ISM-3232
VBA for Excel

VBA stands for Visual Basic for Applications. VBA is the host language for writing macros within Microsoft Word, Excel, PowerPoint and Access. As business students, you will find it very useful to learn how to use VBA within Excel to automate certain tasks. In order to exploit the full potential of VBA in Excel, it is important that you are comfortable with both Excel as well as Visual Basic.

What is a Macro?

A macro, is essentially a computer program (a set of instructions) in a some programming language. A specific task in Excel can be automated by writing a macro. In Excel, macros are written in VBA. Even without any knowledge of Visual Basic, you can write simple macros by simply recording your keystrokes and mouse actions. For more complex tasks, which cannot be recorded through mouse clicks and keystrokes, knowledge of Visual Basic is essential. Once you have developed good skills, you can even combine recording a macro and adding your own code using Visual Basic. The task being automated may be as simple as formatting a cell to be Italicized, or as complex as an Accounts Receivable System. Once a macro is created, it can be executed by the click of a button or by a keystroke.

How to record and play a Macro?

You first need to make sure that you have the Developer tab in the Main Menu. If you don't have it, you can get it using File → Options → Customize Ribbon and then selecting the Developer tab. Once the Developer tab is on the main menu, you can click it to see the Developer Ribbon. On the Developer Ribbon, on the Code Panel (on your left), you will see a button for Record Macro. When you click on it you will see a dialog box in which you can specify the name of the macro, the optional shortcut key and a description. The shortcut key is some letter a through z that you can specify. Say you specify the letter k, then you can run this macro using Cntrl-k. You should not use letters that you frequently use for other purposes, such as c or v or x because you might be using Cntrl-c for "Copy", Cntrl-v for "Paste" and Cntrl-x for "Cut". When you click on OK on the Record Macro dialog box, you are in record mode. When this happens, the "Record Macro" button turns into a Stop Recording button. While in record mode, every keystroke and every mouse click is recorded. You can stop recording anytime by clicking the "stop recording" button. Excel writes VBA code for each recorded macro. Once recorded, macros can be run either by clicking on the Macros button and selecting the macro name or by the shortcut key (Cntrl + key).

Example 1 of a Macro:

Do the following exercise:
Record a macro (call it MyFirstMacro) to make the Style of a range of cells (Say B2:B8) to "Currency". Specify a short-cut key (say m) while recording the macro. Run the macro using the macro button and selecting the macro name and also by using the shortcut key (Ctrl + m). Run the same macro on a new sheet and study its effects. The macro is saved under Module1. Now study the code of this macro in VB Editor. The code should resemble the following code:

```
Range("B2:B8").Select
Selection.Style = "Currency"
```

Note the object-oriented nature of this code. Range("B2:B8") is the object with a method called 'Select'. Selection is an object with a property called 'Style' whose value is being assigned as "Currency" at run-time.

Example 2 of a Macro:

Record a macro (call it MySecondMacro), to adjust the column width of a column (Say column A). Use a shortcut key (say w). Run the macro on several sheets. Study the macro. The code should look something like this. The exact value of the columnwidth may vary.

```
Columns("A:A").Select
Selection.ColumnWidth = 3.14
```

What is Relative Referencing?

The macro in example 1 is only good for range B2:B8 of the current sheet while the macro in example 2 is good only for Column A of the current sheet. What if we want to format any group of seven cells or want to adjust the width of any column? For this, we need to use what is called "Relative Referencing". In the developer ribbon, just below the 'Record macro' button is the relative referencing button. When it is enabled, Excel records the macro a little differently. It records the macro so that the cells or columns affected are not absolute, but relative to the current position of the cell pointer.

The macro of Example 1 above (using relative referencing) looks like this:

```
ActiveCell.Offset(0, 0).Range("A1:A6").Select
Selection.Style = "Currency"
```

The macro for Example 2 above (using relative referencing) looks like this:

```
ActiveCell.Offset(0, 0).Columns("A:A").EntireColumn.Select
Selection.ColumnWidth = 3.14
```

Try running these two macros. Notice that the currency format and the column widths are imposed wherever your cell pointer is at the time of running the macros.

Beyond Recordable Macros

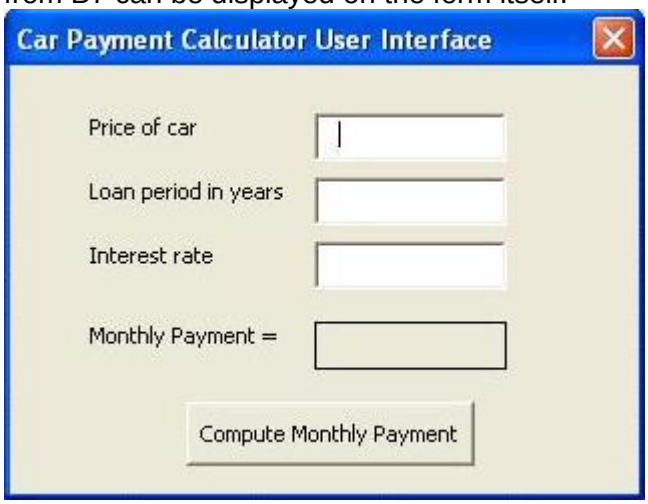
Recordable macros have limitations. They can only record your key strokes and mouse clicks. This limits our ability to program more complex tasks such as writing loops or if statements or creating form interfaces etc. But using Visual Basic, it is possible to write code to do basically anything you want to do. Let us study a few applications where we go beyond recordable macros.

Application 1: User Interface for a Car Payment calculator:

A common use of VBA in Excel is to design user interfaces to interact with an Excel application. Say we have a simple car payment calculator as follows:

	A	B
1	Car Payment Calculation Application	
2		
3	Price of car	=20000
4	Loan period	=5*12
5	Interest	=3/1200
6		
7	Monthly Pmt	=PMT(B5,B4,B3)
8		

In this application, the user is required to enter the price of the car in cell B3, the loan period in months in B4 and the interest rate per month in B5. The result is calculated in cell B7 using the PMT() function. VBA can be used to design a user interface form where the user can enter the input parameters and obtain the output. The interface will transfer the input parameters to cells B3,B4 and B5 and the result from B7 can be displayed on the form itself.




```
Private Sub btnComputeMonthlyPayment_Click()
   Range("B3").Value = txtPriceOfCar.Text
   Range("B4").Value = Val(txtLoanPeriodInYears.Text) * 12
   Range("B5").Value = Val(txtInterestRate.Text) / 1200
   lblMonthlyPayment.Caption = (-1) * Range("B7").Value
End Sub
```

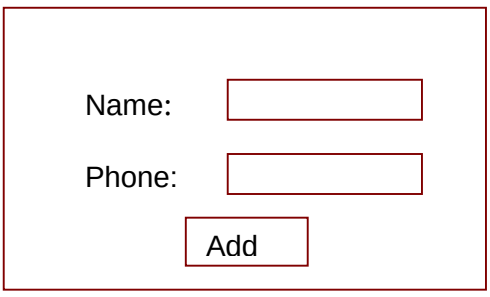
Using this interface, the user can calculate the monthly payment without interacting with the excel worksheet directly. Note that the actual calculations for monthly payment are performed in Excel; the form is used to simply pass input values to the correct cells in Excel and fetch the output value from the correct cell in Excel.

Application 2: Creating a database

Say we have a database for phone numbers. We will design a data entry application. Let us create a phone database on Sheet1 as follows in cells A2:B6

	A	B
1		
2	Name	Phone
3	John	345-1234
4	Carol	354-4321
5	Nathan	355-2345
6	Andrew	355-6543

Name cell A2 as PhoneDatabase. We will now create a data entry form for this database. Create a form with two text boxes, two labels and a command button as follows:



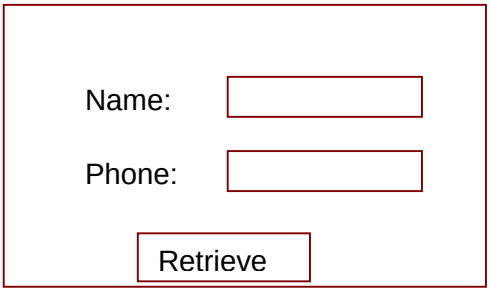
Write the following code under the Add button:

```
Sub btnAdd_Click()
   Application.Goto Reference:="PhoneDatabase"
   Selection.End(xlDown).Select
   ActiveCell.Offset(1, 0).Range("A1").Select
   ActiveCell.Value = txtName
   ActiveCell.Offset(0, 1).Range("A1").Select
   ActiveCell.Value = txtPhone
End Sub
```

This code will add a new row of data in the phone database. Study and understand the code. Note that exception handling code can also be added for validity checks.

Application 3: Retrieving data from a database

This application will retrieve data from the database. Let us create another form to retrieve data.



The user should be able to type a name and click the Retrieve button to obtain the phone number of the person.

```
Sub btnRetrieve_Click()
    Application.Goto reference:="PhoneDatabase"

    Do Until ActiveCell.Text = ""
       If ActiveCell.Text = txtName Then
          ActiveCell.Offset(0, 1).Select
          txtPhone = ActiveCell.Value
          Exit Sub
       Else
          ActiveCell.Offset(1, 0).Select
       End If
    Loop

    MsgBox "Sorry, no " & txtName & " in the database"

End Sub
```