Advanced Excel and Visual Basic for Applications — VBA

IOE 373 Lecture 9

Topics

- Objects
- VBA Basics
- Procedures
- MsgBox Function
- Basic Properties

Why learn VBA?

- It comes with Excel, Excel is the most popular business application available
  - VBA works inside Excel, no special software needed
- Easy access to Excel controls
- Custom dialog boxes
- Custom worksheet functions
- Custom user interface and menus
- Add-in files that can be run directly in excel

Objects

- Objects are Excel elements that can be manually manipulated or via a macro. Here are some examples of Excel objects:
  - The Excel application
  - A workbook
  - A worksheet in a workbook
  - A range or a table in a worksheet
  - A ListBox control on a UserForm (a custom dialog box)
  - A chart in a worksheet
  - A chart series in a chart
  - A data point in a chart
  - …

Names

- We can use names as identifiers that can refer to a cell, range, value, formula, or graphic object.
    - Formulas that use names are much easier to read than formulas that use cell references, and creating formulas that use named references is much easier.

Spreadsheet Apps

- Good spreadsheet applications have the following characteristics:
  - Enable the end user to perform a task that would be difficult from scratch.
  - Provide an appropriate solution to the problem. **(A spreadsheet environment isn't always the optimal approach.)**
  - Produce accurate results and is free of bugs.
  - Use appropriate and efficient methods and algorithms to accomplish its job.
  - Trap errors before the user is forced to deal with them.

Examples

- See attached Excel Apps…

Basics of VBA

- Code – performs actions in VBA. Code is stored in a VBA Module
  - VBA modules are stored in a workbook file, but you view/edit in the Visual Basic Editor (VBE)
  - Workbooks containing VBA code should be saved as .xlsm files

Procedures

- Unit of computer code that performs an action
  - Sub procedures – series of statements executable in different ways. Example:

```
Sub One()
        Sum=1+2
        MsgBox "The Sum of 1+2=" & Sum
End Sub
```

  - Function procedures – return values (or an array). Can be also called from within other procedures or as an excel function in your worksheet. Example:

```
Function AddThree (arg1, arg2, arg3)
        AddThree=arg1+arg2+arg3
End Function
```

Object Hierarchy

- When referring to an object, we have to follow object hierarchy by using dots (.) as separators between hierarchies. For example when referring to a workbook named lecture9.xlsx, we would use:
  - Application.Workbooks("lecture9.xlsx")
- Similarly, if we want to refer to sheet1 inside lecture9.xlsx:
  - Application.Workbooks("lecture9.xlsx").Worksheets("sheet1")
- And if we refer to cell A10 in sheet1:
  - Application.Workbooks("lecture9.xlsx").Worksheets("sheet1").Range("A10")

Active Objects

- To simplify specific references to an object, we can use shortcuts.
- If lecture9-code.xlsm is the active workbook then we can simplify the reference to cell A10 as:
  - Worksheets("sheet1").Range("A10")
- Similarly, if sheet1 is the active sheet in the workbook, we can also just refer to the cell directly:
  - Range("A10")

Object Properties

- Each object has properties (or settings). Examples:
  - Range objects properties: ***Value***, ***Address***
    - We can directly refer to the value in cell A10 of sheet1 with this statement: Worksheets("sheet1").Range("A10").Value

VBA Variables

- Can assign values to variables directly or from cell values:
  - Var1=3000+2000
  - Var1=Worksheets("sheet1").Range("A10")

Object Methods

- Methods are actions performed with the object (similar to clicking a key or button in excel)
- For example: Worksheets("sheet1").Range(“MyRange").ClearContents
  - This would clear the contents of cell A10 in sheet1

Events

- Applicable to some objects:
  - Opening a workbook triggers the event: Workbook_Open
  - Activating a workbook triggers the event: Workbook_Activate
  - Changing a cell in a worksheet triggers the event: Worksheet_Change

MsgBox Function

- The MsgBox function is one of the most useful VBA functions.
- Primarily used to display the value of a variable (and/or custom message).
- This function often is a great substitute for a simple custom dialog box.
- Excellent debugging tool because you can insert MsgBox functions at any time to pause your code and display the result of a calculation or variable assignment.

MsgBox Function

- Most functions return a single value.
  The MsgBox function returns a value and displays a dialog box that the user can respond to.
- Value returned by the MsgBox function represents the user's response to the dialog box.
- Also, use the MsgBox function even when you have no interest in the user's response but want to some type of message display.

MsgBox - Syntax

- MsgBox function has five arguments (those in square brackets are optional):
- MsgBox(prompt[, buttons][, title][, helpfile, context])
  - prompt: (Required) The message displayed in the pop-up display.
  - buttons: (Optional) specifies which buttons and icons, if any, to appear in the message box. Use built-in constants (e.g. vbYesNo)
  - title: (Optional) text that appears in the message box's title bar. Default is Microsoft Excel.
  - helpfile: (Optional) You can have a Help file associated with the message box.
  - context: (Optional) The context ID of the Help topic.

MsgBox

- The value returned can be assigned to a variable, or use the function by itself. Examples:

```
Ans = MsgBox("Continue?", vbYesNo +
vbQuestion, "Tell me")
If Ans = vbNo Then Exit Sub
```

  - Notice the use of the sum of the two built-in constants (vbYesNo + vbQuestion) for the buttons argument. Using vbYesNo displays two buttons: Yes and No. Adding vbQuestion displays a question mark icon. When the first statement is executed, The variable "Ans" contains one of two values, represented by the constant vbYes or vbNo. If the user clicks the No button, the procedure ends.

Example 1 – Writing a simple Sub

- Insert a VBA Module into a project
- Type/Copy the following code:

```
Sub SayHello()
    Msg = "Is your username: " & Application.UserName & "?"
    Ans = MsgBox(Msg, vbYesNo)
    If Ans = vbNo Then
        MsgBox "Oops, My Bad"
        Range("A1") = "User Unknown!"
    Else if Ans=vbYes
        MsgBox "I knew it!"
        Range("A1") = "User: " & Application.UserName
    End If
End Sub
```

```
Sub SayHello2()
Dim Msg as String
Dim Ans as Variant

Do
    Msg = "Is your username: " & Application.UserName & "?"
    Ans = MsgBox(Msg, vbYesNo)
    If Ans = vbNo Then
        MsgBox "Oops, My Bad"
        Range("A1").Value = InputBox("What is your Name?", , "User Unknown!")
    Else
        MsgBox "I knew it!"
        Range("A1").Value = "User: " & Application.UserName
    End If
Loop While Ans = vbNo And (Range("A1").Value = "User Unknown!" Or
Range("A1").Value = "")
End Sub
```

Example 1 - Testing the code

- Locate the cursor in the code window
- Press F5
- Run->Run Sub/UserForm

Example 1 – Recap

- This example makes use of the following:
  - Procedure declaration (Sub SayHello())
  - Value assignments to variables (Msg, Ans)
  - Concatenation of strings (with the & operator)
  - Use built-in VBA function (MsgBox). More comprehensive List: https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/msgbox-function
  - Use If-Then-Else statement. More comprehensive List: https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/ifthenelse-statement
  - Ending the Procedure (End Sub)

Application Properties

Property	**Object Returned**
ActiveCell	The active cell
ActiveChart	Active chart sheet or chart contained in a ChartObject or a worksheet
ActiveSheet	The active sheet, either a worksheet or a chart
ActiveWindow	The active window
ActiveWorkbook	The active workbook
Selection	Could be a Range object (cell or cells), Shape, ChartObject
ThisWorkbook	Workbook containing the VBA procedure in execution

Range Objects

- Two Common Syntaxes:
  - Object.Range(cell1)
  - Object.Range(cell1, cell2)
- Other examples:
  - Assigning a value to a specific cell: Worksheets("Sheet1").Range("A10").Value=15.2
  - Assigning a value to a named cell (e.g. Input): Worksheets("Sheet1").Range("Input").Value=150
  - Assigning a value to a specific range of cells: Worksheets("Sheet1").Range("A1:B10").Value=2

Cells Property

- Another way of referring to a range
- Cells property works like the Range property:
  - Object.Cells(rowIndex, columnIndex):
    - Worksheets("Sheet1").Cells(1,1)=10 (assigns a value of 10 to cell A1)
  - Object.Cells(rowIndex) – The argument in the parenthesis can vary from 1 to 17,179,869,184! In Excel 2010 each cell is numbered starting with cell A1 (1) going right and then down. So Cell A2 is the 16,385…
  - Object.Cells – This can be used when referring to a cell within a range: Range(“D5:E7").Cells(2,1)=2 would assign a value of 2 to cell D6

Offset Property

- Only applies to a Range object…
- Syntax: object.Offset(rowOffset,columnOffset)
- The 2 arguments correspond to the relative position from the upper-left cell of the range object. For example:
  - ActiveCell.Offset(1,0).Value=15 – This enters a value of 15 into the cell directly below the active cell
  - ActiveCell.Offset(-1,-1).Value=15 – This enters a value of 15 into the cell above and to the left of the active cell