Excel VBA Course Notes: 1. Macro Basics
(File: VBA01-Macros.xls)

Using the Macro Recorder

Open Excel and the VBE (Visual Basic Editor). Unless it has been changed, the VBE window contains the *Project Explorer* window and the *Properties* window (these can be accessed from the **View** menu).

Project Explorer: Works like a file manager. Helps you navigate around the code in your workbook.

Properties Window: Shows the properties of the currently active object (e.g. *Sheet1*) of the current workbook (e.g. *Book1*).

Exercise 1: Recording a Macro.

This exercise shows what happens when a macro is recorded and demonstrates the difference between recording absolute and relative references.

1. On an empty worksheet in a new workbook, select cell **C10**
2. Start the **Macro Recorder** with option to save macro in *This Workbook*. At this point the VBE creates a new *Modules* folder. It's quite safe to go and look at it - your actions won't be recorded. Click the **[+]** next to the folder and see that the VBE has places a module in the folder and named it *Module1*. Double-click the module icon to open its code window. Switch back to Excel.
3. Make sure that the **Relative Reference** button on the *Stop Recording* toolbar is NOT pressed in.
4. Select cell **B5** and stop the recorder.
5. Switch to the VBE and look at the code:

```
Range("B5").Select
```

6. Now record another macro, exactly the same way, but this time with the **Relative Reference** button pressed in.
7. Switch to the VBE and look at the code:

```
ActiveCell.Offset(-5, -1).Range("A1").Select
```

8. Now record another macro, but instead of selecting cell B5, select a block of cells 3x3 starting at B5 (select cells B5:F7)
9. Switch to the VBE and look at the code:

```
ActiveCell.Offset(-5, -1).Range("A1:B3").Select
```

10. Play back the macros, having first selected a different cell than C10 (for Macro2 and Macro3 the starting cell must be in row 6 or below - see step 11 below)

    Macro1 - always moves the selection to B5
    Macro2 - moves the selection to a cell 5 rows up and 1 column to the left of the selected cell.
    Macro3 - always selects a block of six cells starting 5 rows up and 1 column to the left of the selected cell.

11. Run Macro2 but force an error by selecting a cell in row 5 or above. The macro tries to select a non-existent cell because its code is telling it to select a cell 5 rows above the starting point, and that's off the top of the sheet. Press **Debug** to be taken to the part of the macro that caused the problem.

    NOTE: When the VBE is in Debug mode the line of code that caused the problem is highlighted in yellow. You must "reset" the macro before you can proceed. Click the **Reset** button on the VBE toolbar or go to **Run ➢Reset**. The yellow highlighting disappears and the VBE comes out of Debug mode.

12. It is important to try and anticipate user error like this. The simplest way is to modify the code to simply ignore errors and move on to the next task. Do this by adding the line…

```
On Error Resume Next
```

    … immediately above the first line of the macro (underneath the line `Sub Macro1()`

13. Run *Macro2* as before, starting too high on the sheet. This time the line you typed tells Excel to ignore the line of code that it can't execute. There is no error message and the macro exits having done all it can. Use this method of handling errors with caution. This is a very simple macro. A more complex macro would probably not perform as expected if errors were simply ignored. Also, the user has no idea that something has gone wrong.
14. Modify the code of *Macro2* to include a more sophisticated error handler thus:

```
Sub Macro2()
  On Error GoTo ErrorHandler
  ActiveCell.Offset(-5, -1).Range("A1").Select
  Exit Sub
ErrorHandler:
   MsgBox "You must start below Row 5"
End Sub
```

15. This time the user is presented with a dialog box when something goes wrong. If there is no error the line `Exit Sub` causes the macro to finish after it has done its job - otherwise the user would see the message even if there were no error.

Improving Recorded Macros

The good way to learn the basics of VBA is to record a macro and see how Excel writes its own code. Often, though, recorded macros contain much more code than is necessary. The following exercises demonstrate how you can improve and streamline code that has been produced by a recorded macro.

Exercise 2: Improving on Recorded Macros

This exercise shows that when macros are recorded, often more code is generated than necessary. It demonstrates the use of the With statement to précis the code.

1. Select any cell or block of cells.
2. Start the macro recorder and call the macro FormatCells. The Relative References setting will not be relevant.
3. Go to **Format ➢Cells ➢Font** and choose *Times New Roman* and *Red*.
   Go to **Patterns** and choose *Yellow*.
   Go to **Alignment** and choose *Horizontal, Center*
   Go to **Number** and choose *Currency*.
4. Click **OK** and stop the recorder.
5. Click the **Undo** button (or *Ctrl+Z*) to undo your changes to the worksheet.
6. Select a block of cells and run the *FormatCells* macro. Note that it can not be undone! Type in the cells to check the result of the formatting.
7. Look at the code:

```
Sub FormatSelection()
    Selection.NumberFormat = "$#,##0.00"
  With Selection
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlBottom
        .WrapText = False
        .Orientation = 0
        .ShrinkToFit = False
        .MergeCells = False
  End With
  With Selection.Font
        .Name = "Times New Roman"
        .FontStyle = "Regular"
        .Size = 10
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = 3
  End With
  With Selection.Interior
        .ColorIndex = 6
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
    End With
End Sub
```

8. Note all the extra instructions that have been recorded. Delete lines of code so that only the following remains:

```
Sub FormatSelection()
    Selection.NumberFormat = "$#,##0.00"
  With Selection
        .HorizontalAlignment = xlCenter
  End With
  With Selection.Font
        .Name = "Times New Roman"
        .ColorIndex = 3
  End With
  With Selection.Interior
        .ColorIndex = 6
  End With
End Sub
```

9. Run the macro to test the edited code. It still works as before.

10. Now modify the code even further:

```
Sub FormatSelection()
  With Selection
        .NumberFormat = "$#,##0.00"
        .HorizontalAlignment = xlCenter
        .Font.Name = "TimesNewRoman"
        .Font.ColorIndex = 3
        .Interior.ColorIndex = 6
  End With
End Sub
```

11. Test the macro. Everything still works and the code will run much faster.

12. Try recording the same macro usi ng toolbar buttons instead of going to the dialog box:
Change the Font to *Times New Roman*
Change the Font Colour to *Red*
Change the Fill Color to *Yellow*
Click the *Center* button
Click the *Currency* button

13. Look at the code. You still get lots of stuff that you don't necessarily want. Excel is recording all the *default* settings. Most of these are safe to delete.

14. Experiment with editing directly into the code to change colours, the font, the number format etc.

Exercise 3: Watch a Macro Being Recorded

This exercise shows that you can learn by watching the macro build as it is being recorded. It is also an example of when sometimes the With statement isn't appropriate.

1. Open the file **VBA01.xls**.

   Whilst this worksheet is visually OK and can be understood by the user, the presence empty cells can cause problems. Try filtering the data and see what happens. Go to **Data ➢Filter ➢Autofilter** and filter by Region or Month. It is clear that Excel does not make the same assumptions that the user does. The empty cells need to be filled.

2. Tile the Excel and VBE windows (vertically) so they are side -by-side.

3. Select any cell within the data. If it is an empty cell it must be adjacent to a cell containing data.

4. Start the macro recorder and call the macro *FillEmptyCells*. Set to record **Relative References**.

5. In the VBE window find and double -click the module (Module1) for the current workbook to open the editing pane, then switch off the Project Explorer window and the Properties window (only to make space).

6. Record the new macro as follows :

   Step 1. **Ctrl+*** (to select the current region)
   Step 2. **Edit ➢Go To ➢Special ➢Blanks ➢OK** (to select all the empty cells in the current region)
   Step 3. Type **=[UpArrow]** then press **Ctrl+Enter** (to place your typing into all the selected cells)
   Step 4. **Ctrl+*** (to select the current region again)
   Step 5. **Ctrl+C** (to copy the selection - any method will do)
   Step 6. **Edit ➢Paste Special ➢Values ➢OK** (to paste the data back into the same place but discarding the formulas)

Step 7. **Esc** (to come out of Copy Mode)
Step 8. Stop Recording.

7. Look at the code:

```
Sub FillEmptyCells()
    Selection.CurrentRegion.Select
    Selection.SpecialCells(xlCellTypeBlanks).Select
    Selection.FormulaR1C1 = "=R[-1]C"
    Selection.CurrentRegion.Select
    Selection.Copy
    Selection.PasteSpecial Paste:=xlValues, Operation:=xlNone, SkipBlanks:= _
    False, Transpose:= False
    Application.CutCopyMode = False
End Sub
```

8. Note the use of the space-and-underscore " _" to denote the splitting of a single line of code on to a new line. Without this Excel would treat the code as two separate statements.

9. Because this macro has been recorded with well thought out commands, there is little unnecessary code. In the *Paste Special* everything after the word "xlValues" can be deleted.

10. Try out the macro. Then use the AutoFilter tool and note the difference.