Julian’s Macro Tips

The following tips have been copied from the following website:
http://www.angelfire.com/biz7/julian_s/julian/julians_macros.htm

An additional extensive resource is provided by JoJo Zawawi at:

Note on Errors in Macros
Ever had a macro running perfectly one day and the next day errors keep on popping up even though you never made changes to that macro? This is no fault of yours. Due to the excel VBA design, macro files get badly fragmented due to heavy editing of macros, insertion of modules & userforms. What you need to do is copy your macros else where, delete the macros, save the file without macros. Open the file again and import the macros and save it once more with the macros. You macros will run properly until it gets fragmented again at a later stage.

Auto Run - Making your macros run automatically when opening your workbook. You can either use the Auto Open method or the Workbook Open method. These macros will display the message "Hello" when you open the workbook.

Sub Auto_Open()
    MsgBox "Hello"
End Sub

This code would be located in the module. However if you use the second method, the code must be in the workbook (double click "This Workbook" in the explorer window). Click on the drop down list (that says General) and select Workbook. Click on the drop down list (that says declarations) and select Open.

Private Sub Workbook_Open()
    MsgBox "Hello"
End Sub

Counting Rows (Columns and Sheets) - When you have selected a range, it is sometimes useful to know how many rows or columns you have selected as this information can be used in your macros (for e.g. when you have reached the end, you will know it is time to stop the macros. This macro will do the trick.

Sub Count()
    mycount = Selection.Rows.Count
    MsgBox mycount
End Sub

To modify this macro to count the number of sheets instead, use mycount = Application.Sheets.Count.
**Active Cell** - Sometimes you need to move your cursor around your worksheet to reposition it before running the next step of a macro. The movement here uses the row, column position method.

```vba
Sub Down()
    ActiveCell.Offset(1, 0).Select
End Sub
Sub up()
    ActiveCell.Offset(-1, 0).Select
End Sub
Sub Right()
    ActiveCell.Offset(0, 1).Select
End Sub
Sub Left()
    ActiveCell.Offset(0, -1).Select
End Sub
```

**Copying A Range** - Copy data from a specific range can be done with this macro. Here data is copied from the current sheet to the active cell.

```vba
Sub CopyRange()
    Range("A1:A3").Copy Destination:=ActiveCell
End Sub
```

To copy from a range in another sheet (e.g., Sheet3) to the active cell you need to change the code to;

```vba
Sheets("sheet3").Range("A1:A3").Copy Destination:=ActiveCell
```

**Carriage Return** - Sometimes you may want to put a line of text on the next row and not let it continue on the first row. See this example in a message box.

```vba
Sub TwoLines()
    MsgBox "Line 1" & vbCrLf & "Line 2"
End Sub
```

**Change Capitalization of Text** – This example uses lowercase capitalization.

```vba
Sub LowerCase()
    Dim cell As Range
    For Each cell In Selection.Cells
        If cell.HasFormula = False Then
            cell = LCase(cell)
        End If
    Next
End Sub
```

Instead of `LCase`, use `UCase` (uppercase) or `Application.Proper` (title capitalization).
Counter - To use a counter in your macro, just assign any cell to retain the value. In this example the cell A1 is chosen. Each time the macro is run, it adds the value 1 to the cell A1.

```vba
Sub Count()
    mycount = Range("a1") + 1
    Range("a1") = mycount
End Sub
```

Current Cell Content - Sometimes we need to know what the cell contains i.e. dates, text or formulas before taking a course of action. In this example a message box is displayed. Replace this with a macro should you require another course of action.

```vba
Sub ContentChk()
    If Application.IsText(ActiveCell) = True Then
        MsgBox "Text" 'replace this line with your macro
    Else
        If ActiveCell = "" Then
            MsgBox "Blank cell" (replace this line with your macro)
        Else
            If ActiveCell.HasFormula Then
                MsgBox "formula" (replace this line with your macro)
            Else
                MsgBox "date" (replace this line with your macro)
            End If
        End If
    End If
End Sub
```

Current Cell Position

```vba
Sub MyPosition()
    myRow = ActiveCell.Row
    myCol = ActiveCell.Column
    MsgBox myRow & "," & myCol
End Sub
```

Deleting Empty Rows - To delete empty rows in a selected range we can use the following macro. The macro here uses the For Next Loop. First the macro counts the rows in a selected range to determine the when the macro should stop. The For Next statement acts as a counter.

```vba
Sub DeleteEmptyRows() 'Your code here
End Sub
```
Sub DelEmptyRow()
Rng = Selection.Rows.Count
ActiveCell.Offset(0, 0).Select
Application.ScreenUpdating = False
For i = 1 To Rng
If ActiveCell.Value = "" Then    'You can replace "" with 0 to delete rows with 'the value zero
Selection.EntireRow.Delete
Else
ActiveCell.Offset(1, 0).Select
End If
Next i
Application.ScreenUpdating = True
End Sub

The statement "Application.ScreenUpdating = False" prevents the screen from updating to ensure the macro runs faster and the screen will not flicker. Don't forget to set it back to "True".

Duplicates (Highlight duplicates in Bold Red) - There are times you need to highlight duplicate data in your worksheet. This macro does the trick.

Sub DupsRed()
Application.ScreenUpdating = False
Rng = Selection.Rows.Count
For i = Rng To 1 Step -1
myCheck = ActiveCell
ActiveCell.Offset(1, 0).Select
For j = 1 To i
If ActiveCell = myCheck Then
Selection.Font.Bold = True
Selection.Font.ColorIndex = 3
End If
ActiveCell.Offset(1, 0).Select
Next j
ActiveCell.Offset(-i, 0).Select
Next i
Application.ScreenUpdating = True
End Sub

Emailing Workbook - To email your current workbook the following code.

Sub Email()
ActiveWorkbook.SendMail recipients:=julsn@yahoo.com
End Sub
**Error Trapping** - This first statement allows the macro to continue the next line of code upon hitting an error.

```
On Error Resume Next
```

The following statement will run an alternative code if there is an error.

```
On Error Goto ErrorTrap1
... more lines of code
ErrorTrap1:... more code (what to do if there is an error)
```

**Excel Functions** - Using Excel functions in VBA is almost the same as using them in a spreadsheet. For example to round an amount to 2 decimal places in a spreadsheet would be;

```
=round(1.2345,2)
```

In VBA you would need to use the term Application followed by the function ie;

```
ActiveCell = Application.round(ActiveCell, 2)
```

**Flickering Screen** - Sometimes when you run a macro, the screen flickers a lot due to the screen updating itself. This slows the macro done especially when the macro has a lot of work to do. You need to include the statement as shown below.

```
Application.ScreenUpdating = False
```

You need to set the screen updating back to true at the end of the macro.

**Functions** - Creating function is useful as complicated formulas can be made easier in code than in a spread sheet. Formulas can be protected so that users cannot see or modify them. The example I use will calculate tax using the Select Case Statement. Here’s the scenario: First $2500 is tax free; Next $2500 is taxable at 5%; Anything above $5000 is taxable at 10%.

In cell A1 type Income and in cell B1 type in your income in numbers say $20000. In cell A2 type Tax payable and in cell B2 type =tax(B1). Put the following code in a module. The tax payable here would be $1625.

```
Public Function tax (income As Single)
Select Case income
Case Is <= 2500
    tax = 0
Case Is <= 5000
    tax = (5000 - 2500) * 0.05            (in this case 125)
Case Else
    tax = (income - 5000) * 0.1 + 125
End Select
End Function
```
**Goto (a range)** - To specify a macro to go to a specific range you can use the Goto method. Here I have already named a range in my worksheet called "Sales". You may also use an alternative method ie the Range select method. Naming a range in excel is recommended rather than specifying an absolute cell reference.

```vba
Sub GoHere()
    Application.Goto Reference:="Sales" OR Range("Sales").Select
End Sub
```

**Going to the 1st Sheet** - You can select the first sheet of the workbook without knowing the name of the sheet by referring to it by the index.

```vba
Sub FirstSheet()
    Sheets(1).Select
End Sub
```

**Hiding Sheets** - To hide your worksheet from users you can use the following code.

```vba
Sub HideSheet()
    Sheet1.Visible = xlSheetVeryHidden
End Sub
```

If you hide your sheets this way, users will not be able to unhide them using the menus. Only using VB codes will be able to display the sheets again.

**Hiding Excel** - You can hide the Excel application with this macro. This disables the user from using the excel menus. Don't forget to set it back to visible.

```vba
Sub HideExcel()
    Application.Visible = False
End Sub
```

**Input Box** - When you need to get input from users, you can use input boxes. This macro will ask for the user's name and will display a message "Hello" plus the user's name.

```vba
Sub GetInput()
    Dim MyInput
    MyInput = InputBox("Enter your name")
    MsgBox ("Hello ") & MyInput
End Sub
```

**Joining Text Together** - There are times where we import text file into Excel an we get text that are separated. I received an email asking how put these text together. Select across your cells first and run this macro.
Sub JoinText()
myCol = Selection.Columns.Count
For i = 1 To myCol
    ActiveCell = ActiveCell.Offset(0, 0) & ActiveCell.Offset(0, i)
    ActiveCell.Offset(0, i) = ""
Next i
End Sub

**Message Box** - When you need to communicate with users, you can use message boxes. This macro will display a message "This macro is created by Julian". The Message Box appearance can be customized to show whether it is Information, Critical Messages. Here the icon in the message box would be different. The buttons can also be customized to show extra Yes, No, Ok buttons. (Refer to vbYesNo macro). This macro will show you 3 different styles.

    Sub MyMessage()
        MsgBox "This macro is created by Julian"
        MsgBox "The icon is different", vbInformation
        MsgBox "The top title is different", vbExclamation, "Julian's Tips"
    End Sub

**Protecting / Unprotecting a sheet** - The macros below will protect/unprotect the current worksheet with a specific password.

    Sub ProtectSheet()
        Dim Password 'This line of code is optional
        Password = "1234"
        ActiveSheet.Protect Password, True, True, True
    End Sub
    Sub UnProtectSheet()
        Password = "1234"
        ActiveSheet.Unprotect Password
    End Sub

To protect all the sheets this macro uses all the methods contained in this page (see counting sheets). An If, Then statement can be used that tests for a condition and if the condition is TRUE it will end the macro otherwise it will select the next sheet. You will also notice the For, Next statement is also used.

**Protecting your VB code** - To protect your VB code from being seen by others, all you need to do is go to the project explorer, point at your project and right click on it. Select VBA project properties, click on the protection tab and check the Lock project for viewing and key your password. That's it.
Running A Sub Routine - To run another macro from within a macro you need to use the Call statement.

    Sub Macro1()
    MsgBox("This is Macro1")
    Call Macro2    'This calls for Macro2 to run
    End Sub

Saving a file - There are times you may want a macro to save a file automatically after running a macro. The second macro will save the file with a name called "MyFile". You may specify the path if you need to.

    Sub Save()
    ActiveWorkbook.Save
    End Sub
    Sub SaveName()
    ActiveWorkbook.SaveAs Filename:="C:\MyFile.xls"
    End Sub

Top of the screen - To make the activecell be at the top of the screen & to the left on the screen try this.

    Sub TopLeft()
    ActiveCell.Select
    With ActiveWindow
       .ScrollColumn = ActiveCell.Column
       .ScrollRow = ActiveCell.Row
    End With
    End Sub

vbYesNo - There are times you may want users to click Yes or No. Just insert this line of code. Here the Select Case statement is used.

    YesNo = MsgBox("This macro will ... Do you want to continue?", vbYesNo + vbCritical, "Caution")Select Case YesNo
    Case vbYes
       'Insert your code here if Yes is clicked
    Case vbNo
       'Insert your code here if No is clicked