Excel VBA 365 Handbook

A Comprehensive Guide to Excel Macro Programming

by

Dr. Liew Voon Kiong

Disclaimer

Excel VBA 365 Made Easy is an independent publication and is not affiliated with, nor has it been authorized, sponsored, or otherwise approved by Microsoft Corporation.

Trademarks

Microsoft, Visual Basic, Excel and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Liability

The purpose of this book is to provide basic guidelines for people interested in Excel VBA 365 programming. Although every effort and care has been taken to make the information as accurate as possible, the author shall not be liable for any error, harm or damage arising from using the instructions given in this book.

Acknowledgement

I would like to express my sincere gratitude to many people who have made their contributions in one way or another to the successful publication of this book.

My special thanks go to my children Xiang, Yi and Xun. My daughter Xiang edited this book while my sons Yi and Xun contributed their ideas and even wrote some of the sample programs for this book. I would also like to appreciate the support provided by my beloved wife Kim Huang and my youngest daughter Yuan. I would also like to thank the millions of visitors to my Excel VBA Tutorial website at https://excelvbatutor.com/ for their support and encouragement.

About the Author

Dr. Liew Voon Kiong holds a bachelor's degree in Mathematics, a master's degree in Management and a doctorate in Business Administration. He has been involved in Visual Basic programming for more than 30 years. He created the popular online Visual Basic Tutorial at www.vbtutor.net which has attracted millions of visitors since 1996. It has consistently been one of the highest ranked Visual Basic websites.

Dr. Liew is also the author of the Visual Basic Made Easy series, which includes **Excel VBA Made Easy**, **Visual Basic 6 Made Easy, Visual Basic 2008 Made Easy, Visual Basic 2010 Made Easy, Visual Basic 2013 Made Easy, Visual Basic 2015 Made Easy, Visual Basic 2017 Made Easy** and **Visual Basic 2019 Made Easy .** Besides the VB books, he has also published **JavaScript Made Easy**, **JavaScript & JQuery Made Easy** and **HTML & CSS Made Easy**. Dr. Liew's books have been used in high school and university computer science courses all over the world.

TABLE OF CONTENTS

Chapter 1 Introduction to Excel VBA 365 15

1.1 The Concept of Excel VBA 15

1.2 The Visual Basic Editor in MS Excel 365 15

1.2.1 Building Excel VBA 365 using the Controls. 17

Example 1.1 Displaying a Message 20

Example 1.2 Populates Cells with Text and Values 21

1.2.2 Building Excel VBA 365 using the Visual Basic Editor 23

1.2.3 Creating Macros 26

Example 1.3 Creating a Macro 30

Example 1.4 Creating a Salary Calculator 33

Example 1.5 Creating the Macro that Add Two Numbers 34

Example 1.6 A Macro that Populates Cells using the For…Next Loop 35

Example 1.7 A Macro that Populates the Cells with Characters using the Chr() Function. 36

1.3 The Excel VBA 365 Code 37

Example 1.8 Populating a Cell using the Value Property of Range 37

Example 1.9 Coloring the Cells with the Color Property 37

Example 1.10 Adding Numbers Using the Do... Loop 38

Example 1.11 A Macro that Accepts Inputs and Add Numbers 39

1.4 Errors Handling 39

1.4.1 Writing the Errors Handling Code 40

Example 1.12 Catching Error for Invalid Division 40

Example 1.13 Nested Errors Handling 41

Chapter 2 Working with Variables 43

2.1 The Concept of Variables 43

2.2 Variable Names 43

2.3 Declaring Variables 44

2.2.1 Numeric Data Types 44

2.2.2 Non-numeric Data Types 44

Example 2.1 Declaration of Different Data Types 45

Example 2.2 Creating a Salary Calculator Using If… Then…Else 46

2.2 Option Explicit 47

Example 2.3 Using Option Explicit to Catch Typo Errors 48

2.3 Assigning Values to the Variables 49

2.4 Performing Arithmetic Operations 50

Example 2.4 Compute Examination Results 50

Example 2.5 Concatenation of Strings 52

2.5 Arrays 53

2.5.1 Declaring an Array 53

2.5.2 One-Dimensional Array 53

Example 2.6 Array of Names 53

Example 2.7 Declare Arrays in a Single Line 55

2.5.3 Two-Dimensional Array 56

Example 2.8 Tracking the Performance of Salespersons 57

Chapter 3 Message box and Input Box **Error! Bookmark not defined.**

3.1 The MsgBox ( ) Function **Error! Bookmark not defined.**

Example 3.1 Using the Name Constant vbOKCancel **Error! Bookmark not defined.**

Example 3.2 Separating the Message into Three Lines using the Chr() Function **Error! Bookmark not defined.**

Example 3.3 A Number Guessing Game **Error! Bookmark not defined.**

3.2 The InputBox() Function **Error! Bookmark not defined.**

Example 3.4 Using InputBox **Error! Bookmark not defined.**

Chapter 4 Using If...Then…Else **Error! Bookmark not defined.**

4.1 Conditional Operators **Error! Bookmark not defined.**

4.2 Logical Operators **Error! Bookmark not defined.**

4.3 Using If...Then...ElseIf… Else **Error! Bookmark not defined.**

Example 4.1 Comparing Two Numbers **Error! Bookmark not defined.**

Example 4.2 Computing the Examination Grades **Error! Bookmark not defined.**

Example 4.3 The Use of the Not Operator **Error! Bookmark not defined.**

Chapter 5 Looping **Error! Bookmark not defined.**

5.1 For…Next Loop **Error! Bookmark not defined.**

5.1.1 The Single For…Next Loop **Error! Bookmark not defined.**

Example 5.1 Populating Cells with Numbers **Error! Bookmark not defined.**

Example 5.2 Populating Alternative Cells **Error! Bookmark not defined.**

Example 5.3 Early Termination of Program **Error! Bookmark not defined.**

5.1.2 The Nested For…Next Loop **Error! Bookmark not defined.**

Example 5.4 Populating a Range of Cells **Error! Bookmark not defined.**

Example 5.5 Analyzing Exam Results **Error! Bookmark not defined.**

5.2 The Do…Loop **Error! Bookmark not defined.**

Example 5.6 A Counter **Error! Bookmark not defined.**

Example 5.7 Another Counter **Error! Bookmark not defined.**

Example 5.8 Decreasing Numbers **Error! Bookmark not defined.**

Example 5.9 Decreasing Numbers **Error! Bookmark not defined.**

Example 5.10 Displaying Numbers **Error! Bookmark not defined.**

Example 5.11 Formatting Contents using with Selection **Error! Bookmark not defined.**

Example 5.12 Prime Number Tester **Error! Bookmark not defined.**

5.3 The While…Wend Loop **Error! Bookmark not defined.**

Example 5.13 Arithmetic Progression **Error! Bookmark not defined.**

Example 5.14 Exiting a While...Wend Loop **Error! Bookmark not defined.**

Example 5.15 A Number Guessing Game **Error! Bookmark not defined.**

Chapter 6 Select Case...End Select **Error! Bookmark not defined.**

Example 6.1 Processing Student Grades **Error! Bookmark not defined.**

Example 6.2 Using Case Is **Error! Bookmark not defined.**

Example 6.3 Processing Grades **Error! Bookmark not defined.**

Chapter 7: Excel VBA 365 Objects **Error! Bookmark not defined.**

7.1: Objects **Error! Bookmark not defined.**

7.2: Properties and Methods **Error! Bookmark not defined.**

7.2.1 Properties **Error! Bookmark not defined.**

Example 7.1 The Value Property **Error! Bookmark not defined.**

7.2.2 Methods **Error! Bookmark not defined.**

a) The Count method **Error! Bookmark not defined.**

Example 7.2 The Count Property **Error! Bookmark not defined.**

b) The ClearContents Method **Error! Bookmark not defined.**

Example 7.3 Clearing Contents **Error! Bookmark not defined.**

c) The ClearFormats Method **Error! Bookmark not defined.**

Example 7.4 Clearing Format **Error! Bookmark not defined.**

d) The Clear Method **Error! Bookmark not defined.**

Example 7.5 Select Range and Clear Contents **Error! Bookmark not defined.**

e) The Select Method **Error! Bookmark not defined.**

Example 7.6 The Select Method **Error! Bookmark not defined.**

Example 7.7 Selecting a Range of Cells **Error! Bookmark not defined.**

Example 7.8 Select and Clear **Error! Bookmark not defined.**

f) The Autofill Method **Error! Bookmark not defined.**

Example 7.9 Autofill a Range **Error! Bookmark not defined.**

Example 7.10 Set the Source and Destination **Error! Bookmark not defined.**

Example 7.11 Autofill Weekdays **Error! Bookmark not defined.**

Example 7.12 Select and Clear Contents by the User **Error! Bookmark not defined.**

Chapter 8: The Workbook Object **Error! Bookmark not defined.**

8.1 Workbook Properties **Error! Bookmark not defined.**

8.1.1 The Name Property **Error! Bookmark not defined.**

Example 8.1 Displaying the Workbook Name **Error! Bookmark not defined.**
8.1.2 The Path Property **Error! Bookmark not defined.**
Example 8.2 Showing the Path of the workbook **Error! Bookmark not defined.**
Example 8.3 Showing the Path and Name of a Workbook**Error! Bookmark not defined.**
8.2 The Workbook Methods **Error! Bookmark not defined.**
8.2.1 The Save Method **Error! Bookmark not defined.**
Example 8.4 Save Workbook **Error! Bookmark not defined.**
8.2.2 The SaveAs Method **Error! Bookmark not defined.**
Example 8.5 SaveAs Method **Error! Bookmark not defined.**
8.2.3 The Open Method **Error! Bookmark not defined.**
Example 8.6 Opening a File **Error! Bookmark not defined.**
8.2.4 The Close Method **Error! Bookmark not defined.**
Example 8.7 Closing a File **Error! Bookmark not defined.**
Chapter 9 The Worksheet Object **Error! Bookmark not defined.**
9.1 Worksheet Properties **Error! Bookmark not defined.**
9.1.1 The Name Property **Error! Bookmark not defined.**
Example 9.1 Return a Worksheet Name **Error! Bookmark not defined.**
9.1.2 The Count Property **Error! Bookmark not defined.**
Example 9.3 Count Number of Columns **Error! Bookmark not defined.**
Example 9.4 Count Number of Rows **Error! Bookmark not defined.**
9.2 Worksheet Methods **Error! Bookmark not defined.**
9.2.1 The Add Method **Error! Bookmark not defined.**
Example 9.5 Add a New Worksheet **Error! Bookmark not defined.**
9.2.2 The Delete Method **Error! Bookmark not defined.**
Example 9.6 Delete a Worksheet **Error! Bookmark not defined.**
9.2.3 The Select Method **Error! Bookmark not defined.**
Example 9.7 Select a Worksheet **Error! Bookmark not defined.**
Example 9.8 Select a Cell **Error! Bookmark not defined.**

Example 9.9 Select a Range of Cells Error! Bookmark not defined.
Example 9.10 Select a Column of a Worksheet Error! Bookmark not defined.
Example 9.11 Select a Row of a worksheet Error! Bookmark not defined.
9.2.4 The Copy and Paste Method Error! Bookmark not defined.
Example 9.12 Copy and Paste Error! Bookmark not defined.
Example 9.13 Copy and Paste Contents Error! Bookmark not defined.
Chapter 10: The Range Object Error! Bookmark not defined.
10.1 Range Properties Error! Bookmark not defined.
10.1.1 Formatting Error! Bookmark not defined.
Example 10.1 Formatting a Range of Cells Error! Bookmark not defined.
Example 10.2 Using ColorIndex Error! Bookmark not defined.
10.1.2 The Formula Property Error! Bookmark not defined.
Example 10.3 Using the Formula Property Error! Bookmark not defined.
10.1.3 Built-in Formulas Error! Bookmark not defined.
Example 10.4 Using the Average Formula Error! Bookmark not defined.
Example 10.5: Using the Mode Formula Error! Bookmark not defined.
Example 10.6: Using the Median Formula Error! Bookmark not defined.
Example 10.7 Using the Interior and Color Properties Error! Bookmark not defined.
10.2 Range Methods Error! Bookmark not defined.
10.2.1 The Autofill Method Error! Bookmark not defined.
Example 10.8 Using the AutoFill Method Error! Bookmark not defined.
10.2.2 Select, Copy and Paste Methods Error! Bookmark not defined.
Example 10.9 Select, Copy and Paste Error! Bookmark not defined.
10.2.3 Copy and PasteSpecial Methods Error! Bookmark not defined.
Example 10.10 Using the Pastespecial Method Error! Bookmark not defined.
Example 10.11 PasteValues and PasteFormuas MethodsError! Bookmark not defined.
10.2.4 The Find Method Error! Bookmark not defined.
Example 10.12 Search for a Name Error! Bookmark not defined.

Example 10.12 Search for a Name in a Range **Error! Bookmark not defined.**

Example 10.13 Search for a Specific Value in a Range **Error! Bookmark not defined.**

Example 10.14 Search for a Specific Value and Replace with New Value**Error! Bookmark not defined.**

Chapter 11 Excel VBA Controls **Error! Bookmark not defined.**

11.1 Check Box **Error! Bookmark not defined.**

Example 11.1 Using the Check Box **Error! Bookmark not defined.**

Example 11.2 Tracking Which Check Box(es) Was(were) Checked**Error! Bookmark not defined.**

Example 11.3 A Shopping Cart **Error! Bookmark not defined.**

11.2 Text Box **Error! Bookmark not defined.**

Example 11.4 Using the Text Box **Error! Bookmark not defined.**

11.3 Option Button **Error! Bookmark not defined.**

Example 11.5 Using the Option Buttons **Error! Bookmark not defined.**

Example 11.6 Using If…Then…Else and the Option Button**Error! Bookmark not defined.**

Example 11.7 Changing the Color of the Font **Error! Bookmark not defined.**

11.4 List Box **Error! Bookmark not defined.**

Example 11.8 Adding Items to a List Box using the AddItem Method**Error! Bookmark not defined.**

11.5 Combo Box **Error! Bookmark not defined.**

Example 11.9 Adding Items to a Combo Box **Error! Bookmark not defined.**

11.6 Toggle Button **Error! Bookmark not defined.**

Example 11.10 Using the Toggle Button **Error! Bookmark not defined.**

11.7 Spin Button **Error! Bookmark not defined.**

Example 11.11 Increase Value Using the Spin Button **Error! Bookmark not defined.**

11.8 Scrollbar **Error! Bookmark not defined.**

Example 11.12 Increase Value Using the Scrollbar **Error! Bookmark not defined.**

11.9 Slider **Error! Bookmark not defined.**

Chapter 12 Functions **Error! Bookmark not defined.**

12.1 The Concept of Functions **Error! Bookmark not defined.**

12.2 Types of Functions **Error! Bookmark not defined.**

12.2 Built-In Functions **Error! Bookmark not defined.**

Example 12.1 Generating a Sales Report **Error! Bookmark not defined.**

12.3 User-Defined Functions **Error! Bookmark not defined.**

Example 12.2 Creating the Formula to Calculate the Area of a Triangle**Error! Bookmark not defined.**

Example 12.3 Compute Grades **Error! Bookmark not defined.**

Example 12.4 Calculate Commissions **Error! Bookmark not defined.**

12.4 Passing variables by reference and by Value in a Function**Error! Bookmark not defined.**

Example 12.5 Demonstrate ByRef and ByVal **Error! Bookmark not defined.**

Chapter 13 Sub Procedures **Error! Bookmark not defined.**

Example 13.1 Create a Font Resizing Sub Procedure **Error! Bookmark not defined.**

Example 13.2 Changing the Font Size Based on the User's Input **Error! Bookmark not defined.**

Example 13.3 Change Font Size **Error! Bookmark not defined.**

Example 13.4 Show a Hidden Text **Error! Bookmark not defined.**

Example 13.5 Buy Decision Sub Procedure **Error! Bookmark not defined.**

Chapter 14 String Handling Functions **Error! Bookmark not defined.**

14.1 InStr **Error! Bookmark not defined.**

14.2. Left **Error! Bookmark not defined.**

14.3. Right **Error! Bookmark not defined.**

14.4. Mid **Error! Bookmark not defined.**

14.5. Len **Error! Bookmark not defined.**

Example 14.1 Executing Several String Functions **Error! Bookmark not defined.**

Chapter 15 Date and Time Functions **Error! Bookmark not defined.**

15.1 Using the Now ( ) Function **Error! Bookmark not defined.**

Example 15.1 Using Several Time and Date Formatting Functions**Error! Bookmark not defined.**

15.2 Date and Time Functions **Error! Bookmark not defined.**

Example 15.2 Usage of Date and Time Functions **Error! Bookmark not defined.**

15.3 DatePart Function **Error! Bookmark not defined.**

Example 15.3 Using the DatePart Function **Error! Bookmark not defined.**

15.4 Adding and Subtracting Dates **Error! Bookmark not defined.**

Example 15.4 Subtracting Years **Error! Bookmark not defined.**

Chapter 16 UseForm **Error! Bookmark not defined.**

16.1 Keyboard Events **Error! Bookmark not defined.**

Example 16.1 Testing the Keyboard **Error! Bookmark not defined.**

Example 16.2 Identify which Key was Pressed **Error! Bookmark not defined.**

16.2 Mouse Events **Error! Bookmark not defined.**

Example 16.3 MouseDown Event **Error! Bookmark not defined.**

Example 16.4 Importing Data from a Worksheet to a List Box**Error! Bookmark not defined.**

Example 16.5 Performing Calculation **Error! Bookmark not defined.**

Example 16.6 Web Browser **Error! Bookmark not defined.**

Chapter 17 Working with Files **Error! Bookmark not defined.**

17.1 Application.GetOpenFilename method **Error! Bookmark not defined.**

Example 17.1 Opening a File **Error! Bookmark not defined.**

17.2 Application.GetSaveAsFilename method **Error! Bookmark not defined.**

Example 17.2 Saving a File **Error! Bookmark not defined.**

17.3 Creating a Text File **Error! Bookmark not defined.**

Example 17.3 Creating a Text file **Error! Bookmark not defined.**

17.4 Reading a File **Error! Bookmark not defined.**

Chapter 18 Class Modules **Error! Bookmark not defined.**

18.1 Creating a Class Module **Error! Bookmark not defined.**

Example 18.1 The BMI Calculator **Error! Bookmark not defined.**

Example 18.2 Grades Calculator **Error! Bookmark not defined.**

Example 18.3 Future Value Calculator **Error! Bookmark not defined.**

18.2 Class Module Properties **Error! Bookmark not defined.**

Example 18.4 The ATM Machine **Error! Bookmark not defined.**

Example 18.5 The Decision-Making App **Error! Bookmark not defined.**

Example 18.6 A Virtual Keyboard **Error! Bookmark not defined.**

Example 18.7 Grades Calculator **Error! Bookmark not defined.**

Chapter 19 Drawing Charts **Error! Bookmark not defined.**

Chapter 20 Dealing with Shapes **Error! Bookmark not defined.**

Example 20.1 Drawing a Hexagon Shape **Error! Bookmark not defined.**

Example 20.2 Manipulating the Color and Transparency **Error! Bookmark not defined.**

Example 20.3 Drawing Shapes **Error! Bookmark not defined.**

Example 20.4 Adding Glow **Error! Bookmark not defined.**

Example 20.5 Declaring Shapes **Error! Bookmark not defined.**

Example 20.6 Creating 3-D Effect **Error! Bookmark not defined.**

Example 20.7 Adding Text to a Shape **Error! Bookmark not defined.**

Chapter 21 Interacting with Database **Error! Bookmark not defined.**

21.1 Working with Microsoft Access Database **Error! Bookmark not defined.**

Example 21.1 Importing Data from Access Database **Error! Bookmark not defined.**

L,21.2 Building a Data Entry Form **Error! Bookmark not defined.**

Example 21.2 Designing a Data Entry Form **Error! Bookmark not defined.**

Chapter 22 Printing **Error! Bookmark not defined.**

22.1 The Basic Syntax **Error! Bookmark not defined.**

22.2 Printing a Particular Worksheet **Error! Bookmark not defined.**

22.3 Printing a Specific Page Range **Error! Bookmark not defined.**

22.4 Printing Several Copies **Error! Bookmark not defined.**

Example 22.1 Printing Several Copies **Error! Bookmark not defined.**

22.5 Print Preview **Error! Bookmark not defined.**

Example 22.2 Print Pryeview **Error! Bookmark not defined.**

Example 22.3 Dialog to Let the User to Continue or Stop Printing **Error! Bookmark not defined.**

22.6 Print a Selected Range **Error! Bookmark not defined.**

Example 22.4 Print a Selected Range **Error! Bookmark not defined.**

Example 1 BMI Calculator **Error! Bookmark not defined.**

Example 2 Financial Calculator **Error! Bookmark not defined.**

Example 3 Investment Calculator **Error! Bookmark not defined.**

Example 4 Prime Number Tester **Error! Bookmark not defined.**

Example 5 Selective Summation **Error! Bookmark not defined.**

Example 6 Excel VBA 365 Windows Media Player **Error! Bookmark not defined.**

Example 7 Animation **Error! Bookmark not defined.**

Example 8 Amortization Calculator **Error! Bookmark not defined.**

Example 9 Boggle **Error! Bookmark not defined.**

Example 10 Calculator **Error! Bookmark not defined.**

Example 11 Scientific Calculator **Error! Bookmark not defined.**

Example 12 Dice **Error! Bookmark not defined.**

Example 13 Geometric Progression **Error! Bookmark not defined.**

Example 14 Password Cracker **Error! Bookmark not defined.**

Example 15 Digital Slot Machine **Error! Bookmark not defined.**

Example 16 Professional Slot Machine **Error! Bookmark not defined.**

Example 17 Quadratic Equation Solver **Error! Bookmark not defined.**

Example 18 Simple Harmonic Motion **Error! Bookmark not defined.**

Example 19 Simultaneous Equation **Error! Bookmark not defined.**

Example 20 Star War **Error! Bookmark not defined.**

Example 21 Stock Trading **Error! Bookmark not defined.**

Example 23 Payback Period Calculator **Error! Bookmark not defined.**

Example 24 Depreciation Calculator **Error! Bookmark not defined.**

Example 25 Non-Linear Simultaneous Equation Solver **Error! Bookmark not defined.**

Example 26 Pythagoras Theorem **Error! Bookmark not defined.**

Example 27 Factors Finder **Error! Bookmark not defined.**

Example 28 Loan Payments Calculator **Error! Bookmark not defined.**

Index **Error! Bookmark not defined.**

Chapter 1 Introduction to Excel VBA 365

This book is based on the latest Microsoft Excel, which is one of the apps of Microsoft Office 365; hence I named this book Excel VBA 365 Made Easy. All the Excel VBA code examples in this book have been tested in Microsoft Excel 365 and proven to be bugs free, therefore you may try them out in your own settings. Although the examples are based on MS Excel 365, they should be workable in older versions of MS Excel because the syntaxes are based largely on Visual Basic 6.

1.1 The Concept of Excel VBA

VBA stands for Visual Basic for Applications. It is an event-driven programming language Visual Basic embedded inside Microsoft Office applications like Microsoft Excel, Microsoft Word, Microsoft PowerPoint and more. By running Visual Basic within the Microsoft Office applications, we can build customized functions and macros to enhance the capabilities of those applications. Besides that, we can build VBA macros that automates processes in the Microsoft Office applications.

Among the Visual Basic applications, Microsoft Excel VBA 365 is the most popular. There are many reasons why we should learn VBA for Microsoft Excel, one of the reasons is you can understand the fundamentals of Visual Basic programming within the MS Excel environment, without having to purchase a copy of Microsoft Visual Basic software. Another reason is by learning Excel VBA; you can build custom-made functions to complement the built-in formulas and functions of Microsoft Excel.

Although MS Excel has numerous built-in formulas and functions, it is still insufficient to cater for many complex calculations and applications. This book was written in such a way that you can learn VBA for MS Excel from scratch, and everyone shall be able to master it in a short time! Basically, Excel VBA code is created using Visual Basic, therefore, its syntaxes remain largely the same for every version of Microsoft Excel. Although this book is based on MS Excel 365, you may apply it in older versions of MS Excel.

1.2 The Visual Basic Editor in MS Excel 365

To create VBA applications in Microsoft Excel 365, you must own a copy of Microsoft office 365 that comes with the basic package comprising Microsoft Word, Microsoft PowerPoints, Microsoft Excel, Microsoft Access and more. If you have already owned a copy of Microsoft Office 365, proceed to program Excel VBA by launching Microsoft Excel 365. Figure 1.1 shows the initial Workbook of Microsoft Excel 365.

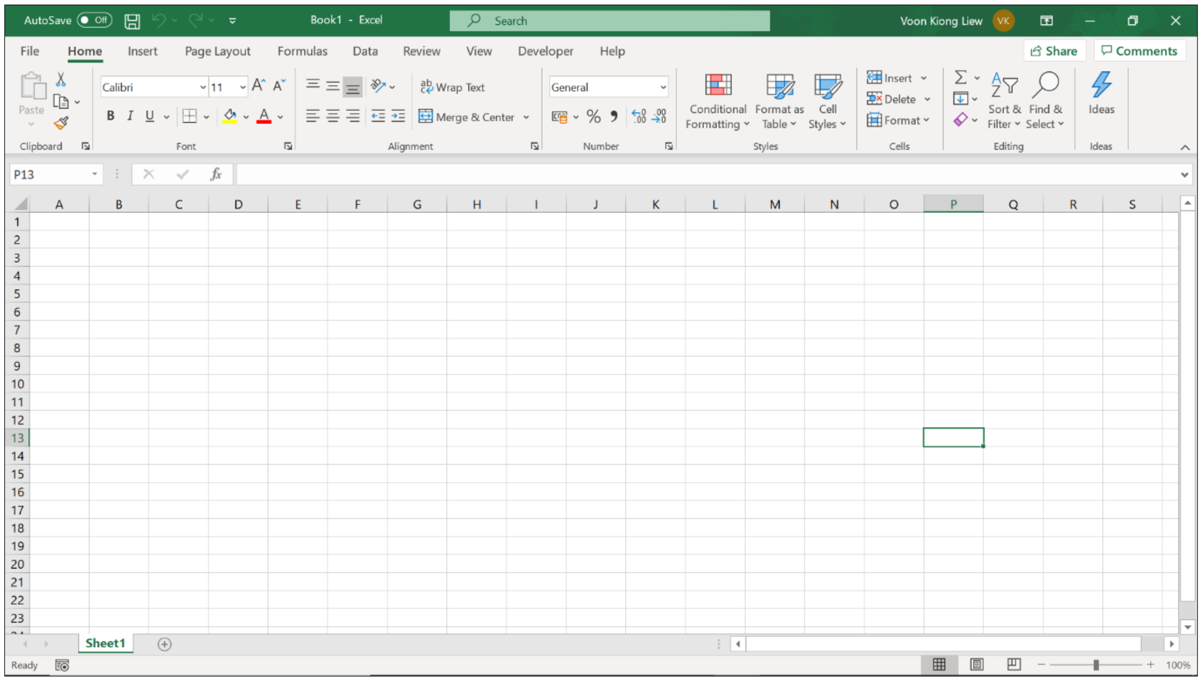


Figure 1.1 Microsoft Excel 365 workbook

Next, click on the Developer tab to access the Developer window, the environment for building Excel 365 Visual Basic applications. In the Developer environment, you may play with all kinds of tools and apps that you can use to develop VBA and macros.

There are three ways to start programming Excel VBA, by placing controls on the worksheet and double click it to enter the Visual Basic Editor. The second way is to enter the Visual Basic Editor directly by clicking the View Code button or the Visual Basic button in the Developer environment. In addition, you can also program VBA by creating macros.

1.2.1 Building Excel VBA 365 using the Controls.

There are two categories of controls, Form controls and ActiveX controls. Form controls are built into Excel whereas ActiveX controls are loaded separately. Though Form controls are simpler to use, ActiveX controls allow for more flexible design.

To use the controls, navigate to the Developer tab then click on the Insert button to access the ActiveX controls and Form Controls, as shown in Figure 1.2.

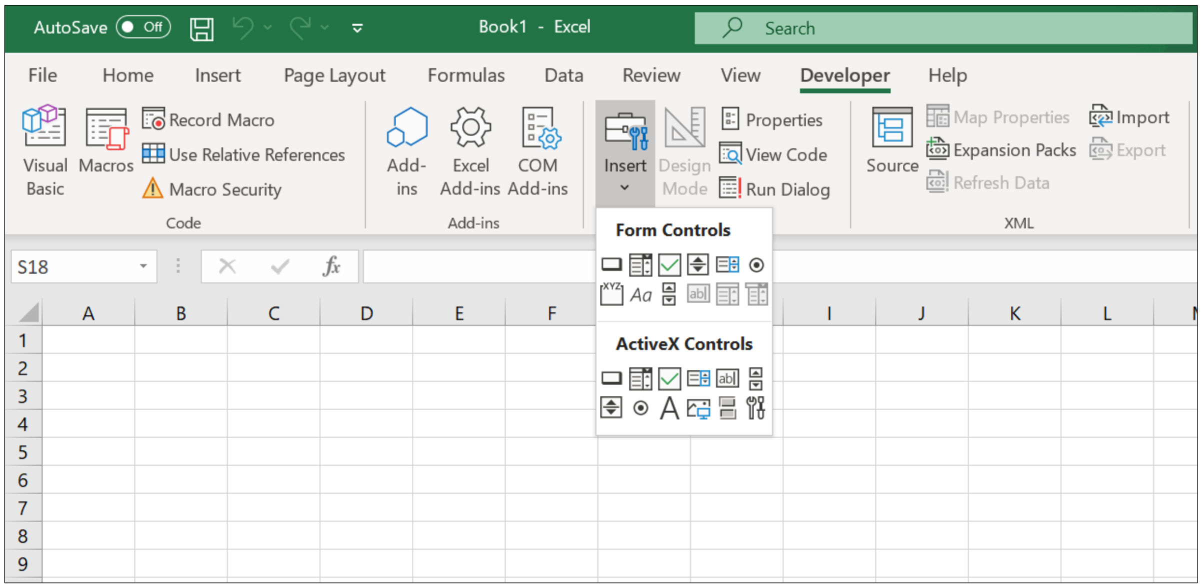
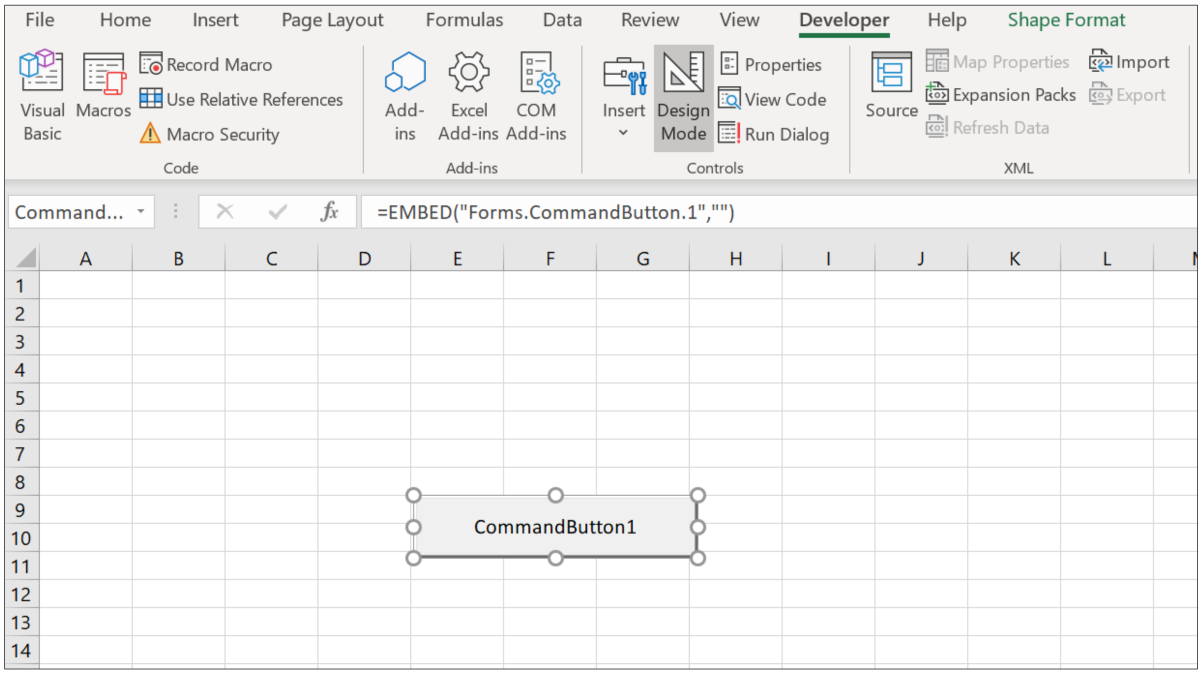


Figure 1.2 **Form and ActiveX Controls**

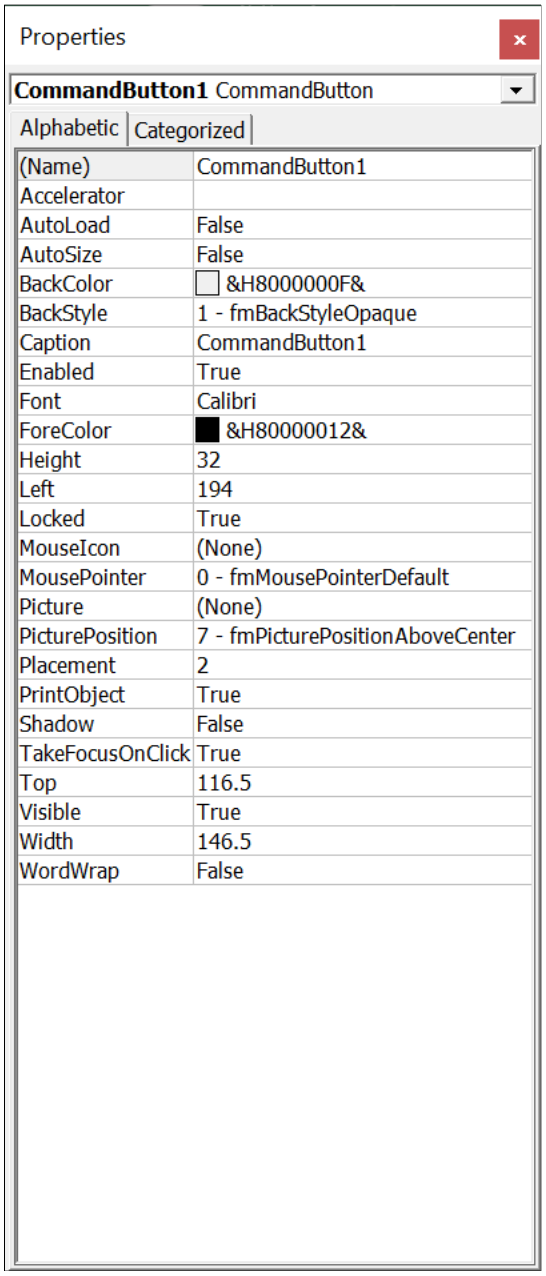
Let us start with the command button. To place a command button on the MS Excel worksheet, click on the command button under ActiveX controls and draw it on the worksheet, as shown in Figure 1.3. Notice that the Developer environment is in the Design Mode at this stage.



=EMBED("Forms.CommandButton.1","")

Figure 1.3 **The Command Button in the Design Mode**

At this stage, you might want to customize the command button by changing some of its properties. To access the properties, right-click the command button and select the Properties option to launch the Properties window, as show in Figure 1.4.

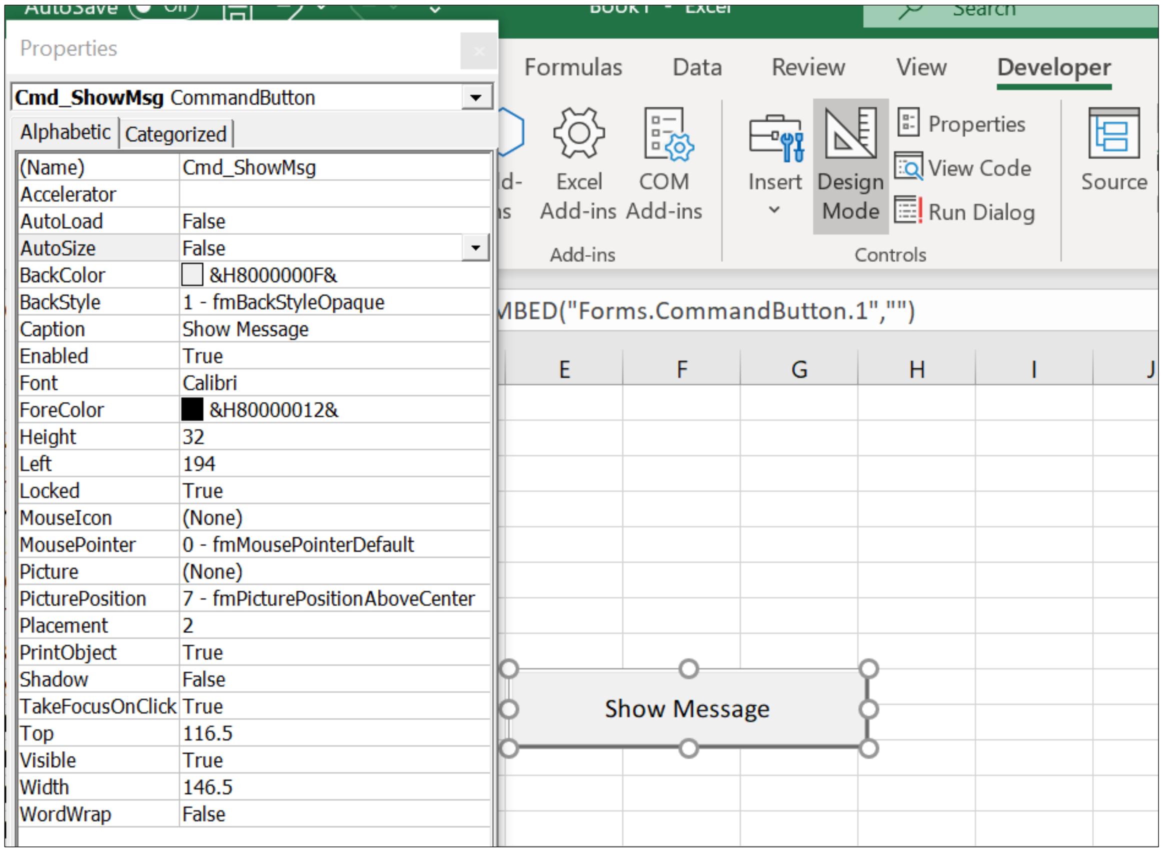


(Name)	CommandButton1
Accelerator	
AutoLoad	False
AutoSize	False
BackColor	☐ &H8000000F&
BackStyle	1 - fmBackStyleOpaque
Caption	CommandButton1
Enabled	True
Font	Calibri
ForeColor	■ &H80000012&
Height	32
Left	194
Locked	True
MouseIcon	(None)
MousePointer	0 - fmMousePointerDefault
Picture	(None)
PicturePosition	7 - fmPicturePositionAboveCenter
Placement	2
PrintObject	True
Shadow	False
TakeFocusOnClick	True
Top	116.5
Visible	True
Width	146.5
WordWrap	False

Figure 1.4 The Properties Window

You may change its name to any name you wish but for learning purposes I suggest you change its name to Cmd_ShowMsg and its Caption to Show Message, as shown in Figure 1.5.

Notice that the caption on the command button has changed to Show Message.



MBED("Forms.CommandButton.1","")

Cmd_ShowMsg CommandButton

Alphabetic | Categorized

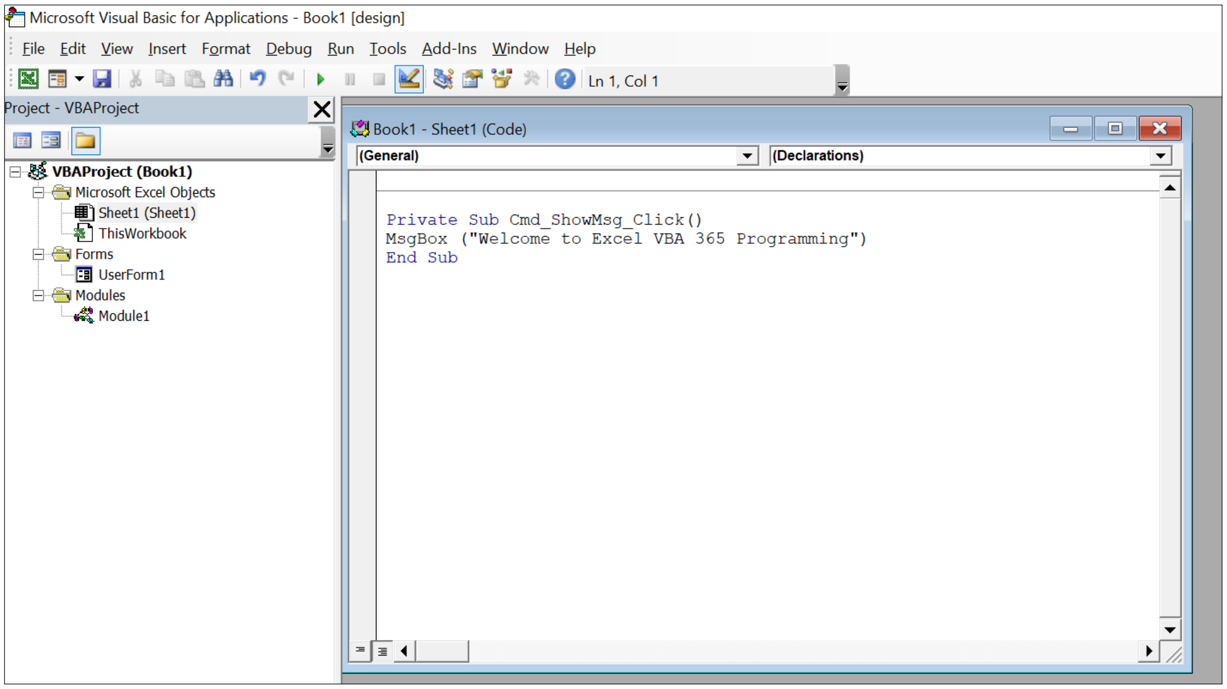
(Name)	Cmd_ShowMsg
Accelerator	
AutoLoad	False
AutoSize	False
BackColor	⬜ &H8000000F&
BackStyle	1 - fmBackStyleOpaque
Caption	Show Message
Enabled	True
Font	Calibri
ForeColor	⬛ &H80000012&
Height	32
Left	194
Locked	True
MouseIcon	(None)
MousePointer	0 - fmMousePointerDefault
Picture	(None)
PicturePosition	7 - fmPicturePositionAboveCenter
Placement	2
PrintObject	True
Shadow	False
TakeFocusOnClick	True
Top	116.5
Visible	True
Width	146.5
WordWrap	False

Figure 1.5

Next, click on the command button to enter the Visual Basic Editor (We will use the short form VBE every now and then in the book). In the VBE, Enter the statements as shown in Example 1.1, as follows:

Example 1.1 Displaying a Message

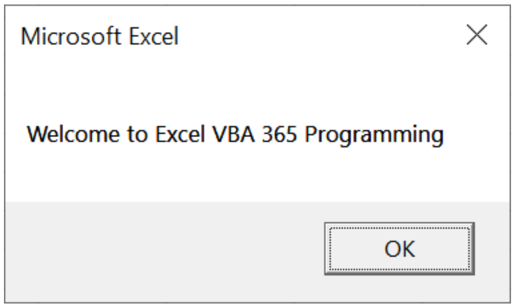
```
Private Sub Cmd_ShowMsg_Click()
MsgBox ("Welcome to Excel VBA 365 Programming")
End Sub
```



```
Private Sub Cmd_ShowMsg_Click()
MsgBox ("Welcome to Excel VBA 365 Programming")
End Sub
```

Figure 1.6 The Visual Basic Editor

To run the VBA program, quit the VBE and the Design Mode and then click on the command button. A message box will appear, as shown in Figure 1.7



Welcome to Excel VBA 365 Programming

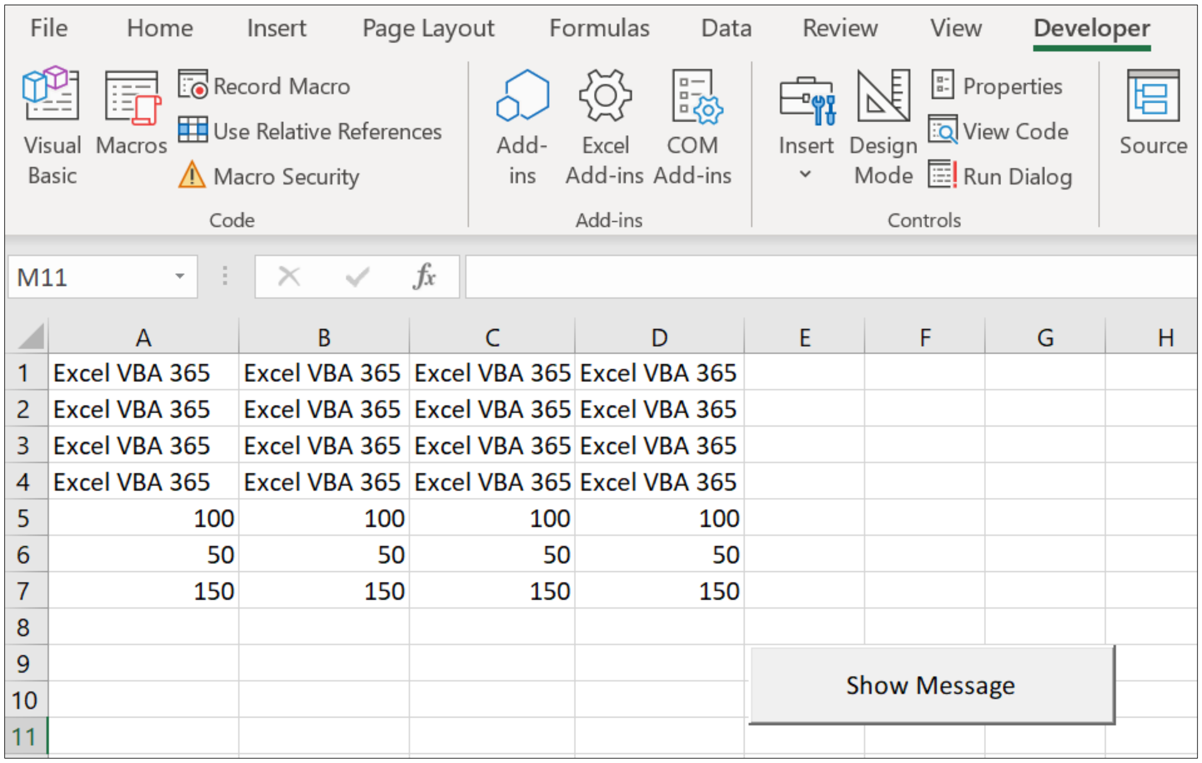
Figure 1.7

The next example involved the use of the Range object and its property Value, as well as the cells object. The program also introduces a `For...Next` loop which you are already familiar if you have been programming in Visual Basic 6.

Example 1.2 Populates Cells with Text and Values

```
Private Sub Cmd_Compute_Click()
 Range("A1:D4").Value = "Excel VBA 365 "
 Range("A5:D5").Value = 100
 Range("A6:D6").Value = 50
For i = 1 To 4
 Cells(7, i) = Cells(5, i) + Cells(6, i)
Next
End Sub
```

The first statement will populate the cells from the range cell A1 to cell D4 with the phrase "Excel VBA 365". The second statement populates the cells from the range cell A5 to cell D5 with the value of 100. The third statement populates the cells from the range cell A6 to cell D6 with the value of 50. The `For…Loop` statement adds the corresponding values of row 5 and row 6 and display them in row 7. Running the VBA produces the output UI as shown in Figure 1.8.



	A	B	C	D	E	F	G	H
1	Excel VBA 365	Excel VBA 365	Excel VBA 365	Excel VBA 365				
2	Excel VBA 365	Excel VBA 365	Excel VBA 365	Excel VBA 365				
3	Excel VBA 365	Excel VBA 365	Excel VBA 365	Excel VBA 365				
4	Excel VBA 365	Excel VBA 365	Excel VBA 365	Excel VBA 365				
5	100	100	100	100				
6	50	50	50	50				
7	150	150	150	150				
8								
9								
10								
11								

Figure 1.8

1.2.2 Building Excel VBA 365 using the Visual Basic Editor

To access Visual Basic Editor directly, click on Visual Basic or View Code in the Developer environment. In the VBE, you are presented with two items, General and Worksheet. General is the declaration section when you can declare some global variables. Worksheet is the object where you can write some VBA code to interact with it. The current active worksheet is sheet1(the name assigned to Worksheet1) as only one worksheet is available, as seen on the right section of the VBE, as shown in Figure 1.9.

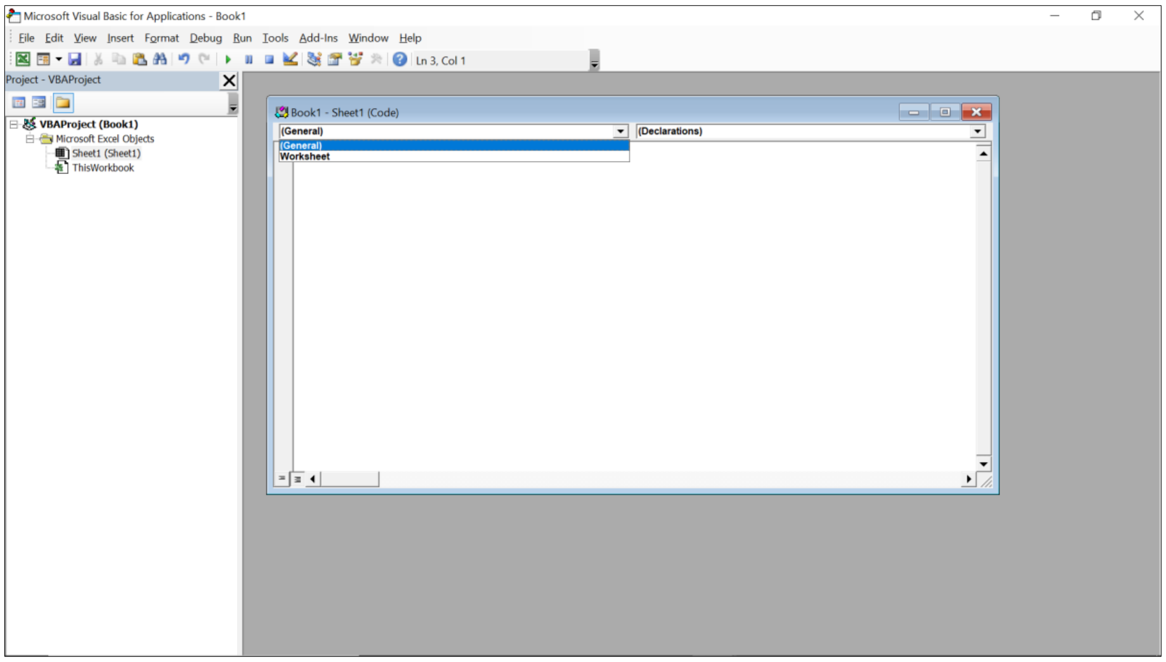


Figure 1.9 The Visual Basic Editor

If you add another worksheet to the workbook, the VBE will shows two worksheets, sheet1 and sheet2, as shown in Figure 1.10

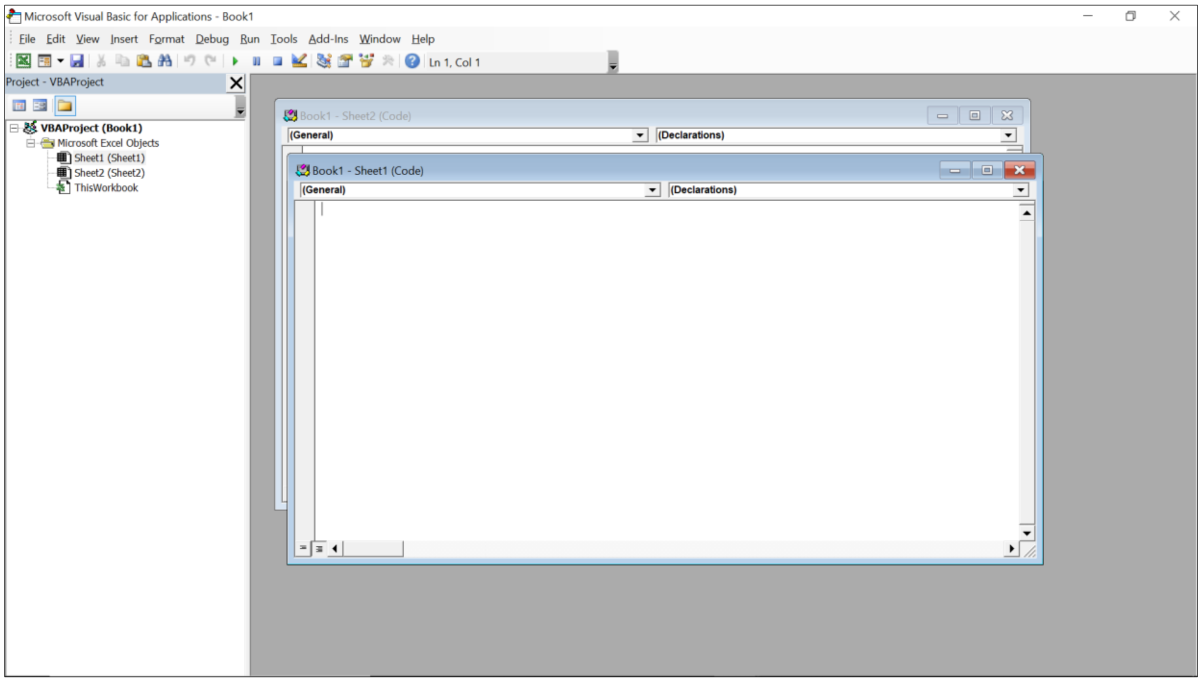
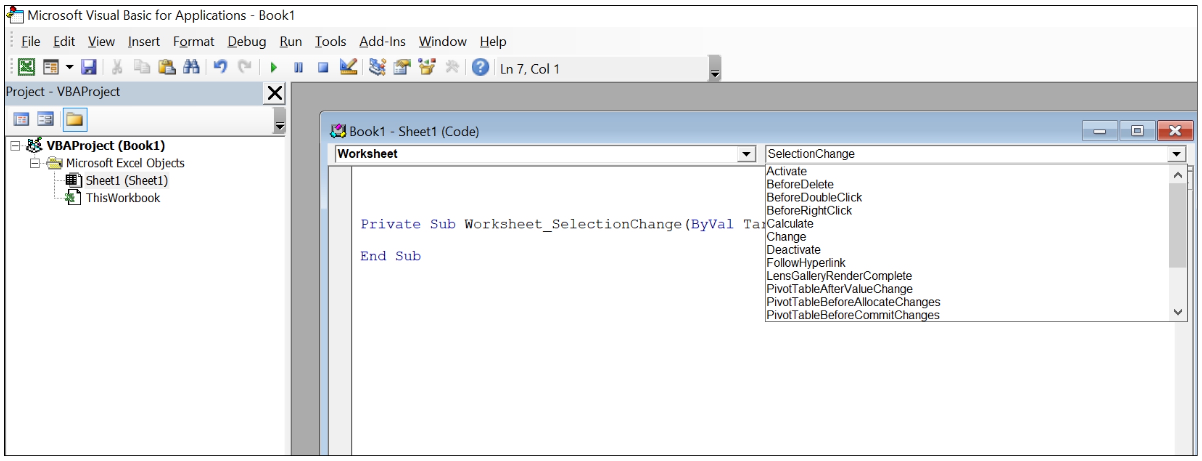


Figure 1.10

When you click the Worksheet, an event procedure will appear, as shown below:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

End Sub
```

A worksheet has many events associated with it (for that matter any Excel VBA objects has events associated with them). The default event is `SelectionChange` , as shown in the event procedure above. To view more events associated with the WorkSheet, click on the small inverted triangle on the top right corner of VBE, you will see a drop-down list of events, as shown in Figure 1.11.



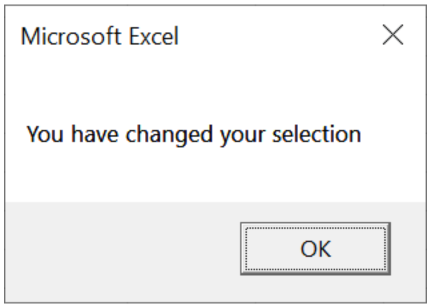
```
Private Sub Worksheet_SelectionChange(ByVal Ta

End Sub
```

Figure 1.11 **The Worksheet Events**

Now let us enter some code into the event procedure, as follows:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
 MsgBox ("You have changed your selection")
 End Sub
```

This code means whenever you click on another cell of the Worksheet, the message " You have changed your selection" message will appear, as shown in Figure 1.12.



You have changed your selection

Figure 1.12

You should proceed to save your Excel Workbook before your work is lost. Remember to save your file with the extension xlsm, which means Excel Macro Enabled Workbook, otherwise your VBA will not run when you open it the next time.

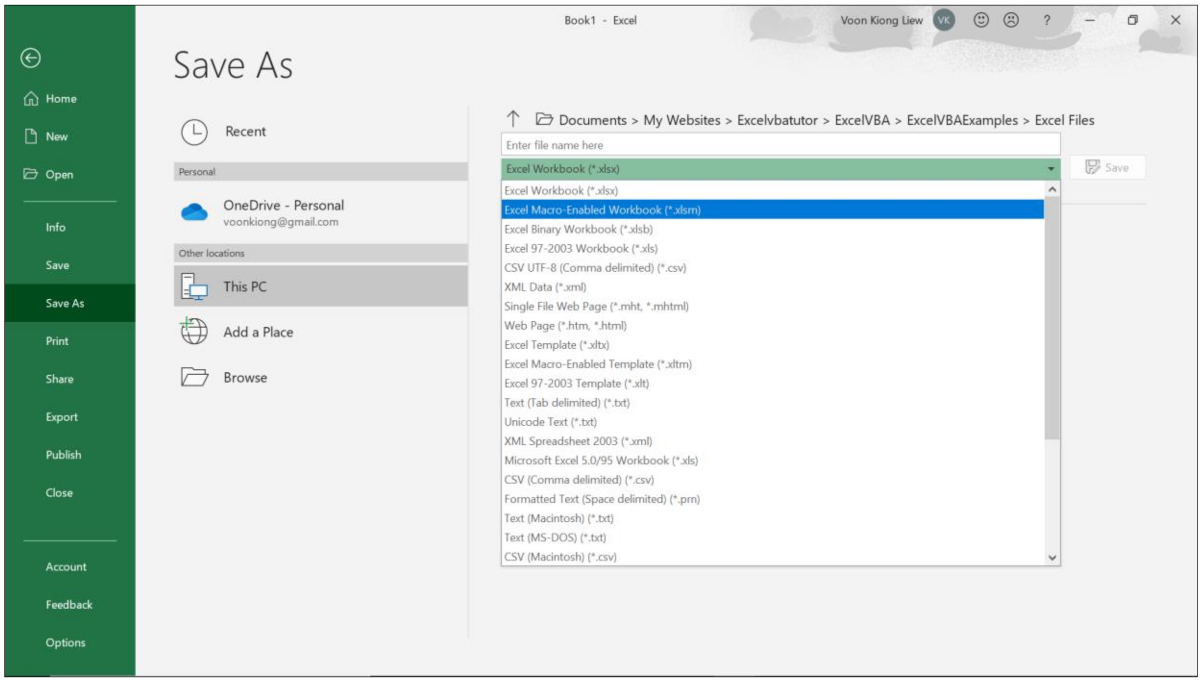

Book1 - Excel
Voon Kiong Liew
Home
New
Open
Info
Save
Save As
Print
Share
Export
Publish
Close
Account
Feedback
Options
Save As
Recent
Personal
OneDrive - Personal
voonkiong@gmail.com
Other locations
This PC
Add a Place
Browse
Documents > My Websites > Excelvbatutor > ExcelVBA > ExcelVBAExamples > Excel Files
Enter file name here
Excel Workbook (*.xlsx)
Save
Excel Workbook (*.xlsx)
Excel Macro-Enabled Workbook (*.xlsm)
Excel Binary Workbook (*.xlsb)
Excel 97-2003 Workbook (*.xls)
CSV UTF-8 (Comma delimited) (*.csv)
XML Data (*.xml)
Single File Web Page (*.mht, *.mhtml)
Web Page (*.htm, *.html)
Excel Template (*.xltx)
Excel Macro-Enabled Template (*.xltm)
Excel 97-2003 Template (*.xlt)
Text (Tab delimited) (*.txt)
Unicode Text (*.txt)
XML Spreadsheet 2003 (*.xml)
Microsoft Excel 5.0/95 Workbook (*.xls)
CSV (Comma delimited) (*.csv)
Formatted Text (Space delimited) (*.prn)
Text (Macintosh) (*.txt)
Text (MS-DOS) (*.txt)
CSV (Macintosh) (*.csv)

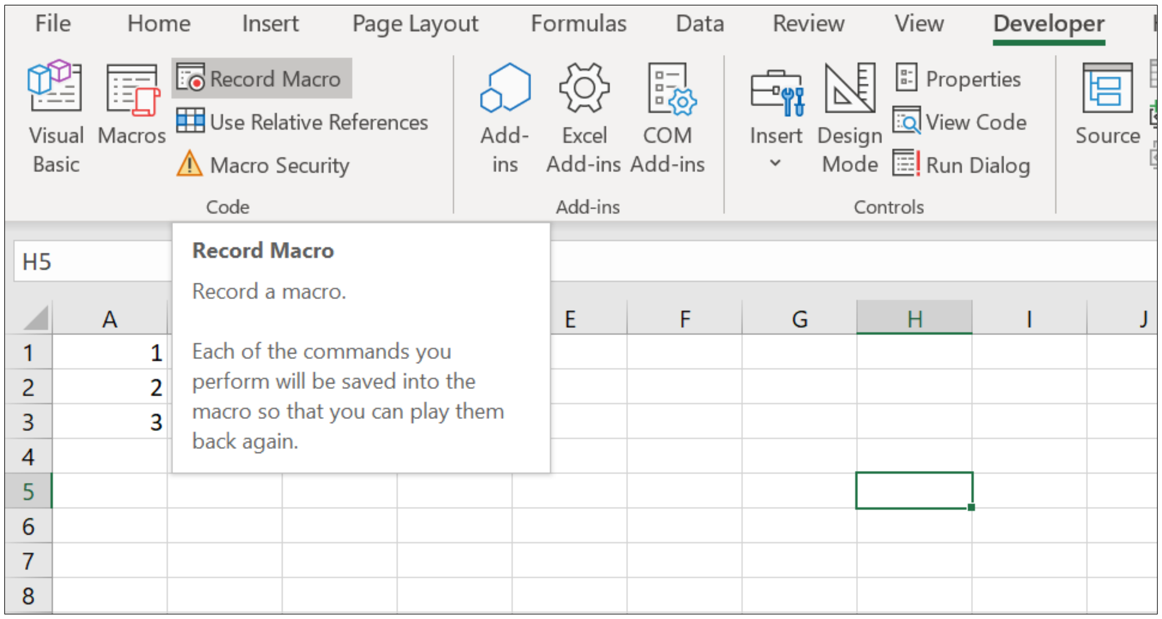
Figure 1.13 **Saving File with Extension xlsm**

In addition, Visual Basic Editor also allow you to insert modules and UserForms to build more advance VBA. Usually the module allows you to develop customized functions whereas the UserForm allows you to build more powerful applications. We will discuss module and UserForm in a later chapter.

1.2.3 Creating Macros

You can also learn Excel VBA 365 programming by creating and editing macros. Macro is a record and playback tool that records and plays back Excel worksheet activities performed by the user. Macros save time as they automate repetitive tasks. It is a programming code that runs in an Excel VBA environment. You can edit a macro as well as creating new macros using Visual Basic syntaxes.

To record a macro, click on the Record Macro button in the Developer environment, as shown in Figure 1.14



Record Macro

Record a macro.

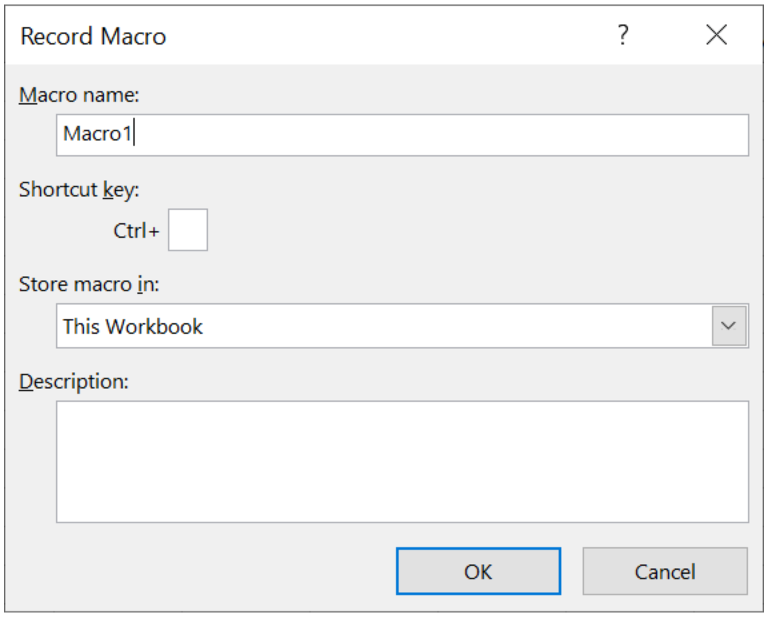
Each of the commands you perform will be saved into the macro so that you can play them back again.

	A	E	F	G	H	I	J
1	1						
2	2						
3	3						
4							
5							
6							
7							
8							

Figure 1.14

Upon clicking the Record Macro button, a dialog box will appear and prompts you to enter the macro name. The macro name cannot have space between characters, underscore is allowed. Following are a few rules in naming a macro:

- Must start with a letter or underscore
- Space is not allowed
- Does not conflict with existing names in the workbook


Record Macro

Macro name:

Macro1

Shortcut key:

Ctrl+

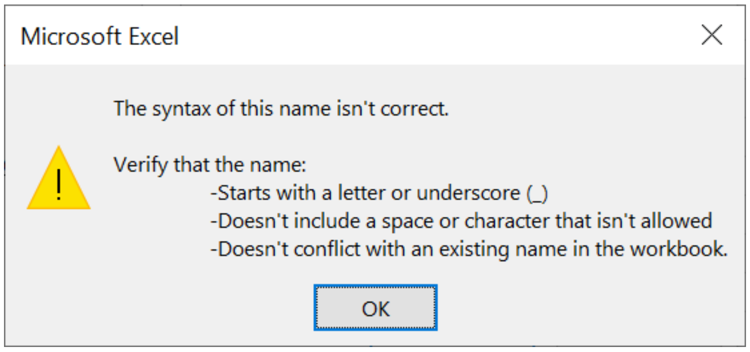
Store macro in:

This Workbook

Description:

Figure 1.15

If you did not follow the rules, the dialog as shown in Figure 1.16 will appear.


Microsoft Excel

The syntax of this name isn't correct.

Verify that the name:
-Starts with a letter or underscore (_)
-Doesn't include a space or character that isn't allowed
-Doesn't conflict with an existing name in the workbook.

Figure 1.16

Let us create a macro named Test_Macro. Next, click OK to start recoding the macro. Perform some activities on the worksheet like entering some numbers and add those numbers, then stop the macro recording.

To view the macro you have just created, click the Macros button and you can see the newly created macro as shown in Figure 1.17. You can run, edit or delete the macro.

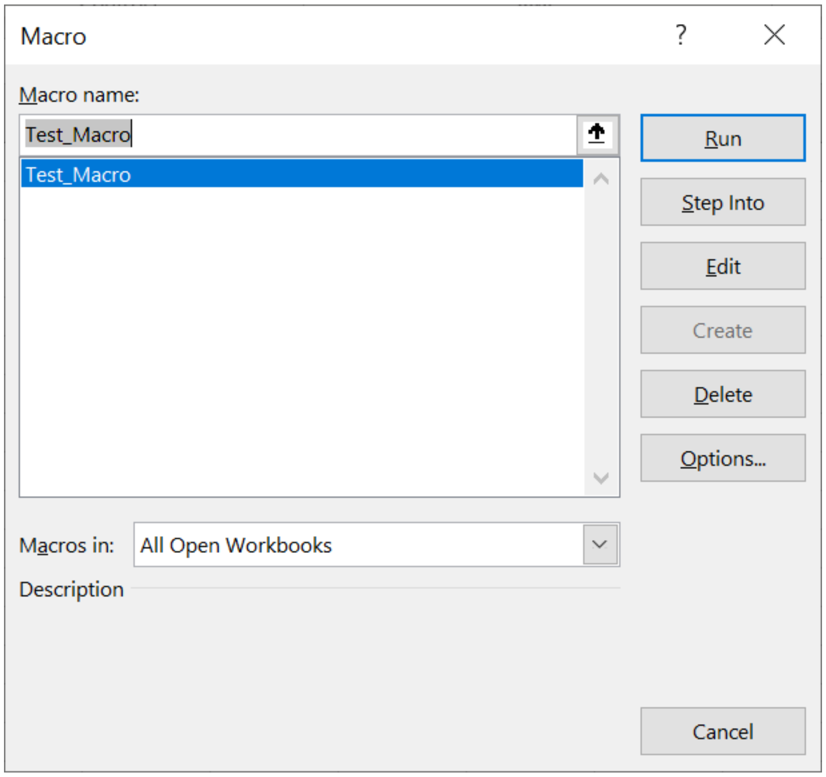
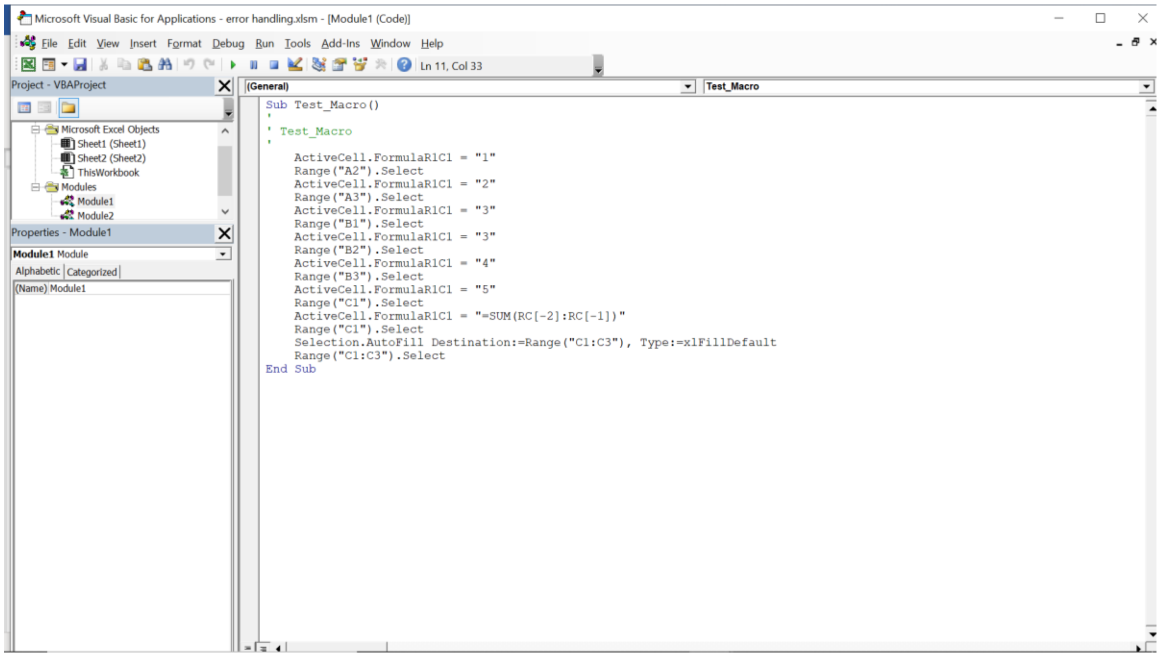


Figure 1.17

Let us edit the macro. When you click on the Edit button, you will be able to see the macro code in the VBE, as shown in Figure 1.18. The code is the same as the code in a VB sub procedure which starts with a Sub keyword and an End Sub keyword.



```
Sub Test_Macro()
'
' Test_Macro
'
    ActiveCell.FormulaR1C1 = "1"
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "2"
    Range("A3").Select
    ActiveCell.FormulaR1C1 = "3"
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "3"
    Range("B2").Select
    ActiveCell.FormulaR1C1 = "4"
    Range("B3").Select
    ActiveCell.FormulaR1C1 = "5"
    Range("C1").Select
    ActiveCell.FormulaR1C1 = "=SUM(RC[-2]:RC[-1])"
    Range("C1").Select
    Selection.AutoFill Destination:=Range("C1:C3"), Type:=xlFillDefault
    Range("C1:C3").Select
End Sub
```

Figure 1.18

Example 1.3 Creating a Macro

Let us create a macro from scratch instead of recoding a macro. To create a macro, click on Macros button and type in a name, as shown in Figure 1.19.

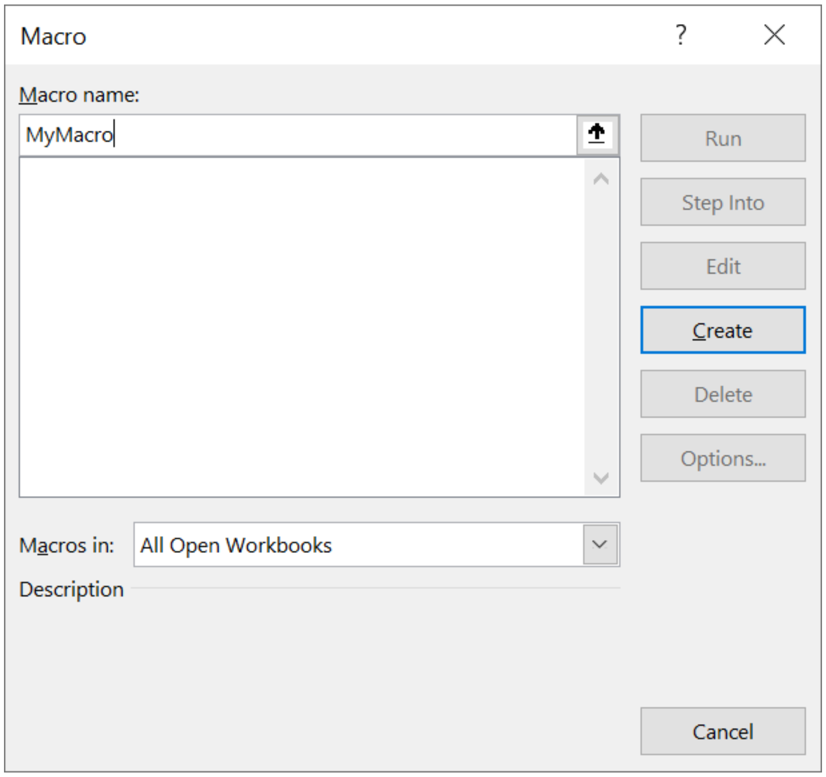
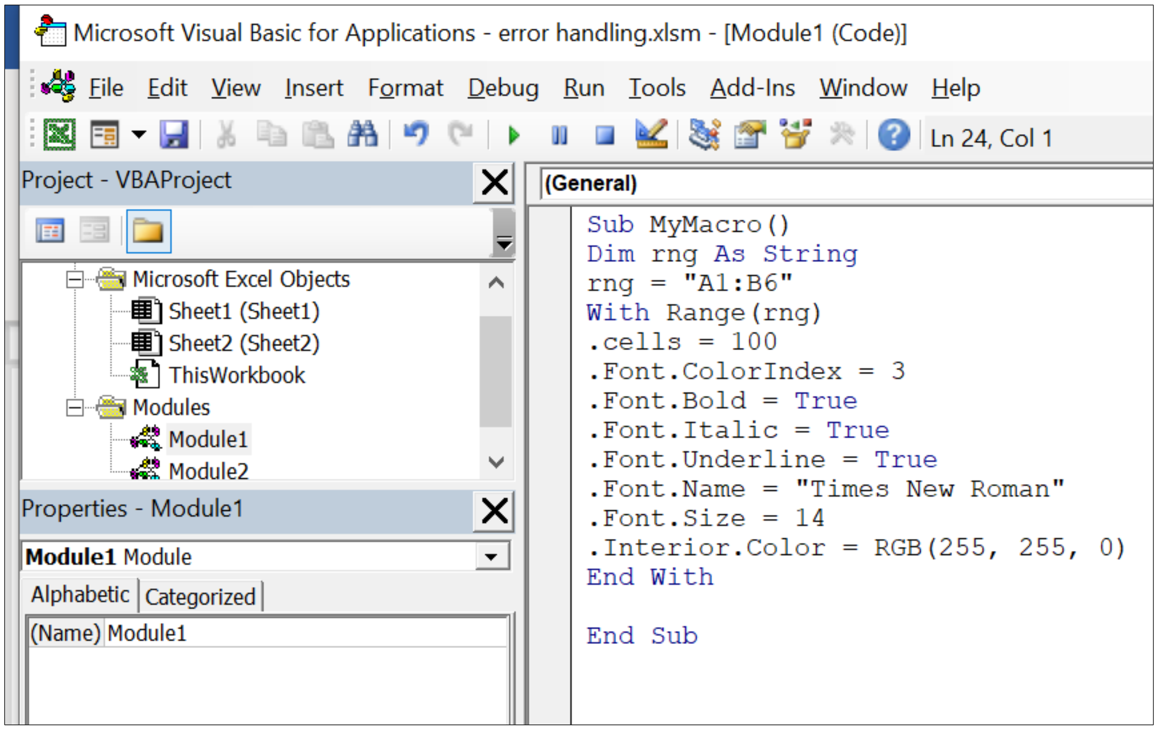


Figure 1.19

Click create to enter the VBE, and type come codes as shown in Figure 1.20.



```
Sub MyMacro()
Dim rng As String
rng = "A1:B6"
With Range(rng)
.cells = 100
.Font.ColorIndex = 3
.Font.Bold = True
.Font.Italic = True
.Font.Underline = True
.Font.Name = "Times New Roman"
.Font.Size = 14
.Interior.Color = RGB(255, 255, 0)
End With

End Sub
```

Figure 1.20

The macro code is using the VB syntaxes. In the macro, we declared a variable rng to store the range value. By using the keyword With and the Range method, the macro formats the targeted range of cells using the font properties, the interior object and the color property. The macro will also populate the cells with a value of 100. We shall learn more about object and properties in later lessons. Save the macro and then click the run (a little green triangle on the tool bar) button or press F5 to run the macro. A dialog will appear prompting you to run, edit or delete the macro as shown in Figure 1.21.

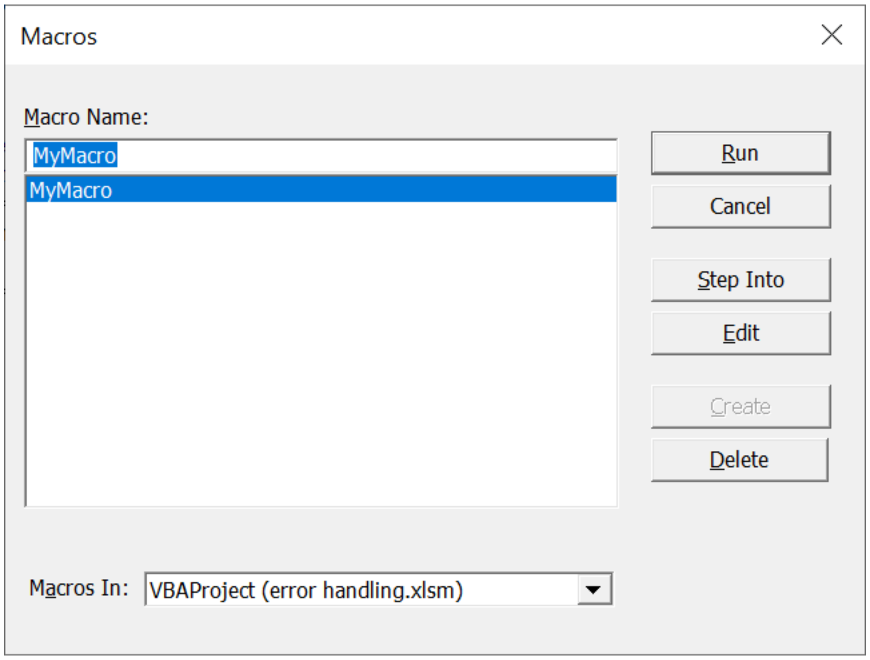
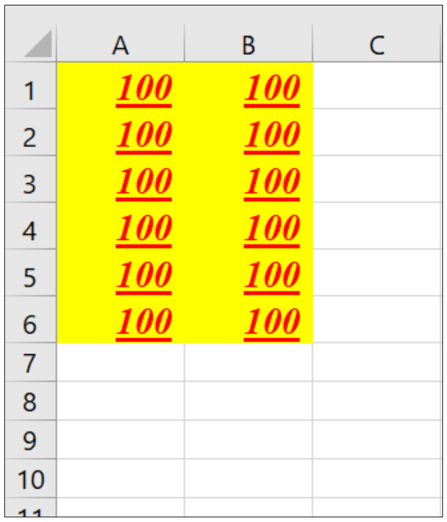


Figure 1.21

Choose Run to execute the macro and you go back to the worksheet to see the results, as shown in Figure 1.22



	A	B	C
1	100	100	
2	100	100	
3	100	100	
4	100	100	
5	100	100	
6	100	100	
7			
8			
9			
10			

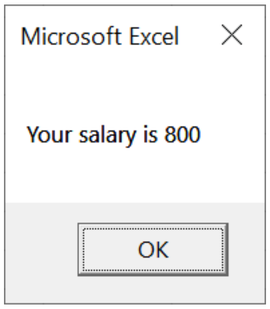
Figure 1.22

Example 1.4 Creating a Salary Calculator

This is a macro that calculates the salary based on the wage and hours worked.

```
Sub Cal_Salary() 'macro name
 salary 8, 100
End Sub

Sub salary(wage As Single, hours As Single) 'sub procedure
 MsgBox "Your salary is " & wage * hours

End Sub
```

When you run the macro, it will call the sub procedure with two arguments, i.e. wage and hours. The values specified by the macro will be passed to the sub procedure to compute the salary. The output is as shown in Figure 1.23


Microsoft Excel

Your salary is 800

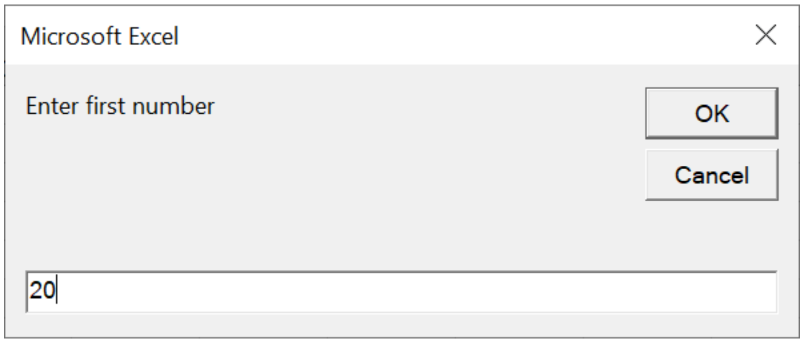
Figure 1.23

Example 1.5 Creating the Macro that Add Two Numbers

This macro adds two numbers input by the user via input boxes and present the sum in a message dialog.

```
Sub Cal_Sum()
Dim x As Single, y As Single
  x = InputBox("Enter first number")
  y = InputBox("Enter second number")
  sum x, y
End Sub
Sub sum(a As Single, b As Single)
  MsgBox ("sum=" & a + b)
End Sub
```

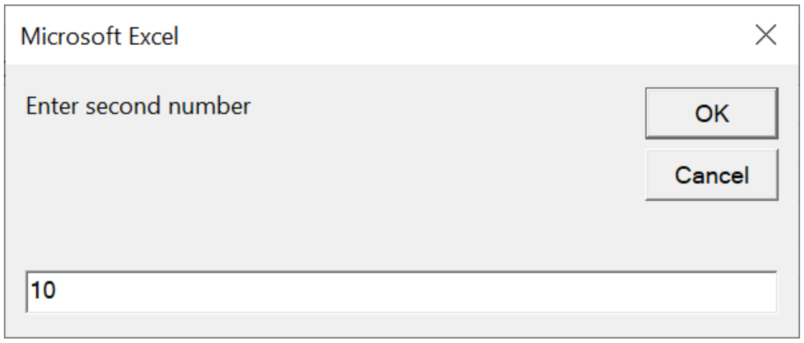
The outputs are shown in Figure 1.24, Figure 1.25 and Figure 1.26


Microsoft Excel

Enter first number

20

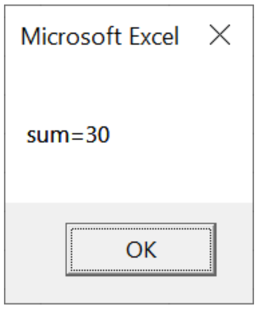
Figure 1.24


Microsoft Excel

Enter second number

10

Figure 1.25


Microsoft Excel

sum=30

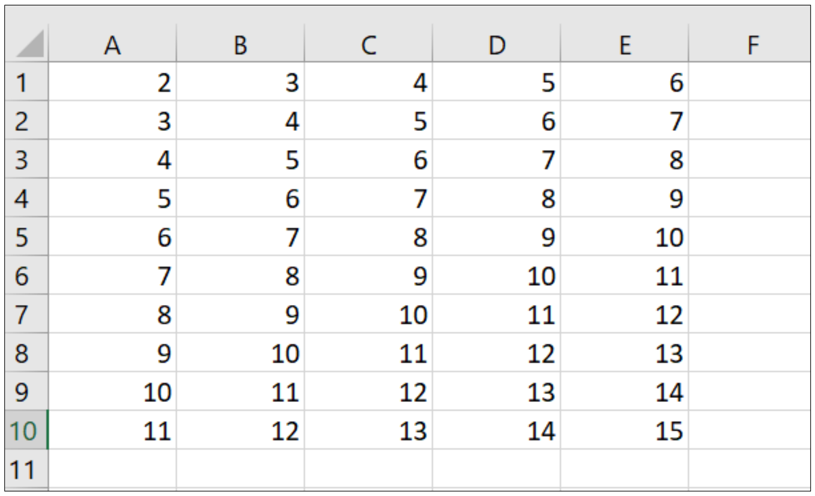
Figure 1.26

Example 1.6 A Macro that Populates Cells using the For…Next Loop

This macro employs a For…Next loop to populate a range of cells on the worksheet. (you will learn more about For…Next loop in a later chapter.)

```
Sub Loop_macro()
Dim i, j As Integer
For i = 1 To 10
For j = 1 To 5
 Cells(i, j).Value = i + j
Next
Next
End Sub
```

When you run the program, the cells in the range A1:E10 will be populated, as shown in Figure 1.27



	A	B	C	D	E	F
1	2	3	4	5	6	
2	3	4	5	6	7	
3	4	5	6	7	8	
4	5	6	7	8	9	
5	6	7	8	9	10	
6	7	8	9	10	11	
7	8	9	10	11	12	
8	9	10	11	12	13	
9	10	11	12	13	14	
10	11	12	13	14	15	
11						

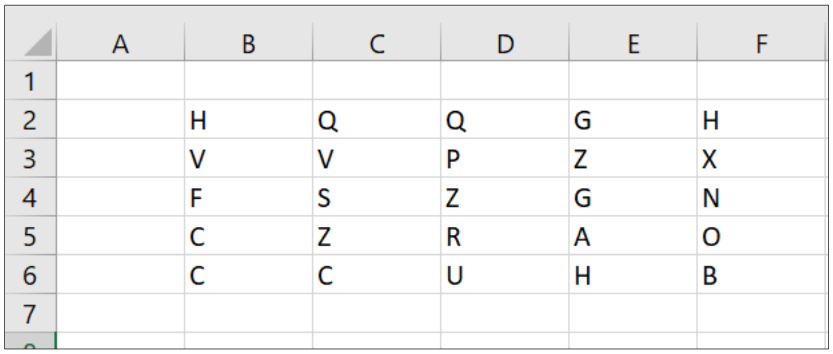
Figure 1.27

Example 1.7 A Macro that Populates the Cells with Characters using the Chr() Function.

This macro generates random characters based on the Chr() function and the ASCII codes.

```
Sub Random_Chr()
Dim m As Integer
For i = 2 To 6
For j = 2 To 6
m = Int(26 * Rnd) + 65
Sheet1.Cells(i, j) = Chr(m)
Next
Next
End Sub
```

The output is as shown in Figure 1.28



	A	B	C	D	E	F
1						
2		H	Q	Q	G	H
3		V	V	P	Z	X
4		F	S	Z	G	N
5		C	Z	R	A	O
6		C	C	U	H	B
7						

Figure 1.28

1.3 The Excel VBA 365 Code

Writing Excel VBA 365 code is like writing code in Visual Basic 6, which means you can use syntaxes like that of Visual Basic 6. However, there are some syntaxes specifically reserved for MS Excel, like the object called `Range`. `Range` is the object that specifies the value of a cell or a range of cells in MS Excel worksheet. The syntax of `Range` is as follows:

```
Range("cell Name").Value=K
```

or

```
Range("Range of Cells").Value=K
```

Value is the property of the Range object and k is a numeric value or a string.

Example 1.8 Populating a Cell using the Value Property of Range

```
Private Sub CommandButton1_Click ()
Range ("A1").Value= "Excel VBA 365"
End Sub
```

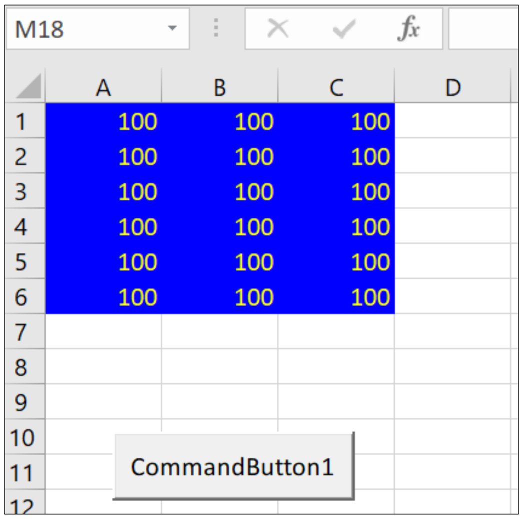
Running the code will fill cell A1 with the text “Excel VBA”. You can also use Range without the Value property, as shown in Example 1.3.

Example 1.9 Coloring the Cells with the Color Property

In this example, clicking the command button will fill cell A1 to C6 with the value of 100, change its background color to blue and its font color to yellow.

```
Private Sub CommandButton1_Click ()
Range("A1:C6")=100
Range("A1:C6").Interior.Color = vbBlue
Range("A1:C6").Font.Color = vbYellow
End Sub
```

The output



	A	B	C	D
1	100	100	100	
2	100	100	100	
3	100	100	100	
4	100	100	100	
5	100	100	100	
6	100	100	100	

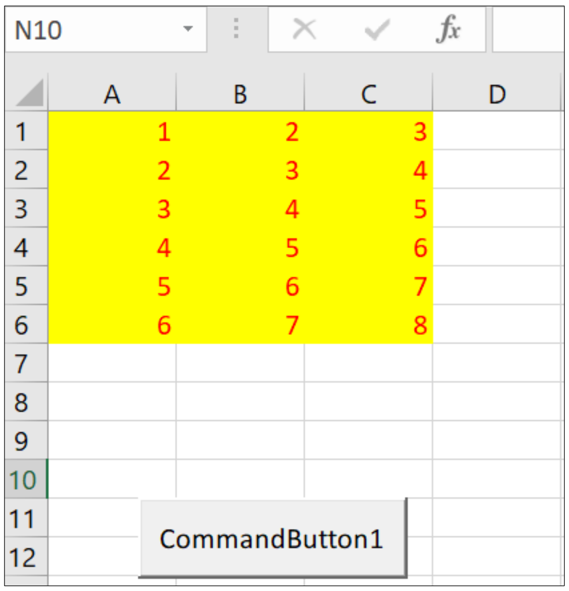
Figure 1.14

Example 1.10 Adding Numbers Using the Do... Loop

This example apply the `Do Loop` to populate `cells(1,1)` to `cells(6,3)` with numbers that follow the formula specified in the code. For example, when i=2, the value of `cells(2,2)` is 2+2=4. On top of that, it also set the background for the specified range to yellow and the font color to red.

```
Private Sub CommandButton1_Click()
i = 1
Do
Cells(i, 1) = i
Cells(i, 2) = i + 1
Cells(i, 3) = i + 2
i = i + 1
Loop Until i > 6
Range("A1:C6").Interior.Color = vbYellow
Range(Cells(1, 1), Cells(6, 3)).Font.Color = vbRed

End Sub
```

The output is as shown in Figure 1.15



	A	B	C	D
1	1	2	3	
2	2	3	4	
3	3	4	5	
4	4	5	6	
5	5	6	7	
6	6	7	8	
7				
8				
9				
10				
11				
12				

Figure 1.15

Example 1.11 A Macro that Accepts Inputs and Add Numbers

This is a macro that accepts inputs from the user and calculate the sum. When you run the macro, the user will be prompted to enter two numbers via two input boxes, then sum them up.

```
Sub Cal_Sum()
Dim x As Single, y As Single
  x = InputBox("Enter first number")
  y = InputBox("Enter second number")
  sum x, y
End Sub


Sub sum(a As Single, b As Single)
  MsgBox ("sum=" & a + b)
End Sub
```

When you run the macro, two input boxes will appear to alert the user to enter two numbers, then present the answer via a dialog message, as shown in Figure 1.16, Figure 1.17 and Figure 1.18.

1.4 Errors Handling

Errors handling is an integral part of coding in Excel VBA 365. Errors often occur when the user enter incorrect values into a cell of an Excel worksheet. For example, an error occurs when instruct the computer to divide a number by zero.

Another example is the user might enter a text (string) to a box that is designed to handle only numeric values, the computer will not be able to perform an arithmetic calculation for text, therefore, will create an error. These errors are known as synchronous errors.

Writing errors handling code should be considered a good practice for Excel VBA 365 programmers, so do not try to finish a program fast by omitting the errors handling code. However, there should not be too many errors handling code in the program as it creates problems for the programmer to maintain and troubleshoot the program later. Fortunately, we can write Excel VBA 365 code to handle those errors efficiently.

1.4.1 Writing the Errors Handling Code

The syntax for errors handling is

```
On Error GoTo program_label
```

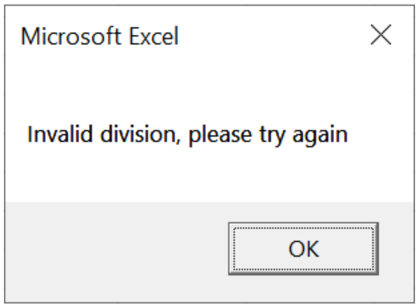
where program_label is the section of code that is designed by the programmer to handle the error committed by the user. Once an error is detected, the program will jump to the program_label section for error handling. You also need to add the statement Exit Sub to prevent the program from jumping to error handling section even though the inputs were correct.

Example 1.12 Catching Error for Invalid Division

```
Private Sub CommandButton1_Click()
On Error GoTo err_handler
 num1 = InputBox("Enter first number")
 num2 = InputBox("Enter second number")
 MsgBox num1 / num2
Exit Sub

err_handler:
 MsgBox "Invalid division, please try again"
```

```
End Sub
```

The program will display the error message “Invalid division, please try again” if the user enters letters instead of numbers or enter the second number as zero, as shown in Figure 1.16


Microsoft Excel

Invalid division, please try again

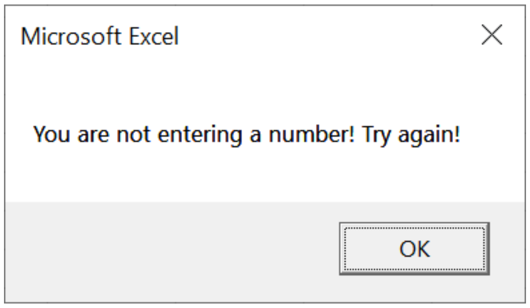
Figure 1.16

Example 1.13 Nested Errors Handling

By referring to Example 1.6, it is better to alert the user the types of error he or she has committed, such as entering non-numeric data like letters or enter zero as denominator. It should be placed in the first place as soon as the user input something in the input box. And the error handler label error_handler1 for this error should be placed after the error_handler2 label. This means the second error handling procedure is nested within the first error handling procedure. Notice that you must put an Exit Sub for the second error handling procedure to prevent to execute the first error handling procedure again. The code is as follow:

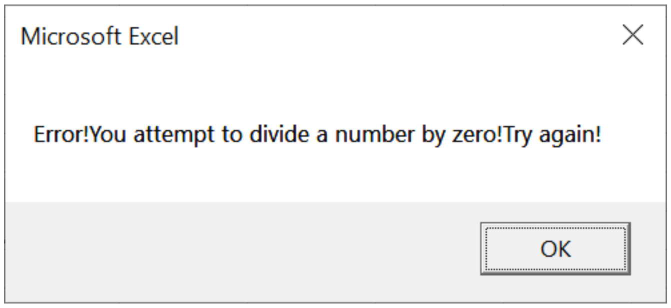
```
Private Sub CommandButton2_Click()
Dim firstNum, secondNum As Double
On Error GoTo error_handler1
 firstNum = InputBox("Enter first number")
 secondNum = InputBox("Enter second number")
On Error GoTo error_handler2
MsgBox firstNum / secondNum
Exit Sub   'To prevent error handling when the inputs are valid

error_handler2:

MsgBox " Error!You attempt to divide a number by zero!Try again!"
 Exit Sub
error_handler1:
 MsgBox " You are not entering a number! Try again!"
```

```
End Sub
```



You are not entering a number! Try again!

Figure 1.17



Error!You attempt to divide a number by zero!Try again!

Figure 1.18

Additionally, you can use the keyword Resume Next to prevent error message from appearing and branch back to the section of the program where error occurred.

```
Private Sub CommandButton1_Click()
On Error Resume Next
 num1 = InputBox("Enter first number")
 num2 = InputBox("Enter second number")
 MsgBox num1 / num2
End Sub
```

Chapter 2 Working with Variables

2.1 The Concept of Variables

Variables are like mailboxes in the post office. The content of the variables changes every now and then, just like the mailboxes. In computer programming, variables are areas allocated by the computer memory to store data. According to Wikipedia:

"

A variable is a storage address identified by a memory address *paired with an associated* symbolic name*, which contains some known or unknown value. The variable name is the usual way to* reference *the stored value, in addition to referring to the variable itself. This separation of name and content allows the name to be used independently of the exact information it represents. The identifier in computer* source code *can be* bound *to a* value *during* run time*, and the value of the variable may thus change during* program execution"

2.2 Variable Names

Like the mailboxes, each variable must be given a name. To name a variable in Excel VBA 365, you must follow the following set of rules:

- It must be less than 255 characters
- No spacing is allowed
- It must not begin with a number
- Period is not permitted
- Cannot use exclamation mark (!), or the characters @, &, $, #
- Cannot repeat names within the same level of scope.

Examples of valid and invalid variable names are displayed in Table 2.1

Table 2.1 Examples of valid and invalid variable names

Valid Name	Invalid Name
My_Car	My.Car
ThisYear	1NewBoy

Long_Name_Can_beUSE	He&HisFather	*& is not acceptable
Group88	Student ID	* Space not allowed

2.3 Declaring Variables

In Excel VBA 365, we must declare the variables before using them. We declare a variable by assigning a name and a data type. Excel VBA 365 data types can be divided into two types, the numeric data types and the non-numeric data types.

2.2.1 Numeric Data Types

Numeric data types are types of data that consist of numbers. In Excel VBA 365, the numeric data are divided into 7 types as summarized in Table 2.2.

Table 2.2 Numeric Data Types

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

2.2.2 Non-numeric Data Types

Non-numeric data types are data that cannot be manipulated using arithmetic operators. They comprise string, date, Boolean and more, as summarized in Table 2.3

Table 2.3 Non-Numeric Data Types

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

You may declare the variables implicitly or explicitly. For example, `sum=text1.text` means that the variable sum is declared implicitly and ready to receive the input in `Textbox1`. For explicit declaration, variables are declared in the general section of the code window using the `Dim` statement. The syntax is as follows:

```
Dim variableName as DataType
```

Example 2.1 Declaration of Different Data Types

```
Dim password As String
Dim yourName As String
Dim firstnum As Integer
Dim secondnum As Integer
Dim total As Integer
Dim birthDay As Date
Dim test As boolean
Dim earning As currency
```

You may also combine the variables into one line, separating each variable with a comma.

```
Dim password As String, yourName As String, firstnum As Integer.
```

If the data type is not specified, Excel VBA 365 will automatically declare the variable as a Variant. For string declaration, there are two possible formats, one for the variable-length string and another for the fixed-length string. For the variable-length string, just use the same syntax as Example 2.1 above. However, for the fixed-length string, you must use the syntax as shown below:

```
 Dim VariableName as String * n
```

n defines the number of characters the string can hold. For example,

```
Dim yourName as String * 10
```

mean yourName can hold no more than 10 Characters.

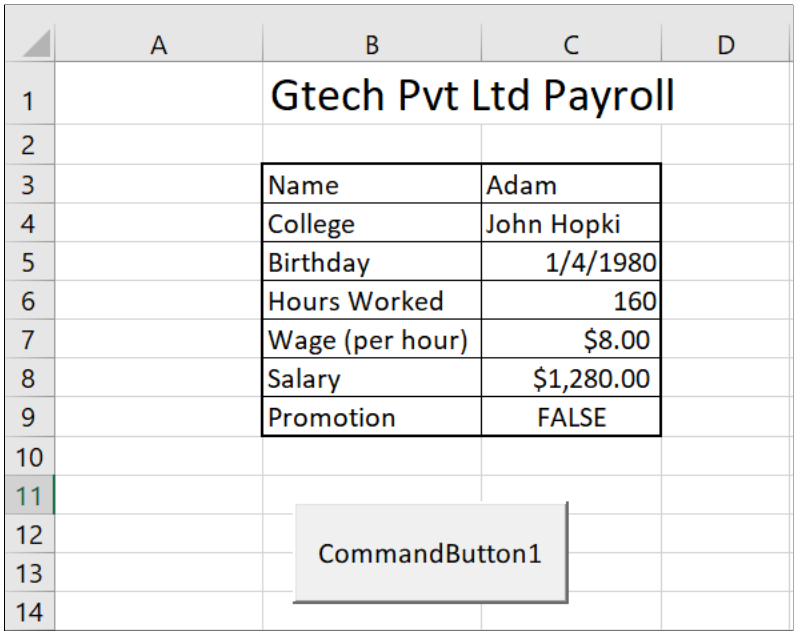
Example 2.2 Creating a Salary Calculator Using If… Then…Else

This is a payroll app that calculate the salary based on wage per hour and hours worked. In this example, we declared four types of variables, namely the string, date, currency and Boolean. The declaration `Dim college As String * 10` implies that the variable college can only holds 10 characters. In addition, we use the `If…Then…Else` statement to determine whether the employee entitle a promotion.

The code

```
 Private Sub CommandButton1_Click()
 Dim yourName As String
 Dim college As String * 10
 Dim birthDay As Date
 Dim workhour As Single
 Dim wage As Currency
 Dim salary As Currency
 Dim promotion As Boolean
  yourName = "Adam"
  college = "John Hopkin University"
  birthDay = "1 April 1980"
  workhour = 160
  wage = 8
  salary = workhour * wage
 If workhour > 160 Then
  promotion = True
 Else
  promotion = False
 End If
  Cells(3, 3) = yourName
  Cells(4, 3) = college
  Cells(5, 3) = birthDay
  Cells(6, 3) = workhour
  Cells(7, 3) = wage
  Cells(8, 3) = salary
  Cells(9, 3) = promotion

 End Sub
```

The output is as seen in Figure 2.1



	A	B	C	D
1		Gtech Pvt Ltd Payroll		
2				
3		Name	Adam	
4		College	John Hopki	
5		Birthday	1/4/1980	
6		Hours Worked	160	
7		Wage (per hour)	$8.00	
8		Salary	$1,280.00	
9		Promotion	FALSE	

Figure 2.1

You can notice that the College name has been truncated to just 10 characters (including spacing).

2.2 Option Explicit

The keyword `Option Explicit` in Excel VB365 programming is to track errors in the usage of variable. For example, if we commit a typo, Excel VBA 365 will pop up an error message “Variable not defined”. Indeed, `Option Explicit` forces the programmer to declare every variable using the `Dim` keyword. It is a good practice to use `Option Explicit` because it will prevent the incorrect use of variable names due to typing errors, especially when the program gets larger. Using `Option Explicit` save time in debugging.

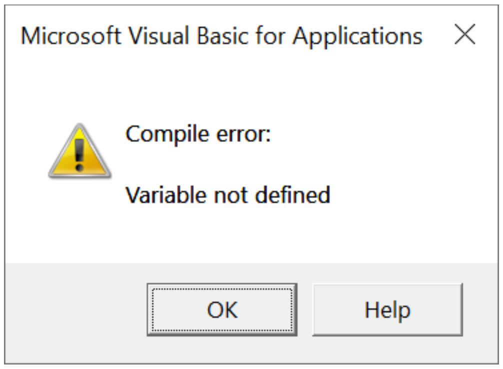
When `Option Explicit` is included in the program code, every variable must be declared using the `Dim` keyword. Any variable that is not declared or wrongly typed will produce the "Variable not defined" error. The error must be corrected before the program can continue to run.

Example 2.3 Using Option Explicit to Catch Typo Errors

This example uses the Option Explicit keyword and it demonstrates how a typo is being tracked.

```
Option Explicit
Private Sub CommandButton1_Click()
Dim YourName As String
Dim password As String
YourName = "John"
password = 12345
Cells(1, 2) = YourNam
Cells(1, 3) = password
End Sub
```

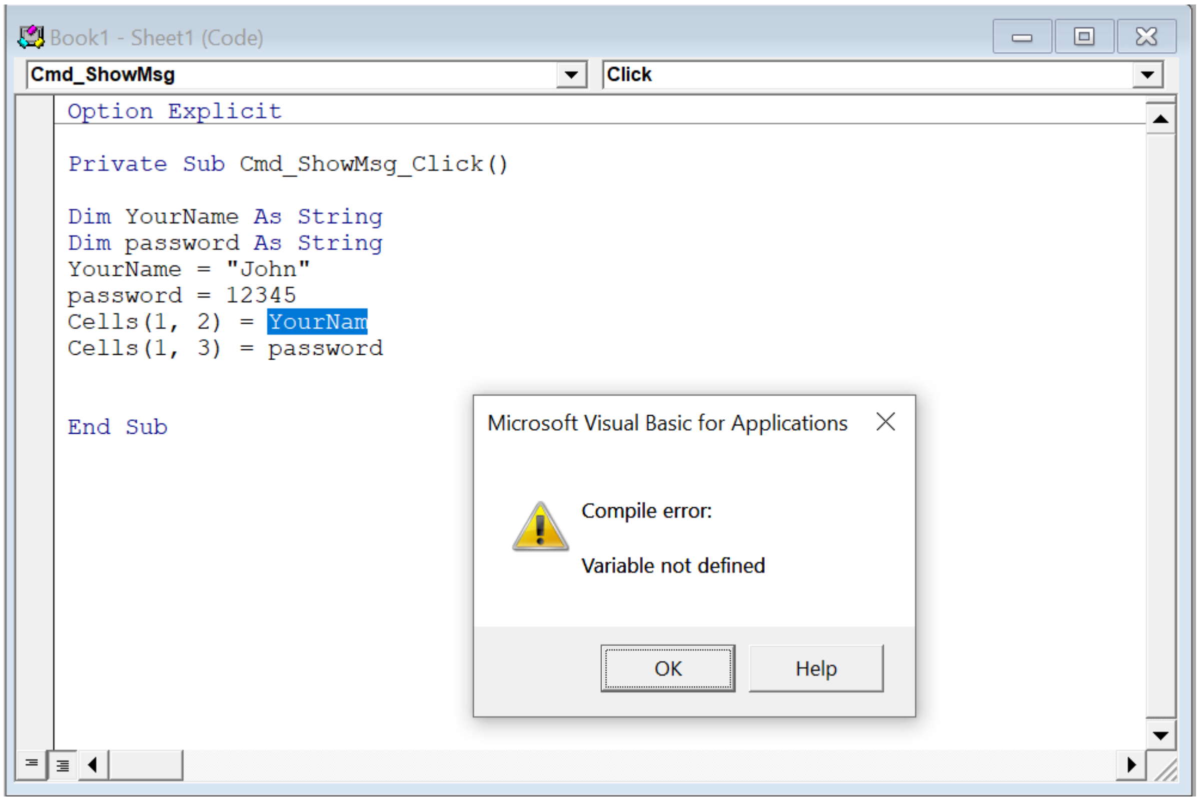
The typo is `YourNam` and so the error message 'variable not defined" will be displayed and the program is suspended, as shown in Figure 2.2. The error Yournam will also be highlighted as shown in Figure 2.3.


Microsoft Visual Basic for Applications

Compile error:

Variable not defined

Figure 2.2



```
Option Explicit

Private Sub Cmd_ShowMsg_Click()

Dim YourName As String
Dim password As String
YourName = "John"
password = 12345
Cells(1, 2) = YourNam
Cells(1, 3) = password


End Sub
```

```
Microsoft Visual Basic for Applications

Compile error:

Variable not defined
```

Figure 2.3 **Error message due to typo error**

2.3 Assigning Values to the Variables

After declaring several variables with the `Dim` statements, we can assign values to them. The syntax of an assignment is

```
Variable=Expression
```

The variable can be a declared variable or a control property value. The expression can be a mathematical expression, a number, a string, a Boolean value (true or false) and more. Here are some examples:

```
firstNumber=100
secondNumber=firstNumber-99
userName="John Lyan"
userpass.Text = password
Label1.Visible = True
Command1.Visible = False
ThirdNumber = Val(usernum1.Text)
```

```
total = firstNumber + secondNumber+ThirdNumber
```

2.4 Performing Arithmetic Operations

To compute numeric values, we shall use arithmetic operators. In Excel VBA 365, the symbols for arithmetic operators are different from normal mathematical operators except for + and -. For example, multiplication is * and division is /. Besides. we must differentiate between / and \, where / is a normal division whilst \ is an integer division. Integer division \ discards the decimals. For example, 27\5 is 5. The Excel VBA 365 arithmetic operators as shown in Table 2.3.

Table 2.3 Arithmetic Operators

Operator	**Mathematical function**	**Example**
^	Exponential	2^4=16
*	Multiplication	4*3=12
/	Division	12/4=3
Mod	Modulus	15 Mod 4=3
\	Integer Division	19\4=4
+ or &	String concatenation	"Visual"&"Basic"="Visual Basic"

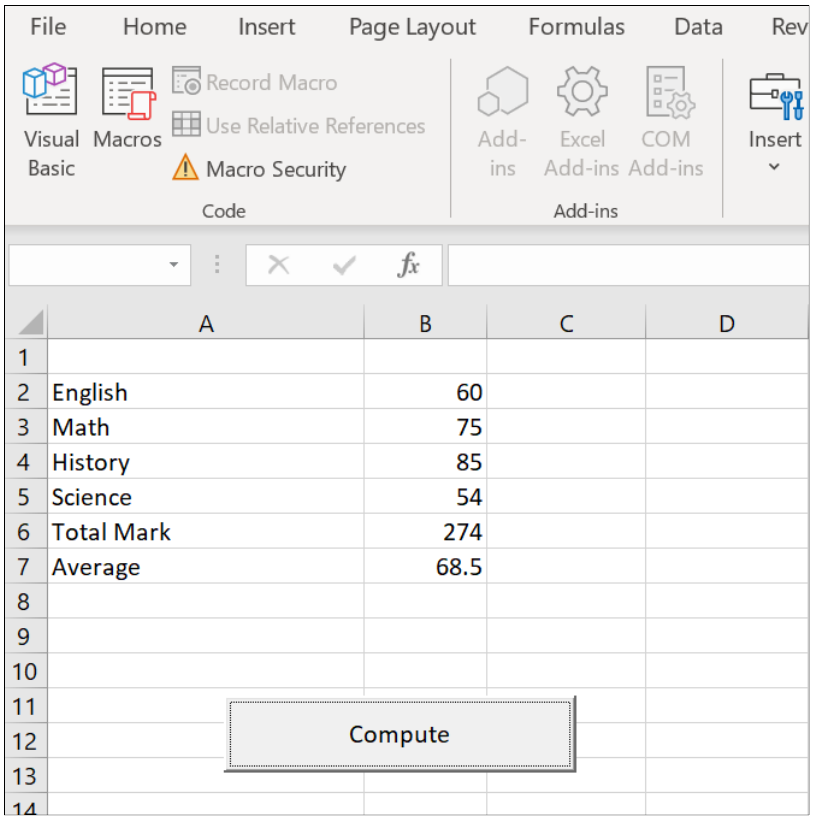
Example 2.4 Compute Examination Results

This example calculates the total mark and the average mark of an examination result. We declared four variables as Single and another two variables as Double. In the code, we use `Worksheetfunction.sum` to add the marks and `Worksheetfunction.count` to count the number of subjects.

The Code

```
Option Explicit
Private Sub Cmd_Calculate_Click()
Dim mark1, mark2, mark3, mark4 As Single
Dim total, average As Double

mark1 = 60
mark2 = 75
mark3 = 85
mark4 = 54
Cells(2, 2) = mark1
Cells(3, 2) = mark2
Cells(4, 2) = mark3
Cells(5, 2) = mark4
total = WorksheetFunction.Sum(Range(Cells(2, 2), Cells(5, 2)))
average = total / WorksheetFunction.Count(Range(Cells(2, 2), Cells(5, 2)))
Cells(6, 2) = total
Cells(7, 2) = average
End Sub
```

The output is shown in Figure 2.4



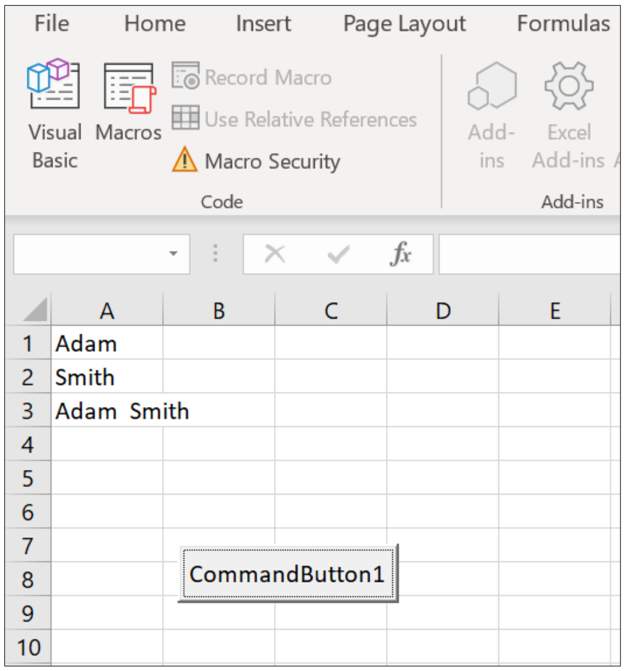
	A	B	C	D
1				
2	English	60		
3	Math	75		
4	History	85		
5	Science	54		
6	Total Mark	274		
7	Average	68.5		

Figure 2.4

Example 2.5 Concatenation of Strings

In this example, three variables are declared as string. The variable firstName and the variable secondName will receive their data entered by the user into Cells(1,1) and cells(2,1) respectively. You will notice that performing arithmetic operation on strings will result in the concatenation of the strings, as shown in Figure 2.5.

```
Option Explicit
Private Sub CommandButton1_Click()
Dim secondName As String
Dim yourName As String
firstName = Cells(1,1)
secondName = Cells(2,1)
yourName = firstName + "  " + secondName
Cells(3,1) = yourName
End Sub
```



	A	B	C	D	E
1	Adam				
2	Smith				
3	Adam Smith				
4					
5					
6					
7					
8					
9					
10					

Figure 2.5 Concatenation of Strings

2.5 Arrays

When we work with a single item in Excel VBA 365, we only need to declare one variable. However, if we need to deal with a list of items, we need to declare an array of variables instead of using a variable for each item. For example, if we need to enter 100 names, instead of declaring 100 different variables, we need to declare only one array.

An array is a group of variables with the same data type and name. We differentiate each item in the array by using subscript, the index value of each item. For example, Studentname (1), Studentname (2), Studentname (3) …Studentname(n)

2.5.1 Declaring an Array

We use the Dim statement to declare an array just as the way we declare a single variable. In Excel VBA 365 we can have a one-dimensional array, two-dimensional array or even a multidimensional array (up to 60)

2.5.2 One-Dimensional Array

The statement to declare a one-dimensional array in Excel VBA 365 is as follows:

```
Dim arrayName(index) as dataType or Dim arrayName(first index to last index) as
dataType
```

For example, the following statement declares an array that comprises 10 elements.

```
 Dim StudentName(10) as String
 Dim StudentName(1 to 10) as String
 Dim StudentMark(10) as Single
 Dim StudentMark( 1 to 10) as Single
```

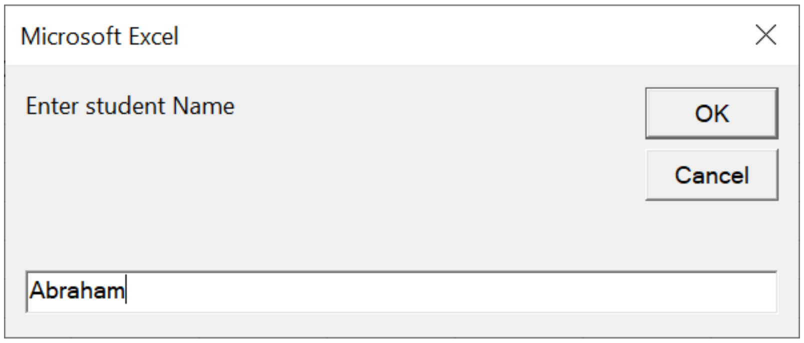
Example 2.6 Array of Names

In this example, we define an array StudentName comprising five names using the Dim keyword. We include an InputBox to accept input from the user. We also use the For…Next loop to accept the input five times and display the five names from cell A1 to cell E1. The code is as follows:

```
Private Sub CommandButton1_Click( )
Dim StudentName(1 to 5) As String
```

```
 For i = 1 To 5
  StudentName(i) = InputBox("Enter student Name")
  Cells(i, 1) = StudentName(i)
 Next
End Sub
```

* You can also declare the array using `Dim StudentName(5) As String` When we run the program, an input box will appear, as shown below. This input box will repeat five times and let the user enter five names, as shown in Figure 2.6.

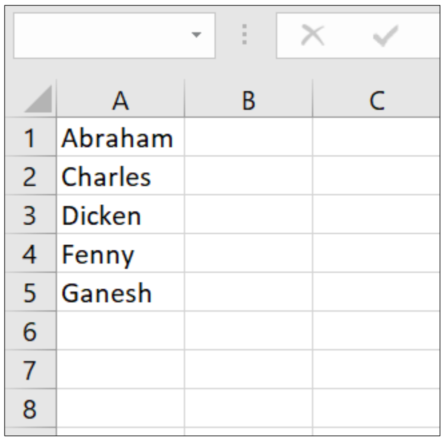

Microsoft Excel

Enter student Name

Abraham

Figure 2.6

Five names will be displayed in the worksheet as shown in Figure 2.6



	A	B	C
1	Abraham		
2	Charles		
3	Dicken		
4	Fenny		
5	Ganesh		
6			
7			
8			

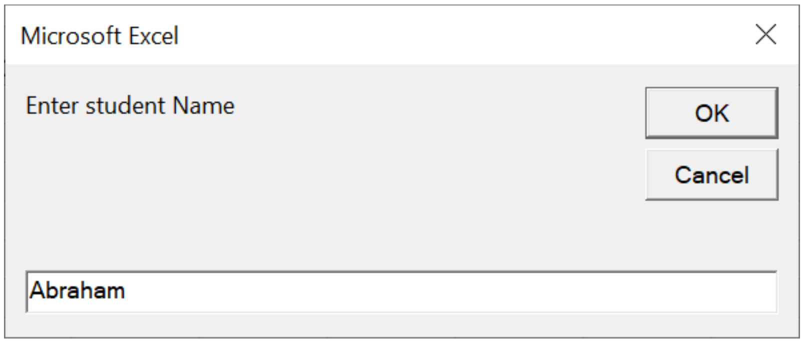
Figure 2.7

You can also declare more than one array on a single line. In Example 2.7, we declare three arrays in a single line, separated by commas.

Example 2.7 Declare Arrays in a Single Line

```
Private Sub CommandButton1_Click( )
Dim StudentName(3) As String, StudentID(3) As String, StudentMark(3) As Single
 For i = 1 To 3 StudentName(i) = InputBox("Enter student Name")
  StudentID(i) = InputBox("Enter student ID")
  StudentMark(i) = InputBox("Enter student Mark")
  Cells(i, 1) = StudentName(i)
  Cells(i, 2) = StudentID(i)
  Cells(i, 3) = StudentMark(i)
 Next
End Sub
```

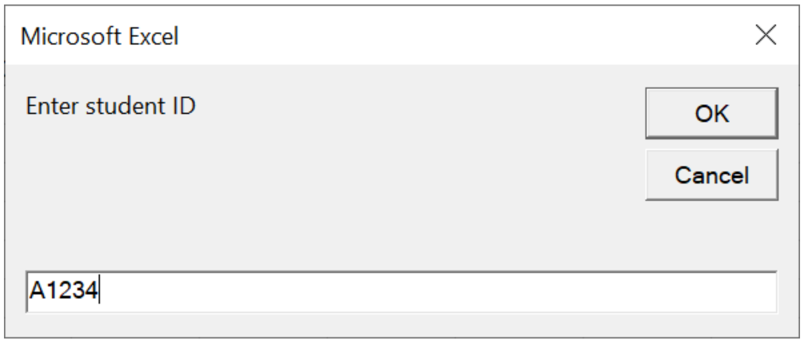
When we run the program, three input boxes will appear consecutively to let the user enter the student name, the student ID and then the student mark. The process will repeat three times until the particulars of all three students have been entered. The three input boxes and the output images are shown below:



Enter student Name

Abraham

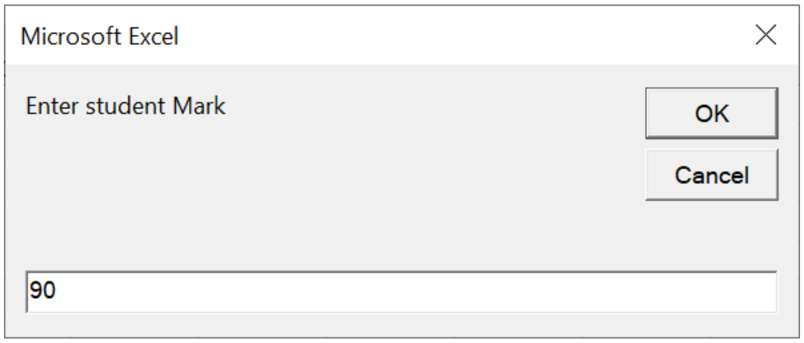
Figure 2.8



Enter student ID

A1234

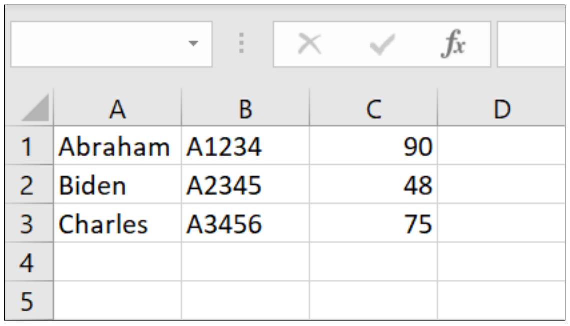
Figure 2.9



90

Figure 2.10

The Output is shown in the Figure 2.11



	A	B	C	D
1	Abraham	A1234	90	
2	Biden	A2345	48	
3	Charles	A3456	75	
4				
5				

Figure 2.11

2.5.3 Two-Dimensional Array

Multidimensional arrays are often needed when we are dealing with a more complex database, especially those that handle a large amount of data. Data are usually organized and arranged in table form; this is where the multidimensional arrays come into play. However, in this tutorial, we are dealing only with the two-dimensional array. A two-dimensional array can be represented by a table that contains rows and columns, where one index represents the rows and the other index represent the columns.The statement to declare a two-dimensional array is

```
Dim arrayName (num1, num2) as datatype
```

Where num1 is the suffix of the first dimension of the last element and num2 is the suffix of the second dimension of the last element in the array. The suffixes of the element in the array will start with (0, 0) unless you set the Option Base to 1. In the case when the Option Base is set to 1, then the suffixes of the element in the array will start with (1, 1). For example,

```
Dim Score (3, 3) as Integer
```

will create a two-dimensional array consists of 16 elements. These elements can be organized in a table form as shown in the table below:

Table 2.1

Score(0,0)	Score(0,1)	Score(0,2)	Score(0,3)
Score(1,0)	Score(1,1)	Score(1,2)	Score(1,3)
Score(2,0)	Score(2,1)	Score(2,2)	Score(2,3)
Score(3,0)	Score(3,1)	Score(3,2)	Score(3,3)

If you set the option base to 1, then there will be only 9 elements, i.e from Score(1,1) to Score(3,3). However, if you want the first element to start with suffixes (1,1) you can also use the following format of declaration:

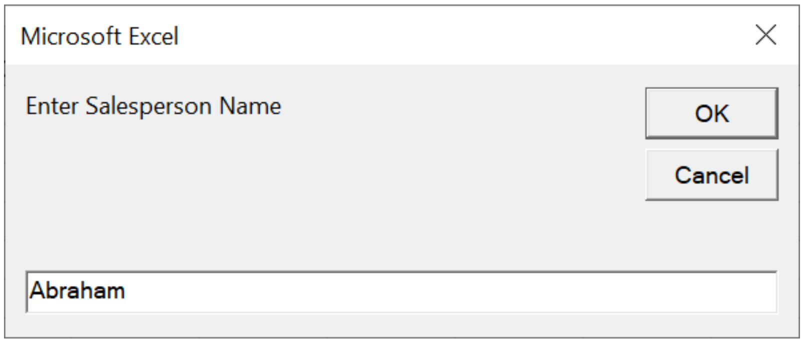
```
Dim Score(1 to 3, 1 to 3) as Integer
```

Example 2.8 Tracking the Performance of Salespersons

If a company wants to track the performance of 5 salespersons over a period of 2 days, you can create a 5×2 array in Excel VBA 365, denoted by a 5X 2 table in a worksheet. Therefore, you can write the following VBA code using a nested For loop.

```
Private Sub CommandButton1_Click()
Dim SalesPersonName As String
Dim SalesPersonID, Day As Integer
Dim SalesVolume(2 To 6, 2 To 3) As Double
For SalesPersonID = 2 To 6
SalesPersonName = InputBox("Enter Salesperson Name")
Cells(SalesPersonID, 1) = SalesPersonName
 For Day = 2 To 3
  SalesVolume(SalesPersonID, Day) = InputBox("Enter Sales Volume for day " & (Day
- 1))
  Cells(SalesPersonID, Day) = SalesVolume(SalesPersonID, Day)
 Next Day
 Next SalesPersonID
End Sub
```

When the user runs the program, the input box that will prompt the user to enter salesperson's name, as shown in the Figure 2.12

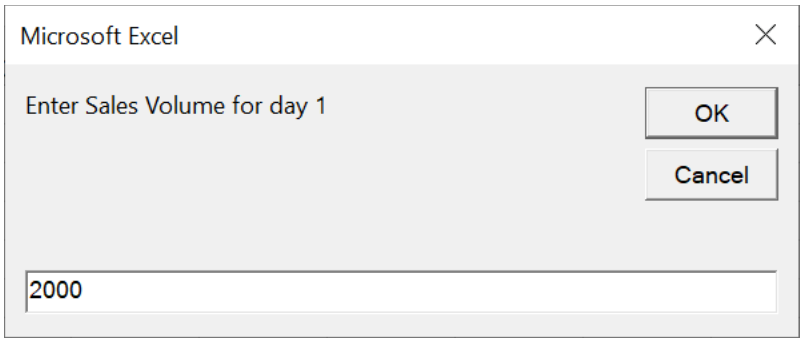


Enter Salesperson Name

Abraham

Figure 2.12

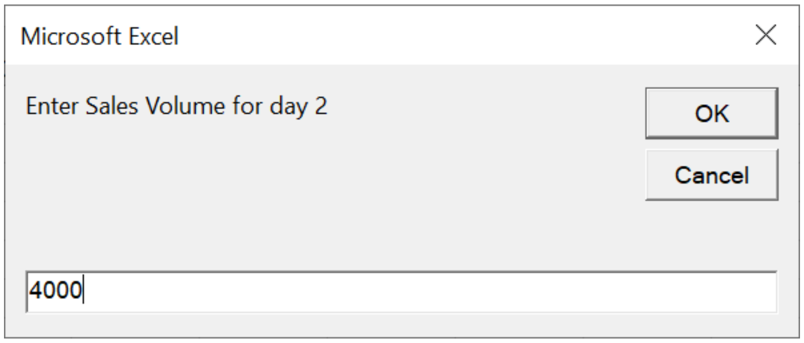
Next, you will be asked to enter the sales volume for day 1 and day 2, as shown in Figure 2.13 and Figure 2.14.



Enter Sales Volume for day 1

2000

Figure 2.13

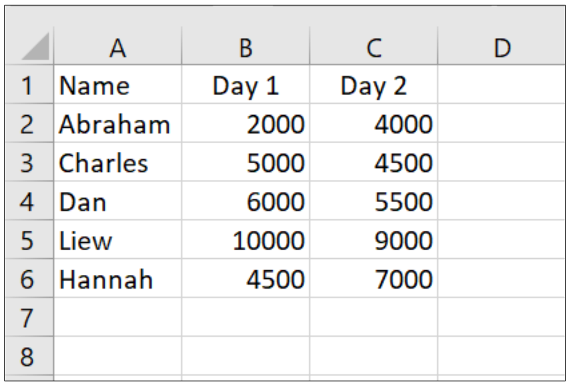


Enter Sales Volume for day 2

4000

Figure 2.14

After entered data for five salespersons, you will obtain a table as shown in Figure 2.15



	A	B	C	D
1	Name	Day 1	Day 2	
2	Abraham	2000	4000	
3	Charles	5000	4500	
4	Dan	6000	5500	
5	Liew	10000	9000	
6	Hannah	4500	7000	
7				
8				

Figure 2.15