comma "," in your code. Like we are doing in

Figure 80.

Author Note: for every use of comma you will switch to next comma for e.g. if you would single comma "," then you will switch to "After" argument, if you use two commas ",," then you will switch to count arguments of the syntax.

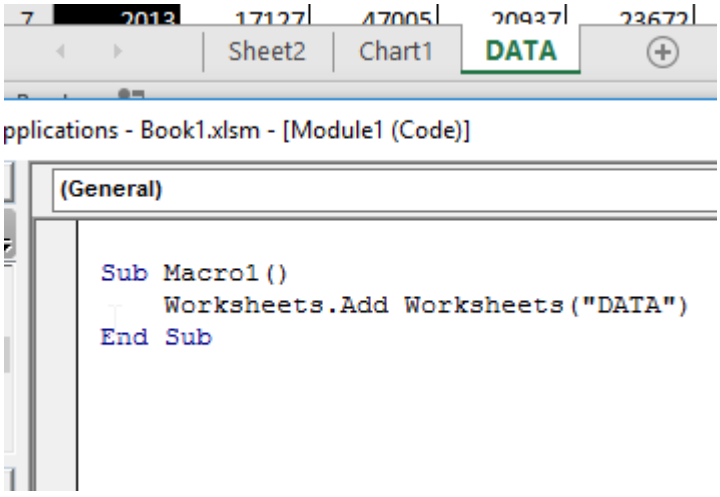


Figure 79

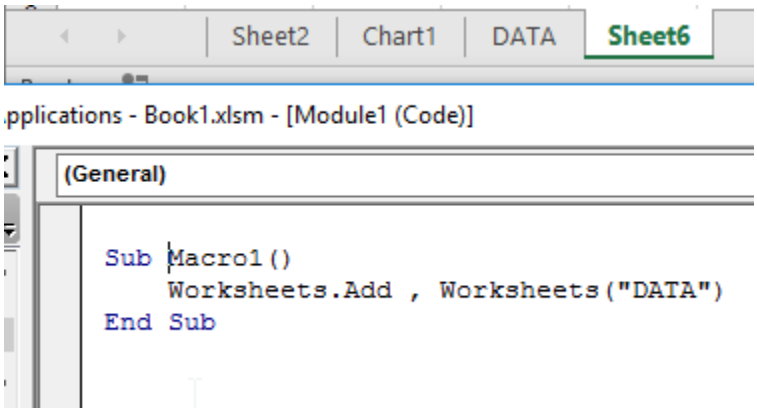


Figure 80

The alternative use of comma is that you can type you whole arguments first then type := then use your code . like we are doing in the Figure 81. In this figure we have used “Before arguments” then used “:=” then we have type the sheet name. See the result worksheet is inserted on the left of the DATA worksheets.

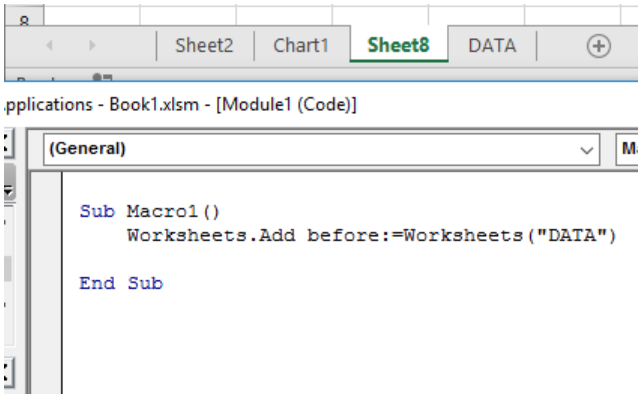


Figure 81

In this example we have used two arguments first is the “After” arguments and the second argument is “Count”. See the result two worksheets has been added to the next of the DATA sheet as shown in Figure 82

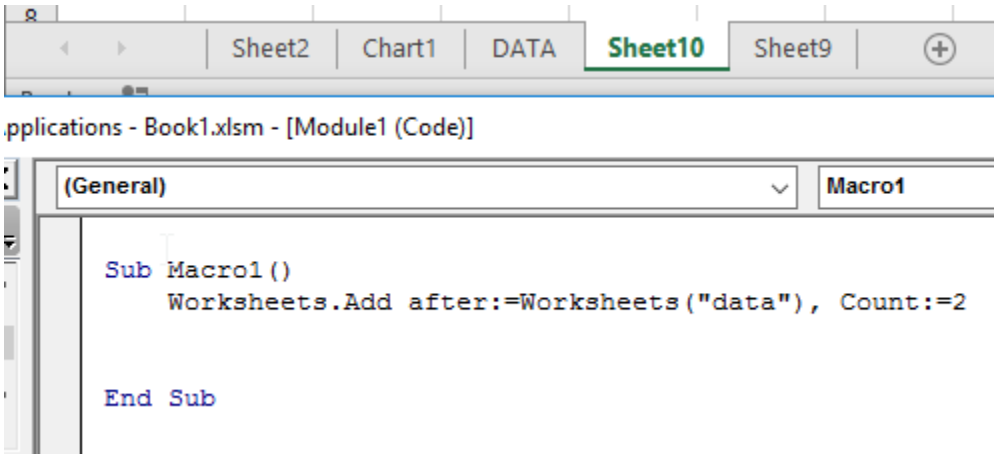


Figure 82

Deleting the worksheets

If you want to delete the worksheets then you can use this code to delete the worksheet

Nameof the sheet.Delete

Worksheet(“sheet10”).delete

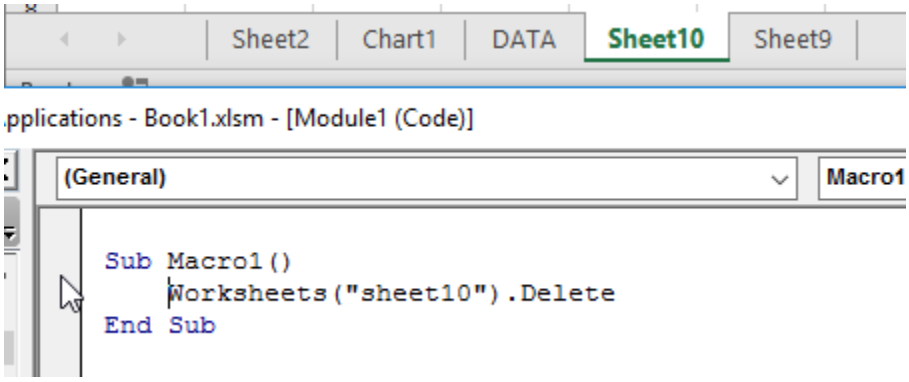


Figure 83

When you will run the code this will open the pop up window as given in Figure 83

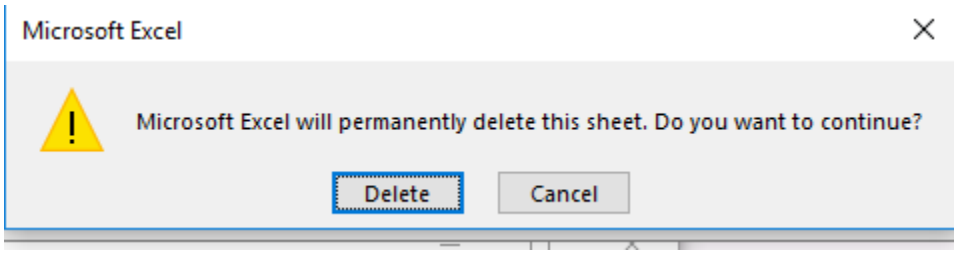


Figure 84

You can also disable this pop up window by using this line of code

Application,Displayalerts = False

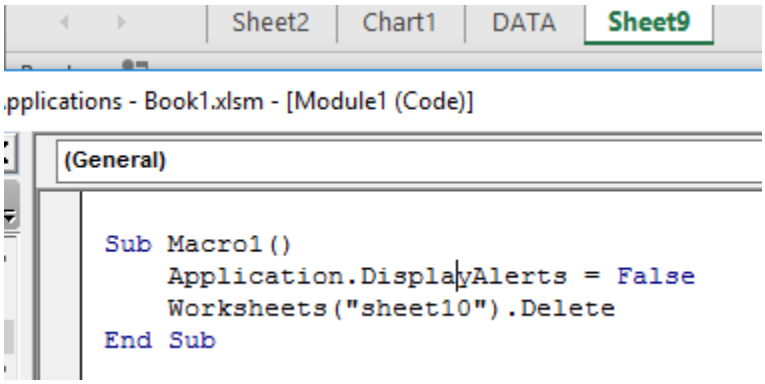


Figure 85

This code will simply delete the sheet without showing any pop up window.

Author Note: You can delete all one type of sheets in one shot

For this you have to use

Sheets.delete

This will delete all sheets

Charts.delete

This will delete the all type of charts sheet

Copying the sheets

Syntax

expression .Copy(Destination)

Parameter

Table 8

Name	Required/Optional	Data Type	Description
Destination	Optional	Variant	Specifies the new range to which the specified range will be copied. If this argument is omitted, Microsoft Excel copies the range to the Clipboard.

Source : https://msdn.microsoft.com/en-us/library/office/ff837760(v=office.15).aspx

Destination may be the after the worksheet or before the worksheet. If you want to add the worksheets

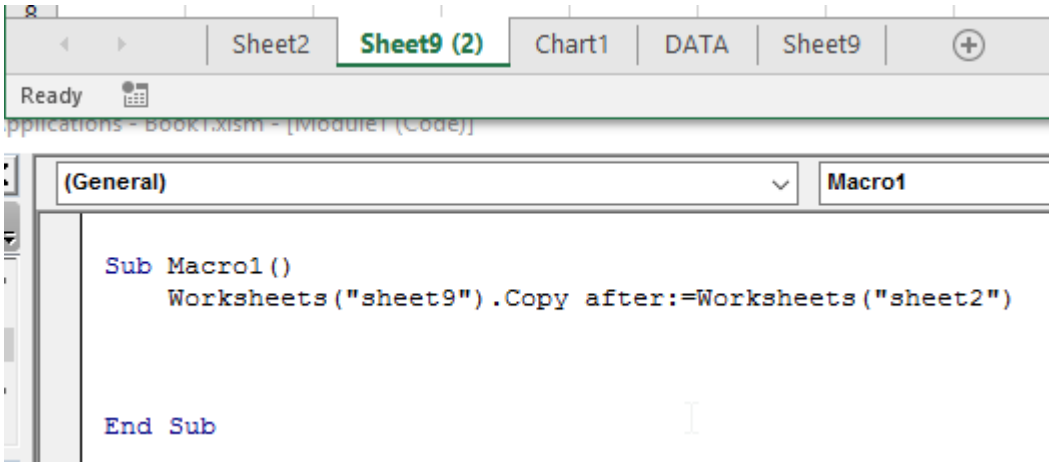


Figure 86

In the Figure 86 we have copied the sheet 9 after the sheet 2

Moving the worksheets

Moving the worksheet is the same as the copying the worksheet instead of copy , just you will have to use the move code

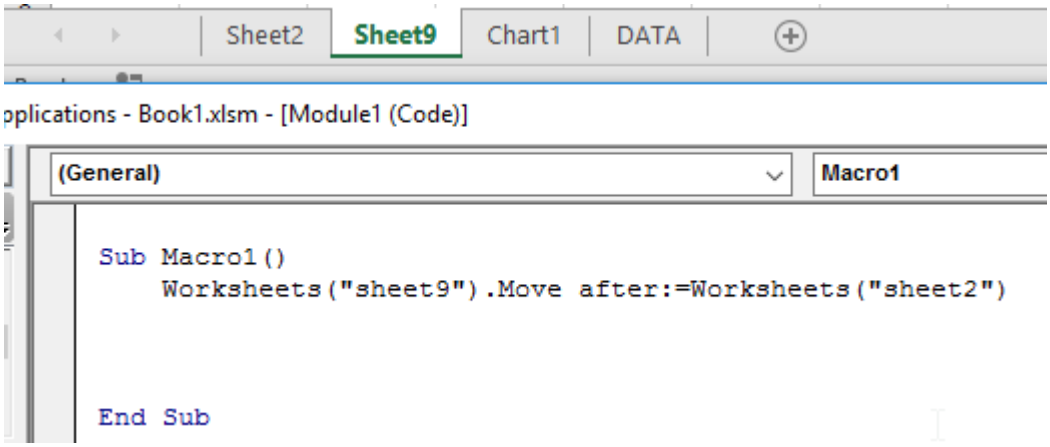


Figure 87

Renaming Sheets

For renaming the worksheet you have to use the NAME property in VBA

Worksheets("sheet2").Name= "BOSS"

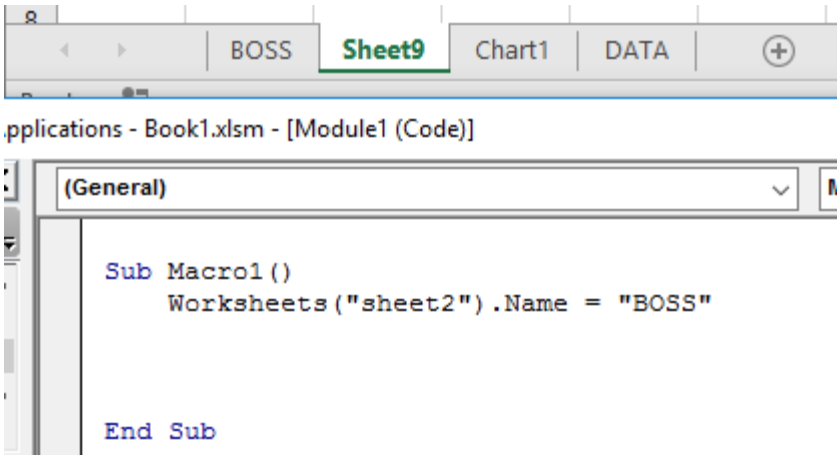


Figure 88

Hiding and Un-hiding the sheets

For hiding and Unhiding the worksheet you have to use the Visible property in VBA. There are three type of hiding and unhiding method in EXCEL VBA

Table 9

Method	What it do
Xlsheetvisible	It's unhide the hidden sheet
Xlsheethidden	It's hide the sheet
Xlsheetveryhidden	It's very hide the sheet, excel can't unhide this sheet without the help of the VBA

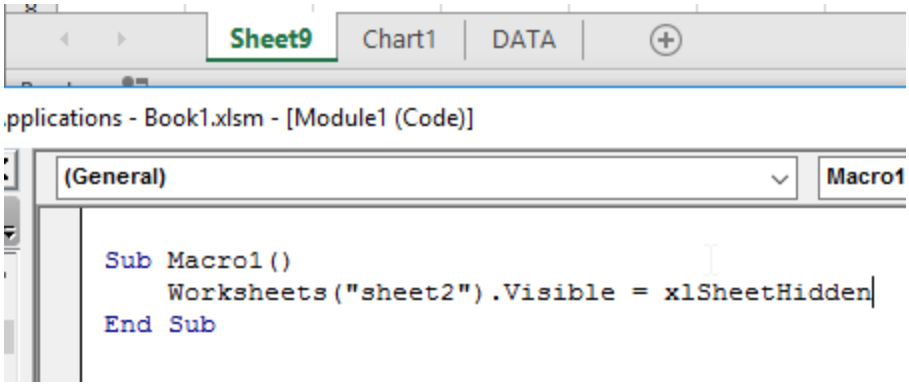


Figure 89

In the Figure 89 we have hidden the sheet 2 by using the Visible property, you can unhide this sheet by using EXCEL also, since it is simply hidden sheet. Just go to excel sheet and right click on any sheet then click on unhide as shown in Figure 90

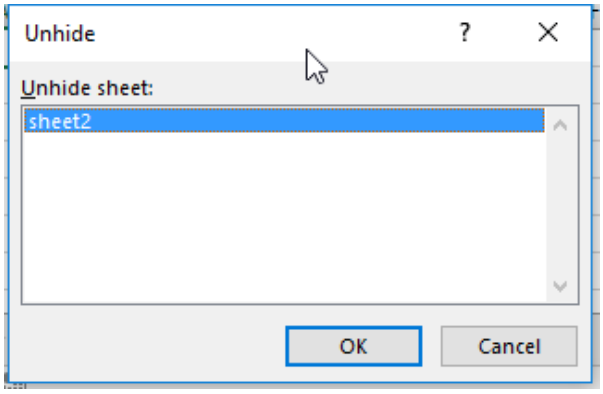


Figure 90

If we will use “Xlsheetveryhidden” then excel can’t unhide it. See the Figure 91 and Figure 92

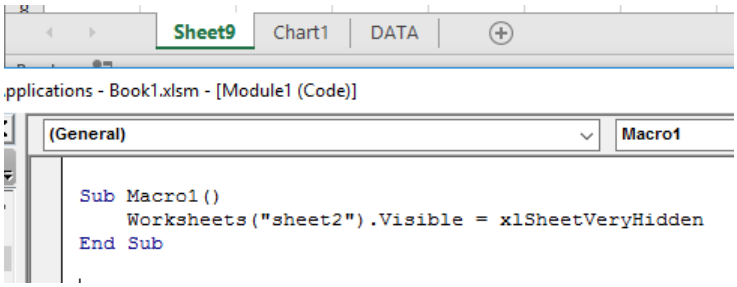


Figure 91

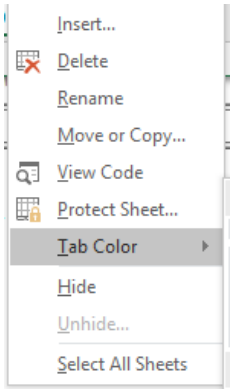


Figure 92

Figure 93 is showing how to unhide the sheets through VBA

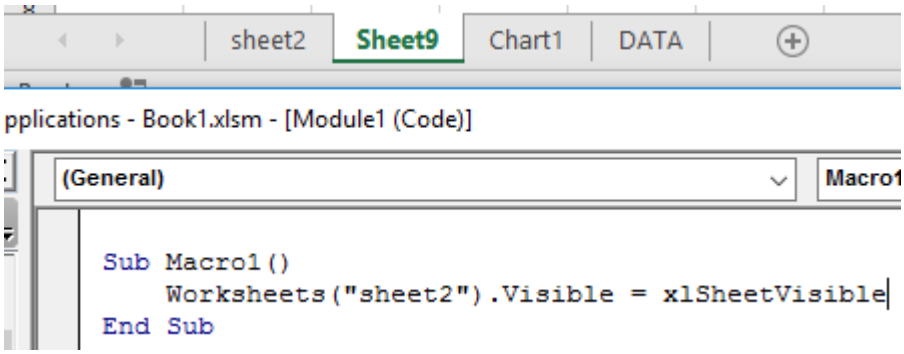

sheet2
Sheet9
Chart1
DATA
pplications - Book1.xlsm - [Module1 (Code)]
(General)
Macro1
Sub Macro1()
Worksheets("sheet2").Visible = xlSheetVisible
End Sub

Figure 93

Chapter 9

WORKBOOKS

Referring the Workbooks

Workbooks can be referred by the Workbooks property. In Figure 94 we have two macro enabled workbooks first is Book1 and second is Book2.

In this figure we have used Workbooks("Book1.xlsm").Activate , it means we are saying to excel please select Book 1 for us. After running this code workbook Book1 will be selected for work.

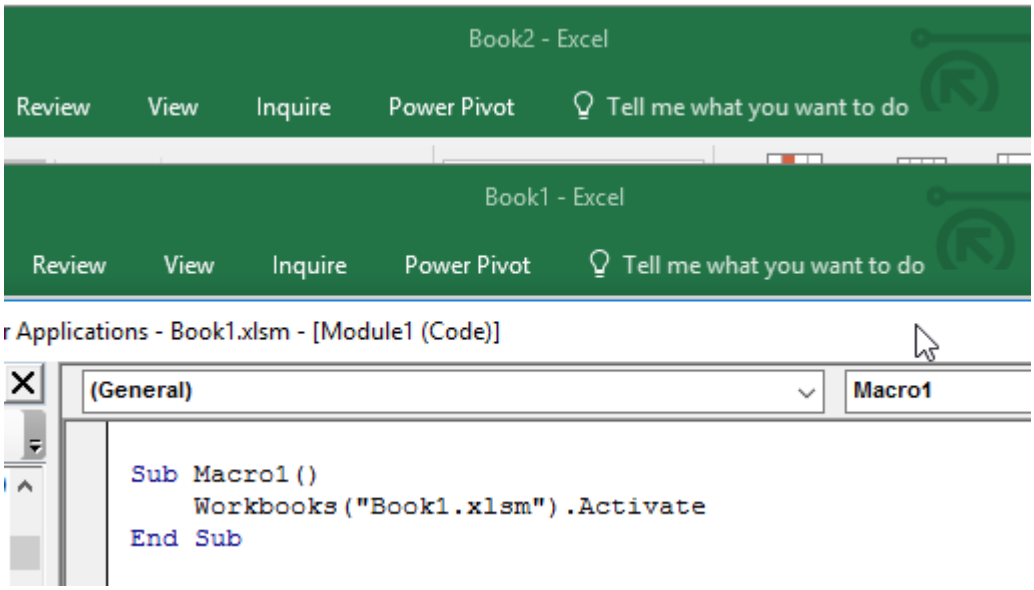


Figure 94

Author Note: if you want to select workbooks other then Macro enabled workbooks then you have to care about the extension i.e. for 2007 workbook use xlsx for 2003, workbooks use xls and so on.

Referring the workbooks by index number

It is same as Indexing through worksheet number.

Opening the workbooks

Workbooks can be opened through OPEN method in VBA

Here is the syntax of the OPEN method

expression .Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword, IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify, Converter, AddToMru, Local, CorruptLoad)

For opening of workbooks you have to remember the location of your workbooks

For instance if your file location is “D:\new folder\ Book1.xlsm” then you have to type this location in the VBA also , so that VBA can refer the location and open your file

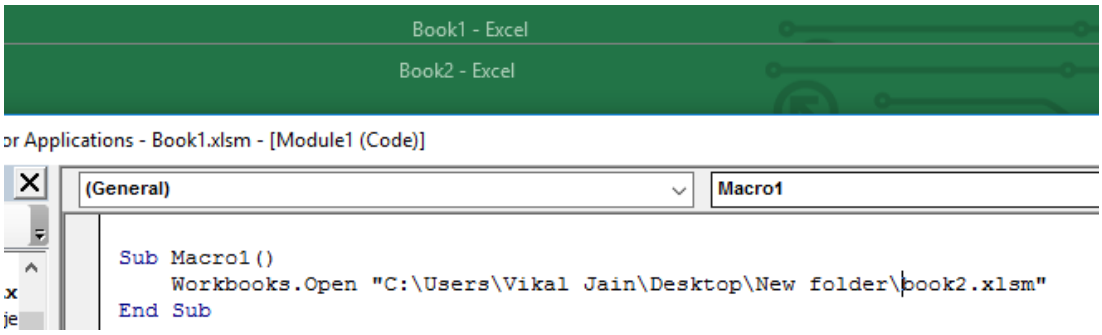


Figure 95

Author Note: The syntax of Open method is too long and difficult to understand, since this book is dealing about the basic learning series, so you can refer the Google to get the advance knowledge for it. The expanded version and link of Microsoft are also given in the Table 17

Closing the workbooks

Closing of workbooks can be done through Close method as shown in Figure 96

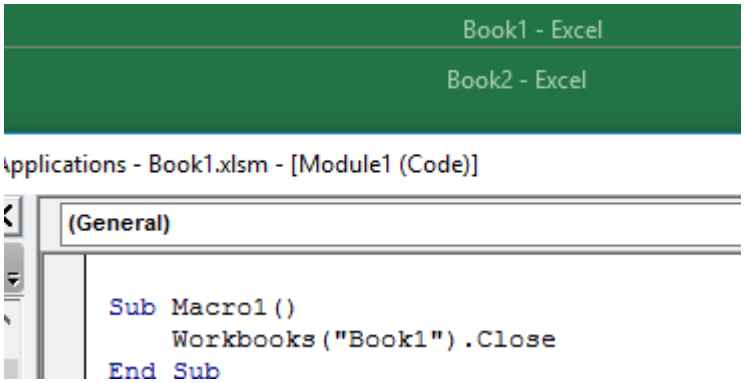


Figure 96

When you will run the code given in the Figure 96 then a pop up window will open as given in Figure 97

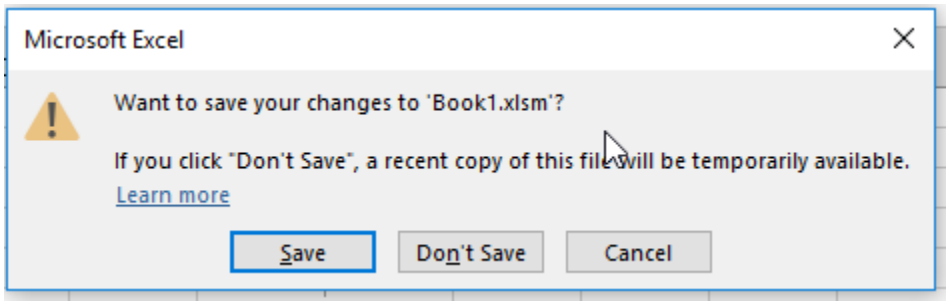


Figure 97

If you don't want to see this window and you always want to save your file then you can add true after the Close method as given in Figure 98. If you will write false then excel will not save your file and will save the unsaved version.

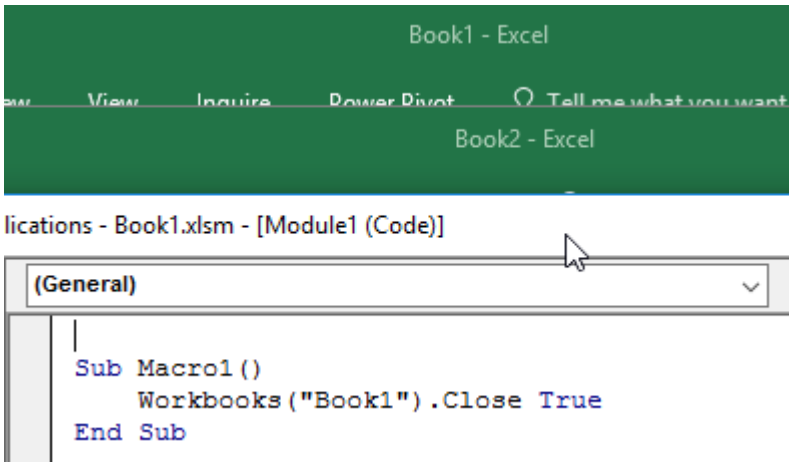


Figure 98

Creating the workbooks

ADD method is used to add the new workbooks as given in Figure 99

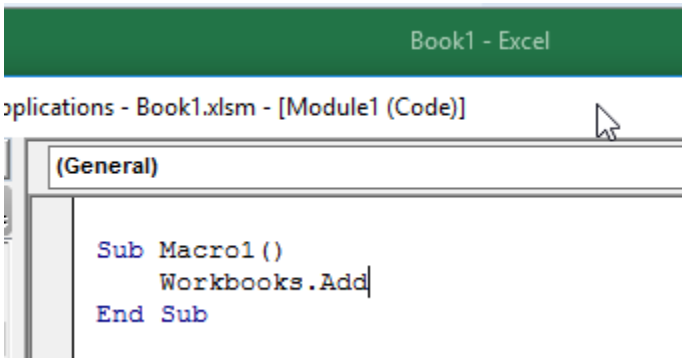


Figure 99

Saving the workbooks

SAVE method can help in the saving the workbook. Workbook saving is also so simple. See Figure 100

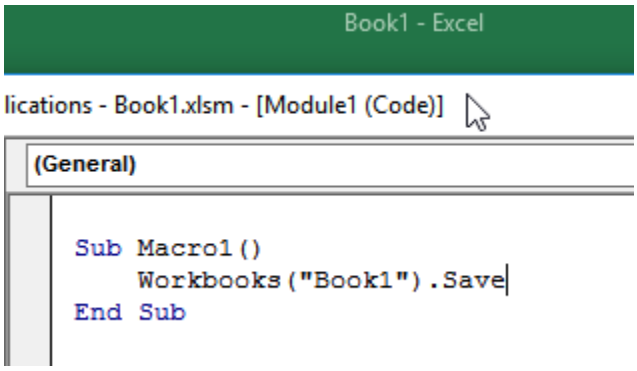


Figure 100

Use SAVEAS

SAVEAS works in same manner as OPEN method do. You have to specify the location. See Figure 101

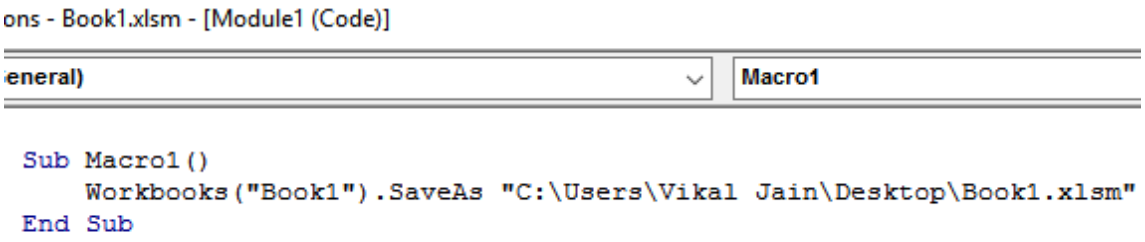


Figure 101

Chapter 10

VARIABLES

Why Variable

VBA stores data in memory using Variables. A variable is a name given by you, to which you assign a piece of data that is stored in an area of the computer's memory, allowing you to refer to that data when you need to later in the macro.

For instance in Figure 102 Cell A1 value is the 125 and it is dynamic i.e. can be changed every time. So you declare the variable that Myvalue must we the value of cell a1. Now when you will change this A 1 cell value then Variable value will get change.

Another advantage is that our code gets shortened through Variable. It is typically hard to write Range("A1").Value again and again. Instead, you can simply write Myvalue in next line of code.

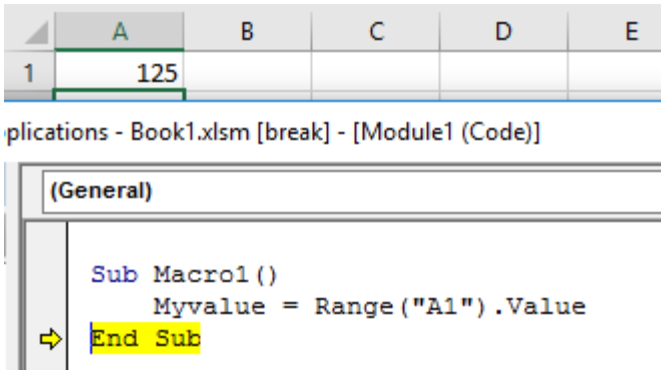


Figure 102

How to declare variable

Without Dim Statement

This is the simplest method to define the Variable in VBA. What you need to do just put, variable name in the front of the code as shown in

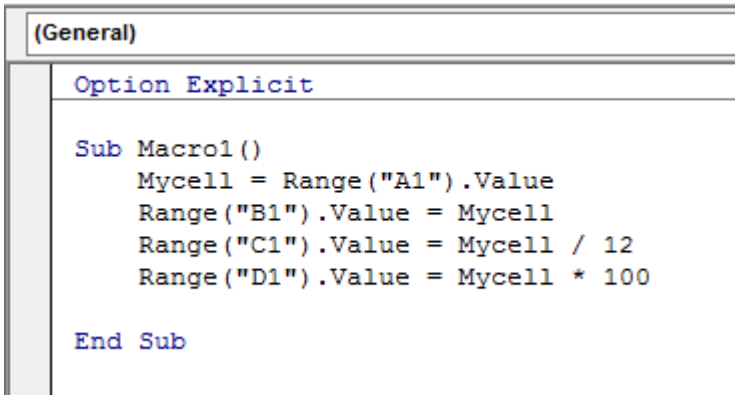


Figure 103

As this method is simple, however there are lots of limitations inherit with this procedure such as

Variable always defined as variant type which is discussed in Table 10

Error tracing is more complicated in this method

With Dim Statement

Basically Dim statement is used to declare the variable.

If we use layman language then DIM statement can be defined as follows

DIM….YOURVARIABLE NAME….AS….TYPE OF DATA BEING STORED

12 34

For defining Dim statement you need 2 things now

Your Variable Name

Before we go into some detail of Variables, there are a few important rules that you must know about.

A Variable name must Start with a letter and not a number. Numbers can be included within the name, but not as the first character.

A Variable name can be no longer than 255 characters.

A variable cannot contain space or special characters like @, $, &, (and so on.

A Variable name cannot be the same as any one of Excel's key words. By this, I mean you cannot name a Variable with such names as Sheet, Worksheet etc.

All Variables must consist of one continuous string of characters only. You can separate words by either capitalising the first letter of each word, or by using the underscore characters if you prefer.

Author Note: In simple way basically, when it comes to naming your variables, keep it simple. Use only letters and maybe numbers after the first character.

Data Type

The following table shows the Visual Basic data types, their supporting common language runtime types, their nominal storage allocation, and their value ranges.

Table 10

Visual Basic type	Common language runtime type structure	Nominal storage allocation	Value range

Boolean	Boolean	Depends on implementing platform	True or False
Byte	Byte	1 byte	0 through 255 (unsigned)
Char (single character)	Char	2 bytes	0 through 65535 (unsigned)
Date	DateTime	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	Decimal	16 bytes	0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) † with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest nonzero number is +/-0.0000000000000000000000001 (+/-1E-28) †
Double (double-precision floating-point)	Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324 † for negative values; 4.94065645841246544E-324 through 1.79769313486231570E+308 † for positive values
Integer	Int32	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long (long integer)	Int64	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807 (9.2...E+18 †) (signed)

Object	Object (class)	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	SByte	1 byte	-128 through 127 (signed)
Short (short integer)	Int16	2 bytes	-32,768 through 32,767 (signed)
Single (single-precision floating-point)	Single	4 bytes	-3.4028235E+38 through -1.401298E-45 † for negative values; 1.401298E-45 through 3.4028235E+38 † for positive values
String (variable-length)	String (class)	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	UInt32	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	UInt64	8 bytes	0 through 18,446,744,073,709,551,615 (1.8...E+19 †) (unsigned)
User-Defined(structure)	(inherits fromValueType)	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members
UShort	UInt16	2 bytes	0 through 65,535 (unsigned)

Source : https://msdn.microsoft.com/en-IN/library/47zceaw7.aspx

You can also define your variable without using any DIM statement like I did in the Figure 102.

Let's understand Variable in more detail

What if Variable will not be there

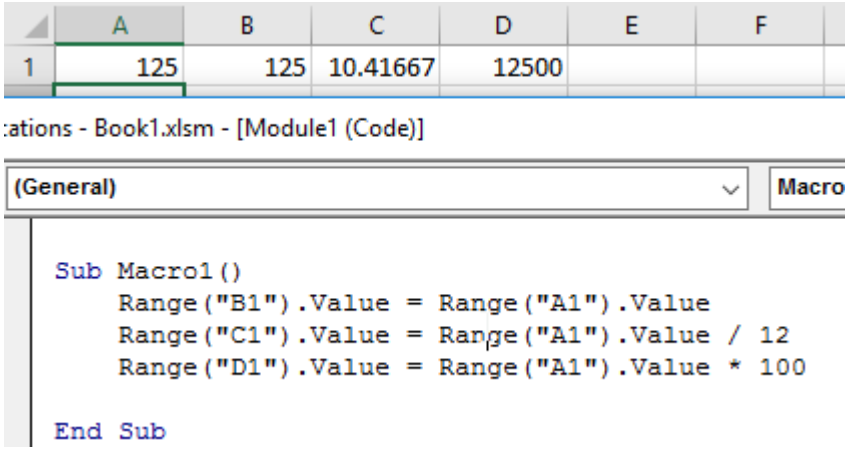


Figure 104

Here Cell B1 value is equal to the Cell A1,

Cell C1 Value is calculated by dividing the 12 from the Value of cell A1 and

Cell D1 Value is calculated by multiplying the 100 into the Value of cell A1.

This can be so typical when there will no Variable

Let's do this with the help of Variable

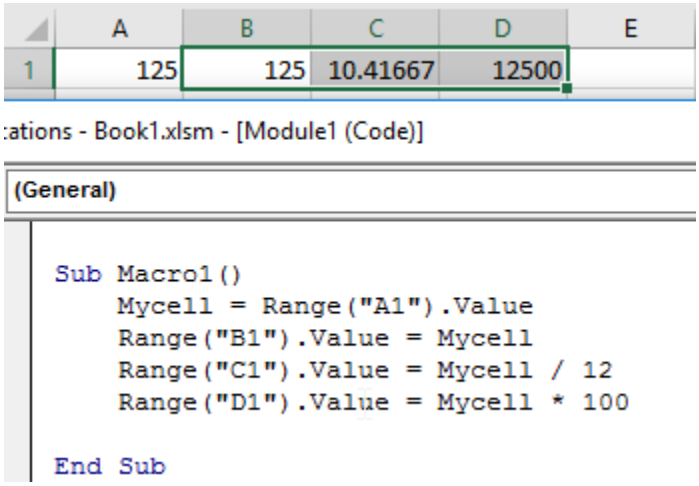


Figure 105

In the Figure 105 we have defined the Mycell as Range("A1").Value. If means it will store the value of Cell A1 in it memory and perform the tasks accordingly.

Author Note: Mycell = Range("A1").Value and Range("A1").Value = Mycell are different in EXCEL VBA since in the former case Mycell is storing the value of Cell A1 and in the later case Cell A1 is storing the Mycell value of defined earlier.

Data Types

Data types are the different kinds of ways you can store data in memory. Table 10 shows a list of common data types with their descriptions and memory usage.

Declaring date Variable

You can assign values to a date variable by enclosing them in the # (number sign) character,

For example

Mydate = #04 October 1991#

Or

Mytime = #04:10 Am#

Or

Mydate2 = #October 04, 1991#

Or

Mytime2 = # 04/10/1991 04:10pm#

Other Data Types

To assign a value to a Numeric or String type Variable, you simply use your Variable name, followed by the equals sign (=) and then the String or Numeric type. eg:

Sub ParseValue()

Dim MyWord as String

Dim MyNumber as Integer

MyWord = Range("A1").Text

MyNumber = Range("A1").Value

End Sub

In the above line of code we have declare Myword as string since it can contain letters, numbers, spaces, and punctuation and My Number as Integer since it can contain number from -32,768 to 32,767.

There are lots of technique to define the Variable, Smart programmers may use

Dim myValue1 as Integer, myValue2 as Integer, myValue3 as Integer

But if you are beginners you can make this mistake also

Dim myValue1, myValue2, myValue3 as Integer

Author Note: If you do not specify a data type after a variable name, such as in the latter case with myValue1 and myValue2, VBA assigns the default Variant data type. Only the Value3 variable has been specified the Integer data type. Variant is a catch-all data type that is the most memory-intensive, and the least helpful in understanding

Option Explicit

Another way to reduce is the force your variable through Option Explicit. What you have to do just go to on the top of your module and write Option Explicit shown in Figure 106

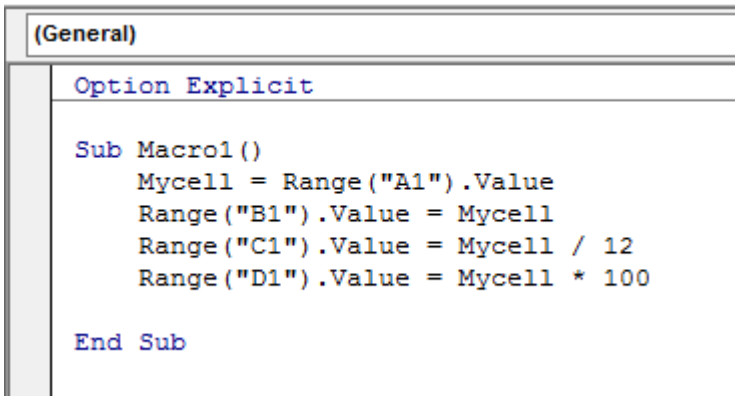


Figure 106

This will force your variable to perform with the DIM statement. If you will not use Dim statement then it will show compile error as shown in Figure 107

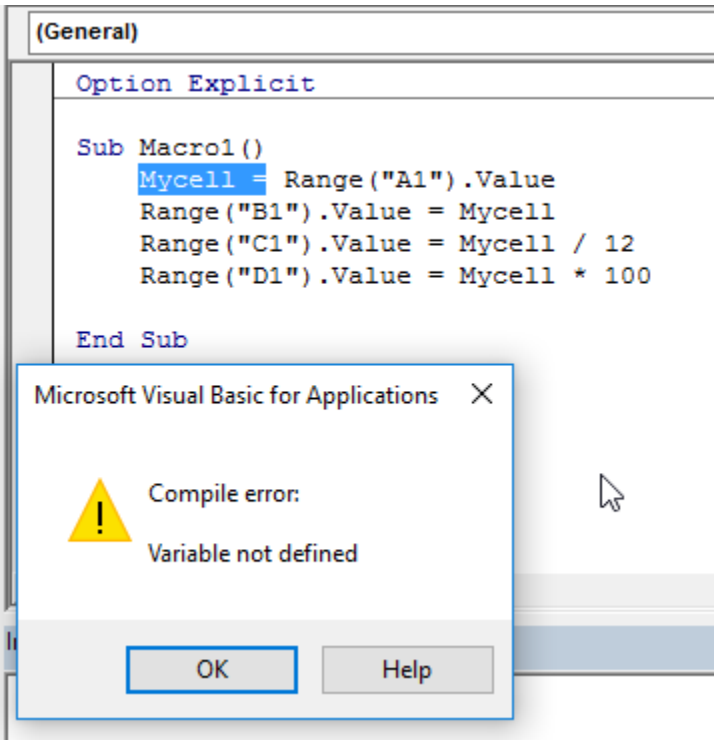


Figure 107

You can also enable the automatic forcing of variable through these steps

Step 1 : Go to Tools menu and click on the options

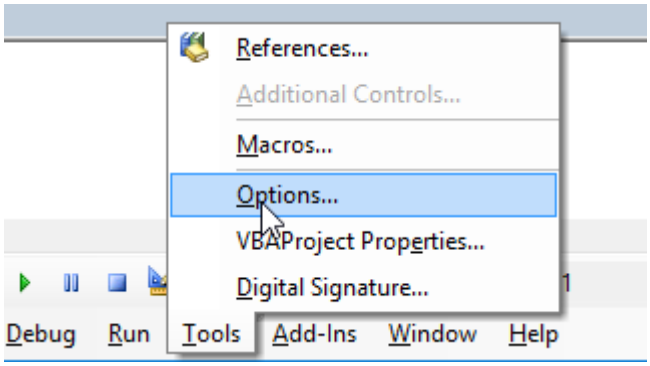


Figure 108

Step 2: Click on the check box "Require Variable declaration" in the editor menu then click ok. Now every time your module will show Option Explicit.

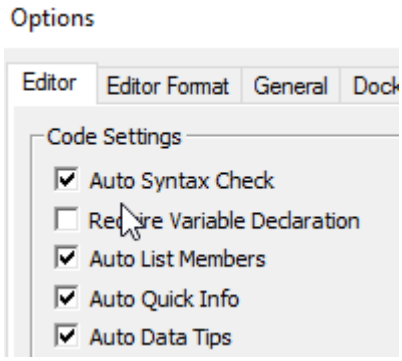


Figure 109

Where to store Variable

Variable cannot be stored whole lifetime in the computer memory, it has some specific time period to perform.

Local Macro level only

When you will perform a code then as and when your code will end your variable will become useless.

For example in the Figure 110 Variable Mycell will end after the performing of Macro1. In Macro2 the Value of Mycell will go to empty again and it will produce some errors

```
Sub Macro1()
    Dim Mycell As Integer
    Mycell = Range("A1").Value
    Range("B1").Value = Mycell
    Range("C1").Value = Mycell / 12
    Range("D1").Value = Mycell * 100
End Sub
Sub Macro2()
    Dim Mycell2 As Integer
    Mycell2 = Mycell * 100
End Sub
```

Figure 110

Module level

In this level your variable will come to end after the performing of whole module, for example in Figure 111 Variable Mycell will come to end after performing of MACRO1 and MACRO 2.

```
Dim Mycell As Integer
Sub Macro1()
    Mycell = Range("A1").Value
    Range("B1").Value = Mycell
    Range("C1").Value = Mycell / 12
    Range("D1").Value = Mycell * 100
End Sub
Sub Macro2()
    Dim Mycell2 As Integer
    Mycell2 = Mycell * 100
End Sub
```

Figure 111

Application Level

Finally, you can declare the variables as Public, which will make them visible to all macros in all modules as shown in Figure 112.

```
Public Mycell As Integer
Sub Macro1()
    Mycell = Range("A1").Value
    Range("B1").Value = Mycell
    Range("C1").Value = Mycell / 12
    Range("D1").Value = Mycell * 100
End Sub
Sub Macro2()
    Dim Mycell2 As Integer
    Mycell2 = Mycell * 100
End Sub
```

Figure 112

Chapter 11

OBJECT VARIABLE

What is Object Variable

Object variable is used to represent the Object, Now the question arise is What is OBJECT.

When we are going to select more than one cells i.e. range or worksheet then it is called as object.

Declare Object Variable

Declaring object variable is as simple as declaring the normal variable you have to follow the same set of rules as given in "With Dim Statement".

For example

Dim Myrange as Range

Use SET word to define object Variable

This is the key difference between normal variable and Object variable that you have to use SET word to assign the value to Object variable.

SET Myrange = Worksheets("Sheet1").Range("A1:B10")

SET Mysheet = Activesheet

SET newworkbook = Workbooks.add

Destroy Variables

A good programmer always destroy variable at the end of the programming so that it cannot effect our future operations and tasks.

How to destroy your variables?

Use NOTHING to destroy the every variable for example

Myrange = Nothing

SET Mysheet = Nothing

Chapter 12

MAKING DECISIONS

If we will talk about EXCEL then we can take decisions through various functions such as

IF

And

Or

Not

If Error and so on

However we cannot simply use these functions if we are using VBA, for VBA we have others type of decisions making tools like

If then

If then else

And

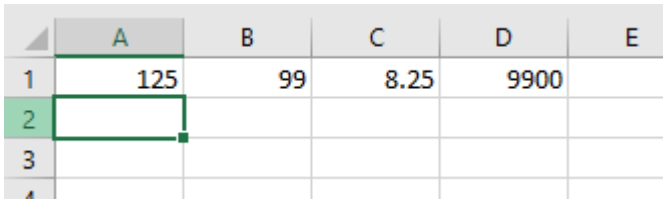
Or

Not

Select case

If ....then

If then is the simplest form of decision making tool in VBA . It works in same manner as the IF function do in the Excel.



	A	B	C	D	E
1	125	99	8.25	9900	
2					
3					

Figure 113

If we will talk about the If function in EXCEL then it has three arguments in its syntax.

If(logical test, value_if_true ,value_if_False)

Let's use If now. In the Figure 113 if our A1 cell value will exceed the 100 then we wil eligible for reward , so we want a message whether we are eligible for reward or not.

In cell A2 we can use =If(A1>100, "Eligible for reward", "Not eligible")

In VBA it also works in the same manner but there is no condition for False argument.

For example

```
Public Mycell As Integer
Sub Macro1()
    Mycell = Range("A1").Value
    Range("B1").Value = Mycell
    Range("C1").Value = Mycell / 12
    Range("D1").Value = Mycell * 100
    If Mycell > 100 Then
    MsgBox "Eligible for reward"
    End If
    End Sub
```

Figure 114

In this code we have provided Mycell as Variable and we are putting a condition that

If the value of Mycell is exceeding than the 100 then a message box will pop up as shown in figure Figure 115

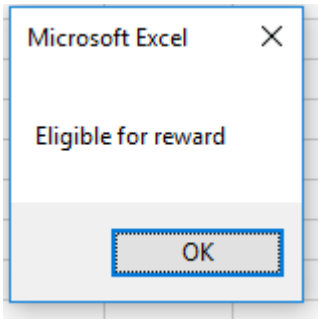


Figure 115

Author Note: If…then structure must be end with Endif otherwise EXCEL will get confused.

If….then….else

If we will talk about If then else then it also provides False arguments for If function, as If then don't.

```
(General)
Public Mycell As Integer
Sub Macro1()
    Mycell = Range("A1").Value
    Range("B1").Value = Mycell
    Range("C1").Value = Mycell / 12
    Range("D1").Value = Mycell * 100
    If Mycell > 100 Then
        MsgBox "Eligible for reward"
    Else
        MsgBox "Not Eligible"
    End If
    End Sub
```

Figure 116

See in the Figure 116we have also provided the False arguments of the if function, now as and when the value of the Mycell will get lower than 100 it will provide different message.

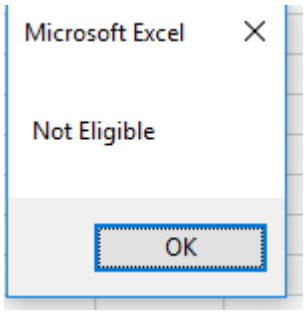


Figure 117

If....then....Elseif

This is as same as Nested if in the EXCEL.

Excel

=IF(A1>125,"Double Bonus",IF(A1>100,"Eligible for Bonus","Not Eligible"))

VBA

If Mycell > 125 then

Msgbox “Double Bonus”

Elseif Mycell > 100 then

Msgbox “Eligible for Bonus”

Elseif Mycell <= 100 then

Msgbox “Not Eligible”

End if

Select Case

Your macros will sometimes need to take into consideration not just one, two, or five courses of action, but possibly ten, hundreds, or even thousands depending on the situation.

For these complex evaluations, the SELECT CASE statement is a perfect solution.

Let me show you some example to explain the Select case

Example 1

```
Public Mycell As Integer
Sub Macro1()
    Mycell = Range("A1").Value
    Range("B1").Value = Mycell
    Range("C1").Value = Mycell / 12
    Range("D1").Value = Mycell * 100

    Select Case Mycell
        Case Is > 125: MsgBox "double bonus"
        Case Is > 100: MsgBox "Eligible for Bonus"
        Case Is <= 100: MsgBox "Not Eligible"
    End Select

    End Sub
```

Figure 118

Example 2

Sub CurrentQuarter()

Select Case Month(VBA.Date)

Case 1 To 3: MsgBox "Quarter 1"

Case 4 To 6: MsgBox "Quarter 2"

Case 7 To 9: MsgBox "Quarter 3"

Case 10 To 12: MsgBox "Quarter 4"

End Select

End Sub

As you can see, you don't need 12 separate statements to handle each conditional month; you can simply state the range of months using the To statement in each Case.

If we summaries then we can conclude that the working of Select case as same as If....then....Elseif.

The procedure of writing the Select case is as follows

Select Case your test-Expression

Case your question

Your answer

Case your question2

Your answer2

Case your question2

Your answer 2

End Select

NOT, OR & AND

These decision makers are also work in the same manner as they do in the EXCEL.

NOT

The NOT operator inverts an expression's True or False evaluation.

Table 11

If expression is	The value of result is
True	False
False	True

Dim a As Integer = 10

Dim b As Integer = 8

Dim c As Integer = 6

Dim firstCheck, secondCheck As Boolean

firstCheck = Not (a > b)

secondCheck = Not (b > a)

The preceding example produces results of False and True, respectively.

Source : https://msdn.microsoft.com/en-us/library/2cwcswt4.aspx

OR

The OR operator performs a logical disjunction, whereby if either condition is True, or if both conditions are True, the result is True. If both conditions are False, the OR operation results in False.

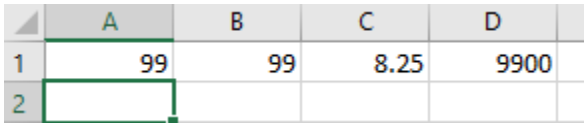


Figure 119

Range("A1").value >100 or Range("D1").value >10000

If any of the condition from the both conditions will true and Or will consider as True.

AND

The result of the AND operation is True only if both conditions are True.

Range("A1").value >100 and Range("D1").value >10000

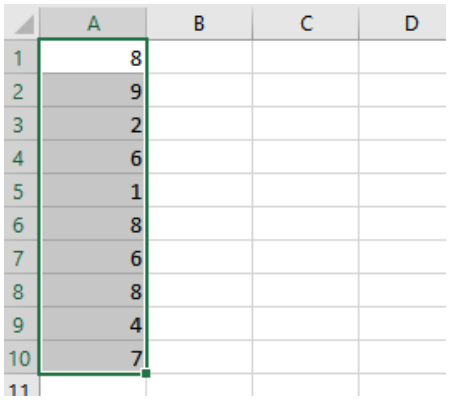
If both condition are correct then only And operator will evaluate true condition otherwise it will evaluate False.

Chapter 13

LOOPING

If you need to perform the same task (i.e. same piece of code) multiple times, then looping can be used.

This example will provide you good understanding about the looping.



	A	B	C	D
1	8			
2	9			
3	2			
4	6			
5	1			
6	8			
7	6			
8	8			
9	4			
10	7			
11				

Figure 120

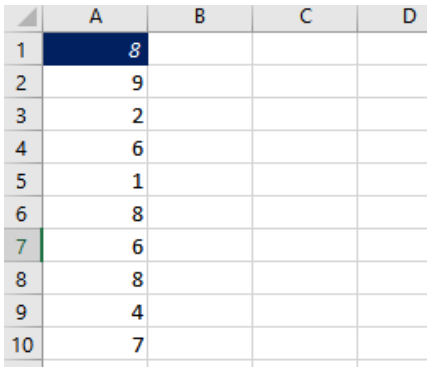
In the Figure 120 we have data from Cell A1 to A10 what I am going to do. I am going to format the cell one by one for the purpose of audit.

I will make the Fill Color = Dark Blue

Font Color = White

Text = Italics

So I will record a macro, assign shortcut CTRL + Q and perform my tasks



	A	B	C	D
1	*8*			
2	9			
3	2			
4	6			
5	1			
6	8			
7	6			
8	8			
9	4			
10	7			

Figure 121

Now for cell A1 to A10 we have to press CTRL + Q every time for the same formatting. This seems to be so easy but, think about large scale of data then we will need looping since we can't

run our macro every time to do a single task. With the help of loop we are able to run our macro only once then our all tasks will be done in one shot.

For… Next loop

The For…Next loop structure is a simple and effective way to repeat an action for a specified number of times.

In the same example given above we will format all cells in one shot with the following code

```
Sub Macro3()
'
' Macro3 Macro
'
' Keyboard Shortcut: Ctrl+Shift+Q
'
    lastrow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 To lastrow
      With Cells(i, 1).Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 6299648
        .TintAndShade = 0
        .PatternTintAndShade = 0
        End With

        With Cells(i, 1).Font
        .ThemeColor = xlThemeColorDark1
        .TintAndShade = 0
    End With
Next i
    Selection.Font.Italic = True
End Sub
```

Figure 122

In the Figure 122 we have written our code in the For….Next Loop

Steps we have taken in the above code

Define Last row to check the last loop position

Use For….Next, so that we can perform our code in one shot

Done formatting

Use STEP

The loop uses the default step size of 1, when looping from 1 to i. However, you may sometimes want to step through a loop using different sized steps. This can be done using the Step keyword, as shown in the following example

```
Sub Macro3()
'
' Macro3 Macro
'
' Keyboard Shortcut: Ctrl+Shift+Q
'
    lastrow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 To lastrow Step 2
        With Cells(i, 1).Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 6299648
        .TintAndShade = 0
        .PatternTintAndShade = 0
        End With

        With Cells(i, 1).Font
        .ThemeColor = xlThemeColorDark1
        .TintAndShade = 0
        End With
        Selection.Font.Italic = True
 Next i
End Sub
```

Figure 123

In the Figure 123 we have used step size 2 it means, it will switch the i value to 1,3,4.... and produce the result like

	A	B
1	8	
2	9	
3	2	
4	6	
5	1	
6	8	
7	6	
8	8	
9	4	
10	7	

Figure 124

We can use the step size as negative value but for that we have to write the i value from large value to small value i.e. i = lastrow to 1 step -2. This means now vba will take i value as 10, 8, 6...

For Each….Next Loop

The working of For Each….Next loop is as same as the For….next loop. However instead of working for normal variable it works for OBJECT variable.

```
Sub CloseWorkbooks()
'Declare your object variable.
Dim wb As Workbook
'Open the For loop structure.
For Each wb In Workbooks
        'Enter the command(s)that will be repeated.
        If wb.Name = “Book.xlsx” then
        wb.Save
        wb.Close
        End If
        'Loop to the next iteration.
Next wb
End Sub
```

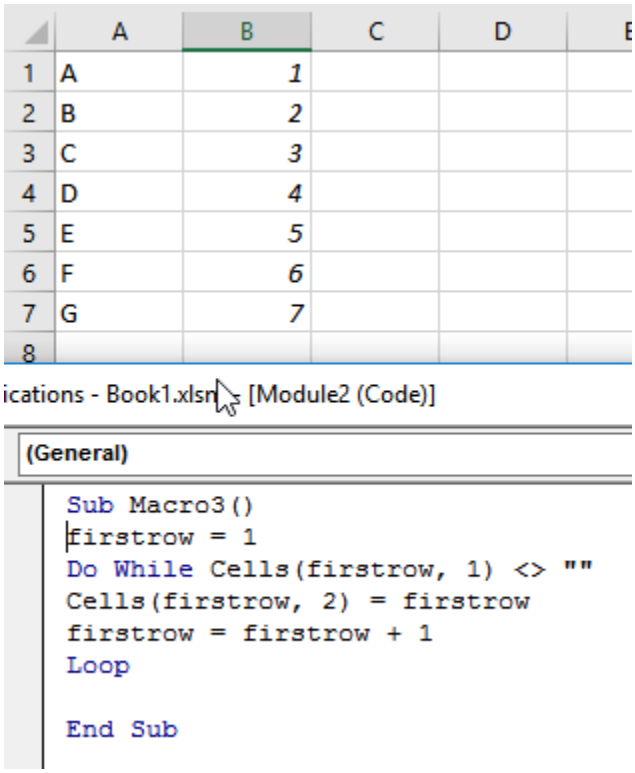
Existing a For….. Loop (EXIT FOR)

If you want to exit from For…loop early, for example you are looking for book1 in your opened workbook and after getting book1 in the loop, it is no use to perform the loop for other workbooks since it will make your macro slow responder in that case you can use EXIT FOR.

```
Sub CloseWorkbooks()
'Declare your object variable.
Dim wb As Workbook
'Open the For loop structure.
For Each wb In Workbooks
        'Enter the command(s)that will be repeated.
        If wb.Name = “Book1.xlsx” then
        wb.Save
        wb.Close
Exit For
        End If
        'Loop to the next iteration.
Next wb
End Sub
```

DO WHILE

In a Do While loop, you test for a condition that must be True before the loop will execute. When the condition is True, the command(s) within the loop are executed.



	A	B	C	D
1	A	1		
2	B	2		
3	C	3		
4	D	4		
5	E	5		
6	F	6		
7	G	7		
8				

ications - Book1.xlsm [Module2 (Code)]

(General)

```
Sub Macro3()
firstrow = 1
Do While Cells(firstrow, 1) <> ""
Cells(firstrow, 2) = firstrow
firstrow = firstrow + 1
Loop

End Sub
```

Figure 125

In Figure 125 we have putted the S. No. in B column, based on DO WHILE LOOP, in this condition we have assigned the value on the basis of column A. if the any cell of column A will be blank then our loop will stop.

Do Until

DO UNTIL LOOP working is purely based on DO WHILE LOOP, but the difference that, DO WHILE responds on the condition that must be true, however DO UNTIL responds only on that condition that must be false.

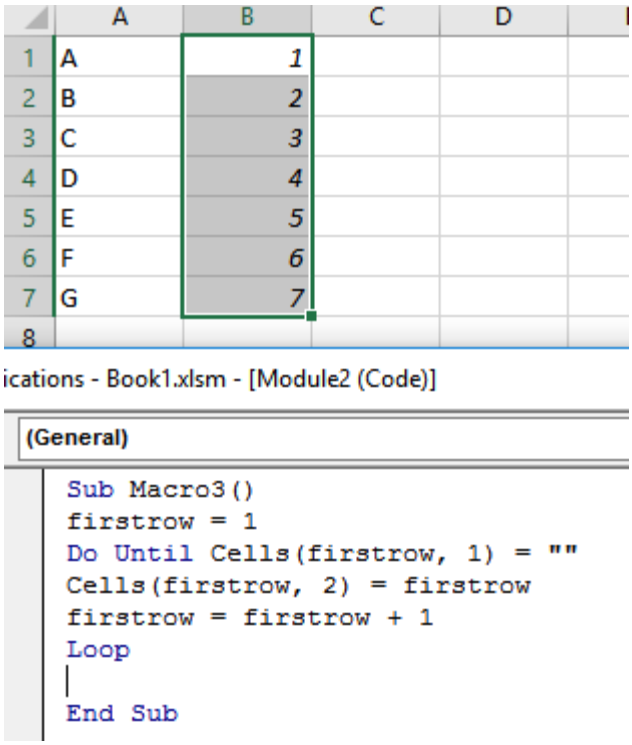


Figure 126

While Wend

While wend works in the same manner as DO WHILE do. While wend becomes obsolete now. VBA still supports While…Wend loops for backward compatibility with prior versions of Excel

Chapter 14

VBA IS OVER

Yes, VBA is over now. As per my concern you have learnt the Basic VBA.

Now the question is, how it could be possible that some basic chapters taught you whole VBA, we still have not discussed any property in detailed, any method, any chart type, any formatting code and so on. Believe me it is over, let me show now.

Here is the Secret, Why don't you record your macro and change some basic tools. It is no need to learn everything, BE SMART AND BE CODE CHEATER.

Example 1

	A	B
1	A	
2	B	
3	C	
4	D	
5	E	
6	F	
7	G	

Figure 127

In Figure 127 we are going to color active cell. Means on which our curser will be placed, the font color of that cell would be white and Background color of that cell would be Black.

Start recording and do the same formatting which i told you

Go to any cell

Make background color Black of that cell

Make font color white.

Now open Visual Basic editor and got to module 1 it will show your code as shown in Figure 128

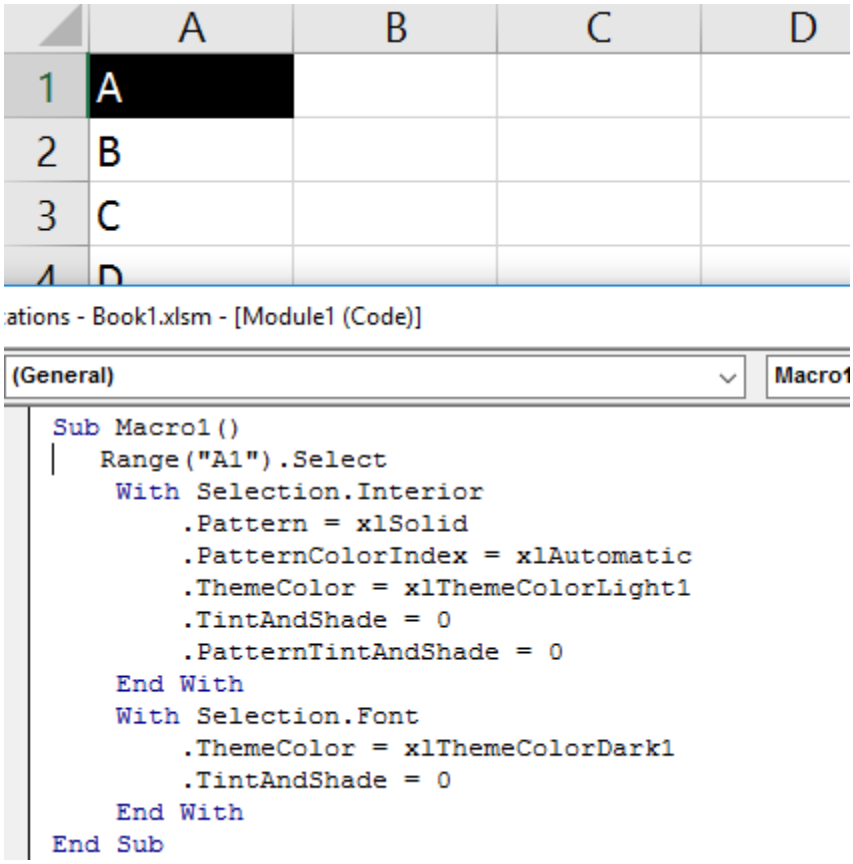


Figure 128

Noticed second line “Range("A1").Select” it is showing that every time macro will select Cell A1. Now change till line to “ActiveCell.Select” as shown in Figure 129

```
Sub Macro1()
   Activecell.Select
    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .ThemeColor = xlThemeColorLight1
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
    With Selection.Font
        .ThemeColor = xlThemeColorDark1
        .TintAndShade = 0
    End With
End Sub
```

Figure 129

Your whole code is ready. Did you notice?

Example 2

This time I am going to make some EXCEL sheets with the macro recorder with the name of values of cells in column A i.e. first sheet will be A, second will be B and so on till the last cell value.

Let's create a new folder first on your desktop; only then you are able to save your files on the same folder.

Here is the location of My folder

"C:\Users\Vikal Jain\Desktop\New folder (2)"

I am going to make 7 sheets. With the name of A,B,C,D,E,F,G as shown in Figure 130

	A	B
1	A	
2	B	
3	C	
4	D	
5	E	
6	F	
7	G	

Figure 130

Let's start recording Macro, here are the steps again.

Press Ctrl + N to open a new workbook

Save your file as A in your folder and

Close your workbook

Stop recording

```
Sub savemyfiles()
'
' savemyfiles Macro
'

'
    Range("A1").Select
    Workbooks.Add
    ActiveWorkbook.SaveAs Filename:= _
        "C:\Users\Vikal Jain\Desktop\New folder (2)\A.xlsx", FileFormat:= _
        xlOpenXMLWorkbook, CreateBackup:=False
    ActiveWindow.Close
End Sub
```

Figure 131

In this case we have to determine, how many records in column A exist we have to go to last row so our macro will stop at last values

From

LAST ROW AND LAST COLUMN you can find the last row

Lastrow = Cells(Rows.Count, 1).End(xlUp).row

Now create a loop from first row to last row and concatenate with the folder name.

```
Sub Savemyfiles()
    lastrow = Cells(Rows.Count, 1).End(xlUp).Row
    For i = 1 To lastrow
    Mycellvalue = Range("A" & i).Value
    Workbooks.Add
    ChDir "C:\Users\Vikal Jain\Desktop\New folder (2)"
    ActiveWorkbook.SaveAs Filename:= _
        "C:\Users\Vikal Jain\Desktop\New folder (2)\" & Mycellvalue & ".xlsx", FileFormat:= _
        xlOpenXMLWorkbook, CreateBackup:=False
        ActiveWindow.Close
        Next i
    Application.WindowState = xlNormal
End Sub
```

Figure 132

Changes made to your code

```
Sub Savemyfiles()
  lastrow = Cells(Rows.Count, 1).End(xlUp).Row
  For i = 1 To lastrow
  Mycellvalue = Range("A" & i).Value
  Workbooks.Add
  ChDir "C:\Users\Vikal Jain\Desktop\New folder (2)"
  ActiveWorkbook.SaveAs Filename:= _
    "C:\Users\Vikal Jain\Desktop\New folder (2)\" & Mycellvalue & ".xlsx", FileFormat:= _
    xlOpenXMLWorkbook, CreateBackup:=False
    ActiveWindow.Close
    Next i
  Application.WindowState = xlNormal
End Sub
```

Figure 133

Name	Date	Type	Size	Tag
A	6/16/2016 7:52 AM	XLSX File	8 KB	
B	6/16/2016 7:52 AM	XLSX File	8 KB	
C	6/16/2016 7:52 AM	XLSX File	8 KB	
D	6/16/2016 7:53 AM	XLSX File	8 KB	
E	6/16/2016 7:53 AM	XLSX File	8 KB	
F	6/16/2016 7:53 AM	XLSX File	8 KB	
G	6/16/2016 7:53 AM	XLSX File	8 KB	

Figure 134

Example 3

	A	B	C	D	E
1	A	141	391	130	338
2	B	154	222	161	413
3	C	197	166	205	401
4	A	180	187	176	200
5	B	344	397	406	489
6	A	266	205	205	265
7	G	174	343	392	110

Figure 135

In this example i am going to filter the data set of Values A and i am going to insert a new sheet and copy into that sheet.

Start recording macro again

Here are the steps

Start recording

Go to cell A1

Pres Ctrl + Shift + L or Alt + D + F + F to apply filter

Filter Values A

Copy the entire selection

Insert the new sheet

Paste it on that sheet

Press esc to exit copy mode

Remove filter from our data set

Stop Recording

```
Sub Macro5()
'
' Macro5 Macro
    Range("A1").Select
    Selection.AutoFilter
    ActiveSheet.Range("$A$1:$E$7").AutoFilter Field:=1, Criteria1:="A"
    Range(Selection, Selection.End(xlToRight)).Select
    Range(Selection, Selection.End(xlDown)).Select
    Selection.Copy
    Sheets.Add
    ActiveSheet.Paste
    Application.CutCopyMode = False
    ActiveSheet.Next.Select
    Selection.AutoFilter
    Range("F13").Select
End Sub
```

Figure 136

In the above code see the line number 6 “ActiveSheet.Range("$A$1:$E$7").AutoFilter Field:=1, Criteria1:="A" this criteria is going to select till column E every time. Let’s make it dynamic to the last column.

For example if there is 4 more columns then EXCEL can filter that,

Find last column first from the

LAST ROW AND LAST COLUMN

Lastcol = Cells(1,Columns.Count).End(xltoleft).column

And concatenate with the filter data as shown in the Figure 137

```
Sub Macro5()
'
' Macro5 Macro
    lastcol = Cells(1, Columns.Count).End(xlToLeft).Column
    Range("A1").Select
    Selection.AutoFilter
    ActiveSheet.Range("$A$1:$E$" & lastcol).AutoFilter Field:=1, Criteria1:="A"
    Range(Selection, Selection.End(xlToRight)).Select
    Range(Selection, Selection.End(xlDown)).Select
    Selection.Copy
    Sheets.Add
    ActiveSheet.Paste
    Application.CutCopyMode = False
    ActiveSheet.Next.Select
    Selection.AutoFilter
    Range("F13").Select
End Sub
```

Figure 137

Macro can do everything, except these

Looping

Provide Message Box and Input boxes

Deal with IF statements

Chapter 15

FORMULAS

A1 Style & R1C1 style

If you work with excel in normal way, it means you only know about A1 style where A is the column and 1 is the row. However EXCEL behind the scene can understand only R1C1 style. Let me show you the exact picture of excel.

	A	B	C	D	E	F	G	H	I
1	A	141	391	130	338	295	316	302	326
2	B	154	222	161	413	200	110	375	336
3	C	197	166	205	401	222	250	180	362
4	A	180	187	176	200	307	310	154	138
5	B	344	397	406	489	126	242	265	111
6	A	266	205	205	265	343	262	194	175
7	G	174	343	392	110	349	275	109	229
8		*1456*	*1911*	*1675*	*2216*	*1842*	*1765*	*1579*	*1677*

Figure 138

In Figure 138, I have recorded macro for the sum and shown the totals in Row 8. When I will open Visual basic editor then it will be seem like this.

(General) Macro1

```
Sub Macro1()
    Range("B8").Select
    ActiveCell.FormulaR1C1 = "=SUM(R[-7]C:R[-1]C)"
    Range("B8").Select
    Selection.Copy
    Range("B8:I8").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
End Sub
```

Figure 139

What? We have done the total of B1:B7 in B8 as per Figure 138 but what VBA is showing? Yes, it is showing "=SUM(R[-7]C:R[-1]C)"

This is R1C1 style.

Learning of R1C1 style can improve your programming skills. R1c1 style can be understood through this example.

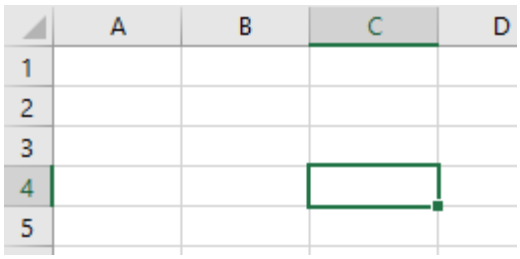


Figure 140

Cell C4 is identified by Excel as the address at the intersection of row 4 and column 3 (because column C is the third column from the left on the worksheet grid), which Excel interprets as R4C3. Cell M92 is interpreted as R92C13, and so on. As you might guess, the R1C1 address of cell A1 is R1C1

Enabling the R1C1 style in EXCEL

Go to File menu

Select EXCEL Options

Then go to Formula tab on the top left side of the window.

Click on the check box of the R1C1 style.

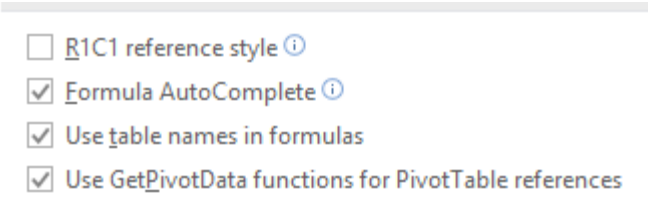


Figure 141

EXCEL Prior to the Enabling of R1C1 style

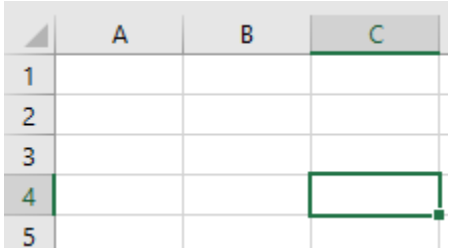


Figure 142

EXCEL after the Enabling of R1C1 style

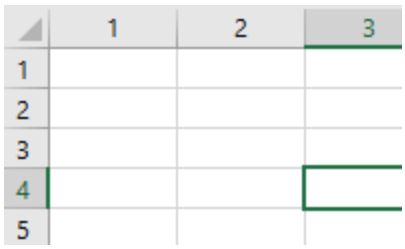


Figure 143

See the formula bar when, I have done sum of two figures

In A1 Style

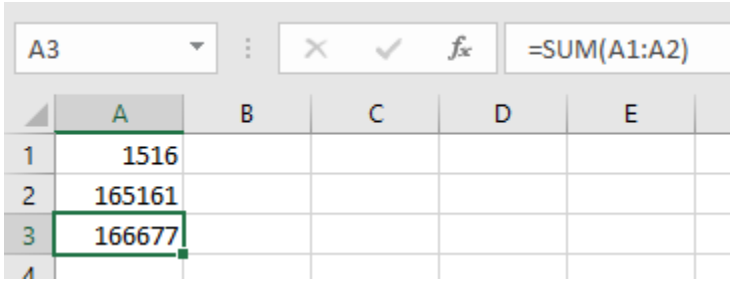


Figure 144

In R1C1 style

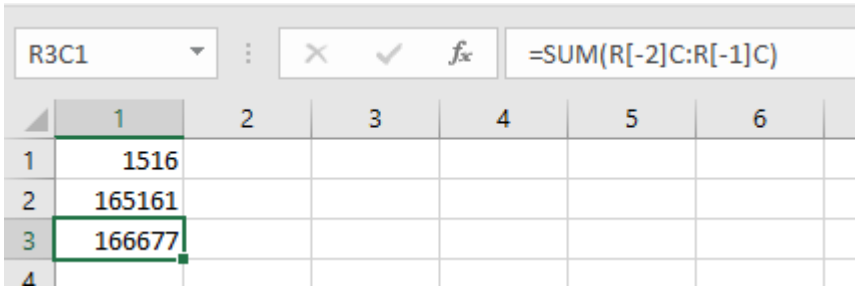


Figure 145

How does R1C1 style work?

As shown in Figure 146 when you will give reference to just one upper cell then it will Show =R[-1]C when you will give reference to two upper cells then it will show =R[-2]C and so on.

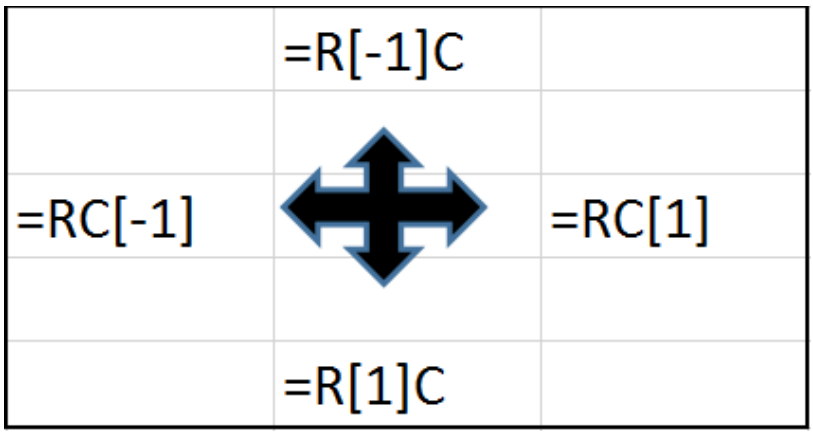


Figure 146

Absolute Vs Relative

The difference between these two can be understand with the help of the Table 12. In this table we are linking the value of the cell E3 to the Value of G5. It means when we will enter the value in the cell E3 then G3 will get updated automatically.

Table 12

	A1 Style	R1C1 style
Relative	ActiveSheet.Range("G5").Formula =E3	ActiveSheet.Range("G5").FormulaR1 C1=R[-2]C[-2]
Absolute	ActiveSheet.Range("G6").Formul a=$E$3	ActiveSheet.Range("G6").FormulaR1 C1=R3C5

So the square brackets make it relative. And not using square brackets makes it absolute.

Author Note: It is worth mentioning that the reference is handled differently when absolute vs. relative.

For relative references you are counting from the cell the formula is in. E3 is R[-2]C[-2] away from G5. i.e. 2 rows up, 2 column left.

For absolute values you are counting from cell A1. So E3 is R3C5 from cell A1. i.e. 3 rows down, 5 columns over.

Using Functions in the code

With the help of VBA you can use inbuilt functions of EXCEL, Make your own functions or customized the functions of EXCEL.

Functions same in VBA and EXCEL

There are lots of inbuilt functions in VBA that are same as EXCEL such as

Table 13

Function Name	What it does
Left	Extract left records from the values

Right	Extract right records from the values
Mid	Extract Middle Characters from the values of the cell
Ucase	Work as upper formula in EXCEL
Lcase	Work as Lower formula in EXCEL

Some inbuilt functions can't work directly

Functions shown as Table 13 can work directly without any further code. However there are lots of functions which cannot works directly in the EXCEL

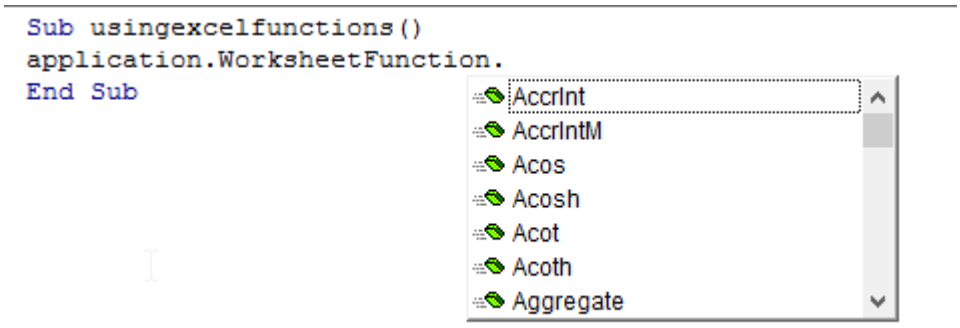


Figure 147

APPLICATION.WORKSHEETFUNCTION allows you to work with those functions which cannot be directly used in VBA, in my example I am going to use the VLOOKUP.

The one things you must know about VBA that it does not show the arguments of the EXCEL functions. For instance VLOOKUP (lookup_value, table_array, col_index_num, [range_lookup]) is the syntax of Vlookup but if we will use Vlookup in VBA then it will be shown as

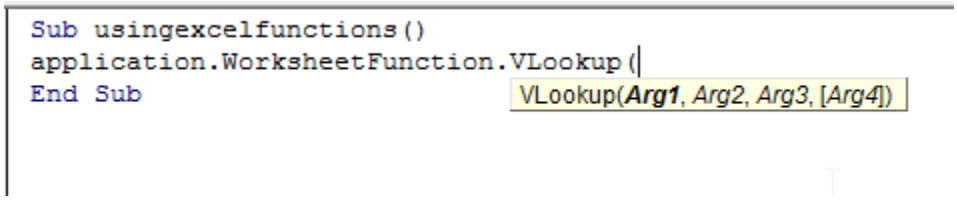


Figure 148

Therefore it is suggested by me, please use your functions first in EXCEL then use in VBA, otherwise VBA can produce absurd results.

	A	B	C	D	E	F	G	H	I
1	***Year***	***2009***	***2010***	***2011***	***2012***	***2013***	***2014***	***2015***	***2016***
2	A	141	391	130	338	295	316	302	326
3	B	154	222	161	413	200	110	375	336
4	C	197	166	205	401	222	250	180	362
5	D	180	187	176	200	307	310	154	138
6	E	344	397	406	489	126	242	265	111
7	F	266	205	205	265	343	262	194	175
8	G	174	343	392	110	349	275	109	229
9		*1456*	*1911*	*1675*	*2216*	*1842*	*1765*	*1579*	*1677*
10									
11		*Sales In 2015*							
12	*D*								

Figure 149

Using Vlookup, I am going to find the Value of sales made by D in year 2015. So this will be the code:-

```
Sub usingexcelfunctions()
Range("B12").Value = Application.WorksheetFunction. _
VLookup(Range("A12").Value, Range("A1:i8"), 8, False)
End Sub
```

Figure 150

Let's run this code now

10		
11		*Sales In 2015*
12	*D*	*154*
13		

Figure 151

Wow Vlookup worked.

Make your own functions in excel

Example 1:- Convert Centimeters into foot.

141	391	130	338	295	316	302	326
154	222	161	413	200	110	375	336
197	166	205	401	222	250	180	362
180	187	176	200	307	310	154	138
344	397	406	489	126	242	265	111
266	205	205	265	343	262	194	175
174	343	392	110	349	275	109	229

Figure 152

In Figure 152 I want to request excel that please show the height in Foot while the value are given in "cm". I will create a functions foot that will automatically convert the value into cm into Foot.

Here is the code

```
Function foot(mycell As Integer)
'1 foot = 30.48 cm
foot = mycell / 30.48
End Function
```

Figure 153

Now Foot function has created, now you can refer any cell to see the value in foot.

	Sales In 2015
D	5.05249 =foot(B3)

Figure 154

Example 2: Finding My workbook Path

As you can noticed that there is no function in excel which can show where the current EXCEL workbook is saved.

Here is the code

```
Function mypath()
mypath = activeworkbook.Path
End Function
```

Figure 155

Here is the result shown by excel

C:\Users\Vikal Jain\Desktop\New folder

Figure 156

Author Note: you must note that if you want to give the arguments to your code then you must type your arguments following by your function name like i did in Figure 153 and if you just want information from any function then you can just use parenthesis following by your function name as shown in Figure 155.

Customized function in EXCEL by VBA

We all know that Vlookup has four arguments, What if we can change it to one

	A	B	C	D	E	F	G	H	I
1	***Year***	***2009***	***2010***	***2011***	***2012***	***2013***	***2014***	***2015***	***2016***
2	A	141	391	130	338	295	316	302	326
3	B	154	222	161	413	200	110	375	336
4	C	197	166	205	401	222	250	180	362
5	D	180	187	176	200	307	310	154	138
6	E	344	397	406	489	126	242	265	111
7	F	266	205	205	265	343	262	194	175
8	G	174	343	392	110	349	275	109	229
9		*1456*	*1911*	*1675*	*2216*	*1842*	*1765*	*1579*	*1677*
10									
11		*Sales In 2015*							
12	*D*								

Figure 157

Figure 157 is showing our data set. Now i am again going to show you, finding the value of Sale made in 2015 by Employee D.

```
Function findSales(Select_Vendor)

findSales = Application.WorksheetFunction. _
VLookup(Select_Vendor, Range("A1:I8"), 8, False)

End Function
```

Figure 158

Figure 158 is showing customize code, now i will have to select only Employee Name and vlookup will show the value automatically.

	Sales In 2015	
E	*265*	***=findSales(A12)***

Figure 159

Isn't Amazing??

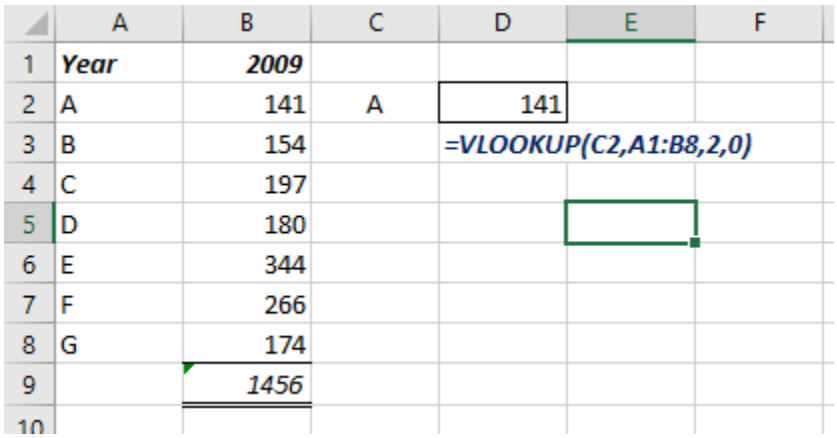
Chapter 16

ERROR HANDLING

There are three types of Error handlers in VBA, which are as follows:-

Table 14

S.No.	Term	Descriptions
1	Go to Line	This term will allow you to go to specific line of defined code. Which you have made in VBA
2	Go to 0	This disables enabled error handler in the current procedure and resets it to Nothing.
3	Resume Next	This term allow you to skip that line of item in the range and move to next values of your ranges



	A	B	C	D	E	F
1	***Year***	***2009***				
2	A	141	A	141		
3	B	154		=VLOOKUP(C2,A1:B8,2,0)		
4	C	197				
5	D	180				
6	E	344				
7	F	266				
8	G	174				
9		*1456*				
10						

Figure 160

In Figure 160, I have used simple Vlookup where I have found, the value of "A" using Vlookup. Let's set a message box for this value using VBA.

```
Sub showmyvalues()
myvalues = Application.WorksheetFunction. _
VLookup(Range("C2").Value, Range("A1:B8"), 2, False)
MsgBox myvalues
End Sub
```

Figure 161

When I will run the code given Figure 161 then it will show a message box as shown in Figure 162

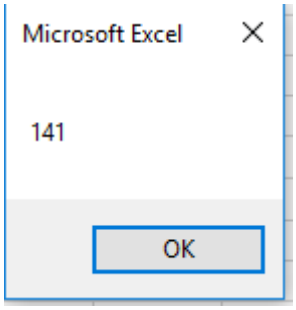


Figure 162

Let's put D in Cell C2, Now message box will show 180. Now try to put H in Cell C2,

	A	B	C	D	E	F
1	***Year***	***2009***				
2	A	141	H	#N/A		
3	B	154		***=VLOOKUP(C2,A1:B8,2,0)***		
4	C	197				
5	D	180				
6	E	344				
7	F	266				
8	G	174				
9		*1456*				

Figure 163

This time Vlookup is showing error as you can see in Figure 163. However when we will try VBA to check the working of message box. It will not run instead, it will show a pop up window as shown in Figure 164.

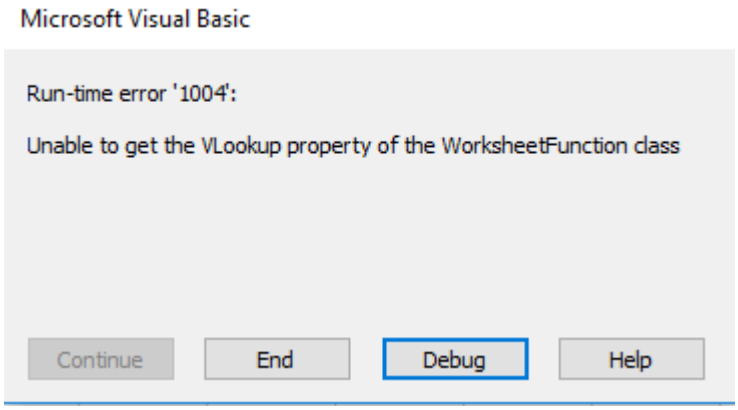


Figure 164

If you will not fix the Error handling then you have to fix the errors every time. This error is occurring, since there is no such value i.e. H exist in our data set.

Let's try to resolve the issues.

Method 1

On Error Goto …

In this case excel will go to next line of code as defined in the code and skip the code that can't be executed.

```
Sub showmyvalues()
On Error GoTo Valuenotexist
myvalues = Application.WorksheetFunction. _
VLookup(Range("C2").Value, Range("A1:B8"), 2, False)
MsgBox myvalues
Valuenotexist: MsgBox "Such Value not Exist"
End Sub
```

Figure 165

As shown in Figure 165 we have shown two variables first is "Myvalues" and the second is "ValuenotExist". We have defined "Myvalues" variable as Vlookup function. But we have defined a condition also that if the myvalues will show error, please skip this line of code and go to "ValuenotExist" i.e. show a message box that such value not exist.

Now VBA will show pop up window as follows

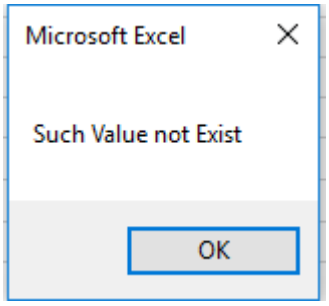


Figure 166

On Error Goto 0 (zero)

In this method VBA will set our result values as nothing. But before defining on "On Error Goto 0" we have to always define "On Error Resume Next" .

```
Sub showmyvalues()
On Error Resume Next
myvalues = Application.WorksheetFunction. _
VLookup(Range("C2").Value, Range("A1:B8"), 2, False)
    MsgBox myvalues
On Error GoTo 0

End Sub
```

Figure 167

Now this time we are saying to excel that, if you will face error then resume next line and the next line is define variable as 0, so message box will show results as shown in Figure 168

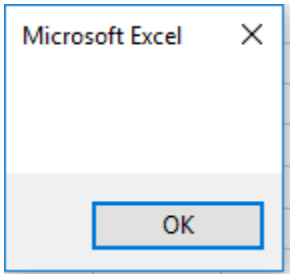


Figure 168

In short "On Error Goto 0" means forget the definition of variable and set it equal to nothing and "***On Error Resume Next***" means resume next line of code and skip the last line of code.

Chapter 17

User Forms

User form allows you to create customized forms that will restrict users to enter specific data in the sheets.

To insert the User forms just go to insert menu then insert Userform

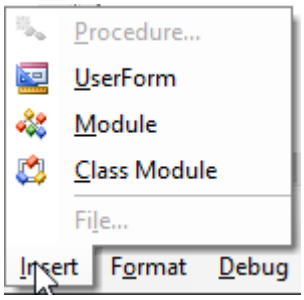


Figure 169

Then till will insert two windows first is the Userform window which allows you to build userforms according to you

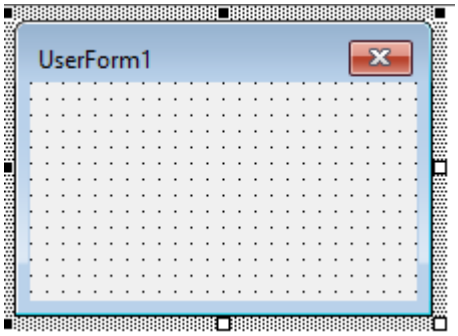


Figure 170

And second toolbox which allows you insert Checkboxes, combo boxes, drop and down boxes and so on according to your requirement.

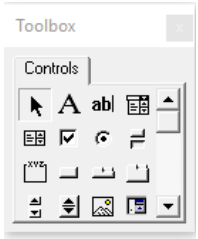


Figure 171

On the left bottom there is property window on the your VBA, this is very important tool since 99% time you have to use it.

Property window seems to be like

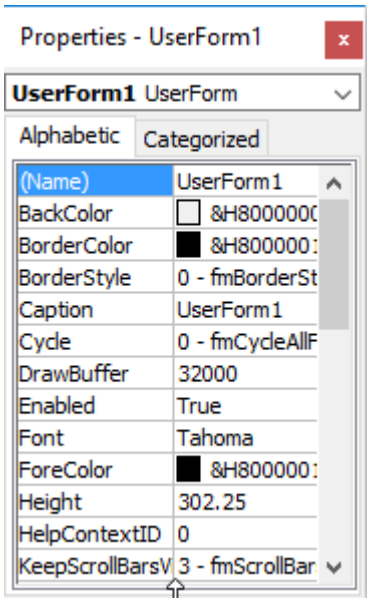


Figure 172

Before discussing Userforms we will discuss the property window main items first.

Table 15

S.No.	Properties	What it does
1	Name	Allow you to specify the form name
2	Caption	Caption is the text which describes and identifies a UserForm or Control.for instance like module 1, module 2 VBA build userform like userform 1, userform 1,
3	TabIndex Property	TabIndex is the position of the control in a UserForm's tab order. It always starts from 0

Now we are going to create a userform as shown in Figure 173.

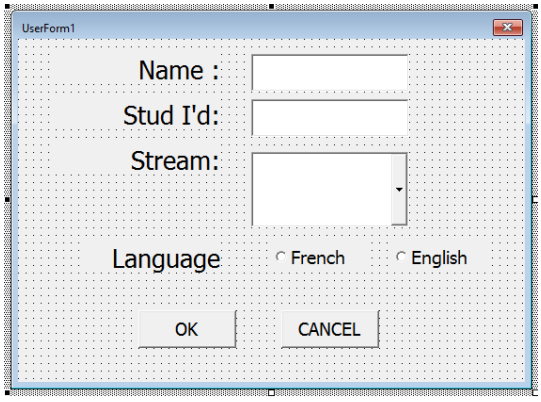


Figure 173

Step 1

We will insert four labels first then rename these as “Name”, “Stud I’d”, “Stream” and “Language”. VBA will insert the Labels with the name of Label 1, label 2 and so on. We have to change the name through “Caption” tab in the properties window. Also we will change the name of labels through “Name” property in Properties window. It will help you to write your code without any confusion. See the Figure 174 i have set the Name as stream and caption as “Stream:”

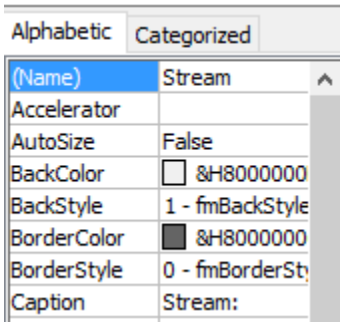


Figure 174

Step 2

Now we will insert two text boxes from the Tool box, to insert the “Name and Unique I,d of the student.

Step 3

We will insert one list box , two check boxes and two combo boxes to insert streams, languages and ok & cancel respectively.

We will change the NAME and Caption one by one from the properties tab.

Let’s begin the code, i want to tell you one very important thing before writing the code, every variable or name of userforms is control by “ME” word for instance see Figure 175

```
Private Sub Cancel_Click()
Unload Me
End Sub
```

```
Private Sub Ok_Click()
lastrow = Cells(Rows.Count, 1).End(xlUp).Row + 1

' enter the name in column a
StudId = lastrow - 1
With Worksheets("Sheet2")
    .Cells(lastrow, 1).Value = Me.Namebox
    .Cells(lastrow, 2).Value = StudId
    .Cells(lastrow, 3).Value = Me.Streambox
    If Me.French = True Then
        .Cells(lastrow, 4).Value = "French"
    Else
        .Cells(lastrow, 4).Value = "English"
    End If
End With

If Cells(lastrow, 1).Value = "" Then
    msgbox "Name cannot be blank"
End If

' ok button
Unload Me

End Sub
```

Figure 175

For cancel button only a simple line of code is required i.e. “Unload Me”.

And the remaining whole code is only for “OK” button.

What is “ME”? – “ME” refers to the parent object from which the code is "sitting" in. For a UserForm, when you are writing the code within the UserForm module, Me will refer to that UserForm. Similarily, if you are writing in a Sheet module, the Me will refer to that specific sheet.

Chapter 18

FILES AND FOLDERS IN EXCEL

You can manage the external files and folders through VBA. But the ways of using VBA are different depending upon the version of excel. If you are using the EXCEL 2003 then you must use OBJECT property and if you are using MS EXCEL 2007 or later version then you may use OBJECT property or MICROSOFT SCRIPTING RUNTIME.

Before starting the topics, i want to tell you that my folder location is "C:\Users\Vikal Jain\Desktop\New folder\Vikal" (I will call it as Maypath) so, I am going to use this path for this whole chapter.

If you are using OBJECT property then you must define you variable as OBJECT

"Dim FSO as Object

Set FSO = CreateObject("Scripting.FileSystemObject")"

If you are using MICROSOFT SCRIPTING RUNTIME. property then you must define you variable as below

"Dim FSO As New Scripting.FileSystemObject"

"Dim FSO as Object

Set FSO = CreateObject("Scripting.FileSystemObject")"

These above two methods are only the magic swords, which helps you to deal with external object through VBA.

I would suggest you to cram these two lines, since i am going to use these two lines in my whole chapter.

Enable Microsoft Scripting Run Time (MSRT) in VBA

For enabling the MSRT you have to go to Tools menu from the menu bar then go to References, then search for Microsoft Run Time.

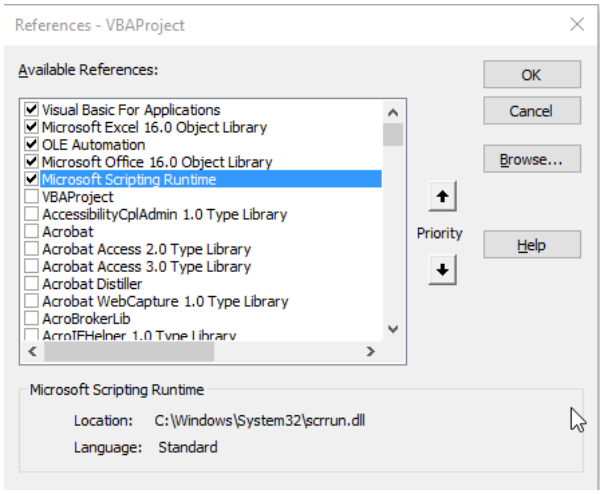


Figure 176

Create Folder through VBA

```
Sub CreatingAFolder()

'Define Variable
    Dim FSO As New Scripting.FileSystemObject
    Dim mypath As String

'Specify the Path
    mypath = "C:\Users\Vikal Jain\Desktop\New folder\Vikal"

'Check whether folder already exists or not
    If Not FSO.FolderExists(mypath) Then
        FSO.CreateFolder (mypath)
        msgbox "New FolderCreated Successfully", vbExclamation, "Done!"
        Else
        msgbox "Specified Folder Already Exists", vbExclamation, _
        "Folder Already Exists!"
    End If
End Sub
```

Figure 177

The code of creating folder through VBA is shown in Figure 177. See the first line of code i.e. "Dim FSO As New Scripting.FileSystemObejct" which is used by me in the Figure 177 will always be the same when you are dealing with the files and folder through VBA. Since this line of code has the power to deal with Folders.

I have put a condition in my code that if the folder exists then give a message box that "Specified Folder Already Exists" otherwise give the message "New folder Created successfully". See the results in Figure 178, Figure 179 and Figure 180.

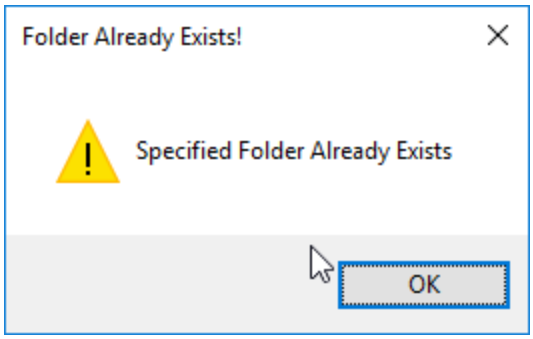


Figure 178

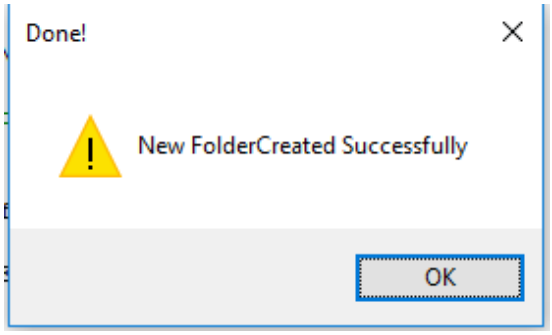


Figure 179

Figure 180

Copying Folders from One Location to Another

When you are using the VBA the first line of code will always be the same, since we have to always define the Scripting object first.

The code for this is shown in Figure 181 and the results of code is shown in Figure 182, Figure 183 & Figure 184.

```
'copy folder

Sub fileandfolders()

'defineVariable

    Dim FSO As New Scripting.FileSystemObject
    Dim mypath As String, newFolder As String

'Specify the Path

    mypath = "C:\Users\Vikal Jain\Desktop\New folder\Vikal"
    newFolder = "C:\Users\Vikal Jain\Desktop\New folder\Boss"

'Check folder exists first if yes then copy the folder otherwise
'Don't copy

    If Not FSO.FolderExists(newFolder) Then
        FSO.CopyFolder mypath, newFolder
        msgbox "Folder Copied Successfully to The Destination", _
        vbExclamation, "Done!"
    Else
        msgbox "Folder Already Exists in the Destination", vbExclamation, _
        "Folder Already Exists!"
    End If
    End Sub
```

Figure 181

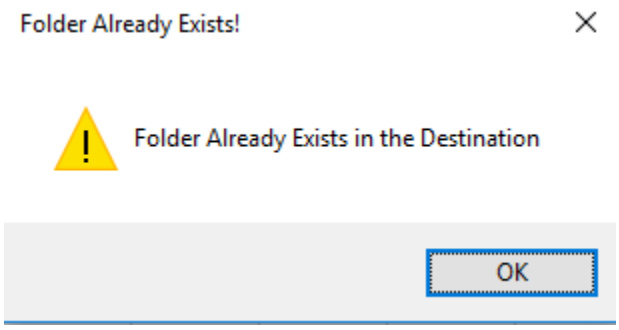


Figure 182

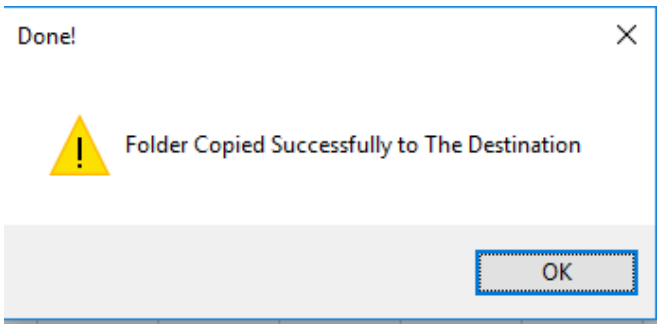


Figure 183

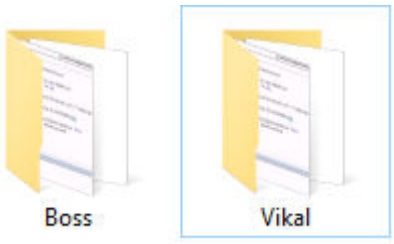


Figure 184

Move Folder

```
'move folder

Sub movefolders()

'defineVariable

    Dim FSO As New Scripting.FileSystemObject
    Dim mypath As String, newFolder As String

'Specify the Path

    mypath = "C:\Users\Vikal Jain\Desktop\New folder\Vikal"
    newFolder = "C:\Users\Vikal Jain\Desktop\New folder\Boss"

'Check folder exists first if yes then copy the folder otherwise
'Don't copy

    If Not FSO.FolderExists(newFolder) Then
        FSO.MoveFolder mypath, newFolder
        msgbox "Folder Moved Successfully to The Destination", _
        vbExclamation, "Done!"
    Else
        msgbox "Folder Already Exists in the Destination", vbExclamation, _
        "Folder Already Exists!"
    End If
    End Sub
```

Figure 185

Figure 185 is showing the code for moving the folder.

Delete Folder

```
Sub Deletefolders()

'defineVariable

    Dim FSO As New Scripting.FileSystemObject
    Dim mypath As String

'Specify the Path

    mypath = "C:\Users\Vikal Jain\Desktop\New folder\Vikal"

'Check folder exists first if yes then delete the folder

    If FSO.FolderExists(mypath) Then
        FSO.DeleteFolder mypath
        msgbox "Folder Deleted Successfully ", _
        vbExclamation, "Done!"
    Else
        msgbox "Folder is not there", vbExclamation, _
        "Deletefolder!"
    End If
    End Sub
```

Figure 186

Figure 186 is showing the code for deleting the folder.

Chapter 19

EVENT HANDLER

Event handler allows you to handle the event like

- Opening the workbook
- Select the range
- Select the cells and so on.

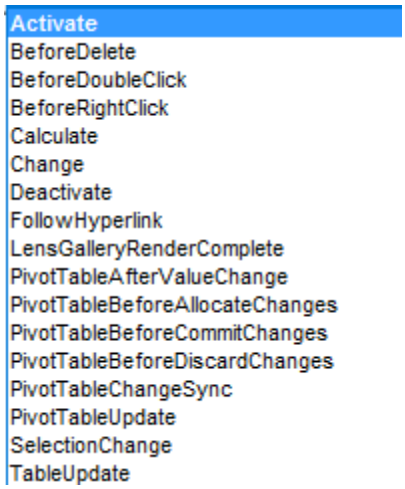


Figure 187

Figure 195 is showing the list of events for a worksheet and Figure 188 is showing the list for events for a workbook.

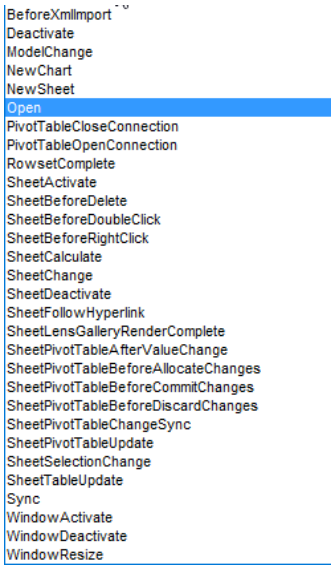


Figure 188

Let me show the example what worksheets event is exactly?

Go to sheet 3 from the project window

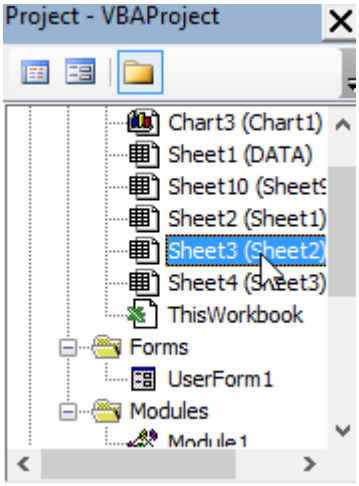


Figure 189

Click on the drop arrow where it says General, and choose Worksheet.

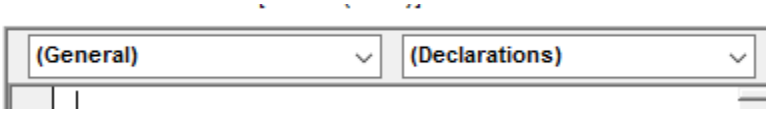


Figure 190

You can now choose the particular event you want to handle by clicking on the right-hand drop arrow:

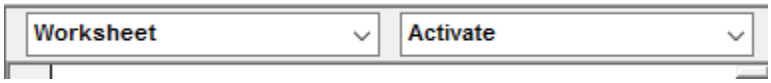


Figure 191

You can now type in any code that you want to run when a user selects your worksheets! When you select your sheet, then all things being well your code will run.

```
Private Sub Worksheet_Activate()
Worksheets("sheet3").Range("a1:a10").Interior.Color = RGB(25, 55, 5)
End Sub
```

Figure 192

Now when i select the sheet 3 it will automatically color the Range A1 to A10

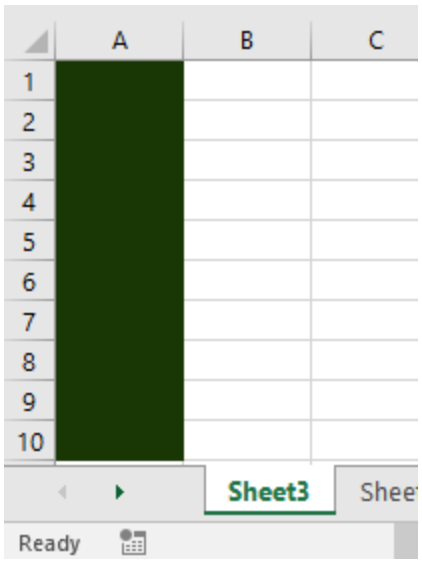


Figure 193

So this is the basics event handler, which allows you to perform the events.

Chapter 20

OTHER TOPIC

Message Box

Displays a message in a dialog box, waits for the user to click a button, and returns an Integer indicating which button the user clicked.

Syntax

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

The MsgBox function syntax has these named arguments:

Table 16

Part	Description
Prompt	Required. String expression displayed as the message in the dialog box. The maximum length of prompt is approximately 1024 characters, depending on the width of the characters used. If prompt consists of more than one line, you can separate the lines using a carriage return character (Chr(13)), a linefeed character (Chr(10)), or carriage return linefeed character combination (Chr(13) &Chr(10)) between each line.
Buttons	Optional. Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If omitted, the default value for buttons is 0.
Title	Optional. String expression displayed in the title bar of the dialog box. If you omit title, the application name is placed in the title bar.
Helpfile	Optional. String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If helpfile is provided, context must also be provided.
Context	Optional. Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If context is provided, helpfile must also be provided.

Source: https://msdn.microsoft.com/en-us/library/aa445082(v=vs.60).aspx

If we analyze the Table 16 then, we will realize that only “Prompt” argument is compulsory, others are optional. Prompt is used to display your message which do you want to show in the message box. You can also separate the lines using line separators there are lots of separators in VBA such as:-

Character No.	Character Name	What is does
Chr(10)+ Chr(13)	vbCrLf – Carriage Return CR	

	+ Line Feed LF	

Code without Line separator

```
Sub mymsgbox()
msgbox " Hello! This is message box showing by VBA"
End Sub
```

Figure 194

Here is the result

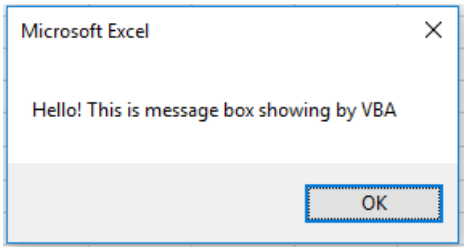


Figure 195

Code with Line separator

```
Sub mymsgbox()
msgbox " Hello!" & vbCrLf & "This is message box showing by VBA"
End Sub
```

Figure 196

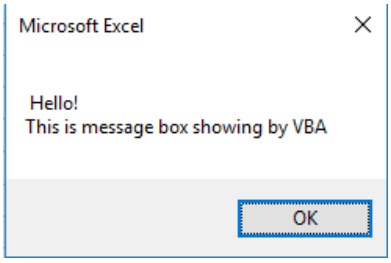


Figure 197

Alternative Methods

Method 1

```
Sub mymsgbox()
msgbox " Hello!" & Chr(10) & "This is message box showing by VBA"
End Sub
```

Figure 198

Method 2

```
Sub mymsgbox()
msgbox " Hello!" & Chr(13) & "This is message box showing by VBA"
End Sub
```

Figure 199

Author Note: List of all characters that are used in VBA are shown in Appendix 2.

Settings

The buttons argument settings are:

Constant	Value	Description
vbOKOnly	0	Display OK button only.
vbOKCancel	1	Display OK and Cancel buttons.
vbAbortRetryIgnore	2	Display Abort, Retry, and Ignore buttons.
vbYesNoCancel	3	Display Yes, No, and Cancel buttons.
vbYesNo	4	Display Yes and No buttons.
vbRetryCancel	5	Display Retry and Cancel buttons.
vbCritical	16	Display Critical Message icon.
vbQuestion	32	Display Warning Query icon.
vbExclamation	48	Display Warning Message icon.
vbInformation	64	Display Information Message icon.
vbDefaultButton1	0	First button is default.
vbDefaultButton2	256	Second button is default.
vbDefaultButton3	512	Third button is default.
vbDefaultButton4	768	Fourth button is default.
vbApplicationModal	0	Application modal; the user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal; all applications are suspended until the user responds to the message box.
vbMsgBoxHelpButton	16384	Adds Help button to the message box
VbMsgBoxSetForeground	65536	Specifies the message box window as the foreground window
vbMsgBoxRight	524288	Text is right aligned
vbMsgBoxRtlReading	1048576	Specifies text should appear as right-to-left reading on Hebrew and Arabic systems

Source: https://msdn.microsoft.com/en-us/library/aa445082(v=vs.60).aspx

These buttons represents type of Messages for instances whether they are critical, info button, and warning button and so on

Example 1

```
Sub mymsgbox()
msgbox " Hello!This is message box showing by VBA", Buttons:=vbInformation
End Sub
```

Figure 200

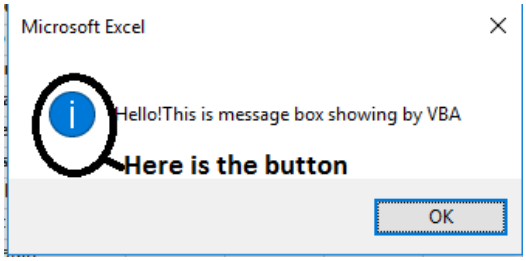


Figure 201

In Figure 201 VBA is showing VBA information button

Example 2

```
Sub mymsgbox()
msgbox " Hello!This is message box showing by VBA", Buttons:=vbCritical
End Sub
```

Figure 202

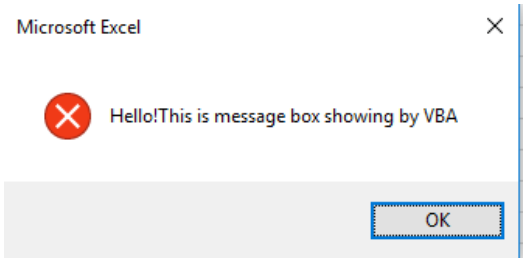


Figure 203

This time VBA is showing Critical button in Figure 203

Next argument is the “Title” , you can give the title to Message box as shown in the Figure 205

```
Sub mymsgbox()
msgbox " Hello!This is message box showing by VBA", _
Buttons:=vbCritical, Title:="Boss"
End Sub
```

Figure 204

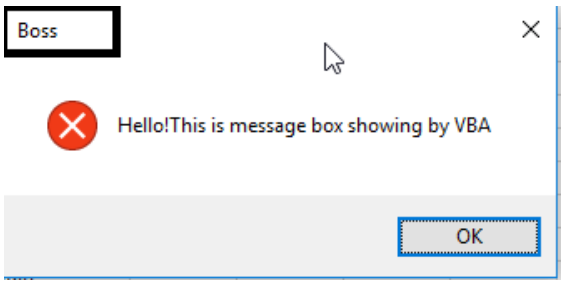


Figure 205

Then the last two arguments are helpfile and context which are rarely used in Message box.

Input Box

Input box allows you to enter the data in the pop up window and produce the results on the basis of Values entered.

Syntax

InputBox(Prompt, Title, Default, Left, Top, HelpFile, HelpContextID, Type)

Parameters

Name	Required / Optional	Data Type	Description
Prompt	Required	String	The message to be displayed in the dialog box. This can be a string, a number, a date, or a Boolean value (Microsoft Excel automatically coerces the value to a String before it is displayed).
Title	Optional	Variant	The title for the input box. If this argument is omitted, the default title is "Input."
Default	Optional	Variant	Specifies a value that will appear in the text box when the dialog box is initially displayed. If this argument is omitted, the text box is left empty. This value can be a Range object.
Left	Optional	Variant	Specifies an x position for the dialog box in relation to the upper-left corner of the screen, in points.
Top	Optional	Variant	Specifies a y position for the dialog box in relation to the upper-left corner of the screen, in points.
HelpFile	Optional	Variant	The name of the Help file for this input box. If the HelpFile and HelpContextID arguments are present, a Help button will appear in the dialog box.
HelpContextID	Optional	Variant	The context ID number of the Help topic in HelpFile.
Type	Optional	Variant	Specifies the return data type. If this

			argument is omitted, the dialog box returns text.

Source: https://msdn.microsoft.com/en-us/library/office/ff839468.aspx

First argument: - Prompt

This allow you to enter the data in the input box for instance

```
Sub whatisyourname()
InputBox "Please enter your name"
End Sub
```

Figure 206

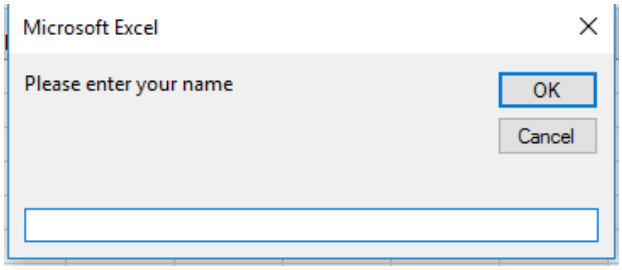


Figure 207

Author Note: This pop up window allows you to enter the data in the Input Box. However, it does not allow you to produce any results only on the basis of "Prompt" arguments.

Second argument: - Title

Title allows you to give the title to your input box for instance

```
Sub whatisyourname()
InputBox "Please enter your name", Title:="NameBox"
End Sub
```

Figure 208

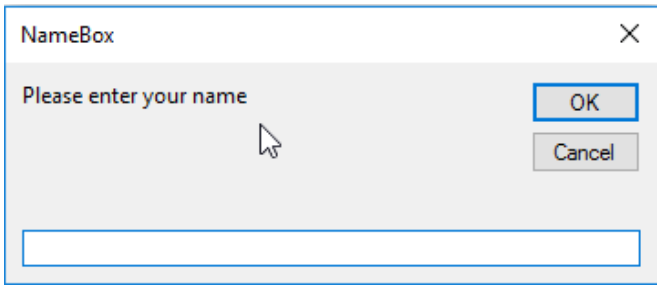


Figure 209

Third arguments: - Default

This argument allow you to show a message in the data entering box, for instance

```
Sub whatisyourname()
InputBox "Please enter your name", _
Title:="NameBox", Default:="Enter Name Here....."
End Sub
```

Figure 210

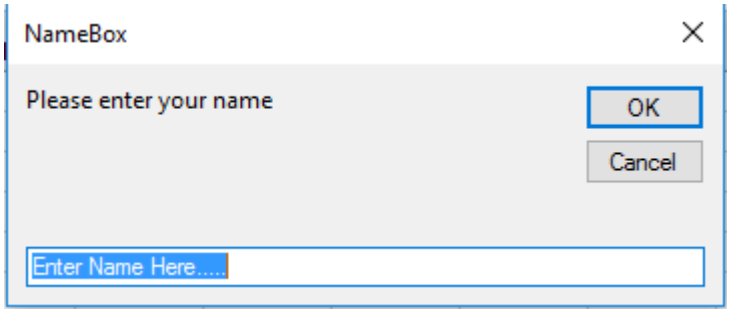


Figure 211

You can link any cell to the input box as shown in the following example

```
Sub whatisyourname()
Range("A1").Value = InputBox("Please enter your name", _
Title:="NameBox", Default:="Enter Name Here.....")
End Sub
```

Figure 212

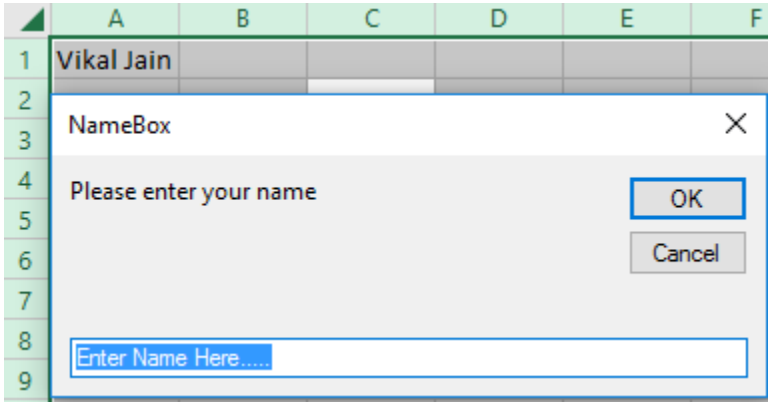


Figure 213

See the Figure 213 when I entered my name in dialogue box. It shows my name in the cell A1.

Author Note: The point to be noted here that if you just want to provide any information on the basis of Message box and Input box then you don't need to provide your arguments in the parenthesis. When you are linking the cells or the range with these boxes then you must enter your data in the Parenthesis.

Till yet OK button and Cancel button are showing the same results. Let's show the different results with the help of these two buttons.

Here is the code

```
Sub whatisyourname()
Myname = InputBox("Please enter your name", _
Title:="NameBox", Default:="Enter Name Here.....")

If Myname = "" Then
msgbox "You didnot enter any data"
Else
msgbox "My name is " & Myname
End If
End Sub
```

Figure 214

When you will run the code as shown in Figure 214 then this pop up window will open.

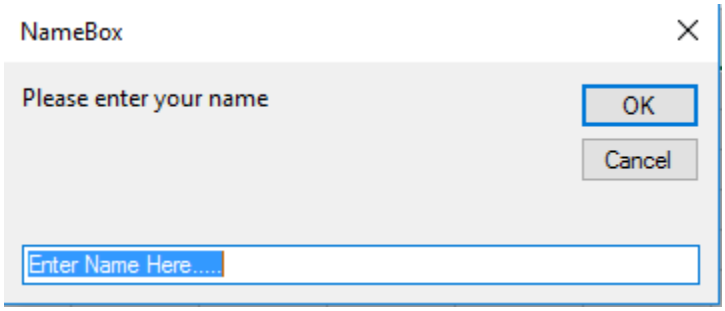


Figure 215

When you will enter the name in the input box and press enter. The result will be as follows.

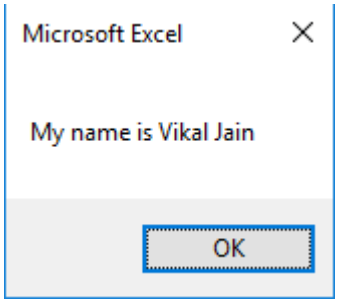


Figure 216

If you will press cancel button then it will show this pop up window.

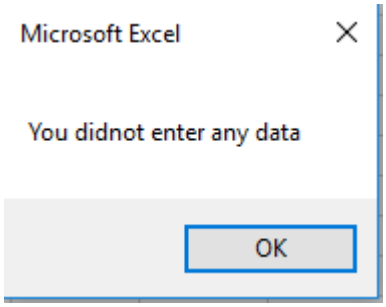


Figure 217

Display Alerts

When you are dealing with the VBA then sometimes EXCEL messages like “Do you want to delete the sheet1” or “Do you want to move the sheet 1” and so on may results into confusion for users of that sheet.

Since you are writing that code and you are asking from users do you want to delete that sheet which users don’t know about. So users will get confused. In this situation we generally disable the EXCEL messages from this code

Application.DisplayAlerts = False

Figure 218 is showing the total numbers of sheets in my workbook, I am going to delete the Sheet2 by using VBA.

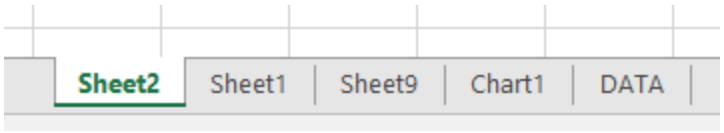


Figure 218

Here is simple line of code

```
Sub displayalerts()
'
' Macro2 Macro

Sheets("Sheet2").Delete

End Sub
```

Figure 219

When you will run that code this will open a pop up window that do you want to delete the sheet.

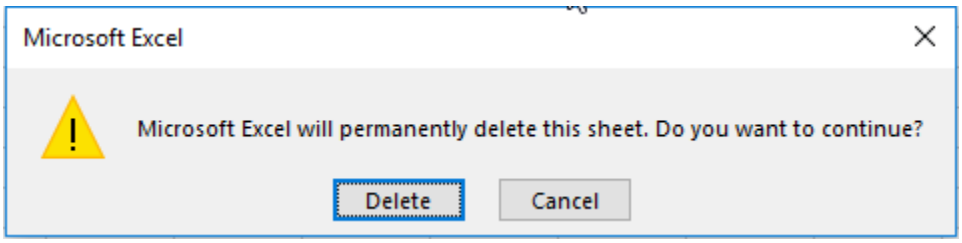


Figure 220

When we will turn off display alerts the code will simply delete that sheet as shown in the Figure 222 and don’t forget to turn on back the display alerts again since it is not going to show you the alerts after switching off the Display alerts.

```
Sub displayalerts()
'
' Macro2 Macro
Application.displayalerts = False
Sheets("Sheet2").Delete
Application.displayalerts = True
End Sub
```

Figure 221

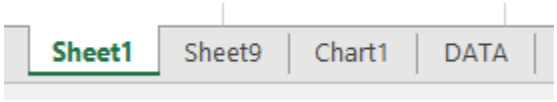


Figure 222

Speed up your Macro

When you use VBA to perform the code in EXCEL, It will take lots of time to perform. Since, it always shows you each and every steps of code during running of macro. It is lengthy process by excel to show you the screen updating for each and every line of code.

You can simply turn off the screen updating, through this line of code and it will not show you screen updating now and boost the speed of your macro.

Application.Screenupdating = False (for turn off the screen updates)

Please ensure, when your code is finished then turn it on again.

Chapter 21

APPENDIXES

OPEN method Syntax

expression .Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword, IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify, Converter, AddToMru, Local, CorruptLoad)

Parameter

Table 17

Name	Required/Optional	Data Type	Description
FileName	Optional	Variant	String. The file name of the workbook to be opened.
UpdateLinks	Optional	Variant	Specifies the way external references (links) in the file, such as the reference to a range in the Budget.xls workbook in the following formula =SUM([Budget.xls]Annual!C10:C25), are updated. If this argument is omitted, the user is prompted to specify how links will be updated. For more information about the values used by this parameter, see the Remarks section. If Microsoft Excel is opening a file in the WKS, WK1, or WK3 format and the UpdateLinks argument is 0, no charts are created; otherwise Microsoft Excel generates charts from the graphs attached to the file.
ReadOnly	Optional	Variant	True to open the workbook in read-only mode.
Format	Optional	Variant	If Microsoft Excel opens a text file, this argument specifies the delimiter character. If this argument is omitted, the current delimiter is used. For more

			information about the values used by this parameter, see the Remarks section.
Password	Optional	Variant	A string that contains the password required to open a protected workbook. If this argument is omitted and the workbook requires a password, the user is prompted for the password.
WriteResPassword	Optional	Variant	A string that contains the password required to write to a write-reserved workbook. If this argument is omitted and the workbook requires a password, the user will be prompted for the password.
IgnoreReadOnlyRecommended	Optional	Variant	True to have Microsoft Excel not display the read-only recommended message (if the workbook was saved with the Read-Only Recommended option).
Origin	Optional	Variant	If the file is a text file, this argument indicates where it originated, so that code pages and Carriage Return/Line Feed (CR/LF) can be mapped correctly. Can be one of the following XlPlatform constants: xlMacintosh, xlWindows, or xlMSDOS. If this argument is omitted, the current operating system is used.
Delimiter	Optional	Variant	If the file is a text file and the Format argument is 6, this argument is a string that specifies the character to be used as the delimiter. For example, use Chr(9) for tabs, use "," for commas, use ";" for semicolons, or use a custom character. Only the first character of the string is used.
Editable	Optional	Variant	If the file is a Microsoft Excel 4.0 add-in, this argument is True to open the add-in

			so that it is a visible window. If this argument is False or omitted, the add-in is opened as hidden, and it cannot be unhidden. This option does not apply to add-ins created in Microsoft Excel 5.0 or later. If the file is an Excel template, True to open the specified template for editing. False to open a new workbook based on the specified template. The default value is False.
Notify	Optional	Variant	If the file cannot be opened in read/write mode, this argument is True to add the file to the file notification list. Microsoft Excel will open the file as read-only, poll the file notification list, and then notify the user when the file becomes available. If this argument is False or omitted, no notification is requested, and any attempts to open an unavailable file will fail.
Converter	Optional	Variant	The index of the first file converter to try when opening the file. The specified file converter is tried first; if this converter does not recognize the file, all other converters are tried. The converter index consists of the row numbers of the converters returned by the FileConverters property.
AddToMru	Optional	Variant	True to add this workbook to the list of recently used files. The default value is False.
Local	Optional	Variant	True saves files against the language of Microsoft Excel (including control panel settings). False (default) saves files against the language of Visual Basic for Applications (VBA) (which is typically United States English unless the VBA project where Workbooks.Open is run from is an old internationalized XL5/95

			VBA project).
CorruptLoad	Optional	XlCorruptLoad	Can be one of the following constants: xlNormalLoad, xlRepairFile a ndxlExtractData. The default behavior if no value is specified is xlNormalLoad and does not attempt recovery when initiated through the OM.

Source: https://msdn.microsoft.com/en-us/library/office/ff194819.aspx

List of Characters in VBA

Table 18

Character	Code Character
8	backspace
9	Tab
10	linefeed
13	carriage return
32	[space]
33	!
34	""
35	#
36	$
37	%
38	&
39	'
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:

59	;
60	<
61	=
62	>
63	?
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	_
96	'
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k

108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	|
125	}
126	~