MS Office 2010

MS Excel VBA Manual

Table of Contents

Introduction

All the Office 2010 applications allow users to create their own Visual Basic code to carry out particular actions in the Application.

But why do you need to do this given that each application comes with a host of powerful features? The answer lies in how these features are used.

VBA is present in Word 2010, Excel 2010, PowerPoint 2010, Outlook 2010, Project 2010 and Publisher 2010.

Access 2010 and Visio 2010 create VBA code a little differently and will be covered in other Courses.

The purpose of this course is to give you the fundamental tools to start down the path of VBA programming in Microsoft Excel and to encourage you to further your knowledge beyond basic functionality; maybe you will use what you learn as the basis to explore VBA in other applications.

Whatever path you decide to take the basic VBA for excel will give you the tools at your fingertips to explore further.

Visual Basic Editor

Visual Basic for Applications (50 mins)

So what is visual basic for applications?

Visual Basic for Applications or VBA is a computer programming language which is used to control Microsoft Excel's functionality.

VBA can also be used to control Microsoft Word, PowerPoint or other programs.

All major Microsoft Office products come standard with VBA.

VBA controls Microsoft Excel by means of macros which are also called procedures.

In order to command Microsoft Excel effectively using Visual Basic for Applications (VBA), Microsoft Excel’s operational capabilities must be well understood along with its program elements.

What is the difference between VBA and VB?

VBA is a subset of VB which runs inside one of the office applications. As a result VBA inherits the current Office object library and application instance by default and any references that are included. However in VB you have to create the application instances if you need to manipulate one or more of the Office application objects

VBA Terminology

Before you start coding in VBA you need to be familiar with some key terms associated with it. The following table describes some of those terms.

Object	VBA object is something like a tool or a thing that has certain functions and properties, and can contain data. For example, an Excel Worksheet is an object, cell in a worksheet is an object, range of cells is an object, font of a cell is an object, a command button is an object, and a text box is an object and more.
Property	Each VBA object has its own properties that control its appearance. When we talk about range as an object typical properties are:- Column Width, Row Height, Font, Text, Value, Formula, Borders
Method	While most objects only provide characteristics to describe them, other objects can perform actions. For example, a house can be used to protect people when it is raining outside. In computer programming, an action that an object can perform is referred to as method.
Procedure	A procedure is a section of code created to carry an assignment, separate from a spread sheet, whose action can be used to complement a spread sheet. You cre6ate the procedure by writing code. One of the advantages of a procedure is that, once it exists, you can access it when necessary and

	as many times as you want.
Comment	A line of text within a procedure, that you can use to describe each line of code or the entire procedure. To comment a line out place an apostrophe at the beginning of the line. The comment will turn green.
Module	Is a file that you can write and edit blocks of code and other VBA code.

Collections and container Objects.

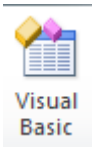
Collections are a set of related objects having the same properties.

Container objects are used to store and manipulate collections of data objects, allowing access to individual members of the collection in a simple and consistent manner container objects may or may not be related to each other.

The Visual Basic Editor

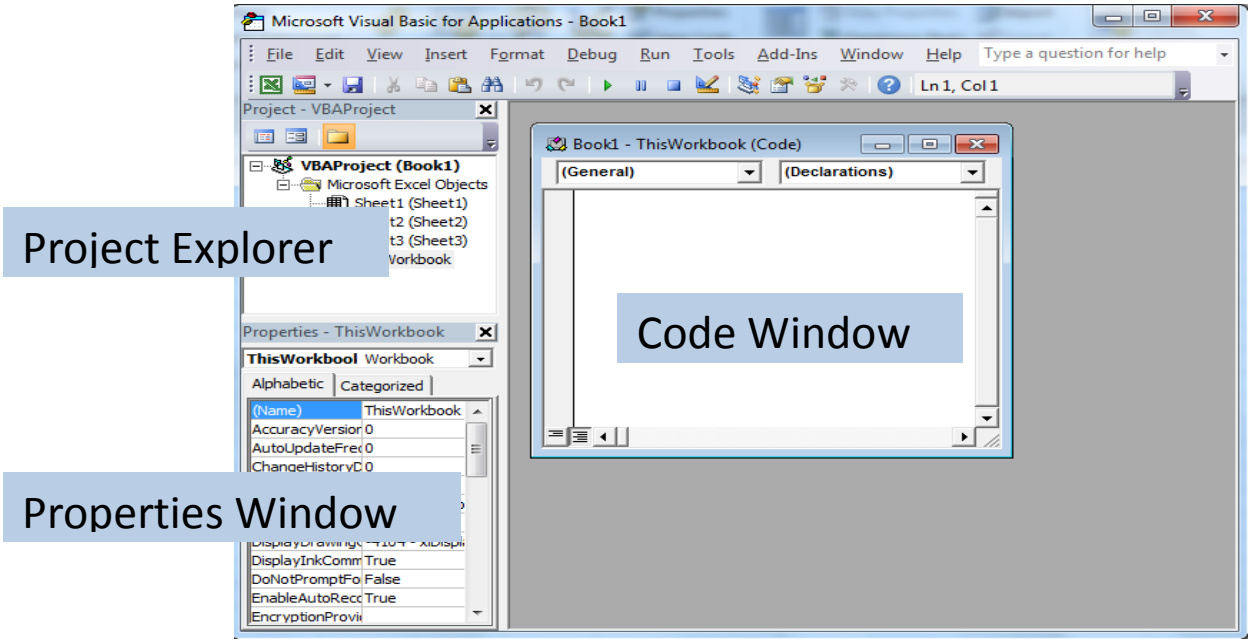
The VBE editor is used to create and edit VBA code from within the VBE editor you can create manipulate both properties and methods of an object. To access the VBE window for Excel the Excel application has to be open.

To launch the VBE editor from Excel you can do one of the following Click on the visual basic icon on the developer tab. The developer tab is not shown by default and you have to select it from: - File – options – Customise Ribbon.



As an alternative use the following keystroke. ALT + F11.

The VBE Window



The Project Explorer

This navigational aid displays a project for each workbook or template that is open in Excel. A project is a collection of modules the name of the project is the same that of the corresponding work book. Each work sheet can also contain Modules.

The Properties Window

This lists the properties of the selected object, you can change the properties for an object in this window.

The Code Window

As it suggests this is the place when you open a module you can create or edit the code view and compile the VBA code.

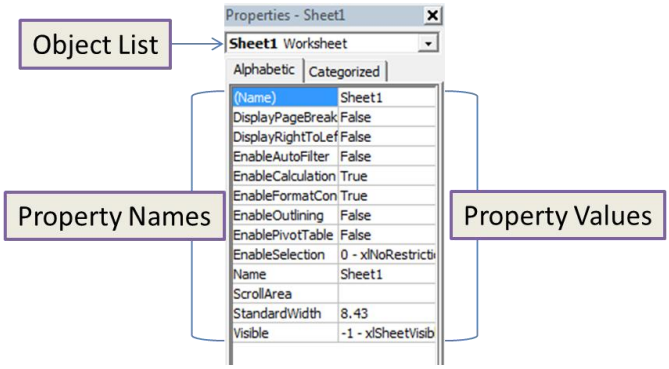
Object Programming

What is object programming?

In excel objects have certain properties, methods, and events. Properties control the appearance and other attributes of an object. Methods are built in procedures that you use to perform specific actions on an object. Events are actions such as mouse click double click or open and close a workbook. Most objects in VBA have events associated with them. For example a worksheet object has an event called activate.

The Properties Window

From the properties control panel you can change the behaviour and appearance of an object by modifying the properties in that panel. The name of the selected object appears in the object list pane, there are two tabs alphabetic and categorized. The alphabetic tab displays the property names alphabetically and categorized tab displays the properties grouped on the basis of the tasks they perform.



Modifying Properties

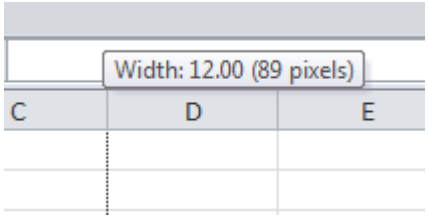
EXERCISE 1

- In the project explorer verify that sheet1 (sheet1) is selected
- In the properties window double click **name**
- Edit the value to read **purchase sales 2012**
- Press the **Enter** Button

In the project explorer the worksheet name changes to purchase sales 2012

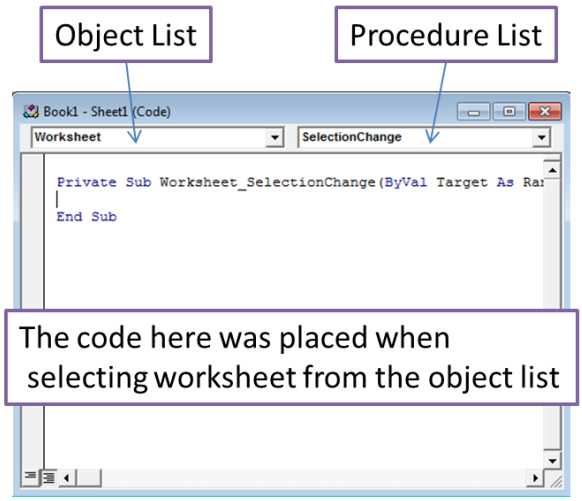
EXERCISE 2

- in the properties window, double click **standardwidth**
- enter **12**
- Select  to view the changes
- Point and Click between C & D a screen tip appears showing as 12



Using the Code Window

At times you may want to change the property of an object through code; each object that appears in the project explorer has a separate code window. To open a code window you double click the object or module in the project explorer.



The code window contains two lists, the object list and the procedure list. The object list displays all objects associated with the current module. The procedures list displays all the procedures in the current module or all the events of the object selected in the object list, the definition of the objects default procedure appears in the code window, for example if you select the object worksheet from the object list the following code appears in the window.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

End Sub
```

The code indicates that **Worksheet_SelectionChange**is the defaultevent of the worksheet object. This event occurs when the user selects a cell in the worksheet. Code written within this procedure will execute every time this event occurs. For example if you write code within this event procedure to display a specific message that message will appear every time you select a different cell.

Type in the following to give it a try**: MsgBox "you selected another cell"**

The general syntax for changing object properties through code is

Object.property = Value

EXAMPLE

```
Sub change_name()
    Worksheets("sheet1").Name = "myworksheet"
End Sub
```

Colours in the code window

Blue	Indicates Keywords that are reserved by VBA
Black	Indicates normal VBA code
Red	Indicates errors in code or the procedure failed to execute as intended
Green	Text prefixed with an apostrophe these are comments, and are ignored when you run the code

Using Methods

Every object can perform certain actions are defined by methods. Some methods need a value as input to complete their actions. For example, the open method of the workbook object takes a file name as input so it knows specifically what workbook to open. The input value is called an argument. An argument is a variable, constant or expression that provides additional information to a method, so that it can execute properly. To use a method in VBA code, you would use the following syntax.

Object.method argument1, argument2, argument3

For example to protect worksheet1 with the password “My Password”, you can use the following code.

Sheet1.protect “mypassword”

EXERCISE 1

Add password protection to sheet 2 in a work book

```
Sub mypasson()
Sheet2.protect "mypassword"
End Sub
```

EXERCISE 2

Remove password protection to sheet 2 in a work book

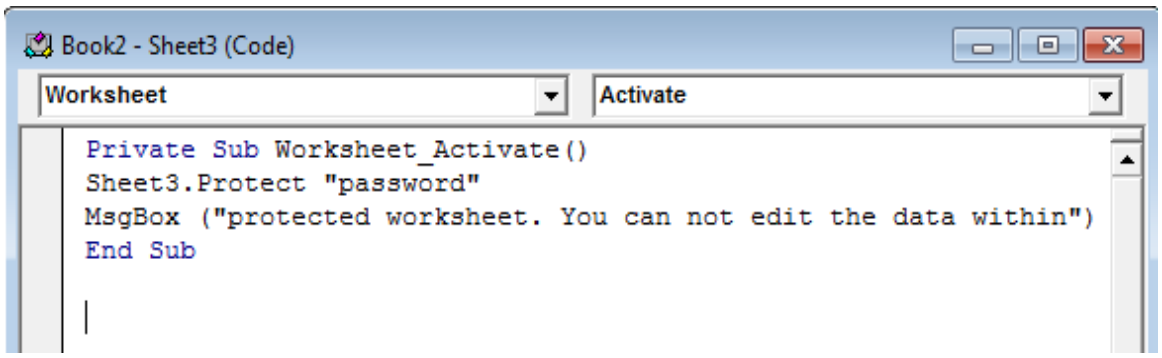
```
Sub mypassoff()
Sheet2.Unprotect "mypassword"
End Sub
```

Using Events

You might want a procedure to run in response to a specific user action. You can do this by associating the code with an event object. This association is created using an event procedure.

An event procedure is a code that is executed when an event occurs. For example, you can write code for the activate event of a worksheet to display a message indicating that you cannot change the data in the worksheet this procedure will activate when the user activates the worksheet.

To program an event, double click the object to display a code window. Select an event from the procedure list and enter the code, the code will run automatically when a user triggers the event for that procedure.



EXERCISE

Switch to the VBE Window

In the project explorer verify that sheet3 is selected

In the Code window, from the object list, select **Worksheet** (left)as shown above from the procedures list select **Activate** (right)

Type the following:

Sheet3.protect “password”

MSGBOX (“protected Worksheet. You cannot edit the data in the list.”)

Switch to excel

Click on sheet3 to see the results

Object-orientated programming overview

Below shows a schematic on how objects properties events and methods are positioned in the programming hierarchy where the car is the object.

Object

Properties

Make

Model

Colour

Year

Price

Events

On-start

On-Parked

On-Brake

Methods

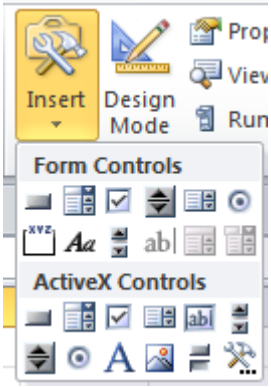
Start

Drive

Park

Using Buttons

The following procedure shows you how to add a button to the worksheet and assign code to that button.



From the ribbon bar navigate to the developers tab, from the controls group select insert then click on the button icon. Draw the button to the size required.

Next the macros dialog box opens, in the macro name box type in calculate, next click on new. Insert the code below in the code window.

```
Sub Calculate()
ActiveCell.FormulaR1C1 = "=Product(rc[-2]:rc[-1])"
End Sub
```

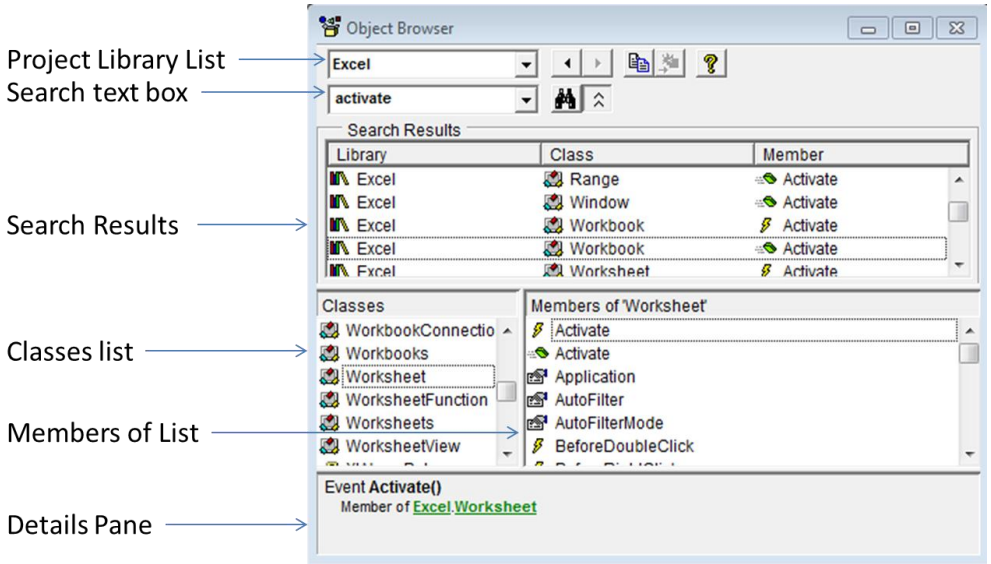
The code when executed adds two numbers to the left of a highlighted cell. Later on we will look at the looping stamen to automatically add up all values in the columns.

The Object Browser

When using code in VBA you will need to identify the methods or properties for an object you may wish to use, the object browser can be used to obtain this information, included in the object browser is a search facility to allow you to search for information on objects, methods, properties or events.

The object Browser

As already discussed the object browser is a window that displays the classes, properties, methods events, and constants in the various object library, below shows the result of the excel library being chosen, searching for the activate method that shows the classes (objects) available. Below shows the activate event for the worksheet object.



The various elements in the object browser are:

- **Project/Library list** shows the names of all the projects and object library's
- **Search Text Box**, accepts text strings and searches for information related to the text typed in.
- **Search results** shows the results of the information typed into the search text box
- **Classes list** shows the classes that are available depending on what has been selected from the project library list.
- **Members of list**, shows the methods and properties that belong to a particular object.
- **Details Pane** shows the definition and syntax of a selected method, property or event.

The object Browser icons

	Object / Class
	Method
	Event
	Properties
	Help Button
	Search Button

The search Feature

To search for a project, workbook or object enter its name in the search text box and then click the search button, the results are shown in the search results panel.

Unit summary

In this unit you have learnt that VBA is a programming language been introduced to the Visual basic Editor, Terminology use in VBA and gained an understanding of how objects, properties, methods and events play a role in Object-orientated programming.

You have learnt how to modify properties and use methods by using the code window, you also learned how to associate code to an event, and finally in this section you added a button to execute some VBA code.

In the last part of this section we looked at how the object browser worked, familiarised ourselves with the interface and icon sets and performed a search to find out information about properties and methods.

☞ Notes:

Programming Basics

Data

Programs receive data as input then process that data to generate output, a good example of this is a calculator where you input numbers, instruct it which operator to use and it then returns an output as the answer. When inputting data the program uses a temporary storage space called a variable, the variable consists of a name and data type. The name is used to identify itself in the program and the data type indicates the type of data to be stored. In VBA it's not mandatory to specify the data type if you don't it will assign a type called variant, however specifying data types makes the code run more efficiently.

Data Types

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

Variables

We have established that variables are used in VBA to store data and the name given to it uniquely identifies the variable in the computer memory. For example in a procedure that calculates a commission value, you would create a variable to contain that value.

The process of defining a variable and its data type is called declaration; there are two types of declaration in VBA either implicitly or explicitly.

Implicit Declaration

You can use a variable without declaring it. This is called implicit declaration, consider the following code.

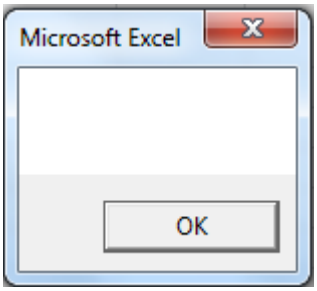
Answer = 100 + 100

In the code above we have a variable called **Answer** this holds the sum of the two answers.

Enter the code below into a new module

```
Sub calc()
Answer = 100 + 100
MsgBox (Anser)
End Sub
```

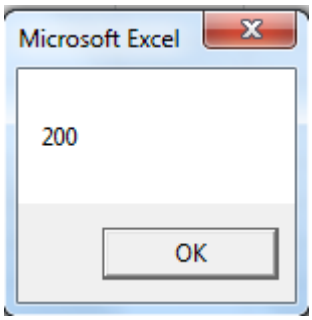
When you run this code it launches the macro box, now click on the macro **calc** and run.



As you can see the display box is empty, this is because of a spelling mistake, and the spelling mistake for the variable is not recognised because we have chosen to run implicitly.

Now change the code to read as below

```
Sub calc()
Answer = 100 + 100
MsgBox (Answer)
End Sub
```



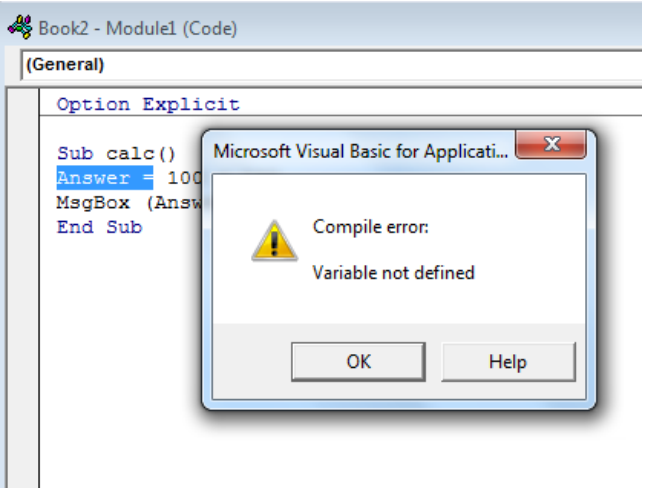
When you run this code note the result is now displayed.

So from this small demonstration we can identify two basic flaws in using implicit declarations.

1. Prone to errors and spelling mistakes causing the code to fail when executed.
2. By default the variant data type is assigned whilst declaring it as an integer would be far more efficient.

Explicit Declaration

As you have discovered implicit declarations are open to errors in the code, to prevent errors and control the type of variable used we need to set an explicit Declaration. To make sure this action takes place type **Option Explicit** at the top of the code window as shown below. With your cursor inside the code press F5 to run the code as you can see from the dialog box we get a compile error complaining the variable is not found.



To make this code run we have to declare the variables add the following code line to the sub routine. **Dim answer As Integer** the code will now run to display the correct result. You can declare more than one variable at a time, for example Employee surname and salary amount as.

Dim Salary_ammount as Double, Employee_name as string

Constants

A constant is a special type of variable that holds static data Tax rates commission rates and scientific constants are good uses for this variable to be used. To declare a constant use following syntax const <constant_name> = Value, for example setting the current vat rate as a constant may be written as **Const VAT=0.20**

Option Explicit

Const VAT = 0.2

Sub calc()

Dim answer As Integer

Answer = (100 + 100) * VAT

MsgBox (answer)

End Sub

Type in the code above this will result in 40 being displayed in the dialog box.

Expressions

An expression is a combination of operators, constants, procedures and names of controls and properties that is evaluated to obtain a result. When you use an expression in code it returns a value in one of the data types provided by VBA.

Operators

Operators are arithmetic symbols

Arithmetic Operators		**Comparison Operators**	
+	Addition	<	Less than
-	Subtraction	>	Great than
/	Division	=	Equal to
*	Multiplication	<>	Not equal to
A comparison operator returns a Boolean value because the result of any comparison is always true or false			

The Cells Object

Whilst we us arithmetic or comparison operators with values contained in a worksheet, we use the **cells** object when referring to in individual cell in a worksheet.

Use the syntax as follows **cells (row, column)** operates similar to the index function the main difference between the cell object function is unlike the excel functions that call column first then row, the cell object calls the row first then the column.

The code below pastes 2012 into cell **D2**

```
Sub Push()
Cells(2, 4).Value = 2012
End Sub
```

To retrieve a value in a cell we could use a variable **myvar**in the following syntax line

Myvar = cells(2,4).value

So you can see this working we have added a message Box

```
Sub Pull()
Dim myvar As Integer
myvar = Cells(2, 4).Value
MsgBox (myvar)
End Sub
```

The two examples above show how the push code places code into a cell and that the pull function extracts data from a cell.

Functions

If you have got to the stage of learning VBA then you will be proficient already in excel and would have used many of the inbuilt functions, functions always return results.

User interaction functions

These functions help to accept user input or display output the two most common user interface functions are the **inputbox** and the **msgbox**

You can use the **input box** function to get data from the user, use the following syntax.

Dim employeename as string

Employeename = inputbox (“enter Name”)

Because we set up a variable we can use the data later.

Message boxes display exactly that messages text or numbers these can be code written or from variables as shown below.

MsgBox (employee_name)

MsgBox "Warning cold spell", vbCritical, "Weather Warning"

Type in the following into the code window

```
Sub testbox()
Dim employee_name As String
employee_name = InputBox("enter Name")
Cells(2, 4).Value = employee_name
MsgBox (employee_name)
MsgBox "Warning cold spell", vbCritical, "Weather Warning"
End Sub
```

This sets a variable employee to accept text

You are then used to enter your name into an input box

The contents of the input box are transferred to cell D2

First message box displays your name in a dialog box

Second message box displays a dialog bow with weather warning and warning cold spell with a big red X in a dialog box.

Concatenating text

You can use the message box function to display text along with the value stored in a variable. To combine variables with text you can use the concatenation operator ampersand (&). The general syntax for using the concatenation operator ampersand is:

Msgbox (“message_text” &<variable_name>

For example to display the message “the amount is”, along with the value that is stored in the variable amount, the code is:

Msgbox (“the amount is:”& amount)

Type in the text below in the code window

```
Sub join()
Dim Amount As Integer
Amount = InputBox("enter a value")
MsgBox "The Amount is:" & "£" & amount
End Sub
```

The code above takes the value you enter in an input box, then displays that as **The amount is £90** that is on the assumption you have typed 90 into the input box.

Try typing in the code below comment out each line to describe what it does.

```
Sub usingfunctions()
Dim saleseast As Integer, saleswest As Integer, sum As Integer
MsgBox "please enter whole numbers only"
saleseast = InputBox("enter the total sales for the east")
saleswest = InputBox("enter the total sales for the west")
sum = saleseast + saleswest
Cells(2, 4).Value = sum
MsgBox "total sales for east and West: £" & sum
End Sub
```

Object Variables

In another example of a variable we can define an object variable in the example shown we are going to change the colour of the tab colour of two worksheets one to red and one to green, using object variables is discussed in depth in a later course.

Example 1 using a variable

```
Sub changewkstabcolour ()
    Dim wks As Worksheet
        Set wks = ThisWorkbook.Worksheets ("sales2011")
        wks.Tab.Color = vbRed
        Set wks = ThisWorkbook.Worksheets ("sales2012")
        wks.Tab.Color = vbGreen
  End Sub
```

The code above sets variable WKS as a worksheet object, worksheets are a member of the workbook family, and inside the parentheses we define the worksheet name. The property of the tab is changed by referencing the object variable and setting the property to the appropriate colour.

Example 2 define the sheet to change

```
Sub changetabsales ()
  ThisWorkbook.Worksheets ("sales").Tab.Color = vbBlue
  ThisWorkbook.Worksheets ("sales2011").Tab.Color = vbGreen
  ThisWorkbook.Worksheets ("sales2012").Tab.Color = vbRed
End Sub
```

Example 3 change the active worksheet

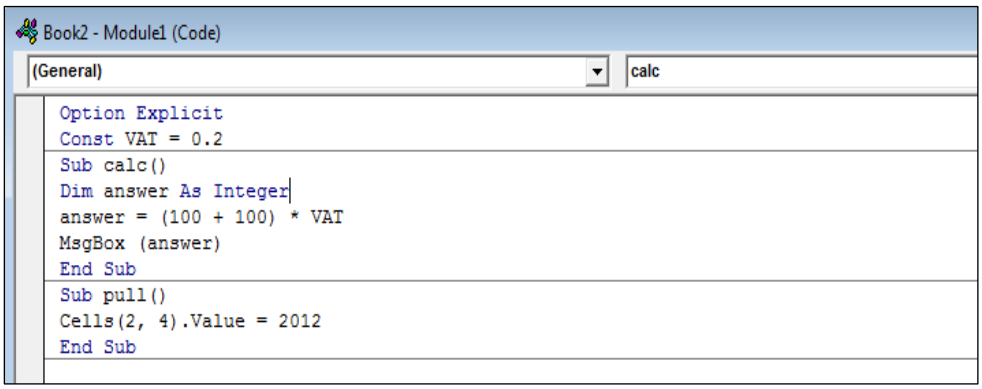
```
Sub changecol ()
  Sheets (ActiveSheet.Name).Select
  ActiveWorkbook.Sheets (ActiveSheet.Name).Tab.ColorIndex = 21
End Sub
```

Scope of Variables

Having set up variables it may become apparent that you need to use it in more than one procedure, rather than copy to each individual module we can broaden its scope or put another way its usability. The accessibility or scope is determined where the variable has been declared.

Declarations Section

The top of the code window includes a declaration section; variables placed here can be used by all procedures within the module remember option explicit is also set in this area.



Types of Scope

Three types of scope are available in VBA as already stated the scope of a variable is determined by the way you declare it.

Procedure-level, when you declare a variable within a procedure it is not accessible outside of the procedure. A procedure level variable is only available within the procedure it is written.

Private module-Level, when declaring a variable in the declaration section of a module using **Dim or Private** keyword the variable is known as a private module-level variable, the variable can be used by any procedure within the module but cannot be accessed by a procedure outside the module.

Public Module Level, when a variable is declared in the declaration section with the prefix keyword public, the variable is called a public module-level variable these variables can be called from any procedure or module.

Scope of Procedures

You may have a procedure that performs a general function like multiplying the value of two numbers; this procedure can be included in different modules by giving it the relevant scope. You can also specify a type for the procedure it can be one of the following, **Sub, function or property** function procedure is similar to a sub function but whereas the sub procedure executes code the function procedure returns a value.

Procedure scopes

The general syntax that determines the procedure scope is as follows

Private/Public sub <procedure name>()

<Procedure Body>

End Sub

Understanding the syntax

Public indicates that the procedure can be used in different modules**.**

Private indicates the procedure can only be used in the current Module.

Sub / End Sub the start and the end of the procedure.

<Procedure name>denotes the name of the procedure, you must give a unique name and it must not be the same as any of the VBA Keywords.

<Procedure Body>, denotes the code for the procedure. A procedure can have one or more statements that are executed sequentially.

Calling a sub procedure

To create a public procedure that can be called globally, you will need to create it with the public keyword. To call the procedure you use the **Call** keyword followed by the name of the procedure you wish to execute. A procedure that calls another procedure is called a calling procedure, the procedure it refers to is known as the called procedure.

When you call a procedure you can pass information to it through arguments. If it's a procedure does not need an argument you can specify an empty set of parentheses. A called procedure runs its code and then returns control to the next line in the calling procedure.

```
Option Explicit


Public Function CalculateSalesTax() As Currency
  'CalculateSalesTax = Cells(5th row, 4th column)=D5.Value * 0.08
  CalculateSalesTax = Cells(5, 4).Value * 0.08
End Function
```

This is the function that you run in turn it calls the information from above, calculatesalestax takes the value and multiply it by .08 this information is then stored in the variable ST. then the contents of cell D5 are added to the Result of ST

```
Public Sub CallFunc()
  Dim ST As Currency
'Set st as a currency variable
  'cell D5 value = D5 value + ST
  'st is the results of a public function calculate sales tax
    ST = CalculateSalesTax()
  Cells(5, 5).Value = Cells(5, 4).Value + ST
End Sub
```

Function procedures

Similar to sub procedures in the way that are written but a function returns a value. The value that the function produces is stored in a predefined variable. The scope of the function will be either private or public.

General Function Syntax:

Function <procedureName)(variables) as data type

<Procedure Body>

End Function

Example Function:

In excel there is no **function** called **mpg** below we have created that function, inside the brackets we have created **three variables** , line 2 of the code indicates the way the mpg function uses those **variables**. Remember the rules of **BODMAS** still apply.

```
Function mpg (start_miles, End_miles, Fuel) As Integer
        mpg = (End_miles - start_miles) / Fuel
End Function
```

Syntax Components

- **Function and End function** specify the start and finish of the function procedure
- **Function <procedure Name>**, this gives the function its name, the name has to be unique and cannot be a VBA keyword.
- **As <data type>.** Though optional if not declared the variables are set to variant remember we said earlier code is more efficient when specified.
- **<Procedure Body>** the lines of code within the function that make up the function and that are executed in sequential steps.

Remember you can also add a function through the add procedure dialog box, you make your selections and the correct deceleration is added to the module when you click ok

Calling function procedures

Below we have a function called sales, this function is called from a subroutine called main. The key word **call** is used to run the function **sales**

Call a function From Code

```
Option Explicit
Public Gtotal As Currency
```

```
Public Function sales () As Currency
        Gtotal = Cells (8, 2).Value + Cells (8, 3).Value
        Cells (8, 5).Value = gtotal
End Function
```

```
PublicSub main ()
        Call sales
End Sub
```

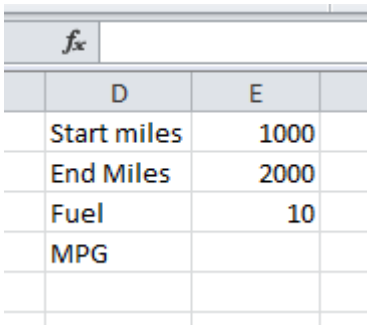
Call a function using insert function dialog box

Earlier on we created the MPG function shown below while some functions you may want to run from code, we can also run UDF functions from our insert function dialog box, they are known as UDF, user defined functions.

```
Function MPG (start_miles, End_miles, Fuel) As Integer
        MPG = (End_miles - start_miles) / Fuel
End Function
```

Using the insert UDF

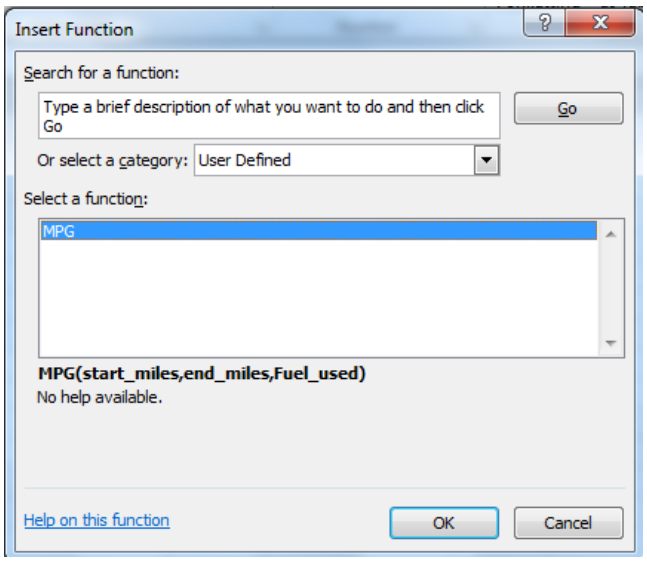
We have laid out an example to make use of the UDF MPG



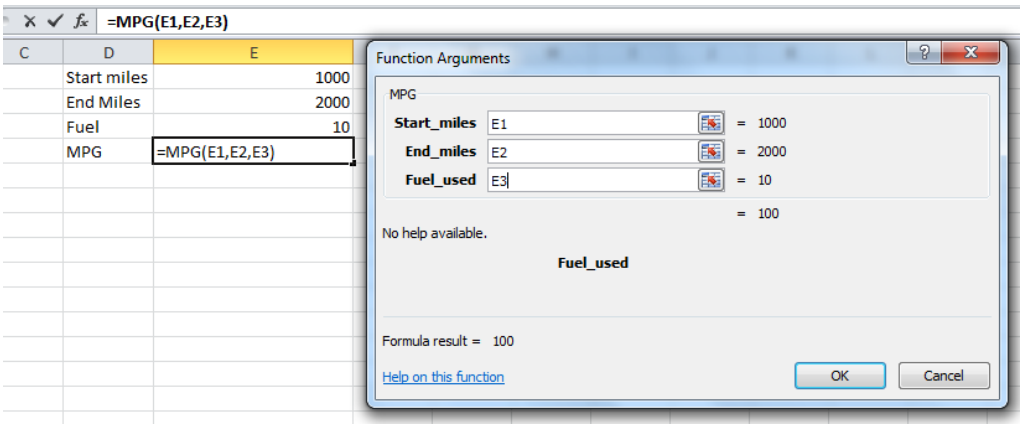
D	E
Start miles	1000
End Miles	2000
Fuel	10
MPG	

Click into cell E4 and click on the icon

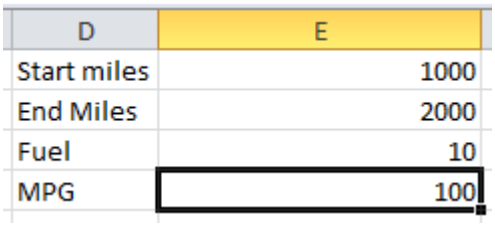
This launches the insert dialog box on the **or select a category** drop down box select **user defined** as shown below, in the select function pane you should see **MPG** select and click **ok.**



Next the input **function arguments** dialog box for **MPG** opens enter the cell references for **start_miles, end_miles and fuel_used** as shown below, click **OK** to complete.



The result is now shown as 100 see below.



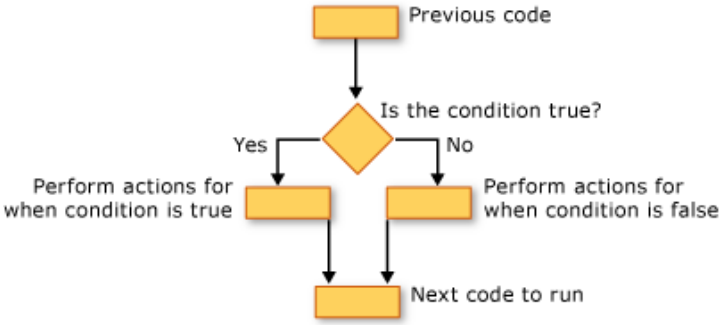
D	E
Start miles	1000
End Miles	2000
Fuel	10
MPG	100

☞ Notes:

Control Structures

Decision Structures

Decision structures allow the program to execute various procedures dependant on an outcome if a condition is met run one procedure if not run an alternative procedure, it is possible to run nested decision structures within VBA using the correct syntax.



If...Then...Else Construction

If...Then...Else constructions allow you to test for one or more conditions and run one or more statements depending on each condition. You can test conditions and take actions in the following ways:

- Run one or more statements if a condition is **True**
- Run one or more statements if a condition is **False**
- Run some statements if a condition is **True** and others if it is **False**
- Test an additional condition if a prior condition is **False**

The control structure that offers all these possibilities is the If...Then...Else Statement in (VBA). You can use a single-line version if you have just one test and one statement to run. If you have a more complex set of conditions and actions, you can use the multiple-line version.

```
Sub test ()
If condition [Then]
[statements]
[ElseIf elseifcondition [Then ]
    [ elseifstatements ] ]
[ Else
    [ elsestatements ] ]
End If
-or-
If condition Then [ statements ] [ Else [ elsestatements ] ]
```

End sub

Syntax property

Condition

Required. Expression. Must evaluate to True or False, or to a data type that is implicitly convertible to Boolean.

Then

Required in the single-line form, optional in the multiple-line form.

Statements

Optional. One or more statements following If...Then that are executed if condition evaluates to True.

Elseifcondition

Required if ElseIf is present. Expression. Must evaluate to True or False, or to a data type that is implicitly convertible to Boolean.

Elseifstatements

Optional. One or more statements following ElseIf...Then that are executed if elseifcondition evaluates to True.

Elsestatements

Optional. One or more statements that are executed if no previous condition or elseifcondition expression evaluates to True.

End If

Terminates the If...Then...Else block.

If then statement

In this procedure we check a cell reference to check to see if the vale is greater than 100 if the cell evaluates greater than 100 the function is deemed as true and places the message “very high mpg” in cell E4 at the end if it breaks out and continues to run the remainder of the procedure, in this case places a value in cell G4

Example 1

```
Sub test1()
        If Cells (4, 5).Value > 100 then
        Cells (4, 6).Value = "Very high MPG"
        End If
        Cells (4, 7).Value = 200
End Sub
```

In this procedure two variables have been set **score** and **grade** score is = to value of cell A1, when run this procedure checks the value of cell A1 if the value is greater than the result is passed. Otherwise the cell stays empty

---

Example 2

```
Sub test2()
        Dim score As Integer, grade As String
        score = Range("A1").Value
        If score >= 60 Then grade = "passed"
        Range("B1").Value = grade
End Sub
```

If then Else statement

In this procedure we check a cell reference to check to see if the vale is equal to 100 if the cell evaluates to 100 the function is deemed as true and this runs a message box that displays Correct Result

```
Sub test ()
        IfCells (4, 5).Value = 100 Then
        MsgBox "correct result"
        Else
        MsgBox "try again"
        End If
End Sub
```

If then elseif Else statement

In this procedure consider it as a nested if, the if function evaluates the first value if it meets that criteria then the first message box is displayed, if the value is not meet it loops through the remainder of the code evaluating the answers until the criteria has been met.

```
Sub test4()
If Cells(4, 5).Value <= 10 Then
    MsgBox "give up"
ElseIf Cells(4, 5).Value <= 40 Then
    MsgBox "poor"
ElseIf Cells(4, 5).Value <= 80 Then
    MsgBox "average"
ElseIf Cells(4, 5).Value <= 90 Then
    MsgBox " above average"
Else
    MsgBox "well done"
  End If
End Sub
```

Select Case statement

The select case statement is another way of running different code based on the value of a variable . in this example you set up a variable called commision and then test it against a series of expression. You can execute various statements for each condition each based on the variable.

```
Sub commision()
totalsales = Cells(7, 5).Value
Select Case totalsales
Case Is>= 100
amount = totalsales * (6 / 100)
Case Is> 200
amount = totalsales * (7 / 100)
Case Is> 500
amount = totalsales * (10 / 100)
Case Is> 1000
amount = totalsales * (15 / 100)
Case Is> 5000
amount = totalsales * (20 / 100)
Case Else
amount = totalsales * (0)
End Select
Cells(8, 5) = amount
End Sub
```

Type the above into the code window, to help you test your results assign a button to the worksheet and then assign your macro to run on click of the button.

Case syntax explained

- **Select case** marks the beginning of the case construct
- **Variable** stores the value you wish to use with the case statement
- **Expression** this is the condition that tests the value stored in the variable
- **Statement** the code that will run if the condition is true
- **Case else** marks the start of the code that will run if the condition is false
- **End Select** ends the case select construct

> *Substitute* **case is>with case 400 to 500** *this checks if a value is between two values*

Loop Structures

You use loop structures such as for… next and for each …next when you want to run a specific block of text repeatedly. Use one of two loop structures depending on the number of iterations that are required.

Fixed iteration This runs a set of statements for a predetermined number of times.Example **For… next loop**

Indefinite Iterationthis runs a set of statement until a defined condition is met.

Example **Do …while loop**

The For… next Loop

in the example below the numbers add up 1 +2+3 to give a result of 6 if you substitute **for count 1 to 3** with **for count 1 to 10** the numbers added 1+2+3+4+5+6+7+8+9+10 resulting in the value 55 returned. You could run **for count 6 to 7** this adds 6+7 **for count 6 to 9** this adds 6+7+8+9.

Example 1

```
Sub count()
        Dim count As Integer
        For count = 1 To 3
'this adds 1+2+3
        Sum = Sum + count
        Next count
        'places answer in a cell
        Cells(2, 2).Value = "the sum of the amount counts " & Sum
'you could use this if the value added to a worksheet is not visible to the user
        MsgBox ("the sum of the amount counts " & Sum)
End Sub
```

Example 2

Next we consider a nested if function and how we can convert it into code.

=IF(A1<500,A1*0,IF(A1<=1000,A1*10%,A1*30%))

```
Sub count2()
   Dim count As Integer
   For count = 1 To 20
   totalsales = Cells(count, 1).Value
   If totalsales >= 500 Then
   commisionamnt = totalsales * 0.1
   ElseIf totalsales > 1000 Then
   commisionamnt = totalsales * 0.3
   Else
   commisionamnt = totalsales * 0
   End If
  Cells(count, 2).Value = commisionamnt
   Next count
End Sub
```

The For Each… next Loop

The For Each..Next statement allows you to repeat a set of actions on the individual elements of a collection or array. For example, if you select a range of cells, then this command allows you to repeat a set of commands on each cell in the range. Another examplewould be to repeat a set of actions on the worksheets in a workbook.

The following is the typical syntax for the For Each..Next statement:

For Each element In *a collection*

statements

Next counter

Example 1

In this example we set a variable as **cell** setting it to **range**

```
Sub ShowValue()
Dim cell As Range
For Each cell In Selection
MsgBox cell.Value
Next
End Sub
```

Example 2

In this exercise if a cell has a number in the range A1:A50 the procedure will Double the cell value, if it contains a number,otherwise clear the cell.

```
Sub ForEachCollectionLoop2()
    For Each cell In Range("A1:G50")
    If IsNumeric(cell) Then
    cell.Value = cell.Value * 2
    Else
    cell.Clear
    End If
    Next
End Sub
```

Do…Loop Statements

Do While Loop

When a process has to be repeated it is best to use a loop structure to make sections ofinstructions repeat rather than have multiple sets of duplicated instructions.

Now we introduce you to Conditional LoopsRepetition while a certain condition is satisfied or until a certain condition is satisfied.

Check for the condition before running the loop:

Do Whilecondition

Statements

Loop

Execute the commands once before checking the condition:

Do

Statements

Loop Whilecondition

Use the keywords Until or While to define the condition, placing them either at the top orat the end of the Do…Loop.

Example 1

X=10 and is our base number DO Until X is greater than 40 Adds 10 to the base

```
Sub DoLoops1()
    x = 10
    Do Until x > 40
    x = x + 10
    MsgBox x
    Loop
End Sub
```

Example 2

```
Sub DoLoops2()
    x = 10
    Do
    x = x + 10
    MsgBox x
    Loop While x < 40
End Sub
```

You can conditionally break out of a Do...Loop using Exit Do.

Save your file before testing the code. It is very easy to get stuck in a conditional loop. You must try to terminate the procedure if you are stuck. Press the ESCAPE key. If this fails, try CTRL and BREAK together. It's bad news after this, CTRL+ALT+DELETE.

Custom Dialog Boxes

User Forms

Display interactive dialogs in the Excel interface by including a User Form in your project. The programming of User Forms can be time-consuming as every action that the User Form performs has to be coded, the OK button does not do anything until you write the code contained in its click event.

You need to be familiar with User Form objects, there is no macro recorder.

Designing the User Form

The general methodology for designing User Forms is as follows:

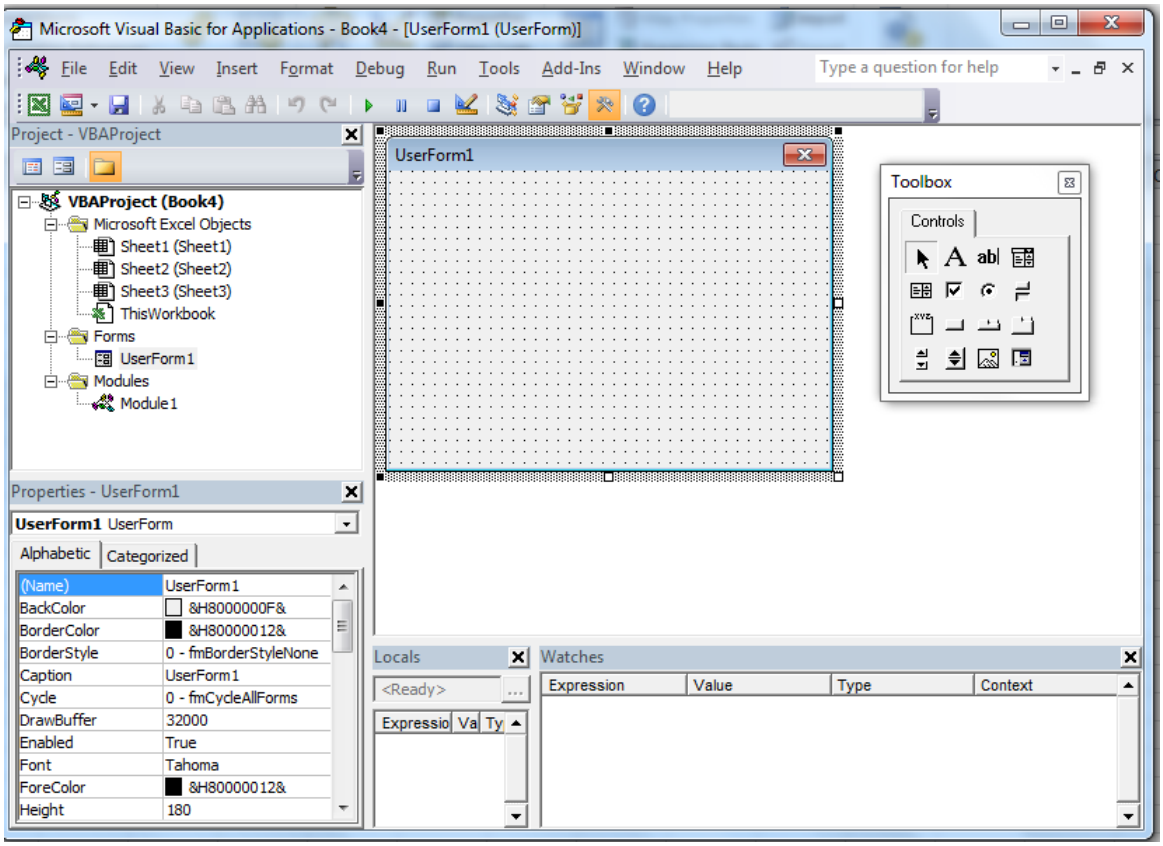
- Insert a User Form into your Project.
- Create the visual image by adding Controls to the Form.
- Name the Controls and set their static properties.
- Write the code in your General Module to show the User Form.
- Fill in the event code shells in the User Form's object module.

Naming objects

It is awkward having to use the default object names when you are completing the event procedures for each control; is the OK button CommandButton1 or is it CommandButton2? Follow the published standard conventions for Control names, add the three-character lower case prefix to your names and you will never have any problems identifying your control objects in code.

Check Box	chk
Combo Box	cbo
Command Button	cmd
Frame	Fra
Label	lbl
List Box	Lst
Option Button	opt
Text Box	txt
Toggle Button	Tog
User Form	frm

User Form Interface



Adding A User form

Create a user form from The VBE Editor Window

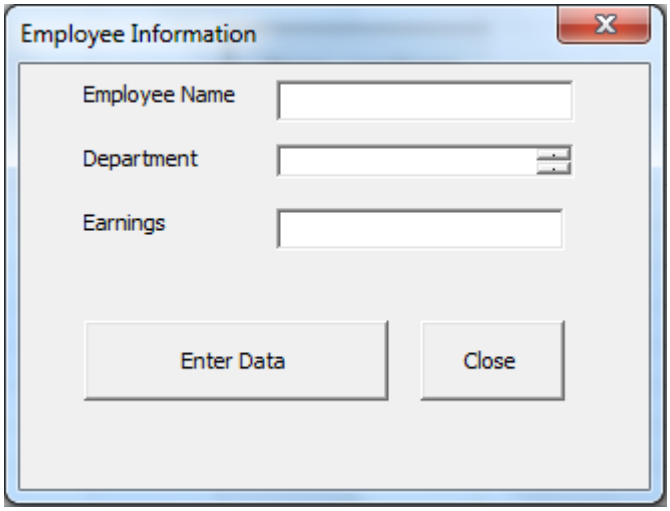
- Open a new Excel Workbook
- Save the workbook as my user form as Excel Macro Enabled
- From the insert menu select user form
- Change the (name) property to frmemployeeinfo
- Change the caption to read Employee Information

Using Controls

Control	Name	Description
	Label	Displays a text label on a form
	Text Box	Accepts data from users
	CommandButton	Performs actions such as saving data
	ListBox	User can select from a list of values
	CheckBox	Yes or no true or false
	Optionbutton	Represents a single option in a group
	ComboBox	Displays a list of values, the user can select or add to the combo box list.

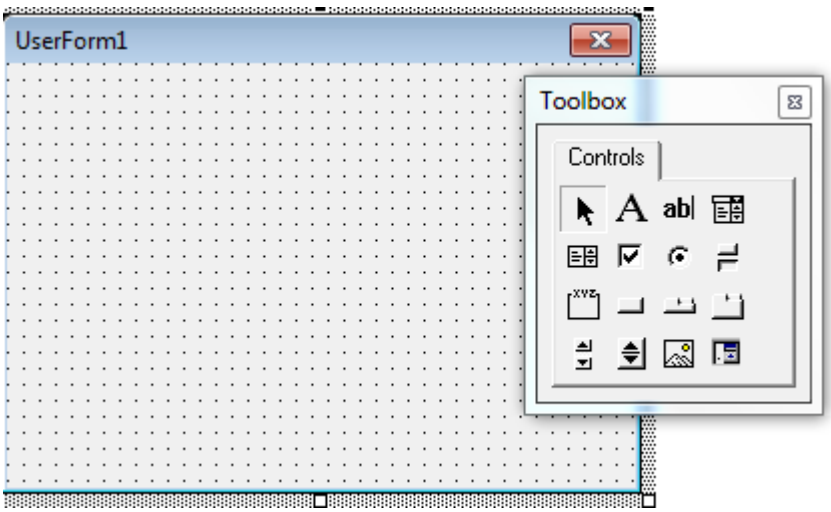
Adding user forms

Follow the steps below to create your first user form

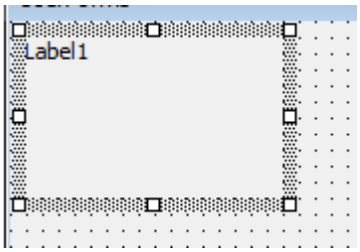


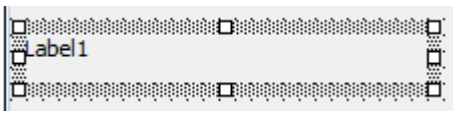
In the example above you will create a user form to enter the employee name, their Department from a list enter their earnings and add two controls one to enter the data and one to close the form.

Switch from the excel view to the **visual basic editor**, from the **insert** menu select **user form**. The screen will now look as shown below. If the toolbox is not displayed from the menu bar select **view toolbox**.

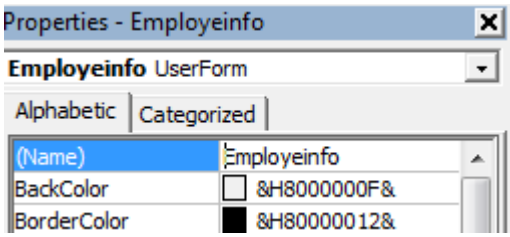


Click the A and draw the label box as shown resize the label as shown below

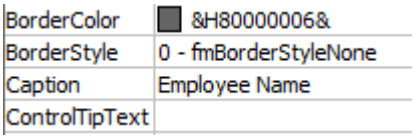




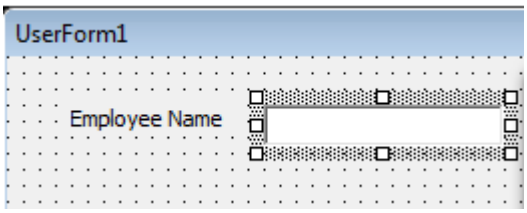
Before we go any further click on the form navigate to the form property's box in name type **Employeeinfo**



Next Change the **caption** property of the label to **Employee name**

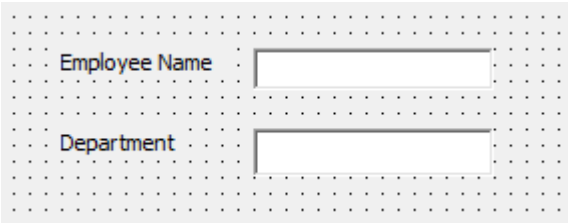


Now add a text box this is the ab| icon



Change the properties of the name to read **txtempname**now add a further label below employee name, change the properties in **caption** to **Department**.

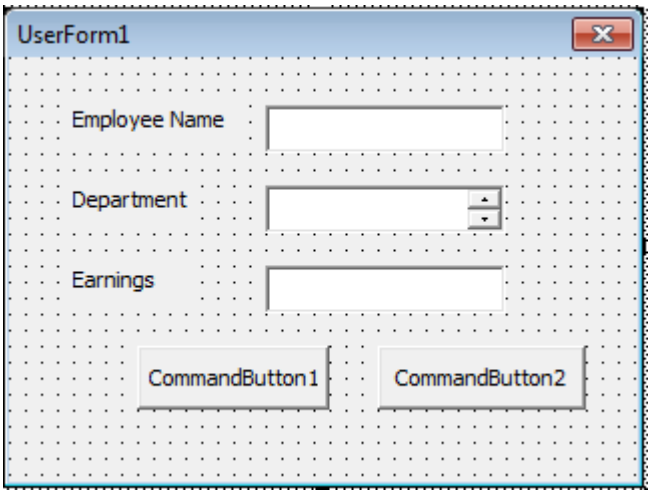
Now we are going to add a List Box click on the icon add the list box below employee name input box. To show as below.



Next change the name property to **lstempdepartment,** locate **RowSource**enter the following details **E4:E10,** the list Box will be populated from this data Range.

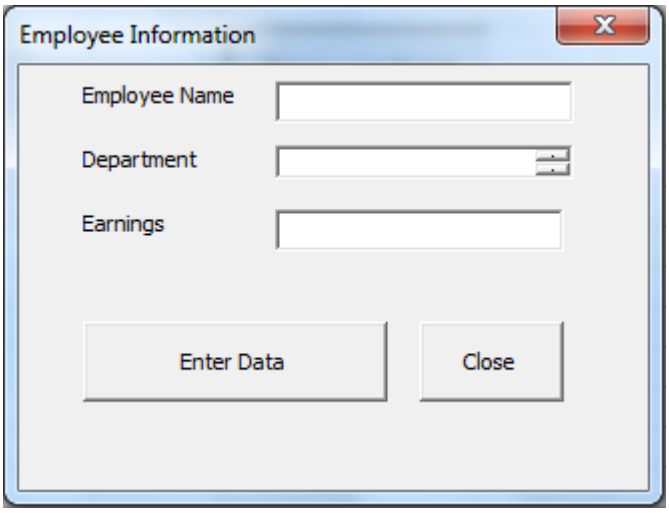
Next add one more label rename the caption to **Earnings,** add a text box change the name property to txtempearnings.

Two more buttons to add these are to be command buttons lay out as shown below.



Now we shall finish it off navigate to the properties of the form in the caption box type the following. **Employee Information** now for the **commandButton1** in the properties change the caption to **Enter Data** change the name property to **CmdAdd**

now for the **commandButton2** in the properties change the caption to **Close** change the name property to **CmdClose** your form should now look like the one below.



We have now created our user form click the save button to update and save your work

In the next section we will look at events and how to transfer data from the user form to the excel data sheet.

Events

VBA is based around event – driven programming. In other words the code doesn't follow a predetermined path, instead it responds to events that a user or condition that occurs. For example the click of a button generates an event, the **on_click** event. Procedures are executed when an event is triggered. User forms and their controls respond to the events that are associated with them. Below is a list of events and their triggers.

User Form Objects	
Event	***Triggers***
Activate	When a user form comes active
Deactivate	When a user form becomes inactive
Click	When a user clicks on any part of a form
DblClick	When a user double clicks any part of a form
Initialize	When a user form is loaded into memory
Terminate	When a user form is unloaded into memory
Control Events	
Event	***Triggers***
Change	A Controls data is changed
Click	A control is clicked
DblClick	A control is double clicked
BeforeUpdate	A controls value is updated through the user form and is about to lose focus. Focus is the ability of a control, user form or window to receive a click or keyboard input
AfterUpdate	A controls value is updated through the user form and loses its focus

Event handling code

Attach an event handler to a control

Double click on the command button **EnterData** the code window opens as below

```
Private Sub CommandButton1_Click()

End Sub
```

Add the following code to read as below this stores the data from the form to the spread sheet in the following Cells A12 ,B12, C12

```
Private Sub CommandButton1_Click()

        Cells(12, 1).Value = Employeinfo.txtempname.Value
        Cells(12, 2).Value = Employeinfo.lstempdepartment.Value
        Cells(12, 3).Value = Employeinfo.txtempearnings.Value

End Sub
```

Double click on the command button **Close** the code window opens as below

This code closes the user form **Employeinfo**

```
Private Sub CommandButton2_Click()

        Employeinfo.Hide

End Sub
```

Now switch to the excel spread sheet and add a control button from the developers tab controls group, insert drop down, button form_control.

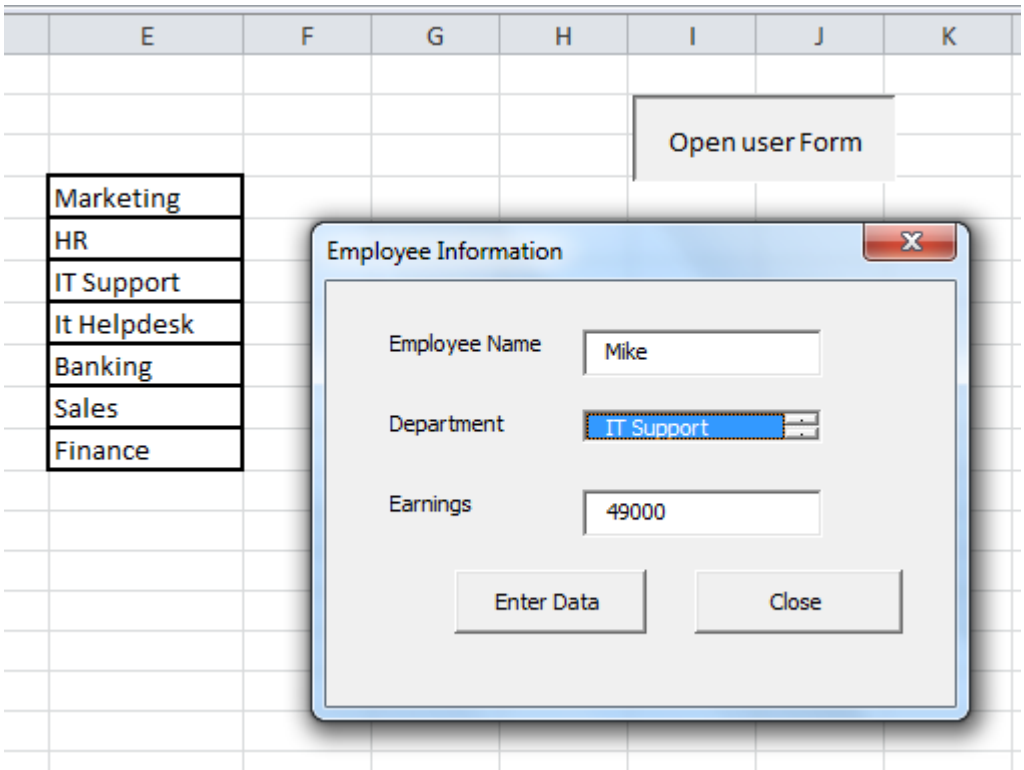
Draw the **button** and the assign macros button dialog box opens, click on **New**
Enter the code as shown below, **Edit** the text on the **Button face** to say **OPEN**

```
Sub Button1_Click()
        Employeinfo.Show
End Sub
```

Enter the following values into cells E4 :E10

Marketing
HR
IT Support
It Helpdesk
Banking
Sales
Finance

Having clicked on the open user form button the employee information form will open Enter the details you wish to add to the spread sheet, click enter data to transfer the contents of the form to the spread sheet, click close to exit the form.



Data Validation

If a user accidently enters the wrong data or data is missing we want to be able to set in place some rules of checking that information, this is known as validation.

Navigate to the VBA Editor window

On the textempearnings enter the following code

```
Private Sub txtempearnings_AfterUpdate()
   If Val(txtempearnings.Value) <= 0 Then
      MsgBox "Null Value Not Valid"
   txtempname.Value = ""
   lstempdepartment.Value = "select one"
   End If
End Sub
```

```
nextrow = Application.WorksheetFunction.CountA(Range("a:a")) + 1

Range("a1").Offset(nextrow, 0).Value = txtName.Value
Range("a1").Offset(nextrow, 1).Value = cmbDept.Value

If optMale.Value = True Then
   Range("a1").Offset(nextrow, 2).Value = "Male"
Else
   Range("a1").Offset(nextrow, 2).Value = "Female"
End If

If chkPension.Value = True Then
   Range("a1").Offset(nextrow, 3).Value = "Yes"
Else
   Range("a1").Offset(nextrow, 3).Value = "No"
End If
```

Form Check List

Before you unleash your UserForm, give it one final check using the following checklist.

- Are the controls aligned with each other?
- Are similar controls the same size?
- Are controls evenly spaced?
- Is the dialog box too overwhelming? If so, you may want to use a series of dialogs (like a Wizard), or use a Multipage control.
- Can every control be accessed with an accelerator key?
- Are any of the accelerator keys duplicated?
- Are the controls grouped logically (by function)?
- Is the tab order set correctly? The user should be able to tab through the dialog box and access the controls sequentially.
- If the UserForm will be stored in an add-in, did you test it thoroughly after creating the add-in? It's important to remember that an add-in will never be the active workbook.
- Will your VBA code take appropriate action if the UserForm is cancelled, or if the user presses Esc?
- Are there any misspellings in the text? Excel's spelling checker doesn't work with UserForms.
- Does the UserForm have an appropriate caption?
- If applicable, will your UserForm fit on the screen in lower screen resolutions?
- If your UserForm will be used in multiple versions of Excel, have you tested the application on all versions?
- If you use help, is the help topic correct? And does it explain all the controls?

Debugging and Error Handling

Errors

When you execute a programme, procedure you may get an incorrect result or the execution of the code causes a permanent stop. This can be because a mistake in the code, called a bug or if a statement in the code tries to call an invalid operation.

Examples of this can be a misspelt variable that will cause an error. When an error occurs VBA will either display an error message or refer you to the help file.

VBA provides you with a set of tools for debugging your code these include, toggle or break points, and the watch window.

Error Types

There are three types of programming error, compile time, run time and logical. The following table describes the type of error.

Error	Description
Compile-error	A compile-time error happens when the program is being compiled. Generally, compile-time errors are syntax errors; and they are caught by the compiler.
Run-time error	Run-time errors occur at at run-time; generally, the program compiles but does not run correctly. An example of a compile-time error might be leaving out a semi-colon in C. An attempt to read past (dereference) a null pointer in C would be an example of a run-time error.
Logical	a logic error is a bug in a program that causes it to operate incorrectly, but not to terminate abnormally (or crash). A logic error produces unintended or undesired output or other behaviour, although it may not immediately be recognized as such. Logic errors occur in both compiled and interpreted languages. Unlike a program with a syntax error, a program with a logic error is a valid program in the language, though it does not behave as intended. The only clue to the existence of logic errors is the production of wrong solutions.

Error Types

Compile Time error

Compile Time error This is because no **Endif** Present

```
Sub calculationbonus()
   Dim bonus As Currency, earnings As Currency
   earnings = InputBox("enter earnings")
   If earnings > 40000 Then
   bonus = earnings * (12 / 100)
   MsgBox "the bonus is " & bonus
End Sub
```

Logical Error

This is a **logical error** the **+** has been used instead of the*****

```
Sub bonus()
   Dim bonusamt As Currency, salesamt As Currency
   salesamt = 10
   bonusamt = salesamt + (6 / 100)
   MsgBox "bonus is " & bonusamt
End Sub
```

Run Time Error

This is **a run time error** the code executes and creates a divide by Zero Error because the **(6 / 0)**

```
Sub bonus()
   Dim bonusamt As Currency, salesamt As Currency
   salesamt = 10
   bonusamt = salesamt * (6 / 0)
   MsgBox "bonus is " & bonusamt
End Sub
```

Debugging

To create an error free application you need to be able to trace an correct errors when they occur, the process is called debugging. VBA provides the following tools to help you debug your code these are located on the debug toolbar.

Debugging tools

The following table describes the tools available to you in VBA to view values of variables and expressions and trace the execution of a program.

Tool	Used To
Break Point	Pause the execution of code at a specified statement. You can insert a breakpoint in the first line of code segment that you suspect to be the cause of the error. You can then monitor the execution of the code
Watch Window	Monitor values of specified variables and expressions while the code is running
Immediate Window	Test your output by assigning different values to variables or expressions
Locals Window	Monitor all the declared variables of the procedure currently running

Break mode and Breakpoint

Whenever a run time is detected in VBA the execution of the program pauses and the program enters what's called break mode. At this point in time the line of code causing the problem is displayed and highlighted in yellow, this helps you trace and debug the error. When you're in break mode you can examine also the variables and properties by pointing to them.

You can set your own breakpoint in the code to pause at a specific place, when the executed code reaches the breakpoint VBA switches into break mode. Breakpoints are temporary markers and are not saved along with the code.

To insert or remove a breakpoint place the insertion point in the code you want to insert the break point and use one of the following methods.

- **Choose debug, toggle breakpoint**
- **Press F9**
- **Click the toggle breakpoint button on the debug toolbar (if not open then, view, toolbars, debug.**
- **Click in the margin indicator bar adjacent to the code click again to remove it**

Setting a Breakpoint

- Open a new work book and layout the data as shown below
- Save as my debugging tools

	A	B	C	D	E	F	G
1	Mouse Training						
2	Performance report Excel						
3	Salesperson	Qtr1	Qtr2	Qtr3	Qtr4	Total sales	Commission
4	Bill MacArthur	£2,500.00	£2,750.00	£3,500.00	£3,700.00	£12,450.00	
5	Jamie Morrison	£3,560.00	£3,000.00	£1,700.00	£2,000.00	£10,260.00	
6	Maureen O'Connor	£4,500.00	£4,000.00	£3,500.00	£3,700.00	£15,700.00	
7	Rebecca Austin	£3,250.00	£2,725.00	£3,000.00	£3,250.00	£12,225.00	
8	Paul Anderson	£2,520.00	£2,000.00	£2,500.00	£4,576.00	£11,596.00	
9	Cynthia Roberts	£1,500.00	£1,700.00	£1,800.00	£2,000.00	£ 7,000.00	
10	Rita Greg	£4,590.00	£4,050.00	£4,500.00	£3,700.00	£16,840.00	
11	Trevor Johnson	£3,660.00	£3,200.00	£3,000.00	£2,250.00	£12,110.00	
12	Kevin Meyers	£1,790.00	£1,800.00	£2,000.00	£2,200.00	£ 7,790.00	
13	Adam Long	£1,700.00	£1,950.00	£2,500.00	£2,750.00	£ 8,900.00	
14	Kendra James	£1,650.00	£2,000.00	£1,500.00	£1,750.00	£ 6,900.00	
15	Michael Lee	£2,050.00	£2,500.00	£2,800.00	£3,200.00	£10,550.00	
16	Sandra Lawrence	£3,425.00	£3,750.00	£4,000.00	£3,120.00	£14,295.00	
17	Mary Smith	£4,540.00	£2,700.00	£3,100.00	£3,200.00	£13,540.00	
18	Annie Philips	£1,200.00	£1,700.00	£1,800.00	£2,000.00	£ 6,700.00	

- Open the VBE
- Insert a module using the following code

```
Option Explicit
Dim TotalSales As Currency, CommissionAmt As Currency

Public Sub Commission()
    TotalSales = Cells(4, 6).Value
If TotalSales > 10000 Then
        CommissionAmt = TotalSales + (10 / 100)
 Else
        CommissionAmt = TotalSales * (7 / 100)
End If
   Cells(4, 7).Value = CommissionAmt
End Sub
```

- Run the procedure you created called **commision**
  this calculates a commision amount if total sales is greater than £10,000 of 10% any sales less than £10,000 then 7%
- Switch back to excel notice cell G4 contains a value of £12450.10 the value we are looking for is £12450.00. this indicates a problem in our logic
- Switch back to the VBE we will find the error using the debug toolbar
- To open the tool bar select, view toolbars, debug

- Place the insertion point at the beginning of the code as shown

```
If TotalSales > 10000 Then
```

- Click the hand icon to set the break point 
- Now click on the run icon 
- Switch back to excel and we can seeno data has been entered into the cell
- Now toggle off the break point and set a break point on the next line of code
- Click on the run icon again repeat for the next line
- The cell is now populated with an incorrect value, as the breakpoint stops the code running we know the line above the breakpoint has the problem lets examine that code.

CommissionAmt = TotalSales + (10 / 100)

- Straight away we can see that the **+**operator has been used instead of *****
- Change the operator from + to *
- Remove the break Point
- Run the procedure again.

You will see that the calculation has now returned the correct value, so using a break point we can identify the problem area, in the next section we will look at the watch window.

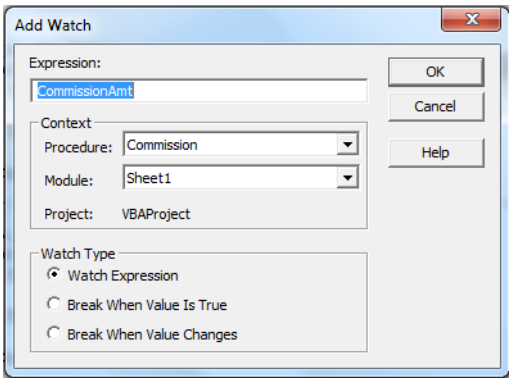
Watch Expressions

Some errors may not be traceable to a single statement in your code, for example in some logical errors it's difficult to isolate the line of code that is causing the error. In cases such as this you need to monitor the behaviour the expressions and variables of the entire procedure. Each expression or variable you monitor is known as a watch, watch expressions can be defined in either break mode or design time.
VBA automatically monitors and displays the expressions in the watch window.
The watch window is displayed automatically when you enter break mode or you choose to open the watch window manually by clicking the watch window button on the debug toolbar. The watch window can be used to change values of variables and expressions this allows you to observe how these changes to affect the code.

How to add a watch expression

- Select debug, click on add watch
- In the expression box to type the variable name
- Under context specify the procedure and module names for the current project, this is important when you have variables of the same name but different scope.
- Click ok to display the watch window

See example of watch window below



- Consider the following, using the file you have already created make sure you are in the VBE window
- Select debug then add watch
- In the expression box type totalsales, the variable to watch, in the context region make sure the procedure list refers to commision and the module list states sheet1 then set watch type to watch expression
- Click ok
- Repeat the procedure this time add CommissionAmt
- Update the code

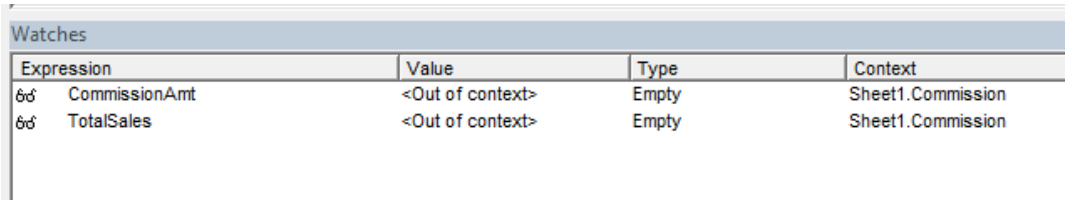
Now these watches are in place we shall now use stepping through code in the next section to see how they work.

Stepping through code

In break mode VBA provides different methods for monitoring the execution of programs, you can execute your program line by line or procedure by procedure or a combination of the two. Running code line by line examining the code line by line is known as steeping through, you can step through the code to determine which statement causes the error. There are three ways to step through the code.

- **Step into**, runs each executable line of code sequentially and steps into the procedure. This allows you to observe the effect of each statement on variables. To step into the code either choose **debug step into** or press **F8** or **click step into button on the debug toolbar**.
- **Step Over**, this runs each procedure as if it was a single statement, you can use this to skip calls to other procedures from the current procedure. To step over code choose **debug, step over**, **click step over button** on the debug toolbar or **press shift + F8.**
- **Step Out**, runs the remaining code in the current procedure as a single statement. If the current procedure is a called procedure the remaining code in the procedure is executed and the debugging stops at the next statement in the calling procedure. To step out of the code, choose **debug, step out**. You can also click **Ctrl+Shift+F8** or **click step out button** on the debug toolbar.

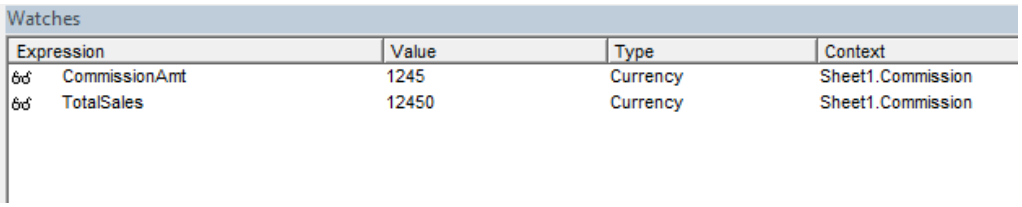
The watch window before running the procedure commission



Watches

Expression	Value	Type	Context
CommissionAmt	<Out of context>	Empty	Sheet1.Commission
TotalSales	<Out of context>	Empty	Sheet1.Commission

The watch window after running the procedure commision



Watches

Expression	Value	Type	Context
CommissionAmt	1245	Currency	Sheet1.Commission
TotalSales	12450	Currency	Sheet1.Commission

Now we understand what stepping through procedure and how it works we shall now put that into practice.

Using the Step Into

- Using the same excel file you have created make sure you switch to the VBE window, if the file is already open, make sure you reset the code.
- Set the break point at the line of code before **if TotalSales >10000 Then**
- Execute the code
- **The yellow highlight** indicates the code has paused and the code is in break mode.
- Press **F8** to advance to the next line of executable line of code, if the value is greater than £10000 the control moves to the statement after the **if** , when the value is lower than £10000 the control moves to the else line then will move to the line directly after.
- As you can see in the window below you can monitor the values you can see at which point the error occurs and rectify.
- Switch back to the excel work book lower the value in the initial cell so that the total sales falls below £10000
- Reset the VBE window set the break point is set **if TotalSales >10000 Then**
- Execute the code
- **The yellow highlight** indicates the code has paused and the code is in break mode.
- **Press F8** to cycle through the code note this time it passes through to the **Else** statement so with the code amended and values changed we have proved the code works.

Delete watch expressions

There are several ways to delete a watch expression

- Select debug, edit watch select the expression to delete and click delete
- Right click on the watch to delete in the watch window, right click on the mouse and from the menu select delete watch.

The immediate Window

The immediate window helps monitor values assigned to variables and expressions, change the values of variables and expressions and test the results of expressions based on these new values. You can use the print method or use a (?) mark followed by the variable name to view the current value of a variable.

To view , immediate window click the immediate window on the debug window.

Immediate window with example of **print** method and **?**

```
Immediate
percentage = 15
?bonus
 2640
percentage = 15
print bonus
 2640
```

Working with the intermediate window

Launch a new work book and type in the information below save as employee information, name the worksheet Employeeinfo.

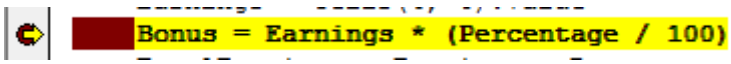
	A	B	C	D	E	F	G
1	**Employee information**						
2							
3	**Name**	**Emp code**	**SSN**	**Region**	**Department**	**Earning (£)**	**Total Earnings**
4	Diana Stone	30	372-12-7281	East	Marketing	£ 44,000.00	
5	Jesse Bennet	23	284-78-9701	South	Sales	£ 250,500.00	
6	Rita Greg	9	612-20-9800	East	Sales	£ 380,050.00	
7	Adam Long	18	640-62-8586	North	Administration	£ 90,000.00	
8	Anna Morris	26	312-13-8162	West	Accounts	£ 150,000.00	
9	Annie Philips	6	553-06-2429	West	Human resources	£ 60,000.00	
10	David Ford	38	631-10-1786	North	Customer support	£ 150,200.00	
11	Davis Lee	37	467-29-9320	East	Accounts	£ 73,500.00	
12	James Overmire	4	283-18-6190	South	Marketing	£ 105,000.00	
13	Jamie Morrison	19	201-90-1901	East	Human resources	£ 62,000.00	
14	Julia Stockton	39	332-21-7283	West	Customer support	£ 96,600.00	
15	Kevin Meyers	17	712-35-6656	West	Accounts	£ 84,000.00	
16	Mary Smith	8	193-33-3314	North	Administration	£ 104,000.00	
17	Maureen O'Connor	20	389-10-7212	West	Accounts	£ 120,000.00	
18	Melinda McGregor	3	336-64-4672	South	Administration	£ 95,000.00	
19	Melissa James	7	423-82-1129	East	Accounts	£ 87,000.00	
20	Michelle Washington	21	283-12-1023	North	Sales	£ 110,000.00	

- With the employee information work book open make sure you are on the employeeinfo worksheet.
- Switch to the VBE window
- Make sure the employeeinfo code window is selected

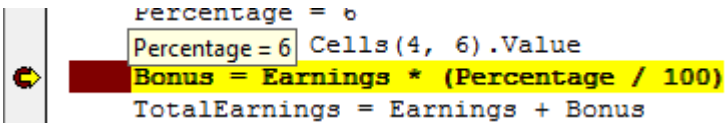
Copy the following code into the code window pane

```
Option Explicit
        Dim Percentage As Single
        Dim Earnings As Currency, TotalEarnings As Currency, Bonus As Currency
    Public Sub NetEarnings()
        Percentage = 6
        Earnings = Cells(4, 6).Value
        Bonus = Earnings * (Percentage / 100)
        TotalEarnings = Earnings + Bonus
        Cells(4, 7) = TotalEarnings
    End Sub
```

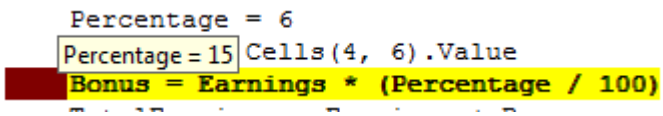
- In the procedure net earnings insert a break point at Bonus = Earnings
- Run the procedure NetEarnings, the code enters break mode



- Point your cursor art the first procedure as shown note the variable percentage as 6



- Click  on the debug toolbar to launch the immediate pane
- In the immediate pane type **Percentage = 15, press the enter key**
- Point your cursor art the first procedure as shown note the variable percentage as 15



- Press **F8** to run the line that calculates the bonus amount and move to the next line
- In immediate window type **Print Bonus** to view the bonus amount in the immediate window
- Press the **Enter Key** to display the bonus amount.
- On the next line of the immediate window type **percentage = 20**
- Press the **Enter Key**

- **Drag the arrow** back to **the break point**
- Press **F8** to run the line that calculates the bonus
- In the immediate window type **?Bonus** press the **Enter Key**
- The bonus is now displayed in the immediate window.
- Remove the Breakpoint
- Close the immediate window
- Update code and run procedure
- Switch back to Excel.
- Save your work

Locals Window

The locals window helps you monitor the values of variables within the current executing procedure or function

The Locals Window

VBA provides one way to access the objects from which Excel is composed. Start to examine these objects by writing a short routine to change the value of a variable. Activate the VB Editor, select Insert/Module and enter the following subroutine in the code window:

```
Sub SampleVariable()
Dim aval As Variant
    aval = 200
    aval = 123.123
    aval = "Cat"
    aval = True
    aval = #12/1/1998#
End Sub
```

Now select the Locals Window command from the View menu and then use the F8 key to step through the subroutine.

You will see the entries in the Locals Window change as each line is executed with the contents and type of each variable displayed in turn.

Error-handling

Error handling refers to the programming practice of anticipating and coding for error conditions that may arise when your program runs. Errors in general come in 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.

We will concern ourselves here only with run time errors. Typical run time errors include attempting to access a non-existent worksheet or workbook, or attempting to divide by zero. The example code in this article will use the division by zero error (Error 11) when we want to deliberately raise an error.

If you have no error handling code and a run time error occurs, VBA will display its standard run time error dialog box. While this may be acceptable, even desirable, in a development environment, it is not acceptable in a production environment.

The goal of well-designed error handling code is to anticipate potential errors, and correct them at run time or to terminate code execution in a controlled, graceful method. Your goal should be to prevent unhandled errors from arising.

The On Error Statement

The heart of error handling in VBA is the On Error statement. This statement instructs VBA what to do when an run time error is encountered. The On Error statement takes three forms.

```
On Error Goto 0
On Error Resume Next
On Error Goto <label>:
```

The on error GoTo 0

The first form, On Error Goto 0, is the default mode in VBA. This indicates that when a run time error occurs VBA should display its standard run time error message box, allowing you 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 second form, On Error Resume Next , is the most commonly used and misused form. It 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. It is the responsibility of your code to test for an error condition and take

appropriate action. You do this by testing the value of Err.Number and if it is not zero execute appropriate code. For example,

```
On Error Resume Next
   N = 1 / 0    ' cause an error
   If Err.Number <> 0 Then
      N = 1
   End If
```

This code attempts to assign the value 1 / 0 to the variable N. This is an illegal operations, so VBA will raise an error 11 -- Division By Zero -- and because we have On Error Resume Next in effect, code continues to the If statement. This statement tests the value of Err.Number and assigns some other number to N.

On Error Goto <label>:

The third form On Error of is On Error Goto <label>:which tells VBA to transfer execution to the line following the specified line label. Whenever an error occurs, code execution immediately goes to the line following the line label. None of the code between the error and the label is executed, including any loop control statements.

```
On Error Goto ErrHandler:
   N = 1 / 0    ' cause an error
   '
   ' more code
   '
   Exit Sub
   ErrHandler:
   ' error handling code
   Resume Next
   End Sub
```

Enabled And Active Error Handlers

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.

Error Handling Blocks And On Error Goto

An error handling block, also called an error handler, is a section of code to which execution is transferred via a On Error Goto <label>: statement. This code should be designed either to fix the problem and resume execution in the main code block or to terminate execution of the procedure. You can't use to the On Error Goto <label>: statement merely skip over lines. For example, the following code will not work properly:

```
        On Error GoTo Err1:
   Debug.Print 1 / 0
   ' more code
Err1:
   On Error GoTo Err2:
   Debug.Print 1 / 0
   ' more code
Err2:
```

When the first error is raised, execution transfers to the line following Err1:. The error hander is still active when the second error occurs, and therefore the second error is not trapped by the On Error statement.

The Resume Statement

The Resume statement instructs VBA to resume execution at a specified point in the code. You can use Resume only in an error handling block; any other use will cause an error. Moreover, Resume is the only way, aside from exiting the procedure, to get out of an error handling block. Do not use the Goto statement to direct code execution out of an error handling block. Doing so will cause strange problems with the error handlers.

The Resume statement takes three syntactic form:

```
Resume
Resume Next
Resume <label>
```

Used alone, **Resume** causes execution to resume at the line of code that caused the error. In this case you must ensure that your error handling block fixed the problem that caused the initial error. Otherwise, your code will enter an endless loop, jumping between the line of code that caused the error and the error handling block.

The following code attempts to activate a worksheet that does not exist. This causes an error (9 - Subscript Out Of Range), and the code jumps to the error handling block which creates the sheet, correcting the problem, and resumes execution at the line of code that caused the error.

```
   On Error GoTo ErrHandler:
   Worksheets("NewSheet").Activate
   Exit Sub

   ErrHandler:
   If Err.Number = 9 Then
      ' sheet does not exist, so create it
      Worksheets.Add.Name = "NewSheet"
      ' go back to the line of code that caused the problem
      Resume
   End If
```

The second form of **Resume is Resume Next** . This causes code execution to resume at the line immediately following the line which caused the error. The following code causes an error (11 - Division By Zero) when attempting to set the value of N. The error handling block assigns 1 to the variable N, and then causes execution to resume at the statement after the statement that caused the error.

```
    On Error GoTo ErrHandler:
N = 1 / 0
Debug.Print N
Exit Sub

ErrHandler:
N = 1
' go back to the line following the error
Resume Next
```

The third form of Resume is **Resume <label>:** . This causes code execution to resume at a line label. This allows you to skip a section of code if an error occurs. For example,

```
 On Error GoTo ErrHandler:
N = 1 / 0
'
' code that is skipped if an error occurs
'
Label1:
'
' more code to execute
'
Exit Sub

ErrHandler:
' go back to the line at Label1:
Resume Label1:
```

All forms of the Resume clear or reset the Err object.

Error Handling With Multiple Procedures

Every procedure need not have an error code. When an error occurs, VBA uses the last On Error statement to direct code execution. If the code causing the error is in a procedure with an On Error statement, error handling is as described in the above section. However, if the procedure in which the error occurs does not have an error handler, VBA looks backwards through the procedure calls which lead to the erroneous code. For example if procedure A calls B and B calls C, and A is the only procedure with an error handler, if an error occurs in procedure C, code execution is immediately transferred to the error handler in procedure A, skipping the remaining code in B.

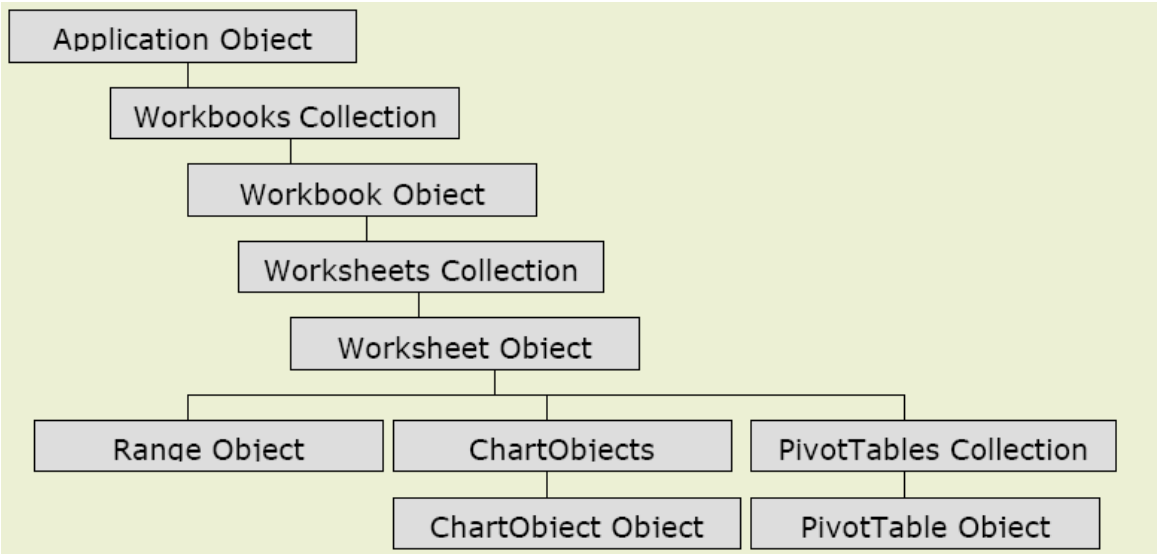
A Note Of Caution

It is tempting to deal with errors by placing an On Error Resume Next statement at the top of the procedure in order to get the code to run without raising an error. This is very bad coding practice. Remember that using On Error Resume Next does not fix errors. It merely ignores them.

The Excel Object Model

The full Excel Object Model has over 200 objects and is too detailed to show on one page. However you tend to only use certain objects on a regular basis and the following diagram shows the relationship between the most commonly used objects.

Search for "Microsoft Excel Objects" in VBA Help to see the full diagram.



Object references: Cells, Sheets and Workbooks

The macro recorder will show you what your object references are but it will not show you the variety of different expressions that can be used to access common Excel objects.

Non-specific Object References	
Selection	The current selection
ActiveCell	The current active cell
ActiveSheet	The current worksheet
ActiveWorkbook	The current workbook
ThisWorkbook	Workbook containing the procedure
Specific Object References, various styles	
Range("A1")	Cell A1
Range("A1:F50")	Range A1:F50
[A1]	Cell A1

[A1:F50]	Range A1:F50
ActiveCell.Range("A2")	The cell below the active cell
Cells(1)	Cell A1
Range(Cells(1,1),Cells(50,6))	Range A1:F50
Range("NamedRange").Cells(1,1)	The first cell in the named range
Range("A:A")	Column A
[A:A]	Column A
Columns(1)	Column A
Range("5:5")	Row 5
[5:5]	Row 5
Rows(5)	Row 5
Sheets("Sheet1")	The Sheet called Sheet1
Worksheets("Sheet1")	The Worksheet called Sheet1
Sheets(2)	The second Sheet in the Workbook
Worksheets(3)	The third Worksheet in theWorkbook
Worksheets("Sheet1").Range("A1")	Cell A1 on Sheet1
[Sheet1].[A1]	Cell A1 on Sheet1
ActiveSheet.Next	The sheet after the active sheet
Workbooks("Basic")	The Workbook file, Basic.xls

Square brackets

The full object reference to the worksheet cell A1 is Range("A1"). If you are typing-in cell references rather than recording, it is easier to use the shortcut notation using square brackets, [A1]. You can use the same style of referencing on other objects as well, such as worksheets but there are a number of rules and restrictions.

It is usually best to restrict the square bracket notation to cell references only, where it is entirely definitive and reliable.

With…End With

The With statement is used so the object reference can be made and then retained so that multiple actions may be carried out without having to repeat the same object reference in each statement.

You can keep the With reference open for as long as you like in the same procedure, just pointing to it using the dot operator. Every With requires an End With. You can have

multiple With pointers. When you are reading code that uses multiple With pointers, the rule is simple; the dot points to the nearest With.

```
With Object
.Property
With .Child Object
.Method
.Method
End With
End With
```

Appendix A How to speed up code

If you have Excel VBA macro code that runs slow, the chances are it's caused by Excel having to recalculate at each line of code. **Deleting rows** is one of many things that can become painfully slow. This can be overcome very easily by switching Excel into manual calculation before your code runs. Just be aware that if your code bugs out, and you have no error trap, the Workbook will be left in manual calculation and **NO** properly designed spread sheet **should ever be used with calculation in manual**.

If you ever get advice to switch Excel into manual calculation to prevent slow saving, closing and data entry...run very fast! This is a band aid approach rather than addressing the underlying cause. If you run any Excel Workbook in manual calculation it's only a matter of time before non-calculated data is read off!

The code below shown how we can switch calculation into manual, run some code, then put it back how it was.

```
Sub GoToManual()
        Dim xlCalc As XlCalculation
         xlCalc = Application.Calculation
         Application.Calculation = xlCalculationManual
        On Error GoTo CalcBack
         'YOUR CODE
        Application.Calculation = xlCalc
Exit Sub
        ‘CalcBack:
        Application.Calculation = xlCalc
End Sub
```

Speed up code and stop screen flickering:

```
Sub NoScreenRePainting()
        Application.ScreenUpdating=False
        'Your code here.
        Application.ScreenUpdating=True
End Sub
```

Preventing calculation while executing code:

```
Sub NoCalculations()
        Application.Calculation = xlCalculationManual
        'Your code here.
        Application.Calculation = xlCalculationAutomatic
End Sub
```

Speeding up code if you have Worksheet or Workbook Events.

Also stops endless loops in Events

```
 Sub StopAllEvents()
        Application.EnableEvents = False
        'Your code here.
        Application.EnableEvents = True
End Sub
```

Inserting a Relative formula into a range of cells:

This is faster than AutoFill or Copy.

```
Sub NoAutoFillOrCopy()
        Range("A1:A200").FormulaR1C1 = "=SUM(RC[1]:RC[5])"
End Sub
```

Avoid the use of Copy and Paste whenever Possible:

```
'Use:
Sub NoCopyAndPaste()
'Instead of:
        Sheet1.Range("A1:A200").Copy
        Sheet2.Range("B1").pasteSpecial
        Application.CutCopyMode=False'Clear Clipboard
'Use:
'By-passes the Clipboard
        Sheet1.Range("A1:A200").Copy Destination:=Sheet2.Range("B1")
```

```
'Or, if only values are needed:
    Sheet2.Range("B1:B200").Value= Sheet1.Range("A1:A200").Value
'Or, if only formulae are needed:
    Sheet2.Range("B1:B200").Formula = Sheet1.Range("A1:A200").Formula
'See also FormulaArray and FormulaR1C1 etc
'Instead of:
    Sheet1.Range("A1:A200").Copy
    Sheet1.Range("A1:A200").PasteSpecial xlPasteValues
    Application.CutCopyMode=False'Clear Clipboard
'Use:
    Sheet1.Range("A1:A200") = Sheet1.Range("A1:A200").Value
End Sub
```

Appendix B Sample Code

Copy Data to another Work Sheet

Option Explicit

```
Sub CopyListOrTable2NewWorksheet()
'Works in Excel 2003 and Excel 2007. Only copies visible data.
   Dim New_Ws As Worksheet,Dim ACell As Range
    Dim CCount As Long,Dim ActiveCellInTable As Boolean
    Dim CopyFormats As Variant, Dim sheetName As String
      If ActiveWorkbook.ProtectStructure = True Or ActiveSheet.ProtectContents = True
          Then
      MsgBox "This macro is not working when the workbook or worksheet is protected"
      Exit Sub
   End If
   'Set a reference to the ActiveCell. You can always use A Cell to
   'point to this cell, no matter where you are in the workbook.
   Set ACell = ActiveCell
   'Test to see if ACell is in a table or list. Note that by using ACell.ListObject, you
   'don't need to know the name of the table to work with it.
   On Error Resume Next
   ActiveCellInTable = (ACell.ListObject.Name <> "")
   On Error GoTo 0
   'If the cell is in a list or table run the code.
   If ActiveCellInTable = True Then
      'Get the new worksheet name.
      With Application
         .ScreenUpdating = False
         .EnableEvents = False
      End With
      'Test if there are more than 8192 separate areas. Excel only supports
      'a maximum of 8,192 non-contiguous cells through VBA macros and manual.
      On Error Resume Next
      With ACell.ListObject.ListColumns(1).Range
         CCount = .SpecialCells(xlCellTypeVisible).Areas(1).Cells.Count
```

```
End With
On Error GoTo 0
If CCount = 0 Then
  MsgBox "There are more than 8192 areas, so it is not possible to " & _
      "copy the visible data to a new worksheet. Tip: Sort your " & _
      "data before you apply the filter and try this macro again.", _
      vbOKOnly, "Copy to new worksheet"
Else
  'Copy the visible cells.
  ACell.ListObject.Range.Copy
  'Add a new Worksheet
  Set New_Ws = Worksheets.Add(after:=Sheets(ActiveSheet.Index))
  'Ask for the Worksheet name
  sheetName = InputBox("What is the name of the new worksheet?", _
                "Name the New Sheet")
  On Error Resume Next
  New_Ws.Name = sheetName
  If Err.Number > 0 Then
    MsgBox "Change the name of sheet : " & New_Ws.Name & _
      " manually after the macro is ready. The sheet name" & _
      " you fill in already exists or you use characters" & _
      " that are not allowed in a sheet name."
    Err.Clear
  End If
  On Error GoTo 0
  'Paste the data in the new worksheet
  With New_Ws.Range("A1")
    .PasteSpecial xlPasteColumnWidths
    .PasteSpecial xlPasteValuesAndNumberFormats
    .Select
    Application.CutCopyMode = False
  End With
  'Call the Create List or Table dialog.
  Application.ScreenUpdating = True
```

```
        Application.CommandBars.FindControl(ID:=7193).Execute
        New_Ws.Range("A1").Select
        ActiveCellInTable = False
        On Error Resume Next
        ActiveCellInTable = (New_Ws.Range("A1").ListObject.Name <> "")
        On Error GoTo 0
        Application.ScreenUpdating = False
        'If you not want to create a table it will run the code below
        If ActiveCellInTable = False Then
          Application.GoTo ACell
          CopyFormats = MsgBox("Do you also want to copy the Formats ?", _
                        vbOKCancel + vbExclamation, "Copy to new worksheet")
          If CopyFormats = vbOK Then
            ACell.ListObject.Range.Copy
            With New_Ws.Range("A1")
              .PasteSpecial xlPasteFormats
              Application.CutCopyMode = False
            End With
          End If
        End If
      End If
      'Select the new worksheet if not active
      Application.GoTo New_Ws.Range("A1")
      With Application
        .ScreenUpdating = True
        .EnableEvents = True
      End With
    Else
      MsgBox "Select a cell in your List or Table before you run the macro", _
          vbOKOnly, "Copy to new worksheet"
    End If
End Sub
```

Copy Data to another Work Book

```
Option Explicit
Sub CopyListOrTableData2NewWorkbook()
'Works in Excel 2003 and Excel 2007. Only copies visible data.
            Dim New_Ws As Worksheet
            Dim ACell As Range
            Dim CCount As Long
            Dim ActiveCellInTable As Boolean
          Dim CopyFormats As Variant
  If ActiveSheet.ProtectContents = True Then
    MsgBox "This macro is not working when the worksheet is protected"
    Exit Sub
  End If
  'Set a reference to the ActiveCell named ACell. You can always use
  'ACell now to point to this cell, no matter where you are in the workbook.
  Set ACell = ActiveCell
  'Test to see if ACell is in a table or list. Note that by using ACell.ListObject, you
  'don't need to know the name of the table to work with it.
  On Error Resume Next
  ActiveCellInTable = (ACell.ListObject.Name <> "")
  On Error GoTo 0
  'If the cell is in a list or table, run the code.
  If ActiveCellInTable = True Then
    With Application
      .ScreenUpdating = False
      .EnableEvents = False
    End With
    On Error Resume Next
    With ACell.ListObject.ListColumns(1).Range
      CCount = .SpecialCells(xlCellTypeVisible).Areas(1).Cells.Count
    End With
    On Error GoTo 0
```

```
'Test if there are more than 8192 separate areas. Excel only supports
'a maximum of 8,192 non-contiguous cells through VBA macros and manual.
If CCount = 0 Then
   MsgBox "There are more than 8192 areas, so it is not possible to " & _
        "copy the visible data to a new workbook. Tip: Sort your " & _
        "data before you apply the filter and try this macro again.", _
        vbOKOnly, "Copy to new workbook"
Else
   'Copy the visible cells to the new workbook
   ACell.ListObject.Range.Copy
   'Add a new workbook with one worksheet
   Set New_Ws = Workbooks.Add(xlWBATWorksheet).Worksheets(1)
   'Paste the data in the worksheet in the new workbook
   On Error Resume Next
   With New_Ws.Range("A1")
      .PasteSpecial xlPasteColumnWidths
      .PasteSpecial xlPasteValuesAndNumberFormats
      .Select
      Application.CutCopyMode = False
   End With
   On Error GoTo 0
   'Call the Create List or Table dialog
   Application.ScreenUpdating = True
   Application.CommandBars.FindControl(ID:=7193).Execute
   New_Ws.Range("A1").Select
   ActiveCellInTable = False
   On Error Resume Next
   ActiveCellInTable = (New_Ws.Range("A1").ListObject.Name <> "")
   On Error GoTo 0
   Application.ScreenUpdating = False

   'If you not want to create a Table it will run the code below
   If ActiveCellInTable = False Then
      Application.GoTo ACell
```

```
            CopyFormats = MsgBox("Do you also want to copy the Formats ?", _
                        vbOKCancel + vbExclamation, "Copy to new workbook")
            If CopyFormats = vbOK Then
              ACell.ListObject.Range.Copy
              With New_Ws.Range("A1")
                .PasteSpecial xlPasteFormats
                Application.CutCopyMode = False
              End With
            End If
          End If

        End If
        'Select the new workbook if not active.
        Application.GoTo New_Ws.Range("A1")
        With Application
          .ScreenUpdating = True
          .EnableEvents = True
        End With
      Else
        MsgBox "Select a cell in your List or Table before you run the macro", _
            vbOKOnly, "Copy to new workbook"
      End If
End Sub
```

Filtering Data

```
Option Explicit
'There are four filter examples in this module
'And a macro example to clear the filter below the filter examples
'1: Filter on InputBox value
'2: Filter on ActiveCell value
'3: Filter on Range value (D1 in this example)
'4: Criteria in the code (Netherlands, with tips below the macro)
```

Filter the first field of the Table/List for the inputbox value

```
Sub FilterListOrTableData()
'Works in Excel 2003 and Excel 2007.
  Dim ACell As Range
  Dim ActiveCellInTable As Boolean
  Dim FilterCriteria As String
  'Use this line if you want to select a cell in the Table with code.
  'Application.GoTo Sheets("Yoursheetname").Range("A24")
  If ActiveSheet.ProtectContents = True Then
    MsgBox "This macro is not working when the worksheet is protected", _
        vbOKOnly, "Filter example"
    Exit Sub
  End If
  'Set a reference to the ActiveCell named ACell. You can always use
  'ACell now to point to this cell, no matter where you are in the workbook.
  Set ACell = ActiveCell
  'Test to see if ACell is in a table or list. Note that by using ACell.ListObject, you
  'don't need to know the name of the table to work with it.
  On Error Resume Next
  ActiveCellInTable = (ACell.ListObject.Name <> "")
  On Error GoTo 0
  'If the cell is in a list or table, run the code.
  If ActiveCellInTable = True Then
```

```
    'Show all data in the table or list.
    On Error Resume Next
    ActiveSheet.ShowAllData
    On Error GoTo 0
    'This example filters on the first column in the List/Table
    '(change the field if needed). In this case the Table starts
    'in A so Field:=1 is column A, field 2 = column B, ......
    'Use "<>" & filtercriteria if you want to exclude the criteria from the filter.
    FilterCriteria = InputBox("What text do you want to filter on?", _
                        "Type in the filter item.")
    ACell.ListObject.Range.AutoFilter _
          Field:=1, _
          Criteria1:="=" & FilterCriteria
  Else
    MsgBox "Select a cell in your List or Table before you run the macro", _
         vbOKOnly, "Filter example"
  End If

End Sub
```

ActiveCell value as criteria

```
Sub FilterListOrTableData2()
'Works in Excel 2003 and Excel 2007.
  Dim ACell As Range
  Dim ActiveCellInTable As Boolean
  Dim FilterCriteria As String
  'Use this line if you want to select a cell in the Table with code.
  'Application.GoTo Sheets("Yoursheetname").Range("A24")
  If ActiveSheet.ProtectContents = True Then
    MsgBox "This macro is not working when the worksheet is protected", _
         vbOKOnly, "Filter example"
    Exit Sub
  End If
```

```
  'Set a reference to the ActiveCell named ACell. You can always use
  'ACell now to point to this cell, no matter where you are in the workbook.
  Set ACell = ActiveCell
  'Test to see if ACell is in a table or list. Note that by using ACell.ListObject, you
  'don't need to know the name of the table to work with it.
  On Error Resume Next
  ActiveCellInTable = (ACell.ListObject.Name <> "")
  On Error GoTo 0
  'If the cell is in a list or table, run the code.
  If ActiveCellInTable = True Then
    'Show all data in the table or list.
    On Error Resume Next
    ActiveSheet.ShowAllData
    On Error GoTo 0
    'This example filter on the ActiveCell value
    ACell.ListObject.Range.AutoFilter _
        Field:=ACell.Column - ACell.ListObject.Range.Cells(1).Column + 1, _
        Criteria1:="=" & ACell.Text
  Else
    MsgBox "Select a cell in your List or Table before you run the macro", _
      vbOKOnly, "Filter example"
  End If
End Sub
```

Filter the first field of the Table/List for the text value of Range("D1")

```
Sub FilterListOrTableData3()
'Works in Excel 2003 and Excel 2007.
  Dim ACell As Range
  Dim ActiveCellInTable As Boolean
  Dim FilterCriteria As String
  'Use this line if you want to select a cell in the Table with code.
  'Application.GoTo Sheets("Yoursheetname").Range("A24")
```

```
  If ActiveSheet.ProtectContents = True Then
    MsgBox "This macro is not working when the worksheet is protected", _
        vbOKOnly, "Filter example"
    Exit Sub
  End If
  'Set a reference to the ActiveCell named ACell. You can always use
  'ACell now to point to this cell, no matter where you are in the workbook.
  Set ACell = ActiveCell
  'Test to see if ACell is in a table or list. Note that by using ACell.ListObject, you
  'don't need to know the name of the table to work with it.
  On Error Resume Next
  ActiveCellInTable = (ACell.ListObject.Name <> "")
  On Error GoTo 0
  'If the cell is in a list or table, run the code.
  If ActiveCellInTable = True Then
    'Show all data in the table or list.
    On Error Resume Next
    ActiveSheet.ShowAllData
    On Error GoTo 0
    'This example filter on the first column in the List/Table.
    '(change the field if needed). In this case the Table starts
    'in A so Field:=1 is column A, field 2 = column B, ......
    'It will use a cell text of Range("C1")for the Criteria.
    ACell.ListObject.Range.AutoFilter _
        Field:=1, _
        Criteria1:="=" & Range("D1").Text
  Else
    MsgBox "Select a cell in your List or Table before you run the macro", _
        vbOKOnly, "Filter example"
  End If
End Sub
```

'Criteria in the code (read the tips below the macro)

This example will filter the first column "Country" for "Netherlands"

```
'ACell.ListObject.Range.AutoFilter Field:=1, Criteria1:="=Netherlands"
Sub FilterListOrTableData4()
'Works in Excel 2003 and Excel 2007.
  Dim ACell As Range
  Dim ActiveCellInTable As Boolean
  Dim FilterCriteria As String
  'Use this line if you want to select a cell in the Table with code.
  'Application.GoTo Sheets("Yoursheetname").Range("A24")
  If ActiveSheet.ProtectContents = True Then
    MsgBox "This macro is not working when the worksheet is protected", _
        vbOKOnly, "Filter example"
    Exit Sub
  End If
  'Set a reference to the ActiveCell named ACell. You can always use
  'ACell now to point to this cell, no matter where you are in the workbook.
  Set ACell = ActiveCell
  'Test to see if ACell is in a table or list. Note that by using ACell.ListObject, you
  'don't need to know the name of the table to work with it.
  On Error Resume Next
  ActiveCellInTable = (ACell.ListObject.Name <> "")
  On Error GoTo 0
  'If the cell is in a list or table, run the code.
  If ActiveCellInTable = True Then
    'Show all data in the table or list.
    On Error Resume Next
    ActiveSheet.ShowAllData
    On Error GoTo 0
    'This example filters on the first column in the List/Table
    '(change the field if needed). In this case the Table starts
    'in A so Field:=1 is column A, 2 = column B, ......
```

```
        'Use "<>Netherlands" if you want to exclude the criteria from the filter
        ACell.ListObject.Range.AutoFilter Field:=1, Criteria1:="=Netherlands"
    Else
        MsgBox "Select a cell in your List or Table before you run the macro", _
            vbOKOnly, "Filter example"
    End If
End Sub
```

In the example I filter on the first column for the Netherlands

```
' ACell.ListObject.Range.AutoFilter Field:=1, Criteria1:="=Netherlands"
'
'But you can also repeat the line for other fields.
'
'This will filter all males from the Netherlands (column A and C in my example)
' ACell.ListObject.Range.AutoFilter Field:=1, Criteria1:="=Netherlands"
' ACell.ListObject.Range.AutoFilter Field:=3, Criteria1:="=M"
''Use this to filter for all males from the Netherlands and the USA (column A and C in my
        example)
'I use two criteria in field 1 (2 is the maximum for AutoFilter in one field)
' ACell.ListObject.Range.AutoFilter Field:=1, Criteria1:="=Netherlands", Operator:=xlOr,
        Criteria2:="=USA"
' ACell.ListObject.Range.AutoFilter Field:=3, Criteria1:="=M"
''Use this to filter for all males born between 23 Feb 1947 and 7 May 1988 from the
        Netherlands and the USA
'(column A, C and D in my example). I use two criteria in field 1 and 4 (2 is the maximum
        for AutoFilter)
' ACell.ListObject.Range.AutoFilter Field:=1, Criteria1:="=Netherlands", Operator:=xlOr,
        Criteria2:="=USA"
' ACell.ListObject.Range.AutoFilter Field:=3, Criteria1:="=F"
' ACell.ListObject.Range.AutoFilter Field:=4, Criteria1:=">=02/23/1947", _
 ' Operator:=xlAnd, Criteria2:="<=05/07/1988"
''Important: Use always the US mm/dd/yyyy format if you filter Dates.
'Note: You only have the use the mm/dd/yyyy format in the code, no problem
'if the format in the worksheet is different.
```

Macro to clear the filter in the Table/List

```
Sub ClearFilterListOrTable()
'Works in Excel 2003 and Excel 2007.
  Dim ACell As Range
  Dim ActiveCellInTable As Boolean
  If ActiveSheet.ProtectContents = True Then
    MsgBox "This macro is not working when the worksheet is protected", _
        vbOKOnly, "Clear filter example"
    Exit Sub
  End If
  'Set a reference to the ActiveCell named ACell. You can always use
  'ACell now to point to this cell, no matter where you are in the workbook.
  Set ACell = ActiveCell
  'Test to see if ACell is in a table or list. Note that by using ACell.ListObject, you
  'don't need to know the name of the table to work with it.
  On Error Resume Next
  ActiveCellInTable = (ACell.ListObject.Name <> "")
  On Error GoTo 0
  'If the cell is in a list or table, run the code.
  If ActiveCellInTable = True Then
    'Show all data in the table or list.
    On Error Resume Next
    ActiveSheet.ShowAllData
    On Error GoTo 0
  Else
    MsgBox "Select a cell in your List or Table before you run the macro", _
        vbOKOnly, "Clear filter example"
  End If
End Sub
```

Print selection or range with one or more areas.

The macro will add a new sheet and copy all the selection areas on it.
Then it will print and delete the sheet.

```
Sub Test()
  Dim Destrange As Range
  Dim Smallrng As Range
  Dim Newsh As Worksheet
  Dim Ash As Worksheet
  Dim Lr As Long
  Application.ScreenUpdating = False
  Set Ash = ActiveSheet
  Set Newsh = Worksheets.Add
  Ash.Select
  Lr = 1
  'You can also use a range with more areas like this
  'For Each smallrng In Ash.Range("A1:C1,D10:G20,A30").Areas
  For Each Smallrng In Selection.Areas
    Smallrng.Copy
    Set Destrange = Newsh.Cells(Lr, 1)
    Destrange.PasteSpecial xlPasteValues
    Destrange.PasteSpecial xlPasteFormats
    Lr = Lr + Smallrng.Rows.Count
  Next Smallrng
  Newsh.Columns.AutoFit
  Newsh.PrintOut
  Application.DisplayAlerts = False
  Newsh.Delete
  Application.DisplayAlerts = True
  Application.ScreenUpdating = True
End Sub
```

Print odd and even pages

This option is not available in Excel but you can use a macro to do it.

```
Sub Print_Odd_Even()
   Dim Totalpages As Long
   Dim StartPage As Long
   Dim Page As Integer
   StartPage = 1  '1 = Odd and 2 = Even
   'Or use the InputBox suggestion from Gord Dibben
   'StartPage = InputBox("Enter 1 for Odd, 2 for Even")
   Totalpages = Application.ExecuteExcel4Macro("GET.DOCUMENT(50)")
   For Page = StartPage To Totalpages Step 2
      ActiveSheet.PrintOut from:=Page, To:=Page, _
                     Copies:=1, Collate:=True
   Next
End Sub
```

Print visible, Hidden or all worksheets

If you want to print a whole workbook you can use this code line
ThisWorkbook.PrintOut Or ActiveWorkbook.PrintOut
But this will not print hidden Worksheets.
You can use this macro to print hidden and visible Worksheets

```
Sub Print_Hidden_And_Visible_Worksheets()
'Dave Peterson
   Dim CurVis As Long
   Dim sh As Worksheet
   For Each sh In ActiveWorkbook.Worksheets
      With sh
         CurVis = .Visible
         .Visible = xlSheetVisible
         .PrintOut
         .Visible = CurVis
      End With
   Next sh
End Sub
```

To print only hidden sheets use

```
With Sh
  CurVis = .Visible
  If CurVis >= 0 Then
    .Visible = xlSheetVisible
    .PrintOut
    .Visible = CurVis
  End If
End With
```

Insert Page Breaks every ? rows

If row 1 is a header row and you want to print it on every page then
change RW + 1 to RW + 2 and use File>Page Setup>Sheet to fill in $1:$1
in the "Rows to repeat at top: " box.

This example will add breaks every 20 rows from row 1 till the last row with data in column A.

```
Sub Insert_PageBreaks()
  Dim Lastrow As Long
  Dim Row_Index As Long
  Dim RW As Long
  'How many rows do you want between each page break
  RW = 20
    With ActiveSheet
    'Remove all PageBreaks
    .ResetAllPageBreaks
    'Search for the last row with data in Column A
    Lastrow = .Cells(Rows.Count, "A").End(xlUp).Row
        For Row_Index = RW + 1 To Lastrow Step RW
      .HPageBreaks.Add Before:=.Cells(Row_Index, 1)
    Next
  End With
End Sub
```