

Dr.Liew Voon Kiong

# Excel VBA 365 Handbook

A Comprehensive Guide to  
Excel Macro Programming

# **Excel VBA 365 Handbook**

## **A Comprehensive Guide to Excel Macro Programming**

**by**

**Dr. Liew Voon Kiong**

# **Disclaimer**

Excel VBA 365 Handbook 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.

Copyright© Liew Voon Kiong 2020. All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, without permission in writing from the author.

## 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](http://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**. 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

1.1 The Concept of Excel VBA

1.2 The Visual Basic Editor in MS Excel 365

    1.2.1 Building Excel VBA 365 using the Controls.

        Example 1.1

        Example 1.2

    1.2.2 Building Excel VBA 365 using the Visual Basic Editor

1.3 The Excel VBA 365 Code

        Example 1.3

        Example 1.4

        Example 1.5

1.4 Errors Handling

    1.4.1 Writing the Errors Handling Code

        Example 1.6

        Example 1.7 Nested Errors Handling

## Chapter 2 Working with Variables

2.1 The Concept of Variables

2.2 Variable Names

2.3 Declaring Variables

    2.2.1 Numeric Data Types

    2.2.2 Non-numeric Data Types

        Example 2.1

        Example 2.2

2.2 Option Explicit

        Example 2.3

2.3 Assigning Values to the Variables

2.4 Performing Arithmetic Operations

        Example 2.4

[Example 2.5](#)

[2.5 Arrays](#)

[2.5.1 Declaring an Array](#)

[2.5.2 One-Dimensional Array](#)

[Example 2.6](#)

[Example 2.7](#)

[2.5.3 Two-Dimensional Array](#)

[Example 2.8](#)

[Chapter 3 Message box and Input Box](#)

[3.1 The MsgBox\( \) Function](#)

[Example 3.1](#)

[Example 3.2](#)

[Example 3.3](#)

[3.2 The InputBox\(\) Function](#)

[Example 3.4](#)

[Chapter 4 Using If...Then...Else](#)

[4.1 Conditional Operators](#)

[4.2 Logical Operators](#)

[4.3 Using If...Then...Elseif... Else](#)

[Example 4.1](#)

[Example 4.2](#)

[Example 4.3](#)

[Chapter 5 Looping](#)

[5.1 For...Next Loop](#)

[5.1.1 The Single For...Next Loop](#)

[Example 5.1](#)

[Example 5.2](#)

[Example 5.3](#)

[5.1.2 The Nested For...Next Loop](#)

[Example 5.4](#)

[Example 5.5](#)

## [5.2 The Do...Loop](#)

[Example 5.6](#)

[Example 5.7](#)

[Example 5.8](#)

[Example 5.9](#)

[Example 5.10](#)

[Example 5.11](#)

[Example 5.12 Prime Number Tester](#)

## [5.3 The While...Wend Loop](#)

[Example 5.13](#)

[Example 5.14](#)

[Example 5.15 Number Guessing Game](#)

## [Chapter 6 Select Case...End Select](#)

[Example 6.1](#)

[Example 6.2](#)

[Example 6.3](#)

## [Chapter 7: Excel VBA 365 Objects](#)

### [7.1: Objects](#)

### [7.2: Properties and Methods](#)

#### [7.2.1 Properties](#)

[Example 7.1](#)

[Example 7.2](#)

#### [7.2.2 Methods](#)

[a\) The Count method](#)

[Example 7.3](#)

[b\) The ClearContents Method](#)

[Example 7.4](#)

[c\) The ClearFormats Method](#)

[Example 7.5](#)

[d\) The Clear Method](#)

[Example 7.6](#)

[e\) The Select Method](#)

[Example 7.7](#)

[Example 7.8](#)

[Example 7.9](#)

[f\) The Autofill Method](#)

[Example 7.10](#)

[Example 7.11](#)

[Example 7.12](#)

[Example 7.13](#)

## [Chapter 8: The Workbook Object](#)

### [8.1 The Workbook Properties](#)

#### [8.1.1 The Name Property](#)

[Example 8.1 Displaying the Workbook Name](#)

#### [8.1.2 The Path Property](#)

[Example 8.2 Showing the Path of the workbook](#)

[Example 8.3 Showing the Path and Name of a workbook](#)

### [8.2 The Workbook Methods](#)

#### [8.2.1 The Save Method](#)

[Example 8.4](#)

#### [8.2.2 The SaveAs Method](#)

[Example 8.5](#)

#### [8.2.3 The Open Method](#)

[Example 8.6](#)

#### [8.2.4 The Close Method](#)

[Example 8.7](#)

## [Chapter 9 The Worksheet Object](#)

### [9.1 The Worksheet Properties](#)

### [9.1 The Worksheet Properties](#)

### [9.1.1 The Name Property](#)

[Example 9.1](#)

### [9.1.2 The Count Property](#)

[Example 9.2](#)

[Example 9.3](#)

[Example 9.4](#)

## [9.2 The Worksheet Methods](#)

### [9.2.1 The Add Method](#)

[Example 9.5](#)

### [9.2.2 The Delete Method](#)

[Example 9.6](#)

### [9.2.3 The Select Method](#)

[Example 9.7](#)

[Example 9.8](#)

[Example 9.9](#)

[Example 9.10](#)

[Example 9.11](#)

### [9.2.4 The Copy and Paste Method](#)

[Example 9.12](#)

[Example 9.13](#)

## [Chapter 10: The Range Object](#)

### [10.1 The Range Properties](#)

#### [10.1.1 Formatting](#)

[Example 10.1](#)

[Example 10.2](#)

#### [10.1.2 The Formula Property](#)

[Example 10.3](#)

#### [10.1.3 The Built-in Formulas](#)

[Example 10.4](#)

[Example 10.5: Mode](#)

[Example 10.6: Median](#)

[Example 10.7](#)

[10.2 The Range Methods](#)

[10.2.1 Autofill Method](#)

[Example 10.8](#)

[10.2.2 Select, Copy and Paste Methods](#)

[Example 10.9](#)

[10.2.2 Copy and PasteSpecial Methods](#)

[Example 10.10](#)

[Example 10.11](#)

[Chapter 11: Excel VBA Controls](#)

[11.1 Check Box](#)

[Example 11.1](#)

[Example 11.2](#)

[Example 11.3](#)

[11.2 Text Box](#)

[Example 11.4](#)

[11.3 Option Button](#)

[Example 11.5](#)

[Example 11.6](#)

[Example 11.7](#)

[11.4 List Box](#)

[Example 11.8](#)

[11.5 Combo Box](#)

[Example 11.9](#)

[11.6 Toggle Button](#)

[Example 11.10](#)

[Chapter 12 Functions](#)

[12.1 The Concept of Functions](#)

[12.2 Types of Functions](#)

## [12.2 Built-In Functions](#)

[Example 12.1](#)

## [12.3 User-Defined Function](#)

[Example 12.2](#)

[Example 12.3](#)

[Example 12.4](#)

## [12.4 Passing variables by reference and by Value in a Function](#)

[Example 12.5](#)

## [Chapter 13 Sub Procedures](#)

[Example 13.1](#)

[Example 13.2](#)

[Example 13.3](#)

[Example 13.4](#)

[Example 13.6](#)

## [Chapter 14 String Handling Functions](#)

[14.1 InStr](#)

[14.2. Left](#)

[14.3. Right](#)

[14.4. Mid](#)

[14.5. Len](#)

[Example 14.1](#)

## [Chapter 15 Date and Time Functions](#)

[15.1 Using the Now \( \) Function](#)

[Example 15.1](#)

[15.2 Date and Time Functions](#)

[Example 15.2](#)

[15.3 DatePart Function](#)

[Example 15.3](#)

[15.4 Adding and Subtracting Dates](#)

[Example 15.4](#)

## Chapter 16 UseForm

### 16.1 The Keyboard Events

Example 16.1

Example 16.2

### 16.2 The Mouse Events

Example 16.3

Example 16.4

Example 16.5

Example 16.6 Web Browser

## Chapter 17 Working with Files

### 17.1 Application.GetOpenFilename method

Example 17.1

### 17.2 Application.GetSaveAsFilename method

Example 17.2

### 17.3 Creating a Text File

Example 17.3 Creating a text file

### 17.4 Reading a File

Example 17.4 Reading a text file

## Chapter 18 Class Modules

### 18.1 Creating a Class Module

Example 18.1

Example 18.2

Example 18.3

### 18.2 Class Module Properties

Example 18.4

Example 18.5

Example 18.6 Virtual Keyboard

Example 18.7

## Chapter 19 Drawing Charts

### Example 1 BMI Calculator

[Example 2 Financial Calculator](#)  
[Example 3 Investment Calculator](#)  
[Example 4 Prime Number Tester](#)  
[Example 5 Selective Summation](#)  
[Example 6 Excel VBA 365 Windows Media Player](#)  
[Example 7 Animation](#)  
[Example 8 Amortization Calculator](#)  
[Example 9 Boggle](#)  
[Example 10 Calculator](#)  
[Example 11 Scientific Calculator](#)  
[Example 12 Dice](#)  
[Example 13 Geometric Progression](#)  
[Example 14 Password Cracker](#)  
[Example 15 Digital Slot Machine](#)  
[Example 16 Professional Slot Machine](#)  
[Example 17 Quadratic Equation Solver](#)  
[Example 18 Simple Harmonic Motion](#)  
[Example 19 Simultaneous Equation](#)  
[Example 20 Star War](#)  
[Example 21 Stock Trading](#)  
[Index](#)

# Chapter 1 Introduction to Excel VBA 365

---

This book is based on the latest Microsoft Excel, which is one of the applications of Microsoft Office 365; hence I named this book Excel VBA 365 Made Easy. All the 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 user-defined 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. This book is based on MS Excel 365, but 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.

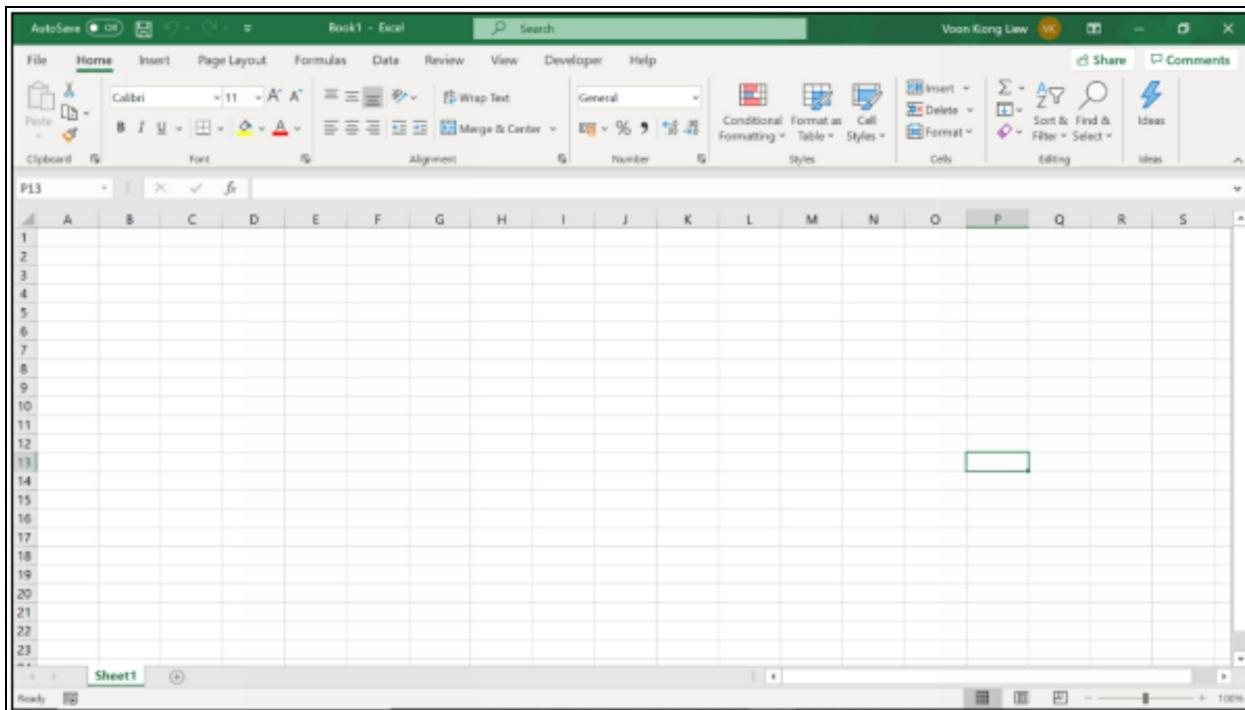


Figure 1.1 Microsoft Excel 365 workbook

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.

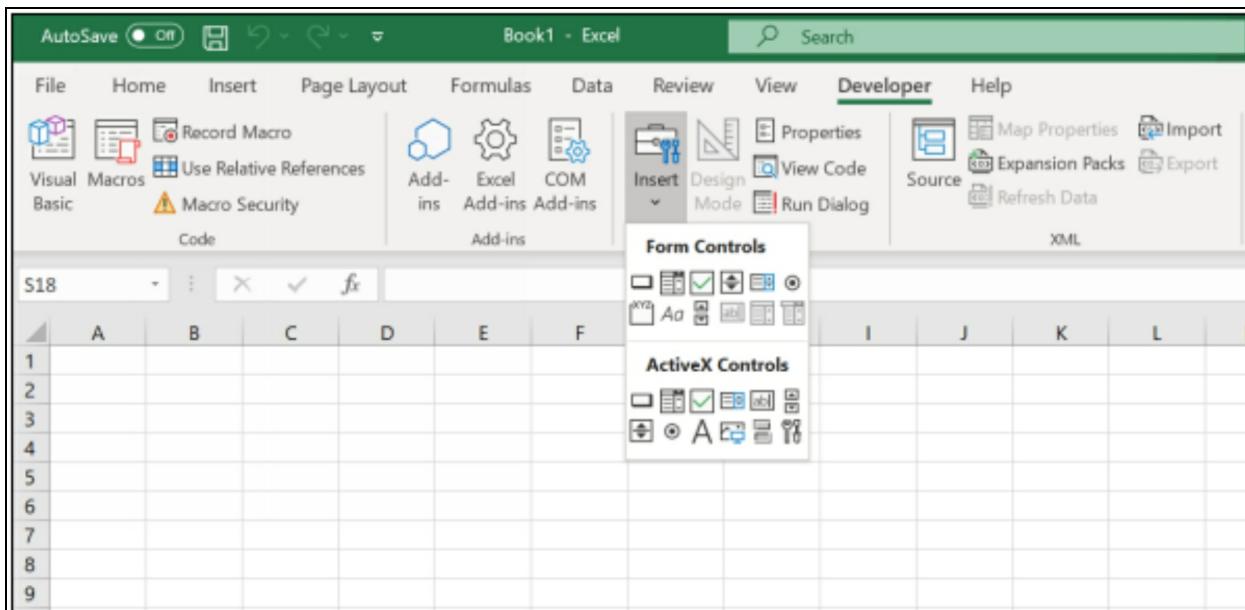
There are two ways to start programming Excel VBA, by placing controls on

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.

### 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.

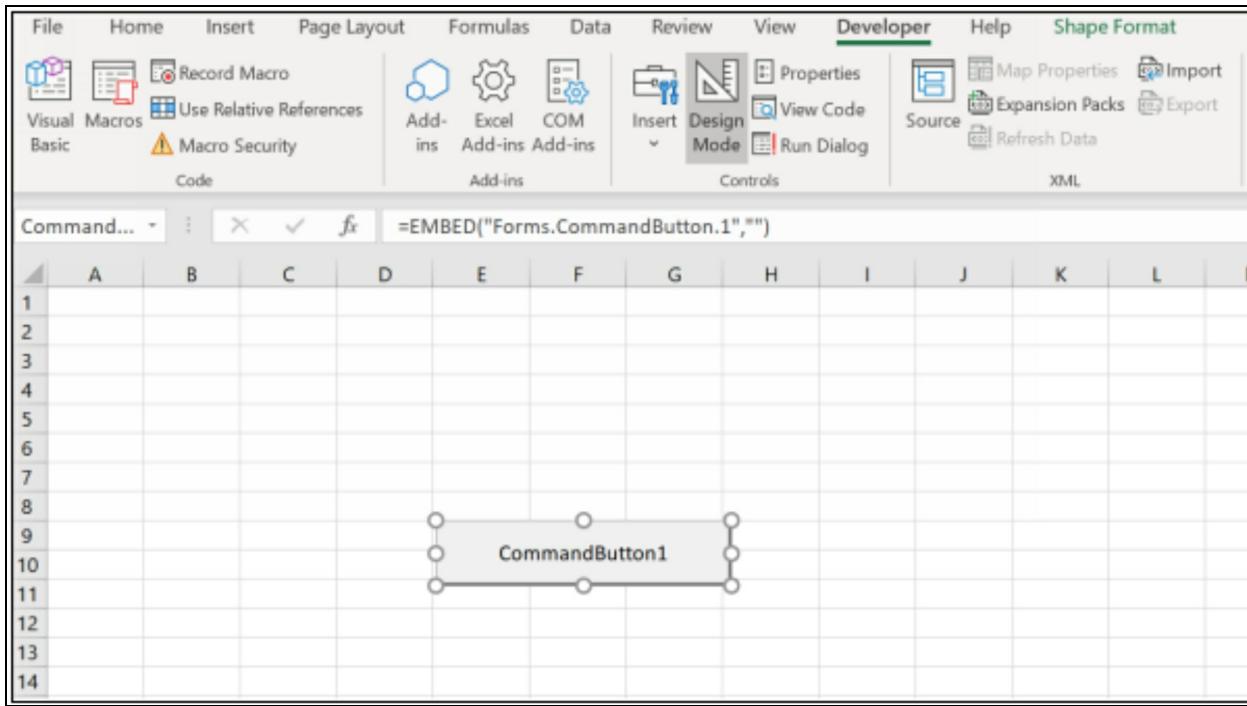
To use the controls, navigate to the Developer tab then click on the Insert button to access the ActiveX controls and Form Controls, as shown in Figure 1.2.



**Figure 1.2 Form and ActiveX Controls**

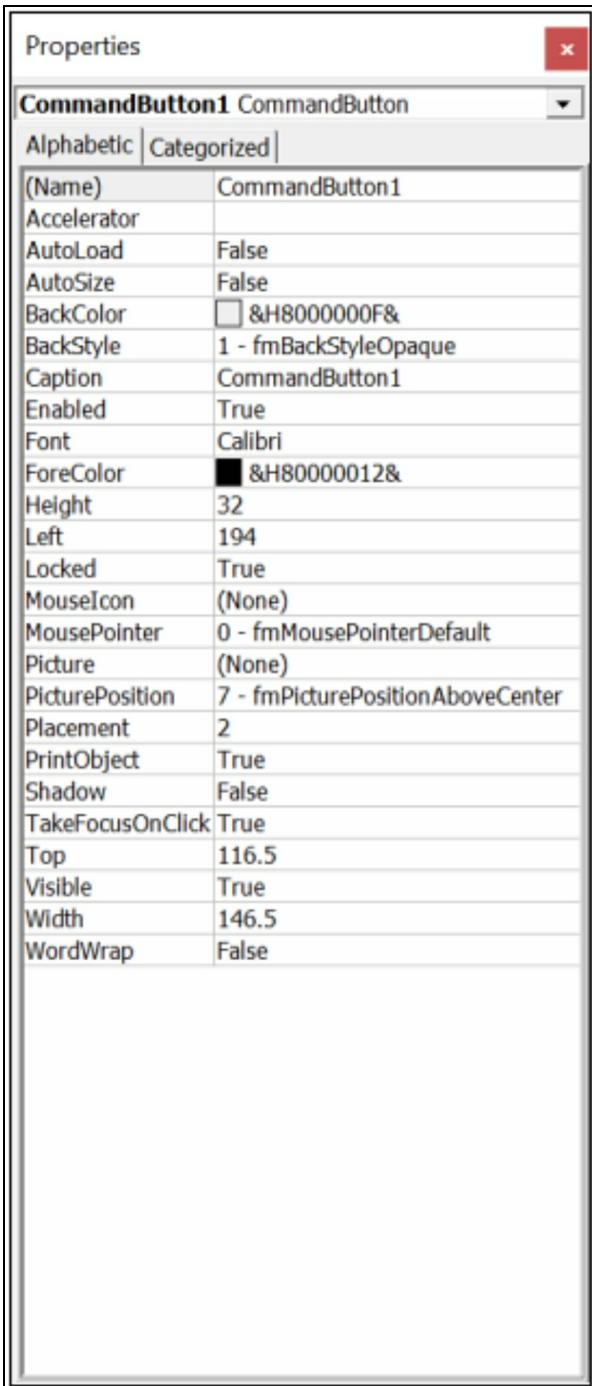
Let us start with the command button. To place a command button on the MS Excel spreadsheet, click on the command button under ActiveX controls and draw it on the worksheet, as shown in Figure 1.3. Notice that the Developer

environment is in the Design Mode at this stage.



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

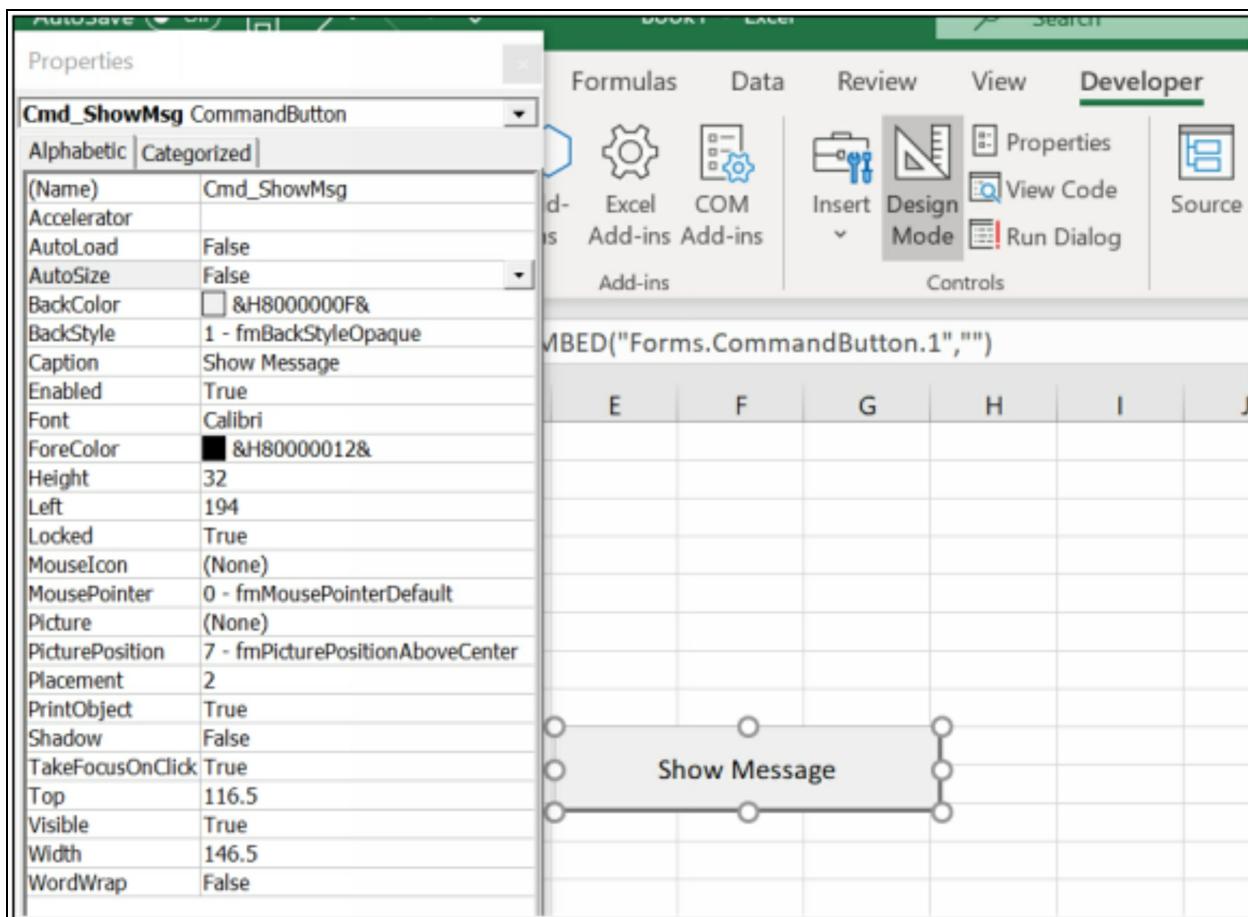
At this stage, you might want to customize the command button by changing some of its properties. To access the properties, right-click the command button and select the Properties option to launch the Properties window, as shown in Figure 1.4.



**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.

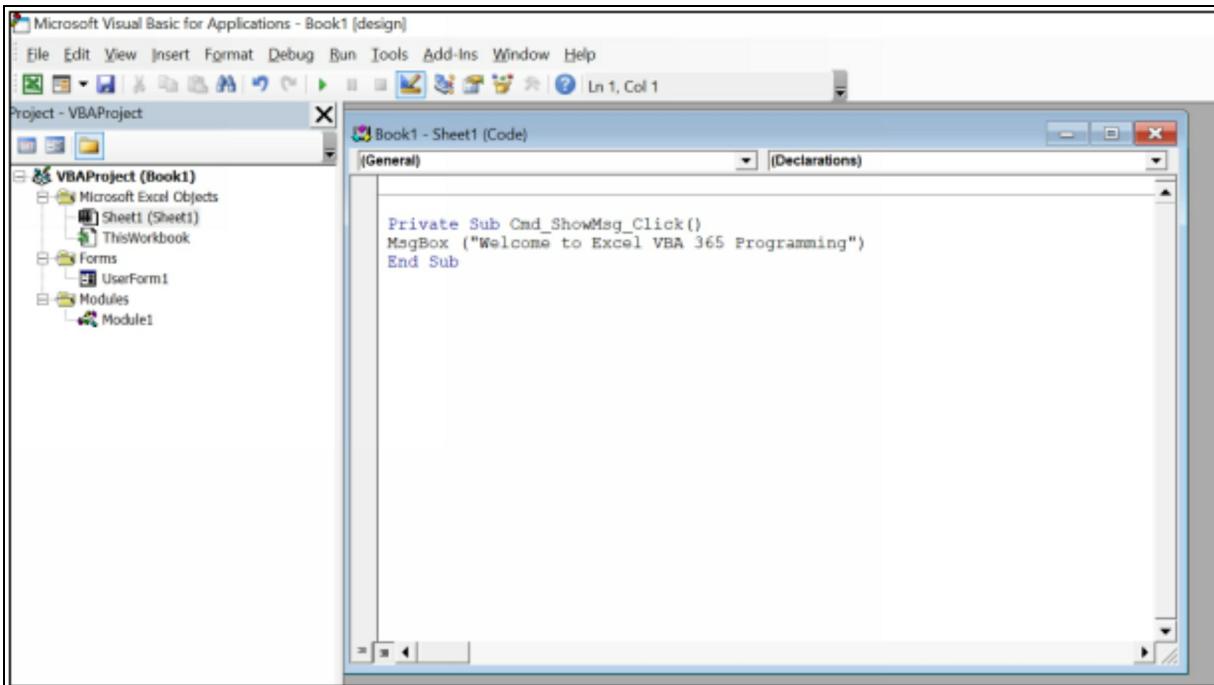


**Figure 1.5**

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

```
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



**Figure 1.7 The Properties window**

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

```
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
```

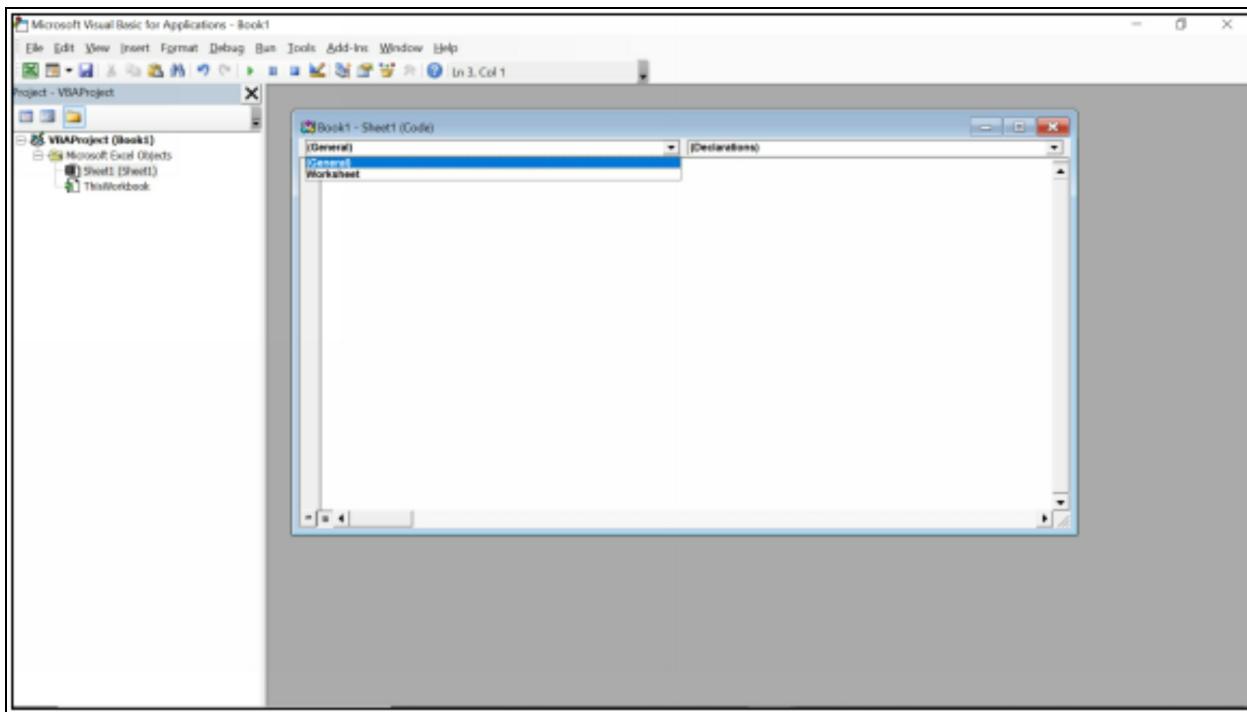
The first statement will populate the cells from the range cell A1 to cell D4 with the phrase "Excel VBA 365". The second statement populate the cells from the range cell A5 to cell D5 with the value of 100. The third statement populate the cells from the range cell A6 to cell D6 with the value of 50. 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					Show Message			

**Figure 1.8**

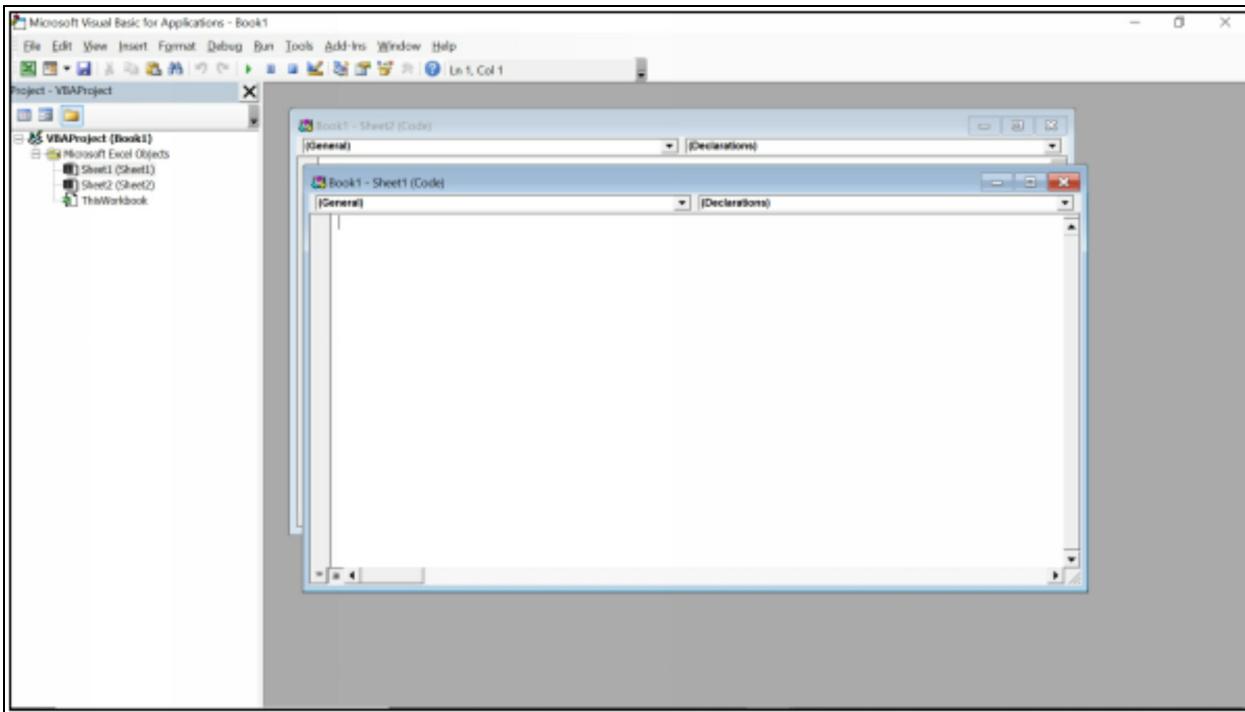
### 1.2.2 Building Excel VBA 365 using the Visual Basic Editor

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

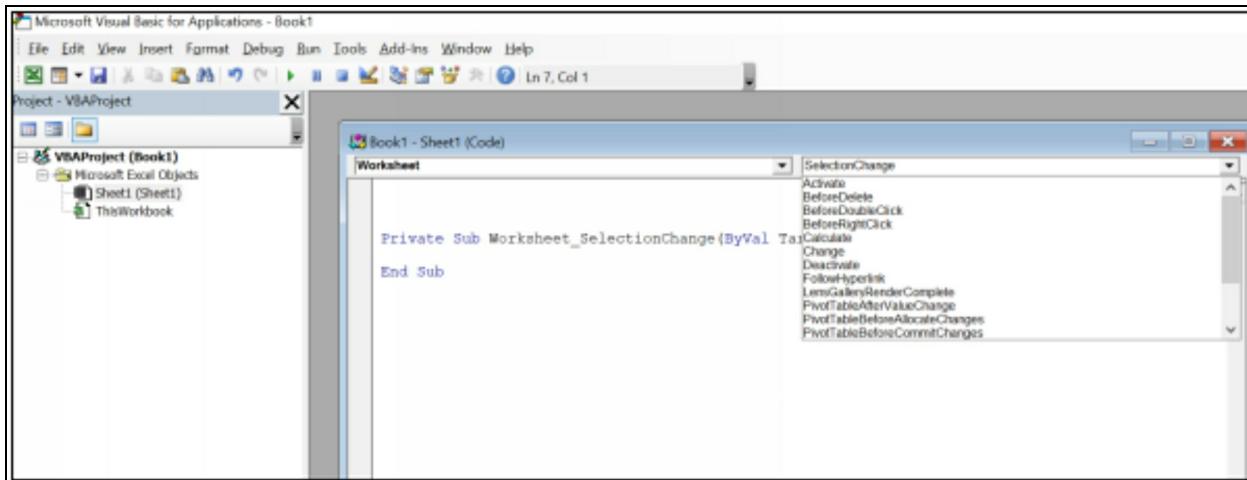


**Figure 1.10**

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.



**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
```

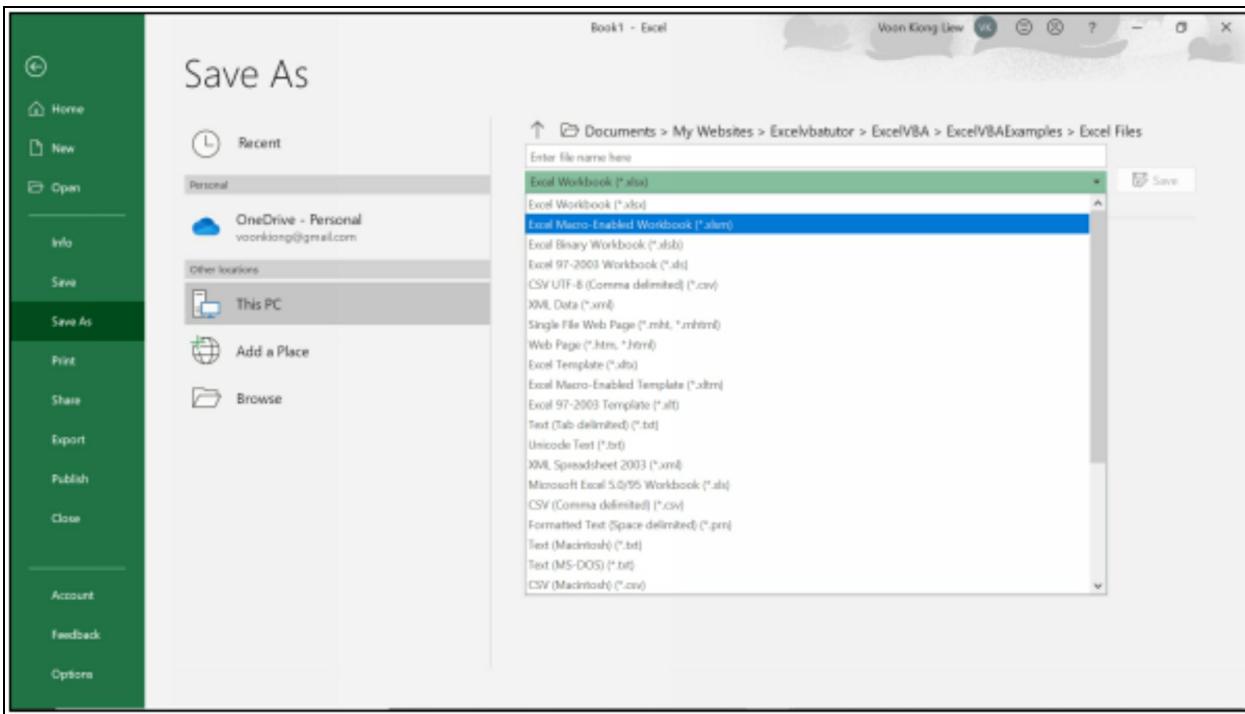
This code means whenever you click on another cell of the Worksheet, the message " You have changed your selection" message will appear, as shown in Figure 1.12.



**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.



**Figure 1.13 Saving File with Extension xlsm**

In addition, Visual Basic Editor also allow you to insert a module and a UserForm 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.

## 1.3 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 spreadsheet. 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 1.3

```
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 1.4

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 is as shown in Figure 1.14.

	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	
7				
8				
9				
10				
11	CommandButton1			
12				

**Figure 1.14**

## Example 1.5

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

N10	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				CommandButton1
11				
12				

Figure 1.15

## 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 spreadsheet. 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 1.6

```
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



**Figure 1.16**

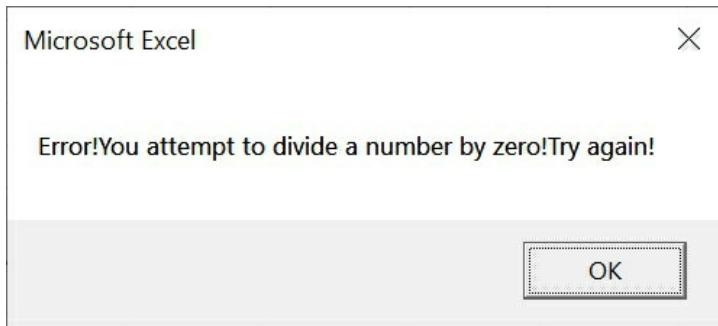
### Example 1.7 Nested Errors Handling

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 follows:

```
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
```



**Figure 1.17**



**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. The numbers can be manipulated arithmetically with various standard operators such as plus, minus, multiply, divide and more. 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

```
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

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
<b>Gtech Pvt Ltd Payroll</b>			
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	
10			
11		CommandButton1	
12			
13			
14			

**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

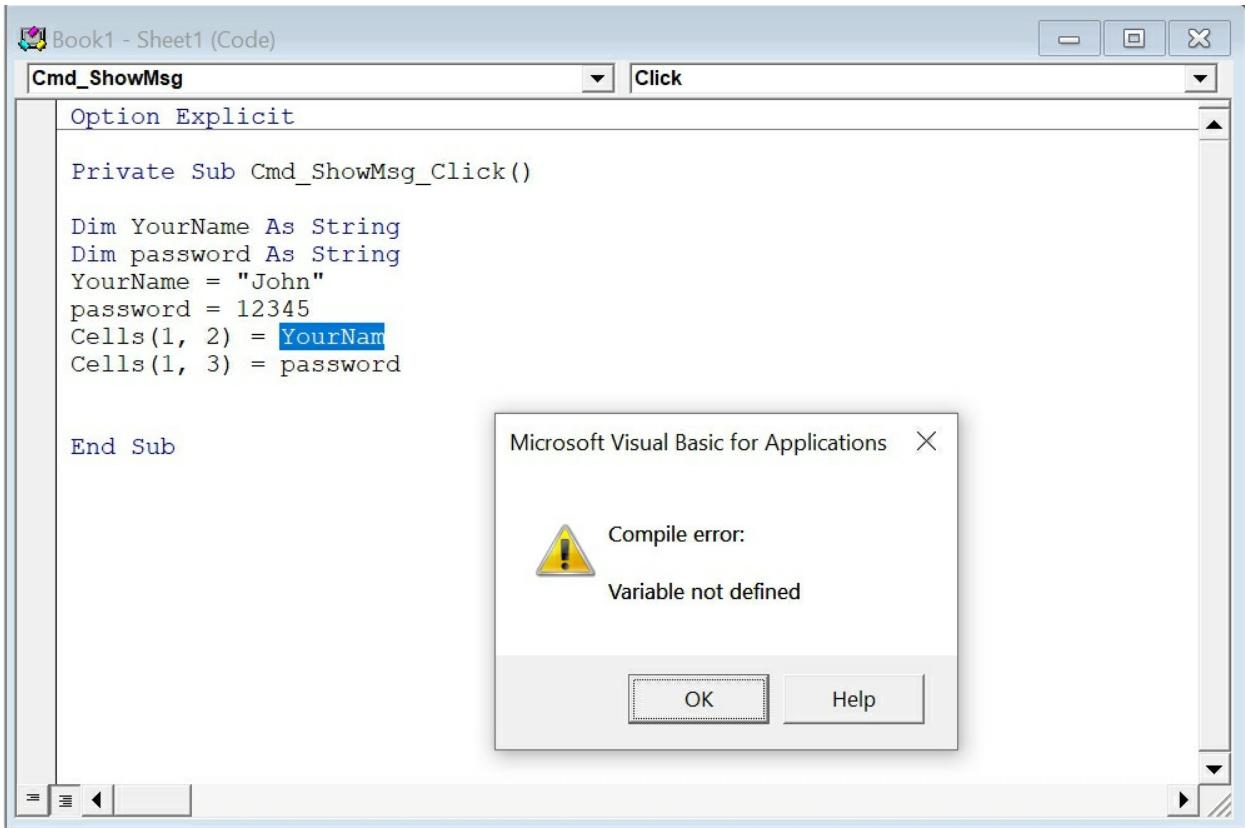
This example uses the Option Explicit keyword and it demonstrates how a typo is being tracked.

```
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.



**Figure 2.2**



**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,  $27\backslash 5$  is 5. The Excel VBA 365 arithmetic operators as shown in Table 2.3.

**Table 2.3 Arithmetic Operators**

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

### Example 2.4

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)))
```

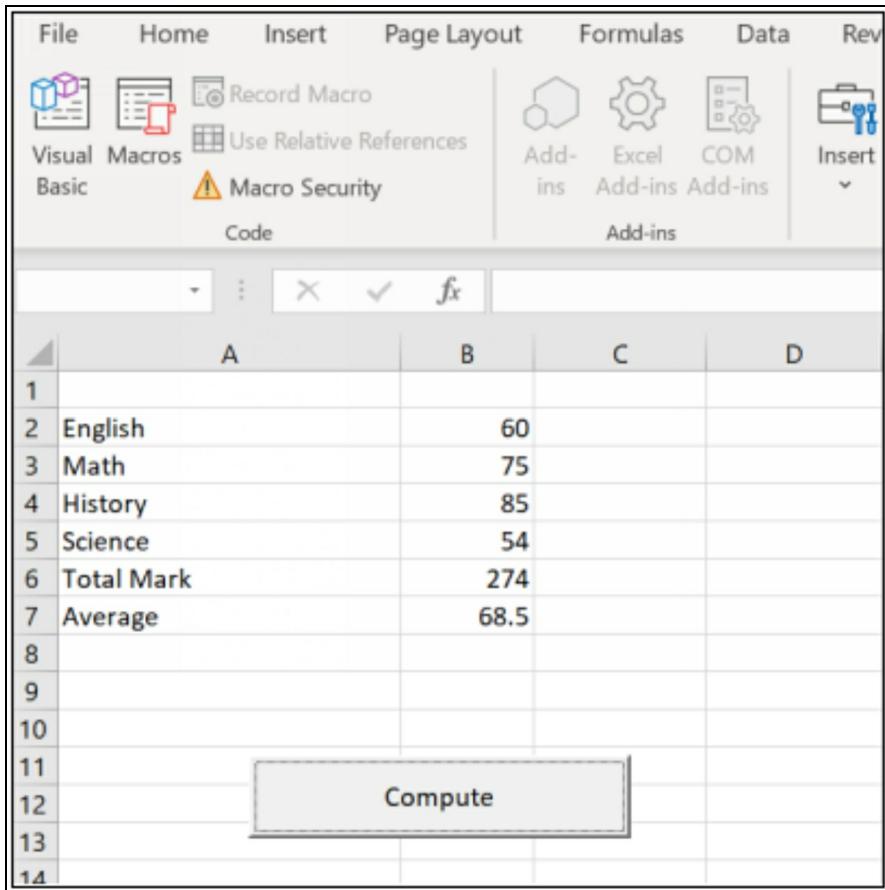
```
average = total / WorksheetFunction.Count(Range(Cells(2, 2), Cells(5, 2)))
```

```
Cells(6, 2) = total
```

```
Cells(7, 2) = average
```

```
End Sub
```

The output is shown in Figure 2.4



**Figure 2.4**

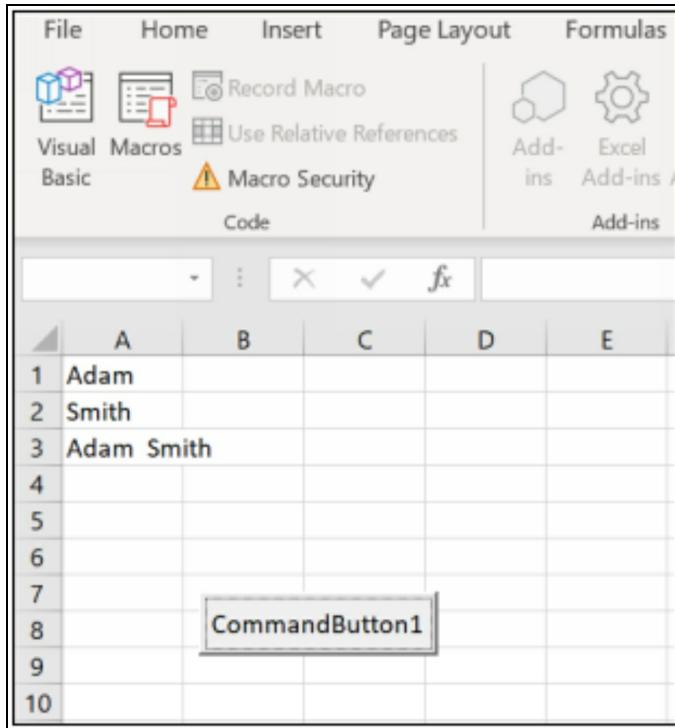
### Example 2.5

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 below

#### 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
```



**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, we can use the following statement to declare 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
```

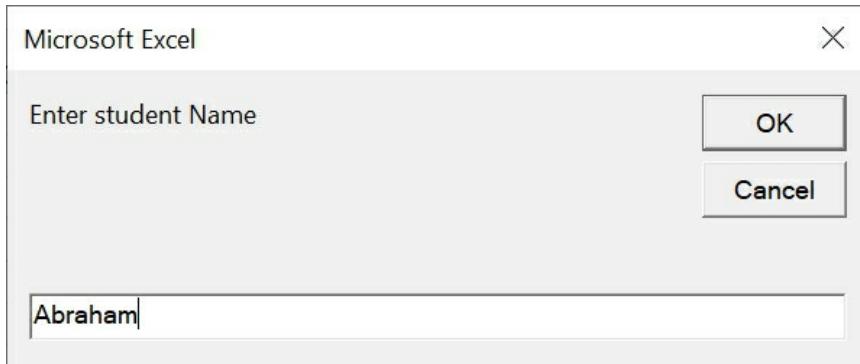
### Example 2.6

In this example, we define an array StudentName of five strings 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.



**Figure 2.6**

The five names will be displayed in the spreadsheet 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

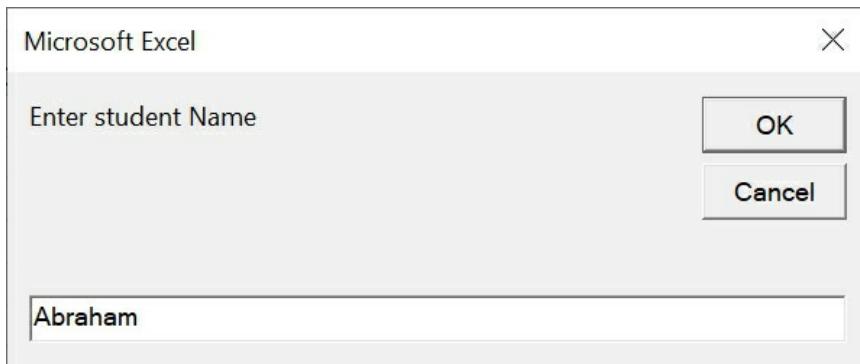
```
Private Sub CommandButton1_Click()
Dim StudentName(3) As String, StudentID(3) As String, StudentMark(3) As Single
For i = 1 To 3 StudentName(i) = InputBox("Enter student Name")
    StudentID(i) = InputBox("Enter student ID")
    StudentMark(i) = InputBox("Enter student Mark")
    Cells(i, 1) = StudentName(i)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

```
Cells(i, 3) = StudentMark(i)
```

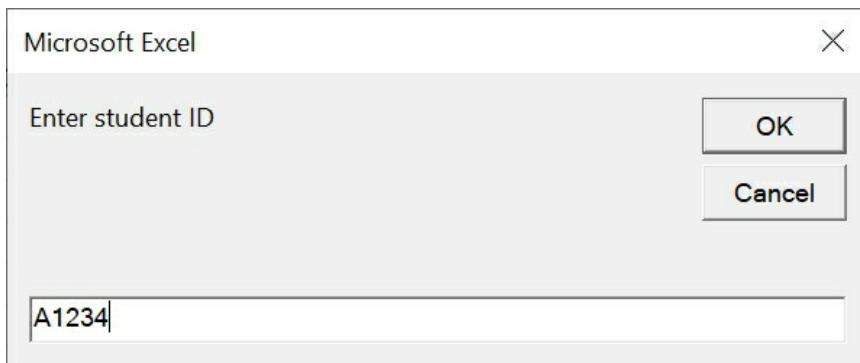
```
Next
```

```
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:



**Figure 2.8**



**Figure 2.9**



**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:

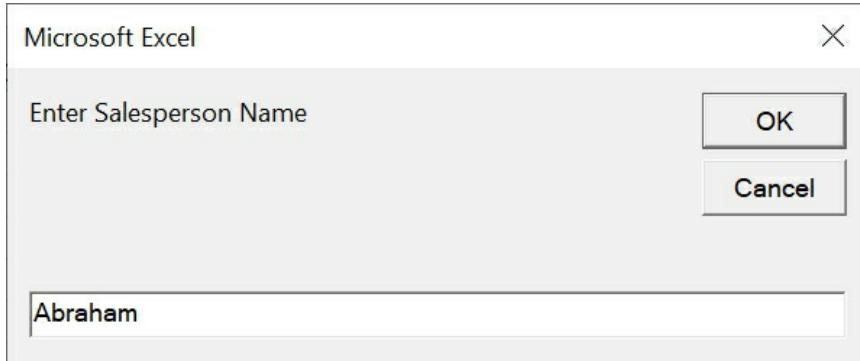
Dim Score(1 to 3, 1 to 3) as Integer

### Example 2.8

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

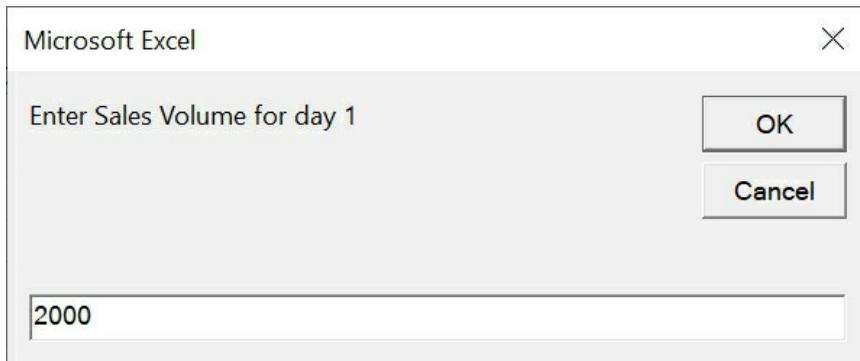
```
Private Sub CommandButton1_Click()
Dim SalesPersonName As String
Dim SalesPersonID, Day As Integer
Dim SalesVolume(2 To 6, 2 To 3) As Double
For SalesPersonID = 2 To 6
    SalesPersonName = InputBox("Enter Salesperson Name")
    Cells(SalesPersonID, 1) = SalesPersonName
    For Day = 2 To 3
        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

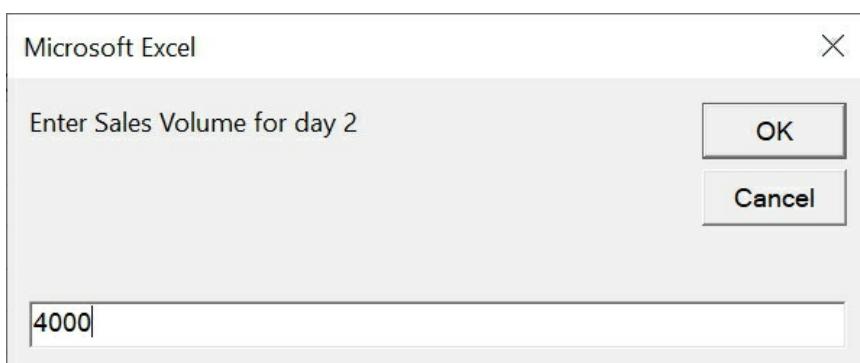


**Figure 2.12**

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.



**Figure 2.13**



**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

# Chapter 3 Message box and Input Box

Excel VBA 365 has many built-in functions. Among these functions, MsgBox() and InputBox() are the two most commonly used . These two functions serve the purpose of interaction with the users. InputBox() lets the user inputs data whereas MsgBox() display a message in dialog box after the user clicks on a button .

## 3.1 The MsgBox ( ) Function

MsgBox() is an Excel VBA 365 function that displays a message in a dialog box, waits for the user to click a button, and returns an Integer indicating which button the user clicked. The MsgBox () function contains three arguments, as follows:

yourMsg=MsgBox(*prompt*,*style value*, *title*)

- prompt - a string expression displayed as the message in the dialog box. The maximum length of *prompt* is approximately 1024 characters, depending on the width of the characters used. If *prompt* consists of more than one line, you can separate the lines by using a carriage return character (Chr(13)) or a linefeed character (Chr(10)) .
- style value- this is a numeric expression that specifies the number and type of buttons to display as well as the icon style to use. This argument is optional. If omitted, the default value for buttons is 0.
- Title argument displays the title of the message box.

Table 3.1 lists the types of button that can be displayed on the message box.

**Table 3.1 Style Values and Type of Button(s)**

<b>Style Value</b>	<b>Named Constant</b>	<b>Type of Button(s) Displayed</b>
0	vbOkOnly	Ok button
1	vbOkCancel	Ok and Cancel buttons
2	vbAbortRetryIgnore	Abort, Retry and Ignore buttons.
3	vbYesNoCancel	Yes, No and Cancel buttons
4	vbYesNo	Yes and No buttons
5	vbRetryCancel	Retry and Cancel buttons

We can use the named constant in place of integers for the second argument to make the programs more readable. In fact, Excel VBA will automatically show a list of named constants where you can select one of them. For example, `yourMsg=MsgBox("Click OK to Proceed", 1, "Startup Menu")` and `yourMsg=Msg("Click OK to Proceed". vbOkCancel,"Startup Menu")` are the same. `yourMsg` is a variable that holds values that are returned by the `MsgBox()` function. The retuned values are determined by the type of buttons being clicked by the users. It must be declared as Integer data type in the procedure or in the general declaration section. Table 3.2 shows the list of returned values, the corresponding named constants and the type of buttons.

**Table 3.2 Returned Values**

<b>Value</b>	<b>Named Constant</b>	<b>Button Clicked</b>
1	vbOk	Ok
2	vbCancel	Cancel
3	vbAbort	Abort
4	vbRetry	Retry
5	vbIgnore	Ignore
6	vbYes	Yes
7	vbNo	No

## Example 3.1

This example demonstrates the use of chr(13) to separate the message into two lines. The named constant is vbOKCancel(can use 1). It also uses the If...ElseIf and Else statement to determine the type of buttons that was clicked.

The code

```
YourMsg = MsgBox("Hi, welcome!" & Chr(13) & "Please click OK to Proceed", vbOKCancel, "Welcome Message")
```

```
If YourMsg = 1 Then
```

```
    MsgBox ("Your click the OK button")
```

```
Else
```

```
    MsgBox ("Your click the Cancel button")
```

```
End If
```

Running the VBA produces the following dialog message, as shown in Figure 3.1.



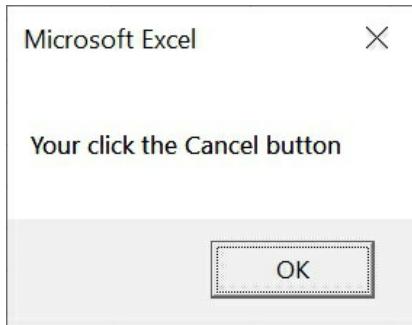
**Figure 3.1**

\*Notice that the message was displayed in two separate lines.

If the user clicks the OK button, it will display another dialog message, indicating the OK button was clicked, as shown in Figure 3.2. On the other hand, if the Cancel button was clicked, the dialog message will indicate the Cancel button was clicked, as shown in Figure 3.3.



**Figure 3.2**



**Figure 3.3**

### Example 3.2

This example illustrates the use of `chr(10)` to separate the message into three lines. The named constant is `vbYesCancel`(can use style value 3). It also uses the `If...ElseIf` and `Else` statement to determine the type of buttons that was clicked and display the relevant messages.

The code

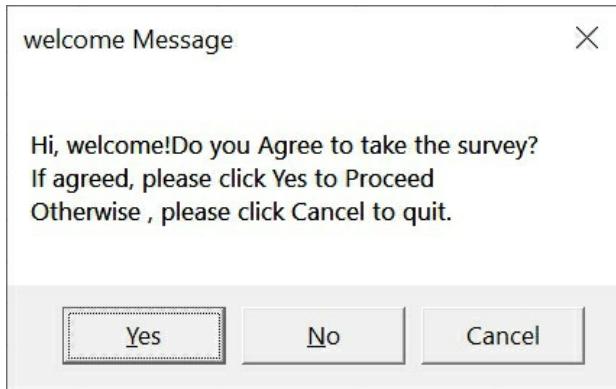
```
Private Sub CommandButton2_Click ()  
YourMsg = MsgBox("Hi, welcome!Do you Agree to take the survey?" &  
Chr(10) & "If agreed, please click Yes to Proceed" & Chr(10) & "Otherwise ,  
please click Cancel to quit.", vbYesNoCancel, "welcome Message")  
If YourMsg = 6 Then  
MsgBox ("Your click the Yes button, click OK to enter the survey.")  
ElseIf YourMsg = 2 Then  
MsgBox ("Your click the Cancel button, please try again.")  
Else
```

```
MsgBox ("Your click the No button, may I know why?")
```

```
End If
```

```
End Sub
```

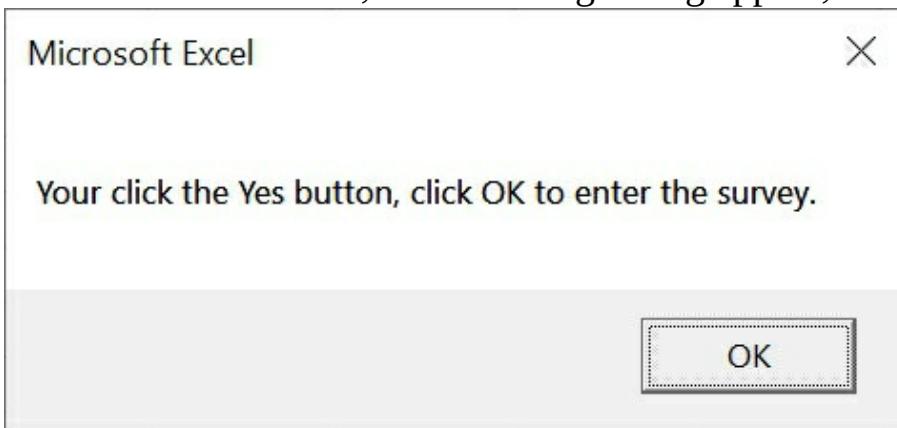
The output



**Figure 3.4**

Notice that the dialog box displayed three buttons, Yes, NO and Cancel. Besides that, the message was displayed in three lines.

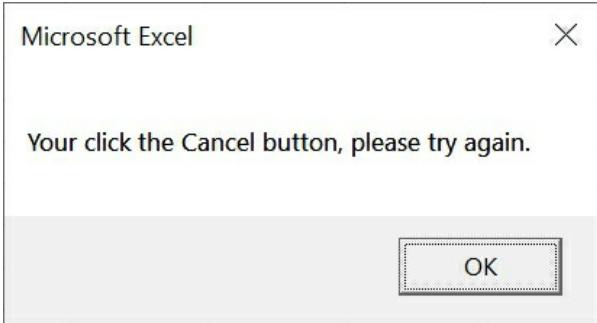
If the user clicked the Yes button, the following dialog appear, as shown in



**Figure 3.5.**

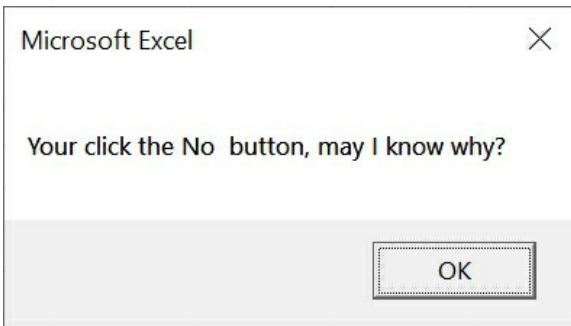
**Figure 3.5**

If the user clicked the Cancel button, the following dialog message will be displayed, as shown in Figure 3.6.



**Figure 3.6**

If the user clicked the No button, the following dialog message will be displayed, as shown in Figure 3.7.



**Figure 3.7**

To make the message box more communicative, you can add an icon by using the icon style value or its named constant. There are four types of icons available in Excel VBA 365, as shown in Table 3.3

**Table 3.3 Icon Style**

<b>Value</b>	<b>Named Constant</b>	<b>Icon Style</b>
16	vbCritical	

3	vbQuestion	
48	vbExclamation	
64	vbInformation	

### Example 3.3

This example is a number guessing game. The input box was added using the InputBox function to accept input from the user. This program also added the named constant vbExclamation as the third argument besides the vbAbortRetryIgnore named constant. The two named constants can be joined using the “+” sign. In addition, we have also added a sub procedure guessNum (which we will discuss in details in a later chapter) to handle the guess number procedure. This guess number procedure generates a random number from 1 to 6 and let the user guess the answer via an input box. Further, it uses the If...Then...ElseIf...Else statements to check the answers and output appropriate responses.

```

Private Sub CommandButton4_Click()
    YourMsg = MsgBox("Do you wish to guess the number?" & Chr(13) &
        "You may get the wrong answer!", vbYesNoCancel + vbExclamation, "Try
        Your Luck")
    If YourMsg = 6 Then
        guessNum
    ElseIf YourMsg = 7 Then
        MsgBox ("Try next Time")
    Else
        MsgBox ("Please try again")
    End If

```

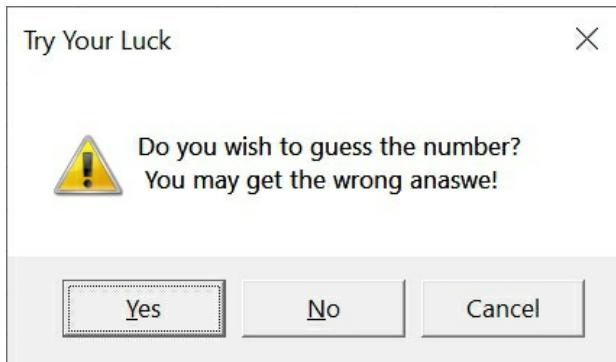
```

End Sub

Sub guessNum()
    Dim ans, myans As Integer
    ans = Int(Rnd * 6) + 1
    myans = InputBox("Enter your answer")
    If myans = ans Then
        MsgBox ("Your answer is right!")
    ElseIf myans <> ans And myans > 0 And myans < 7 Then
        MsgBox ("Your answer is wrong")
    Else
        MsgBox ("Your answer is out of range")
    End If
End Sub

```

Running the program produces the dialog message box as shown in Figure 3.8



**Figure 3.8**

Clicking the Yes button brings up an input box, as shown in Figure 3.9.



**Figure 3.9**

Clicking the OK button produces the final dialog message that tell the answer is wrong, right or out of the range, as shown in Figure 3.10



**Figure 3.10**

## 3.2 The InputBox() Function

An InputBox() is an Excel VBA 365 function that prompt the user to enter a value or a message in the form of text in a box. The syntax is

```
myMessage=InputBox(Prompt, Title, default_text, x-position, y-position)
```

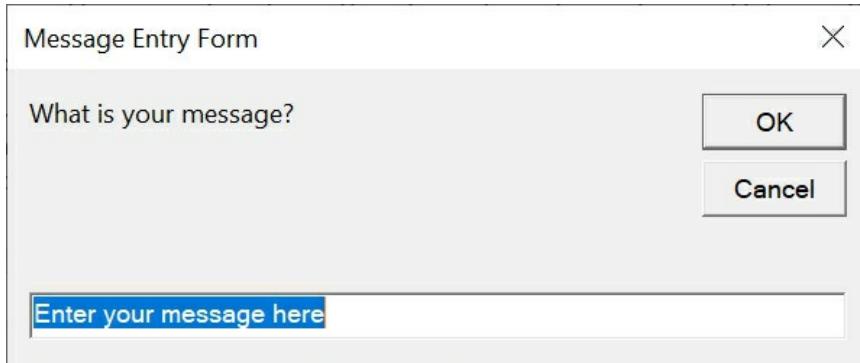
myMessage is a variant data type which accepts the message input by the user. The arguments are explained as follows:

- Prompt - the message displayed in the input box.
- Title - The title of the input box.
- default-text - The default text that appears in the input field where users can use it as his intended input, or he may change it to another message.
- x-position and y-position - the position of the input box.

## Example 3.4

```
Private Sub CommandButton1_Click()  
Dim userMsg As String  
userMsg = InputBox("What is your message?", "Message Entry Form",  
"Enter your message here", 500, 700)  
MsgBox ("Your message is " & userMsg)  
End Sub
```

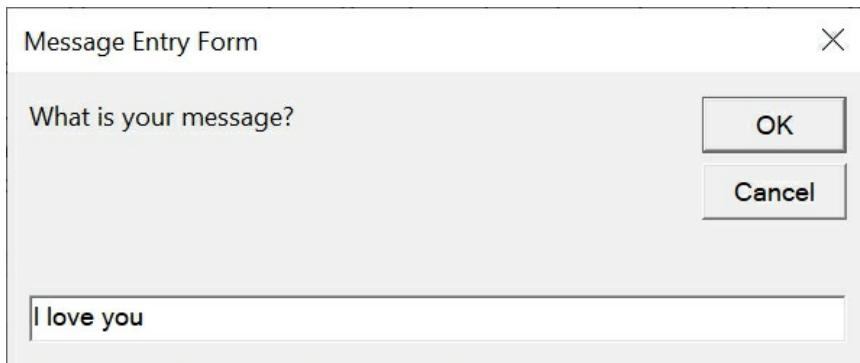
When the user clicks on the OK button, the input box will appear, as shown in Figure 3.11.



**Figure 3.11**

Notice that the default text is "Enter your message here".

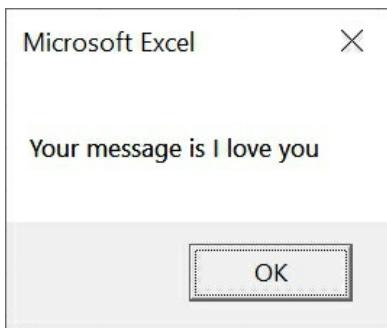
Next, enter the message "I love you", as shown in Figure 3.12.



**Figure 3.12**

Clicking the OK button will pop up a message box that display the message

as shown in Figure 3.13.



**Figure 3.13**

# Chapter 4 Using If...Then...Else

---

In this chapter, you will learn how to create Excel VBA 365 codes that can handle decisions when it processes inputs from the user and then determine the program flow. Excel VBA 365 uses the If ...Then...Else statement, conditional operators and logical operators to control the program flow.

## 4.1 Conditional Operators

To control the Excel VBA 365 program flow, we can use several conditional operators. Basically, they resemble mathematical operators. Conditional operators allow the Excel VBA code to compare some data values and then decide what action(s) to take. For example, it can decide whether to execute or terminate a program. These operators are shown in Table 4.1.

**Table 4.1 Conditional Operators**

Operator	Meaning
=	Equal to
>	More than
<	Less Than
>=	More than and equal
<=	Less than and equal
<>	Not Equal to

\* You can also compare strings with the above operators. However, there are certain rules to follow: Upper case letters are lesser than lowercase letters, "A" < "B" < "C" < "D" ..... < "Z" and numbers are lesser than letters.

## 4.2 Logical Operators

In addition to conditional operators, there are a few logical operators that

enable decision making based on some conditions. These operators are shown in Table 4.2.

**Table 4.2 Logical Operators**

Operator	Meaning
And	Both sides must be true
or	One side or other must be true
Xor	One side or other must be true but not both
Not	Negates truth

### 4.3 Using If...Then...ElseIf... Else

To deal with a more complex Excel VBA 365 program, we shall use the If...Then...ElseIf and Else statements . The syntax is as follows:

```
If conditions Then  
VB expressions  
ElseIf  
VB expressions  
Else  
VB expressions  
End If
```

#### Example 4.1

In this example, the program prompts the user to enter two numbers, then compares their values to determine which number is greater. The result is displayed in a message box. We declare the two numbers as variant to prevent type mismatch error if the user left the input box empty.

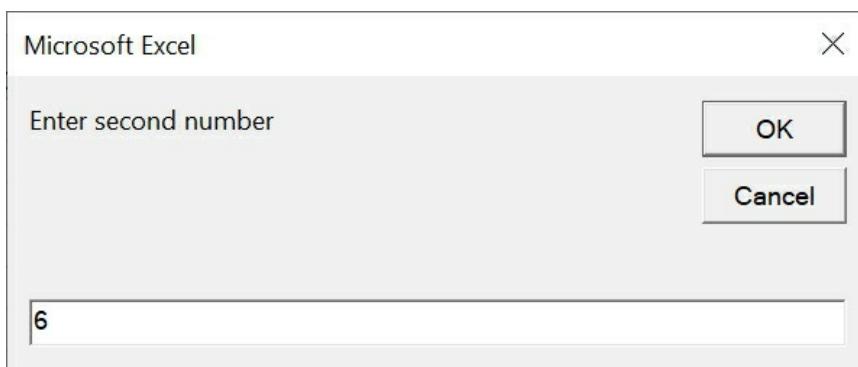
```
Private Sub Cmd_CheckNum_Click()  
Dim firstNum, secondNum As Variant  
firstNum = InputBox("Enter first number")
```

```
secondNum = InputBox("Enter second number")
If firstNum > secondNum Then
    MsgBox (" The first number is greater than the second number")
ElseIf firstNum < secondNum Then
    MsgBox (" The first number is less than the second number")
Else
    MsgBox (" Two numbers are equal ")
End If
End Sub
```

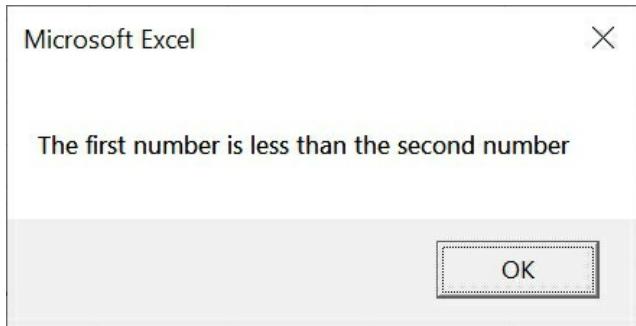
The outputs are illustrated in Figure 4.1, Figure 4.2 and Figure 4.3



**Figure 4.1**



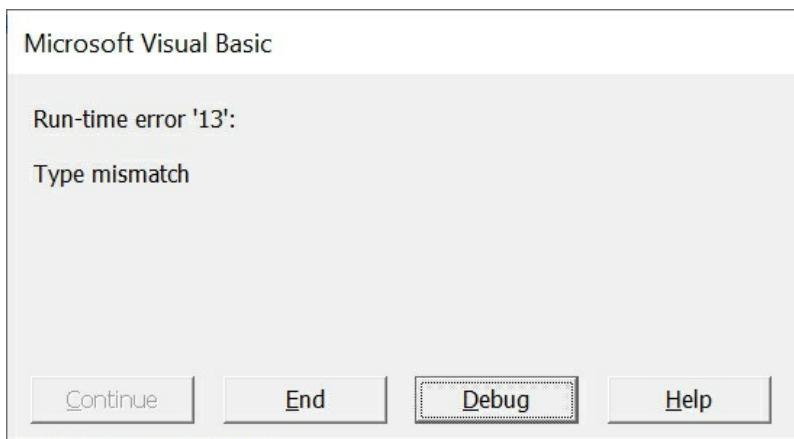
**Figure 4.2**



**Figure 4.3**

## Example 4.2

This Excel VBA 365 computes the grade of the mark entered via an input box. This program employs the If...Then...ElseIf statements to generate the grade and output the result via a message dialog box. We declare the variable mark as a variant to avoid type mismatch error. Let say we declare mark as integer and the user entered a letter or left the input box empty and proceed to click the OK button, the type mismatch error will occur, as shown in Figure 4.4



**Figure 4.4 Type Mismatch**

## The Code

```
Private Sub Cmd_Compute_Grade_Click()
    Dim mark As Variant
    Dim grade As String
```

```

mark = InputBox("Enter the mark")
If mark < 20 And mark >= 0 Then
grade = "F"
ElseIf mark < 30 And mark >= 20 Then
grade = "E"
ElseIf mark < 40 And mark >= 30 Then
grade = "D"
ElseIf mark < 50 And mark >= 40 Then
grade = "C-"
ElseIf mark < 60 And mark >= 50 Then
grade = "C"
ElseIf mark < 70 And mark >= 60 Then
grade = "C+"
ElseIf mark < 80 And mark >= 70 Then
grade = "B"
ElseIf mark <= 100 And mark > 80 Then
grade = "A"
Else
MsgBox ("The mark is out of range")
grade = "Not Defined"
End If
MsgBox ("Your grade is " & grade)
End Sub

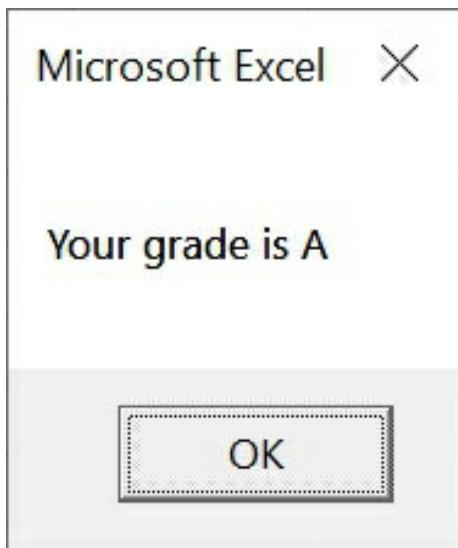
```

Running the program produces and input box, as shown in Figure 4.5.



**Figure 4.5**

If the user entered 80, the result will be A, as shown in Figure 4.6



**Figure 4.6**

### Example 4.3

This example demonstrates the use of the `Not` operator.

```
Private Sub CommandButton1_Click()
Dim x, y As Integer
x = Int(Rnd * 10) + 1
y = x Mod 2
If Not y = 0 Then
    MsgBox " x is an odd number"
Else
    MsgBox " x is an even number"
End If
End Sub
```

In the example, the `Rnd` function produces random numbers between 0 and 1. Therefore `Rnd*10` produces a random number between 0 and 9. `Int(n)` is a function that returns an integer less than a given number n. Therefore, `Int(Rnd*10)+1` generates random integers between 1 and 10. `Mod` is the operator that returns the remainder when a number is divided by another number. If x is an even number, `x Mod 2` produces a zero. Based on this

logic, if  $x \text{ Mod } 2$  is not zero, it is an odd number; otherwise it is an even number.

# Chapter 5 Looping

We can write an Excel VBA 365 procedure that allows the program to run repeatedly until a set of conditions are met. This procedure is known as looping. Looping is an especially useful feature of Excel VBA 365 because it makes repetitive works easier. There are three kinds of loops in Visual Basic, the For...Next loop , the Do...Loop , and the While...Wend Loop .

## 5.1 For...Next Loop

### 5.1.1 The Single For...Next Loop

The syntax of a single For...Next loop is

```
For counter=startNum to endNum (Step increment)  
    One or more statements  
    Next
```

#### Example 5.1

```
Private Sub CommandButton1_Click()  
    Dim i As Integer  
    For i = 1 To 10  
        Cells(i, 1) = i  
    Next  
End Sub
```

In this example, executing the Excel VBA 365 macro will populate cells(1,1) with the value of 1, cells(2,1) with the value of 2, cells(3,1) with the value of 3.....until cells (10,1) with the value of 10. The position of each cell in the Excel spreadsheet is referenced with cells (i,j) , where i represents the row and j represents the column.

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

CommandButton1

**Figure 5.1 For...Next loop with single step increment**

## Example 5.2

In this example, the step increment is used where the value of i increases by 2 after each loop. Therefore, running the macro will populate alternate cells after each loop. When you run the macro, cells (1, 1) will be populated with the value of 1, cells (2, 1) remain empty, cells (3, 1) populated with value of 3 etc.

```
Private Sub CommandButton1_Click()
Dim i As Integer
For i = 1 To 15 step 2
Cells(i, 1) = i
Next
End Sub
```

	A	B	C	D	E
1	1				
2					
3	3				
4					
5	5				
6					
7	7				
8					
9	9				
10		CommandButton1			
11	11				
12					
13	13				
14					
15	15				
16					

**Figure 5.2 For...Next loop with step increment**

To exit the For...Next loop, you can use the Exit For statement.

### Example 5.3

In this example, the program will terminate once it completed 10 loops.

```

Private Sub CommandButton1_Click ()
Dim i As Integer
For i = 1 To 15
Cells(i, 1) = "Round "&i
If i >= 10 Then
    MsgBox("You have completed 10 rounds, exit now")
    Exit For
End If
Next i
End Sub

```

Clicking the command button produces the following output, as shown in

Figure 5.3

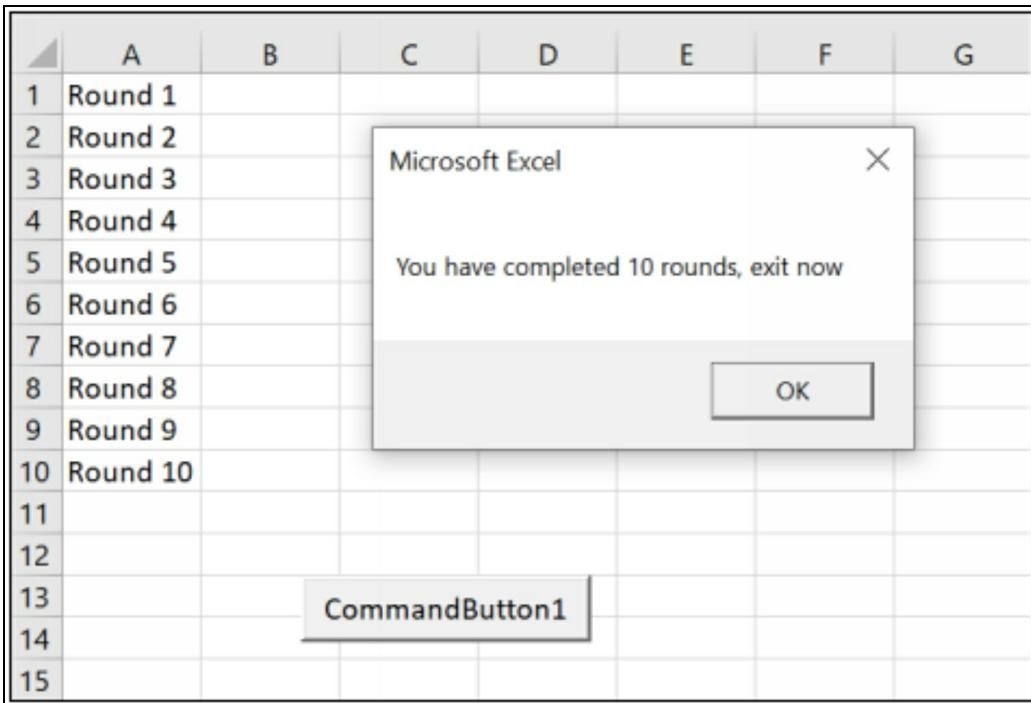


Figure 5.3

### 5.1.2 The Nested For...Next Loop

In previous examples, the For...Next loop will only populate one column or one row only. To populate an entire range of cells, we can use the nested loops, or loops inside loops. This is illustrated in Example 5.4.

#### Example 5.4

```
Private Sub CommandButton1_Click ()  
Dim i, j As Integer  
For i = 1 To 10  
    For j = 1 To 5  
        Cells (i, j) = i + j  
    Next j  
    Next i  
End Sub
```

In this example, when  $i=1$ , the value of  $j$  will iterate from 1 to 5 before it goes

to the next value of i, where j will iterate from 1 to 5 again. The loop will end when i=10 and j=5. In the process, it sums up the corresponding values of i and j, as shown in Figure 5.4

	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						
12						
13						
14						
15	CommandButton1					
16						
17						

**Figure 5.4**

### Example 5.5

This is a counter that can count the number of passes and the number of failures for a list of marks obtained by the students in an examination. The program also differentiates the passes and failures with blue and red colors, respectively. Let us examine the code below:

```
Private Sub Cmd_Analyze_Click()
Dim i, counter As Integer
For i = 1 To 20
If Cells(i, 2) > 50 Then
    counter = counter + 1
    Cells(i, 2).Font.ColorIndex = 5
Else
    'do nothing
    Cells(i, 2).Font.ColorIndex = 3
```

```

End If
Next i
Cells(21, 2).Value = counter
Cells(22, 2).Value = 20 - counter
End Sub

```

This program combines the For...Next loop and the If ...Then...Else statements to control the program actions. If the value in that cell is more than 50, the value of counter is increased by 1 and the font color is changed to blue (ColorIndex = 5) , otherwise there is no increment in the counter and the font color is changed to red (ColorIndex=3).

ColorIndex is a font property that specifies the color of the font. The syntax is

`Cells(i,j).Font.ColorIndex=k`

where k is a color value. For example, if k=3, the color is red and if k=5 the color is blue. The values of ColorIndex are shown in Table 5.2

**Table 5.2 ColorIndex**

<b>Color Value</b>	<b>Color</b>	<b>Color Value</b>	<b>Color</b>
1		6	
2		7	
3		8	
4		9	
5		10	

The output is shown in Figure 5.5

	A	B	C	D	E	F	G
1	Abraham	45					
2	Ben	60					
3	Chan	30					
4	Danish	70					
5	Elaine	85					
6	Fenny	56					
7	Ganesh	68					
8	Hannah	91					
9	Irene	40					
10	Jenkin	25					
11	Kiran	55					
12	Liew	100					
13	Menon	58					
14	Nathan	78					
15	Oprah	88					
16	Peter	95					
17	Robert	33					
18	Satish	43					
19	Tarzan	77					
20	Vinesh	90					
21	Pass	14					
22	Fail	6					
23							

Analyze Marks

Figure 5.5 The Excel VBA counter

## 5.2 The Do...Loop

Another type of looping in Excel VBA 365 is the Do...Loop . There are four different ways you can use the Do Loop, as shown below.

a)

Do While condition

Block of one or more statements

Loop

b)

Do

Block of one or more statements  
Loop While condition

c)

Do Until condition  
Block of one or more statements  
Loop

d)

Do  
Block of one or more statements  
Loop Until condition

### Example 5.6

This program will keep adding 1 to the preceding counter value as long as the counter value is less than 10. It displays 1 in cells (1,1) , 2 in cells(2,1) ..... until 10 in cells (10,1) .

```
Private Sub CommandButton1_Click()
Dim counter As Integer
Do While counter < 10
    counter = counter + 1
    Cells(counter, 1) = counter
Loop
End Sub
```

### Example 5.7

```
Dim counter As Integer
Do counter = counter + 1
    Cells(counter, 1) = counter
Loop While counter < 10
End Sub
```

Example 5.6 and 5.7 produces the same result, as shown in Figure 5.6

	A	B	C	D	E
1	1				
2	2				
3	3				
4	4				
5	5				
6	6				
7	7				
8	8				
9	9				
10	10				
11					
12					
13		CommandButton1			
14					
15					

**Figure 5.6**

### Example 5.8

This program will keep adding 1 to the preceding counter value until the counter value reaches 10. It displays 10 in cells(1,1) , 9 in cells(2,1) ..... until 1 in cells (10,1) .

```
Private Sub CommandButton1_Click()
Dim counter As Integer
Do Until counter = 10
    counter = counter + 1
    Cells(counter, 1) = 11 - counter
Loop
End Sub
```

### Example 5.9

```
Private Sub CommandButton1_Click()
Dim counter As Integer
Do counter = counter + 1
    Cells(counter, 1) = 11 - counter
Loop Until counter = 10
```

```
End Sub
```

Example 5.8 and Example 5.9 produces the same output, as shown below:

	A	B	C	D	E
1	10				
2	9				
3	8				
4	7				
5	6				
6	5				
7	4				
8	3				
9	2				
10	1				
11					CommandButton1
12					

**Figure 5.7**

### Example 5.10

This program will display the values of X in cells(1,1) to cells(11,1). The values of Y are  $X^2$  and the values are displayed in column 2, i.e. from cells(2,1) to cells(2,11). Finally, it shows the values of X+Y in column 3, i.e. from cells(3,1) to cells(3,11).

```
Private Sub CommandButton1_Click()
Dim counter , sum As Integer
'To set the alignment to center
Range("A1:C11").Select
With Selection
    .HorizontalAlignment = xlCenter 'to set alignment to center
End With
Cells(1, 1) = "X"
Cells(1, 2) = "Y"
Cells(1, 3) = "X+Y"

Do While counter < 10
```

```
counter = counter + 1  
Cells(counter + 1, 1) = counter  
Cells(counter + 1, 2) = counter * 2  
sum = Cells(counter + 1, 1) + Cells(counter + 1, 2)  
Cells(counter + 1, 3) = sum  
Loop  
End Sub
```

You will notice that we used the selection procedure

With Selection

End With

This is to format the style of the content in the selected range. Let us examine the following Example.

### Example 5.11

```
Private Sub CommandButton1_Click()  
With Selection  
.HorizontalAlignment = xlCenter 'to set alignment to center  
.Font.ColorIndex = 5  
.Font.Bold = True  
.Font.Italic = True  
.Font.Name = "Verdana"  
.Font.Size = 14  
End With
```

The output

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

**Figure 5.8**

### Example 5.12 Prime Number Tester

This program can test whether a number is a prime number or not. A prime number is a number that cannot be divided by other numbers other than by itself. Examples are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 and more.

In this program, we can use the `Select Case...End Select` statement to determine whether a number is a prime number or not. For case 1, all numbers that are less than 2 are prime numbers. In Case 2, if the number is 2, it is a prime number. In the last case, if the number N is more than 2, we need to divide this number by all the numbers from 3, 4, 5, 6, ..... up to N-1, if it can be divided by any of these numbers, it is not a prime number, otherwise it is a prime number. To control the flow, we can use the `Do.....Loop While` statement. Besides, we need to create a variable tag to store the result of the test. If `tag="Not Prime"`, the number is not a prime number, otherwise it is a prime number.

#### The Code

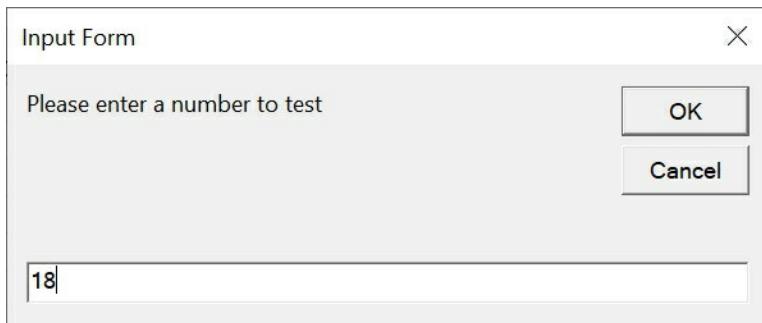
```
Dim N, D As Single
```

```
Dim tag As String
```

```
Private Sub Cmd_Input_Click()
N = InputBox("Please enter a number to test", "Input Form")
End Sub
```

```
Private Sub Cmd_Test_Click()
Select Case N
Case Is < 2
MsgBox ("It is not a prime number")
Case Is = 2
MsgBox ("It is a prime number")
Case Is > 2
D = 2
Do
If N / D = Int(N / D) Then
MsgBox ("It is not a prime number")
tag = "Not Prime"
Exit Do
End If
D = D + 1
Loop While D <= N - 1
If tag <> "Not Prime" Then
MsgBox ("It is a prime number")
End If
End Select
End Sub
```

### The Output



**Figure 5.9**



**Figure 5.10**

### 5.3 The While...Wend Loop

The structure of a While... Wend Loop is remarkably like that of the Do... Loop . The syntax is:

```
While conditions  
    statements  
Wend
```

Please be aware that the While...End While syntax in VB.net is not applicable in Excel VBA 365.

#### Example 5.13

This program generates an arithmetic progression and display the list in a list box. The default of the list box is ListBox1 and you can change it to any name you like. In addition, you can use the list box method AddItem to populate the list.

```
Private Sub Cmd_Generate_Click()  
    MyListBox.AddItem "n" & vbTab & "sum"  
    While n <> 15  
        n = n + 1  
        Sum = Sum + n  
        MyListBox.AddItem n & vbTab & Sum  
    Wend  
End Sub
```

To clear the list box, we use the list box Clear method. The procedure is as

shown below:

```
Private Sub Cmd_Clear_Click()
MyListBox.Clear
End Sub
```

The output is shown in Figure 5.11

E	F	G	H	I
n	sum			
1	1			
2	3			
3	6			
4	10			
5	15			
6	21			
7	28			
8	36			
9	45			
10	55			
11	66			
12	78			
13	91			
14	105			
15	120			

**Generate List**      **Clear List**

**Figure 5.11**

To exit the While...Wend loop, we do not have a keyword like Exit Do in the Do...Loop . However, we can use an indirect method by using a flag variable and set it as Boolean. Let us examine the following example.

#### Example 5.14

In this example, we declare status as Boolean and it will act as a flag variable. The initial value of status is set to True . To exist the While...Wend loop, we use the If...Then statement to change the status to False when the loop

reaches a value, in this case 10.

```
Private Sub Cmd_Generate_Click()
MyListBox.AddItem "n" & vbTab & "sum"
Dim status As Boolean
status = True
While status = True
n = n + 1
Sum = Sum + n
If n = 10 Then
status = False
End If
MyListBox.AddItem n & vbTab & Sum
Wend
End Sub
```

The output is as shown in Figure 5.12

E	F	G	H	I
<b>n      sum</b>				
1	1			
2	3			
3	6			
4	10			
5	15			
6	21			
7	28			
8	36			
9	45			
10	55			

**Figure 5.12**

### Example 5.15 Number Guessing Game

This is a number guessing game where the user tries to guess a preset number. The program uses the While...wend loop and a flag variable to control the number of tries the player can attempt. In this example when the number of tries exceeded 10 and failed to get the correct answer, the program will terminate.

#### The Code

```

Dim n, ans As Integer
Dim guessNum as Variant
Dim flag As Boolean
Private Sub Cmd_Start_Click()
flag = True
ans = 48

```

```

While flag = True
try
n = n + 1
If n = 10 And guessNum <> ans Then
MsgBox ("Too many attempts, you lost your chance, bye")
flag = False
ElseIf guessNum <> ans Then
MsgBox ("Attempt " & n & " Keep trying")
ElseIf guessNum = ans Then
MsgBox ("Congratulations! Your got it right")
flag = False
n = 0
End If
Wend
End Sub

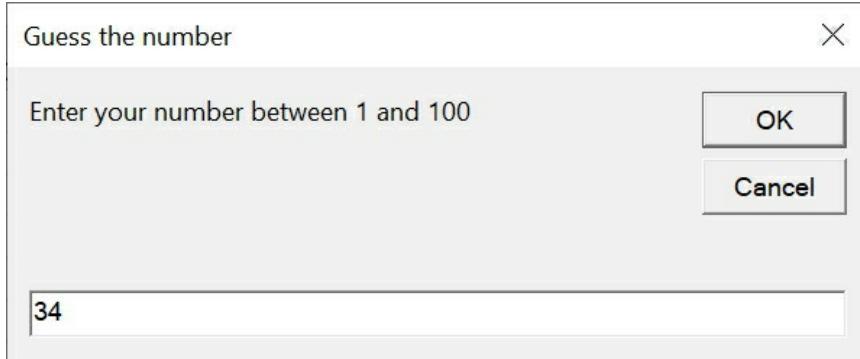
```

```

Sub try()
guessNum = InputBox("Enter your number", "Guess the number")
End Sub

```

When you run the program, an input box will appear, prompting the user to enter a number, as shown in Figure 5.13.



**Figure 5.13**

If the answer is incorrect, the following dialog will appear, as shown in Figure 5.14.



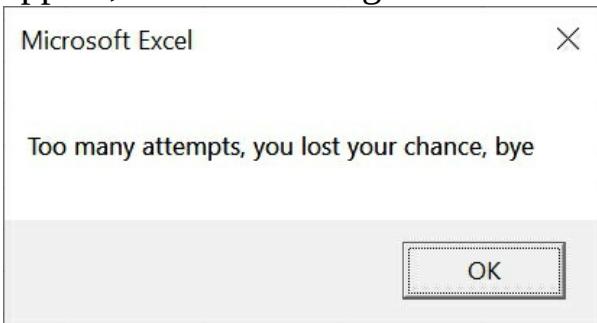
**Figure 5.14**

If the answer is correct, the following dialog message will pop up, as shown in Figure 5.15.



**Figure 5.15**

If the user failed to get the answer after 10 attempts, the following dialog will appear, as shown in Figure 5.16.



**Figure 5.16**

## Chapter 6 Select Case...End Select

Usually, it is enough to use the conditional statement If...Then...Else for multiple options. However, if there are too many different cases, the If...Then...Else structure could become too bulky. Fortunately, Excel VBA provides another way to handle complex multiple-choice cases, that is, the Select Case...End Select decision method. The Select Case...End Select syntax is written as follow:

Select Case variable

Case value 1

    Statement

Case value 2

    Statement

Case value 3

    Statement

.

.

Case Else

End Select

Example 6.1 shows you how to process the grades of students according to the marks. In this example, the user enters a mark via an input box, and he will know his grade via a message box. The output is shown in Figure 6.2

### Example 6.1

```
Private Sub CommandButton1_Click()
Dim mark As Variant
Dim grade As String
mark = InputBox("Enter your mark")
Select Case mark
```

```
Case 0 To 20
grade = "F"
Case 20 To 29
grade = "E"
Case 30 To 39
grade = "D"
Case 40 To 59
grade = "C"
Case 60 To 79
grade = "B"
Case 80 To 100
grade = "A"
Case Else
grade = "Error!"
End Select
MsgBox ("Your grade is " & grade)
End Sub
```



**Figure 6.1**



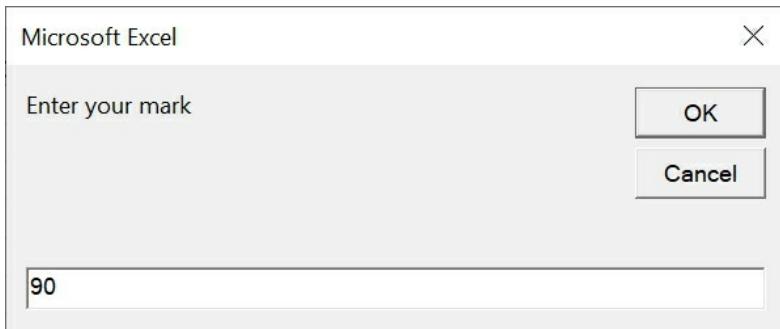
**Figure 6.2**

### Example 6.2

This example uses the `Case Is` keyword to define the Case value.

```
Private Sub CommandButton2_Click()
Dim mark As Single
mark = InputBox("Enter your mark")
Select Case mark
Case Is >= 85
    MsgBox ("Excellence")
Case Is >= 70
    MsgBox ("Good")
Case Is >= 60
    MsgBox ("Above Average")
Case Is >= 50
    MsgBox ("Average")
Case Else
    MsgBox ("Need to work harder")
End Select
End Sub
```

The output is as shown in Figure 6.3



**Figure 6.3**



**Figure 6.4**

### Example 6.3

This is another example that computes the grade. The value for the Case is a string.

```
Private Sub CommandButton1_Click()
Dim grade As String
grade = TxtGrade.Text
Select Case grade
Case "A"
    MsgBox("High Distinction")
Case "A-"
```

```
MsgBox("Distinction")
```

```
Case "B"
```

```
MsgBox("Credit")
```

```
Case "C"
```

```
MsgBox("Pass")
```

```
Case Else
```

```
MsgBox("Fail")
```

```
End Select
```

```
End Sub
```

#### Example 6.4 Number Guessing Game

This is an Excel VBA 365 number guessing game that involves the use of Select Case.

```
Dim Secret_Num As Integer
```

```
Private Sub Cmd_Guess_Click()
```

```
Dim Your_Number As Integer
```

```
Secret_Num = 4
```

```
Your_Number = InputBox("Enter a number between 1 and 10 inclusive")
```

```
Select Case Your_Number
```

```
Case Is < Secret_Num
```

```
MsgBox ("Your number is smaller than the secret number, try again!")
```

```
Case Is > Secret_Num
```

```
MsgBox ("Your number is greater than the secret number, try again!")
```

```
Case Else
```

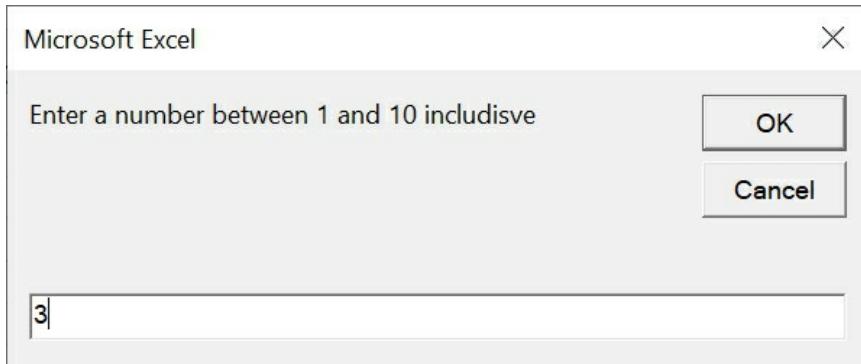
```
Beep 'produces a beep sound
```

```
MsgBox ("Your number is correct, congratulations!")
```

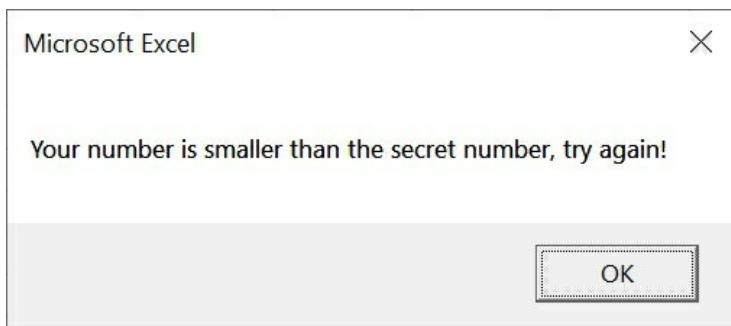
```
End Select
```

```
End Sub
```

The outputs are shown in Figure 6.5 and Figure 6.6



**Figure 6.5**



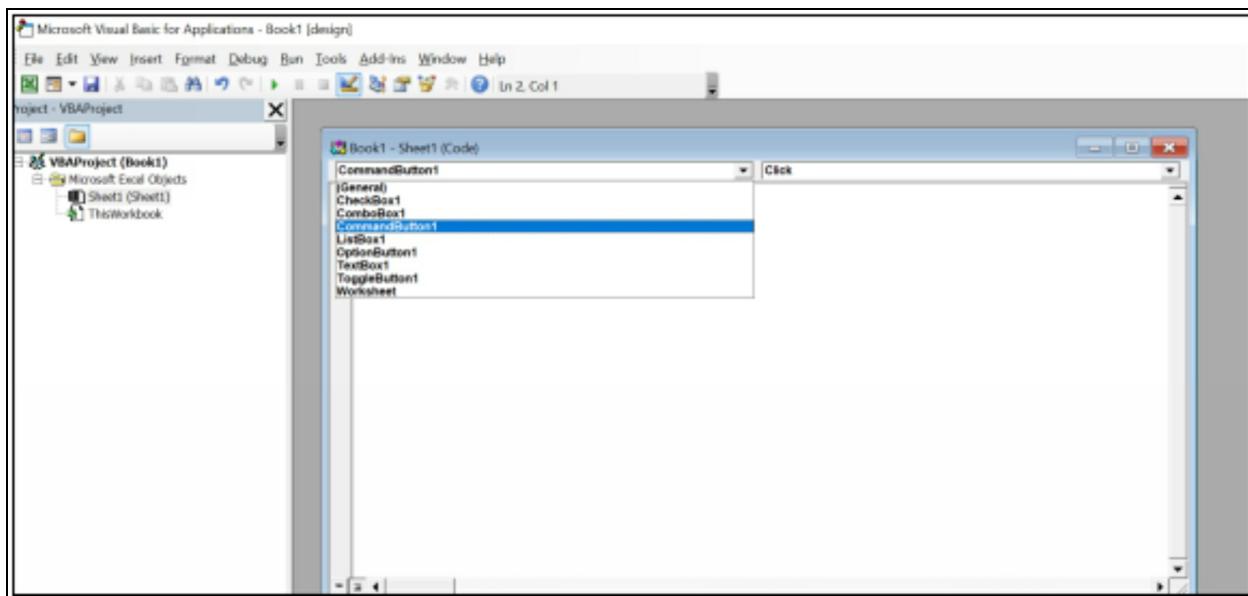
**Figure 6.6**

# Chapter 7: Excel VBA 365 Objects

---

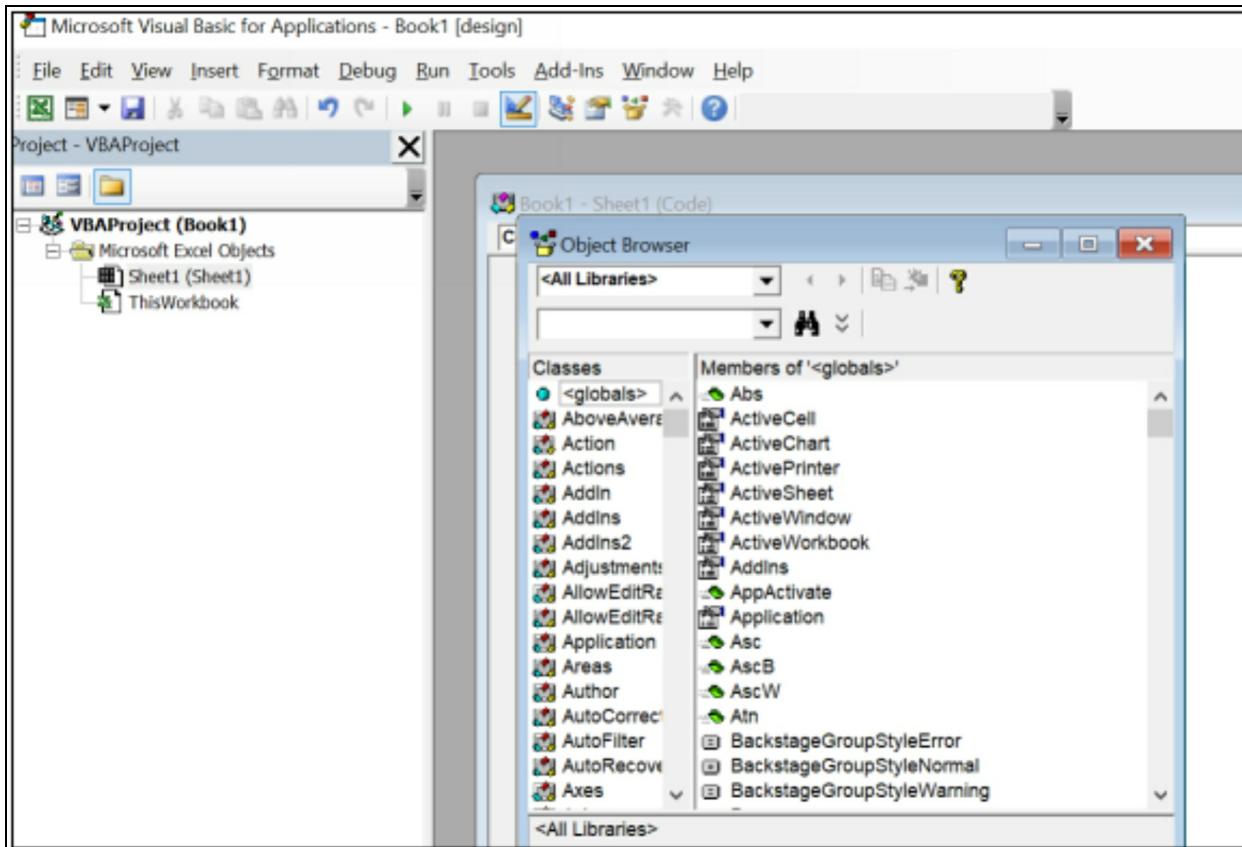
## 7.1: Objects

Excel VBA 365 comprises many objects. For example, an Excel Workbook is an object, Excel Worksheet is an object, a cell in a worksheet is an object, a range of cells is an object, the font of a cell is an object, a command button is an object, and a text box is an object and more. To view the Excel VBA 365 objects, you can insert several objects or controls into the worksheet and click the command button to access the Visual Basic Editor environment. The upper left pane of the code window contains the list of objects you have inserted into the worksheet; you can view them in the dropdown dialog when you click the down arrow, as shown in Figure 7.1. The right pane represents the events associated with the objects.



**Figure 7.1** Excel VBA 365 Objects

To view all the available objects, you can click on the object's browser in the code window, as shown in Figure 7.2.



**Figure 7.2 Objects browser that lists the entire Excel VBA objects.**

## 7.2: Properties and Methods

### 7.2.1 Properties

An Excel VBA object contains properties and methods. Properties are the characteristics or the attributes of an object. For example, Range is an Excel VBA object and one of its properties is value. We can access the property of an object by connecting the object to its property by a period. The following example shows how we can access the property value of the Range object.

#### Example 7.1

```
Private Sub CommandButton1_Click()
    Range("A1:A6").Value = 10
End Sub
```

In this example, by using the value property, we can fill cells A1 to A6 with the value of 10. However, because value is the default property, it can be omitted. Therefore, the above procedure can be rewritten as

## Example 7.2

```
Private Sub CommandButton1_Click()
Range("A1:A6")= 10
End Sub
```

An object can also be a property, it depends on the hierarchy of the objects. For example, Cells is an object, but it is also the property of the range object. The Range object has a higher hierarchy than the Cells object, and the Interior object has a lower hierarchy than the Cells object, so we can write the objects according to their hierarchies:

```
Range("A1:A3").Cells(1, 1).Interior.Color = vbYellow
```

This Interior object will set the background color of cells (1, 1) to yellow. Notice that although the Range object specifies a range from A1 to A3, but the cells property specifies only cells (1,1) to be filled with yellow color, it sorts of overwrites the range specified by the Range object.

The Font object belongs to the Range object and has a few properties, one of the properties is Color . For example,

```
Range("A1:A4").Font.Color=vbYellow
```

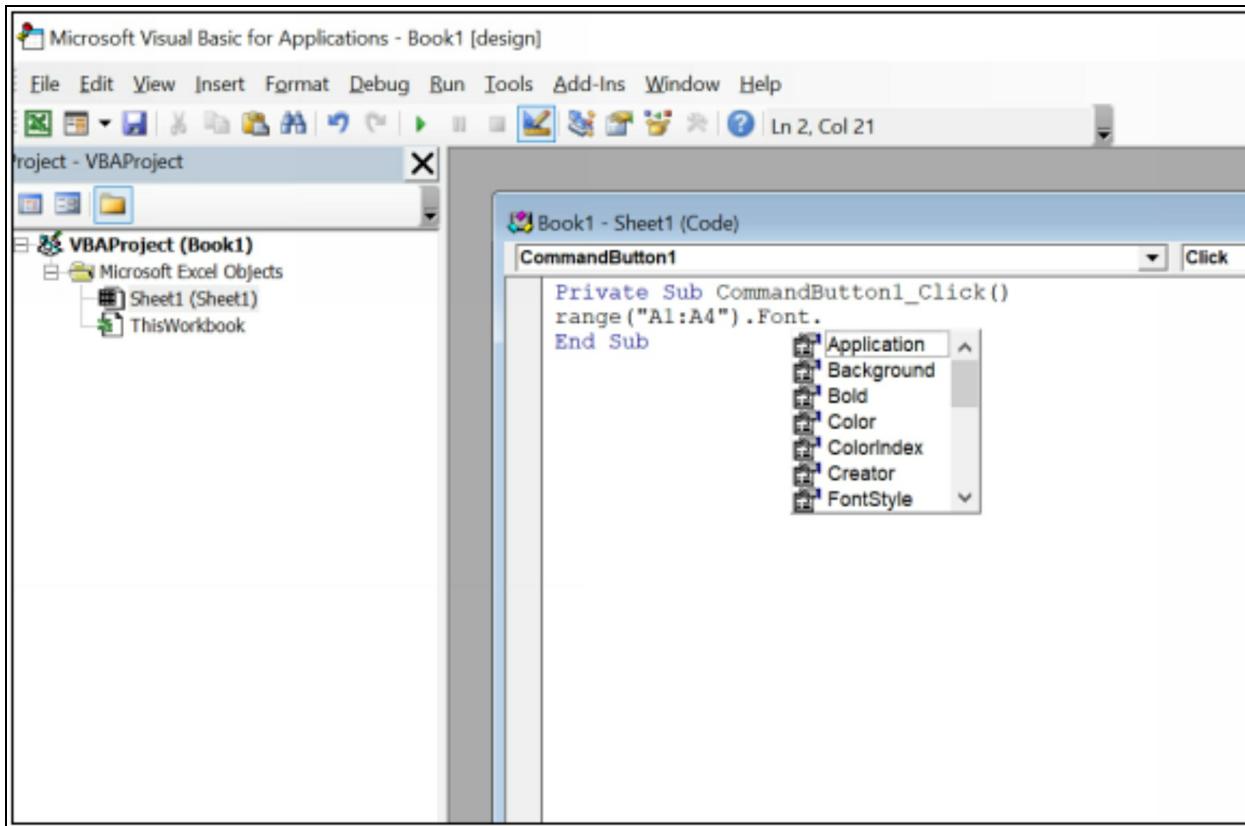
\*vbYellow is one of the eight ColorConstants class, the other colors are vbRed, vbBlue, vbGreen, vbCyan, vbMagenta, vbWhite, vbBlack.

The statement above will change the contents from cell A1 to cell A4 to yellow color. Another property of the Font object is Bold that sets the text to bold style in a cell or a range of cells. For example,

```
Range("A1:A10").Font.Bold=True
```

The statement above will change the contents from cell A1 to cell A10 to

bold. It is not necessary to type the properties because Excel VBA 365 IntelliSense will display a drop-down list of properties as you type a period at the end of the object name. You can then select the property you want by double clicking it or by highlighting it then press the Enter key. The IntelliSense drop-down is shown in Figure 7.3.



**Figure 7.3 The IntelliSense Drop-Down List**

### 7.2.2 Methods

Besides having properties, Excel VBA 365 objects also have methods. Methods are used to perform some operations. We shall discuss some of the methods:

#### a) The Count method

The Count is a method of the Range object. It counts the number of cells in the specified range. For example,

```
Range ("A1:A10").Count
```

returns a value of 10. To display the number of cells returned, you can write the following code.

### Example 7.3

```
Private Sub CommandButton1_Click()
Dim mycounter As Integer
mycounter = Range("A1:A6").Count
MsgBox("The count is" & mycounter)
End Sub
```

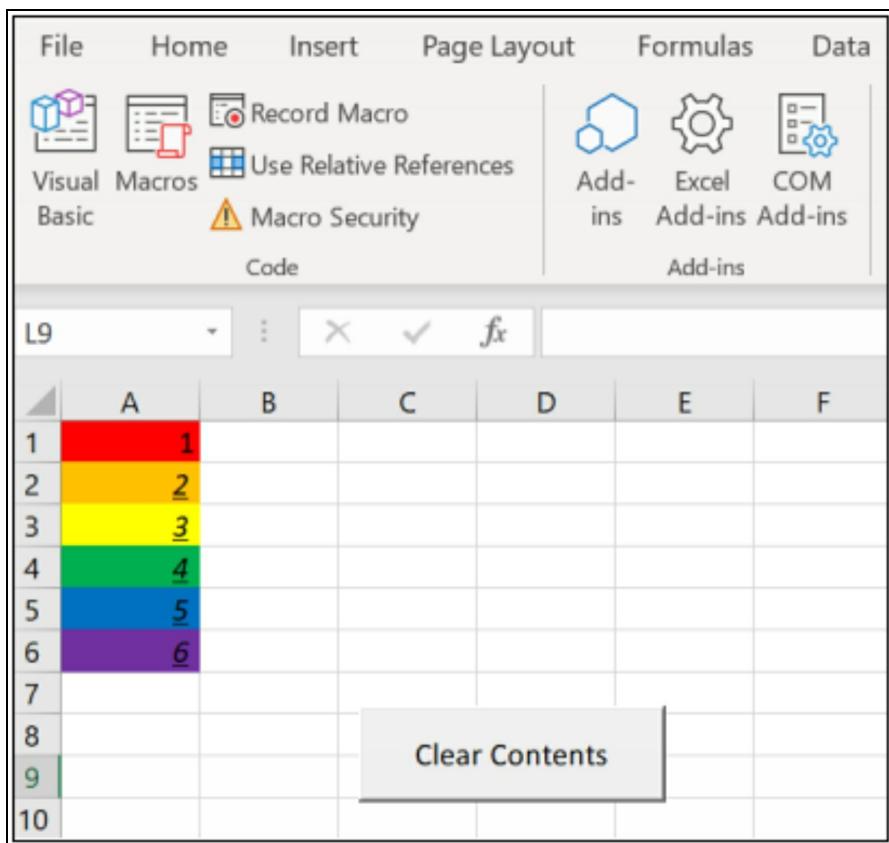
### b) The ClearContents Method

ClearContents is a method of the range object that clears the contents of a cell or a range of cells, without removing the formatting.

### Example 7.4

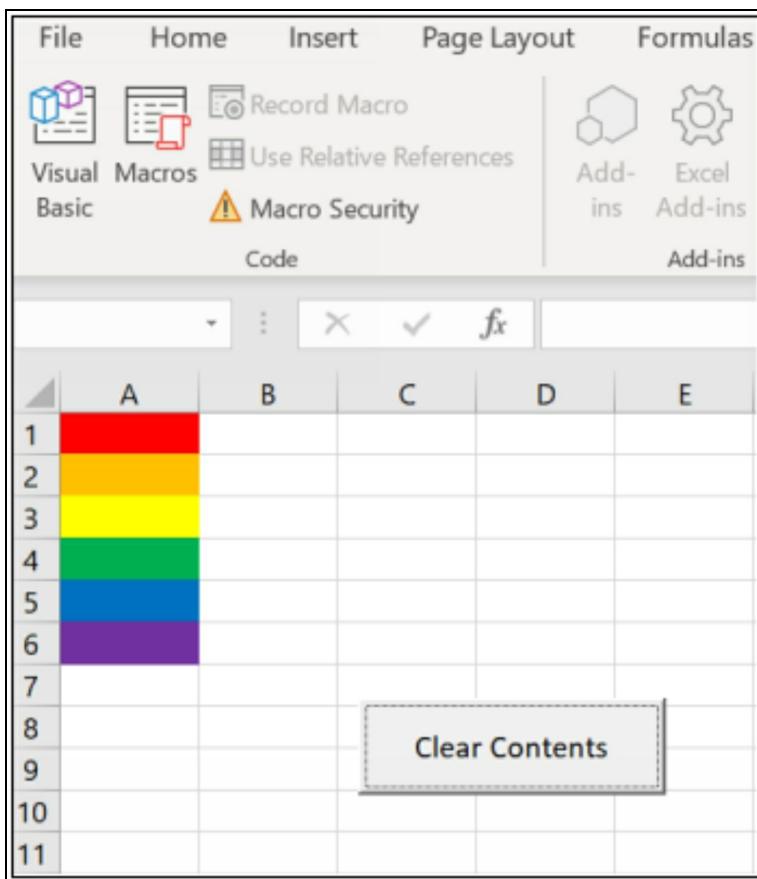
The following Excel VBA 365 code to clear the contents from cell A1 to A6, without removing their formats.

```
Private Sub CommandButton1_Click()
Range("A1:A6").ClearContents
End Sub
```



**Figure 7.4**

When the user clicks clear contents, it will only clear the numbers, but the formatting remains intact, as shown in Figure 7.5



**Figure 7.5**

You can also let the user select his own range of cells and clear the contents by using the `InputBox` function, as shown in Example 7.5. Not only the background color remains intact, if you retype the numbers, they will be bold and italic.

To clear the contents of the entire worksheet, you can use the following syntax

```
Sheet1.Cells.ClearContents
```

### c) The `ClearFormats` Method

If you just want to clear the formats of a range of cells or an entire worksheet, you can use the `ClearFormats` method. The syntax to clear the format of a

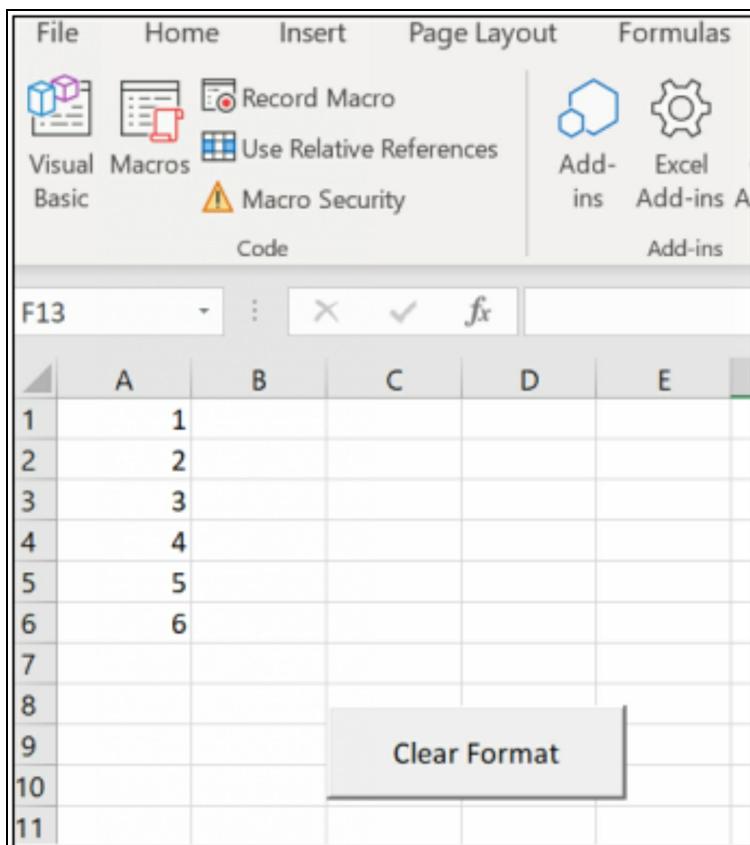
range of cells is as follows :

```
Range(cells(i,j),cells(m,n)).ClearFormats
```

### Example 7.5

We use back the same worksheet as Example 7.4. It will clear the formatting but not the contents, as shown in Figure 7.6.

```
Private Sub Cmd_ClearContents_Click()  
    Range("A1:B6").ClearFormats  
End Sub
```



**Figure 7.6**

To clear the formatting of the entire worksheet, use the following syntax

```
Sheet1.Cells.ClearFormats
```

d) The Clear Method

The clear method clears everything in the specified range of cells, including its contents and its formats. The syntax is

```
Range(cells(i,j),cells(m,n)).Clear
```

To clear the contents and formats of an entire sheet, the syntax is

```
Sheet1.Cells.Clear
```

You can also let the user specify a range of cells via an input box, then clear its contents, as shown in the following example.

### Example 7.6

```
Private Sub CommandButton1_Click()
Dim selectedRng As String
selectedRng = InputBox("Enter your range")
Range(selectedRng).ClearContents
End Sub
```

### e) The Select Method

The Select method selects a range of cells specified by the Range object. The syntax is

```
Range("A1:A5").Select
```

### Example 7.7

```
Private Sub CommandButton1_Click()
Range("A1:A5").Select
End Sub
```

### Example 7.8

This example allows the user to specify the range of cells to be selected.

```
Private Sub CommandButton1_Click()
Dim selectedRng As String
selectedRng = InputBox("Enter your range")
Range(selectedRng).Select
End Sub
```

### Example 7.9

In this example, we insert two command buttons, the first one is to select the range and the second one is to deselect the selected range.

```
Private Sub CommandButton1_Click()
Range("A1:B5").Select
End Sub
```

```
Private Sub CommandButton2_Click()
Range("A1:B5").Clear
End Sub
```

Instead of using the `Clear` method, you can also use the `ClearContents` method.

### f) The Autofill Method

Another particularly useful method is the `Autofill` method. This method performs an autofill in the cells in the specified range with a series of items including numbers, days of week, months of year and more. Formats are copied from the source range to the target range repeating if necessary. The syntax is:

```
Expression.AutoFill (Destination, Type)
```

`Expression` is a variable that represents a `Range` object. `Destination` means the required `Range` object of the cells to be filled. The destination must include the source range. `Type` specifies the fill type like days of week, month of year and more. `Type` is optional, if not specified, Excel 365 will auto detect the trend. `AutoFill` type is shown in Table 7.1

**Table 7.1 Auto Fill Type**

Name	Value	Description
<b>xlFillDefault</b>	0	Excel auto detect the values and formats used to fill the target range.
<b>xFillCopy</b>	1	Copy the values and formats from the source range to the target range.
<b>xlFillDays</b>	5	Extends the names of the days of the week in the source range into the target range.
<b>xlFillWeekdays</b>	6	Extend the names of the days of the workweek in the source range into the target range.
<b>xlFillMonths</b>	7	Extend the names of the months in the source range into the target range.
<b>xlFillYears</b>	8	Extend the years in the source range into the target range.
<b>xlLinearTrend</b>	9	Extend the numeric values from the source range into the target range in an arithmetic sequence.
<b>xlGrowthTrend</b>	10	Extend the numeric values from the source range into the target range in a geometric sequence.

*Source: Adapted from XlAutoFillType enumeration (Excel) by Microsoft*

### Example 7.10

```
Private Sub CommandButton1_Click()
```

```

Range("A1")=1
Range("A2")=2
Range("A1:A2").AutoFill Destination:=Range("A1:A10")
End Sub

```

In this example, the source range is A1 to A2. When the user clicks on the command button, the program will first fill cell A1 with 1 and cell A2 will 2, and then automatically fills the Range A1 to A10 with a series of numbers from 1 to 10. If you do not preset the values in cells A1 and A2, you must key in the values manually cells A1 and A2. The output is shown in Figure 7.7

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

**Figure 7.7**

### Example 7.11

This example uses the Set keyword to define the source range and the destination range.

```

Private Sub CommandButton2_Click()
Range("A1") = 2
Range("A2") = 4
Set SourceRange = Worksheets("Sheet1").Range("A1:A2")
Set destRange = Worksheets("Sheet1").Range("A1:A10")

```

```
SourceRange.AutoFill Destination:=destRange  
End Sub
```

The output is shown in Figure 7.8. Notice that it is an arithmetic sequence.

	A	B	C	D	E
1	2				
2	4				
3	6				
4	8				
5	10				
6	12				
7	14				
8	16				
9	18				
10	20				
11					

**Figure 7.8**

If you add the fill type xlGrowthTrend to Example 7.11 as shown below, you will get a geometric sequence.

```
SourceRange.AutoFill Destination:=destRange, Type:=xlGrowthTrend
```

The output

	A	B	C	D	E
1	2				
2	4				
3	8				
4	16				
5	32				
6	64				
7	128				
8	256				
9	512				
10	1024				
11					
12					

**Figure 7.9**

## Example 7.12

```
Private Sub CommandButton1_Click()
Cells(1, 1).Value = "Monday"
Cells(2, 1).Value = "Tuesday"
Range("A1:A2").AutoFill Destination:=Range("A1:A10"),
Type:=xlFillDays
End Sub
```

In this example, when the user clicks on the command button, the program will first populate cell A1 with “Monday” and cell A2 with “Tuesday”, and then automatically populate the Range A1 to A10 with the days of a week. The output is shown in Figure 7.10

	A	B	C	D	E
1	Monday				
2	Tuesday				
3	Wednesday				
4	Thursday				
5	Friday				
6	Saturday				
7	Sunday				
8	Monday				
9	Tuesday				
10	Wednesday				
11					

**Figure 7.10**

## Example 7.13

This example allows the user to select the range of cells to be automatically filled using the `Autofill` method. This can be achieved with the use of an input box. Since each time we want to autofill a new range, we must clear the contents of the entire worksheet using the `Sheet1.Cells.ClearContents` statement.

```
Private Sub CommandButton1_Click()
```

```
Dim selectedRng As String  
Sheet1.Cells.ClearContents  
selectedRng = InputBox("Enter your range")  
Range("A1") = 1  
Range("A2") = 2  
Range("A1:A2").AutoFill Destination:=Range(selectedRng)  
End Sub
```

We shall discuss more about Excel VBA objects in the next few chapters.

# Chapter 8: The Workbook Object

In the previous chapter, we have learned how to work with objects in general with several examples. We have also learned how to work with properties and methods associated with the objects. In this chapter, we will learn specifically about the `Workbook` object. It is at the top of the hierarchy of all the Excel VBA 365 objects and one of the most used objects.

## 8.1 The Workbook Properties

The keyword to represent `Workbook` object is `Workbooks`. As we are dealing with a collection of workbooks most of the time, therefore using `Workbooks` enables us to manipulate multiple workbooks simultaneously.

When we deal with multiple workbooks, we use indices to denote different opened workbooks, using the keyword `Workbooks(i)`, where `i` is an index. For example, `Workbooks(1)` denotes `Workbook1`, `Workbooks (2)` denotes `Workbook2` and so forth.

Properties are attributes that define a `Workbook`. A workbook has several properties, namely `Name`, `Path`, `FullName` and more. Let us examine the following example:

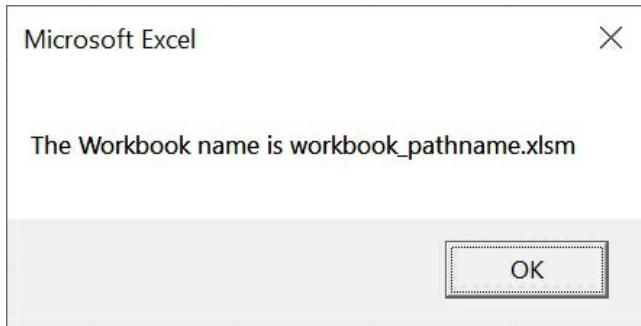
### 8.1.1 The Name Property

The `name` property displays the name of the active `Workbook`.

#### Example 8.1 Displaying the Workbook Name

```
Private Sub CommandButton1_Click()
    MsgBox "The Workbook name is "& Workbooks(1).Name
End Sub
```

The program produces a dialog box that displays the first workbook name, i.e. `Book1`, as shown in Figure 8.1.



**Figure 8.1**

If we have only one open workbook, we can also use the syntax ThisWorkbook in place of Workbook (1) , as follows:

```
Private Sub CommandButton1_Click ()  
    MsgBox ThisWorkbook.Name  
End Sub
```

### 8.1.2 The Path Property

The Path property displays the path of the Workbook file.

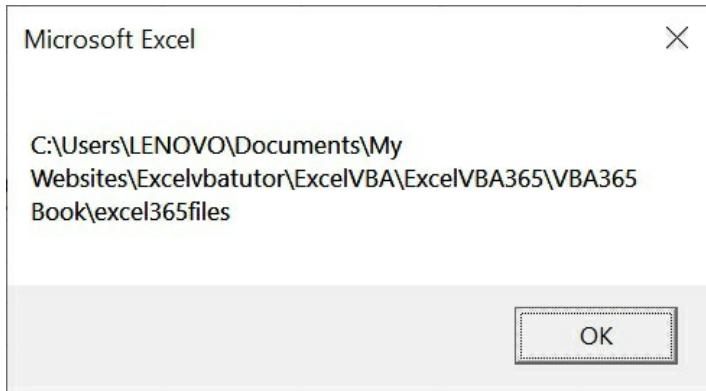
#### Example 8.2 Showing the Path of the workbook

```
Private Sub CommandButton1_Click ()  
    MsgBox ThisWorkbook.Path  
End Sub
```

You can also use the following code

```
Private Sub CommandButton1Click ()  
    folderPath = Workbooks(1).Path  
    MsgBox folderPath  
End Sub
```

The output



**Figure 8.2**

### Example 8.3 Showing the Path and Name of a workbook

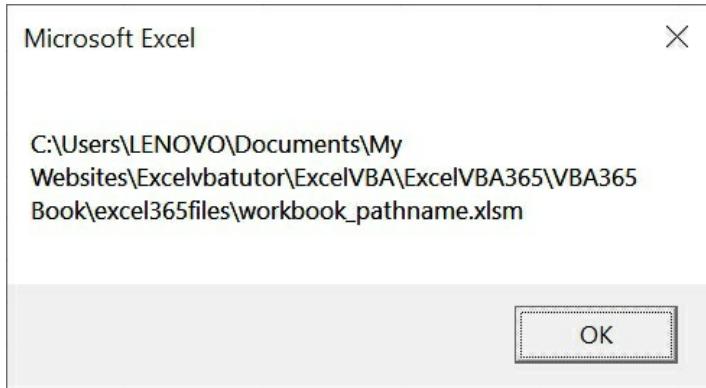
This example will display the path and name of the opened workbook. The code is:

```
Private Sub CommandButton1_Click ()  
    MsgBox ThisWorkbook.FullName  
End Sub
```

Or

```
Private Sub CommandButton1Click()  
    MsgBox Workbooks(1).FullName  
End Sub
```

The output.



**Figure 8.3**

## 8.2 The Workbook Methods

Besides properties, there are several methods associated with the workbook object. Methods enable the Workbook to perform some operations. Some of these methods are Save, SaveAs, Open, Close and more.

### 8.2.1 The Save Method

The save method let the user save the opened workbook without altering the file name.

The syntax is

Workbooks(1).Save

Or

Thisworkbook.Save

#### Example 8.4

```
Private Sub CommandButton1_Click()
ThisWorkbook.Save
End Sub
```

### 8.2.2 The SaveAs Method

The SaveAs method let the user save the file with a new name. The syntax is

```
fName = Application.GetSaveAsFilename
ThisWorkbook.SaveAs
```

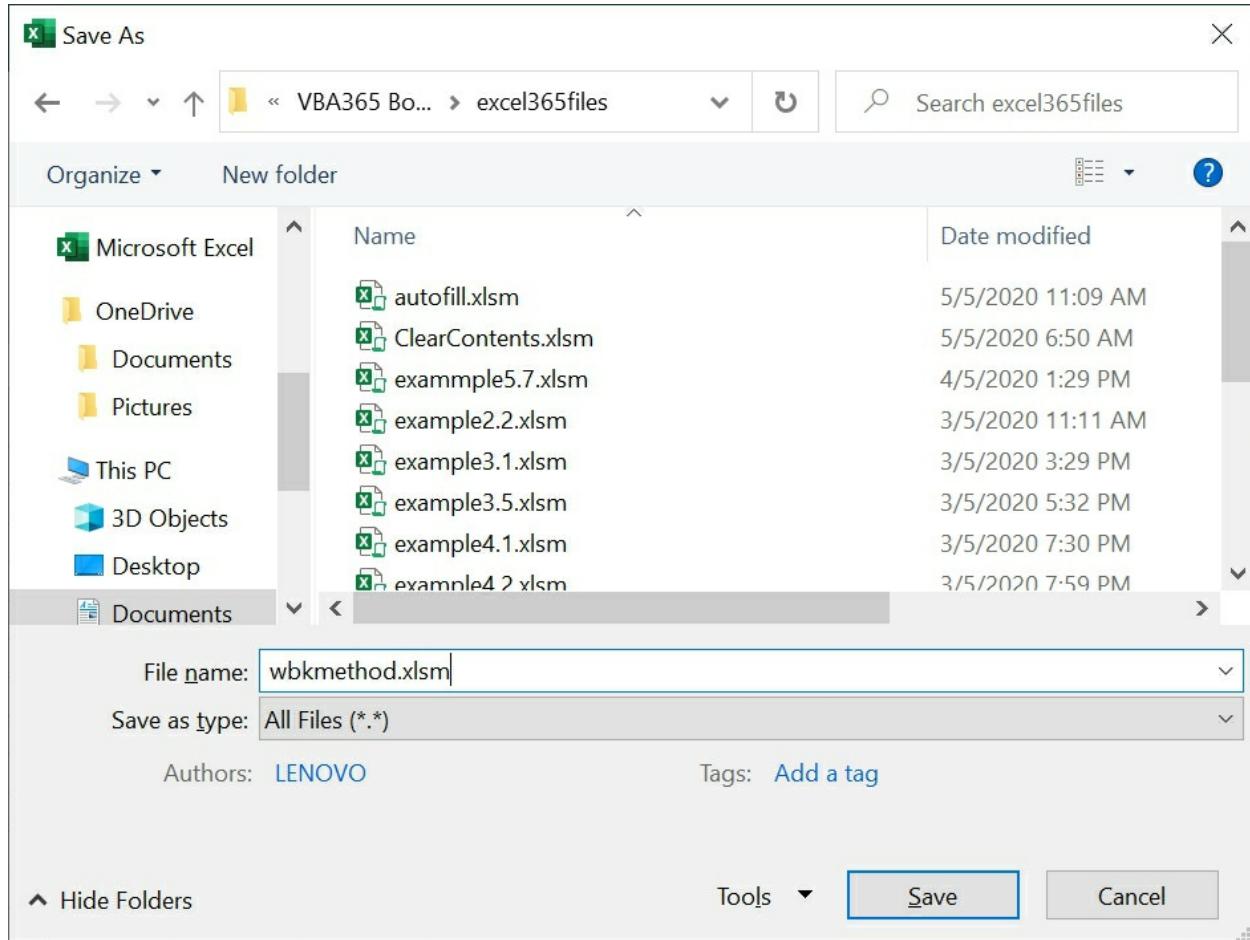
#### Example 8.5

In this example, when the user clicks on the command button, it will open up a dialog box and ask the user to specify a path and type in the file name, and then click the save button, not unlike the standard windows SaveAs dialog, as shown in Figure 8.5.

```
Private Sub CommandButton1_Click()
fName = Application.GetSaveAsFilename
ThisWorkbook.SaveAs Filename:=fName
```

```
End Sub
```

The output is shown in Figure 8.4



**Figure 8.4 The Save As Dialog**

### 8.2.3 The Open Method

The Open method allows the Excel VBA 365 programs to open a file. The syntax is

```
Workbooks.Open ("File Name")
```

#### Example 8.6

In this example, when the user clicks on the command button, it will open the

```
file workbook_object1.xls under the path C:\Users\liewvk\Documents\  
  
Private Sub CommandButton1_Click()  
Dim myfile As String  
myfile = "C:\Users\Documents\ExcelVBA365\autofill.xlsm"  
Workbooks.Open (myfile)  
End Sub
```

Running the program opens the file specified in the code.

#### 8.2.4 The Close Method

The Close method is the command that closes a workbook. The syntax is

```
Workbooks (i).Close
```

#### Example 8.7

In this example, when the user clicks the command button, it will close Workbooks (1).

```
Private Sub CommandButton1_Click()  
Workbooks (1).Close  
End Sub
```

# Chapter 9 The Worksheet Object

## 9.1 The Worksheet Properties

Like the Workbook Object, the Worksheet object has its own set of properties and methods. The ExcelVBA 365 identifier for the Worksheet is Worksheets .

## 9.1 The Worksheet Properties

Some of the common properties of the worksheet are name, count, cells, columns, rows and columnWidth .

### 9.1.1 The Name Property

The name property returns the name of the active worksheet.

#### Example 9.1

```
Private Sub CommandButton1_Click()
    MsgBox Worksheets(1).Name
End Sub
```

The above example pop-ups a dialog that displays the worksheet name as sheet 1 .

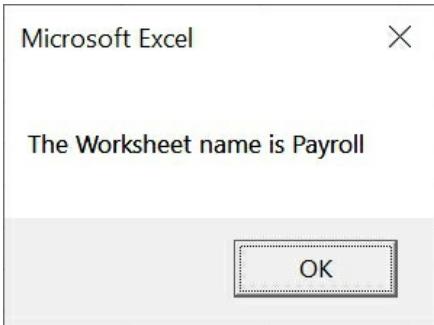


**Figure 9.1**

If you change the default name sheet1 to payroll, the new name will be displayed.

```
Private Sub CommandButton1_Click()
    MsgBox "The Worksheet name is " & Worksheets(1).Name
End Sub
```

The output is as shown in Figure 9.2



**Figure 9.2**

### 9.1.2 The Count Property

The Count property returns the number of worksheets in an opened workbook.

## Example 9.2

```
Private Sub CommandButton1_Click()
MsgBox "Number of Worksheets = " & Worksheets.Count
End Sub
```

The output is as shown in Figure 9.3.



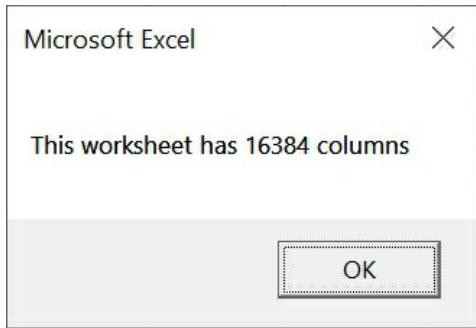
**Figure 9.3**

## Example 9.3

The Count property can also return the number of columns in the worksheet.

```
Private Sub CommandButton2_Click()
Dim countCol As Variant
countCol = Worksheets(1).Columns.Count
MsgBox "This worksheet has " & countCol & " columns"
End Sub
```

\* It is sufficient to write Columns.Count as the worksheet is considered the active worksheet. The output is shown in Figure 9.4.

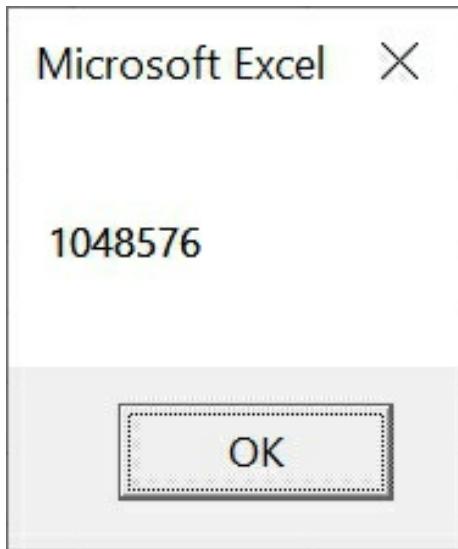


**Figure 9.4**

### Example 9.4

The `Count` property in this example will return the number of rows in the worksheet.

```
Private Sub CommandButton1_Click()
    MsgBox Worksheets(1).Rows.Count
End Sub
```



**Figure 9.5**

## 9.2 The Worksheet Methods

The Worksheet object comprises several methods. Some of the worksheet methods are Add, Delete, Select, SaveAs, Copy, Paste and more.

### 9.2.1 The Add Method

The Add method adds a new Worksheet to the Workbook

#### Example 9.5

When the user clicks the second command button, it will add a new worksheet to the Workbook

```
Private Sub CommandButton1_Click()  
Worksheets.Add  
End Sub
```

### 9.2.2 The Delete Method

The Delete method delete a Worksheet from the Workbook

#### Example 9.6

```
Private Sub CommandButton2_Click()  
Worksheets(1).Delete  
End Sub
```

### 9.2.3 The Select Method

The Select Method select a Worksheet. Besides that, it also work with the Worksheet's properties(objects) Cells, Range, Columns and Rows .

#### Example 9.7

In this example, worksheet 2 will be selected.

```
Private Sub CommandButton1_Click()  
Worksheets(2).Select  
End Sub
```

#### Example 9.8

In this example, cells A1 of Worksheet1 will be selected.

```
Private Sub CommandButton1_Click()
'Cell A1 will be selected
Worksheets (1).Cells (1,1).Select
End Sub
```

### Example 9.9

In this example, cells from range A1 to B6 in Worksheet2 will be selected.

```
Private Sub CommandButton1_Click()
Worksheets(2).Activate
Worksheets(2).Range("A1:B6").Select
End Sub
```

### Example 9.10

In this example, Column1 of woksheet2 will be selected.

```
Private Sub CommandButton1_Click()
'Column 1 will be selected
Worksheets (1).Columns (1).Select
End Sub
```

### Example 9.11

In this example, Row1 of woksheet2 will be selected.

```
Private Sub CommandButton1_Click()
'Row 1 will be selected
Worksheets (1).Rows (1).Select
End Sub
```

## 9.2.4 The Copy and Paste Method

Excel VBA 365 allows us to write code for copy and paste using the Copy and Paste Method.

### Example 9.12

This example copies the contents of cells(1,1) and pastes it to cells(2,2)

```
Private Sub CommandButton1_Click()
```

```
    Worksheets(1).Cells(1,1).Select  
    Selection.Copy  
End Sub
```

```
Private Sub CommandButton2_Click()
```

```
'To paste the content of cell 1 to cell 2
```

```
    Worksheets(1).Cells(2,2).Select
```

```
    ActiveSheet.Paste
```

```
End Sub
```

### Example 9.13

This example copy the contents of a range of cells from A1 to B6 then paste them to another range of cells starting from cell C1.

```
Private Sub CommandButton1_Click()
```

```
    Worksheets(1).Range("A1:B6").Select
```

```
    Selection.Copy
```

```
End Sub
```

```
Private Sub CommandButton2_Click()
```

```
'To paste the content of cell 1 to cell 2
```

```
    Worksheets(1).Range("C1").Select
```

```
    ActiveSheet.Paste
```

```
End Sub
```

# Chapter 10: The Range Object

---

We have discussed quite a bit of the Range object in chapter 7, but we shall study more about the Range object in this chapter.

## 10.1 The Range Properties

The Range object is ranked lower than the Worksheet object in the hierarchy. We can also say that the Worksheet object is the parent object of the Range object. Therefore, the Range object also inherits the properties of the Worksheet object. Some of the common properties of the range object are Columns, Rows, Value and Formula .

### 10.1.1 Formatting

There are many Range properties that we can use to format the cells in the Excel Worksheet. The properties that format the font are Bold, Italic, Underline, Size, Name, FontStyle, ColorIndex and Color . These properties are used together with the Font object.

The Bold , Italic , Underline and FontStyle properties are used to specify the font style. The syntax for using Bold , Italic and Underline are follows:

```
Range ("YourRange").Font.Bold=True  
Range ("YourRange").Font.Italic=True  
Range ("YourRange").Font.Underline=True
```

The FontStyle property can be used to replace some the properties above to achieve the same formatting effects. Acceptable styles are Regular, Italic, Bold, and Bold Italic. The syntax is as follows:

```
Range ("YourRange").Font.FontStyle="Bold Italic"
```

The Name property is used to format the type of font you wish to display in the designated range. The syntax is as follows:

```
Range ("A1:A3").Font. Name = "Time News Roman"
```

The formatting code is illustrated in Example 10.1.

### Example 10.1

```
Private Sub CommandButton1_Click()
Range("A1:A3").Font.Bold = True
Range("A1:A3").Font.Italic = True
Range("A1:A3").Font.Underline = True
Range("A1:A3").Font.Size = 20
Range("A1:A3").Font.Name = "Time News Roman"
End Sub
```

We can use `Fontstyle` to achieve the same effects, as shown in Example 10.2

```
Private Sub CommandButton1_Click()
Range("A1:A3").Font.Underline = True
Range("A1:A3").Font.FontStyle="Bold Italic"
Range("A1:A3").Font.Size = 20
Range("A1:A3").Font.Name = "Time News Roman"
End Sub
```

The `Font` and `ColorIndex` properties are used together to format the font color. You can also use the `color` property to display the font color.

### Example 10.2

```
Private Sub CommandButton2_Click()
Range("A4:B10").Font.ColorIndex = 4
End Sub
```

In this example, the font color will be displayed in green (Corresponding with `ColorIndex =4`).

We can also use the `Color` property to format the font color, the syntax is as follows:

```
Range("A4:B10").Font.Color = VbRed
```

### 10.1.2 The Formula Property

The Formula property is a member of the Range object in Excel VBA 365. We can use it to set a formula for a single cell or range of cells.

#### Example 10.3

```
Private Sub CommandButton1_Click()
Range("A1:B3").Columns(3).Formula = "=A1+B1"
End Sub
```

In this example, the formula A1+B1 will be copied down column 3 (column C) from cell C1 to cell C3. The program automatically sums up the corresponding values down column A and column B and displays the results in column C, as shown in Figure 10.1.

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

**Figure 10.1**

The above example can also be rewritten and produces the same result as below:

```
Range("A1:B3").Columns(3).Formula = "=Sum(A1:B1)"
```

### 10.1.3 The Built-in Formulas

There are many formulas in Excel VBA which we can use to simplify complex calculations. The formulas are categorized into Financial, Mathematical, Statistical, Date and Time and others. For example, in the statistical category, we have Average (Mean), Mode and Median.

#### Example 10.4

In this example, the program computes the average (mean) of the corresponding values down column A and column B and displays the results down column C. For example, the mean of values in cell A1 and Cell B1 is computed and displayed in Cell C1. Subsequent means are automatically copied down Column C until C3.

```
Private Sub CommandButton1_Click()
Range("A1:B3").Columns(3).Formula = "=Average(A1:B1)"
End Sub
```

#### Example 10.5: Mode

In this example, the program computes the mode for every row in the range A1:E4 and displays them in column F. It also makes the font bold and red in color, as shown in Figure 10.2.

```
Private Sub CommandButton1_Click()
Range("A1:E4").Columns(6).Formula = "=Mode(A1:E1)"
Range("A1:E4").Columns(6).Font.Bold = True
Range("A1:E4").Columns(6).Font.ColorIndex = 3
End Sub
```

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

Figure 10.2

### Example 10.6: Median

This program computes the median for every row in the range A1:E4 and displays them in column F. It also makes the font bold and red in color, as shown in Figure 10.3.

```
Private Sub CommandButton1_Click()
Range("A2:E5").Columns(6).Formula = "=Median(A2:E2)"
Range("A2:E45").Columns(6).Font.Bold = True
Range("A2:E5").Columns(6).Font.ColorIndex = 5
End Sub
```

	A	B	C	D	E	F	G	H
1						Mode	Median	
2	12	4	8	8	5	8	8	
3	5	5	7	10	5	5	5	
4	6	7	9	8	6	6	7	
5	8	8	6	9	6	8	8	
6								
7								
8								
9								
10						Calculate Mode	Calculate Median	
11								
12								

**Figure 10.3**

### Example 10.7

In this example, the Interior and the Color properties will fill the cells in the range A1:A3 with yellow color.

```
Private Sub CommandButton1_Click()
Range("A1:A3").Interior.Color = vbYellow
End Sub
```

## 10.2 The Range Methods

The Range method allows the Range object to perform certain operations. It enables automation and performs customized calculations that greatly speed up otherwise time-consuming works if carried out manually.

There are many Range methods which we can use to automate our works. Some of the methods are Autofill, Clear, ClearContents, Copy, cut, PasteSpecial, Select and so forth.

### 10.2.1 Autofill Method

This program allows the cells in range A1 to A20 to be filled automatically

following the sequence specified in the range A1 to A2. The Destination keyword is being used here.

### Example 10.8

```
Private Sub CommandButton1_Click ()  
Set myRange = Range ("A1:A2")  
Set targetRange = Range ("A1:A20")  
myRange.AutoFill Destination: =targetRange  
End Sub
```

### 10.2.2 Select, Copy and Paste Methods

We use the `Select` method to select a specified range, copy the values from that range and then paste them in another range, as shown in the following example:

### Example 10.9

```
Private Sub CommandButton1_Click ()  
Range ("C1:C2").Select  
Selection.Copy  
Range ("D1").Select  
ActiveSheet.Paste  
End Sub
```

\*We can also use the `Cut` method in place of `Copy` in the above example.

### 10.2.2 Copy and PasteSpecial Methods

The `Copy` and the `PasteSpecia l` methods are performed together. The `copy` method will copy the contents in a specified range and the `PasteSpecial` method will paste the contents into another range. However, unlike the `paste` method, which just pastes the values into the target cells, the `PasteSpecial` method has a few more options. The options are `PasteValues` , `PasteFormulas` , `PasteFormats` and `PasteAll` . The `PasteValues` method will just paste the values of the original cells into the targeted cells while the

`PasteFormulas` will copy the formulas and update the values in the targeted cells accordingly.

### Example 10.10

```
Private Sub CommandButton1_Click()
Range("C1:C2").Copy
Range("D1:D2").PasteSpecial Paste:=xlPasteValues
Range("E1:E2").PasteSpecial Paste:=xlPasteFormulas
Range("F1:F2").PasteSpecial Paste:=xlPasteFormats
Range("G1:G2").PasteSpecial Paste:=xlPasteAll
End Sub
```

The output is displayed in Figure 10.4. The original values are pasted to the range D1:D2 while the formula is updated in the range E1:E2 but not the formats. The original formats for the font are bold and red. The formats are reflected in range F1:F2 but the formulas were not pasted there. Lastly, everything is copy over to the range G1:G2.

A screenshot of a Microsoft Excel spreadsheet. The spreadsheet has columns A through H and rows 1 through 11. Row 1 contains values 1, 2, 3, 4, 6, and 6. Row 2 contains values 3, 4, 7, 7, 14, and 14. The cells in row 2 are bold and red. A command button labeled "CommandButton1" is located in cell C8. The status bar at the bottom shows "J8".

	A	B	C	D	E	F	G	H
1	1	2	3	3	6	a	6	
2	3	4	7	7	14	b	14	
3								
4								
5								
6								
7								
8								
9								
10								
11								

**Figure 10.4**

We can also modify the code above and paste them according to the `PasteValues` option and the `PasteFormulas` option, as shown in Example

10.11.

### Example 10.11

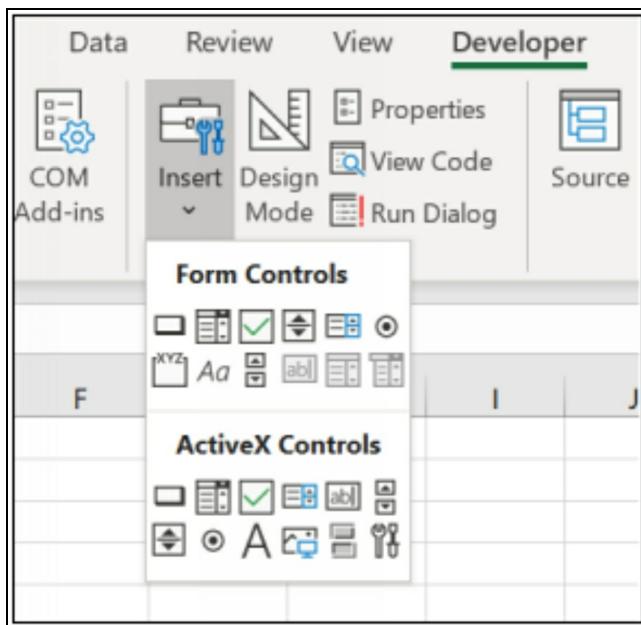
```
Private Sub CommandButton1_Click()
Range("C1:C2").Select
Selection.Copy
Range("D1:D2").PasteSpecial Paste:=xlPasteValues
Range("E1:E2").PasteSpecial Paste:=xlPasteFormulas
End Sub
```

# Chapter 11: Excel VBA Controls

---

Excel VBA 365 Development environment provides several controls that can be used to build Excel VBA 365 applications. There are two categories of controls, the Form controls and the ActiveX controls. The Form controls are easier to use but the ActiveX controls are more powerful and flexible. ActiveX controls allows you to build any applications you wish; the only limit is your imagination.

As these controls are objects, they have their own properties, methods and events. You can view them when you click on the Developer tab and then click Insert, as shown in Figure 11.1



**Figure 11.1**

Some of the most used controls are Command Button, Label, Text Box, List Box, Combo Box, Check Box and Image. Since we have learned how to program the Command Button in previous chapters, we will deal with other controls in this chapter.

## 11.1 Check Box

The Check Box control lets the user selects or unselects an option. One of the most used properties of the Check Box is `Value`. When the Check Box is checked, its value is set to True and when it is unchecked, the value is set to False.

### Example 11.1

In this example, the user can choose to display the sale volume of one type of fruits sold or total sale volume.

```
Private Sub CommandButton1_Click()
If CheckBox1.Value = True And CheckBox2.Value = False Then
    MsgBox "Quantity of apple sold is" & Cells(4, 2).Value
ElseIf CheckBox2.Value = True And CheckBox1.Value = False Then
    MsgBox "Quantity of orange sold is " & Cells(4, 3).Value
Else
    Cells(4, 4) = "=sum(B4:C4)"
    MsgBox "Quantity of Fruits sold is" & Cells(4, 4).Value
End If
End Sub
```

Sales Volume					
	A	B	C	D	E
1					
2					
3		Apple	Orange	Total	
4		100	150	250	
5					
6					
7				✓ Apple	
8					
9				☐ Orange	
10					
11					
12				CommandButton1	
13					
14					

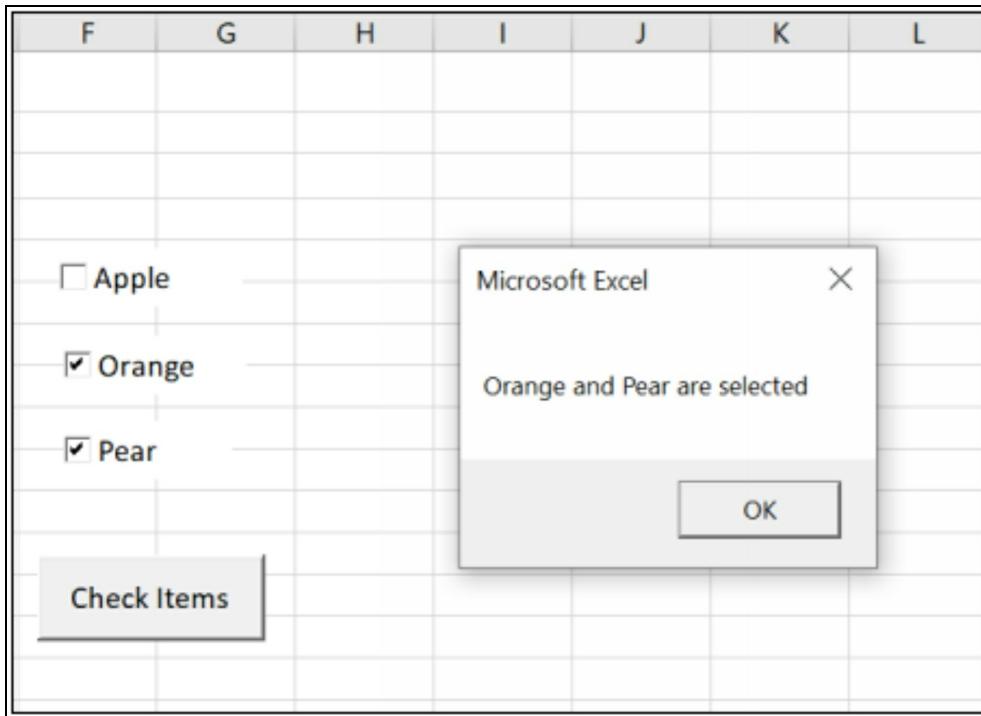
## Figure 11.2 Check Box

### Example 11.2

This example can ascertain the user clicks which check box(es).

```
Private Sub Chk_Check_Click()
If Chk_Apple.Value = True And Chk_Orange.Value = 0 And
Chk_Pear.Value = 0 Then
    MsgBox "Apple is selected"
ElseIf Chk_Orange.Value = True And Chk_Apple.Value = 0 And
Chk_Pear.Value = 0 Then
    MsgBox "Orange is selected"
ElseIf Chk_Pear.Value = True And Chk_Apple.Value = 0 And
Chk_Orange.Value = 0 Then
    MsgBox "Orange is selected"
ElseIf Chk_Orange.Value = True And Chk_Apple.Value = True And
Chk_Pear.Value = 0 Then
    MsgBox "Apple and Orange are selected"
ElseIf Chk_Pear.Value = True And Chk_Apple.Value = True And
Chk_Orange.Value = 0 Then
    MsgBox "Apple and Pear are selected"
ElseIf Chk_Orange.Value = True And Chk_Pear.Value = True And
Chk_Apple.Value = 0 Then
    MsgBox "Orange and Pear are selected"
Else
    MsgBox "All are selected"
End If
End Sub
```

The Output is shown in Figure 11.3



**Figure 11.3**

### Example 11.3

This is a shopping cart that allows the customer to select items with certain quantities.

```
Private Sub Cmd_Cal_Click()
```

```
Const price_orange = 3  
Const price_apple = 4  
Const price_pear = 2  
Const price_grape = 5  
Const price_melon = 2.5  
Const price_lemon = 1.5  
Dim qt_orange As Single  
Dim qt_apple As Single  
Dim qt_pear As Single  
Dim qt_grape As Single  
Dim qt_melon As Single  
Dim qt_lemon As Single  
Dim sum As Single
```

```

qt_orange = Cells(6, 5)
qt_apple = Cells(8, 5)
qt_pear = Cells(10, 5)
qt_grape = Cells(6, 9)
qt_melon = Cells(8, 9)
qt_lemon = Cells(10, 9)
If Chk_Orange.Value = True Then
    sum = sum + qt_orange * price_orange
End If

If Chk_Apple.Value = True Then
    sum = sum + qt_apple * price_apple
End If

If Chk_Pear.Value = True Then
    sum = sum + qt_pear * price_pear
End If

If Chk_Grape.Value = True Then
    sum = sum + qt_grape * price_grape
End If

If Chk_Melon.Value = True Then
    sum = sum + qt_melon * price_melon
End If

If Chk_Lemon.Value = True Then
    sum = sum + qt_lemon * price_lemon
End If

Lbl_Total.Caption = Format(Str(sum), "$#.00")
End Sub

```

The program also calculate the total bill, as shown in Figure 11.4

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										
4					Quantity(Kg)				Quantity(Kg)	
5										
6		Orange \$3 /Kg		<input checked="" type="checkbox"/>	4	Grape \$5/Kg		<input checked="" type="checkbox"/>	5	
7										
8		Apple \$4/Kg		<input checked="" type="checkbox"/>	2	Melon \$2.50/Kg		<input checked="" type="checkbox"/>	2	
9										
10		Pear \$2/Kg		<input checked="" type="checkbox"/>	3	Lemon \$1.50/Kg		<input checked="" type="checkbox"/>	1	
11										
12										
13					Calculate		Total		\$57.50	
14										
15										
16										

**Figure 7 Shopping Cart**

## 11.2 Text Box

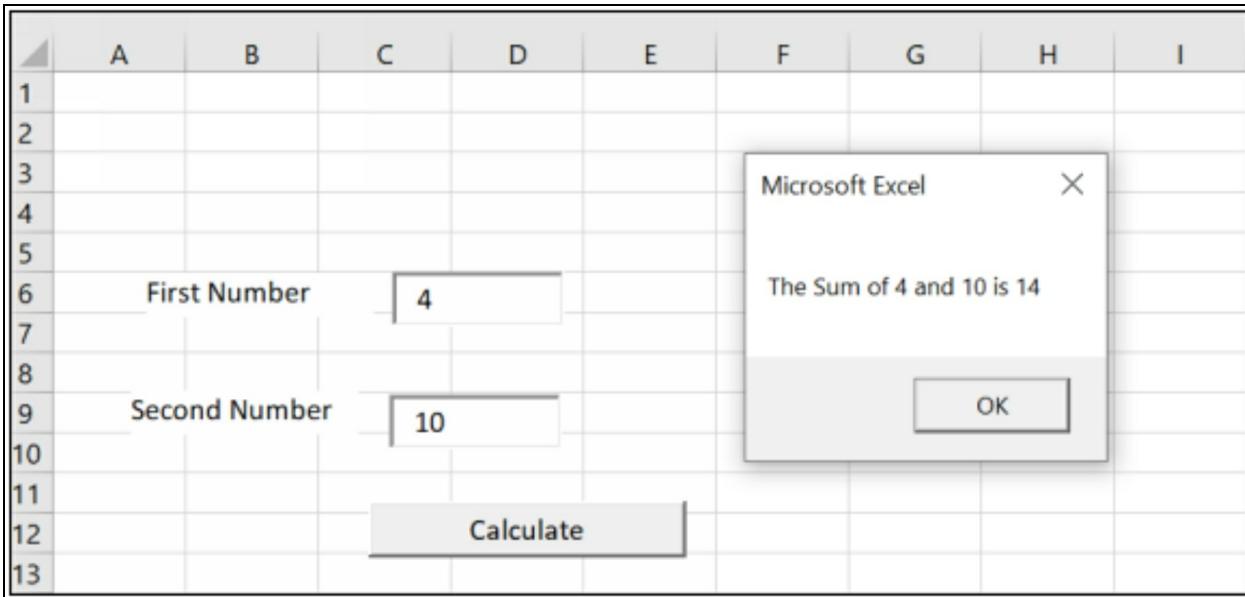
Text Box is the standard control for accepting input from the user as well as to display the output. It can handle string and numeric data but not images.

### Example 11.4

In this example, we inserted two text boxes and displayed the sum of numbers in a message box. The Val function is used to convert string into numeric values because the Textbox treats the number entered as a string.

```
Private Sub CommandButton1_Click ()
Dim x As Variant, y As Variant, z As Variant
x = TextBox1.Text
y = TextBox2.Text
z = Val(x) + Val(y)
MsgBox "The Sum of " & x & " and " & y & " is " & z
```

```
End Sub
```



**Figure 11.5**

## 11.3 Option Button

The option button control also lets the user select one of the choices. Two or more option buttons must work together because as one of the option buttons is selected, the other option button will be unselected. In fact, only one option button can be selected at one time. When an option button is selected, its value is set to “True” and when it is unselected; its value is set to “False”.

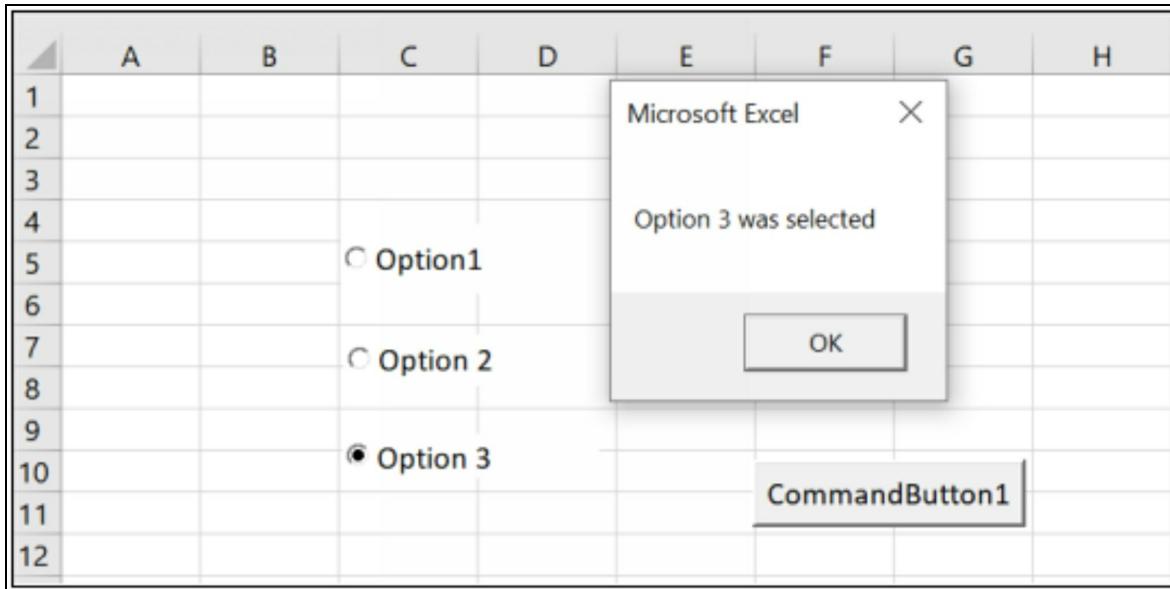
### Example 11.5

This example demonstrates the usage of the option buttons. In this example, the Message box will display the option button selected by the user.

```
Private Sub CommandButton1_Click()
If OptionButton1.Value = True Then
    MsgBox "Option 1 was selected"
ElseIf OptionButton2.Value = True Then
    MsgBox "Option 2 was selected"
```

```
Else
    MsgBox "Option 3 was selected"
End If
End Sub
```

The output is shown in Figure 11.6

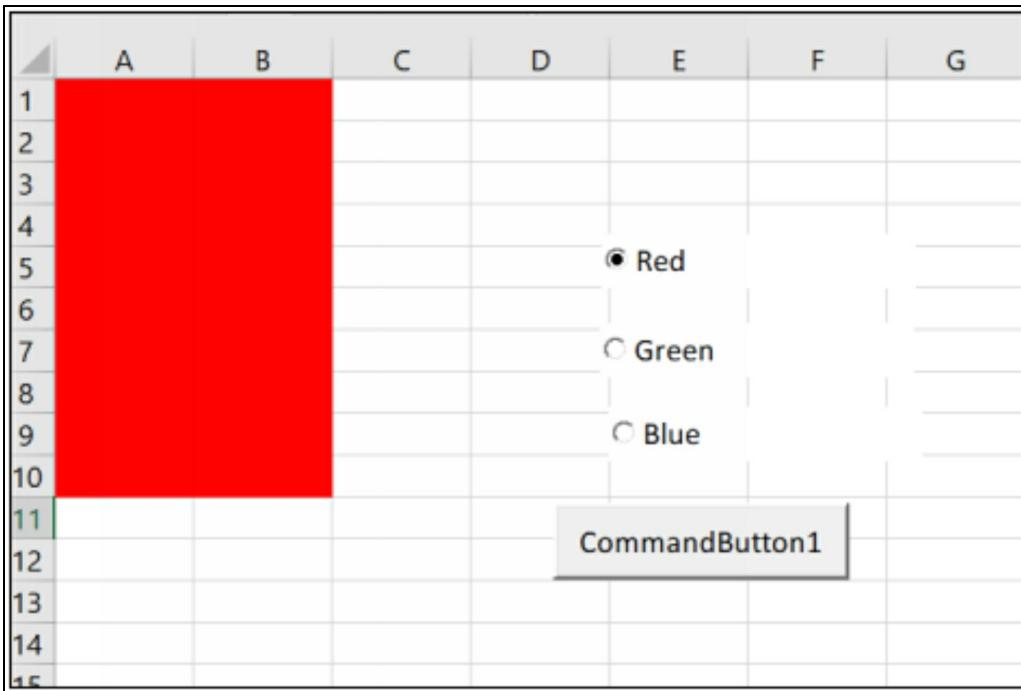


**Figure 11.6 Option Buttons**

### Example 11.6

In this example, If...Then...ElseIf statements are used to control the action when an option button is being selected, i.e., changing the background color of the selected range.

```
Private Sub CommandButton1_Click()
    If OptionButton1.Value = True Then
        Range("A1:B10").Interior.Color = vbRed
    ElseIf OptionButton2.Value = True Then
        Range("A1:B10").Interior.Color = vbGreen
    ElseIf OptionButton3.Value = True Then
        Range("A1:B10").Interior.Color = vbBlue
    End If
End Sub
```



**Figure 11.7 Option Buttons**

### Example 11.7

In this example, the program will change the font color of the item selected, as seen in Figure 11.8

```
Private Sub OptionButton1_Click()
Dim i As Integer
For i = 1 To 12
If Cells(i, 2) = "apple" Then
Cells(i, 2).Font.Color = vbGreen
End If
Next
End Sub
```

```
Private Sub OptionButton2_Click()
For i = 1 To 12
If Cells(i, 2) = "orange" Then
Cells(i, 2).Font.Color = vbRed
End If
Next
```

```
End Sub
```

	A	B	C	D
1	apple			
2	apple			
3	apple			
4	orange			● Apple
5	orange			
6	grape			
7	apple			○ Orange
8	orange			
9	grape			
10	orange			
11	pear			
12	apple			
13				
14				

Figure 11.88

## 11.4 List Box

The function of the List Box is to present a list of items where the user can click and select the items from the list. To add items to the list, we can use the `AddItem` method. To clear all the items in the List Box, you can use the `Clear` method. The usage of `AddItem` method and the `Clear` method is shown Example 11.6.

### Example 11.8

This Excel VBA program adds item "Apple" for even rows and adds item "Orange" for odd rows

```
Private Sub Cmd.AddItem_Click()
Dim x As Integer
```

```
For x = 1 To 11  
If x Mod 2 = 0 Then  
    ListBox1.AddItem "Apple"  
Else  
    ListBox1.AddItem "Orange"  
End If  
  
Next  
End Sub  
Private Sub Cmd_ClearItem_Click()  
For x = 1 To 10  
    ListBox1.Clear  
Next  
End Sub
```

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

**Figure 11.9**

## 11.5 Combo Box

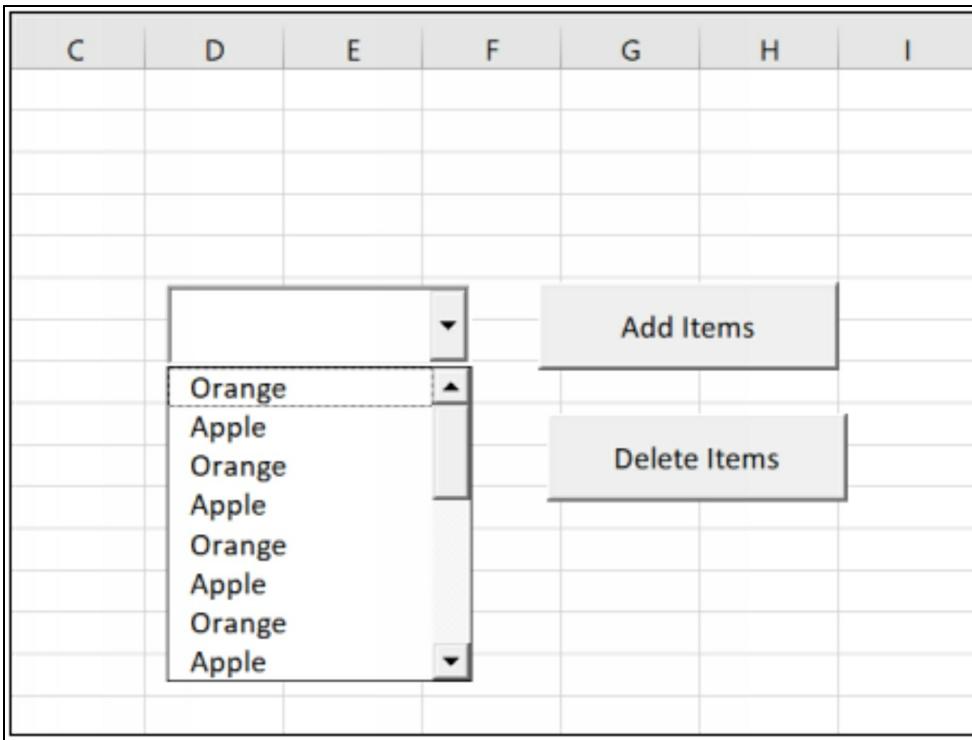
The purpose of the Combo Box is also to present a list of items where the user can click and select the items from the list. However, the user needs to click on the small arrowhead on the right of the combo box to see the items in a drop-down list. To add items to the list, you can also use the AddItem method.

### Example 11.9

This Excel VBA program adds item "Apple" for even rows and adds item "Orange" for odd rows

```
Private Sub CommandButton1_Click()
For x = 1 To 10
If x Mod 2 = 0 Then
    ComboBox1.AddItem "Apple"
Else
    ComboBox1.AddItem "Orange"
End If
Next
End Sub
```

```
Private Sub CommandButton2_Click()
    ComboBox1.Clear
End Sub
```



**Figure 11.10**

## 11.6 Toggle Button

Toggle button lets the user switch from one action to another alternatively. When the Toggle button is being depressed, the value is true and when it is not depressed, the value is false. By using the If and Else program structure, we can thus switch from one action to another by pressing the toggle button repeatedly.

### Example 11.10

In this example, the user can toggle between

```
Private Sub ToggleButton1_Click ()  
    If ToggleButton1.Value = True Then  
        Cells (1, 1) = "Apple"  
        Cells (1, 1).Font.Color = vbRed  
    Else  
        Cells (1, 1) = "Orange"  
        Cells (1, 1).Font.Color = vbBlue
```

End If  
End Sub

# Chapter 12 Functions

---

## 12.1 The Concept of Functions

We have learned about Excel VBA 365 procedures in our previous chapters. However, all of them are event procedures. Event procedures are VBA programs that are associated with Excel VBA objects such as command buttons, check boxes, and option buttons.

In this chapter, we shall learn how to create procedures that are independent from the event procedures. They are usually called into the event procedures to perform certain tasks. There are two types of the procedures, namely Functions and Sub Procedures . In this chapter, we will discuss functions. We will deal with Sub Procedures in the next chapter.

## 12.2 Types of Functions

There are two types of functions; one is the built-in function while the other one is the user-defined function. We can use built-in functions in Excel 365 to perform automatic calculations.

However, built-in functions can only perform some basic calculations, for more complex calculations, user-defined functions are often required. User-defined functions are procedures created independently from the event procedures. A Function can receive arguments passed to it from the event procedure and then return a value in the function name. It is normally used to perform some calculations.

## 12.2 Built-In Functions

Some of the built-in functions are Sum , Average , Min (to find the minimum value in a range), Max (To find the maximum value in a range), Mode , Median and more. We must use the WorksheetFunction object to use the built-in functions. The syntax is as follows:

WorksheetFunction.fn

For example

```
WorksheetFunction.sum(Range("A1:A5"))
```

Another example:

```
Set myRng = Worksheets("Sheet1"). Range("A1:B10")
MinVal= Application.WorksheetFunction.Min(myRange)
MsgBox MinVal
```

### Example 12.1

This example generates a sales report using the built-in functions

```
Private Sub Cmd_Generate_Click()
Dim rng, total, avg, min_val, max_val, median_val, mode_val As Variant
For i = 5 To 12
    rng = Range(Cells(i, 1), Cells(i, 13))
    WorksheetFunction.Sum (Range("A1:B3"))

    total = WorksheetFunction.Sum(rng)
    Cells(i, 14) = total
    avg = Round(WorksheetFunction.Average(rng), 2)
    Cells(i, 15) = avg
    min_val = WorksheetFunction.Min(rng)
    Cells(i, 16) = min_val
    max_val = WorksheetFunction.Max(rng)
    Cells(i, 17) = max_val
    median_val = WorksheetFunction.Median(rng)
```

```

Cells(i, 18) = median_val
mode_val = WorksheetFunction.Mode(rng)
Cells(i, 19) = mode_val
Next
End Sub

```

## The Output

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1																				
2	Sales Report of Hi_Tech Inc. For Year 2019																			
3																				
4		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total	Average	Minimum	Maximum	Median	Mode	
5	Huawei Pro 20	100	300	120	200	300	350	300	250	220	180	170	390	2880	240	100	390	235	300	
6	Huawei Pro 30	200	180	150	230	290	320	380	230	270	230	195	200	2875	239.58	150	380	230	230	
7	Iphone 11	80	180	180	150	190	180	200	220	140	170	120	160	1970	164.17	80	220	175	180	
8	Iphone SE	110	115	200	120	120	117	230	190	150	130	110	170	1762	146.83	110	230	125	110	
9	Samsung Note 10	130	150	120	200	300	200	400	250	220	180	170	390	2710	225.83	120	400	200	200	
10	Samsung Note 10+	210	180	150	230	390	380	380	240	270	190	195	200	3015	251.25	150	390	220	380	
11	Xiaom Redmi 8	180	130	180	180	190	190	200	220	140	170	120	160	2060	171.67	120	220	180	180	
12	Xiaom Redmi Note	210	115	230	120	150	150	230	190	150	130	110	270	2055	171.25	110	270	150	150	
13																				
14																				
15																				
16																				
17																				
18																				

Generate

**Figure 12.1**

## 12.3 User-Defined Function

To create a user-defined function, we use the following syntax:

```

Function FunctionName (arguments) As DataType
    Statements
End Function

```

### Example 12.2

In this example, we shall create a function that calculate the area of a rectangle. This function contains two arguments, one is to accept the value of width and the other is to accept the value of height. Note that the function Area\_Rect is called from the event procedure (clicking the command button) and the values to be passed to the arguments are enclosed in the parentheses.

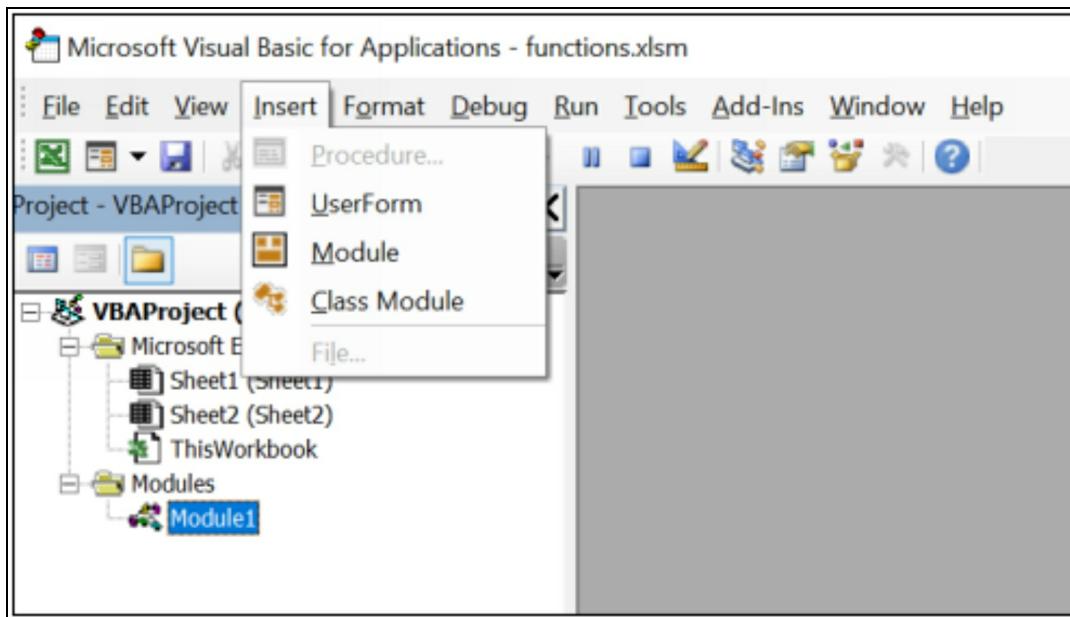
```

Private Sub CommandButton1_Click()
Dim a As Variant, b As Variant
a = InputBox("Enter Width of Rectangle")
b = InputBox("Enter Height of Rectangle")
MsgBox "The area of the rectangle is" & Area_Rect(a, b)
End Sub

Function Area_Rect(x As Variant, y As Variant) As Variant
Area_Rect = x * y
End Function

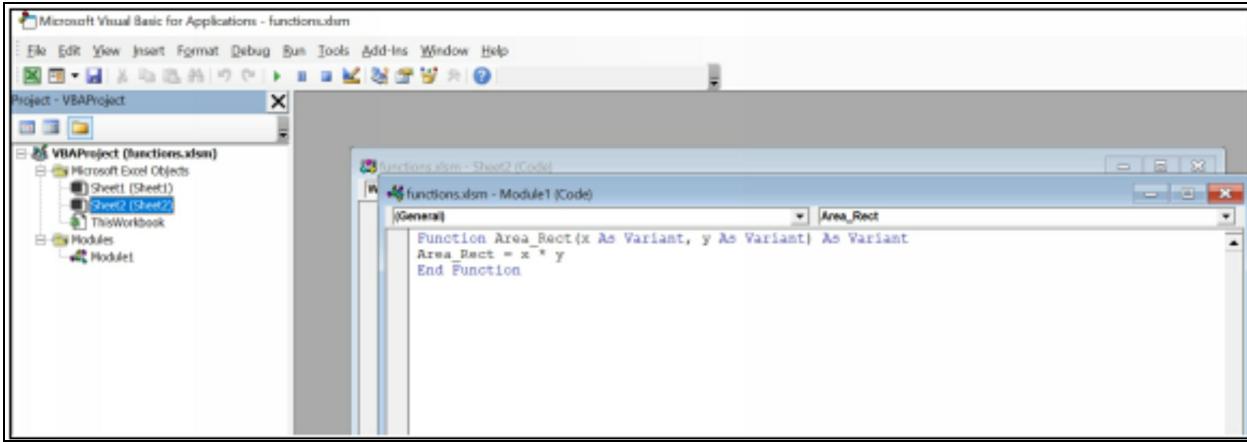
```

We can also create a user-defined function by inserting a module in the Visual Basic Editor and entering the function code there. After creating the function, we can then return to the spreadsheet and use this function as any other built-in functions. To insert the module, click on Tool in the menu bar, select Macro and then click on Visual Basic Editor. In the Visual Basic Editor window, insert a module by clicking Insert on the menu bar, and then click on Module.



**Figure 12.2**

In the module environment, enter the function code for the function `Area_Rect`, as shown in the Figure 12.3.



**Figure 12.3**

Now, you can return to the Excel spreadsheet and enter the function in any cell. In this Example , the function is entered in cell C7 and the values of width and height are entered in cell c5 and cell c6 respectively. Now type = in cells C7 and in the formula, type Ar and the IntelliSense will suggest Area\_Rect, as shown in Figure 12.4. When you click on Area\_Rect, the formula will appear on the Formula bar, as shown in Figure 12.5

A screenshot of an Excel spreadsheet. The formula bar shows "=Ar". The spreadsheet has columns A, B, C, E, and F. Row 1 contains "SUM" in cell A1. Row 2 is blank. Row 3 is blank. Row 4 is blank. Row 5 contains "Height" in cell C5 and "5" in cell E5. Row 6 contains "Width" in cell C6 and "4" in cell E6. Row 7 contains "Area of Rectangle" in cell C7 and the formula "=Ar" in cell E7, which is highlighted with a green border. A dropdown menu is open over cell E7, showing three options: "ARABIC", "Area\_Rect" (which is highlighted with a blue border), and "AREAS". Rows 8 and 9 are blank.

**Figure 12.4**

	A	B	C	D	E
1					
2					
3					
4					
5		Height		5	
6		Width		4	
7		Area of Rectangle	=Area_Rect(		
8					
9					
10					
11					
12					

**Figure 12.5**

Now type in the reference cells and press, you will obtain the area of the triangle, as shown in Figure 12.6

	A	B	C	D	E
1					
2					
3					
4					
5		Height		5	
6		Width		4	
7		Area of Rectangle	=Area_Rect(D5,D6)	20	
8					
9					
10					

**Figure 12.6**

The formula can be copied and updated to other cells by using the autofill method, i.e. by dragging the placeholder on the bottom right corner of the cell, as shown in Figure 12.7 below.

The screenshot shows an Excel spreadsheet with a green border around the range A1:B10. Cell C1 contains the formula =Area\_Rect(A1,B1). The data in columns A and B is as follows:

	A	B	C	D	E
1	3	20	60		
2	4	18	72		
3	5	16	80		
4	6	14	84		
5	7	12	84		
6	8	10	80		
7	9	8	72		
8	10	6	60		
9	11	4	44		
10	12	2	24		
11					
12					
13					

**Figure 12.7**

The user-defined function not only calculates numerical values, it can also return a string, as shown in Example 12.3.

### Example 12.3

This Excel VBA program computes the grades of an examination based on the marks obtained. It used the `Select Case...End Select` statements to compute the grade. Insert a module and type the function, as follows:

```
Function grade(mark As Single) As String
Select Case mark
Case 0 To 20
grade = "F"
Case 20 To 29
grade = "E"
Case 30 To 39
grade = "D"
Case 40 To 59
grade = "C"
Case 60 To 79
```

```

grade = "B"
Case 80 To 100
grade = "A"
Case Else
grade = "Error!"
End Select
End Function

```

In the example, we want to compute the grade for four subjects, History, Science, English and Math. Enter the formula for cells D4 and drag it down column D to generates the Grade for History. For other columns, just copy and then use Paste special to paste the formulas to all relevant cells. The results are shown in Figure 12.8

	A	B	C	D	E	F	G	H	I	J	K
1	ABC School Examination Results										
2		History	Grade	Science	Grade	English	Grad	Math	Grade		
3	Adam	40	C	70	B	78	B	90	A		
4	Chan	56	C	64	B	43	C	78	B		
5	Daniel	78	B	44	C	80	A	87	A		
6	Eden	50	C	34	D	54	C	60	B		
7	Fabian	67	B	57	C	72	B	78	B		
8	Ganesh	89	A	90	A	95	A	99	A		
9	Han	95	A	89	A	96	A	95	A		
10	Irina	28	E	34	D	35	D	24	E		
11	Jenkin	36	D	41	C	45	C	38	D		
12	Kanna	60	B	65	B	70	B	72	B		
13	Lam	100	A	99	A	98	A	100	A		
14	Menon	19	F	23	E	30	D	23	E		
15	Nathan	55	C	56	C	60	B	45	C		
16											
17											

Figure 12.8

#### Example 12.4

In this example, we create a function that calculates commissions entitlement based on the sales volume and commission rate in table 12.1. We can use the If...Then...Elseif statements to compute the commissions.

**Table 12.1 Commission Table**

Sales Volume (V)	Commissions
<500	3%
500≤V<1000	6%
1000≤V<2000	9%
2000≤V<5000	12%
5000≤V<10000	15%
10000≤V<15000	18%
V>15000	21%

## The Code

```
Function Comm(Sales_V As Variant) As Variant
Dim rate As Variant
If Sales_V < 500 Then
    rate = 0.03
ElseIf Sales_V >= 500 And Sales_V < 1000 Then
    rate = 0.06
ElseIf Sales_V >= 1000 And Sales_V < 2000 Then
    rate = 0.09
ElseIf Sales_V >= 2000 And Sales_V < 5000 Then
    rate = 0.12
ElseIf Sales_V >= 5000 And Sales_V < 10000 Then
    rate = 0.15
ElseIf Sales_V >= 10000 And Sales_V < 15000 Then
    rate = 0.18
ElseIf Sales_V >= 15000 Then
    rate = 0.21
End If
Comm = Format(Sales_V * rate, "$#,##0.00") 'Format the output with
dollar sign
End Function
```

After creating the Comm Function, we can then enter the sales volume in one

column and enter the formula based on the function Comm in another column. The commissions will be automatically computed and updated accordingly. The output screen is as shown in Figure 12.9

	A	B	C	D	E
1					
2				Commissions Payout	
3					
4		Sale Volume	Commissions		
5		1000	\$90.00		
6		3000	\$360.00		
7		10000	\$1,800.00		
8		800	\$48.00		
9		15000	\$3,150.00		
10		20000	\$4,200.00		
11		100	\$3.00		
12		50	\$1.50		
13		1700	\$153.00		
14		30000	\$6,300.00		
15					

Figure 12.9

## 12.4 Passing variables by reference and by Value in a Function

Variables in a function can be passed by reference or by value, using the keywords `ByRef` and `ByVal` respectively. The main difference between `ByRef` and `ByVal` is that `ByRef` will change the value of the variable while `ByVal` will retain the original value of the variable. By default, the function uses `ByRef` to pass variables.

### Example 12.5

```
Private Sub CommandButton1_Click()
Dim N As Variant
N = 27
Range("A1") = CRoot(N)
Range("A2") = N
Range("A3") = CRoot2(N)
Range("A4") = N
End Sub
```

```
Function CRoot(ByRef r As Variant)
r = r ^ (1 / 3)
CRoot = r
End Function
Function CRoot2( ByVal w As Variant)
w = w ^ (1 / 3)
CRoot2 = w
End Function
```

In this example, we created two similar functions, CRoot and CRoot2, respectively. However, the first function uses the `ByRef` keyword and the second function use the  `ByVal` keyword. Notice that the value of N has changed to 3 by the function CRoot, as shown in cell B3. Now the function CRoot2 computes the cubic root of N based on the new value of N, i.e. 3, and shows the result in cell B4. However, it does not change the value of N, it remains as 3, as shown in cell B5 of Figure 12.10

	A	B	C	D	E
1	Orginal Value of N	27			
2	Cubic Root of N	3			
3	Value of N	3			
4	Cubic Root of N	1.44225			
5	Value of N	3			
6					
7					
8					
9					
10			CommandButton1		
11					

**Figure 12.10**

# Chapter 13 Sub Procedures

---

A sub procedure is a procedure that performs a specific task and returns values, but it does not return a value associated with its name. However, it can return a value through a variable name. Sub procedures are usually used to accept input from the user, display information, print information, manipulate properties or perform some other tasks. It is a program code by itself and it is not an event procedure because it is not associated with a runtime procedure or a control such as a command button. It is called by a main program whenever it is required to perform a specified task. Sub procedures help to make programs smaller and easier to manage and debug.

A Sub procedure begins with a Sub statement and ends with an End Sub statement, as follows:

```
Sub ProcedureName (arguments)
    Statements
End Sub
```

## Example 13.1

In this example, a sub procedure ResizeFont is created to resize the font in the range if it fulfills a value greater than 40. There are two parameters or arguments associated with the sub procedure, namely x for font size and rng for range. This sub procedure is called by the event procedure Sub CommandButton1\_Click () and passed the values 15 to x (for font size) and Range ("A1:A10") to rng to perform the task of resizing the font to 15 for values>40 in range A1 to A10. The Excel VBA code is as follows:

```
Private Sub CommandButton1_Click()
    ResizeFont 15, Range("A1:A10")
End Sub
```

```
Sub ResizeFont(x As Variant, rng As Range)
    Dim myRange As Range
    For Each myRange In rng
```

```
If myRange.Value > 40 Then  
    myRange.Font.Size = x  
End If  
Next  
End Sub
```

The output is as shown in Figure 13.1

A screenshot of a Microsoft Excel spreadsheet. The table has columns A through F and rows 1 through 11. The data is as follows:

	A	B	C	D	E	F
1	20					
2	30					
3	40					
4	30					
5	60					
6	70					
7	80					
8	90					
9	40					
10	60					
11						

A context menu is open over the cell containing the value 80. The menu item "Resize Fonts" is highlighted.

**Figure 13.1**

To make the program more flexible and interactive, we can modify the above program to accept input from the user. The values input by the user through the input boxes will be passed on to the procedure to execute the job, as shown in Example 13.2.

### Example 13.2

```
Private Sub CommandButton1_Click()  
    Dim rng As String  
    rng = InputBox("Input range")  
    x = InputBox("Input Font Size")  
    ResizeFont x, Range(rng)
```

```
End Sub
```

### Example 13.3

```
Sub ResizeFont(x As Variant, Rge As Range)
Dim cel As Range
For Each cel In Rge
If cel.Value > 40 Then
cel.Font.Size = x
End If
Next cel
End Sub
```

### Example 13.4

In this example, the main program calls the subroutine `findHidden` and execute it. The result is a message box that displays the hidden text.

```
Private Sub CommandButton1_Click()
ShowHidden
End Sub
Sub ShowHidden()
hidden_txt = "@#%43&*"
MsgBox hidden_txt
End Sub
```



**Figure 13.2**

### Example 13.5

In this example, the main program calls the subroutine salary and passes the parameters 10 and 300 to it. It will calculate the salary based on wage per hour and the number of hours and display on the message box.

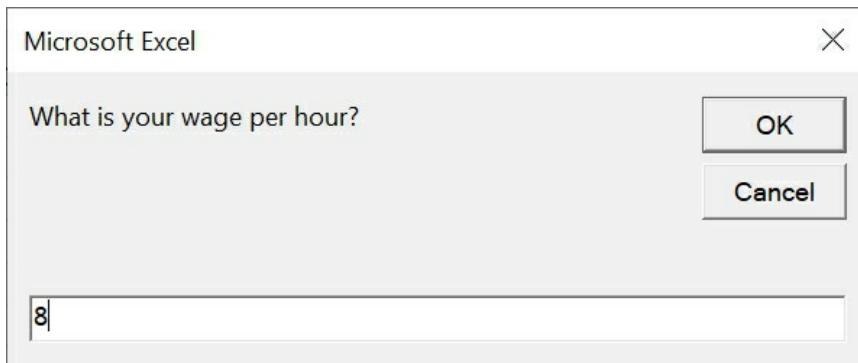
```
Private Sub Cmd_Calculate_Click()
    salary 10, 300
End Sub
```

```
Sub salary(wage As Single, hours As Single)
    MsgBox wage * hours
End Sub
```

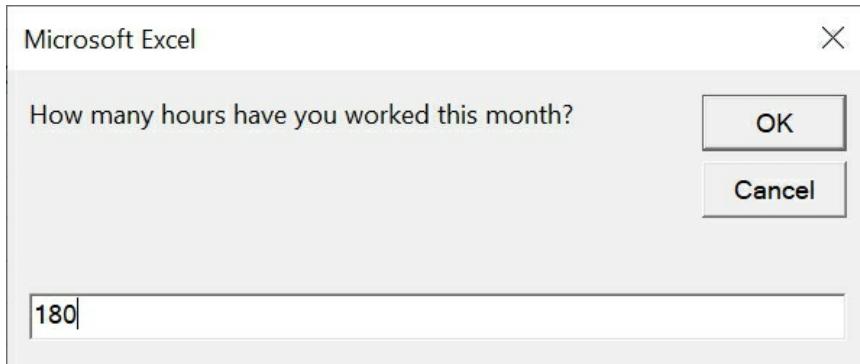
We can add make the program above more interactive by accept inputs from the user via input boxes, as follows:

```
Sub salary(wage As Single, hours As Single)
    wage = InputBox("What is your wage per hour?")
    hours = InputBox("How many Hours have you worked this month")
    MsgBox "Your Salary is $ " & wage * hours
End Sub
```

Running the program two input boxes for the user to enter the wage and the hours worked, as shown in Figure 13.3 and Figure 13.4



**Figure 13.3**



**Figure 13.4**

After accepting the values above, the Excel VBA program calculates the salary, as shown in Figure 13.5



**Figure 13.5**

### Example 13.6

This Excel VBA 365 program determines the buying decision based on shoe's size and its price. In this program, we create a sub procedure known as `buy_decision` that has two arguments, `size` and `price`. We declared a Boolean variable `buy` to assist in decision-making. It takes a value of true or false. If the size and price fulfilled the requirements, `buy` is assigned a value of true, else it is assigned a value of false.

```
Sub buy_decision(size As Integer, price As Single)  
Dim buy As Boolean
```

```

If size >= 7 And price <= 200 Then
    buy = True
    MsgBox ("Buy")
Else: buy = False
    MsgBox ("Don't Buy")
End If
End Sub

```

```

Private Sub CmdDecide_Click()
    Dim shoe_size As Integer, shoe_price As Single
    shoe_size = val(TxtSize.Text)
    shoe_price = val(TxtPrice.Text)
    buy_decision shoe_size, shoe_price
End Sub

```

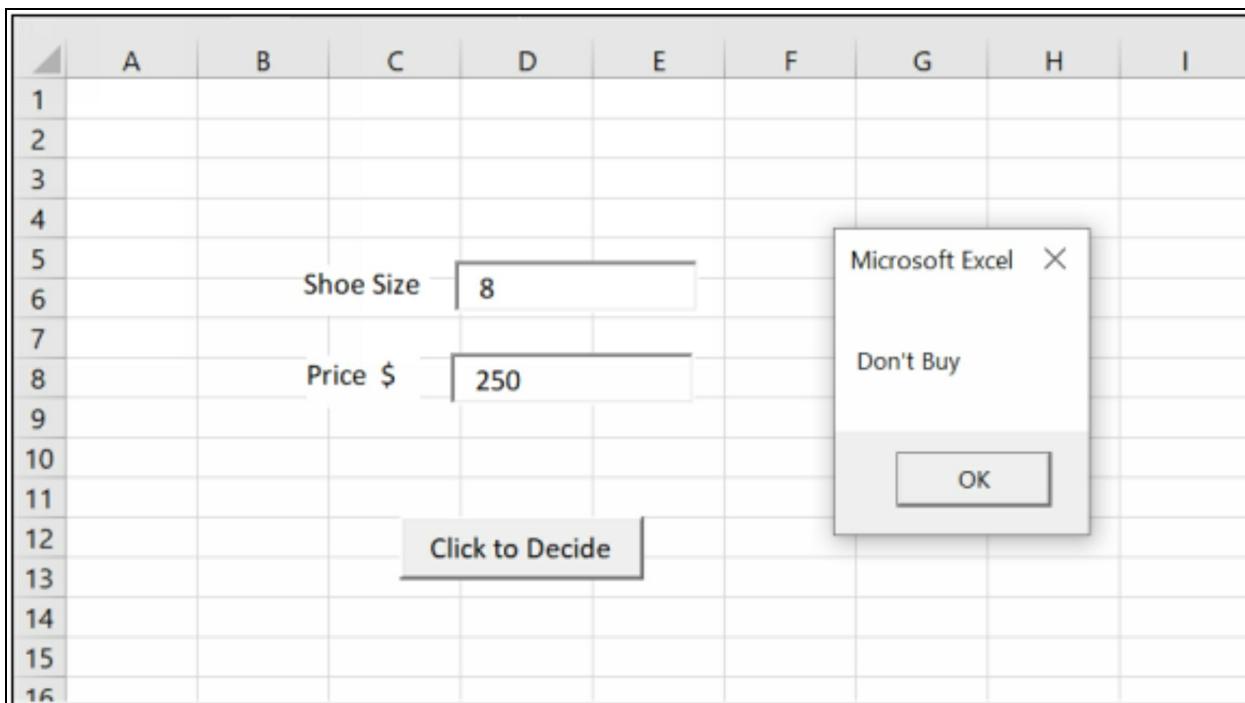


Figure 13.6

# Chapter 14 String Handling Functions

String handling functions are used to manipulate text in Excel VBA 365. Some of the string handling functions are listed and explained below:

## 14.1 InStr

The InStr function searches for a phrase that is embedded within the original phrase and returns the starting position of the embedded phrase. The syntax is

Instr (n, original phrase, embedded phrase)

n is the position where the Instr function will begin to look for the embedded phrase. For example,

Instr(1, "Visual Basic", "Basic")=8

## 14.2. Left

The Left function extracts a substring from a phrase, starting from the left. The syntax is .

Left(Phrase, n)

n indicates the number of characters that you wish to extract starting from the left-most character.

```
Dim phrase, myphrase As String  
phrase="Visual Basic for Applications"  
myphrase=Left(phrase, 6)  
Msgbox(myphrase)
```

Running the code produces a message box displaying the extracted phrase “Visual”

## 14.3. Right

The Right function extracts a substring from a phrase, starting from the Right. The syntax is

Right(phrase, n)

n indicates the number of characters that you wish to extract starting from the right-most character.

Right (phrase, 5) means 5 characters are extracted from the phrase, starting from the rightmost position. For example

```
Dim phrase, myphrase As String  
phrase="Visual Basic for Applications"  
myphrase=Right(phrase, 8)  
MsgBox(myphrase)
```

Running the code produces a message box displaying the extracted phrase “ications”

## 14.4. Mid

The Mid function extracts a substring from a phrase, starting from the position specified by the second parameter in the bracket. The syntax is

Mid(phrase, position, n)

position is the starting position of the phrase from which the extraction process will start, and n is the number of characters to be extracted. For example

```
Dim phrase, myphrase As String  
phrase = "Visual Basic for Applications"  
myphrase = Mid(phrase, 8, 5)  
MsgBox(myphrase)
```

Running the code produces a message box displaying the extracted phrase "Basic"

## 14.5. Len

Len is a function that returns the length of a phrase, including spacing. The syntax is

Len(Phrase)

For example, if phrase="Excel VBA 365" then

Len(Phrase)=13

### Example 14.1

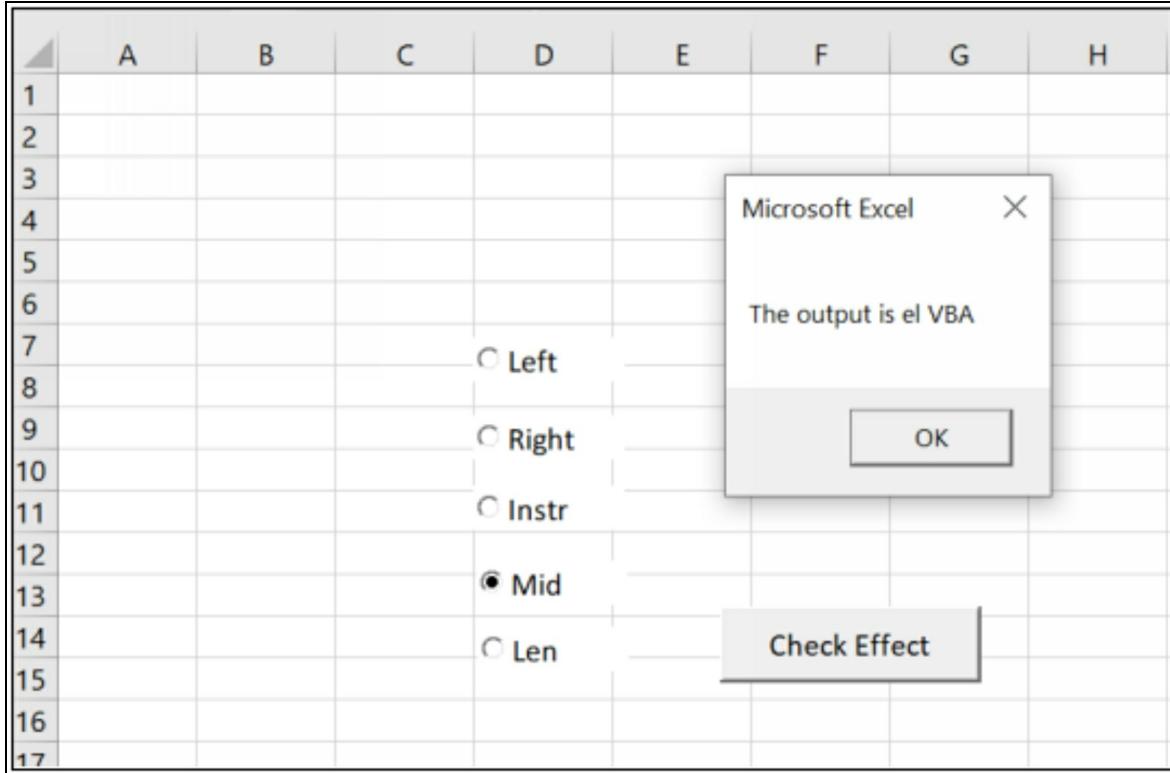
In this example, we insert five option buttons, lable them as Left, Right, Instr, Mid and Len.

When the user selects an option and click check effect, he will see the effect correspond to the string functions.

```
Private Sub Cmd_Check_Click()
Dim choice As Boolean
Dim phrase As String
Dim output As String
phrase = "Excel VBA 365"
If OptLeft.Value = True Then
    output = Left(phrase, 3)
ElseIf OptRight = True Then
    output = Right(phrase, 4)
ElseIf OptInstr = True Then
    output = InStr(1, phrase, "VBA")
ElseIf OptMid = True Then
    output = Mid(phrase, 4, 7)
Else
    output = Len(phrase)
End If
```

```
MsgBox ("The output is " & output)  
End Sub
```

The output is shown in Figure 14.1



**Figure 14.1**

# Chapter 15 Date and Time Functions

---

We can use various built-in date and time handling functions to manipulate date and time.

## 15.1 Using the Now ( ) Function

The Now () function returns the current date and time according to your computer's regional settings. We can also use the Format function in addition to the function Now to customize the display of date and time using the syntax

Format (Now, "style argument")

The usage of Now and Format functions are explained in the Table 15.1

**Table 15.1 Date and Time Formats**

Format	Output
Format(Now, "s")	Current Time in seconds
Format(Now, "n")	Current Time in minutes
Format(Now, "h")	Current Time in hours
Format(Now, "m")	Current Month in numeric form
Format(Now, "mmm")	Current Month in short form
Format(Now, "mmmm")	Current Month in full name
Format(Now, "y")	Number of days to date in current year
Format(Now, "yyyy")	Current Year

### Example 15.1

```
Private Sub Cmd_ShowTimeDate_Click()
```

```

Cells(1, 2).Value = Now()
Cells(2, 2).Value = Format(Now, "s")
Cells(3, 2).Value = Format(Now, "n")
Cells(4, 2).Value = Format(Now, "h")
Cells(5, 2).Value = Format(Now, "m")
Cells(6, 2).Value = Format(Now, "mmm")
Cells(7, 2).Value = Format(Now, "mmmm")
Cells(8, 2).Value = Format(Now, "y")
Cells(9, 2).Value = Format(Now, "yyyy")
End Sub

```

	A	B	C
1	Current Date and Time	8/5/2020 8:33	
2	Current Time in seconds	26	
3	Current Time in Minutes	33	
4	Current Time in hours	8	
5	Current Month in numeric form	5	
6	Current Month in short form	May	
7	Current Month in Full form	May	
8	Number of days to date in current year	129	
9	Current Year	2020	
10			
11		Show Current Date & Time	
12			
13			
14			

Figure 15.1

## 15.2 Date and Time Functions

The usage of these functions is illustrated in the Table 15.2

Table 15.2 Date and Time Functions

FUNCTION	OUTPUT
Date	Current date
Day(Date)	Day part of the current date

Weekday(Date)	Weekday of the current week in numeric form.
WeekdayName(Weekday(Date))	Weekday name of the current date
Month(Date)	Month of the current year in numeric form
MonthName(Month(Date))	Full name of the current month
Year(Date)	Current year in long form

## Example 15.2

```

Private Sub Cmd_ShowDateTime_Click()
Cells(1, 2) = Date
Cells(2, 2) = Day(Date)
Cells(3, 2) = Weekday(Date)
Cells(4, 2) = WeekdayName(Weekday(Date))
Cells(5, 2) = Month(Date)
Cells(6, 2) = MonthName(Month(Date))
Cells(7, 2) = Year(Date)
End Sub

```

	A	B	C
1	Current Date	8/5/2020	
2	Current Day	8	
3	Weekday of the current week in numeric form.	6	
4	Weekday name of the current date	Friday	
5	Month of the current year in numeric form	5	
6	Full name of the current month	May	
7	Current year in long form	2020	
8			
9			
10			
11			
12	Show Current Date and Time		
13			
14			

Figure 15.2

## 15.3 DatePart Function

The DatePart function is used together with the Now function to retrieve part of date or time specified by the arguments. The DatePart function is written as

DatePart (Part of date to be returned, Now)

Several DatePart expressions and the corresponding outputs are shown in Table 15.3

**Table 15.3 DatePart Expressions**

<b>DATEPART EXPRESSION</b>	<b>PART OF DATE /TIME RETURNED</b>
DatePart("s",Now)	Current second
DatePart("n",Now)	Current minute
DatePart("h",Now)	Current hour
DatePart("w",Now)	Current weekday
DatePart("m",Now)	Current month
DatePart("y",Now)	Current day of the year
DatePart("yyyy",Now)	Current year

### Example 15.3

```
Private Sub CommandButton1_Click ()  
Cells (2, 2) = DatePart ("yyyy", Now)  
Cells (3, 2) = DatePart ("m", Now)  
Cells (4, 2) = DatePart ("d", Now)  
Cells (5, 2) = DatePart ("w", Now)  
Cells (6, 2) = DatePart ("h", Now)  
Cells (7, 2) = DatePart ("n", Now)  
Cells (8, 2) = DatePart ("s", Now)  
End Sub
```

	A	B	C	D
1				
2	Current Year	2020		
3	Current Month	5		
4	Current Day	8		
5	Current Weekday	6		
6	Current Hour	9		
7	Current Minute	9		
8	Current Second	42		
9				
10				
11				
12				
13	Show Current Date and Time			
14				
15				

**Figure 15.3**

## 15.4 Adding and Subtracting Dates

Dates can be added using the `DateAdd` function. The syntax of the `DateAdd` function is

`DateAdd(interval, value to be added, date)`

interval=part of date to be added. For example,

`DateAdd ("yyyy", 3, Now)`

3 years will be added to the current year.

Similarly, Dates can be subtracted using the `DateDiff` function. The syntax of the `DateDiff` function is

`DateDiff(interval, first date, second date)`

interval=part of date to be subtracted. For example,

```
DateDiff ("yyyy", Now, "6/6/2012")
```

3 years will be subtracted from the current year. Both the aforementioned functions use the argument “s” for second, “n” for minute, “h” for hour, “d” for day, “w” for week, “m” for month and “yyyy” for year.

#### Example 15.4

```
Private Sub CommandButton1_Click ()
```

```
Cells (1, 1) = Date  
Cells (2, 1) = DateAdd ("s", 300, Now)  
Cells (3, 1) = DateAdd ("n", 30, Now)  
Cells (4, 1) = DateAdd ("h", 3, Now)  
Cells (5, 1) = DateAdd ("d", 2, Now)  
Cells (6, 1) = DateAdd ("m", 3, Now)  
Cells (7, 1) = DateAdd ("yyyy", 2, Now)  
Cells (8, 1) = DateDiff ("yyyy", Now, "8/6/2012")  
Cells (9, 1) = DateDiff ("d", Now, "13/6/2009")  
Cells (10, 1) = DateDiff ("m", Now, "8/10/2011")  
Cells (11, 1) = DateDiff ("d", Now, "8/10/2009")  
Cells (12, 1) = DateDiff ("n", Now, "8/10/2009")  
Cells(13, 1) = DateDiff("s", Now, "8/10/2009")
```

```
End Sub
```

	A	B	C	D
1	8/5/2020			
2	8/5/2020 9:24			
3	8/5/2020 9:49			
4	8/5/2020 12:19			
5	10/5/2020 9:19			
6	8/8/2020 9:19			
7	8/5/2022 9:19			
8	-8			
9	-3982			
10	-103			
11	-3865			
12	-5566159			
13	-333969554			
14				
15				
16				
17	CommandButton1			
18				

Figure 15.4

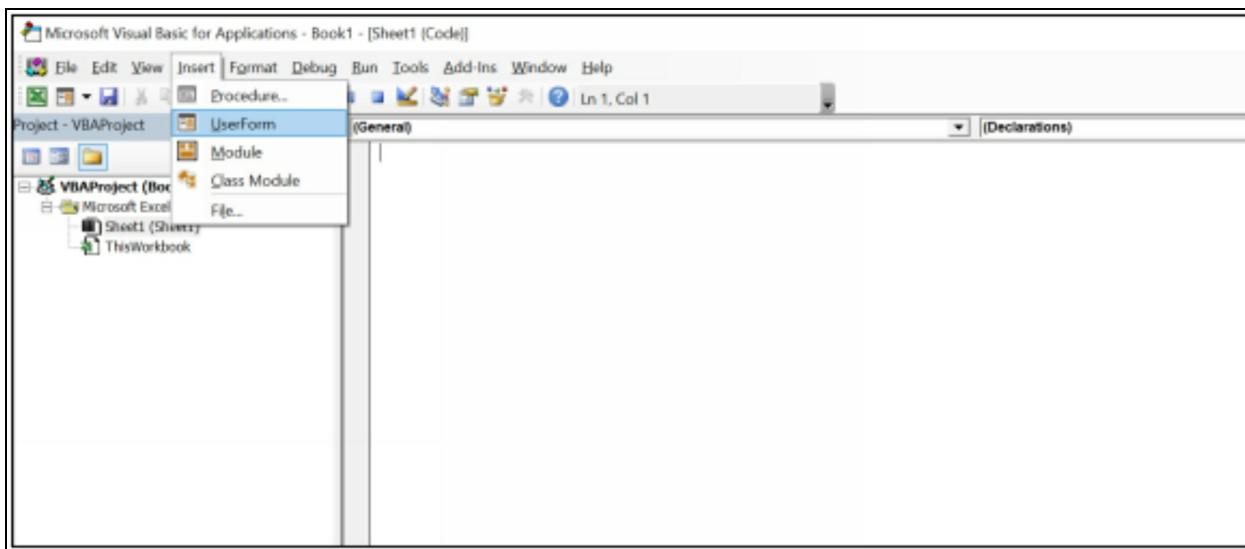
---

# Chapter 16 UseForm

---

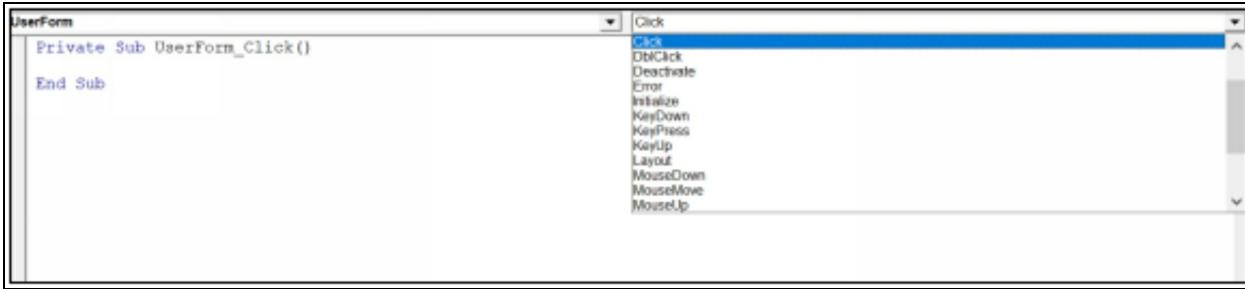
According to Microsoft, a UserForm object is a window or dialog box that makes up part of Microsoft Excel application's user interface. In layman terms, it is a tool to build custom Excel VBA applications. In fact, it is like the mini Visual Basic integrated development environment that lets you build any applications you wish.

To use the UserForm , enter the Excel VBA 365 editor by clicking View Code or Visual Basic in the Developer environment. In the Excel VBA 365 editor, select Insert then choose UserForm from the drop-down menu, as shown in Figure 16.1



**Figure 9.1**

The UserForm has many events associated with it, as shown in Figure 16.2



**Figure 16.2 Events**

Some of the events are Click, Dblclick, Initialize, KeyDown, KeyUp, MouseDown, MouseMove, MouseUp and more. You can write Excel VBA 365 code for the events.

## 16.1 The Keyboard Events

The keyboard events are KeyPress, KeyDown and KeyUp. These events capture the keys triggered by the user.

### Example 16.1

This is the Exel VBA 365 code that can check whether the user presses the Enter key or other keys. It involved the use of the ASCII code.

```
Private Sub UserForm_KeyPress(ByVal KeyAscii As
MSForms.ReturnInteger)
If KeyAscii = 13 Then '13 is the ASCII value for the Enter key
    MsgBox "You have pressed the Enter key"
Else
    MsgBox "You have pressed other key"
End If
End Sub
```

### The Output



**Figure 16.3 The user pressed Enter key**



**Figure 16.4 The User Pressed Other keys**

### ASCII Explanation

The keyboard event occurs when the user presses any key that corresponds to a certain alphanumeric value or an action such as Enter, spacing, backspace and more. Each of those value or action is represented by a set of code known as the ASCII. ASCII stands for American Standard Code for Information Interchange. ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. In order to write code for the Keyboard events, we need to know the ASCII and the corresponding values. Chr is the function that returns the string that corresponds to an ASCII code. For example, chr(65)=A. Table 16.1 show all the ASCII code and its values.

**Table 16.1 ASCII Code**

ASCII	Chr	ASCII	Chr	ASCII	Chr
8	Backspace	61	=	98	b

13	Enter key	62	>	99	c
32	Space	63	?	100	d
33	!	64	@	101	e
34	"	65	A	102	f
35	#	66	B	103	g
36	\$	67	C	104	h
37	%	68	D	105	i
38	&	69	E	106	j
39	'	70	F	107	k
40	(	71	G	108	l
41	)	72	H	109	m
42	*	73	I	110	n
43	+	74	J	111	o
44	,	75	K	112	p
45	-	76	L	113	q
46	.	77	M	114	r
47	/	78	N	115	s
48	0	79	O	116	t
49	1	80	P	117	u
50	2	81	Q	118	v
51	3	82	R	119	w
52	4	83	S	120	x
53	5	84	T	121	y
54	6	85	U	122	z
55	7	86	V	123	{
56	8	87	W	124	
57	9	88	X	125	}
58	:	89	Y	126	~
59	;	90	Z	127	DEL
60	<	97	a		

## Example 16.2

This example can identify which key is being pressed. The Chr function returns the character (or action) that corresponds to the ASCII code, the syntax is Chr(ASCII code). For example, Chr(65)=A . When a key is pressed, automatically it returns a value corresponding to the ASCII code defined by its argument KeyAscii

```
Private Sub UserForm_KeyPress(ByVal KeyAscii As  
MSForms.ReturnInteger)  
MsgBox ("You pressed the ") & Chr(KeyAscii) & " key"  
End Sub
```

If you pressed the # key, the following message box will appear, as shown in Figure 16.5.



**Figure 16.5**

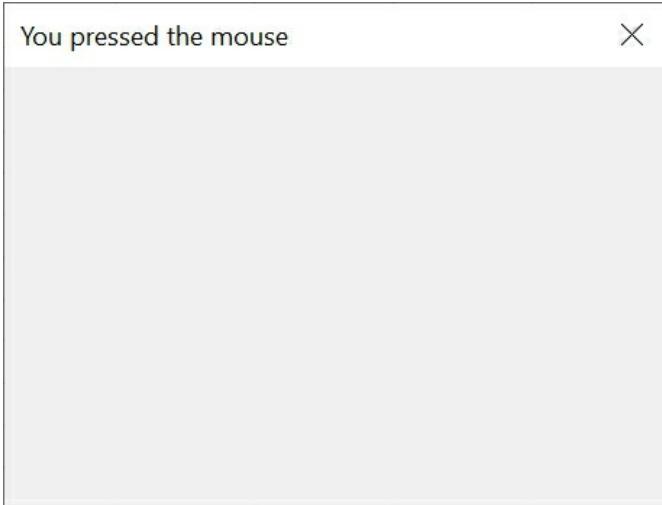
## 16.2 The Mouse Events

### Example 16.3

You can also try out the following Excel VBA code that changes the caption of the UserForm when the user presses the mouse button.

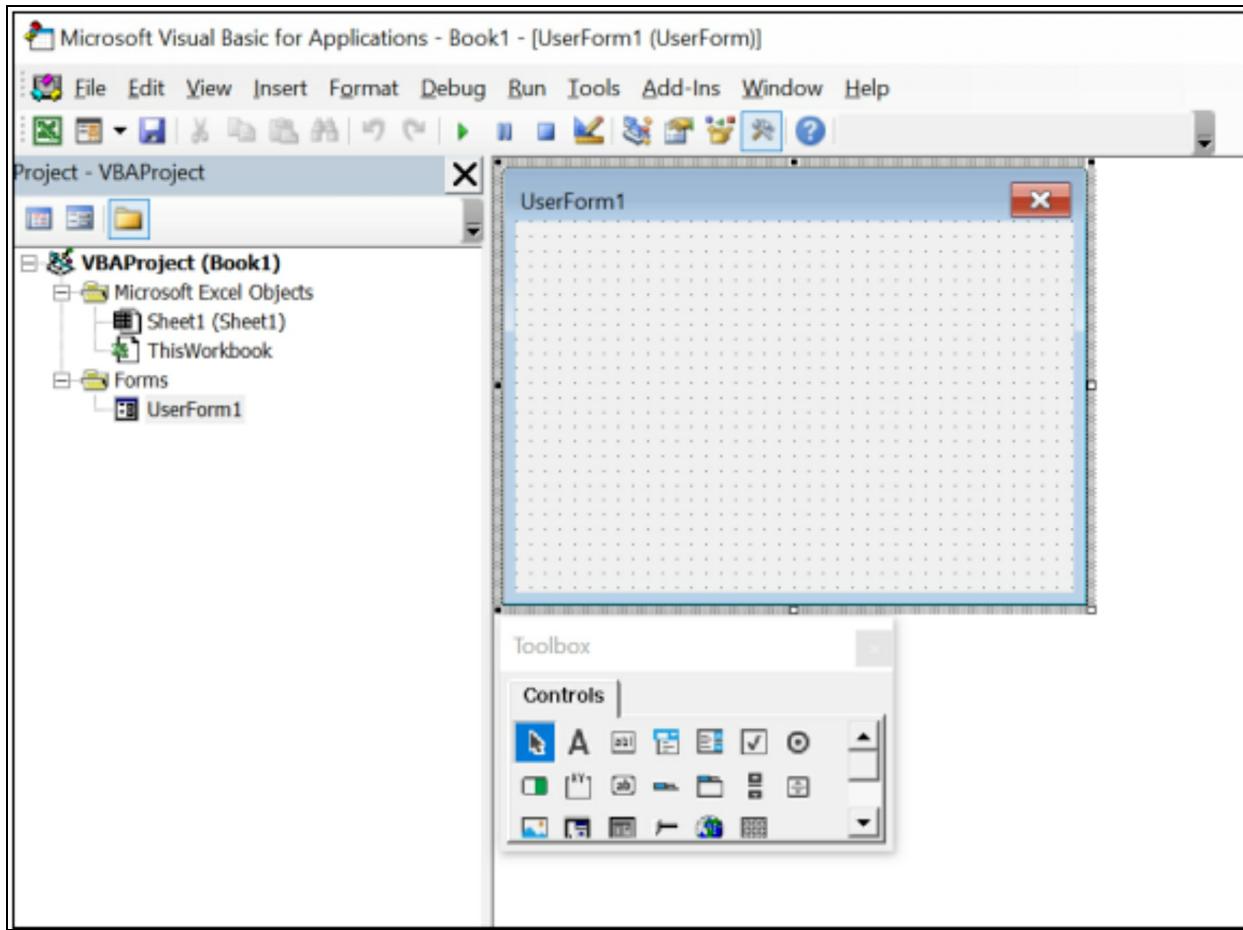
```
Private Sub UserForm_MouseDown(ByVal Button As Integer, ByVal Shift  
As Integer, ByVal X As Single, ByVal Y As Single)  
UserForm3.Caption = "You Pressed the mouse"  
End Sub
```

The output is as seen in Figure 16.6



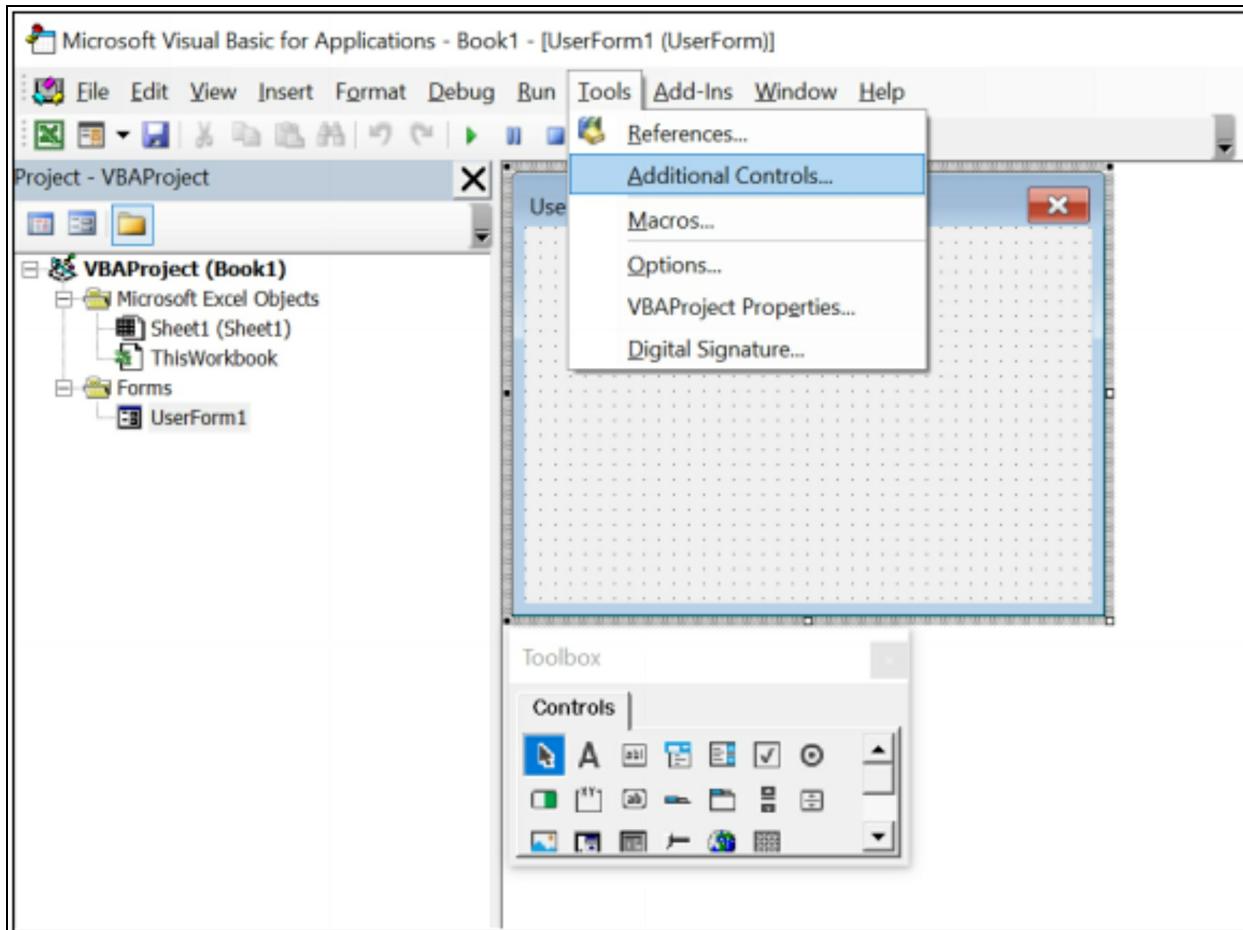
**Figure 16.6**

To develop more complex applications, you may place more ActiveX controls on the UserForm and write code for them. The UserForm comes with a Toolbox comprising Active-X controls such as TextBox, CheckBox, CommandButton, ListBox , Label, ComboBox, Image , OptionButton and more, as shown in Figure 16.7. Since you have learned how to write codes for the controls above, we shall proceed to discuss one particularly important control, the Common Dialog Control.



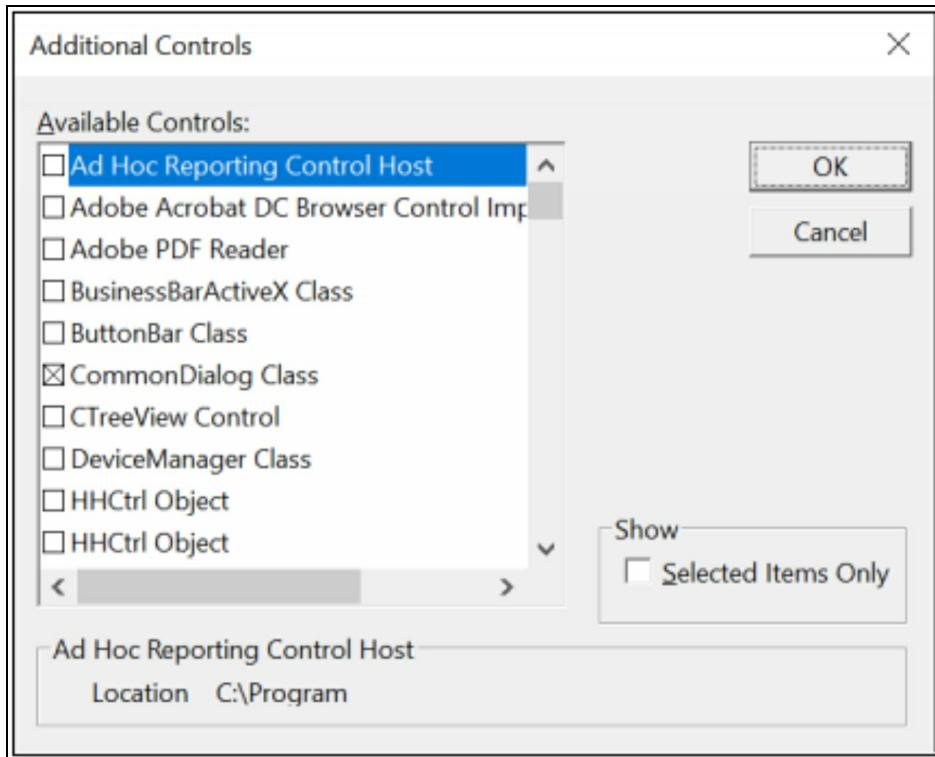
**Figure 16.7**

You can add more ActiveX controls by clicking Tools on the menu bar and select Additional Tools, as shown in Figure 16.8



**Figure 16.8**

Clicking on Additional Controls will produce an Additional Control dialog box, as shown in Figure 16.9 You can then select the controls you wish to add to the UserForm.



**Figure 16.9**

#### Example 16.4

You can display information from the spreadsheet in a List box on the UserForm. The following code display information from column 1 and column 2 from the spreadsheet with headings on the ListBox.

ColumnCount is the property of the ListBox to display the number of columns, ColumnHeads to display the heads, RowSource to define the range of cells. The output is as shown in Figure 16.11.

```
Private Sub CommandButton1_Click()
    ListBox1.ColumnCount = 2
    ListBox1.ColumnHeads = True
    ListBox1.RowSource = "Sheet1!A2: B5"
End Sub
```

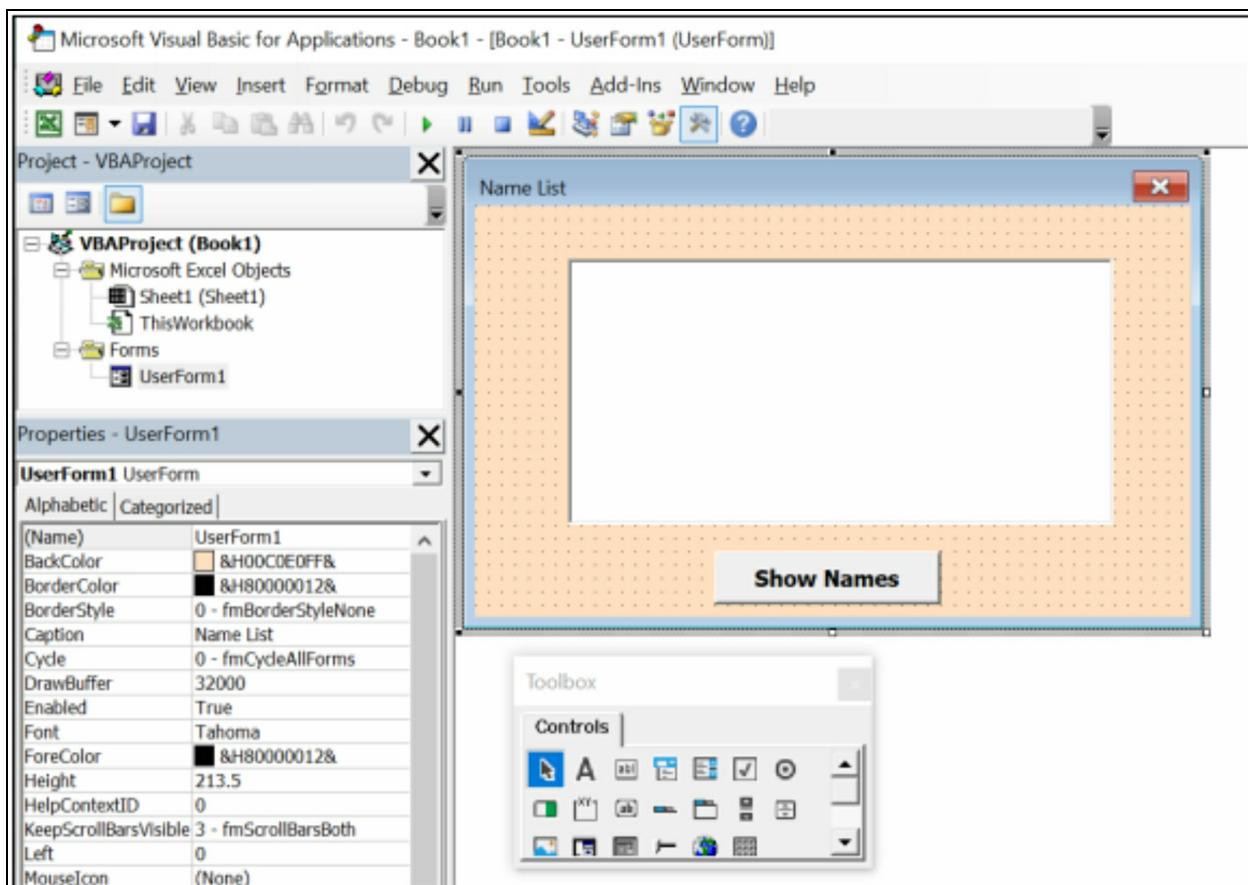
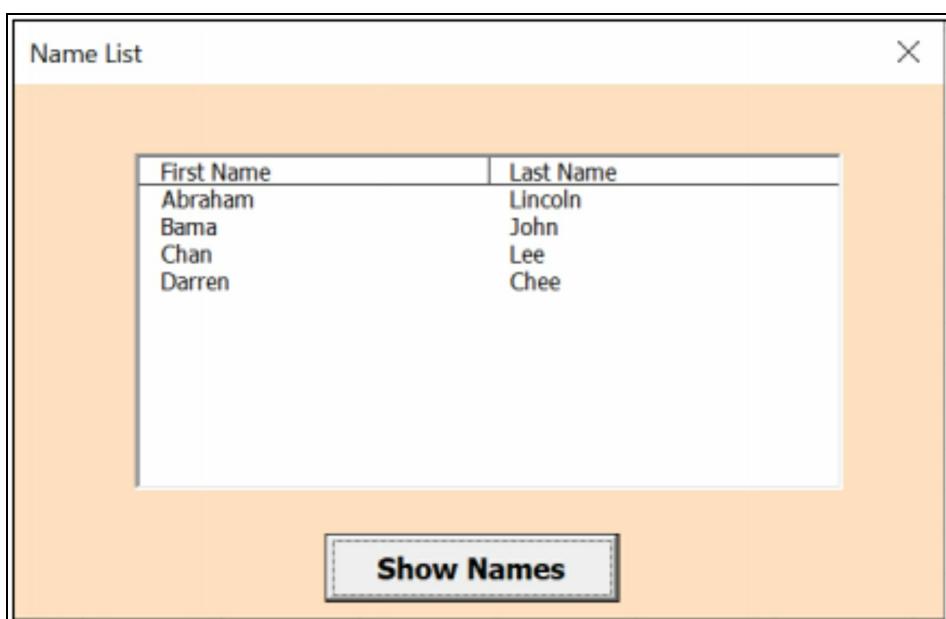


Figure 16.10 The Design UI



## Figure 16.11 The Runtime UI

### Example 16.5

You can create a simple calculator to perform a simple calculation involving the information on a spreadsheet. Insert a Userform, then place two text boxes, one label control, and a command button onto the form. Set the ControlSource property of the first text box to Sheet1!A1 and the ControlSource property of the second text box to sheet1!B1. Add three more labels, place the first label besides TextBox1 and change the caption to number 1 and place the second label besides TextBox2 and change the caption to number 2, and place the last label besides the label that you will display the sum of the two numbers in the text boxes and change to caption to Total. Lastly, change the caption of the command button to Calculate. Now click on the command button and enter the following code:

```
Private Sub CommandButton1_Click()  
Label1.Caption = Val(TextBox1.Text) + Val(TextBox2.Text)  
Cells(1, 3 = Label1.Caption  
End Sub
```

Press F5 to run the program and you will be presented the interface of a simple calculator. Click on the calculate button and you will obtain the sum of the two numbers from cells A1 and cells B1. The sum will be displayed in cells C3, as shown in Figure 16.12.

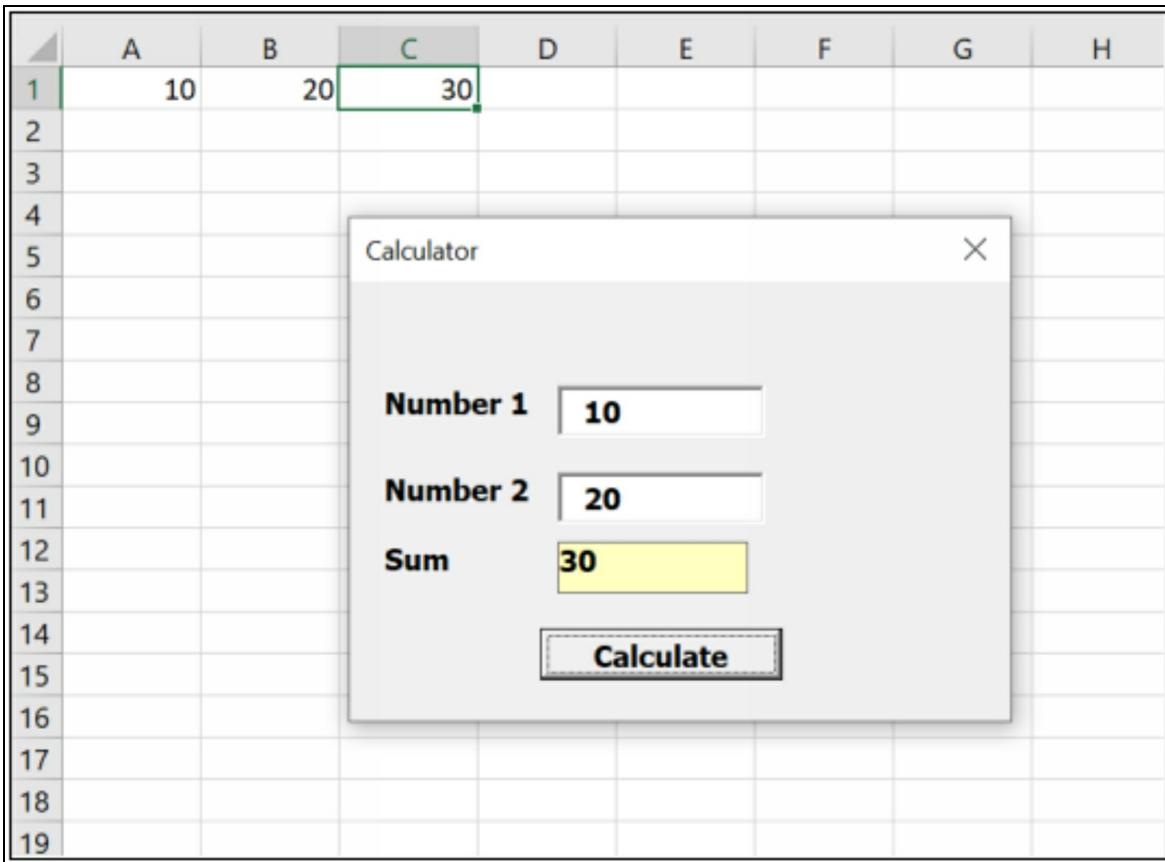
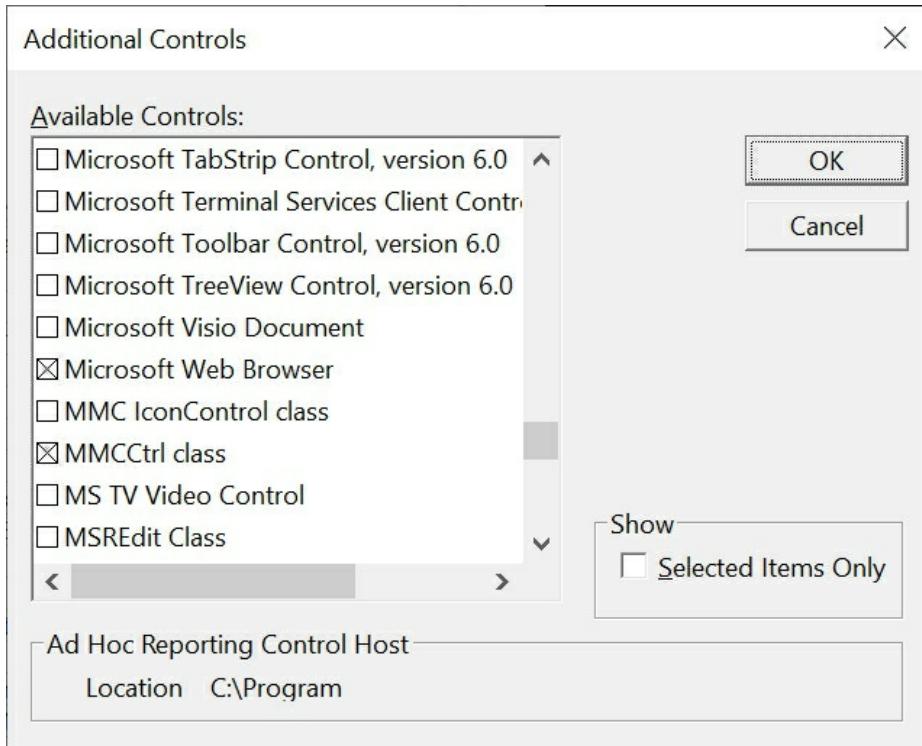


Figure 16.12

### Example 16.6 Web Browser

If you are bored with existing web browsers, you can create your very own web browser using Excel VBA 365. First, insert a UserForm and name it as Frm\_Webbrowser and change its caption to Excel VBA 365 Browser.

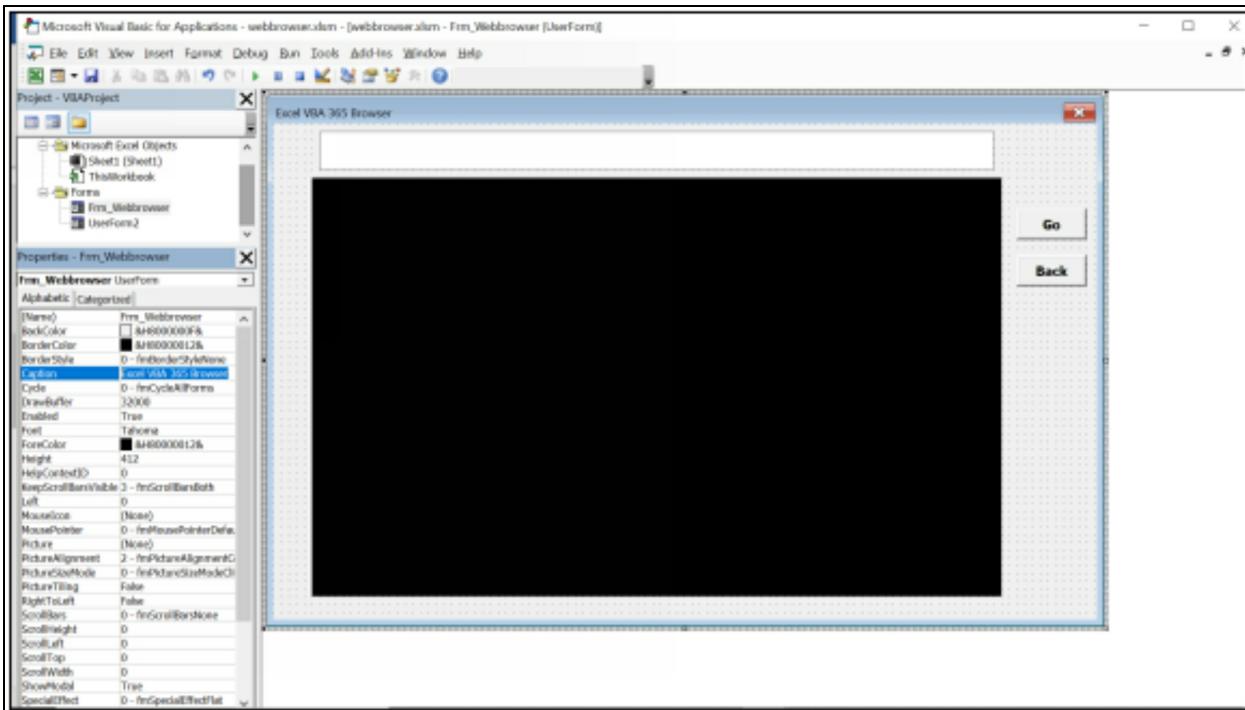
Next, insert the Web Browser control into your UserForm. The Web Browser control is not available in the default toolbox, so you need to add it from the Add Additional Controls menu and select Microsoft Web Browser, as shown in Figure 16.13.



**Figure 16.13**

Upon clicking the OK button, you will see the Web Browser control in the control toolbox.

To design the UI, insert the Web Browser into your UserForm and drag it to a suitable size. Additionally, insert a text box to be used as the URL panel, name it as `TxtURL`. Besides that, insert two command buttons, name one of them as `Cmd_Go` and the other one as `Cmd_Back`. These two buttons act as the Go and Back navigators. The design UI is as shown in Figure 16.14



**Figure 16.14**

## The Code

```
Private Sub UserForm_Initialize()
    MyWebBrowser.Navigate ("https://www.excelvbatutor.com")
End Sub
```

```
Private Sub Cmd_Back_Click()
    On Error Resume Next
    MyWebBrowser.GoBack
End Sub
```

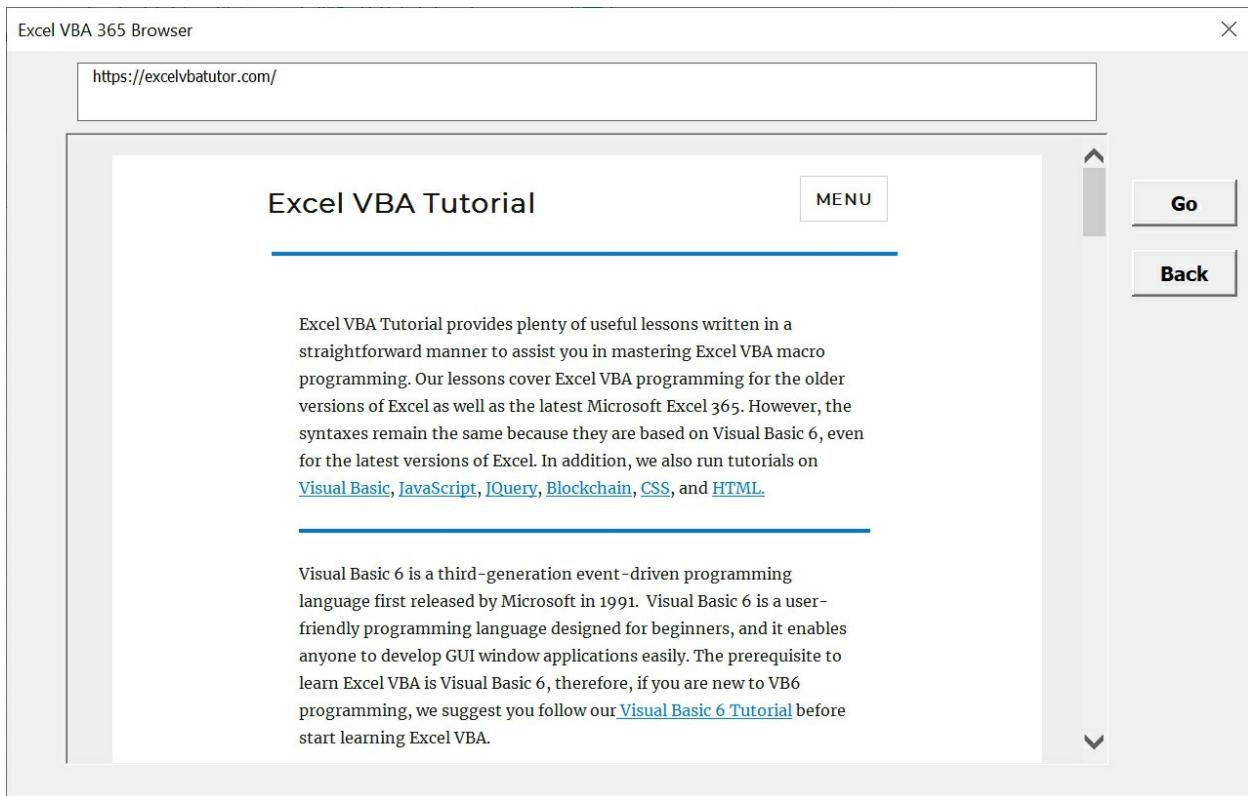
```
Private Sub Cmd_Go_Click()
    MyWebBrowser.Navigate (TxtURL.Text)
End Sub
```

'To show the URL in the text box when loading is completed

```
Private Sub MyWebBrowser_DocumentComplete(ByVal pDisp As Object,
    URL As Variant)
    TxtURL.Text = URL
End Sub
```

End Sub

The Output is shown in Figure 16.15.



**Figure 16.15**

# Chapter 17 Working with Files

---

In this chapter, you will learn how to work with files in Excel VBA 365. Working with files includes creating files, saving files and opening files.

In the older version of Excel, you can use Microsoft Common Dialog control to create dialog boxes for opening and saving files. However, Excel VBA 365 has done away with this control. Instead, it provides several built-in dialog boxes to handle opening and saving of files.

Excel VBA 365 file handling methods includes `GetOpenFilename` method, `GetSaveAsFilename` method, and the `Dialogs` object of the `Application` object, which contains all the built-in Excel dialog boxes.

## 17.1 Application.GetOpenFilename method

This method displays the standard **Open** dialog box and gets a file name from the user without actually opening any files. The syntax is

`Application.GetOpenFilename (FileFilter, FilterIndex, Title, ButtonText, MultiSelect)`

To open the file, you must use a `Workbook` method `Open`, the syntax is  
`Workbooks.Open(Filename)`

### Example 17.1

In this example, start Excel and insert a `UserForm` in the VBE. Insert a command button on the `UserForm` and change its caption to Open File. Click the command button and enter the following Excel VBA code:

```
Dim myfileName As Variant
myfileName = Application.GetOpenFilename("Excel Files (*.xlsx),
*.xlsx,Excel Macro Files (*.xlsm), *.xlsm", Title:="Open File")
If myfileName <> False Then
    Workbooks.Open (myfileName)
End If
```

When you press F5 and run the program, a dialog box will appear ad prompt the user to select a file from a folder, as shown in Figure 17.1

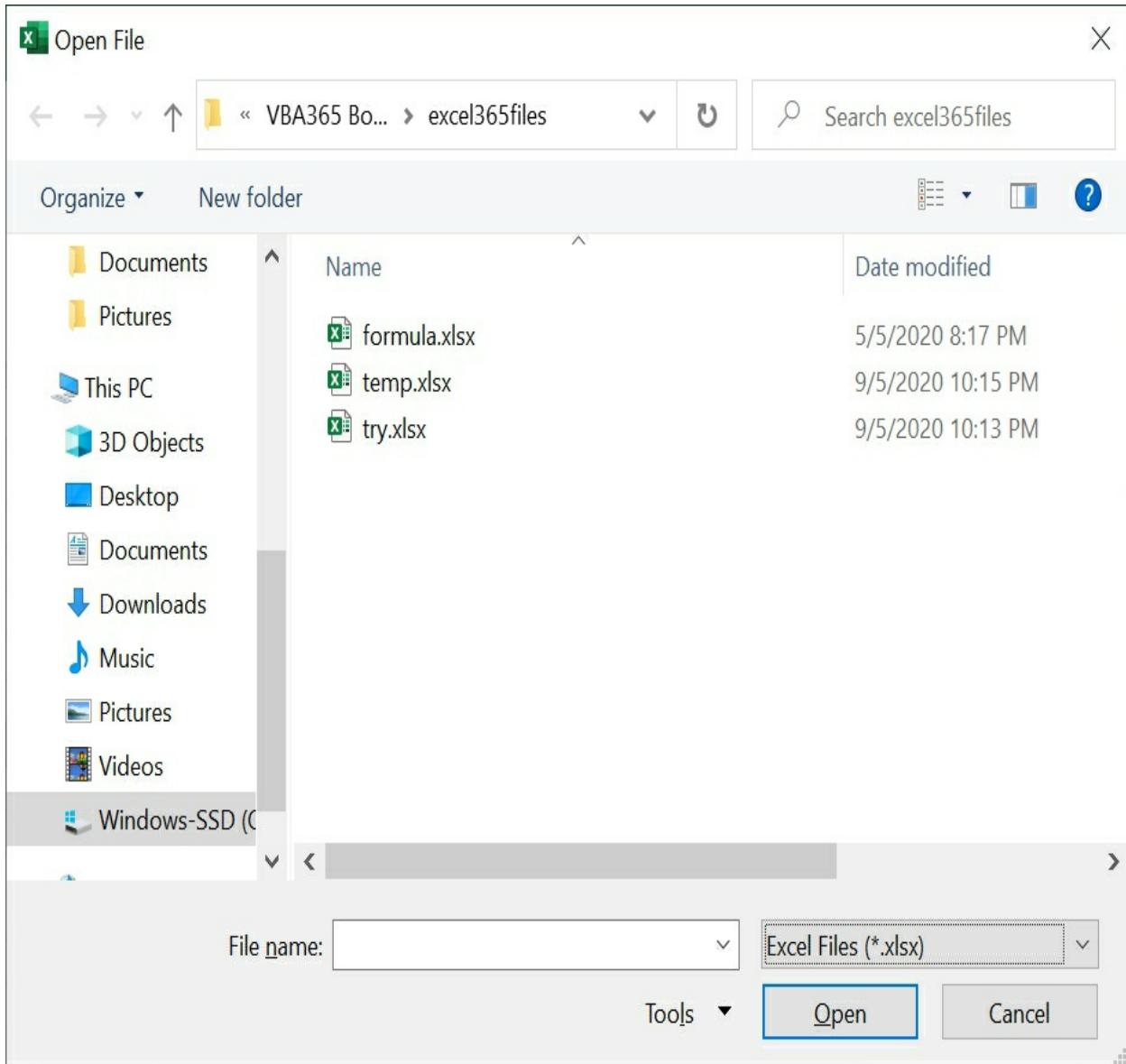


Figure 17.1

Selecting a file from the folder and clicking the Open button will open the Excel file.

## 17.2 Application.GetSaveAsFilename method

This method displays the standard **Save As** dialog box and gets a file name from the user without actually saving any files. The syntax is

`Application.GetSaveAsFilename (InitialFilename, FileFilter, FilterIndex,`

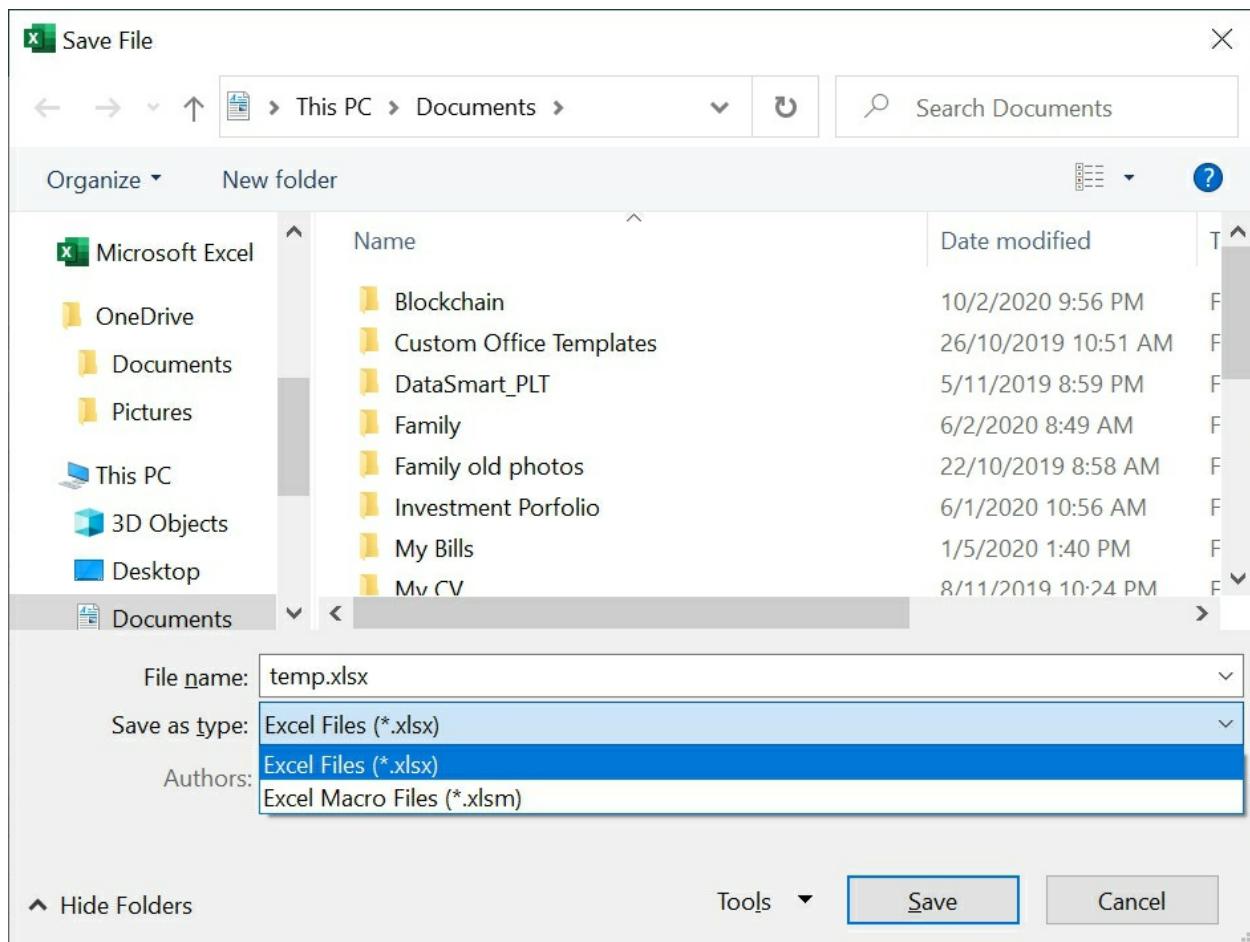
Title, ButtonText)

To save the file, use the `ActiveWorkbook.SaveAs` method, the syntax is  
`ActiveWorkbook.SaveAs Filename:=myfileName,`  
`FileFormat:=xlWorkbookNormal`

### Example 17.2

This example allows the user to save the file as normal Excel file(xlsx) or macro-enabled file(xlsm). Insert a command button on the UserForm and change its caption to Save File. Click the command button and enter the Excel VBA code below. When you run the program and click the Save button, a Save As dialog box will appear and prompt you to select a file type and type the file name to save the file, as show in Figure 17.2

```
Private Sub Cmd_Save_Click()
Dim myfileName As Variant
myfileName = Application.GetSaveAsFilename(FileFilter:= _
"Excel Files (*.xlsx), *.xlsx,Excel Macro Files (*.xlsm), *.xlsm",
Title:="Save File", InitialFileName:="temp")
If myfileName <> False Then
    ActiveWorkbook.SaveAs Filename:=myfileName,
    FileFormat:=xlWorkbookNormal
Exit Sub
End If
End Sub
```



**Figure 17.2**

### 17.3 Creating a Text File

We have learned how to open and saving files. In this chapter, we shall how to create text files in Excel VBA 365.

To create a text file, you can use the following command

Open "fileName" For Output As #fileNumber

Each text file created must have a file name and a file number for identification. As for the file name, you must also specify the path where the file will reside.

For example:

Open "c:\My Documents\sample.txt" For Output As #1

will create a text file by the name of sample.txt in the My Document folder.

The accompanying file number is 1. If you wish to create and save the file in drive A, simply change the path, as follows"

Open "A:\\sample.txt" For Output As #1

If you wish to create a HTML file, simple change the extension to .html

Open "c:\\My Documents\\sample.html" For Output As # 2

### Example 17.3 Creating a text file

```
Private Sub create_Click ()  
Dim intMsg As String  
Dim myText As String  
Dim fileSaveName As String  
Dim mypath As String  
fileSaveName = "sample.txt"  
mypath = "C:\\Users\\LENOVO\\Documents\\My Websites\\"  
Open mypath & fileSaveName For Output As #1 Open "c:\\My  
Documents\\sample.txt" For Output As #1  
intMsg = MsgBox ("File sample.txt opened")  
myText = InputBox ("Enter some words")  
Print #1, myText  
intMsg = MsgBox ("Writing a" & MyText & "to sample.txt ")  
Close #1  
intMsg = MsgBox ("File sample.txt closed")  
End Sub
```

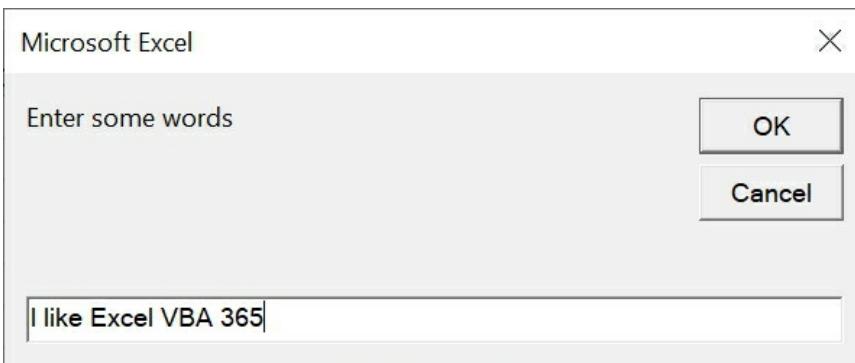
The above program will create a file sample.txt in the My Documents' folder which is ready to receive input from users. Any data input by users will be saved in this text file. Instead of print, you can also use write to save the file. After opening the file, you must always close it with the command close.

The first dialog when you run this program is shown in Figure 17.3



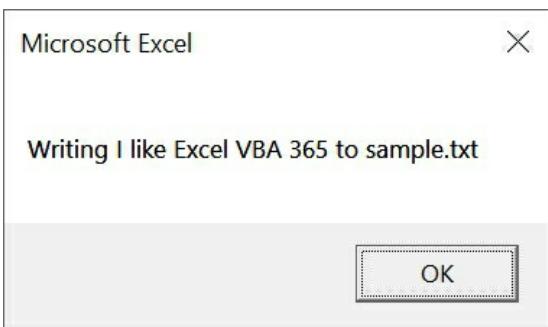
**Figure 17.3**

After clicking the OK button, you shall see this dialog where you can enter texts, as shown in Figure 17.4



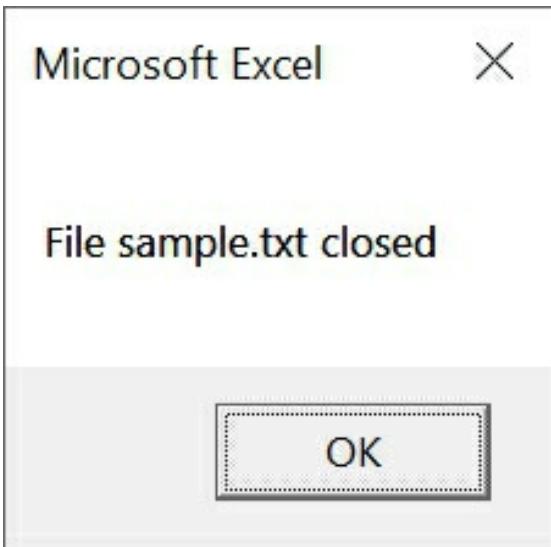
**Figure 17.4**

Upon clicking the OK button, the following dialog box will appear, as shown in Figure 17.5



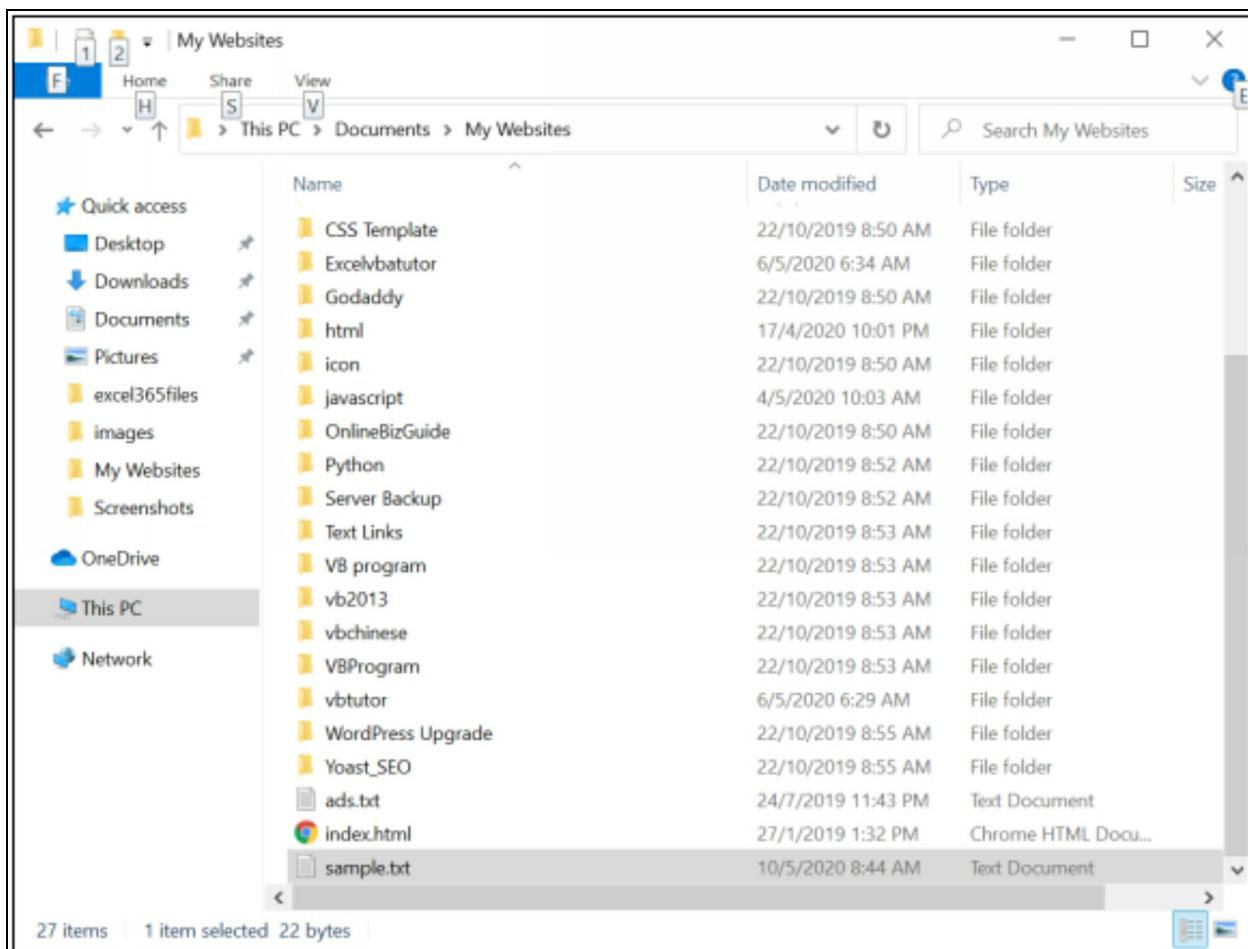
### **Figure 17.5**

Finally, after clicking the OK, the close file dialog will appear, as shown in Figure 17.6



### **Figure 17.6**

To check whether the file sample.txt has been created, you can browse the folder. Notice that the file sample.txt has been created, as shown in Figure 17.7



**Figure 17.7**

You can modify Example 17.3 by accepting input from the user via a text box. To do this, insert a text box and set its multiline property to true so that the user can type in multiline of text.

The modified code is shown below:

```
Private Sub create_Click ()  
Dim intMsg As String  
Dim myText As String  
Dim fileSaveName As String  
Dim mypath As String  
fileSaveName = "sample.txt"
```

```
mypath = "C:\Users\LENOVO\Documents\My Websites\"  
Open mypath & fileSaveName For Output As #1  
intMsg = MsgBox("File sample.txt opened")  
myText = myTextBox.Text  
Print #1, myText  
intMsg = MsgBox("sample.txt has been created ")  
Close #1  
intMsg = MsgBox("File sample.txt closed")  
End Sub
```

The output



**Figure 17.8**

## 17.4 Reading a File

To read a file created in section 13.2, you can use the input # statement. The syntax is shown below:

Open "fileName" For Input As #1

You must open the file according to its file number and the variable that holds the data. You also need to declare the variable using the DIM

command.

#### Example 17.4 Reading a text file

```
Private Sub Cmd_Read_Click()
Dim mypath As String
Dim strTextLine As String
mypath = "C:\Users\LENOVO\Documents\My Websites\sample.txt"
Open mypath For Input As #1
Do While Not EOF(1)      ' Loop until end of file.
    Line Input #1, strTextLine ' Read line into variable.
Loop
myTextBox.Text = strTextLine
Close #1
End Sub
```

This program will open the sample.txt file and display its contents in the text box, as shown in Figure 17.9

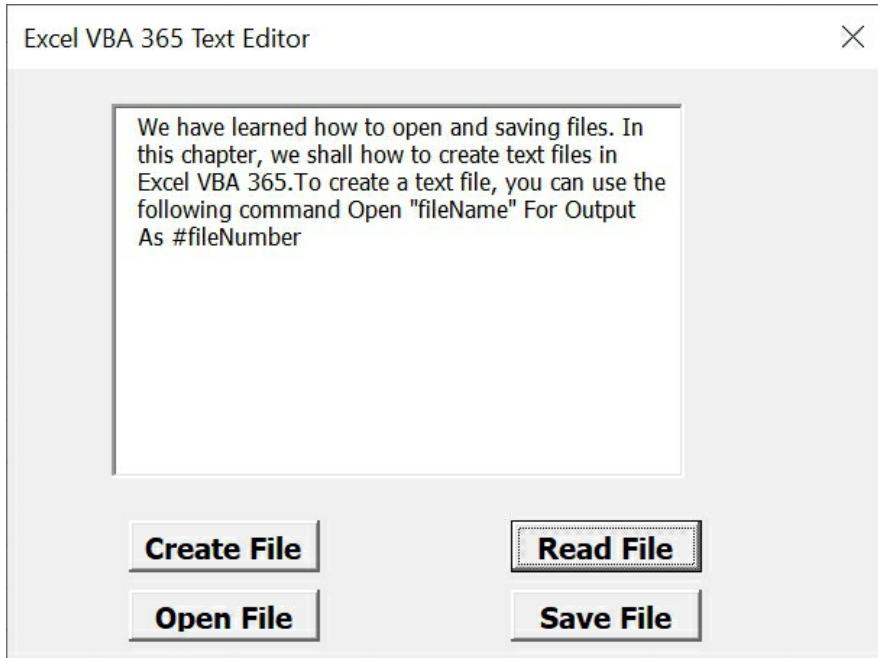


Figure 17.9

# Chapter 18 Class Modules

---

We have learned how to use modules in previous chapters, these modules are classified as standard modules. Besides modules, the userForm also come with a special kind of module known as the class module.

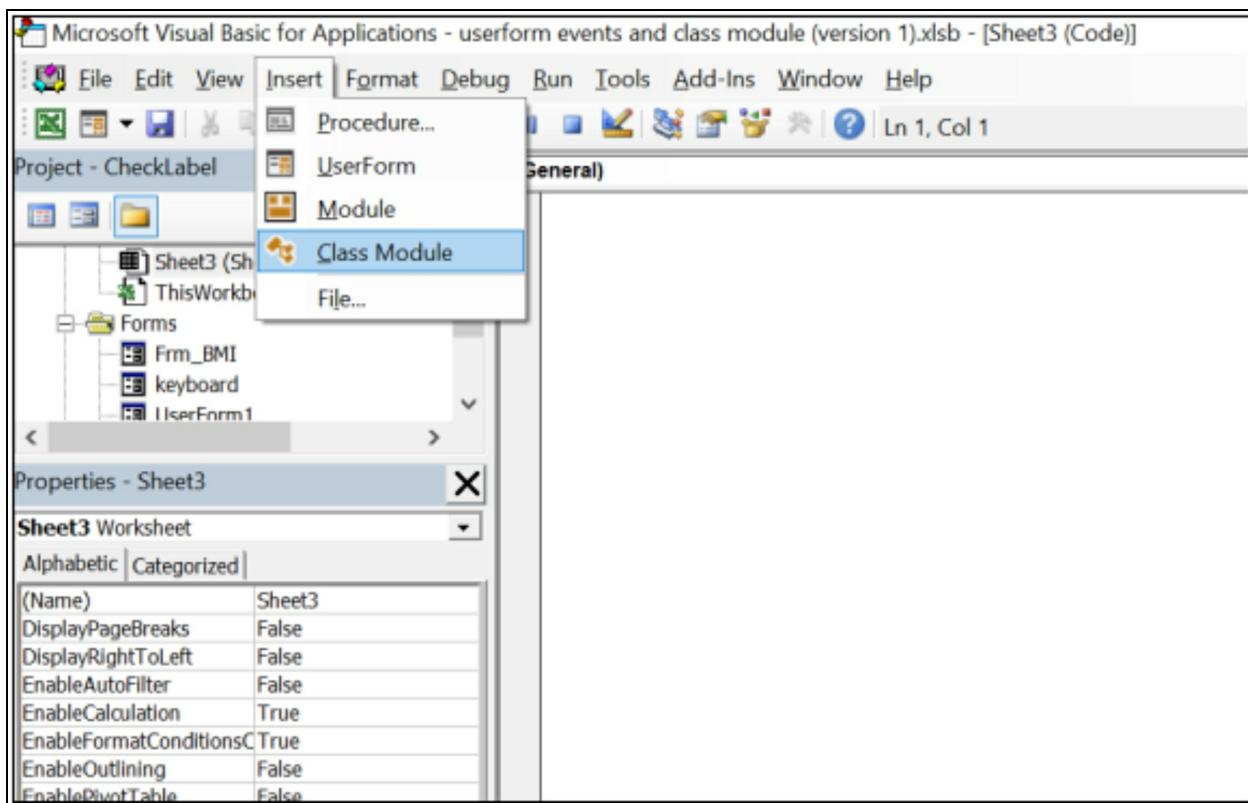
The class module allow you to create a component object model (COM) . The **object model** is a large hierarchy of all the **objects** used in Excel **VBA 365**. As discussed in earlier chapters, an Excel VBA **object** contains data, properties and methods. Examples of objects are Workbook, Worksheet, Range, Cells, command buttons and more. Using the class module, you can create your very own Excel VBA 365 objects.

The Class module is like the master design that we can based on to create new objects. Basically, there are four different components in a class module, as shown below:

- Methods (functions or sub procedures)
- Member variables
- Properties
- Events

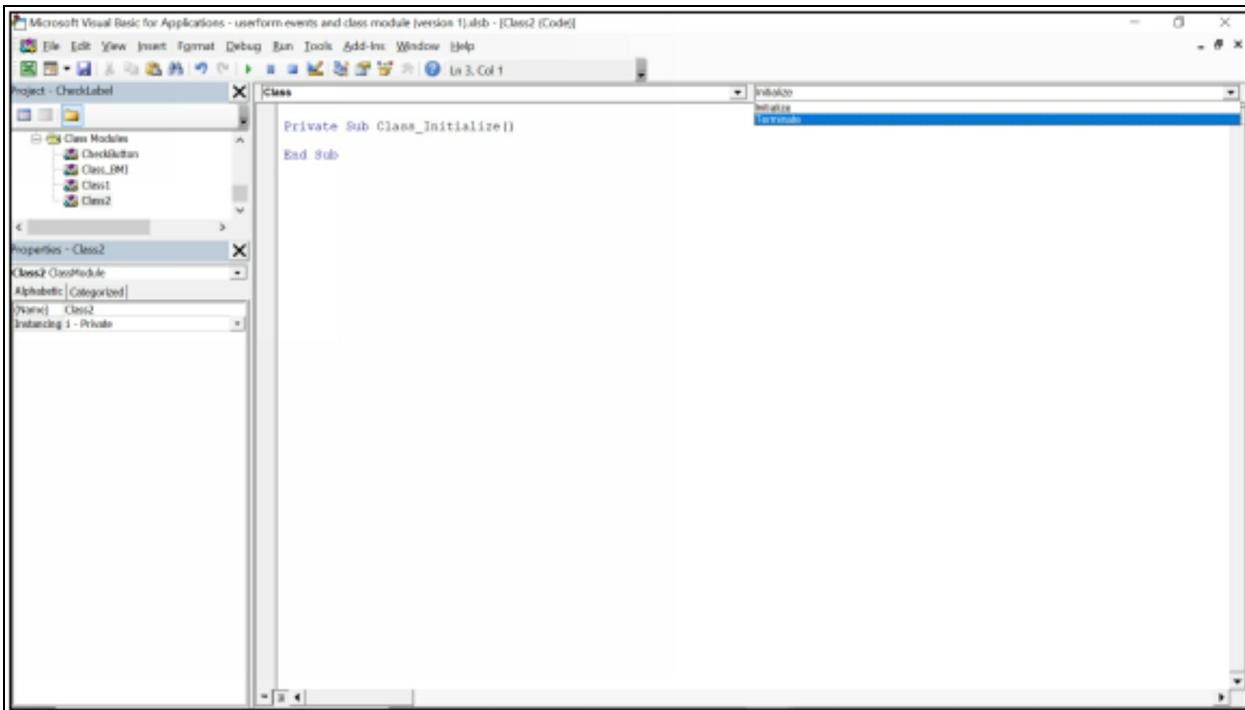
## 18.1 Creating a Class Module

To create a class module, click Insert and select Class Module from the drop-down list, as shown in Figure 18.1.



**Figure 11.1**

The class module has only one property, name and two events associated with him, i.e. Initialize and Terminate, as shown in Figure 18.2.



**Figure 18.2**

In the class module environment, you can create functions, sub procedures, properties and events. However, you cannot run the sub procedures or events in the class module itself, you need to connect a UserForm to reference them and run them in the UserForm. Let us examine Example 18.1, a BMI calculator.

### Example 18.1

This Excel VBA code calculates BMI based on your height and weight. First, insert a class module and name it as Class\_BMI. Key in the following code

```
Public Function bmi(ByVal height As Single, ByVal weight As Single)
bmi = Format((weight) / (height ^ 2), "0.00")
End Function
```

Please be reminded that we cannot use Dim to declare a member variable like bmi in the class module, we must use the declaration keyword Public or Private .

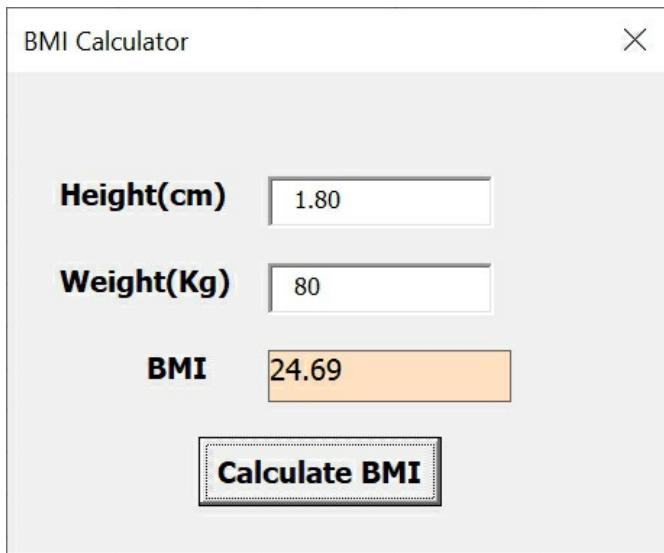
Next, insert a UserForm and name it as Frm\_BMI. To design the UI, insert two text boxes and name them as Txt\_Height and Txt\_Weight. These text

boxes are for the user to input the height and weight. In addition, insert a Label to display the BMI, name it as Lbl\_BMI.

Last, insert a command button and name it as Cmd\_BMI. Now click on the command button and enter the follow code:

```
Private Sub Cmd_BMI_Click()
Dim h, w, bmi As Variant
Dim MyBMI As New Class_BMI 'Create an instance of Class_BMI
h = Txt_Height.Text
w = Txt_Weight.Text
Lbl_BMI.Caption = MyBMI.bmi(h, w) 'Reference the method in Class_BMI
End Sub
```

The output is shown in Figure 18.3



**Figure 18.3**

## Example 18.2

This Excel VBA 365 program computes the grades of students based on their marks. In this program, insert a class module and name it as Cls\_ExamGrade. Next, insert a UserForm and name it as Frm\_ExamGrade. To design the UI, insert a text box to accept the input for marks and a Label for displaying the grades.

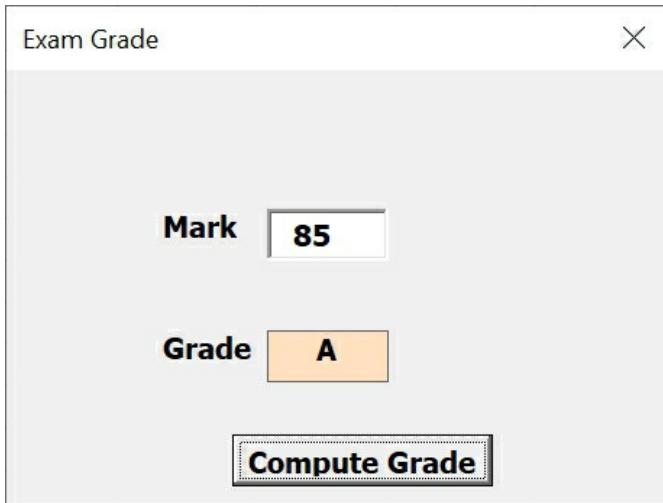
The code for Cls\_ExamGrade is as follows:

```
Public Function grade(ByVal mark As Single) As String
Select Case mark
Case 0 To 20
grade = "F"
Case 20 To 29
grade = "E"
Case 30 To 39
grade = "D"
Case 40 To 59
grade = "C"
Case 60 To 79
grade = "B"
Case 80 To 100
grade = "A"
Case Else
grade = "Error!"
End Select
End Function
```

The code for Frm\_ExamGrade

```
Private Sub Cmd_Compute_Click()
Dim mrk As Variant
Dim myGrade As New Cls_ExamGrade
mrk = Txt_Mark.Text
Lbl_Grade.Caption = myGrade.grade(mrk)
End Sub
```

The Output



**Figure 18.4**

### Example 18.3

This Excel VBA code calculate the future value (FV)of an investment. The calculation is based on the investment amount (Also known as present value or simply PV), the compound interest rate and the number of years from now. The formula to calculate this future value is

$$FV = PV * (1 + i / 100)^n$$

We will use a class module to create the function(method) to calculate the FV based on three arguments, PV, interest rate and number of years. Insert a class module and name it as Cls\_FV. Next, insert a UserForm and name it as Frm\_FV.

In the class module, key in the following code:

```
Public Function FV(ByVal PV As Double, ByVal Num_Year As Single,  
ByVal Int_Rate As Single)  
FV = Format(PV * (1 + Int_Rate / 100) ^ Num_Year, "#,##0.00")  
End Function
```

To design the UI for Frm\_FV, insert three text boxes and name them as Txt\_PV, Txt\_NumYear and Txt\_Interest. Also insert a Label to display the PV. Last, insert a command button and name it as Cmd\_Calculate.

The Code

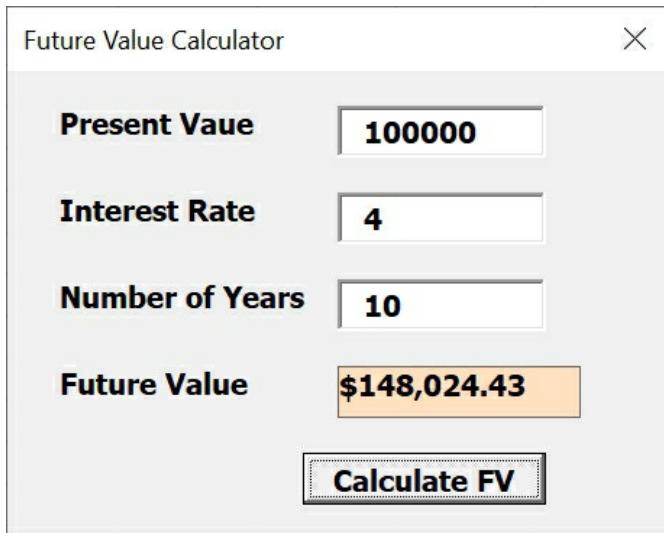
```
Private Sub Cmd_Calculate_Click()
```

```

Dim PVal As Double
Dim num_y, i As Single
Dim FValue As New Cls_FV
PVal = Txt_PV.Text
num_y = Txt_NumYear.Text
i = Txt_Interest.Text
Lbl_FV.Caption = FValue.FV(PVal, num_y, i)
End Sub

```

### The Output



**Figure 18.5**

## 18.2 Class Module Properties

Having created your object in the class module, you can create properties for the object. There are three methods you must use to create and access the properties. The methods are Let, Get and Set. Let sets a value in the class, Get returns a value or object from the class and Set sets an object in the class.

The syntaxes to use Let, Get and Set are as follows:

```

Public Property Get varname() As Type
End Property

```

```

Public Property Let varname (argument As Type )
End Property

```

```
Public Property Set (varname As Type )  
End Property
```

### Example

```
Public Property Let Fname(mdata As string)  
    xFname=mdata  
End Property
```

To read the property, use the Get method, as follows:

```
Public Property Get Fname() As String  
Fname=xFname  
End Property
```

Make sure that the Let and Get property statements have the same name, i.e. Fname in the case above.

### Example 18.4

This example simulates an ATM machine where the account holder can check the balance, deposit money as well as withdraw money.

First, insert a class module and change its name to ClsATM then enter the following code:

```
' Member variable  
Private x_balance As Double  
  
' Properties  
Property Get AcctBalance() As Double  
    AcctBalance = x_balance  
End Property  
  
Property Let AcctBalance(value As Double)  
    x_balance = value  
End Property
```

'Event - triggered when class initializes

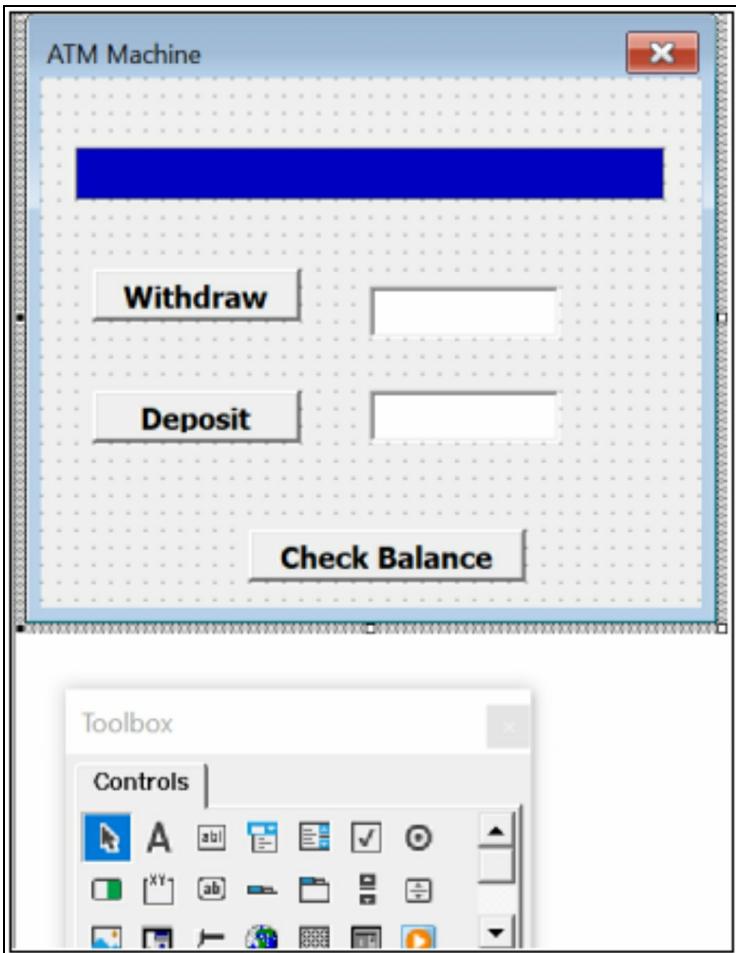
```
Private Sub Class_Initialize()
    x_balance = 100
End Sub

' Methods
Public Function Withdraw(ByVal amount As Double)
    x_balance = x_balance - amount
End Function

Public Function Deposit(ByVal amount As Double)
    x_balance = x_balance + amount
End Function
```

To design the UI of the virtual ATM machine, insert a UserForm and change its name to Frm\_ATM. Next, insert a Label to serve the purpose as a display panel. In addition, insert two text boxes to let the account holder enter the amount to withdraw and the amount to deposit. Last, add a command button to let the account holder to check his or her balance.

The design UI is as shown in Figure 18.6



**Figure 18.6**

Next, enter the code in Frm\_ATM, as follows:

```
Dim myAccount As New ClsATM 'Declare a new instance of the object  
ClsATM
```

```
    Dim d, w As Variant
```

```
Private Sub Cmd_CheckBalance_Click()
```

```
    CheckAccount
```

```
End Sub
```

```
Private Sub Cmd_Check_Click()
```

```
    Lbl_Panel.Caption = "Your balance is: " & myAccount.AcctBalance  
End Sub
```

```
Private Sub Cmd_Deposit_Click()
```

```
d = TxtD_Amount.Text  
On Error GoTo err 'to prevent error when the user enters a string
```

```
myAccount.Deposit d  
TxtD_Amount.Text = ""  
Exit Sub
```

```
err:  
Lbl_Panel.Caption = Amount cannot be 0"  
End Sub
```

```
Private Sub Cmd_Withdraw_Click()  
w = TxtW_Amount.Text  
If w > myAccount.AcctBalance Then  
    Lbl_Panel.Caption = "Withdrawal denied, not enough money"  
Else  
    myAccount.Withdraw w  
    End If  
    TxtW_Amount.Text = ""  
End Sub
```

Note that we are using the property AcctBalance that we have created in the class module ClsATM. When you run the application and click the Check Balance button, the output is as shown in Figure 18.7

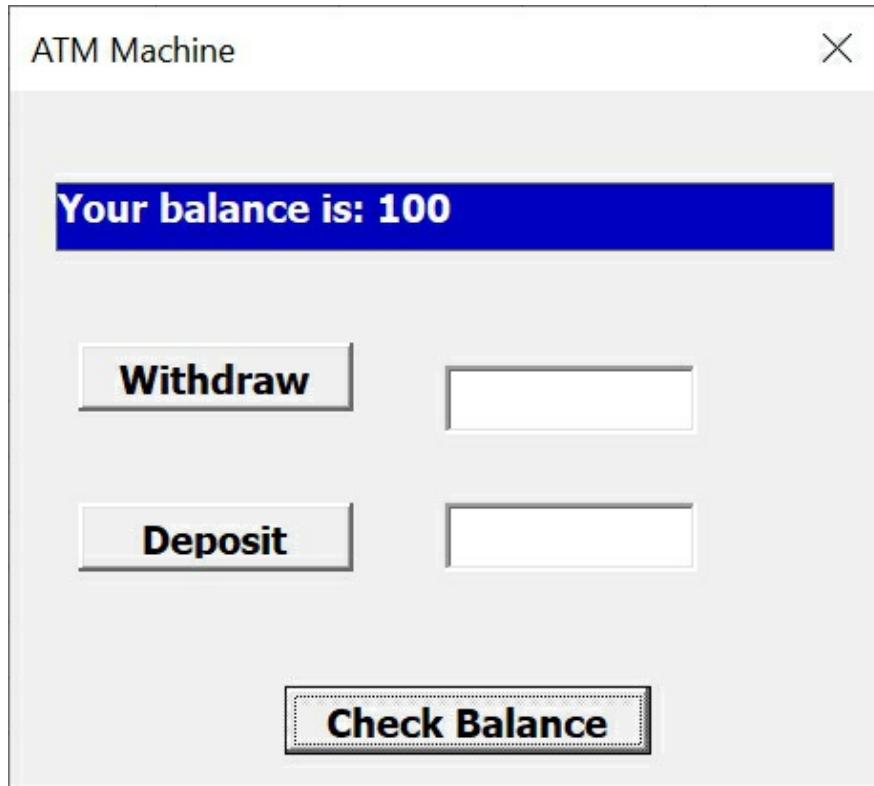
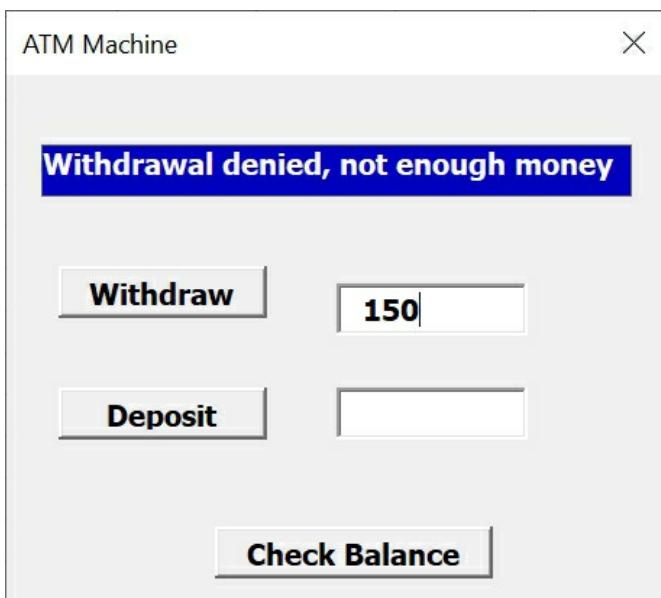


Figure 18.7

Notice that the balance is 100 as it was initialized in the class module. If you try to withdraw an amount more than the balance, you will get the following message shown on the panel, as in Figure 18.8



## Figure 18.8

### Example 18.5

This is a decision-making app that can determine the buying decision based on shoe's size and its price. In this Excel VBA program, insert a class module and name it as `Cls_BuyDecision`. In the class module environment, declare a Boolean variable `buy` as follows:

```
Private buy As Boolean
```

Next, use the `Let` and `Get` to create the property `BuyDecision`.

```
Property Get BuyDecision() As Boolean
```

```
    BuyDecision = buy
```

```
End Property
```

```
Property Let BuyDecision(value As Boolean)
```

```
    BuyDecision = value
```

```
End Property
```

Next, create a sub procedure known as `buy_decision` that has two arguments, `size` and `price`. A Boolean variable `buy` is included to help in decision making. It takes a value of true or false. If the size and price entered fulfilled the requirement via the `If...Then...Else` statement, `buy` is assigned a value of true, else it is assigned a value of false. The code is as follows:

```
Public Sub buy_decision(ByVal size As Integer, ByVal price As Single)
```

```
If size >= 7 And price <= 200 Then
```

```
    buy = True
```

```
    MsgBox ("Buy")
```

```
Else: buy = False
```

```
    MsgBox ("Don't Buy")
```

```
End If
```

```
End Sub
```

We also set the `buy` value as True when the class initializes:

```
Private Sub Class_Initialize()
```

```
    buy = True
```

```
End Sub
```

Now, proceed to insert a UserForm to build the UI. Insert a UserForm and

name it as Frm\_BuyDecision. Next, insert two text boxes and name them as Txt\_ShoeSize and Txt\_ShoePrice. Finally, insert two command buttons, one is to decide on the buying decision and the other one to check on the decision.

The code is shown overleaf.

```
Dim mbuy As New Cls_BuyDecison 'Create a new instant of  
Cls_ButDecison  
Dim shoe_size As Integer, shoe_price As Single  
  
Private Sub Cmd_BuyDecison_Click()  
    shoe_size = Txt_ShoeSize.Text  
    shoe_price = Txt_ShoePrice.Text  
    mbuy.buy_decision shoe_size, shoe_price  
End Sub  
  
'Checking Buying Decision  
Private Sub Cmd_Check_Click()  
    Dim dec As String  
    MsgBox "What was the buying decison?"  
    If mbuy.BuyDecision = True Then  
        MsgBox ("The decison is buy")  
    Else  
        MsgBox (" The decison is don't buy")  
    End If  
End Sub
```

When you run the application and click check decision, you will get the following output



**Figure 18.9**

The decision is buy because the class initialized with `buy=True`. Now, try entering the following values and see what the buying decision is:

A screenshot of a "Decision Making App" window. The title bar says "Decison Making App" and has a close button "X". There are two input fields: "Shoe Size" containing "6" and "Shoe Price" containing "180". Below the inputs are two buttons: "Buy Decision" and "Check Decision".

**Figure 18.10**

The decision is Don't buy.



**Figure 18.11**

Now, click on the Check Decision button, the output is as shown in Figure 18.12



**Figure 18.12**

Note that the default decision has been changed to False.

### Example 18.6 Virtual Keyboard

In this example, we create a virtual keyboard in Excel VBA 365. To design the UI, we need to insert 69 command buttons as representation of the keys on the keyboard. If we use a normal module, we need to program each button which will make writing code cumbersome and inefficient. Therefore, we need to create a new object that replaces the default command buttons using the class module. This object will be an array or a collection of command buttons, therefore, we just need to write code for this object instead of each

command button.

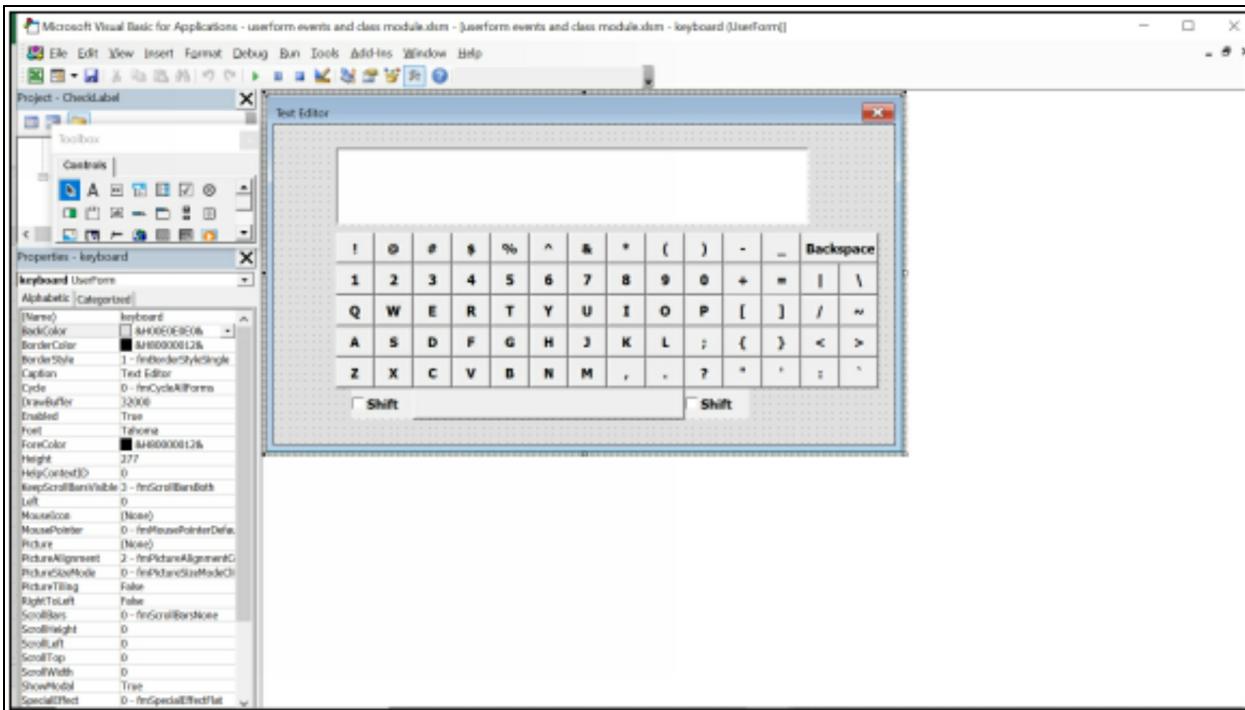
First, insert a class module and name it as CheckButton. Next, we must declare a public object variable that represents all possible events of a certain controltype, in our case, CommandButton. So, the first line of code is

```
Public WithEvents ctl_commandbutton As MSForms.CommandButton
```

Subsequently, we must write an event procedure for our object variable. However, before we proceed, we need to insert a UserForm and name it as keyboard. In the UserForm, we need to create a collection or array of the command buttons, as follows:

```
Public ctr_collection As New Collection  
Private Sub Userform_Initialize()  
    For Each ctl In Controls  
        If TypeName(ctl) = "CommandButton" Then  
            ctr_collection.Add New CheckButton, ctl.name  
            Set ctr_collection(ctl.name).ctl_commandbutton = ctl  
        End If  
    Next  
End Sub
```

Before we go back to the class module, let us design the UI as shown in Figure 18.13



**Figure 18.13 The Keyboard UI**

We insert 69 command buttons to represent the keys, and two check boxes to act as the shift keys. We use the caption property of the command buttons to write the character in the TextBox (names as `Txt_Panel`). To use this text box, we must include its parent UserForm name `keyboard` as this object is not created in the class module.

Now we go back to the class module environment and enter the following code:

```
Sub ct_CommandButton_Click()
    Dim char As String
    Dim i, j As Integer
    i = ct_CommandButton.TabIndex

    'To check whether the shift keys are being checked
    'If False, we convert the characters to lowercases
    '69 is the TabIndex value of the Shift keys
    If keyboard.Chk_Shift.value = False And keyboard.Chk_Shift2.value = False
        And i <> 69 Then
            char = LCase(ct_CommandButton.Caption)
        ElseIf Display.Chk_Shift.value = True Or keyboard.Chk_Shift2.value = True
            And i <> 69 Then
```

```

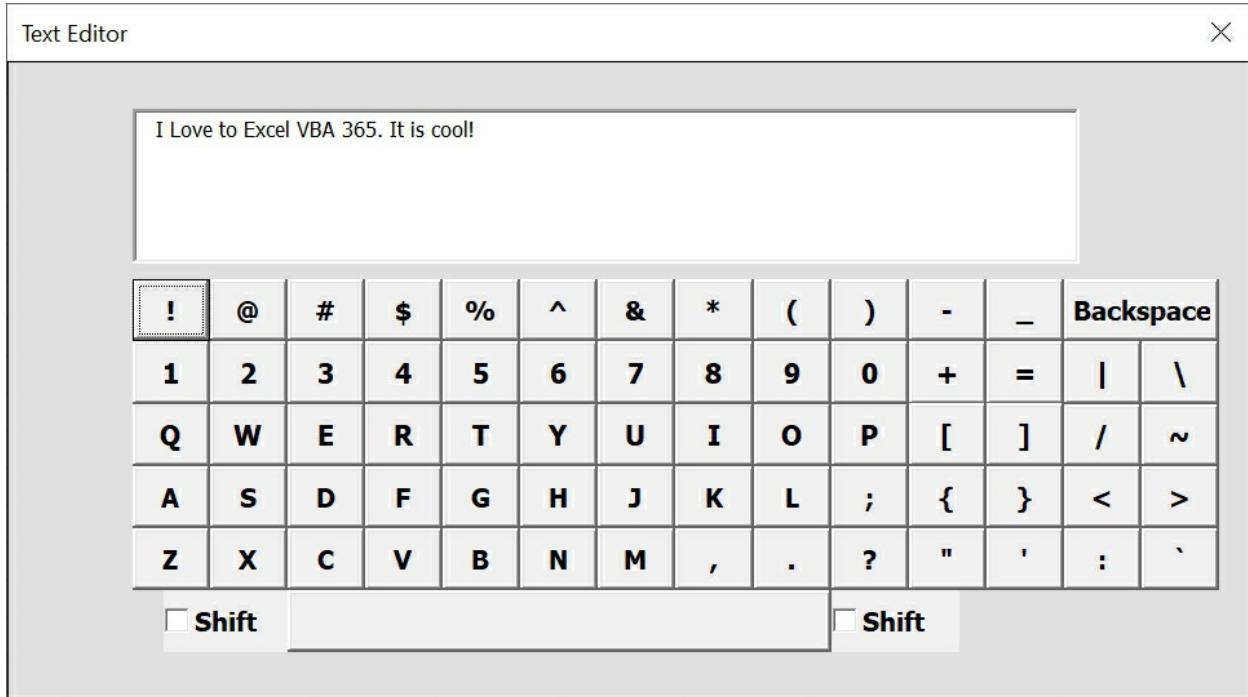
char = ct_CommandButton.Caption
End If
If ct_CommandButton.Caption = "" Then
    char = Chr(32)
End If
'This is the code for the backspace key.
'We use the Len and Left function to delete the left most character one by one
    If ct_CommandButton.TabIndex = 69 Then
        j = Len(Display.Txt_Panel.Text) - 1

        keyboard.Txt_Panel.Text = Left(keyboard.Txt_Panel.Text, j)
    End If
    keyboard.Txt_Panel.Text = keyboard.Txt_Panel.Text + char
End Sub

```

This code will apply to each command button. Therefore, you can see that we only need to write one event procedure for clicking the command buttons, instead of writing the same procedure for all the command buttons.

The Output is as shown in Figure 18.14



**Figure 18.14 The Virtual Keyboard**

## Example 18.7

This example shows you how to use Get and Set methods. This excel VBA program compute the exam grades based on the students' marks.

First, insert a class module and name it as ClsStudentExam. Enter the following code:

'Declare the member variables

Private StudentName As String

Private StudentMarks As Double

'Create the object properties using let and gets

Public Property Let Name(strName As String)

StudentName = strName

End Property

Public Property Get Name() As String

Name = StudentName

End Property

Public Property Let Marks(Marks As Double)

StudentMarks = Marks

End Property

Public Property Get Marks() As Double

Marks = StudentMarks

End Property

'Create the method to compute the grades

Public Function grade() As String

Dim ExamGrade As String

grade = ExamGrade

Select Case StudentMarks

Case 0 To 20

grade = "F"

Case 20 To 29

grade = "E"

Case 30 To 39

```
grade = "D"
Case 40 To 59
grade = "C"
Case 60 To 79
grade = "B"
Case 80 To 100
grade = "A"
Case Else
grade = "Error!"
End Select
```

End Function

Next, insert a UserForm and name it Frm\_StudentMark. Additionally, insert a command button and name it as Cmd\_Compute as well as a list box and name it ListStuName.

First, create a sub procedure to compute the grade

```
Sub ComputeMark()
```

'Declare a variable and define it as a reference to the class.

```
Dim TheStudent As clsStudentExam
```

'Use the Set method and the New keyword to instantiate the clsStudentExam object

```
Set TheStudent = New clsStudentExam
```

```
StuName = InputBox("Enter the Student Name")
```

```
TheStudent.Name = StuName
```

```
MsgBox "The student name is " & TheStudent.Name
```

```
StuMark = InputBox("Enter the Student Mark")
```

```
If StuMark > 100 Or StuMark < 0 Then
```

```
MsgBox "Please reenter the mark"
```

```
StuMark = InputBox("Enter the Student Mark")
```

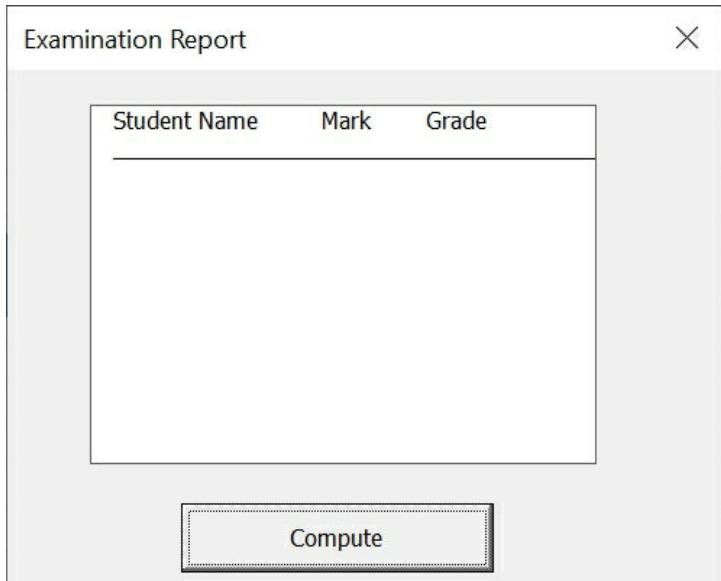
```
End If
```

```
TheStudent.Marks = StuMark  
MsgBox TheStudent.Name & "'s Mark is " & TheStudent.Marks  
MsgBox TheStudent.Name & " obtained Grade " & TheStudent.grade  
  
MsgBox TheStudent.Name & " has got " & TheStudent.Marks & " marks  
with a Grade " & TheStudent.grade  
'List the results in the list box using the AddItem method  
ListStuName.AddItem TheStudent.Name & vbTab & vbTab &  
TheStudent.Marks & vbTab & TheStudent.grade  
  
End Sub  
Private Sub Cmd_Compute_Click()  
'Call the sub procedure  
ComputeMark  
End Sub
```

Add headings to the list box when the UserForm initializes

```
Private Sub UserForm_Initialize()  
ListStuName.AddItem "Student Name" & vbTab & "Mark" & vbTab &  
"Grade"  
ListStuName.AddItem  
"  
-----"  
End Sub
```

When you run the program, you will be presented with a blank list box with headings, as shown in Figure 18.15



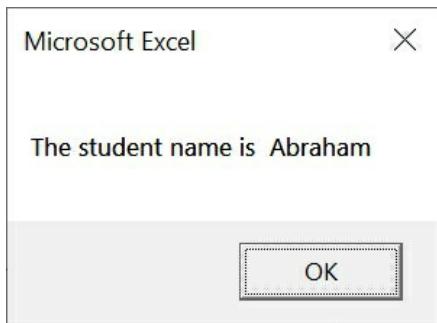
**Figure 18.15**

When you click compute, an input box will appear, asking you to enter the student name, as shown in Figure 18.16



**Figure 18.16**

After pressing OK, a message box will appear showing the student name was entered correctly, as shown in Figure 18.17.



**Figure 18.17**

Upon pressing the OK button, you will be prompted to enter the student marks, as shown in Figure 18.18



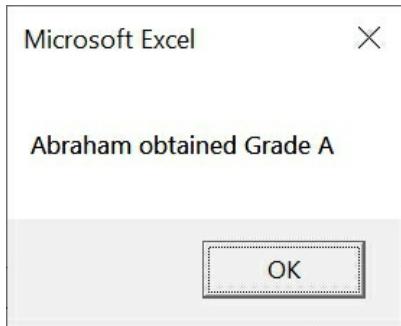
**Figure 18.18**

Pressing the OK button will confirm the mark, as shown in Figure 18.19



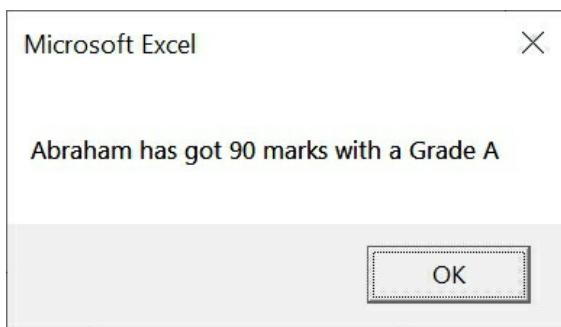
**Figure 18.19**

Pressing the OK button will show the grade obtained by the student, as shown in Figure 18.20.



**Figure 18.20**

The next dialog will show the mark and grade obtain by the student, as shown in Figure 18.21



**Figure 18.21**

Finally, clicking the OK button will register the results in the list box, as shown in Figure 18.22

Examination Report		
Student Name	Mark	Grade
Abraham	90	A
<b>Compute</b>		

**Figure 18.22**

After entering 10 results, you will see that the list box has been populated with the results, as shown in Figure 18.23

Student Name	Mark	Grade
Abraham	90	A
Bowen	70	B
Casey	50	C
Dan	85	A
Francis	40	C
Gan	25	E
Hannah	37	D
Irene	65	B
Lim	56	C
Menon	10	F

**Figure 18.23**

# Chapter 19 Drawing Charts

---

Charts are important as they are the visual representations of the spreadsheet data. This chapter will teach you how to create various types of charts using Excel VBA code.

Excel VBA 365 has made charting engine as part of the Shape object. It is also an object by itself. We can create charts on a worksheet of their own or embed them into an existing worksheet. The chart sheet is the Chart object whereas the embedded chart is part of the shape collection for the worksheet.

To create this chart drawing app, start a new Excel VBA 365 workbook. Next, click on the Developer tab to launch the Excel VBA design environment. Insert three command buttons, one is for the user to enter the customized range, another one is to draw the chart and the final one is to clear charts. Besides that, insert 6 option buttons to let the user select the type of chart he or she wants to draw.

The type of chart we can draw depends on the ChartType attributes(properties). The list of ChartType attributes is as shown in the table below:

**Table 19.1**

Property	Chart type
xlArea	Area Chart
xlBar	Bar Chart
xlColumn	Column Chart
xlLine	Line Chart
xl3DLine	3D Line Chart
xlPie	Pie Chart
xl3DPieExploded	Exploded 3D Pie Chart
xl3DBarStacked	3D Stacked Bar Chart
xlXYScatter	XY Scatter Chart

xl3DArea	3D Area Chart
xl3DColumn	3D Column Chart
xl3DLine	3D Line Chart
xlBubble	Bubble Chart

The next step is to create a table with some hypothetical data. In our example, we call it a sales performance table. The design UI is as shown in Figure E19.1

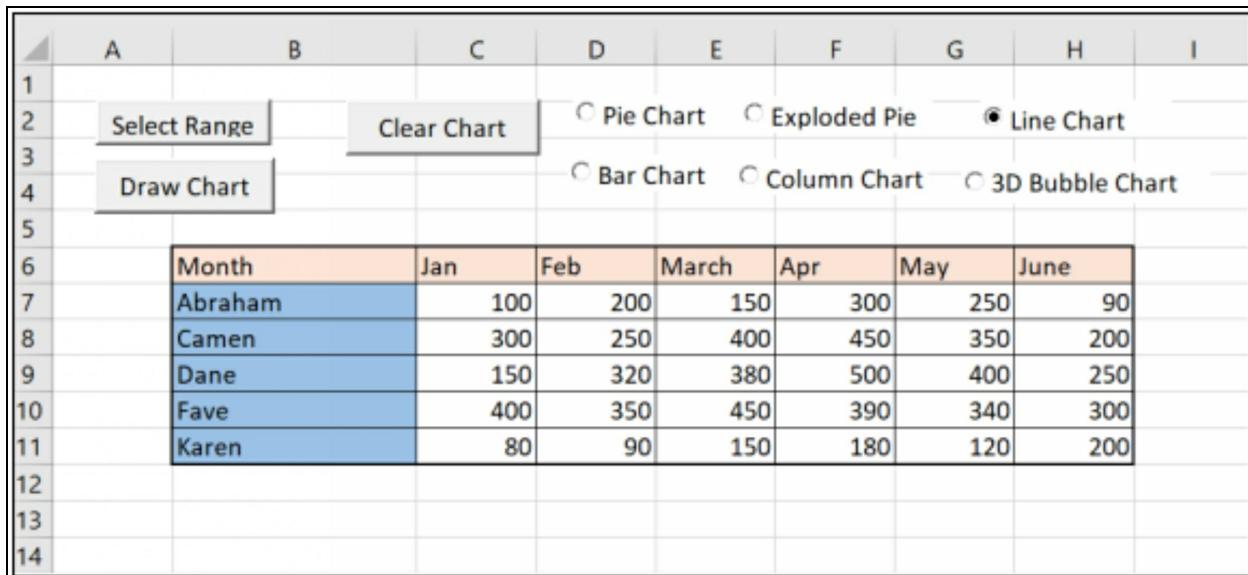


Figure 19.1

### The Code to Select the Range

We create a sub procedure to let user enter the range of data via an InputBox.

```
Static Sub selectRng()
rng = InputBox("Enter the range, eg. a1:e12")
End Sub
```

This sub procedure will be called when the user clicks the Select Range command button. The code is as follows:

```
Static Sub Cmd_SelectRng_Click()
selectRng
End Sub
```

In the general section of the code, we define two variables, rng and ctype. rng is for accepting the range value input by the user and ctype is to accept different values of ChartType

### **The Sub Procedure to Draw the Charts**

```
Sub draw()
    ActiveSheet.Shapes.AddChart.Select
    ActiveChart.ChartType = ctype
    ActiveChart.PlotArea.Select
    ActiveChart.SetSourceData Source:=Range(rng)
    ActiveChart.HasTitle = True
    ActiveChart.ChartTitle.Text = "Sales Performance"
End Sub
```

### **The Code to Draw the Charts**

```
Private Sub Cmd_DrawChart_Click()
    If Opt_Pie.Value = True Then
        ctype = xlPie
    ElseIf Opt_ExplodePie.Value = True Then
        ctype = xl3DPieExploded
    ElseIf Opt_Line.Value = True Then
        ctype = xlLine
    ElseIf Opt_Bar.Value = True Then
        ctype = xlBarStacked
    ElseIf Opt_ColumnChart.Value = True Then
        ctype = xlColumnStacked
    Else
        ctype = xlBubble3DEffect
    End If

    draw

    With ActiveChart.Parent
```

'To position the chart

.Top = Range("J4").Top

.Left = Range("J10").Left

'define the height of the chart

.Height = 250

.Width = 350

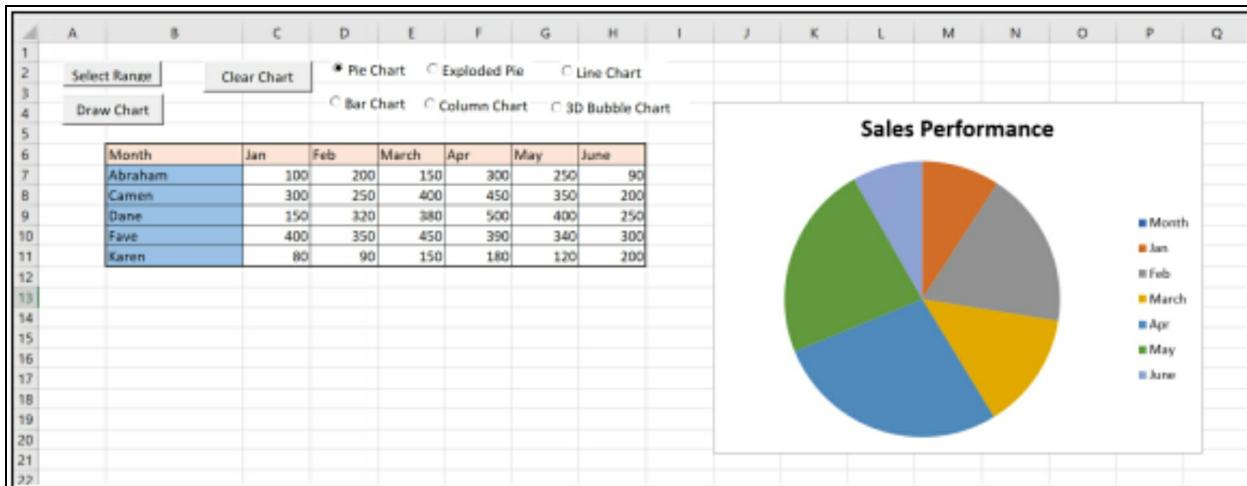
End With

End Sub

## The Code to Delete the Chart

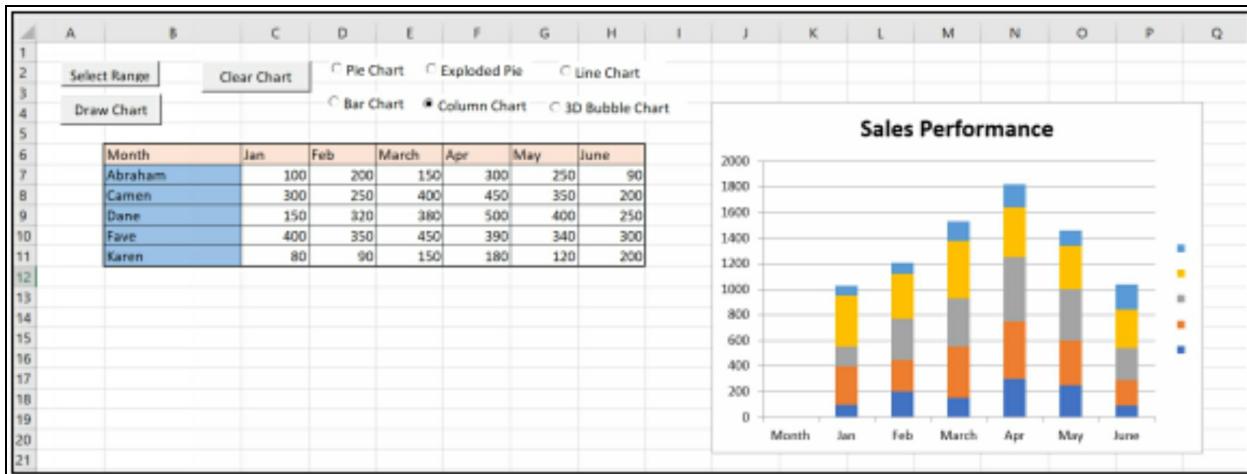
```
Private Sub Cmd_ClearChart_Click()
    Worksheets("Sheet1").ChartObjects.Delete
End Sub
```

The pie chart is shown in Figure 19.2



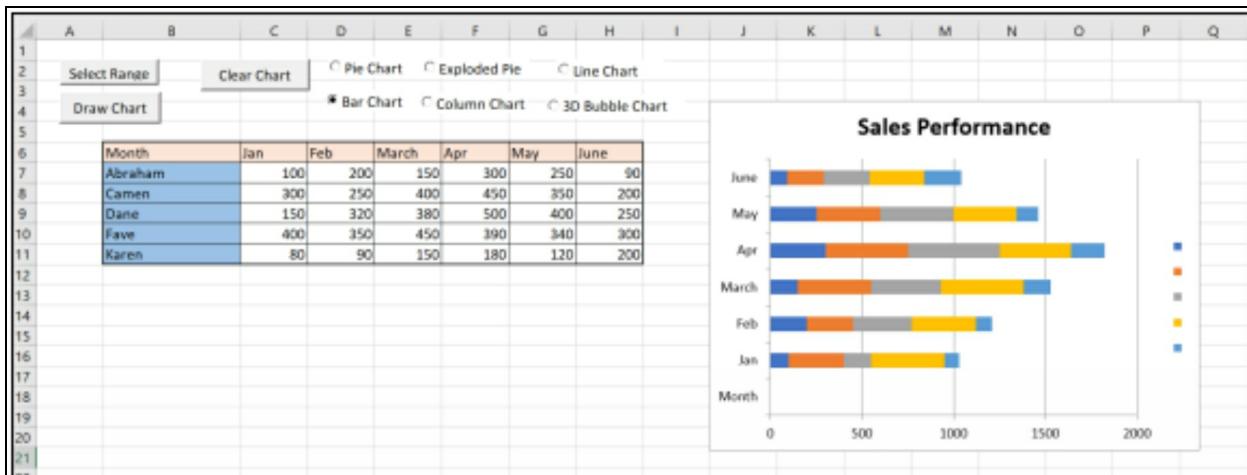
**Figure 19.2**

The column chart is as shown in Figure 19.3



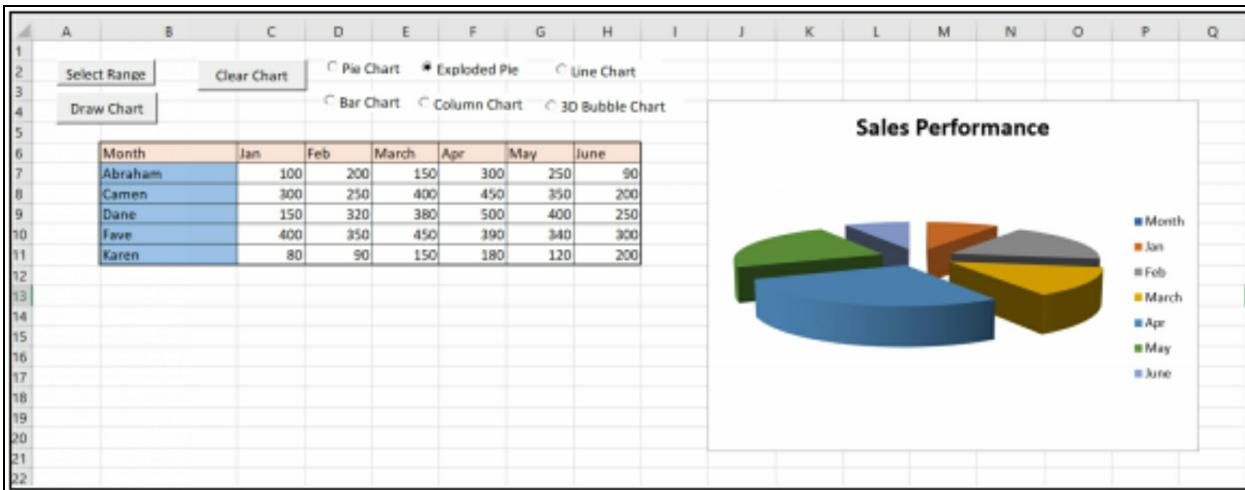
**Figure 19.3**

The bar chart is as shown in Figure 19.4



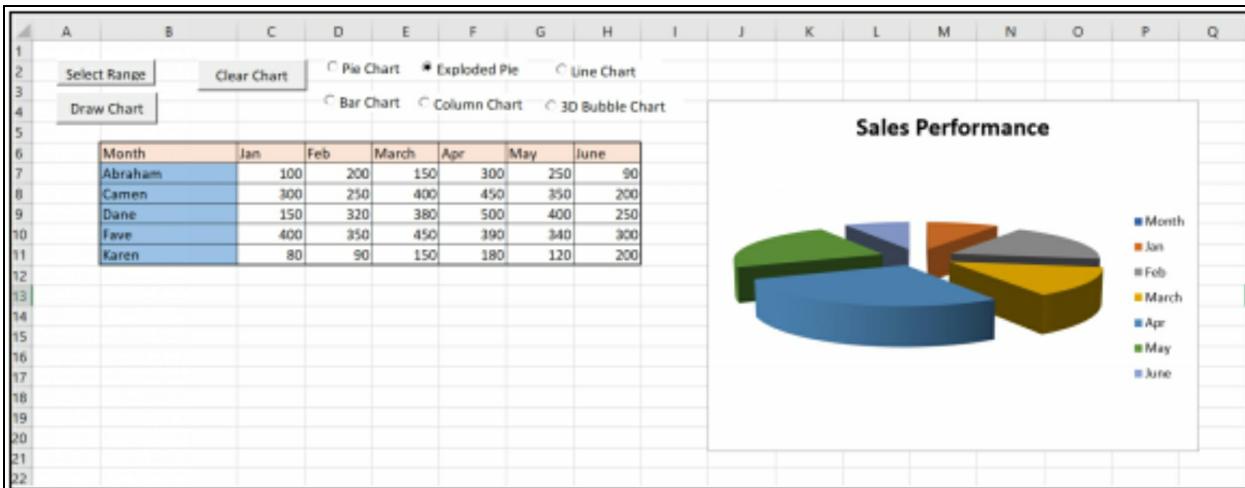
**Figure 19.4**

The explode pie chart is as shown in Figure 19.5



**Figure 19.5**

The bubble chart is as shown in Figure 19.6



**Figure 19.6**

# Excel VBA 356 Examples

## Example 1 BMI Calculator

Body Mass Index (BMI) is a standard measure for our health status. If your BMI is too high, it means you are overweight and would likely face a host of potential health issues associated with high BMI, such as hypertension, heart diseases, diabetics and many others.

The formula for calculating BMI is

$$\text{BMI} = \text{weight} / (\text{height})^2$$

The function Round is to round the value to a certain decimal place. It takes the format Round(x, n), where n is the number to be rounded and n is the number of decimal places. The second part of the program uses the If...Then...Else statement to evaluate the weight level. The code for BMI calculator is illustrated below:

```
Private Sub CommandButton1_Click()
Dim weight, height, bmi, x As Single
weight = Cells(2, 2)
height = Cells(3, 2)
bmi = (weight) / height ^ 2
Cells(4, 2) = Round(bmi, 1)
If bmi <= 15 Then
    Cells(5, 2) = "Under weight"
ElseIf bmi > 15 And bmi <= 25 Then
    Cells(5, 2) = "Optimum weight"
Else
    Cells(5, 2) = "Over weight"
End If
End Sub
```

	A	B	C
1			
2	Weight(Kg)	85	
3	Height(cm)	1.8	
4	BMI	26.2	
5	Comment	Over weight	
6			
7			
8	Calculate BMI		
9			
10			
11			
12			

**Figure E.g.1 BMI Calculator**

## Example 2 Financial Calculator

This is an Excel VBA 365 program that can calculate monthly payment for the loan taken from the bank. The formula to calculate periodic payment is shown below, where PVIFA is known as present value interest factor for an annuity.

$$\text{Payment} = \text{Initial Principal} / \text{PVIFA},$$

The formula to compute PVIFA is

$$1/i - 1/(i(1+i)^n)$$

where n is the number of payments. Normally you can check up a financial table for the value of PVIFA and then calculate the payments manually. The function Format is to determine the number of decimal places and the use of the \$ sign.

Below is the code for the financial calculator:

```
Private Sub CommandButton1_Click()
    Dim N As Integer
    Dim p, pmt, rate, I, PVIFA As Double
    p = Cells(2, 2)
    rate = Cells(3, 2)
    N = Cells(4, 2) * 12
    I = (rate / 100) / 12
    PVIFA = 1 / I - 1 / (I * (1 + I) ^ N)
    pmt = p / PVIFA
    Cells(5, 2) = Format(pmt, "$#,##0.00")
End Sub
```

The above financial VBA calculator can also be programmed using the built-in worksheet function, PMT. It is very much easier to program than the

previous one. The format of this function is

WorksheetFunction.pmt (rate, N, amount)

Where rate is the interest rate, N is the period of payments (of number of periodic payments) and amount is the amount borrowed.

People usually key in the annual interest rate as an integer rather than in decimal form, so we need to divide the rate by 100 and then divide again by 12 to get the monthly rate.

The negative sign is placed in front of the amount borrowed because this is the amount the borrower owed the financial institute, so it should be negative. If we don't put a negative, the payment will have a negative sign. The code is shown overleaf.

```
Private Sub CommandButton1_Click ()  
  
    Dim rate, N As Integer  
    Dim amt, payment As Double  
    amt = Cells(2, 2)  
    rate = (Cells(3, 2) / 100) / 12  
    N = Cells(4, 2) * 12  
    payment = WorksheetFunction.pmt(rate, N, -amt)  
    Cells(5, 2) = Format(payment, "$##,##,.00")  
  
End Sub
```

	A	B	C
1			
2	Amount Borrowed	\$80,000.00	
3	Interest Rate		4
4	Number of Years		25
5	Monthly Payment		\$422.27
6			
7			
8			
9			
10	Calculate		
11			
12			
13			
14			

**Figure E.g.2 Financial Calculator**

## Example 3 Investment Calculator

In order to get one million dollars in the future, we need to calculate the initial investment based on the interest rate and the length of a period, usually in years. The formula is

WorksheetFunction.PV (rate, N, periodic payment, amount, due)

Where rate is the interest rate, N is the length of the period and amount is the amount borrowed. Below is the Excel VBA code for the investment Calculator:

```
Private Sub Cmd_Calculate_Click ()  
Dim F_Money, Int_Rate, Investment As Double  
Dim numYear As Single  
F_Money = Cells(2, 2)  
Int_Rate = (Cells(3, 2) / 100)  
numYear = Cells(4, 2)  
Investment = PV(Int_Rate, numYear, 0, F_Money, 1)  
Cells(5, 2) = Format(-Investment, "$##,##0.00")  
End Sub
```

	A	B	C
1			
2	Future Value	\$1,000,000.00	
3	Interest Rate	5	
4	Number of Years	30	
5	Initial Investment	\$231,377.45	
6			
7			
8			
9		Calculate	
10			
11			
12			

Figure E.g.3

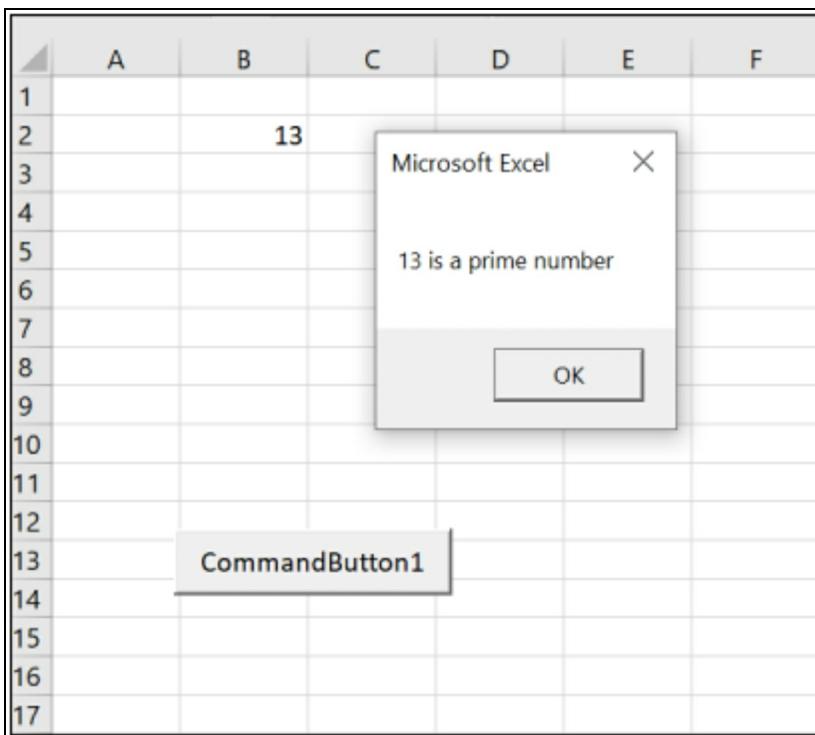
## Example 4 Prime Number Tester

This Excel VBA 365 program will test whether a number entered by the user is a prime number or not. Prime number is a number that cannot be divided by other numbers other than itself, it includes 2 but exclude 1 and 0 and all the negative numbers.

In this program, we use the `Select Case...End Select` statement to determine whether a number entered by a user is a prime number or not. For case 1, all numbers that are less than 2 are not prime numbers. In Case 2, if the number is 2, it is a prime number. In the last case, if the number N is more than 2, we divide this number by all the numbers from 3,4,5,6,.....up to N-1, if it can be divided by any of these numbers, it is not a prime number, otherwise it is a prime number. Besides, we also used a tag="Not Prime" to identify the number that is not prime, so that when the routine exits the loop, the label will display the correct answer. Below is the code:

```
Private Sub CommandButton1_Click ()  
Dim N, D As Single  
Dim tag As String  
N = Cells (2, 2)  
Select Case N  
Case Is < 2  
    MsgBox "It is not a prime number"  
Case Is = 2  
    MsgBox "It is a prime number"  
Case Is > 2  
    D = 2  
    Do  
        If N / D = Int(N / D) Then  
            MsgBox "It is not a prime number"  
            tag = "Not Prime"  
        Exit Do  
    End If
```

```
D = D + 1  
Loop While D <= N - 1  
If tag <> "Not Prime" Then  
    MsgBox "It is a prime number"  
End If  
End Select  
End Sub
```



**Figure E.g.4**

## Example 5 Selective Summation

This is an Excel VBA 365 program that can perform selective summation according to a set of conditions. For example, you might just want to sum up those figures that have achieved sales targets and vice versa. This VBA program can sum up marks that are below 50 as well as those marks which are above 50.

In this program, rng is declared as range and we can set it to include a certain range of cells, here the range is from A1 to A10.

Then we used the For...Next loop to scan through the selected range

`rng.Cells(i).Value` read the value in `cells(i)` and then passed it to the variable mark.

To do selective addition, we used the statement `Select Case....End Select`

Finally, the results are shown in a message box

Here is the code:

```
Private Sub CommandButton1_Click ()  
Dim rng As Range, i As Integer  
Dim mark, sumFail, sumPass As Single  
sumFail = 0  
sumPass = 0  
Set rng = Range("A1:A10")  
For i = 1 To 10  
mark = rng.Cells(i).Value  
Select Case mark  
Case Is < 50  
sumFail = sumFail + mark  
Case Is >= 50  
sumPass = sumPass + mark  
End Select
```

```
Next i  
MsgBox "The sum of Failed marks is" & Str(sumFail) & vbCrLf & "The  
sum of Passed marks is" & Str(sumPass)  
End Sub
```

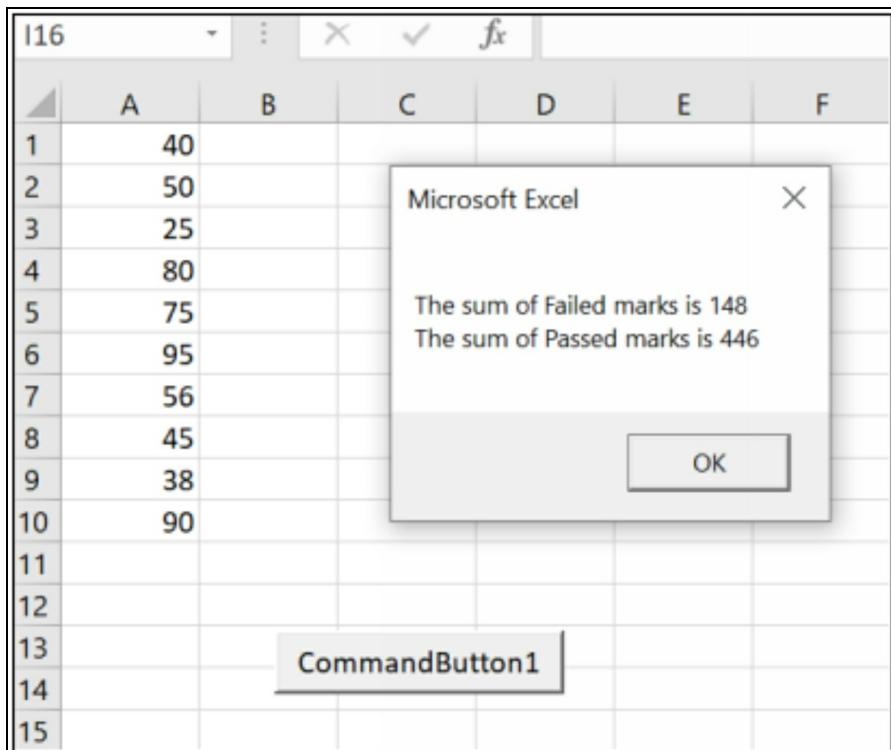


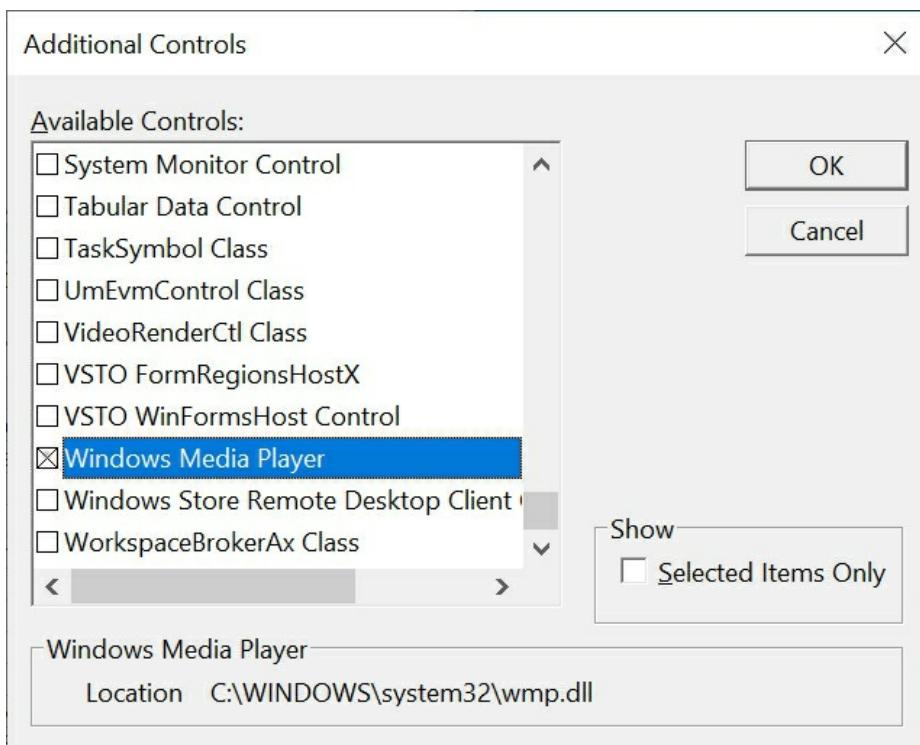
Figure E.g.5

## Example 6 Excel VBA 365 Windows Media Player

---

This is an Excel VBA 365 Windows media player that can play all kinds of media files like wav, midi, mp3, mpeg video, avi video and more. When you start the player, you can select media files of different types from different sources such as your hard drives, CD and even the URL of a webpage. After selecting the file, you can play it using the Play button or you can use the buttons of the Windows Media Player

To build this application, you must insert the Windows Media Player Control. This control is not available in the default VBE, you need to choose Tools from the menu and select Add Controls from the drop down list. In the Add Controls list, select Windows Media Player then click OK, as shown in Figure E.g.6.1.



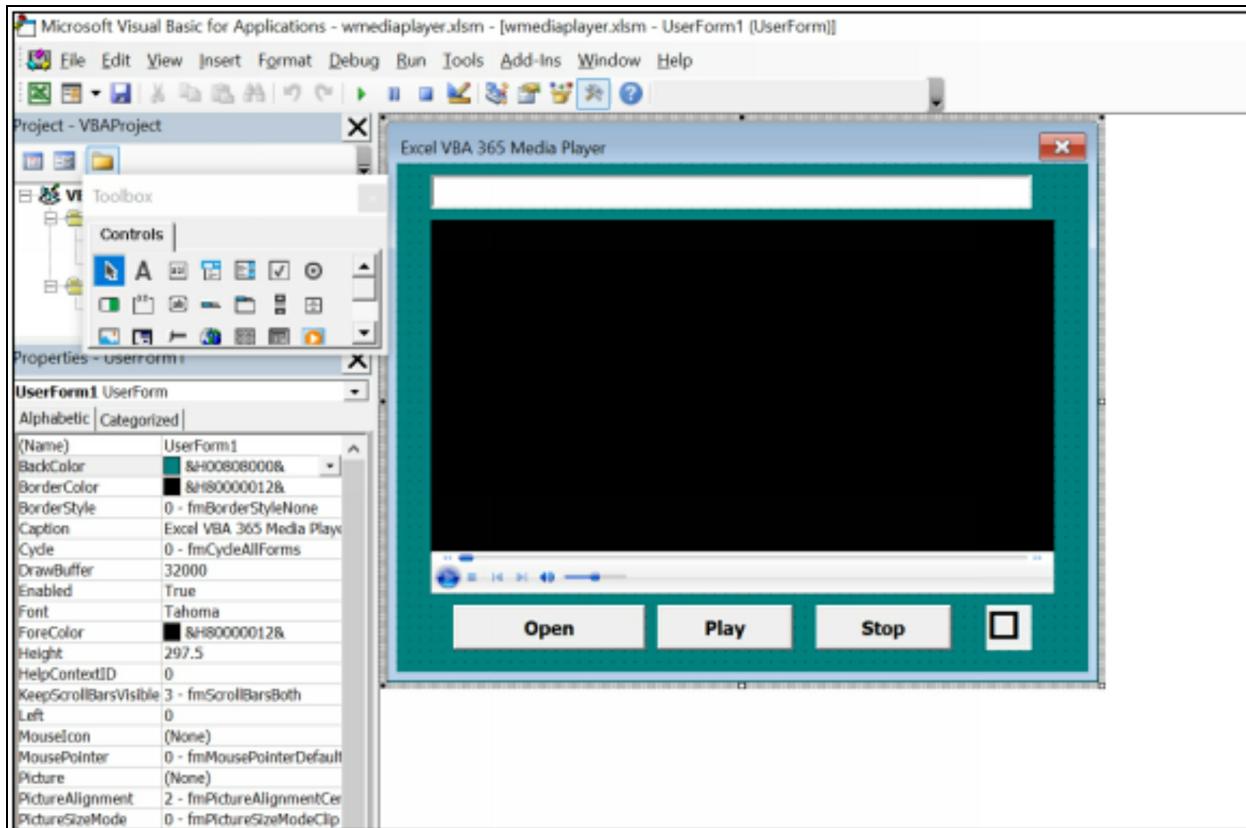
**Figure E.g. 6.1**

When you click OK, you will see that the Windows Media Player control is added to the tool box, as shown in Figure E.g.6.2



**Figure E.g.6.2**

Now you are ready to design the player's interface. First, drag the Windows Media Player control into the form and resize it to a certain size. Set its fullScreen property to false. Insert three command buttons in the form. Change the caption of first button to Open, second one to Play, third one to Stop. Next, insert a label and change its caption to o (using windings font by typing o) which will be used to launch Full Screen. Finally, insert a text box for the purpose to display the URL of the selected media file. The design would look like the one shown in Figure E.g.6.3.



### **Figure E.g.6.3**

In the General section of the VBE code window, declare the variable myfileName as follows:

```
Dim myfileName As Variant
```

Now click on the Open button and key in the following code

```
Private Sub Cmd_Open_Click()
myfileName = Application.GetOpenFilename("Wave Files (*.wav),
*.wav, MP3 Files (*.mp3), *.mp3, MP4 Files (*.mp4), *.mp4, Video Files
(*.avi), *.avi, MPEG Files (*.mpg), *.mpg")
If myfileName <> False Then
End If
End Sub
```

Remember that in Chapter 17, Application.GetOpenFilename is the Excel VBA 365 method to get the file name. We use the filter for several types of media files that include wav, mp3, mp4, avi and mpg.

Before we go on, remember that we should disable Auto Start property of the Windows Media Player. If we allows Auto Start, the Windows Media Player will resize to its default size when you launch the player. To do this click on Custom property and uncheck Auto Start Property (under Play Back Option)in the Windows Media Player property dialog, as shown in Figure E.g.6.4 and Figure E.g. 6.5.

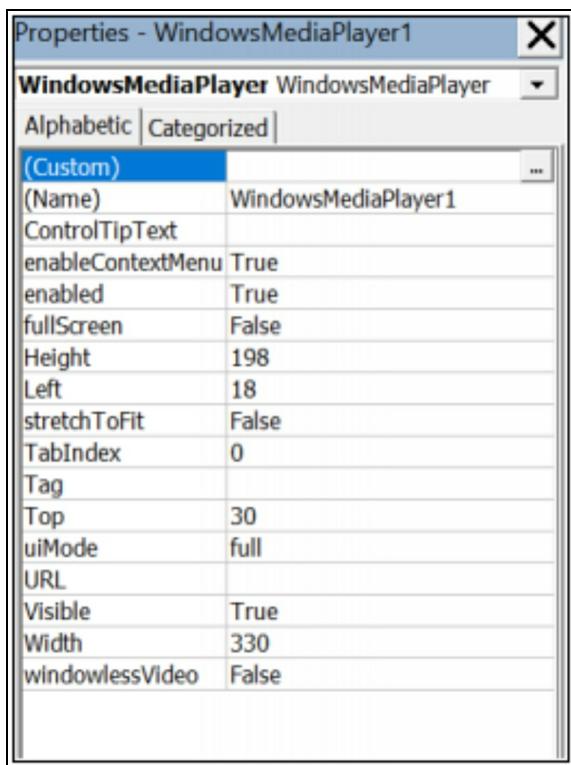
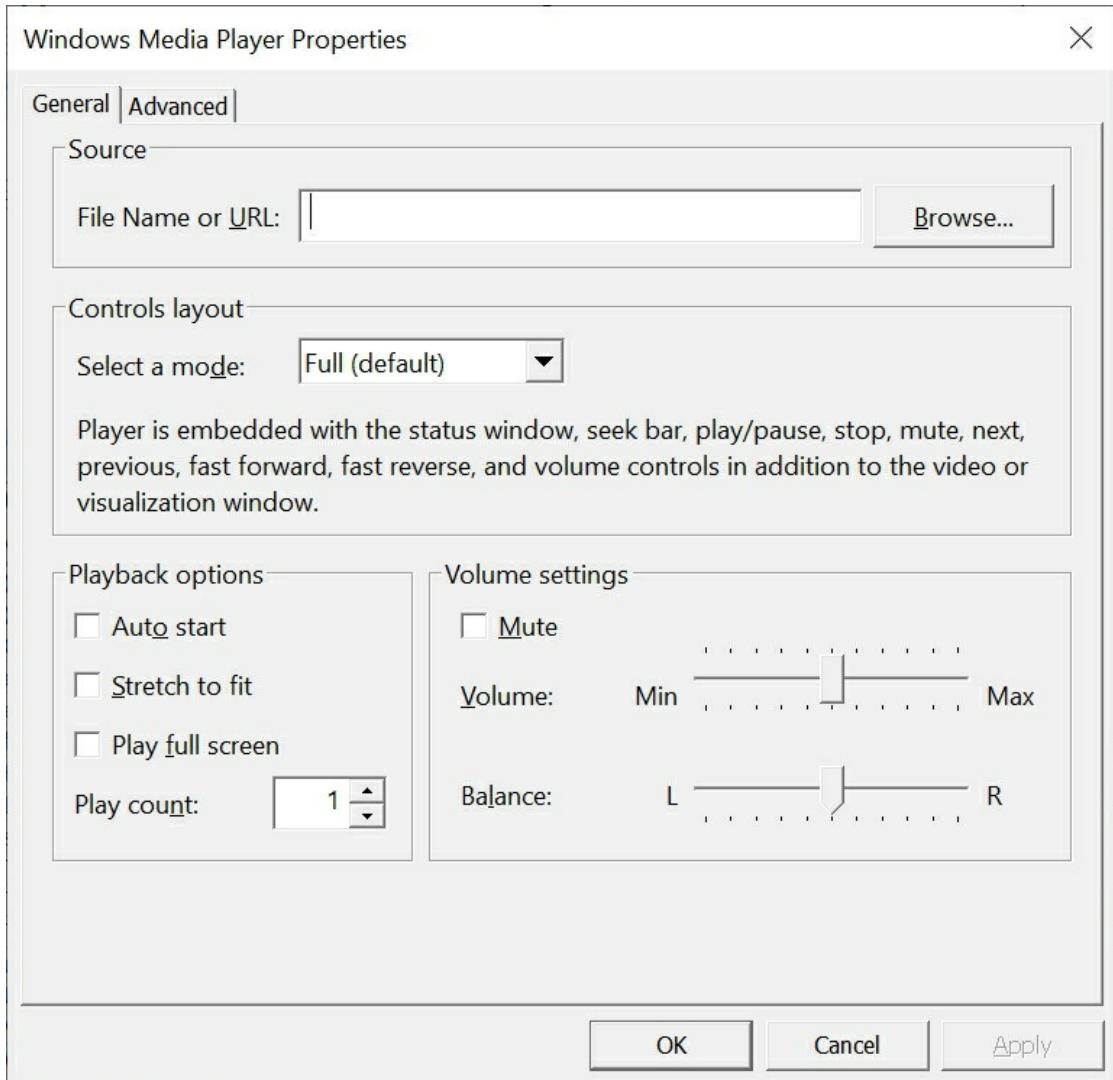


Figure E.g.6.4 Custom Property



**Figure E.g.6.5**

Next, click on the Play button as enter the following code:

```
Private Sub Cmd_Play_Click()
WindowsMediaPlayer1.Url = myfileName
WindowsMediaPlayer1.Controls.Play
End Sub
```

For the Stop button, write the following code:

```
Private Sub Cmd_Stop_Click()
WindowsMediaPlayer1.Close
End Sub
```

Finally, for the Label button, enter the following code. This code allows the

Windows Media Player to play at full screen.

```
Private Sub Lbl_FullScreen_Click()
    WindowsMediaPlayer1.fullScreen = True
End Sub
```

The runtime interface is shown in Figure E.g.6.6

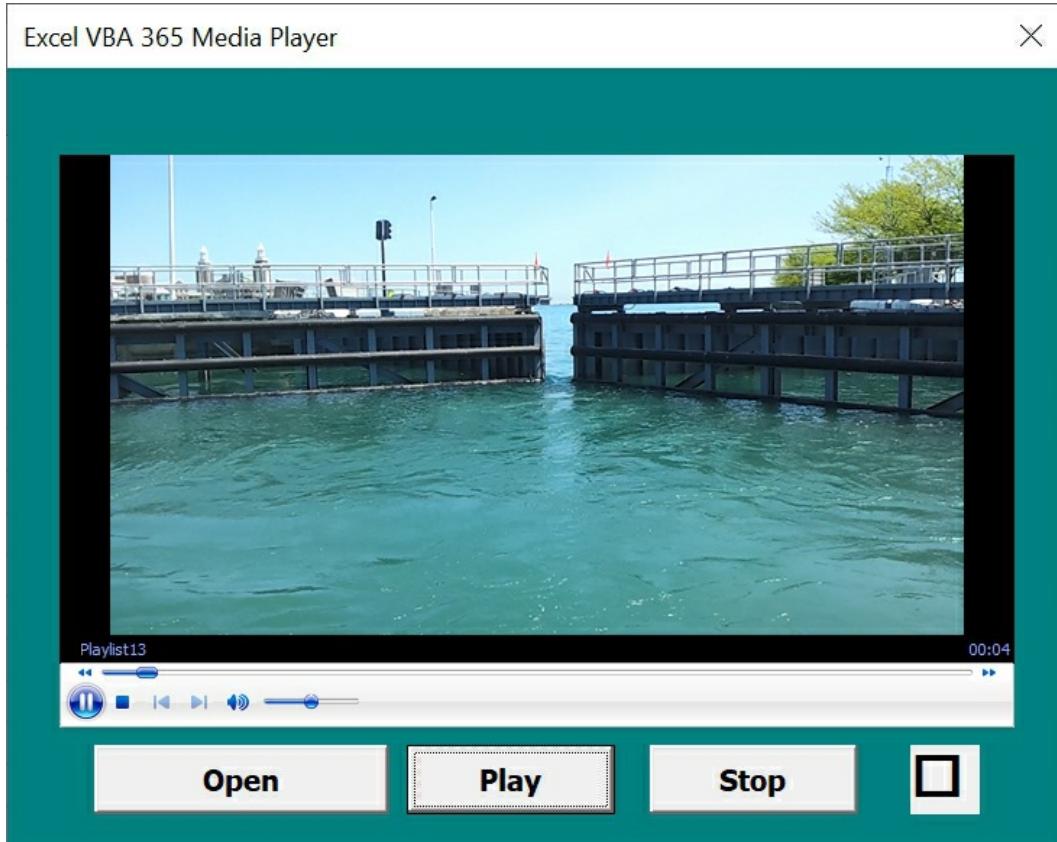


Figure E.g.6.6 The Runtime Interface

## Example 7 Animation

Besides creating Excel VBA 365 code for mathematical and financial calculations, it is also possible to create some fun applications in Excel VBA 365, including games and animation. Although professionals' programmers might not be interested in writing such applications, it is worthwhile trying them out as a hobby and for personal satisfaction.

Animation can be achieved by changing the position of an object continuously using a looping sub procedure. It also involved the use of the DoEvents command. DoEvents is an Excel VBA command that temporarily pauses the execution of the macro to refresh the screen and execute any pending events in Excel.

Two properties that are required to change the positions of the object are the Left and Top properties. The Left property specifies the distance between the left edge of the object in pixel from the left border of the screen and the Top property specifies the distance between the top edge of the object and the top border of the screen. The following code makes the object move from left to right then back to left again repeatedly until the user press the stop button.

```
Private Sub CmsStart_Click()
repeat:
With VBAProject.Sheet1.Image1
.Left = .Left + 1
DoEvents
If .Left > 200 Then .Left = 1
End With
GoTo repeat

End Sub
```

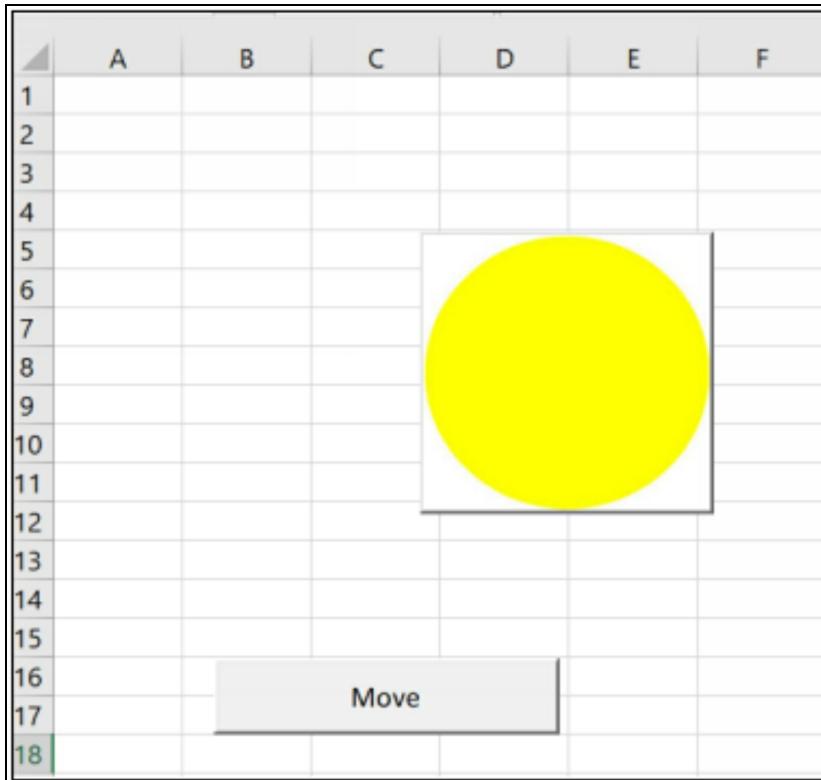
There also a code for the reset button:

```
Private Sub CmdReset_Click()
With VBAProject.Sheet1.Image1
.Left = 0
End With
```

```
End Sub
```

\*The above code reset the position of the image to the far left

The UI is as shown in Figure E.g.7.1



**Figure E.g.7.1**

You can also write the following code to make the motion vertical.

```
Private Sub StartButton_Click()
repeat:With VBAProject.Sheet1.Image1
    .Top= .Top+ 1
    DoEvents
    If .Top> 200 Then .Top = 1
    End WithGoTo repeat
End Sub
```

If you wish to make the object move diagonally, then use the properties Top and Left at the same time, as follows:

```
Private Sub StartButton_Click()
repeat:
    With VBAProject.Sheet1.Image1
        .Top = .Top + 5
    End With
End Sub
```

```
.Left = .Left + 5  
DoEvents  
If .Top > 200 Then .Top = 1  
If .Left > 200 Then .Left = 1  
End With  
GoTo repeat  
End Sub
```

## Example 8 Amortization Calculator

---

Before we delve into the program code, we need to know some basic financial concepts. The term loan amortization means the computation of the amount of equal periodic payments necessary to provide lender with a specific interest return and repay the loan principal over a specified period.

The loan amortization process involves finding the future payments whose present value at the loan interest rate equal the amount of initial principal borrowed. Lenders use a loan amortization schedule to determine these payment amounts and the allocation of each payment to interest and principal.

The formula to calculate periodic payment is  
payment=Initial Principal/PVIFA<sub>n</sub>

where PVIFA<sub>n</sub> is known as present value interest factor for an annuity.

The formula to compute PVIFA<sub>n</sub>

$$PVIFAn = \frac{1}{i} - \frac{1}{i(1+i)^n}$$

where n is the number of payments. Normally you can check up a financial table for the value of PVIFA<sub>n</sub> and then calculate the payments manually. You can also use a financial calculator to compute the values. However, if you already know how to write program in Excel VBA, why not create your very own financial calculator?

To calculate the payments for interest, you can multiply the initial principal with the interest rate, then use periodic payment to minus payment for interest. To calculate the balance at the end of a period, we use the formula

$$\text{End-of\_year principal} = \text{Beginning\_of\_year principal} - \text{periodic payment}$$

We insert two command buttons into the Worksheet, one of them is for the calculation of the periodic payment and the other one is to generate the amortization table.

The Code to Calculate the periodic payment

```
Public Sub Cmd_Calculate_Click()
```

```
P = Cells(4, 3)
Num = Cells(5, 3)
r = Cells(6, 3)
I = r / 100
PVIFA = 1 / I - 1 / (I * (1 + I) ^ Num)
pmt = P / PVIFA
Cells(7, 3) = Round(pmt, 2)
End Sub
The Code to Generate the Amortization Table
Dim Num, n As Integer
Dim I, P, PVIFA, r, pmt, PI, PP As Double
```

```
Private Sub Cmd_Generate_Click()
Dim Num, n As Integer
n = 0
P = Cells(4, 3)
Num = Cells(5, 3)
r = Cells(6, 3)
I = r / 100
PVIFA = 1 / I - 1 / (I * (1 + I) ^ Num)
pmt = P / PVIFA
```

```
Do
n = n + 1
PI = P * I
PP = pmt - PI
P = P - PP
Cells(n + 11, 3) = n
Cells(n + 11, 4) = Round(pmt, 2)
Cells(n + 11, 5) = Round(PI, 2)
Cells(n + 11, 6) = Round(PP, 2)
Cells(n + 11, 7) = Round(P, 2)
If n = Num Then
Exit Do
End If
Loop
```

End Sub

The Output UI is as shown in Figure Eg.8.1

A	B	C	D	E	F	G	H	I	J
<b>Amortization Table</b>									
n	Installment	Interest Payment	Princiapl Payment	Balance					
1	700.98	480	220.98	5779.02					
2	700.98	462.32	238.66	5540.37					
3	700.98	443.23	257.75	5282.62					
4	700.98	422.61	278.37	5004.25					
5	700.98	400.34	300.64	4703.61					
6	700.98	376.29	324.69	4378.93					
7	700.98	350.31	350.66	4028.26					
8	700.98	322.26	378.72	3649.55					
9	700.98	291.96	409.01	3240.53					
10	700.98	259.24	441.73	2798.8					
11	700.98	223.9	477.07	2321.73					
12	700.98	185.74	515.24	1806.49					
13	700.98	144.52	556.46	1250.03					
14	700.98	100	600.98	649.05					
15	700.98	51.92	649.05	0					

Figure E.g.8.1

## Example 9 Boggle

---

Boggle is a type of words puzzle game where the players can form as many words as possible from the characters displayed on a nxn square. Words can be formed in many ways, from left to right, from right to left, top to bottom, bottom to top, diagonal, zigzag manner and more.

In this example, we have designed a 5x5 boggle. This is only a boogle board generator so the players have to write the words on a piece of paper.

Each time we press the shuffle button, a different set of characters will appear. In order to achieve this, we use the chr() function and the Rnd function to randomly generate the characters.

Alphabet A to Z correpond to Chr(65) to chr(90), therefore we need to generate random numbers between 65 to 90. The formula to generate random numbers between two numbers is:

$$m = \text{Int}((\text{.MaxValue} - \text{.MinValue} + 1) * \text{Rnd}) + \text{.MinValue}$$

Therefore, the formula to generate random numbers between 65 and 90 is RndNum=Int(26)+65, which means we can generate random alphabet using chr(m) in a For Next Loop.

To display the alphabets in 25 cells on the worksheet, we use a nexted For Next loop.

### The Code

```
Private Sub cmd_Shuffle_Click()
    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
```

	A	B	C	D	E	F	G	H	I	J
1	BOGGLE									
2	Q	O	E	Y	R					
3	N	K	C	U	L					
4	T	P	V	A	F					
5	B	C	I	D	A					
6	N	R	O	V	C					
7										

Figure E.g.9.1

# Example 10 Calculator

This is a typical calculator created using Excel VBA 365. It consists of the number buttons, the operator buttons and some additional command buttons such as the memory button, the clear button and more. To design the interface, insert 23 command buttons and a label. The label is designated to be the display panel. Next, allocate ten command buttons as the number buttons and change the captions according to their numbers. Besides that, we need to assign four command buttons for the basic operators and one command button for the equal operator. In addition, you need to add the dot command button for decimals. We have also added the M+, M-, MR, C, Sqrt and the MC command buttons.

The design UI is as shown below:

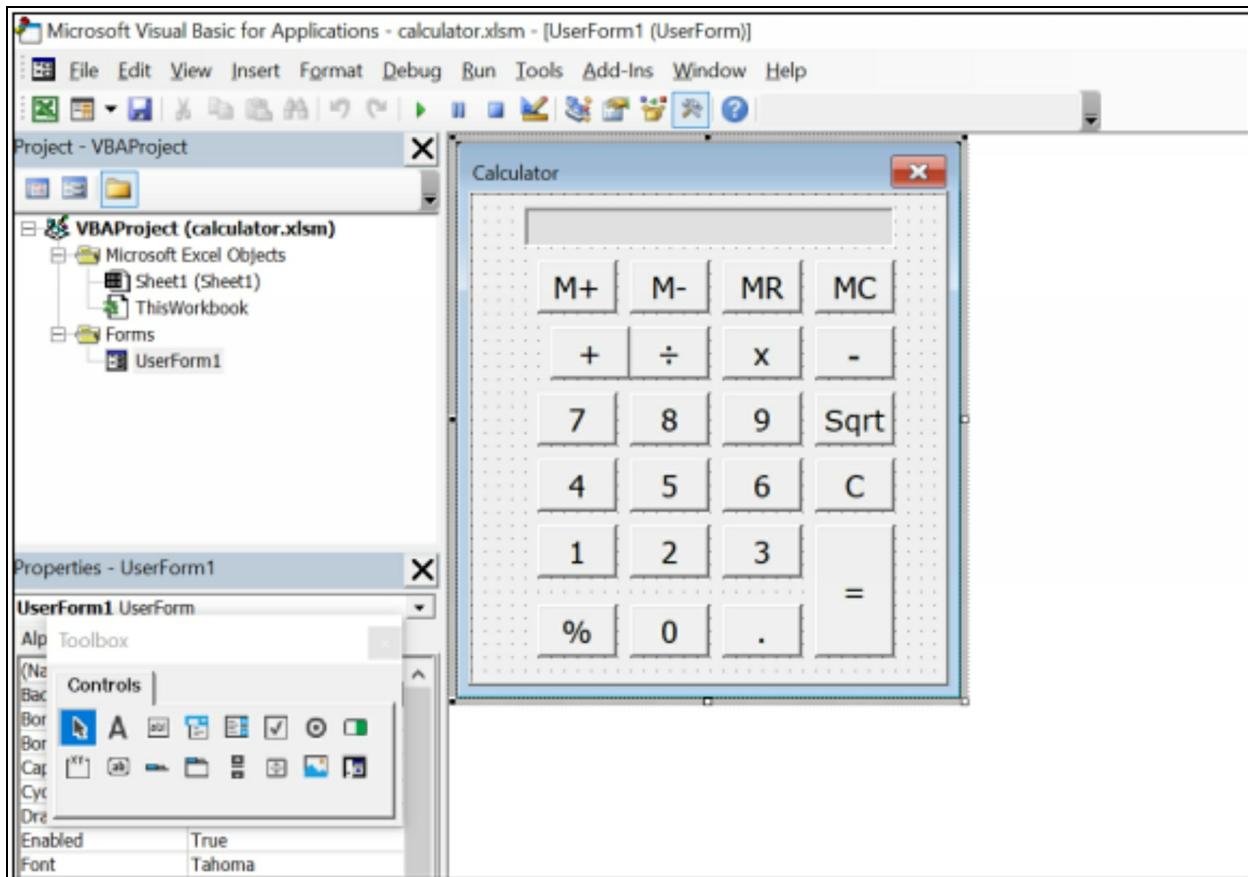


Figure E.g.10.1

Table E.g.10.1 lists the controls and their corresponding properties:

**Table E.g.10.1**

CONTROL	PROPERTIES
Cmd_Zero	<ul style="list-style-type: none"><li>• Caption: 0</li><li>• BackStyle:fmBackStyleOpaque</li><li>• BackColor:&amp;H8000000F&amp;</li><li>• Font:Tahoma</li><li>• Font Size: 14 Regular</li></ul>
Cmd_one	<ul style="list-style-type: none"><li>• Caption: 1</li><li>• BackStyle:fmBackStyleOpaque</li><li>• BackColor:&amp;H8000000F&amp;</li><li>• Font:Tahoma</li><li>• Font Size: 14 Regular</li></ul>
Cmd_two	<ul style="list-style-type: none"><li>• Caption: 2</li><li>• BackStyle:fmBackStyleOpaque</li><li>• BackColor:&amp;H8000000F&amp;</li><li>• Font:Tahoma</li><li>• Font Size: 14 Regular</li></ul>
Cmd_three	<ul style="list-style-type: none"><li>• Caption: 3</li><li>• BackStyle:fmBackStyleOpaque</li><li>• BackColor:&amp;H8000000F&amp;</li><li>• Font:Tahoma</li><li>• Font Size: 14 Regular</li></ul>
Cmd_Four	<ul style="list-style-type: none"><li>• Caption: 4</li><li>• BackStyle:fmBackStyleOpaque</li><li>• BackColor:&amp;H8000000F&amp;</li><li>• Font:Tahoma</li><li>• Font Size: 14 Regular</li></ul>
Cmd_Five	<ul style="list-style-type: none"><li>• Caption: 5</li></ul>

	<ul style="list-style-type: none"> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Six	<ul style="list-style-type: none"> <li>• Caption: 6</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Seven	<ul style="list-style-type: none"> <li>• Caption: 7</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Eight	<ul style="list-style-type: none"> <li>• Caption: 8</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Nine	<ul style="list-style-type: none"> <li>• Caption: 9</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Dot	<ul style="list-style-type: none"> <li>• Caption: .</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Percent	<ul style="list-style-type: none"> <li>• Caption: %</li> <li>• BackStyle:fmBackStyleOpaque</li> </ul>

	<ul style="list-style-type: none"> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Plus	<ul style="list-style-type: none"> <li>• Caption: +</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Minus	<ul style="list-style-type: none"> <li>• Caption: -</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Multiply	<ul style="list-style-type: none"> <li>• Caption: x</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Divide	<ul style="list-style-type: none"> <li>• Caption: ÷</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Equal	<ul style="list-style-type: none"> <li>• Caption: =</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Sqrt	<ul style="list-style-type: none"> <li>• Caption: Sqrt</li> <li>• BackStyle:fmBackStyleOpaque</li> </ul>

	<ul style="list-style-type: none"> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_C	<ul style="list-style-type: none"> <li>• Caption: C</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_MC	<ul style="list-style-type: none"> <li>• Caption: MC</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Sqrt	<ul style="list-style-type: none"> <li>• Caption: Sqrt</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_Plus	<ul style="list-style-type: none"> <li>• Caption: M+</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_MSubtract	<ul style="list-style-type: none"> <li>• Caption: M-</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_MR	<ul style="list-style-type: none"> <li>• Caption: MR</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> </ul>

	<ul style="list-style-type: none"> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Cmd_MC	<ul style="list-style-type: none"> <li>• Caption: MC</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>
Lbl_Panel	<ul style="list-style-type: none"> <li>• SpecialEffect:fmSpecialEffectSunken</li> <li>• BackStyle:fmBackStyleOpaque</li> <li>• BackColor:&amp;H8000000F&amp;</li> <li>• Font:Tahoma</li> <li>• Font Size: 14 Regular</li> </ul>

The code involves declaring several variables, as shown below:

```
Dim a, b, c, d, e, f, sqrt, memoplus, memominus As Double
```

```
Dim key As Integer
```

```
Dim newNum As Boolean
```

```
Dim display As Single
```

The variable newNum is to control the display on the display panel. The following syntax instruct the label panel to clear the display panel if we enter a new number.

```
If newNum = True Then
```

```
    Lbl_Panel.Caption = ""
```

The code for every number button (using command button 1 as an example is as follows:

```
Private Sub Cmd_One_Click()
```

```
If newNum = True Then
```

```
    Lbl_Panel.Caption = ""
```

```
End If
```

```
display = Val(Lbl_Panel.Caption + Str(1))
```

```
Lbl_Panel.Caption = Str(display)
```

```
newNum = False
```

```
End Sub
```

We create a sub procedure for all the operator buttons CheckValue:

```
Static Sub CheckValue()
```

```
displayValue = Val(Lbl_Panel.Caption)
```

```
If key = 1 Then
```

```
a = displayValue
```

```
key = 1
```

```
ElseIf key = 2 Then
```

```
b = displayValue
```

```
key = 2
```

```
ElseIf key = 3 Then
```

```
c = displayValue
```

```
key = 3
```

```
ElseIf key = 4 Then
```

```
d = displayValue
```

```
key = 4
```

```
ElseIf key = 5 Then
```

```
e = displayValue
```

```
key = 5
```

```
ElseIf key = 6 Then
```

```
memoplus = memoplus + displayValue
```

```
key = 6
```

```
ElseIf key = 7 Then
```

```
memominus = Abs(memominus) - displayValue
```

```
key = 7
```

```
ElseIf key = 8 Then
```

```
sqrt = displayValue
```

```
key = 8
```

```
End If
```

```
newNum = True
```

```
End Sub
```

The code for the equal button is as follows:

```
Private Sub Cmd_Equal_Click()
If key = 1 Then
f = Val(Lbl_Panel) + a
ElseIf key = 2 Then
f = b - Val(Lbl_Panel)
ElseIf key = 3 Then
f = c * Val(Lbl_Panel)
ElseIf key = 4 Then
f = d / Val(Lbl_Panel)
ElseIf key = 5 Then
f = (e * Val(Lbl_Panel)) / 100
End If
```

```
Lbl_Panel.Caption = Str(Round(f, 4))
newNum = True
End Sub
```

The code for the M+ button is as follows:

```
Private Sub Cmd_MPlus_Click()
key = 6
CheckValue
End Sub
```

The code for the M- button is as follows:

```
Private Sub Cmd_MSubtract_Click()
key = 7
CheckValue
End Sub
```

The code for the MR button is as follows:

```
Private Sub Cmd_MR_Click()
If key = 6 Then
f = memoplus
ElseIf key = 7 Then
f = memominus
End If
```

```
Lbl_Panel.Caption = Str(Round(f, 4))
newNum = True
End Sub
```

The code for the C button is as follows:

```
Private Sub Cmd_C_Click()
Lbl_Panel.Caption = "0"
newNum = True
End Sub
```

The code for the MC button is as follows:

```
Private Sub Cmd_MC_Click()
Lbl_Panel.Caption = "0"
displayValue = 0
memoplus = 0
memominus = 0
newNum = True
End Sub
```

The code for the % button is as follows:

```
Private Sub Cmd_Percent_Click()
key = 5
CheckValue
End Sub
```

The code for the Sqr button is as follows:

```
f = Sqr(Val(Lbl_Panel.Caption))

Lbl_Panel.Caption = Str(Round(f, 4))
newNum = True
```

The Runtime UI

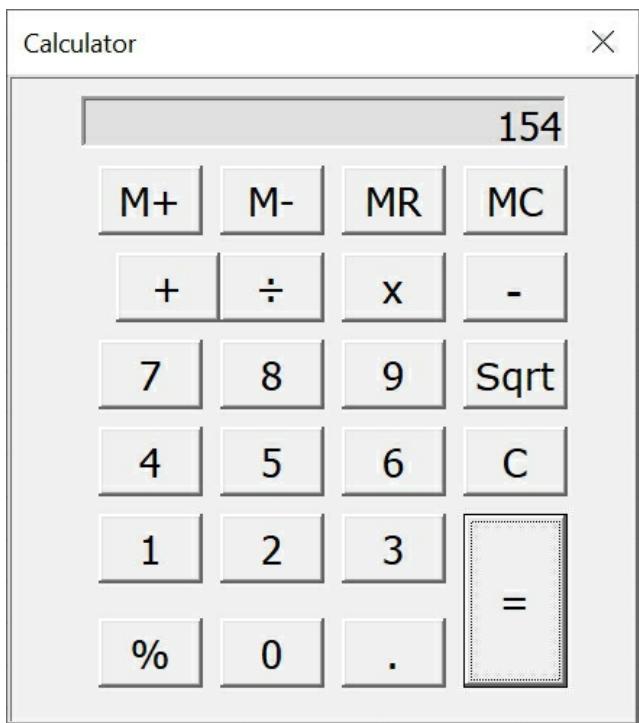


Figure E.g.10.2

## Example 11 Scientific Calculator

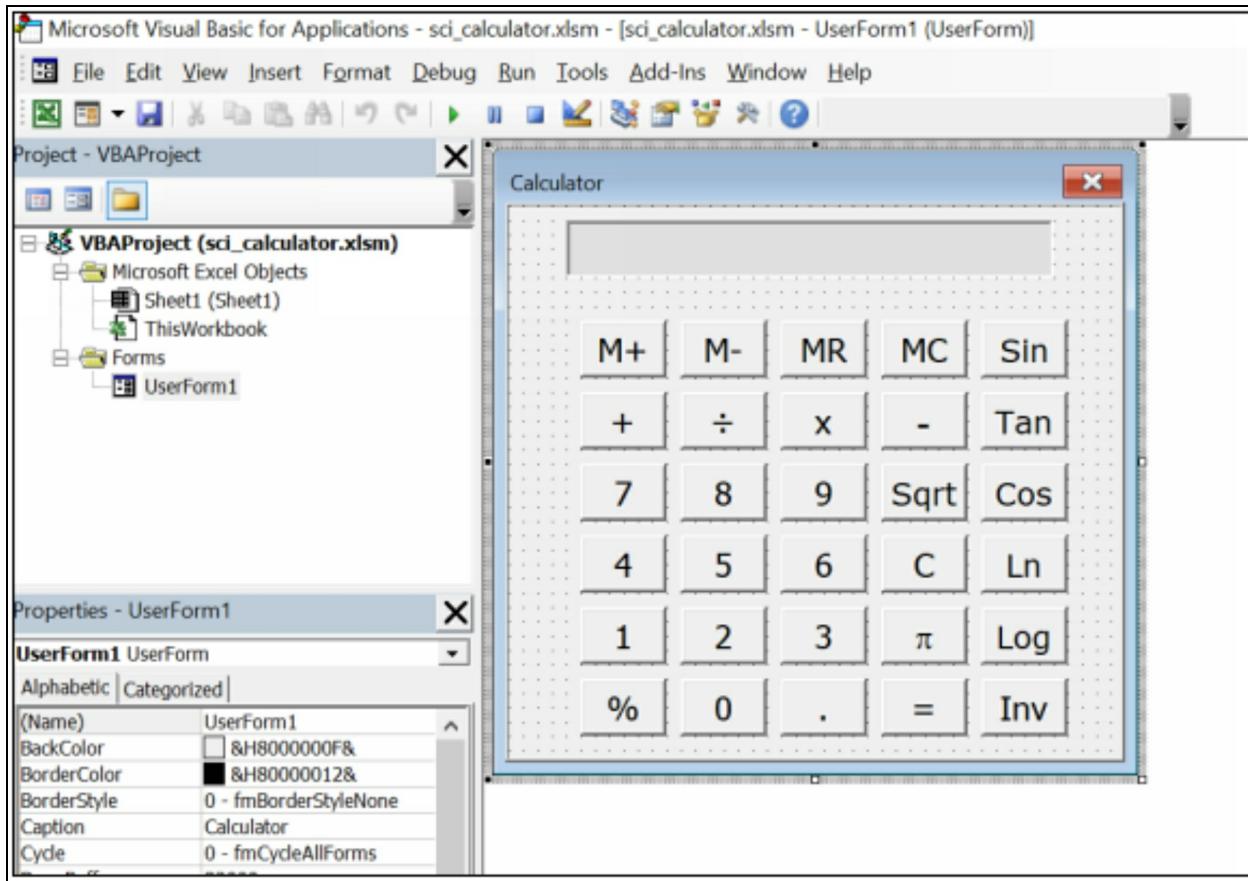
---

In this example, we will enhance the basic features of the calculator and upgrade it to a scientific calculator. We use back the same design as the basic calculator, and add eddital command buttons.

The additonal command buttons cater for the following mathematical functions:

- Sine
- Cosine
- Tangent
- Log
- Ln
- Arcsine
- Arccosine
- Arctangent
- $\pi$

The design UI is as shown in Figure E.g.11.1



**Figure E.g.11.1**

The Code for the Sine function

```
Private Sub Cmd_Sin_Click()
```

```
f = Val(Lbl_Panel.Caption)
If inv = False Then
    sinx = Round(Sin(f * 4 * Atn(1) / 180), 4)
    Lbl_Panel.Caption = sinx
Exit Sub
Else
```

```
On Error GoTo error_handler
arcsinx = Round((WorksheetFunction.Asin(f) * 180) / (4 * Atn(1)))
Lbl_Panel.Caption = arcsinx
inv = False
Exit Sub
```

```

End If

error_handler:
MsgBox ("Please enter a number less than or equal to 1 or more than or equal
to -1")

newNum = True
End Sub

```

Notice that we need to add an error\_handler to handle the error when the user entered a value outside the range of 1 and -1 for arcsine. Besides that, we need to use the syntax

`WorksheetFunction.Asin(x)`

instead of just Asin. For the value of  $\pi$ , we use the formula  $4 * \text{Atn}(1)$  because  $\tan(\pi/4)=1$ .

The Code for the Cosine function

```

Private Sub Cmd_Cos_Click()
f = Val(Lbl_Panel.Caption)
If inv = False Then

```

`cosx = Round(Cos(f * 4 * Atn(1) / 180), 4)`

`Lbl_Panel.Caption = cosx`

`Exit Sub`

`Else`

`On Error GoTo error_handler`

`arccosx = Round((WorksheetFunction.Acos(f) * 180) / (4 * Atn(1)))`

`Lbl_Panel.Caption = arccosx`

`inv = False`

`Exit Sub`

`End If`

`error_handler:`

```

MsgBox ("Please enter a number less than or equal to 1 or more than or equal
to -1")

```

```
newNum = True  
End Sub
```

The Code for the Tan function

```
Private Sub Cmd_Tan_Click()  
Lbl_Panel.Caption = tanx  
  
If inv = False Then  
f = Val(Lbl_Panel.Caption)  
tanx = Round(Tan(f * 4 * Atn(1) / 180), 4)  
Else  
arctanx = Round(Atn(f) * 180) / (4 * Atn(1))  
  
Lbl_Panel.Caption = arctanx  
End If
```

```
newNum = True  
inv = False  
End Sub
```

The Code for the natural Logarithm function

```
Private Sub Cmd_Ln_Click()  
f = Val(Lbl_Panel.Caption)  
Lbl_Panel.Caption = Str(Log(f))  
newNum = True  
inv = True  
End Sub
```

The Code for the Log base 10 function

```
Private Sub Cmd_Log_Click()  
f = Val(Lbl_Panel.Caption)  
Lbl_Panel.Caption = Str(Log(f) / Log(10))  
newNum = True  
End Sub
```

We declare a variable inv as a Boolean to invert between Sine and arcsine, cosine and arccosine as well as tan and arctangent.

The runtime UI is as shown in Figure E.g.11.2

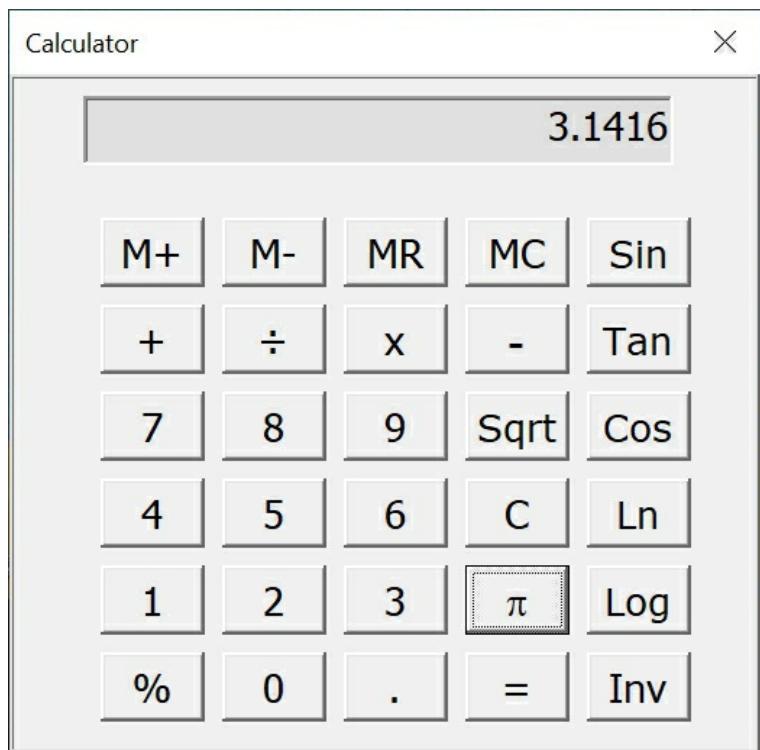


Figure E.g.11.2

## Example 12 Dice

A dice is often required as a tool to play board games. It can also be incorporated into VBA games that require a dice. VBA games that you can create in Excel VBA are step and ladder game, monopoly and more. This example demonstrates how to create an animated dice. When the user click the Spin button, it will start to animate and stop after a certain interval.

To design the interface, insert a UserForm in the VBE. Next, insert a command button and label it as SPIN. Finally, insert seven dot images, as shown in the figure below. Excel VBA will label the images as Image1 to Image7.

The design UI is as shown in Figure E.g.12.1

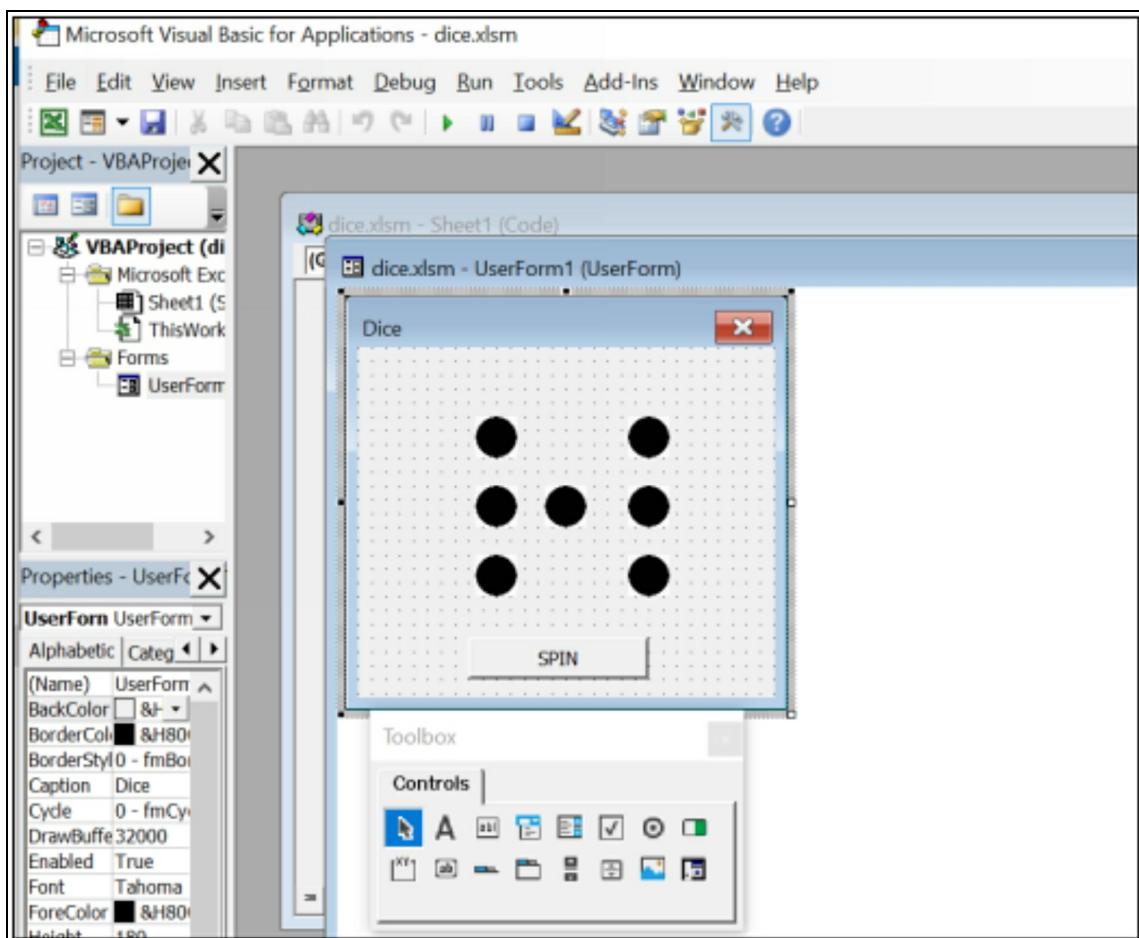


Figure E.g.12.1

The Code

```
Private Sub CommandButton1_Click()
```

```
    x = 0
```

```
    Do
```

```
        n = Int(1 + Rnd * 6)
```

```
        If n = 1 Then
```

```
            Image4.Visible = True
```

```
            Image1.Visible = False
```

```
            Image2.Visible = False
```

```
            Image3.Visible = False
```

```
            Image5.Visible = False
```

```
            Image6.Visible = False
```

```
            Image7.Visible = False
```

```
        End If
```

```
        If n = 2 Then
```

```
            Image1.Visible = True
```

```
            Image7.Visible = True
```

```
            Image2.Visible = False
```

```
            Image3.Visible = False
```

```
            Image5.Visible = False
```

```
            Image6.Visible = False
```

```
            Image4.Visible = False
```

```
        End If
```

```
        If n = 3 Then
```

```
            Image1.Visible = True
```

```
            Image4.Visible = True
```

```
            Image7.Visible = True
```

```
            Image2.Visible = False
```

```
            Image3.Visible = False
```

```
            Image5.Visible = False
```

```
            Image6.Visible = False
```

```
        End If
```

```
        If n = 4 Then
```

```
Image1.Visible = True  
Image2.Visible = True  
Image6.Visible = True  
Image7.Visible = True  
Image4.Visible = False  
Image3.Visible = False  
Image5.Visible = False  
End If
```

```
If n = 5 Then  
Image1.Visible = True  
Image4.Visible = True  
Image2.Visible = True  
Image6.Visible = True  
Image7.Visible = True  
Image3.Visible = False  
Image5.Visible = False
```

```
End If
```

```
If n = 6 Then  
Image1.Visible = True  
Image2.Visible = True  
Image5.Visible = True  
Image3.Visible = True  
Image6.Visible = True  
Image7.Visible = True  
Image4.Visible = False  
End If
```

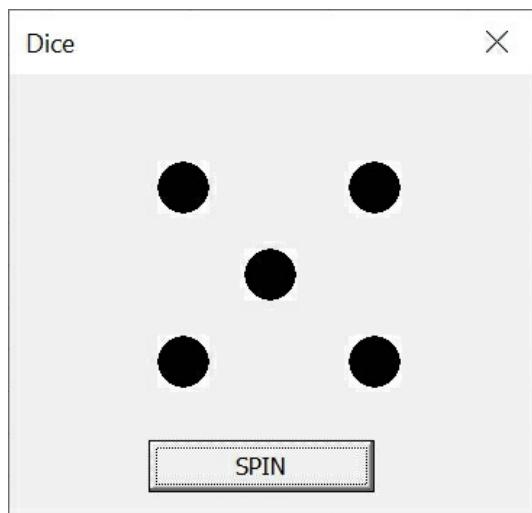
```
x = x + 2
```

```
DoEvents
```

```
Loop Until x > 10000
```

```
End Sub
```

The runtime UI is as shown in Figure E.g.12.2



**Figure E.g.12.2**

## Example 13 Geometric Progression

---

This is an Excel VBA program that generates a geometric progression (Also called geometric series or geometric sequence) and displays the results in a range of cells.

Geometric progression is a sequence of numbers where each subsequent number is found by multiplying the previous number by a fixed number. The fixed number is called the common ratio. The common ratio can be negative, an integer, a fraction and any number but it must not be a zero.

The formula to find the nth term of the geometric progression is  $ar^{n-1}$ , where a is the first number and r is the common ratio.

In Excel VBA, we employ the Do.... Loop Until statement to generate the geometric progression. In this program, we insert two command buttons, one is to generate the geometric sequence and the other one is to clear contents. We need to clear contents because we intend to display the GP in a table with border on the worksheet.

The formula to add border with a certain color is :

```
Set Rng = Range(start_cell, end_cell)
```

```
With Rng.Borders
```

```
    .LineStyle = xlContinuous
```

```
    .Color = vbBlue
```

```
    .Weight = xlThick 'Think border. Use xlThin if you want a thin  
border
```

```
End With
```

The formula to clear contents is :

```
Range(Cells(start_cell, end_cell).ClearContents
```

```
Range(Cells(start_cell, end_cell).ClearFormats
```

The Code

```
Private Sub Cmd_Generate_Click()
```

```
Dim n, num As Integer
```

```
Dim a, x, r As Single
```

```
a = Cells(3, 3)
r = Cells(4, 3)
num = Cells(5, 3)
```

```
n = 1
Do
x = a * r ^ (n - 1)
Cells(n + 5, 2) = "Term " & n
```

```
Cells(n + 5, 3) = x
n = n + 1
```

```
Loop Until n = num + 1
```

```
Set Rng = Range(Cells(6, 2), Cells(n + 4, 3))
```

```
With Rng.Borders
    .LineStyle = xlContinuous
    .Color = vbBlue
    .Weight = xlThick
End With
End Sub
```

```
Private Sub Cmd_Clear_Click()
Range(Cells(6, 2), Cells(1000, 3)).ClearContents
Range(Cells(6, 2), Cells(1000, 3)).ClearFormats
End Sub
```

The Runtime UI is as shown in Figure E.g.14.1

A	B	C	D	E	F
<b>Geometric Progression Generator</b>					
First Term	2				
Common Ratio	3				
Number of Terms	8				
Term 1	2				
Term 2	6				
Term 3	18			Generate	
Term 4	54				
Term 5	162				
Term 6	486			Clear Contents	
Term 7	1458				
Term 8	4374				

Figure E.g.13.1

# Example 14 Password Cracker

---

This is a password cracker application created using Excel VBA 365(can use older version). This app can generate possible passwords and check each of them with the actual password. If the generated password found to be equal to the actual password, login will be successful, and a secret word will be revealed.

In this application, we use a Do loop and the DoEvents command to iterate through the process of generating the passwords.

The password is a combination of three alphanumeric characters(you can use more characters), so we need to generate random alphanumeric characters using the chr() functions. The ASCII code for all alphanumeric characters are from 33 to 126. Therefore, we need to generate random alphanumeric characters using the formula:

`Int(Rnd * 93) + 33`

## The Code

```
Private Sub Cmd_Generate_Click()
Dim crackpass, password, secretword As String
Dim num1, num2, num3 As Integer

password = "#@9"
secretword = "Happy Birthday"
Do
    num1 = Int(Rnd * 93) + 33
    num2 = Int(Rnd * 93) + 33
    num3 = Int(Rnd * 93) + 33
    crackpass = Chr(num1) & Chr(num2) & Chr(num3)
    TxtPW.Text = crackpass

    If crackpass = password Then
        Exit Do
    End If
End Sub
```

```
DoEvents
```

```
Loop
```

```
Lbl_Msg.Caption = "Password Cracked!Login Successful!"
```

```
Cells(2, 2) = secretword
```

```
End Sub
```

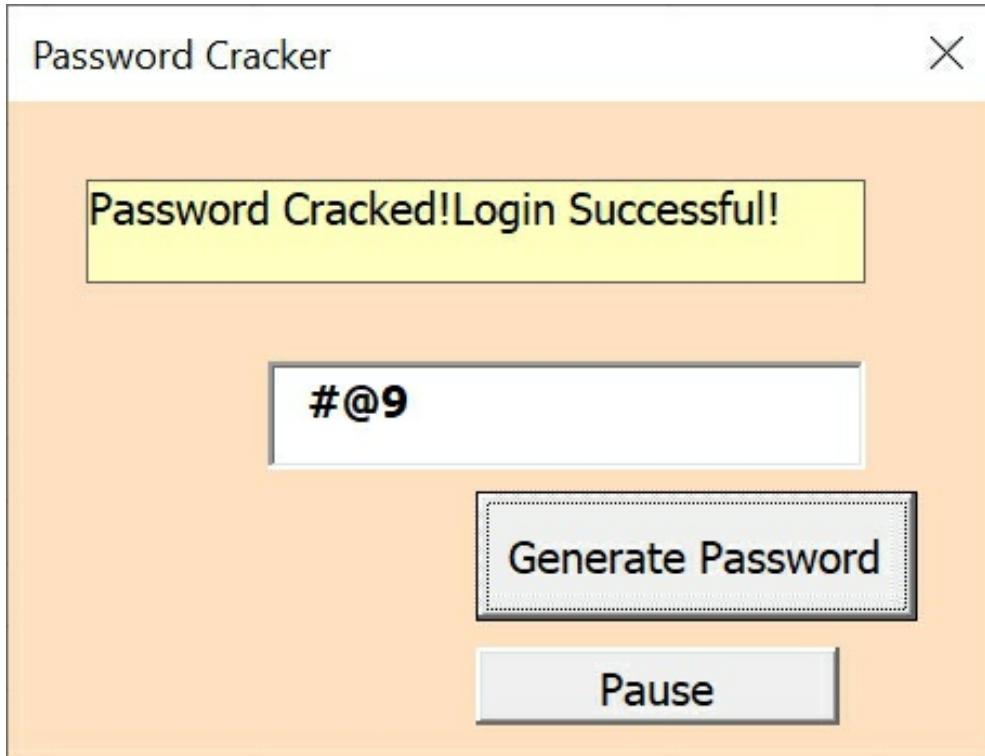


Figure E.g.14.1 The Password

	A	B	C
1			
2	Secret Word	Happy Birthday	
3			
4			
5			
6			
7	Password Cracker		X
8			
9		Password Cracked! Login Successful!	
10			
11			
12		#@9	
13			
14			
15		<input type="button" value="Generate Password"/>	
16			
17		<input type="button" value="Pause"/>	
18			
19			
20			

Figure E.g.14.2 The Secret Word

# Example 15 Digital Slot Machine

---

This is a digital slot machine created using Excel VBA 365. In this example, start MS Excel and click the Developer tab. Click View Code to enter the Visual Basic editor. Insert a UserForm and design the UI for the slot machine. Insert three labels for displaying the digits, one label to display the "Slot Machine " name, two command buttons which is used for spinning and ending the program.

We use the Rnd function to generate random numbers from 1 to 9 using the formula Int(Rnd \* 9)+1 that will be shown on the three labels.

We use the Do..Loop procedure to generate the random numbers and pause it momentarily between each loop using the DoEvents command. This will create the animation effect, simulating a spinning slotmachine. DoEvents is an Excel VBA command that temporarily pauses the execution of the macro to refresh the screen and execute any pending events in Excel.

We use the If...Then...ElseIf statements to check for JackPot and Mini Jackpots. If all numbers are 7 then it will be the Jackpot, if any two same numbers appear then it will be a mini jackpot.

## The Code

```
Private Sub CommandButton1_Click()
```

```
Dim x As Integer
```

```
x = 0
```

```
Do
```

```
x = x + 2
```

```
Label1.Caption = Int(Rnd * 9) + 1
```

```
Label2.Caption = Int(Rnd * 9) + 1
```

```
Label3.Caption = Int(Rnd * 9) + 1
```

```
DoEvents
```

```
Loop Until x = 10000
```

```
If (Label1.Caption = 7) And (Label2.Caption = 7) And (Label3.Caption = 7)
```

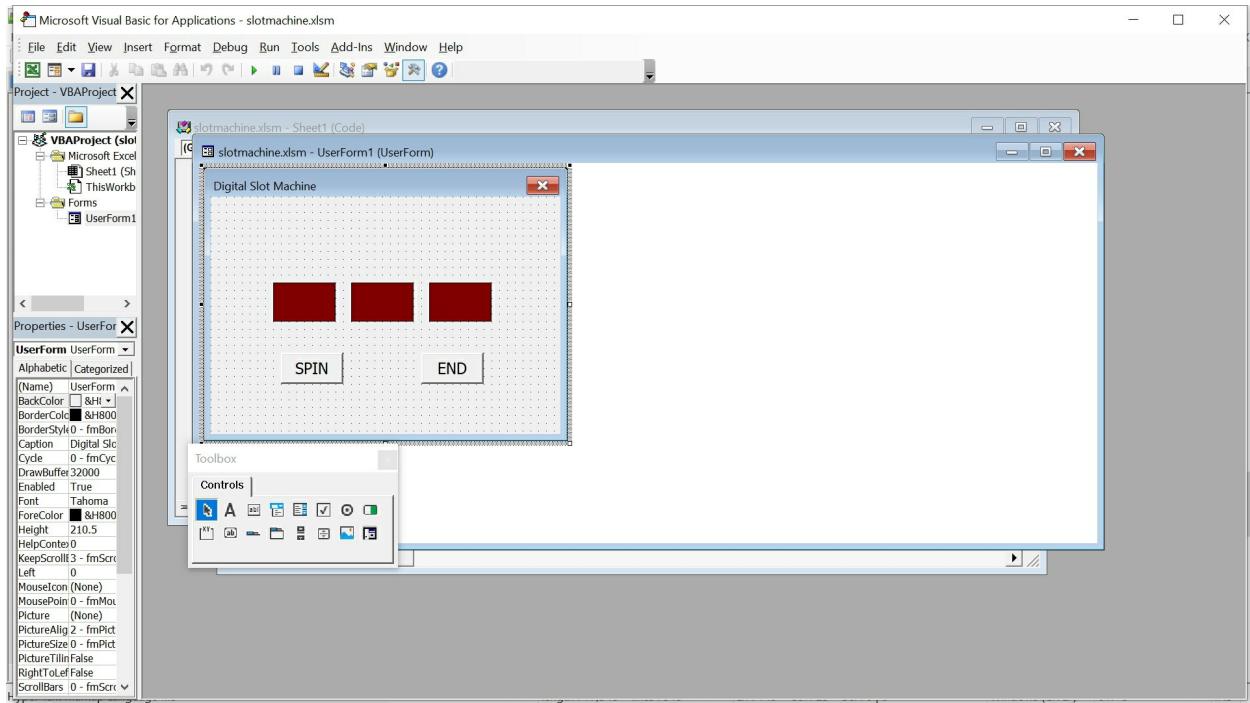
```
Then
```

```

    MsgBox ("Your strike the Jackpot")
    ElseIf (Label1.Caption = Label2.Caption) Or (Label1.Caption =
Label3.Caption) Or (Label2.Caption = Label3.Caption) Then
        MsgBox ("Your strike the Mini Jackpot")
    End If

End Sub

```



**Figure E.g.15.1 The Design UI**

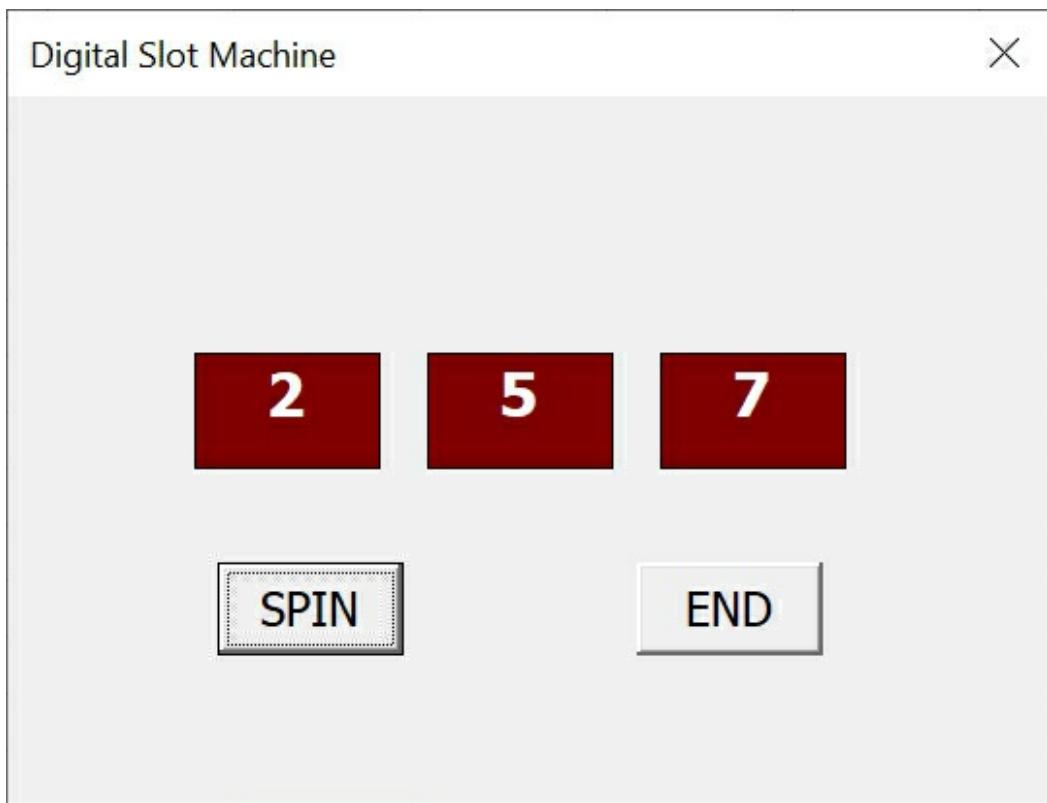


Figure E.g.15.2 The Runtime UI

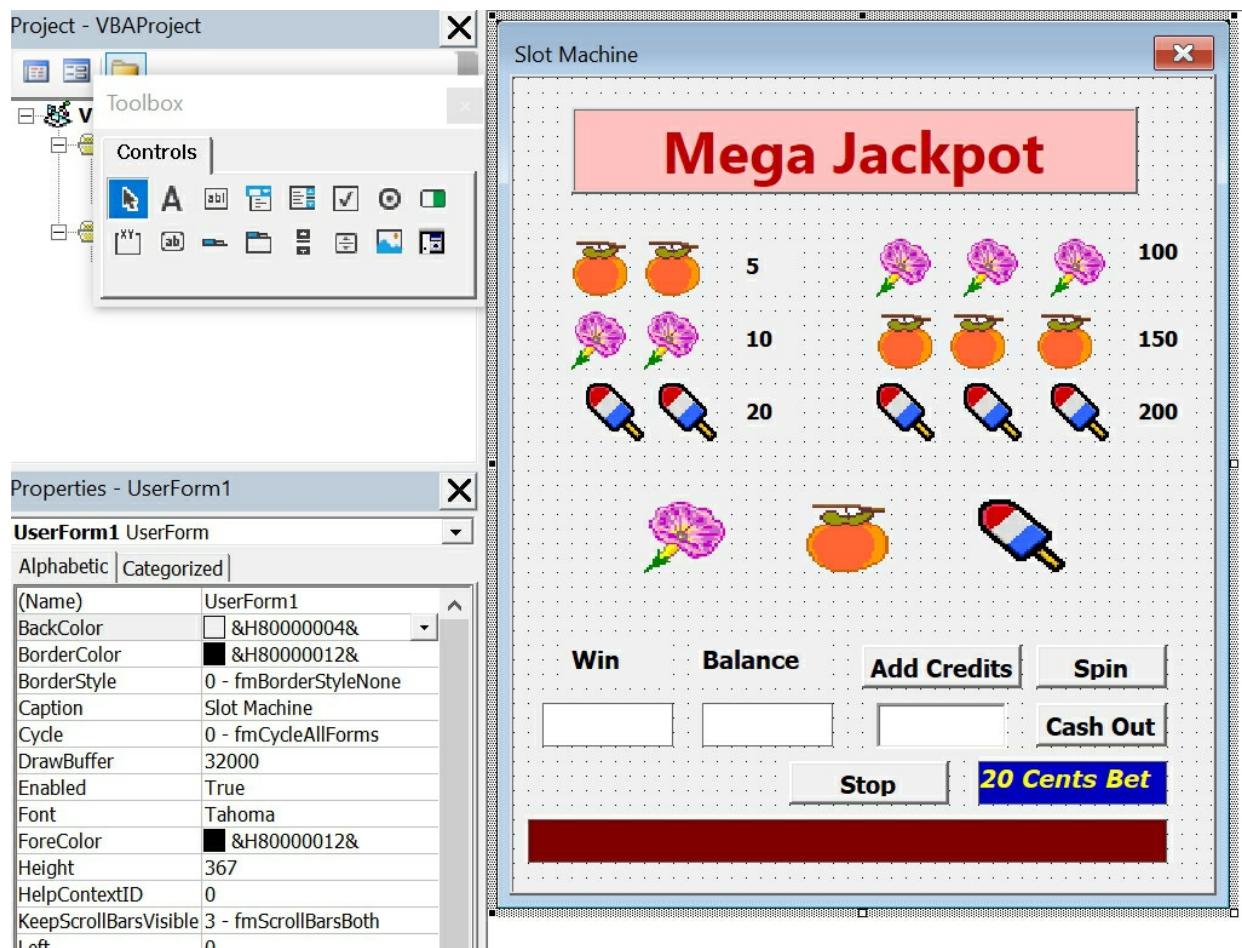


Figure E.g.15.3 The Mini Jackpot Dialog

# Example 16 Professional Slot Machine

We shall create slot machine that resembles the real slot machines in the casino, using Excel VBA 365(Older versions should be able to work too).

To create the project, start an Excel 365 workbook and save the file as slotmachine.xlsm or some other file name you like. Click the Developer tab and enter the Visual Basic Editor. In the VBE, insert a UserForm and design the user interface, as shown in Figure E.g.17.1



**Figure E.g.16.1 The Design UI**

First, insert three image boxes and load some pictures into them, we used the images of a flower, an orange and an ice cream. We shall program them to appear randomly when the user presses on the spin button, resembling a slot machine.

In addition, we insert a few labels and command buttons and a text box.

Label one command as Spin , another one as Add Cash and the last one as Cash Out. Besides that, one label is to display the balance and the other one is to display the winning amount of each spin.

We shall program the slot machine as a 20¢ machine, where each spin costs 20¢. We shall also determine how much money won for different combinations.

We will create a sub procedure and name is as spin(). This code for this sub procedure uses a randomize process to generate different combinations of the images. Besides that, we use a Do Loop and the DoEvents command to create an animation effect. The code for the Spin() sub procedure is:

### **The Code for Spin()**

```
Sub spin()
Do

    Lbl_Message.Visible = False
    Txt_Amount.Text = ""

    a = 1 + Int(Rnd * 3)
    b = 1 + Int(Rnd * 3)
    c = 1 + Int(Rnd * 3)

    If a = 1 Then
        Image1.Picture = LoadPicture("C:\Users\LENOVO\Documents\My
        Websites\Excelvbatutor\ExcelVBA\ExcelVBAExamples\images\orange.gif")
    End If

    If a = 2 Then
        Image1.Picture = LoadPicture("C:\Users\LENOVO\Documents\My
        Websites\Excelvbatutor\ExcelVBA\ExcelVBAExamples\images\flower.gif")
    End If

    If a = 3 Then
        Image1.Picture = LoadPicture("C:\Users\LENOVO\Documents\My
        Websites\Excelvbatutor\ExcelVBA\ExcelVBAExamples\images\icecream.gif")
    End If
```

```
If b = 1 Then
Image2.Picture = LoadPicture("C:\Users\LENOVO\Documents\My
Websites\Excelvbatutor\ExcelVBA\ExcelVBAExamples\images\orange.gif")
End If

If b = 2 Then
Image2.Picture = LoadPicture("C:\Users\LENOVO\Documents\My
Websites\Excelvbatutor\ExcelVBA\ExcelVBAExamples\images\flower.gif")
End If

If b = 3 Then
Image2.Picture = LoadPicture("C:\Users\LENOVO\Documents\My
Websites\Excelvbatutor\ExcelVBA\ExcelVBAExamples\images\icecream.gif")
End If

If c = 1 Then
Image3.Picture = LoadPicture("C:\Users\LENOVO\Documents\My
Websites\Excelvbatutor\ExcelVBA\ExcelVBAExamples\images\orange.gif")
End If

If c = 2 Then
Image3.Picture = LoadPicture("C:\Users\LENOVO\Documents\My
Websites\Excelvbatutor\ExcelVBA\ExcelVBAExamples\images\flower.gif")
End If

If c = 3 Then
Image3.Picture = LoadPicture("C:\Users\LENOVO\Documents\My
Websites\Excelvbatutor\ExcelVBA\ExcelVBAExamples\images\icecream.gif")
End If

If (a <> b And b <> c And c <> a) Then
    win = 0
End If

If (a = 1 And b = 1 And c <> 1) Or (a = 1 And c = 1 And b <> 1) Or (b = 1
And c = 1 And a <> 1) Then
    win = 5
End If
```

If (a = 2 And b = 2 And c <> 2) Or (a = 2 And c = 2 And b <> 2) Or (b = 2 And c = 2 And a <> 2) Then

    win = 10

End If

If (a = 3 And b = 3 And c <> 3) Or (a = 3 And c = 3 And b <> 3) Or (b = 3 And c = 3 And a <> 3) Then

    win = 20

End If

If (a = 1 And b = 1 And c = 1) Then

    win = 100

End If

If (a = 2 And b = 2 And c = 2) Then

    win = 150

End If

If (a = 3 And b = 3 And c = 3) Then

    win = 200

End If

x = x + 2

DoEvents

Loop Until x = 1000

Lbl\_Win.Caption = Str(win)

If win = 100 Then

    Lbl\_Message.Visible = True

    Lbl\_Message.Caption = " Mini Jackpot!"

End If

If win = 150 Then

    Lbl\_Message.Visible = True

    Lbl\_Message.Caption = "Jackpot!"

End If

If win = 200 Then

```
Lbl_Message.Visible = True  
Lbl_Message.Caption = "Mega Jackpot!"  
End If
```

```
updatebalance
```

```
End Sub
```

We also need to program the Spin command button to call the Spin() sub procedure. The code is as follows:

```
Private Sub Cmd_Spin_Click()  
Dim x, a, b, c, win As Integer  
If Val(Lbl_Balance.Caption) < 20 Then 'The user cannot spin when the  
balance falls below 20  
MsgBox ("Please add credit") 'Prompts the user to add cash when the  
balance falls below 20  
Txt_Amount.SetFocus  
Else  
spin  
End If
```

In addition, we need to create a sub procedure to keep track of the balance. We use the static keyword to create the sub procedure to prserve the value of the balance amount. The code is as follows:

```
Static Sub updatebalance()  
Dim myamount, balance As Integer  
  
myamount = Val(Lbl_Win.Caption) - 20  
balance = Val(Lbl_Balance.Caption) + myamount  
Lbl_Balance.Caption = Str(balance)  
  
End Sub
```

The runtime UI is as shown in Figure E.g.17.2

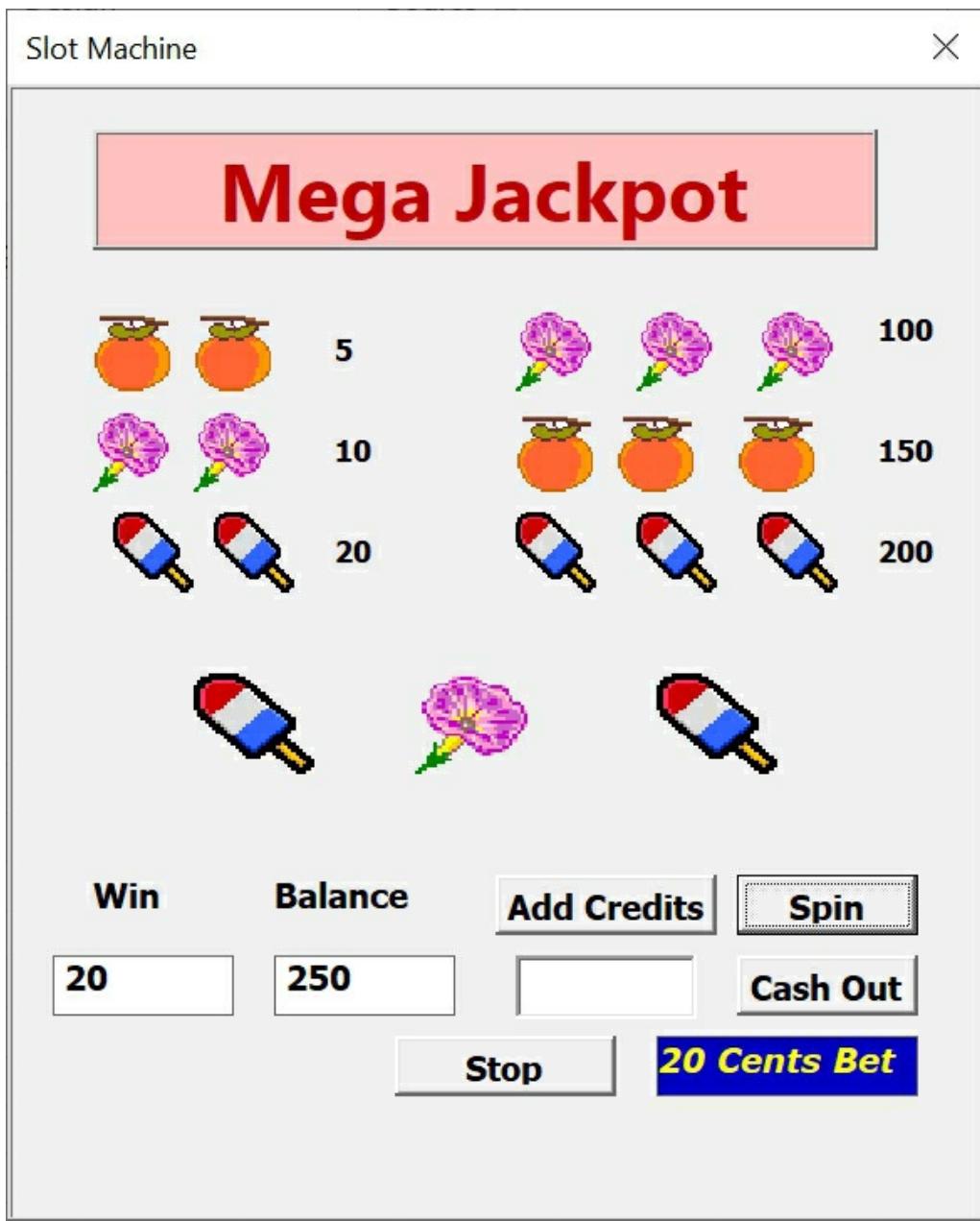


Figure E.g.16.2 The Runtime UI

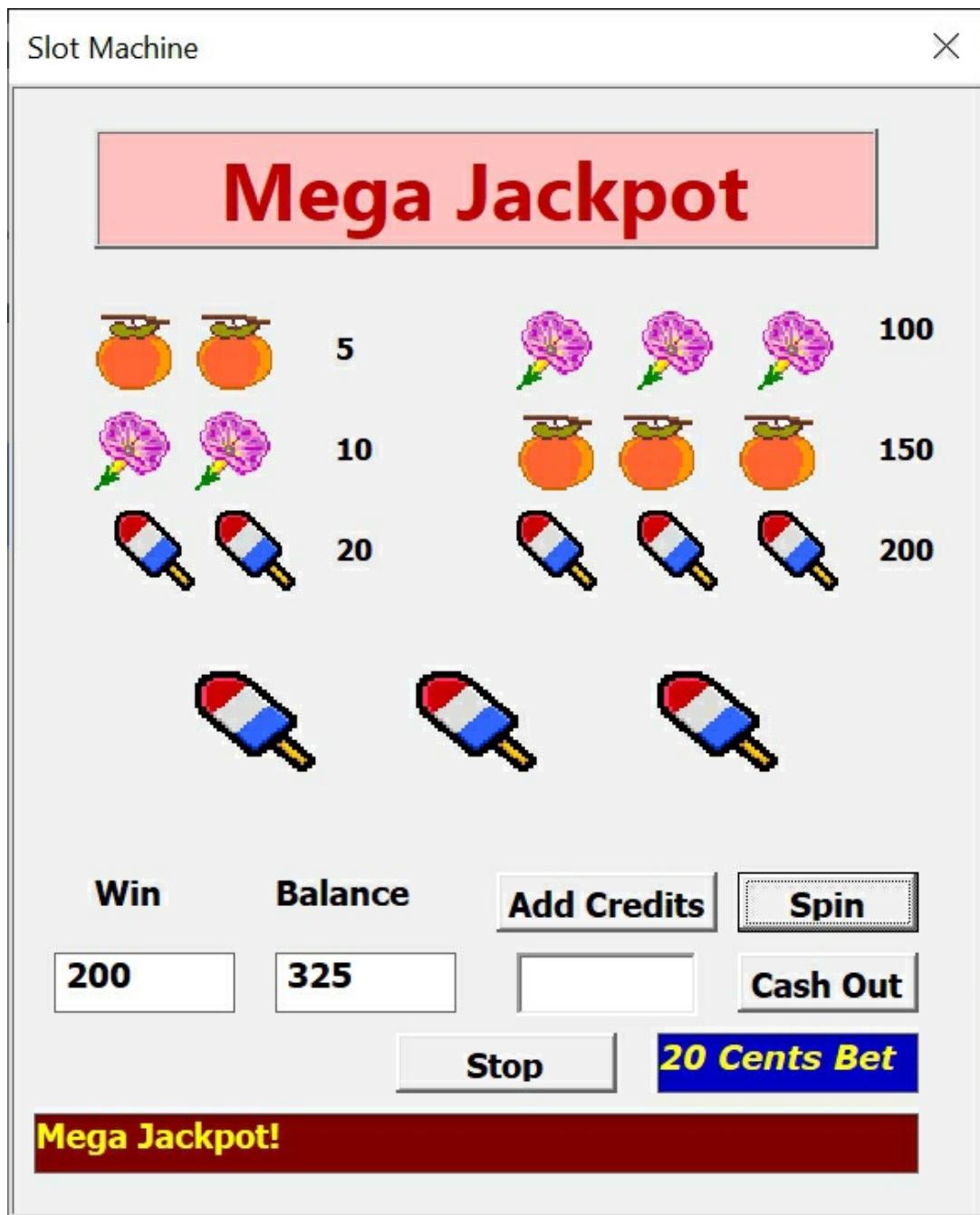


Figure E.g.16.3

Example 17 Quadratic Equation Solver

Quadratic equation is a straightforward high school mathematics problem. The quadratic equation solver was programmed to determine the number of roots the equation has as well as to compute the roots. It uses the determinant  $b^2 - 4ac$  to solve the problems.

If  $b^2 - 4ac > 0$ , then it has two roots and if  $b^2 - 4ac = 0$ , then it has one root, else it has no root.

To obtain the roots, the program uses the standard quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

To create the quadratic equation solver, start an Excel Worksheet and design the interface, as shown below:

A	B	C	D	E	F
<u><b>Quadratic Equation Solver</b></u>					
<b><math>ax^2+bx+c = 0</math></b>					
Please enter the value of a,b and c					
a					
b					
c					
Number of Roots					
x					
<b>Compute</b>					

**Figure E.g.17.1 The Design UI**

### The Code

```
Private Sub Cmd_Compute_Click()
Dim a, b, c, det As Integer
Dim root1, root2 As Single
a = Cells(8, 4)
b = Cells(9, 4)
c = Cells(10, 4)
det = (b ^ 2) - (4 * a * c)
```

If det > 0 Then

```
Cells(11, 4) = 2  
root1 = (-b + Sqr(det)) / (2 * a)  
root2 = (-b - Sqr(det)) / (2 * a)  
Cells(12, 4) = Round(root1, 4)  
Cells(12, 5) = Round(root2, 4)
```

```
ElseIf det = 0 Then
```

```
    root1 = (-b) / 2 * a
```

```
    Cells(11, 4) = 1
```

```
    Cells(12, 4) = Round(root1, 4)
```

```
    Cells(12, 5) = ""
```

```
Else
```

```
    Cells(11, 4) = 0
```

```
    Cells(12, 4) = "No Root"
```

```
    Cells(12, 5) = ""
```

```
End If
```

```
End Sub
```

A	B	C	D	E	F
<h2>Quadratic Equation Solver</h2>					
$ax^2+bx+c = 0$					
Please enter the value of a,b and c					
a	2				
b	3				
c	1				
Number of Roots	2				
x	-0.5		-1		
<input type="button" value="Compute"/>					

Figure E.g.17.2 The Runtime UI

## Example 18 Simple Harmonic Motion

Simple harmonic motion is the motion of a simple harmonic oscillator. The motion is periodic, as it repeats itself at standard intervals in a specific

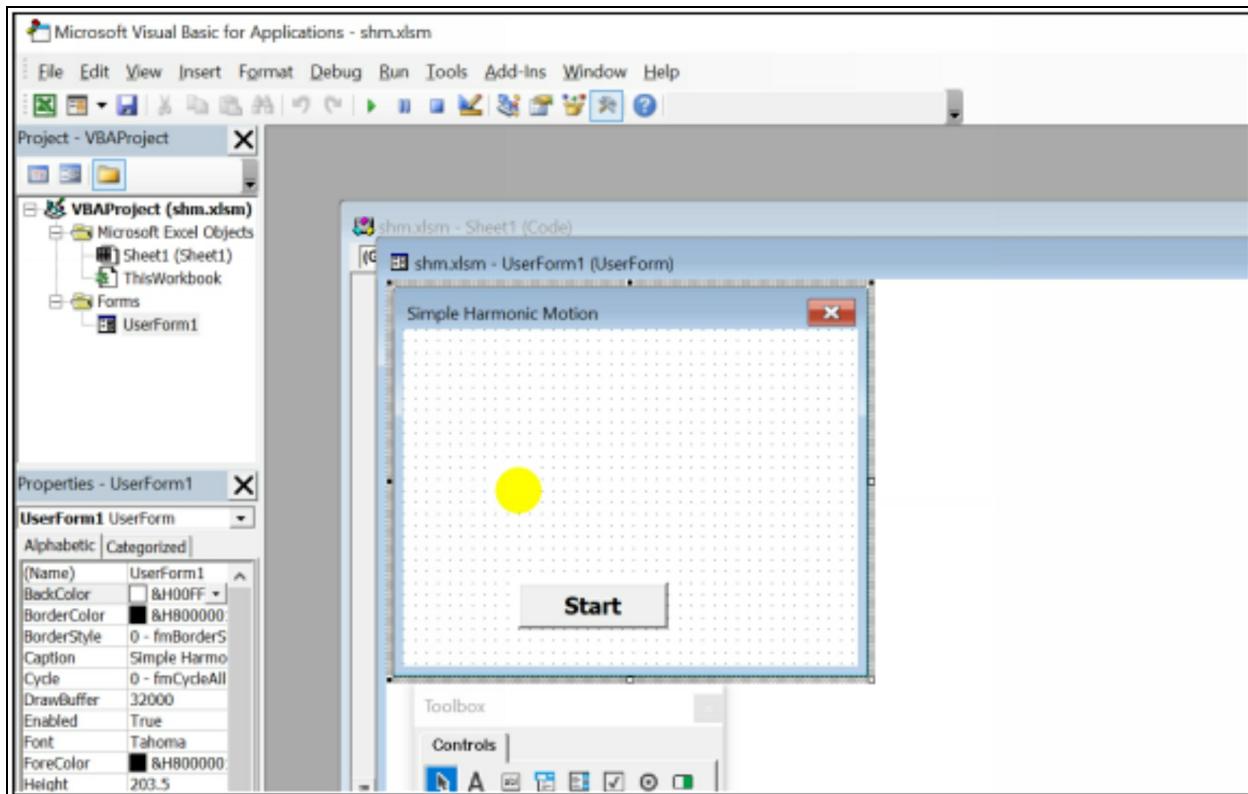
manner with constant amplitude. It is characterized by its amplitude, its period(the time for a single oscillation), its frequency(the number of cycles per unit time), and its phase(which determines the starting point on the sine wave). The period, and its inverse the frequency, are constants determined by the overall system, while the amplitude and phase are determined by the initial conditions (position and velocity)of that system. (Wikipedia, 2008).

The equation of a simple harmonic motion is:

$$x = A \cos(2\pi f t + \phi)$$

where x is the displacement, A is the amplitude of oscillation, f is the frequency, t is the elapsed time, and  $\phi$  is the phase of oscillation.

To create a simple model of simple harmonic motion in Excel VBA 365 , use the equation  $x=A\cos(\omega t)$ , and assign a value of 500 to A and a value of 50 to  $\omega$ .

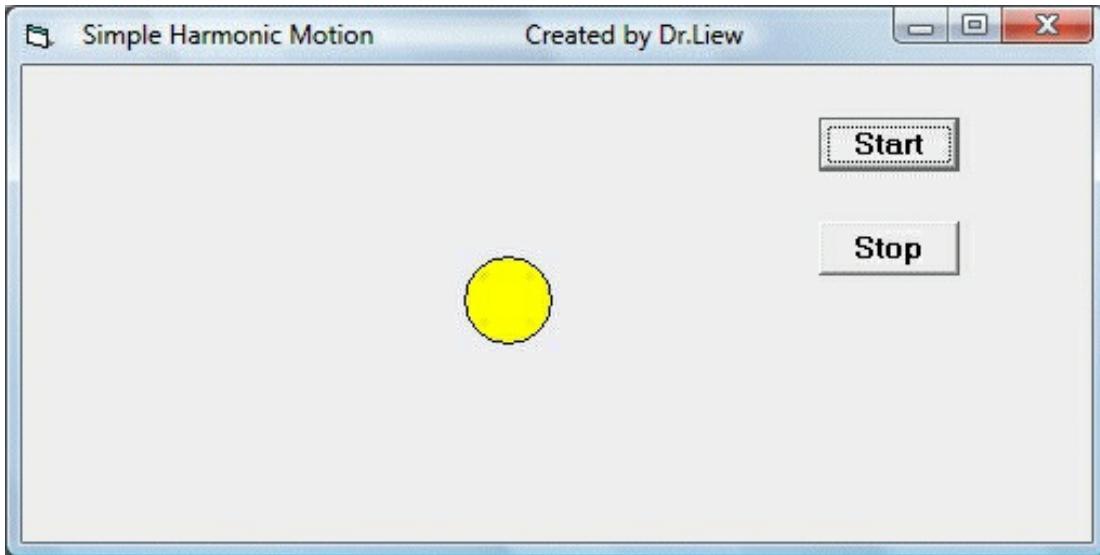


**Figure E.g.18.1 The Design UI**

In this example, insert a picture box and upload an image to the picture container. Next, insert a command button and change the captions to start.

We can use the move method to move the image whose path is determined by the formula  $x = 500 * \text{Cos}(50 * t)$ . We use a Do Loop and the DoEvents command to create the motion.

When you run the program, the image will move in an oscillating motion as shown in the figure below:



**Figure E.g.18.2**

### The Code

```
Private Sub Cmd_Start_Click()
Dim y As Integer
Dim t As Double
x = 0
Do
    t = t + 1
    y = 100 * Cos(50 * t)
    Image1.Move Image1.Left + y
    DoEvents
    x = x + 2
Loop
End Sub
```

# Example 19 Simultaneous Equation

---

Simultaneous equations are equations that involves two or more unknown variables. There must be as many equations as the number of unknown variables for us to solve the problem. In this example, we shall only solve linear simultaneous equations. Linear simultaneous equations take the following forms:

$$ax+by=m$$

$$cx+dy=n$$

There are two ways to solve simultaneous equations, substitution or elimination. In this program, we use the substitution method. Reorganizing the equations derived the following formulas:

$$x = (b * n - d * m) / (b * c - a * d)$$

$$y = (a * n - c * m) / (a * d - b * c)$$

To design the UI, we allocate cells for entering the values of a, b, c,d, m ,n and also cells to display the values of x and y. Insert a command button and change its caption to Solve Equation.

## The Code

```
Private Sub CmdSolve_Click()
Dim a, b, c, d, m, n As Integer
Dim x, y As Double

a = Cells(7, 4)
b = Cells(7, 6)
m = Cells(9, 4)
c = Cells(8, 4)
d = Cells(8, 6)
n = Cells(9, 6)
x = (b * n - d * m) / (b * c - a * d)
y = (a * n - c * m) / (a * d - b * c)
Cells(11, 4) = Round(x, 2)
Cells(12, 4) = Round(y, 2)
```

End Sub

To limit the answers to two decimal places, we use the round function. The runtime UI is shown in the Figure below:

A	B	C	D	E	F	G	H	I
<b>Simultaneous Equation Solver</b>								
$ax+by=m$								
$cx+dy=n$								
a=	5	b=	2					
c=	2	d=	4					
m=	10	n=	12					
x=	1							
y=	2.5							
<b>Solve Equation</b>								

Figure E.g.19.1 The Runtime Interface

## Example 20 Star War

---

This Excel VBA program demonstrates the principle of projectile in physics. At an angle and a launch velocity, the projectile can reach a certain range and certain height. The maximum range is at an angle of 45 degree. This principle can be applied in the military field where a missile can be launched at a velocity and an angle to hit a remote target. It can also be applied in other scientific and technological fields. This game provides a good training for students in their abilities in making estimation.

In this program, we use the formula  $v \sin \theta - \frac{1}{2}gt^2$  as the vertical component of the displacement and  $v \cos \theta$  as the horizontal component of the displacement( where  $g$  is the gravitational acceleration ,  $v$  the launch velocity and  $\theta$  the launch angle). To enable the missile to fly, we use a combination of the Object.Move method and the object coordinate system , i.e. object. left and object.Top.

We also apply the random function Rnd to launch the objects at random positions at each new game. Besides that, we also keep track of the total score. The code to keep track of the total score is as follows:

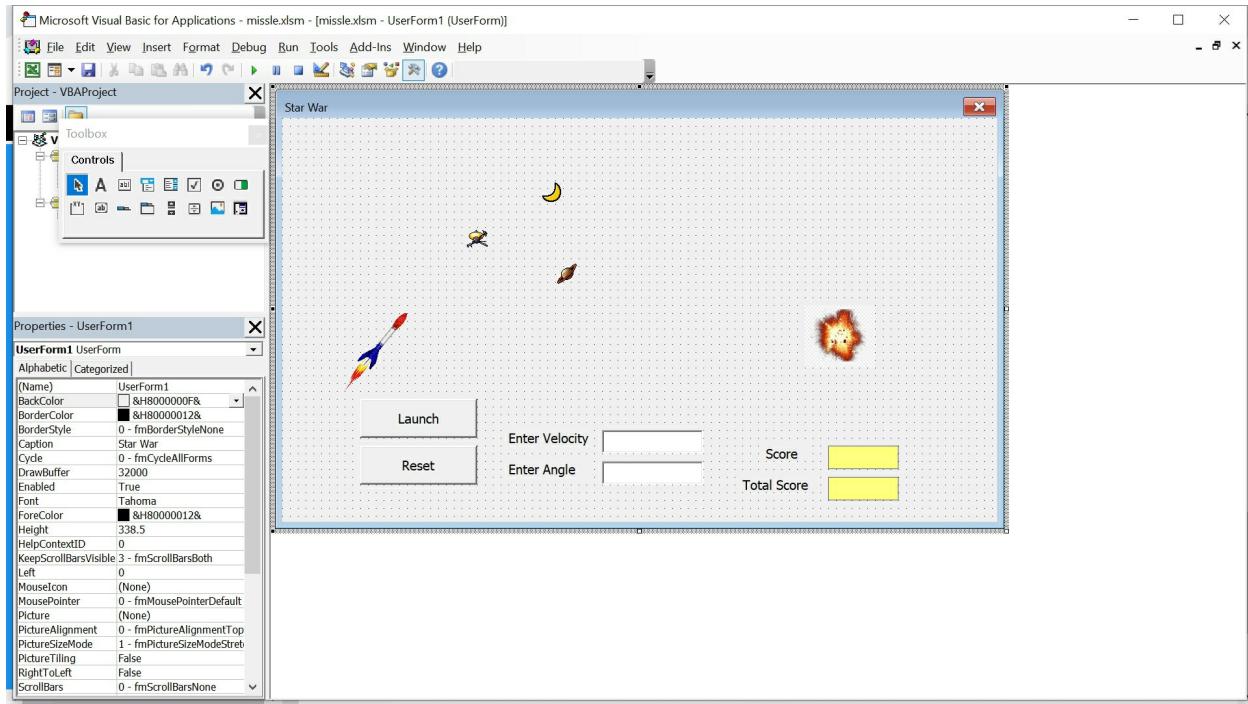
```
Static Sub countscore()
Dim myscore, Tscore As Integer

myscore = Val(Lbl_Score.Caption)
Tscore = Tscore + myscore
Lbl_Total.Caption = Str(Tscore)
End Sub
```

\* Static is the declaration keyword that stores the memory even after a procedure is executed.

In the design stage, insert a Userform then insert a missile image, three objects and an explosion image. In addition, we insert a text box for entering the velocity and another text box for entering the angle. Also, we inserted two labels, one to show the current score and another one to display the total score.

The design interface is as shown below:



**Figure E.g.20.1 The Design UI**

## The Code

```

Private Sub Launch()
Dim a As Variant
Dim t As Variant
Dim Y As Variant
Dim w As Variant
Dim i As Variant
Dim score As Integer
Dim left1, left2, left3, top1, top2, top3 As Variant
Dim backgr As Integer
Dim totalscore As Integer

left1 = Int(Rnd * 200) + 100
left2 = Int(Rnd * 200) + 100
left3 = Int(Rnd * 200) + 100
top1 = Int(Rnd * 40) + 10
top2 = Int(Rnd * 40) + 10
top3 = Int(Rnd * 40) + 10

```

t = 0

Do

v = Val(TxtVelocity.Text)

a = Val(TxtAngle.Text)

t = t + 1

Y = v \* Sin(a \* 3.141592654 / 180) \* (t / 100) - 4.9 \* ((t / 100) ^ 2)

x = v \* Cos(a \* 3.141592654 / 180) \* (t / 100)

Image1.Move Image1.Left + x, Image1.Top - Y

t = t + 1

DoEvents

Loop Until t = 10

If Image4.Visible = True And (Image1.Left < left3 + 100 And Image1.Left > left3 - 100) And (Image1.Top < top3 + 300 And Image1.Top > top3 - 100)

Then

i = 2

Image5.Left = Image4.Left

Image5.Top = Image4.Top

Image5.Visible = True

Image4.Visible = False

Image1.Visible = False

score = score + 50

ElseIf Image3.Visible = True And (Image1.Left < left2 + 100 And Image1.Left > left2 - 100) And (Image1.Top < top2 + 100 And Image1.Top > top2 - 100) Then

i = 1

Image5.Left = Image3.Left

Image5.Top = Image3.Top

Image5.Visible = True

Image3.Visible = False

Image1.Visible = False

```
score = score + 100
```

```
ElseIf Image2.Visible = True And (Image1.Left < left1 + 100 And  
Image1.Left > left1 - 100) And (Image1.Top < top1 + 100 And Image1.Top >  
top1 - 100) Then
```

```
i = 0
```

```
Image5.Left = Image2.Left
```

```
Image5.Top = Image2.Top
```

```
Image5.Visible = True
```

```
Image2.Visible = False
```

```
Image1.Visible = False
```

```
score = score + 200
```

```
End If
```

```
Lbl_Score.Caption = Str(score)
```

```
countscore
```

```
End Sub
```

```
Static Sub countscore()
```

```
Dim myscore, Tscore As Integer
```

```
myscore = Val(Lbl_Score.Caption)
```

```
Tscore = Tscore + myscore
```

```
Lbl_Total.Caption = Str(Tscore)
```

```
End Sub
```

```
Private Sub Cmd_launch_Click()
```

```
Launch
```

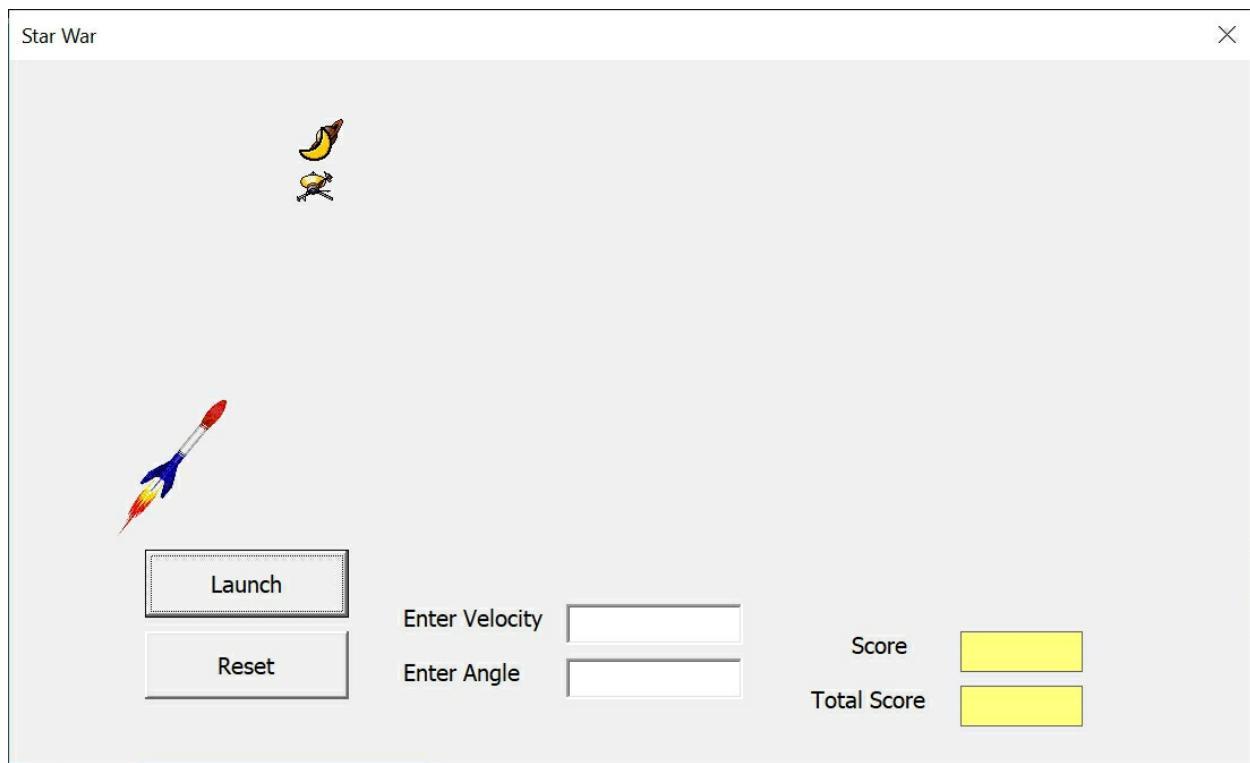
```
End Sub
```

```
Private Sub Cmd_Reset_Click()
```

```
Image1.Visible = True
```

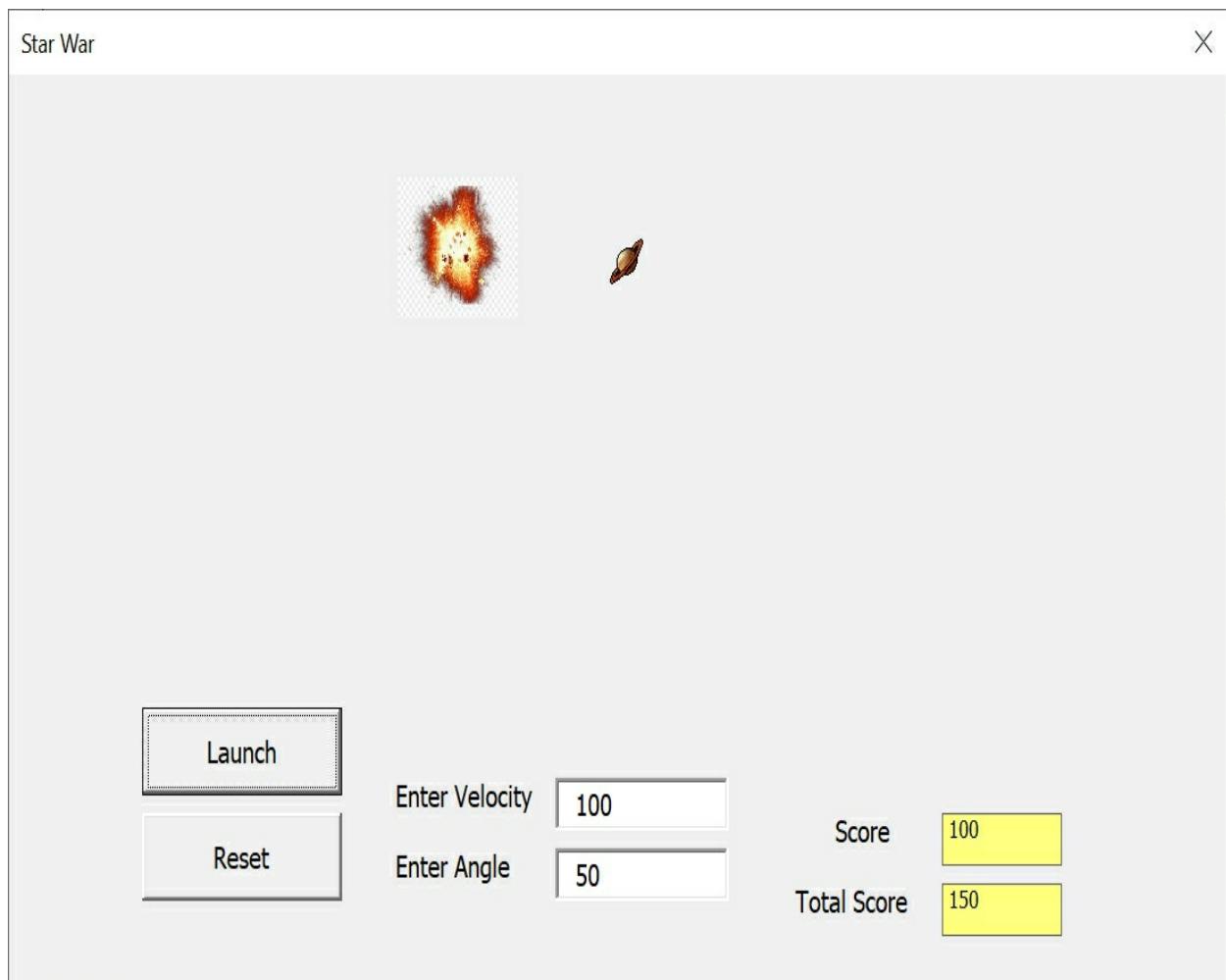
```
left1 = Int(Rnd * 200) + 100
left2 = Int(Rnd * 200) + 100
left3 = Int(Rnd * 200) + 100
top1 = Int(Rnd * 40) + 10
top2 = Int(Rnd * 40) + 10
top3 = Int(Rnd * 40) + 10
Image2.Left = left1
Image3.Left = left2
Image4.Left = left3
Image2.Top = top1
Image3.Top = top2
Image4.Top = top3
Image1.Left = 48
Image1.Top = 150
Image2.Visible = True
Image3.Visible = True
Image4.Visible = True
Image5.Visible = False
```

End Sub



**Figure E.g.20.2 The Runtime UI**

Starwar Start



**Figure E.g.20.3 Explosion**

## Example 21 Stock Trading

We can create a stock trading simulation application in Excel VBA 365(Any version of MS Excel will do).

For stock trading, we need to consider the following data:

- Asking Price- Price offered by the sellers
- Bidding Price- Price offered by the buyers
- Selling Quantity-Total number of shares available for sale on the stock market
- Buying Quantity- Total number of shares the buyers bid on the stock market
- Last Done Price- The price for the last transaction
- Order Price – Price bid by the trader to buy or to sell
- Order Quantity- Number of Shares ordered by the trader
- Average Price- Average share price
- Buy Value – Average value of shares paid by the trader
- Gross Market Value- The current market of shares owned by the trader
- The total number of shares in the hand of the trader.
- Profit or Loss

We shall use the following variables to represent the data:

- AP- Asking Price
- BD- Bidding Price
- SQ- Selling Quantity
- BQ- Buying Quantity
- LP- Last Done Price
- OP- Order Price
- OQ- Order Quantity
- AVP- Average Price
- BV- Buy Value
- MV- Gross Market Value
- TQ- Total Number of Shares in Hand

In this example, we need to design the UI on the Worksheet, as shown in the following figure. We insert three command buttons, one to open the market,

another one to close the market and one more to submit the order. Also insert two option buttons for the user to select buy or sell.

Global Stock Exchange					
	Asking Price	Sell Quantity	Bid Price	Bid Quantity	Last Done Price
7	Order Panel				
9	<input type="radio"/> Buy	Quantity			
10	<input type="radio"/> Sell	Price		Submit	
13	Shares at Hand				
15	Average Price	Total Quantity	Buy Value	Gross Market Value	Profit/Loss
19				Open Market	Close Market

**Figure E.g.21.1 The Design UI**

We need to use the Do Loop procedure and the DoEvents command to generate various random values and present them on several cells on the spreadsheet. The values are the Asking Price, the Bidding Price, the Selling Quantity, the Buying Quantity and the Last Done Price.

We can control the prices change interval by setting the incremental value of  $x$ .

We declare the variables in the General section, as follows:

```

Dim ranNum1, ranNum2, randNum3, ranNum4 As Double
Dim AP As Single
Dim SQ As Integer
Dim BD As Single
Dim BQ As Integer
Dim OQ As Long
Dim OP As Single

```

```
Dim LP As Single  
Dim LVOL As Integer  
Dim AVP As Single  
Dim TQ As Long  
Dim BV As Single  
Dim MV As Single  
Dim PL As Single
```

To set the initial values when we start the worksheet, we use the following code:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)  
    Opt_Sell.Value = False  
    Opt_Buy.Value = False  
    ranNum1 = Rnd * 2 + 3  
    ranNum2 = Rnd * 2 + 3  
    ranNum3 = Rnd * 2 + 3  
    ranNum4 = Rnd * 2 + 3  
    AVP = Round(ranNum1, 3) * 5 ' Average Price  
    TQ = Int(Round(Rnd(), 2) * 10000) + 1000 'Total Shares at Hand  
    BV = AVP * TQ 'Buy Value  
    AP = ranNum2 * 5 'Asking Price  
    SQ = Int(Round(Rnd, 2) * 10000) + 1000  
    BD = Round(ranNum2, 3) * 5 'Bidding Price  
    BQ = Int(Round(Rnd, 2) * 10000) + 1000  
    LP = Round(ranNum2, 3) * 5 'Last Done Price  
    MV = LP * TQ  
    PL = MV - BV  
    LVOL = Int(Round(Rnd, 2) * 10000) + 1000 'Last Done Volume  
    sheet1.Cells(5, 2) = Format(AP, "#,##.00")  
    sheet1.Cells(5, 3) = SQ  
    sheet1.Cells(5, 4) = Format(BD, "#,##.00")  
    sheet1.Cells(5, 5) = Format(BQ, "#,##.00")  
    sheet1.Cells(11, 4) = Format(OP, "#,##.00")  
    sheet1.Cells(16, 2) = Format(AVP, "#,##.00")  
    sheet1.Cells(16, 3) = Format(TQ, "#,##.00")
```

```
sheet1.Cells(16, 5) = Format(MV, "#,##.00")
sheet1.Cells(16, 6) = Format(PL, "#,##.00")
sheet1.Cells(5, 6) = Format(LP, "#,##.00")
sheet1.Cells(16, 4) = Format(BV, "#,##.00")
sheet1.Cells(5, 7) = LVOL
End Sub
```

The code to open the market is as follows:

```
Private Sub Cmd_Open_Click()
Dim x As Single
x = 0

Do
x = x + 50
ranNum1 = Rnd * 2 + 3
ranNum2 = Rnd * 2 + 3
ranNum3 = Rnd * 2 + 3
AP = ranNum1 * 5
SQ = Int(Rnd * 10000) + 1000
BD = ranNum2 * 5
BQ = Int(Rnd * 10000) + 1000
LP = ranNum3 * 5 'Last Done Price
MV = LP * TQ ' Market Value
PL = MV - BV
LVOL = Int(Round(Rnd(), 2) * 10000) + 1000 'Last Done Volume
sheet1.Cells(5, 2) = Format(AP, "#,##.00")
sheet1.Cells(5, 3) = Format(SQ, "#,##.00")
sheet1.Cells(5, 4) = Format(BD, "#,##.00")
sheet1.Cells(5, 5) = Format(BQ, "#,##.00")
sheet1.Cells(5, 6) = Format(LP, "#,##.00")
sheet1.Cells(5, 7) = LVOL
sheet1.Cells(16, 5) = Format(MV, "#,##.00")
sheet1.Cells(16, 6) = Format(PL, "#,##.00")

DoEvents
Loop
```

```
End Sub
The Code to Submit the Order
Private Sub Cmd_Submit_Click()
OQ = sheet1.Cells(9, 4)
OP = sheet1.Cells(11, 4)
LP = sheet1.Cells(5, 6)
If Opt_Buy.Value = True And BV >= 1000 Then
    AVP = (AVP * TQ + OP * OQ) / (TQ + OQ)
    TQ = TQ + OQ
    sheet1.Cells(16, 3) = TQ
    MV = LP * TQ
    BV = TQ * AVP
    PL = MV - BV
    sheet1.Cells(5, 6) = Format(LP, "#,##.00")
    sheet1.Cells(16, 5) = Format(MV, "#,##.00")
    sheet1.Cells(16, 4) = Format(BV, "#,##.00")
    sheet1.Cells(16, 6) = Format(PL, "#,##.00")
    sheet1.Cells(16, 2) = Format(AVP, "#,##.00")
ElseIf Opt_Buy.Value = True And BV < 1000 Then MsgBox ("You don't
have enough fund to buy, reduce order")
End If
If Opt_Sell.Value = True And TQ >= OQ Then
    AVP = (AVP * TQ + OP * OQ) / (TQ + OQ)
    TQ = TQ - OQ
    sheet1.Cells(16, 3) = TQ
    MV = LP * TQ
    BV = TQ * AVP
    PL = MV - BV
    sheet1.Cells(16, 5) = Format(MV, "#,##.00")
    sheet1.Cells(16, 4) = Format(BV, "#,##.00")
    sheet1.Cells(16, 6) = Format(PL, "#,##.00")
    sheet1.Cells(16, 2) = Format(AVP, "#,##.00")

ElseIf Opt_Sell.Value = True And TQ < OQ Then
    MsgBox ("Not enough shares, reduce order")
End If
```

End Sub

The code to close the market:

Private Sub Cmd\_Close\_Click()

Cancel = True

End

End Sub

A	B	C	D	E	F	G	H
<b>Global Stock Exchange</b>							
Asking Price		Sell Quantity	Bid Price	Bid Quantity	Last Done Price	Last Volume	
19.64		5,799.00	16.9	1,632.00	16.48	4899	
<b>Order Panel</b>							
<input checked="" type="radio"/> Buy		Quantity	1000	Submit			
<input type="radio"/> Sell		Price	20				
<b>Shares at Hand</b>							
Average Price		Total Quantity	Buy Value	Gross Market Value	Profit/Loss		
17.77		2,600.00	46,202.00	42,856.19	-3,345.81		
				Open Market	Close Market		

Figure E.g.21.2 The Runtime UI

# Index

---

## A

ActiveWorkbook · 200  
ActiveX controls · 17, 137  
AddItem · 86, 88, 147, 148, 149, 150, 229  
amortization · 260  
Application object · 198  
arguments · 54, 62, 152, 154, 164, 169, 178, 213, 220  
ASCII · 184, 185, 186, 286  
asteSpecial · 133  
Autofill · 107, 112, 133, 134  
Average · 94, 131, 153, 314, 316

---

## B

BMI · 210, 211, 241, 242  
boggle · 263  
Boolean · 37, 38, 39, 42, 87, 88, 89, 169, 173, 220, 269, 277  
by reference · 162  
by Value · 162  
ByRef · 162, 163  
ByVal · 26, 27, 162, 163, 184, 187, 197, 210, 212, 213, 216, 220, 316

---

## C

caption · 21, 187, 193, 194, 198, 200, 225, 252, 306  
Case Is · 84, 94, 96, 247, 249  
cells · 23, 28, 30, 45, 72, 73, 75, 79, 81, 82, 98, 100, 101, 102, 104, 105, 106, 107, 108, 109, 112, 121, 125, 126, 127, 128, 130, 133, 134, 135, 156, 157, 158, 159, 191, 194, 249, 263, 283, 306, 315  
Check Box · 138, 139  
chr · 56, 57, 185, 263, 286  
Chr · 54, 56, 57, 60, 185, 186, 187, 226, 263, 286  
class module · 208, 210, 213, 220, 223  
clear · 86, 102, 103, 104, 105, 106, 112, 147, 234, 265, 269, 283  
ClearContents · 102, 104, 105, 106, 107, 112, 113, 133, 283, 284  
ClearFormats · 105, 106, 283, 284  
Click · 21, 23, 29, 30, 39, 41, 44, 45, 55, 57, 60, 63, 67, 69, 71, 72, 73, 74, 76, 77, 79, 81, 82, 83, 84, 86, 88, 89, 93, 94, 95, 96, 100, 102, 105, 106, 107, 109, 110, 112, 113, 114, 115, 116, 117, 118, 120, 121, 122, 123, 124, 125, 126, 127, 129, 130, 131, 132, 133, 134, 135, 136, 138, 139, 141, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 155, 162, 164, 166, 167, 169, 173, 176, 177, 179, 181, 184, 191, 193, 194, 196, 198, 200, 202, 205, 206, 211, 212, 214, 217, 218, 221, 225, 229, 235, 236, 237,

241, 243, 244, 246, 247, 249, 253, 255, 256, 257, 258, 259, 261, 263, 270, 271, 272, 275, 276, 277, 280, 283, 284, 286, 289, 292, 296, 300, 305, 306, 311, 317, 318  
Close method · 120  
Color · 29, 30, 78, 100, 128, 130, 133, 145, 146, 147, 151, 283, 284  
ColorIndex · 77, 78, 83, 128, 129, 131, 132  
column · 72, 75, 82, 130, 131, 132, 159, 161, 191, 237  
ColumnCount · 191  
Columns · 123, 125, 126, 128, 130, 131, 132  
COM · 208  
Combo Box · 138, 149  
command button · 18, 19, 21, 22, 29, 75, 98, 109, 112, 118, 119, 120, 124, 154, 164, 193, 198, 200, 211, 213, 216, 223, 226, 228, 235, 265, 270, 279, 296, 304, 306  
Command Button · 19, 138  
component object model · 208  
Conditional operators · 65  
controls · 17, 18, 98, 137, 138, 188, 189, 190, 266  
ControlSource · 193  
Count · 44, 102, 122, 123, 124  
creating files · 198

---

## D

Date · 37, 38, 39, 131, 175, 176, 177, 179, 181  
DateAdd function · 180  
DateDiff function · 180  
DatePart function · 178  
Dblclick · 184  
Delete method · 125  
Developer tab · 17, 137, 234, 289, 292  
Dim · 37, 38, 39, 40, 41, 42, 44, 45, 61, 63, 67, 69, 71, 72, 73, 74, 76, 77, 79, 80, 81, 82, 84, 88, 89, 93, 94, 95, 96, 102, 106, 107, 113, 120, 123, 141, 143, 146, 148, 153, 155, 161, 162, 165, 166, 169, 171, 172, 173, 198, 200, 202, 205, 206, 210, 211, 212, 214, 217, 221, 225, 228, 241, 243, 244, 246, 247, 249, 253, 261, 263, 269, 284, 286, 289, 296, 300, 305, 306, 308, 309, 311, 315, 316, 317  
Do Loop · 30, 79, 293, 304, 315  
Do.....Loop While · 84  
Do...Loop · 72  
DoEvents · 257, 258, 259, 281, 286, 289, 293, 295, 304, 305, 306, 310, 315, 317

---

## E

Else · 39, 56, 57, 58, 60, 61, 65, 66, 67, 69, 71, 77, 92, 93, 94, 95, 96, 138, 140, 144, 148, 150, 151, 159, 169, 173, 184, 212, 218, 220, 221, 228, 236, 241, 275, 276, 277, 296, 301  
Errors handling · 31  
event procedure · 26, 27, 152, 154, 164, 224, 226  
Excel VBA code · 15, 65, 164, 187, 198, 200, 210, 213, 246

---

## **F**

fixed-length string · 38  
Font object · 100, 101, 128  
FontStyle property · 128  
For...Next · 23, 72, 73, 75  
Form controls · 17, 137  
Format function · 175  
Formatting · 128  
functions · 15, 28, 54, 152, 153, 155, 163, 171, 173, 175, 176, 181, 208, 210, 274, 286  
Functions · 152, 153, 171, 175, 176, 177

---

## **G**

Get · 214, 215, 220, 227, 228  
GetOpenFilename · 198, 199, 253  
GetSaveAsFilename · 118, 198, 200

---

## **I**

If...Then...ElseIf · 66, 68, 145, 289  
If...ElseIf · 56, 57  
If...Then...Else · 39  
Image · 138, 188  
Initialize · 184, 196, 209, 216, 220, 224, 229  
InputBox · 54, 60, 61, 62, 63, 67, 69, 84, 90, 93, 94, 96, 104, 106, 107, 113, 155, 166, 167, 202, 229, 235  
InStr · 171, 173  
Int · 61, 71, 84, 213, 246, 247, 263, 280, 286, 289, 293, 309, 311, 312, 316, 317  
IntelliSense · 101, 156  
Interior · 29, 30, 100, 133, 145

---

## **K**

keyboard events · 184  
KeyDown · 184  
KeyPress · 184, 187  
KeyUp · 184

---

## **L**

Label · 138, 188, 211, 212, 213, 216, 256, 293  
Left · 171, 173, 226, 236, 257, 258, 259, 305, 310, 311, 312  
Len · 173, 174, 226  
Let · 18, 68, 77, 82, 87, 114, 210, 214, 215, 220, 227, 228  
List Box · 138, 147

ListBox · 188, 191  
logical operators · 65, 66  
looping · 72, 79, 257

---

## M

Macro · 28, 155, 199, 200  
Max · 153  
median · 132, 153  
Median · 131, 132, 153  
Method · 102, 105, 106, 107, 117, 118, 119, 120, 124, 125, 126, 134  
Mid · 172, 173  
Min · 153  
Mode · 18, 19, 22, 131, 153, 154  
module · 28, 155, 156, 158, 208, 209, 210, 212, 213, 214, 215, 218, 219, 220, 223, 224, 225, 227  
MouseDown · 184, 187  
MouseMove · 184  
MouseUp · 184  
MsgBox · 21, 27, 54, 55, 56, 57, 58, 60, 61, 63, 67, 69, 70, 71, 75, 84, 85, 89, 93, 94, 95, 96, 114, 115,  
116, 121, 122, 123, 124, 138, 139, 140, 143, 144, 153, 155, 166, 167, 169, 172, 174, 184, 187, 202,  
205, 220, 221, 229, 247, 250, 276, 290, 296, 318  
multiline property · 205

---

## N

named constant · 55, 56, 57, 59, 60  
Non-numeric data types · 37  
Now () Function · 175  
Numeric data types · 36

---

## O

objects · 23, 26, 98, 99, 100, 102, 113, 114, 125, 137, 152, 208, 308  
Open method · 119  
opening files · 198  
option button · 144, 145  
Option Explicit · 40, 41, 44, 45

---

## P

PasteAll · 135  
PasteFormulas · 135  
PasteValues · 135, 136  
pmt · 243, 244, 261  
Properties window · 19, 22  
PV · 213, 214, 246

PVIFA · 243, 261

---

## R

Range · 23, 26, 27, 28, 29, 30, 36, 37, 44, 82, 99, 100, 101, 102, 105, 106, 107, 108, 109, 110, 112, 113, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 145, 153, 162, 164, 166, 208, 235, 236, 249, 283, 284, 316  
Reading a File · 206  
Right · 172, 173  
Rnd · 61, 71, 263, 280, 286, 289, 293, 308, 309, 311, 312, 316, 317  
Round · 74, 153, 241, 261, 271, 272, 275, 276, 277, 301, 306, 316, 317  
row · 23, 72, 75, 131, 132  
Rows · 124, 125, 126, 128  
RowSource · 191

---

## S

Save · 117, 119, 200  
SaveAs · 117, 118, 124, 200  
Select Case · 84, 92, 93, 94, 95, 96, 158, 159, 212, 228, 247, 249  
Select method · 106, 134  
Set · 110, 134, 153, 193, 214, 215, 224, 227, 229, 249, 252, 283, 284  
step increment · 73, 74  
string · 29, 37, 38, 42, 45, 54, 95, 143, 158, 171, 173, 185, 215, 218  
style value · 54, 57, 59  
sub procedure · 60, 164, 169, 220, 228, 229, 235, 257, 270, 293, 296  
Sub Procedures · 152, 164  
Sum · 44, 86, 88, 131, 143, 153

---

## T

Text Box · 138, 143  
Text File · 201  
Time · 60, 129, 131, 175, 176, 177, 179  
Toggle button · 150  
Top · 236, 257, 258, 259, 308, 310, 311, 312  
type mismatch · 66, 68

---

## U

user-defined function · 152, 154, 155, 158  
UserForm · 28, 183, 184, 187, 188, 190, 191, 194, 195, 196, 198, 200, 210, 211, 212, 213, 216, 221, 224, 225, 228, 229, 279, 289, 292

---

## V

variables · 24, 35, 36, 37, 38, 42, 44, 45, 162, 208, 227, 235, 269, 306, 314, 315  
variant · 62, 66, 68  
VBA · 1, 2, 3, 15, 16, 17, 21, 22, 23, 24, 26, 28, 29, 35, 36, 38, 40, 43, 54, 55, 56, 59, 62, 65, 66, 68, 72, 78, 79, 86, 92, 96, 98, 99, 101, 102, 113, 114, 119, 126, 130, 131, 137, 148, 149, 152, 158, 164, 168, 169, 171, 173, 183, 184, 187, 194, 198, 200, 201, 208, 210, 212, 213, 220, 223, 227, 234, 240, 243, 246, 247, 249, 251, 253, 257, 260, 265, 279, 283, 286, 289, 292, 303, 308, 314  
vbAbortRetryIgnore · 55, 60  
VBE · 21, 22, 24, 25, 26, 198, 251, 253, 279, 292  
vbExclamation · 60  
vbYesCancel · 57  
Visual Basic 6 · 3, 15, 23, 28  
Visual Basic Editor · 16, 17, 21, 22, 24, 25, 28, 98, 155, 292  
Visual Basic for Applications · 15, 171, 172

---

## W

Web Browser · 194, 195  
Web Browser control · 195  
While.... Wend Loop · 86  
Windows Media Player control · 251, 252  
Workbook · 16, 28, 98, 114, 115, 117, 121, 124, 125, 198, 208  
worksheet · 17, 18, 24, 25, 26, 98, 104, 105, 106, 112, 121, 123, 124, 125, 234, 243, 263, 283, 316  
Worksheet Methods · 124  
Worksheet Properties · 121  
Worksheetfunction · 44

---

## X

xlGrowthTrend · 109, 111  
xlsm · 28, 120, 199, 200, 292  
Xor · 66