



Office Scripts Introduction

Sample manual - first two chapters



Wise Owl
Training

TABLE OF CONTENTS (1 of 5)

1	GETTING STARTED	Page
1.1	Introduction to Office Scripts	7
	<i>What are Office Scripts?</i>	7
	<i>What do I need to use Office Scripts?</i>	7
	<i>Can't I already Automate Excel?</i>	8
	<i>Why use Office Scripts and not VBA?</i>	8
	<i>What are the Limitations?</i>	8
1.2	Preparing to Write Office Scripts	9
	<i>The Automate Ribbon Tab</i>	9
	<i>The Code Editor</i>	10
1.3	Writing Your First Script	11
	<i>Creating a New Script</i>	11
	<i>The Basic Structure of a Script</i>	11
	<i>Adding a Comment</i>	12
	<i>Writing Basic Instructions</i>	12
	<i>Adding More Instructions</i>	13
	<i>Running the Script</i>	13
1.4	Code Editor Settings	14
	<i>Changing Code Editor Settings</i>	14
	<i>A Note on Visual Studio Code</i>	14

2	WORKING WITH SCRIPT FILES	Page
2.1	Office Script Files	15
	<i>Viewing Existing Script Files</i>	15
	<i>The More Options Menu</i>	15
	<i>Creating a New Script File</i>	16
	<i>Opening an Existing Script</i>	16
	<i>Renaming an Open Script</i>	16
	<i>Deleting a Script</i>	16
2.2	Saving Script Files	17
	<i>Where Office Scripts are Saved</i>	17
	<i>Saving an Open Script</i>	17
	<i>The Save As Option</i>	17
2.3	Sharing Scripts	18
	<i>Sharing a Script</i>	18
	<i>Stopping Sharing a Script</i>	18

3	WRITING OFFICE SCRIPTS	Page
3.1	TypeScript Language Basics	19
	<i>TypeScript Language Elements</i>	19
3.2	Tools for Writing Code	20
	<i>IntelliSense</i>	20
	<i>Showing More Information</i>	20
	<i>Showing Help Popups</i>	21
	<i>The Context Menu</i>	21
	<i>The Command Palette</i>	21
3.3	Writing Neat Code	22
	<i>Writing Comments</i>	22
	<i>Commenting Out Code</i>	22
	<i>Writing Long Instructions</i>	23
	<i>White Space and Indenting</i>	23
	<i>Auto Formatting a Script</i>	24
	<i>Using Semicolons</i>	24
3.4	Dealing with Errors	25
	<i>Spotting Compile-time Errors</i>	25
	<i>Viewing Error Information</i>	26
	<i>Cycling Through Errors</i>	26
	<i>Fixing Errors Automatically</i>	26
3.5	Getting Help	27
	<i>Microsoft Documentation</i>	27
	<i>Other Documentation</i>	27
	<i>Recording Scripts</i>	28
	<i>Searching for Help</i>	28

4	RUNNING SCRIPTS	Page
4.1	Introduction	29
	<i>Running a Script</i>	29
	<i>Stopping a Script that is Running</i>	29
4.2	When Things Go Wrong	30
	<i>Spotting Runtime Errors</i>	30
4.3	Using Buttons	31
	<i>Creating a Button</i>	31
	<i>Using a Button to Run a Script</i>	31
	<i>Editing a Button</i>	32
	<i>Removing Buttons</i>	33

TABLE OF CONTENTS (2 of 5)

5	RANGES	Page
5.1	Referring to Range Objects	34
	<i>Referring to the Active Cell</i>	34
	<i>Referring to a Cell by Reference</i>	34
	<i>Referring to Multiple Cells</i>	35
	<i>Using Named Ranges</i>	35
	<i>Using Row and Column Numbers</i>	36
	<i>Multiple Cells using Indexes</i>	36
5.2	The RangeAreas Object	37
	<i>Referring to Multiple Separate Ranges</i>	37
	<i>RangeAreas vs. Range Objects</i>	37
5.3	Getting and Setting Cell Values	38
	<i>Changing the Value of a Cell</i>	38
	<i>Changing Values without Selecting Cells</i>	38
	<i>Reading the Value of a Cell</i>	39
	<i>Cell Data Types</i>	39
5.4	Cell Formulas	40
	<i>Creating Formulas using Cell References</i>	40
	<i>Creating Formulas using R1C1 Notation</i>	40
5.5	Formatting Cells	41
	<i>Fill Formatting Properties</i>	41
	<i>Font Formatting Properties</i>	41
	<i>Formatting Cell Borders</i>	42
	<i>Changing Number Formats</i>	42
	<i>Using Cell Styles</i>	42
5.6	Moving Between Worksheets	43
	<i>Activating a Worksheet</i>	43
	<i>Modifying Cells without Switching Sheets</i>	43
5.7	Relative Cell References	44
	<i>Finding the Last Cell in a List</i>	44
	<i>Selecting to the End of a Filled Range</i>	44
	<i>Referring to a Region of Cells</i>	45
	<i>Offsetting Rows and Columns</i>	45
	<i>Resizing a Range</i>	46
	<i>Entire Rows and Columns</i>	46
	<i>Getting the Used Range</i>	47
	<i>Finding Special Cells</i>	47
5.8	Inserting, Deleting and Clearing Cells	48
	<i>Inserting Cells</i>	48
	<i>Deleting Cells</i>	48
	<i>Clearing Cells</i>	48
5.9	Moving and Copying Cells	49
	<i>Moving Cells</i>	49
	<i>Copying Cells</i>	50
	<i>Options for Copying Cells</i>	50
5.10	Finding Ranges	51
	<i>Finding a Single Cell</i>	51
	<i>Finding Multiple Cells</i>	51

6	VARIABLES	Page
6.1	What are Variables?	52
	<i>Declaring a Variable</i>	52
	<i>Reassigning Variables</i>	53
	<i>Declaring Constants</i>	53
	<i>Using the Var Statement</i>	53
6.2	Variable Scope	54
	<i>Scope with the Let Statement</i>	54
	<i>Scope with the Var Statement</i>	54
6.3	Variables and Types	55
	<i>Primitive and Object Types</i>	55
	<i>Implicitly Typed Variables</i>	55
	<i>Explicit Variable Types</i>	56
	<i>Casting Types</i>	56
	<i>Printing the Type of a Value</i>	57
	<i>The Unknown Type</i>	57

7	CONDITIONAL STATEMENTS	Page
7.1	The If Statement	58
	<i>Basic If Statements</i>	58
	<i>The Else Clause</i>	59
	<i>Else If Clauses</i>	59
7.2	Writing Logical Tests	60
	<i>Comparison Operators</i>	60
	<i>Combining Conditions</i>	60
	<i>The Not Operator</i>	60
7.3	Conditions and Objects	61
	<i>Testing if an Object Exists</i>	61
	<i>Conditional Actions</i>	61
7.4	Conditions and Cell Values	62
	<i>Testing the Type of a Value</i>	62
	<i>Testing for Empty Cells</i>	62
7.5	Other Conditional Statements	63
	<i>The Conditional Operator</i>	63
	<i>The Switch Statement</i>	63

TABLE OF CONTENTS (3 of 5)

8	WORKING WITH DATA	Page
8.1	Working with Numbers	64
	<i>Basic Operators for Numbers</i>	64
	<i>Order of Operations</i>	64
	<i>Number Assignment Operators</i>	65
	<i>Formatting Numbers</i>	65
	<i>Converting Strings to Numbers</i>	66
	<i>The Number Object</i>	66
	<i>The Math Object</i>	67
	<i>Rounding Numbers</i>	67
	<i>Generating Random Numbers</i>	67
8.2	Working with Strings	68
	<i>Concatenating Text</i>	68
	<i>String Templates</i>	68
	<i>Escaping Characters</i>	69
	<i>New Lines</i>	69
	<i>Editing Strings</i>	70
	<i>Comparing Strings</i>	70
	<i>Searching Strings</i>	71
	<i>Regular Expressions</i>	71
8.3	Working with Dates	72
	<i>Creating a Date Object</i>	72
	<i>Getting and Setting Date Parts</i>	72
	<i>Formatting Dates</i>	73
	<i>Converting Excel Dates to TypeScript Dates</i>	73
	<i>Testing for Date Values</i>	74
	<i>A Quirk of TypeScript Dates</i>	74

10	CONDITIONAL LOOPS	Page
10.1	Conditional Loops	82
	<i>While Loops</i>	82
	<i>Do While Loops</i>	83
	<i>Nesting Loops</i>	83
10.2	Exiting a Loop	84
	<i>The Break Statement</i>	84
	<i>The Continue Statement</i>	84
	<i>Using Line Labels</i>	85
10.3	Finding Text with a Loop	86

11	FOR LOOPS	Page
11.1	Introduction to For Loops	87
	<i>For Loop Syntax</i>	87
	<i>A Basic Example</i>	87
11.2	For Loops and Ranges	88
	<i>Getting a Cell by Row and Column Number</i>	88
	<i>Offsetting from a Range</i>	88
	<i>Looping Over Rows and Columns</i>	89
11.3	More For Loop Features	90
	<i>Continue and Break Statements</i>	90
	<i>Optional Loop Statements</i>	90
	<i>Multiple Loop Statements</i>	90

9	FUNCTIONS AND PARAMETERS	Page
9.1	Functions	75
	<i>Why Create Extra Functions?</i>	75
	<i>Declaring a Function</i>	76
	<i>Calling a Function</i>	76
9.2	Parameters	77
	<i>Declaring Parameters</i>	77
	<i>Default Parameter Values</i>	77
	<i>Optional Parameters</i>	78
9.3	Returning Values	79
	<i>The Return Statement</i>	79
	<i>Specifying the Return Type</i>	79
9.4	Function Expressions	80
	<i>Creating a Function Expression</i>	80
	<i>Anonymous Function Expressions</i>	80
9.5	Arrow Function Expressions	81
	<i>Creating an Arrow Function Expression</i>	81
	<i>Including Multiple Statements</i>	81

TABLE OF CONTENTS (4 of 5)

12	ARRAYS	Page
12.1	Introduction to Arrays	91
	<i>Declaring and Populating an Array</i>	91
	<i>Viewing Array Contents</i>	91
	<i>Referring to Array Elements</i>	92
	<i>Declaring an Empty Array</i>	92
12.2	Modifying Array Contents	93
	<i>Sorting Arrays</i>	93
	<i>Adding Items</i>	93
	<i>Removing Items</i>	94
	<i>Altering Array Values</i>	94
12.3	Searching Arrays	95
	<i>Basic Array Search Methods</i>	95
	<i>Searching Arrays with Arrow Functions</i>	95
12.4	Extracting Values from Arrays	96
	<i>Extracting Items to New Arrays</i>	96
	<i>Combining Arrays into a New Array</i>	96
	<i>Converting an Array to a String</i>	96
12.5	Looping Over Arrays	97
	<i>While Loops</i>	97
	<i>For Loops</i>	97
	<i>For Of Loops</i>	98
	<i>The forEach Method</i>	98
12.6	Multi-Dimensional Arrays	99
	<i>Creating a Multi-Dimensional Array</i>	99
	<i>Referring to Elements</i>	99
	<i>Looping Over Multi-Dimensional Arrays</i>	100
12.7	Arrays and Ranges	101
	<i>Getting Cell Contents as an Array</i>	101
	<i>Writing an Array to a Range</i>	102
	<i>Looping Through Range Values</i>	103
	<i>RangeAreas and Arrays</i>	104
12.8	Arrays and Function Parameters	105
	<i>The Argument Array</i>	105
	<i>Passing an Array to a Function</i>	105
	<i>The Rest Parameter</i>	106
	<i>Returning an Array</i>	106

13	COLLECTIONS	Page
13.1	Introduction to Collections	107
	<i>What are Collections?</i>	107
13.2	Collections and Objects	108
	<i>Referencing Collections</i>	108
	<i>Collections as Arrays</i>	108
	<i>Referencing Objects in Collections</i>	109
	<i>Referencing Objects as Items in Arrays</i>	109
13.3	Adding and Deleting Objects	110
	<i>Adding Items to Collections</i>	110
	<i>Deleting Items from a Collection</i>	110
13.4	Looping Through Collections	111
	<i>Deleting Objects in a Loop</i>	111
13.5	Worksheet Examples	112
	<i>Protecting and Unprotecting Worksheets</i>	112
	<i>Hiding all but One Worksheet</i>	112
	<i>Consolidating Worksheets</i>	113
	<i>Dividing a List into New Worksheets</i>	114
13.6	Chart Examples	115
	<i>Changing Chart Types and Formats</i>	115
	<i>Creating a Chart on Every Worksheet</i>	115
	<i>Looping Over Chart Series</i>	116
	<i>Looping Through Data Points</i>	116
13.7	Table Examples	117
	<i>Creating a Table on Each Worksheet</i>	117
	<i>Looping Through Tables</i>	117
	<i>Table Columns</i>	118
	<i>Adding Table Rows</i>	118

14	ERROR HANDLING	Page
14.1	What is Error Handling?	119
	<i>The Try Catch Statement</i>	119
	<i>The Finally Block</i>	120
	<i>The Error Object</i>	120
14.2	Throwing Exceptions	121
	<i>The Throw Statement</i>	121
14.3	Errors in Functions	122
	<i>Catching Errors in the Calling Function</i>	122
	<i>Throwing Errors in a Called Function</i>	122
	<i>Catching Errors in a Called Function</i>	123

TABLE OF CONTENTS (5 of 5)

15	CRIB SHEET	Page
15.1	Office Scripts Reference	124
	<i>Selecting and Activating Things</i>	124
	<i>Selecting a Range Relatively</i>	125
	<i>Using Variables</i>	126
	<i>Conditional Statements</i>	127
	<i>Looping</i>	128
	<i>Creating Functions</i>	129

CHAPTER 1 - GETTING STARTED

1.1 Introduction to Office Scripts

This chapter introduces you to *Office Scripts*, beginning with the answers to a few simple questions.

What are Office Scripts?

Office Scripts are small programs used to automate tasks in Microsoft Excel. You write code in the *TypeScript* language and you can either record the steps or write your program from scratch.

```
1 function main(workbook: ExcelScript.Workbook) {
2
3     // Get a reference to the Input worksheet
4     let ws: ExcelScript.Worksheet = workbook
5         .getWorksheet("Input");
6
7     // Check that the worksheet exists
8     if (!ws) {
9         console.log("Input sheet missing");
10        return;
11    }
12
13    // Get reference to next blank cell
14    let rng: ExcelScript.Range = ws
15        .getRange("A1")
16        .getRangeEdge(ExcelScript.KeyboardDirection.down)
17        .getOffsetRange(1, 0);
18
19    // Enter value and format cell
20    rng.setValue("Wise Owl");
21    rng.getFormat().getFill().setColor("#f28400");
22    rng.getFormat().getFont().setName("Arial");
23 }
```

A basic Office Script to add a value to the end of a list and apply some formatting.

What do I need to use Office Scripts?

According to Microsoft, these are the three requirements for using Office Scripts:

- 1) Excel for Windows, for Mac, or on the web.
- 2) OneDrive for Business.
- 3) Any commercial or educational Microsoft 365 license with access to the Microsoft 365 Office desktop apps.

Can't I already Automate Excel?

If you're thinking that Office Scripts sound a lot like *VBA macros*, you're right! In fact, there are three main ways to automate Excel, summarised in the table below:

Method	What you can do
<i>Office Scripts</i>	Automate the desktop or online version of Excel.
<i>VBA Macros</i>	Automate any desktop Office application.
<i>Office Add-Ins</i>	Create an add-in to extend the features of any desktop or online Office application.

Why use Office Scripts and not VBA?

Office Scripts allow you to do some things you can't do with VBA macros, as described in the table below:

Feature	Description
<i>Excel online</i>	You can use Office Scripts to automate workbooks in the web version of Excel.
<i>Power Automate</i>	You can run Office Scripts from a Power Automate flow.
<i>External APIs</i>	Office Scripts support calls to external APIs which can provide data to your files.
<i>Better security</i>	An Office Script only has access to the workbook in which it is running, unlike VBA which has access to your entire computer.

What are the Limitations?

Office Scripts have several limitations compared to VBA and Office Add-Ins, as described in the table below:

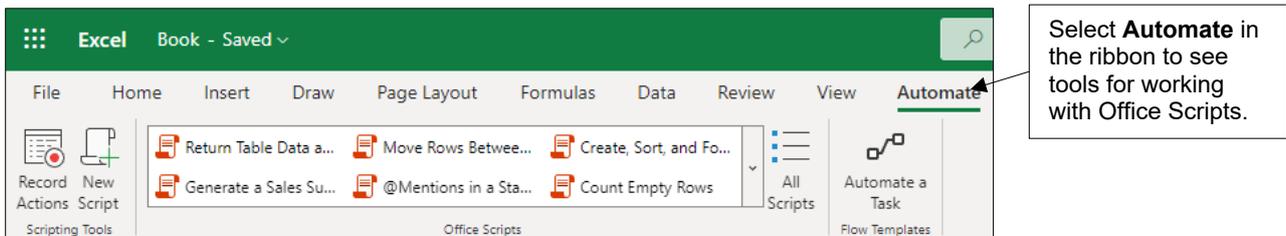
Feature	Description
<i>Excel only</i>	Currently, Office Scripts can only be used in Excel. Office Add-Ins can also work with Word, PowerPoint, Outlook, OneNote and Project. VBA can use OLE and COM libraries to control a variety of other applications, including the Microsoft Office apps.
<i>No events</i>	VBA and Office Add-Ins can respond to events to make code run automatically. Office Scripts must be run explicitly by the user.
<i>Single workbook</i>	An Office Script only has access to the workbook in which it is running. Even something as simple as copying a value from one workbook to another requires the use of a Power Automate flow.
<i>User interface</i>	Office Scripts don't have access to any of the user interface elements of Excel. This means that you can't display dialog boxes or similar UI elements. If you need to create a user-interface it would be better to use VBA or to create a full Office Add-In.

1.2 Preparing to Write Office Scripts

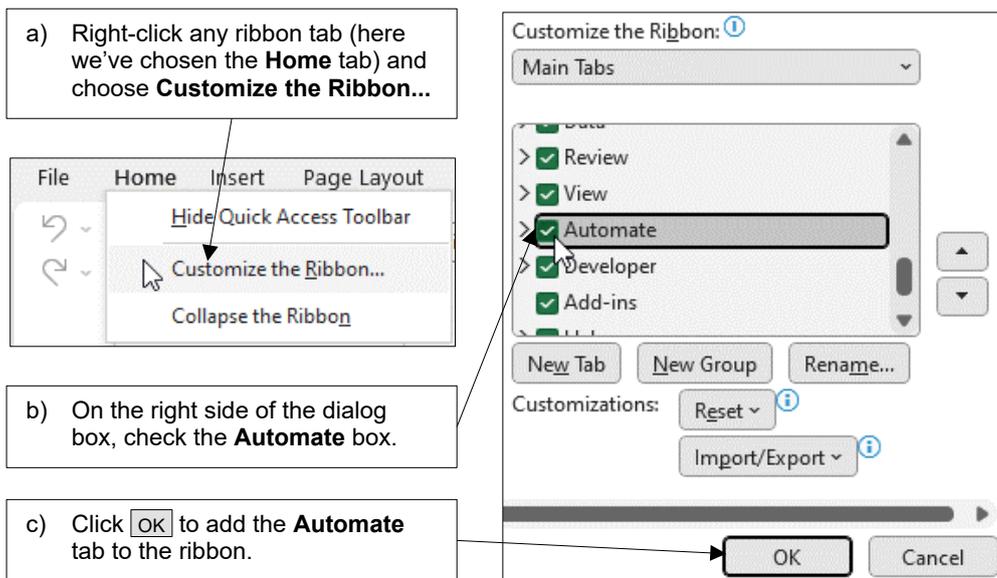
This section shows you the basic things you need to do before you can write your first script.

The Automate Ribbon Tab

The *Automate* tab of the ribbon contains the tools you need to write Office Scripts.



If you're using the desktop version of Excel, the **Automate** tab may not appear automatically. You can see how to display it in the diagram below:



The Code Editor

You edit Office Scripts in the *Code Editor* window in Excel. You can open the **Code Editor** in several ways; a simple option is to choose **Automate | All Scripts** from the Excel ribbon.

Click **All Scripts** to see the **Code Editor**.

The **Code Editor** appears on the right of the Excel window.

You can click here to create a new script.

This area will show a list of existing scripts and gives you a quick way to open them.

You can change the width of the **Code Editor** (you will definitely want to increase this!) by clicking and dragging the vertical divider.

In the desktop version of Excel you can move and resize the Code Editor window by clicking and dragging its title bar.

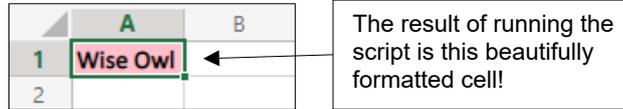
Click and drag the title bar of the Code Editor to move it.

If you leave the Code Editor floating over the Excel window you can click and drag the window borders to resize it.

You can drag the Code Editor window to the left or right of the Excel window to dock it in position.

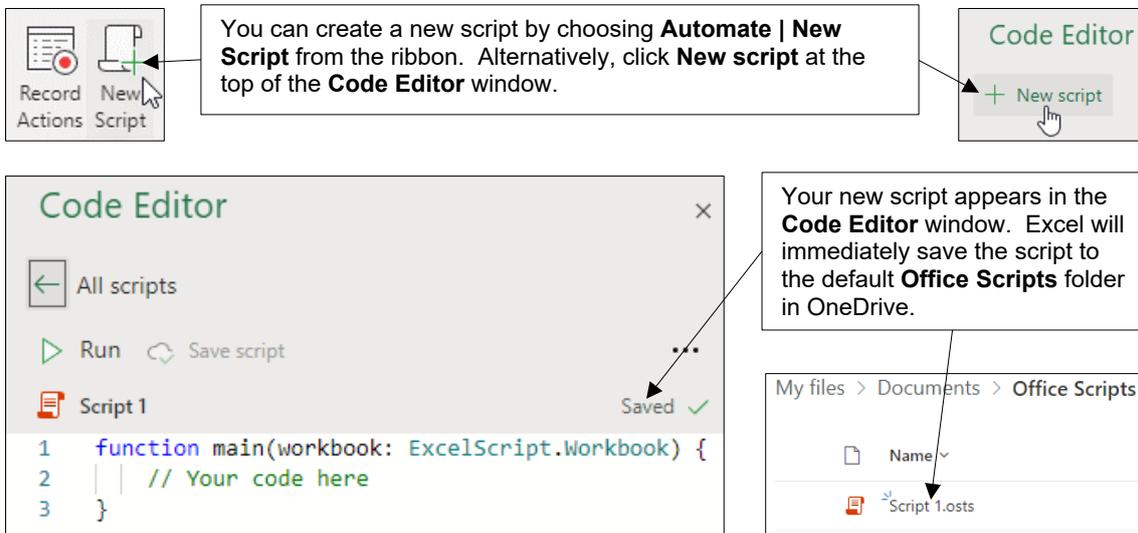
1.3 Writing Your First Script

This section shows you how to write a simple program to add text to a cell and apply some formatting to get used to the basics of writing code.



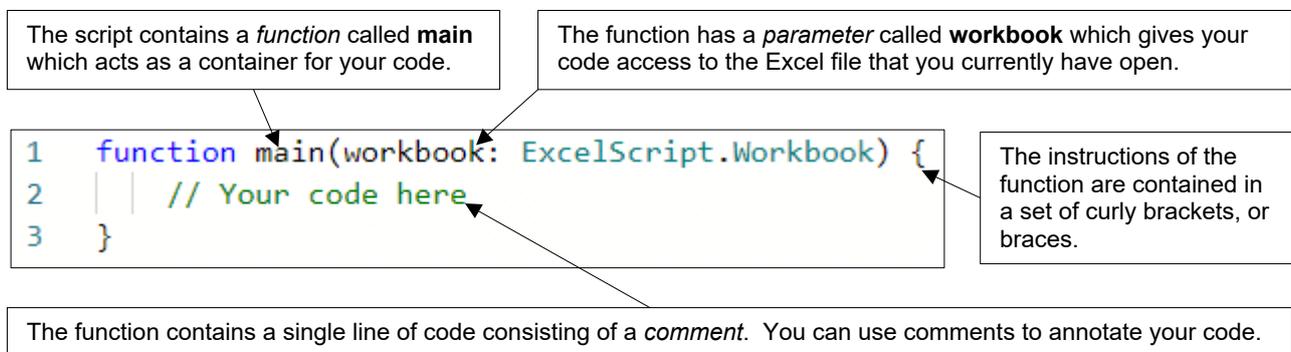
Creating a New Script

Start by creating a new workbook in either the desktop or online version of Excel and then choose to create a new Office Script.



The Basic Structure of a Script

You can see the basic structure of your new script in the diagram below:



Adding a Comment

You can add your own comments to your code to provide helpful reminders of what it is meant to do.

```
1 function main(workbook: ExcelScript.Workbook) {
2     // Your code here
3
4
5 }
```

Click at the end of the line 2 then press **Enter** twice to provide a new line on which to write your comment.

```
1 function main(workbook: ExcelScript.Workbook) {
2     // Your code here
3
4     // Enter my name in the selected cell
5 }
```

Type two forward slashes to begin the comment and follow these with anything you feel is appropriate.

Writing Basic Instructions

Our first instruction will write the name **Wise Owl** into the currently selected cell in the workbook. To do this:

- 1) On a new line in the script, begin typing the word **workbook** (take care when doing this – TypeScript is a case-sensitive language!).

As you begin typing you'll see a list of options matching what you've typed so far. To select an item from this *Intellisense* list you can press **Tab** or **Enter** or click on the word with the mouse.

```
1 function main(workbook: ExcelScript.Workbook) {
2     // Your code here
3
4     // Enter my name in the selected cell
5     work
6 }
```

- 2) After the word **workbook**, enter a full stop and begin typing **getActiveCell**.

```
4     // Enter my name in the selected cell
5     workbook.getA
6 }
```

Use the Intellisense list to help you finish entering the word.

- 3) After **getActiveCell**, open and close some round brackets.

```
4     // Enter my name in the selected cell
5     workbook.getActiveCell()
6 }
```

You can type in the open round bracket and the closing bracket will be added for you automatically.

- 4) Enter another full stop followed by **setValue** then an open round bracket and double-quotes.

```
4     // Enter my name in the selected cell
5     workbook.getActiveCell().setValue("")
6 }
```

The other double-quote and closing round bracket will be added for you automatically.

- 5) Between the double quotes, enter the name you want to write into the cell.

```
4 // Enter my name in the selected cell
5 workbook.getActiveSheet().setValue("Wise Owl");
6 }
```

You can change the text in the double quotes to anything you like.

- 6) Complete the instruction by adding a semicolon to the end of the line.

```
4 // Enter my name in the selected cell
5 workbook.getActiveSheet().setValue("Wise Owl");
6 }
```

Each TypeScript instruction ends with a semicolon.

Adding More Instructions

You can continue adding as many instructions as you need to complete your program. The diagram below shows the complete script with two extra instructions and relevant comments:

```
1 function main(workbook: ExcelScript.Workbook) {
2
3     // Enter my name in the selected cell
4     workbook.getActiveSheet().setValue("Wise Owl");
5
6     // Change the cell fill colour
7     workbook.getActiveSheet().getFormat().getFill().setColor("Pink");
8
9     // Make the font bold
10    workbook.getActiveSheet().getFormat().getFont().setBold(true);
11 }
```

We've deleted the first comment to tidy up the code.

The pattern of the new instructions is similar to the first one we added.

Running the Script

You can see how to run your finished script in the diagram below:

Click **Run** at the top of the Code Editor to run the script that you currently have open.

A message appears to tell you the code is running or, if you're unlucky, that the code has failed! You can click **Stop** if you need to abandon a script that appears to be stuck.

The beautiful end result!



*Experienced VBA programmers should be careful not to press **F5** to run your code. In the online version of Excel this will refresh the page and close the Code Editor!*

1.4 Code Editor Settings

Changing Code Editor Settings

To change Code Editor settings you first need to open a script file.

Click at the top right of the Code Editor and choose **Editor Settings** from the menu.

To return to your script, click this arrow.

You can change the **Theme** to alter the appearance of the editor. The example below uses the **Dark** theme.

```
1 function main(workbook:
2   ExcelScript.Workbook) {
3     // Enter my name in
```

You can enable a **Minimap** of your script which appears at the top right of the code window.

The **Folding** option allows you to collapse and expand sections of your code contained within different types of brackets.

With the **Folding** option enabled you can click arrows like this one to collapse and expand sections of your script.

This option allows you to edit a script in the online version of Visual Studio Code.

A Note on Visual Studio Code

If you're working in the online version of Excel, you can enable the **Visual Studio Code connection** option. You can then open a script in the Visual Studio Code web application, as shown below:

Use the menu at the top right of the code window and choose **Open in VS Code**.

You can edit the script in the new tab.

You can close the Visual Studio tab or click **Close** to return to the Code Editor in Excel.

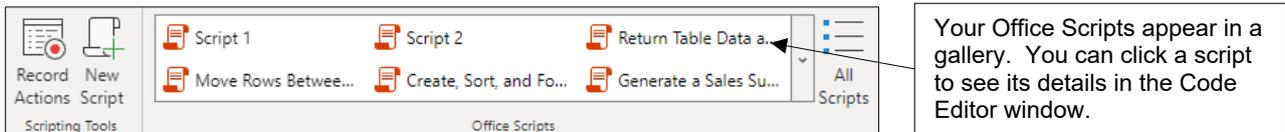
CHAPTER 2 - WORKING WITH SCRIPT FILES

2.1 Office Script Files

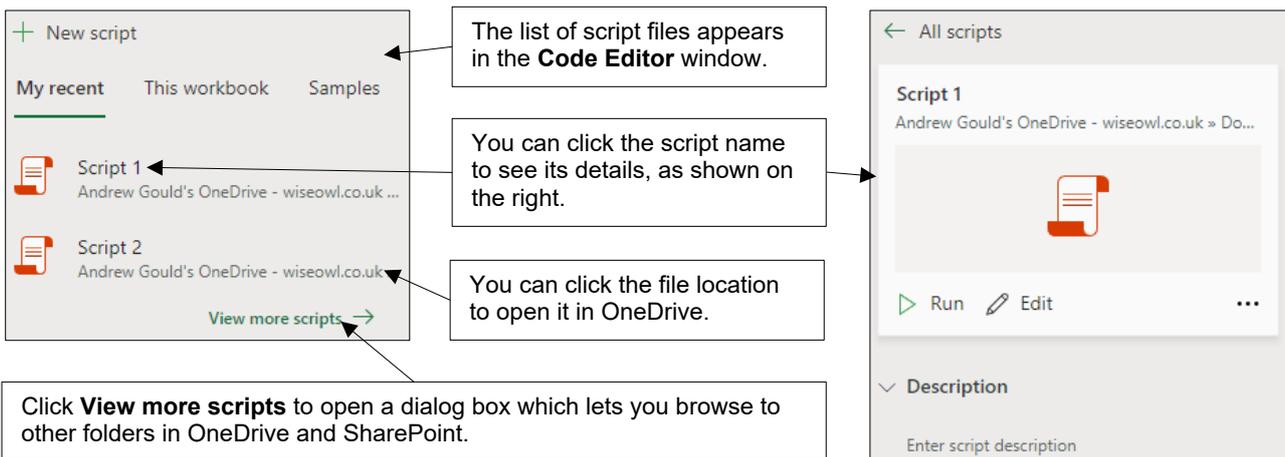
This section explains the basics of working with Office Script files.

Viewing Existing Script Files

You can see your recently used and sample scripts in the **Automate** tab in the Excel ribbon.

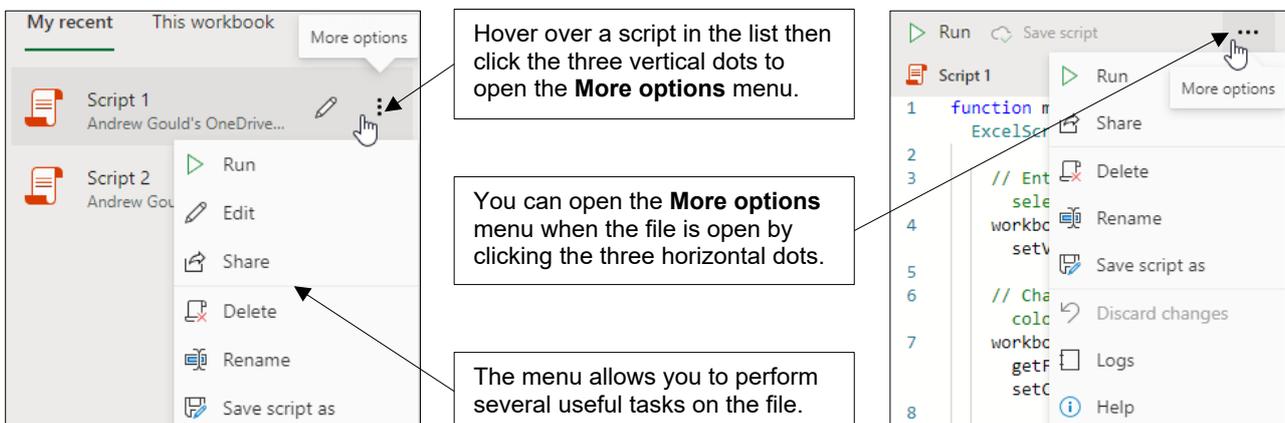


To see all your scripts, choose **Automate | All Scripts** from the Excel ribbon.



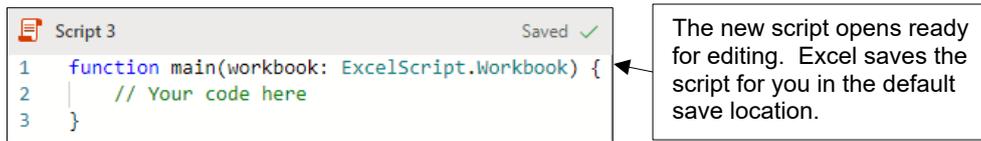
The More Options Menu

You can use a script's **More options** menu to perform several useful tasks on it.



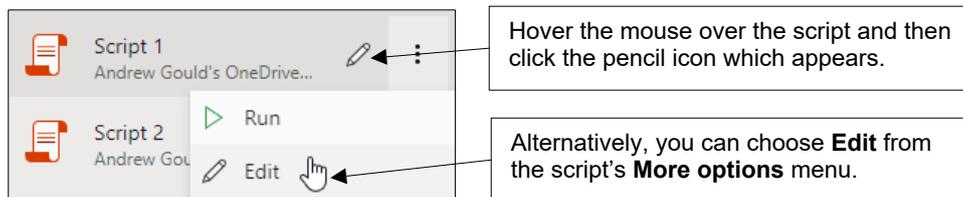
Creating a New Script File

The simplest way to create a new script is to choose **Automate | New Script** from the Excel ribbon.



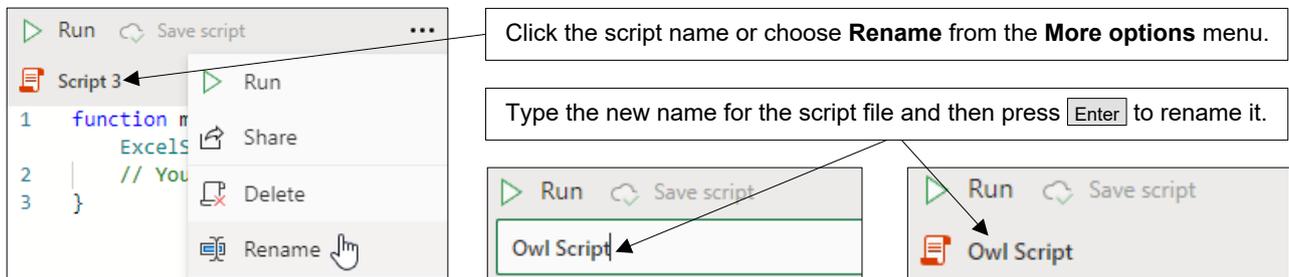
Opening an Existing Script

You can open an existing script to edit its code as shown in the diagram below:



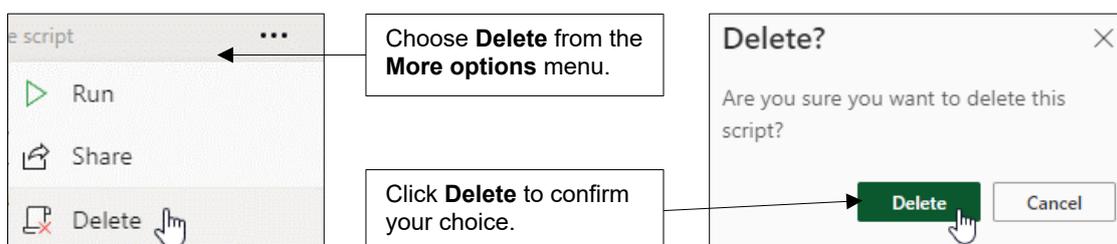
Renaming an Open Script

You can rename a script file, even if you're currently editing it.



Deleting a Script

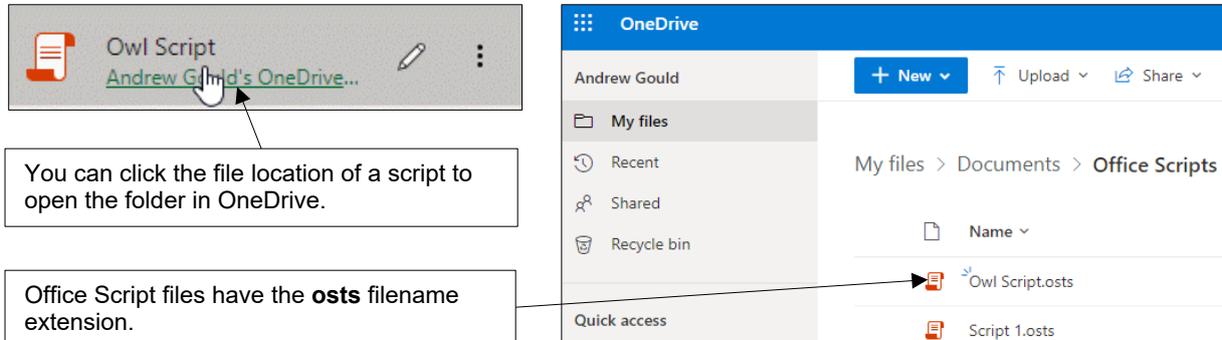
You can delete a script from the **More options** menu, even if you are currently editing it.



2.2 Saving Script Files

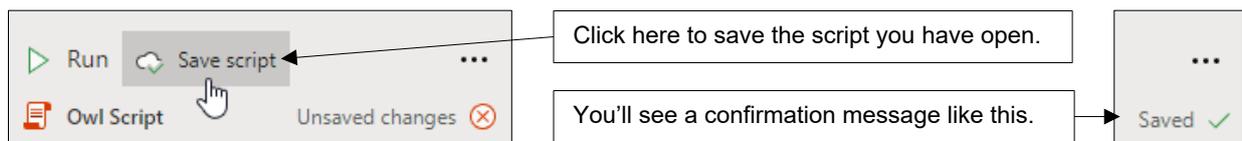
Where Office Scripts are Saved

By default, Office Script files are saved in the **Documents/Office Scripts** folder in OneDrive.



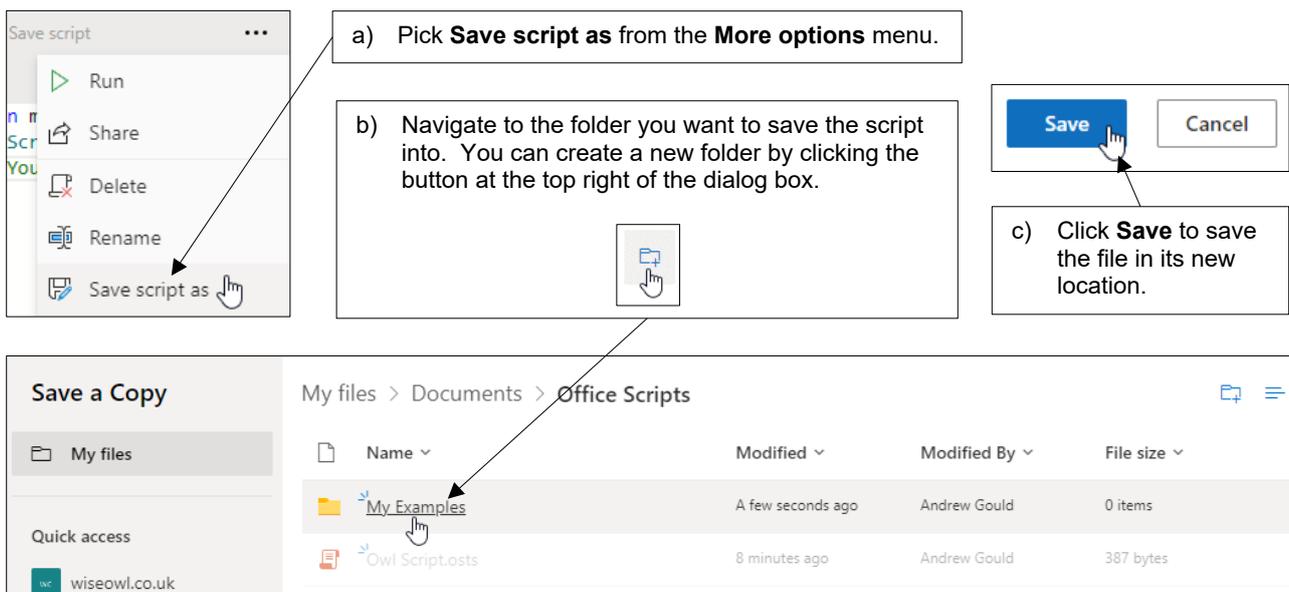
Saving an Open Script

You can save the script that you're currently working on in the default folder as shown below:



The Save As Option

You can choose to save your script in a different location as shown below:



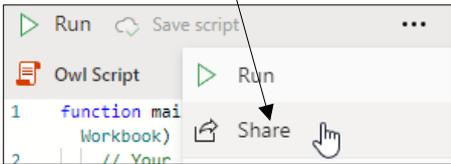
2.3 Sharing Scripts

Scripts saved to your OneDrive are accessible only to you. If you'd like others to use your scripts, you'll need to share them in some way.

Sharing a Script

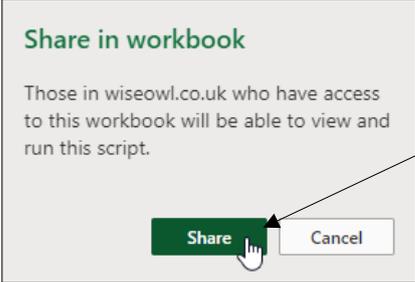
The first step in sharing a script is to share it in a workbook. To do this, first open the workbook you want to share the script in.

a) Choose **Share** from the script's **More options** menu.



Share in workbook

Those in wiseowl.co.uk who have access to this workbook will be able to view and run this script.

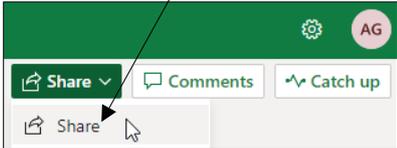


b) Click **Share** to confirm that you want to share the script. The script's icon changes to show that it is being shared:



You'll also need to give other people access to the workbook, as shown in the diagram below:

a) Click the **Share** menu in the top right of the Excel window and choose **Share**.



b) In the dialog box, configure with whom the workbook is to be shared.

Send link

Book.xlsx

People you specify can edit >

To: Name, group or email

Message...

Send

Copy link

People you specify can edit >

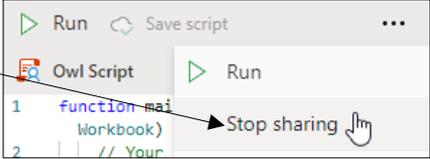
Copy

c) Share the file by sending a link in an email, or by copying the link and distributing it another way.

Stopping Sharing a Script

The diagram below shows how you can reverse the process of sharing an Office Script:

a) From the script's **More options** menu, choose **Stop sharing**.



b) On the dialog box which appears, choose whether to stop sharing in all workbooks or just the currently open one, and then click **Stop sharing**.

Stop sharing in workbook

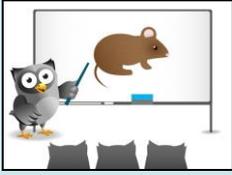
Those in wiseowl.co.uk who have access to this workbook except for the owner will not be able to view and run this script.

Stop sharing in all workbooks

Stop sharing

Cancel

WHAT WE DO

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



**WiseOwl
Training**

