DATA ANALYTICS WITH EXCEL

SESSION 13

Acadgild

Agenda

1. Controlling Program Flow
   - For … Next Loops
   - While … Wend Loops
   - For Each … Next Loops
   - Do … Loop Loops
2. Exercise – Loops & Conditional statements
3. User Form
   - Where do you find the ActiveX & Form controls
   - Types of Form Controls
   - Types of ActiveX Controls
   - Form vs. ActiveX Controls
4. Event Handler
   Different Type of Event Handler
5. Exercise – User forms
6. The IDE Debugger
   - Breakpoints
   - Watches
   - Stepping Through Code
7. Good VBA Coding Practices
8. Optimization in VBA
9. Error Handling In VBA
   - On Error Goto 0
   - On Error Resume Next
   - On Error Goto <label>
11. Password Protection of VBA Code
12. Using VBA to Create or Update Charts
13. Using VBA for Sending an automatic email using outlook
11. Connecting to Database using VBA

Controlling Program Flow

The two basic kinds of control structures in programming:

- Conditional statements
  - Execute different sets of commands depending upon the current values of the program variables.

    If … Then … ElseIf … Else … End If

    Select Case … Case … Case Else … End Select

- Loops
  - Repeat an instruction set

    For … Next

    For Each … Next

    Do While … Loop

For … Next Loops

- FOR … NEXT loops
  - Execute a set of VBA statements a predetermined number of times [(end-start)/increment]
  - The counter variable can be used in expressions within the loop

```
FOR counter = start TO end [STEP increment]
VBA statements
 NEXT counter
```

- To terminate a FOR-NEXT loop,
  - Embed the EXIT FOR command within a conditional statement that evaluates the termination criteria
  - Example :

```
Public Sub ForToNext_IntegerCounter()
        Dim intCounter As Integer
        For intCounter = 1 to 10
            Debug.Print intCounter
            If intCounter = 9 Then Exit For End If
        Next intCounter
        Debug.Print "End value: " & intCounter
    End Sub
```

For … Next Example

What does the following code do?

```
Sub Label_Sheets()
        Const cintNumber_of_Sheets = 20
        Dim intSheetNumber As Integer
        For intSheetNumber = 1 To cintNumber_of_Sheets
        ThisWorkbook.Worksheets(intSheetNumber).Name = _
            "Terr" & intSheetNumer
        Next intSheetNumber
End Sub
```

- Output -

  It will change the names of sheets to Terr1, Terr2 and so on till Terr20 if we have twenty sheets in current workbook.

While … Wend Loops

Executes a series of statements as long as a given condition is True

Syntax:

```
While condition

    [statements]

Wend
```

- The While...Wend statement syntax has these parts
  - **Condition** - Numeric expression or string expression that evaluates to True or False. If condition is Null, condition is treated as False.
  - **Statements** - One or more statements executed while condition is True.

➢ If condition is True, all statements are executed until the Wend statement is encountered. Control then returns to the While statement and condition is again checked. If condition is still True, the process is repeated. If it is not True, execution resumes with the statement following the Wend statement.

While.. Wend Loop Example

What does the following code do?

```
Sub PrintCounting()

    Dim lngCounter  as Long  ' Declare variable.

    While lngCounter < 20 ' Test value of Counter.
        lngCounter = lngCounter + 1 ' Increment Counter.
        Debug.Print  lngCounter
    Wend   ' End While loop when Counter > 19.

End Sub
```

Output:

It prints 1 to 20 in the immediate window.

For Each … Next Loops

- FOR EACH … NEXT loop
  - Executes a set of VBA statements for each object in a collection

```
FOR EACH object/ element IN collection/array
    VBA statements
NEXT object/element
```

- What does the following code do?

```
Dim Range_Cell As Range
For Each Range_Cell In ActiveSheet.UsedRange
        Range_Cell.Font.ColorIndex = 3
Next Range_Cell
```

Output :

It will check for the cells that have values in them (used cells) and then will change the color of the respective cells' text to Color Index 3.

Do … Loop Loops

- Do … Loop loops
  - Execute a set of VBA statements while an expression is true

```
Do [{While | Until} condition]
[statements]
[Exit Do]
[statements]
Loop
```

(OR)

```
Do
[statements]
[Exit Do]
[statements]
Loop [{While | Until}
condition]
```

- To terminate a Do-Loop loop,
  - Embed the EXIT Do command within a conditional statement that evaluates the termination criteria

Do … Loop Example

This procedure calculates the factorial of the input number

```
Public Sub Factorial(intNumber As Integer)
    Dim intLoop As Integer
    Dim intFact As Integer
    intLoop = 1
    intFact = 1
    Do While intLoop <= intNumber
        intFact = intFact * intLoop
        intLoop = intLoop + 1
    Loop
    MsgBox intFact
End Sub
```

Five Routines That Give The Same Answer

```
Public Sub DoLoop_WhileAtStart()
  Dim intInteger As Integer
  Do While intInteger < 100
    intInteger = intInteger + 1
  Loop
  Debug.Print intInteger
End Sub
```

```
Public Sub DoLoop_UntilAtStart()
  Dim intInteger As Integer
  Do Until intInteger >= 100
    intInteger = intInteger + 1
  Loop
  Debug.Print intInteger
End Sub
```

```
Public Sub DoLoop_WhileAtEnd()
  Dim intInteger As Integer
  Do
    intInteger = intInteger + 1
  Loop While intInteger < 100
  Debug.Print intInteger
End Sub
```

```
Public Sub DoLoop_UntilAtEnd()
  Dim intInteger As Integer
  Do
    intInteger = intInteger + 1
  Loop Until intInteger >= 100
  Debug.Print intInteger
End Sub
```

```
Public Sub WhileWend()
  Dim intInteger As Integer
  While intInteger < 100
    intInteger = intInteger + 1
  Wend
  Debug.Print intInteger
End Sub
```

Exercise – Loops & Conditional statements

Duration: 15 min

1. Write a function to fill cells in Sheet1 from A1 to A100 with values from 1 to 100

2. Write a function to fill only even numbered cells with their row numbers in column B from B1 to B100

3. Fill Column C with remainder of division by 5 in each row (remainder should be filled in words), i.e. cell C14 should have "FOUR" written in it

User Form

Form Controls:

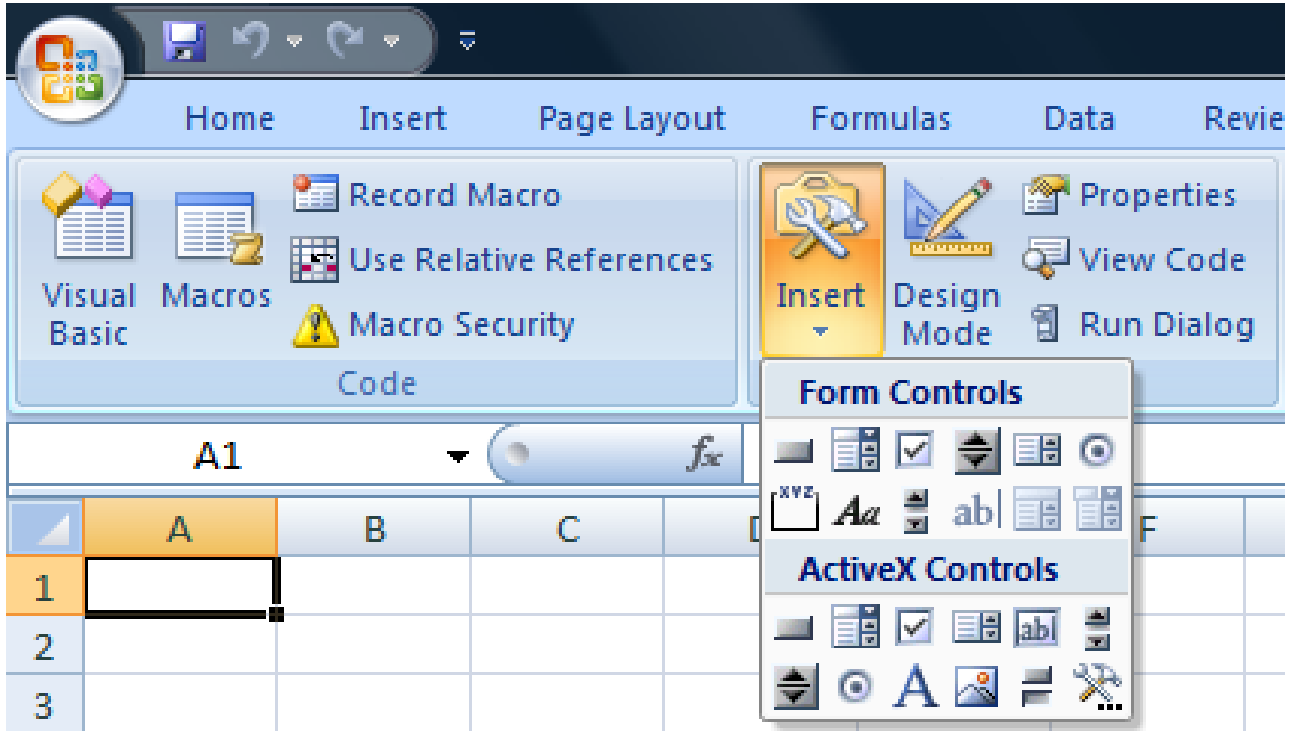
- Simple to use
- built in methods to easily place values in worksheet cells
- No VBA knowledge needed
- Available since earlier versions of excel

ActiveX Controls:

- Properties can be changed. Eg. borders, background, height etc.
- Built in macro events
- Need to know VBA to program
- More properties can be customized as they are relatively newer than Form Controls

Where do you find the ActiveX & Form controls

- Both the ActiveX and Forms controls are located in the Controls group on the Developer tab.
- Click “Insert” to get the list shown below:



Types of Form Controls

- Button
  - Runs a macro that performs an action when a user clicks it. A button is also referred to as a push button

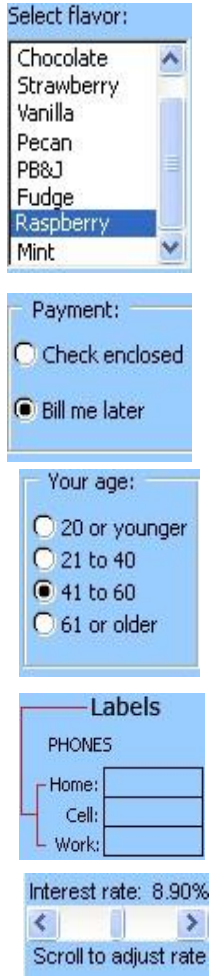
- Combo Box
  - Combines a text box with a list box to create a drop-down list box
  - Is more compact than a list box but requires the user to click the down arrow to display the list of items
  - Use a combo box to enable a user choose only one item from the list
  - The control displays the current value in the text box, regardless of how that value is entered

- Check Box
  - Shows True or False on selecting
  - Can select more than one check box on a worksheet or in a group box

- Spin button
  - Increases or decreases a value, such as a number increment, time, or date
  - To increase the value, click the up arrow; to decrease the value, click the down arrow
  - Typically, a user can also type a text value directly into an associated cell or text box.

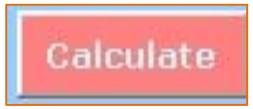
Types of Form Controls (Contd)

- List box
  - Displays a list of one or more items of text from which a user can choose
  - Use a list box for displaying large numbers of choices that vary in number or content
  - Three types of list boxes: Single selection, Multiple selection, Extended Selection
- Option button (Radio Button)
  - Allows a single choice within a limited set of mutually exclusive choices
  - an option button is usually contained in a group box or a frame
  - An option button is also referred to as a radio button
- Group Box
  - Groups related controls into one visual unit in a rectangle with an optional label
  - Typically, option buttons, check boxes, or closely related contents are grouped.
- Label
  - Identifies the purpose of a cell or text box, or displays descriptive text (such as titles, captions, pictures) or brief instructions
- Scroll bar
  - Scrolls through a range of values when clicked on the scroll arrows or drag the scroll box
  - A user can also type a text value directly into an associated cell or text box



Types of ActiveX Controls

- **Label**
  - Identifies the purpose of a cell or text box, or displays descriptive text (such as titles, captions, pictures) or brief instructions
- **Text Box**
  - Enables to view, type, or edit text or data that is bound to a cell
  - A text box can also be a static text field that presents read-only information
- **Command Button**
  - Runs a macro that performs an action when a user clicks it
- **Check box**
  - Shows True or False on selecting
  - Can select more than one check box on a worksheet or in a group box



Types of ActiveX Controls (Contd)

- **Option button (Radio Button)**
  - Allows a single choice within a limited set of mutually exclusive choices
  - an option button is usually contained in a group box or a frame
  - An option button is also referred to as a radio button

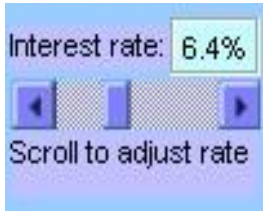
- **List box**
  - Displays a list of one or more items of text from which a user can choose
  - Use a list box for displaying large numbers of choices that vary in number or content

- **Combo box (Drop Down Box)**
  - Combines a text box with a list box to create a drop-down list box
  - Is more compact than a list box but requires the user to click the down arrow to display the
    list of items
  - Use a combo box to enable a user choose only one item from the list
  - The control displays the current value in the text box, regardless of how that value is entered

Types of ActiveX Controls (Contd)

- Frame Control
  - Groups related controls into one visual unit in a rectangle with an optional label
  - Typically, option buttons, check boxes, or closely related contents are grouped

- Scroll bar
  - Scrolls through a range of values when clicked on the scroll arrows or drag the scroll box
  - A user can also type a text value directly into an associated cell or text box



- Spin button
  - Increases or decreases a value, such as a number increment, time, or date
  - To increase the value, click the up arrow; to decrease the value, click the down arrow
  - Typically, a user can also type a text value directly into an associated cell or text box.

Form vs. ActiveX Controls

Always use Form Controls over ActiveX controls

- ActiveX controls trigger events while Form controls call macros assigned to them
- Sometimes, events for ActiveX controls are triggered on its own when some other event is triggered which might cause errors
- ActiveX controls are visually more appealing and have more formatting options available than form controls
- At times, on opening the workbook, ActiveX controls are not recognised and give errors

Event Handler

- **An event handler procedure is a specially named procedure that's executed when a specific event occurs**
- **Following are examples of types of events that Excel can recognize:**

  - A workbook is opened or closed
  - A worksheet is activated or deactivated
  - An object is clicked
  - A worksheet is changed
  - A workbook is saved

Different Types of Event Handler

- **Workbook Events**
  - Events that occur for a particular workbook. Examples for this events are Open, Close, and BeforeSave
- **Worksheet Events**
  - Events that occur for a particular worksheet. Examples include Change, and SelectionChange
- **Application Events**
  - Events that occur for a particular the application (Excel itself)
- **UserForm Events**
  - Events that occur for a particular UserForm or an object that contained on the UserForm. For example Click event
- **Chart Events**
  - Events that occur for a particular chart. Examples include Select
- **Events not associated with objects**
  - For examples OnTime and OnKey events

The IDE Debugger

Debugging

- Is the process of identifying and correcting programming errors

The IDE has two options for debugging

- The breakpoint (Debug, Toggle Breakpoint or [F9]) option
  - Causes Excel to pause before the selected line of code is executed
- The watch (Debug, Quick Watch or [Shift]-[F9]) option
  - Tells Excel which variables you want to examine.
  - F8 (step into) can work as an alternative for watch window for step by step execution to understand the execution and values of variables in middle of code

Breakpoints

Microsoft Visual Basic - Book1 - [Module1 (Code)]

File Edit View Insert Format Debug Run Tools Window Help

Ln 10, Col 3

Project - VBAProject

VBAProject (Book1)
- Microsoft Excel Objects
  - Sheet1 (Sheet1)
  - Sheet2 (Sheet2)
  - Sheet3 (Sheet3)
  - ThisWorkbook
- Modules
  - Module1
- vbaXYChartLabeler (XY

Properties - Module1

Module1 Module

Alphabetic | Categorized

(Name) Module1

(General) Break | Test

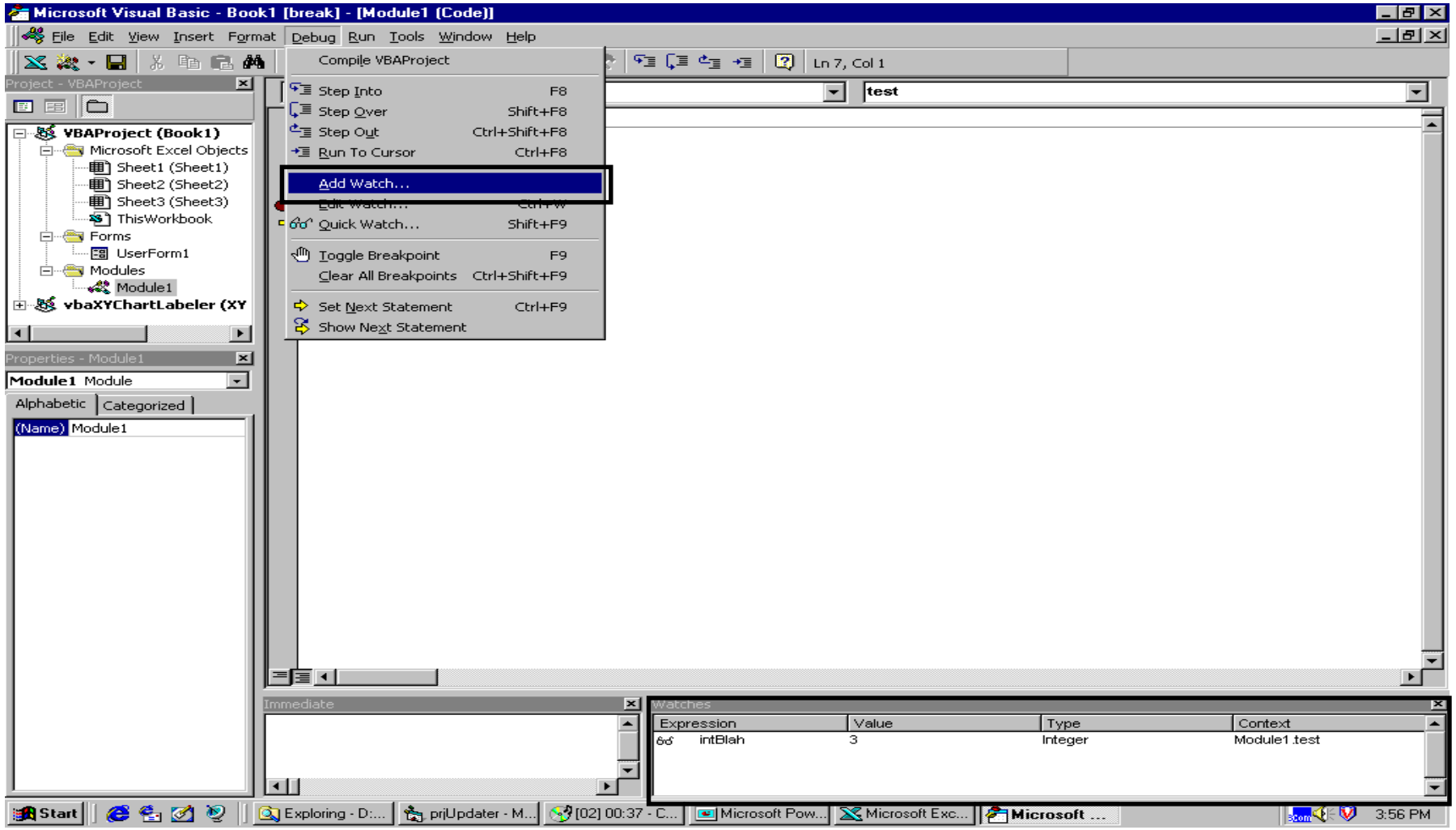
```
Option Explicit

'This subroutine does nothing. It is pretty much only here so that
'I can demonstrate how one uses breakpoints. So there.
Sub Test()
    Dim intTest As Integer
    Dim blnTest As Boolean

    intTest = 2
    If intTest > 3 Then
        blnTest = False
    Else
        blnTest = True
    End If

    MsgBox blnTest
End Sub
```

This is a breakpoint

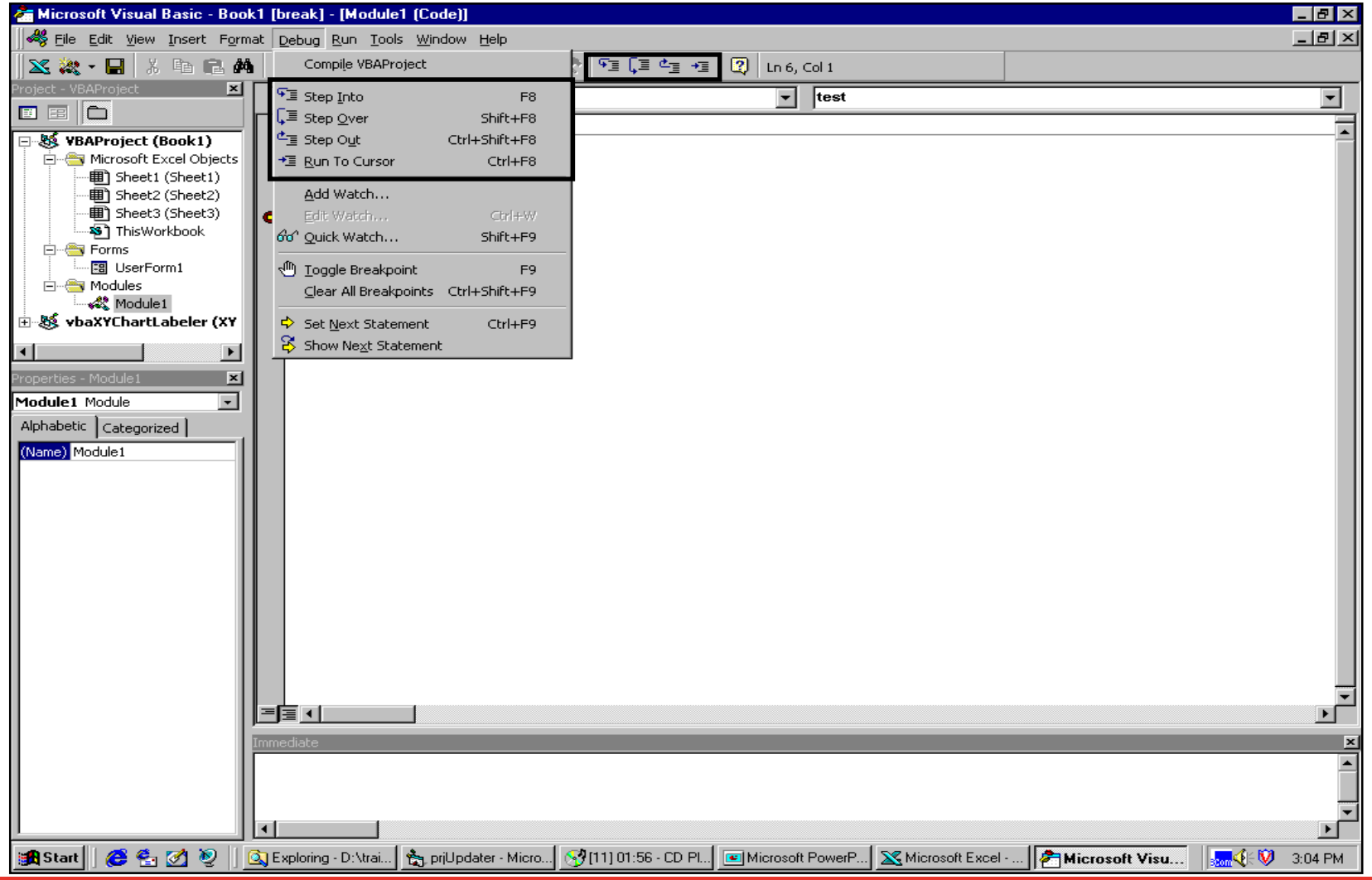
Immediate

Start | Exploring - N:... | Microsoft Pow... | Microsoft Exc... | Microsoft ... | Greg Sarjeant... | 5:15 PM

Watches


Microsoft Visual Basic - Book1 [break] - [Module1 (Code)]
File Edit View Insert Format Debug Run Tools Window Help
Compile VBAProject
Step Into F8
Step Over Shift+F8
Step Out Ctrl+Shift+F8
Run To Cursor Ctrl+F8
Add Watch...
Edit Watch...
Quick Watch... Shift+F9
Toggle Breakpoint F9
Clear All Breakpoints Ctrl+Shift+F9
Set Next Statement Ctrl+F9
Show Next Statement
Ln 7, Col 1
test
Project - VBAProject
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook
Forms
UserForm1
Modules
Module1
vbaXYChartLabeler (XY
Properties - Module1
Module1 Module
Alphabetic Categorized
(Name) Module1
Immediate
Watches
Expression Value Type Context
intBlah 3 Integer Module1.test
Start
Exploring - D:...
prjUpdater - M...
[02] 00:37 - C...
Microsoft Pow...
Microsoft Exc...
Microsoft ...
3:56 PM

Stepping Through Code



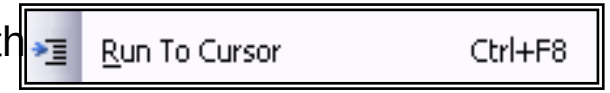
Stepping Through Code

- Step Into
  - Executes current line
    - if current line is a sub/function call, the debugger will enter (Step Into) the sub/function

- Step Over
  - Executes current line
    - if current line is a sub/function call, the debugger will execute the sub/function without stepping through it

- Step Out
  - Completes execution of the current sub/function and returns to the sub/function which called it

- Run To Cursor
  - Executes all code until th



Optimization in VBA

- Every VBA method or property call takes time to execute
- The number of method and property calls is equal to the number of "." (dot) operators
- So, minimizing the number of "." operators will speed up your code

- "." operators could be minimized by
  - using object variables

```
Workbooks(1).Worksheets(1).Range("A1").Value = 10
Workbooks(1).Worksheets(1).Range("A2").Value = 20
```

could be replaced by

```
Set wksSheet = Workbooks(1).Worksheets(1)
wksSheet.Range("A1").Value = 10
wksSheet.Range("A2").Value = 20
```

  - using the With statement

```
With Workbooks(1).Sheets(1)
    .Range("A1").Value = 10
    .Range("A2").Value = 20
End With
```

Loops

- Use For Each … Next Loops
  - they are much faster than indexed loops to iterate through collections or arrays

- Keep Properties and Methods outside loops
  - Change

```
For intLoop = 1 To 200
        Cells(intLoop,1).Value = Cells(1,1).Value
Next intLoop
```

  - to

```
intCV = Cells(1,1).Value
For intLoop = 1 To 200
        Cells(intLoop,1).Value = intCV
Next intLoop
```

Other Optimizations

- Minimize object Activation and Selection
- Remove unnecessary recorded code
- Minimize use of Variant data type

- Use specific object types
  - Dim wksSheet as Worksheet

- Use constants wherever possible
  - Constant strFileName As String = “MyFile.xls”

- Turn off screen updating
  - Application.Screenupdating = False

- Use worksheet functions whenever one is available
  - Application.WorksheetFunction.Average(Range(“A1:A20”))

Good Coding Practices

- Your VBA Code
  - should be clear
    - use highly descriptive variable names
    - use comments to explain the program flow/ variable usage
    - align comments to make them readable
    - indent the code
  - should be flexible
    - avoid hard coding cell addresses, constant values, client names, directory structures,
  - should be fast
    - try to optimize you code as much as you can

Error Handling In VBA

- **Error handling refers to the programming practice of anticipating and coding for error conditions that may arise when your program runs**

- **Errors are of three types**
  - Compiler Errors such as undeclared variables that prevent your code from compiling
  - User Data Entry Error such as a user entering a negative value where only a positive number is acceptable
  - Run Time Errors that occur when VBA cannot correctly execute a program statement

The On Error Statement

- **The heart of error handling in VBA is the On Error statement**

- **The On Error statement takes three forms**
  - On Error Goto 0
  - On Error Resume Next
  - On Error Goto <label>:

On Error Goto 0

- Default mode in VBA

  - Indicates that when a run time error occurs VBA should display its standard run time error message box, allowing to enter the code in debug mode or to terminate the VBA program
  - When On Error Goto 0 is in effect, it is the same as having no enabled error handler
  - Any error will cause VBA to display its standard error message box

On Error Resume Next

- The most commonly used and misused form

  - Instructs to VBA to essentially ignore the error and resume execution on the next line of code
  - It is very important to remember that On Error Resume Next does not in any way "fix" the error.
  - It simply instructs VBA to continue as if no error occurred
  - However, the error may have side effects, such as uninitialized variables or objects set to Nothing

On Error Goto <label>:

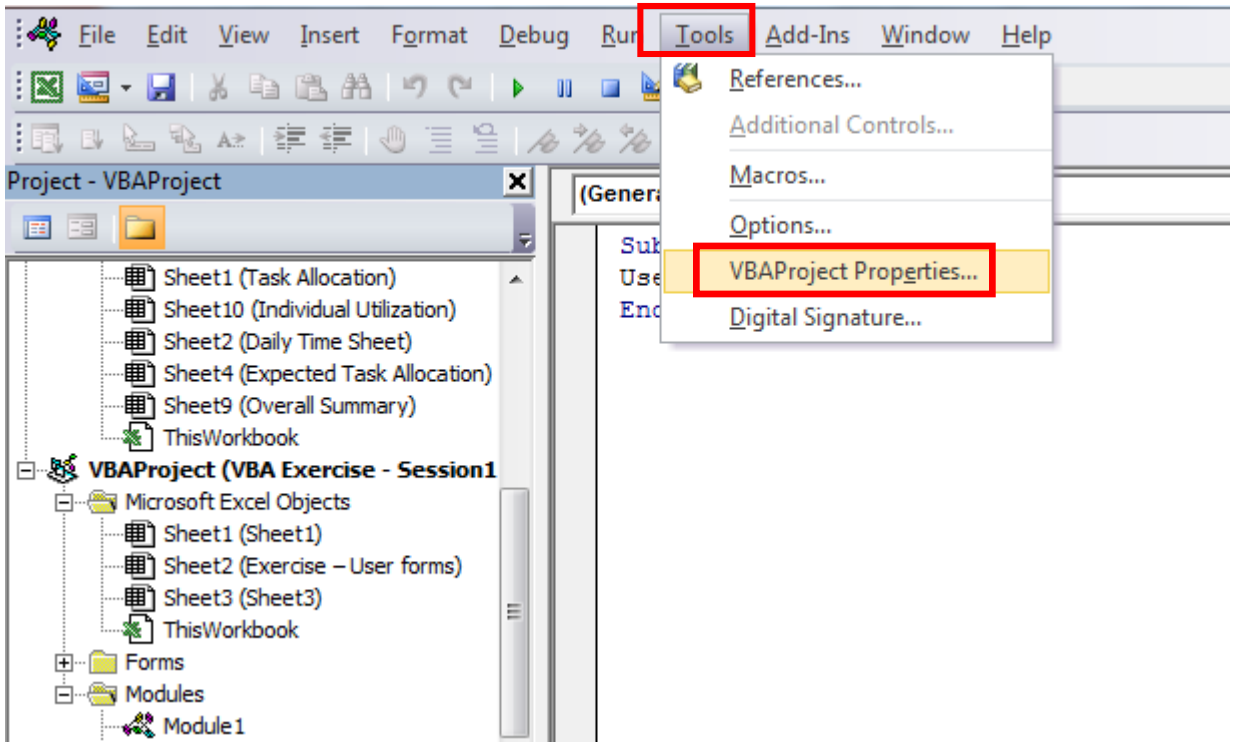
- An error handler is said to be **enabled** when an On Error statement is executed

  - Only one error handler is enabled at any given time, and VBA will behave according to the enabled error handler
  - An active error handler is the code that executes when an error occurs and execution is transferred to another location via a On Error Goto <label>: statement

Password Protection of VBA Code

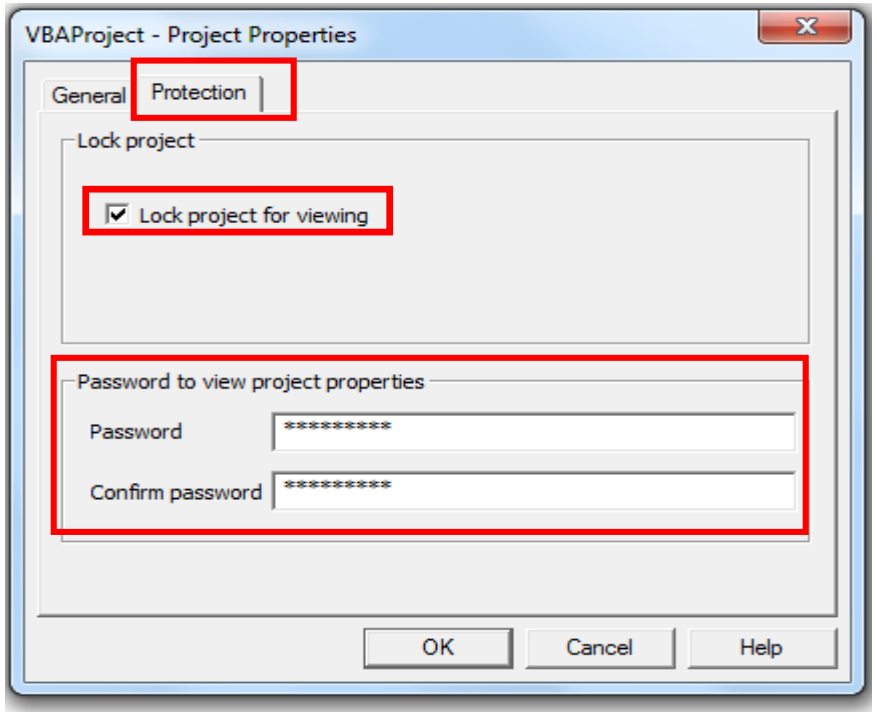
Just like we can password protect our workbooks, we can password protect a macro in Excel from being viewed (and executed).

1. In VBA window, click on Tools and then VBAProject Properties

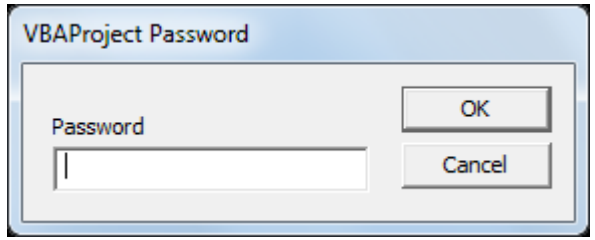


Password Protection of VBA Code

2. On the Protection tab, check "**Lock project for viewing**" and enter a password twice and Click on OK.



3. No w Save this file, close and reopen the Excel file. Try to view the code. The following dialog box will appear:

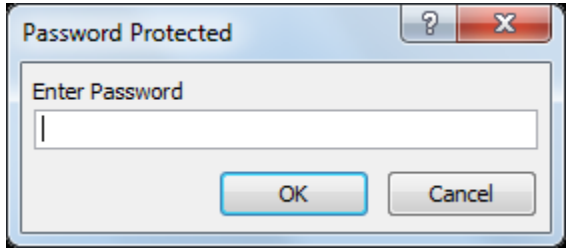


Password Protection of VBA Code

4. We can still execute the code by clicking on the command button but we cannot view or edit the code anymore (unless you know the password).
5. If we want to password protect the macro from being executed, add the following code lines in the code of command button:-

```
Dim password As Variant
password = Application.InputBox("Enter Password", "Password Protected")

Select Case password
   Case Is = False
      'do nothing
   Case Is = "easy"
      Range("A1").Value = "This is secret code"
   Case Else
      MsgBox "Incorrect Password"
End Select
```

2. Now, when we click on the command button it asks for the password.



Using VBA to Create or Update Charts

- A chart is simply packed with objects, each of which has its own properties and methods. Because of this, manipulating charts with VBA can be a bit of a challenge.
- In Excel, a chart can be located in either of two places within a workbook:
  - As an embedded object on a worksheet: A worksheet can contain any number of embedded charts
  - In a separate chart sheet: A chart sheet holds a single chart
  - Most charts are created manually, by using the Chart Wizard. But you can also create charts by using VBA. And, of course, you can use VBA to modify existing charts.
  - The fastest way to create a chart on a new sheet is to select your data and then press F11. Excel creates a new chart sheet and uses the default chart type

Chart object model

- The Worksheet object contains a ChartObject object, which contains a Chart object. The Chart object has a ChartTitle object, and the ChartTitle object has a Text property which stores the text that's displayed as the chart's title.
  - Here's another way to look at this hierarchy for an embedded chart:
    - Application. Workbook.Worksheet.ChartObject.Chart.ChartTitle
  - For a chart sheet, the object hierarchy is a bit different because it doesn't involve the Worksheet object or the ChartObject object. For example, here's the hierarchy for the ChartTitle object for a chart in a chart sheet:
    - Application.Workbook.Chart.ChartTitle

Create a Chart

- In the following code the Location method is used to move the chart to a worksheet.

```
Sub CreateChart()

    Dim objChart As Chart
    Application.ScreenUpdating = False
    Set objChart = Charts.Add 'Adding the chart
    Set objChart = objChart.Location(Where:=xlLocationAsObject, Name:="Sheet1")
    With objChart
        .SetSourceData Sheets(1).Range("A1:C10"), PlotBy:=xlColumns 'Providing data source in the form
    of range
        .HasTitle = True ' Giving titles to the chart
        .ChartTitle.Text = "Sales“
        .ChartType = xlColumnClustered  'Giving the chart type (e.g xlColumnClustered,xlColumnstacked
    etc)
        .HasLegend = False
        .ApplyDataLabels Type:=xlDataLabelsShowValue
      .Axes(xlCategory).TickLabels.Orientation = xlHorizontal 'Aligning the ticklabels of X Axis
        .ChartTitle.Font.Bold = True
        .ChartTitle.Font.Size = 12
        .Deselect
      End With
   Application.ScreenUpdating = True

End Sub
```

Chart Formatting

```
Sub ChartMods2()

With Sheets(“Sheet1”).ChartObjects(“Chart 1”).Chart
    .Type = xlArea
    .ChartArea.Font.Name = “Arial”
    .ChartArea.Font.FontStyle = “Regular”
    .ChartArea.Font.Size = 9
    .PlotArea.Interior.ColorIndex = xlNone
    .Axes(xlValue).TickLabels.Font.Bold = True
    .Axes(xlCategory).TickLabels.Font.Bold = True
    If .HasLegend = True Then .Legend.Position = xlBottom
End With

End Sub
```

Looping through all charts

```
Sub ChangeChartType()
        Dim chtobj as ChartObject
For Each chtobj In ActiveSheet.ChartObjects
        chtobj.Chart.ChartType = xlArea  'It changes the chart type of all the charts
to area chart in the active sheet
Next chtobj
End Sub


Sub ChangeChartType2()
        Dim cht as Chart
For Each cht In ActiveWorkbook.Charts
        cht.ChartType = xlArea  'It changes the chart type of all the charts to area
chart in the whole workbook
Next cht
End Sub
```

Using VBA for Sending an automatic email using outlook

- The Outlook object model provides all of the functionality necessary to manipulate data that is stored in Outlook folders, and it provides the ability to control many aspects of the Outlook user interface (UI).
- Below is the vba code module which can be used to send email via your outlook account:

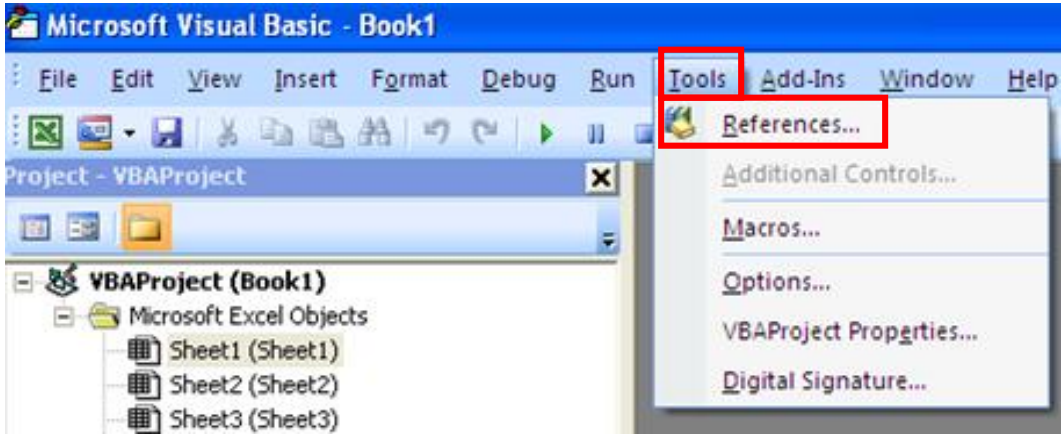
```
Sub SendEmail()
    Dim objOutlook      As Object
    Dim objItem         As Object

    Application.ScreenUpdating = True
    Set objOutlook = CreateObject("Outlook.Application")
    Set objItem = objOutlook.CreateItem(0)
    With objItem
        .to = "xyz@zzz.com"             ' Enter the email id to which you want to send your email
        .Subject = "Test Mail"                          'Subject of the email
        .body = "Testing Email Sending Via Outlook Object"       'Body text of the email
        .cc = "abc@yyy.com"    'Enter the email id to which you want to copy your email to
        .send
    End With
    Application.ScreenUpdating = True
```

Connecting to Database using VBA

- Database is a collection of information that is organized so that it can easily be accessed, managed, and updated

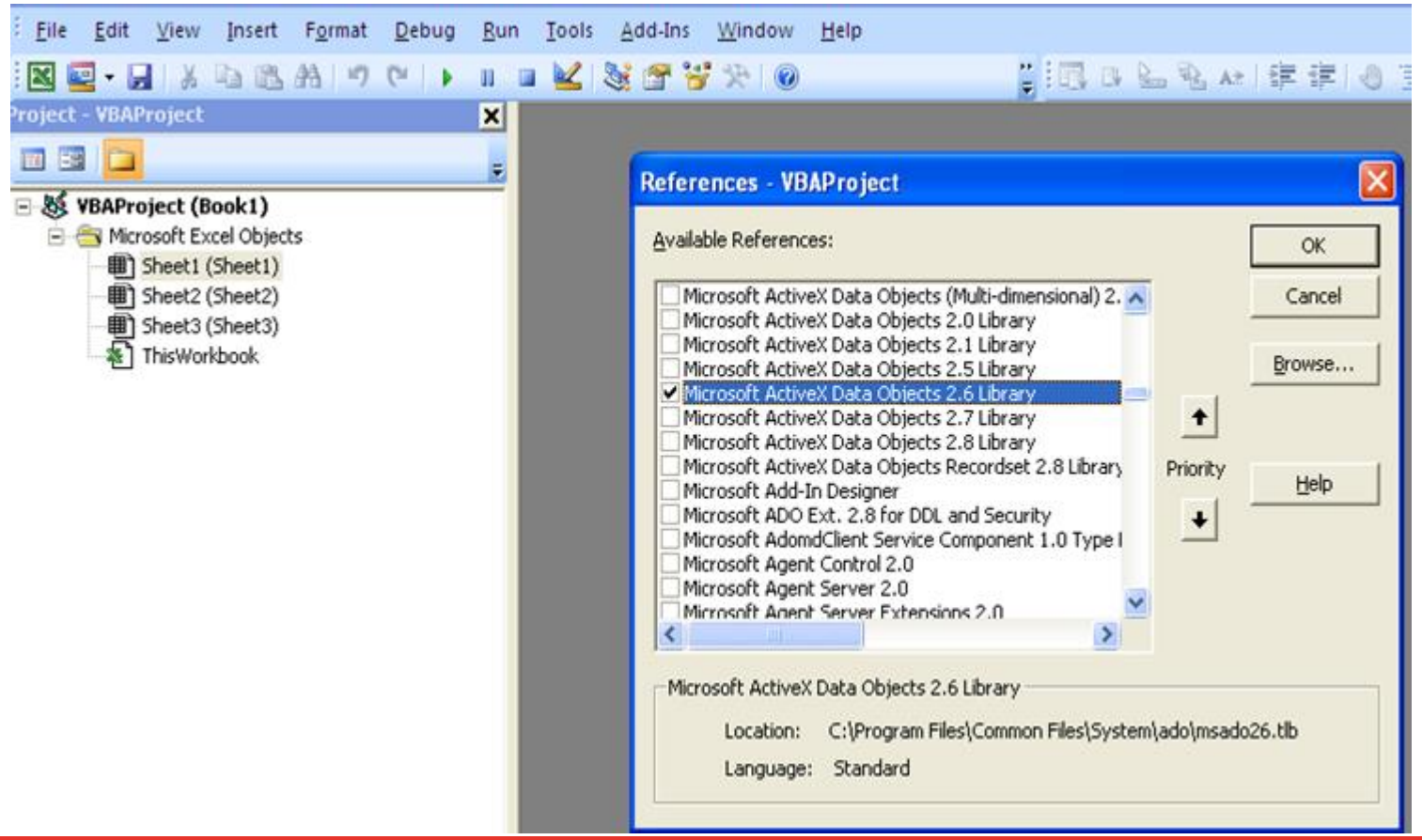
- Databases available
  - Access
  - SQL
  - Oracle
  - MYSQL
  - Sybase
  - DB2

We use **ActiveX Data Objects (ADO)** tool to connect to database.
Step 1: Add reference for Microsoft Activex Data Objects Library



Connecting to Database using VBA

Then select ” Microsoft Activex Data Objects Library” from the list.



Parameters to Connect Excel with Database

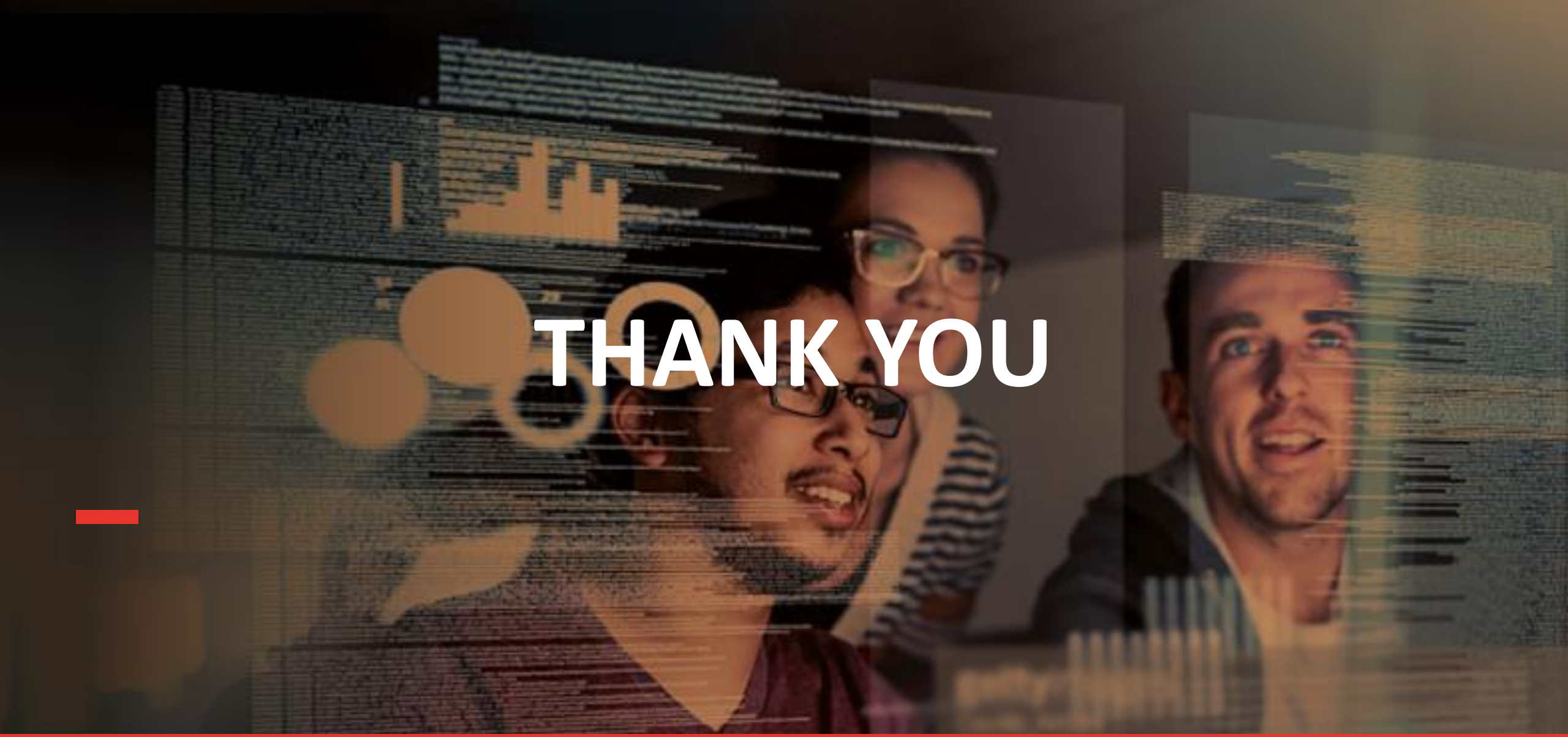
- Provider
  - Microsoft.Jet.OLEDB.4.0
- Data Source
  - Path of the Database and Database
- Database Password (Optional)
  - Password

Connection String

Code

```
Public Conn As New ADODB.Connection

Public Sub ConnectDatabase()
   Dim strConn As String
   'Close the Connection before opening
   If Conn.State <> adStateClosed Then Exit Sub
   Set Conn = New ADODB.Connection
   'Connection String
   strConn = “Provider=Microsoft.ACE.OLEDB.12.0;” &
         "Data Source = " & ThisWorkbook.Path &
"\Population.mdb;"
   'Open the Connection string
   Conn.Open strConn
   Exit Sub
End Sub
```

```
Sub GetEmp()
   Dim strQry As String
   Dim rsData As ADODB.Recordset
   Dim strID As String
   Dim wksTable as worksheet
   wksTable.Range("B4:G38").ClearContents
   Set rsData = New ADODB.Recordset
   strQry = "SELECT Population.* FROM
Population"
   ConnectDatabase
   rsData.Open strQry, Conn
   wksTable.Range("B4").CopyFromRecordset
rsData
   Set rsData = Nothing
End Sub
```



THANK YOU

Acadgild