Contents

1 **Getting started** 7

Introducing Excel VBA 8
Recording a macro 10
Viewing macro code 12
Testing a macro 14
Editing macro code 15
Referencing relatives 16
Saving macros 18
Trusting macros 20
Summary 22

2 **Writing macros** 23

Exploring the Editor 24
Creating a macro 26
Adding toolbar buttons 28
Adding Form Controls 30
Recognizing hierarchy 32
Identifying a range 34
Addressing cells 36
Summary 38

3 **Storing values** 39

Creating variables 40
Defining data types 42
Managing strings 44
Producing arrays 46
Describing dimensions 48
Representing objects 50
Declaring constants 52
Summary 54

4 Performing operations 55

Doing arithmetic 56
Making comparisons 58
Assessing logic 60
Joining strings 62
Understanding precedence 64
Summary 66

5 Making statements 67

Choosing a branch 68
Branching alternatives 70
Selecting branches 72
Performing loops 74
Looping while true 76
Breaking from loops 78
Iterating for each 80
Including with 82
Summary 84

6 Executing procedures 85

Calling subroutines 86
Modifying scope 88
Passing arguments 90
Adding modules 92
Fixing values 94
Debugging code 96
Handling errors 98
Summary 100

7 Employing functions 101

Defining a function 102
Calling a function 104
Scoping a function 106
Passing array arguments 108
Stating options 110
Returning errors 112
Debugging functions 114
Describing functions 116
Summary 118

8 Recognizing events 119

Creating event-handlers 120
Opening workbook events 122
Changing workbook events 124
Closing workbook events 126
Spotting worksheet changes 128
Catching worksheet clicks 130
Listening for keystrokes 132
Observing the time 134
Summary 136

9 Opening dialogs 137

Acquiring input 138
Showing messages 140
Importing files 142
Saving files 144
Producing data forms 146
Executing Ribbon commands 148
Summary 150

10 Providing UserForm dialogs 151

Inserting a UserForm 152
Adding controls 154
Comparing control types 156
Adjusting properties 158
Naming controls 160
Displaying forms 162
Handling form events 164
Managing lists 166
Summary 168

11 Developing apps 169

Ignoring modes 170
Indicating progress 172
Controlling MultiPages 174
Tabbing data pages 176
Showing chart info 178
Creating Add-ins 180
Installing Add-ins 182
Adding Ribbon buttons 184
Summary 186

Index 187

1 Getting started

Welcome to the exciting world of Excel VBA (Visual Basic for Applications). This chapter demonstrates how to create a VBA macro for Excel workbooks.

8 Introducing Excel VBA
10 Recording a macro
12 Viewing macro code
14 Testing a macro
15 Editing macro code
16 Referencing relatives
18 Saving macros
20 Trusting macros
22 Summary

Introducing Excel VBA

Visual Basic for Applications (VBA) is the programming language that is built into the Excel spreadsheet application and other Microsoft Office applications. It extends Excel so it can perform tasks that can't be done with standard Excel tools, and provides the capability to automate many routine tasks.

The examples in this book assume the reader is an experienced Excel user who can accomplish these fundamental operations:

- Create workbooks and insert worksheets
- Navigate around a workbook and worksheet
- Use the Excel Ribbon interface
- Name cells and ranges
- Use the Excel worksheet functions

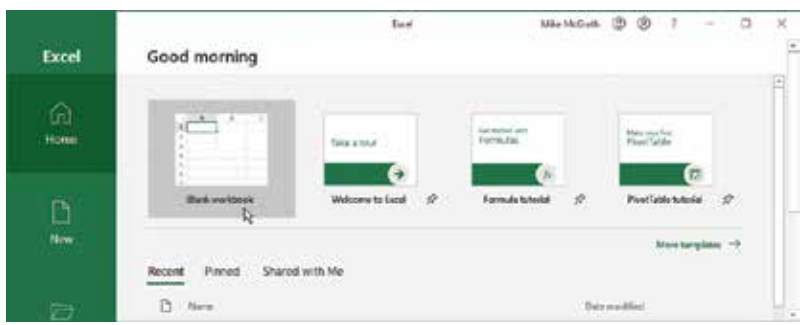
All examples are demonstrated using Excel 2019, although most examples are also applicable to earlier versions of Excel.

If you're just starting out with Excel, please refer to our companion book Excel 2019 in easy steps.

Enabling VBA

Before you can get started using the capabilities of VBA, it must first be enabled in your Excel installation:

1. Launch Excel, then choose to open a **Blank workbook**

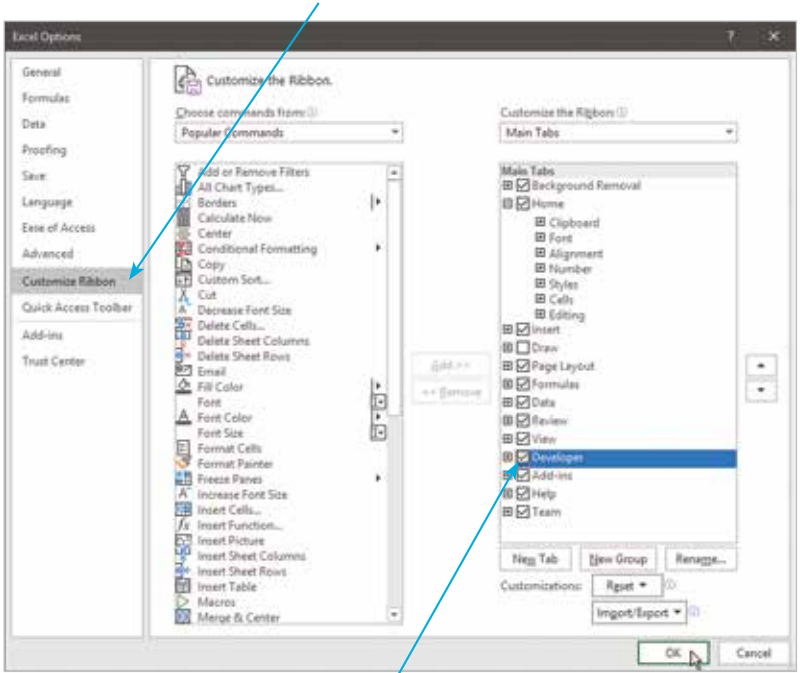


2. When the workbook opens, choose the **File** item on the Excel Ribbon

All the examples in this book are available for free download at www.ineasysteps.com/resource-centre/downloads

3 Next, select the **Options** item – to open the "Excel Options" dialog box

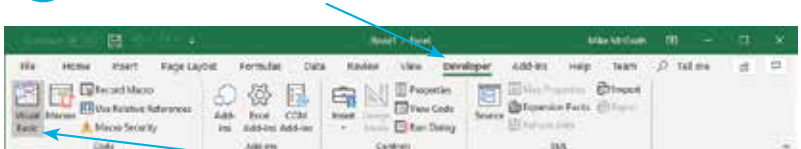
4 In the Excel Options dialog, choose the **Customize Ribbon** item on the left-hand pane



5 Now, check the **Developer** option box in the right-hand pane

6 Click the **OK** button to apply the change and to close the Excel Options dialog box

7 See that a **Developer** tab has been added to the Ribbon



8 Choose the Developer tab to see a **Visual Basic** button in the Ribbon's "Code" group – VBA is now enabled

You can also open the Excel Options dialog box by pressing the **Alt** + **F** + **T** keys.

In the Excel Options dialog you can click the **+** button beside the **Developer** item to reveal the groups it contains. If you right-click on any group, a context menu offers you options to modify the groups that will appear on the Developer tab.

Recording a macro

Having enabled VBA, as described on pages 8-9, you can create a simple app by recording a "macro" to store actions:

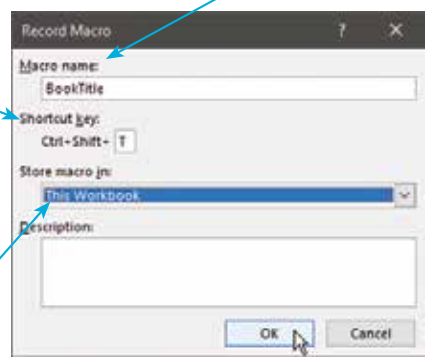
1. Open a blank workbook in Excel, then select worksheet cell **A1**

2. On the Developer tab, click the **Record Macro** button in the Code group to launch the "Record Macro" dialog box

3. Type a name of your choice in the dialog's **Macro name** field – for example, type "BookTitle"

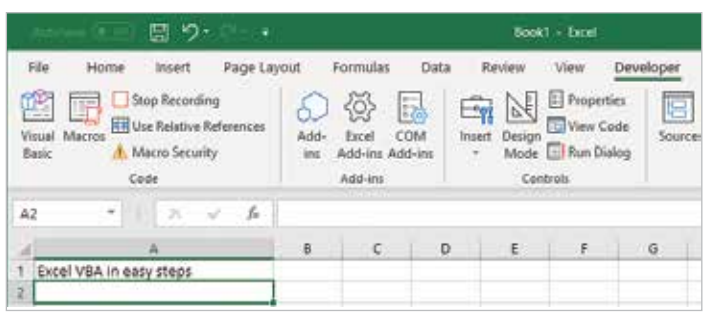
4. Next, type a letter in the dialog's **Shortcut key** field – for example, type "T", to create a **Ctrl** + **Shift** + **T** shortcut

5. Now, choose to store the macro in **This Workbook**



6. Click the **OK** button to close the Record Macro dialog, and to begin recording actions

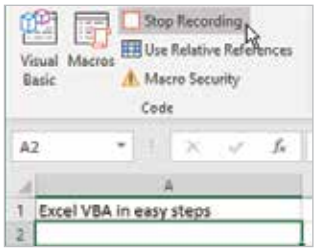
7. Type the title of this book into previously selected cell A1, then hit **Enter** – to enter the title text into the cell



In the Record Macro dialog you can add a **Description** of what the macro will perform.

A macro is a set of programming instructions stored in VBA code.

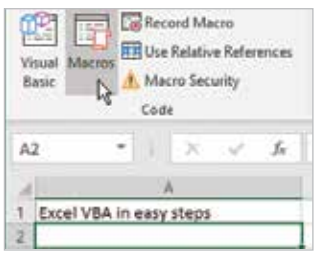
8 Notice that focus has moved, so cell A2 is now automatically selected after you hit the **Enter** key



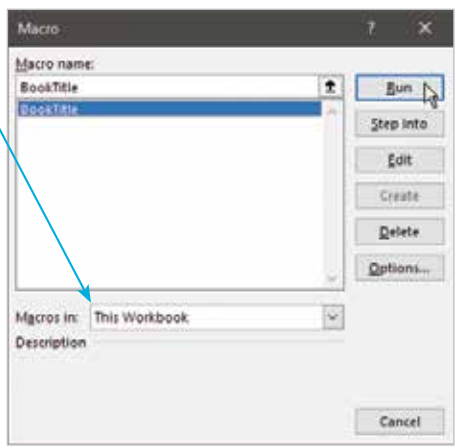
9 Now, click the **Stop Recording** button in the Code group on the Developer tab – to stop recording your actions

The **Record Macro** button changes to **Stop Recording** when recording is taking place.

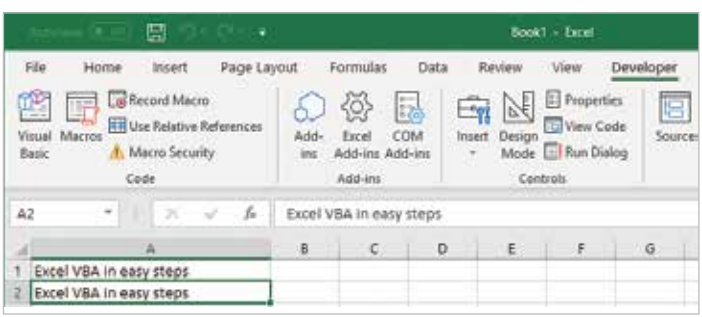
10 Click the **Macros** button in the Code group to launch the "Macro" dialog box and choose to see macros in **This Workbook**



11 Select the "BookTitle" macro, then click the **Run** button to execute the macro and see the book title text appear in the automatically selected cell **A2**



You can also use the shortcut keys **Alt** + **F8** to open the Macro dialog box at any time.

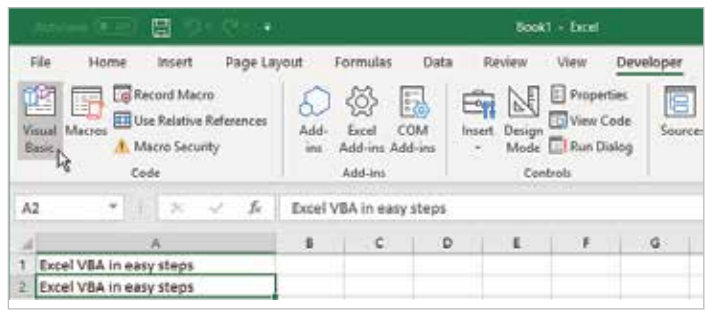


Viewing macro code

Having created a macro, as described on pages 10-11, the VBA programming instructions that were written when the macro was created can be viewed in the Visual Basic Editor:

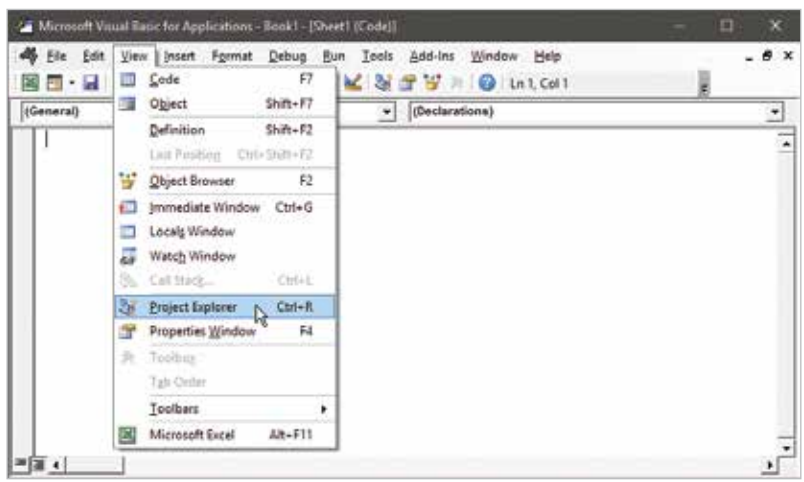
You can also use the shortcut keys **Alt** + **F11** to open the Visual Basic Editor at any time.

1 On Excel's Developer tab, click the **Visual Basic** button in the Code group – to launch the Visual Basic Editor

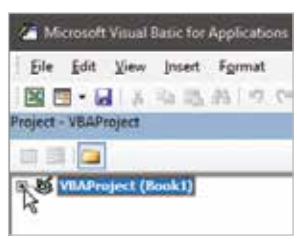


2 In the Visual Basic Editor, select **View, Project Explorer** – to open the "Project Explorer" window

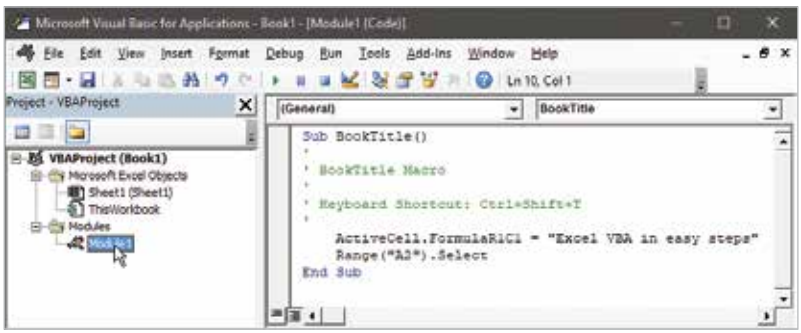
The **Project Explorer** window may already be visible when the Visual Basic Editor opens, and the project may already be expanded, but it is useful to practice opening and closing these items to become familiar with the Visual Basic Editor interface.



3 In Project Explorer, click the + button beside the **Book1** project to expand its contents



4 Now, in Project Explorer, double-click the **Module1** node within the "Modules" folder – to see the macro VBA code



Code analysis

- **Sub BookTitle ( )** – This declares the beginning of a "subroutine" (**Sub**) with the same name you gave to the macro (**BookTitle**) and was written when it began recording.
- **' BookTitle Macro** – This is a comment, confirming that this subroutine is for a macro of your chosen name.
- **' Keyboard Shortcut: Ctrl+Shift+T** – This is another comment, describing the shortcut keys you chose to run this macro.
- **ActiveCell.FormulaR1C1 = "Excel VBA in easy steps"** – This is an instruction that was written when you typed the book title into the cell and hit the **Enter** key.
- **Range("A2").Select** – This is an instruction that was written as focus moved to cell A2 after you hit the **Enter** key.
- **End Sub** – This denotes the end of this macro subroutine, and was written when you stopped recording.

The color used in the code is the default syntax highlighting that the Visual Basic Editor automatically applies for easier reading. Blue is applied to "keywords" that have special meaning in Visual Basic code, and green is applied to comments describing the code. For clarity, the same color syntax highlighting is also used in the example code listed in the steps provided throughout this book.

The **( )** parentheses that appear in the first line of code can contain a parameter list. This is demonstrated later, on page 90.

All lines that begin with an apostrophe are simply ignored when the macro is executed.

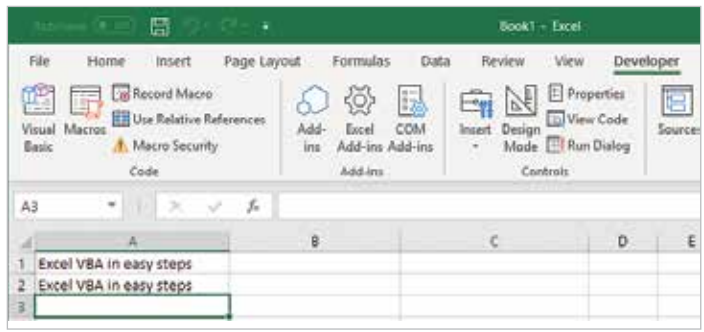
Testing a macro

Before starting to record the macro, as described on page 10, shortcut keys were specified in the Record Macro dialog and these can now be tested to ensure they can run the macro:

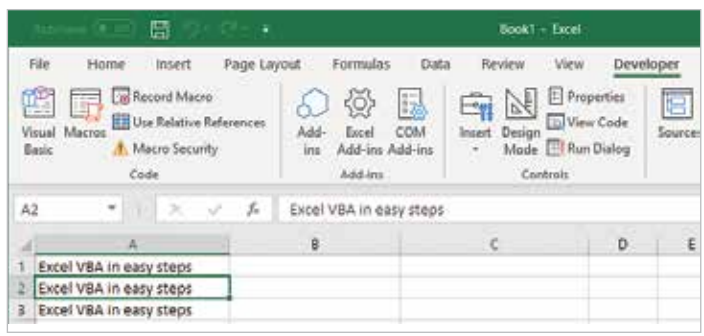
You can use the shortcut keys **Alt** + **F11** to close the Visual Basic Editor.

1. With the Visual Basic Editor open, select **View**, **Microsoft Excel**, or click the button on the toolbar to return to the Excel interface

2. Next, select empty cell **A3**



3. Now, press the **Ctrl** + **Shift** + **T** shortcut keys to test run the macro – the book title should appear in the cell you selected and the focus returned to cell **A2** as instructed in code



If you try to specify a shortcut key that is already designated for use by another macro in the same workbook, a dialog will appear requesting you to specify an alternative shortcut key – so you cannot accidentally duplicate.

It is important to remember that cell A1 was selected before the macro recording began, otherwise the action of selecting that cell would be written as an instruction in the macro. This would mean the book title could only be written into cell A1 each time the macro was run.

Editing macro code

Now you are sure the macro can be run by both the **Run** button in the Macro dialog, and by the **Ctrl** + **Shift** + **T** shortcut keys you specified, but you probably will not need it to return focus to cell A2 after each run. The code can be edited to remove the instruction to return focus, and also to style the text it writes:

1. On Excel's Developer tab, click the **Visual Basic** button in the Code group to launch the Visual Basic Editor

2. In Project Explorer, double-click the project's **Module1** item to see the macro VBA code

3. Next, delete this instruction line that returns focus
   Range("A2").Select

4. Now, add these instructions anywhere within the subroutine to style the text in bold red
   ActiveCell.Font.Bold = True
   ActiveCell.Font.Color = vbRed

5. Click the **Save** button

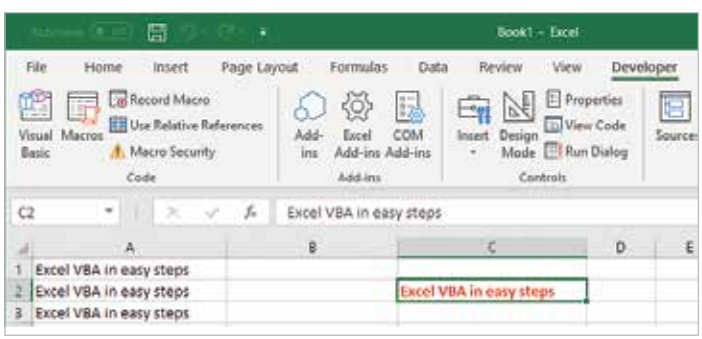
```
(General)                                    BookTitle

Sub BookTitle()
'
' BookTitle Macro
'
' Keyboard Shortcut: Ctrl+Shift+T
'
    ActiveCell.Font.Bold = True
    ActiveCell.Font.Color = vbRed

    ActiveCell.FormulaR1C1 = "Excel VBA in easy steps"

End Sub
```

6. Return to Excel and select any cell – press the **Ctrl** + **Shift** + **T** shortcut keys to run this edited macro



As you type instructions a pop-up box will often appear when you type a period/full stop, to offer a list of suggestions from which you can choose an item.

The eight Visual Basic color constants are **vbRed**, **vbGreen**, **vbBlue**, **vbYellow**, **vbMagenta**, **vbCyan**, **vbBlack**, and **vbWhite** – see pages 52-53 for more on constants.

Although the lines of VBA code are executed from top to bottom, their order is unimportant in this macro – the cell's styling can be set before or after its content is added.

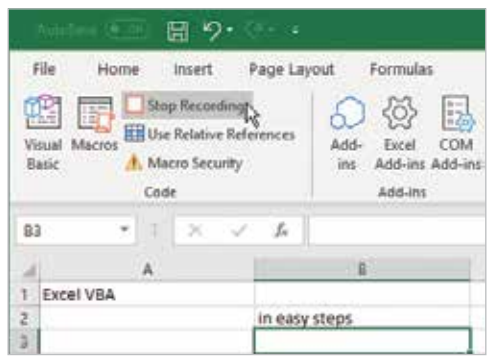
Referencing relatives

Macros recorded using relative referencing are often more flexible, as they can be applied anywhere in a workbook.

Excel has two macro recording modes that differ in the way they refer to cells on the worksheet. The default recording mode, used in the previous examples, refers to cells by their "absolute" position on the worksheet – cell A1, A2, A3, and so on. The alternative recording mode refers to cell locations by their position on the worksheet "relative" to other cells – offset by a specified number of rows and columns from another cell. The difference between the two recording modes is important, as macros that use absolute referencing always reference the same cell locations regardless of the currently selected cell, whereas macros that use relative referencing reference cells at locations offset from the selected cell:

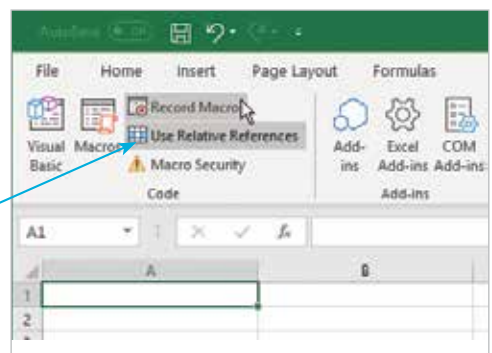
1. Clear all worksheet cells, then select cell **A1** and begin a macro named "AbsoluteBookTitle"

2. Type this book's topic, then select cell **B2** and type this book's series name



3. Hit **Enter**, then click **Stop Recording**

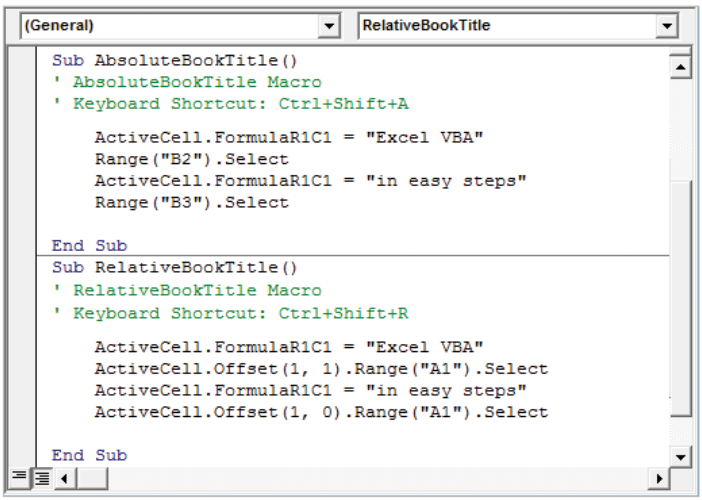
4. Clear all worksheet cells, then select cell **A1** and click the **Use Relative References** button in the Code group



5. Begin a macro named "RelativeBookTitle", then repeat Steps 2 and 3 to complete the macro

Shortcut keys might also be specified to run these macros – for example, **Ctrl** + **Shift** + **A** (Absolute) and **Ctrl** + **Shift** + **R** (Relative).

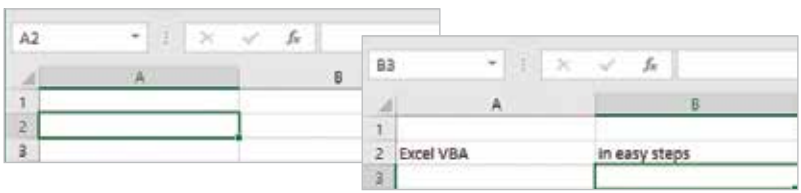
6 Click the **Visual Basic** button to open the Visual Basic Editor, then compare the VBA code of each macro



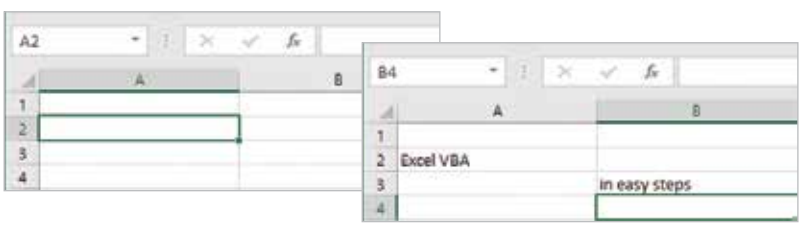
Empty comment lines are removed from this screenshot to save space.

When selecting cell B2, absolute referencing refers to it by name, but relative referencing refers to it as offset by 1 row and 1 column from the initially selected cell. To compare performance:

7 Clear all cells, then select cell **A2** and run the macro named "AbsoluteBookTitle"



8 Again, clear all cells, then once more select cell **A2** and run the macro named "RelativeBookTitle"



In this example, the macro using absolute referencing writes the book series name in the cell named **B2**, whereas the macro using relative referencing writes the book series name in cell **B3** – as it is offset by 1 row and 1 column from the initially selected cell.

Saving macros

Since Excel 2007, workbook files have had the standard file extension of ".xlsx", but these cannot contain Visual Basic macros. In order to save an Excel workbook and its macros, it must instead be saved as an Excel Macro-Enabled Workbook that is given a file extension of ".xlsm". If you save a workbook containing a macro as a standard ".xlsx" file, all macro code will be lost – but Excel will warn you before this happens:

Choose a folder location where you want to save workbooks. Here, it's a folder named "Excel Workbooks" within the system's **Documents** folder.

1. In Excel, select **File**, **Save As**, then type "BookTitle" as the workbook name and click the **Save** button

2. If the workbook contains a macro, a warning dialog will appear asking if you wish to proceed – unless you want to save the workbook without its macro, click the **No** button

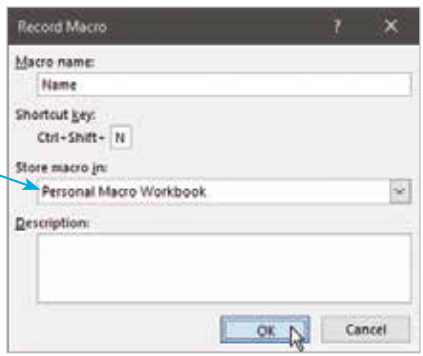
Click the ▾ button to reveal a drop-down list of file types from which to choose.

3. Change the file type to Excel Macro-Enabled Workbook, then click the **Save** button to save the complete workbook

Although most macros are intended for use in a specific workbook, general-purpose macros that may be useful in many workbooks can be saved in the special Personal Macro Workbook. This is a file named "personal.xlsb" that automatically opens in the background when Excel starts up – so the macros it contains are available to any other workbook. To save a macro in the Personal Macro Workbook, simply choose that option in the Record Macro dialog before you begin recording a macro:

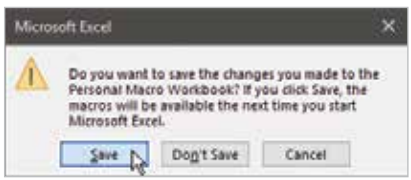
The Personal Macro Workbook runs in a hidden window that you can reveal by selecting the **View** tab, then choosing **Unhide** in the Window group.

1. Click the **Record Macro** button and call the macro "Name", then choose the **Personal Macro Workbook** option



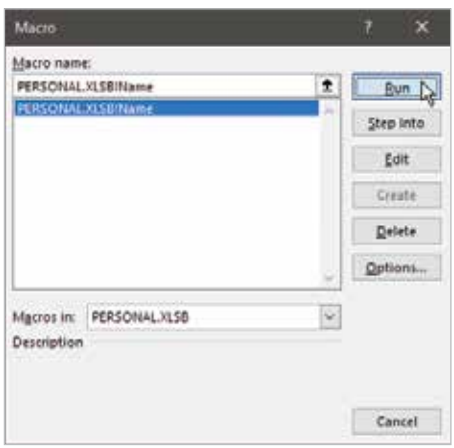
2. Type your name into the selected cell, then select **Stop Recording** and close Excel

3. A dialog will appear asking if you wish to save changes made to the Personal Macro Workbook – click the **Save** button to retain the macro



4. Next, start Excel and begin a new **Blank workbook**, then click the **Macros** button in the Code group

5. Now, choose the saved "Name" macro and click **Run** to write your name into a cell



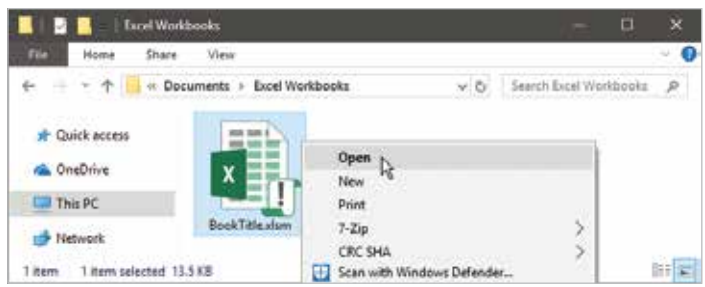
Trusting macros

Excel Workbook files (.xlsx) are regarded as safe, as they merely contain data, whereas Excel Macro-Enabled Workbook files (.xlsm) may pose a potential threat, as they are executable. Recognizing this, Excel automatically disables the macros in an Excel Macro-Enabled Workbook until the user consents to trust their safety. On opening a workbook that contains macros, a security warning offers the user the option to enable macros. If the user consents to enable macros, the workbook is regarded as trustworthy and the security warning will never appear again.

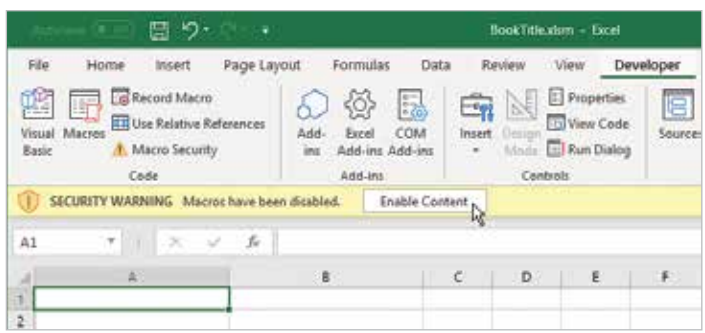
As an alternative to enabling macros in individual workbooks, a folder can be nominated as a trusted location. This then allows Excel Macro-Enabled Workbook files to be placed inside that folder and run without security restrictions:

Both **.xlsx** and **.xlsm** file types store workbook data in XML format. Excel also supports **.xlsb** files that store workbook data in binary format. This is favored by some users, but workbook content is more accessible to other software when stored as XML data.

1. Navigate to the folder containing an Excel Macro-Enabled Workbook, and open it in Excel

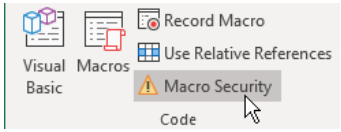


2. Click the **Enable Content** button if you consent to permanently enable macros in this workbook



Macros have been used to distribute malware – be wary of enabling macros in a workbook from an unknown source.

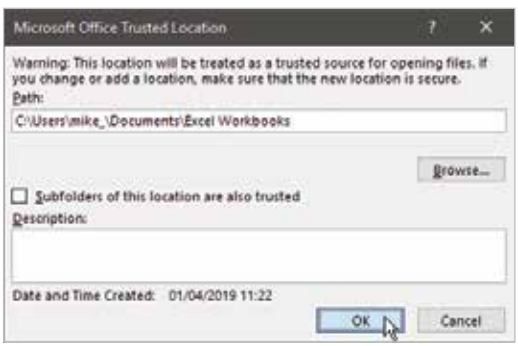
3 Next, click the **Macro Security** button in the Code group to open the "Trust Center" dialog box



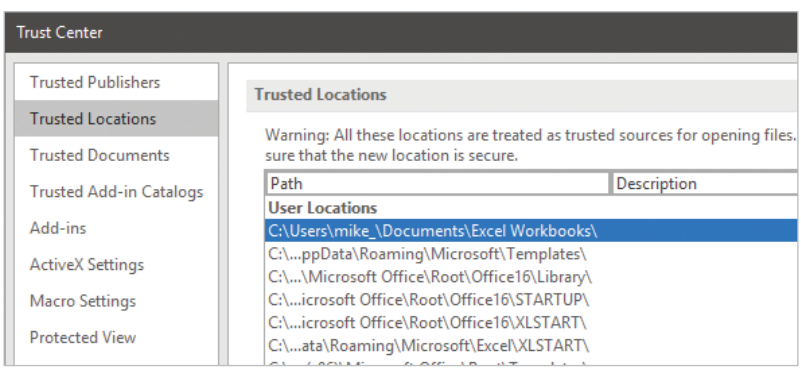
4 In the Trust Center dialog, select the **Trusted Locations** item in the left-hand panel

5 Now, click the **Add new location...** button to open the "Microsoft Office Trusted Location" dialog

6 Browse to select the folder you wish to nominate as a trusted location for Excel Macro-Enabled Workbooks



7 Click the **OK** button to see your nominated folder added to the list of **Trusted Locations** in the Trust Center



You can also use **Trusted Documents** to nominate a workbook so it will run without security restrictions.

All workbooks in **Trusted Locations** will run without security restrictions.

Summary

- **VBA** (Visual Basic for Applications) is the programming language built into Excel that extends its capabilities beyond the standard Excel tools.
- The Excel Developer option enables VBA and adds a **Developer** tab to the Excel Ribbon.
- The **Code** group on the Developer tab contains a **Record Macro** button with which to create VBA macros.
- The **Macros** button in the Code group lists available macros.
- The **Visual Basic** button in the Code group opens the Visual Basic Editor to inspect macro programming instructions.
- Macro subroutines are stored in a project's **Module1** node.
- Subroutines contain programming instructions and comments.
- Specified shortcut keys, or the **Run** button in the **Macro** dialog, can be used to run a macro.
- Recorded macro programming instructions can be edited in the **Visual Basic Editor** to change the macro's behavior.
- Excel's default macro recording mode references cells by their absolute position.
- The **Use Relative References** button in the Code group enables Excel's alternative macro recording mode, which references cells by their relative position.
- A workbook that contains macro code must be saved as an **Excel Macro-Enabled Workbook** and ".xlsm" file extension.
- General-purpose macros can be saved in the **Personal Macro Workbook** so they are available in other workbooks.
- Excel automatically disables macros in an Excel Macro-Enabled Workbook until the user consents to trust them.
- A folder can be nominated as a **trusted location** where Excel Macro-Enabled Workbooks can be placed and run without security restrictions.