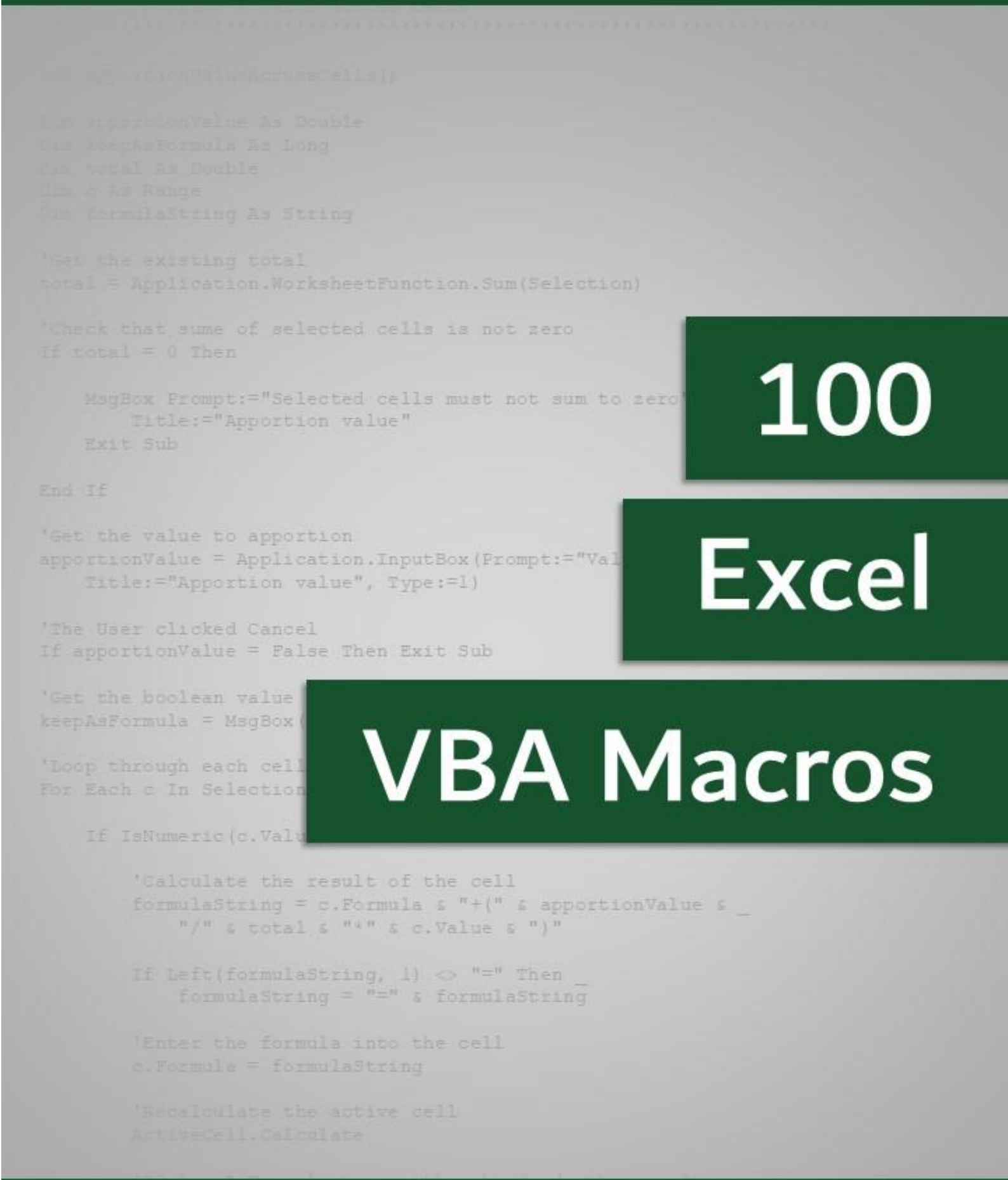
100 Excel VBA Macros



EXCEL OFF THE GRID

100 Excel VBA Macros

https://excelofffthegrid.com

Limit of liability/disclaimer of warranty

Because of the possibility of human or mechanical error, the copyright holder does not guarantee the accuracy, adequacy or completeness of any information. The copyright holder accepts no liability for any inaccuracy, error of omission, or for the results obtained, regardless of cause from the use of any information.

The copyright holder does not warrant or guarantee that the information contained in the work will meet your requirement or its fitness for a particular purpose.

Contents

PART ONE:

How to use VBA macros

How to use this book

The macros and techniques contained in this book are illustrations of what can be achieved with VBA. In most circumstances, the code will need to be customized to your specific needs. As the macro segments are illustrations, they are not all useful in their own right.

I have tried to write the code so it can be (a) understood by those with limited experience of VBA and (b) easily customized to meet user requirements. This means that each macro is not necessarily written in the most efficient manner and excludes extensive error checking.

Support files

All the macros are available in the support file, which was distributed in the same zip file as this Ebook.

Found an error?

Whilst I try to create safe and reliable code segments, I can (and often do) make mistakes. Please backup copies of your files before using any code in this book. Backing up ensures that if anything goes seriously wrong, you can revert to a previous working version.

If you do find errors, please let me know. Go to https://excelofftthegrid.com/contact/ to contact me and provide as much information about the error as possible. Hopefully, over time, with your feedback, I can eradicate all the errors and turn this into an even better resource.

What is VBA?

Visual Basic for Applications (VBA) is the programming language created by Microsoft to control parts of their applications. Most things which you can do with the mouse or keyboard in the Microsoft Office suite, you can also do using VBA. For example, in Excel, you can create a chart; you can also create a chart using VBA, it is just another method of achieving the same thing.

Advantages of using VBA

Since VBA code can do the same things as we could with the mouse or keyboard, why bother to use VBA at all?

Saves time:

VBA code will operate at the speed your computer will allow, which is still significantly faster than you can operate. For example, if you have to open 10 workbooks, print the documents, then close the workbook, it might take you 2 minutes with a mouse and keyboard, but with VBA it could take seconds.

Reduces errors:

Do you ever click the wrong icons or type the wrong words? Me too, but VBA doesn't. It will do the same task over and over again, without making any errors. Don't get me wrong, you still have to program the VBA code correctly. If you tell it to do the wrong things 10 times, then it will. But if we can get it right, then it can remove the errors created by human interaction.

Completes repetitive actions without complaining:

Have you ever had to carry out the same action many times? Maybe creating 100 charts, or printing 100 documents, or changing the heading on 100 spreadsheets. That's not fun, nobody wants to do that. But VBA is more than happy to do it for you. It can do the same thing in a repetitive way (without complaining). In fact, repetitive tasks is one of the things VBA does best.

Integration with other applications:

You can use VBA in Word, Access, Excel, Outlook and many other programs, including Windows itself. But it doesn't end there, you can use VBA in Excel to control Word and PowerPoint, without even needing to open those applications.

What is programming?

Programming is simply writing words in a way which a computer can understand. However, computers are not particularly flexible, so we have to be very specific about what we want the computer to do, and how we tell it to do it. The skill of programming is learning how to convey the request to the computer as clearly, as simply and as efficiently as possible.

What is the difference between a Macro and VBA?

This is a common question which can be confusing. Put simply, VBA is the language used to write a macro – just in the same way as a paragraph might be written using the English language.

The terms 'macro' and 'VBA' are often used interchangeably.

The golden rule of learning VBA

If you are still learning to write VBA, there is one thing which will help you. While it may be common practice, to copy and paste code, it will not help you to learn VBA quickly. Here is the one rule I am going to ask you to stick to... **type out the code yourself.**

Why am I asking you to do this? Because it will help you learn the VBA language much faster.

Let's get started

Now you know what VBA is, why you should use it, and the golden rule, so there is only one thing left to do… let's get started!

Setting up Excel

Before you can get stuck in with using the code in this book, you must first have Excel set up correctly. This involves:

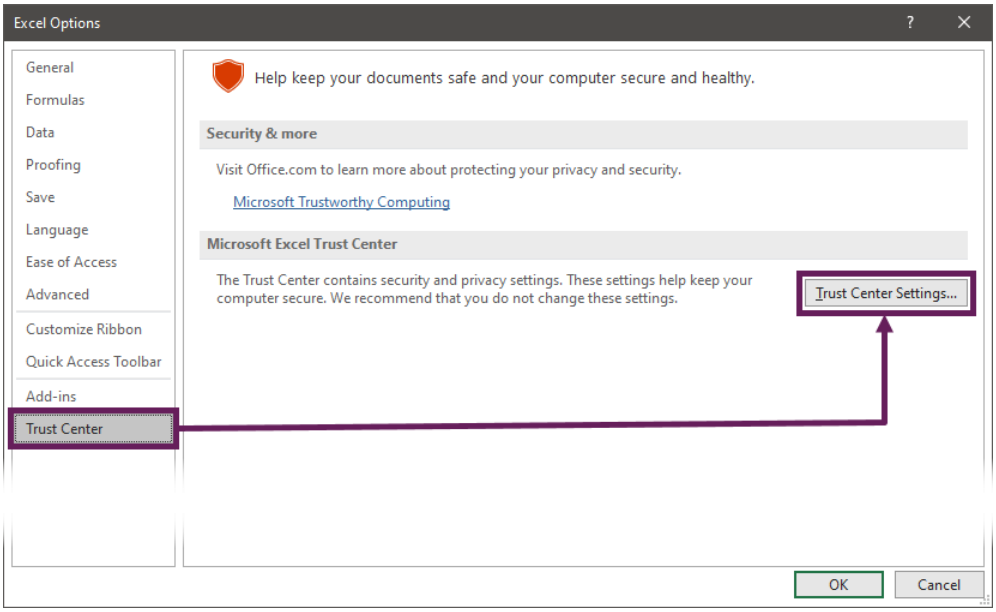
1) Ensuring the correct macro security settings have been applied
2) Enabling the Developer ribbon.

Macro security settings

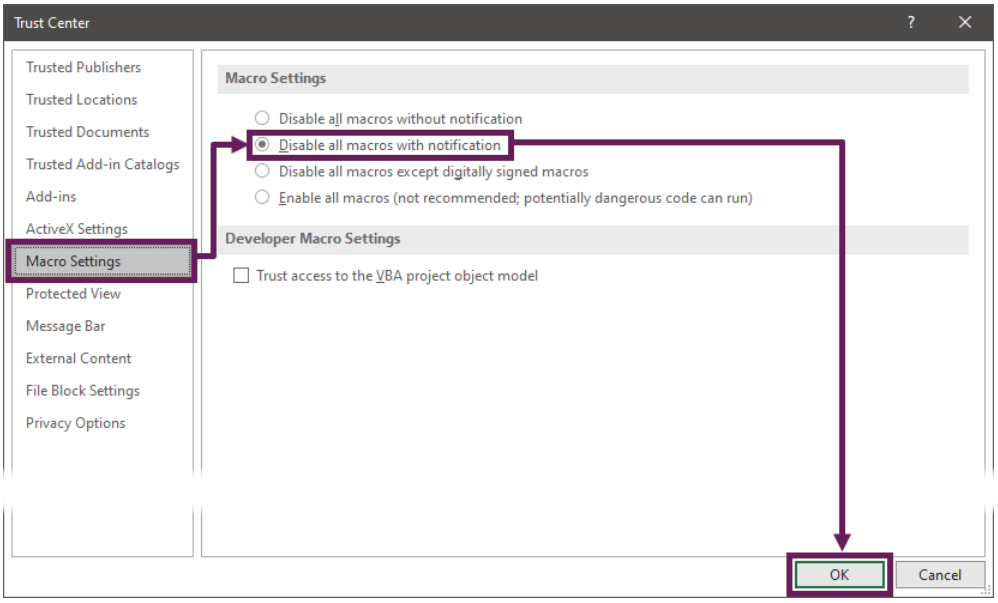
Macros can be used for malicious purposes, such as installing a virus, recording key-strokes, etc. This can be blocked with the security settings. However, if the settings are set too high, you cannot run any macros, or too low, you will not be protected. Neither of these is a good option.

Let's apply suitable settings which will give you the power to decide when to allow macros or not.

1. In Excel, click **File > Options**
2. In the Excel Options dialog box, click **Trust Centre > Trust Centre Settings…**



3. In the Trust Centre dialog box, click **Macro Settings > Disable all macros with notification**.



4. Click **OK** to close the Trust Centre, then **OK** again to close the Excel Options.

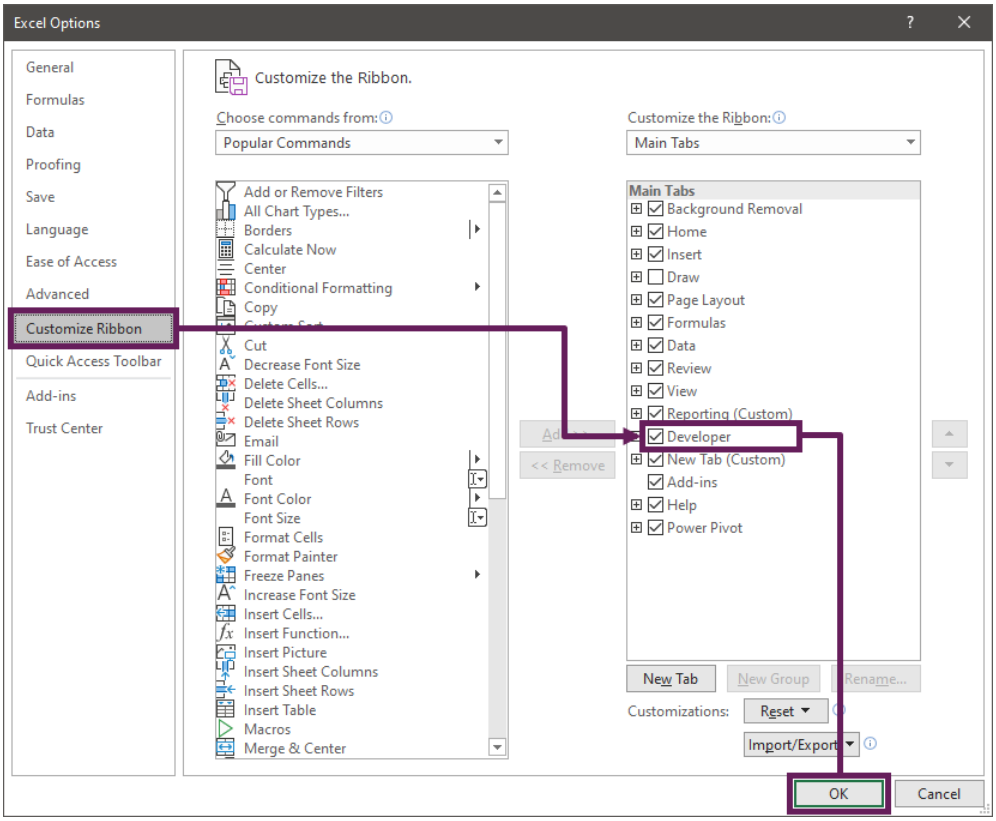
Workbooks containing macros will now be automatically disabled until you click the Enable Content button at the top of the screen.

Enable the Developer ribbon

The Developer ribbon is the place where all the VBA tools are kept. It is unlikely that this is already enabled, unless you or your IT department have already done so.

Look at the top of your Excel Window if you see the word 'Developer' in the menu options, then you are ready to go. You can skip straight ahead to the next part. However, if the 'Developer' ribbon is not there, just follow these instructions.

1. In Excel, click **File > Options**
2. In the Excel Options dialog box, click **Customize Ribbon**
3. Ensure the **Developer** option is checked

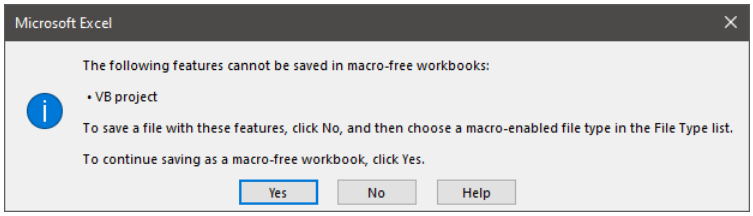


4. Click **OK** to close the Excel Options

The Developer ribbon should now be visible at the top of the Excel window.

File format for macro enabled files

To save a workbook containing a macro, the standard .xlsx format will not work.



Generally, the .xlsm (Excel Macro-Enabled Workbook) file format should be used for workbooks containing macros. However .xlam (Excel Add-in), .xlsb (Excel Binary Workbook) and .xltx (Excel Macro-Enabled Template) are scenario specific formats which can also contain macros.

The legacy .xls and .xla file formats can both contain macros. They were superseded in 2007, and should now be avoided.

Basic rule is… **if you don't know, go for .xlsm.**

Personal macro workbook

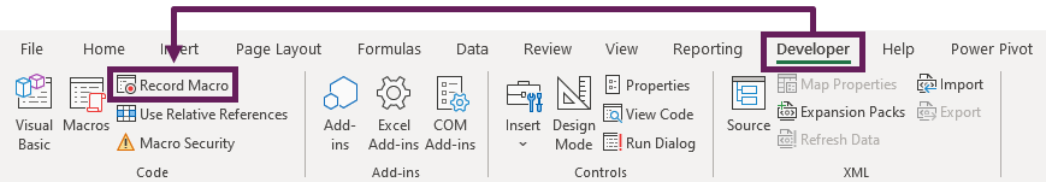
If we want macros to be reusable for many workbooks, often the best place to save them is in the personal macro workbook.

A personal macro workbook is a hidden file which opens whenever the Excel application opens.

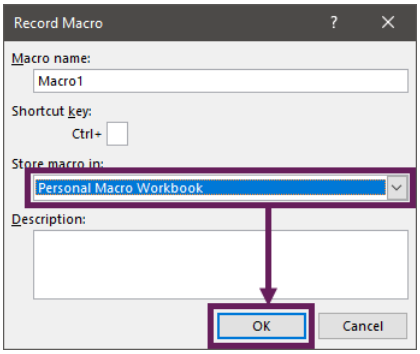
How to create a personal macro workbook?

A personal macro workbook does not exist by default; we have to create it. There are many ways to do this, but the easiest is to let Excel do it for us.

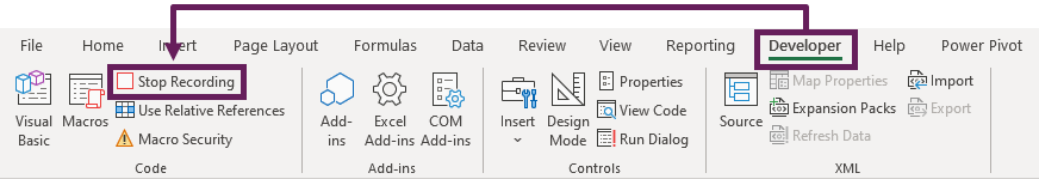
1. In the ribbon, click **Developer > Record Macro**.



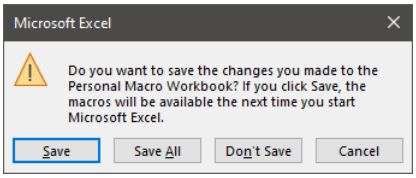
2. In the Record Macro dialog box, select **Personal Macro Workbook** from the drop-down list.



3. Click **OK**.
4. Do anything in Excel, such as typing your name into cell A1.
5. Click **Developer > Stop Recording**



6. Close all the open workbooks in Excel, this will force the personal macro workbook to be saved. A warning message will appear, click **Save**.



In the next part, we will learn how to use the Visual Basic Editor, which gives us access to the personal macro workbook.

Using the Visual Basic Editor

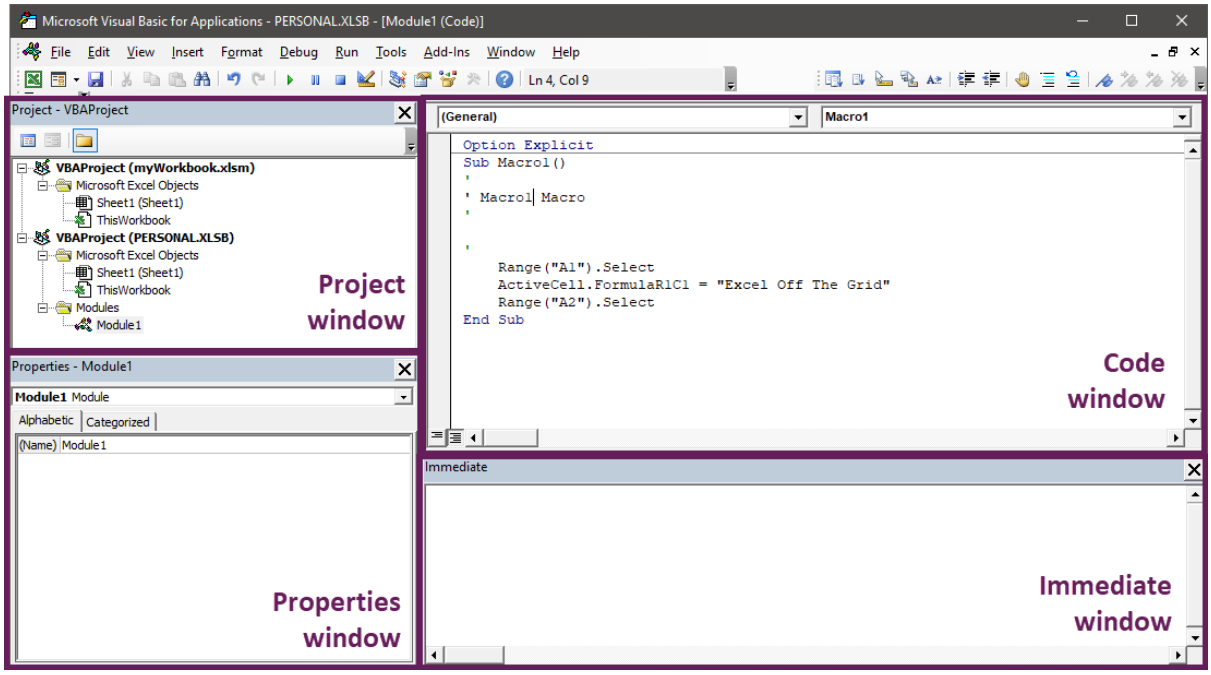
The Visual Basic Editor (or VBE as it can be known) is the place where we enter or edit VBA code. The Visual Basic Editor is found within the Developer Ribbon

In Excel, click **Developer > Visual Basic** to open the VBE.

Alternatively, you could use the keyboard; press ALT+F11 (the + indicates that you should hold down the ALT key, press F11, then release the ALT key), which toggles between the Excel window and the VBE.

The Visual Basic Editor Window

The Visual Basic Editor contains four main sections.



Within the top left of the VBE, we will see a list of items which can contain VBA code (known as the project window)

Double-clicking any sheet name, workbook or module, will open the code window associated with that item. VBA code is entered into the code window.

Unless you have specific reasons, the best option is to enter the macro into a module. To create a module, click **Insert > Module** within the VBE.

Running a macro

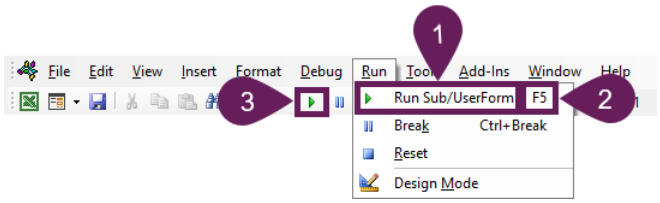
There are many ways to run VBA code. This section is not exhaustive, but is intended to provide an overview of the most common methods.

Running a macro from within Visual Basic Editor

When testing VBA code, it is common to execute that code from the VBE.

Click anywhere within the code, between the Sub and End Sub lines, choose one of the following options:

1. Click **Run > Run Sub/UserForm** from the menu at the top of the VBE
2. Using the keyboard, you can press ALT+F5
3. Click the play button at the top of the VBE



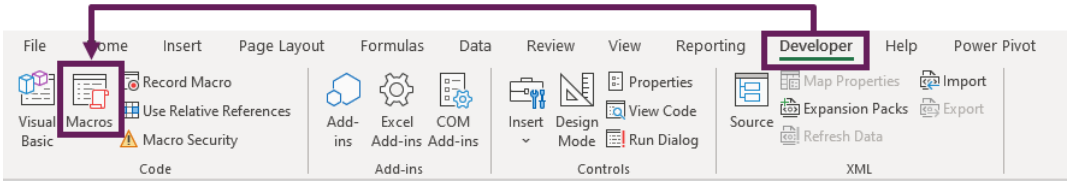
The code you entered will be executed.

Running a macro from within Excel

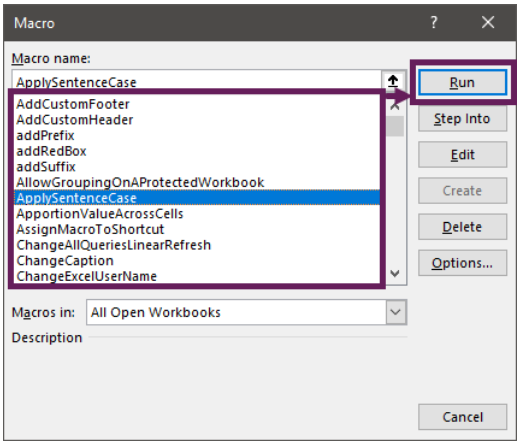
Once the code has been tested and in working order, it is common to execute it directly within Excel. There are lots of options for this too (including events, or user defined functions), however the three most common methods I will show you are:

Run from the Macro window

1. Click **View > Macros** or **Developer > Macros**



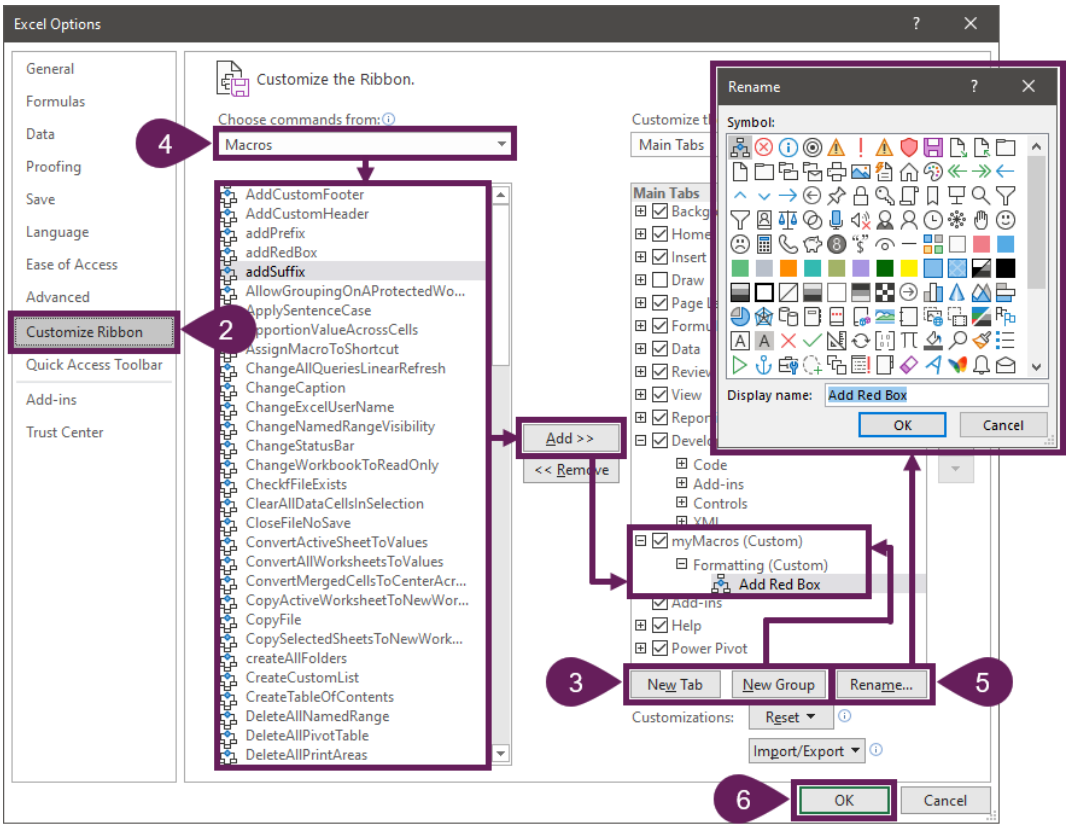
2. Select the macro from the list and click **Run**.



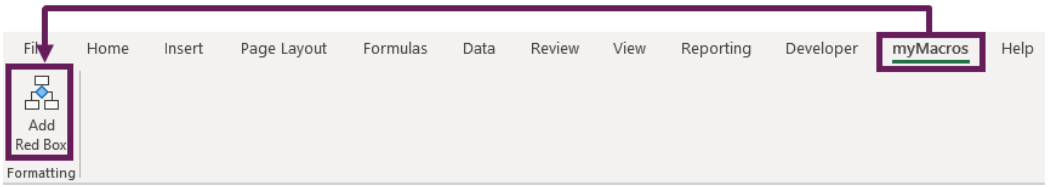
Create a custom ribbon

Having macros always available in the ribbon is a great time saver. Therefore, learning how to customize the ribbon is useful.

1. In Excel, click **File > Options**
2. In the Excel Options dialog box, click **Customize Ribbon**
3. Click **New Tab** to create a new ribbon tab, then click **New Group** to create a section within the new tab.
4. In the **Choose commands from** drop-down, select Macros. Select your macro and click **Add >>** to move the macro it into your new group.
5. Use the **Rename…** button to give the tab, group or macro a more useful name.



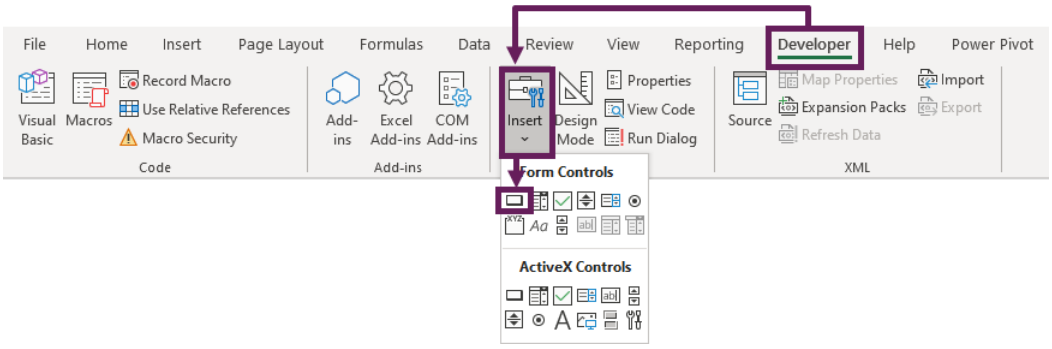
6. Click **OK** to close the window.
7. The new ribbon menu will appear containing your macro. Click the button to run the macro.



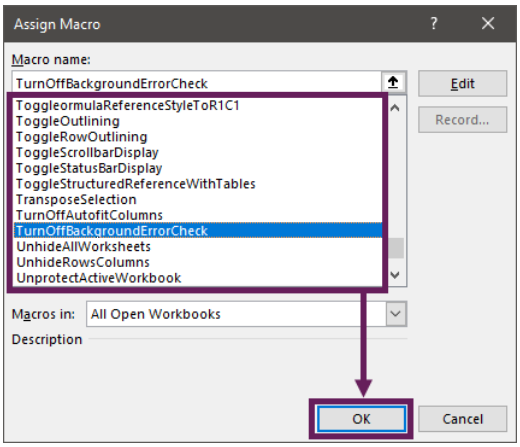
Create a button/shape on a worksheet

Macros can be executed using buttons or shapes on the worksheet.

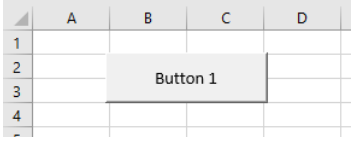
1. To create a button, click **Developer > Insert > Form Control > Button**



2. Draw a shape on the worksheet to show the location and size of the button
3. The Assign Macro dialog will appear, select the macro and click **OK**.

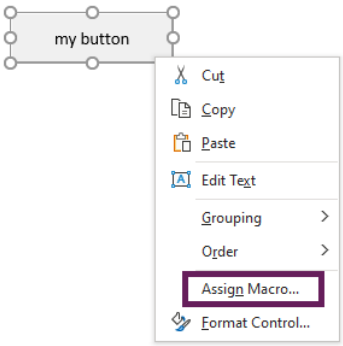


4. The button will appear. Clicking the button will run the macro



5. **Right-click** on the button to **change the description**

To assign a different macro, **right-click** on the button and select **Assign Macro…** from the menu.



Alternatively, a macro can be assigned to a shape. After creating a shape, **right-click** on it and select **Assign Macro…** from the menu, then follow the same process as for a button.

PART TWO:
100 Excel VBA Macros

General Macros

001 - Macro to call at the start of each macro

What does it do?

Excel contains various settings that can slow down your macros. We want to turn off those settings before running a macro (which is example code 001), then restore them again after the macro finishes (which example code 002).

VBA Code

```
'Place at the start of the macro
Public calcMode As Long
Public pageBreakStatus As Boolean

Sub SettingsStartOfMacro()

With Application

    calcMode = .Calculation
    pageBreakStatus = ActiveSheet.DisplayPageBreaks

    'Turn calculation mode to manual
    .Calculation = xlCalculationManual

    'Turn off screen updating (i.e. no annoying screen flash)
    .ScreenUpdating = False

    'Alert windows will not be displayed
    .DisplayAlerts = False

    'Turn off page breaks on active sheet
    .DisplayAlerts = False

End With

End Sub
```

Notes:

To call this macro, include the following code at the start of each of your macros after the Sub statement.

```
Call SettingsStartOfMacro
```

The VBA code above includes Public variables, these must be included at the top of the code window (directly after the Option Explicit statement), and before any Subs are created.

002 - Macro to call at the end of each macro

What does it do?

Restores all the settings which were changed in the macro above.

VBA Code

```
Sub SettingsEndOfMacro()

With Application

    'Return calculation to automatic
    .Calculation = calcMode

    'Turn screen updating back on
    .ScreenUpdating = True

    'Enable alerts to be shown
    .DisplayAlerts = True

    'Reset page breaks on active sheet
    .DisplayAlerts = pageBreakStatus

End With

End Sub
```

Notes:

To call this macro, include the following code at the end of your macro, directly before the End Sub statement.

```
Call SettingsEndOfMacro
```

Hiding and displaying worksheets

003 - Hide all selected sheets

What does it do?

Hides all the selected sheets.

VBA Code

```
Sub HideAllSelectedSheets()

'Create variable to hold worksheets
Dim ws As Worksheet

'Ignore error if trying to hide the last worksheet
On Error Resume Next

'Loop through each worksheet in the active workbook
For Each ws In ActiveWindow.SelectedSheets

    'Hide each sheet
    ws.Visible = xlSheetHidden

Next ws

'Allow errors to appear
On Error GoTo 0

End Sub
```

Notes:

Excel requires at least one active worksheet. If all the visible sheets are selected, to avoid an error, the VBA code will not hide the last sheet.

004 – Very hide all selected sheets

What does it do?

Makes the selected sheets very hidden.

Worksheets which are very hidden do not appear in the list of hidden worksheets. They can only be seen and made visible using VBA.

VBA Code

```
Sub VeryHideAllSelectedSheets()

'Create variable to hold worksheets
Dim ws As Worksheet

'Ignore error if trying to hide the last worksheet
On Error Resume Next

'Loop through each worksheet in the active workbook
For Each ws In ActiveWindow.SelectedSheets

    'Very hide each sheet
    ws.Visible = xlSheetVeryHidden

Next ws

'Allow errors to appear
On Error GoTo 0

End Sub
```

Notes:

Excel requires at least one active sheet. If all the visible sheets are selected, to avoid an error, the VBA code will not hide the last sheet.

005 – Unhide all sheets

What does it do?

Makes all worksheets visible.

VBA Code

```
Sub UnhideAllWorksheets()

'Create variable to hold worksheets
Dim ws As Worksheet

'Loop through each worksheet in the active workbook
For Each ws In ActiveWorkbook.Worksheets

    'Unhide each sheet
    ws.Visible = xlSheetVisible

Next ws

End Sub
```

006 – Delete all hidden worksheets

What does it do?

Deletes all hidden worksheets.

VBA Code

```
Sub DeleteHiddenWorksheets()
'If worksheet is very hidden it will not be delete

'Create variable to hold worksheets
Dim ws As Worksheet

'Prevent the warning message appearing
Application.DisplayAlerts = False

'Loop through each worksheet in the active workbook
For Each ws In ActiveWorkbook.Worksheets
```

```
    'Check if the sheet is hidden
    If ws.Visible = xlSheetHidden Then

        'Delete workbook
        ws.Delete

    End If

Next ws

'Restore warning messages
Application.DisplayAlerts = True

End Sub
```

Notes:

Take care that there are no interrelationships between the remaining sheets and those being deleted. Deleting sheets could cause your formulas or other functionality to break.

007 – Hide all worksheets except active sheet

What does it do?

Hides all the worksheets, except the active sheet.

VBA Code

```
Sub HideAllWorksheetsExceptActive()

'Create a variable to hold worksheets
Dim ws As Worksheet

'Loop through each worksheet in the active workbook
For Each ws In ActiveWorkbook.Worksheets

    'If the ws in the loop is not the active sheet
    If ActiveSheet.Name <> ws.Name Then

        'Make each worksheet hidden
        ws.Visible = xlSheetHidden
```

```
    End If

Next ws

End Sub
```

008 – Sort worksheets alphabetically

What does it do?

Sorts all the worksheets in alphabetical order (Excel doesn't provide any features to do this automatically).

VBA Code

```
Sub SortSheetsTabName()

Dim wsCount As Integer
Dim i As Integer
Dim j As Integer

wsCount = ActiveWorkbook.Sheets.Count

For i = 1 To wsCount - 1

    For j = i + 1 To wsCount

        If Sheets(j).Name < Sheets(i).Name Then
            Sheets(j).Move before:=Sheets(i)
        End If

    Next j

Next i

End Sub
```

Applying protection

009 – Protect all selected worksheets

What does it do?

Protects all the selected worksheets with a password determined by the user.

VBA Code

```
Sub ProtectSelectedWorksheets()

Dim ws As Worksheet
Dim sheetArray As Variant
Dim myPassword As Variant

'Set the password
myPassword = Application.InputBox(prompt:="Enter password", _
    Title:="Password", Type:=2)

'The User clicked Cancel
If myPassword = False Then Exit Sub

'Capture the selected sheets
Set sheetArray = ActiveWindow.SelectedSheets

'Loop through each worksheet in the active workbook
For Each ws In sheetArray

    On Error Resume Next

    'Select the worksheet
    ws.Select

    'Protect each worksheet
    ws.Protect Password:=myPassword

    On Error GoTo 0

Next ws
```

```
sheetArray.Select

End Sub
```

010 – Unprotect all worksheets

What does it do?

Unprotects all worksheets with a password determined by the user.

VBA Code

```
Sub UnprotectAllWorksheets()

'Create a variable to hold worksheets
Dim ws As Worksheet

'Create a variable to hold the password
Dim myPassword As Variant

'Set the password
myPassword = Application.InputBox(prompt:="Enter password", _
    Title:="Password", Type:=2)

'The User clicked Cancel
If myPassword = False Then Exit Sub

'Loop through each worksheet in the active workbook
For Each ws In ActiveWorkbook.Worksheets

    'Unprotect each worksheet
    ws.Unprotect Password:=myPassword

Next ws

End Sub
```

011 – Protect active workbook

What does it do?

Protects the workbook structure with a password determined by the user.

VBA Code

```
Sub ProtectActiveWorkbook()

'Create a variable to hold the password
Dim myPassword As Variant

'Set the password
myPassword = Application.InputBox(prompt:="Enter password", _
    Title:="Password", Type:=2)

'The User clicked Cancel
If myPassword = False Then Exit Sub

'Protect the active workbook
ActiveWorkbook.Protect Password:=myPassword

End Sub
```

012 – Unprotect active workbook

What does it do?

Unprotects the workbook structure with a standard password of *myPassword.*

VBA Code

```
Sub UnprotectActiveWorkbook()

'Create a variable to hold the password
Dim myPassword As String

'Input the password
myPassword = "myPassword"
```

```
'Unprotect the active workbook
ActiveWorkbook.Unprotect Password:=myPassword

End Sub
```

013 – Lock cells containing formulas

What does it do?

Password protects a single worksheet with cells containing formulas locked, all other cells are unlocked.

VBA Code

```
Sub LockOnlyCellsWithFormulas()

'Create a variable to hold the password
Dim myPassword As Variant

'If more than one worksheet selected exit the macro
If ActiveWindow.SelectedSheets.Count > 1 Then

    'Display error message and exit macro
    MsgBox "Select one worksheet and try again"
    Exit Sub

End If

'Set the password
myPassword = Application.InputBox(prompt:="Enter password", _
    Title:="Password", Type:=2)

'The User clicked Cancel
If myPassword = False Then Exit Sub

'All the following to apply to active sheet
With ActiveSheet

    'Ignore errors caused by incorrect passwords
    On Error Resume Next
```

```
    'Unprotect the active sheet
    .Unprotect Password:=myPassword

    'If error occured then exit macro
    If Err.Number <> 0 Then

        'Display message then exit
        MsgBox "Incorrect password"
        Exit Sub

    End If

    'Turn error checking back on
    On Error GoTo 0

    'Remove lock setting from all cells
    .Cells.Locked = False

    'Add lock setting to all cells
    .Cells.SpecialCells(xlCellTypeFormulas).Locked = True

    'Protect the active sheet
    .Protect Password:=myPassword

End With

End Sub
```

014 – Hide formulas when protected

What does it do?

When the active sheet is protected, formulas will not be visible in the formula bar. Uses a predefined password of *mypassword.*

VBA Code

```
Sub HideFormulasWhenProtected()

'Create a variable to hold the password
Dim myPassword As String

'Set the password
myPassword = "myPassword"

'All the following to apply to active sheet
With ActiveSheet

   'Unprotect the active sheet
   .Unprotect Password:=myPassword

    'Hide formulas in all cells
   .Cells.FormulaHidden = True

   'Protect the active sheet
   .Protect Password:=myPassword

End With

End Sub
```

Turning settings on & off

015 – Toggle gridlines on selected sheets

What does it do?

On the selected sheets, the gridlines are toggled on/off. The setting applied is based on the active worksheet.

VBA Code

```
Sub ToggleGridlines()

'Create a variable to hold worksheets
Dim ws As Worksheet

'Create a variable to hold the active worksheet
Dim currentWs As Worksheet

'Create a variable to hold of gridlines are currently displayed
Dim currentSetting As Boolean

'Turn off screen updating
Application.ScreenUpdating = False

'Record the gridlines setting of the current worksheet
currentSetting = ActiveWindow.DisplayGridlines

'Record the active sheet
Set currentWs = ActiveSheet

'Loop through each worksheet in active workbook
For Each ws In ActiveWindow.SelectedSheets

    'Activate the sheet
    ws.Activate

    'Remove the gridlines
    ActiveWindow.DisplayGridlines = Not currentSetting
```

```
Next ws

'Revert back to active sheet
currentWs.Activate

'Turn on screen updating
Application.ScreenUpdating = True

End Sub
```

016 – Toggle worksheet tabs

What does it do?

Hides or displays the worksheet tabs.

VBA Code

```
Sub ToggleWorksheetTabs()

'Toggle the worksheet tabs
ActiveWindow.DisplayWorkbookTabs = _
    Not ActiveWindow.DisplayWorkbookTabs

End Sub
```

017 – Toggle worksheet headings

What does it do?

Hides or displays the row numbers and column letters on all selected sheets.

VBA Code

```
Sub ToggleHeadings()

'Toggle the display of headings
ActiveWindow.DisplayHeadings = _
    Not ActiveWindow.DisplayHeadings

End Sub
```

018 – Toggle formula bar display

What does it do?

Toggles the formula bar between hidden or visible.

VBA Code

```
Sub ToggleFormulaBarDisplay()

Application.DisplayFormulaBar = _
    Not Application.DisplayFormulaBar

End Sub
```

019 – Toggle status bar display

What does it do?

Toggles the status bar display between visible and hidden.

VBA Code

```
Sub ToggleStatusBarDisplay()

Application.DisplayStatusBar = _
    Not Application.DisplayStatusBar

End Sub
```

020 – Toggle scrollbar display

What does it do?

Horizontal and vertical scrollbars are separate objects; each can be visible or hidden. This macro rotates between the four options.

VBA Code

```
Sub ToggleScrollbarDisplay()

With ActiveWindow

    'Toggle between the 4 combined display options.
    Select Case .DisplayHorizontalScrollBar & _
        .DisplayVerticalScrollBar

        Case True & True
            .DisplayHorizontalScrollBar = False
            .DisplayVerticalScrollBar = False

        Case False & False
            .DisplayHorizontalScrollBar = True
            .DisplayVerticalScrollBar = False

        Case True & False
            .DisplayHorizontalScrollBar = False
```

```
            .DisplayVerticalScrollBar = True

        Case False & True
            .DisplayHorizontalScrollBar = True
            .DisplayVerticalScrollBar = True

    End Select

End With

End Sub
```

021 – Toggle background error checking

What does it do?

Turn on/off background error checking. This will make the small warning triangle in the corner of the cell appear or disappear.

VBA Code

```
Sub TurnOffBackgroundErrorCheck()

With Application.ErrorCheckingOptions

    'Turn off error checking
    .BackgroundChecking = Not .BackgroundChecking

End With

End Sub
```

022 – Toggle between R1C1 and A1

What does it do?

Toggle between A1 style cell referencing (i.e., standard Excel ranges reference) or R1C1 style referencing (for advanced users).

VBA Code

```
Sub ToggleormulaReferenceStyleToR1C1()

With Application

    'If currently in R1C1
    If .ReferenceStyle = xlR1C1 Then

        'Change to A1 style
        .ReferenceStyle = xlA1

    Else

        'Change to R1C1 style
        .ReferenceStyle = xlR1C1

    End If

End With

End Sub
```

023 – Toggle structured references with tables

What does it do?

Turn on/off structured references when using the tables feature.

VBA Code

```
Sub ToggleStructuredReferenceWithTables()

With Application

    If .GenerateTableRefs = xlGenerateTableRefStruct Then
```

```
        'Turn off structured references
        .GenerateTableRefs = xlGenerateTableRefA1

    Else

        'Turn on structured references
        .GenerateTableRefs = xlGenerateTableRefStruct

    End If

End With

End Sub
```

024 – Toggle setting workbook as final

What does it do?

Marking a workbook as final indicates to the user that it is a version which can be relied upon. This macro toggles the Final setting on/off.

VBA Code

```
Sub ToggleWorkbookFinal()

ActiveWorkbook.Final = Not ActiveWorkbook.Final

End Sub
```

025 – Display username in cell

What does it do?

Insert the username of the current user in the active cell.

VBA Code

```
Sub UserNameInCell()

'Put user name in the active cell
ActiveCell.Value = Application.UserName

End Sub
```

Notes:

The username in Excel and Windows are not the same thing. This code, and the code below only refer to the Excel username.

026 – Changes the Excel username

What does it do?

Changes the username based on input from the user.

VBA Code

```
Sub ChangeExcelUserName()

'Create variable to hold the User Name
Dim getUserName As Variant

'Enter User Name in an InputBox
'Entering blank will revert back to default user name
getUserName = Application.InputBox(prompt:="Enter User name", _
    Title:="User name", Type:=2)

If getUserName = False Then Exit Sub
```

```
'Change the Excel UserName
Application.UserName = getUserName

End Sub
```

027 – Change status bar message

What does it do?

Changes the message in the status bar.

VBA Code

```
Sub ChangeStatusBar()

Dim statusBarMessage As Variant

statusBarMessage = Application.InputBox(prompt:="Status bar message:", _
    Title:="Status bar", Type:=2)

'The User clicked Cancel
If statusBarMessage = False Then Exit Sub

If statusBarMessage = "" Then

    'Value is blank, so reset status bar
    Application.StatusBar = False

Else

    'Apply new status bar text
    Application.StatusBar = statusBarMessage

End If

End Sub
```

028 – Change caption at top of Excel workbook

What does it do?

Changes the text at the top of the Excel window.

VBA Code

```
Sub ChangeCaption()

Dim myCaption As Variant

myCaption = Application.InputBox(prompt:="Caption text:", _
    Title:="Caption", Type:=2)

'The User clicked Cancel
If myCaption = False Then Exit Sub

'Apply new caption text
ActiveWindow.caption = myCaption

End Sub
```

029 – Display Excel in full screen mode

What does it do?

Puts Excel into full screen mode.

VBA Code

```
Sub ToggleFullScreenDisplay()

Application.DisplayFullScreen = _
    Not Application.DisplayFullScreen

End Sub
```

030 – Toggle direction of row group outlining

What does it do?

Changes the direction of row group outlining.

VBA Code

```
Sub ToggleRowOutlining()

With ActiveSheet.Outline

    If .SummaryRow = xlBelow Then

        'Outline above
        .SummaryRow = xlAbove

    Else

        'Outline below
        .SummaryRow = xlBelow

    End If

End With

End Sub
```

031 – Toggle direction of column group outlining

What does it do?

Changes the direction of column group outlining.

VBA Code

```
Sub ToggleColumnOutlining()

With ActiveSheet.Outline

    If .SummaryColumn = xlRight Then
```

```
        'Outline left
        .SummaryColumn = xlLeft

    Else

        'Outline right
        .SummaryColumn = xlRight

    End If

End With

End Sub
```

032 – Toggle outline display

What does it do?

Hides the [+]. [-] and [number] symbols used in group outlining.

VBA Code

```
Sub ToggleOutlining()

'Toggle to hide or display outlining
ActiveWindow.DisplayOutline = _
    Not ActiveWindow.DisplayOutline

End Sub
```

033 – Toggle comment display

What does it do?

Toggles through all options for displaying Comments (known as Notes in more recent versions of Excel).

VBA Code

```
Sub ToggleCommentDisplay()

With Application

    'Toggle between 3 states of comment display
    Select Case .DisplayCommentIndicator

        Case xlNoIndicator
            .DisplayCommentIndicator = xlCommentIndicatorOnly

        Case xlCommentIndicatorOnly
            .DisplayCommentIndicator = xlCommentAndIndicator

        Case xlCommentAndIndicator
            .DisplayCommentIndicator = xlNoIndicator

    End Select

End With

End Sub
```

034 – Allow groups on protected worksheet

What does it do?

Enables users to interact with groupings on protected sheets.

VBA Code

```
Sub AllowGroupsOnAProtectedWorksheet()

Dim ws As Worksheet

'Loop through each worksheet in active workbook
For Each ws In ActiveWorkbook.Worksheets

    'Apply settings to every worksheet
    ws.Protect Password:="", UserInterfaceOnly:=True
    ws.EnableOutlining = True

Next ws

End Sub
```

035 – Freeze panes on all selected sheets

What does it do?

Freeze the pane on all selected sheets at the same time.

VBA Code

```
Sub FreezePaneOneAllSelectedSheets()

Dim ws As Worksheet
Dim currentWs As Worksheet

Application.ScreenUpdating = False

'Record the active sheet
Set currentWs = ActiveSheet
```

```
'Loop through each worksheet in active workbook
For Each ws In ActiveWindow.SelectedSheets

    'Activate the sheet
    ws.Activate

    'Remove existing freeze pane
    ActiveWindow.FreezePanes = False

    'Apply new freeze pane
    ActiveWindow.FreezePanes = True

Next ws

'Revert back to active sheet
currentWs.Activate

Application.ScreenUpdating = True

End Sub
```

Saving

036 – Save file with password to open

What does it do?

Save the file with password protection. The password is provided by the user.

VBA Code

```
Sub SaveFileWithPasswordToOpen()

'Create a variable to hold the password
Dim myPassword As Variant
Dim myPassword2 As Variant

'Set the password
myPassword = Application.InputBox(prompt:="Enter password", _
    Title:="Password", Type:=2)

'The User clicked Cancel
If myPassword = False Then Exit Sub

'Set the password
myPassword2 = Application.InputBox(prompt:="Re-enter password", _
    Title:="Password", Type:=2)

'The User clicked Cancel
If myPassword2 = False Then Exit Sub

'If passwords to not match then exit the macro
If myPassword <> myPassword2 Then

    MsgBox "The passwords do not match"
    Exit Sub

End If
```

```
'Save the file
ActiveWorkbook.SaveAs Password:=myPassword

End Sub
```

037 – Close workbook without saving changes

What does it do?

Close a workbook without saving and avoiding the Save Changes warning message.

VBA Code

```
Sub CloseFileNoSave()

'Force Excel to think no changes occured
ActiveWorkbook.Saved = True

'Close the file
ActiveWorkbook.Close

End Sub
```

038 – Save time stamped backup file

What does it do?

Save a backup copy of the workbook with a time stamp.

VBA Code

```
Sub SaveTimeStampedBackup()

'Create variable to hold the new file path
Dim saveAsName As String

'Set the file path
saveAsName = ActiveWorkbook.Path & "\" & _
    Format(Now, "yymmdd-hhmmss") & " " & ActiveWorkbook.Name
```

```
'Save the workbook
ActiveWorkbook.SaveCopyAs Filename:=saveAsName

End Sub
```

039 – Save and close all open workbooks

What does it do?

Save and close all open workbooks.

VBA Code

```
Sub SaveAllOpenWorkbooks()

'Create a variable to hold workbooks
Dim wb As Workbook

'Loop through each open workbook
For Each wb In Workbooks

    'Close each workbook and save changed
    wb.Close SaveChanges:=True

Next wb

End Sub
```

040 – Change workbook to read-only

What does it do?

Change the active workbook to read-only.

VBA Code

```
Sub ChangeWorkbookToReadOnly()

'Change workbook to read only
ActiveWorkbook.ChangeFileAccess Mode:=xlReadOnly

End Sub
```

041 – Prepare workbook for saving

What does it do?

The macro will, for each worksheet:

- Close all group outlining
- Set the view to the normal view
- Remove gridlines
- Hide all row numbers and column numbers
- Select cell A1

The first sheet is selected.

After running the macro, every worksheet in the workbook will be in a tidy state for the next use.

VBA Code

```
Sub PrepareWorkbookForSaving()

'Declare the worksheet variable
Dim ws As Worksheet

'Loop through each worksheet in the active workbook
For Each ws In ActiveWorkbook.Worksheets

    'Activate each sheet
    ws.Activate
```

```
    'Close all of groups
    ws.Outline.ShowLevels RowLevels:=1, ColumnLevels:=1

    'Set the view settings to normal
    ActiveWindow.View = xlNormalView

    'Remove the gridlines
    ActiveWindow.DisplayGridlines = False

    'Remove the headings on each of the worksheets
    ActiveWindow.DisplayHeadings = False

    'Get worksheet to display top left
    ws.Cells(1, 1).Select

Next ws

'Find the first visible worksheet and select it
For Each ws In Worksheets

    If ws.Visible = xlSheetVisible Then

        'Select the first visible worksheet
        ws.Select

        'Once the first visible worksheet is found exit the sub
        Exit For

    End If

Next ws

End Sub
```

Named Ranges

042 – Delete all named ranges

What does it do?

Deletes all existing named ranges (but ignores print areas).

VBA Code

```
Sub DeleteAllNamedRange()

'Create a variable to hold the named range
Dim n As Name

'Loop though each named range
For Each n In ActiveWorkbook.Names

    'Check if not a Print Area
    If Right(n.Name, 10) <> "Print_Area" Then

        'Delete name
        n.Delete

    End If

Next n

End Sub
```

043 – Delete all print areas

What does it do?

Delete all existing print areas from the active workbook.

VBA Code

```
Sub DeleteAllPrintAreas()

'Create a variable to hold the named range
Dim n As Name

'Loop though each named range
For Each n In ActiveWorkbook.Names

    'Check if not a Print Area
    If Right(n.Name, 10) = "Print_Area" Then

        'Delete name
        n.Delete

    End If

Next n

End Sub
```

044 – Hide named ranges

What does it do?

Makes a named range invisible to the named range window.

VBA Code

```
Sub ChangeNamedRangeVisibility()

'Make named range hidden from Name Manager
ActiveWorkbook.Names("myNamedRange").Visible = False

End Sub
```

Ranges & Cells

045 – Convert merged cells to center across

What does it do?

Changes all single row merged cells into center across formatting.

VBA Code

```
Sub ConvertMergedCellsToCenterAcross()

Dim c As Range
Dim mergedRange As Range

'Loop through all cells in Used range
For Each c In ActiveSheet.UsedRange

    'If merged and single row
    If c.MergeCells = True And c.MergeArea.Rows.Count = 1 Then

        'Set variable for the merged range
        Set mergedRange = c.MergeArea

        'Unmerge the cell and apply Centre Across Selection
        mergedRange.UnMerge
        mergedRange.HorizontalAlignment = xlCenterAcrossSelection

    End If

Next

End Sub
```

046 – Unhide all rows and columns

What does it do?

Makes all hidden rows and columns visible

VBA Code

```
Sub UnhideRowsColumns()

'Unhide the columns
ActiveSheet.Columns.EntireColumn.Hidden = False

'Unhide the rows
ActiveSheet.Rows.EntireRow.Hidden = False

End Sub
```

047 – Fit selection to screen

What does it do?

Zoom the screen on the selected cells.

VBA Code

```
Sub FitSelectionToScreen()

'To zoom to a specific area, then select the cells
Range("A1:I15").Select

'Zoom to selection
ActiveWindow.Zoom = True

'Select first cell on worksheet
Range("A1").Select

End Sub
```

PDFs

048 – Save each worksheet as a separate PDF

What does it do?

Save each worksheet as a single PDF file. The file name is based on the name of the tab.

VBA Code

```
Sub SaveEachWorksheetAsPDF()

Dim ws As Worksheet

'Loop through each worksheet in the active workbook
For Each ws In ActiveWorkbook.Worksheets

    'Save each sheet as PDF
    ws.ExportAsFixedFormat Type:=xlTypePDF, fileName:=ws.Name

Next

End Sub
```

049 – Save selected worksheets as a single PDF

What does it do?

Saves the selected worksheets into a single PDF file.

VBA Code

```
Sub SaveSelectedSheetsAsPDF()

'Save the selected sheets as PDF
ActiveSheet.ExportAsFixedFormat Type:=xlTypePDF, _
    Filename:=ActiveSheet.Name

End Sub
```

Copying worksheets

050 – Copy active sheet to a new workbook

What does it do?

Copy the active worksheet to a new workbook.

VBA Code

```
Sub CopyActiveWorksheetToNewWorkbook()

'Copy the sheet
ActiveSheet.Copy

End Sub
```

051 – Copy selected sheets to new workbooks and save

What does it do?

Copies the selected sheets into new workbooks. Values are hardcoded, then the new workbook is saved and closed.

VBA Code

```
Sub CopySelectedSheetsToNewWorkbookHardCodeSaveClose()

'Create variables to hold worksheets
Dim ws As Worksheet

'Create variables to hold the source and target workbook
Dim wbTarget As Workbook
Dim wbSource As Workbook

'Set wbSource to the ActiveWorkbook
Set wbSource = ActiveWorkbook

For Each ws In ActiveWindow.SelectedSheets
```

```
    'Copy the worksheet to a new workbook
    ws.Copy

    'New workbook becomes the ActiveWorkbook
    Set wbTarget = ActiveWorkbook

    'Hardcode the values in the new workbook
    With wbTarget.Sheets(ws.Name).UsedRange
        .Value = .Value
    End With

    'Save and close the workbook
    Application.DisplayAlerts = False
    wbTarget.SaveAs wbSource.Path & "\" & ws.Name & ".xlsx"
    wbTarget.Close
    Application.DisplayAlerts = True

Next ws

End Sub
```

Files and folders

052 – Check if a file exists

What does it do?

Checks if a file already exists.

VBA Code

```
Sub CheckfFileExists()

'Create variable to hold the file path
Dim filePath As String

'Set file path to a specific file
filePath = "C:\Users\marks\Documents\FileName.xlsx"

'Display message box True = file exists, false = does not exist
MsgBox prompt:=Dir(filePath) <> "", Title:="File exists"

End Sub
```

053 – Rename or move file or folder

What does it do?

Rename or move a file or folder.

VBA Code

```
Sub RenameMoveFileOrFolder()

'Create variables to hold the file paths
Dim currentFilePath As String
Dim newFilePath As String

'Set the file paths
currentFilePath = "C:\Users\marks\Documents\CurrentFileName.xlsx"
newFilePath = "C:\Users\marks\Documents\NewFileName.xlsx"
```

```
'Move the file
Name currentFilePath As newFilePath

End Sub
```

054 – Copy a file

What does it do?

Copy a file and save it to a new location.

VBA Code

```
Sub CopyFile()

'Create variables to hold the file paths
Dim copyFilePath As String
Dim pasteFilePath As String

'Set the file paths
copyFilePath = "C:\Users\marks\Documents\copyThisFile.xlsx"
pasteFilePath = "C:\Users\marks\Documents\pasteFileHere.xlsx"

'Copy the file
FileCopy copyFilePath, pasteFilePath

End Sub
```

055 – Delete a file

What does it do?

Deletes a file.

VBA Code

```
Sub DeleteFile()

'Create variable to hold the file path
Dim filePath As String
```

```
'Set file path to a specific file
filePath = "C:\Users\marks\Documents\DeleteMe.xlsx"

'Alternatives
'Delete all .xlsx files
'filePath = "C:\Users\marks\Documents\DeleteFolder\*.xlsx"
'Delete all files in a folder
'filePath = "C:\Users\marks\Documents\DeleteFolder\*.*"

'Delete the file
Kill filePath

End Sub
```

056 – Create all folder in a file path

What does it do?

Creates a folder, including any folders along the file path which do not already exist.

VBA Code

```
Sub createAllFolders()

'Create necessary variables
Dim folderPath As String
Dim individualFolders() As String
Dim tempFolderPath As String
Dim arrayElement As Variant

'The desired folder path
folderPath = "C:\Users\marks\Documents\New Folder\New Folder\" & _
    "New Folder\New Folder"

'Split the folder path into individual folder names
individualFolders = Split(folderPath, "\")

'Loop though each individual folder name
For Each arrayElement In individualFolders
```

```
'Build string of folder path
    tempFolderPath = tempFolderPath & arrayElement & "\"

    'If folder does not exist, then create it
    If Dir(tempFolderPath, vbDirectory) = "" Then

        MkDir tempFolderPath

     End If

Next arrayElement

End Sub
```

057 – Delete a folder and its contents

What does it do?

Deletes a folder with all its contents

VBA Code

```
Sub DeleteFolder()

'Create variable
Dim folderPath As String

'Ensure the folder path has a "\" at the end of the string
folderPath = "C:\Users\marks\Documents\Delete Folder\"

'Use wildcards to delete all the files in the folder
Kill folderPath & "*.*"

'Delete the now empty folder
RmDir folderPath

End Sub
```

Adjusting cell values

058 – Flip number signage on selected cells

What does it do?

Flips the number signage of all numeric values in the selected cells

VBA Code

```
Sub FlipNumberSignage()

'Create variable to hold cells in the worksheet
Dim c As Range

'Loop through each cell in selection
For Each c In Selection

    'Test if the cell contents is a number
    If IsNumeric(c) Then

        'Convert signage for each cell
        c.Value = -c.Value

    End If

Next c

End Sub
```

059 – Convert sheet to hardcoded values

What does it do?

Hard codes all cells on the worksheet.

VBA Code

```
Sub ConvertActiveSheetToValues()

'Uses the Used Range
With ActiveSheet.UsedRange

    'Hardcode the values
    .Value = .Value

End With

End Sub
```

060 – Convert all worksheets in workbook to hardcoded values

What does it do?

Hard codes all cells in a workbook.

VBA Code

```
Sub ConvertAllWorksheetsToValues()

'Create a variable to hold worksheets
Dim ws As Worksheet

'Loop through each worksheet
For Each ws In ActiveWorkbook.Worksheets

    'Convert the worksheet to values
    With ActiveSheet.UsedRange

        .Value = .Value


    End With

Next ws

End Sub
```

061 – Swap selected ranges

What does it do?

Rotates two or more selected ranges

VBA Code

```
Sub SwapSelectedRanges()

'Create variables to hold the ranges
Dim rng As Range
Dim tempRng As Variant
Dim areaCount As Long
Dim areaRows As Long
Dim areaCols As Long
Dim i As Integer
Dim j As Integer

Set rng = Selection
areaCount = rng.Areas.Count

'There must be at least two areas selected
If areaCount < 2 Then
    MsgBox "Please select atleast two ranges."
    Exit Sub
End If

'All areas must be the same shape
areaRows = rng.Areas(1).Rows.Count
areaCols = rng.Areas(1).Columns.Count

For i = 2 To areaCount
    If rng.Areas(i).Rows.Count <> areaRows Or _
        rng.Areas(i).Columns.Count <> areaCols Then
        MsgBox "All ranges must have the same number of rows and
columns."
        Exit Sub
    End If
Next i
```

```
'Check that ranges don't intersect with each other
For j = 1 To areaCount - 1
    For i = 1 + j To areaCount
        If Not Intersect(rng.Areas(i), rng.Areas(j)) Is Nothing Then
            MsgBox "Selected areas must not overlap."
        End If
    Next i
Next j

'Swap the ranges
tempRng = rng.Areas(areaCount).Cells.Formula
For i = areaCount To 2 Step -1
rng.Areas(i).Cells.Formula = rng.Areas(i - 1).Cells.Formula
Next i
rng.Areas(1).Cells.Formula = tempRng

End Sub
```

062 – Clear all data cells

What does it do?

Clears all cells in the selection which are constants (i.e. not formulas).

VBA Code

```
Sub ClearAllDataCellsInSelection()

'Clear all hardcoded values in the selected range
Selection.SpecialCells(xlCellTypeConstants).ClearContents

End Sub
```

063 – Apply sentence case to selection

What does it do?

Applies sentence case (i.e., the first letter in each sentence has a capital letter) to all the cells in the selection.

VBA Code

```
Sub ApplySentenceCase()

Dim rng As Range
Dim c As Range
Dim letter As String
Dim capitalize As Boolean
Dim finalString As String
Dim i As Integer

Set rng = Selection

'loop through each cell in selection
For Each c In rng

    finalString = ""
    capitalize = True

    'loop through each letter in next string
    For i = 1 To Len(c)

        letter = Mid(c.Value, i, 1)

        'If letter is a period, then turn on capitalize switch
        If letter = "." Then capitalize = True

        'If capitalize switch is on, then make upper case
        If capitalize = True Then

            letter = UCase(letter)
```

```
            'Turn off capitalize switch if capital found
            If letter >= "A" And letter <= "Z" Then
                capitalize = False
            End If

        'If letter is not to be capitalized, then make lower case
        Else
            letter = LCase(letter)
        End If

        'Add the letter onto the new text string
        finalString = finalString & letter

    Next i

    'Return the value back to cell
    c.Value = finalString

Next c

End Sub
```

064 – Apportion a value across cells

What does it do?

Apportion a value across all the selected cells.

VBA Code

```
Sub ApportionValueAcrossCells()

Dim apportionValue As Double
Dim keepAsFormula As Long
Dim total As Double
Dim c As Range
Dim formulaString As String
```

```
'Get the existing total
total = Application.WorksheetFunction.Sum(Selection)

'Check that sume of selected cells is not zero
If total = 0 Then

    MsgBox Prompt:="Selected cells must not sum to zero", _
        Title:="Apportion value"
    Exit Sub

End If

'Get the value to apportion
apportionValue = Application.InputBox(Prompt:="Value to apportion:", _
    Title:="Apportion value", Type:=1)

'The User clicked Cancel
If apportionValue = False Then Exit Sub

'Get the boolean value to keep the formula or hardcode the result
keepAsFormula = MsgBox("Keep formula?", vbYesNo)

'Loop through each cell in selection
For Each c In Selection

    If IsNumeric(c.Value) Then

        'Calculate the result of the cell
        formulaString = c.Formula & "+(" & apportionValue & _
            "/" & total & "*" & c.Value & ")"

        If Left(formulaString, 1) <> "=" Then _
            formulaString = "=" & formulaString

        'Enter the formula into the cell
        c.Formula = formulaString

        'Recalculate the active cell
        ActiveCell.Calculate
```

```
        'If keepAsFormula is no, then hardcode the result
        If keepAsFormula = vbNo Then
            c.Value = c.Value
        End If

    End If

Next c

End Sub
```

065 – Add prefix to each cell in selection

What does it do?

Adds a prefix to each cell in the selected cells (excludes formulas and blanks).

VBA Code

```
Sub AddPrefix()

Dim c As Range
Dim prefixValue As Variant

'Display inputbox to collect prefix text
prefixValue = Application.InputBox(Prompt:="Enter prefix:", _
Title:="Prefix", Type:=2)

'The User clicked Cancel
If prefixValue = False Then Exit Sub

For Each c In Selection

    'Add prefix where cell is not a formula or blank
    If Not c.HasFormula And c.Value <> "" Then

        c.Value = prefixValue & c.Value

    End If
```

```
Next

End Sub
```

066 – Add suffix to each cell in selection

What does it do?

Adds a suffix to each value in the selected cells (excludes formulas and blanks).

VBA Code

```
Sub AddSuffix()

Dim c As Range
Dim suffixValue As Variant

'Display inputbox to collect prefix text
suffixValue = Application.InputBox(Prompt:="Enter Suffix:", _
    Title:="Suffix", Type:=2)

'The User clicked Cancel
If suffixValue = False Then Exit Sub

'Loop through each cellin selection
For Each c In Selection

    'Add Suffix where cell is not a formula or blank
    If Not c.HasFormula And c.Value <> "" Then

        c.Value = c.Value & suffixValue

    End If

Next

End Sub
```

067 – Insert rows between existing data

What does it do?

Adds a blank row every n rows.

VBA Code

```
Sub InsertRowsBetween()

Dim rng As Range
Dim i As Long
Dim interval As Integer

Set rng = Selection
interval = Application.InputBox(prompt:="Insert row every:", _
    Title:="Insert rows", Type:=1)

'The User clicked Cancel
If interval = False Then Exit Sub

'Loop through all cells from the end
For i = rng.Rows.Count To 1 Step -1



    'If row is every n rows insert row
    If i Mod interval = 0 Then

        Rows(rng.Row + i).EntireRow.Insert

    End If

Next i

End Sub
```

068 – Remove characters from start

What does it do?

Removes the first n characters from all cells in the selection.

VBA Code

```
Sub RemoveCharacterFromStart()

Dim c As Range
Dim rng As Range
Dim chrToRemove As Variant

Set rng = Selection

'Get the characters to be removed from the user
chrToRemove = Application.InputBox(prompt:= _
    "Number of characters to remove from start:", _
    Title:="Number of characters", Type:=1)

'The User clicked Cancel
If chrToRemove = False Then Exit Sub

'Loop through all cell in selection
For Each c In rng


    'If characters is less then string length, then blank
    If chrToRemove < Len(c) Then

        'Record the values with characters removed
        c.Value = Right(c, Len(c) - chrToRemove)

    Else

        c.Value = ""

    End If
```

```
Next c

End Sub
```

069 – Remove characters from end

What does it do?

Removes the last n characters from all cells in the selection.

VBA Code

```
Sub RemoveCharacterFromEnd()

Dim c As Range
Dim rng As Range
Dim chrToRemove As Variant

Set rng = Selection

'Get the characters to be removed from the user
chrToRemove = Application.InputBox(prompt:= _
    "Number of characters to remove from end:", _
    Title:="Number of characters", Type:=1)

'The User clicked Cancel
If chrToRemove = False Then Exit Sub



'Loop through all cell in selection
For Each c In rng

    'If characters is less then string length, then blank
    If chrToRemove < Len(c) Then
        'Record the values with characters removed
        c.Value = Left(c, Len(c) - chrToRemove)
    Else
            c.Value = ""
    End If
```

```
Next c

End Sub
```

070 – Reverse row order

What does it do?

Reverses the order of all rows of data in the selection.

VBA Code

```
Sub ReverseRows()

'Create variables
Dim rng As Range
Dim rngArray As Variant
Dim tempRng As Variant
Dim i As Long
Dim j As Long
Dim k As Long

'Record the selected range and it's contents
Set rng = Selection
rngArray = rng.Formula

'Loop through all cells and create a temporary array
For j = 1 To UBound(rngArray, 2)
    k = UBound(rngArray, 1)
    For i = 1 To UBound(rngArray, 1) / 2
        tempRng = rngArray(i, j)
        rngArray(i, j) = rngArray(k, j)

        rngArray(k, j) = tempRng
        k = k - 1
    Next
Next
```

```
'Apply the array
rng.Formula = rngArray

End Sub
```

071 – Reverse column order

What does it do?

Reverses the order of all column data in the selection.

VBA Code

```
Sub ReverseColumns()

'Create variables
Dim rng As Range
Dim rngArray As Variant
Dim tempRng As Variant
Dim i As Long
Dim j As Long
Dim k As Long

'Record the selected range and it's contents
Set rng = Selection
rngArray = rng.Formula



'Loop through all cells and create a temporary array
For i = 1 To UBound(rngArray, 1)
    k = UBound(rngArray, 2)
    For j = 1 To UBound(rngArray, 2) / 2
        tempRng = rngArray(i, j)
        rngArray(i, j) = rngArray(i, k)
        rngArray(i, k) = tempRng

        k = k - 1
    Next
Next
```

```
'Apply the array
rng.Formula = rngArray

End Sub
```

072 – Transpose selection

What does it do?

Transposes the selected cells with a single click.

VBA Code

```
Sub TransposeSelection()

'Create variables
Dim rng As Range
Dim rngArray As Variant
Dim i As Long
Dim j As Long
Dim overflowRng As range
Dim msgAns As Long

'Record the selected range and it's contents
Set rng = Selection
rngArray = rng.Formula

'Test the range and identify if any cells will be overwritten
If rng.Rows.Count > rng.Columns.Count Then

    Set overflowRng = rng.Cells(1, 1). _
        Offset(0, rng.Columns.Count). _
        Resize(rng.Columns.Count, _
        rng.Rows.Count - rng.Columns.Count)

ElseIf rng.Rows.Count < rng.Columns.Count Then

    Set overflowRng = rng.Cells(1, 1).Offset(rng.Rows.Count, 0). _
        Resize(rng.Columns.Count - rng.Rows.Count, rng.Rows.Count)
```

```
End If

If rng.Rows.Count <> rng.Columns.Count Then

    If Application.WorksheetFunction.CountA(overflowRng) > 0 Then

    msgAns = MsgBox("Worksheet data in " & overflowRng.Address & _
        " will be overwritten." & vbNewLine & _
        "Do you wish to continue?", vbYesNo)

    If msgAns = vbNo Then Exit Sub

    End If

End If

'Clear the rnage
rng.Clear

'Reapply the cells in transposted position
For i = 1 To UBound(rngArray, 1)

    For j = 1 To UBound(rngArray, 2)

        rng.Cells(1, 1).Offset(j - 1, i - 1) = rngArray(i, j)

    Next

Next

End Sub
```

Shapes and pictures

073 – Create red box around selected areas

What does it do?

Draws a rectangle shape to fit around the selected cells.

VBA Code

```
Sub AddRedBox()

Dim redBox As Shape
Dim selectedAreas As range
Dim i As Integer
Dim tempShape As Shape

'Loop through each selected area in active sheet
For Each selectedAreas In Selection.Areas

    'Create a rectangle
    Set redBox = ActiveSheet.Shapes.AddShape(msoShapeRectangle, _
        selectedAreas.Left, selectedAreas.Top, _
        selectedAreas.Width, selectedAreas.Height)

    'Change attributes of shape created
    redBox.Line.ForeColor.RGB = RGB(255, 0, 0)
    redBox.Line.Weight = 2
    redBox.Fill.Visible = msoFalse

    'Loop to find a unique shape name
    Do
        i = i + 1
        Set tempShape = Nothing

        On Error Resume Next
        Set tempShape = ActiveSheet.Shapes("RedBox_" & i)
        On Error GoTo 0

    Loop Until tempShape Is Nothing
```

```
    'Rename the shape
    redBox.Name = "RedBox_" & i

Next

End Sub
```

074 – Delete all red boxes on active sheet

What does it do?

Having created the red boxes in the macro above. This code removes all the red boxes on the active sheet with a single click.

VBA Code

```
Sub DeleteRedBox()

Dim shp As Shape

'Loop through each shape on active sheet
For Each shp In ActiveSheet.Shapes

    'Find shapes with a name starting with "RedBox_"
    If Left(shp.Name, 7) = "RedBox_" Then

        'Delete the shape
        shp.Delete

    End If

Next shp

End Sub
```

075 – Paste cells as picture

What does it do?

Copies the selected cells and pastes as a static picture.

VBA Code

```
Sub PasteCellaAsPicture()

'Copy the selection as a picture
Selection.CopyPicture Appearance:=xlScreen, Format:=xlPicture

'Offset the picture to start in column next to selection
ActiveCell.Offset(1, Selection.Columns.Count).Select

'Paste the picture
ActiveSheet.Paste

End Sub
```

076 – Paste cells as linked picture

What does it do?

Copies the selected cells and pastes as a linked picture. If the cells in the original picture range change, so does the picture.

VBA Code

```
Sub PasteCellaAsLinkedPicture()

'Copy the selection
Selection.Copy

'Offset the picture to start in column next to selection
ActiveCell.Offset(1, Selection.Columns.Count).Select

'Paste the copy an a linked image
ActiveSheet.Pictures.Paste Link:=True
```

```
'Remove the marching ants
Application.CutCopyMode = False

End Sub
```

Charts

077 – Save selected chart as an image

What does it do?

Saves the selected chart as a picture to the file location contained in the macro.

VBA Code

```
Sub ExportSingleChartAsImage()

'Create a variable to hold the path and name of image
Dim imagePath As String
Dim cht As Chart

imagePath = "C:\Users\marks\Documents\myImage.png"
Set cht = ActiveChart

'Export the chart
cht.Export (imagePath)

End Sub
```

078 – Resize all charts to same as active chart

What does it do?

Select the chart with the dimensions you wish to use, then run the macro. All the charts will resize to the same dimensions.

VBA Code

```
Sub ResizeAllCharts()

'Create variables to hold chart dimensions
Dim chtHeight As Long
Dim chtWidth As Long

'Create variable to loop through chart objects
Dim chtObj As ChartObject
```

```
'Get the size of the first selected chart
chtHeight = ActiveChart.Parent.Height
chtWidth = ActiveChart.Parent.Width

For Each chtObj In ActiveSheet.ChartObjects

    chtObj.Height = chtWidth
    chtObj.Width = chtWidth

Next chtObj

End Sub
```

Power Query

079 – Refresh a Power Query connection

What does it do?

Refreshes a specific query. Can be used to control the order which queries are updated.

VBA Code

```
Sub RefreshPowerQueryConnections()

'Name of Query comes from Data -> Existing Connections
'Use "Query - " then the name of the  connection
ActiveWorkbook.Connections("Query - ImportCSV").Refresh

End Sub
```

080 – Change all connections to prevent background refresh

What does it do?

Background refresh enables Power Query to refresh data while you keep working. However, this can lead to incorrect values as Pivot Tables will refresh before the query is refreshed. Instead, turning off background refresh ensures the refresh occurs in the correct order.

VBA Code

```
Sub ChangeAllQueriesPreventBackgroundRefresh()

Dim counter As Long

For counter = 1 To ActiveWorkbook.Connections.Count

    'Exclude PowerPivot connections
    If ActiveWorkbook.Connections(counter).Type = _
        xlConnectionTypeOLEDB Then
```

```
        'Change Background Query refresh to false
        ActiveWorkbook.Connections(counter).OLEDBConnection. _
            BackgroundQuery = False

    End If

Next counter

End Sub
```

Pivot Tables

081 – Refresh all Pivot Tables in workbook

What does it do?

Refresh all the Pivot Tables in the active workbook.

VBA Code

```
Sub RefreshAllPivotTables()

'Refresh all pivot tables
ActiveWorkbook.RefreshAll

End Sub
```

082 – Delete all Pivot Tables in workbook

What does it do?

Deletes all the Pivot Tables in the active workbook.

VBA Code

```
Sub DeleteAllPivotTable()

'Create a variable to hold worksheets
Dim ws As Worksheet
'Create a variable to hold pivot tables
Dim pvt As PivotTable

'Loop through each sheet in the activeworkbook
For Each ws In ActiveWorkbook.Worksheets

    'Loop through each pivot table in the worksheet
    For Each pvt In ws.PivotTables

        'ClearPivot Tables
        pvt.TableRange2.Clear
```

```
    Next pvt

Next ws

End Sub
```

083 – Remove subtotals from Pivot Table

What does it do?

If the active cell is within a Pivot Table, the macro will remove all the sub-totals from that Pivot Table.

VBA Code

```
Sub HidePivotTableSubtotals()

Dim pvt As PivotTable
Dim pvtField As PivotField

On Error Resume Next

'Get the pivot tabled based on active cell
Set pvt = ActiveSheet.PivotTables(ActiveCell.PivotTable.Name)

'Check if a pivot table is found
If pvt Is Nothing Then

    MsgBox "Select a cell from a Pivot Table."
    Exit Sub

End If

'Loop through fields
For Each pvtField In pvt.PivotFields

    'Hide the pivot table fields
    pvtField.Subtotals(1) = False
```

```
Next pvtField

End Sub
```

084 – Turn off auto fit columns on all Pivot Tables

What does it do?

By default, PivotTables resize columns to fit the contents. This macro changes the setting for every PivotTable in the active workbook, so that column widths set by the user are maintained.

VBA Code

```
Sub TurnOffAutofitColumns()

'Create a variable to hold worksheets
Dim ws As Worksheet

'Create a variable to hold pivot tables
Dim pvt As PivotTable

'Loop through each sheet in the activeworkbook
For Each ws In ActiveWorkbook.Worksheets

    'Loop through each pivot table in the worksheet
    For Each pvt In ws.PivotTables

        'Turn off auto fit columns on PivotTable
        pvt.HasAutoFormat = False

    Next pvt

Next ws

End Sub
```

085 – Toggle GetPivotDataFormula

What does it do?

Toggles the setting which switches between GetPivotData and standard cell referencing when using a cell in a PivotTable.

VBA Code

```
Sub SetGetPivotData()

'Toggle GetPivotData
Application.GenerateGetPivotData = _
    Not Application.GenerateGetPivotData

End Sub
```

Miscellaneous

086 – Get color code from cell fill color

What does it do?

Returns the RGB and Hex for the active cell's fill color.

VBA Code

```
Sub GetColorCodeFromCellFill()

'Create variables hold the color data
Dim fillColor As Long
Dim R As Integer
Dim G As Integer
Dim B As Integer
Dim Hex As String

'Get the fill color
fillColor = ActiveCell.Interior.Color

'Convert fill color to RGB
R = (fillColor Mod 256)
G = (fillColor \ 256) Mod 256
B = (fillColor \ 65536) Mod 256

'Convert fill color to Hex
Hex = "#" & Application.WorksheetFunction.Dec2Hex(fillColor)

'Display fill color codes
MsgBox "Color codes for active cell" & vbNewLine & _
    "R:" & R & ", G:" & G & ", B:" & B & vbNewLine & _
    "Hex: " & Hex, Title:="Color Codes"

End Sub
```

087 – Open calculator app

What does it do?

Opens the calculator app.

VBA Code

```
Sub OpenCalculatorApp()

'Open the Calculator app
Application.ActivateMicrosoftApp Index:=0

End Sub
```

088 – Word count

What does it do?

Counts the number of words within the selected range.

VBA Code

```
Sub WordCount()

'Create a variable to hold a cell
Dim c As Range

'Create a variable to track the word count
Dim wordCount As Long

'Create a temporary variable
Dim tempString As String

'Loop through each cell in selected cells
For Each c In Selection

        'temporary variable is set to each cell
        tempString = c.Value

        'temporary variable has spaces removed
        tempString = Trim(tempString)
```

```
        'If temporary variable is not empty then count it
        If tempString <> "" Then

            wordCount = wordCount + 1

        End If

        'Create look to count spaces between words
        Do While InStr(tempString, " ") > 0

            wordCount = wordCount + 1
            tempString = Mid(tempString, InStr(tempString, " "))
            tempString = Trim(tempString)

        Loop

Next c

'Display the number of words
MsgBox "Total word count in selection = " & wordCount

End Sub
```

089 – Insert custom header

What does it do?

Inserts a custom header which is displayed when the document is printed.

VBA Code

```
Sub AddCustomHeader()

'Create variable to hold the header text
Dim headerText As Variant

'Show input box to collect the header text
headerText = Application.InputBox(prompt:="Header text:", _
    Title:="Header", Type:=2)
```

```
'The User clicked Cancel
If headerText = False Then Exit Sub

With ActiveSheet.PageSetup
    '.LeftHeader = ""
    .CenterHeader = headerText
    '.RightHeader = ""
End With

End Sub
```

090 – Insert custom footer

What does it do?

Inserts a custom footer which is displayed when the document is printed.

VBA Code

```
Sub AddCustomFooter()

'Create variable to hold the footer text
Dim footerText As Variant

'Show input box to collect the header text
footerText = Application.InputBox(prompt:="Footer text:", _
    Title:="Footer", Type:=2)

'The User clicked Cancel
If footerText = False Then Exit Sub

With ActiveSheet.PageSetup
    '.LeftFooter = ""
    .CenterFooter = footerText
    '.RightFooter = ""
End With

End Sub
```

091 – Create a table of contents

What does it do?

Creates or refreshes a hyperlinked table of contents on a worksheet called “TOC”, which is placed at the start of a workbook.

VBA Code

```
Sub CreateTableOfContents()

Dim i As Long
Dim TOCName As String

'Name of the Table of contents
TOCName = "TOC"

'Delete the existing Table of Contents sheet if it exists
On Error Resume Next
Application.DisplayAlerts = False
ActiveWorkbook.Sheets(TOCName).Delete
Application.DisplayAlerts = True
On Error GoTo 0

'Create a new worksheet
ActiveWorkbook.Sheets.Add before:=ActiveWorkbook.Worksheets(1)
ActiveSheet.Name = TOCName

'Loop through the worksheets
For i = 1 To Sheets.Count

    'Create the table of contents
    ActiveSheet.Hyperlinks.Add _
        Anchor:=ActiveSheet.Cells(i, 1), _
        Address:="", _
        SubAddress:="'" & Sheets(i).Name & "'!A1", _
        ScreenTip:=Sheets(i).Name, _
        TextToDisplay:=Sheets(i).Name
```

```
Next i

End Sub
```

092 – Excel to speak the cell contents

What does it do?

Excel speaks back the contents of the selected cells

VBA Code

```
Sub SpeakCellContents()

'Speak the selected cells
Selection.Speak

End Sub
```

093 – Fix the range of cells which can be scrolled

What does it do?

Fixes the scroll range to the selected cell range. It prevents a user from scrolling into other parts of the worksheet.

If a single cell is selected, the scroll range is reset.

VBA Code

```
Sub FixScrollRange()

If Selection.Cells.Count = 1 Then

    'If one cell selected, then reset
    ActiveSheet.ScrollArea = ""

Else
    'Set the scroll area to the selected cells
    ActiveSheet.ScrollArea = Selection.Address
```

```
End If

End Sub
```

094 – Force message box to front of all windows

What does it do?

This message box will appear in front of all applications, even if Excel is not the active application.

VBA Code

```
Sub ForceMsgBoxToFrontOfApplications()

'Use vbSystemModal to force a message box to the front
MsgBox "Forced to the front", vbSystemModal

End Sub
```

095 – Invert the sheet selection

What does it do?

Select some worksheet tabs, then run the macro to reverse the selection.

VBA Code

```
Sub InvertSheetSelection()

'Create variable to hold list of selected worksheet
Dim selectedList As String

'Create variable to hold worksheets
Dim ws As Worksheet

'Create variable to switch after the first sheet selected
Dim firstSheet As Boolean
```

```
'Convert selected sheest to a text string
For Each ws In ActiveWindow.SelectedSheets
    selectedList = selectedList & ws.Name & "[|]"
Next ws

'Set the toggle of first sheet
firstSheet = True

'Loop through each worksheet in the active workbook
For Each ws In ActiveWorkbook.Sheets

    'Check if the worksheet was not previously selected
    If InStr(selectedList, ws.Name & "[|]") = 0 Then

        'Check the worksheet is visible
        If ws.Visible = xlSheetVisible Then

            'Select the sheet
            ws.Select firstSheet

            'First worksheet has been found, toggle to false
            firstSheet = False

        End If

    End If

Next ws

End Sub
```

096 – Remove external links

What does it do?

Removes external links from the cells of a workbook.

VBA Code

```
Sub RemoveExternalLinks()

Dim linkArray As Variant
Dim i As Integer

'Create array of each link source
linkArray = ActiveWorkbook.LinkSources(1)

On Error Resume Next

'Look through each link source
For i = 1 To UBound(linkArray)

    'Break the link
    ActiveWorkbook.BreakLink linkArray(i), xlLinkTypeExcelLinks

Next i

On Error GoTo 0

End Sub
```

097 – Create a custom List

What does it do?

Custom lists are used for sorting or entering data into a predefined order. For example, the days of the week are not in alphabetical order, so to sort by the days of the week a custom list is used.

The macro below creates a custom list from the selected cells.

VBA Code

```
Sub CreateCustomList()

'Add a new Custom List from range of cells
Application.AddCustomList ListArray:=range(Selection.Address)
```

```
'Add a new Custom List from list
'Application.AddCustomList ListArray:=Array("Element1", _
    "Element2", "Element3")

End Sub
```

098 – Delete a custom list

What does it do?

Having created a custom list in the macro above, we may decide to then delete that custom list. The macro deletes any custom list where the first value matches the active cell.

VBA Code

```
Sub DeleteCustomList()

'Find the listNum of a Custom List based on all items
Dim listNumFound As Integer
Dim i As Integer
Dim arrayItem As Variant
Dim customListContents() As Variant

listNumFound = Application.GetCustomListNum(Array( _
    Selection.Address))

For i = 1 To Application.CustomListCount

    'Set the CustomList array to a variable
    customListContents = Application.GetCustomListContents(i)



    'Loop through each element in the CustomList
    For Each arrayItem In customListContents

        'Test if the element has a specific value
        If arrayItem = Selection.Cells(1, 1).Value Then
```

```
        'Delete the custom list
        Application.DeleteCustomList listNum:=i

        End If

    Next arrayItem

Next i

End Sub
```

099 – Assign a macro to a shortcut key

What does it do?

Assigns a macro to a shortcut key.

VBA Code

```
Sub AssignMacroToShortcut()

'+ = Ctrl
'^ = Shift
'{T} = the shortcut letter

Application.OnKey "+^{T}", "nameOfMacro"

'Reset shortcut to default - repeat without the name of the macro
'Application.OnKey "+%{T}"

End Sub
```

100 – Apply single accounting underline to selection

What does it do?

Single accounting underline is a formatting style which is not available in the ribbon. The macro below applies single accounting underline to the selected cells.

VBA Code

```
Sub SingleAccountingUnderline()

'Apply single accounting underline to selected cells
Selection.Font.Underline = xlUnderlineStyleSingleAccounting

End Sub
```