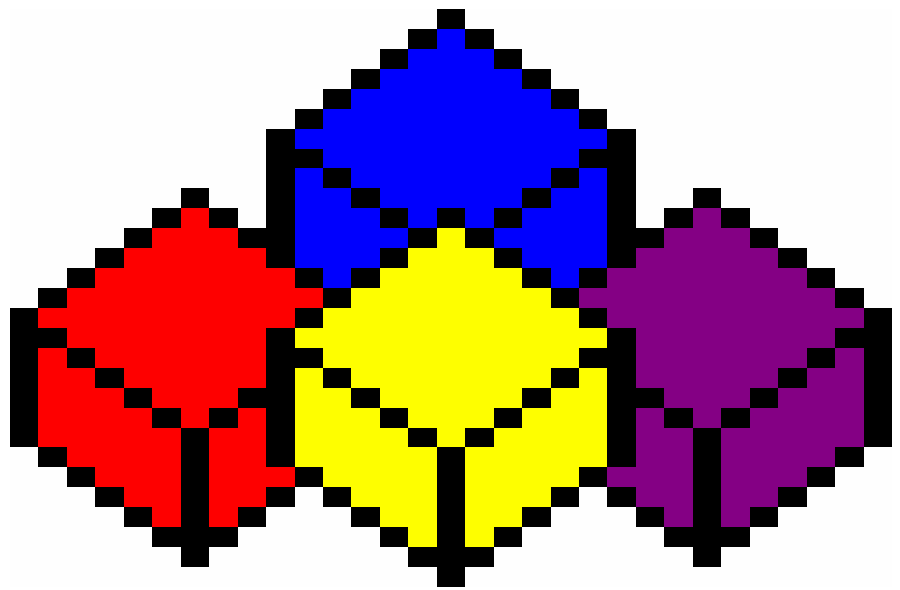


Excel

Macro Development
using

VBA



Marisa Dass
Computing Services
20 November, 2003

Acknowledgement

These notes are based upon course notes originally written by Fariborz Kiannejad, Computing Services, Queen Mary, University of London

Table of Contents

INTRODUCTION.....	1
WHAT IS A MACRO.....	1
WHAT IS VBA.....	1
VBA-RELATED TERMINOLOGY.....	1
CREATING A MACRO.....	4
RECORDING A MACRO.....	4
SPECIFYING WHERE TO STORE A MACRO.....	4
RUNNING A MACRO.....	5
EXAMINING AND EDITING A MACRO.....	6
CREATING A MESSAGE BOX.....	7
ASSIGNING A MACRO TO A BUTTON.....	9
ASSIGNING A SHORTCUT KEY TO A MACRO.....	10
ABSOLUTE AND RELATIVE REFERENCES.....	10
STEPPING THROUGH A MACRO.....	11
ENTERING VBA CODE MANUALLY.....	11
GOING FURTHER WITH MESSAGE BOXES AND INPUT BOXES.....	12
GOING FURTHER WITH ENTERING VBA CODE MANUALLY.....	13
USING HELP.....	13
ESSENTIAL VBA LANGUAGE ELEMENTS.....	14
COMMENTS.....	14
VARIABLES.....	15
DATA TYPES.....	15
SCOPE OF VARIABLES.....	16
CONSTANTS.....	16
ARRAYS.....	17
PROGRAM CONTROL.....	18
GoTo STATEMENT.....	18
IF...THEN STATEMENT.....	18
LOOPS.....	19
FOR STATEMENTS.....	20
DO STATEMENTS.....	20
WHILE STATEMENTS.....	21
WORKSHEET RANGES AND SOME USEFUL CODE.....	22
CREATING USER FORMS.....	23
APPENDIX A.....	27
SAMPLE CODE FOR EXERCISE 16.....	27
SAMPLE CODE FOR EXERCISE 19 – WHILE...WEND.....	27
SAMPLE CODE FOR EXERCISE 19 – DO...WHILE.....	27
SAMPLE CODE FOR EXERCISE 20.....	28

Introduction

What is a Macro

Excel is a powerful spreadsheet package containing many built-in functions as well as graphical data presentation facilities. The built-in functions in Excel are generally sufficient in most cases. However, Excel can also be used to automate tasks. In Excel, a macro is a module containing commands or functions, that is a series of actions used to automate a task. This has the following advantages:

- a) saving time in performing repeated actions;
- b) reducing the possibility of introducing errors into the spreadsheet;
- c) enabling users to perform complex operations in a single step.

For example, to calculate a monthly balance sheet for a company, or perform data analysis on multiple sets of experimental results, one of the following methods can be adopted:

1. apply the functions, formulas etc. for each set of data independently,
2. record all the required actions (functions, formulas) in an independent module (a macro) and apply this macro to any similar data set.

What is VBA

VBA corresponds to Visual Basic for Applications, and is a programming language developed by Microsoft. Many Microsoft applications already use VBA, such as Access, Excel, Project, Word.

It is worth noting that:

- ① VBA is not the same as VB (Visual Basic). VB is a programming language that creates stand-alone applications like Excel itself. VBA is embedded in another product, such as Excel. Hence, VBA provides the basic functionality of VB for programming facilities within Microsoft applications. Although VBA and VB have a lot in common, it is important to know that they are different.
- ① VBA is a programming language (like Fortran, Basic C), but it also serves as a macro language. Excel documentation refers to a program written in VBA and executed in Excel as a macro.
- ① Prior to the introduction of VBA, Excel (versions 2.0, 3.0 and 4.0) used another development language called XLM (Excel Macro Language). Later versions of Excel support both XLM and VBA.

VBA-Related Terminology

Functions

A function is a procedure that performs one or more actions in a specific order and returns a single value or an array of values.

For example, the following function (named “AddUp”) takes two arguments, adds them up and returns a single value (the sum of the arguments):

```
Function AddUp(num1, num2)
    AddUp = num1 + num2
End
```

Argument

An argument is a value that is passed to a function or a subroutine procedure. The procedure then uses this value to perform its task. For instance, an argument can be a number, text, cell, references, constants or even other functions. In the above example, the arguments are *num 1* and *num 2*.

Object

Excel provides over 100 objects which can be manipulated by using VBA code. Examples of objects include: a workbook, a worksheet, a range of cells on a worksheet, a toolbar. Each object can contain other objects. For example, Excel itself is an object called **application** and it contains other objects such as **workbook** objects and **toolbar** objects. The **workbook** object contains **worksheet** objects and **chart** objects.

Objects are arranged in a hierarchy. The term Object Model refers to the arrangement of these objects.

Subroutine

A subroutine is a procedure that performs a number of actions, including functions, on or with objects. A subroutine, unlike a function, does not return values, but performs some action. VBA modules consist of subroutine procedures.

① A VBA module always appears in the following format:

```
Sub Name1( )  
    Action 1  
    Action 2  
    .....  
End Sub  
Sub Name2( )  
    Action 3  
    Action 4  
    .....  
End Sub  
Etc.
```

Where: 'Sub' stands for subroutine

'Name' is user-defined

() contains arguments, if there are any

'Action' could be any operation, including a function.

Collection

Objects of the same type form a collection. For example, the **worksheets** collection consists of all the worksheets in a workbook and the **toolbar** collection consists of all the objects in a toolbar. Collections are themselves objects.

Reference to an object is made by specifying its position in the hierarchy, separated by dots. For example, reference to a cell, A1, in Sheet1 of a given workbook, Book1, in an application would be as follows:

```
Application.Workbooks("Book1").Worksheets("Sheet1").Range("A1")
```

① Active objects can be left out of the list. For example if Application and Workbook (Book1) are active the list can be reduced to:

```
Worksheets("Sheet1").Range("A1")
```

and if Sheet1 is also active:

```
Range("A1")
```

Properties

A property is a setting for an object. For example, a range object can have such properties as **value** or **name**. A chart object has such properties as **type**, **has legend**, etc.

A property of an object is referred to by combining the object's name with the property's name, separated by a dot.

For example, *Worksheets("Sheet1").Range("A1").Value*, refers to the value property of Cell A1.

Methods

A method is an action Excel performs with an object. For example, **ClearContents** is a method that can be performed with either a collection of cell objects or an individual cell, as follows: `Worksheets("Sheet1").Range("A1").ClearContents`.

Exercise 1

Objective: To enter data and manually perform actions on the data in a worksheet.

1. Open Excel.
2. Enter the following data into Sheet 1:

	A	B	C	D	E
1				Euro Tourist Rate	1.34
2					
3	Item	Value - £	Value - Euro		
4	Train Tickets	25			
5	Eating Out	150			
6	Shopping	200			
7	Sight-seeing	75			
8	Car Hire	80			
9	Souvenirs	40			
10	Miscellaneous	100			

3. Save the workbook as **Travel** in the 'My Documents' folder.
4. Enter a formula in cell C4 to calculate the equivalent value in Euros given the tourist rate in cell E1. **Hint:** Use an *absolute cell reference* to cell E1.
5. Copy the formula in cell C4 to cells C5:C10.
6. Save the file.
7. Select cells A3:A10, then **Edit** ☞ **Copy**.
8. Select Sheet 2, select cell A1 and paste the copied data **Edit** ☞ **Paste**.
9. Select cells C3:C10 from Sheet 1, then **Edit** ☞ **Copy**.
10. Select Sheet 2, select cell B1 and paste the copied data **Edit** ☞ **Paste**.

- ① This happens because the formula is updated to reflect the value in cell E1 of Sheet 2.

B
Value - Euro
0
0
0
0
0
0
0

11. To paste the actual values, select B1 (on sheet 2) then **Edit** ☞ **Paste Special** to get the Paste Special dialog box.

Choose the **Values** button and click **OK**.

Paste Special [?] [X]

Paste

All Validation

Formulas All except borders

Values Column widths

Formats Formulas and number formats

Comments Values and number formats

Operation

None Multiply

Add Divide

Subtract

Skip blanks Transpose

Paste Link OK Cancel

Creating a Macro

There are two ways of creating a macro:

- i) Automatically – By recording the actions performed using Excel's macro recorder.
- ii) Manually – By entering the code manually into a VBA module sheet.

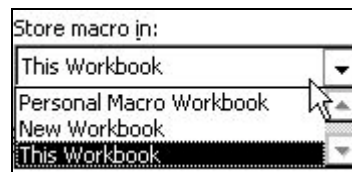
Both methods result in the creation of a macro module containing VBA code, which can then be tested, examined and edited.

Recording a Macro

Before recording a macro, plan the actions which are to be recorded. Remember that recording a macro will register every individual keystroke and mouse-click.

Specifying Where to Store a Macro

When a macro is being recorded, the user can choose where to store the macro.



For a macro to work:

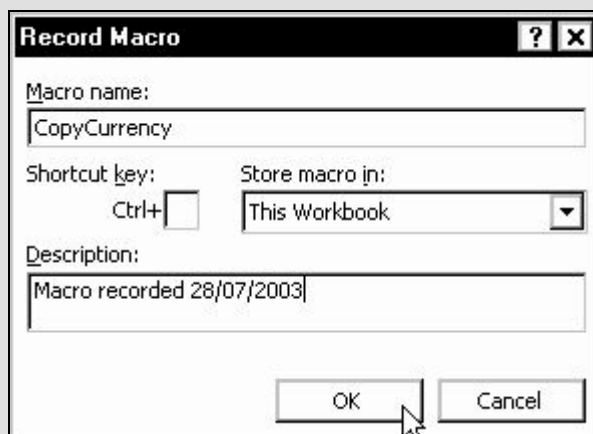
- Its workbook must be open, but not necessarily active
- Once the workbook in which the macro is saved is open, the macro can then be run from any workbook.

Exercise 2


Objectives: To record a macro which will:

- i) copy the item and Euro value columns into a different sheet;
- ii) format the relevant cells as currency, two decimal places with the sterling (£) or Euro (€) symbol.

1. Clear the entire contents of sheet 2 – select the entire sheet then **Edit** ⇨ **Clear** ⇨ **All**.
2. Select cell A1 on sheet 2.
3. Select sheet 1.
4. Choose **Tools** ⇨ **Macro** ⇨ **Record New Macro**.
5. In the Record macro window, enter the macro name – **CopyCurrency**. Then click **OK**.

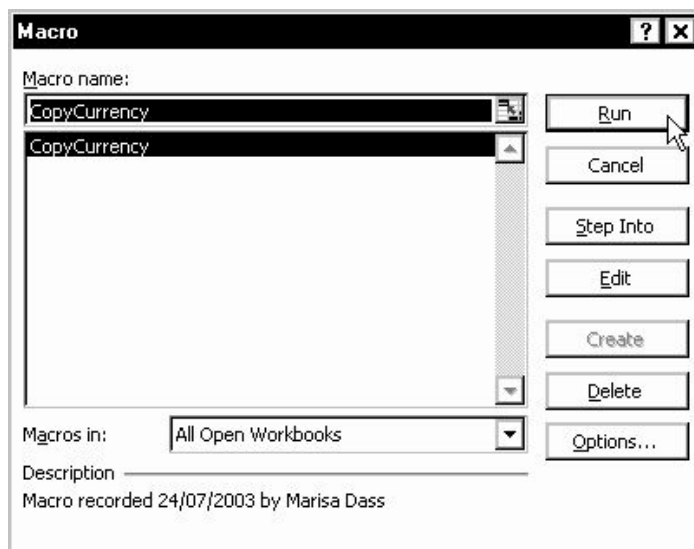


(CONTINUED ON NEXT PAGE)

6. All your actions will now be recorded until the Stop Recording button  is pressed.
7. Select sheet 2.
8. Type the word *Item* in cell A1.
9. Type *Value – Euro* in cell B1.
10. Go to sheet 1 and select cells A4:A10, then select **Edit** \Rightarrow **Copy**.
11. Go to sheet 2, select cell A2, then select **Edit** \Rightarrow **Paste**.
12. Return to sheet 1 and select cells C4:C10, then select **Edit** \Rightarrow **Copy**.
13. Go to sheet 2, select cell B2, then select **Edit** \Rightarrow **Paste Special** \Rightarrow **Values, OK**.
14. Select columns A and B, **Format** \Rightarrow **Column** \Rightarrow **Autofit Selection**.
15. Format cells B2:B8 as currency, two decimal places, with the Euro (€) symbol.
16. Click the **Stop Recording** button.

Running a Macro

To run an existing macro in Excel, select **Tools** \Rightarrow **Macro** \Rightarrow **Macros**, select the desired macro and click **Run**.



ⓘ Remember that a macro cannot be un-done like other Excel operations.

Exercise 3

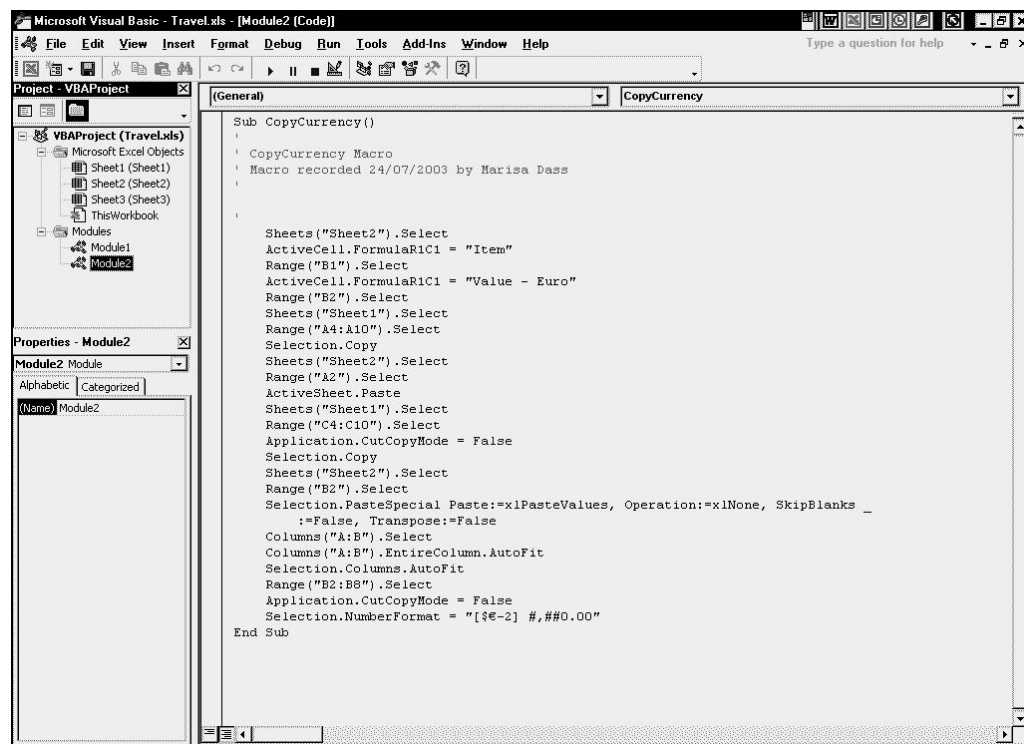
Objective: To run the CopyCurrency macro.

1. Clear the entire contents of sheet 2 – select the entire sheet then **Edit** \Rightarrow **Clear** \Rightarrow **All**.
2. Select cell B11 on sheet 2.
3. Run the macro – **Tools** \Rightarrow **Macro** \Rightarrow **Macros**, select CopyCurrency, and click **Run**.
4. Observe that the word *Item* appears in cell B11, which is the cell selected before applying the macro, rather than cell A1.
5. Clear sheet 2 again and try starting from a different cell before applying the macro.
6. The next section will address why this occurs.

	A	B
1		Value - Euro
2	Train Tickets	€ 33.50
3	Eating Out	€ 201.00
4	Shopping	€ 268.00
5	Sight-seeing	€ 100.50
6	Car Hire	€ 107.20
7	Souvenirs	€ 53.60
8	Miscellaneous	€ 134.00
9		
10		
11		Item
12		

Examining and Editing a Macro

When a macro is recorded, VBA code is created. This code can be examined and edited – **Tools** ➤ **Macro** ➤ **Macros**, select the desired macro and then click on **Edit**. This opens a Microsoft Visual Basic window which contains the VBA code:



The macro begins with a **Sub** statement, followed by the macro name “CopyCurrency”; next are the recorded actions, and the macro ends with an **End Sub** statement.

- ① Comments in a programming language are completely ignored by the program, and serve as an explanation of the code.
Comment lines begin with an apostrophe ‘.
- ① By default, Excel adds the information typed in the Record – New – Macro sheet at the beginning of the macro as comments.

Examine the code of the macro. Each line of code refers to one operation. The code can be edited by typing directly in the Visual Basic window.

Creating a Message Box

Message boxes are used to display a message in a dialog box. The user must then click a button to select a response, and based on the response the macro will perform different actions.

Exercise 4

Objectives: To edit the VBA code so that:

- i) the macro always starts at cell A1;
 - ii) a message box appears asking for confirmation prior to running the macro.
1. Look at the existing code and add an appropriate line of code so that the macro starts in cell A1.
 2. Type in the following code before the first line of code:
Answer = MsgBox("Copy currency values to sheet 2?", vbYesNo)
If Answer <> vbYes Then Exit Sub
 3. Save the VBA code – **File** ☞ **Save**.
 4. Close the Microsoft Visual basic window.
 5. Clear sheet 2 and run the macro "CopyCurrency".

The code typed in step 2 does the following:

- Display a message box with two buttons: *Yes* and *No*.
 - The user's choice is recorded in a variable called "**Answer**".
 - If the answer is not equal to "<>" Yes, then Excel exits the macro without executing it.
- ① The code entered in step 2 are examples of VBA code that cannot be recorded and can only be entered manually.

Exercise 5

Objective: To write a macro to calculate the number of holidays left for each member of staff and the cost if replaced by hourly staff.

1. Open a new workbook, and enter the following data:

	A	B	C	D	E
1		Holiday Entitlement	Holidays Taken	Holidays Left	Cost at £8.26 per Hour
2	Staff 1	30	2		

2. Create a macro named **Holidays** stored in a **new workbook**, which will:
 - i) calculate the number of holidays left for Staff 1;
 - ii) calculate the cost if Staff 1 was replaced by hourly paid staff during their holidays (assume 1 working day = 8 hours);
 - iii) format the cost as currency format.

Hint: Ensure that cell D2 is the active cell when beginning to record the macro.
3. Note that the Excel has created a new workbook to store the macro.
4. Starting at cell A3, enter the following data:

Staff 2	30	8
Staff 3	30	21
Staff 4	28	12
Staff 5	28	14

5. Select cell D3 and run the macro.
6. Observe that the cell E3 is not updated and that cell E2 is the active cell.
7. Examine the code for this macro: **Tools** ☞ **Macro** ☞ **Macros**, select the **Holidays** macro and then click on *Edit*.

(CONTINUED ON NEXT PAGE)

8. The following is the code for the **Holidays** macro:

```
Sub Holidays()  
'  
' Holidays Macro  
' Macro recorded 15/08/2003  
'  
'  
ActiveCell.FormulaR1C1 = "=RC[-2]-RC[-1]"  
Range("E2").Select  
ActiveCell.FormulaR1C1 = "=RC[-2]*8*8.26"  
Range("E2").Select  
Selection.Style = "Currency"  
End Sub
```

9. This code can be edited so that the cost is calculated for the appropriate staff member:

i) Replace the first occurrence of Range("E2").Select with **ActiveCell.Next.Select**

Ⓛ This selects the cell immediately to the right of the active cell and is equivalent to using the TAB key.

ii) Delete the second occurrence of Range("E2").Select.

10. Close the Microsoft Visual basic window.

11. Select cell D3 and run the macro again.

12. Follow the steps in Exercise 6 following to assign the **Holidays** macro to a button on the toolbar.

Ⓛ Note: **ActiveCell.FormulaR1C1** will return or set the value or formula of the currently selected cell.


R[x]C[y] will select a cell *x* rows downward (or $-x$ rows upward) and *y* columns to the right (or $-y$ columns to the left) of the currently selected cell.

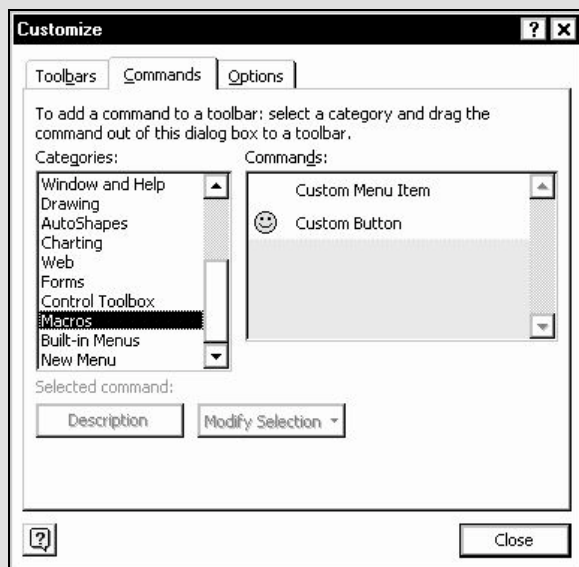
Assigning a Macro to a Button


A macro can be assigned as an item under a menu, or a button on a toolbar. This makes it easier to run the macro, by clicking the button on the toolbar or selecting the item under a menu.

Exercise 6

Objective: To assign a macro to a button on a toolbar.


1. Select **Tools**  **Customize**, then click the **Command** tab.
2. Under **Categories**, select **Macros**.

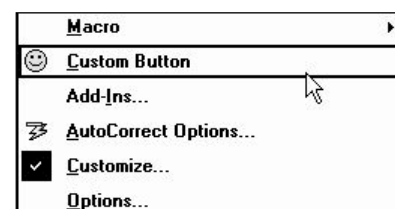


3. Under **Commands**, click and drag the **Custom Button** icon,  and position the icon at the end of the Formatting toolbar.
4. Right-click the Custom Button icon which was positioned on the toolbar.
5. Type in the macro name **Holidays** leaving the & symbol in place.



6. Click on **Assign Macro** and select the macro to be assigned to this button – in this case, **Holidays** – then click **OK**.
7. If desired, a different picture can be selected for the icon by clicking on **Change Button Image**.
8. Click the **Close** button in the Customize window.
9. Use this button to run the macro for the remaining staff members.

- ① Similarly, in step 3, the macro can be assigned as an item under a menu by dragging the **Custom Button** icon  to a menu and placing it below an existing menu item.



Assigning a Shortcut Key to a Macro

When first recording a new macro, it is possible to specify a shortcut key to be used to run the macro. **Tools** ➤ **Record New Macro**, then specify the shortcut key. For example, **Ctrl + I** can be assigned as a shortcut key so that when they are simultaneously pressed the macro will execute.



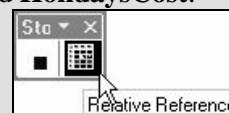
Absolute and Relative References

When a macro is recorded, absolute references to cells are used so that the location of a specific cell (for example, Range("E2")) is recorded on the macro. Alternatively, the **Offset** property can be used to record relative references to the active cell; for instance, 2 columns to the right and 1 row down.

Exercise 7

Objective: To re-do Exercise 5 using the Offset property.

1. Select cells D2:E6, then **Edit** ➤ **Clear** ➤ **All**.
2. Delete the existing **Holidays** macro: **Tools** ➤ **Macro** ➤ **Macros**, select the **Holidays** macro then click **Delete**. Confirm deletion of the **Holidays** macro.
3. Select cell D2.
4. Record a new macro: **Tools** ➤ **Macro** ➤ **Record New Macro**, named **HolidaysCost**.
5. Click the **Relative Reference** button, on the record macro toolbar:



6. Enter a formula in D2 to calculate the number of holidays left.
7. Enter a formula in E2 to calculate the cost if Staff 1 was replaced by hourly paid staff during their holidays (assume 1 working day = 8 hours).
8. Format cell E2 as currency format.
9. Click the **Stop Recording** button.
10. Select cell D3 and run the macro. Observe that the cell E3 is now correctly updated.
11. Examine the code for this macro: **Tools** ➤ **Macro** ➤ **Macros**, select the **HolidaysCost** macro and then click on **Edit**.

```
Sub HolidaysCost()  
  
' HolidaysCost Macro  
' Macro recorded 15/08/2003  
  
ActiveCell.FormulaR1C1 = "=RC[-2]-RC[-1]"  
ActiveCell.Offset(0, 1).Range("A1").Select  
ActiveCell.FormulaR1C1 = "=RC[-2]*8*8.26"  
ActiveCell.Select  
Selection.Style = "Currency"  
End Sub
```

12. Close the Microsoft Visual basic window.

Stepping Through a Macro

This is useful to see the effect of the VBA statements and also to debug the code.

Exercise 8

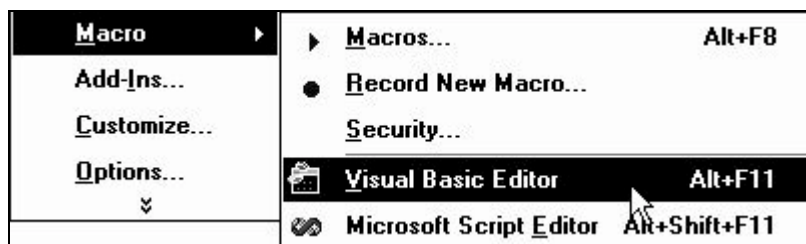
Objective: To step through the **HolidaysCost** macro.

1. Select cell D5.
2. Select **Tools** ➤ **Macro** ➤ **Macros**, highlight **HolidaysCost**, then click on **Step Into**.
3. Resize the windows so that both the Excel workbook and the code in the Visual Basic window are visible.
4. Press the F8 function key to step through the code line by line. Each line of code is highlighted when stepping through the code.
5. Observe the effect of each line of code on the worksheet as the code is stepped through.
6. Step through all lines of code in the macro.

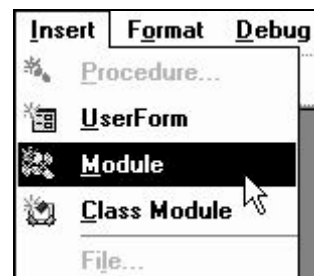
Entering VBA Code Manually

It is sometimes necessary to enter VBA code manually, rather than through the macro recorder. For example, entering the code for a message box. To create a macro in this way, first create an empty module sheet:

1. Select **Tools** ➤ **Macro** ➤ **Visual Basic Editor**:



2. In the Visual Basic book window, **Insert** ➤ **Module**:



3. The code can be entered in the module window which opens.

This VBA module sheet is identical to the one that is opened by the macro recorder. A single VBA module sheet can contain any number of subroutines, functions and declarations.

- This sheet behaves in almost the same way as a word-processing sheet.
- Text can be cut, copied and pasted.
- The layout of the text can be formatted using tabs.

A single VBA line can be as long as desired. However, it is better practise to make lines shorter and joined with continuation marks “_”.

Going Further with Message Boxes and Input Boxes

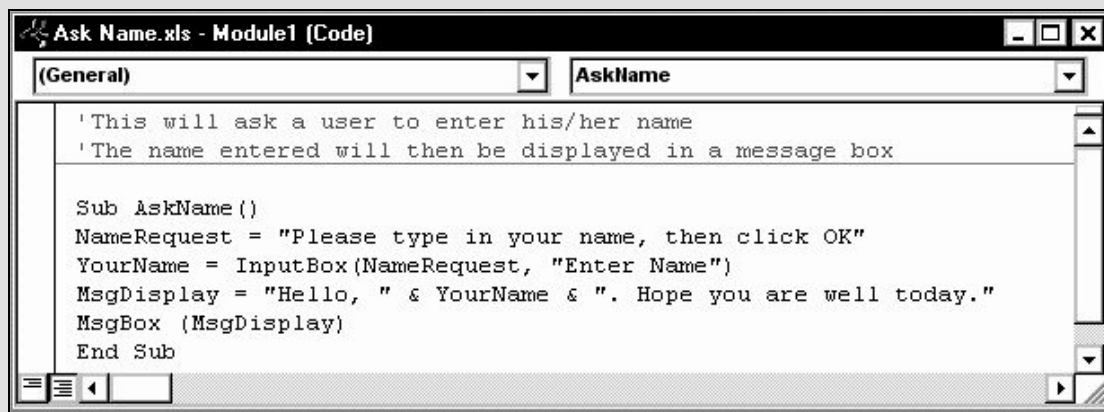
An input box is used to request information from a user. A dialog box is displayed prompting the user for information; the user must then input the information or click a button to proceed.

① This is another example of VBA code that cannot be recorded and can only be entered manually.

Exercise 9


Objective: To ask the user to input their name and then display the name in a message box.

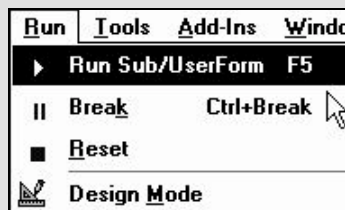
1. Enter the following code in to the module window:



```
' This will ask a user to enter his/her name
' The name entered will then be displayed in a message box

Sub AskName()
    NameRequest = "Please type in your name, then click OK"
    YourName = InputBox(NameRequest, "Enter Name")
    MsgBoxDisplay = "Hello, " & YourName & ". Hope you are well today."
    MsgBox (MsgBoxDisplay)
End Sub
```

2. On the third line of code “Enter Name” is the title of the input box and will be displayed at the top of the input box. This is optional and if omitted, the default title is “Input”.
3. Save the macro with the name **Ask Name** in the ‘My Documents’ folder.
4. Run the macro: **Run**  **Run Sub/User Form F5**.



5. If nothing is entered in the input box and the user clicks **Cancel**, a zero-length string ("") is returned.

Going Further with Entering VBA Code Manually

Exercise 10

Objective: To count the number of occurrences of a word in a given range of cells on a worksheet.

1. Open a new Excel workbook and enter the following data:


	A	B
1		
2		London
3		Reading
4		Bristol
5		Liverpool
6		Liverpool
7		Manchester
8		Brighton
9		London
10		Cardiff
11		Manchester
12		Brighton
13		Cardiff
14		Birmingham
15		Liverpool

2. Open a Visual Basic Module window and type in the following code:

```
' This macro counts the number of occurrences of the word Liverpool from cells B2:B15
Sub CountData()
Range("A1").Select
ActiveCell.FormulaR1C1 = "=countif(R[1]C[1]:R[14]C[1], " & """" & "Liverpool" & """" & ")"
End Sub
```

3. Save the macro as CountData.
4. Run the macro and observe the results in cell A1 of the workbook.
5. Add the macro as an item under the Tools menu.

Using Help

When using the Visual Basic Editor, help can be accessed at any time by pressing **F1** or via the menu: **Help**  **Microsoft Visual Basic Help**. Help can be gained on the various commands and the correct syntax.

Essential VBA Language elements

Comments

Comments refer to lines of information which play a descriptive role for the benefit of the user. They do not have any effect on the running of the program and could therefore be left out if so desired. Comments are therefore inserted into a program as a reminder of why something was done, or what a particular segment of the code does.

① Comments are useful for explaining a program.

In VBA comments begin with an apostrophe ('). Try the following exercise.

Exercise 11

Objective: To use comments

1. Open the Visual Basic Editor – **Tools** ☞ **Macro** ☞ **Visual Basic Editor** and start a new macro module.
2. Type in the following code:

```
Sub Comments()  
    'This subroutine does nothing of value  
    num = 0 'Assign a value of 0 to a variable named num  
    MsgBox (num) 'Display the value assigned to num in a message box  
End Sub
```

3. Run the code: **Run** ☞ **Run Sub/User Form F5**.

① An apostrophe within a set of quotation marks is not treated as a comment indicator. For example:

```
Msg = "This isn't a comment"
```

① A comment line can also be indicated using the keyword "Rem". For example:

```
Rem This is similar to a BASIC comment line
```

Exercise 12

Objective: To practise using more comments.

1. Type in the following code then run the code:

```
Sub MoreComments()  
    Rem – This subroutine assigns values to two variables  
    a = 200 'Assign value to 1st variable  
    b = 300 'Assign value to 2nd variable  
    c = a + b 'Add the two variables and assign the result to 3rd variable  
    MsgBox (a & " + " & b & " = " & c)  
    'Display the calculation and result  
    Rem – This may be too many comments  
End Sub
```


Object	4	Any Object reference
String	1 byte/ char	0 to approximately 2 billion (approximately 65,535 for Microsoft Windows version 3.1 and earlier)
Variant (with numbers)	16	Any numeric value up to the range of a Double
Variant (with characters)	22 + string length	Same range as for variable-length String
User-defined (using Type)	Number required by elements	The range of each element is the same as the range of its data type

VBA automatically specifies the Variant data type if a data type is not explicitly specified. However, letting VBA handle the data type selection will result in slower program execution and inefficient use of memory. To force declaration of all the variables used in a module, include the following statement as the first statement in the VBA code: *Option Explicit*

Scope of Variables

There can be more than one module in any workbook. Variables used in any module can be:

- Available to just one procedure
Declared by using a Dim or other declaration statements within the procedure
- Available to any procedure within the module
Declared by using Dim or Private before the first Sub statement in a module
- Available to all procedures in all modules
Declared by using a Public statement before the first Sub statement in a module

Example:

```
Public A as String
Dim W as integer
Private X as Boolean
Sub MySub1()
Dim Y as Date
'Program code here
End Sub

Sub MySub2()
Dim Z as Integer
'Program code here
End Sub
```

In this example variable A will be available to all modules in the workbook, Variables W and X to all subroutines in the module (that is MySub1 and MySub2), and variables Y and Z only to subroutines MySub1 and MySub2 respectively.

Constants

It is sometimes useful to declare constants rather than variables. A constant is an element whose value remains unchanged throughout the procedure, module or in fact the entire workbook.

Using constants could have a distinct advantage. For example, if a procedure needs to refer to a specific value, (e.g. interest rate), several times, it is better to declare the value as a constant and refer to the constant's name rather than the actual value. This would have the advantages of making the program more readable and easier to change.

Example:

```
Public Const Rate = 0.0725, Period = 12
Const X as Integer = 20
Sub MySub()
Dim CurrentPeriod as Integer
CurrentPeriod = Period - 1
'Program code here
End Sub
```

In the above example, the constant named 'Period' is referred to in MySub by its name rather than its value.

Arrays

All programming languages support arrays. An array is a group of variables which have a common name. A specific variable in an array is referred to by using the array name and an index number. For example, you might define an array of 52 integer variables, to hold the number of weeks of the year. If you name the array "Week", you can refer to the first element of the array as Week(0), the second as Week(1), and so on. Declaring an array is done in the same way as for other variables, using Dim, Public or Private. However, the size of the array must be specified. Array indexes usually begin at 0, unless otherwise specified, so:

```
Dim MyArray (9) as Integer
```

is a fixed-size, integer array with 10 rows.

Multi-dimensional arrays are also supported:

```
Dim MyArray (9, 4) as Integer
```

is a fixed-size, integer array with 10 rows and 5 columns. The first argument refers to the number of rows and the second argument refers to the number of columns.

Leave the parentheses empty to declare a dynamic array whose size can be changed while a program is running.

❶ To change VBA's default setting for an array index to begin from 1 instead of 0, use the following statement before any Sub statements:

```
Option Base 1
```

Exercise 13

Objective: To use an array

1. Start a new macro module, type then run following code:

```
Sub GetGrade()  
Dim Grades(3) As Variant 'Array with 4 elements  
  
'Set data for each element in the array  
Grades(0) = "A"  
Grades(1) = "B"  
Grades(2) = "C"  
Grades(3) = "F"  
  
PercentageScore = InputBox("Please Enter Your Percentage Score", "Get Grades")  
If PercentageScore <= 100 And PercentageScore >= 85 Then  
    MsgBox ("You have scored Grade " & Grades(0))  
ElseIf PercentageScore < 85 And PercentageScore >= 65 Then  
    MsgBox ("You have scored Grade " & Grades(1))  
ElseIf PercentageScore < 65 And PercentageScore >= 45 Then  
    MsgBox ("You have scored Grade " & Grades(2))  
ElseIf PercentageScore < 45 And PercentageScore >= 0 Then  
    MsgBox ("You have scored Grade " & Grades(3))  
Else  
    MsgBox ("Invalid Percentage Score Entered")  
End If  
End Sub
```

Program Control

Some VBA procedures start at the top and progress line by line to the end of the procedure, VBA modules produced using Excel's macro recorder are of this type. However, there are times when the control of the program needs to be controlled by skipping some statements, executing some statements multiple times and testing conditions to determine what the routine should do next. This is achieved by control statement such as GoTo, If, ElseIf.

GoTo Statement

A GoTo statement offers the most straight-forward means for changing the flow of a program. A GoTo statement unconditionally branches to a labelled line in the procedure. The syntax is as follows:

GoTo Label1

Exercise 14

Objective: To use the GoTo Statement

2. Start a new macro module, type then run following code:

```
Sub GoToName()  
    UserName = InputBox("Please enter your full name: ")  
    If UserName <> "Bill Gates" Then GoTo WrongName  
    MsgBox ("Welcome Bill...")  
    Exit Sub  
WrongName:  
    'This is the GoTo line label  
    MsgBox ("Sorry, only Bill Gates can enter.")  
End Sub
```

If ...Then Statement

This flow control structure in VBA will conditionally execute a statement or group of statements depending on whether specified criteria are met. The syntax is as follows:

```
If condition1 Then statement1  
Else If condition2 Then  
    statement2  
    statement3  
Else  
    statement 4  
End If
```

Exercise 15

Objective: To practice using an If Statement

1. Open a new Excel workbook and type the following data into Sheet 1:

	A	B	C
1	Name	Sales	Commission
2	Joe	10000	
3	Sam	7500	
4	Sara	12500	
5	John	9300	
6	Alex	6000	

(CONTINUED ON NEXT PAGE)

2. Open a Visual Basic module window and type in the following code:

```
Sub CalcCommission()  
'This procedure calculates the commission for each person based on their  
sales  
  
Dim SalesAmt As Currency  
Const Comm1 = 0.0175, Comm2 = 0.0125, Comm3 = 0.0075  
  
    ActiveCell.Select  
    SalesAmt = ActiveCell.Offset(0, -1)  
    If SalesAmt > 10000 Then  
        ActiveCell.FormulaR1C1 = SalesAmt * Comm1  
    ElseIf SalesAmt >= 7500 And SalesAmt <= 10000 Then  
        ActiveCell.FormulaR1C1 = SalesAmt * Comm2  
    Else 'Sales < 7500  
        ActiveCell.FormulaR1C1 = SalesAmt * Comm3  
    End If  
End Sub
```

3. Close the Visual Basic module window.
4. Click in cell C2 then run the macro – **Tools** ☞ **Macro** ☞ **Macros**, select CalcCommission, and click **Run**.
5. Observe that the sales commission is calculated as (10000 * 0.0125) since Joe's sales fall in the range >= 7500 and <= 10000.
6. Click in cell C3 and run the macro again.
7. Save this workbook as **Sales Commission** in the 'My Documents' folder.
8. Close the workbook.

① Note: **ActiveCell.Offset(R, C)** will select a cell **R** rows downward (or **-R** rows upward) and **C** columns to the right (or **-C** columns to the left), of the currently selected cell.

Exercise 16

Objective: To write a subroutine using an If Statement

1. Open a new Excel workbook.
2. Write a subroutine which will check the system time and display a message box as follows:
 - If the time is between 9:00 am and 11:59 am , message – “Good Morning” & Time
 - If the time is between 12:00 noon and 5:59 pm, message – “Good Afternoon” & Time
 - If the time is between 6:00 pm and 11:59 pm, message – “Good Night” & Time

Hints: Use the **Time** function which returns the current system time.
The time will be of the format **hh:mm:ss am/pm**, and is delimited by the # character.

See Appendix A for sample code

Loops

The term looping refers to the process of repeating a block of statements a known number of times or until a certain condition is met. Loops take the form of the following statements:

- For
- Do...Until, Do...While
- While...Wend

For Statements

This will repeat a block of statements a precise number of times.

The syntax is as follows:

```
For Counter1 = [Start] To [End] [Step]
    [Statements]
[Exit For]
For Counter2 = [Start] To [End] [Step]
    [Statements]
Next [Counter]
Next [Counter]
```

- ❶ Step – optional, defaults to one. This is the amount Counter increments each time through the loop.
- ❶ Exit For – is an alternate way to exit the loop and is often used with an ‘If...Then’ statement to pass control to the first statement outside the loop (that is, after Next).
- ❶ ‘For’ loops can also be nested, that is a ‘for’ loop within another ‘for’ loop.

Exercise 17

Objective: To practice using a For Statement

1. Type in and run the following 2 pieces code in a Visual Basic module window:

```
Sub FillRange()
    StartVal = CInt(InputBox("Enter a starting value:"))
    NumOfCells = CInt(InputBox("Enter number of cells here:"))
    For Counter = 1 To NumOfCells
        ActiveCell.Offset(Counter - 1, 0) = StartVal + Counter - 1
    Next Counter
End Sub

Sub FillCells()
    For Col = 3 To 5 'Col is a counter
        For Row = 1 To 15 'Row is a counter
            Cells(Row, Col) = Rnd 'Enter a random number into the cell
        Next Row
    Next Col
End Sub
```

- ❶ CInt – a function which converts an expression to an integer.
- ❶ Rnd – a function which returns a random number that is less than 1 but greater than or equal to 0.

Do Statements

This will repeat a block of statements *while* a certain condition is true or *until* a certain condition is met.

The syntax is as follows:

```
Do While/Until [Condition]
    [Statements]
[Exit Do]
Statements
Loop
```

- ❶ Exit Do – is an alternate way to exit the loop and is often used with an ‘If...Then’ statement to pass control to the first statement outside the loop.

Exercise 18

Objective: To practice using a Do...Until Statement

1. Enter the following data into column A:

	A
1	4
2	10
3	16
4	18
5	12
6	6
7	1
8	2
9	8
10	14
11	
12	5
13	8
14	7

2. Type in and run the following code in a Visual Basic module window:

```
Sub DoDemo()  
    Range("A1").Select  
    Do Until IsEmpty(ActiveCell) = True  
        ActiveCell.Offset(0, 1).Value = ActiveCell.Value * 2  
        If ActiveCell.Value = 1 Then  
            Exit Do  
        End If  
        ActiveCell.Offset(1, 0).Select  
    Loop  
End Sub
```

3. Clear the contents of cells B1:B7, change the value of cell A7 from '1' to '3' and run the macro again.

While Statements

This will repeat a block of statements while a certain condition is true.

The syntax is as follows:

```
While [Condition]  
    [Statements]  
Wend
```

Exercise 19

Objective: To write a subroutine using a While...Wend Statement

1. Open the **Sales Commission** Excel workbook which was created in Exercise 15.
2. Clear any data in cells C2:C6.
3. Re-write the CalcCommission macro using a While...Wend loop.

Note: A Do...While loop can also be used

See Appendix A for sample code

Exercise 20

1. Write a procedure to guess someone's age, based on the following algorithm:
 - Create a message box with two buttons – Yes, No, which asks if the person is younger than 16 years of age.
 - If the person is younger than 16 years, then go to a message box which informs them that they are too young to play this game.
 - Set a variable "age counter" to be equal to 16.
 - Create a message box with three buttons – Yes, No, Cancel, which asks if the person is "age counter" years of age.
 - If user clicks No, then increment "age counter" by 1, and ask if the person is "age counter" years of age. Repeat this process (loop) until the person responds with Yes or Cancel.
 - If user clicks Yes, display a message saying that the age was correctly guessed.
 - If user clicks Cancel, display a message saying that the procedure has given up trying.

See Appendix A for sample code

Worksheet Ranges and Some Useful Code

Most VBA programming involves worksheet ranges. When dealing with range objects, the following points should be noted:

1. It is not necessary to select ranges to work with them.
2. It is better to use named ranges rather than cell references. For example, use *Range("Total")* rather than *Range("D45")* as in the latter case, if a row is added before row 45 then the macro would need to be modified accordingly.
3. When working with ranges, an entire column or row may be selected.

Example 1: The following code will copy cells A1:C11 and paste the contents in cells E12:G22.

```
Sub CopyRange()  
    Range("A1:C11").Copy Range("E12:G22")  
End Sub
```


In some cases, the range of cells to be copied vary in row and column dimension. For example, in a workbook which keeps track of weekly sales, the number of rows with data may change weekly.

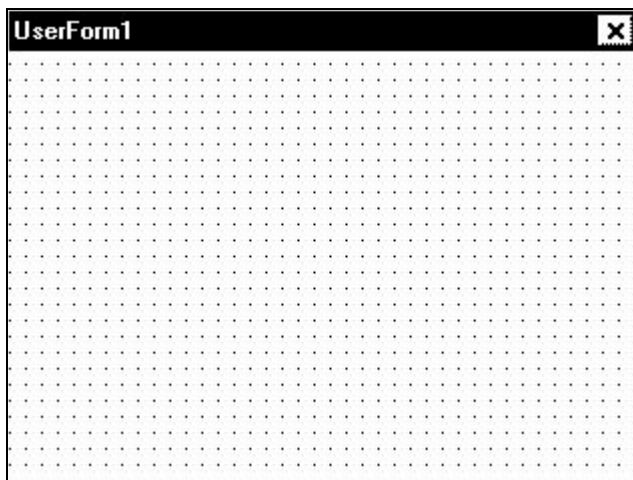
Example 2: The following code will copy the range of cells bounded by any combination of blank rows and blank columns (CurrentRegion) and paste the contents into a new worksheet.

```
Sub CopyRange()  
    Range("A1").CurrentRegion.Copy Worksheets("Sheet2").Range("A1")  
End Sub
```

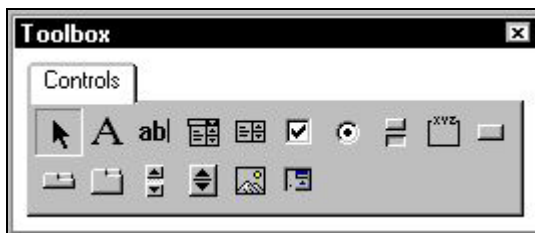
① Moving a range is similar to copying. Use the **Cut** instead of **Copy** command.

Creating User Forms

A user form allows you to create windows or dialog boxes. To create a user form, select **Insert**  **UserForm** from the Visual Basic window:



When a UserForm is activated, Excel displays the Controls toolbar, which can be used to add controls to the UserForm to receive user input, display output, and trigger event procedures. As the cursor is moved over each button on the toolbar, a description window displays the function of the button.






To use a UserForm, VBA code is written to perform the following:

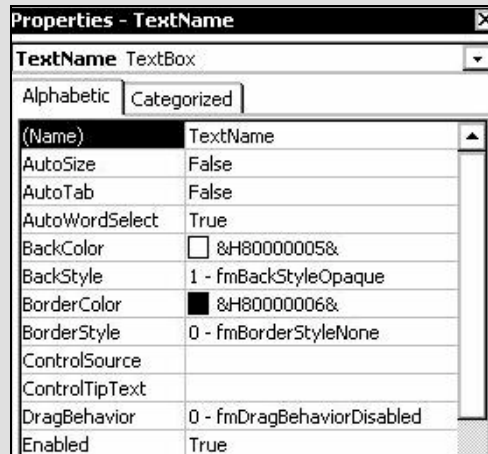
1. Initialise the UserForm controls.
2. Display the UserForm.
3. Take some action with the information supplied.

Exercise 21

Objective: To create a userform with controls as follows:

- A freeform text box to type in a name.
 - Two option buttons – Male and Female, grouped in a frame so that only one option can be selected.
 - An 'OK' command button, which will put the data entered on the form into the next available row in an Excel worksheet, and then clear the contents of the form.
 - A 'Cancel' button which will close the form.
1. Open the Visual Basic Editor – **Tools**  **Macro**  **Visual Basic Editor**.
 2. Insert a userform – **Insert**  **UserForm**.
 3. To insert a control, click the desired button on the controls toolbox, click inside the userform and drag to size the control. The control can also be re-positioned within the userform by dragging it to a different location.
 4. The properties for each control created will be displayed in a 'Properties Window' when the control is selected:

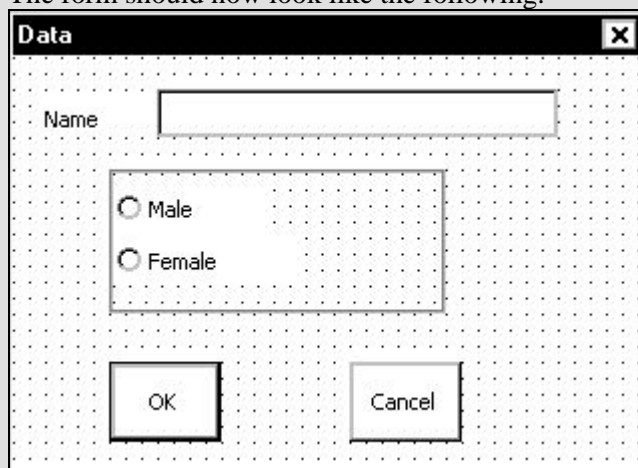
(CONTINUED ON NEXT PAGE)



5. Insert controls on the userform as follows:

Control Type	Property	Setting
User Form	<i>Name</i> <i>Caption</i>	UserForm1 Data
Label	<i>Name</i> <i>Caption</i>	LabelName Name
Text Box	<i>Name</i>	TextName
Frame	<i>Name</i> <i>Caption</i>	FrameGender
Option Button (positioned inside the frame)	<i>Name</i> <i>Caption</i>	OptionMale Male
Option Button (positioned inside the frame)	<i>Name</i> <i>Caption</i>	OptionFemale Female
Command Button	<i>Name</i> <i>Caption</i> <i>Default</i> (sets this button as the default command button on the form and is invoked if the Enter key on the keyboard is pressed)	CommandOK OK True
Command Button	<i>Name</i> <i>Caption</i> <i>Cancel</i> (sets this button as the cancel button on the form)	CommandCancel Cancel True

6. The form should now look like the following:



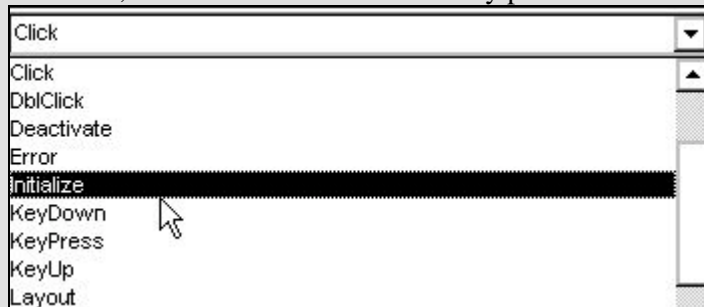
(CONTINUED ON NEXT PAGE)

7. As it stands, the userform created above will not do anything until code is written to run the various controls created.
8. Firstly, the form must be initialised when opened so that all the fields are empty. Ensure the userform is selected, then view the form's code – **View** ☞ **Code**.
9. The code is displayed in a window:

```
Private Sub UserForm_Click()

End Sub
```

10. Use the drop-down list of procedures at the top right of the code window to select **Initialize**, which will create the necessary procedure.



Delete the UserForm_Click procedure.

11. Type in the following code for UserForm_Initialize:

```
Private Sub UserForm_Initialize()

'Sets all fields to empty and places the cursor in the TextName box so the user can start typing
immediately

    TextName.Value = ""
    OptionMale.Value = False
    OptionFemale.Value = False
    TextName.SetFocus
End Sub
```

12. Next the 'Cancel' button must be coded so that it will close the userform when clicked.
13. Go back to the userform view: **Window** ☞ **UserForm1 (UserForm)**, and double-click the 'Cancel' command button on the userform to view its code.
14. Type in the following code:

```
Private Sub CommandCancel_Click()
    Unload Me
End Sub
```

15. The 'OK' button must also be coded so that it will put the data entered on the userform into the next empty row in an Excel worksheet, and then clear the contents of the userform.
16. Go back to the userform view: **Window** ☞ **UserForm1 (UserForm)**, and double-click the 'OK' command button on the userform to view its code.

(CONTINUED ON NEXT PAGE)

17. Type in the following code:

```
Private Sub CommandOK_Click()
    ActiveWorkbook.Worksheets("Sheet1").Activate
    Range("A1").Select
    Do Until IsEmpty(ActiveCell) = True
        If IsEmpty(ActiveCell) = False Then
            ActiveCell.Offset(1, 0).Select
        End If
    Loop
    ActiveCell.Value = TextName.Value
    If OptionMale = True Then
        ActiveCell.Offset(0, 1).Value = "Male"
    ElseIf OptionFemale = True Then
        ActiveCell.Offset(0, 1).Value = "Female"
    End If
    Range("A1").Select
    Call UserForm_Initialize
End Sub
```

18. Finally, a macro must now be created to run the userform. Create a new module **Insert Module** and type in the following code:

```
Sub OpenForm()
    UserForm1.Show
End Sub
```

19. Close the Visual Basic window, and save the file as **User Data** in the 'My Documents' folder.

20. Run the macro and enter data on the userform to observe how it works.

Appendix A

Sample Code for Exercise 16

```
Sub CheckTime()  
    If (Time >= #9:00:00 AM# And Time <= #11:59:00 AM#) Then  
        MsgBox ("Good Morning. The time is: " & Time)  
    ElseIf (Time >= #12:00:00 PM# And Time <= #5:59:00 PM#) Then  
        MsgBox ("Good Afternoon. The time is: " & Time)  
    ElseIf (Time >= #6:00:00 PM# And Time <= #11:59:00 PM#) Then  
        MsgBox ("Good Night. The time is: " & Time)  
    End If  
End Sub
```

Sample Code for Exercise 19 – While...Wend

```
Sub CalcCommission()  
    Dim SalesAmt As Currency  
    Const Comm1 = 0.0175, Comm2 = 0.0125, Comm3 = 0.0075  
  
    Range("C2").Select  
    While IsEmpty(ActiveCell.Offset(0, -1)) = False  
        SalesAmt = ActiveCell.Offset(0, -1)  
        If SalesAmt > 10000 Then  
            ActiveCell.FormulaR1C1 = SalesAmt * Comm1  
        ElseIf SalesAmt >= 7500 And SalesAmt <= 10000 Then  
            ActiveCell.FormulaR1C1 = SalesAmt * Comm2  
        Else 'Sales < 7500  
            ActiveCell.FormulaR1C1 = SalesAmt * Comm3  
        End If  
        ActiveCell.Offset(1, 0).Select  
    Wend  
End Sub
```

Sample Code for Exercise 19 – Do...While

```
Sub CalcCommission()  
    Dim SalesAmt As Currency  
    Const Comm1 = 0.0175, Comm2 = 0.0125, Comm3 = 0.0075  
  
    Range("C2").Select  
    Do While IsEmpty(ActiveCell.Offset(0, -1)) = False  
        SalesAmt = ActiveCell.Offset(0, -1)  
        If SalesAmt > 10000 Then  
            ActiveCell.FormulaR1C1 = SalesAmt * Comm1  
        ElseIf SalesAmt >= 7500 And SalesAmt <= 10000 Then  
            ActiveCell.FormulaR1C1 = SalesAmt * Comm2  
        Else 'Sales < 7500  
            ActiveCell.FormulaR1C1 = SalesAmt * Comm3  
        End If  
        ActiveCell.Offset(1, 0).Select  
    Loop  
End Sub
```

Sample Code for Exercise 20

```
Sub GuessAge()  
Dim AgeCounter As Byte 'Byte will store a number in the range 0 to 255  
  
    AgeRange = "Are you younger than 16 years?"  
    AgeAnswer = MsgBox(AgeRange, vbYesNo)  
    If AgeAnswer = vbYes Then GoTo NotValid  
  
    'Code to be executed if person is 16 years or over  
    AgeCounter = 16  
    Age = "Is your age " & AgeCounter & "?"  
    Answer = MsgBox(Age, vbYesNoCancel, "Age Guesser")  
    If Answer = vbNo Then  
        Do Until Answer = vbYes Or Answer = vbCancel  
            AgeCounter = AgeCounter + 1  
            Age = "Is your age " & AgeCounter & "?"  
            Answer = MsgBox(Age, vbYesNoCancel, "Age Guesser")  
        Loop  
    End If  
    If Answer = vbCancel Then MsgBox ("I give up :-( Bye")  
    If Answer = vbYes Then MsgBox ("Congratulations!!! I guessed correctly :-)")  
  
    NotValid:  
        MsgBox ("Sorry, you are too young to play this game.")  
End Sub
```