

## CHAPTER 1

## InFocus

# UNDERSTANDING EXCEL VBA

**Visual Basic**, or **VB** for short, is a programming language that was developed many years ago by Microsoft to help people program their computers. Microsoft Office has a number of applications that each have a derivative version of **VB** which has become known as **Visual Basic for Applications**, or **VBA** for short.

VBA is an object-oriented programming language that performs operations by manipulating **objects**, which in turn have **methods**, **properties** and **events**. Each application in the Microsoft Office suite, including Microsoft Excel, has its own specific set of objects.

**In this session you will:**

- ✓ gain an understanding of programming in **Microsoft Excel**
- ✓ gain an understanding of **VBA** terminology
- ✓ learn how to display the **DEVELOPER** tab
- ✓ gain an understanding of the **VBA Editor** screen and its features
- ✓ learn how to open, close and switch to the **VBA Editor**
- ✓ gain an understanding of objects used in **VBA**
- ✓ learn how to access the **Excel** object model
- ✓ learn how to use the **Immediate Window**
- ✓ learn how to work with object collections
- ✓ learn how to work with property values
- ✓ learn how to work with worksheets in **VBA**
- ✓ learn how to use the **Object Browser**
- ✓ learn how to program with the **Object Browser**
- ✓ gain an understanding of how to access good and reliable help for **VBA**
- ✓ gain an understanding of the codes used in this chapter.

# PROGRAMMING IN MICROSOFT EXCEL

Microsoft Excel has had some form of programming language available to it ever since it was first released. Its current programming language, **VBA**, is a very powerful and versatile

tool. It is not overly difficult to learn and use, but it sometimes can seem complicated until you really fully grasp its background and what it is primarily designed to do.

## The Difference Between Macros And VBA

There is really no difference between macros and VBA. When spreadsheets were introduced to computing it was possible to program them to perform repetitive procedures. These programs were simply lists of commands from the standard menu that you wanted to run in a particular sequence. This early form of a program was known as a **macro**.

The term **macro** is still used today – indeed you'll see it on the **DEVELOPER** tab of the ribbon in Microsoft Excel 2013. However, these macros are now based on a full-blown programming language (VBA) rather than a list of simple commands. So the two terms, *macro* and *VBA*, are interchangeable.

## Why Use VBA?

The sole justification behind putting a programming language behind an application like Microsoft Excel is to extend its capabilities beyond those that you can find on the ribbon. There are several reasons people decide to program in VBA:

- **Repetition** – The primary use of VBA is to automate operations (especially those performed in a repetitive manner). For example, if you need to download accounting data, extract the sales information and create a chart on a weekly basis, you can write code in VBA to automate these tasks and ensure they are done quickly and accurately.
- **Limitation and Interaction** – Imagine developing the world's best workbook full of facts and formulas that plots the economies of several dozen developing nations around the world. Then imagine handing that workbook to one of your colleagues only to find they have accidentally deleted the area that contained your formulas! With carefully written VBA coding you can guide inexperienced users through your workbook, giving them access to specific areas and precluding them from others, providing them with specialised dialog boxes (known in VBA as **forms**) and prompting them for key information.
- **Cross-application Communication** – You can use VBA to write programs that communicate across applications. For example, using VBA you can pull Word data and charts into a PowerPoint presentation, or insert sales figures from Excel into a Word document.

## Time versus Effort

VBA isn't always the best way to go about a task. If you only need to perform an operation (even a complex one) once, then it would be a waste of time to write a program to do it – no matter how clever the program makes you look!

Also, there are many actions that can quickly and easily be performed using a command or commands that already exist on the ribbon. Before you sit down to write a VBA program, check to make sure that the task can't already be done in other ways. This is why it is important to have a good grasp of Microsoft Excel before you attempt VBA.

# VBA TERMINOLOGY

**Visual Basic for Applications** is a derivative of the programming language **Visual Basic**. Each Office application has its own particular kind of VBA depending on the objects and operation of

the application. For example, Microsoft Excel uses worksheets while Microsoft Word works with documents. The following notes explain some of the key concepts in VBA programming.

## Object Oriented and Procedure Driven

**Visual Basic for Applications** and **Visual Basic** are both **object-oriented** programming languages because they work with **objects**. Most of these objects appear on the screen, hence the term 'visual'.

They are also **procedure-driven** languages using commands and structures from the **BASIC** programming language to bind object statements into workable applications.

## Objects, Properties, Methods and Events

In VBA an **object** is anything in an application that you can see and manipulate in some way.

For example, you can manipulate a worksheet by adding rows, deleting columns, displaying gridlines, and so on. A worksheet is therefore an example of an object. Rows are also objects, as are columns too. These are **child** objects of the **parent** object – the worksheet. This way of organising objects into a hierarchy is known as an **object model**.

Objects can be manipulated in one of three ways. You can:

- change the way an object looks or behaves by changing its **properties**
- make an object perform a task by using a **method** that is associated with the object
- run a procedure whenever a particular **event** happens to an object.

Objects therefore have **properties**, **methods** and **events**.

### A real-world example...

Let's look at a simple real-world analogy to get a better idea about **objects**, **properties**, **methods** and **events**. Consider a car: it is an **object** because you can see it and manipulate it. Its:

- **properties** are its physical characteristics such as its make, model, colour, and so on
- **methods** define what you can do with the car such as reversing, accelerating, turning, stopping, and so on
- **events** are the actions that happen to the car that generate an automatic response from the car. For example, if you remove the keys from the ignition while the car's headlights are on (event), most cars will sound a warning alarm or turn off the lights (response).

## The Active Object

In VBA, **active** describes the object item that you're currently working on.

For example, the worksheet cell that you're editing or formatting in Excel is the **active** cell. The workbook that you are currently working on is said to be the **active** workbook. The object that is currently active is said to have the **focus**.

This is an important concept to understand because most of your VBA programming will be performing an action on a particular object. If you don't identify that object correctly, you may find that Excel shifts focus behind the scenes to a different object and your program will fail.

# DISPLAYING THE DEVELOPER TAB

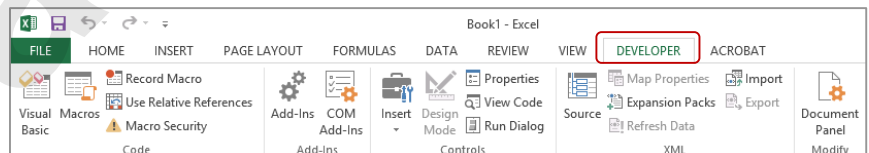
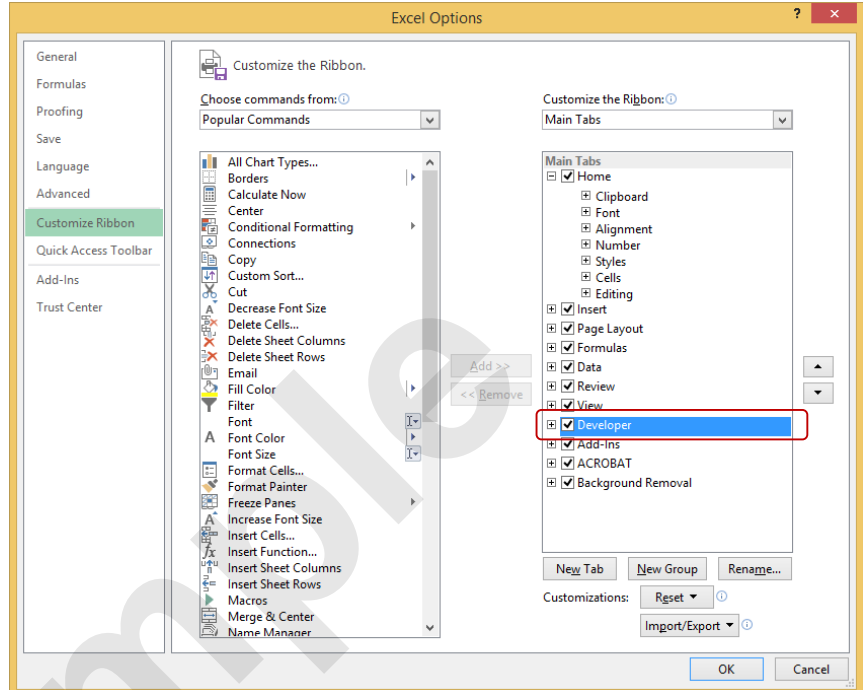
Before you start working with VBA, you will need to ensure that the **DEVELOPER** tab is displayed on the ribbon in Excel. This tab allows you to access the commands to display the **VBA Editor**

and run macros, among others. The **DEVELOPER** tab is not displayed by default; you must enable it using the **Excel Options** dialog box.

## Try This Yourself:

*Before starting this exercise ensure Excel has started and a new, blank workbook is displayed...*

- 1 Click on the **FILE** tab to display the **Backstage**
  - 2 Click on **Options** to display the **Excel Options** dialog box
  - 3 Click on **Customise Ribbon** in the left pane to display the options for customising the ribbon
  - 4 In the right pane, click on **Developer** so it appears ticked
  - 5 Click on **[OK]** to save the settings and return to Excel
- The DEVELOPER tab is now displayed on the ribbon...*
- 6 Click on the **DEVELOPER** tab to see the available commands



## For Your Reference...

To **display** the **DEVELOPER** tab on the **ribbon**:

1. Click on the **FILE** tab
2. Click on **Options**
3. Click on **Customise Ribbon**
4. Click on **Developer** in the right pane so it appears ticked, then click on **[OK]**

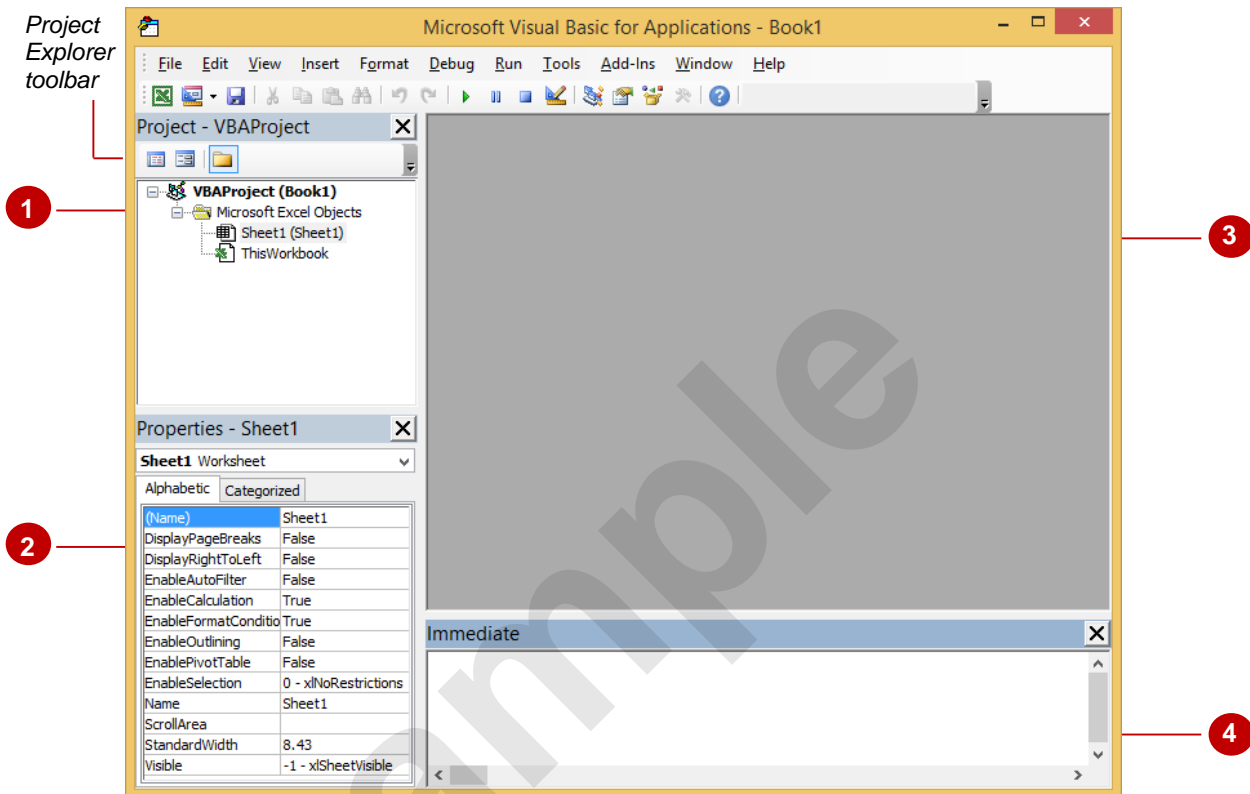
## Handy to Know...

- Once the **DEVELOPER** tab is displayed, it will remain on the ribbon until you choose to remove it again. You can do this by displaying the **Customise Ribbon** screen of the **Excel Options** dialog box and clicking on **Developer** so it appears unticked.

# THE VBA EDITOR SCREEN

The **VBA Editor** is a separate application designed to help you create and edit VBA procedures. The **VBA Editor** comprises four main components: the **Project Explorer**, the

**Properties Window**, the **work area** in which you create VBA code in **code modules** and build **UserForms**, and an area for special **panes** that can be displayed.



- 1 Project Explorer** The **Project Explorer** displays the contents of the current *VBA project* which, in the example above, is called **VBAProject**. (In general terms, a **project** is an Office file and all of its associated VBA items, including its macros and user forms). Contents of a project can include **forms**, **objects** such as worksheets, and **modules**.

Using the tools in the **Project Explorer's** toolbar you can display the code window (**View Code**) so you can write and edit code associated with the selected item; display the object window (**View Object**) for the selected item, an existing workbook or user form; and hide or show the object folders while still showing the individual items contained within them (**Toggle Folders**).
- 2 Properties Window** The **Properties Window** lists the properties for the object that is currently selected in the **Project Explorer** or in the work area (in the case of a **form**) and displays its current settings.
- 3 Work Area** The **work area** is the larger area to the right of the **Project Explorer** and **Properties Window**. It is in this area where the text editor is used to create VBA code in a **module** (a window that is designed to hold programming code) and where user forms are built. (A **UserForm** is a window or dialog box that makes up part of an application's user interface.)
- 4 Pane Area** The **Pane Area** is used to display additional mini-windows or panes that can assist in the development and debugging of programs. The example above shows the **Immediate** pane.

# OPENING AND CLOSING THE EDITOR

The **Visual Basic Editor** is a separate application that can be accessed from any Microsoft Office application. When you are programming you can keep the Editor open and

switch between the Editor and the workbook as required. Alternatively, you can close the Editor and return to the workbook, then re-open the Editor when you need it.

## Try This Yourself:

Open File

Before starting this exercise you **MUST** open the file **VBE1301 Understanding Excel VBA\_1.xlsm...**

- 1 If the macro security warning appears while opening the file click on **[Enable Content]**

Your workbook should now be open...

- 2 Click on the **DEVELOPER** tab, then click on **Visual Basic** in the **Code** group to open the **VBA Editor**

The Editor will open in its own window...

- 3 Select **File > Close and Return to Microsoft Excel**

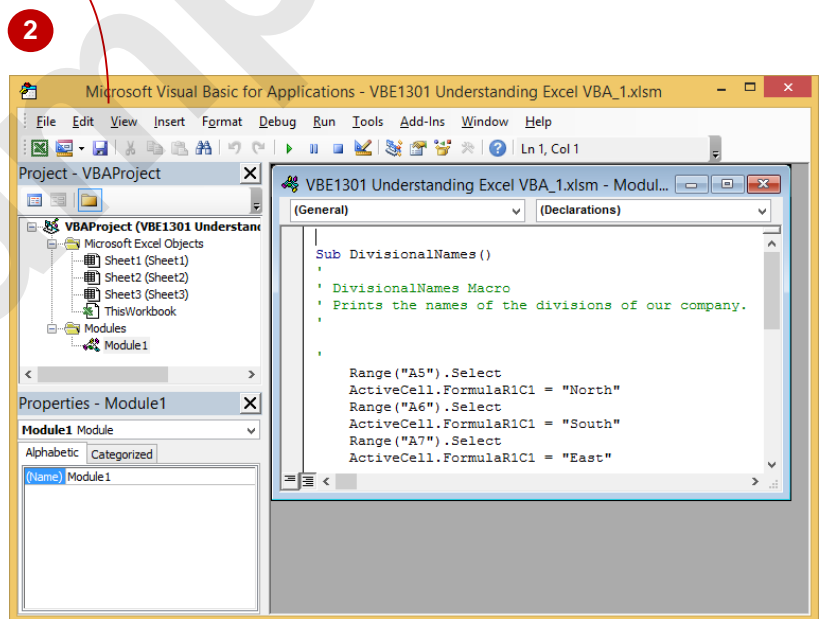
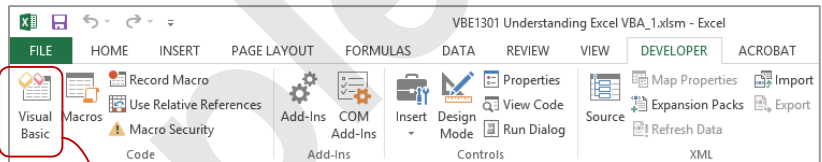
The Editor will close and you will return to the Excel workbook...

- 4 Press **[Alt] + [F11]** to reopen the Editor window

- 5 Press **[Alt] + [Q]** to close the Editor window

1

A1								
	A	B	C	D	E	F	G	H
1								
2								
3	Divisions	Year 1	Year 2	Year 3				
4								
5	North	250	288					
6	South	360	414					
7	East	254	292					
8	West	156	179					
9								
10								
11								



## For Your Reference...

To **open** the **VBA Editor**:

- On the **DEVELOPER** tab, click on **Visual Basic** in the **Code** group, or
- Press **[Alt] + [F11]**

To **close** the **VBA Editor**:

- Select **File > Close and Return to Microsoft Excel**

## Handy to Know...

- **[Alt] + [F11]** acts as a toggle between the Editor and the workbook. When you use **[Alt] + [F11]** to swap back to the workbook, it keeps the Editor open rather than closing it.

# UNDERSTANDING OBJECTS

In VBA, an **object** is anything in an application that you can manipulate in some way. For example, your code may open a workbook, rename a worksheet, select a cell or range,

maximise a window, set a specific application option, and so on. Each of these items – workbook, worksheet, cell, range, window and application – is an **object**.

## Object Collections

As you know, you can have several Excel workbooks open at any time. Each of these workbooks is a separate **object** that belongs to a **collection**. A **collection** is simply a set of similar objects. For example, Excel's **Workbooks** collection is the set of all open **Workbook** objects. Because collections are objects themselves, they have their own properties and methods that you can use to manipulate one or more objects in the collection.

The members of a collection are called **elements**. You can refer to an individual element by the object's name or its index value (the **index** is the number that Excel lists beside the filename in **Switch Windows** in the **VIEW** tab). For example, you can programmatically close a workbook named **Monthly Sales.xlsx** using the following two commands (assuming that **Monthly Sales.xlsx** is the first workbook opened in the current Excel session):

```
Workbooks("Monthly Sales.xlsx").Close Or  
Workbooks(1).Close
```

Notice how the object (**Workbooks**) and the method (**Close**) are delimited by a full stop (.).

If you don't specify an element – such as ("**Monthly Sales.xlsx**") or (**1**) – VBA assumes you are working with the entire collection.

## Object Properties

Every object has a defining set of characteristics. These characteristics are called the object's **properties** and they control the appearance and position of the object. For example, the **ActiveCell** object has a **Value** property that you can use to *get* or *set* the contents of the cell.

When you refer to a property you use the syntax **Object.Property**. For example:

```
ActiveCell.Value
```

**Properties** appear in a listing with a property icon .

## Object Methods

An object's **method** describes what you can do with the object. For example, you can **save** (method) the **active workbook** (object).


When you refer to a method you use the syntax **Object.Method**. For example:

```
ActiveWorkbook.Save
```

Some methods have **arguments**. In this case you use the syntax **Object.Method argument1, argument2, ...**

```
Workbooks.Open "D:\Sales Data\Monthly Sales.xlsx"
```

The above statement will open the workbook called **Monthly Sales.xlsx**. This statement has included only one of the available arguments – the *filename* and path. Sometimes arguments are compulsory. For example, VBA needs to know what file to open so failing to provide a filename here will crash the program. Sometimes arguments are optional such as whether to open the file as *Read Only*.

The names of the valid arguments, plus the **constants** that can be used with them, are listed in the **Help** system for each particular method. **Methods** appear in a listing with the method icon .

## Object Events

An **event** is something that happens to an object. The opening of a workbook in Excel is an example of an event. Although Excel has an **Open** method that you can use to open a workbook, this **method** only initiates the procedure; the actual process of the file being opened is the **event**.

For example, you may write a procedure (which is called an **event handler**) that will display a message box each time a specific workbook is opened.

# VIEWING THE EXCEL OBJECT MODEL

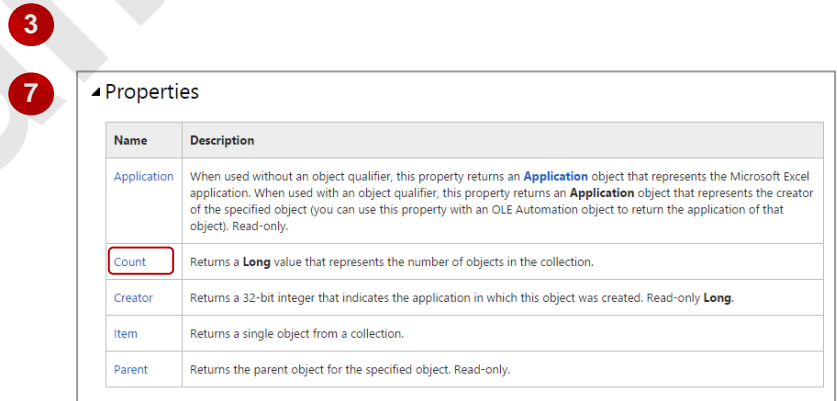
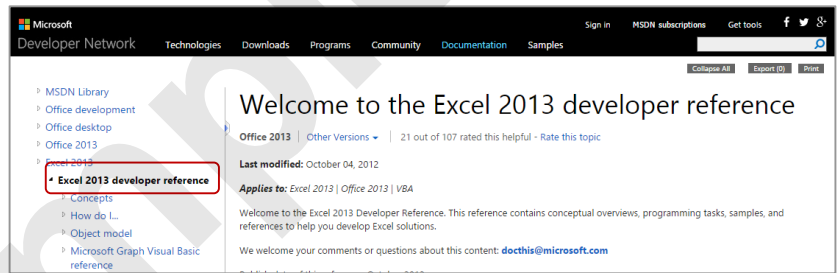
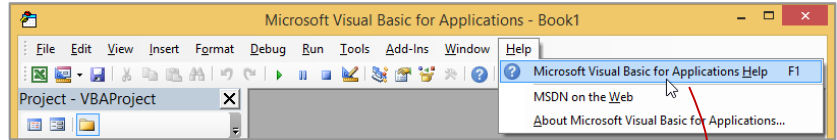
The **object model** for Excel is a very extensive hierarchy of **objects** and **collections**. In previous versions of Excel you could readily find a pictorial representation of the entire model; however, in

Excel 2013 you can only view pictorial representations of the individual objects of the model. Nevertheless, this is still a great way of finding the methods and properties of an object.

## Try This Yourself:

*Before starting this exercise ensure that your computer is connected to the internet...*

- 1 Open a new blank workbook, then open the Editor
- 2 In the Editor, select **Help > Microsoft Visual Basic for Applications Help** to open a new browser window
- 3 Click on **Excel 2013 developer reference** in the left pane
- 4 Click on **Object model** in the left pane to display the list of objects in the model
- 5 Scroll to and click on **Workbooks Object**
- 6 Click on **Workbooks Members**, then examine the list of methods and properties shown to the right
- 7 Scroll down to **Properties**, then click on **Count** to read about the **Count** property
- 8 Close the browser tab



## For Your Reference...

To **access** the **Excel object model**:

1. Select **Help > Microsoft Visual Basic for Applications Help**
2. Click on **Excel 2013 developer reference**
3. Click on **Object model**

## Handy to Know...

- It is more important to know how to find the object model than it is to memorise the entire structure. Understanding how these objects, methods and properties work together will come with time and experience.

# USING THE IMMEDIATE WINDOW

The Editor has an **Immediate Window** that lets you type instructions and test expressions. It runs statements immediately as if they were run from a procedure. You can also precede variables and

expressions with a **question mark** to perform **what is** operations. For example, **?Workbooks.Count** asks "How many workbooks are open?".

## Try This Yourself:

*Before starting this exercise ensure that a new, blank workbook is open and that the VBA Editor is displayed...*

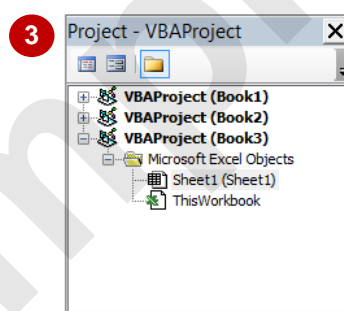
- 1 Select **View > Immediate Window** to display the **Immediate Window** as a pane at the bottom of the Editor
- 2 Type **Workbooks.Add**, then press **Enter** to create a new workbook
- 3 Click on **Workbooks.Add**, then press **Enter** to create another workbook  
*You can see the workbooks that are open in the Project Explorer...*
- 4 Repeat step 3 three more times to build up a **collection** of open workbooks
- 5 Click under the last command, type **Workbooks.Open "C:\Course Files for Microsoft Excel 2013 VBA\Test\_1.xlsx"**, then press **Enter** to open the workbook called **Test\_1.xlsx**
- 6 Repeat step 5 for **Test\_2.xlsx**
- 7 Type **?Workbooks.Count** and press **Enter** to display the number of workbooks that are open  
*Leave the workbooks and the VBA Editor open for the next exercise*



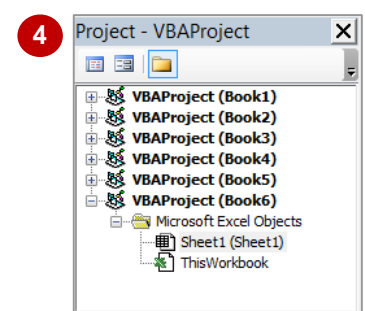
1



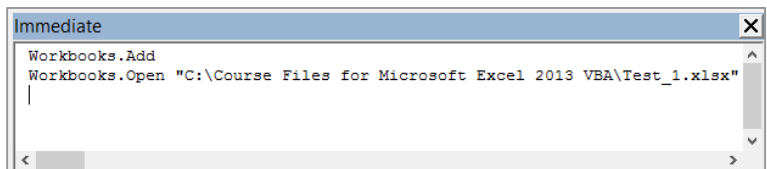
2



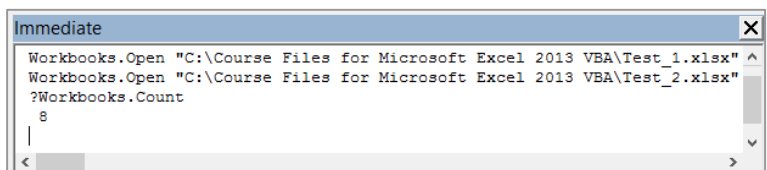
3



4



5



7

## For Your Reference...

To **display** the **Immediate Window**:

- Click on **View**, then select **Immediate Window**

To **use** the **Immediate Window**:

1. Type a command in the window
2. Press **Enter**

## Handy to Know...

- Computer programming requires a much higher degree of accuracy than you may be used to. For example, if you have typed the file and file path details incorrectly or if the files are in a different location on your computer, the programming instructions you type will result in an error message.

# WORKING WITH OBJECT COLLECTIONS

In VBA some objects belong to **collections**. The **Workbooks** collection is the set of all the **workbook** objects currently open in the **application**. Each item in a collection is known

as an **element** and must be referenced as part of the collection, either by name or by its position in the collection (known as the **index**), where **1** is the first workbook you opened, **2** is the second, etc.

## Try This Yourself:

*Continue using the previous files with this exercise...*

- 1 Ensure the cursor is positioned on a blank line in the **Immediate Window**, type **?Workbooks.Item(1).Name**, then press **Enter** to return the name of the first workbook you added
- 2 Type **Workbooks.Item(1).Activate**, then press **Enter** to make this the active workbook
- 3 Type **Workbooks.Item("Test\_1.xlsx").Activate**, then press **Enter** to make this workbook active
- 4 Type **?Workbooks.Item("Test\_1.xlsx").Saved**, then press **Enter** to see whether the file has been saved – **True** – or **False** if it hasn't been saved
- 5 Type **Workbooks.Item("Test\_1.xlsx").Close**, then press **Enter** to close the workbook – if the file hasn't been saved you will be prompted to do so
- 6 If prompted, click on **[Don't Save]**
- 7 Type **Workbooks.Close**, then press **Enter** to close all workbooks – don't save if prompted  
*Leave the Editor open for the next exercise*

```

Immediate
?Workbooks.Count
8
?Workbooks.Item(1).Name
Book1
  
```

1

```

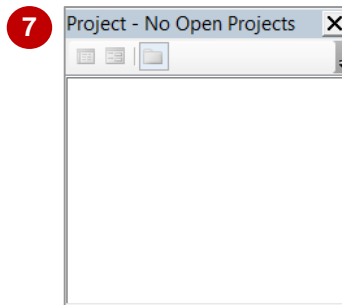
Immediate
Workbooks.Item(1).Activate
Workbooks.Item("Test_1.xlsx").Activate
?Workbooks.Item("Test_1.xlsx").Saved
True
  
```

4

```

Immediate
Workbooks.Item("Test_1.xlsx").Activate
?Workbooks.Item("Test_1.xlsx").Saved
True
Workbooks.Item("Test_1.xlsx").Close
  
```

5



7

*When all of the workbooks are closed, Project Explorer is empty*

## For Your Reference...

To **work** with **collections** use the structures:  
**CollectionName.Item(index or name).Method**  
**CollectionName.Item(index or name).Property = value**

## Handy to Know...

- You can find out information about the currently active workbook by using the object **ActiveWorkbook**. For example, **?ActiveWorkbook.Name** will display the name of the active workbook.

# SETTING PROPERTY VALUES

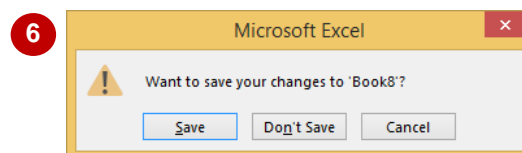
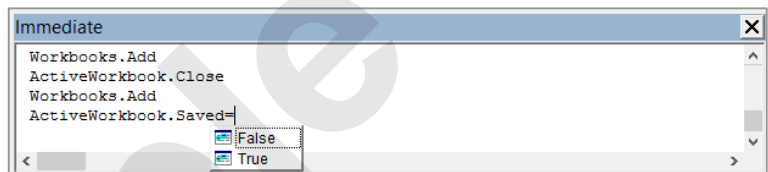
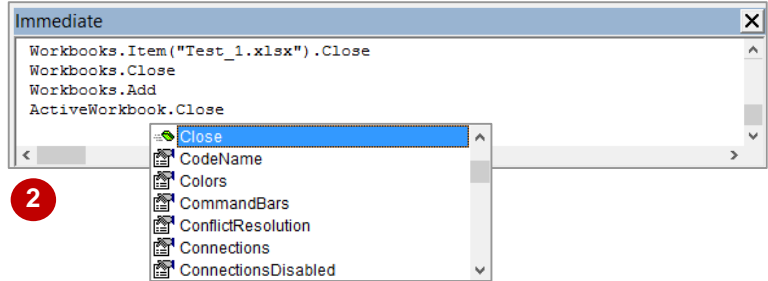
**Properties** affect the way an object looks or behaves. The properties for objects can be listed by typing the name of the **object** followed by a full stop. VBA includes a built-in **list** system

(known as **IntelliSense**) that will list methods, properties and values as you type your command. **Property values** are set by specifying the **object**, the **property**, an **equal sign** and the new **value**.

## Try This Yourself:

*Before starting this exercise ensure your cursor is positioned in the Immediate Window in the VBA Editor...*

- 1 Type **Workbooks.Add**, then press  to add a workbook (and make it active)
- 2 Type **ActiveWorkbook.Close**, then press   
*Notice how a list of methods and properties appears when you press the full stop or get to the end of a method or property...*
- 3 Type **Workbooks.Add**, then press
- 4 Type **ActiveWorkbook.Saved =** *IntelliSense will list the possible values for the Saved property. Let's use the logical value False to force Excel to display a prompt to save the currently unsaved workbook when it is closed...*
- 5 Type **False**, then press
- 6 Type **ActiveWorkbook.Close**, then press   
*You will be prompted to save the workbook...*
- 7 Click on **[Don't Save]**  
*Leave the Editor open for the next exercise*



## For Your Reference...

To **set** a **property value** use the structure:  
`Object.Property = Value`

## Handy to Know...

- It can take some time to get used to working with properties. Some properties can only be read, while others can be changed. When you change a property you are said to be **setting** its value. When you read a property you are said to be **getting** its value.

# WORKING WITH WORKSHEETS

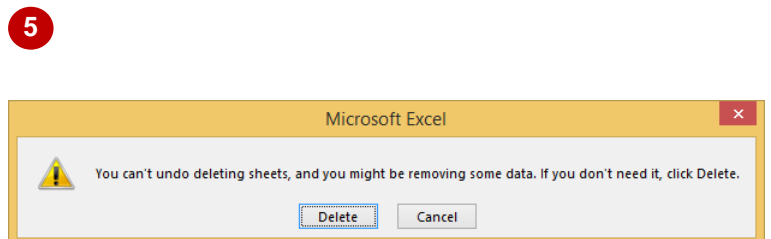
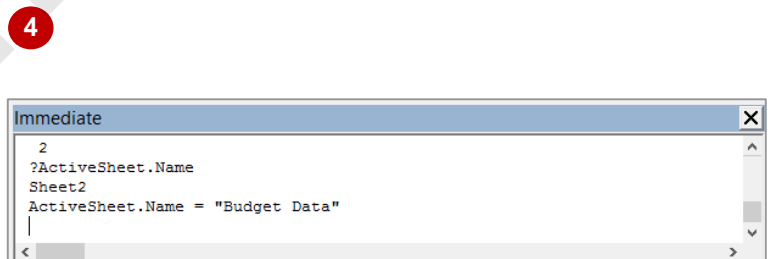
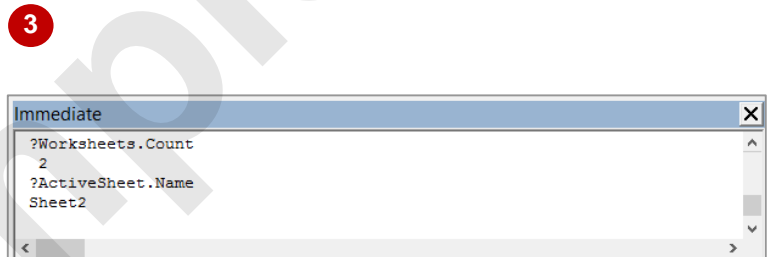
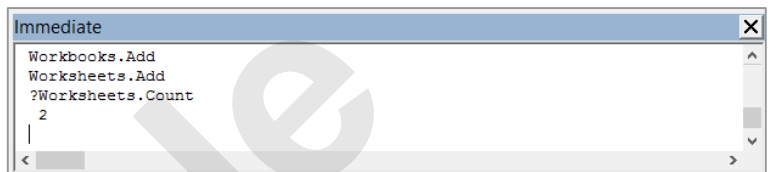
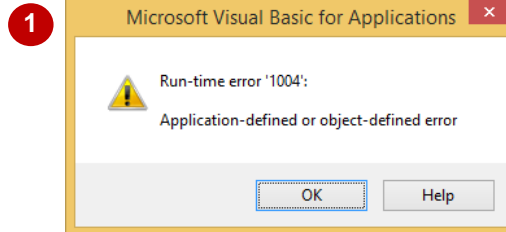
**Worksheets** is a **collection** of worksheet objects. It refers to all of the worksheets in the active workbook. You can insert, modify and delete worksheets using properties and methods,

but you can only manipulate worksheets if you have a workbook open. The following example demonstrates some of the techniques that you can use with the **Worksheets** collection.

## Try This Yourself:

*Before starting this exercise ensure all workbooks are closed and your cursor is positioned in the Immediate Window in the VBA Editor...*

- 1 Type **Worksheets.Add**, then press   
*An error message will appear – there is no workbook open in which to insert a worksheet...*
- 2 Click on **[OK]**
- 3 Type the following commands, pressing  after each:  
**Workbooks.Add**  
**Worksheets.Add**  
**?Worksheets.Count**  
*The number of worksheets in the workbook will be returned...*
- 4 Type **?ActiveSheet.Name**, then press  to display the name of the active worksheet
- 5 Type **ActiveSheet.Name = "Budget Data"**, then press  to name the active worksheet
- 6 Type **Worksheets.Item(2).Delete**, then press   
*A warning will appear notifying you that you may lose data...*
- 7 Click on **[Delete]**  
*Leave the workbook and the VBA Editor open for the next exercise*



## For Your Reference...

To **work** with **Worksheets** use the structure:  
**Worksheets.Item(Index or "Name").method**  
**Worksheets.Item(Index or "Name").property = value**

## Handy to Know...

- Cryptic error messages are common occurrences in VBA, both for beginners and for experienced programmers.