

Contents



Lesson	Summary	Page

Your First Module



Activating the Developer Tab

Select Options->Customise Ribbon

either check or uncheck the Developer check box.

Visual Basic Editor

Most of your work will be done in macros - so you can open up Visual Basic Editor by <alt>f11, or Developer Tab->Visual Basic Editor

Your First Module

Modules can be attached to a specific worksheet or the whole workbook.

Double click on a sheet or the work book to open up that pane.

Just type 'sub fred' then return - it creates the module and capitalises the words for you. You can then start typing your code in, eg

```
Sub fred()  
    Range("a5") = 12  
End Sub
```

Now click the play button or F5 to run it - the cell A5 now has '12' in it.

Ranges



Ranges

A range is just a set of cells. They are normally next to each other - but they don't have to be...

```
Range("a5") = 12          ' denotes a single cell
Range("c2:c6") = 15       ' denotes top left, bottom right
Range("d5:g8") = 19       ' denotes top left, bottom right
Range("a1,a6:a8,g8") = 5  ' mixture of single and ranges
Range("A2") = "Hello"     ' denotes a single cell
```

Are all valid ranges

Debugging



Debugging

You can press F8 or Step Into, this will start the module and stop just before the first statement.

You can then...

F8 - step in - if the current statement is a call to another module, it starts debugging that module. If the current statement is a single command - it gets executed.

Shift F8 - step over, if the current statement is a call to another module, the module is executed at full speed, then stops just before the next statement after it returns.

CtrlShift F8 - step out, runs the rest of this module and stops at the first statement when it returns.

Ctrl F8 - run to cursor, exactly what you would expect.

Saving



Saving

You will normally want to save your macros with the workbook. In this case you need to save it as a 'macro enabled' workbook (xlsm) instead of the normal.xlsx.

You can change the default extension by selecting options->save.

Clickable Controls



Shapes as buttons

The easiest way to run macros/modules is to use buttons.

Basically a button is an object with an on-click event.

The most common use is to place a shape on your spreadsheet, right click - and Assign Macro.

Buttons

The easiest way to run macros/modules is to use buttons.

Basically a button is an object with an on-click event.

The most common use is to place a shape on your spreadsheet, right click - and Assign Macro.

The Developer Tab has some nifty controls though. If the Design Mode button is pressed - you can insert controls onto the sheet, resize, recolour etc. Once you are out of Design Mode, if you click on the button - it actions the macro.

There are two types of controls - Form Controls and ActiveX controls.

ActiveX appear to be the best.

If you add a button control, then you add code behind it by double clicking it (when in design mode) - a new sub opens up.

Macros



Recording Macros

Sometimes you don't know what the commands are - but you know how to access them from the Excel menus.

If this is the case - simply click 'record macro' - then perform the actions you want - when you click 'stop recording' - all the commands will appear in the window. You can then tweak it as you like.

Running Macros

Simply select 'developer tab/macros'.

Select the macro and dbl-click it or....

Select an object on the spreadsheet, right click and assign macro

Named Ranges



You can assign a name to a group of cells. The cells can be in a block or they can be individual cells all over the place.

There are lots of ways to assign a name, here are a few...

Select the cells you want included

- > type in a name in the cell dropdown (top left)
- > right click -> define name -> enter a name
- > select formulas tab -> name manager -> new

You can alter or delete them from the 'name manager' on the formulas tab.

You can then use this 'name' wherever you need a cell or range of cells, eg.

```
Range("my_table") = 0
```


Messages



Simple message

Msgbox "hello world"

Question/Answer Dialog

```
answer = MsgBox("message", vbYesNo, "title")  
if answer = vbYes then
```

ActiveX Controls



Use ActiveX instead of form controls because you have more control.

You can change the colour, the cursor, picture all sorts of stuff.

put an ActiveX button on the form.

Select it, then go into VB code.

With the CommandButton selected in the left dropdown, you have lots of events in the right dropdown.

Microsoft have lots of constants for each key or button.

Here is a link to the full set...

<https://docs.microsoft.com/en-us/office/vba/Language/Reference/User-Interface-Help/keycode-constants>

KeyDown

There are two parameters here. The first is the key code. The second is the shift mask.

This shift mask can have a value between 1 and 7 to represent 3 bits.

Bit 1 - shift key pressed

Bit 2 - ctrl key pressed

Bit 3 - alt key pressed

KeyPress

This is similar to KeyDown but it uses the ascii code instead of the Microsoft Key code.

MouseMove

You get passed a Button (xlNoButton, xlPrimaryButton-left, xlSecondaryButton-right, xlMiddleButton)

You also get passed a Shift Mask - as abot and a mouse x and y pos.

MouseUp/MouseDown- same parameters as MouseMove

GotFocus, LostFocus

When someone clicks or tabs onto or off that control.

Cell/Range Properties



This page caters for a single cells (`Cells(2,3).Text`) or a range (`Range("a1:f5").Text`).

Take, for example £12.50

The `.value` property is the content of the cell with no formatting, so here it would be 12.5

The `.text` property is the content of the cell with formatting, so here it would be £12.50

The `.row` property is the row of the cell (starting at 1 from the top)

The `.column` property is the column of the cell (starting at 1 from the left)

The `.select`, this selects the cells in the range.

The `.count`, this counts the number of cells in the range - even if they are empty.

The `.address`, (absolute row, absolute column)

`MsgBox Range("a1:c6").address(true,true)` shows `a1:c6`

`MsgBox Range("a1:c6").address(false,true)` shows `$a1:$c6`

`MsgBox Range("a1:c6").address(true,false)` shows `a$1:c$6`

`MsgBox Range("a1:c6").address(false,false)` shows `a1:c6`

The `.formula`, gets or sets the formula

The `.numberformat`, gets or sets the number format

`Range("a1:a6").numberformat = "0.00%"`

The `.font` has a `.bold`, `.underline`, `italic` part

`Range("a1:c6").Font.Bold = true`

Cells object



Cells object

You can access cells by the 'range' object (which takes a cell or range of cells where the format is A1:C6, etc)

You can also access the cells like a two dimensional array...

```
Cells(4,2).Font.Bold = true
```

You can even use letters

```
Cells(4,"b").Font.Bold = true
```

This selected cell B4

You can specify just a single number, Cells(5) sets the 5th cell.

The 'Range' also has a cells object.

Range("a2:f8").Cells(2,2), this will actually update c4 because you take a2 as a starting point and move two cells to the right and two cells down.

Ranges is good for selecting areas but it relies on strings, cells use numbers but can only select a single cell at a time the trick is to use both...

Cells and Range together

```
Range(cells(3,5),cells(6,9))
```

This selected e3:i9 - using numbers / programmatically

Variables pg 1



Untyped

myVar = 34

Here excel will store it as untyped, it has no limits and will always try to return the correct value. good for quick and nasty variables - but not efficient.

Typed

Dim myVar as Boolean

uses 2 bytes of memory, can only contain true or false

Dim myVar as Integer

uses 2 bytes of memory, can contain -32768 to 32767

Dim myVar as Long

uses 4 bytes of memory, can contain -2,147,483,648 to 2,147,483,647

Dim myVar as Double

uses 8 bytes of memory can contain -1.797e308 -> 1.787e308

Dim myVar as Currency

uses 8 bytes of memory, can contain -900,000,000,000.000 to +900,000,-000,000.0000

Dim myVar as Date myVar, can contain 1st Jan 100 to 31 Dec 9999

Dim myVar as String, 10 bytes plus string length

Variables pg 2



Scope

Inside a sub
Dim hi as String

Whole module
Private hi as String

Everywhere
Public hi as String

Constants

Const myPi as double = 3.14
Private Const myPi as double = 3.14
Public Const myPi as double = 3.14

String Concatenation

MsgBox "Hello " & "world"

Sub Procedures



You can call a procedure with no parameters or 1 or many.

If a parameter is declared as ByVal - then the value of the parameter is passed in - if this is changed - it does not get changed when returning to calling procedure. If ByRef is used - then a pointer to the value is passed - in this event - any changes made to this parameter are kept when returning.

```
sub myMain()  
  a=10  
  b=20  
  call mySecondSub()  
  call myThirdSub(a)  
  call myFourthSub(b)  
  MsgBox(a & ", " & b )  
end sub
```

```
sub mySecondSub()  
end sub
```

```
sub myThirdSub(ByVal a)  
  a = 100  
end sub
```

```
sub myFourthSub(ByRef b)  
  b = 200  
end sub
```

This would display 'a' as 10 (because the parameter was defined as byVal, b would display 200 - because it was defined as ByRef.

Dates



Assigning a today (no time element)

myDate = Date

Assigning current date/time

myDate = Now

Adding/Subtracting days

myDate = Date + 5

myDate = Date - 4

Adding/Subtracting house/mins

myDate = Now + (1/24) ' adds 1 hour

myDate = Now - (3/24) ' deletes 3 hours

Text to Date and Back

myDate = cdate(Range("a1")) ' converts text to date

myDate = myDate + 1 ' adds 1 day

Range("a1") = myDate ' converts date to text

Import / Export modules/classes

You can export your modules to a .cls file (class), you can also import modules from .cls (class) files

Block Indent/Unindent

Select the block of code you want then press tab to indent, shift tab to unindent.

Bookmarks

You can set or unset bookmarks and goto them.

Immediate Window

This lets you execute code on the fly - quite useful if you want to mess around with some parameters. You can change variables (temporarily), Using debug.print will send information to the Immediate window

Locals Window

This shows all the variables (and their values) in the current window

Watch Window

You can 'watch' certain variables - this shows you them.

Debugging

You can press F8 to start a macro and stop at the first line.

Breakpoints

These toggle on / off

Tools / Macros

Shows you macros available in this book.

Last Row/Col



This is quite useful and at the same time quite annoying. There is no easy way to get the last filled in cell - so I've devoted a whole page to it.

Here is what you have to do...

- a. Identify the range of cells you want (could be whole sheet or not)
- b. Go to the bottom/right of that range of cells
- c. Go up/left and find the first non-blank cell
- d. Then take the row/col of that cell

eg.

```
cells(rows.count, 1).end(xlup).row
```

This gets the row of the bottom-most filled in cell of column 1

```
cells(1, cols.count).end(xlleft).col
```

This gets the column of the right-most filled in cell of row 1

```
cells(6, 10).end(xlleft).col
```

This gets the column of the right-most filled in cell of row 6 before column 10

You can do the same by recording a macro...

- record macro
- select a cell
- <ctrl-left cursor>
- stop recording

Referencing Cells



You can specify a range of cells in quite a few different ways...

`Cells("b5:d7")`, selects cells between b5 and d7

`Columns("J:L")`, selects columns, J, K and L

`Rows("5:8")`, selects rows 4, 6, 7 and 8

`ActiveCell.offset(3,7)`, select 3 rows down and 7 rows across from active

`Range("b" & i & ":" & "c" & j)`. if the variable 'i' is 3 and 'j' is 5, then this selects cells between b3 and c5

With and Endwith



You can save yourself a lot of time if you have lots of actions to perform on the same object...

With activecell

.Font.bold = true

.Font.italic = true

Endwith

Operators



Comparison operators...

=

<>

<

>

<=

>=

The 'IF' statement



If x=5 then msgbox("hi")

if x=5 then
elseif
else
end if

if not x=5 then
else
end if

Handy Functions



isNumeric(cell)

Returns true if cell contains a number

Goto and Labels



```
goto myEnding
```

```
myEnding:  
    MsgBox "hi"
```


Case Statements



```
Select case range("c2")  
  case 12  
    MsgBox("hi C2 is 12")  
  case is < 5  
    MsgBox("less than 5")  
Select end
```

Custom Functions



If you want to create a function which you can call from a cell.....

```
Function Kgrams(lbs)
  Kgrams = lbs * 0.451
End function
```

```
fred = Kgrams(100)
```

You can have optional parameters....

```
Function Kgrams(lbs as integer, x as string, optional y as string)
  if isMissing(y) then
    Do not use it
  else
    use it
  end if
```

Loops (for)



For loops

```
For x = 1 to 10
    Cells(x,1) = x
    Cells(x,1).bold = true
Next x
```

Function myReport

```
Lastrow = cells(rows.count, 1).end(xlup).row
For x = 2 to LastRowq
    Mymsg=MyMsg & vbnewline & cells(x,1)
Next x
MsgBox myMsg
End Function
```

You can go forwards in chunks -or even go backwards

```
For i = 2 to 400 step 20
Next i
```

```
For i=400 to 5 step -1
Next i
```

```
For i=400 to 5 step -20
Next i
```

Loops (foreach)



```
Foreach <variable> in <group>  
  if IWantToExitEarly exit for  
Next or Next <variable>
```

eg.

```
Foreach mycell in Range("mynamedrange")  
  myCell.font....  
Next mycell
```

Loops (do)



```
'infinite loop  
do  
  if IWantToExitEarly exit do  
loop
```

```
'do until condition, do not even do 1 loop unless condition is met  
do until cells(x,y) = ""  
  .. something  
  x=x+1  
loop
```

```
'do until condition, but do atleast on loop  
do  
  .. something  
  x=x+1  
loop until cells(x,y)=""
```

```
'do while condition, do not even do 1 loop unless condition is met  
do while cells(x,y) = ""  
  .. something  
  x=x+1  
loop
```

```
'do while condition, but do atleast on loop  
do  
  .. something  
  x=x+1  
loop while cells(x,y)=""
```

Input Box



```
Fred = InputBox("prompt", "title", "default", x, y, helpfulcontext)
```

Simple Report



Basically the same concept as in the forloops example - but instead of displaying a message - the output is written to a new worksheet.

This report has two worksheets, a data and a rpt sheet

```
Dim dSheet as Worksheet ' data worksheet
Dim rSheet as Worksheet ' report worksheet
```

```
Set dSheet = thisWorkbook.Sheets("data")
Set rSheet = thisWorkbook.Sheets("rpt")
```

Simply use forloops - to read from dSheets.Cells.... and write to rSheet.Cells.

To clear your report simply get the last row...

```
dLr = dSheet.Cells(rows.count, 1).End(xlup).row
dSheet.Range("a2:La"&dLr).ClearContents
```

You can hide your report tab until you have generated the report if you like - this saves you having lots of tabs visible. Simply hide your report tabs, then when you're ready - simply...

```
rSheet.Visible = true
```

You can also

```
rSheet.Visible = xlSheetHidden (same as false, can be unhidden)
```

```
rSheet.Visible = xlSheetVisible (same as true)
```

```
rSheet.Visible = xlSheetVeryHidden (cannot be unhidden)
```

When you have created your report - you may want to make that report the current worksheet by...

```
rSheet.Select
```

You can also do a print preview

```
rSheet.PrintPreview
```

You can also print it automatically (simply omit the parameters you don't want to specify)

```
rSheet.PrintOut , , 3 'sends 3 copies to the default printer
```

Errors



Sometimes you ask for input and the user cancels. Excel will show the message but will keep going.

You can stop the messages yourself by....

On error resume next

You can then perform your own checks and display your own messages

if Fred = empty then exit sub

File System Objects



This lets you access the files and directories - maybe load other spreadsheets up.

```
' create the object
```

```
Set fso = createObject("Scripting.FileSystemObject")
```

```
'get the folder we want
```

```
Set fldr = fso.GetFolder("c:\temp\")
```

```
'for each excel file...
```

```
foreach wbfile in fldr.files
```

```
  if fso.GetExtension(wbfile) = "xlsx" then
```

```
    set wb = Workbooks.open(wbfile)
```

```
    foreach ws in wb.sheets
```

```
      if ws.cells(x,y) etc...
```

```
    next ws
```

```
  end if
```

```
next wbfile
```

Events pg 1



This lets you execute code when specific events happen.

Simply open up VB Code Editor, select the left dropdown and pick workbook, select the right dropdown and select the event.... here are some.

SelectionChange Event

This would run if the current cell selection changes.

A parameter target is passed into the function, this is the newly selected range - it is passed byVal so you cannot change it.

WorksheetActivate Event

This executes when the user moves onto this sheet.

WorksheetDeactivate Event

This executes when the user moves away from the sheet.

Handy for hiding report worksheets after they have finished. Be careful though the variable 'activesheet' usually means the worksheet you are moving to - not from.

WorksheetBeforeDelete Event

This executes before the worksheet is deleted. There is no way of cancelling it, so you cannot use it as an 'are you sure' method.

WorksheetBeforeDoubleClick Event

When you double click a cell or range - it executes a default action (usually goes into edit mode on that cell).

This will execute when you double click a cell or range. You get passed a 'cancel' parameter - if you set it to true it does not perform the normal/default double click function.

WorksheetBeforeRightClick Event

Same here - a context menu normally appears - you can stop it and perform your own action.

Worksheet Calculate Event

Gets called when anything is recalculated - be careful - this gets called all the time.

Events pg 2



Disabling and Enabling events

`Application.EnableEvents = false`

`Application.EnableEvents = true`

You can use this to temporarily disable all events - so you can muck around with cells without triggering anything - then put it back.

WorksheetChange event

This is called when any text of any cell is changed.

Events pg 3



WorksheetFollowHyperLink Event

is activated when you click a hyper link, this could be a web page or another sheet. It triggers after the event - so after the web page has loaded.

The parameter to the function (target) is a hyper link.

This contains lots of information depending on the type of link. If it is a URL, then it contains the address.

You can also do lots of other stuff like, AddToFavourites.

WorksheetTableUpdate Event.

This is when you load a table from an external source not when you are typing into a table.

Intersect



Intersect is where you can check to see if the target range is within a cell range you want - a bit easier then checking start/end row cols etc.

eg. if `intersect(Range("a3:c5"),Range("b3:c5")).count > 0` then

Now the problem is that if they do not intersect - then 'nothing' is returned - and you cannot get a count of nothing.... so to test if there are any rows you need to use 'is nothing' - but negate it....

if `intersect(Range("a3:c5"),Range("g3:h5")) is nothing`
this would be true

if `not intersect(Range("a3:c5"),Range("b3:c5")) is nothing`
this would also be true

Workbook Events pt 1



To get to the Workbook events in the VB Editor, double click the workbook item (green icon to the left) - select 'workbook' in the left dropdown, then the actual event in the right drop down.

You can debug these functions by pressing F8 - you don't need to close and reopen the book to activate the openbook event.

WorkbookOpen event

This executes as soon as the workbook opens, you can use it to clear the last report or set active sheets, hide tabs or give a greeting.

WorkbookActivate and WorkbookDeactivate

These trigger when you have multiple workbooks and you move from one to another. It does not work if you click away from Excel and back.

WorkbookSave

This executes before it saves. You can cancel the save by setting 'cancel' to true.

WorkbookAfterSave

Runs after it has saved.

It executes even if cancel was set to true in the WorkbookSave.

The success parameter is true if they wanted to save - False if they did not want to save.

WorkbookBeforeClose

Triggers when the user closes a book - you do get to cancel it.

WorkbookBeforePrint

You can stop them printing if you want.

WorkbookNewChart

Executes when user adds a new chart to any sheet in the workbook. This is useful if you want to preset colours or axis names.

WorkbookNewSheet

This activates when the user adds a new sheet.

Workbook Events pt 2



WorkbookWindowActivate / WorkbookWindowDeactivate

Triggers when you go from workbook to workbook.

WorkbookWindowResize

Triggers when the workbook dimensions change.

Status Bar



Basically, this is a single line at the bottom of the Excel Window.

You can display messages there or clear them.

`Application.StatusBar = "Hi There"`

To clear it set it to "" or false

Another tip is to create a label, resize it so that it is bigger than your control which sets the status bar.

Set its background to transparent.

Set the text to empty.

Send it backward.

On mouseMove you can clear the status bar, this means that it gets set during the button but cleared when you come off the button.

Checkbox



Simply add this control to your sheet.

In your code you can...

```
if me.Checkbox1 then  
end if
```

LinkedCell property

This is where you can specify a cell to get its value (true or false) from.

Note, changing the value of the cell does not activate the onClick event.

If you change this cell, it affects the control, if you change the control it affects the cell.

If the cell has invalid items in it, eg. tru instead of true, then the control is temporarily greyed out until you click the control or enter a valid value.

OptionsButton



This is basically a radiobutton.

If you place lots on a sheet, each time you click one, all the others become unclicked.

If you have multiple sets of radio buttons, you can group them together. When they are grouped, click on one only resets the buttons in the same group.

To group them, simply select multipleoptions, click properties and set the group name.

Aligning and Grouping



You can select a group of controls, then click the alignment button.

You can also click left or right to nudge the controls in that direction.

You can also group them together. This means that wne you first click the control - it selects the whole group. If you second click - you select an item of that group.

SpinButton



This gives you two buttons, there is a spinup event and a spindown button - each with its own event.

If you resize the button it will attempt to determine the correct direction: up/down; left/right. This can be controlled in the properties to force it in one direction or the other.

PrintObject in properties means that it will show on a printout (or not)

DbtClick in design mode to show the OnChange event.

If you use a SpinButton on a cell and it is empty and you add/subtract, it will assume the value is zero this is because a cell can have a numeric format.

A textBox though will always assume a text value - so if it is empty, you'll get an error - you could use 'On error resume next' - to ignore errors.

Combobox



The property "ListFillRange", is the named range or actual cell range for that list box.

You can have multiple columns, it displays then in the list - but only uses the value of the first column.

You specify the columncount in the properties and also column widths ('35;35')

This will show two columns each with 35 points.

If you have a named range with two columns they will automatically show.

DynamicRanges



You can use these with ComboBoxes or anything really.

Sometimes you want a range to automatically increase - you do this by using the 'offset' function/

Go into named range manager alt i, n, d

Enter

= offset(startcell, offsetrow, offsetcol, numberofrows, number of cols)

If you have your list in column A, with column headings it might look like this

= offset(sheet1!\$A\$1, 1, 0, counta(sheet1!\$A:\$A)-1, 1)

count a will count the number of filled in cells of the range - so where we have selected the whole column A.

Our list has column headings so we subtract 1 from the count.