QUICK GUIDE FOR VBA EXCEL

LEARN VBA FASTER THAN ANYONE

If you want to learn VBA, This book is the place where you wanted to start. This book is the first part what I will be writing. Be engaged with my blog to get an update of what is coming next.

https://www.itchat.in/blog/

FOR MORE INFORMATION CONTACT: EXCEL-CONSULTING@ITCHAT.IN

RV

Contents

CHAPTER 1 - VBA BASICS

What is VBA?

VBA stands for Visual Basic for Application, is a scripting language which can manipulate MS office Objects. Prior to VBA there was a language called XLM but it was very cryptic and hard to use, Visual Basic is very easy and widely used language, that was probably the reason Microsoft implemented this into MS office products.

What is MS Office Object?

As you know, VBA can manipulate MS office object, Here MS office objects refer to MS office applications such as Excel, Access, Outlook, PowerPoint. Every MS office Application have its own object hierarchy. You will understand this next.

What is object hierarchy/Model?

Microsoft Office Applications have their own object model, this Book is about following this object model and manipulate these objects by using VBA. So, we will be following Microsoft Excel and its object model throughout this book. This is the basic object model of Excel.

- **Excel Application**
  - **Workbooks**
    - **Worksheets**
      - **Range**
      - **Shapes**
    - **ChartSheets**

This model is telling that in one instance of excel you can open multiple workbooks. In one workbook you can have so many worksheets, Charts sheets, in one worksheets you have range, Shapes etc. Excel model is very complex, thousands of objects are bounds to each other.

What is object itself?

Object is an entity that perform task, every object has its own properties and methods written in it class. For example, Range Object. Range object have some properties and methods. If you want to assign a value to a range, you will write `Range("A1").value  So,` Value is a property of Range object. Properties and something which can get inputs or return a value. Some properties are Read-Only, you cannot assign a value to a Read-Only property. Like `Application.Version`. you can only read the Excel Version, but you cannot change it.

Unlike properties, Methods doesn't return or get input as value. They complete a task. For example, `Range("A1:A100").Sort`. So, Sort is a method here, it will not return any value but it will sort the Range.

What is class?

Class is the definition of an object, all properties and methods are written in class, once class is written, N number of objects can be created by referencing the class. Imaging class is blueprint of a building, once blueprint is finalized, countless building can be created by using the parameters declared in blueprint.

If you call yourself Human, you are an object of human class, of course, because you fit in the definition written in human class. E.g. you will have two eyes, two hands, two legs and an intelligent brain.

How VBA Works?

Unlike other language there is no complier to create an .exe (executable fie). VBA interpreter decode each line one by one and give instruction to machine.

VBA stays with Excel file itself, it means, there is no extra file that you need to create to run your VBA program, you will write all your VBA code in Excel file itself. Every Excel file comes with VBObject Model that save all your code.

Files type which support VBA

Remember to save excel file in Excel Macro enabled file else your file won't be able to hold the VBA program. So best practice is to save your first before starting any program. You can save your file in any of these extensions.

XLSM, XLS, XLSB, XLTM, XLA, XLAM

Understanding VBA components

I am not going to tell you to record macro and then edit it to make it work for you dynamically, instead, I will teach how to write VBA program from scratch. Before writing your code you need to understand few basic of any programming language. These are basically programming component that you should aware of.

What is Variables?

In any programming language, Variable is the first thing that you need to understand, Variable is nothing but memory holder, A variable hold memory so that it can save any value for further use in program.

Example –

```
Dim bytAge as byte
bytAge = Inputbox("Please enter your age")
```

Explanation –

In the program above, bytAge is a variable which will save entered age in the memory, so whenever you need to use age in your program, you will use bytAge variable, you will not need to have user to enter age again and again.

What are the rules of declaring variable?

1. No Special characters are allowed except underscore (_)
2. First character must be an alphabet.
3. Max length of a variable is 255
4. Reserved keyword cannot be used as a variable name.
5. Keep in mind the ambiguity.
6. Must have a data type
7. Use a meaningful name for your variable
8. Use Hungarian notation for your variable
   a. Dim **lngCounter** as long
   b. Dim **strFileData** as string

What is Data type?

Data type is another thing which is associated with variable, Data type is basically a validation on data which is going to store in a variable.

Example –

```
Dim bytAge as byte
bytAge = Inputbox("Please enter your age")
```

Explanation –

bytAge variable is a Byte type variable. Byte data type allow only numbers to get store in the variable, if user enter a string like “twenty-six”, your program will throw an error.

There are few types of data type which are available in VBA.

Data Type or **Subtype**	**Required Memory**	**Default Value**	**VBA Constant**	**Range**
Integer	2 bytes	0	`vbInteger`	–32,768 to 32,767
Long Integer	4 bytes	0	`vbLong`	–2,147,483,648 to 2,147,486,647
Single	4 bytes	0	`vbSingle`	–3402823E38 to –1.401298E–45 or 1.401298E–45 to 3.402823E38
Double	8 bytes	0	`vbDouble`	–1.79769313486232E308 to –4.94065645841247E–324 or 1.79769313486232E308 to 4.94065645841247E–324
Currency	8 bytes	0	`vbCurrency`	–922,337,203,477.5808 to 922,337,203,685,477.5807
Date	8 bytes	00:00:00	`vbDate`	January 1, 100 to December 31, 9999
Fixed String	String's length	Number of spaces to accommodate string	`vbString`	1 to 65,400 characters
Variable String	10 bytes plus the number of characters	Zero-length string ("")	`vbString`	0 to 2 billion characters
Object	4 bytes	Nothing (`vbNothing`)	`vbObject`	Any Access object, ActiveX component or Class object
Boolean	2 bytes	False	`vbBoolean`	–1 or 0
Variant	16 bytes	Empty (`vbEmpty`)	`vbVariant`	Same as Double
Decimal	14 bytes	0	`vbDecimal`	–79,228,162,514,264,337,593,543,950,335 to 79,228,162,514,264,337,593,543,950,335 or –7.2998162514264337593543950335 to 7.9228162514264337593543950335

Byte	1 byte	0	vbByte	0 to 255

You don't need to remember all these Ranges. But, to choose data type wisely you need memorize a couple ranges like byte, Integer and long. when you remember these you should be able to determine to choose correct data type for variable like in above program, we chose **byte** for bytAge because storage capacity of byte is 255, A normal person on earth cannot live that long, so this is the best data type of Age.

What are Constants?

A variable is something which values can varies at any time in the program, you can change the value of a variable anywhere in your program. But you cannot change value of constants. You need to assign a value to constant while writing your code.

Example –

```
Public Const strApplicationName as string = "Excel VBA Application"
```

Explanation-

`strApplicationName` is a constant here, Constants are basically read only memory where you can assign value in design time and use that value in runtime. We will use these ahead in this book.

What is Scope?

I am sure you noticed the keywork "**Public**" in the last example. There are three keyword **Dim**, **Public** and **Private** to define the availability of any variable, constant, procedure or function in VBA.

1. If anything is declared with **Public** keyword, it will be available in entire program.
2. If program is declared with **Private** keyword, it will be available in module only.
3. If variables are declared with **Dim** keyword, it will be available in module only. Or if variables are declared in any procedure with **Dim** keyword, those will be available only in that procedure.

Example –

```
Public Const strApplicationName as string = "My Application Name"
Public Const dblPI as Double = 3.14
Dim bytAge as Byte

Private Function GetData() as long
     Dim varData
End Sub
```

Explanation –

You can access `strApplicationName` in entire project. It means this constant will be available in any module, any user form, class module. Same goes with publicly declared constant `dblPI`.

bytAge is declared with **Dim** keyword, it will be accessible only in the module it is declared in.

varData will only be accessible in the function GetData() only, you can not use this variable outside this function. Look at the image #1, it will give you more clarity on this.

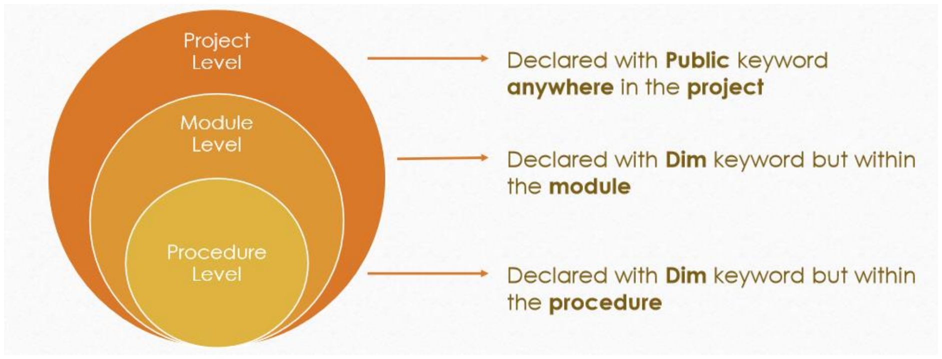

Project Level
Module Level
Procedure Level
Declared with Public keyword anywhere in the project
Declared with Dim keyword but within the module
Declared with Dim keyword but within the procedure

Image #1

What is Ambiguity?

Ambiguity is an error occurred when you declared variable/Constants/Functions/Procedure in same scope.

Example –

```
Dim strProjectName as string
Dim strProjectName as string
```

Of course, both variables are declared with same keyword and at same place, so VBA compiler will get confused which one to use when any program will call them. So, VBA compiler won't let the process start and will throw an error.

VBA Procedures?

You can write three types of procedures in VBA.

1. Sub Procedure
2. Function Procedure
3. Events Procedure

What is Sub Procedure?

Sub procedure is group of VBA statements which can run the process. A Sub procedure is used to complete a task. It starts with **Sub** keyword and ends with **End Sub**

Example –

```
Sub MergeCells()
     Range(“A1:A10”).Merge
End Sub
```

This Sub procedure will merge A1:A10 on active sheet

What is Function Procedure?

Function procedure are set of instructions which do some calculation and return the calculated value. It starts with **Function** keyword and ends with **End Function** keyword. Function procedure also have one of data type. Calculated value needs to be assign the function name to get the result from a function procedure.

Example –

```
Function CalculateAge(dtDob as date) as byte
     CalculatedAge = Year(Date)- Year(dtDob)
End Sub
```

Explanation –

Here we have created a parametrized function procedure. We are passing date of birth as a parameter. Next, we assign the difference of years to the function name.

What is Event Procedure?

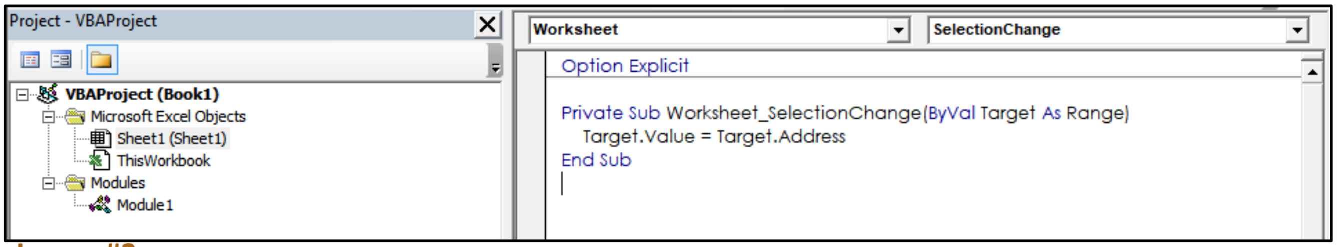
Event procedures are built-in program in VBA for some objects, they automatically get invoked when something got triggered. These procedures must be written in objects code window.

Example –

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Target.Value = Target.Address
End Sub
```

How to insert an event procedure?

First of all, open up the object windows, for above example is a worksheet even, so open up your VBE (Alt + F11) and double click on sheet1, here you will see two dropdowns, Select worksheet in the first one and in second one select the SelectionChange. Once you select event it will automatically insert an empty event. You will write your code here. e.g. `Target.Value = Target.Address`

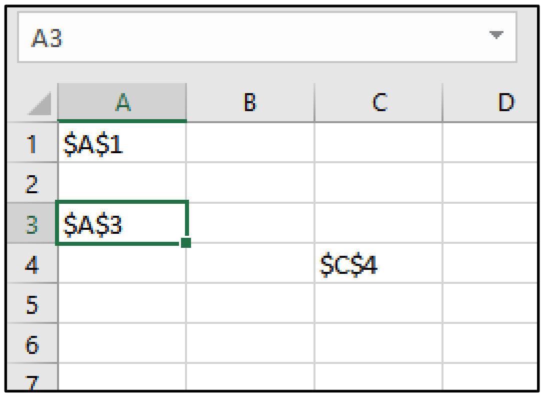


```
Option Explicit

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Target.Value = Target.Address
End Sub
```

Image #2

Explanation –

Now, when you select any cell on sheet one, it will insert cell address in the selected cell.



A3

	A	B	C	D
1	A1			
2				
3	A3			
4			C4	
5				
6				
7				

Image #3

How to comment VBA code?

Comments in your code are very important, you should make it your habit to write comments as much as possible. You should write What your program is doing, you should try to explain the process and calculation in your comments, so that if any other programmer sees your code, he/she will be able to understand it quicker. Even if you see your code again after a year, without comments you won't be able to understand what you did and why you did. So, Comments are very important.

1. Use single quote sign (') before line
2. User Rem before line
3. If you want to comment/uncomment an entire section in code. Use the tool bar shown in screenshot

```vba
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    ' This event will insert address of selected cell
    Target.Value = Target.Address
End Sub
```

Chapter 2 – Controlling program

In previous chapter, we tried to understand how to write procedures in VBA. Now, we will understand how to write VBA statements and control them so that they work dynamically. We will understand how to let program to run statement on specific condition. We will learn how to use loops to reduce the repetitive statements and make program even more logical. We will also learn how to handle error in programs

IF Statement

If statement is probably the most useful statement in VBA you will write, It let you decide to run lines of code based on conditions, there are couple of ways we can use IF statement.

Syntax 1

```
IF <Condition> Then Statement
```

If you need to write only **one line of code** after if statement, you should use the above syntax.

```
Function GetDayName(bytDayNumber As Byte) As String

    If bytDayNumber >= 1 And bytDayNumber <= 7 Then
        If bytDayNumber = 1 Then GetDayName = "Sunday"
        If bytDayNumber = 2 Then GetDayName = "Monday"
        If bytDayNumber = 3 Then GetDayName = "Tuesday"
        If bytDayNumber = 4 Then GetDayName = "Wednesday"
        If bytDayNumber = 5 Then GetDayName = "Thurday"
        If bytDayNumber = 6 Then GetDayName = "Friday"
        If bytDayNumber = 7 Then GetDayName = "Saturday"
    End If

End Function
```

Syntax 2

```
IF <Condition> Then
```

```
        ‘–Statements here
    Endif
```

If there only **one condition** you should use the syntax above. See example –

```
    Private Sub Worksheet_SelectionChange(ByVal Target As Range)
        ' This event will insert address of selected cell

        If Target.address = “$A$1” then
          Target.Value = Target.Address
        Endif

    End Sub
```

SYNTAX 3

```
    IF <Condition> Then
          ‘–Statements here
    Elseif <Condition> then
          ‘–Statements here
    Else
          ‘–Statements here
    Endif
```

If there only **multiple condition** you should use the syntax above. See example –

```
Function CheckifSheetExists(strSheetName As String, Optional wbkFile As Workbook)
As Boolean
        Dim wbkCheck As Workbook
    Dim wksSheet As Worksheet

    If Not wbkFile Is Nothing Then '-- if passed
        Set wbkCheck = wbkFile
    Else
        Set wbkCheck = ThisWorkbook '-- if missing
    End If
    For Each wksSheet In wbkCheck.Worksheets
        If wksSheet.Name = strSheetName Then
            CheckifSheetExists = True
            Exit Function
        End If
    Next wksSheet

End Function
```

How to build a condition?

A condition must return either TRUE or FALSE and based on these TRUE and False, IF statement will decide which set of statement it should execute. <condition> most of the time is comparison between two variables. Sometimes we need to use VBA informative functions Like `IsNumeric`, `IsDate` etc. here is the list of comparison operator and informative function you can use in VBA.

Comparison Operator

```
= Equal to
<> Does not equal to
>= Greater than equal to
<= Lesser than equal to
> Greater Than
< Lesser Than
```

Information Function

```
isNumeric
isArray
isDate
isEmpty
isError
isMissing
isNull
isObject
```

You will see all these in use in further program in this book.

Select Case

Case also works like IF statement but in different way. We use Case when there are so many conditions. See example

```
Function GetDayNameByCase(bytDayNumber As Byte) As String

        Select Case bytDayNumber
            Case 1
                GetDayNameByCase = "Sunday"
            Case 2
                GetDayNameByCase = "Monday"
            Case 3
                GetDayNameByCase = "Tuesday"
            Case 4
                GetDayNameByCase = "Wednesday"
            Case 5
                GetDayNameByCase = "Thursday"
            Case 6
                GetDayNameByCase = "Friday"
            Case 7
                GetDayNameByCase = "Saturday"
            Case Else
                GetDayNameByCase = "Invalid input"
        End Select

End Function
```

Loops

Loops are very important to understand to make your program optimized and logical. In any program in any language you will see 90% of codes are written in Loops and Ifs. There are few types of loop in VBA.

For Loop
For Each Loop
Do Loop

For Loop

Loop are basically used to reduce repetitive syntax. For example, if you want to print 1 to 5 counting in Range A1:A5, without loop you will write code like this.

```
Sub PrintNumber()

    Range("A1").Value = 1
    Range("A2").Value = 2
    Range("A3").Value = 3
    Range("A4").Value = 4
    Range("A5").Value = 5

End Sub
```

Now you see the beauty of For Loop.

```
Sub PrintNumber()

  Dim lngR As Long
  For lngR = 1 To 5
    Range("A" & lngR).Value = lngR
  Next lngR
```

```
     End Sub
```

Explanation –

For loop need a numeric variable to iterate the statement, you can define how many times you want to run the loop. Here `lngR` is our loop variable and we defined that it needs to run 1 to 5. Now, the statement `Range(“A” & lngR).value = lngR` will run five time and each time value of variable `lngR` will increase by 1. `lngR` concatenated with “A” will create a string range reference and Range function will convert that to an actual object. And this way it will assign value to the range in active worksheet.

Steps in For loop

By Default, For Loop increase its variable value by 1, but we can change this to increase or decrease value for our requirement. Let’s say you want to highlight every alternate row in the data. This is how you will use for loop.

```
Sub HighlightAlternateRow()

    Dim lngRow As Long
    Dim lngLastRow As Long
     ‘—Get the last row of your data
    lngLastRow = Sheet1.Range("A" & Rows.Count).End(xlUp).Row

     ‘—Run loop for every alternate row
    For lngR = 1 To lngLastRow Step 2
        Sheet1.Range("A" & lngRow).EntireRow.Interior.Color = vbYellow
    Next lngR

End Sub
```

In above program, Notice the **Step** keyword, you can use any positive/Negative number to run your step except 0. Because if you use 0, your loop will go in infinity and will never come out. Because execution comes out of loop when variable value become greater than the upper value of Loop. E.g. `lngLastRow`

For Each Loop

For Each loop iterate every object in a collection, Like For Loop need a numerical variable to iterate statements, For Each loop need an object type variable to iterate object in the same type of collections. Like if you want to access every cell in a collection of cells, every sheet in all sheets in a workbook, every shape in a sheet, every pivot table in a sheet and so on.

See Couple examples here –

Here is a program which will fill green color in cells if cell value is > 25.

```
Sub HightlightCells()

    Dim rngCell As Range
    For Each rngCell In Sheet1.Range("A1:D25")
        If rngCell.Value > 50 Then
            rngCell.Interior.Color = vbGreen
        Else
            rngCell.Interior.Color = xlNone
        End If
    Next rngCell

End Sub
```

Explanation –

We declared a Range type variable rngCell, which will refer to every cell in For Each loop iteration. Then we initiate For each Loop. Inside for each loop, we put an IF statement to check if cell value is greater than 25 or not, if condition is TRUE, it will execute statement written in TRUE block and highlight cell in green color, if condition is False, it will remove any color from the cell.

```
Example –

Function isSheetAvailable(strSheetName As String) As Boolean

    Dim wksSheet As Worksheet
    For Each wksSheet In ThisWorkbook.Worksheets
        If wksSheet.Name = strSheetName Then
            isSheetAvailable = True
            Exit For
        End If
    Next wksSheet

End Function
```

Explanation –

Here we have a function which will return TRUE if a specific sheet is available or not, We are passing a sheet name as a argument in this function (you will learn argument passing technique next in this chapter). We declared a worksheet type variable `wksSheet`. Then we initiate For Each Loop to iterate each worksheet in the workbook. Inside For each loop we have in IF statement to match worksheet name with the iterating object, if variable `strSheetName` value match with `wksSheet` object, it will assign TRUE on function name.

Do Loop

Unlike For Loops, Do Loop need a condition to determine how many time loops will iterate the statements. There are two types of Do Loop. 1. Do While 2. Do Until.

Example – Have you ever noticed ATM Machine, That will keep asking you if you want to continue transaction after you complete your transaction. If you press YES, it restarts the process. Let's see how that process works.

Do While Loop

```
Sub ATMMachineProcess()

Dim lngCounter As Long
Dim blnReponse As Boolean
Dim dblMoneyWithdraw As Double

'--Initiate loop
Do
  lngCounter = lngCounter + 1
  dblMoneyWithdraw = dblMoneyWithdraw + CDbl(InputBox("Please enter  Amount"))
blnReponse = MsgBox("Do you want to continue?", vbQuestion + vbYesNo + vbDefaultButton1) = vbYes

Loop While blnReponse = True '-- Or you can use only blnResponse, because it is a Boolean variable and its value will either be TRUE or False

 MsgBox "You have done " & lngCounter & " Transcation, total Amount " & dblMoneyWithdraw, vbInformation

End Sub
```

Above program will run the statement inside Do Loop without checking the condition because we are checking condition in the last line of loop, but we check condition at starting of the loop, have a look

```
Do While blnReponse
  lngCounter = lngCounter + 1
  dblMoneyWithdraw = dblMoneyWithdraw + CDbl(InputBox("Please enter  Amount"))
blnReponse = MsgBox("Do you want to continue?", vbQuestion + vbYesNo + vbDefaultButton1) = vbYes
```

```
Loop
```

While will keep iterating statements inside Do Loop, if blnResponse is TRUE every time**.** As soon as blnResponse value is False, Execution will come out of Loop.

Do Until Loop

While will keep iterating statements if condition is TRUE, Until is same but it will keep iterating statements **until** the condition become TRUE, it means it will keep continue if condition is FALSE.

```
Do
     '–Statements here

Loop Until msgbox("Do you want to Exit?", vbQuestion + vbYesNo + vbDefaultButton1)
= vbYes
```

Execution will come out of loop if you press Yes

Calling Procedures

Best practice in any programming language is to break your main program in many small program and then in the Main program call those small programs to do their part and go out of memory. So, prior to writing your code you should determine type of work and develop code for each process separately.

When you have separate programs, those are.

1. **Easy to debug**
    a. When error pops up, these codes will be very easy to debug. And if required to replace the entire process, you can just replace that small process to new one, you don't need to mess up whole program.
2. **Easy to understand**
    a. 1+1 is of course easy to understand in comparison to **$E=Mc^2$** , Meaning that if you have formula sliced in to multiple parts, it will be easy to understand and make amendments

3. **Using less memory**
    a. Memory management is very important. When you have small process, they will use less memory in comparison to large process, when you call small process in Main program one by one, they will load into memory, will do their job and go away, then

next process will come and so on. But if you have one large program, all variables, objects etc. will load into memory and may make your program slower.

Example –

```
Sub MainProgram()
     Call  Process1
     Call Process2
End Sub

Sub Process1()
     '–Statements here
End Sub

Sub Process2()
     '–Statements here
End Sub
```

Procedures in the same module can be called doesn't matter if they are public or private, but if you want to call procedure from another module, those procedure must have declared as public else they cannot be accessed. You can also call procedures from another workbook, but you need to this special command to do so.

```
Application.Run "'" & OtherWorkbookFullPath & "'!ProcessName",Aguments…
```

Passing Arguments/Parameters

Most of the time when we call other process we need to supply some inputs to the calling function, So that they can work dynamically based on the inputs we supplied. Look at the example below.

```
Sub Main()

     Dim bytDay as byte
     Dim strDayName as string
     bytDay = 1
     strDayName= GetDayName(bytDay)

End Sub
Function GetDayName(byval bytDayNumber As Byte) As String

    If bytDayNumber >= 1 And bytDayNumber <= 7 Then
        If bytDayNumber = 1 Then GetDayName = "Sunday"
        If bytDayNumber = 2 Then GetDayName = "Monday"
        If bytDayNumber = 3 Then GetDayName = "Tuesday"
        If bytDayNumber = 4 Then GetDayName = "Wednesday"
        If bytDayNumber = 5 Then GetDayName = "Thurday"
        If bytDayNumber = 6 Then GetDayName = "Friday"
        If bytDayNumber = 7 Then GetDayName = "Saturday"
    End If

End Function
```

Above function is retuning weekday name, we supplied the day number and based on that day number it will return day name. e.g. if you supply 1, it will return Sunday.

There are two methods we pass the arguments in a procedure

1. Pass byVal
2. Pass byRef

Passing Variable byVal

See the previous program, you will notice that there is keyword we are using `byVal` just before `bytDayNumber`. When we use `byVal`, `GetDayName` function will declare a variable in memory named `bytDayNumber` and will assign the value of `bytDay` from the main program.

In other words, Passing `byVal` only supply the copy of variable value, not the memory reference. Called function will create its own variable in memory.

Passing Variable byRef

Every variable declared have it memory address. So, when we pass variable `byRef`, calling function will not create another variable, it will point to the memory of main function's variable. For example

```
Sub Main()

     Dim bytDay as byte
     Dim strDayName as string
     bytDay = 1
     strDayName= GetDayName(bytDay)

End Sub
Function GetDayName(ByRef bytDayNumber As Byte) As String

    If bytDayNumber >= 1 And bytDayNumber <= 7 Then
        If bytDayNumber = 1 Then GetDayName = "Sunday"
        If bytDayNumber = 2 Then GetDayName = "Monday"
        If bytDayNumber = 3 Then GetDayName = "Tuesday"
        If bytDayNumber = 4 Then GetDayName = "Wednesday"
        If bytDayNumber = 5 Then GetDayName = "Thurday"
        If bytDayNumber = 6 Then GetDayName = "Friday"
        If bytDayNumber = 7 Then GetDayName = "Saturday"
    End If

End Function
```

Now, GetDayName function will not create a variable in memory with the name of bytDayNumber, but not bytDayNumber will refer the memory address of bytDay variable from the main program, if you try to change value in bytDayNumber in calling function (GetDayName), value of bytDay in the main program will also change, because both variables are using same memory address.

Optional Parameters

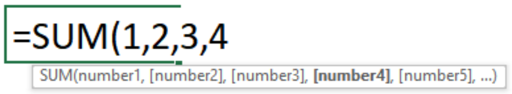
When you call a parameterized procedure, you need to supply all the parameters to the calling procedure, But there is a way to make those parameters Optional. You can declare your parameter following by keyword **Optional. Optional Parameters must be declared as last parameter.** See example

```
Function CheckifSheetExists(strSheetName As String, Optional wbkFile As Workbook)
As Boolean

'-- if wbkFile is missing (not supplied),check if the passing worksheet is existing
in thisworkbook
'-- else check worksheet in the supplied workbook.
    Dim wbkCheck As Workbook
    Dim wksSheet As Worksheet


    If Not wbkFile Is Nothing Then '-- if passed
        Set wbkCheck = wbkFile
    Else
        Set wbkCheck = ThisWorkbook '-- if missing
    End If

    For Each wksSheet In wbkCheck.Worksheets
        If wksSheet.Name = strSheetName Then
            CheckifSheetExists = True
            Exit Function
        End If
    Next wksSheet

End Function
```

In above example, `wbkFile` is an optional parameter, means if programmer don't supply this when calling this function. It will use other options to run the procedure, in this `CheckIfSheetExists` function will check sheet availability in the `ThisWorkbook` (`ThisWorkbook` **is the workbook which your code is executing in**)

Parameters Array or ParamArray

Have you ever noticed SUM function in excel? Sum function keep adding parameters when you keep supplying value to it. Also notice that all other parameters are coming in big brackets "[]" except first one. It means first parameter is required and the second parameters is an optional `paramArray`. Unfortunately, But VBA doesn't support Optional ParamArray, see rules after Example


=SUM(1,2,3,4

SUM(number1, [number2], [number3], **[number4]**, [number5], ...)

Sum functions programming might be very complicated, but here is a simpler version, you can understand ParamArray by this example

```
Function SumNumber(dblNumber1 As Double, ParamArray Arg()) As Double

    Dim lngR As Long
    Dim dblResult As Double

    '--Iterate each item of array of parameters
    For lngR = LBound(Arg) To UBound(Arg)
        If IsNumeric(Arg(lngR)) Then
            dblResult = dblResult + CDbl(Arg(lngR))
        End If
    Next lngR
    dblResult = dblResult + dblNumber1
    SumNumber = dblResult

End Function
```

Rules –

- ParamArray must be last parameter
- ParamArray must be a variant type, Means you must not declare a data type for `Paramarray`
- You cannot use Optional parameter with ParamArray, as both Optional and `Paramarray` must be declared as last parameter.

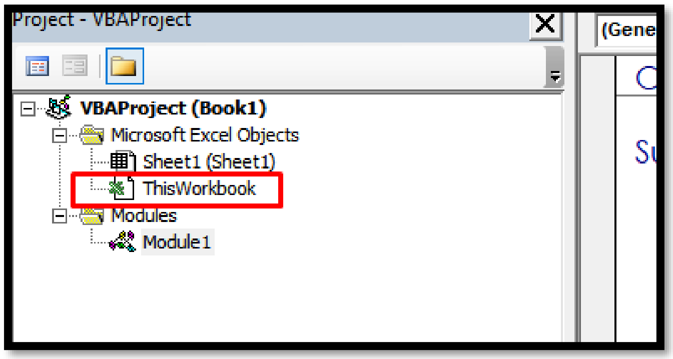
There are few things that you need to understand about Excel workbook. Excel provides many types of extensions which you can save your file in, But if you want to save macro in your file, there are certain file type you need to choose. Most of the time it is Macro Enabled Excel Workbook or .xlsm which you will use, but there are other types which also support VBA such as .xls, .xlsb, xla, xlam.

How to refer workbook

To refer any object in VBA, best practice is to make object and then use it further in program. There are two pre-defined objects in VBA which you can use to refer a workbook.

ThisWorkbook

When you use `ThisWorkbook` in code, this object will refer the workbook you are executing your code in. you don't need to set any special object for the workbook you are writing code, just use `ThisWorkbook` object. Like if you want to know the path of the file, you can use `ThisWorkbook.Path`



ActiveWorkbook

`ActiveWorkbook` object refer to the workbook which is currently activated on the main window. Best practice is to avoid using this one.

Create Object of workbook

If you want to refer any other working instead of the `ThisWorkbook`. You must create an object of that. See the syntax below.

```
Dim wbkData As Workbook
Set wbkData = Workbooks("Data.xlsm")
```

Above code will set Data.xlsm file in `wbkData` object. But only if Data.xlsm is opened. If you want to open the workbook and set it into object at the same time you will use this syntax. And most of the time we do like this.

```
Dim wbkData As Workbook
Dim strFilePath As String
strFilePath = "Set your path here"
Set wbkData = Workbooks.Open(strFilePath)
```

So, for best practice, either use `ThisWorkbook` if you want to refer the workbook you are running your code in. or set an object if you want to work with another workbook. Never use `ActiveWorkbook` or any other object which start with Active word. It can cause some serious error in your program.

In Above code, you see how you can open a workbook and set it into an object. Now, further you can use this object to perform all operation.

Open a new workbook

You can use .Add method of workbooks object to open a new file. See the syntax

```
Set wbkOutput = workbook.add()
```

Close workbook

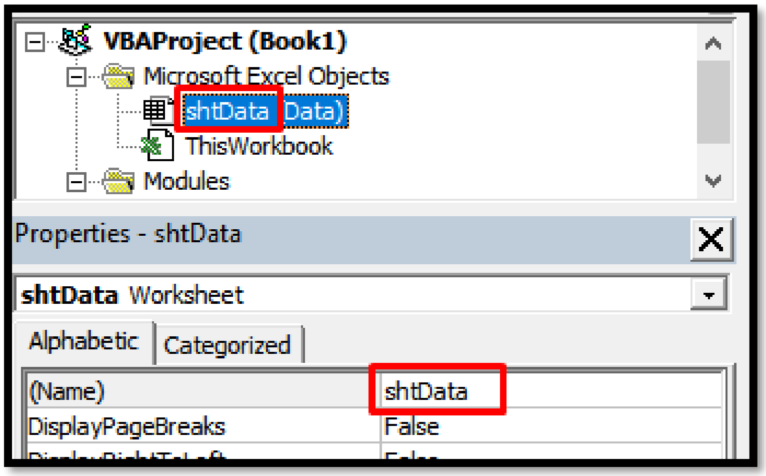
After .close put 0 if you want to close without saving or 1 if you want to save and close.

```
wbkOutput.Close 1
```

Next thing you need to understand is how to manipulate worksheets, How to Add, Delete, Rename, but most important thing to understand is how to refer a worksheet in your program.

How to refer a worksheet

There are couple ways you can refer worksheet in VBA. Best way to use the worksheet's code name to refer a worksheet.



(Name)	shtData
DisplayPageBreaks	False

If you want to refer any worksheet in `ThisWorkbook`, you should always use code name. like in above screenshot, notice the highlighted name `shtData.shtData` is the code name for worksheet Data. You can change this name in the property window. When you use code name you don't need to refer workbook, you also don't need to set an object. With code name, VBA knows that the sheet is available in `ThisWorkbook`. If you want to refer worksheet in other workbook, you need to use this syntax.

```
Dim wbkData As Workbook
Dim wksSheet As Worksheet
Dim strFilePath As String
strFilePath = "Set your path here"
Set wbkData = Workbooks.Open(strFilePath)
Set wksSheet = wbkData.Worksheets("Data")
```

Here wksSheet will refer the Data worksheet in wbkData workbook. You cannot use worksheet's code name from another workbook.

Add/Remove Worksheets

You can use the .Add or .Delete method of worksheet object to Add/Remove worksheet.

```
Thisworkbook.worksheets.add Before:=Thisworkbook.worksheets(1)
```

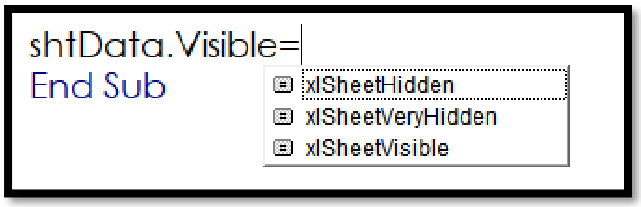
Above code will insert a worksheet before first available worksheet in the file. You can use below code to delete a worksheet.

```
Application.DisplayAlerts=False
Thisworkbook.Worksheets("Data").Delete
Application.DisplayAlerts=True
```

When we delete a worksheet, excel displays a popup asking for permission, because once a sheet is deleting it cannot be Undo. But in Automation, you cannot have user to click the popup every time when it code is trying to delete worksheet. So, use `Application.DisplayAlerts = False` to disable that popup. And after deleting sheet you must turn it back to TRUE. Because if it kept False it will change the application setting and you might delete a worksheet accidently without having popup asking you the permission.

Hide/Unhide worksheet

You can hide or unhide worksheet by using .Visible property of worksheet object. See example



```
shtData.Visible=
End Sub
```

`xlSheetHidden` will hide the sheet, but user can unhide this from excel window. `xlSheetVeryHidden` will hide worksheet but user will not be able to unhide it from excel window. Hidden sheet with `xlSheetVeryHidden` can only be visible by VBA. `xlSheetVisible` will make hidden sheet visible.

You can also assign True/False to .Visible property. If `shtData.Visible=True` , it will unhide the sheet, if it is False then it will hide the sheet with `xlSheetHidden`

Copy/Move Worksheet

You can use .Copy or .Move methods to do so.

```
shtData.Copy After:=shtData
```

this will copy the shtData sheet right after it. If you want to copy a sheet to other workbook you need to refer that other workbook in destination.

```
shtData.Copy After:=wbkOutput.Worksheets(1)
```

this code will copy shtData in wbkData workbook after first worksheet.

Referring a Range in VBA?

There are couple ways that you can use to refer a Range in VBA, Say, if you want to enter your name in A1, you can use any of these methods.

```
Range(“A1”).value = “Your Name”
[A1] = “Your Name”
Cells(1,1).value =”Your Name”
Cells(1, “A” ).value =”Your Name”
```

If you have defined a named range in excel you can use that here too.

```
Range(“rngNameRange”).value = “Your Name”
```

```
Note :-  if your name range contain a formula, it won’t work with above method.
```

Best Practice is to declare a Range type object variable, Assign a Range in that object and then use this object further in your program. See example here.

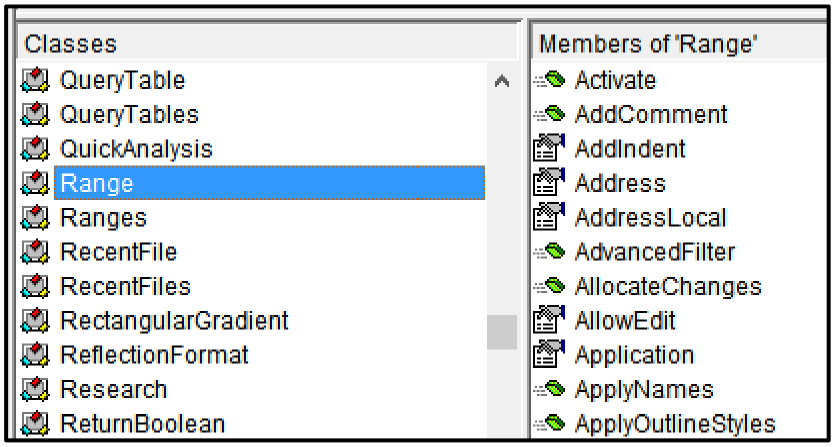
```
Sub AssignName()

        '--Declare a Range type Variable
    Dim rngRange As Range
        '--Assign Range to the Range type variable
    Set rngRange = Range("A1")
        '--Assign Your Name to the range.
    rngRange.Value = "Your Name"

End Sub
```

Example 2-

Let's say, you want to print row number in a column by VBA, see how you will do it

```
Sub RowNumber()
    '--Declare a Range type Variable
    Dim rngRange As Range
    '--Declare a Range variable to run For Each loop
    Dim rngCell As Range
        '--Assign Range to the Range type variable
    Set rngRange = Range("A1:100")
        '--Initial for each loop
    For Each rngCell In rngRange.Cells
        '-- Assign Row number to cell
        rngCell.Value = rngCell.Row
    Next rngCell
End Sub
```

There are so many operations that you can do with Range object, You can see all the properties and method by looking at object browser. To open Object Browse press F2 in VBE(Visual Basic Editor) and in the Class windows select Range Class.



On the right side, you will notice that some icons are green, and some are Black, you should know that all green ones are methods and black ones are properties. You will notice that you can find all methods here to automate the things you do manually. Like Subtotal, Sort, Group, Ungroup, Merge and so on. List is too long, but I will show you a few properties and method in this work you that you have idea of how to use them all by your own.

We will use our best method to use Range object by declaring a variable, Assign a Range to that variable and work with it.

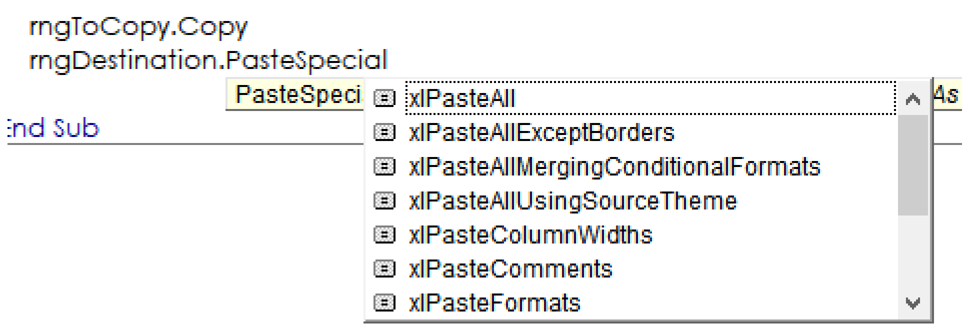
How to copy and paste Range

```
Sub CopyPasteRange()
    '--This procedure will copy Range A1:A100 to B1

    '--Declare Variables
    Dim rngToCopy As Range
    Dim rngDestination As Range

    '--Set Range to the variables
    Set rngToCopy = Range("A1:A100")
    Set rngDestination = Range("B1")

    '-- Copy Range to destination range, This method will copy paste ALL copied
    '-- Data, Formats, Conditional Formats, Data Validation etc
    rngToCopy.Copy rngDestination

End Sub
```

This is how you can use Paste Special in VBA

```
    rngToCopy.Copy
    rngDestination.PasteSpecial xlPasteFormats
```

You can use all paste special parameters by choosing from the list when you use PasteSpecial method in VBA

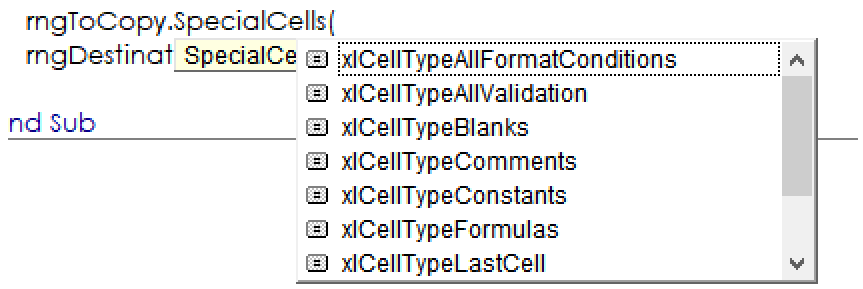


```
rngToCopy.Copy
rngDestination.PasteSpecial
End Sub
```

Special Cells

VBA provides a way to work with Special Cells, this is how you can access those Special Cells in VBA.

```
rngToCopy.SpecialCells(xlCellTypeVisible).Copy
rngDestination.PasteSpecial xlPasteValues
```

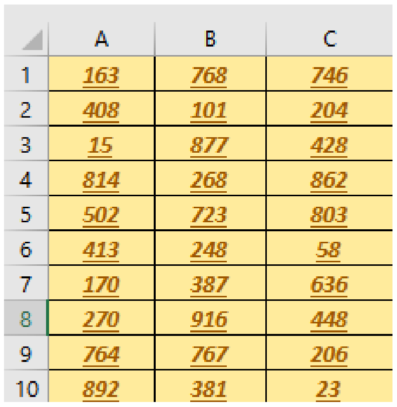


```
rngToCopy.SpecialCells(
rngDestinat SpecialCe
nd Sub
```

Format a Range

There are so many properties and methods are available in VBA to manipulate Range object to give a nice formatting. See few of them here.

```
Sub FormatRange()

    Dim rngRange As Range

    Set rngRange = Range("A1:C10")
    With rngRange
        With .Font
            .Bold = True
            .Color = 22428
            .Italic = True
            .Underline = True
        End With
        .Interior.Color = 10284031
        .Borders.LineStyle = xlContinuous
    End With

End Sub
```



	A	B	C
1	163	768	746
2	408	101	204
3	15	877	428
4	814	268	862
5	502	723	803
6	413	248	58
7	170	387	636
8	270	916	448
9	764	767	206
10	892	381	23

Creating a Dynamic Range in VBA

Always try to write a dynamic code, for example, let’s say want to copy a range to another sheet, you have a range A1:B100, and you write this code

```
Sheet1.Range("A1:B100").copy Sheet2.Range("B1")
```

This code will work. But if you append more rows after A100, this code will not be able to copy those new rows. So, to make your code read those new rows dynamic you need to use either of these methods.

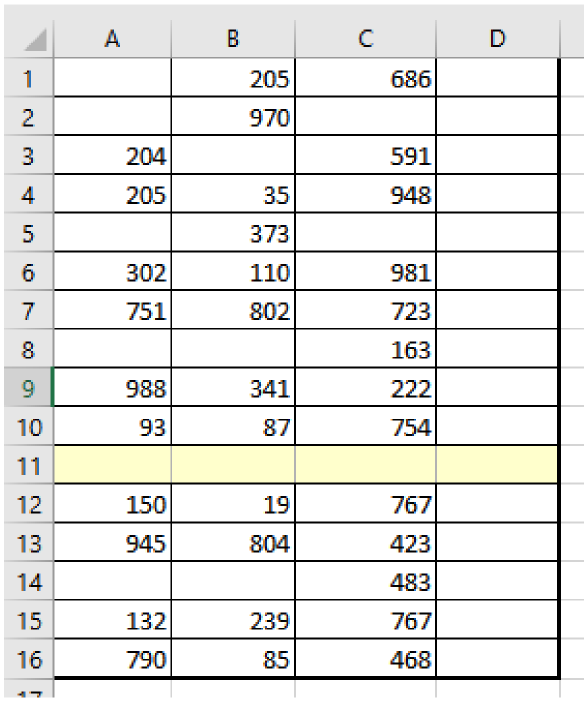
1. Create Range By CurrentRegion Method
2. Determine the LastRow prior to set a range
3. Create range by usedRange method

Create Range By CurrentRegion

Current Region is Range object property which return Range object and contains all cell which have no space in between. Like a set of Range which have no blank rows and columns in between. See an example of CurrentRegion here.

If you write `Range("A1").CurrentRegion`, it will return a range referring to this entire section of A1:C10. You can use any cell address instead of A1, and it will return same Range reference.

Notice the highlighted blank row in this range, this blank row is separating two current regions. If you write Range("A12").CurrentRegion, it will return a section of A12:C16



	A	B	C	D
1		205	686	
2		970		
3	204		591	
4	205	35	948	
5		373		
6	302	110	981	
7	751	802	723	
8			163	
9	988	341	222	
10	93	87	754	
11				
12	150	19	767	
13	945	804	423	
14			483	
15	132	239	767	
16	790	85	468	

But if you take current Region of D1 or D12, it will include col D in CurrentRegion too. Same like blank row is separating current regions, a blank column will do so.

Find Last used row in data

Let's say you got a data file, that contain survey data with multiple columns. Some of column doesn't have detail for every record such as Email ID, Address etc. To find the last row you need to first determine a column which have data in all cells. Such as Record ID. Let's say col A is record ID column, This is how you can create Dynamic range by finding last row.

```
Sub DynamicRange()
    Dim rngRange As Range
    Dim lngLastRow As Long
    Dim rngDestination As Range
    '---Use of End Method of Range object
    lngLastRow = Range("A" & Rows.Count).End(xlUp).Row
    Set rngRange = Range("A1:D" & lngLastRow)
    Set rngDestination = Sheet2.Range("B1")
    rngRange.Copy rngDestination
```

End Sub

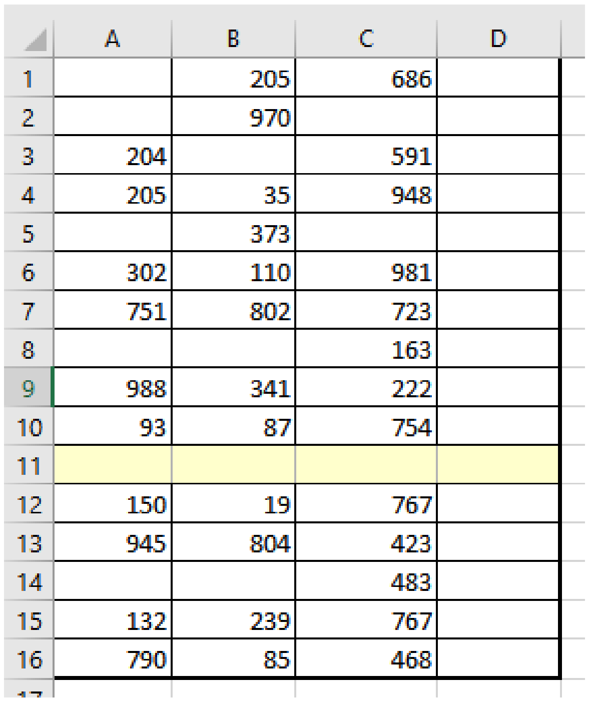
We used End property of Range object, which also return a range object, and then we used .Row property of range object, which return the row number. So basically, By `“A” & Rows.count` we are telling VBA to go down to the last cell in the worksheet in Column A. e.g A1048576 and then press Ctrl + Up key, when you go to last cell and press ctrl+Up it will select the last cell which have a value.

Create Range by UsedRange

UsedRange is also a property of Worksheet object which returns Range object and contain all rows which have data in any cells or have been used.

```
Dim rngRange As Range
Set rngRange = Sheet1.UsedRange
```

Above code will return range A1:D16



	A	B	C	D
1		205	686	
2		970		
3	204		591	
4	205	35	948	
5		373		
6	302	110	981	
7	751	802	723	
8			163	
9	988	341	222	
10	93	87	754	
11				
12	150	19	767	
13	945	804	423	
14			483	
15	132	239	767	
16	790	85	468	

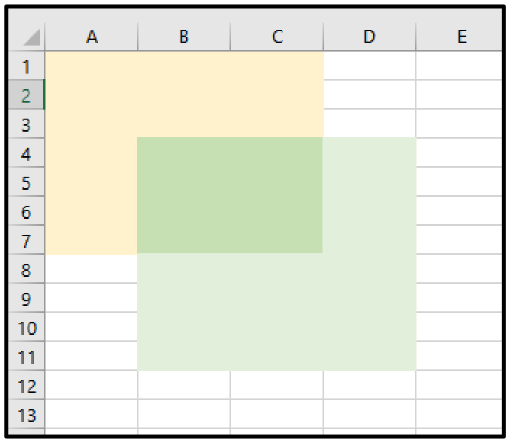
Choosing the best way

However, all three methods can be used to determine the data range, but we need to identify the best one which suit our requirement

1. If you have data without blank rows and columns, you should go with `CurrentRegion` method
2. If you have blank rows, you should go with `lastRow` or `UsedRange` method.

How to get intersection of two Ranges?

So many times, we need to get the intersected range, VBA provide a great function to do so. Have a look.



```
Dim rngRange as Range
Dim rngRange2 as Range
Dim rngIntersect as Range
Set rngRange = Range(“A1:C7”)
Set rngRange2 = Range(“B4:D11”)
Set rngIntersect = Intersect(rngRange,rngRange2)
Debug.print rngIntersect.address
```

Above program will print $B$4:$C$7 in immediate window.

How to get combination (Union) of Range?

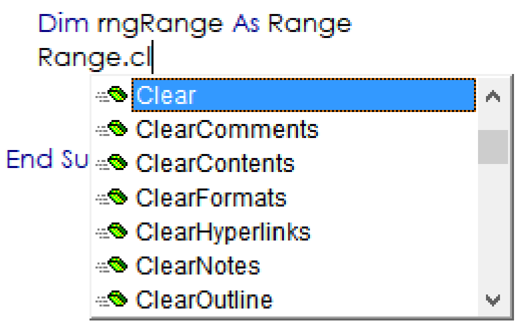
Union is a VBA function which can set multiple non-continuous ranges into a variable, For example if you want to highlight rows only with certain value in column A. you will need to use Union method. Have a look at the example here –

```
Sub HighLightRow()

    Dim strInput As String
    Dim rngRange As Range
    Dim rngUnion As Range
    Dim rngCell As Range

    '–The value you want to search in column A
    strInput = InputBox("Please enter a value")
     '–Set Range in to the variable
    Set rngRange = Range("A1").CurrentRegion

'–Initiate for each look to iterate every cell in col A
    For Each rngCell In rngRange.Columns(1).Cells
      '–check if cell value is equal to entered value by user
        If rngCell.Value = strInput Then
           '–Check if rngUnion is blank
            If rngUnion Is Nothing Then
                '–if value in column A match with entered value
                 '–Set that cell row into range variable
                Set rngUnion = rngCell.EntireRow
            Else
                '–if there is already a range set in the variable
                 '–Union next matched cell with the previously set range
                Set rngUnion = Union(rngUnion, rngCell.EntireRow)
            End If
        End If
    Next rngCell

    If Not rngUnion Is Nothing Then
        rngUnion.Interior.Color = vbYellow
    End If

End Sub
```

Clear a Range

VBA provide different method to clear different kind of attributes from a Range. See the image below to see all the methods you can use to clear range.

If you want to clear everything from a Range, you will use .Clear Method. As names implies, to clear all comments from a range, you will use `.ClearComments`. if you want to clear only data and not formatting, validations, conditional formatting etc, you will use `.ClearContents`.



```
Dim rngRange As Range
Range.cl
End Su
```

Insert Rows or columns

You can use .Insert method to insert rows and column of Range objects. You need refer appropriate Range to use this method. Suppose you want to insert a row. You will use it like this.

```
Range(“A2”).EntireRow.insert
```

To insert a column you will use

```
Range(“A2”).EntireColumn.insert
```

If you want to insert more than one row or column, you need refer Range which contain the same number of rows or columns you want to insert. Say , if you want to insert 5 rows. You will use it like this

```
Range(“A1:A5”).EntireRow.insert or Range(“A1”).Resize(5).EntireRow.Insert
Range(“A1:E1”).EntireColumn.insert or Range(“A1”).Resize(,5).EntireColumn.Insert
```

Delete Rows or Column

You can use .Delete method to insert rows and column of Range objects. You need refer appropriate Range to use this method. Suppose you want to insert a row. You will use it like this.

```
Range(“A2”).EntireRow.Delete
```

To delete a column, you will use

```
Range("A2").EntireColumn.Delete
```

If you want to delete more than one row or column, you need refer Range which contain the same number of rows or columns you want to delete. Say , if you want to delete 5 rows. You will use it like this

```
Range("A1:A5").EntireRow.Delete or Range("A1").Resize(5).EntireRow.Delete
Range("A1:E1").EntireColumn.Delete or Range("A1").Resize(,5).EntireColumn. Delete
```

Filter Range

You can filter your data by using .AutoFilter method of Range object. See this example here.

```
Range("A1").CurrentRegion.AutoFilter 1, strCountry
```

You will notice in above syntax that, we have created a dynamic range by using .CurrentRegion and then we are filtering col A (by giving 1 as filter field). strCountry variable have the filer value. So, when this line will execute, it will filter col A. Now, lets see how you can filter multiple countries in A column.

```
Sub FilterData()

    Dim strCountry1 As String
    Dim strCountry2 As String
    Dim rngRange As Range

    strCountry1 = "USA"
    strCountry2 = "INDIA"
    Set rngRange = Range("A1").CurrentRegion

    rngRange.AutoFilter 1, strCountry1, xlOr, strCountry2

End Sub
```

Notice that we are using `xlOR` operator in filter, So, if value match with either one of the countries it will filter that out. But what if you have a list of countries to filter. See the next example.

```
Sub FilterData()

    Dim varCountries
    Dim rngRange As Range

    varCountries = Array("INDIA", "USA", "JAPAN", "CHINA", "RUSSIA")
    Set rngRange = Range("A1").CurrentRegion
    rngRange.AutoFilter 1, varCountries, xlFilterValues

End Sub
```

In Above example, we have created an array of countries and then pass that to the criteria, but you need to remember that when we are passing array you need to supply `xlFilterValues` as operator.

Sort Range

There are two ways you can sort data in VBA. If you want to sort data on up to three columns, you can use this syntax.

```
Sub FilterData()

    Dim rngRange As Range
    Set rngRange = Range("A1").CurrentRegion

    rngRange.Sort key1:=rngRange.Cells(1, 1), order1:=xlAscending,
key2:=rngRange.Cells(1, 2), order2:=xlAscending, key3:=rngRange.Cells(1, 3),
order3:=xlAscending, Header:=xlYes

End Sub
```

Notice that we have given three Keys and orders to sort data on three columns. Remember that in the key you need to only supply the first cell of sorting column. Since `rngRange` referring multiple column, we are using `rngRange.cells(1,1)` to refer first cell or column A. also see the last parameter Header, if your data have headers, assign `xlYes` to this parameter or else it will sort the headers along with your data.

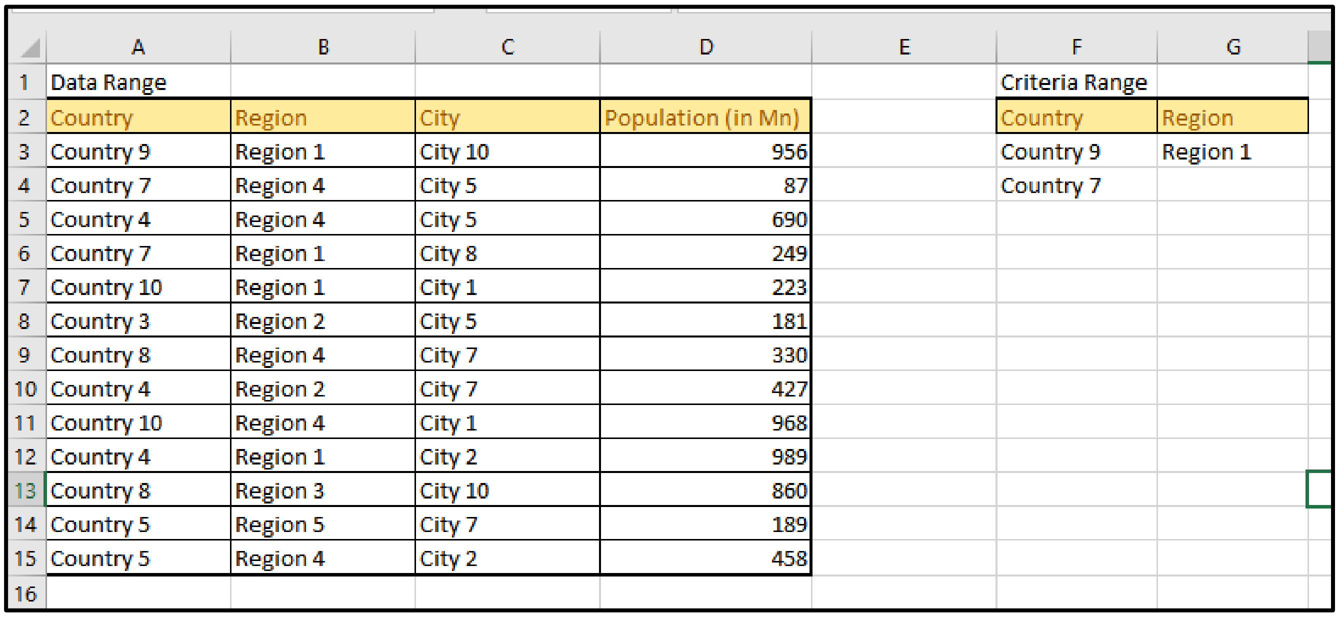
If you have more than three columns you want to sort your data on. You will use this method.

```
Sub FilterData()

    Dim rngRange As Range
    Set rngRange = Range("A1").CurrentRegion

    With ThisWorkbook
        With Worksheets("Sheet1").Sort
            '–Remove all sort fields from worksheet
            .SortFields.Clear
            '–Add as many sort fields as you want.
            .SortFields.Add rngRange.Cells(1, 1), xlRows, xlAscending
            .SortFields.Add rngRange.Cells(1, 2), xlRows, xlAscending
            .SortFields.Add rngRange.Cells(1, 3), xlRows, xlAscending
```

```
                .SortFields.Add rngRange.Cells(1, 4), xlRows, xlAscending
                '–Assign a range to the sort
                .SetRange rngRange
                 '–xlYes if your data have headers
                .Header = xlYes
                .SortMethod = xlPinYin
                 '–Then apply sort at last.
                .Apply
            End With
        End With

    End Sub
```

Advance Filter

As name implies, Advance Filter is little more advance than the normal filter, for using advance filter you can use as many fields you want in the shorter syntax, even you can filter unique data and copy it to another location. This filter work very well on large data. For advance filter you need two ranges. 1. Data Range 2. Criteria Range. Criteria range must have the headers matching with the headers in Data Range to tell Advance Filter the fields to filter data on. See the syntax below.

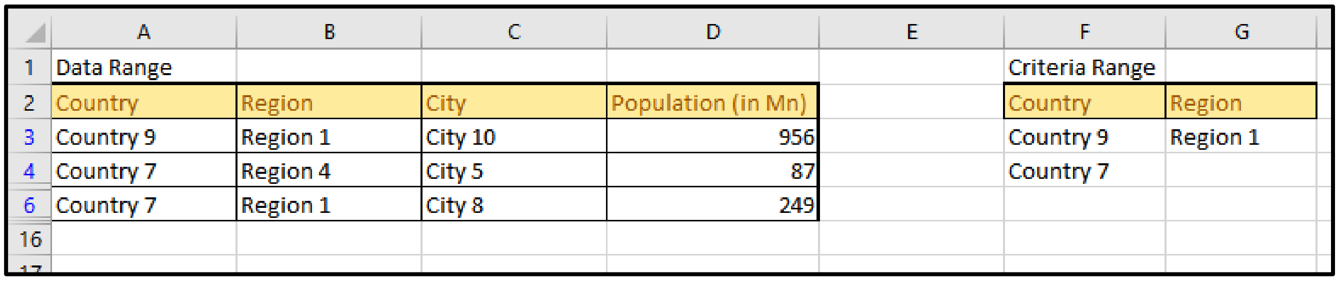


	A	B	C	D	E	F	G
1	Data Range					Criteria Range	
2	Country	Region	City	Population (in Mn)		Country	Region
3	Country 9	Region 1	City 10	956		Country 9	Region 1
4	Country 7	Region 4	City 5	87		Country 7	
5	Country 4	Region 4	City 5	690			
6	Country 7	Region 1	City 8	249			
7	Country 10	Region 1	City 1	223			
8	Country 3	Region 2	City 5	181			
9	Country 8	Region 4	City 7	330			
10	Country 4	Region 2	City 7	427			
11	Country 10	Region 4	City 1	968			
12	Country 4	Region 1	City 2	989			
13	Country 8	Region 3	City 10	860			
14	Country 5	Region 5	City 7	189			
15	Country 5	Region 4	City 2	458			
16							

See the two range above, A:D is my data range and F:G is my criteria range, lets use the code below and it will filter all rows where Country 9 and Region 1, it will also filter all rows with Country 7 with all region.

```
Sub FilterData()

    Dim rngData As Range
    Dim rngCriteria As Range

    Set rngData = Sheet1.Range("A1").CurrentRegion
    '--Exclude Title from data range
    Set rngData = Intersect(rngData, rngData.Offset(1))

    Set rngCriteria = Sheet1.Range("F1").CurrentRegion
    '--Exclude Title from Criteria range
    Set rngCriteria = Intersect(rngCriteria, rngCriteria.Offset(1))

    '-- this statement will filter data
    rngData.AdvancedFilter xlFilterInPlace, rngCriteria

End Sub
```

This is the filtered result



	A	B	C	D	E	F	G
1	Data Range					Criteria Range	
2	Country	Region	City	Population (in Mn)		Country	Region
3	Country 9	Region 1	City 10	956		Country 9	Region 1
4	Country 7	Region 4	City 5	87		Country 7	
6	Country 7	Region 1	City 8	249			
16							

If you want to copy this filtered data to another location, you can use this syntax

```
rngData.AdvancedFilter xlFilterCopy, rngCriteria, Sheet1.Range("rngToPaste")
```

if you want to filter and copy only unique data out of filtered data, you can use this syntax

```
rngData.AdvancedFilter xlFilterCopy, rngCriteria,Sheet1.Range("rngToPaste"),True
```

How to create name range

It is always best to use name range reference in VBA when you set a range object. It gives you flexibilities to insert or delete ranges without changing the reference in your code. Let's say, you are using `Range("A5")` but for some reason you had to insert a row in first row, then you will have to change your reference to A6. Because the range you wanted to set is moved down by one row. If you are using name range like `Range("rngStart")`, you don't need to worry about this movement. Name range references will automatically shift in all directions. Most of the time we create name range manually and then use them later in program, but sometimes you will need to create a name for range, you can use this syntax.

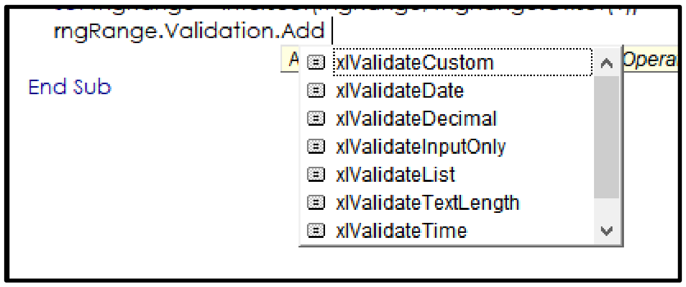
```
Range(“A1:B10”).name = “rngData”
```

Data Validation

Data validation provide the functionality to restrict user to enter invalid data. There are some set of rules you can use to validate an entry. You can also define a custom formula to specific conditions. I have developed so many projects in VBA, but most of the time I didn’t need to add validations by VBA. But sometimes, VBA is life saver when we need to add validations automatically. See the example

```
Sub AddListValidation()
    Dim rngRange As Range
    Set rngRange = Range("A1").CurrentRegion.Columns(2)
    Set rngRange = Intersect(rngRange, rngRange.Offset(1))
    rngRange.Validation.Add xlValidateList, , , "India, US, China, Canada", ""
End Sub
```

After .Add, VBA will list all type of validations option, you can choose which one you need.



```
    rngRange.Validation.Add
End Sub
```

Conclusion

I hope now you get a good understanding of how VBA works, How you can create objects and use properties and method to automate your work. I will write another part of this book soon. Which will teach you the most advance use of Excel VBA.

Write me at excel-consulting@itchat.in for

1. Suggestion of this book
2. Hire me to Excel VBA Automation
3. Personalized VBA Training.

Thanks for your time.