**VBA and MACROS Notes : (LearnIT)**

1. VBA works with other MSOFT Office applications also.
2. VBA workbook code : Quick Access toolbar : Project panel, Properties Panel, Code workbook.
3. VBA : Programming editing and running application code. In this case our application is Excel. Our host application is Excel.
4. VBA is NOT a standalone program: it can run it’s code inside a host application: in our case the host application is Excel.
5. So our VBA code will run inside Excel and must eb compatible with Excel.
6. VBA : Custom Design our own functions. IMP.
7. I need to do something for which there is no existing available Excel program available.
8. This can be achieved with VBA. For ex. Area of a Triangle. Etc.
9. VBA is a mix of old-fashioned MS programming languages with incremental additions as new versions are rolled out.
10. The beginning of every programming statement must start with the object which I want to make a change to. For example to delete/colour a cell, the cell must be selected first in Excel.
11. Enable the Developer tab -> Goto file -> Options -> Customize the ribbon and then turn on the Developer Tab. This will show the Developer tab in our Excel ribbon.
12. Developer Tab -> Code subgroup at the left side -> Record Macro.
13. Record Macro -> Watches everything that we do in Excel (for ex. Using available functions) records and then converts and stores into Code. Once that code is stored we can re-execute that whenever we want.
14. For ex : I have a set of steps but I am worried about missing a step or delegating the task to someone else : who might not know how to do one of the steps. Then one can store these steps via record Macro : In form of a VBA code and then re-use it in a hassle-free manner.
15. If you have tasks in Microsoft Excel that you do repeatedly, you can record a macro to automate those tasks. A macro is **an action or a set of actions that you can run as many times as you want**. When you create a macro, you are recording your mouse clicks and keystrokes.
16. Click on record Macro in the Code Window of the Developer tab -> Give a name to your Macro (Start with letter and avoid space in the Macro name), then assign a shortcut key : Also use Shift in the Shortcut key : Ctrl + Shift + something of your choice. Then we have an option to save the Macro : We can save it to this workbook. It can travel with the current workbook (but not work for other workbooks unless the current workbook is open) , or save it to Personal Macro Workbook : Then it can be used for any other workbook.
17. If we execute using Ctrl+Enter it will not navigate me away. If we just use Enter then it would navigate me downward one cell.
18. First select a cell and apply some formatting steps to it. (Ex. Bold Italic change colour etc. )
19. Then STOP RECORDING : these steps now have been saved to our defined Macro.
20. To test the Macro -> We can launch a new sheet and click on Macro (Developer tab) and then Run the Macro you want. The steps would be replicated.
21. MACROS are going to record with ABSOLUTE CELL REFERENCING. $A$1 is absolute cell referencing, whereas A1 is relative cell referencing. Absolute cell referencing means fixed cell reference.
22. Note that we had selected a cell at the Very start of the Macro recording: So the Macro would run at that place only : Absolute cell referencing. If we want it to run at some other place: then we should use relative cell referencing in the Code group.
23. Very Interesting: Relative Cell references Macro: For example, we want to run the macro at a different location. Remember we specify a cell location at the start of the record MACRO procedure. Suppose we record the MACRO operations in cell C2, but while running we want to run it in cell G2. Then we have to first select Use Relative References in the code pane before recording the MACRO.
24. So turn on use relative reference button and then turn on the MACRO.
25. Once we click on Stop recording, then we can click on the Macros button to run our steps which have been saved into the specified Macro.
26. Sometimes absolute references might be useful, but relative references come in handy quite often. Note that the relative referencing feature is a turn ON/OFF feature. Needs to be disabled when not in use.
27. Note : Whether we are using relative references or absolute references : before turning ON the MACRO recording we should select the cell in which we want to carry out the desired steps. And after we have started recording the Macro : then DO NOT again click on the cell (or object wherein steps are being carried out), just select the object (or cell) once before recording the MACRO, that’s all.
28. Do not assign shortcuts to Macros unless necessary. Assign Ctrl+Shift+something based shortcut if required.
29. For example the tutor did some formatting to a cell and saved it to a Macro. Then she selected all the cells on which the formatting was necessary (Ctrl+Shift+Downarrow shortcut to select all cells in a column) and then did a run of the Macro. Example changing the formatting of a column.
30. If we want to repeat the operation in different places then we should choose relative referencing.
31. Note the stop recording step. And the relative referencing option is selected once we click on it.
32. If we want to do in place changes : A formula cannot reference the same cell in which it is defined : for ex. : RIGHT(C2,4) is invalid if defined in C2, rather define in some other cell and then just paste the values in the special paste options. Paste as values is a good workaround for the circular reference error. Cyclical reference error.
33. Sort and Filter Macros work well with absolute references : so no need to choose relative references.
34. So, while manipulating the Social Security Number Column : we needed to change the whole of the column, changing a single cell did not help.
35. Data tab -> sort add levels (sorting by multiple columns). Note that the entire table needs to be selected for this sorting option to work.
36. Suppose our end users do not know how to sort or filter or do not even know that we created a Macro for this purpose, then we can still convey the MACRO information by inserting a button : Go to developer tabs and then insert a macro button. We can change the name of the button after associating it with a specific Macro.
37. When we are using a Macro to record steps like sorting and filtering a table : first select any cell present in the table, then click on record Macro and then stop after recording the steps. IMPORTANT. Also when using the MACRO to carry out the respective functions : select a cell in any other table and then run the macro. No need of selecting the Entire table : it gives rise to complications.
38. So any Macro can be buttonized!!! (Insert in developer Menu)
39. Another unrelated but important point : Suppose you are working on a sheet which can take input values from a user. Then it does some calculations using formulas and produces some results. Also, it has some text based fields as well. We want the end user only to be able to modify the input values: The we can go to Home -> Find and Select -> Numerical constants and then -> Format -> Unlock and then change colour and then Protect the rest of sheet (Format tab), from being edited. The rest of the sheet will be protected against edits. Protecting and Formatting Sheets with the Macro Recorder
40. Now on to the VBA interface:
41. Macro code can be accessed inside VBA (modules in the properties tab).
42. Macros would be stored inside Module folder in the properties pane in VBA codebook. Else we can insert one Module by going to Insert and then insert a module.
43. View -> Immediate window : run little bits of code and ask questions of our workbook. Example : ?Worksheets.Count and press Enter.
44. So immediate window -> A place to test out your code as well as query your worksheet.
45. Object Browser in the View tab inside VBA codebook. This contains the libraries which can help us to code in VBA. View->Object Browser. Object Browser and Immediate Window.
46. Project Properties Code Immediate and Object
47. If we select the
48. Common area between row and column headers selects everything in the worksheet.
49. From the moment we start to record a Macro, we get a new folder in the Projects pane of the VBA codebook. There we can see a new module macro being created. We can see the equivalent VBA code for the particular Macro.
50. Sub and End Sub : beginning and End statements. Sub : sub-procedures.
51. ‘ add comments to our Macro.
52. So a simple Macro of Times New Roman text font: recorded a larger quite unnecessary code. Unnecessary stuff were also recorded in the so generated Macro code.
53. Cells.Select : The code is to be run for the selected cells ( Remember that initially we had selected some cells before recording the Macro).
54. With and End With
55. Sub : Subprocedure. Run the Sub button is available in the VBA codebook.
56. So we can run the VBA code from the VBA codebook equivalent to running the equivalent Macro.
57. So from the default generated code of the selected Macro, we can delete some lines to trim upto what’s required. For example we have a change font macro which contains unnecessary lines of code (default).
58. Sub is followed by the Macro name. Then we can edit comments too. And then of course we can edit the code which generated the Macro.
59. So code can commented out using ‘.
60. With and End With would be used for start and end of the codes.
61. Right click on the empty space of the quick access toolbar. Customize and then click and drag the tools to place into the Quick access toolbar.
62. Step Into and Compile Project and Toggle Breakpoint place in the Quick access toolbar.
63. Anything a Macro does to your spreadsheet we cannot undo : Ctrl+Z doesn’t work for Macro. So never run a Macro on the original data unless we are extremely confident.
64. Step Into : The tool which can help our Macro run line by line. In the VBA editor of course.
65. It runs the code line by line. For example : start would be sub MACRO\_NAME : Then we can see what the Macro is accomplishing step by step.
66. We can call another Macro from the current Macro : Just write the Macro name without using (), parenthesis is used for defining a Macro.
67. So debugging a Macro VBA code is very similar to debugging which takes place in Python.
68. So it’s always better to test our Macro/VBA code on backup files. After we are sure 100% of it’s working we can go on to the original file.
69. Sheets Tables Charts Columns Rows are all objects and then we can choose what do with them. Start statements with Objects.
70. Properties : Describe the objects.
71. Methods : What does the object do?
72. Rows(“n:m”).Insert : Insert some rows starting at the nth row and ending at the mth row : m>n. Also similar for inserting Columns. So here Insert is a method pertaining to the Rows object.
73. Range(“A1”).Value = “Employee ID” : Value is a property of the A1 cell object here.
74. Rows(“1:1”).Font.Bold = True : Font change of the whole row.
75. Range(“A1”).Select just places the cursor on the cell A1. Deselection is a pretty good step.
76. Insert drop down menu -> Insert a new Module. Modules can have multiple Macros in it.
77. Insert a Procedure. A sub procedure is just like a Macro. IT can call other sub procedures within it. Also it can be called if Public. Function Procedures are different : we can build a custom function out of it. For example the UNIQUE function doesn’t exist by default in Excel-2019. We can construct a user defined Excel procedure to make up our own function.
78. Rows(“1:1”).Insert : Insert a row at the top. Note that when we call methods in VBA using the . operator, using objects, then we do not need to put parenthesis : ().
79. Range(“A1”).Value = “Emp ID” : Header cell named.
80. Different methods to select a cell : Offset and all.
81. We can also define variables : myr = “A1”, Range(myr)
82. Range().Range() to offset.
83. .Select method to select the cell.
84. .Offset(down,right)
85. Range() : Absolute cell reference
86. Selection : Dynamic cell reference for ex. Selection.Font.Bold and Selection.Interior.Color
87. Cell color and themecolor
88. Range(“A1”).Value = “bhushan”
89. Range(“A1”).Clear similarly Selection.Value and Selection.Clear
90. ActiveSheet.Name = “Sheet 4” : Active sheet name changed. For sheet based objects, rather than using Selection to point to the current object under selection, use ActiveSheet instead.
91. Sheets(1).Select : Selects the first sheet in the Workbook.
92. Sheets(1).Select followed by ActiveSheet.Name = “Sheet 1” would rename the first sheet in the workbook.
93. Select is a Method. Most often to select a cell, but could be used with a group of cells as well as sheets. Once we select an object, that object can be referenced (used) using Selection. Selection will simply point to the current object which is selected. Selection is an object itself : whatever has already been selected.
94. When we are in a list of contiguous values : and then we do a Ctrl+A : then it selects whole of the contiguous region associated with that cell. The Current Region property does the same : it returns the current region associated with the Selection. Selection.CurrentRegion.Select
95. Selection.CurrentRegion.Select : Selects the current contiguous region associated with the current cell/selection. Note that for Selection to work properly a cell/region should be selected. (Object must be known)
96. Whatever can be done by recording a Macro, should be done : Unless necessary do not write VBA code.
97. Variables : Containers to store numbers/text/charts/cells/anything which is a object or a value.
98. Dim aNumber as Integer : Dim stands for declare in memory. aNumber is the variable name which will store an Integer value. Then data-size allocated to the variable is fixed.
99. Dim aNumber Integer after that aNumber = 2 after that Range(“A5”).Value=aNumber. Everything is self-explanatory.
100. Numbers data type. Text data type. Boolean Date data types. Object data type. Range data type.
101. Range(“A5”).Value is used to either fetch a value from a Range or assign a value to a range.
102. Dim as statement is not necessary: But then the default data type assigned to the Object is the Variant data type. Which requires large amount of memory as the type of the Object is not known.
103. To Force users to put the Dim as statement : One can put the Option Explicit statement at the top of the Macro code.
104. Do not name the variables with existing objects/properties/methods, reserved keywords are not to be used.
105. Cells(x,y) is an object pointing to the cell (x,y) in your sheet.
106. Whenever we run a query in our immediate window : Always begin the Query with ? if we want to display it’s output on the next line.
107. For next loop and the do loop :
108. For Next loop :
     1. For x = 1 To 10

Code to Execute

Next x :: This acts as a closing statement here

1. Double for Next loop : Nested For Next Loop
   1. For intRow = 1 to 3

For intCol = 1 to 10

Block of Code

Next intCol

Next intRow

1. So the double for next loop was able to populate a Table. Filling up values in rows and columns.
2. A triple For Next loop can populate multiple sheets as well. Adding a 3rd dimension to the already present 2 dimensions.
3. Worksheets(x) would point to the x numbered worksheet. Returns an object. Also Worksheets(x).Cells(z,y).Value can be used to change/retrieve the value in cell z,y in worksheet number x.
4. For Each loop is very similar to For Next loop, but it operates on a collection. For example it operates on a set of worksheets. For every single item of the collection the block of code gets executed.
5. Dim x as Worksheet : Worksheet type object x.
6. For Each x in Worksheets
   1. Do something
   2. Next x : Terminate the For each loop
7. Note that in Excel : Text and Numbers can also be concatenated : For example : “B”&678 : Would result in B678.
8. Breaking a for loop abruptly based upon a condition : Exit For condition
   1. IF Condition Then
      1. Exit For :: Outermost for loop would be exited
   2. ElseIF Condition Then
      1. Do something
   3. End If :: Ends the If statement clause
9. Note the Next x condition for terminating the For Loops (For Each or For Next).
10. Blank space in a cell can be conditioned upon (equals or not equals) using the <> operator.
    1. For example : Cells(4,5).Value<>”” : Checks whether the Cell 4,5 is not blank.
11. Do loop : do while loop :
    1. Should have an initial condition. Dim x as Integer followed by x=1 (acts as Initial condition)
    2. The n should have an update condition (most commonly an increment condition) : x = x + 1
    3. Loop standalone command is used to End the do While loop.
12. Do While x < 10
    1. Body
    2. Update : x = x + 10
    3. Loop : statement to end the do while loop :
13. Date arithmetic : Can just add integers to dates : date\_field + 10 : Adds 10 days to the date\_field as internally in Excel dates as stored as integers.
14. Note the Loop keyword to finish the do while loop.
15. Do Until loop :
    1. Do until some condition is met : Do Until isEmpty(Cell(x,y))
       1. Body of the loop
       2. Update condition
       3. Loop statement to end the do until loop.
16. So Do Until loop : Performs actions until a test is True. Syntax is exactly similar to Do while loop.
17. So Do while performs the set of actions by first checking the condition and executes if the condition is True.
18. And do until keeps performing the actions (body of code) until a specific condition is met.
19. Another one is Do loop until :
    1. This will follow :
       1. Do
       2. Specific block of code
       3. Loop Until Condition
20. So the Do Loop Until would execute atleast once : Unlike the Do Until loop which might not execute Even once : IT checks the condition before every iteration of the loop. Whereas the Do Loop Until first executes the block of code and then before the next iteration checks the condition.
21. Try to use one of the above loops : stick with Do Until Loop. As both the conditions are equivalent.
22. Worksheets.Count : How many worksheets are currently present in my workbook?
23. Select a cell. Then Selection.Offset(4,5) would traverse 4 rows down and 5 columns to the right of the selected cell. Note that the Selection keyword will default to the currently selected object. Negative numbers in the Offset command would move in opp. Direction.
24. <https://stackoverflow.com/questions/36810625/paste-vs-pastespecial> : Paste vs PasteSpecial of the Active Sheet. Paste method is only associated with the ActiveSheet object. If we wanted to paste in a range : we need to use the Paste Special Option. IMPORTANT
25. Now ActiveSheet.Paste can have a destination wherein we can paste the copied data. Also linking is possible if applicable.
26. Selection.Copy will copy the current selection.
27. Cells(x,y).Select will select the cell at location x,y.
28. Worksheets(x).Select or Sheets(x).Select will select the current worksheet.
29. ActiveSheet will return the currently selected sheet.
30. A **Selected cell** is a cell that has been clicked. You can have more than one selected cell. (Usually by holding down CTRL and selecting multiple cells) However the only active cell is the one the user is currently on.
31. Range(“A1”).Select : Will select cell A1. Selection.CurrentRegion : will select the contiguous region associated with current cell.
32. If you don't specify the Destination argument, you must select the destination range before you use this method.
33. This method may modify the sheet selection, depending on the contents of the Clipboard.
34. ActiveSheet.Paste will paste in the currently active sheet. Also if the destination is not provided then the current selection would be used as the destination address.
35. Rows(“3:6”).Delete : will delete the rows specified.
36. End Property : Suppose you have a current selection. The End property enables us to travel to the end of the address which follows the property of the current selection. For example if our current selection contains blanks then the End will help us traverse up or down or right or left till we hit a cell which contains something. Similarly if our current selection cell contains some info : such as number or text then End will help us Navigate to the location where we encounter a first blank.
    1. xlDown, xlUp, ToRight, ToLeft are the arguments associated with the End method : Selection.end(xlDown) and so on.
37. Selection.Address : returns the address of the current selection. Absolute address is returned.
38. So this End property has application in : Selecting the region of text which is at an unfixed number of rows/columns in the sheet.
39. So say we have a task of copying data from multiple sheets (user selects one of those sheets) to a Target sheet. But the data region in the individual sheets although continuous (and whose num\_rows and num\_columns are known) is not present at a fixed start\_location.
    1. Then we can first use End to determine the starting row. Then select the current\_region (knowing that our data region is contiguous), then copying it and pasting it at the destination.
    2. The initial active sheet should determine the sheet from which data needs to be selected for copy-pasting.
40. Columns(“B:E”).Select then Selection.Columns.Autofit : Equivalent to autofitting selected columns B through E.
41. Sub Procedures can be called using the Call command.
42. Okay this is how find based Macro VBA code is being implemented: Record the first instance of the find result and keep finding and keep executing the block of code (for ex. It may change the format of the word found), until we land on the cell which was first found.
43. ActiveCell works on only 1 cell at a time. Selection works on all cells that have been selected. ActiveCell.Address would give you the cell address for the current selection : Like B4, or C4, etc. Important. So, for cells in particular, use ActiveCell, while for a range, go ahead with Selection.
44. To find the exact code of the find operation we can record a Macro and then go to find and then type the word which we need to find. Then we can look into our VBA codebook and see the code generated for that particular module.
45. With Selection.Font =

.Name = “Calibri”

.bold = True   
End

1. So, using With and End we can have the Selction.Font attribute defined and then it’s properties could be changed from the next few lines.
2. So Cells.FindNext would locate a cell and select it, hence it can be accessed via Selection or ActiveCell.
3. Msgbox : Pops a message which gives our End user a particular message.
4. Input Box : Retrieve data from the user and then process the input in order to do something useful. Different Type of MsgBoxes are available in VBA Excel.
5. The message box can have a feature to incorporate clickable buttons vbYes, vbNo, vbCancel, which the user can return.
6. vbYesNoCancel in MsgBox. MsgBox can return the clicked value by the User.
7. So these button clicks can help you get the user input from the Msgbox. Each of the buttons is associated with an Integer.
8. Another way is the InputBox, which can take in a message from the user.
9. We can store the user’s input to a variable and then do something with it. InputBox can return the input types by the user.
10. For example we can use the user’s input to change the name of a sheet.
11. Code continuation character can help us to change the line of the code to the next line. Space+\_ is the CCC.
12. vbCrLf constant to dump the text on the next line with concatenators on either side of this constant.
13. The variables returned by the Msgbox can be directly compared with vbYes and vbNo which are default button outputs linked to the Msgbox (after vbYesNo is enabled in the Msgbox line). These vbYes and vbNo, etc. are constants.
14. vbCrLf hard return.
15. If we have multiple if then then better use Select case.
16. Better to use indents with If then Else statements in Excel VBA.
17. Select case variable\_name, Case Else to encapsulate everything which is not defined above and End Select to close out the Select statement.
18. Dim var1,var2,var3,… as String. Multiple variables in a single line here.
19. Space followed by \_ is the Code continuation character. & vbCrLf & is the Hard return, basically acting as a newline.
20. InputBox can have default text, Title text as well as the Message which is to be types inside the Message Box.
21. Worksheets.Name and Worksheets.Add.
22. Worksheets(1).Select : Selects the very first ordered worksheet in my workbook.
23. Few different naming conventions for specifying a worksheet : Using Numerical index, using Name assigned to the sheet, etc. .Delete method, .Add method, .Name method, etc.
24. Dateserial : Yearvalue; Monthvalue and then numeric date number : spits out the entire date.
25. Format(variable/value,format\_type) : For example : Format(date\_value, “Long Date”).
26. So dateserial is actually a valid date format : it can return a date (which is internally stored as a number in the back-end).
27. While Format can format the date into Text based field.
28. Worksheets(1).Copy After:=Worksheets(2) :: Copies worksheet 1 and then pastes it after worksheet 2. Similarly Worksheets(1).Copy Before:=Worksheets(4) : Copies worksheet 1 and then pastes it before worksheet 4.
29. So DateSerial takes in a Year and a Month and a number date to generate a date formatted value. Format can then extract the Month/Year/Date/Long Date from that date. For example using “mmmm”, Month can be extracted.
30. So in this case the Macro VBA loop was really effective in producing a sheet for each of the months of the year.
31. .Move method available with Worksheets : worksheets(10).Move Before:=Worksheets(1). Places the sheet at the very beginning of the Workbook.
32. Worksheets(2).PrintPreview : In the Immediate window one can see : the print preview so generated. PrintOut is also an option available. If we do not specify a particular worksheet then all the worksheets will be used to generate the Printout/preview.
33. See inputbox syntax : The arguments specifically.
34. End and Offset as discussed earlier.
35. Now creating some UDFs : User defined functions.
36. Add procedure -> Function Procedure.
37. Public function CostforGas() : We can specify arguments inside the parenthesis.
38. MilesofStretch as Double can be printed inside the function argument as well inside the parenthesis.
39. Insert-> Add Procedure-> Function rather than sub procedure.
40. End Function at the End of the function.
41. Our UDFs do not have intellisense embedded as in other default Excel functions. So we can click on the small fx symbol in order to check the arguments of the function which we defined.
42. So we do not return anything from the function. End Function to end the function Block.
43. We can write custom functions for XLOOKUP,UNIQUE, and those things which are unavailable in Excel 2019 but only present in Excel Pro Versions (Office 365).