EXCEL VBA MADE EASY

Dr. Liew Voon Kiong

Disclaimer

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

Trademarks

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

Liability

The purpose of this book is to provide basic guidelines for people interested in Excel VBA 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 2009. 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 **VBA Tutorial** website at **http://www.vbtutor.net/VBA/vba_tutorial.html** for their support and encouragement.

About the Author

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

To provide more support for Visual Basic students, teachers, and hobbyists, Dr. Liew is also the author of the Visual Basic Made Easy series, which includes **Visual Basic 6 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

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 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 formulae 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 2003, but you may apply it in other versions of MS Excel, for example, MS Excel Office 365.

1.2 The Visual Basic Editor in MS Excel

There are two ways to write VBA code in MS Excel. The first is to place a command button on the spreadsheet and start programming by clicking the command button to launch the Visual Basic Editor. The other way is to launch the Visual Basic Editor by clicking on the Tools menu then select Macro from the drop-down menu and choose Visual Basic Editor.

Let us start with the command button. To place a command button on the MS Excel spreadsheet, click View on the MS Excel menu bar followed by clicking on Toolbars and then select the Control Toolbox to bring up the control toolbox, as shown in Figure 1.1. Place the command button on the spreadsheet, as shown in Figure 1.2.

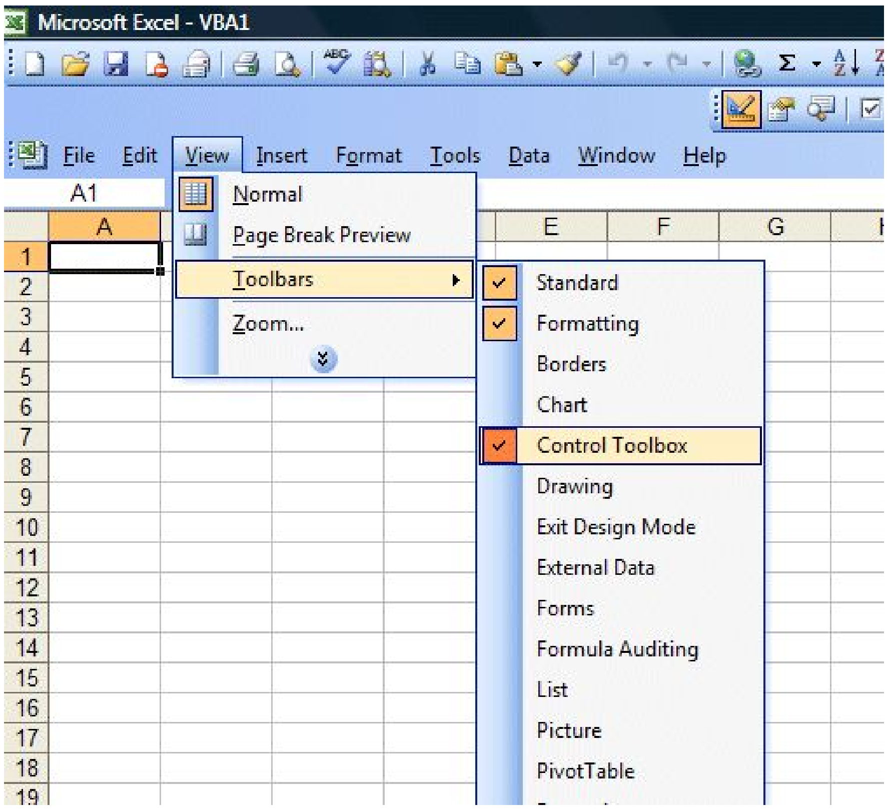
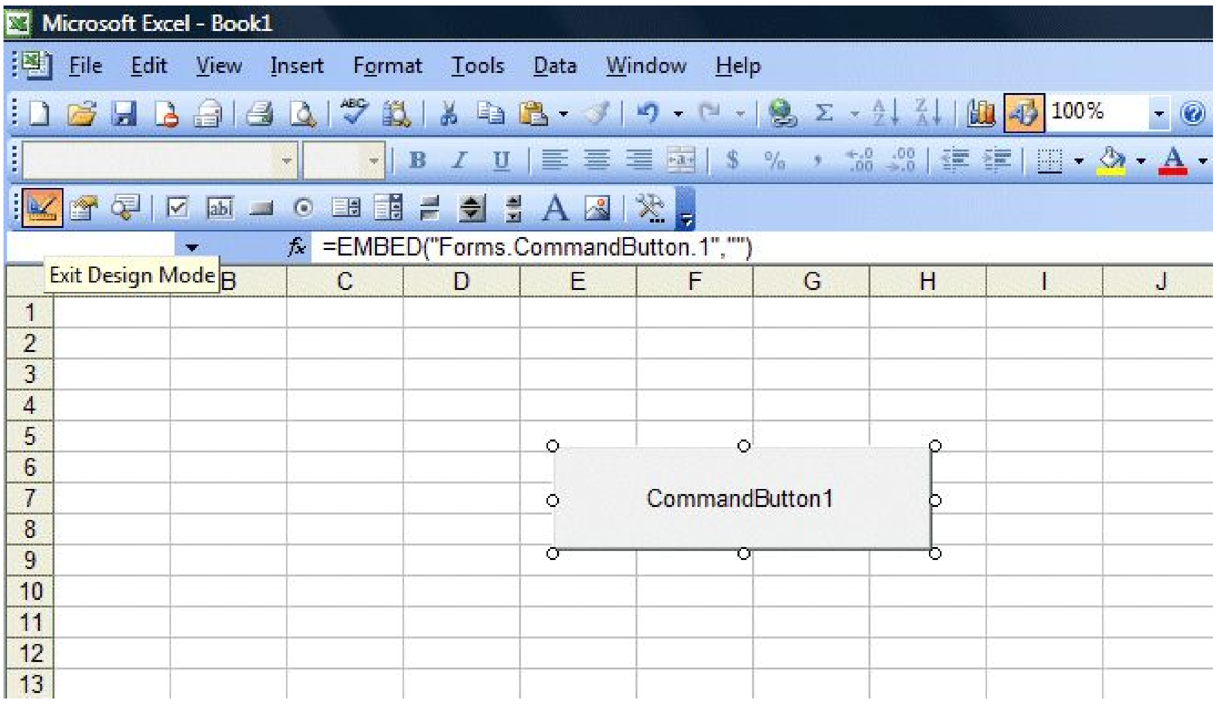


Figure 1.1 Displaying Control Toolbox in MS Excel.



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

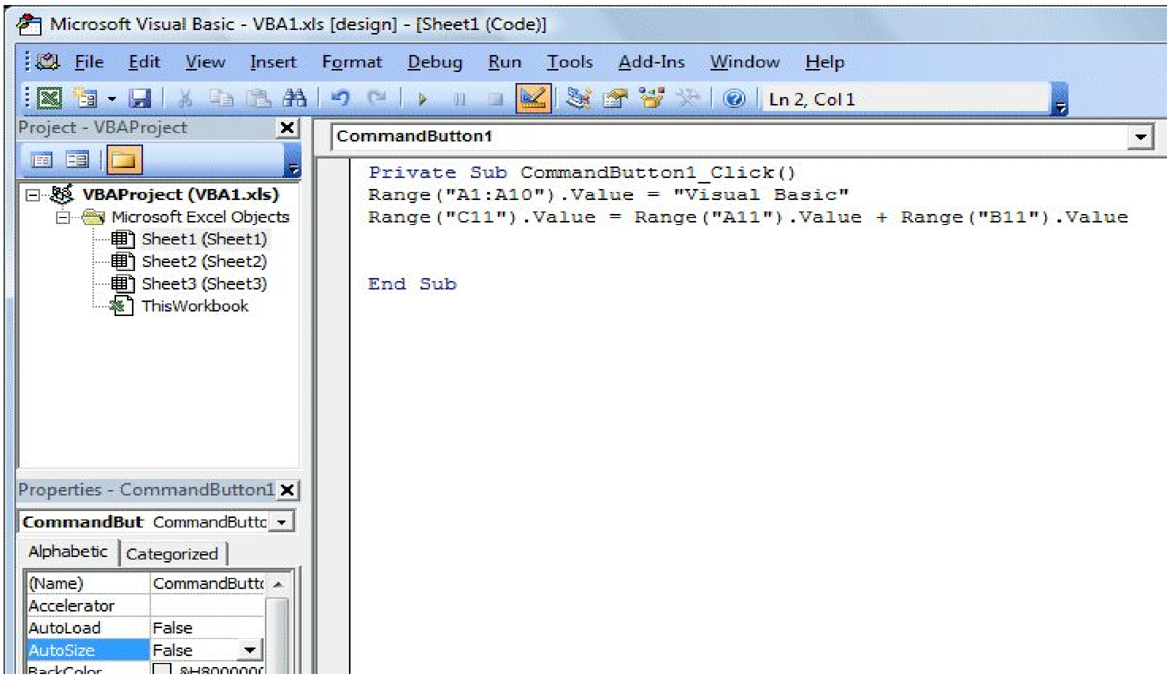
Figure 1.2 The Command Button in Design Mode

Next, click on the command button to launch the Visual Basic Editor. Enter the statements as shown in Figure 1.3. Enter the code as follows:

Example 1.1

```
Private Sub CommandButton1_Click ()
Range ("A1:A10).Value="Visual Basic "
Range ("C11").Value=Range ("A11").Value +Range ("B11").Value
End Sub
```

The first statement will populate cell A1 to cell A10 with the phrase "Visual Basic". The second statement adds the values in cell A11 and cell B11 and displays the sum in cell C11. To run the program, you must exit the Visual Basic Editor by clicking the Excel button on the far-left corner of the tool bar. When you are in the MS Excel environment, run the macro by clicking on the command button. The output is as seen in Figure 1.4.



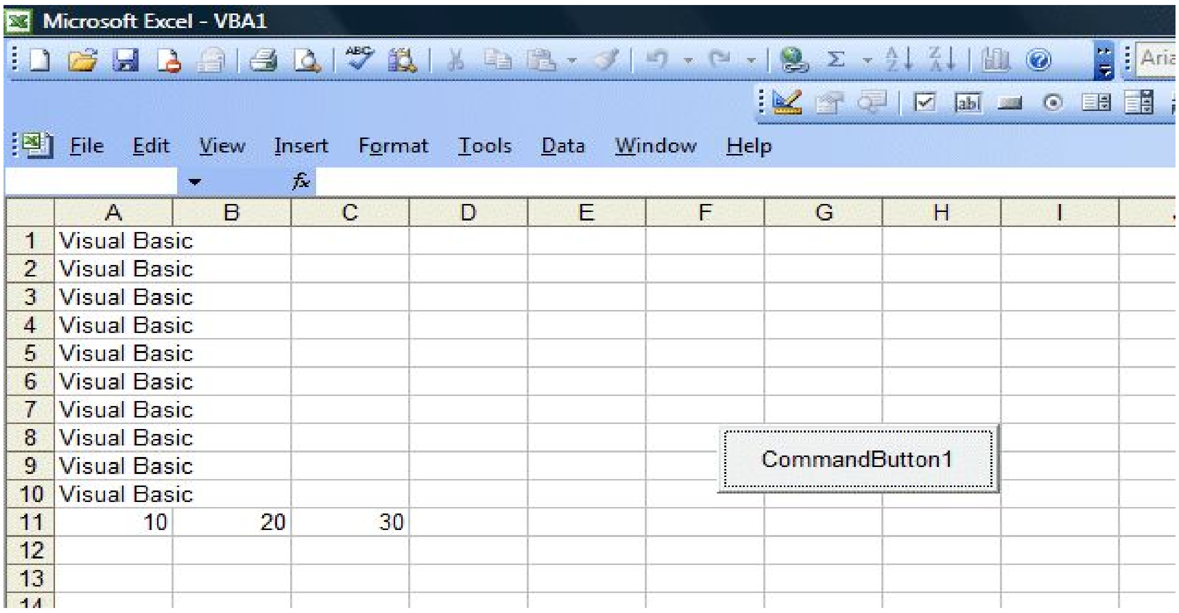
```
Private Sub CommandButton1_Click()
Range("A1:A10").Value = "Visual Basic"
Range("C11").Value = Range("A11").Value + Range("B11").Value


End Sub
```

(Name)	CommandButt
Accelerator	
AutoLoad	False
AutoSize	False

Figure 1.3: The Visual Basic Editor IDE in MS Excel

Running the above VBA will produce the following output.



	A	B	C	D	E	F	G	H	I
1	Visual Basic								
2	Visual Basic								
3	Visual Basic								
4	Visual Basic								
5	Visual Basic								
6	Visual Basic								
7	Visual Basic								
8	Visual Basic								
9	Visual Basic								
10	Visual Basic								
11	10	20	30						
12									
13									

Figure 1.4

1.3 The Excel VBA Code

Excel VBA code is similar to Visual Basic, which means you can use most of the syntaxes in Visual Basic to write VBA code. 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 1.2

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

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

Example 1.3

In this example, clicking the command button will fill cell A1 with the value of 100.

```
Private Sub CommandButton1_Click ()
Range ("A1") = 100
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. Variables are areas allocated by the computer memory to store data. Like the mailboxes, each variable must be given a name. To name a variable, you must follow a set of rules, as follows:

2.2 Variable Names

The following are the rules when naming the variables in VBA

- ❖ It must be less than 255 characters
- ❖ No spacing is allowed
- ❖ It must not begin with a number
- ❖ Period is not permitted

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

2.2.1 Numeric Data Types

Numeric data types are types of data that consist of numbers. In Excel VBA, 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 others, 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
```

You may also combine them 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 VBE 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 format as Example 2.1 above.

However, for the fixed-length string, you must use the syntax as shown below:

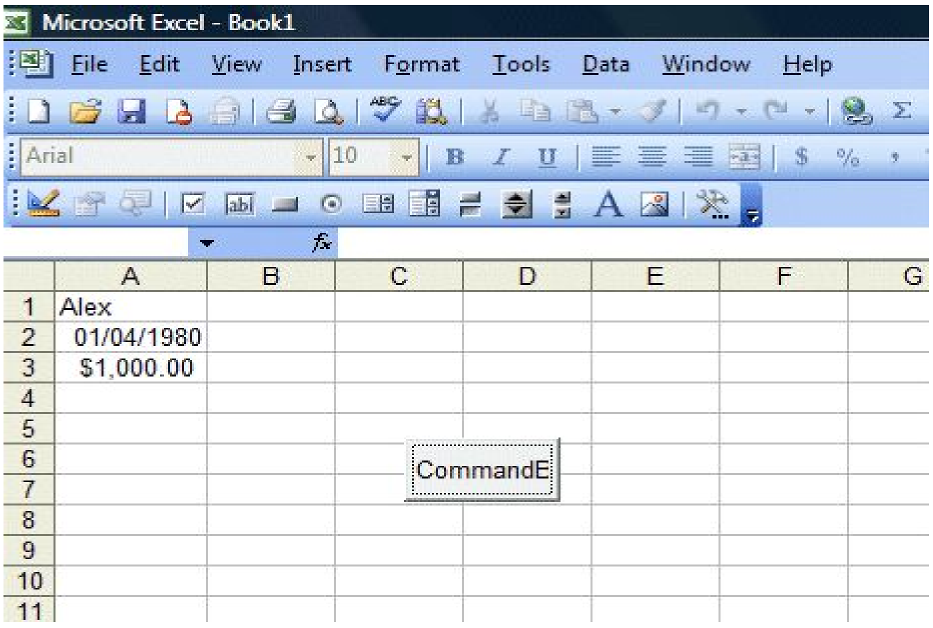
```
Dim VariableName as String * n
```

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

In this example, we declared three types of variables, namely the string, date and currency.

```
Private Sub CommandButton1_Click()
Dim YourName As String
Dim BirthDay As Date
Dim Income As Currency
YourName = "Alex"
BirthDay = "1 April 1980"
Income = 1000
Range("A1") = YourName
Range("A2") = BirthDay
Range("A3") = Income
End Sub
```



	A	B	C	D	E	F	G
1	Alex						
2	01/04/1980						
3	$1,000.00						

Figure 2.1 Output screen for Example 2.2

2.2 Option Explicit

The use of `Option Explicit` is to track errors in the usage of variable. For example, if we have committed a typo, Excel VBE 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 incorrect use of variable names due to typing errors, especially when the program gets larger. Using `Option Explicit` save time in debugging.

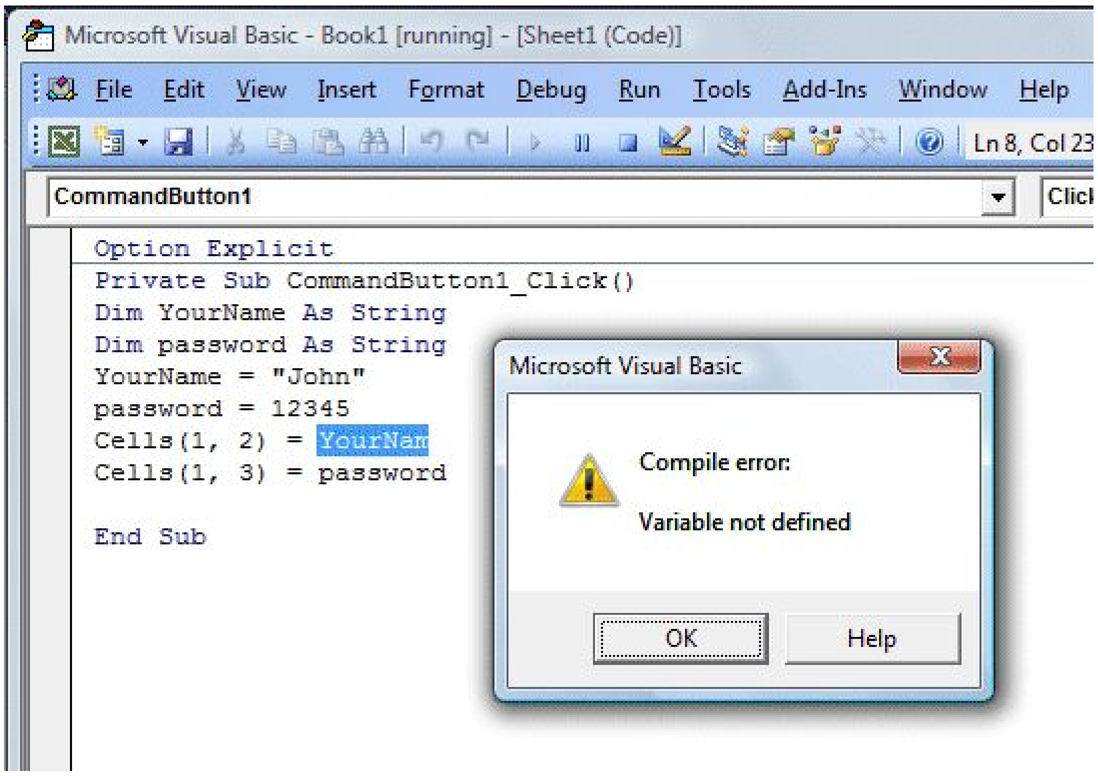
When `Option Explicit` is included in the program code, every variable must be declared using the `Dim` keyword. Any variable that is not declared or wrongly typed will produce the “Variable not defined” error. The error must be corrected before the program can continue to run.

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.


Microsoft Visual Basic - Book1 [running] - [Sheet1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 8, Col 23
CommandButton1
Click
OK
Help

```
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
```

```
Microsoft Visual Basic

Compile error:

Variable not defined
```

Figure 2.2: 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, the symbols for arithmetic operators are different from normal mathematical operators except for + and -. For example, multiplication is * and division are /. Besides. we must differentiate between / and \, where / is a normal division whilst \ is an integer division. Integer division \ discards the decimals. For example, 27\5 is 5. The Excel VBA arithmetic operators as shown in Table 2.3.

Table 2.3: Arithmetic Operators

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

Example 2.4

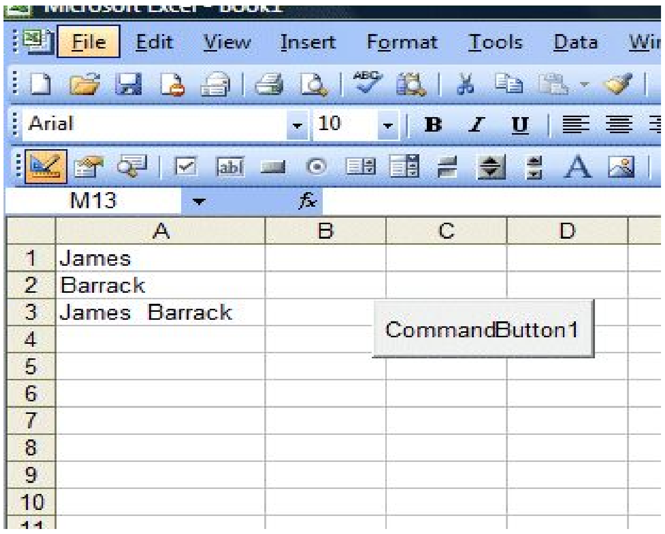
```
Option Explicit
Private Sub CommandButton1_Click ()
Dim number1, number2, number3 As Single
Dim total, average As Double
number1=Cells (1, 1).Value
number1=Cells (2, 1).Value
number3= Cells (3, 1).Value
 Total=number1+number2+number3
Average=Total/3
Cells (5, 1) =Total
Cells (6, 1) =Average
End Sub
```

In Example 2.4, three variables were declared as single and another two variables were declared as variants. Variant means the variable can hold any numeric data type. The program computes the total and average of the three numbers.

Example 2.5

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

In this example, the variable firstName and the variable secondName will store data entered into Cells(1,1) and cells(2,1) respectively. The variable yourName will be assigned the data by combining the first two variables. Finally, yourName is displayed in Cells (3, 1). You will notice that performing arithmetic operation on strings will result in the concatenation of the strings, as shown in figure 2.3 below. Names in A1 and A2 are joined and displayed in A3.



	A	B	C	D
1	James			
2	Barrack			
3	James Barrack			

Figure 2.3: Concatenation of Strings

Chapter 3 Message box and Input Box

Excel VBA has many built-in functions. Among these functions, there are two commonly used functions, `MsgBox()` and `InputBox()`. `InputBox()` allows the user to enter the data while `MsgBox()` displays the output to the user.

3.1 The MsgBox ( ) Function

The objective of the `MsgBox()` function is to produce a pop-up message box and prompt the user to click on a command button before he or she can continue. The message box format is as follows:

```
 yourMsg=MsgBox(Prompt, Style Value, Title)
```

The first argument, `Prompt`, displays the message in the message box. The `Style Value` determines what type of command button will appear in the message box. Table 3.1 lists the command buttons that can be displayed. The `Title` argument displays the title of the message box.

Table 3.1 Styles Values and Command Buttons

Style Value	Named Constant	Button Displayed
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 the 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 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 values, the corresponding named constants and the buttons.

Table 3.2: Returned Values and Command Buttons

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

Example 3.1

In this example, the message in cell (1,2) “Your first VBA program” will be displayed in the message box. As no named constant is specified, the message will simply display the message and the “OK” button.

```
Private Sub CommandButton1_Click()
Dim YourMsg As String
Cells(1, 2) = "Your first VBA program"
YourMsg = Cells(1, 2)
MsgBox YourMsg
End Sub
```

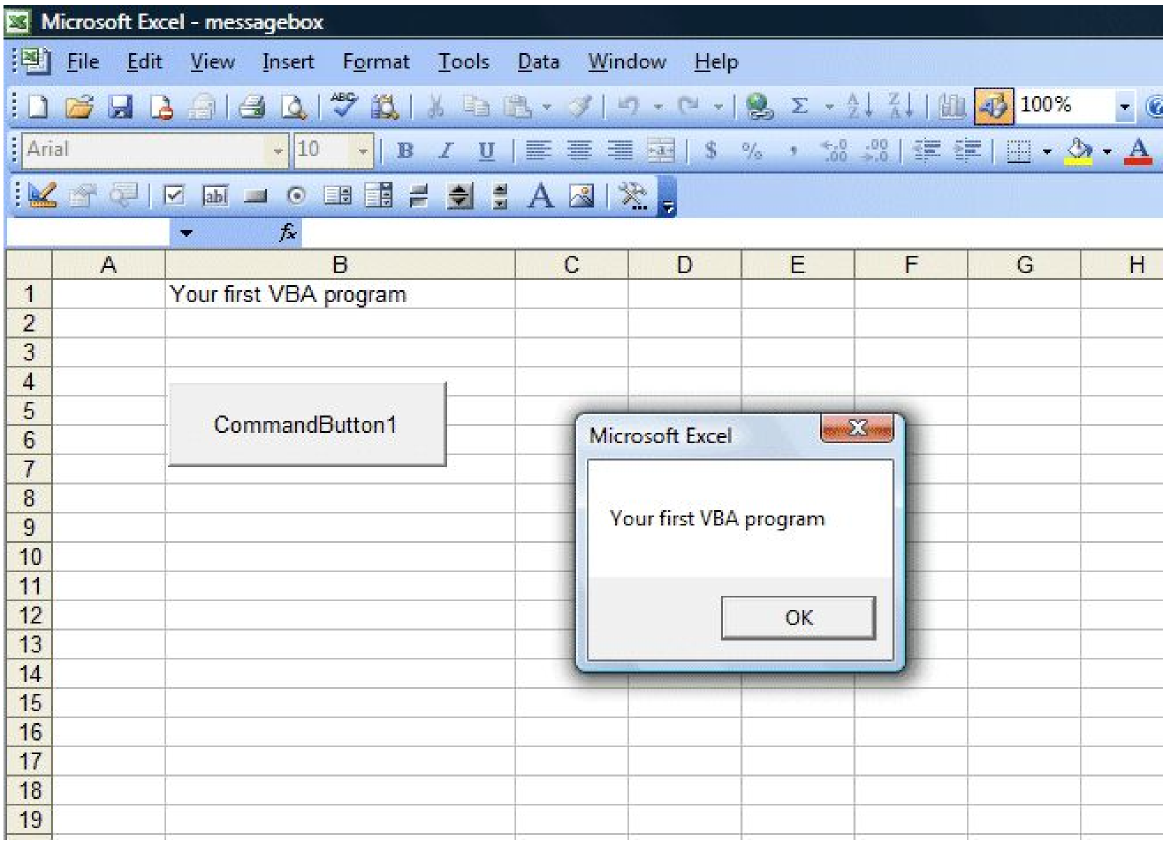

Microsoft Excel - messagebox
File Edit View Insert Format Tools Data Window Help
Your first VBA program
CommandButton1
Microsoft Excel
Your first VBA program
OK

Figure 3.1: Message box with the OK button

Example 3.2

In this Example, the named constant **vbYesNoCancel** is included as the second argument, therefore the message box will display the Yes, No and the Cancel buttons, as shown in Figure 3.2.

```
Private Sub CommandButton1_Click()
Dim YourMsg As String
Cells(1, 2) = "Your first VBA program"
YourMsg = Cells(1, 2)
MsgBox YourMsg, vbYesNoCancel
End Sub
```

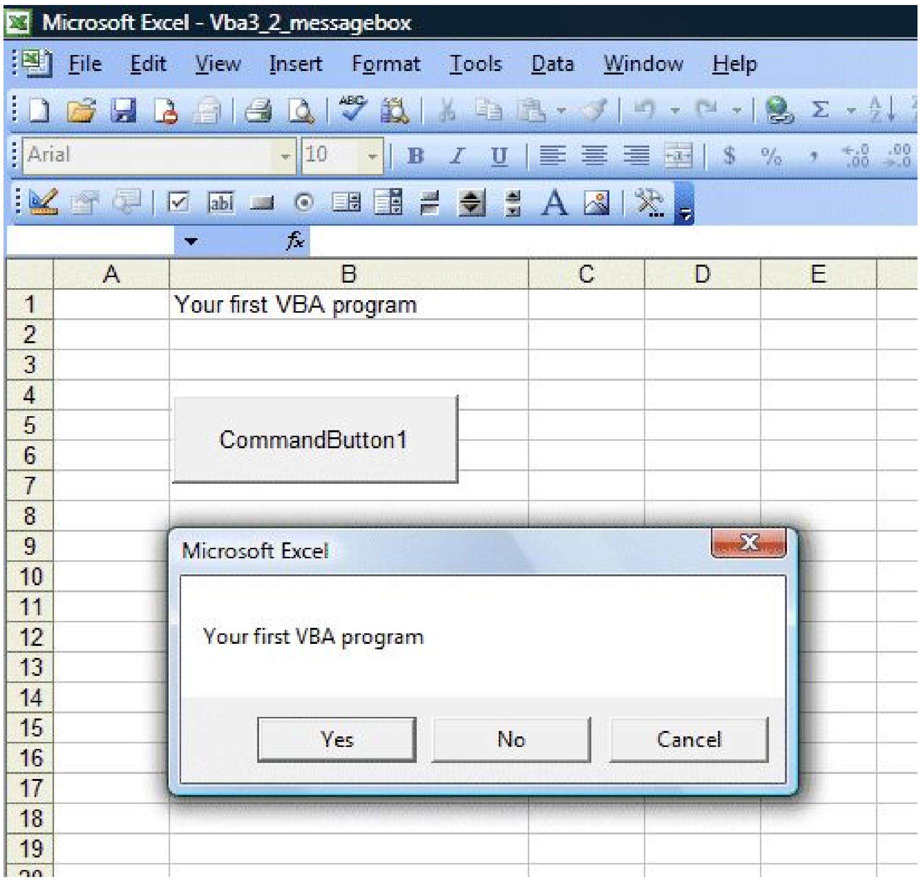


Figure 3.2: Message box with the Yes, No and Cancel buttons

To make the message box look more attractive, you can add an icon besides the message. There are four types of icons available in Excel VBE, as shown in Table 11.3.

Table 3.3

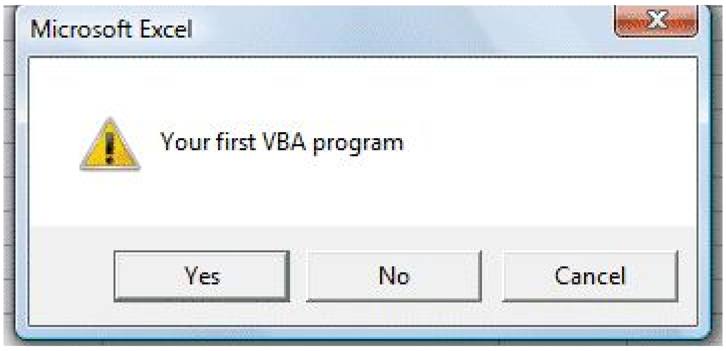
Value	Named Constant	Icon
16	vbCritical	⊗

32	vbQuestion	❓
48	vbExclamation	⚠
64	vbInformation	ℹ

Example 3.3

The code in this example is the same as Example 3.2, but the named `vbExclamation` is added as the third argument. The two-name constants can be joined using the "+" sign. The message box will now display the exclamation icon, as shown in Figure 3.3.

```
Private Sub CommandButton1_Click()
Dim YourMsg As String
Cells(1, 2) = "Your first VBA program"
YourMsg = Cells(1, 2)
MsgBox YourMsg, vbYesNoCancel + vbExclamation
End Sub
```



Your first VBA program

Figure 3.3: Message box with the exclamation icon.

You can even track which button is clicked by the user based on the returned values shown in Table 3.2. In Example 3.4, we use the `If...Then...Else` statement to determine which button has been clicked. You do not have to really understand the program logics at this stage, I will explain them in later chapters.

Example 3.4

```
Private Sub CommandButton1_Click()
Dim testMsg As Integer
testMsg = MsgBox("Click to Test", vbYesNoCancel + vbExclamation,
"Test Message")
If testMsg = 6 Then
Cells(1,1).Value = "Yes button was clicked"
ElseIf testMsg = 7 Then
Cells(1,1).Value = "No button was clicked"
Else
Cells(1,1).Value = "Cancel button was clicked"
End If
End Sub
```

3.2 The InputBox( ) Function

The `InputBox( )` is a function that displays an input box where the user can enter a value or a message in the form of text. The syntax is

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

myMessage is a variant data type but typically it is declared as a string, which accepts the message input by the users. The arguments are explained as follows:

- Prompt - The message displayed in the inputbox.
- 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 or the coordinates of the input box.

Example 3.5

```
Private Sub CommandButton1_Click()
Dim userMsg As String
```

```
userMsg = InputBox("What is your message?", "Message Entry Form",
"Enter your message here", 500, 700)
Cells(1,1).Value=userMsg
End Sub
```

When the user clicks on the OK button, the input box will appear, as shown in Figure 3.4 . Notice that the caption of the input box is "Message Entry Form" and the message is “What is your message?”. After the user entered the message and clicked the OK button, the message will be displayed in cell A1

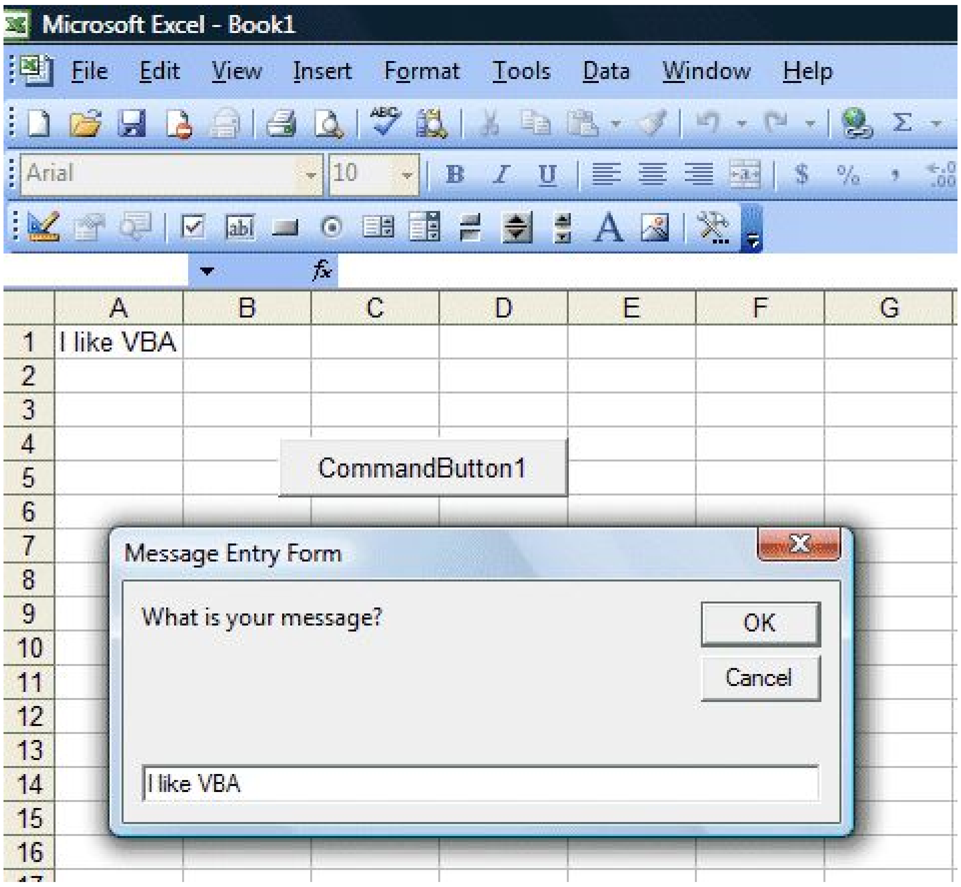


Figure 3.4: The input box

Chapter 4 Using If...Then…Else

In this chapter, you will learn how to create Excel VBA codes that can make decision when it processes inputs from the user and control the program flow. For example, we can write an Excel VBA macro that can instruct the computer to perform certain tasks and terminate the execution when a condition is met. Excel VBA uses the `If...Then...Else` statement and conditional operators as well as logical operators to control the program flow.

4.1 Conditional Operators

To control the Excel VBA 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 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 condition. 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 program, we shall use the `If...Then...Elseif` and `Else` statements . The syntax is as follows:

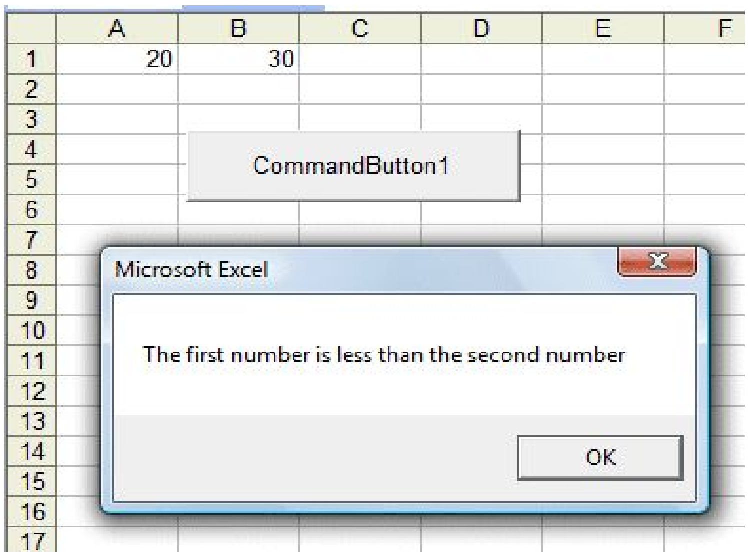
```
If conditions Then
VB expressions
ElseIf
VB expressions
Else
VB expressions
End If
```

Example 4.1

```
Private Sub CommandButton1_Click()
Dim firstnum, secondnum  As Single
firstnum = Cells(1,1).Value
secondnum = Cells(1,2).Value
If firstnum>secondnum Then
MsgBox " The first number is greater than the second number"
If firstnum<secondnum Then
MsgBox " The first number is less than the second number"
```

```
Else
MsgBox " The two numbers are equal "
End If
End Sub
```

In this example, the program compares the values in cells (1, 1) and cells (1, 2) then displays the appropriate comment in a message box. For example, If the first number is less than the second number, it will show the message “The first number is less than the second number”, as shown in Figure 4.1.



The first number is less than the second number

Figure 4.1

Example 4.2

```
Private Sub CommandButton1_Click()
Dim mark As Integer
Dim grade As String

mark = Int(Rnd * 100)
Cells(1, 1).Value = mark
If mark < 20 And mark >= 0 Then
grade = "F"
Cells(2, 1).Value = grade
ElseIf mark < 30 And mark >= 20 Then
grade = "E"
Cells(2, 1).Value = grade
ElseIf mark < 40 And mark >= 30 Then
```

```
grade = "D"
Cells(2, 1).Value = grade
ElseIf mark < 50 And mark >= 40 Then
grade = "C-"
Cells(2, 1).Value = grade
ElseIf mark < 60 And mark >= 50 Then
grade = "C"
Cells(2, 1).Value = grade
ElseIf mark < 70 And mark >= 60 Then
grade = "C+"
Cells(2, 1).Value = grade
ElseIf mark < 80 And mark >= 70 Then
grade = "B"
Cells(2, 1).Value = grade
ElseIf mark <= 100 And mark > -80 Then
grade = "A"
Cells(2, 1).Value = grade
End If
End Sub
```

In this example, we use the `Rnd` function to generate random numbers. To generate random integers between 0 and 100, we used the syntax `Int(Rnd*100)`. `Int(n)` is a Visual Basic function that returns the largest integer less than the number `n`. For example, when `Rnd=0.6543`, then `Rnd*100=65.43`, therefore `Int(65.43)=65`.

Using the statement `cells (1,1)`.Value=mark will display the value of 65 in `cells(1,1)`. Based on the mark in `cells(1,1)`, we use the `If.......Then....ElseIf` statements to display the corresponding grade in `cells(2,1)`. So, when you click on command button 1, it will generate a random number between 1 and 100 and displays it in `cells (1, 1)` and the corresponding grade in `cells (2,1)`. The output is shown in Figure 4.2.

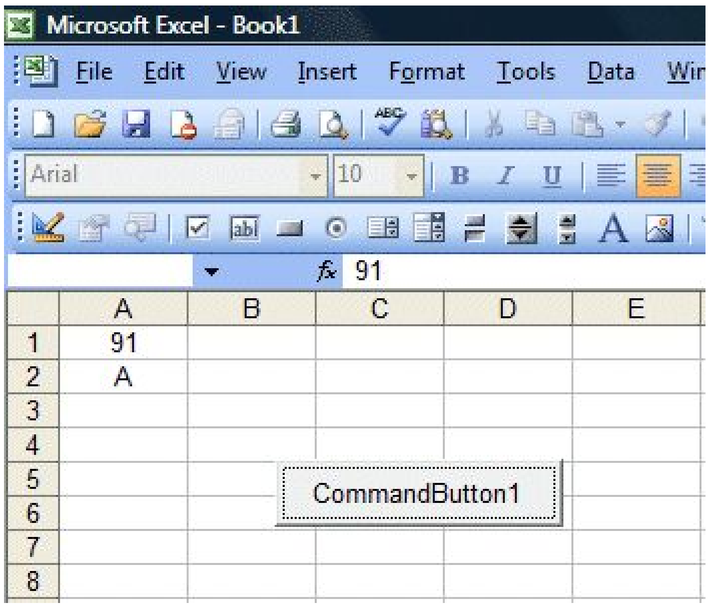


Figure 4.2

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 Mod 2` is not zero, it is an odd number; otherwise it is an even number.

Chapter 5 For……Next Loop

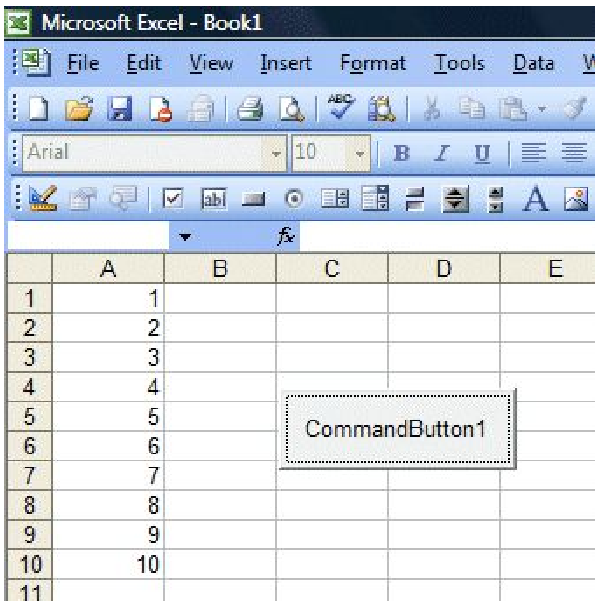
Looping is a feature of Excel VBA that makes repetitive work easier. There are two kinds of loops, the `For...Next loop` and the `Do...Loop`. In this chapter, we will discuss the `For....Next` loop. The syntax of a `For...Next` loop is

```
For counter=startNumber to endNumber (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).Value = i
Next
End Sub
```

In this example, executing the Excel VBA 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.



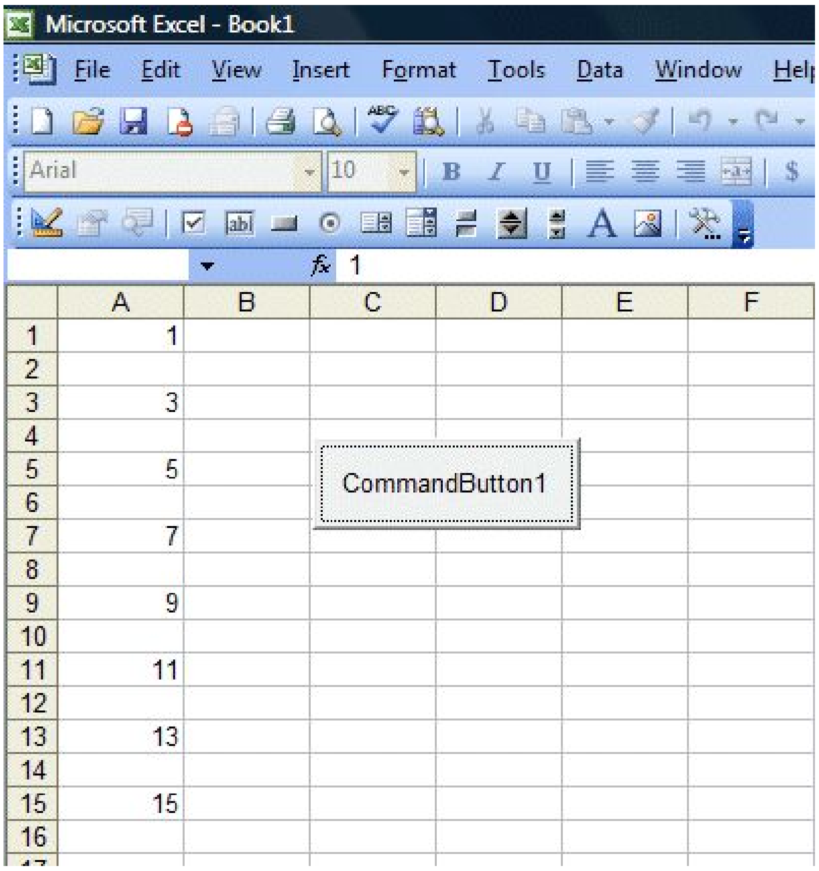
	A	B	C	D	E
1	1				
2	2				
3	3				
4	4				
5	5				
6	6				
7	7				
8	8				
9	9				
10	10				
11					

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).Value = i
Next
End Sub
```



fx 1

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

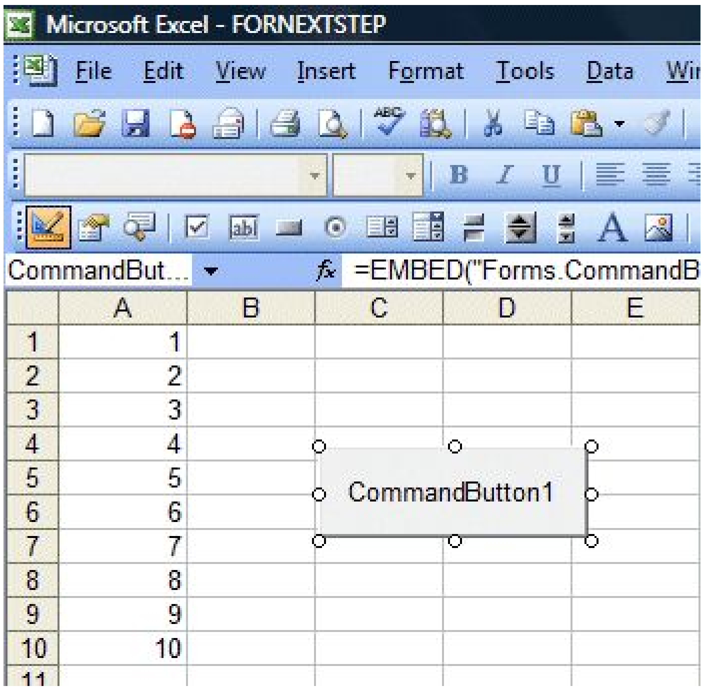
Figure 5.2: For...Next loop with step increment

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

Example 5.3

In this example, the program will stop once the value of I reaches the value of 10.

```
Private Sub CommandButton1_Click()
Dim i As Integer
For i = 1 To 15
Cells(i, 1).Value = i
If i >= 10 Then
Exit For
End If
Next i
End Sub
```


Microsoft Excel - FORNEXTSTEP
File Edit View Insert Format Tools Data
CommandBut...
=EMBED("Forms.CommandB
CommandButton1

	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				

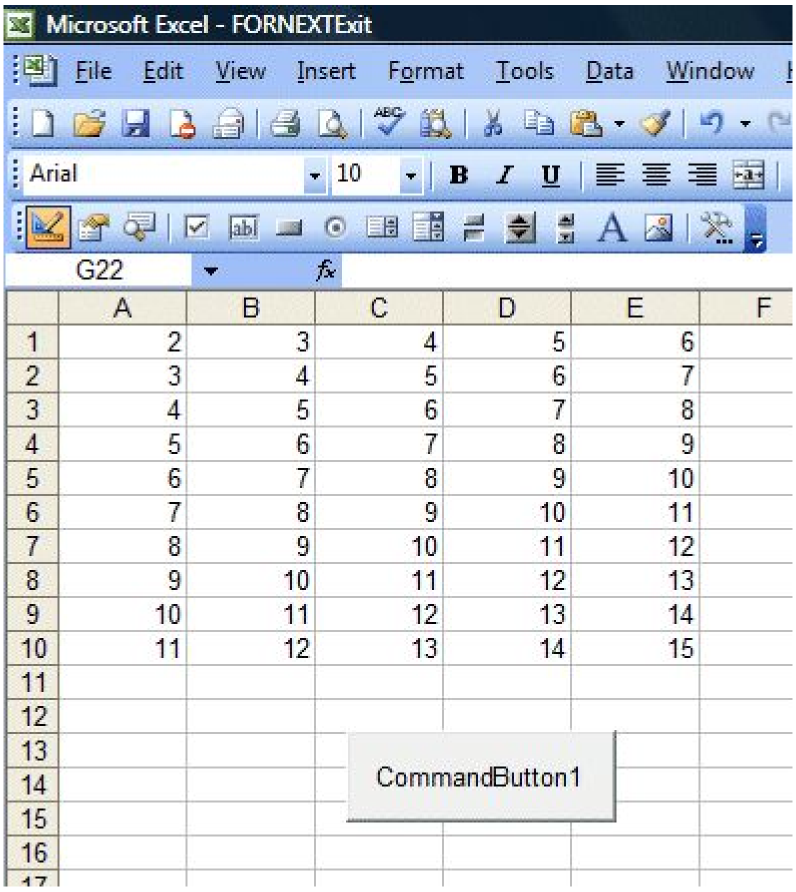
Figure 5.3: The output of Example 5.3

In previous examples, the `For...Next` loop will only fill up values through one column or one row only. To fill up an entire range of values across rows and columns, 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).Value = 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	

Figure 5.4: The output of Example 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's examine the code below:

```
Private Sub CommandButton1_Click()
Dim i, counter As Integer
```

```
For i = 1 To 20
If Cells(i, 2).Value > 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` and and the `If ...Then...Else` statements to control the program flow. 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). The output is shown in Figure 5.5.

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

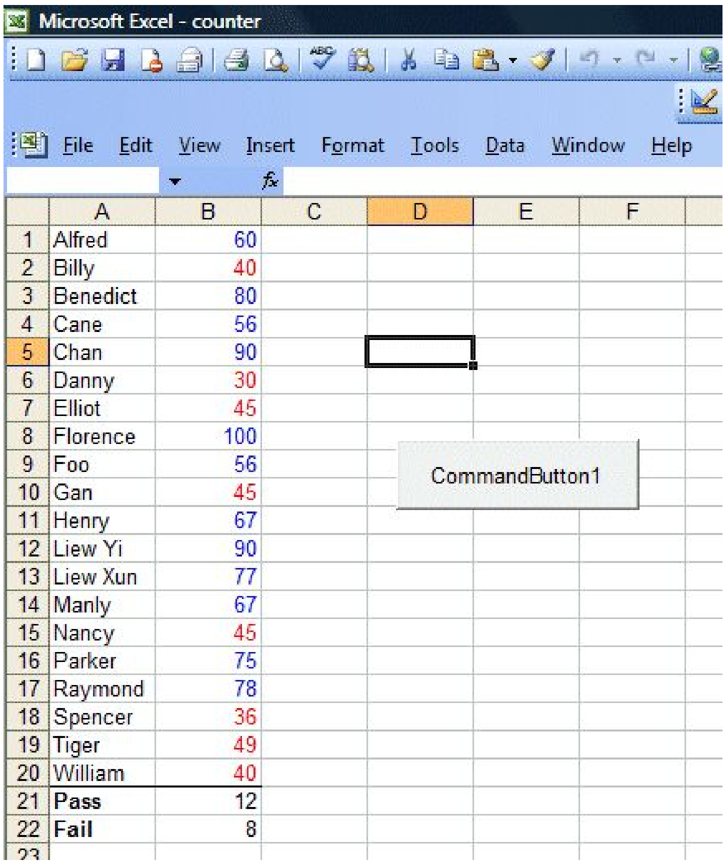
`Cells(i,j).Font.ColorIndex=k`

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 The Values of ColorIndex

Color Value	Color	Color Value	Color	Color Value	color	Color Value	Color
1	⬛	6	🟨	11	🟦	16	▨
2	⬜	7	🟪	12	🟩	17	▨
3	🟥	8	🟦	13	🟪	18	🟪
4	🟩	9	🟥	14	🟩	19	⬜

5	🟦	10	🟩	15	⬜	20	⬜



	A	B	C	D	E	F
1	Alfred	60				
2	Billy	40				
3	Benedict	80				
4	Cane	56				
5	Chan	90				
6	Danny	30				
7	Elliot	45				
8	Florence	100				
9	Foo	56				
10	Gan	45				
11	Henry	67				
12	Liew Yi	90				
13	Liew Xun	77				
14	Manly	67				
15	Nancy	45				
16	Parker	75				
17	Raymond	78				
18	Spencer	36				
19	Tiger	49				
20	William	40				
21	**Pass**	12				
22	**Fail**	8				
23						

Figure 5.5: The VBA counter