



# Excel VBA Fast Track

Sample manual - first two chapters



## TABLE OF CONTENTS (1 of 12)

1	THE VISUAL BASIC EDITOR	Page
1.1	The Visual Basic Editor	14
	<i>Displaying the Developer Ribbon Tab</i>	14
	<i>Opening the VB Editor</i>	14
1.2	The VBE Screen	15
	<i>Opening and Closing Windows</i>	15
	<i>Repositioning Windows</i>	16
	<i>Docking Windows</i>	16
1.3	The Main VBE Windows	17
	<i>The Project Explorer</i>	17
	<i>The Properties Window</i>	17
1.4	VBE Settings	18
	<i>The Options Dialog Box</i>	18
	<i>Changing Font Formatting Options</i>	18

2	WRITING SIMPLE VBA CODE	Page
2.1	Modules	19
	<i>Inserting a Module</i>	19
	<i>Opening and Closing Modules</i>	20
	<i>Renaming Modules</i>	20
	<i>Naming Rules in VBA</i>	20
	<i>Naming Conventions</i>	21
	<i>Removing Modules</i>	21
	<i>Exporting Modules</i>	21
	<i>Importing Modules</i>	22
	<i>Copying Modules to Other Projects</i>	22
2.2	Writing Procedures	23
	<i>Types of VBA Procedure</i>	23
	<i>Inserting Procedures</i>	23
	<i>Starting a Subroutine</i>	24
	<i>Switching off Syntax Error Messages</i>	25
	<i>Setting the Scope of a Procedure</i>	25
2.3	Writing Neat Code	26
	<i>Commenting Your Code</i>	26
	<i>Commenting Out Multiple Lines of Code</i>	27
	<i>Using Blank Lines and Indenting</i>	27
	<i>Indenting Multiple Lines</i>	28
	<i>Changing Indenting Settings</i>	28
	<i>The Continuation Character</i>	28
2.4	Writing Simple VBA Instructions	29
	<i>Objects</i>	29
	<i>Methods and Properties</i>	29
2.5	Tools to Help with Writing Code	30
	<i>Choosing Which Tools are Enabled</i>	30
	<i>Using IntelliSense to Write Code Faster</i>	30
	<i>Using Tooltips</i>	31
	<i>Viewing Data Tips</i>	31

3	SAVING AND OPENING FILES	Page
3.1	Saving VBA Code	32
	<i>Where is Code Stored?</i>	32
	<i>Saving VBA Code</i>	32
	<i>Choosing the Correct File Type</i>	33
3.2	The Personal Macro Workbook	34
	<i>Creating the Personal Macro Workbook</i>	34
	<i>Viewing the Personal Macro Workbook in the VBE</i>	34
	<i>Viewing the Personal Macro Workbook in Excel</i>	35
	<i>Saving the Personal Macro Workbook</i>	35
	<i>Where the Personal Macro Workbook is Stored</i>	35
3.3	Opening Files Which Contain VBA Code	36
	<i>Choosing to Enable VBA Content</i>	36
	<i>Macro Security Settings</i>	36
3.4	Trusted Documents	37
	<i>Viewing Trusted Document Settings</i>	37
	<i>Disabling Trusted Documents</i>	37

4	RUNNING VBA CODE	Page
4.1	Running Code from Excel	38
	<i>Choosing from a List of Macros</i>	38
4.2	Running Code from the VBE	39
	<i>Running a Subroutine</i>	39
	<i>The Debug Toolbar</i>	40
	<i>Compiling Code</i>	40
	<i>Stepping Into and Through Code</i>	41
	<i>Reaching the End of a Procedure</i>	41
	<i>Interrupting a Running Procedure</i>	42
4.3	When Things Go Wrong	43
	<i>Syntax Errors</i>	43
	<i>Compile Errors</i>	43
	<i>Run-Time Errors</i>	44

## TABLE OF CONTENTS (2 of 12)

5	BASIC USER INTERFACES	Page
5.1	Keyboard Shortcuts	45
	<i>Assigning Keyboard Shortcuts in Excel</i>	45
	<i>Assigning Keyboard Shortcuts in Code</i>	45
5.2	Form Control Buttons	46
	<i>Drawing Form Control Buttons</i>	46
	<i>Editing Form Control Buttons</i>	46
5.3	ActiveX Command Buttons	47
	<i>Drawing ActiveX Command Buttons</i>	47
	<i>Attaching Code to the Click Event</i>	47
5.4	AutoShapes and Pictures	48
	<i>Inserting Shapes and Pictures</i>	48
	<i>Assigning a Macro to a Shape or Picture</i>	48
5.5	The Excel Ribbon	49
	<i>Modifying the Quick Access Toolbar</i>	49
	<i>Creating Ribbon Tabs</i>	50

6	WORKING WITH RANGES	Page
6.1	Referring to a Range Object	51
	<i>Referring to a Single Cell</i>	51
	<i>Referring to a Block of Cells</i>	51
	<i>Using Range Names to Refer to Cells</i>	52
	<i>Referring to Non-Contiguous Ranges</i>	52
6.2	The Cells Property	53
	<i>Referring to a Cell with Row and Column Numbers</i>	53
	<i>Referring to a Block of Cells</i>	53
	<i>Referring to Every Cell on a Worksheet</i>	53
6.3	Rows and Columns	54
	<i>The Range, Rows and Columns Properties</i>	54
	<i>The EntireRow and EntireColumn Properties</i>	54
6.4	Referring to the Active or Selected Cell	55
	<i>ActiveCell and Selection</i>	55
	<i>Activate vs. Select</i>	55
6.5	The Offset Property	56
	<i>Offsetting from the ActiveCell</i>	56
	<i>Offsetting a Block of Cells</i>	56
6.6	The End Property	56
	<i>The Four Directions of the End Property</i>	57
	<i>Finding the Start of a List</i>	57
	<i>Finding the Bottom Right Corner of a Table</i>	58
	<i>Finding the Next Blank Cell in a Column</i>	58
	<i>Selecting from the Top to the Bottom of a List</i>	59
	<i>Dealing with Blank Cells</i>	59
6.7	Referring to Regions of Cells	60
	<i>The CurrentRegion Property</i>	60
	<i>The CurrentArray Property</i>	60
6.8	Referring to Special Cells	61
6.9	Referring to Used Cells	62
	<i>Referring to the Used Range</i>	62
	<i>The Last Used Cell</i>	62

7	COLOURS IN VBA	Page
7.1	Excel Colours	63
	<i>The Two Colour Properties of a Range</i>	63
	<i>The ColorIndex Colours</i>	63
7.2	Colour Numbers and Names	64
	<i>VBA Colour Constants</i>	64
	<i>Excel's RGB Constants</i>	64
	<i>The RGB Function</i>	64

## TABLE OF CONTENTS (3 of 12)

<b>8</b>	<b>DISPLAYING MESSAGES</b>	<b>Page</b>
8.1	The MsgBox Function	65
	<i>Syntax and Parameters of a Message Box</i>	65
8.2	Displaying Messages	66
	<i>Displaying a Simple Message</i>	66
	<i>A Note on Using Parentheses</i>	66
	<i>Concatenating a Message</i>	67
	<i>Changing Text Alignment</i>	67
	<i>Using Multiple Lines</i>	68
	<i>Customising the Title</i>	68
8.3	Icons and Buttons	69
	<i>Modifying the Buttons</i>	69
	<i>Setting the Default Button</i>	69
	<i>Displaying Icons</i>	70
	<i>Combining Buttons and Icons</i>	70

<b>9</b>	<b>USER INPUTS</b>	<b>Page</b>
9.1	Asking Users for Input	71
	<i>Where to Store User Input</i>	71
9.2	Asking a Question with a Message Box	72
	<i>The Possible Results of a Message Box</i>	72
	<i>Storing the Result of a Message Box</i>	72
	<i>Testing Which Button was Clicked</i>	73
9.3	The VBA InputBox Function	74
	<i>Syntax and Parameters of the InputBox Function</i>	74
	<i>Using an Input Box to Ask a Question</i>	75
	<i>Setting a Default Value</i>	75
	<i>What Happens if You Click Cancel?</i>	75
	<i>Inputting Different Types of Data</i>	76
9.4	The Excel-Specific InputBox Method	77
	<i>Using the Excel Input Box</i>	77
	<i>Customising the Title and Default Value</i>	77
	<i>Setting the Data Type of the Input Box</i>	78
	<i>Entering an Invalid Value</i>	78
	<i>Selecting Cells</i>	79
	<i>Returning a Reference to a Range</i>	79

<b>10</b>	<b>VARIABLES AND DATA TYPES</b>	<b>Page</b>
10.1	Data Types in VBA	80
	<i>Summary of the VBA Data Types</i>	80
10.2	Declaring and Using Variables	81
	<i>The Dim Statement</i>	81
	<i>Writing To and Reading From Variables</i>	81
10.3	Declared vs. Non-Declared Variables	82
	<i>Non-Declared Variables</i>	82
	<i>Explicitly-Declared Variables</i>	82
	<i>Forcing Explicit Variable Declaration</i>	83
10.4	Variables and Data Types	84
	<i>The Variant Data Type</i>	84
	<i>Declaring Multiple Variables</i>	84
	<i>Choosing the Correct Data Type</i>	85
10.5	Converting Variable Data Types	86
	<i>Implicit Data Type Conversion</i>	86
	<i>The Problem with Implicit Type Conversion</i>	86
	<i>Explicit Data Type Conversion</i>	87
	<i>Checking for Dates and Numbers</i>	87
10.6	The Scope of Variables	88
	<i>Procedure Level Variables</i>	88
	<i>Module Level Variables</i>	89
	<i>Project Level Variables</i>	89
10.7	Constants	90
	<i>Declaring a Constant</i>	90
	<i>The Scope of Constants</i>	90

## TABLE OF CONTENTS (4 of 12)

11	WORKING WITH DATA	Page
11.1	Manipulating Data	91
	<i>The Three Main VBA Data Types</i>	91
	<i>The Basic VBA Operators</i>	91
	<i>Manipulating Values using Functions</i>	92
	<i>Why Some Functions End with a \$ Sign</i>	92
	<i>Using Excel's Worksheet Functions</i>	92
11.2	Working with Numbers	93
	<i>Testing if a Value is a Number</i>	93
	<i>Useful Numeric Functions</i>	93
11.3	Working with Dates	94
	<i>Testing if a Value is a Date</i>	94
	<i>Writing Dates in the VBE</i>	94
	<i>Arithmetic with Dates</i>	94
	<i>Useful Date Functions</i>	95
	<i>Intervals for Date Functions</i>	96
	<i>Setting the First Day of the Week</i>	96
	<i>Formatting Dates</i>	96
11.4	Working with Strings	97
	<i>Concatenating Strings</i>	97
	<i>Character Codes</i>	97
	<i>Special Character Constants</i>	98
	<i>Case Sensitivity</i>	98
	<i>Useful String Functions</i>	99

12	TESTING CONDITIONS	Page
12.1	The If Statement	100
	<i>Single-Line If Statements</i>	100
	<i>The Else Clause</i>	100
	<i>Block If Statements</i>	101
	<i>Nested Ifs</i>	102
	<i>The Elself Statement</i>	102
12.2	Logical Tests and Operators	103
	<i>Comparison Operators</i>	103
	<i>Logical Tests and Boolean Values</i>	103
12.3	Combining Conditions	104
	<i>The Or Operator</i>	104
	<i>The And Operator</i>	104
	<i>The Xor Operator</i>	104
12.4	Comparing Strings	105
	<i>Testing if Two Strings are Equal</i>	105
	<i>Converting the Case of Text</i>	105
	<i>Making All Text Comparisons Case-Insensitive</i>	105
	<i>Relative Comparisons with Strings</i>	105
	<i>The Like Operator and Wildcards</i>	106
12.5	Conditional Functions	107
	<i>The If Function</i>	107
	<i>The Switch Function</i>	107
12.6	The Select Case Statement	108
	<i>A Basic Select Case Statement</i>	108
	<i>Testing Multiple Values</i>	108
	<i>Testing a Range of Values</i>	108

13	FOR NEXT LOOPS	Page
13.1	The For Next Loop	109
	<i>Looping a Set Number of Times</i>	109
	<i>The Step Statement</i>	110
	<i>Exiting from a For Next Loop</i>	110
	<i>Nesting For Next Loops</i>	111
	<i>Looping a Variable Number of Times</i>	111

14	CONDITIONAL LOOPS	Page
14.1	The Do Loop	112
	<i>Exiting from a Do Loop</i>	112
14.2	Do Until Loops	113
	<i>Writing a Do Until Loop</i>	113
	<i>The Loop Until Statement</i>	113
	<i>Breaking Out of a Loop</i>	113
14.3	Do While Loops	114
	<i>Writing a Do While Loop</i>	114
	<i>The Loop While Statement</i>	114

## TABLE OF CONTENTS (5 of 12)

15	HOW VBA WORKS	Page
15.1	Object Oriented Programming	115
	<i>The Building Blocks of an Object Oriented Language</i>	115
15.2	Objects	116
	<i>Referring to Objects by Name</i>	116
	<i>Referring to Objects by Index Number</i>	116
	<i>Qualifying References to Objects</i>	117
	<i>Using Keywords to Reference Objects</i>	117
	<i>Using Object Codenames</i>	118
	<i>Using Object Variables</i>	118
15.3	Collections	119
	<i>Referring to Collections</i>	119
15.4	Methods	120
	<i>Applying Methods to Objects</i>	120
	<i>Passing Arguments to Methods</i>	120
	<i>Returning Values and References from Methods</i>	121
	<i>When to use Parentheses</i>	121
15.5	Properties	122
	<i>Writing to a Property</i>	122
	<i>Read-Only Properties</i>	122
	<i>Property Data Types</i>	122
	<i>Reading from a Property</i>	123
	<i>Properties and Parameters</i>	123
15.6	Getting Help in VBA	124
	<i>The Object Browser</i>	124
	<i>Context Sensitive Help</i>	125
	<i>Recording a Macro</i>	125

16	FOR EACH LOOPS	Page
16.1	Looping Through Collections	126
	<i>The For Each Loop</i>	126
	<i>A Basic Example</i>	126
16.2	Looping Over Worksheets, Charts and Sheets	127
	<i>Protecting all Worksheets</i>	127
	<i>Excluding Worksheets</i>	127
	<i>Looping Through Chart Sheets</i>	128
	<i>Looping Through All Sheets</i>	128
	<i>Looping Through Objects on a Sheet</i>	128
16.3	Looping Over the Workbooks Collection	129
	<i>Processing all Open Workbooks</i>	129
16.4	Looping Over a Collection of Range Objects	130
	<i>Specifying the Range to Loop Over</i>	130
	<i>Looping Through a Column of Data</i>	130
16.5	Nesting For Each Loops	131
	<i>Looping Over Shapes on All Worksheets</i>	131
	<i>Looping Through Sheets in All Open Workbooks</i>	131

## TABLE OF CONTENTS (6 of 12)

<b>17</b>	<b>DEBUGGING</b>	<b>Page</b>
17.1	Debugging Code	132
	<i>The Debug Toolbar</i>	132
17.2	Running Code	133
	<i>Running a Procedure from Start to End</i>	133
	<i>Running a Procedure in Break Mode</i>	133
	<i>Stepping Through Code</i>	134
	<i>Changing the Next Instruction</i>	134
	<i>Editing Code in Break Mode</i>	134
17.3	Debugging Modular Code	135
	<i>Viewing the Definition of a Procedure</i>	135
	<i>Stepping Over a Procedure Call</i>	135
17.4	Breakpoints	136
	<i>Setting and Removing Breakpoints</i>	136
	<i>The Stop Statement</i>	136
	<i>Breaking Conditionally</i>	136
17.5	The Immediate Window	137
	<i>Executing Instructions in the Immediate Window</i>	137
	<i>Asking Questions in the Immediate Window</i>	137
	<i>Printing to the Immediate Window</i>	137
17.6	The Locals Window	138
	<i>Observing Variables</i>	138
17.7	The Watch Window	139
	<i>Adding an Expression to Watch</i>	139
	<i>Types of Watch</i>	139
	<i>Adding a Quick Watch</i>	140
	<i>Editing and Removing Watches</i>	140
17.8	The Call Stack	141
	<i>Displaying the Call Stack</i>	141
	<i>Using the Call Stack</i>	141

<b>18</b>	<b>HANDLING ERRORS</b>	<b>Page</b>
18.1	Run-Time Errors in VBA	142
18.2	Error Handling in VBA	143
	<i>Identifying Potential Run-Time Errors</i>	143
	<i>The On Error Statement</i>	143
18.3	Using the On Error Statement	144
	<i>Ignoring Run-Time Errors</i>	144
	<i>Disabling an Error Handler</i>	144
18.4	Creating a Custom Error Handler	145
	<i>Redirecting Your Code</i>	145
	<i>Writing the Error-Handling Section</i>	145
	<i>Exiting a Procedure before the Error-Handling Code</i>	146
	<i>The Complete Example</i>	146
18.5	Resuming After an Error	147
	<i>Resuming at the Original Line</i>	147
	<i>Resuming at the Next Line</i>	147
	<i>Resuming at a Specified Line</i>	148
	<i>Why use Resume and Not GoTo?</i>	148
18.6	The Err Object	149
	<i>Getting the Error Number and Description</i>	149
	<i>A Catch-All Approach to Error-Handling</i>	149

## TABLE OF CONTENTS (7 of 12)

19	EVENTS	Page
19.1	Event Handlers	150
	<i>Objects Which Have Events</i>	150
	<i>Event Procedures vs. Normal Procedures</i>	150
19.2	Creating a Simple Event Handler	151
	<i>Accessing the Object's Code</i>	151
	<i>Choosing the Event</i>	151
	<i>Writing the Code</i>	152
	<i>Triggering the Event</i>	152
19.3	Workbook Events	153
	<i>The Before Close Event</i>	153
	<i>The Before Save Event</i>	154
	<i>The Before Print Event</i>	154
	<i>The New Sheet Event</i>	155
	<i>New Chart</i>	155
19.4	Worksheet Events	156
	<i>The Selection Change Event</i>	156
	<i>The Change Event</i>	157
	<i>Checking if the Target is Within a Specific Range</i>	157
19.5	ActiveX Controls	158
	<i>Drawing ActiveX Controls</i>	158
	<i>Changing Properties of the Control</i>	158
	<i>Adding Code to the Control's Events</i>	159
	<i>Prevent Controls from Taking the Focus</i>	159

20	CREATING USER FORMS	Page
20.1	User Forms	160
	<i>Creating a Working Form</i>	160
	<i>Our Example</i>	160
20.2	Creating a User Form	161
	<i>Inserting a User Form into a Project</i>	161
	<i>Switching Between Form Views</i>	162
	<i>Removing Forms</i>	162
20.3	Form Properties	163
	<i>Changing the Properties of a Form</i>	163
	<i>Some Common Form Properties</i>	163
	<i>Choosing Colours</i>	164
	<i>Setting Font Properties</i>	164
20.4	Form Controls	165
	<i>The Toolbox</i>	165
	<i>Drawing a Control on a Form</i>	165
20.5	Manipulating Controls	166
	<i>Selecting a Control</i>	166
	<i>Selecting Multiple Controls</i>	166
	<i>Resizing Controls</i>	167
	<i>Moving Controls</i>	167
	<i>Deleting Controls</i>	167
	<i>Copying and Pasting Controls</i>	167
20.6	Laying Out Controls	168
	<i>The Form Grid</i>	168
	<i>The UserForm Toolbar</i>	168
20.7	Grouping Controls	169
	<i>Grouping a Set of Controls</i>	169
	<i>Using Frames to Group Controls</i>	169
20.8	Control Properties	170
	<i>Naming Controls</i>	170
	<i>Naming Conventions for Controls</i>	170
	<i>Size and Position Properties</i>	171
	<i>Formatting Properties</i>	171

21	RUNNING USER FORMS	Page
21.1	Running a Form	172
	<i>Choosing to Run a Form</i>	172
	<i>Closing a Running Form</i>	172
21.2	Navigating a Form	173
	<i>Tab Order</i>	173
	<i>Accelerator Keys</i>	174
	<i>Keyboard Shortcuts</i>	174
	<i>The Default and Cancel Buttons</i>	175



## TABLE OF CONTENTS (8 of 12)

22	ADDING CODE TO FORMS	Page
22.1	Making Forms Work	176
	<i>Our Example</i>	176
22.2	Running User Forms	177
	<i>Running a Form as a Developer</i>	177
	<i>Running a Form as a User</i>	177
22.3	Adding Code to a Form	178
	<i>Viewing a Form's Code</i>	178
22.4	Referring to Forms and Controls	179
	<i>Referring to a Form</i>	179
	<i>The UserForms Collection</i>	179
	<i>Looping Over the UserForms Collection</i>	180
	<i>Referring to Controls on a Form</i>	180
	<i>Looping Over the Controls Collection</i>	180
22.5	Form and Control Events	181
	<i>Initialising a Form</i>	181
	<i>Clicking the Cancel Button</i>	181
	<i>Clicking the Add to List Button</i>	182
	<i>Writing Modular Code in Forms</i>	182
22.6	Validating User Inputs	183
	<i>The Data Events of a Text Box</i>	183
	<i>Deciding on Your Validation Rules</i>	183
	<i>Creating Basic Validation Code</i>	184
	<i>Selecting the Text in a Text Box</i>	184
	<i>Ideas for Less-Intrusive Validation</i>	185
	<i>Resetting the Formatting Properties</i>	185
	<i>Using Hidden Labels</i>	185
	<i>Validation at the Form Level</i>	186
	<i>Setting the Focus to a Control</i>	186
	<i>Looping over Controls</i>	187
	<i>Validating Every Text Box in One Pass</i>	187

23	ADVANCED FORM CONTROLS	Page
23.1	Beyond the Basics	188
	<i>The Advanced Controls Available</i>	188
23.2	Frames	189
	<i>Drawing Frames and Controls</i>	189
	<i>Looping Through Controls in a Frame</i>	189
23.3	Combo Box and List Box Controls	190
	<i>Setting the Row Source</i>	190
	<i>The List Property</i>	191
	<i>Adding Items Individually</i>	191
	<i>Removing and Clearing Items</i>	192
	<i>Referring to the Selected Item</i>	192
	<i>Changing the List Style</i>	193
	<i>Restricting Choices in a Combo Box</i>	193
	<i>Allowing Multiple Selections in a List Box</i>	193
	<i>Referring to Multiple Selected Items</i>	194
	<i>Working with Multiple Columns</i>	194
23.4	Option Buttons	195
	<i>Grouping Option Buttons</i>	195
	<i>Framing Option Buttons</i>	195
	<i>Setting a Default Option for a Group</i>	196
	<i>Using the Value of an Option Button</i>	196
	<i>The Click Event</i>	196
23.5	Check Boxes and Toggle Buttons	197
	<i>Check Box and Toggle Button Values</i>	197
	<i>The Click Event</i>	197
23.6	Spin Buttons and Scroll Bars	198
	<i>Drawing Spin Buttons and Scroll Bars</i>	198
	<i>Scrolling Properties</i>	198
	<i>The Value Property</i>	198
	<i>The Change Event</i>	199
	<i>The SpinUp and SpinDown Events</i>	199
23.7	MultiPage Controls	200
	<i>Selecting Parts of a MultiPage Control</i>	200
	<i>Working with Pages</i>	200
	<i>The Index and Value Properties</i>	201
	<i>Looping Through Pages and Controls</i>	201

## TABLE OF CONTENTS (9 of 12)

24	<b>CONTROLLING OTHER APPLICATIONS</b>	Page
24.1	Referencing Object Libraries	202
	<i>Setting a Reference to an Object Library</i>	202
	<i>The Default References</i>	203
	<i>References and the Object Browser</i>	203
	<i>Microsoft Office Version Numbers</i>	203
24.2	An Example for Word	204
	<i>Setting a Reference to the Word Object Library</i>	204
	<i>Declaring a Variable for Word</i>	204
	<i>Creating a New Instance of Word</i>	205
	<i>Auto-Instancing Variables</i>	205
	<i>Showing and Activating Word</i>	206
	<i>Creating a New Document</i>	206
	<i>Writing and Formatting Text in Word</i>	207
	<i>Copying from Excel to Word</i>	207
	<i>Saving the Document and Closing Word</i>	208
	<i>The Complete Example</i>	208
24.3	An Example for PowerPoint	209
	<i>Setting a Reference to the PowerPoint Object Library</i>	209
	<i>Opening PowerPoint and Creating a Presentation</i>	209
	<i>Creating a Title Slide</i>	209
	<i>Copying from Excel to PowerPoint</i>	210
	<i>Moving and Resizing PowerPoint Objects</i>	210
	<i>Saving the Presentation and Closing PowerPoint</i>	211
	<i>The Complete Example</i>	211
24.4	An Example for Outlook	212
	<i>Setting a Reference to the Outlook Object Library</i>	212
	<i>The Complete Example</i>	212
24.5	Controlling Applications without References	213
	<i>The CreateObject Function</i>	213
	<i>Using Object Variables</i>	213
	<i>Converting Constants to Numbers</i>	214
	<i>Getting a Reference to a Running Application</i>	215
	<i>Testing the Version of an Application</i>	216
24.6	Referencing Other VBA Projects	217
	<i>Setting a Reference to a VBA Project</i>	217
	<i>Creating Excel Add-Ins</i>	218
	<i>Loading Excel Add-Ins</i>	218

25	<b>CONNECTING TO DATABASES</b>	Page
25.1	ActiveX Data Objects	219
	<i>A Brief Version History</i>	219
	<i>Referencing the ADO Library</i>	219
25.2	Connecting to an External Database	220
	<i>Setting the Connection String</i>	220
25.3	Creating Connections in Access	221
	<i>Referencing the CurrentProject's Connection</i>	221
25.4	ADO Recordsets	222
	<i>Creating a Recordset</i>	222
	<i>Setting the Source of the Recordset</i>	222
	<i>Setting the Lock Type</i>	223
	<i>Setting the Cursor Type</i>	223
	<i>Opening and Closing a Recordset</i>	224
	<i>Copying Data into Excel</i>	224
25.5	Moving in a Recordset	225
	<i>Moving the Cursor</i>	225
	<i>Reaching the End of a Recordset</i>	225
	<i>Looping Over a Recordset</i>	226
	<i>Referring to Fields</i>	226
25.6	Finding and Filtering Records	227
	<i>The Find Method</i>	227
	<i>Repeated Finds</i>	227
	<i>Applying a Filter</i>	228
	<i>Removing a Filter</i>	228
	<i>Adding Criteria to a SQL Select Statement</i>	229
	<i>Creating Dynamic SQL Statements</i>	229
25.7	Modifying Data	230
	<i>Adding New Records</i>	230
	<i>Editing Existing Records</i>	230
	<i>Deleting Records</i>	230
25.8	ADO Commands	231
	<i>Creating a New Command Object</i>	231
	<i>Setting the Command Text</i>	231
	<i>Executing the Command</i>	231
25.9	Using DAO	232
	<i>Referencing the Correct Object Library</i>	232
	<i>Opening a Database</i>	232
	<i>Creating a Recordset</i>	232

## TABLE OF CONTENTS (10 of 12)

<b>26</b>	<b>FILES AND FOLDERS</b>	<b>Page</b>
26.1	Working with Files and Folders	233
	<i>The Scripting Runtime Library</i>	233
	<i>Creating a FileSystemObject</i>	233
26.2	Basic File and Folder Techniques	234
	<i>Testing if a File or Folder Exists</i>	234
	<i>Creating a Folder</i>	234
	<i>Copying and Moving Files and Folders</i>	234
	<i>Deleting Files and Folders</i>	235
	<i>Renaming Files and Folders</i>	235
	<i>Getting a Reference to a File or Folder</i>	235
26.3	Looping Over Files and Folders	236
	<i>Looping Over Files</i>	236
	<i>Looping Over Folders</i>	236
	<i>Recursively Looping Over Subfolders</i>	237
26.4	Working with Text Files	238
	<i>Creating and Writing to a Text File</i>	238
	<i>Opening a Text File</i>	238
	<i>Reading from a Text File</i>	239
26.5	Using VBA's FileSystem Methods	240
	<i>Creating Folders</i>	240
	<i>Deleting Files and Folders</i>	240
	<i>Copying Files</i>	240
	<i>Renaming Files</i>	240

<b>27</b>	<b>FILE DIALOG BOXES</b>	<b>Page</b>
27.1	Working with File Dialogs	241
	<i>Types of File Dialog Box</i>	241
	<i>Displaying a File Dialog Box</i>	242
	<i>Performing the Default Action</i>	242
27.2	Customising File Dialogs	243
	<i>Changing the Title and Button Name</i>	243
	<i>Setting the Initial Location</i>	243
	<i>Allowing Multiple Selections</i>	244
	<i>Creating File Filters</i>	244
27.3	Picking Files and Folders	245
	<i>Returning a File or Folder Path</i>	245
	<i>Testing Which Button was Clicked</i>	245
	<i>Dealing with Multiple Selections</i>	246
	<i>Using Multiple File Dialogs</i>	246

<b>28</b>	<b>CLASS MODULES</b>	<b>Page</b>
28.1	What are Class Modules?	247
	<i>Why Create Classes?</i>	247
	<i>Important Terminology</i>	248
	<i>Debugging in Class Modules</i>	248
28.2	Designing a Class	249
	<i>Our Example Film Class</i>	249
28.3	Creating a Class	250
	<i>Inserting a Class Module</i>	250
	<i>Renaming a Class Module</i>	250
	<i>Creating a New Instance of a Class</i>	250
28.4	Creating Basic Properties	251
	<i>Basic Properties</i>	251
	<i>Disadvantages of Basic Properties</i>	251
28.5	Creating Full Properties	252
	<i>Assigning a Value to a Property</i>	252
	<i>Reading a Value from a Property</i>	253
	<i>Assigning an Object to a Property</i>	253
	<i>Writing Additional Code in Properties</i>	254
	<i>Read-Only Properties</i>	254
28.6	Creating Methods	255
	<i>Writing Methods in a Class Module</i>	255
	<i>Using Class Methods</i>	255
28.7	Class Module Events	256
	<i>Creating Class Module Event Handlers</i>	256
	<i>Triggering Class Events</i>	256
28.8	Sharing Class Modules	257
	<i>Step 1 – Rename the VBA Project</i>	257
	<i>Step 2 – Make the Class Public</i>	257
	<i>Step 3 – Create a Function to Return an Instance of the Class</i>	257
	<i>Step 4 – Reference the Class Project</i>	258
	<i>Step 5 – Consume the Class</i>	258

## TABLE OF CONTENTS (11 of 12)

29	COLLECTIONS AND DICTIONARIES	Page
29.1	What are Collections?	259
	<i>Custom Collections and Dictionaries</i>	259
29.2	Untyped Collections	260
	<i>Creating a New Collection</i>	260
	<i>Adding Items to a Collection</i>	260
	<i>Adding Custom Classes to a Collection</i>	261
	<i>Referencing Collection Items</i>	261
	<i>Removing Items from a Collection</i>	261
	<i>Looping Over Collections</i>	262
29.3	Typed Collections	263
	<i>The Problem with Untyped Collections</i>	263
	<i>Creating a Collection Class</i>	263
	<i>Populating a Typed Collection</i>	264
	<i>Looping Over a Typed Collection</i>	264
	<i>Referencing Items in a Typed Collection</i>	264
29.4	Dictionaries	265
	<i>Referencing the Scripting Runtime Library</i>	265
	<i>Creating a New Dictionary</i>	265
	<i>Adding Items to a Dictionary</i>	266
	<i>Referring to Dictionary Items</i>	266
	<i>Automatically Creating Keys</i>	267
	<i>Checking if a Key Exists</i>	267
	<i>The Compare Mode</i>	268
	<i>Removing Items from a Dictionary</i>	268
	<i>Replacing Dictionary Values</i>	269
	<i>Replacing Dictionary Objects</i>	269
	<i>Looping Over Dictionaries</i>	270

30	ARRAYS	Page
30.1	Overview of Arrays	271
	<i>Viewing the Contents of Arrays</i>	271
30.2	Declaring Arrays	272
	<i>Setting the Dimensions of an Array</i>	272
	<i>Changing the Base of Arrays</i>	272
	<i>Declaring Multi-Dimensional Arrays</i>	272
30.3	Populating Arrays	273
	<i>Assigning Values to an Array</i>	273
	<i>Assigning Objects to Arrays</i>	273
30.4	Reading from Arrays	274
	<i>Referring to a Specific Element</i>	274
	<i>Looping Over an Array</i>	274
	<i>The Bounds of an Array</i>	275
	<i>Using For Each Loops</i>	275
30.5	Dynamic Arrays	276
	<i>Declaring an Empty Array</i>	276
	<i>Re-Dimensioning an Array</i>	276
	<i>Preserving the Contents of an Array</i>	276
30.6	Arrays in Excel	277
	<i>Assigning a Range to an Array</i>	277
	<i>Calculating in an Array</i>	277
	<i>Assigning an Array to a Range</i>	277

## TABLE OF CONTENTS (12 of 12)

<b>31</b>	<b>MODULAR CODE, PARAMETERS AND FUNCTIONS</b>	<b>Page</b>
31.1	Modular Code	278
	<i>Our Example</i>	278
31.2	Breaking a Procedure into Parts	279
	<i>Creating Module Level Variables</i>	279
	<i>Getting Input from the User</i>	279
	<i>Retrieving the Related Values</i>	280
	<i>Building and Showing a Message</i>	280
	<i>Putting it all Together</i>	280
31.3	Procedures and Parameters	281
	<i>Our Example</i>	281
	<i>Defining Parameters</i>	281
	<i>Calling a Procedure which has Parameters</i>	282
	<i>Optional Parameters</i>	282
	<i>Assigning Default Values to Parameters</i>	283
	<i>Testing for Missing Arguments</i>	283
	<i>ParamArrays</i>	283
31.4	Passing Arguments ByRef and ByVal	284
	<i>Passing Arguments by Reference</i>	284
	<i>Passing Arguments by Value</i>	285
	<i>Passing Arguments in Parentheses</i>	285
31.5	Functions vs. Subroutines	286
	<i>Returning a Value from a Function</i>	286
	<i>Returning a Reference from a Function</i>	286
	<i>Calling a Function</i>	287
	<i>Using Functions in a Worksheet</i>	287
	<i>Defining Function Parameters</i>	287
31.6	Debugging Modular Code	288
	<i>Viewing the Definition of a Procedure</i>	288
	<i>Stepping Over a Procedure Call</i>	288


<b>32</b>	<b>CONSTANTS AND ENUMERATIONS</b>	<b>Page</b>
32.1	Working with Constants	289
	<i>Declaring Constants</i>	289
	<i>Referencing Constants</i>	289
32.2	Enumerations	290
	<i>Declaring Enumerations</i>	290
	<i>Referencing Enumerations</i>	290
	<i>Using Enumerations as Data Types</i>	291
	<i>Converting an Enumeration to Text</i>	291
	<i>Enumerations for Colours</i>	292

<b>33</b>	<b>SHAPES</b>	<b>Page</b>
33.1	Introduction to Shapes	293
	<i>The Shapes Collection</i>	293
33.2	Referring to Shapes	294
	<i>Names and Index Numbers</i>	294
	<i>Referring to a Range of Shapes</i>	294
	<i>Referring to Selected Shapes</i>	294
	<i>Referring to Newly Added Shapes</i>	295
	<i>Looping Over the Shapes Collection</i>	295
33.3	Shape Size and Position	296
	<i>Changing the Size and Position</i>	296
	<i>Sizing and Positioning Relative to Other Objects</i>	296
33.4	Adding Shapes	297
	<i>Adding a Basic AutoShape</i>	297
	<i>Labels and Textboxes</i>	297
	<i>WordArt</i>	298
	<i>Pictures</i>	298
	<i>Form Controls</i>	299
33.5	Formatting Shapes	300
	<i>Changing Shape Colours</i>	300
	<i>Colour Gradients</i>	301
	<i>Other Formatting Options</i>	302
	<i>Setting Default Shape Formats</i>	303
	<i>Copying Formats between Shapes</i>	303
	<i>Using Shape Styles</i>	303
33.6	Shape Adjustments	304
	<i>Referring to Adjustments</i>	304
	<i>Adjusting Adjustments</i>	304
33.7	Adding Text to AutoShapes	305
	<i>The TextFrame and TextFrame2 Objects</i>	305
	<i>Adding Text to a Shape</i>	305
33.8	Formatting Text in a Shape	306
	<i>Basic Font Formatting</i>	306
	<i>Changing the Colour of Text</i>	306
	<i>Formatting Part of the Text</i>	307
	<i>Aligning Text in a Shape</i>	307
	<i>Changing Text Orientation</i>	307
33.9	Connectors and Lines	308
	<i>Drawing Straight Lines</i>	308
	<i>Adding Multi-Point Lines and Curves</i>	308
	<i>Drawing Freeform Lines</i>	309
	<i>Creating Enclosed Shapes</i>	309
	<i>Connectors</i>	310

# 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.

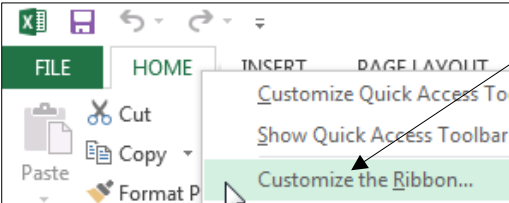


**Wise Owl's Hint**

*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.*

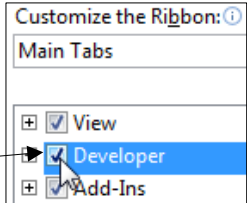
### Displaying the Developer Ribbon Tab

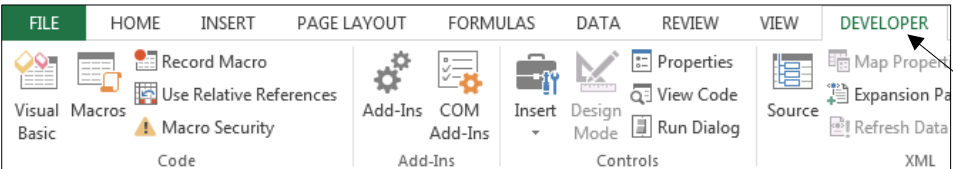
Although you can use the VBE without it, the *Developer* ribbon tab contains some useful tools for working with your VBA code. To display the **Developer** tab:



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

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





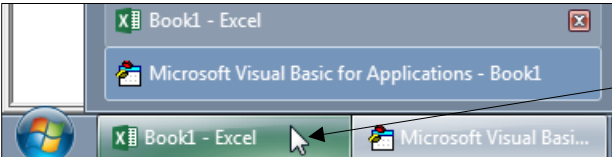
c) Click here to select the **Developer** tab and see the extra tools to which you now have access.

### Opening the VB Editor

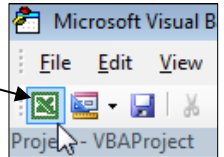
You can open the VBE using one of these options:

<i>Ribbon</i>	<i>Keyboard</i>
<b>Developer   Visual Basic</b>	<b>Alt + F11</b>

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:

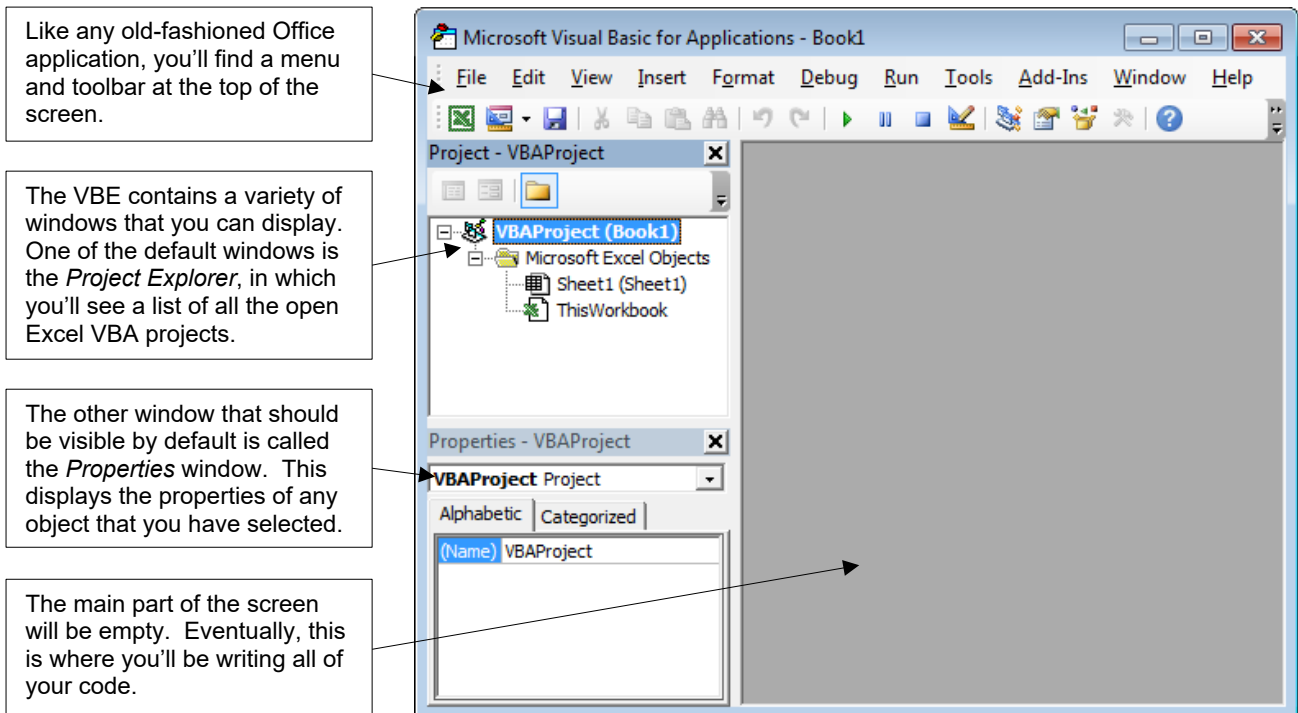


You can use the Windows task bar to select the Excel workbook that you want to see. You can also just click this button on the VBE toolbar.



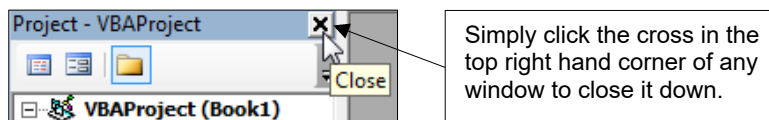
## 1.2 The VBE Screen

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

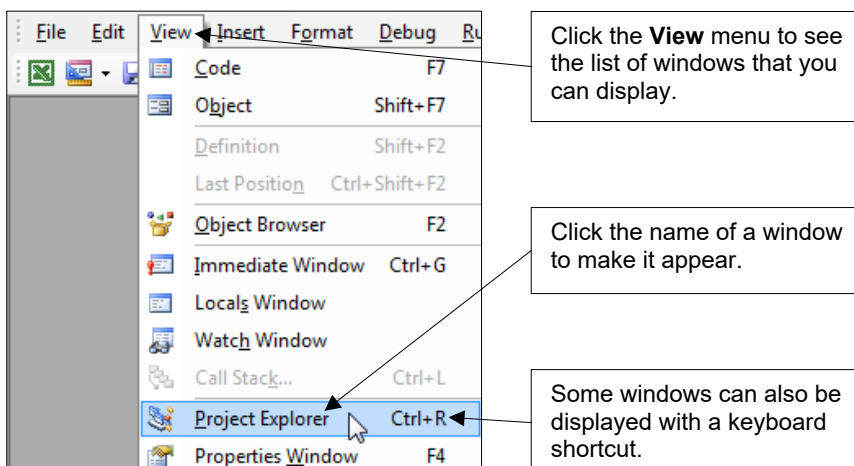


### Opening and Closing Windows

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

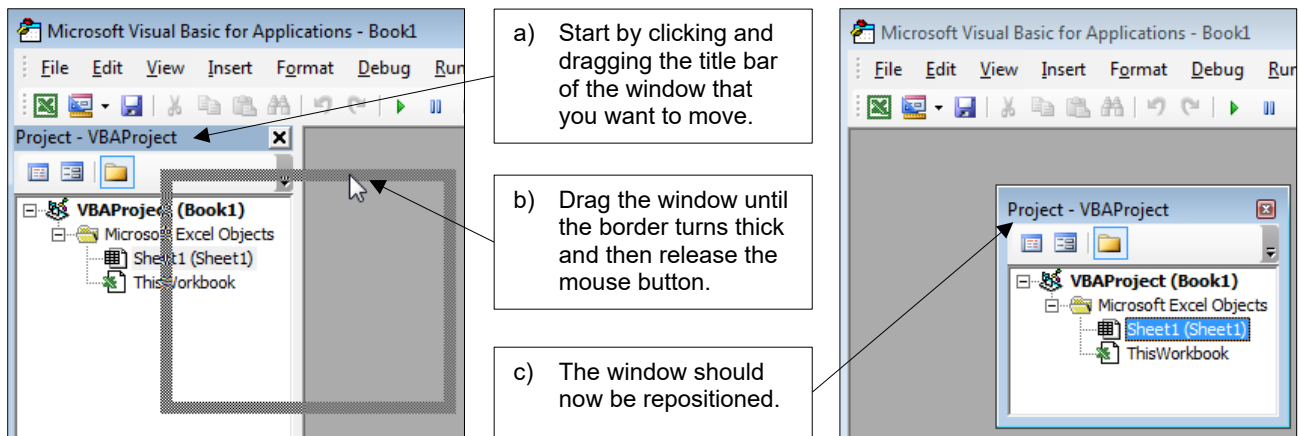


You can use the **View** menu to display any window that you've closed down, and also to view the other available windows.



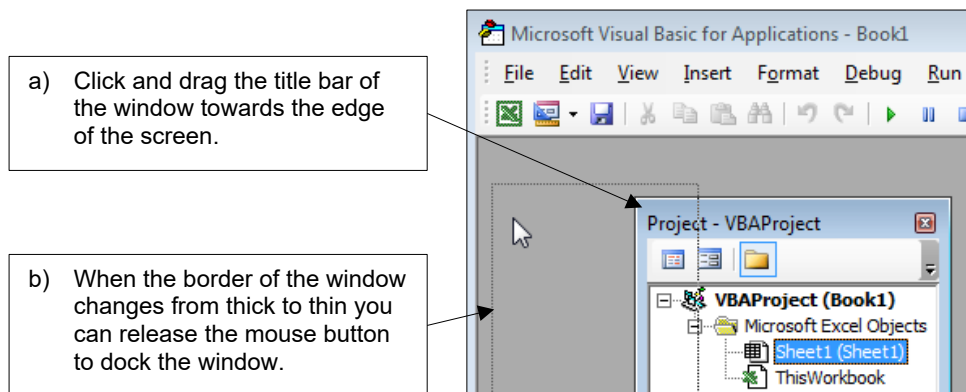
## Repositioning Windows

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.

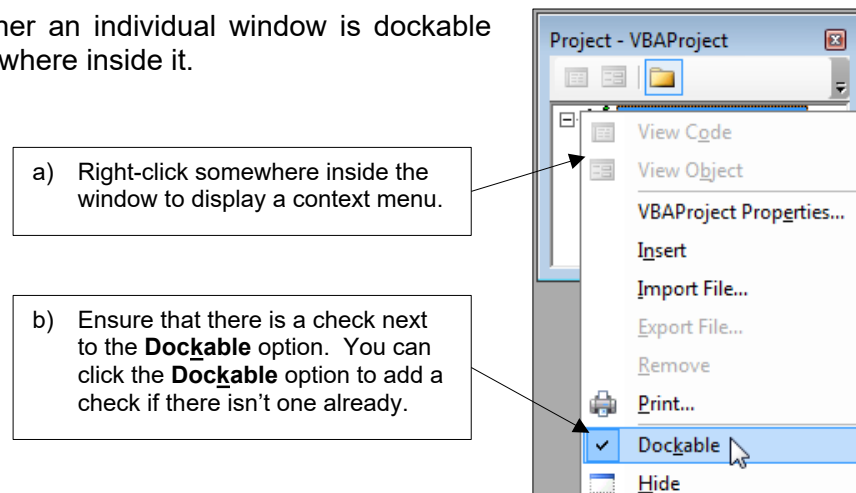


## Docking Windows

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.

Each Excel workbook has its own VBA project which is displayed in the Project Explorer. In this example we have two workbooks and their corresponding VBA projects open.

A VBA project can contain several different types of item. You'll learn about most of them in the rest of this manual.

Click the yellow folder to change how items are displayed: either organised into different folders, or displayed in a single list.

You can collapse and expand the items in a project or a folder by clicking the + and - symbols.

### The Properties Window

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

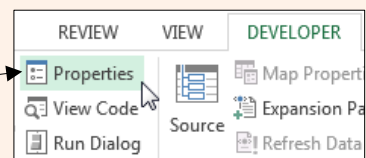
You can use this drop down list at the top of the Properties window to select a different object.

You can display the list of properties either alphabetically or categorised by clicking the tabs at the top of the Properties window.



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: **T**ools | **O**ptions...

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

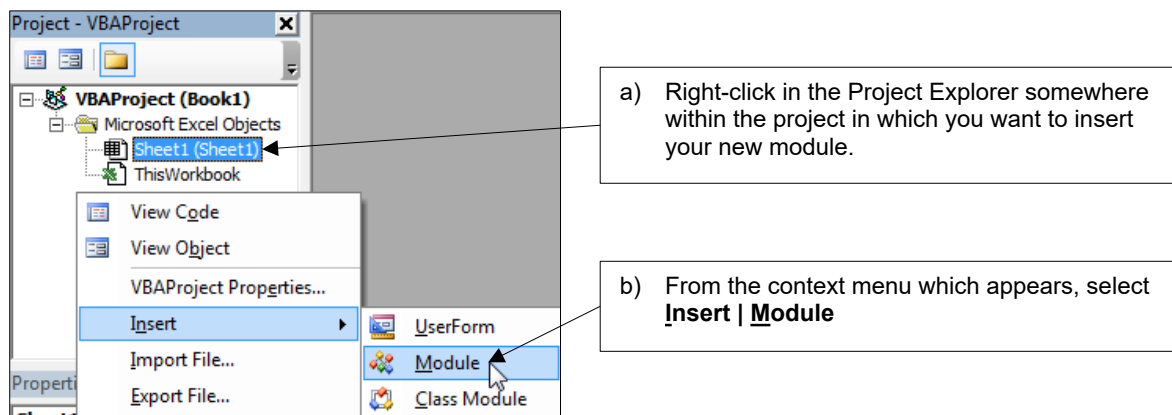
This chapter introduces you to the basics of writing VBA code. You won't create a world-changing application here, but you will learn the fundamental techniques you'll need to start writing one.

### 2.1 Modules

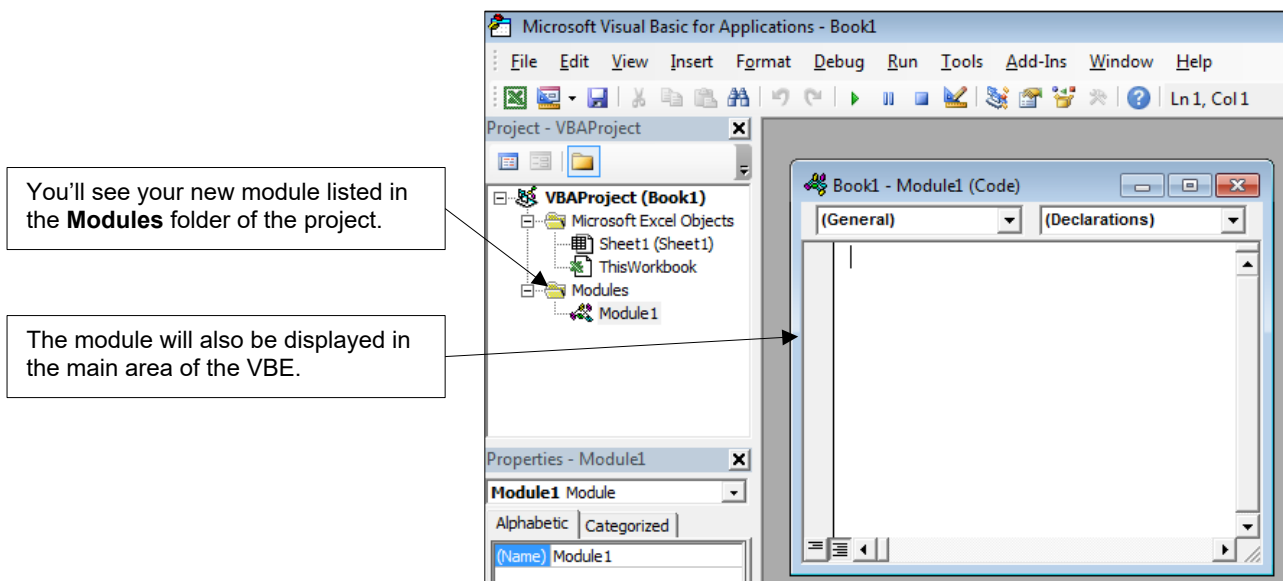
Before you can start writing code you'll need somewhere to put it. You can write VBA code in a variety of places in a project but the most common location is in a *module*.

#### 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:

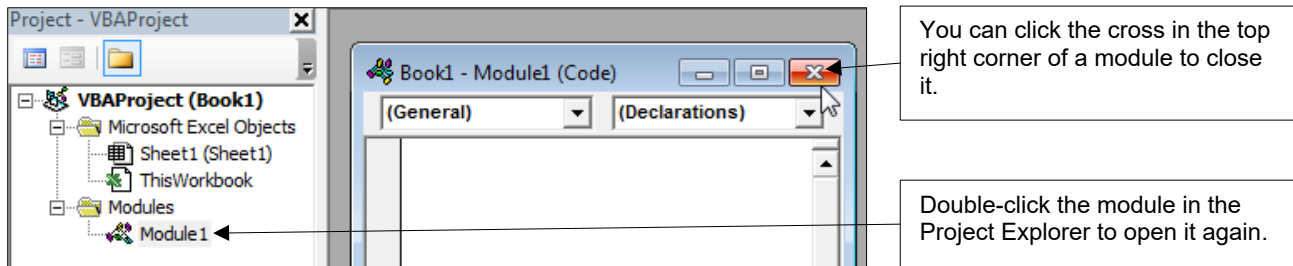


Your new module will appear in the **Modules** folder of your project and will automatically open in the main window of the VBE.



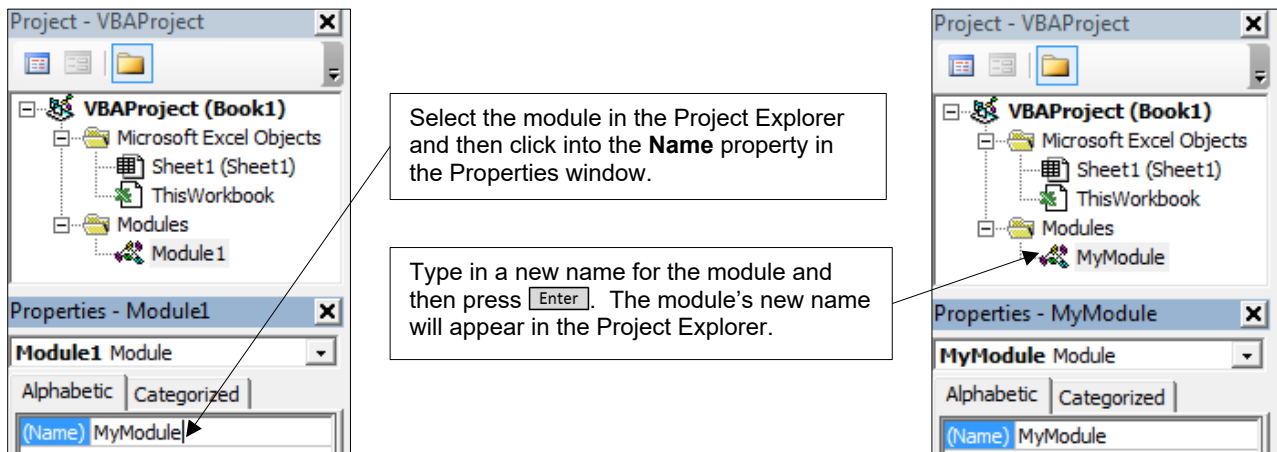
## Opening and Closing Modules

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 <b>Workbook</b> or <b>Worksheet</b> .

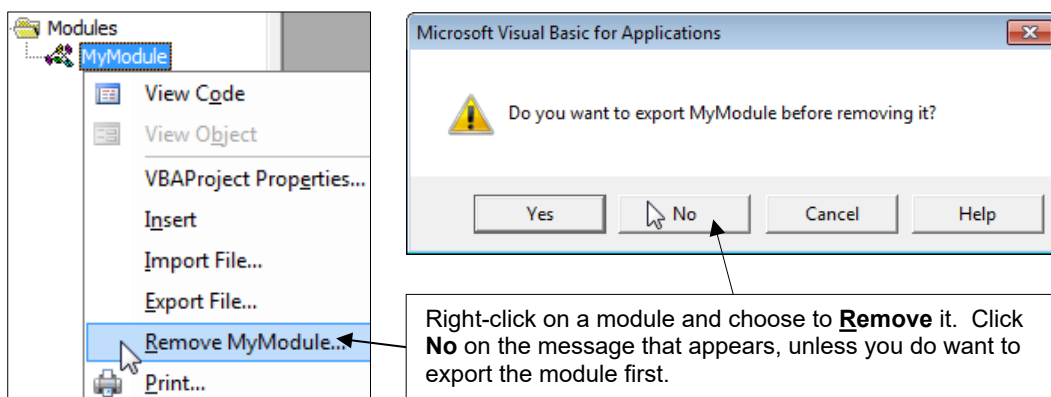
## Naming Conventions

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

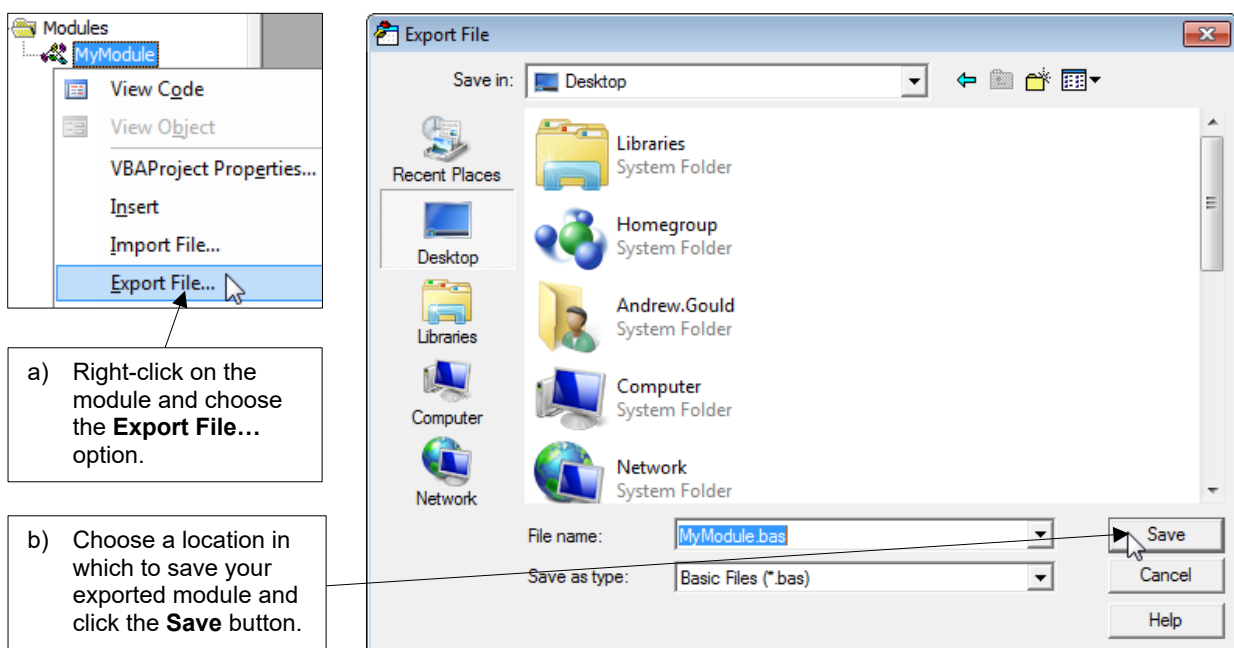
## Removing Modules

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



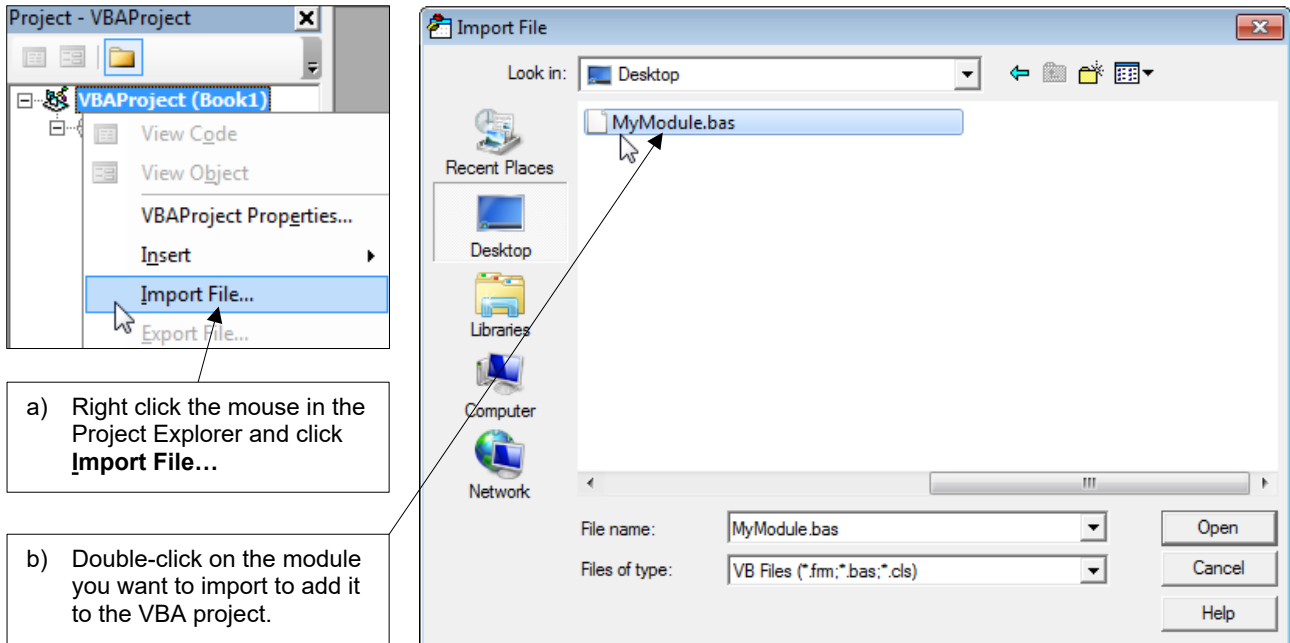
## Exporting Modules

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



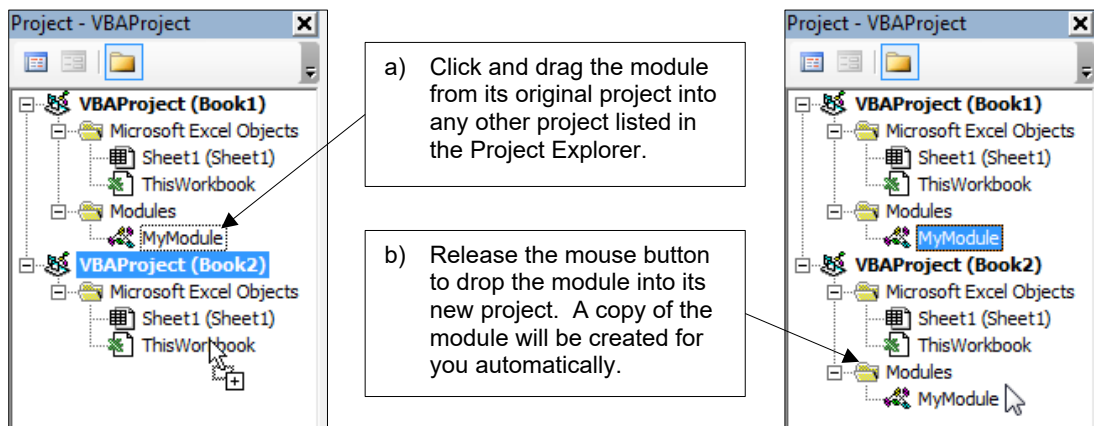
## Importing Modules

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



*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

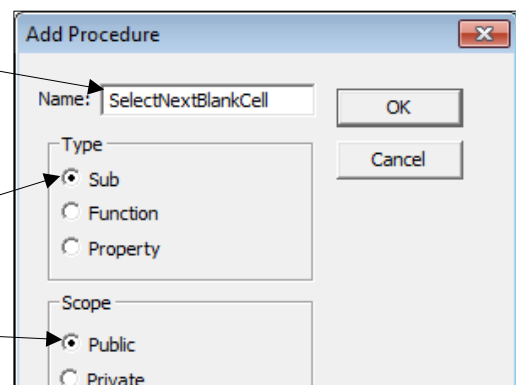
There are three types of procedure you can write in VBA: *subroutines*; *functions*; and *properties*. The table below summarises what each one is, and shows a fairly useless example of each.

Procedure	Description	Example
<i>Subroutine</i>	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 <i>subs</i> or <i>macros</i> .	<pre>Sub MyUselessSubroutine ()     MsgBox "This is useless" End Sub</pre>
<i>Function</i>	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.	<pre>Function IsThisUseless() As Boolean     IsThisUseless = True End Function</pre>
<i>Property</i>	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: <i>Let</i> , <i>Get</i> and <i>Set</i> .	<pre>Property Get Uselessness() As String     Uselessness = "Very useless" End Property</pre>

### 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...**

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

The diagram illustrates the process of starting a subroutine in the VB Editor. It consists of two screenshots of the code editor window and three numbered instructions.

**Step 1:** The first screenshot shows the code editor with the text `sub MyFirstSubroutine` entered. An arrow points from the text to instruction (a).

**Step 2:** The second screenshot shows the code editor with the text `Sub MyFirstSubroutine()` entered. An arrow points from the text to instruction (b).

**Step 3:** The third screenshot shows the code editor with the text `Sub MyFirstSubroutine()  
|  
End Sub` entered. An arrow points from the text to instruction (c).

**Instructions:**

- Start by typing the word `sub` followed by a space and the name that you want to give your procedure.
- Once you've typed in a name for the sub, simply press `Enter` on the keyboard.
- 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.

The diagram illustrates an error message in the VB Editor. It consists of two screenshots of the code editor window and three numbered instructions.

**Step 1:** The first screenshot shows the code editor with the text `sub MyFirst Subroutine` entered. An arrow points from the text to instruction (a).

**Step 2:** The second screenshot shows the code editor with the text `sub MyFirst Subroutine` entered. The word `Subroutine` is highlighted in red. An arrow points from the text to instruction (b).

**Step 3:** The third screenshot shows the code editor with the text `sub MyFirst Subroutine` entered. The word `Subroutine` is highlighted in red. An arrow points from the text to instruction (c).

**Instructions:**

- 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!

The diagram illustrates the instructions to write out the instructions for a subroutine. It consists of one screenshot of the code editor window and one instruction.

**Step 1:** The screenshot shows the code editor with the text `Sub MyFirstSubroutine()  
|  
End Sub` entered. An arrow points from the text to instruction (a).

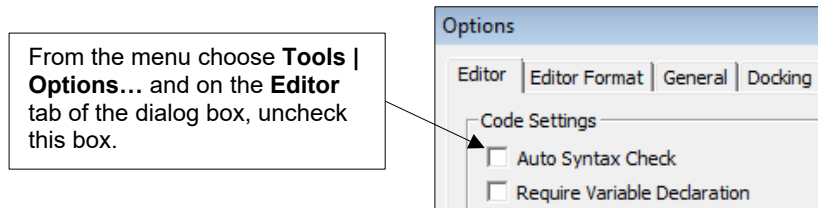
**Instruction:**

- 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.



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.


```
Sub FormatSelectedCells()
    With Selection.Font
        .Name = "Arial"
        .Size = 12
        .Bold = True
        .Italic = False
        .Color = rgbBlack
    End With
End Sub
```

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
```

## 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

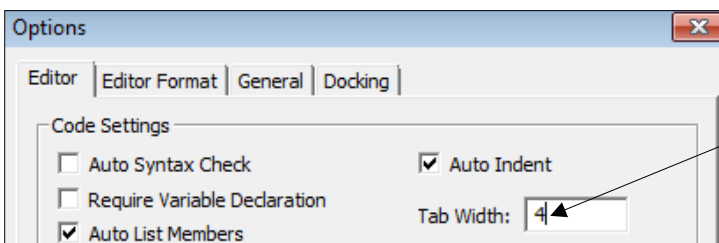
```
With SingleCell
.Interior.Color = rgbPink
.Font.Color = rgbRed
.Font.Italic = True
.Font.Bold = True
End With
```

Select at least a part of each line that you want to indent and then press **Tab** to indent them. You can outdent the selected lines by pressing **Shift** and **Tab**.

```
With SingleCell
. Interior.Color = rgbPink
. Font.Color = rgbRed
. Font.Italic = True
. Font.Bold = True
End With
```

## Changing Indenting Settings

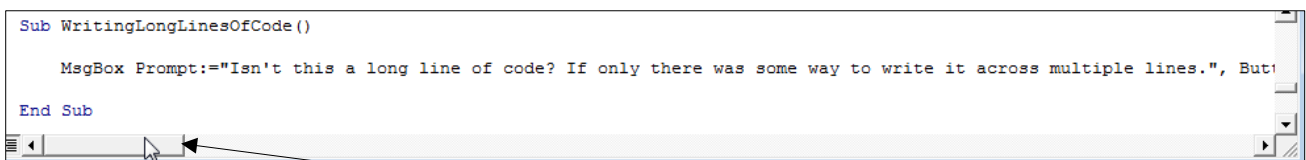
The default width of a tab space in the VB Editor is equivalent to four spaces. You can change this setting by choosing **Tools | Options...** from the menu.



On the **Editor** tab of the dialog box you can type a number into this box to change the width of a tab space in the VB Editor.

## 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.



When your code extends past the width of a single screen you can use the scroll bar to move left and right to see it all.

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.

```
Sub WritingLongLinesOfCode ()
    MsgBox _
        Prompt:="Isn't this a long line of code?", _
        Buttons:=vbYesNo + vbQuestion, _
        Title:="A Long Message"
End Sub
```

To begin a new line in the middle of a single instruction you must type in a space followed by an underscore before pressing **Enter**.

You can't have blank lines between the lines which make up the complete instruction.

## 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 **T**ools | **O**ptions...

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.

```
workbooks("Book1.xlsm").
```

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.

The diagram illustrates how VBA tooltips work. It shows two examples:

- Example 1:** The keyword `range` is typed, followed by an open parenthesis `(` and a space. A tooltip appears showing the full signature: `Range(Cell1, [Cell2]) As Range`. A text box explains: "Tooltips will appear after you type in a keyword followed either by an open parenthesis or a space."
- Example 2:** The keyword `msgbox` is typed, followed by an open parenthesis `(`. A tooltip appears showing the full signature: `MsgBox(Prompt, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult`. The `Prompt` parameter is highlighted in bold. A text box explains: "The tooltip shows the parameter list for the particular keyword you have typed in. You can see the currently active parameter highlighted in bold text." Another text box explains: "Optional parameters are displayed enclosed in a set of square brackets, while compulsory parameters aren't."

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.








































The diagram shows how to redisplay a tooltip. It shows the keyword `MsgBox` typed with a cursor on the same line. A text box explains: "With the text cursor positioned on the same line, press **Ctrl** + **I** to display the tooltip for the command you're writing." Below this, the tooltip is shown redisplayed, showing the full signature: `MsgBox(Prompt, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult`.

## 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.

The diagram shows a VBA procedure being stepped through. A yellow arrow points to the `Range` keyword in the line `Range("A1").Select`. A text box explains: "The yellow arrow indicates that you're stepping through a procedure – more on this later." Another text box explains: "Hover the mouse cursor over a keyword to see a data tip appear with more information." The data tip for `Range` is shown as a yellow box containing `Range("A1").Select`. The data tip for `ActiveCell.Value` is shown as a yellow box containing `ActiveCell.Value = "Wise Owl"`.

## What we do!

		Basic training	Advanced training	Systems / consultancy
<b>Office</b>	Microsoft Excel			
	VBA macros			
	Office Scripts			
	Microsoft Access			
<b>Business Intelligence</b>	Power BI			
	Power Apps			
	Power Automate / PAD			
<b>SQL Server</b>	SQL			
	Reporting Services			
	Report Builder			
	Integration Services			
	Analysis Services			
<b>Coding</b>	Visual C# programming			
	VB programming			
	DAX			
	Python			



**WiseOwl**  
Training



