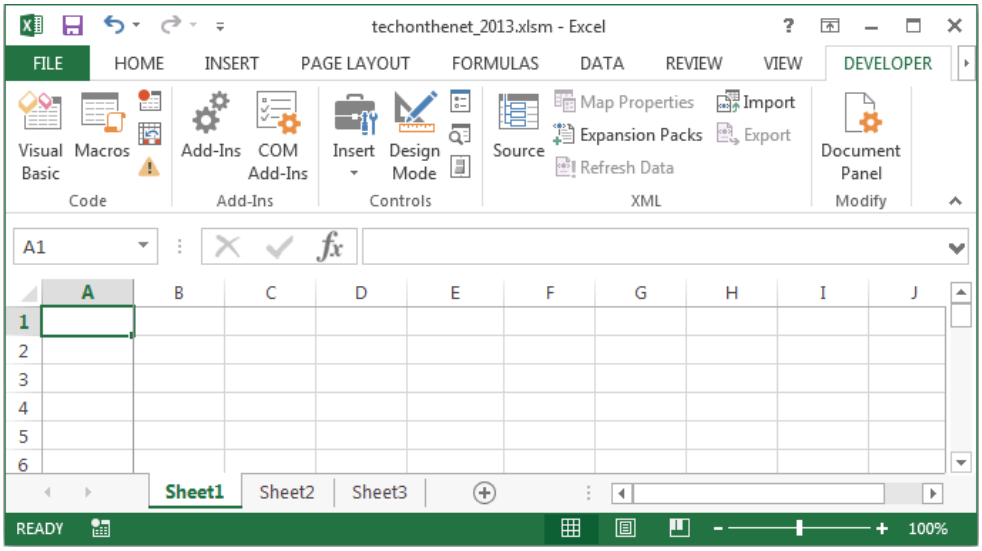
Introduction to Visual Basic for Applications

Course Notes

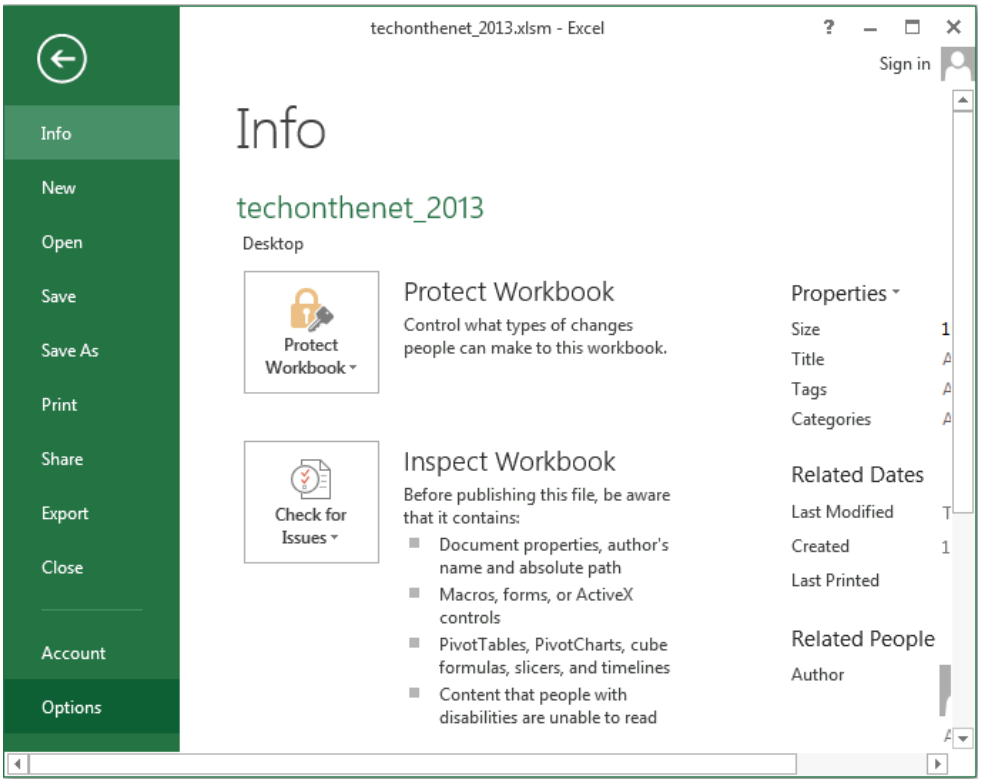
Getting Started

The first thing you will need to do is to get the 'Developer' ribbon displayed on Excel.

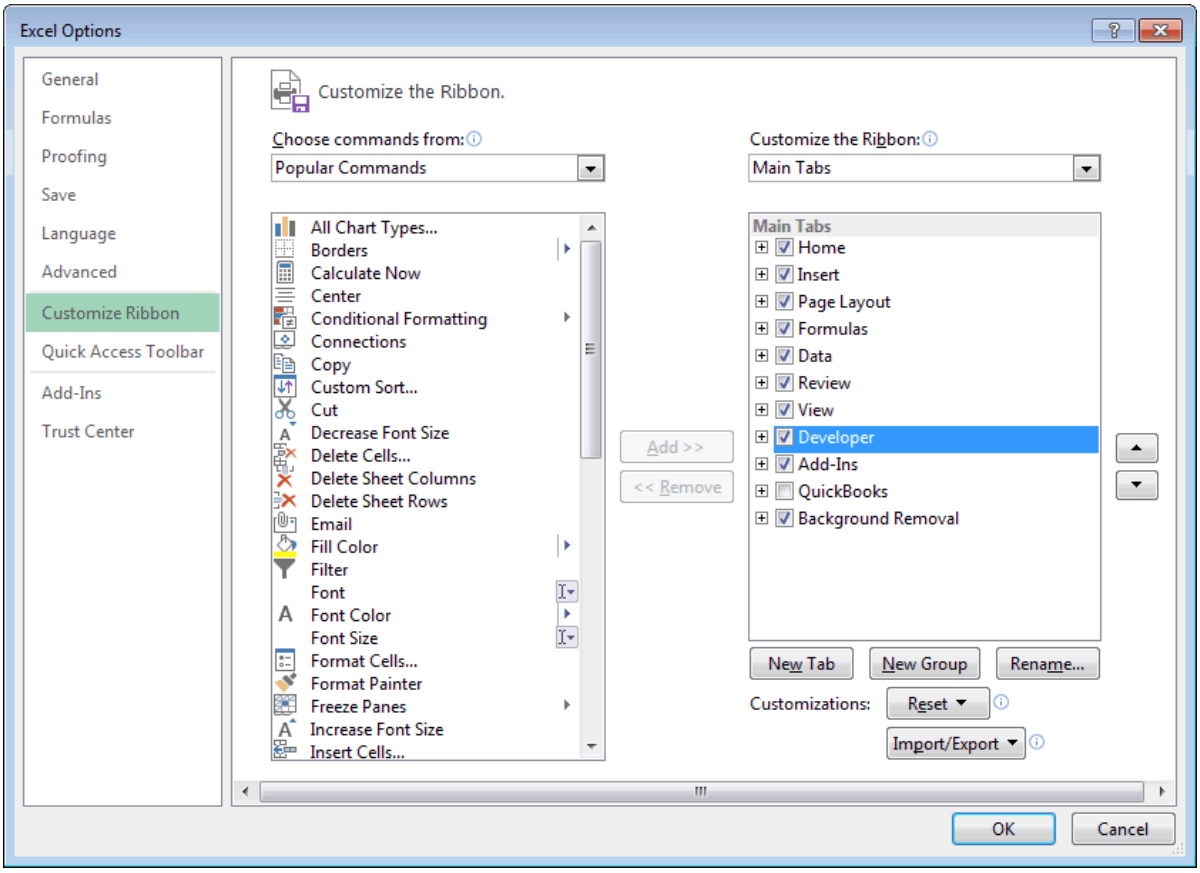
Open up Excel and Click on the *File* tab



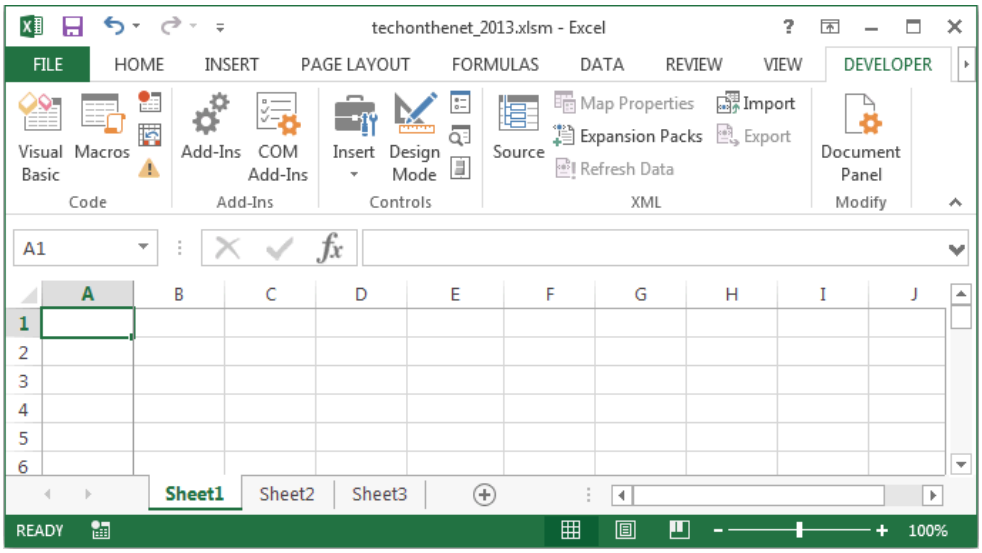
Under the file tab select *Options*



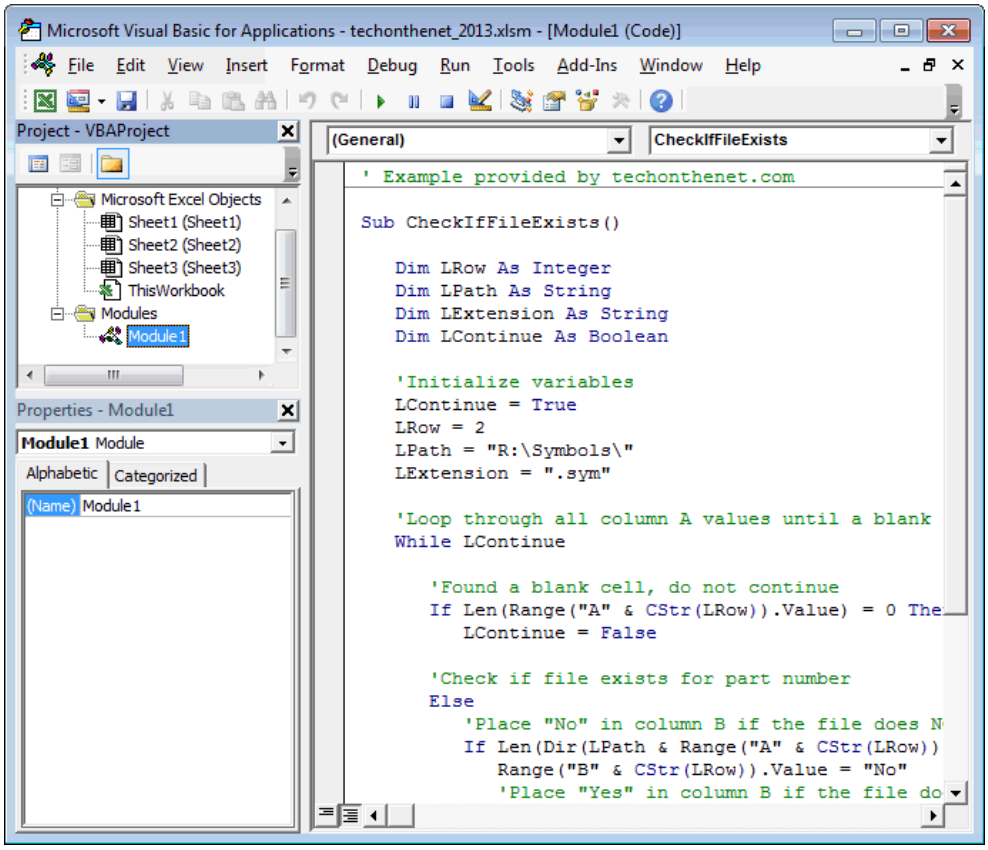
Select *Customize Ribbon* and then the check box for Developer and press OK



You will then see the *DEVELOPER* ribbon on Excel, which you should select.



Then by clicking on the *Visual Basic* button, you will see the Visual Basic Editor as below



This is where you write your code

Your First Program

Select the excel button in the top left hand corner of the visual basic screen to bring back the excel spreadsheet

Click on the *Insert* Button and then select Command Button from the ActiveX Controls

Click on the surface of the workbook where you want the button to appear

Design Mode should now be selected. If it is then clicking on the button (you have just created) will select it. If *Design Mode* is off then clicking on the button will operate it and you will see the normal button behaviour

Make sure *Design Mode* is selected and double click the button you have made. This takes you back into the Visual Basic editor. This time some code is written for you.

```
Private Sub CommandButton1_Click()
End Sub
```

Between the two lines of code type the following code:

```
  MsgBox ("Hello World")
```

Then go back to excel, switch *Design Mode* OFF and click on your button and see what happens

A message box should appear with the message *"Hello World"*

Congratulations - you have written your first computer program

Modules and Functions

In the VBA editor find the window that says 'Project VBAProject'. This is where you control the whole project from.

As we wrote out code having double clicked the button on the excel spreadsheet then the code was stored 'behind the button relating to that sheet'

If you expand Microsoft Excel Objects and then double click on Sheet1 (sheet1) you will see the code you have entered

However suppose we want to write some code that all the sheets in the spreadsheet can 'see'. In this case we need to write the code in a Module.

Right click on the 'Project VBAProject' window and select *Insert* > *Module*. A new module called Module1 will be created.

Type the following code:

```
function square_number(x as double) as double
  answer = x * x
  square_number = answer
 end function
```

Now type the numbers 1 to 10 into cells A1 to A10 and then put the formula *=square_number(A1)* into cell B2 and drag it down to B10. You will see that your formula has worked and calculated the squares of each of the numbers from 1 to 10.

Can you understand what each of the lines in the above program are doing

Locating Cells

Using a function argument is one way we can send information from excel to VBA and visa-versa. However there are many others

Clear your spreadsheet (but leave your function) and label the cell C3 as "topleft". Also label the range of cells from C3 to F6 as "matrix"

Now enter some random numbers into each of the cells in the range: "matrix"

Put a new button on the screen, double click it and enter the following code

```
cells(3,9)=[topleft]
cells(4,9)=[topleft].offset(2,1)
cells(5,9)=[matrix].cells(1,1)
cells(6,9)=[matrix].cells(2,1)
cells(7,9)=cells(3,3)
cells(8,9)=Range("matrix").cells(2,1)
cells(9,9)=Range("topleft").offset(2,1)
cells(10,9)=Range("matrix").columns.count
cells(11,9)=Range("matrix").rows.count
```

Can you see what each of these instruction is doing when you run the code

Try inserting a column of cells between A and B. Then run the code again. Can you explain what has happened

Variable Types

We have already seen `double` used as a variable definition

There are many different variable types in VBA but the main ones we need to worry about are:

Type	Contains	Range
Integer	Integers	-32,768 through 32,767
Long	Integers	-2,147,483,648 through 2,147,483,647
Double	Floating point numbers	+/-1.79769313486231570E+308 through +/-4.94065645841246544E-324
String	Words and phrases	0 to approximately 2 billion Unicode characters
Boolean	True or false	True or false

A more detailed description is given on the msdn website

VBA is generally quite good at guessing which type of variable you are using

However it is best practice to start all modules with the line `option explicit`, which forces you to define each of your variables before using them

Go back to Module1 and type `option explicit` above your function.

Now try and use your function: `square_number` again.

You will see that your function no longer works. The solution is to type:

```
Dim answer as double
```

on the line below the function declaration

what do you think this line of code is doing

Function and Subs

We have already seen one example of a function and a sub

Can you explain the difference

If you want a set of tasks performing but do not want a result returned then use a sub.

Just because they are automatically created when you create a button does not mean that you can only use them behind a button you can use them any other time you wish.

Functions can be used to return values to excel or to other functions or to subs. We will see examples of this under loops.

Both functions and subs can have parameters as we saw with our square_number function earlier on

```
function square_number(x as double) as double
```

`x` is sent to the function and is defined as type double. This means that it does not have to be redefined even if you are using "option explicit"

Functions: other issues

Public or Private

Functions can be Public or Private. A public function is visible outside of its own module whereas a private one can only be seen within the module in which it is written. Public is the default and this is what we use most of the time as we wish to use functions from excel and from worksheet modules.

Using excel functions in VBA

Some functions such as 'NormSDist' are excel functions but not VBA functions. These can be accessed from VBA by using the following syntax:

```
p = Application.NormSDist(2)
```

Is the probability that a normal random variable is less than 2

```
Application.Pi()
```

returns a very accurate value of π

Log vs Ln

In VBA we use the function `Log` to give us the natural logarithm (to the base e). Ln is not defined in VBA

RAND() excel vs RND() VBA

Random numbers in VBA are generated with the function `RND()`. This produces a random number between 0 and 1 each time it is called

Rounding to integer values

If you wish to round down a floating point number to the integer value below use the function `int()`

Inline functions

VBA can also take a function name as an argument to a function and then use that function within another function. We will see this in operation later in the course. The syntax is as follows:

```
x = Run ("square_number", 6)
```

Runs the `square_number` function with the parameter 6. This feature will be used later in the course when we write integration routines to which we which to pass different functions to integrate

Write a function which generates numbers randomly from a Weibull distribution

Remember the pdf is $f(x) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$, for x >= 0 and 0 elsewhere

Loops

For Next Loop

Suppose we wanted to produce a simple function to calculate factorials

Using the formula $factorial(x) = \prod_{r=1}^{x} r$

We will need a loop to calculate this function

In VBA we code this type of loop with a For Next loop like so:

```
prod = 1
for r = 1 to x
  prod = prod * r
 next r
```

See if you can put together a whole function and use it in Excel

Do Loops

Sometimes we don't know in advance how many times we wish to loop through the code until we have looped through it

In these circumstances we can use a Do Loop, where the condition can either be at the start or the end of the loop

```
Do {while | Until} condition
  statement1
  statement2
  ...
 Loop
```

```
Do
  statement1
  statement2
  ...
 Loop {while | Until} condition
```

Use a Do Loop to find the first triangle number which exceeds 1000

Arrays

You can set up array in VBA and you can also pass arrays from excel ranges

Consider the following code:

```
Function sum_matrix(m) As Double
Dim sum As Double
Dim i As Integer
Dim j As Integer
sum = 0
For i = 1 To m.Rows.Count
  For j = 1 To m.Columns.Count
    sum = sum + m(i, j)
   Next j
 Next i
sum_matrix = sum
End Function
```

Type some numbers into a range of cells in Excel then put the function:

=sum_matrix("B3:D7") or whatever your range of cells is and you will see the result is that the your range has been converted into an array in VBA and all the elements have been summed

If you need to define a new matrix directly in VBA then you need the following code

```
Dim new_matrix() as double
ReDim new_matrix(n,n)
```

In this code the `Dim` statement sets up the matrix (but with no dimensions) and says the values are `double`. The second statement then resizes the output array to the size required. There is no particular logic as to why the `Dim` and `ReDim` statements cannot be combined - it is just how VBA works

You may also wish to write some code to multiply two matrices and return the answer matrix to excel. You will need to do this with a sub as function cannot communicate directly with an excel spreadsheet. Here is some example code which does this.

Conditional Statements

`If` statements can be used to execute some code on one condition and other code on another condition

The following code will return true if x is divisible by d and false otherwise

Note the use of a boolean variable as a return value and the `Mod` function, which provides modulo arithmetic

```
Function is_div(x, d As Integer) As Boolean
  If x Mod d = 0 Then
    is_div = True
   Else
    is_div = False
   End If
End Function
```

Use this function to test which numbers in your matrix are divisible by 3.

You do not need to have an else statement in the `if` block if nothing actually happens in the alternative hypothesis

the following code is perfectly acceptable:

```
Function is_div(x, d As Integer) As Boolean
  is_div = false
  If x Mod d = 0 Then
    is_div = True
   End If
End Function
```

You can also put a single `if` statement all on one line if you wish

```
Function is_div(x, d As Integer) As Boolean
  is_div = false
  If x Mod d = 0 Then is_div = True
End Function
```

Notice that you do not then need the `End if` statement

Excel Objects

Excel VBA is an object-oriented language, like C++

Each Excel object represents a feature or a piece of functionality in Excel.

Examples: Workbooks, Worksheets, Ranges, Charts and even Excel itself (the Application object)

Your programs in VBA manipulate the properties and apply methods to Excel objects.

Each Excel object has values or properties associated with it, and also functions called 'methods' that operate on the object and its values and properties.

The objects in Excel can either be singular, or come in collections, and are organized in tree-like hierarchy.

At the top level is the 'Excel program' itself; it is called the 'Application' object. It is a singular object: Excel has only one Application object.

Then come the 'workbooks', stored in the 'Workbooks Collection' which consists of all open workbooks.

Each workbook has a collection of worksheets or Sheets (all the sheets in a workbook).

Each sheet has the collection of cells, called 'Ranges' and possibly 'Charts' in the 'Chart collection'.

Assessing parts uses the dot notation:

```
Application.Ln("A1")
Application.Workbooks("Book1.xls")
Sheets("Sheet1").Range("F4:F9")
```

You don't always need to the full hierarchy of names if the object with the 'focus', for example the current or 'Active sheet', is intended.

```
Sheets("Sheet1").Range("F4:F9")
ActiveWorkbook.Sheets("Sheet1").Range("F4:F9")
```

or even

```
ActiveSheet.Range("F4:F9")
Range("F4:F9")
```

Object properties:

Objects have properties, which could be attributes of the object, or values or settings of the object.

```
Application.ScreenUpdating = False
Range("A1").Name = "Year"
Range("A1").Value = 2006
```

Object methods:

They also have methods, which are a set of predefined activities that an object can carry out or have applied to it.

```
Sheets("Sheet1").Delete
Range("A1:A10").Select
Range("A1:A10").Copy
Range("B1:B10").PasteSpecial
```

There are many web sites offering VBA help just Google what you want to know and the answers should be there

Exercises

Exercise 1

How many prime numbers are there between 1000 and 2000?

Solution

0:00

If your browser does not play the video you can download the mp4 file here by using ALT + click on the link

Exercise 2

The probability of a student passing a test is 72%. What is the probability that more than 40 students out of a class of 60 pass the test?

Solution

0:00

You borrow £200,000 to buy a house. Your monthly mortgage payments are £1,000. What annual rate of interest are you being charged?

If your browser does not play the video you can download the mp4 file here by using ALT + click on the link

Solution

These e-lectures cover the whole of chapter one of the course (i.e. the 2 week induction). They are not required for years 2021 onwards but can still be used as a student resource.

Here is the SIR model we produced in the online lecture (2020)

0:00

If your browser does not play the video you can download the mp4 file here by using ALT + click on the link