Chapter Review - Custom Formulas

The VB Editor

	The VB Editor is used to write our code, and can be accessed via the developer ribbon, or the ALT + F11 shortcut.
Front End vs Back End	In the context of VBA, the VBA editor refers to the backend, whilst the traditional Excel view can be thought of as the front end.
Sheets Vs Modules	Wherever possible, write your code in modules. People can delete sheets!
"ThisWorkbook Code"	This special module is used later on when we starting dealing with workbook events. For example we want code to run when the workbook opens.

Error Tracking (Debugging)

The Immediate Window	The immediate window will help us track down errors in our code.
The Locals Window	The locals window helps us interrogate our data, and helps us track down errors in our code.
The Properties Window	The properties window helps us change the characteristics of objects, such as their name and visibility.

Working in VBA

Function NameHere()	A function in VBA is like a formula in Excel. It returns an answer based on a series of defined steps.
Variables	You can create and assign a value to a variable simply by writing your desired name followed by it's value. They store one piece of information. For example: AmountInUSD = 12
Strings	Variables don't have to be numbers. They can be strings of text too, which can be combined or split. String1 = "a". String2 = "x" String1 & String2 = "ax"
Objects	Objects are things, just like cars, bridges etc. Workbooks, Sheets, Tables, Cells, Ranges etc are all examples of objects. Objects can often be broken down into smaller parts, and usually have properties that we can change such as their size, color & value etc.
Is it an object?	If a 'thing' can be broken down into constituent parts, or has properties that you can change, or holds more than one piece of information, it's almost certainly an object.
Comments	We can write comments by starting a line with a single apostrophe '. These are not interpreted by the code and simply help a reader to understand or search your code. They are REALLY important! Comments will appear in **green** to help you distinguish them from your code.

Variable Name Conventions

Spaces	Notice that when we write AmountInUSD, or any other variable name, we cannot include spaces. Underscores are acceptable, however.
Camel Case	Notice that the first letters of each word in AmountInUSD are capitalized. This is to help readability.
Consistency	There are no set conventions on how to name your variables, or what format to use. The key is to be clear, and consistent, which will help readers of your code immensely!

Working with Worksheets

Sheet Code Names	Are a more robust way of referencing sheets, and won't be changed when someone changes your sheet names.
Values from a Sheet	You can assign a value from a worksheet cell to a variable using this syntax: Var_Name = ws_CodeName.range("rangename").value
Application.Volatile	Allows your function to auto-update on the worksheet if Excel's automatic calculations are activated.

Chapter Review – Errors & Exceptions

Declaring Variables

Declaring Variables helps prevent typos and helps Excel to use memory more efficiently.

Option Explicit	We should ALWAYS use this line of code at the top of our VBA editor, to force us to declare all of our variables.
Dim	Dim is short for 'Dimension' and is used to declare new variables. It must be followed by a variable name and type. For example: 'Dim String1 as String'

Variable Types

Integer	Any whole number
Double	Any number with a decimal
Boolean	True or False
String	Any text, including numbers formatted as text
Variant	Use this type sparingly, when it's not certain if the variable will be of one type

Error Tracking (Debugging)

Debugging is the process of working out why you code isn't working the way you intended.

Breakpoints	Help you stop the code while it is running, so you can investigate errors in your code.
Yellow Code Line	When your code reaches a breakpoint, it'll turn yellow. You can move the yellow line in the left margin to help decide which line to run next.
F5 & F8	These shortcuts help you to either continue running, or run one line of your code.
Debug.Print	Prints values or strings into the Immediate window to help interrogate your code.
The Locals Window	The locals window can be used to quickly check what value your variables currently hold.
Variable Hovering	Hovering over variables in your code will reveal their current value.

Error Handling

IF – THEN statements	If statements help us change the outcome of the code, depending on a specific condition. For example: '**If** *x > y* **then** *Message = "X is greater than Y"*'
On Error Resume Next	Sometimes it's helpful to force VBA to continue working, even after an error occurs. You can combine the following lines to deal with an error. Always reset the error handling using line 4.

1. On Error Resume Next
2. Code with potential error here
3. If Err > 0 then error handling code here
4. On Error Goto 0

Code Navigation

Usually, code is executed one line at a time, in the order in which they are written, but there are a few ways we can change that logic if necessary. It's also fairly normal for one thing to happen per line, but there's a handy way to perform more than one short step on a single line that I've described below.

Exit Function	This short command allows you to exit the code early.
Step 1: Step 2	The colon : allows you to follow one code step with another, on the same line.
Function Answer	When creating your own functions for use in a worksheet, if the answer is not available due to an error, you can set the result of the function to a message. This way, the user knows exactly why the formula has failed. Usually if his has happened, you For example *'FunctionName = "ExchangeRateMissing": Exit Function'*

Chapter Review – Finalize and Publish Functions

IF THEN, ELSE, END IF

This is a slightly more complex syntax for writing IF statements, and allows us to define steps for both outcomes of the tested condition, not just the true one.

The IF statement tests a binary condition, meaning that it's outcome should be true, or false. For example if a > b or if C + Y > 10. The full syntax is as follows:

```
If CONDITION THEN
        CODE HERE FOR TRUE CONDITION
ELSE
        CODE HERE FOR FALSE CONDITION
END IF
```

Code Formatting

Making your code easy to read and navigate should be a priority. Whilst comments and clear variable names are vital, there are also other ways:

Indentation

Use TAB, or SHIFT TAB to manage the amount of indentation on a line of code. For example in the IF statement above we indented each of the outcomes to show that it belongs inside each outcome of the IF statement.

Working with Add-ins

Add-ins let you share code with colleagues, allowing you to share functions and useful formulas.

Saving an Add-in

Save your file as a xlam extension file to make it into an add-in. Save it somewhere central that all your team have access to.

Loading an Add-in

Use the File -> Options -> Add-ins - > Manage -> Excel Add-ins -> OK -> and then browse to find the add-in in the central location.

Avoiding Code Overlap

Once you've installed an add-in on your Excel, you'll need to make sure that you don't have that same code in your workbook, as then you'll have repeated functions, which will confuse Excel. It will either result in errors, or double calculation.

Chapter Review – Events & Automation (Part 1)

MsgBox

A msgbox allows you to send pop-up message to your user. Use them sparingly as usually the user has to click something each time it pops up. They can also be useful to test your code.

Sub (Subroutine)

A subroutine is very similar to a function in that it executes some code. The difference is that a function always returns an answer. A sub simply executes code. For example:

```
Sub UpdateReminder()
    Msgbox "Please remember to update the FOREX assumptions."
End Sub
```

Workbook Events

Events allow you to make code run automatically, when a specific thing happens. Workbook events like open, close and save file must be written in the ThisWorkbook module.

```
Private Sub Workbook_Open()
    Call UpdateReminder
End Sub
```

Calling a Sub

Calling a sub allows you to call one piece of code, from another. You can even pass data between them. For example, in the above examples, the workbook open routine **calls** the Update reminder routine.

Calling a Function

Similarly, you can call a function from inside another sub or function. You can even pass variables into it. For example in the below example, the MathCalculation routine calls another function to find an answer.

This helps keeps our code neat and tidy, and means we can re-use the function elsewhere:

```
Sub MathCalculations()
    StartAtNum = 3
    EndAtNum = 5
    Answer = AddAllConsecutiveNumbers(StartAtNum, EndAtNum)
End Sub
Function AddAllConsecutiveNumbers(StartAtNum as Integer, EndAtNum as Integer)
    Answer = ( ( EndAtNum + StartAtNum ) * ( EndAtNum – StartAtNum + 1 ) ) / 2
    AddAllConsecutiveNumbers = Answer
End Function
```

Working with Other Workbooks

Central Assumptions File

It can be helpful to keep team assumptions in a central location. If all files reference this central file, you can be more certain you're all working with the same data.

Ranged Names

If you're repeating range names in your central file, be sure to name them slightly differently to avoid any chance of confusion.

Range Reference

You can reference a range from a closed workbook by using the following syntax:

```
Value = ExecuteExcel4Macro( DataLocationRef )
    where DataLocationRef = "'" & FilePath & FileName & "'!" & RangeName
```

Caution: Be careful to include the single quotation marks and exclamation.

File Paths

A file path must end in a \, so you can use an IF statement to check this by checking the right hand character of the FilePath string.

Chapter Review – Events & Automation (Part 2)

RED Code

When your code turns red, VBA is telling you that line is incomplete. The technical term for this is a Syntax Error, which is another way of saying that your code doesn't make sense.

Arrays

In it's most basic form, an array is like a grid of variables. Think of it like a table of data, organised into rows and columns. In this chapter we covered a special kind of array, which was simply a list. You can think of this like a grid of 1 column, and n rows.

To manually declare a list in VBA, you'll need to use this syntax, although in the next chapter we'll cover the more common method of declaring an array.

Declaring a list array

```
Dim Currencies as Variant
```

Creating the list array

You can create the list of values manually, as a list, using the following syntax:

```
Currencies = array("EUR", "GBP", "CAD", "CHF")
```

Note: This is a unique case of the array type, and usually you would try to avoid using the Variant type.

For Each Loop

A for each loop can be used to cycle through all the items in a list or array. For example:

```
Dim CurrencyCode as variant
For Each CurrencyCode in Currencies
        'Repeat a set of actions
Next CurrencyCode
```

Chapter Review – Working With Arrays (Part 1)

Getting Data From A Workbook

Open another workbook

To work with and extract multiple values at once from another workbook, you'll have to open it. To do this, you'll need to define a workbook object for both the current workbook, and the one you want to open. That way, you can specify to VBA which one you want to modify each time you change or select something.

```
'Declaring the Current Workbook (The one your code is in)
Dim ThisWB As Workbook      'This declares a workbook but is not yet assigned
Set ThisWB = ActiveWorkbook      'This assigns the activeworkbook as This WB.
'Note: This should be done at the start of your code, to ensure that the activeworkbook is the one from which your code is being run. If your code selects another workbook before this is done, that other workbook becomes the activeworkbook.
```

```
'Declaring the Other Workbook (The one you want to get data from)
Dim WB_assumptions As Workbook
```

Since the other workbook is not yet open, we can't assign it to the workbook object above. But what we can do is to assign it at the same time as opening it. We can do this using the Open method of the workbooks object, which just Opens our chosen workbook.

```
Set WB_assumptions = Workbooks.Open(FilePath & FileName)
```

To activate a workbook

Each time you read or write anything from either file, you should ensure that the correct workbook is activated. You can do this with the following code. Note that you can do this with or without the sheet reference. If you're going to read or write anything, I recommend you be as specific as possible.

```
WB_assumptions.Activate
WB_assumptions.Worksheets("ForexHistory").Activate
ThisWB.Activate
ThisWB.Worksheets("ForexHistory").Activate
```

Arrays

An array is a set of numbers, arranged into a grid like layout. This grid is typically two dimensional, which is great for representing data from a worksheet for example. Note that it's possible to have an array with more than two dimensions.

Declaring an array

You can identify a variable as an array by using the double brackets () as follows:

```
Dim Array1() as integer
```

In this case we've created an array, in which all the values in all the rows and columns are integers.

Assigning values

You can assign values from a range to an array using the following code:

```
Array1() = Range("A1:B5")
```

This will give you an array of five rows and two columns.

Referencing values

If we want to get the value of one item within the array, we can reference it's location using row and column number. For example, Array1(1,2) would return the value of cell B1 from the above example.

Careful: Sometimes VBA will make the first row or column of an array, row 0, or column 0. Use the locals window to check this when you create an array. This is called Zero Indexing. It can be confusing, but it's actually standard practice in many coding languages. In this case row 1, column 1 would be Array1(0,0).

Chapter Review – Working With Arrays (Part 2)

Working with Tables and Ranges

To assign the data rows of a table to an array, you can use the following syntax:

Array1() = WB_assumptions.Worksheets("Forexhistory").ListObjects("ForexHistoryTable").DataBodyRange.Value

Although we didn't directly cover it in the course, you can also assign values from a regular range using the following:

Array1() = WB_assumptions.Worksheets("Forexhistory").Range("A1:B20").Value

Pasting Values From an Array

As with a variable, it's easy to paste data from an array to a range. We just use the reverse of the above syntax:

To assign the data rows of a table to an array, you can use the following syntax:

WB_assumptions.Worksheets("Forexhistory").ListObjects("ForexHistoryTable").DataBodyRange.Value = Array1()

Although we didn't directly cover it in the course, you can also paste values to a regular range using the following:

WB_assumptions.Worksheets("Forexhistory").Range("A1:B20").Value = Array1()

Checking the Size of An Array

Ubound() The Ubound function will return an answer of array size, if you give it an array, and specify the dimension you are referring to. Typically, for a two dimensional array with values taken from the worksheet, the 1st dimension will be the rows, and the 2nd dimension will be the columns. Use the Locals window to check the layout of your array.

RowCount = Ubound(Array1, **1**) ' The 1 refers to the 1st dimension, which is usually rows.

Resizing a Table to fit the Contents of the array

Before you can paste your new data to the table, you'll need to resize the table.

1) First, declare a range variable called "NewTableRange" (the name doesn't matter)

2) Set NewTableRange equal to a resized version of your current table. This just defines a range. It doesn't change anything.

3) Now use the resize method on your table, and give it the NewTableRange as the new size.

```
Dim NewTableRange as Range
Set NewTableRange = Range("ForexDataTable[#All]").Resize(RowCount + 1, ColCount)
ws_ForexData.ListObjects("ForexDataTable").Resize NewTableRange
```

To Create a For Loop

For Loops can be used to repeat some steps a defined or variable number of times.

```
Dim RowNum as Integer
For RowNum = 1 To RowCount
    'steps to repeat go here.
    'ensure you reference RowNum to get the current value for each loop. Eg:
     If ForexHistoryArray(RowNum, 2) = "" Then ErrorMsg = "RateBlank"
Next RowNum
```