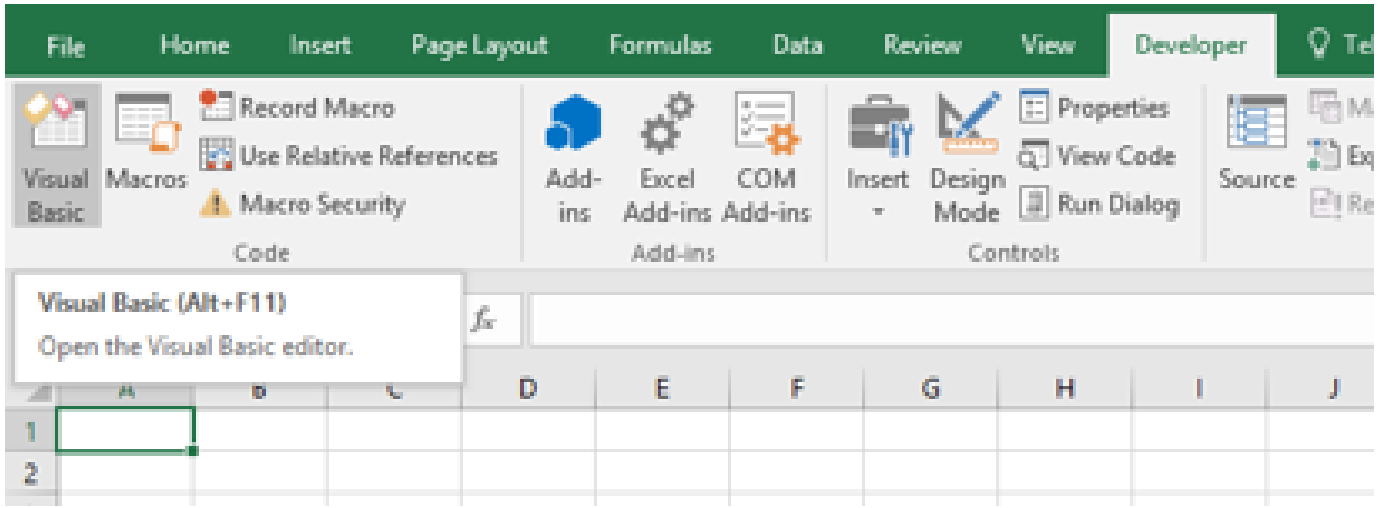
VBA
(Visual Basic for Applications)

Session 1

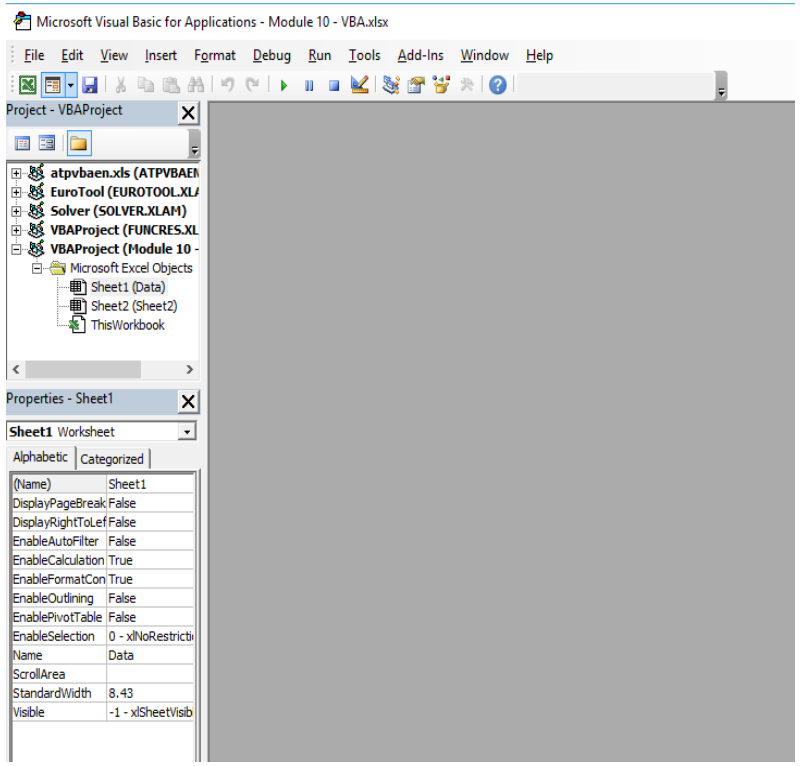
Visual Basic Editor
Record a Macro
Write a VBA Macro
Declaring a Variable

Visual Basic Editor

Go to ‘Visual Basic’ in ‘Developer’ tab



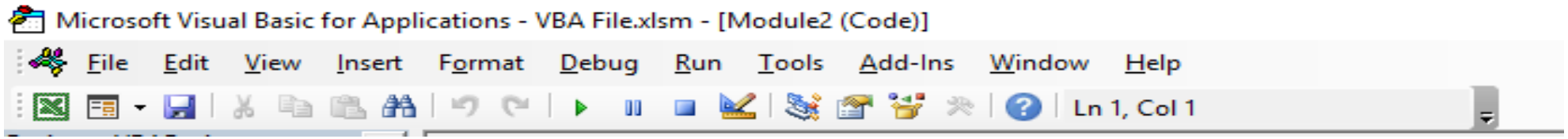
When you click on it, a new Window will open which is the Visual Basic Editor



Visual Basic Editor Components

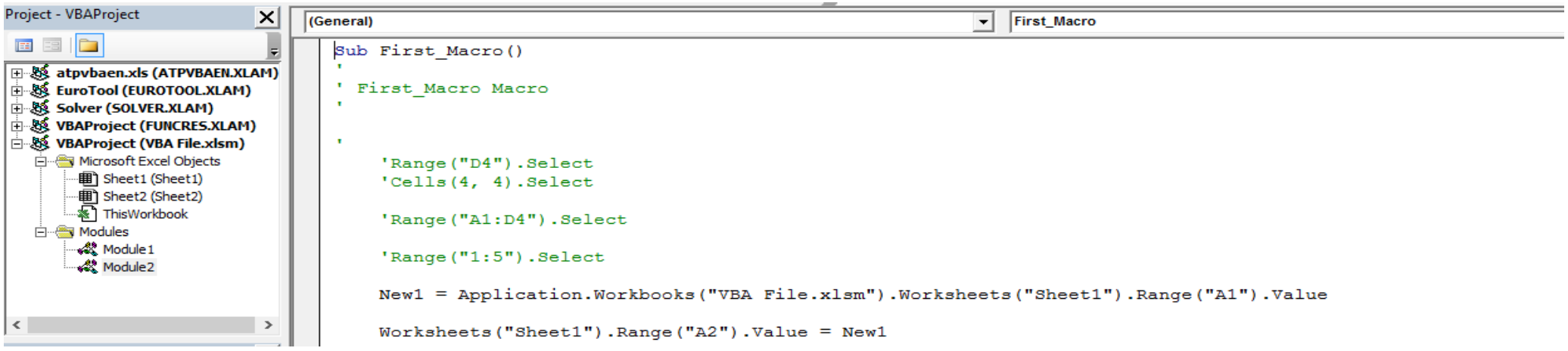
Menu and Tool Bar

These bars consists of various commands that you can use to do things in VBA.

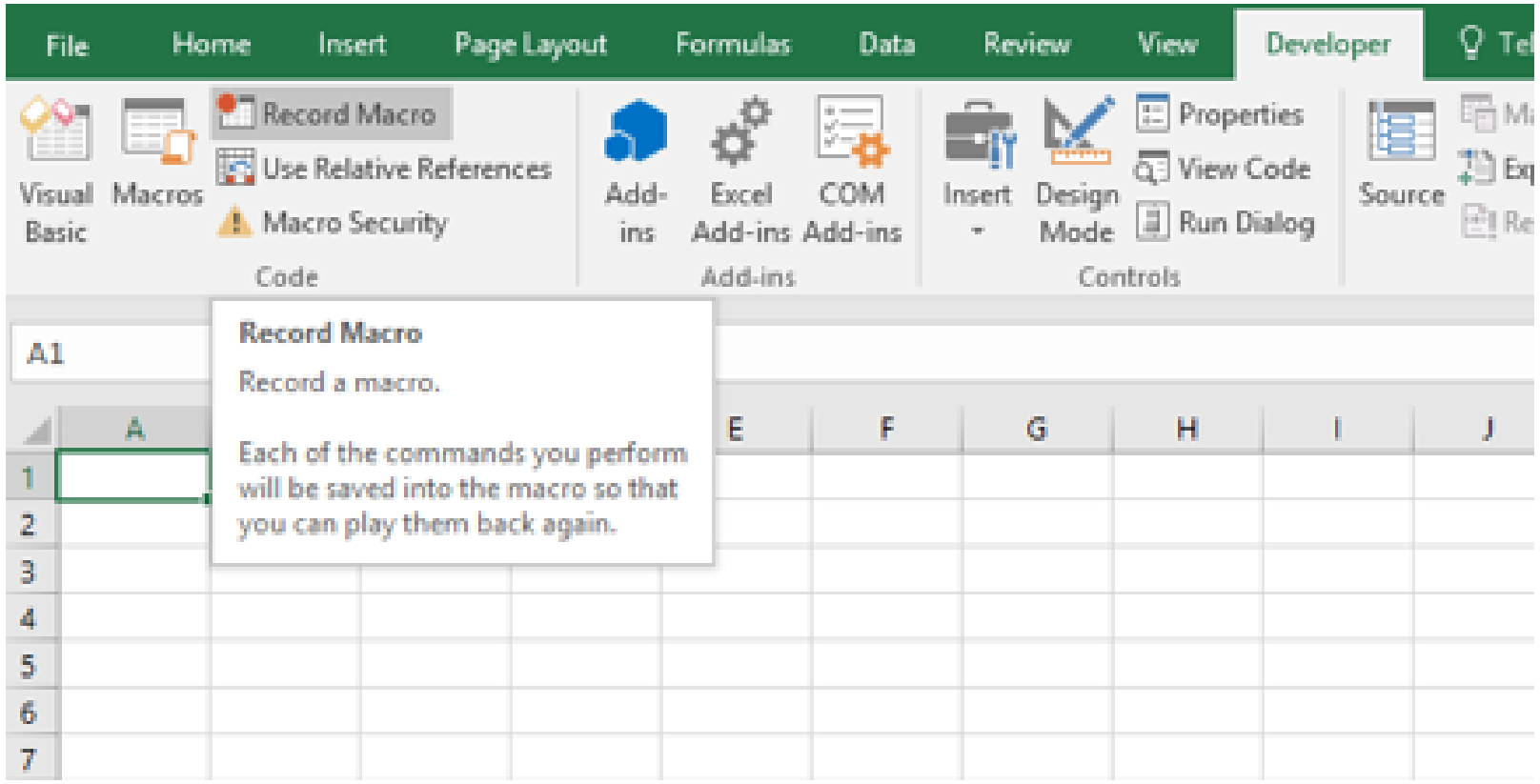


Project Window/Project Explorer

The top right section gives details like which all files are open and within those files how many sheets are present. For each sheet, workbook, you have different code modules. It is beneficial to write programs in Modules rather than individual sheets as long as the macros are not just for that particular sheet - modules can be exported imported to other workbooks; deleting a sheet deletes the corresponding macros.
Write macros in Sheet modules or 'ThisWorkbook' in cases of event handler codes which we will study later.



Recording a macro



Refer cell(s)

Using Range
Examples:
Range("A1:B6")
Range("A1:B6,D5:M8")
Range("A:E"), for entire columns
Range("1:3"), for entire rows

Using Cells - Cells(rownumber, columnnumber)
Examples:
Cells(2,4) is same as Range(“D2”)

Ranges and Cells can also be used together
Example:
Range(Cells(1,1), Cells(4,4)) is same as Range(“A1:D4")

Selecting cell(s)

Selecting one cell:
Range(“cellname”).Select

Selecting multiple cells:
Range(“cellname1:cellname2”).Select

If the row number is dynamic:
Range(“columnname” & rownumber).Select

If the column number is dynamic:
Range(columnname &“rownumber”).Select

Procedures/VBA Macros/Sub Routines

Sub keyword is used to start the macro program and this keyword is followed by the name of the macro. In the parenthesis following macro name, a list of parameters can be supplied and if there are no parameters to be passed, it is kept blank.

The statements that should be executed by the macro follow the sub declaration and the macro ends with an End Sub statement/keyword.

The following shows the declaration of a macro:

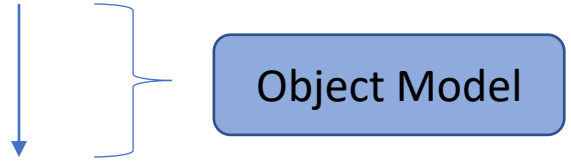
```
Sub MacroName()
[Statements]
End Sub
```

Objects, Properties & Methods

VBA is an Object Oriented Language – combination of different projects like Workbook, Worksheet, Ranges, Charts, Pivots etc.

Object Model
Whenever you use Objects – you need to do so in an orderly manner.
For ex: If there is no workbook, a worksheet cannot exist, similarly, if there is no worksheet, a pivot table or a graph cannot exist.

Workbook
Worksheet
Ranges/Pivots/Charts



A group of similar objects is called Collection. For ex: Collection of all workbook objects is referred to as Workbook collections – similarly collection of all worksheet collections is referred to as Worksheet collection

Objects, Properties & Methods contd..

All Objects have properties – a property can be thought of as a setting or an attribute. For ex: Chart object has properties like ChartTitle, ChartType, Legend

Similarly, Range has properties like Value, Count –

Value Property – Using this property, we can set a value of a particular cell or can read the value from a particular cell to a new variable

Examples:

```
New1 = Application.Workbooks("VBA File.xlsm").Worksheets("Sheet1"). Range("A1").Value
Worksheets("Sheet1").Range("A2").Value = New2
```

Count property: The number of cells in a range can be counted as

Examples:

```
Range(“A1:A10").Count,
Range(“A1:A100").Rows.Count
Range(“A1:D100").Columns.Count
```

Objects, Properties & Methods contd..

Just like all Objects have properties, they have Methods as well – a method is an action that is performed using an object.

For ex: for the Range object we have Methods like ClearContents, Delete

For Charts we have Methods like Activate, Select, BeforeDoubleClick etc..

Declare a Variable

Variables are declared using ‘Dim’ –

```
Dim Population As Long
Dim My_Name As String
Dim Age As Integer
```

‘Option Explicit’ should be used as it forces the user to declare all the variables beforehand

Rules –

- Should not exceed 40 characters
- No space
- No special characters

Advantages –

- Good Programing Standards
- Removes chances of unexpected results by letting VBA know what a variable type is
- Makes code more readable for other users

Type	Range of Values
Integer	-32,768 to 32,767
Long	-2,147,483,648 to 2,147,483,648
Single	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values
Double	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places)
Byte	0 to 255
String	1 to 65,400 characters
Date	January 1, 100 to December 31, 9999
Boolean	True or False

Session 2

Conditional Constructs
Looping Constructs

Conditional Constructs

IF…..THEN Statement

You can check one condition and on the basis of that then run one or multiple statements if the condition holds.
Example: The below program checks if in ‘Sheet1’, we have UK mentioned or not in the cell A2 and if UK is present, it will return London

```
Sub Capitalcity()
If Worksheets("Sheet1").Cells(2, 1).Value = "UK" Then
Worksheets("Sheet1").Cells(2, 2).Value = "London"
End If
End Sub
```

IF…..THEN …ELSE Statement

You can check one condition and on the basis of that then run one of the two statement blocks present.
Example: The below program checks if in ‘Sheet1’, US is present or not – if it is, then it will return ‘Washington DC’ else ‘Not US city’

```
Sub Capitalcity1()
If Worksheets("Sheet1").Cells(2, 1).Value = "US" Then
Worksheets("Sheet1").Cells(2, 2).Value = "Washington D.C"
Else
Worksheets("Sheet1").Cells(2, 2).Value = "Not US city"
End If
End Sub
```

IF program will always have a closing ‘END IF’ statement else VBA will give an error.

Looping Constructs

For….NEXT

This loop is used for a fixed number of times. In the below code, rownum is a variable and it will run 50 times till value 50 is reached increasing by 1 each time –

```
Sub Forexample1()
Dim rownum as integer
FOR rownum = 1 to 50
Worksheets("Sheet1").Cells(rownum,1).Value = rownum
Next rownum
End Sub
```

Looping Constructs contd..

DO WHILE….Loop

This is like FOR statement just that it will keep on looping till the condition is true –

```
Sub DoWhileexample1()
Dim I As Integer
I = 1
Do While I <= 10
Worksheets("Sheet2").Cells(I, 1).Value = I
I = I + 2
Loop
End Sub
```

Looping Constructs contd..

DO UNTIL….Loop

This is like DO WHILE statement just that it will keep on looping till the condition is not met –

```
Sub DoUntilexample1()
Dim I As Integer
I = 1
Do Until I = 11
Worksheets("Sheet2").Cells(I, 2).Value = I
I = I + 2
Loop
End Sub
```

In the above example, if you change the condition of Do Until I = 10 instead of 11, since I will never reach this value 10, as 1+2 = 3, then 3+2 = 5 and so on.. i.e. we will have only odd numbers, excel will keep on running till value of I is 32,767 which is the maximum value a variable declared as integer can take.

This is how Do Until is different from Do While.

Looping Constructs contd..

Do Loop….While and Do Loop…Until

These test the condition is true or not after the statements have been executed – so the loops will run at least once even if the condition was not true and this is how they are different from Do While…Loop and Do Until…Loop

```
Sub DoWhileexample2()
Dim I As Integer
I = 1
Do
Worksheets("Sheet2").Cells(I, 1).Value = I
I = I + 2
Loop While I <= 10
End Sub

Sub DoUntilexample2()
Dim I As Integer
I = 1
Do
Worksheets("Sheet2").Cells(I, 2).Value = I
I = I + 2
Loop Until I = 11
End Sub
```

Looping Constructs contd..

Select Case construct

Using this condition, you can choose from two or more options – let's suppose we have country name in cell A2 which can be either UK, US or India and depending on which country we have, we will place the value of its capital in cell B2. The program will look like following –

```
Sub selectcaseexample1()
Select Case Worksheets("Sheet1").Cells(2, 1).Value

Case Is = "UK"
Worksheets("Sheet1").Cells(2, 2).Value = "London"

Case Is = "US"
Worksheets("Sheet1").Cells(2, 2).Value = "Washington D.C"

Case Is = "India"
Worksheets("Sheet1").Cells(2, 2).Value = "New Delhi"
End Select
End Sub
```

Session 3

Custom Functions

Custom Functions

One of the advantages of VBA is that you can create your own functions using macros. These functions can be called and used as other functions in excel and use them.

Example: Let's suppose you want to calculate area of a circle – now area of a circle is πr^2 where the value of π is 3.14 and r is the radius of the circle –

- Click on 'Visual Basics' and then click on 'Module'
- In the program editor, you will write –

```
Function Area(Radius As Double)
Area = 3.14 * Radius * Radius
End Function
```

- Now go back to excel and call the function as you normally call any other function i.e. by using equal to operator and the name of the function = AREA(Cell containing radius or value of radius can be entered manually)

Custom Functions contd..

Syntax

- Declaring a custom function starts with keyword ‘Function’ and ends with ‘End Function’
- Following keyword Function, we write the name of the function – in our case we have named the function as AREA
- Now, the functions will have some arguments which will be declared in the brackets – here we just have one argument which is radius – double if you may recall is one of the data types we have in VBA – so we are telling VBA that this value can be too large
- In the next line, we show how to perform the calculation – so over here, the area is going to be calculated by πr^2 where the value of π is 3.14 and r is radius which we want the end user to fill in

Note

- Giving data types of the arguments is not always necessary but it is considered a good practice
- You can look for the functions you have created by going to ‘User Defined Functions’ in ‘Formula’ toolbar and change their description so that whenever you open this particular excel file, they are readily available for your use.

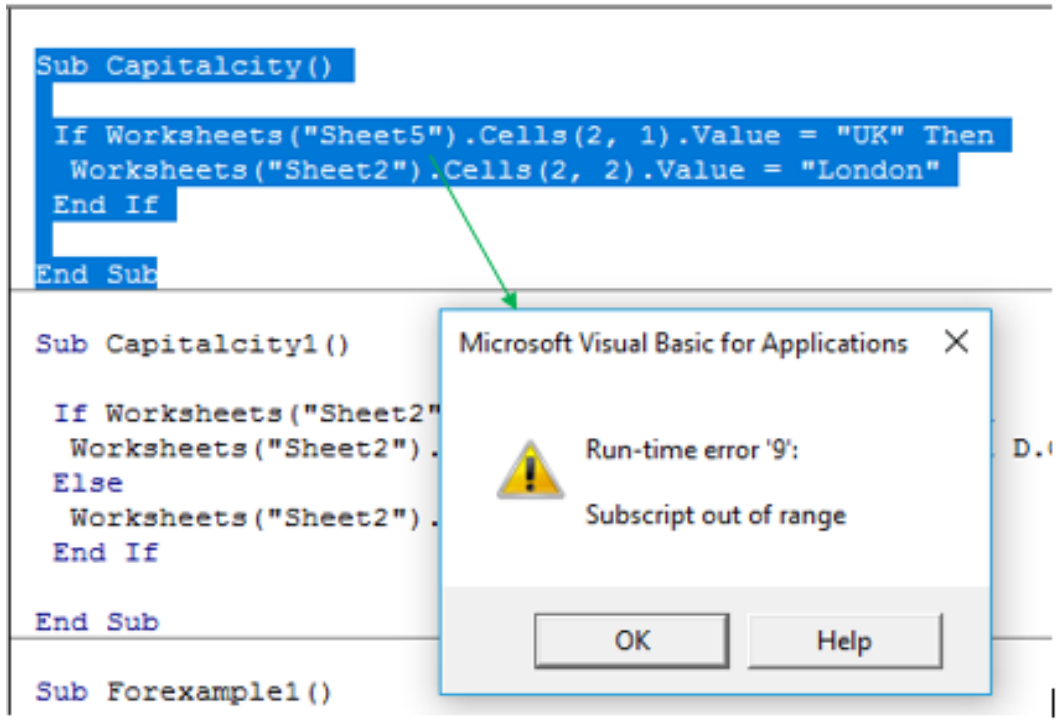
Session 4

Types of Errors
Error Handling
Debugging Programs

Errors

Run-time error

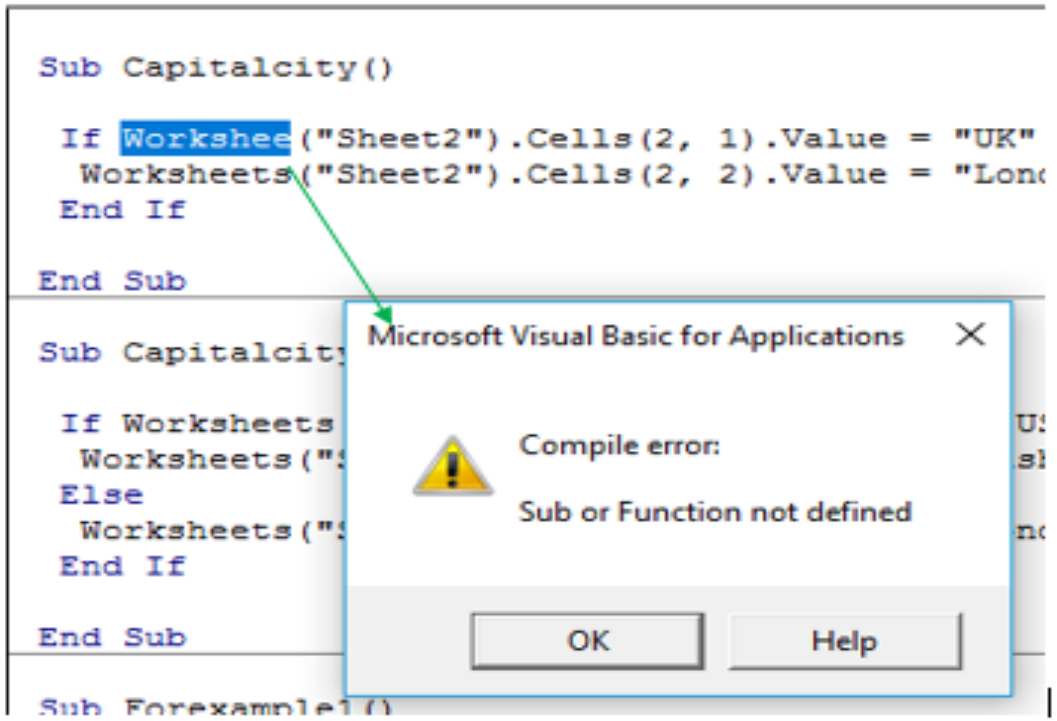
This error occurs when the code is executed and there may be a type mismatch or like in the example below, the workbook did not have any ‘Sheet5’ but because we mentioned that in the code, it gave an out of range error. Excel will throw a message box as shown below –



Errors contd..

Compile error

This error occurs when one of the statements has not been typed correctly or like in the example below, the keyword Worksheets has been misspelled. Excel will throw a prompt as shown below –



Errors contd..

Logical Error

Excel will not throw any prompt for this type of error as it is related to the logic of the code. This may result in the code giving unexpected outputs

Errors Handling

You can add an ‘Error Handling’ functionality to your code which if error occurs, will skip the rest of the code and give you a notification or perform other steps. It is always beneficial to get a notification as you will be aware of an error in the code but then it also depends upon what you are intending to do.

Example: – in the code below, since the workbook doesn’t have Sheet5 in it, VBA directly jumps to errorjump: statement and then through message box displays “Error has occurred”

```
Sub Capitalcity()
On Error GoTo errorjump
If Worksheets("Sheet5").Cells(2, 1).Value = "UK" Then
Worksheets("Sheet2").Cells(2, 2).Value = "London"
End If
Exit Sub

errorjump:
MsgBox "Error has occurred"
End Sub
```

A VBA procedure starts with Sub and ends with End Sub and excel will run the statements between them, but when we add error handlers to the code, the code will run between Sub and Exit Sub and if an error occurs, excel will jump all the statements and go straight to error handler which is declared after Exit Sub.

Debugging an error contd..

To understand which line gave the error, we use ‘Step-Into’ from Debug menu or press F8 –

```
Sub Debug1()
Dim D1 As Date
D1 = Worksheets("Sheet1").Range("A3")
End Sub
```

As we keep on pressing F8, VBA will keep on highlighting the lines –

```
Sub Debug1()
Dim D1 As Date
D1 = Worksheets("Sheet1").Range("A3")
End Sub
```

As we try to move from this line to another, VBA gives the same error which we got when we ran the entire code suggesting that the error occurred because of this 3rd line which is right – as we are passing a text value to a variable which has been declared as a date variable.

This feature is even more useful when the code is complex or has a lot of lines – this example had just 4 lines.

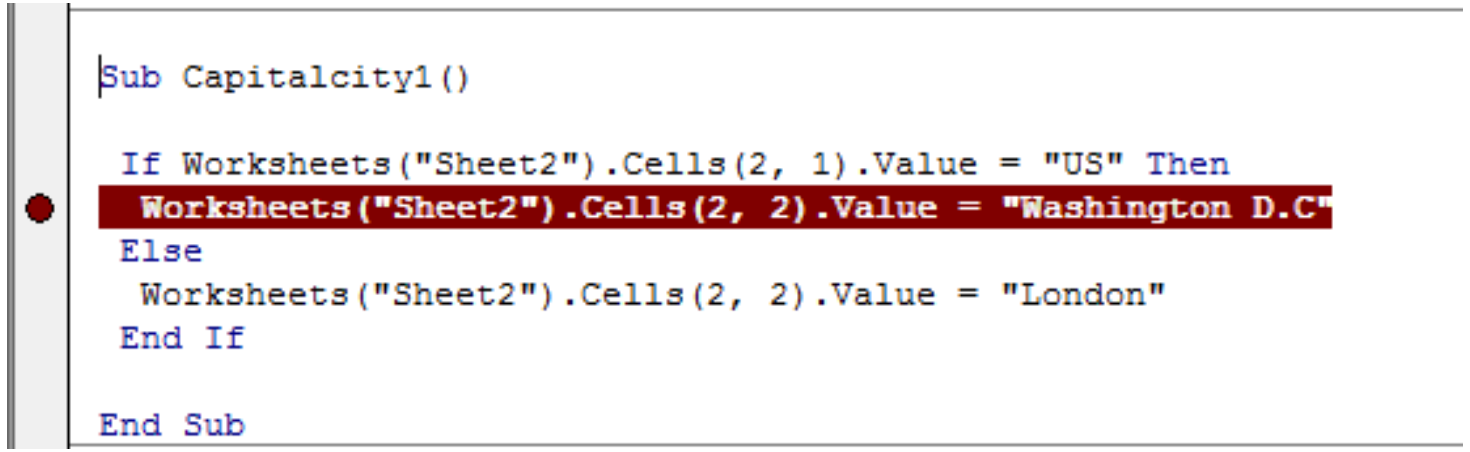
Debugging an error

There are different ways by which you can identify the error in the code and resolve it

Breakpoint

A selected program line at which execution will automatically stop and then you can see if the error has occurred till that line or not – if the error hasn't occurred then you can be sure that the remaining part of the code has the error statement.

To add a breakpoint, click on the right-side margin of the line you want the execution to stop at – so in the below program, the execution will stop at 3rd line of the program before 'Else'



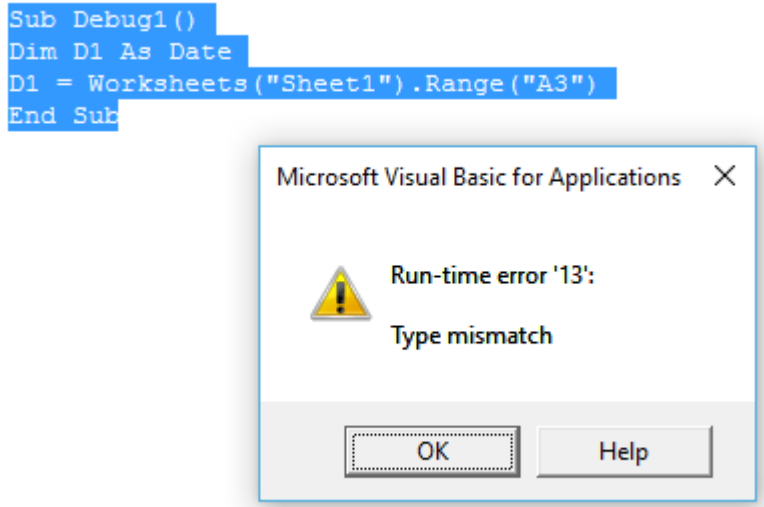
Debugging an error contd..

Stepping (for Run-time error debugging)

If while running the code, excel throws an error, you can step into the code and go from one line to another by pressing F8 and see which line excel has thrown an error for. In the example below, Cell A3 has a text value whereas we have declared in our program a variable D1 as date and are then passing A3 as its value –

```
Sub Debug1()
Dim D1 As Date
D1 = Worksheets("Sheet1").Range("A3")
End Sub
```

VBA gives the following error – >



Session 5

Form Controls

Form Controls

To add a control -

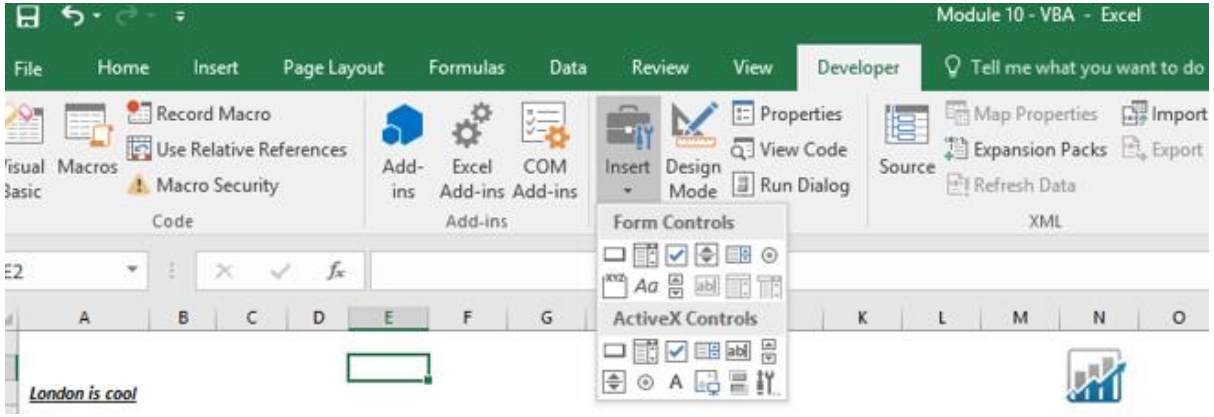
- Go to Developer tab
- In the Controls group, click on insert
- Select the control you want to use and create that on your worksheet using your mouse

You can modify the size or position then or can do it later as well by entering design mode.

When you add a control, to change its properties, you can go into ‘Design mode’ by clicking on the button next to ‘Insert’ in ‘Controls’.

Different controls in form control –

- Button
- Label
- Check Box
- Option Button
- List Box
- Combo Box
- Spin Button
- Scroll Bar
- Group Box



Form Controls contd..

Button

By pressing the Button you can execute a macro. In the below example, button has been used to execute a macro which will clear the cells in Range A1 to D4.

Clear Button

```
Sub clear()
Range("A1:D4").clear
End Sub
```

Label

This can also be used as a button but is more often used put in front of another Control to explain what that control stands for

Check Box

You can have as many check boxes you want and they all will be independent of each other. If a check-box is checked in, then it will return ‘TRUE’ else ‘FALSE’ to a linked cell.
Once you have created the check box, do a right-click and click on ‘Format Controls’ and click on ‘Cell Link’

Form Controls contd..

Option Button

Slightly different to check boxes just that you can copy multiple option buttons together but all of them would be dependent on each other i.e. at one time, only one will be selected the others will be turned off.
And when you link a cell, the cell will give the number of the option which is currently selected

List Box

The List box allows the selection of one or more items from a list. To give the range, once you have created a list box, just do a right click and select ‘Format Controls’ – in the ‘Input Range’, select the cells which you want to see in the list box and then select a cell where you want to output which item of the list is selected currently

Combo Box

The Combo Box is similar to list box just that it has a drop-down – so you can only see the selection and no other value. It will return the position of the selected item.

Form Controls contd..

Spin Button

This will allow you to increase or decrease value of a linked cell by a pre-defined amount. To use this, simply select spin button and create one – do a right click and go to ‘Format Controls’ – write the values as per your requirements and select the linked cell – using spin button then, you can increase or decrease value of the linked cell

Scroll Bar

The Scroll Bar Form Control often referred to as a Slider is a simple linear slider that allows the increase or decrease of a linked cells value by sliding a bar either left/right or up/down. The values can be filled similarly to the rest of the controls we have discussed so far.

Group Box

It is used to group similar controls together for better pictorial representation in a user form

Session 6

Events
Workbook Events
Worksheet Events

Events

What are Events?

Anything you do in an excel file is an event (an action) – like –
Opening a workbook
Going to a specific worksheet
Editing a worksheet
Entering data
Saving/closing file

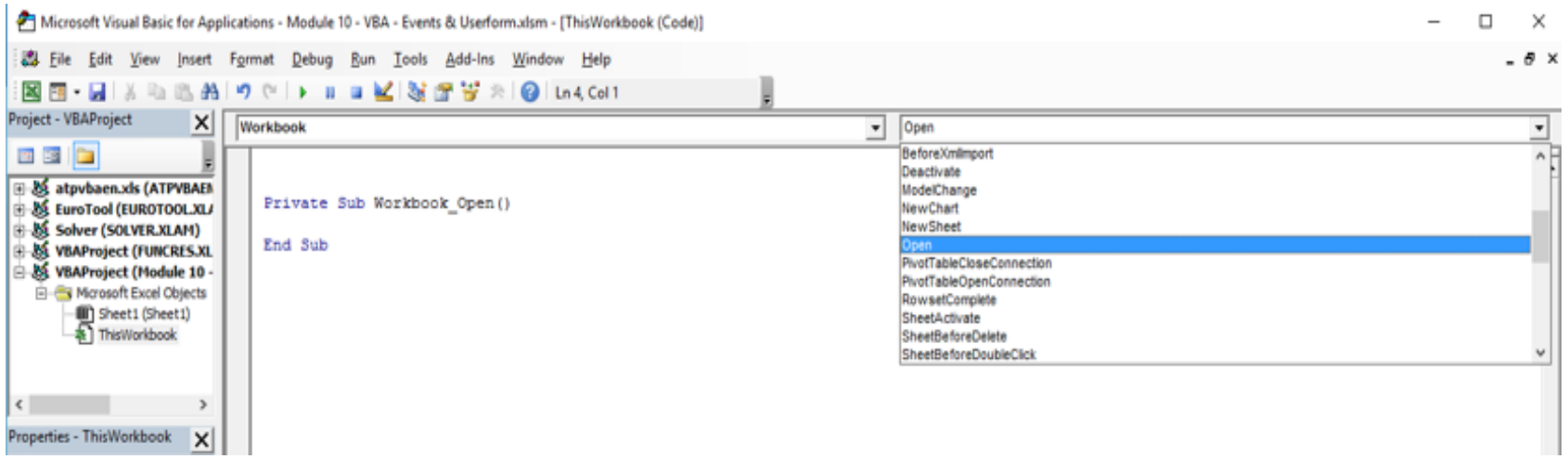
We can use these events by adding VBA codes (event handlers) to make excel perform in a certain way.

Example:
If I want a greeting message ‘Good Day’ whenever an excel file is opened
Or if I want excel to automatically capture the date and time whenever a specific range of cells are altered etc.

Workbook Events

Events for workbooks need to be stored in the ‘ThisWorkbook’ module whereas events for a worksheet need to be stored in the code module for that specific sheet only.

To look at Workbook events –
- Open Visual Basic editor
- Go to ‘ThisWorkbook’ module
- From the left drop-down, select ‘Workbook’
- Now you will notice that on the right drop-down, ‘Open’ is present and in the program editor a code shell has been automatically appeared (starts with Private Sub and ends with End Sub)
- We can add anything we want excel to perform whenever the excel file is opened
- In order to look at other events available, go to the right-drop down and click on the ‘Combo Box’ to see more options



Workbook Events contd..

Some of the most used workbook actions that trigger the Event-

- Open - The workbook is opened.
- Activate - The workbook is activated.
- BeforeClose - The workbook is about to be closed.
- BeforeSave - The workbook is about to be saved.
- NewSheet – Whenever a new sheet is created in the workbook.
- SheetActivate – Whenever any sheet in the workbook is activated.
- SheetChange - Any worksheet in the workbook is edited/changed by the user
- WindowActivate - Any window of the workbook is activated.

Workbook Open Event Examples

This event is triggered when the workbook is opened and this executes the Workbook_Open procedure. As stated earlier, it can be used to achieve a lot of things like –

- A Greeting Message
- Activating a specific sheet or a specific cell in a sheet
- Saving date and time or name of the user who opened the workbook

But for Workbook_Open event to fire, VBA macros should be enabled – so if the default setting is that the macros are disabled, whenever you open the file, the event will not execute on its own (till the time the macros are not enabled).

The following example gives a greeting message whenever someone opens the file with today's date –

```
Private Sub Workbook_Open()
Msg = "Hello!! " & Now()
MsgBox Msg, vbInformation
End Sub
```

Workbook Open Event Examples contd..

Below is an example of saving a person's name when he opens the file – as soon as the person opens an excel file which has the below macro, VBA will open another file 'Module 10 - VBA.xlsx' which will store the name of the person who opened the file along with the time. This type of code is used a lot to keep an audit trail of who accesses the file -

```
Private Sub Workbook_Open()
Dim xOpened As String
Dim xtime As String
xOpened = Environ("Username")
xtime = Format(Now, "dd/mm/yyyy hh:mm")
Workbooks.Open ("C:\PATH TO SECOND FILE\Module 10 - VBA.xlsx")
Workbooks("Module 10 - VBA.xlsx").Activate
i = Workbooks("Module 10 - VBA.xlsx").Sheets("Audit").Range("A65536").End(xlUp).Row
Workbooks("Module 10 - VBA.xlsx").Sheets("Audit").Cells(i + 1, 1).Value = xOpened
Workbooks("Module 10 - VBA.xlsx").Sheets("Audit").Cells(i + 1, 2).Value = xtime
Workbooks("Module 10 - VBA.xlsx").Save
Workbooks("Module 10 - VBA.xlsx").Close
End Sub
```

Workbook Open Event Examples contd..

You can also use conditional constraints or loops –

```
Private Sub Workbook_Open()
If Weekday(Date, vbSunday) = 2 Then
Msg = "Hello!!" & Now
MsgBox Msg, vbInformation
End If
End Sub
```

We can also select a specific cell when the excel file is opened –

```
Private Sub Workbook_Open()
Worksheets("Sheet1").Range("D4").Select
End Sub
```

Workbook Newsheet and BeforeSave Event Examples

Newsheet Event

The following procedure executes whenever a new sheet is added to the workbook. The sheet is passed to the procedure as an argument/parameter. Because a new sheet can be either a worksheet or a chart sheet, this procedure determines the sheet type. If it's a worksheet, it inserts a date and time stamp in cell A1.

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
If TypeName(Sh) = "Worksheet" Then _
Range("A1") = "Sheet added " & Now()
End Sub
```

BeforeSave Event

This event triggers just before the workbook is saved – so as soon as you press Cntrl+S or click on Save button, the event will trigger. Below is an example of a file which is being saved for the first time because that will throw the prompt of 'Save As' (if a file is in read-only mode, then also the below event will trigger) –

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)
If SaveAsUI Then
MsgBox "Save the file with xlsm i.e. macro-enabled format for enabling macros."
End If
End Sub
```

If the Save As box appears, i.e. the file is being saved for the first time (or in read-only mode).

WorkSheet Events

These events are for individual worksheets and are hence stored in the code module of the worksheet for which we plan to use them –

- Activate - The worksheet is activated.
- BeforeDoubleClick - The worksheet is double-clicked.
- BeforeRightClick - The worksheet is right-clicked.
- Calculate - The worksheet is calculated (or recalculated).
- Change - Cells on the worksheet are changed by the user.
- PivotTableUpdate A PivotTable on the worksheet has been updated.
- SelectionChange The selection on the worksheet is changed.

WorkSheet Event Examples

Worksheet_Change Event

It will trigger whenever a specified range or cell in a worksheet is changed by the user (not by a formula) – the code below will add the current date and time to column A of the row where change has taken place. The range we are interested in is from column B to column G – so if a user makes any changes to say cell B5, current date and time will be added to A5 – any changes to cell G10 and date and time will be added to A10 (and so on..)

```
Private Sub Worksheet_Change(ByVal Target As Range)
If Not Intersect(Target, Range("B:G")) Is Nothing Then
Cells(Target.Row, 1) = Now
End If
End Sub
```

Target argument/parameter has been defined as a Range object and refers to the cells which were changed – so if the user is making any changes to cell B5 – target variable will hold the value B5 – using intersect function, VBA then checks if there is an intersect i.e. common cell between target and range we have specified – and using 'Not' and 'Nothing' we are just saying that if there is an intersect, we want to add the date and current time to the row in which change took place.

Worksheet_Activate Event

This event executes whenever we go back to a specific sheet (which has the VBA code in its module) – the below example just throws a message box cautioning the user against deleting any formulas while he is on this sheet –

```
Private Sub Worksheet_Activate()
MsgBox "Do not delete the formula", vbOKOnly
End Sub
```

Session 7

Userforms

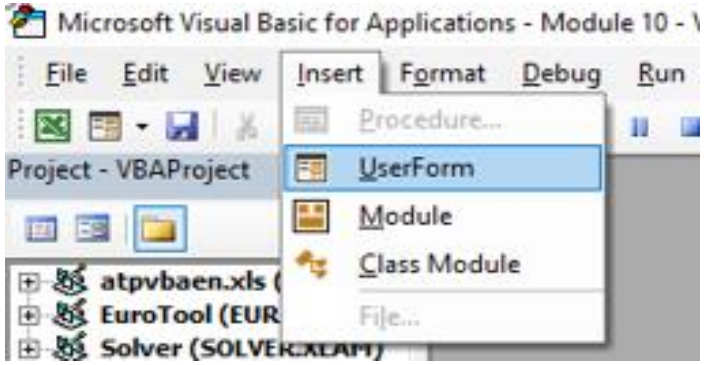
Userforms

A UserForm is a dialog box (or a sort of form) which is custom-built based upon the requirements or data required by the end user.

It is made in excel by using controls like form controls.

Before starting off with the creating a UserForm, it is very important to have a clear idea as to why it is required so that you can design it according to the requirements. Once this is done –

- Open the VBA Editor by going clicking on 'Visual Basic' or using Alt+F11
- Go to Insert and click on UserForm



- Once you have clicked on UserForm, a toolbox with controls will start appearing on the VBA editor – you can drag and drop these buttons as per your requirements

Userforms contd..

- You can edit the properties, sizes of these controls while dragging them or you can do them at the very end also once you have completed all your dragging
- Once the structure of the Userform has been created, it is time to write a VBA code which will make your UserForm visible – you can write it in the module for the sheet in which you want the UserForm to appear
- Now, the main purpose of creating a UserForm is to store the data being entered by a user – so once the data in the fields are entered, the user should have a ‘Submit’ or ‘OK’ button which would save the data –and this is where event handlers come in handy.
- Event handlers can also be used to ‘Cancel’ i.e. remove all the data or unload all the data from the UserForm