

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 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
```

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



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

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

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 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
```

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.