Excel VBA

Sample manual - first two chapters

Wise Owl Training

TABLE OF CONTENTS (1 of 9)

TABLE OF CONTENTS (2 of 9)

TABLE OF CONTENTS (3 of 9)

TABLE OF CONTENTS (4 of 9)

TABLE OF CONTENTS (5 of 9)

TABLE OF CONTENTS (6 of 9)

TABLE OF CONTENTS (7 of 9)

TABLE OF CONTENTS (8 of 9)

TABLE OF CONTENTS (9 of 9)

CHAPTER 1 - THE VISUAL BASIC EDITOR

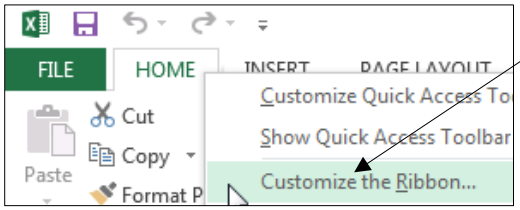
1.1 The Visual Basic Editor

To write any Visual Basic for Applications (VBA) code you'll need to use the Visual Basic Editor (VBE). This chapter explains how to set up the VBE to make writing code as simple as possible.

All of the Microsoft Office applications share the same VBE. This means that if you change any settings in one application those changes will be inherited by the other applications.

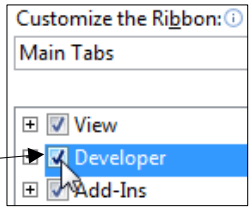
Displaying the Developer Ribbon Tab

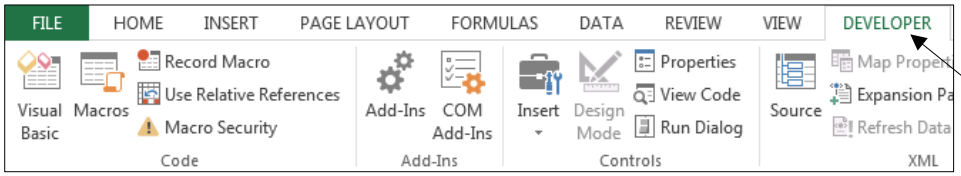
Although you can use the VBE without it, the *Developer* ribbon tab contains some useful tools for working with your VBA code. To display the **Developer** tab:



a) Right-click any existing ribbon tab and choose this option.

b) On the dialog box which appears, check this box and click OK.





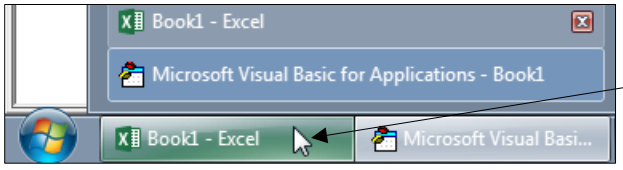
c) Click here to select the **Developer** tab and see the extra tools to which you now have access.

Opening the VB Editor

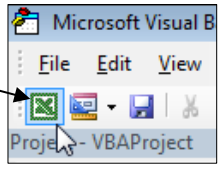
You can open the VBE using one of these options:

Ribbon	*Keyboard*
Developer | Visual Basic	**Alt** + **F11**

When you want to switch back to Microsoft Excel, you can do so by pressing **Alt** + **F11** again. Alternatively, you can use one of the methods shown below:



You can use the Windows task bar to select the Excel workbook that you want to see. You can also just click this button on the VBE toolbar.



1.2 The VBE Screen

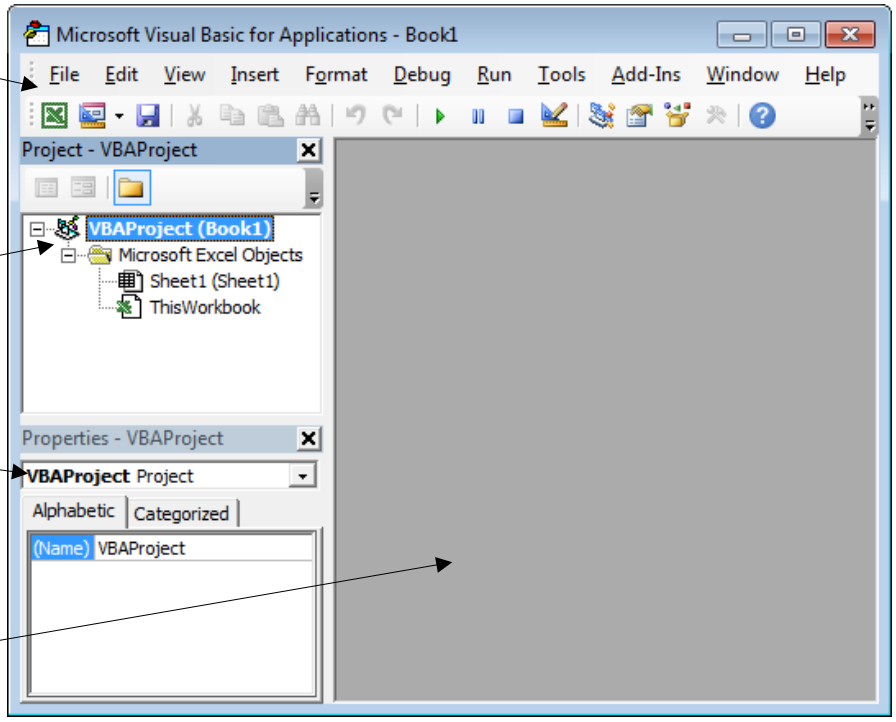
When you first open the VBE you should find that the default layout of the screen resembles the diagram shown below:

Like any old-fashioned Office application, you'll find a menu and toolbar at the top of the screen.

The VBE contains a variety of windows that you can display. One of the default windows is the *Project Explorer*, in which you'll see a list of all the open Excel VBA projects.

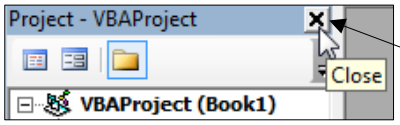
The other window that should be visible by default is called the *Properties* window. This displays the properties of any object that you have selected.

The main part of the screen will be empty. Eventually, this is where you'll be writing all of your code.



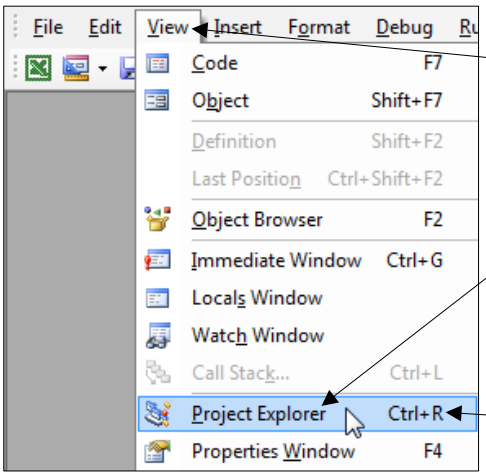
Opening and Closing Windows

You can close any window in the VBE to remove it from the screen.



Simply click the cross in the top right hand corner of any window to close it down.

You can use the **View** menu to display any window that you've closed down, and also to view the other available windows.



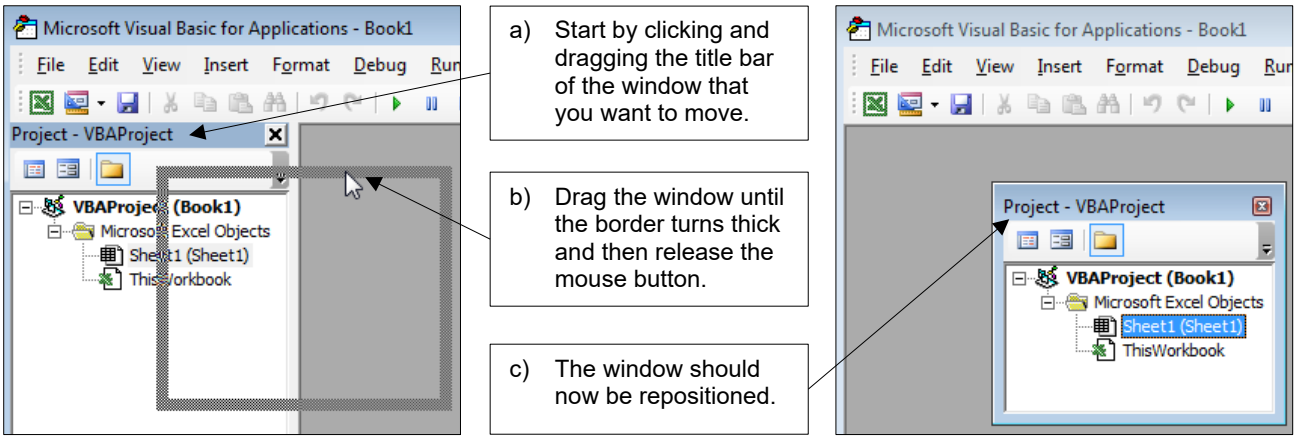
Click the **View** menu to see the list of windows that you can display.

Click the name of a window to make it appear.

Some windows can also be displayed with a keyboard shortcut.

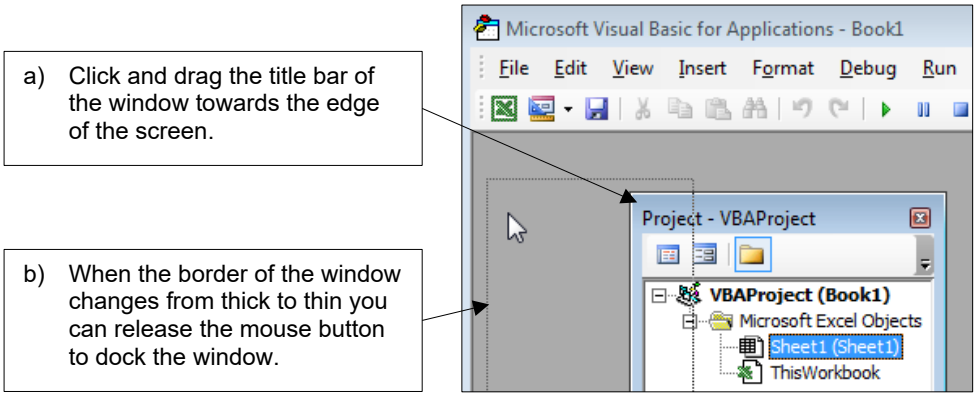
Repositioning Windows

You don't have to accept the default position of the VBE windows. To move a window around you can simply click and drag in the title bar of the window.

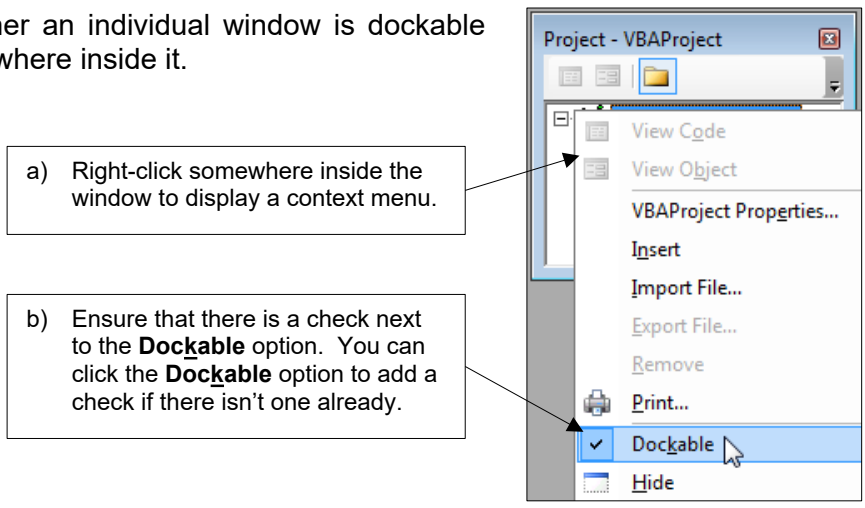


Docking Windows

Returning a window to its original position can be incredibly fiddly. The basic process involves dragging a window towards one of the edges of the screen in order to *dock* it.



You can check whether an individual window is dockable by right-clicking somewhere inside it.

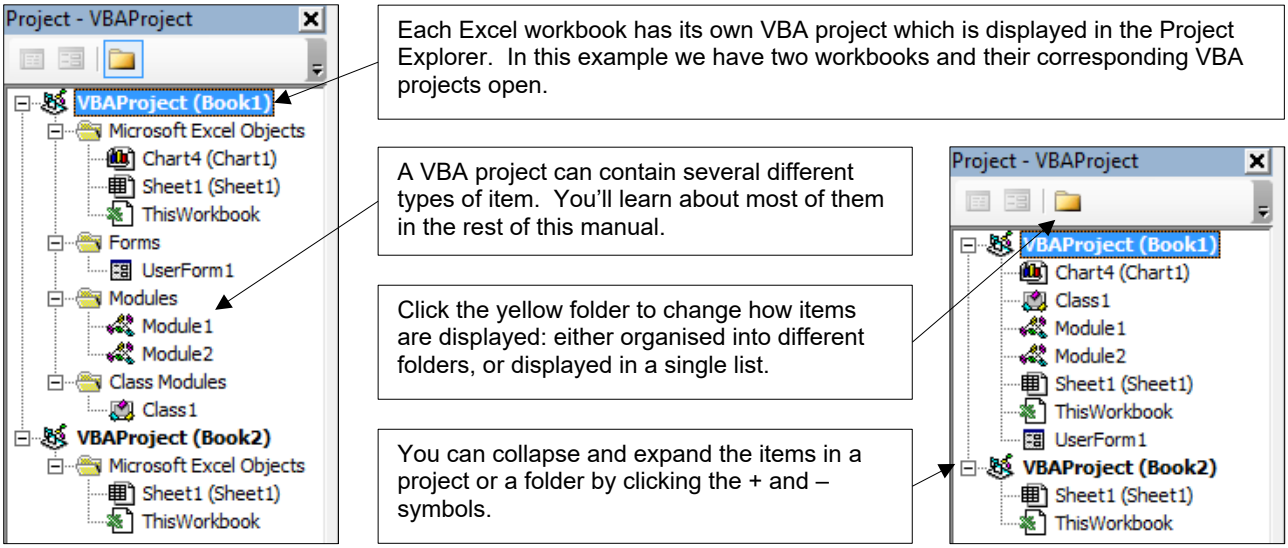


1.3 The Main VBE Windows

You'll find that some of the VBE windows become more useful as you gain experience. There are also some windows which you'll need to learn to use early on in your VBA career.

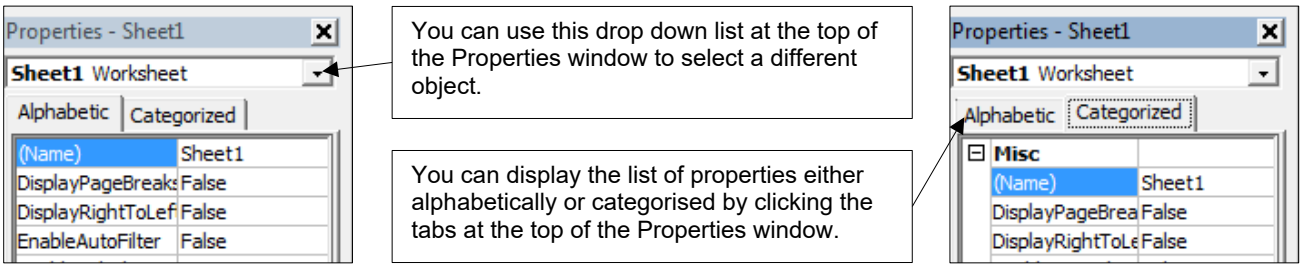
The Project Explorer

The *Project Explorer* window displays a list of all of your open VBA projects, as well as any items contained within these projects.



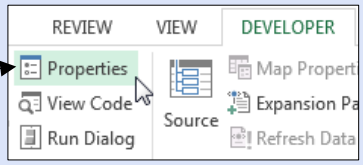
The Properties Window

The *Properties* window shows the attributes of any object that you have selected.



You can also display the Properties window within Excel using a tool on the Developer ribbon tab. Take care though: if you close the window in Excel it will also be closed in the VB Editor.

Click this tool on the Developer tab to show the Properties window in Excel.



1.4 VBE Settings

The VBE has numerous settings that you can alter to suit your preferences when writing code.

The Options Dialog Box

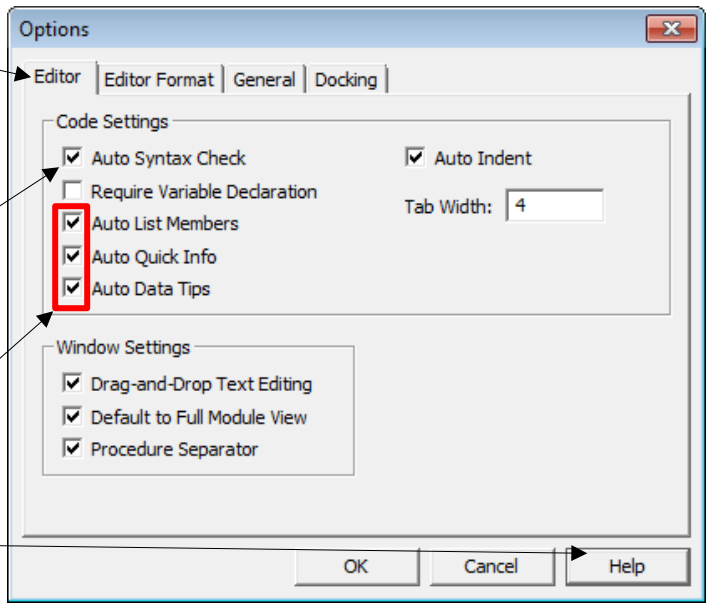
To display the **Options** dialog box, from the menu select: **Tools | Options…**

The default tab you'll see is the **Editor** tab. The options here control the behaviour of the VBE as you're writing code.

The options shown in this diagram represent the default settings you'll see when you first install Excel.

Having these three boxes checked ensures that you'll see as much help as possible as you write your code.

Click this button to open a webpage which describes what each of the options on this tab of the dialog box does.



Changing Font Formatting Options

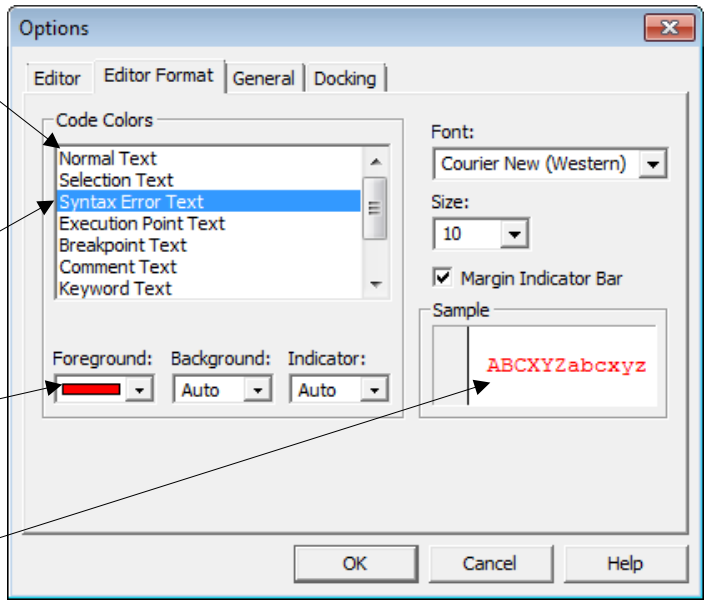
The **Editor Format** tab of the **Options** dialog box has settings that allow you to change the appearance of your code.

The VBE displays different items in your code using different formats. This list shows you the different types of text that you'll see when you're writing code.

Clicking an item in the list reveals the default formatting for that type of code. Here we've selected **Syntax Error Text** which appears in a bright red font in the VBE.

If you don't like the default formatting for any type of code you can change it by selecting different colours, fonts and sizes for the text.

This box shows what the selected text type will look like with your current options. Feel free to click Cancel if it looks horrible!



CHAPTER 2 - WRITING SIMPLE VBA CODE

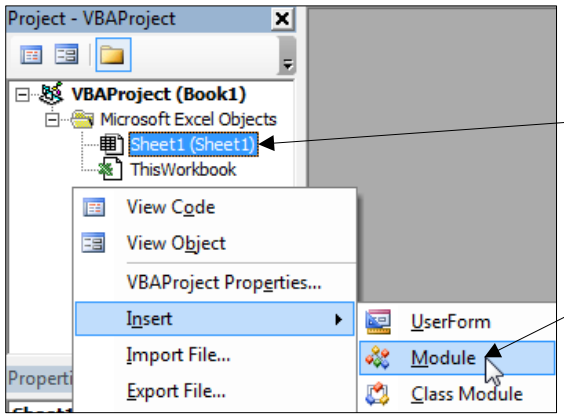
This chapter introduces you to the basics of writing VBA code. You won't create a world-changing application here, but you will learn the fundamental techniques you'll need to start writing one.

2.1 Modules

Before you can start writing code you'll need somewhere to put it. You can write VBA code in a variety of places in a project but the most common location is in a *module*.

Inserting a Module

You can insert a module into a project by selecting **Insert | Module** from the menu. You can also do this using the Project Explorer, as shown in the diagram below:



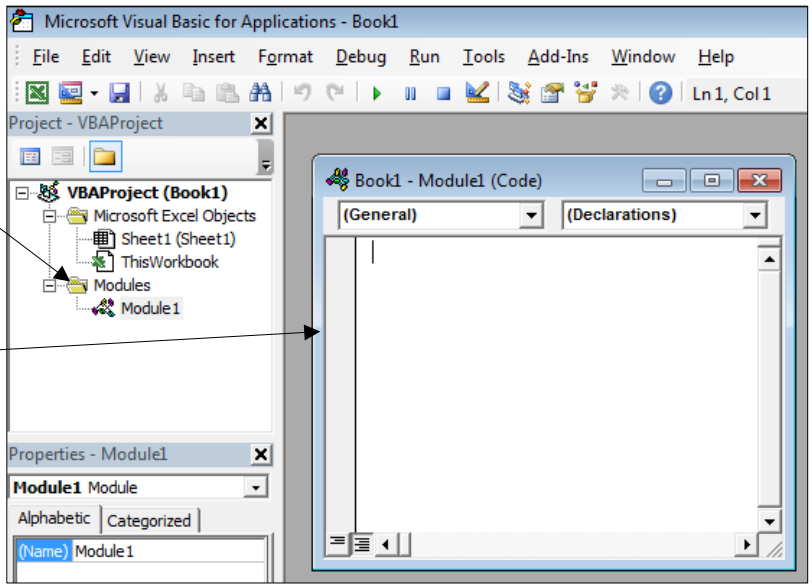
a) Right-click in the Project Explorer somewhere within the project in which you want to insert your new module.

b) From the context menu which appears, select **Insert | Module**

Your new module will appear in the **Modules** folder of your project and will automatically open in the main window of the VBE.

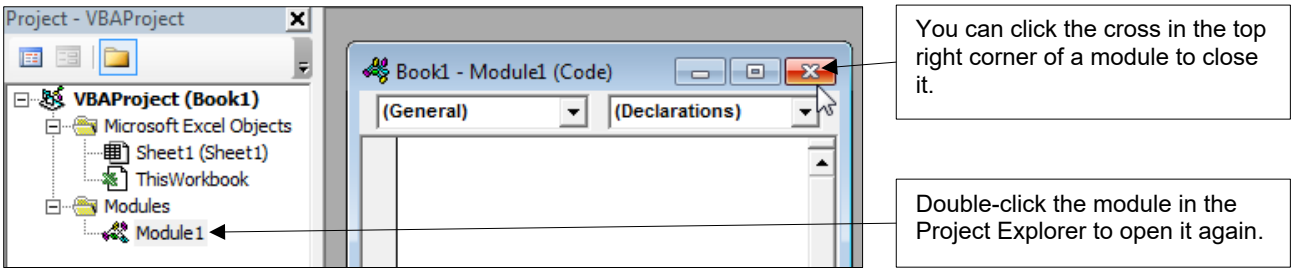
You'll see your new module listed in the **Modules** folder of the project.

The module will also be displayed in the main area of the VBE.



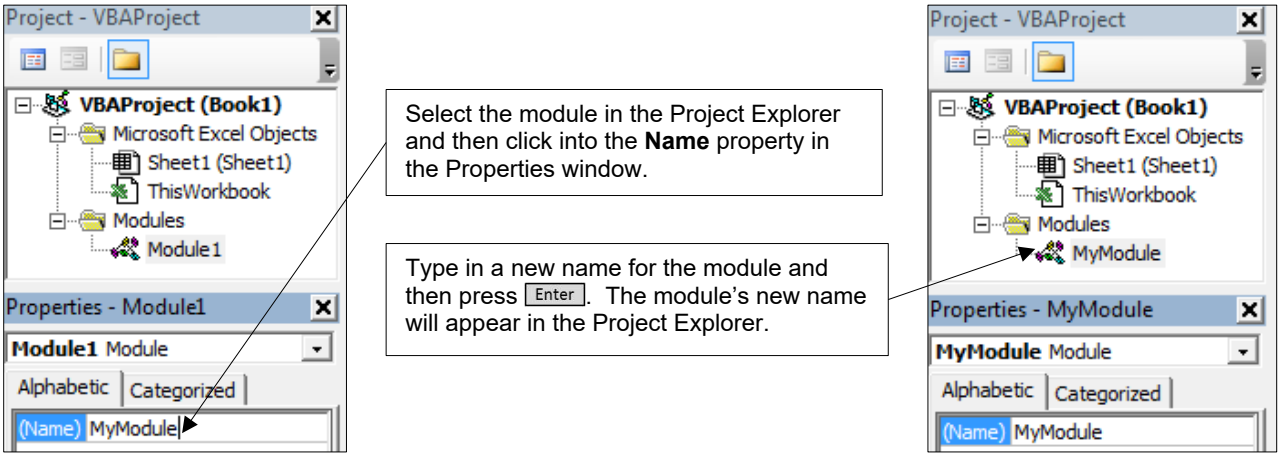
Opening and Closing Modules

When you insert a module it automatically opens. You can close and reopen modules easily, as shown below:



Renaming Modules

To rename a module you change its **Name** property in the Properties window.



Naming Rules in VBA

The rules for module names apply to the names of everything to which you can assign a name in VBA. These rules are summarised in the table below:

Rules for naming things in VBA
The first character must be a letter.
The name cannot contain a space, or any of the following characters **. ! @ $ & #**
The maximum length of a name is 255 characters.
You can't have duplicates of a name in the same scope. So, for example, you can't have two modules in the same project with the same name, but you can have modules in separate projects with the same name.
It's best to avoid using the names of existing VBA things. For example, don't call a module something like **Workbook** or **Worksheet**.

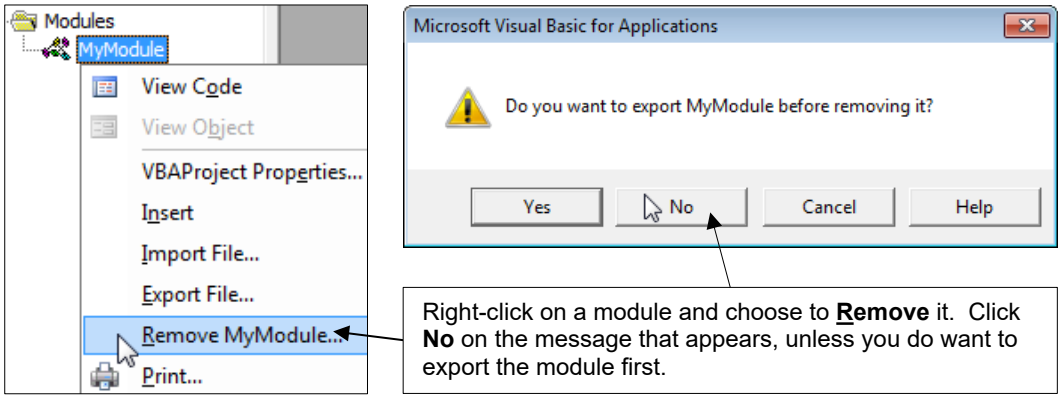
Naming Conventions

As well as the rules that you must follow for naming things in VBA, there are some conventions that you could choose to adopt in order to make your names consistent.

Convention	Description	Example
Capital Letters	Use a capital letter at the start of each word in the name. This is called *Pascal Case* or, sometimes, *Camel Case*.	MyFirstModule
Underscores	Use an underscore instead of a space to separate words.	My_First_Module

Removing Modules

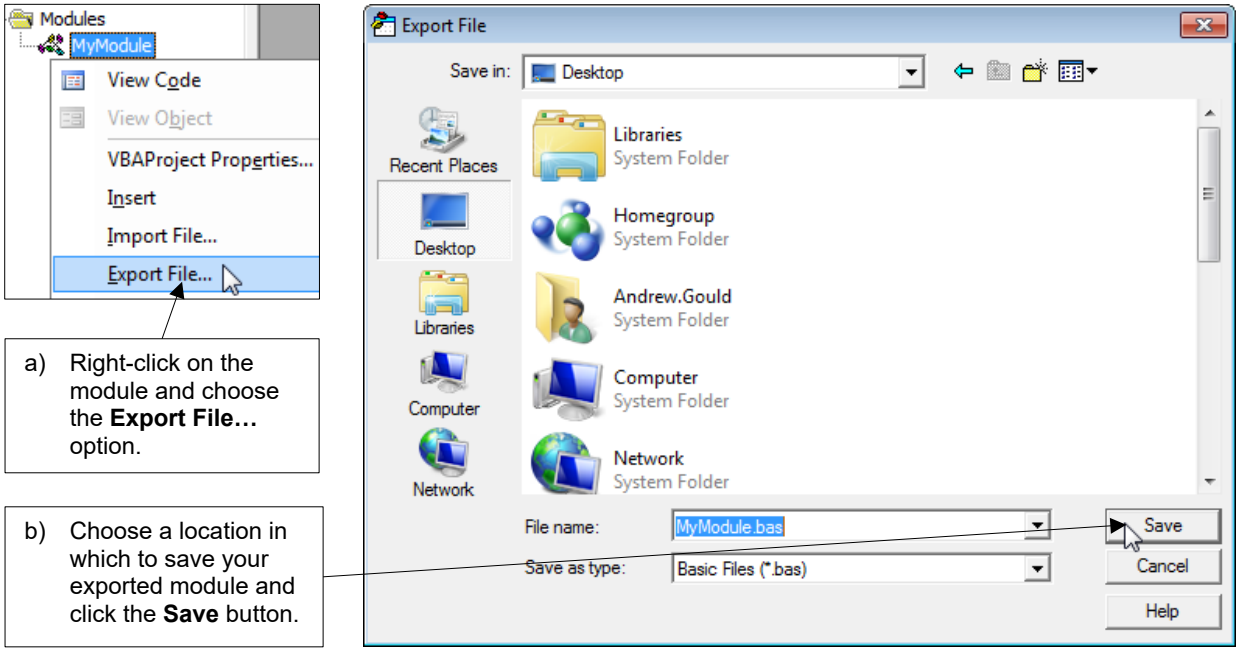
You can delete a module from a project by choosing to *remove* it.



Right-click on a module and choose to **Remove** it. Click **No** on the message that appears, unless you do want to export the module first.

Exporting Modules

You can *export* a module to a file which can be moved around independently of a VBA project.

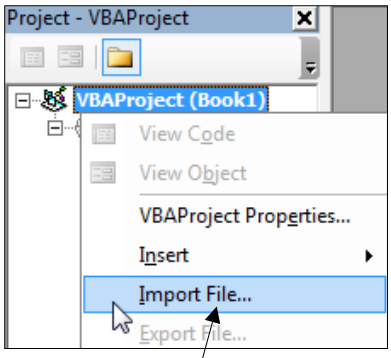


a) Right-click on the module and choose the **Export File…** option.

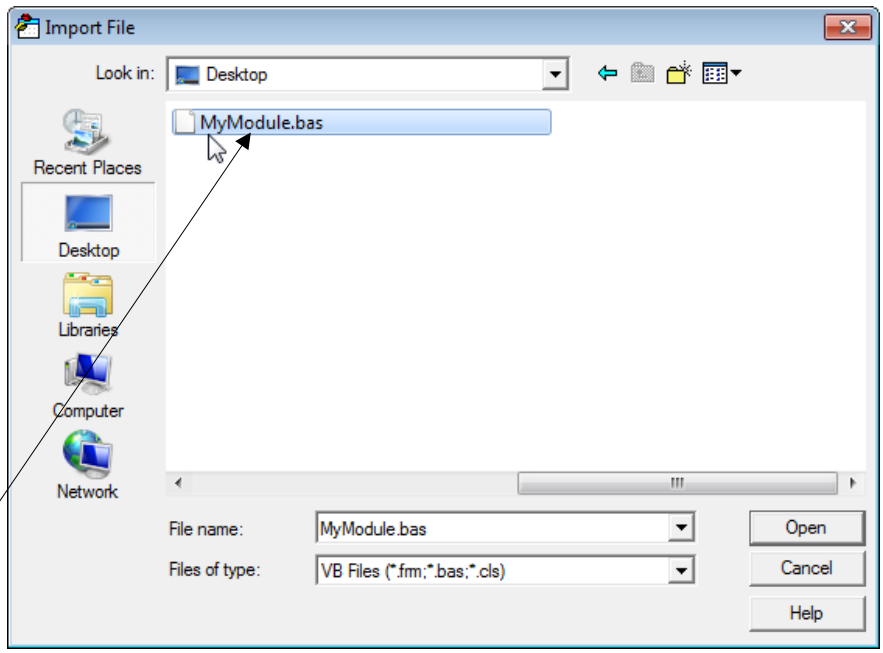
b) Choose a location in which to save your exported module and click the **Save** button.

Importing Modules

You can't run or edit the code in an exported module. First, you must import it into a VBA project.



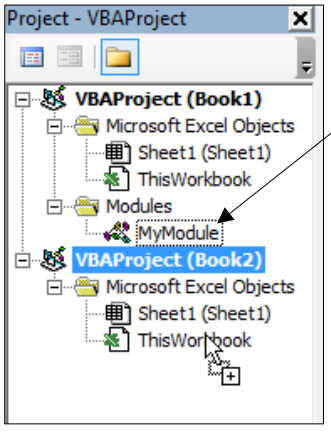
a) Right click the mouse in the Project Explorer and click **Import File…**



b) Double-click on the module you want to import to add it to the VBA project.

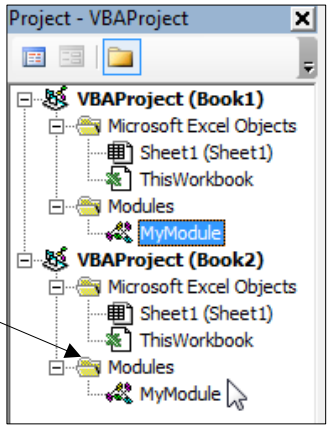
Copying Modules to Other Projects

If you have more than one project open at the same time it's easy to copy modules between them.



a) Click and drag the module from its original project into any other project listed in the Project Explorer.

b) Release the mouse button to drop the module into its new project. A copy of the module will be created for you automatically.



If the destination project already contains a module with the same name, the one that you're copying will be renamed automatically to avoid a conflict.

2.2 Writing Procedures

Procedure is a generic term used to describe a variety of different programs that you can write in VBA. This section explains how to start writing the simplest type of procedure; a *subroutine*.

Types of VBA Procedure

There are three types of procedure you can write in VBA: *subroutines*; *functions*; and *properties*. The table below summarises what each one is, and shows a fairly useless example of each.

Procedure	Description	Example

Subroutine	This is the simplest type of procedure you can write. A subroutine contains a list of instructions for the program to carry out in a specific order. Subroutines are commonly referred to as *subs* or *macros*.	`Sub MyUselessSubroutine()`
		`MsgBox "This is useless"`
		`End Sub`

Function	A function is similar to a subroutine in that it contains a list of instructions to be executed in a particular order. The main thing which distinguishes this type of procedure is that it can also return some kind of value or reference.	`Function IsThisUseless() As Boolean`
		`IsThisUseless = True`
		`End Function`

Property	Properties are written primarily inside class modules. In basic terms, a property is an attribute of an object. There are three different forms of the property statement: *Let*, *Get* and *Set*.	`Property Get Uselessness() As String`
		`Uselessness = "Very useless"`
		`End Property`

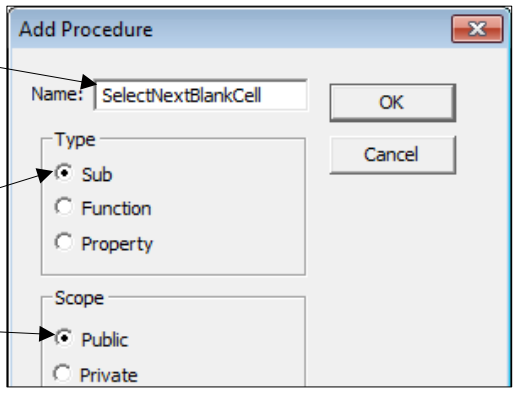
Inserting Procedures

The easiest way to begin a procedure is simply to start typing in your module. If you'd like a little help you can also insert a procedure from the menu by choosing **Insert | Procedure…**

a) Give the procedure a name. Compound names like this one are ideal, as they describe what the procedure does and are unlikely to be confused with existing Excel VBA keywords.

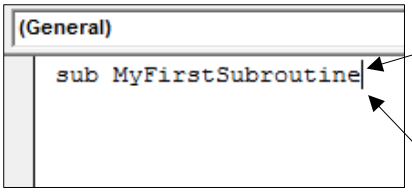
b) Choose which type of procedure you want to create. In this chapter we're sticking with subroutines.

c) Choose the *scope* of your procedure. *Public* procedures can be called from any module in the project, while *private* ones can only be called in the module in which they are written.



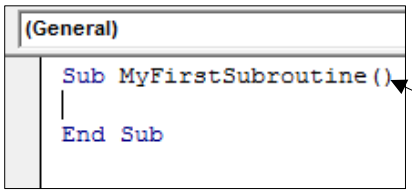
Starting a Subroutine

Although inserting a procedure can help to remind you of the syntax, most of the time you'll find it easier just to type directly into your module. The diagram below shows you how to get started.



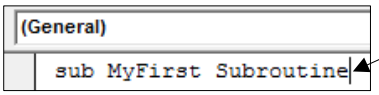
a) Start by typing the word *sub* followed by a space and the name that you want to give your procedure.

b) Once you've typed in a name for the sub, simply press Enter on the keyboard.



c) Several things should then happen:
- The letter *s* in the word *sub* will be capitalised.
- The word *Sub* turns blue.
- Parentheses appear at the end of the procedure's name.
- The words *End Sub* appear.

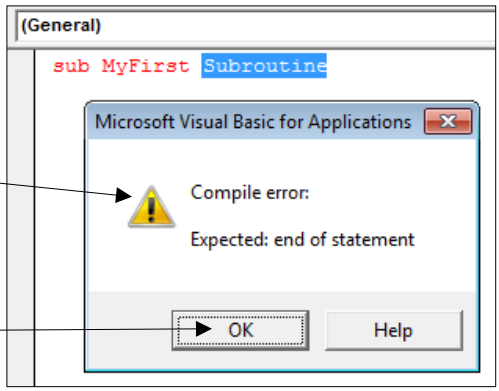
If, on the other hand, you've done something wrong, the VB Editor should make it immediately apparent by displaying an error message.



Here we're trying to create a sub with a space in its name.

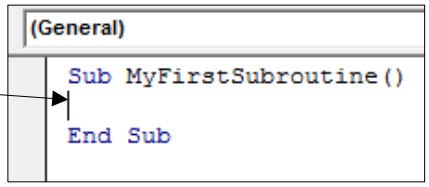
The VB Editor makes it obvious that you've done something wrong by highlighting the text in red and, by default, displaying an error message.

The error message is often difficult to interpret, but in this case we know exactly what we've done wrong. Click OK on the message so that you can remove the space and fix the problem.



When you've successfully created the procedure you can start writing out the instructions to make it do something!

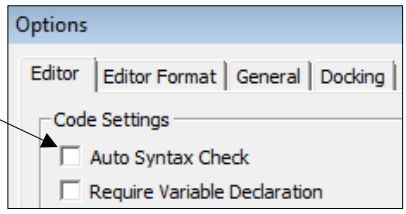
To make your procedure actually do something you just need to write the instructions in between the **Sub** and **End Sub** lines.



Switching off Syntax Error Messages

When you make a mistake it can be annoying to have to click OK on the (often useless) error message before you can fix the problem. Fortunately, you can turn these messages off.

From the menu choose **Tools | Options…** and on the **Editor** tab of the dialog box, uncheck this box.



Now when you make a syntax error the line of code will be highlighted in red, but you'll no longer have to clear the error message before you go about fixing the problem.

Setting the Scope of a Procedure

The *scope* of a procedure determines its availability to other modules in your project. Unless you specify otherwise, all procedures that you create are public.

You can write the word **Public** at the start of a procedure to explicitly show that it is public, but as this is the default you can happily omit this word.

```
Public Sub MyFirstSubroutine()

End Sub
```

Public procedures are available to all of the modules in a project. If you want to restrict the scope of a procedure to a single module, use the word **Private** instead.

```
Private Sub MyFirstSubroutine()

End Sub
```

2.3 Writing Neat Code

Taking the time to write neat code can be a difficult habit to get into, but you'll thank yourself for doing it later on! Neatly-written code is quicker and easier to read and debug.

```
Sub GoodCode()

    'declare some variables
    Dim ProductStatus As String
    Dim SingleCell As Range

    'go to the correct worksheet
    Worksheets("Sheet1").Select

    'loop over the products in column A
    For Each SingleCell In Range("A1:A100")
        'get the status of product from column C
        ProductStatus = SingleCell.Offset(0, 2).Value

        'test if the product is obsolete
        If ProductStatus = "Obsolete" Then
            'if so, format the product ID cell
            With SingleCell
                .Interior.Color = rgbPink
                .Font.Color = rgbRed
                .Font.Italic = True
                .Font.Bold = True
            End With
        End If
    Next SingleCell

End Sub
```

These two procedures perform exactly the same task at exactly the same speed. The one on the left takes slightly longer to write due to the added comments and careful indenting of lines, but if you had to solve an issue with the code the one below is much more difficult to work with.

```
Sub BadCode()
Dim ProductStatus As String
Dim SingleCell As Range
Worksheets("Sheet1").Select
For Each SingleCell In Range("A1:A100")
ProductStatus = SingleCell.Offset(0, 2).Value
If ProductStatus = "Obsolete" Then
With SingleCell
.Interior.Color = rgbPink
.Font.Color = rgbRed
.Font.Italic = True
.Font.Bold = True
End With
End If
Next SingleCell
End Sub
```

Commenting Your Code

Comments are a useful way to help other people (or future you) interpret the code you've written. You can begin a comment by typing an apostrophe followed by your comment text.

```
Sub MyFirstSubroutine()

    'This is a useless comment

    Worksheets.Add 'Comments can appear after code

End Sub
```

You can write comments on separate lines like this one.

You can also write comments at the end of a line of code.

Old-school (or just old) programmers may be interested to learn that you can also add comments using the **Rem** statement.

Rem is short for remark and behaves just like the apostrophe except that you can't use it to add comments at the end of a line of code.

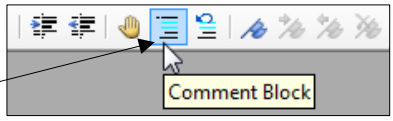
```
Sub NotMyFirstSubroutine()

    Rem A really old-fashioned comment
```

Commenting Out Multiple Lines of Code

Sometimes you'll want to temporarily remove some lines of code from your procedures. Rather than deleting them entirely you can simply turn them into comments.

a) Start by selecting at least part of each line that you want to comment out.

b) Click this button which you can find on the **Edit** toolbar. If you can't see this toolbar, from the menu select **View | Toolbars | Edit**



c) All of the selected lines will be turned into comments.

d) To uncomment the lines, select them and click this tool on the toolbar.

```
Sub FormatSelectedCells()

    With Selection.Font
        .Name = "Arial"
        .Size = 12
        .Bold = True
        .Italic = False
        .Color = rgbBlack
    End With

End Sub
```

```
Sub FormatSelectedCells()

    With Selection.Font
        .Name = "Arial"
'       .Size = 12
'       .Bold = True
'       .Italic = False
'       .Color = rgbBlack
    End With

End Sub
```

Using Blank Lines and Indenting

As you saw in the screenshot at the start of this section, you can write your procedures in one continuous wall of text. It's much better to spend time laying out your code neatly however.

After typing the name of a new procedure press Enter twice to create a blank line between the procedure name and the start of the code.

Press the Tab key to indent the code within the procedure by one level.

```
Sub IndentingLines()

    |
End Sub
```

Within a procedure you should use blank lines at your discretion to make the code as easy to read as possible. The conventions for indenting code depend on which statements you're writing.

Some VBA statements have a corresponding end statement, for example **Sub** always has a matching **End Sub**.

All of the code written between the beginning and end of a statement such as **Sub** and **End Sub** should be indented one level.

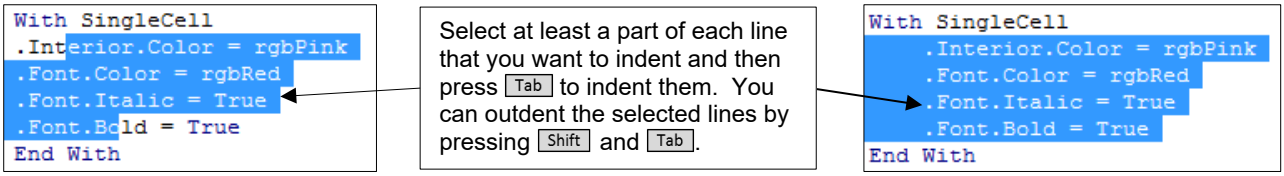
You should continue to indent code each time you begin another statement with a beginning and end, such as **If** and **End If**.

The line at the end of a statement should be written at the same indent level as the start of the statement. You can press Backspace or Shift + Tab to outdent code.

```
Sub IndentingLines()

    Dim ProductStatus As String
    Dim SingleCell As Range

    Worksheets("Sheet1").Select

    For Each SingleCell In Range("A1:A100")
        ProductStatus = SingleCell.Offset(0, 2).Value

        If ProductStatus = "Obsolete" Then
            With SingleCell
                .Interior.Color = rgbPink
                .Font.Color = rgbRed
                .Font.Italic = True
                .Font.Bold = True
            End With
        End If
    Next SingleCell

End Sub
```

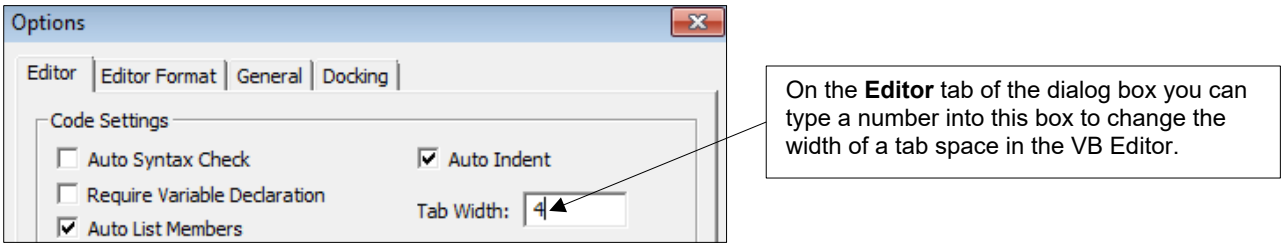
Indenting Multiple Lines

You can indent multiple lines of code at the same time



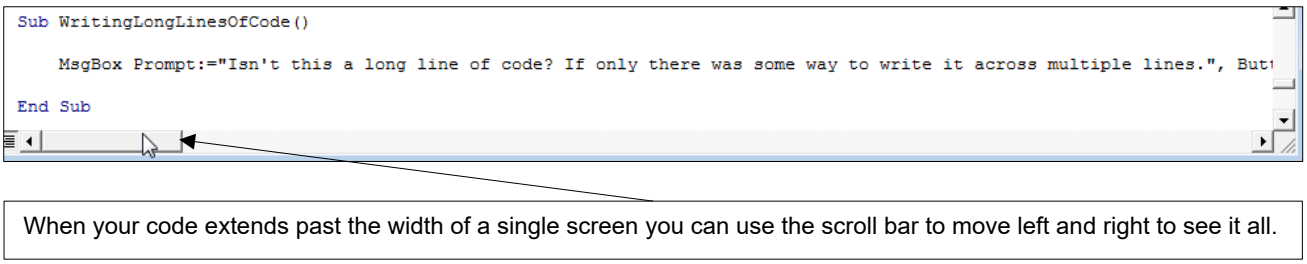
Changing Indenting Settings

The default width of a tab space in the VB Editor is equivalent to four spaces. You can change this setting by choosing **Tools | Options…** from the menu.

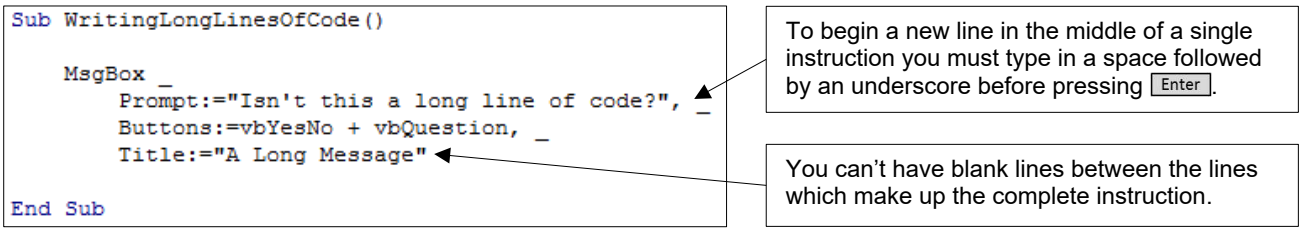


The Continuation Character

As you begin writing longer, more complex instructions you'll often find that your screen isn't wide enough to display the code without scrolling left and right.



You can break one line of code into multiple separate lines using the continuation character. Each time you want to split an instruction onto a new line, type in a space followed by an underscore.



2.4 Writing Simple VBA Instructions

This section is designed as a brief introduction to how the VBA language works to help you get started. We'll discuss these basic ideas in much more detail in a later chapter.

Objects

VBA is based around the concept of *objects*. Some of the main objects you'll encounter are ones that you'll be familiar with from working with Excel, such as workbooks, worksheets and cells.

Generally speaking, whenever you want to perform an action in VBA, you begin the instruction by referring to an object.

After referencing the object you enter a full stop and then use another VBA keyword to do something to the object. The code shown in this example activates a workbook, then selects a worksheet, and finally changes the value of a range object.

```
Sub ReferencingObjects()

    Workbooks("Book1.xlsm").Activate

    Worksheets("Sheet1").Select

    Range("A1").Value = "Something"

End Sub
```

*Basic VBA sentence structure follows a **Thing.Action** pattern, where the **Thing** is the object that you want to manipulate and the **Action** is what you want to do to it. The **Thing** is always separated from the **Action** using a full stop.*

Methods and Properties

In order to manipulate an object you can either apply one of its *methods*, or modify one of its *properties*.

```
Sub ReferencingObjects()

    Workbooks("Book1.xlsm").Activate

    Worksheets("Sheet1").Select

    Range("A1").Value = "Something"

End Sub
```

The name of a method is usually a verb and represents some kind of action that will be performed on an object. Different objects have different methods that can be applied to them. **Activate** and **Select** are both examples of methods.

Properties are attributes of objects whose value you can often change. To assign a value to a property you make it equal to something. Here we're assigning the word **Something** to the **Value** property of a **Range** object.

It may seem complicated at first but the rules of grammar in VBA are relatively simple and, more importantly, consistent. Give it some time and you'll soon be speaking VBA like a pro!

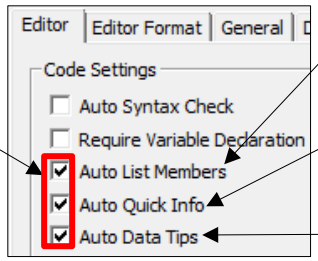
2.5 Tools to Help with Writing Code

There are several features built in to the VBE that are designed to provide you with help as you write your code.

Choosing Which Tools are Enabled

To choose which tools are enabled, from the menu select **Tools | Options…**

On the **Editor** tab of the dialog box, checking these three boxes ensures that you'll receive the maximum amount of help as you write your code. If any of these features annoys you, simply uncheck the box to disable them.



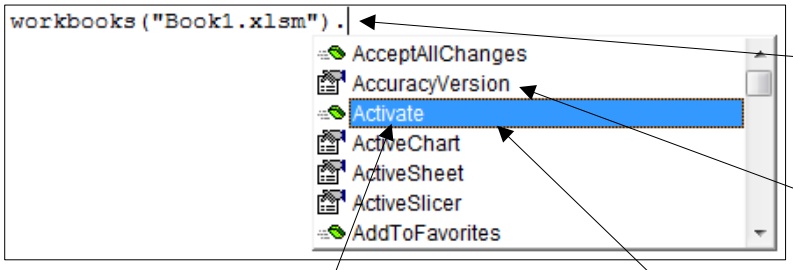
Checking *Auto List Members* ensures that the *IntelliSense* list will appear automatically.

Auto Quick Info determines whether *tooltips* will appear to help you.

Auto Data Tips means you see tooltips when hovering the mouse over certain bits of code.

Using IntelliSense to Write Code Faster

IntelliSense is a useful feature which attempts to present you with a list of valid options as you write your code. This happens automatically if you've checked the **Auto List Members** option.



After referencing an object you can type a full stop to make the IntelliSense list appear.

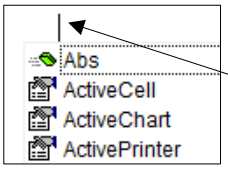
The IntelliSense list displays all of the methods and properties for the class of object that you've referenced.

You can highlight an item in the list either by scrolling through it using the cursor keys or by starting to type the name of the method or property that you want to use.

To type in the highlighted word you can either press Tab to remain on the same line, or Enter to move to the next line.

Beware that not all objects display an IntelliSense list when you type in a full stop immediately after referencing them. A notable example of this is the worksheet object.

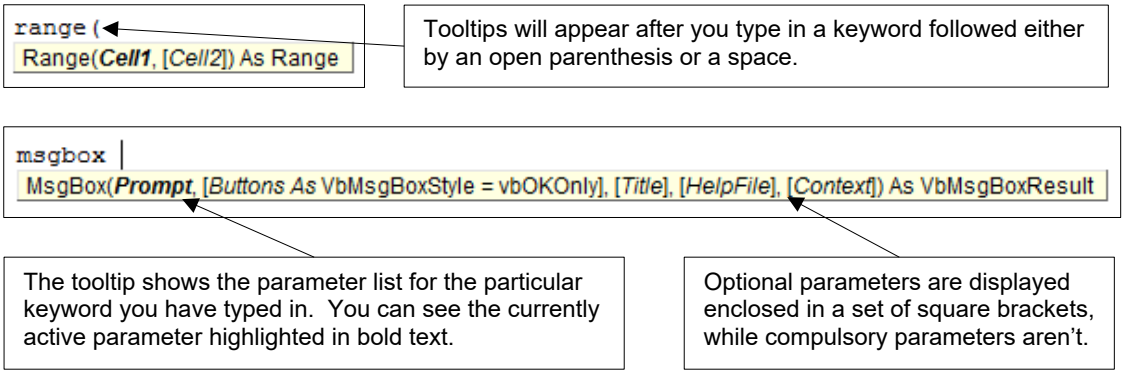
You can also attempt to force the IntelliSense list to appear using a keyboard shortcut. Pressing **Ctrl** + **J** or **Ctrl** + **Spacebar** will achieve this.



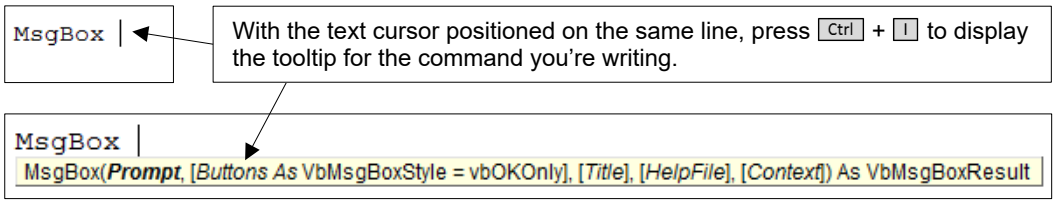
You can even make the IntelliSense list appear at the start of a blank line using one of the two keyboard shortcuts listed above.

Using Tooltips

Tooltips provide you with information on the parameters of VBA keywords. These tooltips will appear automatically as long as you have the **Auto Quick Info** option checked.

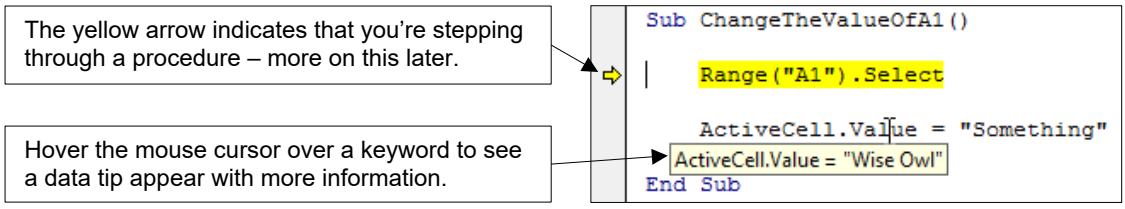


If a tooltip disappears and you want to redisplay it, press Ctrl + I (that's a capital **i** rather than a lower case **L**) on the keyboard.

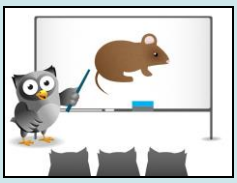


Viewing Data Tips

Data tips only appear while you're stepping through your code – a technique that you'll learn about in a later chapter. To see a data tip simply hover the mouse cursor over a keyword.



WHAT WE DO

		 ONLINE TRAINING	 MANCHESTER OR LONDON	 AT YOUR OFFICE	 BESPOKE CONSULTANCY
OFFICE 365	Microsoft Excel	✓	✓	✓	✓
	VBA macros	✓	✓	✓	✓
	Office Scripts	✓		✓	
	Microsoft Access				✓
POWER PLATFORM	Power BI and DAX	✓	✓	✓	✓
	Power Apps	✓		✓	
	Power Automate	✓	✓	✓	✓
SQL SERVER	Reporting Services	✓	✓	✓	✓
	Report Builder	✓		✓	✓
	Integration Services	✓	✓	✓	✓
	Analysis Services	✓		✓	
CODING LANGUAGES	SQL	✓	✓	✓	✓
	Visual C#	✓	✓	✓	✓
	Python	✓	✓	✓	✓

Wise Owl Training


Wise Owl
Training