Index

Excel VBA Lesson 1: Introduction to Excel VBA

EXCEL VBA is the acronym for Visual Basic for Applications. It is an integration of the Microsoft's event-driven programming language Visual Basic with Microsoft Office Applications such as Microsoft Excel. By running EXCEL VBA within the Microsoft Office applications, you can build customized solutions and programs to enhance the capabilities of those applications. A lot of people might not realize that they can actually learn the fundamentals of Visual Basic programming without having a copy of Visual Basic professional. Why? Because there is a built-in Visual Basic Editor in Microsoft Excel, and you can use it to customize and extend the capabilities of MS Excel. The applications you build with MS Excel is called Visual Basic for Applications, or simply EXCEL VBA.

There are two ways which you could program a EXCEL VBA, one is to place a command button on the spreadsheet and start programming by clicking the command button, another one is to write Visual Basic functions inside the Visual Basic Editor. Let's start with the command button first. In order to place a command button on the spreadsheet, you need to click View on the MS Excel menu bar and then click on the toolbar and finally select the Control Toolbox to launch the control toolbox bar, as shown in Figure 1.1.

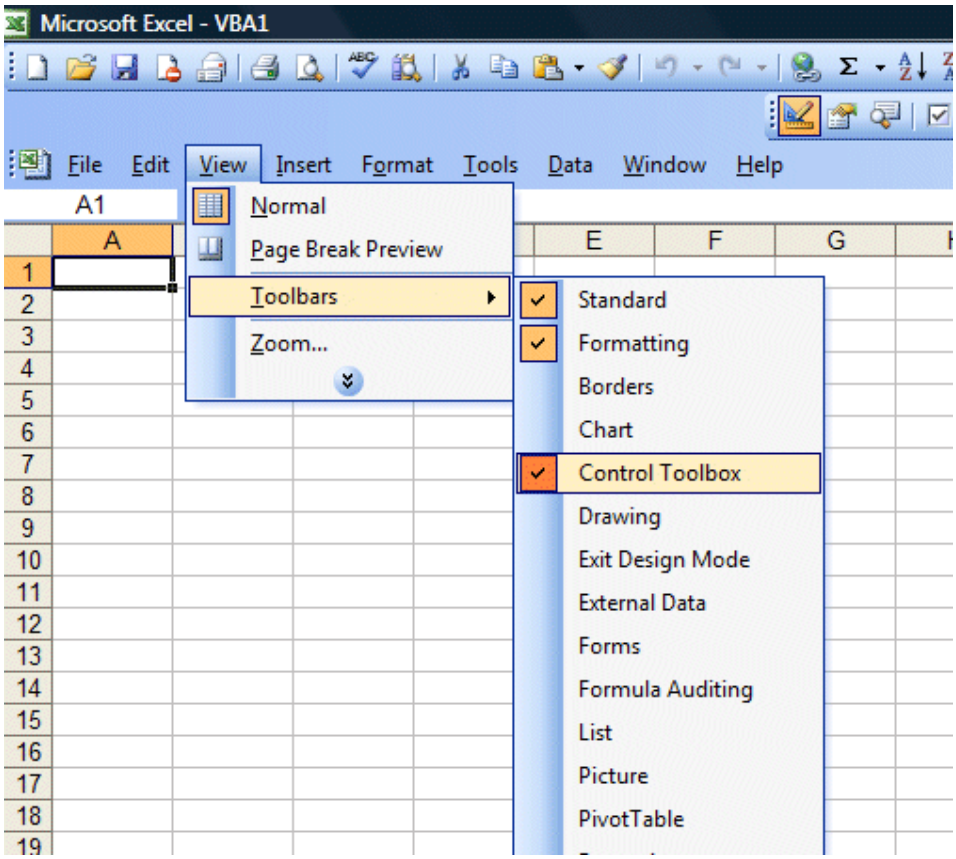


Figure 1.1

The control toolbox comprises various controls. In this chapter, you only use the command button. Now click on the command button and draw it on the spreadsheet, as shown in Figure 1.2

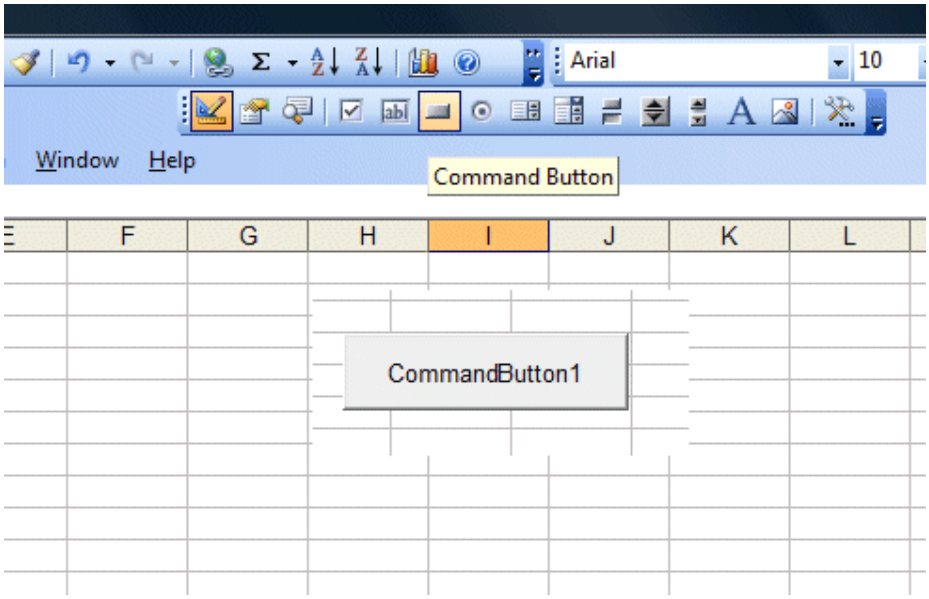


Figure 1.2

Next, click on the command button to bring up the Visual Basic Editor . In the Visual Basic Editor, enter the statements as shown in Figure 1.2. The first statement fills up cell A1 to cell A10 with the phrase “Visual Basic” while the second statement adds the value in cell A11 and cell B11 and then shows the sum in cell C11. It is that simple.

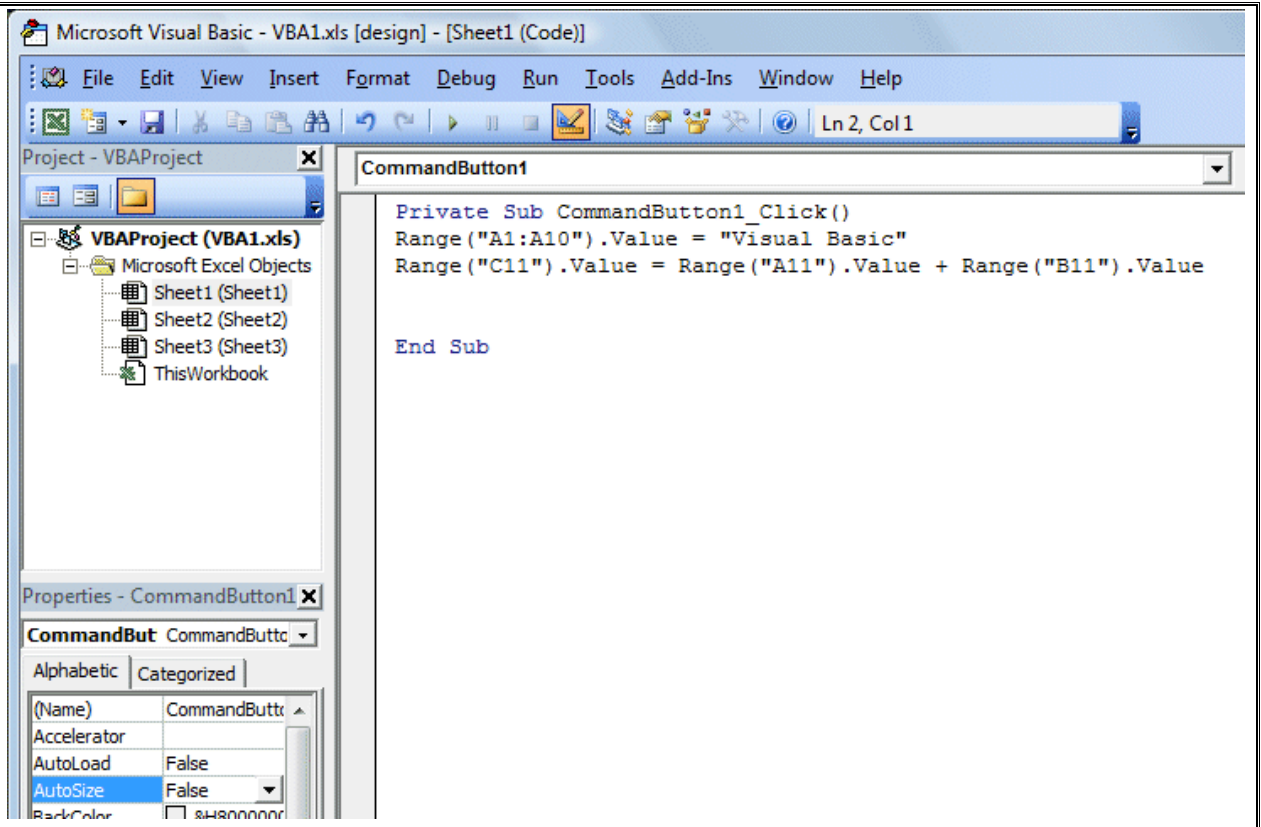


Figure 1.2

Run the macro and you can see output as shown in Figure 1.3 below:

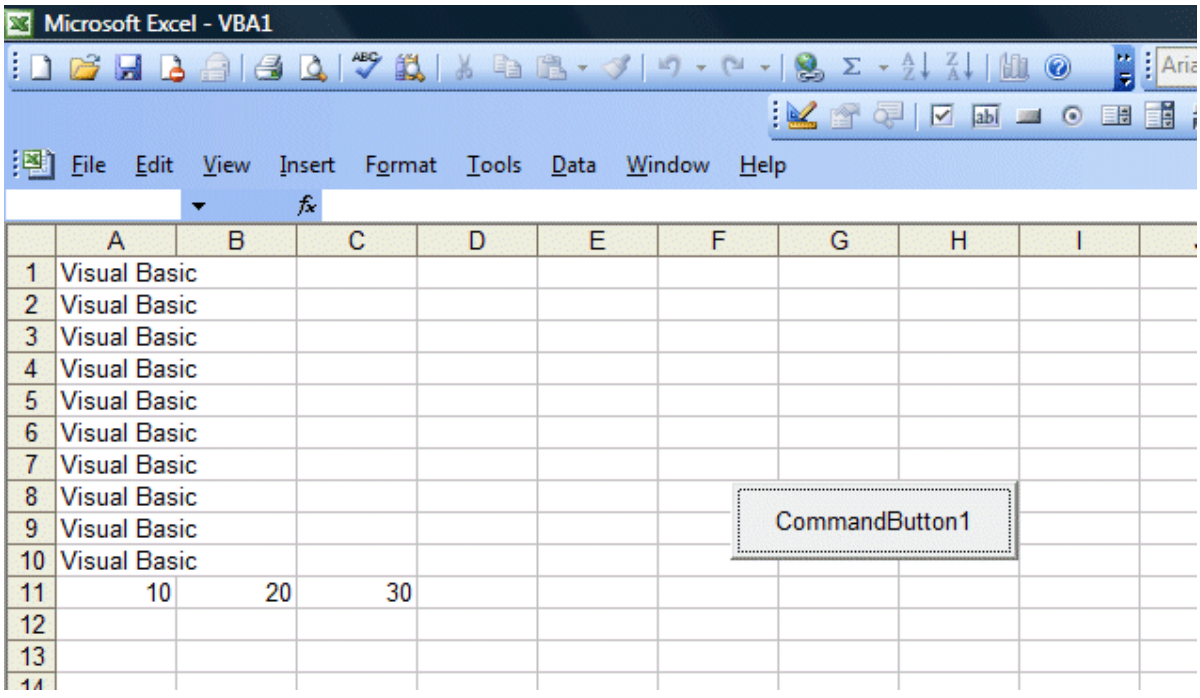


Figure 1.3

Excel VBA Lesson 2: Working with Variables in Excel VBA

2.1 The Concept of Variables in Excel VBA

Variables are like mail boxes in the post office. The contents of the variables changes every now and then, just like the mail boxes. In Excel VBA, variables are areas allocated by the computer memory to hold data. Like the mail boxes, each variable must be given a name. To name a variable in Excel VBA, you have to follow a set of rules, as follows:

a) Variable Names in Excel VBA

The following are the rules when naming the variables in Excel 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

Valid Name	Invalid Name
My_Car	My.Car
ThisYear	1NewBoy
Long_Name_Can_beUSE	He&HisFather *& is not acceptable
Group88	Student ID * Spacing not allowed

Table 2.1 : Example of valid and invalid variable names

b) Declaring Variables

In Excel VBA, one needs to declare the variables before using them by assigning names and data types. There are many Excel VBA data types, which can be grossly divided into two types, namely the numeric data types and non-numeric data types

i) Numeric Data Types

Numeric data types are types of data that consist of numbers, which can be computed mathematically with various standard operators such as add, minus, multiply, divide and so on. In Excel VBA, the numeric data are divided into 7 types, which are summarized in Table 2.2

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).

Table 2.2: Numeric Data Types

ii) Non-numeric Data Types

Non-numeric data types are summarized in Table 2.3

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

Table 2.3: Nonnumeric Data Types

You can 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 Text1 textbox. Other examples of implicit declaration are volume=8 and label="Welcome". On the other hand, for explicit declaration, variables are normally declared in the general section of the codes' windows 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 in one line, separating each variable with a comma, as follows:

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

If the data type is not specified, Excel VBA 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 have to use the format as shown below:

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

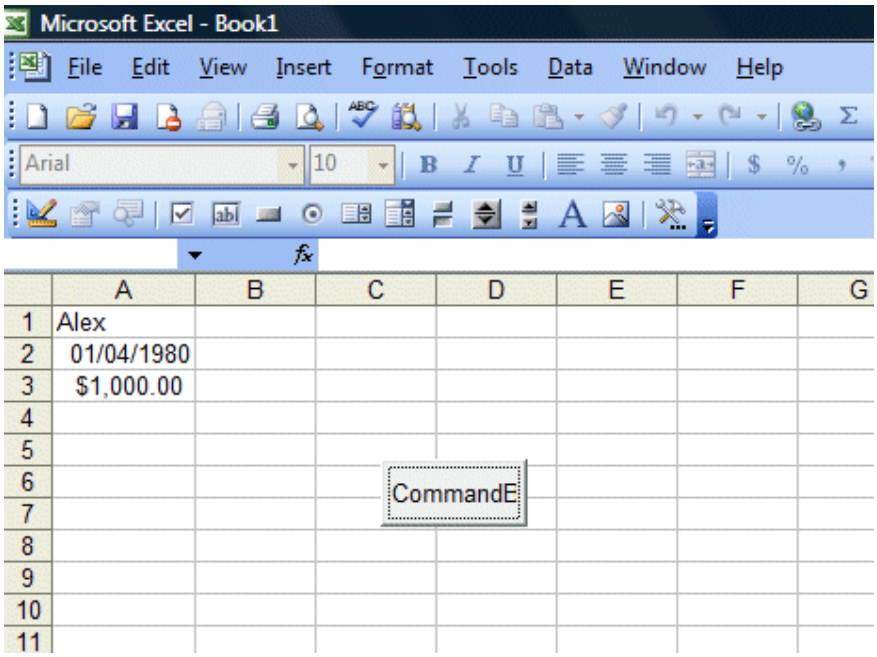
where n defines the number of characters the string can hold. For example, Dim yourName as String * 10 mean yourName can hold no more than 10 Characters.

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
```

The output screen of Example 2.2 is as follows:



Excel VBA Lesson 3: Excel VBA Message Box

In previous lesson we have shown that how you can create Excel VBA to display phrases in a range of cells and also perform arithmetic operations in MS Excel. Today, I shall demonstrate how we can display message boxes in a MS Excel worksheet . A message box normally act as a dialog box where users can interact with the computer, it is able to perform certain actions in response to what the user clicks or selects. The syntax for a message box in Excel VBA is as follows:

message=MsgBox(Prompt, Style Value,Title)

The first argument, Prompt, will display the message in the message box. The Style Value determines what type of command button will appear in the message box. . The Title argument will display the title of the message board. message 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 has to be declared as Integer data type in the procedure or in the general declaration section. Please refer to Lesson 10 of Visual Basic Tutorial for the detail listings of the Style Value as well as the returned value.

In this example, I create three command buttons which show different Options. I put in a bit of program codes in the last button which involve the use of If...Then...Elseif statements.

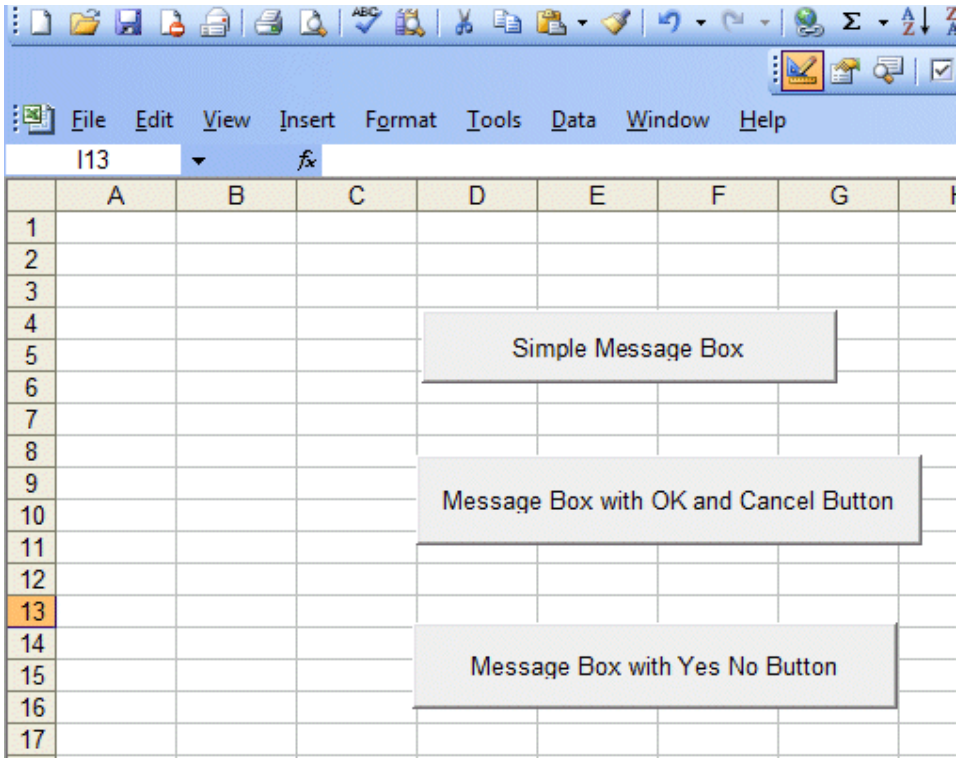


Figure 3.1

Double-click the top button and enter the following code:

```
Private Sub CommandButton1_Click()

MsgBox (“Welcome to VBA Programming”)

End Sub
```

By clicking the first message box, you will see the the message as shown in Figure 3.2

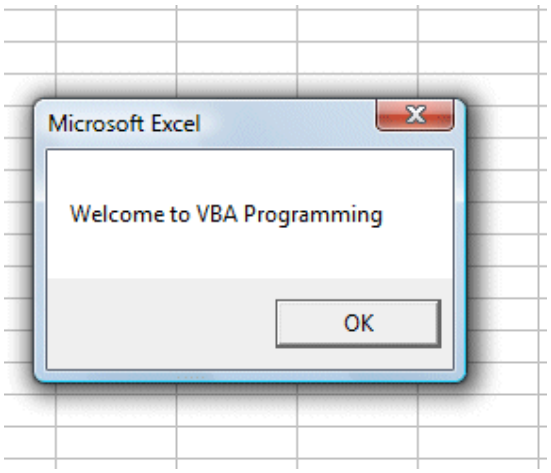


Figure 3.2

Now double-click on the second button and enter the following code:

```
Private Sub CommandButton1_Click()

Dim message As Integer
message = MsgBox(“Click Yes to Proceed, No to stop”, vbYesNoCancel, “Login”)
If message = 6 Then
Range(“A1″).Value = “You may proceed”
ActiveWorkbook.Activate
ElseIf message = 7 Then
ActiveWorkbook.Close
End If

End Sub
```

Click on the button and you shall see the following dialog. You will notice the Yes, No,Cancel buttons:

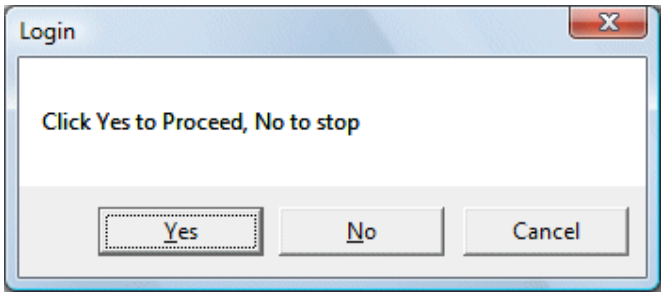


Figure 3.3

Now double-click on the second button and enter the following code:

```
Private Sub CommandButton3_Click()
Dim message As Integer
message = MsgBox("Click Yes to Proceed, No to stop", vbYesNo, "Login")
If message = 6 Then
Range("A1").Value = "You may proceed"
ActiveWorkbook.Activate
ElseIf message = 7 Then
ActiveWorkbook.Close
End If
End Sub
```

Click on the button and you would notice that the displayed message box comprises only the Yes and No button, as shown in Figure 3.4

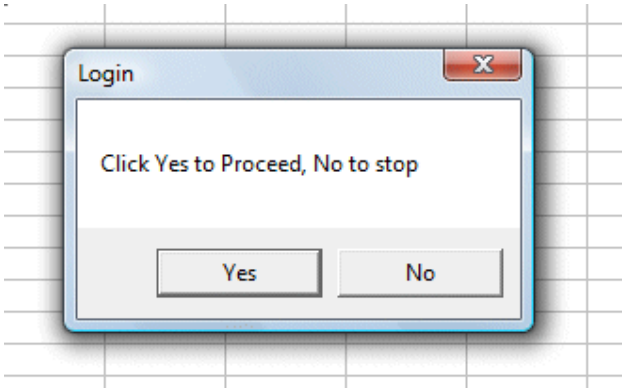


Figure 3.4

Excel VBA Lesson 4: Using If.....Then....Else

In this lesson, you will learn how to create Excel VBA code that can make decision . Decision making process is an important part of Excel VBA programming because it can help to solve practical problems intelligently so that it can provide useful feedback to the users.

In Excel VBA, decision making involves the use of the If..Then...Else syntax to process data and display the output based on certain conditions. To effectively control the VB program flow, we shall use **If...Then...Else** statement together with the conditional operators and logical operators. These operators are shown in Table 4.1 and Table 4.2 respectively.

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

Table 4.1:Conditional 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

Table 4.2:Logical Operators

We shall demonstrate the usage of If....Then...Else with the following example.In this program, we place the command button1 on the MS Excel spreadsheet and go into the VB editor by clicking on the button. At the VB editor, enter the program code as shown below.

We use the RND function to generate random numbers. In order to generate random integers between 0 and 100, we combined the syntax Int(Rnd*100). For example, when Rnd=0.6543, then Rnd*100=65.43, and Int(65.43)=65. Using the statement cells(1,1).Value=mark will place the value of 65 into cell(1,1).

Now, based on the mark in cells(1,1), I use the If.....Then....Elseif statements to put the corresponding grade in cells(2,1). So, when you click on command button 1, it will put a random number between 1 and 100 in cells(1,1) and the corresponding grade in cells(2,1).

The Code

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

The output is shown in Figure 4.1

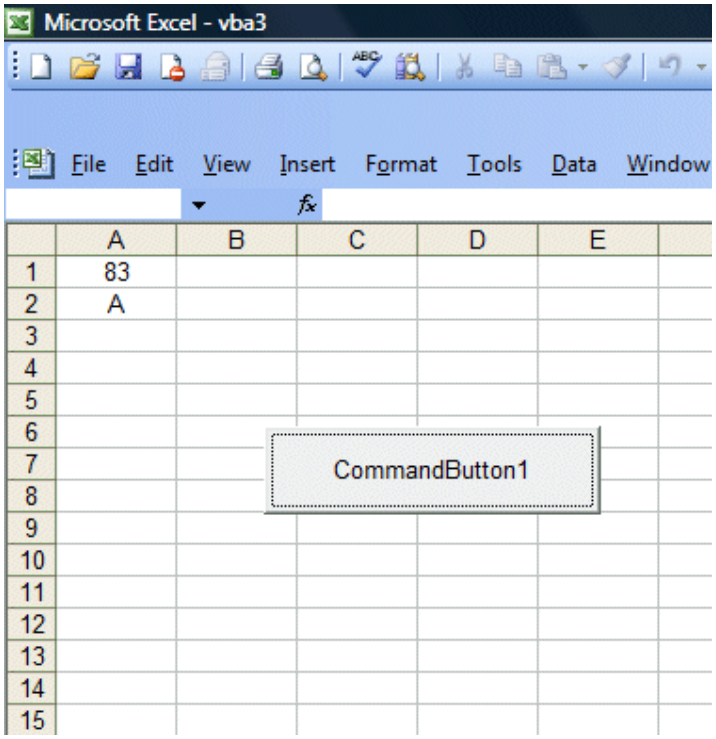


Figure 4.1

Excel VBA Lesson 5: Looping

Another procedure that involves decision making is looping. Excel VBA allows a procedure to be repeated many times until a condition or a set of conditions is fulfilled. This is generally called looping . Looping is a very useful feature of Excel VBA because it makes repetitive works easier. There are two kinds of loops in Visual Basic, the For.....Next loop, the While....Wend loop and the the Do...Loop . We shall deal with the For.....Next loop and the While....Wend loop first and we will deal with the Do....Loop in the next lesson.

5.1 One level For.....Next Loop

The one level For....Next Loop event procedure is written as follows:

For counter=startNumber to endNumber (Step increment)

One or more VB statements

Next

Example 5.1

In this Excel VBA program, you place the command button 1 on the spreadsheet then click on it to go into the Visual Basic editor. When you click on the button , the Excel VBA program will fill 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 row and j represent column.

The Code

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

The Output is shown in Figure 5.1

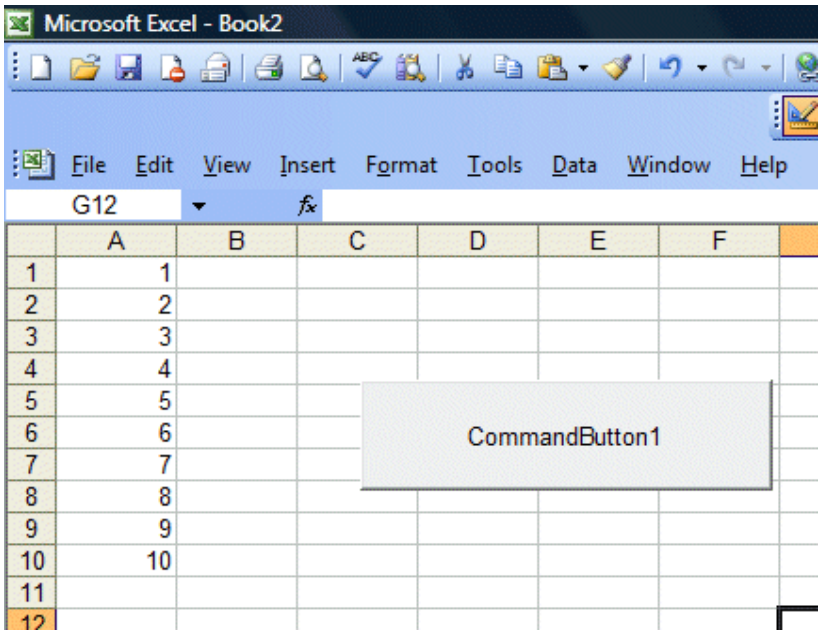


Figure 5.1

Example 5.2

For……..Next loop with step increments. In this example, the number increases by 2 at a time. The resulting series is 1,3,5,7,9 displayed in Cells(1,1), Cells(3,1),Cells(5,1), Cells(7,1) and Cells(9,1) respectively.

The code

```
Private Sub CommandButton1_Click()
Dim i As Integer
For i = 1 To 10  Step 2
Cells(i, 1).Value = i
Next
End Sub
```

The Output

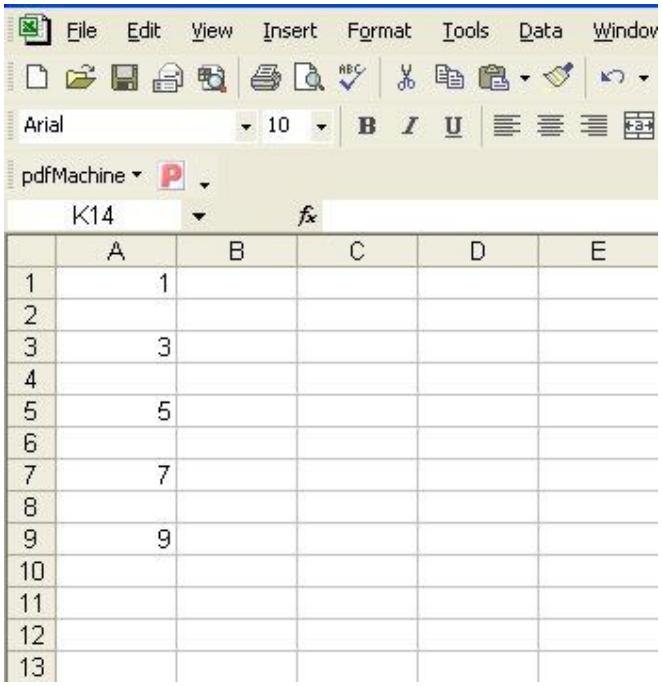


Figure 5.2

5.2 Nested For….Next Loops

Nested For…….Next loops means there more more than one For…Next Loops are nested within the first level For…..Next loop. The structure is as follows:

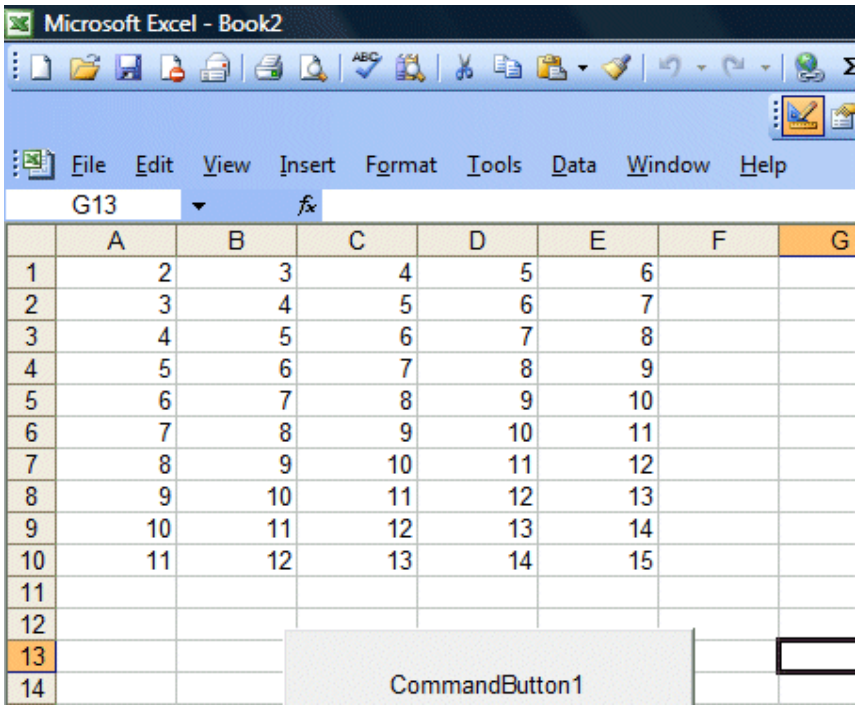
For counter=*startNumber1* to *endNumber1*
For counter=*startNumber2* to *endNumber2*
For counter=*startNumber3* to *endNumber3*
One or more VB statements
Next
Next
Next

Example 5.3

In this example , we use nested loop to put the values of i+j from cells(1,1),cells(1,2),cells(1,3),cells(1,4),cells(1,5) ……..until cells(10,5). The code and output are shown below.

```
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
Next
End Sub
```

The Output



	A	B	C	D	E	F	G
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							

Figure 5.3

5.3 While....Wend Loop

The structure of a While....Wend Loop is very similar to the Do Loop. it takes the following form:

```
While condition
Statements
Wend
```

The above loop means that while the condition is not met, the loop will go on. The loop will end when the condition is met. Let's examine the program listed in example 5.4.

Example 5.4

In this example, the Excel VBA program will compute the square of numbers from 1 to 15 and displays them from Cells(1,1) to Cells(15,1)

The Code

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

i = 1
While i <= 15

Cells(i, 1) = i ^ 2
i = i + 1
Wend

End Sub
```

The Output

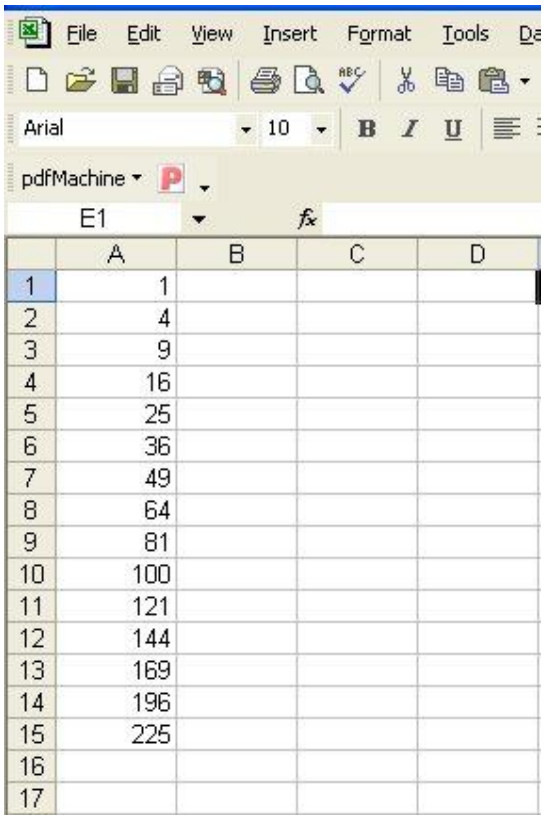


Figure 5.4

Excel VBA Lesson 6: DO……..LOOP

In the previous lesson, you have learned how to use the **For……..Next** loop and the **While…..Wend** loop to execute a repetitive process. In this lesson, you will learn about another looping method know as the **Do** Loop. There are four ways you can use the Do Loop, as shown below:

(i) Do……..Loop While

(ii) Do until……..Loop

(iii) Do while……..Loop

(iv) Do……..Loop until

We shall illustrate the four structures of Do Loops in the following examples:

Example 6.1

Arranging numbers in ascending order

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

End Sub
```

In this example, the program will keep on 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).

The Output

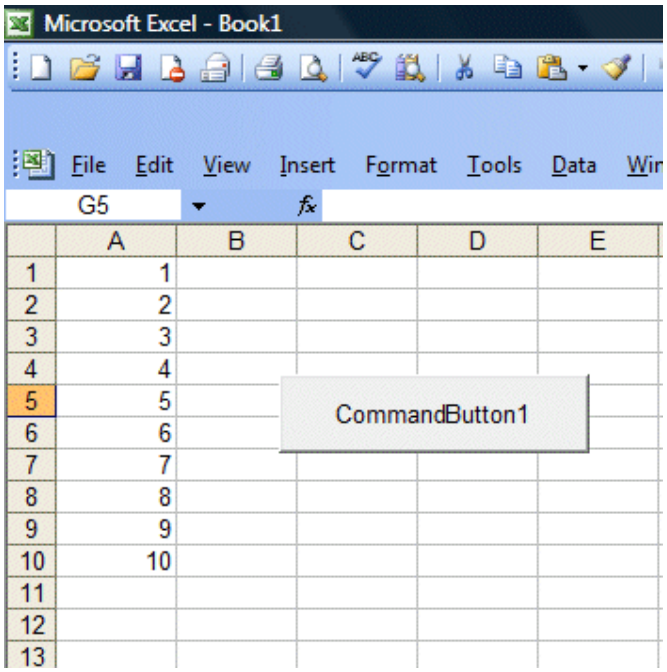


Figure 6.1

Example 6.2

Arranging numbers in descending order

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

In this example, the program will keep on 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).

The Output

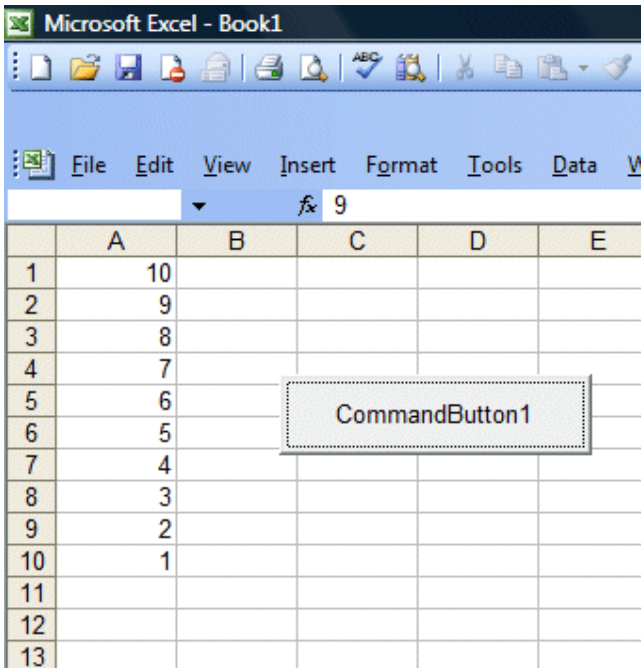


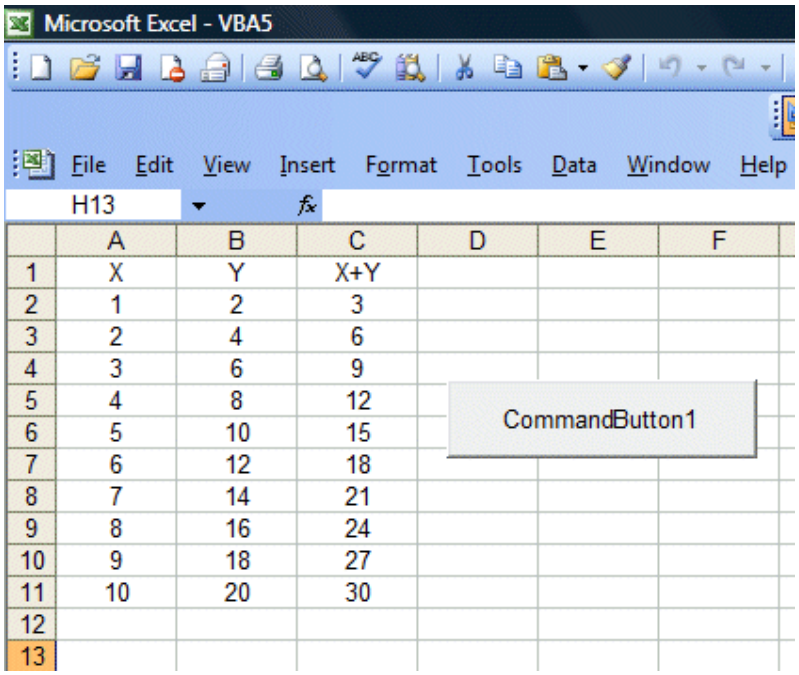
Figure 6.2

Example 6.3

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

'To set the alignment to center
Range("A1:C11").Select
With Selection
.HorizontalAlignment = xlCenter
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
```

In this example, the program will display the values of X in cells(1,1) to cells(11,1). The value of Y is $X^2$and the values are display 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)

The Output



	A	B	C	D	E	F
1	X	Y	X+Y			
2	1	2	3			
3	2	4	6			
4	3	6	9			
5	4	8	12			
6	5	10	15			
7	6	12	18			
8	7	14	21			
9	8	16	24			
10	9	18	27			
11	10	20	30			
12						
13						

Figure 6.3

Excel VBA Lesson 7: Using Select Case……..End Select

In lesson 4, we have learned how to handle codes in Excel VBA that involves decision making process through the use the conditional statement If....Then....Else. However, for multiple options or selections programs, the If...Then...Else structure could become too bulky and difficult to debug if problems arise. Fortunately, Excel VBA provides another way to handle complex multiple choice cases, that is, the Select Case.....End Select decision structure. The general format of a Select Case...End Select structure is as follow:

```
Select Case variable
Case value 1
Statement
Case value 2
Statement
Case value 3
Statement
.
.
.
.
Case Else

End Select
```

We shall demonstrate the use of Select Case...End Select in the following examples:

Example 7.1

In this program, we want to automatically assign a remark in relation to a certain examination grade. For example, if the grade is A, we shall assign remark as High distinction, A- for distinction and so forth. To enter the code, start MS Excel then place a button and a label onto the spreadsheet. The label is for displaying the remark. Cells(1,1) is for the user to enter the grade.

The code:

```
Private Sub CommandButton1_Click()
Dim grade As String
grade = Cells(1, 1)
```

```
Select Case grade

Case “A”
label1.Caption = “High Distinction”

Case “A-”
label1.Caption = “Distinction”

Case “B”
label1.Caption = “Credit”

Case “C”
label1.Caption = “Pass”

Case Else
label1.Caption = “Fail”

End Select
End Sub
```

The Output

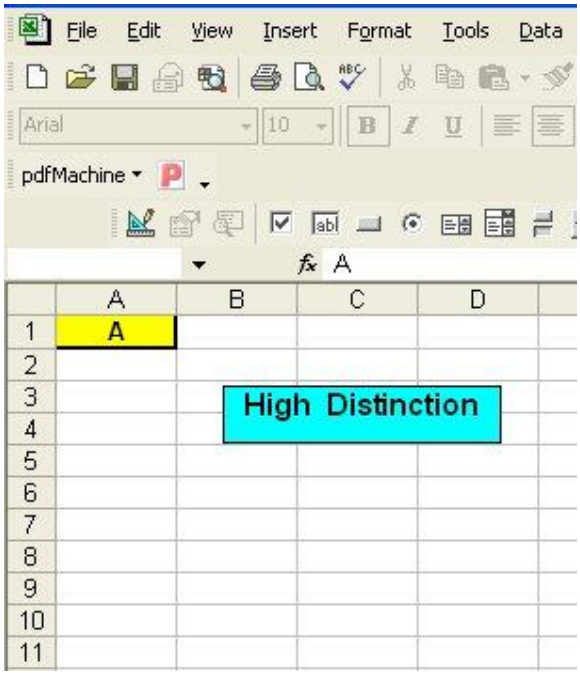


Figure 7.1

Example 7.2

In the this example, we shall show you how to process the grades of students according to the marks given. For example, if a student's mark is 85, the corresponding grade is A and if the mark is 20 the grade will be F and so forth.

The code

```
Private Sub CommandButton1_Click()
Dim mark As Single
Dim grade As String
mark = Cells(1, 1).Value
'To set the alignment to center
Range("A1:B1").Select
With Selection
.HorizontalAlignment = xlCenter
End With
Select Case mark
Case 0 To 20
grade = "F"
Cells(1, 2) = grade
Case 20 To 29
grade = "E"
Cells(1, 2) = grade
Case 30 To 39
grade = "D"
Cells(1, 2) = grade
Case 40 To 59
grade = "C"
Cells(1, 2) = grade
Case 60 To 79
grade = "B"
Cells(1, 2) = grade
Case 80 To 100
grade = "A"
Cells(1, 2) = grade
Case Else
grade = "Error!"
Cells(1, 2) = grade
End Select
End Sub
```

Explanation:

To set the cell align alignment to center, we use the following procedure:

```
Range("A1:B1").Select
With Selection
.HorizontalAlignment = xlCenter
End With
```

We can use the statement case *value1* to *value 2* to specify the range of values that fulfill the particular case.

You should also include the error case where the values entered are out of the range or invalid. For example, if the examination mark is from 0 to 100, then any value out of this range is invalid. In this program, I use case else to handle the error entries.

Figure 7.2 illustrates the output of this example.

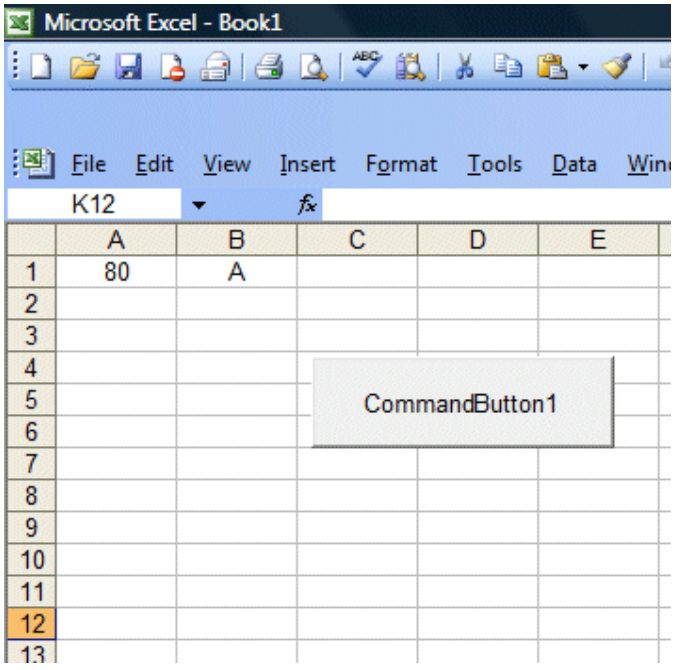


Figure 7.2

Excel VBA Lesson 8: Introduction to Excel VBA Functions

In Excel VBA, a function is similar to a procedure but the main purpose of the function is to accept a certain input from the user and return a value which is passed on to the main program to finish the execution. There are two types of functions, the built-in functions (or internal functions) and the functions created by the programmers, or simply called user-defined functions. We shall deal with built-in functions in this lesson and the user-defined functions in the next lesson.

The first built-in function that we have already learned and familiar with its usage is the Message Box. We are not going to repeat here but we shall take a look at its syntax once more, i.e.

message=MsgBox(Prompt, Style Value,Title)

Now we shall examine the next commonly used function in Excel VBA, it is none other than the InputBox function.

8.1 The InputBox( ) Function

An InputBox( ) function displays a message box where the user can enter a value or a message in the form of text. The format is

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

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

- Prompt – The message displayed normally as a question asked.
- 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 to the message he wish to key in.
- x-position and y-position – the position or the coordinate of the input box.

Example 8.1

In this example, we insert a label and a command button into the MS Excel spreadsheet. Double click on the command button and enter the Excel VBA code as follows:

```
Private Sub CommandButton1_Click()
Dim userMsg As String
userMsg = InputBox("What is your message?", "Message Entry Form", "Enter your
messge here", 500, 700)
If userMsg <> "" Then
Label1.Caption = userMsg
Else
Label1.Caption = "No Message"
End If
End Sub
```

When you run the Excel VBA macro, the following input box will pop up, as shown in Figure 8.1

Figure 8.1

Upon entering your message and click OK, the message will appear on the label, as shown in Figure 8.2

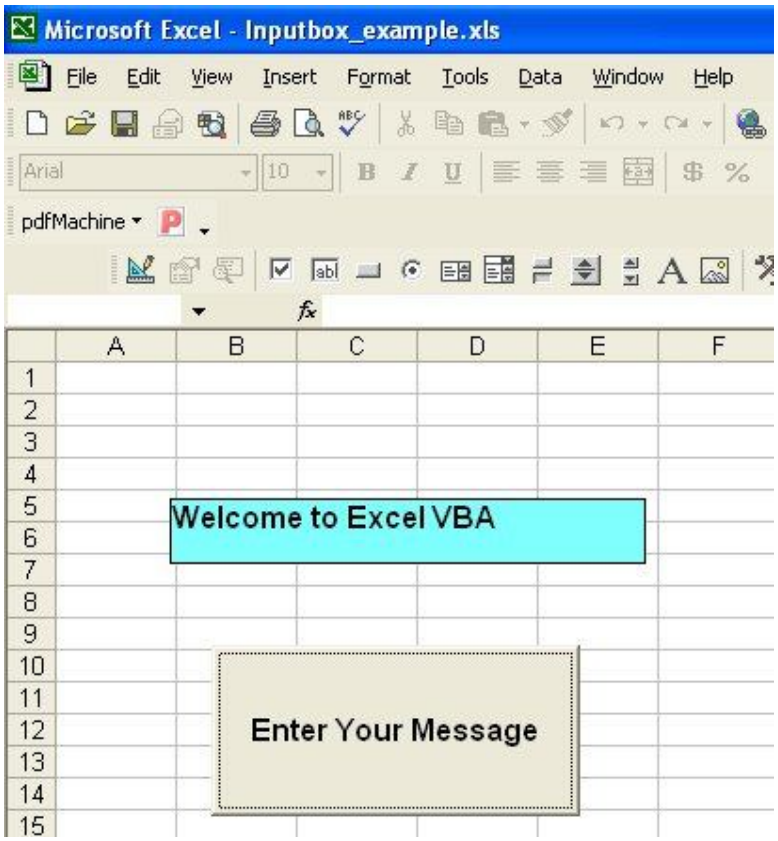


Figure 8.2

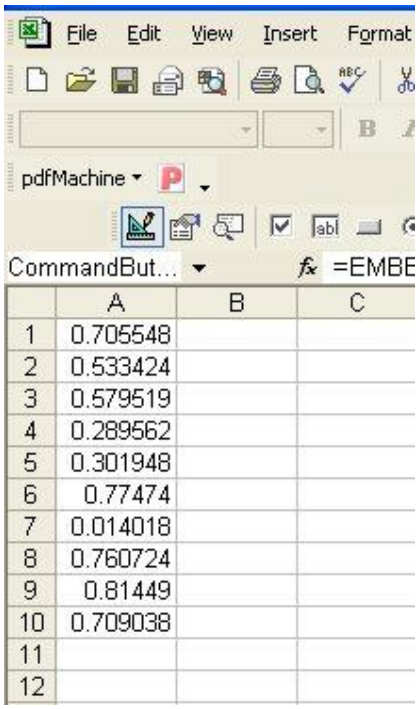
8.2 The RND function

Rnd is very useful when we deal with the concept of chance and probability. The Rnd function returns a random value between 0 and 1. In Example 8.2(a). When you run the program, you will get an output of 10 random numbers between 0 and 1 occupying Cells(1,1) to Cells(10,1) respectively.

Example 8.2(a)

```
Private Sub CommandButton1_Click()
For x = 1 To 10
Cells(x, 1) = Rnd
Next x
End Sub
```

The output



	A	B	C
1	0.705548		
2	0.533424		
3	0.579519		
4	0.289562		
5	0.301948		
6	0.77474		
7	0.014018		
8	0.760724		
9	0.81449		
10	0.709038		
11			
12			

Figure 8.3

Random numbers in its original form are not very useful in programming until we convert them to integers. For example, if we need to obtain a random output of 6 random integers ranging from 1 to 6, which make the program behave as a virtual die, we need to convert the random numbers using the format **Int(Rnd*6)+1**.

In Example 8.2(b), Int(Rnd*6) will generate a random integer between 0 and 5 because the function **Int** truncates the decimal part of the random number and returns an integer. After adding 1, you will get a random number between 1 and 6 every time you click the command button. For example, let say the random number generated is 0.98, after multiplying it by 6, it becomes 5.88, and using the integer function Int(5.88) will convert the number to 5; and after adding 1 you will get 6.

In this example, you place a command button and change its caption to 'roll die'. You also need to insert a label into the form and clear its caption at the designing phase and make its font bigger and bold. Then set the border value to 1 so that it displays a border; and after that set the alignment to center. The statement Label1.Caption=Num means the integer generated will be displayed as the caption of the label.

Example 8.2(b)

```
Private Sub CommandButton1_Click()
Dim num As Integer
num = Int(Rnd * 6) + 1
Label1.Caption = num

End Sub
```

When you run the Excel VBA macro and each time you click on the command button, an integer between 1 and 6 inclusive will appear, thus resembles a die.

The output

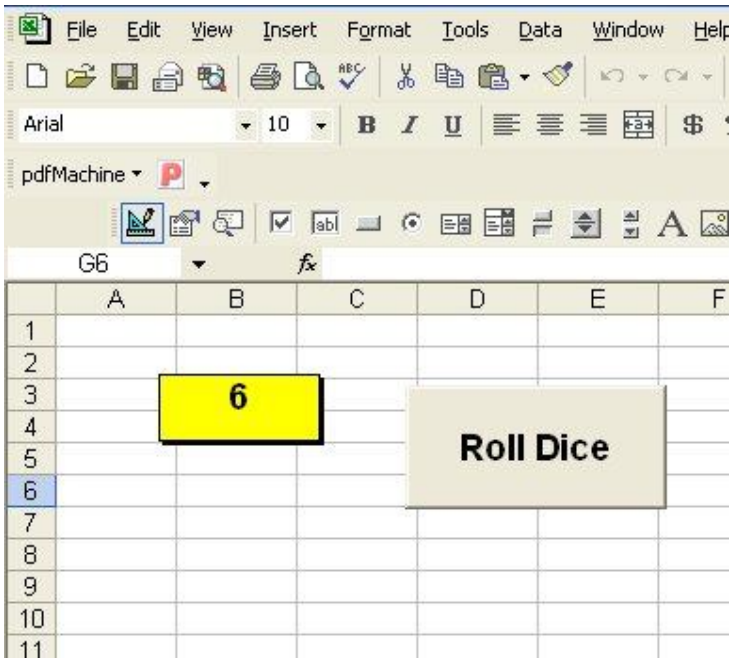


Figure 8.4

8.3 Mathematical Functions

Mathematical functions are very useful and important in programming because very often we need to deal with mathematical concepts in programming such as chance and probability, variables, mathematical logics, calculations, coordinates, time intervals and etc. The common mathematical functions in Visual Basic are **Int, Sqr, Abs, Exp, Log, Sin, Cos, Tan , Atn, Fix** and **Round.**

a) Int is the function that converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than the number. For example, Int(2.4)=2, Int(4.8)=4, Int(-4.6)= -5, Int(0.032)=0 and so on.
b) Sqr is the function that computes the square root of a number. For example, Sqr(4)=2, Sqr(9)=2 and etc.
c) Abs is the function that returns the absolute value of a number. So Abs(-8) = 8 and Abs(8)= 8.
d) Exp of a number x is the value of ex. For example, Exp(1)=e^1 = 2.7182818284590
e) Fix and Int are the same if the number is a positive number as both truncate the decimal part of the number and return an integer. However, when the number is negative, it will return the smallest integer that is larger than the number. For example, Fix(-6.34)= -6 while Int(-6.34)=-7.
f) Round is the function that rounds up a number to a certain number of decimal places. The Format is Round (n, m) which means to round a number n to m decimal places. For example, Round (7.2567, 2) =7.26
g) Log is the function that returns the natural Logarithm of a number. For example, Log (10)= 2.302585

Example 8.3 (a)

```
Private Sub CommandButton1_Click()
For i = 1 To 10

Cells(i, 1) = Sqr(i )
Next

End Sub
```

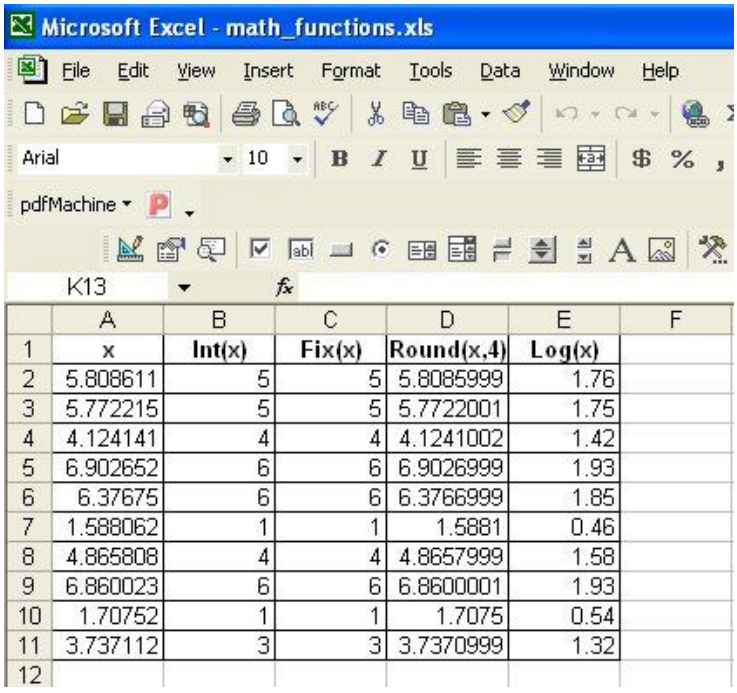
This program will find square root for number 1 to 10 and displays them from Cells(1,1) to Cells(10,1)

Example 8.3 (b)

In this example , we created an Excel VBA to compute the values of Int(x), Fix(x) , Round(x,4) and Log(x) and displays them in respective cells . Notice that you can combine two or more functions in your code, like Round(Log(x)). It uses the Do Loop statement and the Rnd function to generate random numbers. The statement x = Rnd * 7 generate random numbers between 0 and 7 . Using commas in between items will create spaces between them and hence a table of values can be created. The program and output are shown below:

```
Private Sub CommandButton1_Click()

Dim n As Integer
Dim x As Single
n = 2
Do While n < 12
x = Rnd * 7

Cells(n, 1) = x
Cells(n, 2) = Int(x)
Cells(n, 3) = Fix(x)
Cells(n, 4) = Round(x, 4)
Cells(n, 5) = Round(Log(x), 2)

n = n + 1
Loop

End Sub
```

The output is shown in Figure 8.5:



	A	B	C	D	E	F
1	x	Int(x)	Fix(x)	Round(x,4)	Log(x)	
2	5.808611	5	5	5.8085999	1.76	
3	5.772215	5	5	5.7722001	1.75	
4	4.124141	4	4	4.1241002	1.42	
5	6.902652	6	6	6.9026999	1.93	
6	6.37675	6	6	6.3766999	1.85	
7	1.588062	1	1	1.5881	0.46	
8	4.865808	4	4	4.8657999	1.58	
9	6.860023	6	6	6.8600001	1.93	
10	1.70752	1	1	1.7075	0.54	
11	3.737112	3	3	3.7370999	1.32	
12						

Figure 8.5

Excel VBA Lesson 9: Formatting Functions

Formatting output is a very important part of Excel VBA programming so that the data can be presented systematically and clearly to the users. Data in the previous lesson were presented fairly systematically through the use offunctions like **Int, Fix** and **Round**. However, to have better control of the output format, we can use a number of formatting functions in Excel VBA.

The Format function is a very powerful formatting function in Excel VBA. It can display the numeric values in various forms. There are two types of Format function, one of them is the built-in or predefined format while another one can be defined by the users.

9.1 Predefined Format function

The syntax of the predefined Format function is

Format (n, "style argument")

where n is a number and the list of style arguments are listed in Table 9.1

Style argument	Explanation	Example
General Number	To display the number without having separators between thousands.	Format(8972.234, "General Number")=8972.234
Fixed	To display the number without having separators between thousands and rounds it up to two decimal places.	Format(8972.2, "Fixed")=8972.23
Standard	To display the number with separators or separators between thousands and rounds it up to two decimal places.	Format(6648972.265, "Standard")= 6,648,972.27
Currency	To display the number with the dollar sign in front, has separators between thousands as well as rounding it up to two decimal places.	Format(6648972.265, "Currency")= $6,648,972.27
Percent	Converts the number to the percentage form and displays a % sign and rounds it up to two decimal places.	Format(0.56324, "Percent")=56.32 %

Table 9.1

Example 9.1

```
Private Sub CommandButton1_Click()
Cells(1, 1) = Format(8972.234, “General Number”)
Cells(2, 1) = Format(8972.2, “Fixed”)
Cells(3, 1) = Format(6648972.265, “Standard”)
Cells(4, 1) = Format(6648972.265, “Currency”)
Cells(5, 1) = Format(0.56324, “Percent”)
End Sub
```

The Output is shown in Figure 9.1

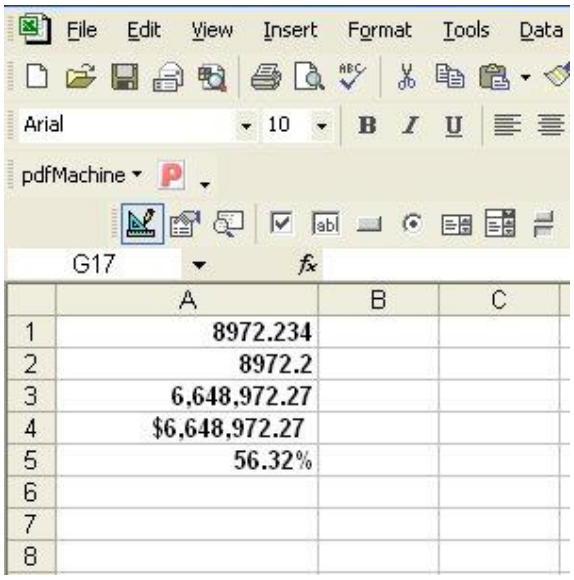


Figure 9.1

9.2 user-defined Format function

The syntax of the user-defined Format function is

Format (n, “user’s format”)

Although it is known as user-defined format, we still need to follows certain formatting styles. Examples of user-defined formatting style are listed in Table 9.2

Example	Explanation	Output
Format(781234.57,"0")	Rounds to whole number without separators between thousands.	781235
Format(781234.57,"0.0")	Rounds to 1 decimal place without separators between thousands.	781234.6
Format(781234.576,"0.00")	Rounds to 2 decimal places without separators between thousands.	781234.58
Format(781234.576,"#,##0.00")	Rounds to 2 decimal places with separators between thousands.	781,234.58
Format(781234.576,"$#,##0.00")	Shows dollar sign and rounds to 2 decimal places with separators between thousands.	$781,234.58
Format(0.576,"0%")	Converts to percentage form without decimal places.	58%
Format(0.5768,"0.00%")	Converts to percentage form with 2 decimal places.	57.68%

Table 9.2

Example 9.2

```
Private Sub CommandButton1_Click()

Cells(1, 1) = Format(781234.57, "0")
Cells(2, 1) = Format(781234.57, "0.0")
Cells(3, 1) = Format(781234.576, "0.00")
Cells(4, 1) = Format(781234.576, "#,##0.00")
Cells(5, 1) = Format(781234.576, "$#,##0.00")
Cells(6, 1) = Format(0.576, "0%")
Cells(7, 1) = Format(0.5768, "0.00%")

End Sub
```

The output is shown in Figure 9.2

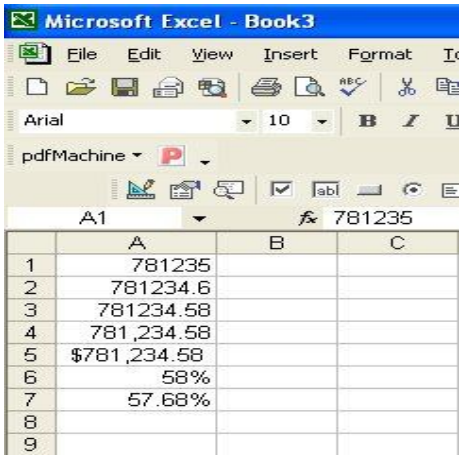


Figure 9.2

Excel VBA Lesson 10: String Manipulation Functions

Excel VBA can handle strings just as well as the stand-alone Visual Basic program. All the string handling functions in Visual Basic such as Len, Right, Left, Mid, Trim, Ltrim, Rtrim, Ucase, Lcase, Instr, Val, Str ,Chr and Asc can be used in Excel VBA.

10.1 The InStr function

InStr is a function that looks for and returns the position of a substring in a phrase

Example 10.1

```
Private Sub cmdInstr_Click()
Dim phrase As String
phrase = Cells(1, 1).Value
Cells(4, 1) = InStr(phrase, “ual”)
End Sub
```

The function InStr(phrase,”ual”) will find the substring “ual” from the phrase “Visual Basic” entered in cells(1,1) and then return its position, in this case, it is 4 from the left.

10.2 The Left function

Left is a function that extracts the characters from a phrase, starting from the left.

Left(phrase,4) means 4 characters are extracted from the phrase, starting from the leftmost position.

Example 10.2

```
Private Sub cmdLeft_Click()
Dim phrase As String
phrase = Cells(1, 1).Value
Cells(2, 1) = Left(phrase, 4)
End Sub
```

This code returns the substring “Visu” from the phrase “Visual Basic” entered in cells(1,1)

10.3 The Right function

Right is a function that extracts the characters from a phrase, starting from the Right.

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

Example 10.3

```
Private Sub cmdRight_Click()
Dim phrase As String
phrase = Cells(1, 1).Value
Cells(3, 1) = Right(phrase, 5)
```

This code returns the substring “Basic” from the phrase “Visual Basic” entered in cells(1,1)

10.4 The Mid function

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

Mid(phrase,8,3) means a substring of three characters are extracted from the phrase, starting from the 8th position from the left, including empty space.

Example 10.4

```
Private Sub cmdMid_Click()
Dim phrase As String
phrase = Cells(1, 1).Value
Cells(5, 1) = Mid(phrase, 8, 3)
End Sub
```

This code returns the substring “Bas” from the phrase “Visual Basic” entered in cells(1,1)

10.5 The Len function

Len is a function that return the length of a phrase(including empty space in between)

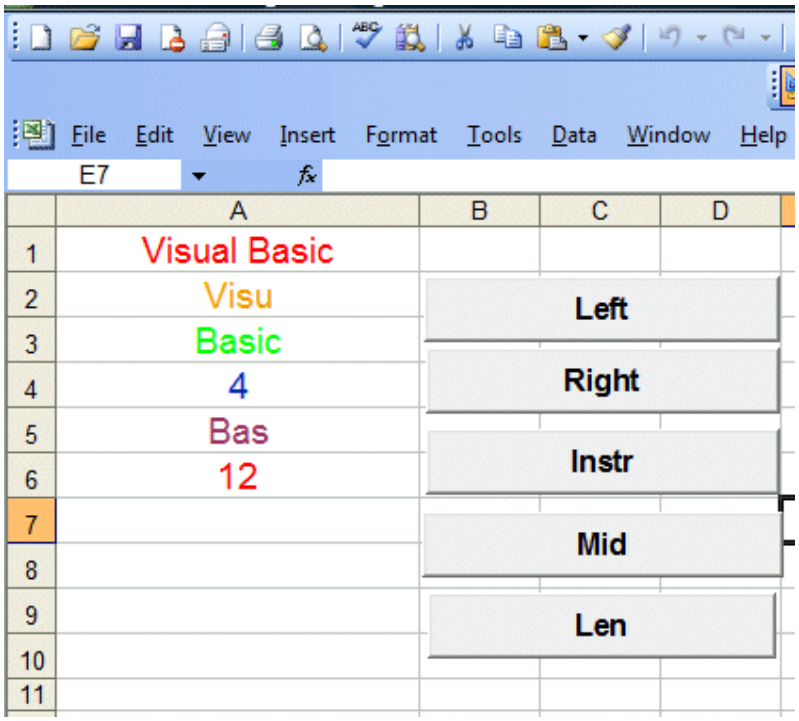
Example 10.5

```
Private Sub cmdLen_Click()
Dim phrase As String
phrase = Cells(1, 1).Value
Cells(6, 1) = Len(phrase)
End Sub
```

The code returns 12 for the phrase “Visual Basic” entered in cells(1,1)

Visual Basic Editor in MS Excel can handle strings just as good as a stand-alone VB program. All the string handling functions in Visual Basic such as Left, Right, Instr, Mid and Len can be used in Excel Visual Basic Editor.

The output of all the examples are shown in the Figure below:



10.6 The Ucase and the Lcase functions

The Ucase function converts all the characters of a string to capital letters. On the other hand, the Lcase function converts all the characters of a string to small letters. For example,

Ucase(“excel vba”) =EXCEL VBA

Lcase(“Excel VBA”) =excel vba

10.7 The Str and Val functions

The Str is the function that converts a number to a string while the Val function converts a string to a number. The two functions are important when we need to perform mathematical operations.

10.8 The Chr and the Asc functions

The Chr function returns the string that corresponds to an ASCII code while the Asc function converts an ASCII character or symbol to the corresponding ASCII code. ASCII stands for “American Standard Code for Information Interchange”. Altogether there are 255 ASCII codes and as many ASCII characters. Some of the characters may not be displayed as they may represent some actions such as the pressing of a key or produce a beep sound. The format of the Chr function is

Chr(charcode)

and the format of the Asc function is

Asc(Character)

The following are some examples:

Chr(65)=A, Chr(122)=z, Chr(37)=% , Asc(“B”)=66, Asc(“&”)=38

Excel VBA Lesson 11: Creating User-Defined Functions

One of the most power features of Excel VBA is that you can create your own functions. Creating user-defined functions allows you to write simpler code instead of having to use complicated formulas that come with MS Excel,thus allows you to accomplish a job easier. The user-defined functions can be entered into any cell or on the formula bar of the spreadsheet just like entering the built-in formulas of the MS Excel spreadsheet.

The syntax of a function is as follows:

Public Function functionName (Arg As dataType,…….) As dataType

or

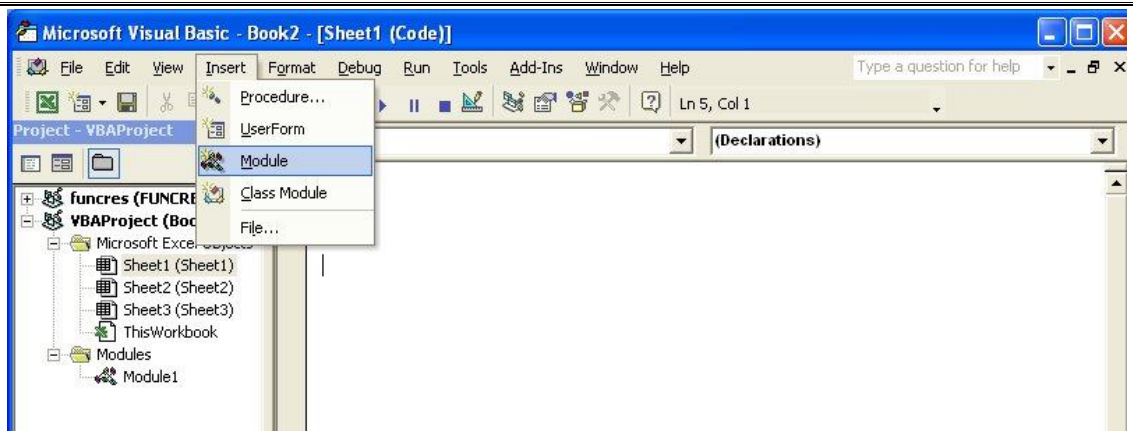
Private Function functionName (Arg As dataType,…….) As dataType

*** Public indicates that the function is applicable to the whole project while Private indicates that the function is only applicable to a certain module or procedure.**

In order to create a user-defined function in Excel VBA, you need to go into the Visual Basic Editor in MS Excel Spreadsheet. To enter the Visual Basic Editor, click on the View Code button, as shown below:

Figure 11.1

In the Visual Basic Editor, click on Insert on the menu bar to insert a module into the project, as shown in Figure 11.2.



Now you ready to create any function by typing the code in the editor.

Example 11.1: Cube Root Function

This cube root function calculates the cube root of a number. The code is as follows:

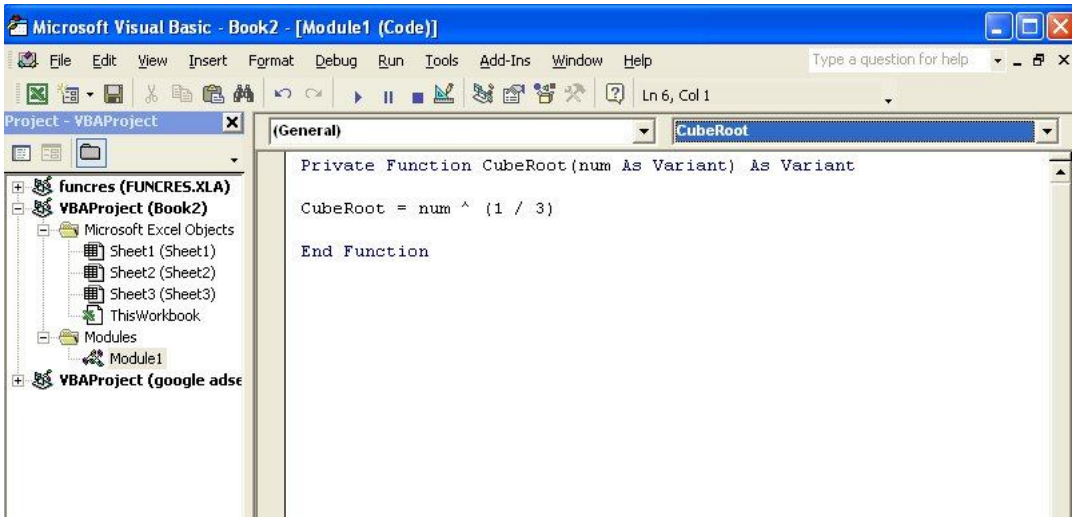


Figure 11.3

Now enter the function CubeRoot just like you enter the formula of MS Excel, as shown in Figure 11.4

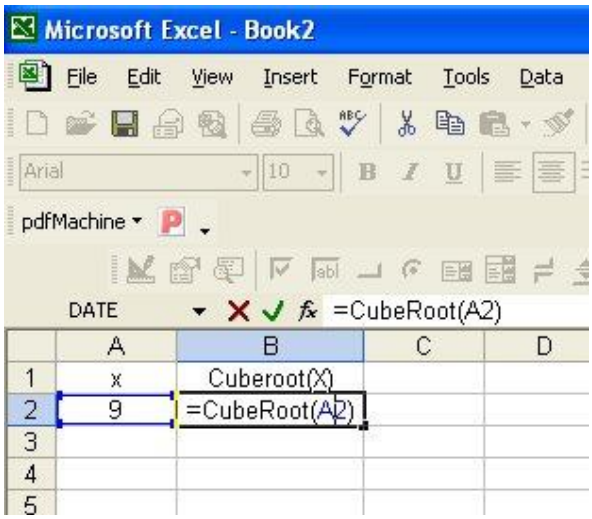


Figure 11.4

Press the Enter key and you will get the cube root of the number entered in A2, as shown in Figure 11.5

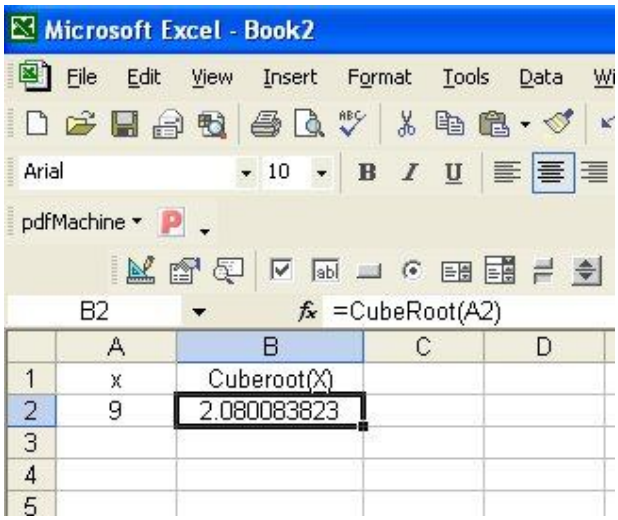
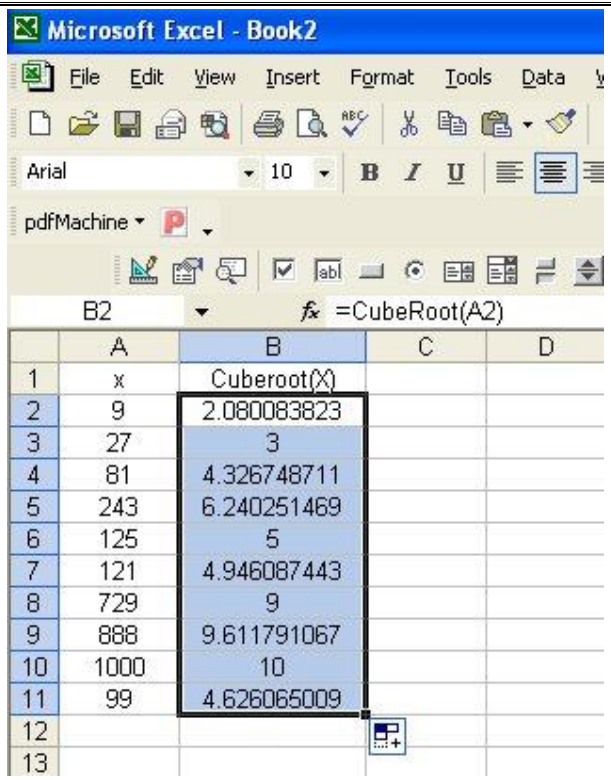


Figure 11.5

You can now copy the function to other cells by dragging the fill handler (i.e the bottom right corner of the cell), so the values will be updated automatically, as shown in Figure 11.6 below:



	A	B	C	D
1	x	Cuberoot(X)		
2	9	2.080083823		
3	27	3		
4	81	4.326748711		
5	243	6.240251469		
6	125	5		
7	121	4.946087443		
8	729	9		
9	888	9.611791067		
10	1000	10		
11	99	4.626065009		
12				
13				

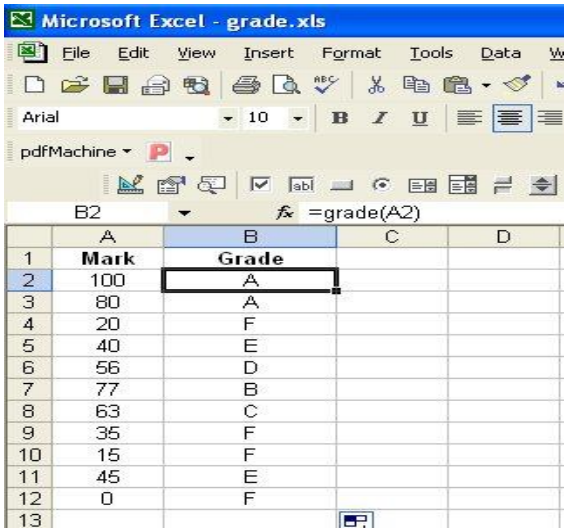
Figure 11.6

Example 11.2: Grade Function

The grade will automatically compute examination grades based on the marks that a student obtained. The code is shown below:

```
Public Function grade(mark As Variant) As String

Select Case mark
Case Is >= 80
grade = “A”
Case Is >= 70
grade = “B”
Case Is >= 60
grade = “C”
Case Is >= 50
grade = “D”
Case Is >= 40
grade = “E”
Case Else
grade = “F”
End Select

End Function
```

Now, enter the marks of the student and then enter the function in Cells B2 and drag the fill handle to copy the function to other cells, you will obtain the output as shown in Figure 11.7



	A	B	C	D
1	Mark	Grade		
2	100	A		
3	80	A		
4	20	F		
5	40	E		
6	56	D		
7	77	B		
8	63	C		
9	35	F		
10	15	F		
11	45	E		
12	0	F		
13				

Figure 11.7

Example 11.3 : Commissions Calculator

This functions calculate commissions payment based on the sales volume shown in the Table below:

Sales Volume($)	Commissons
<500	3%
<1000	6%
<2000	9%
<5000	12%
>5000	15%

The function is as follows:

```
Function Comm(Sales_V  As Variant) as Variant

If Sales_V <500 Then
Comm=Sales_V*0.03
Elseif Sales_V>=500 and Sales_V<1000 Then
Comm=Sales_V*0.06
Elseif Sales_V>=1000 and Sales_V<2000 Then
Comm=Sales_V*0.09
Elseif Sales_V>=200 and Sales_V<5000 Then
Comm=Sales_V*0.12
Elseif Sales_V>=5000  Then
Comm=Sales_V*0.15
End If

End Function
```

Sample output is shown in Figure 11.8

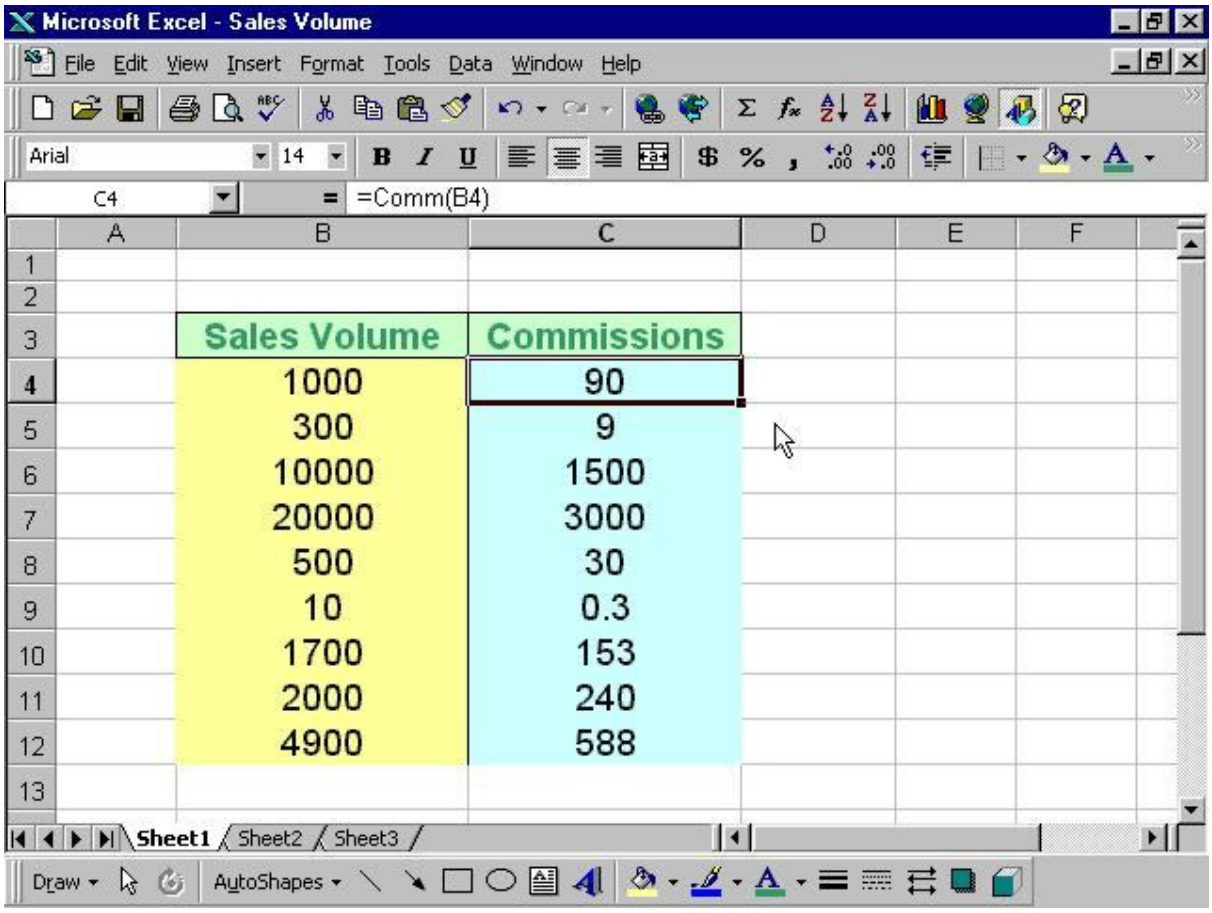


Figure 11.8

Example 11.4 BMI Function

Body Mass Index (BMI) is so popular today that it has become 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 problems associated with high BMI, such as hypertension, heart disease, diabetics and many others. We will show you how to create an Excel VBA function to calculate BMI .

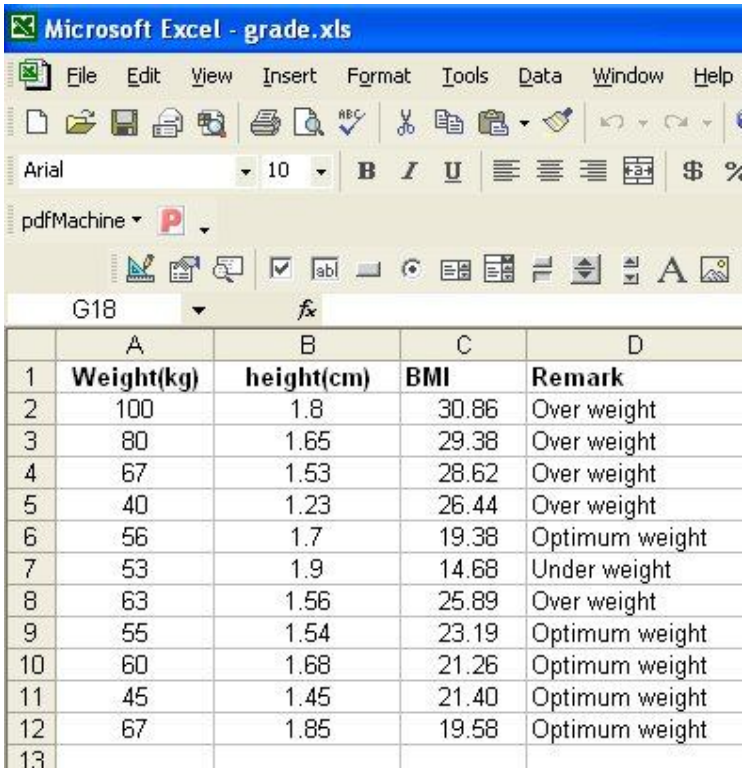
Base on the BMi formula BMI=wieght(/(height2), we created the following code:

```
Private Function BMI(weight As Single, height As Single) As Single

BMI = Round((weight) / height ^ 2,2)

End Function
```

This function comprises two arguments, weight and height.The use of Round function is to round up the BMI value to two decimal places. To implement this function, we allocate one column in MS Excel for entering weight and another column for entering height. We reserved the third column for displaying BMI. Besides, we use the fourth to show the remark. We need to create another function for remark, the code is as shown below:

```
Private Function Remark(bmi_value As Single) As String

If bmi_value <= 15 Then Remark = “Under weight” ElseIf bmi_value > 15 And
bmi_value <= 25 Then
Remark = “Optimum weight”
Else
Remark = “Over weight”
End If
End Function
```

The sample output is shown in Figure 11.9



	A	B	C	D
1	**Weight(kg)**	**height(cm)**	**BMI**	**Remark**
2	100	1.8	30.86	Over weight
3	80	1.65	29.38	Over weight
4	67	1.53	28.62	Over weight
5	40	1.23	26.44	Over weight
6	56	1.7	19.38	Optimum weight
7	53	1.9	14.68	Under weight
8	63	1.56	25.89	Over weight
9	55	1.54	23.19	Optimum weight
10	60	1.68	21.26	Optimum weight
11	45	1.45	21.40	Optimum weight
12	67	1.85	19.58	Optimum weight
13				

Excel VBA Lesson 12 : Formatting Font and Background Colors

In this Lesson, we will explore how to write Excel VBA code that formats the color of a MS Excel spreadsheet. Using Excel VBA code, we can change the font color as well as the the background color of each cell effortlessly.

Alright, let's create a program that can format random font and background colors using a randomize process. Colors can be assigned using a number of methods in Excel VBA, but it is easier to use the RGB function. The RGB function has three numbers corresponding to the red, green and blue components. The range of values of the three numbers is from 0 to 255. A mixture of the three primary colors will produce different colors.

The syntax to set the font color is

cells(i,j).Font.Color=RGB(x,y,x)

where x ,y , z are any numbers between 1 and 255

For example

cells(1,1).Font.Color=RGB(255,255,0) will change the font color to yellow

The syntax to set the cell's background color is

cells(i,j).Interior.Color=RGB(x,y,x)

Where x ,y , z can be any number between 1 and 255

Some RGB Color Codes

Color	RGB Code
	(0,0,0)
	(255,0,0)
	(255,255,0)
	(255,165,0)
	(0,0,255)
	(0,128,0)
	(128,0,128)

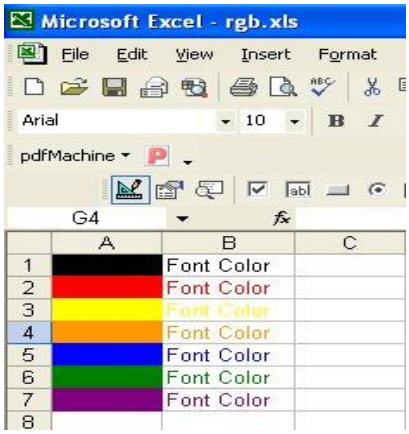
Example 12.1

In this example, clicking the command button changes the background colors from Cells(1,1) to Cells(7,1) according to the specified RGB color codes. It also format the font colors from Cells(1,2) to cells(7,2) using specified RGB color codes.

The code

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

'To fill the cells with colors using RGB codes
Cells(1, 1).Interior.Color = RGB(0, 0, 0)
Cells(2, 1).Interior.Color = RGB(255, 0, 0)
Cells(3, 1).Interior.Color = RGB(255, 255, 0)
Cells(4, 1).Interior.Color = RGB(255, 165, 0)
Cells(5, 1).Interior.Color = RGB(0, 0, 255)
Cells(6, 1).Interior.Color = RGB(0, 128, 0)
Cells(7, 1).Interior.Color = RGB(128, 0, 128)
'To format font color with RGB codes
For i = 1 To 7
Cells(i, 2).Value = "Font Color"
Next
Cells(1, 2).Font.Color = RGB(0, 0, 0)
Cells(2, 2).Font.Color = RGB(255, 0, 0)
Cells(3, 2).Font.Color = RGB(255, 255, 0)
Cells(4, 2).Font.Color = RGB(255, 165, 0)
Cells(5, 2).Font.Color = RGB(0, 0, 255)
Cells(6, 2).Font.Color = RGB(0, 128, 0)
Cells(7, 2).Font.Color = RGB(128, 0, 128)

End Sub
```

The Output



Example 12.2

In this example, the font color in cells(1,1) and background color in cells(2,1) are changing for every click of the command button due to the randomized process.Rnd is a random number between 0 and 1, therefore 255* Rnd will produce a number between 0 and 255 and Int(255*Rnd) will produce integers that take the values from 0 to 254

So we need to add 1 to get random integers from 0 to 255.

For example;Rnd=0.229

255*Rnd=58.395

Int(58.395)=58

The code Private Sub CommandButton1_Click()Randomize Timer

```
Dim i, j, k As Integer
i = Int(255 * Rnd) + 1
j = Int(255 * Rnd) + 1
k = Int(255 * Rnd) + 1
Cells(1, 1).Font.Color = RGB(i, j, k)
Cells(2, 1).Interior.Color = RGB(j, k, i)
End Sub
```

The Output

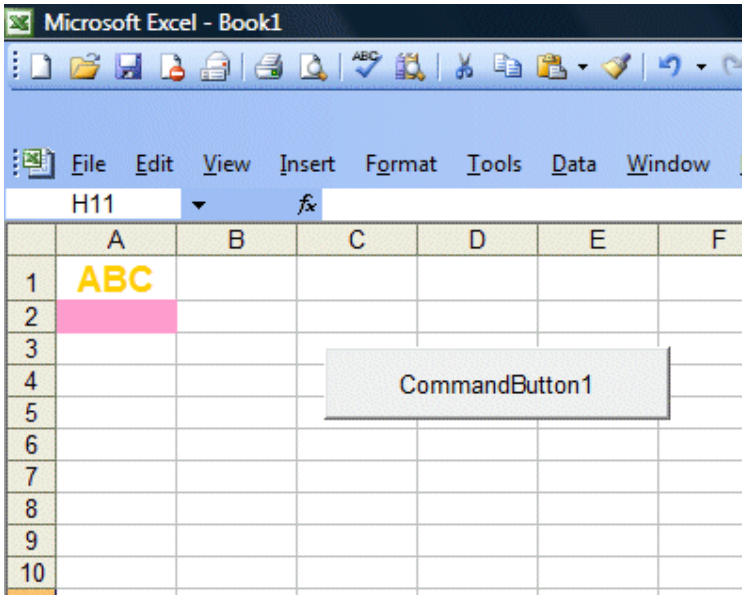


Figure 12.1

Excel VBA Lesson 13 : Introduction to Excel VBA Object Part 1

13.1 The Concept of Object in Excel VBA

Most programming languages today deal with objects, a concept called object oriented programming. Although Excel VBA is not a truly object oriented programming language, it does deal with objects. Excel VBA object is something like a tool or a thing that has certain functions and properties, and can contain data. For example, an Excel Worksheet is an object, cell in a worksheet is an object, range of cells is an object, font of a cell is an object, a command button is an object, and a text box is an object and more.

In order to view the ExcelVBA objects, click object browser in the Excel VBA IDE and you will be presented with a list of objects(or classes) together with their properties and methods, as shown in Figure 13.1.

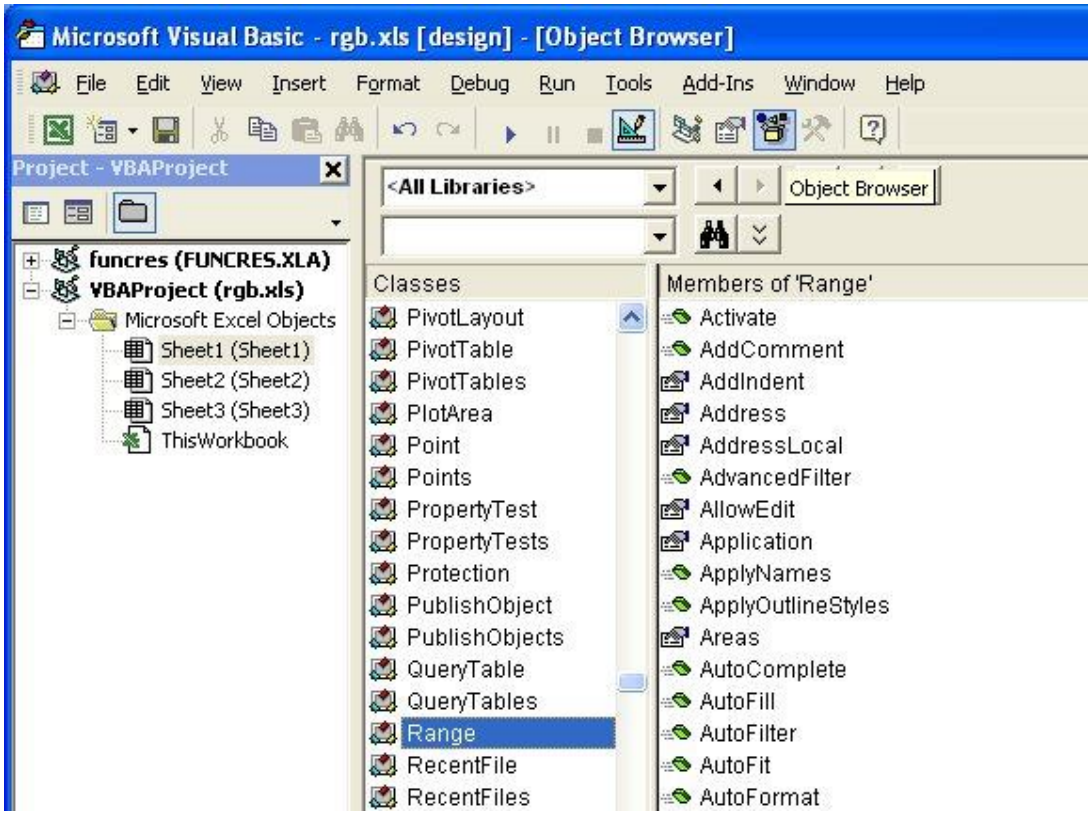


Figure 13.1

If you have inserted some Active-X controls into the worksheet, clicking on the relevant worksheet will reveal the objects together with the associated events, as shown in Figure 13.2

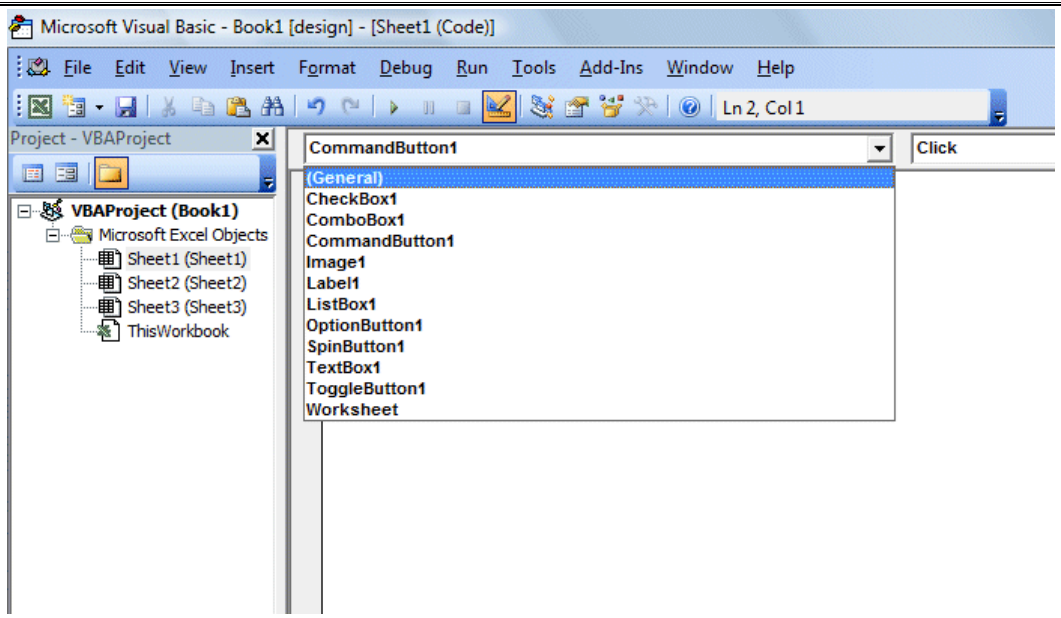


Figure 13.2

13.2: Object Properties

An Excel VBA object has properties and methods. Properties are like the characteristics or attributes of an object. For example, Range is an Excel VBA object and one of its properties is value. We connect an object to its property by a period(a dot or full stop). The following example shows how we connect the property value to the Range object.

Example 13.1

```
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. So the above procedure can be rewritten as

Example 13.2

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

Cells is also an Excel VBA object, but it is also the property of the range object. So an object can also be a property, it depends on the hierarchy of the objects. Range

has higher hierarchy than cells, and interior has lower hierarchy than Cells, and color has lower hierarchy than Interior, so you can write

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

This statement will fill cells (1,1) with yellow color. 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 overwrite the range specified by the Range object.

Another object is font that belongs to the Range object. And font has its properties.For example, Range("A1:A4").Font.Color=vbYellow , the color property of the object Font will result in all the contents from cell A1 to cell A4 to be filled in yellow color.

Sometime it is not necessary to type the properties, Excel VBA IntelliSense will display a drop-down list of proposed properties after you type a period at the end of the object name. You can then select the property you want by double clicking the it or by highlighting it then press the Enter key. The IntelliSense drop-down is shown in Figure 13.3

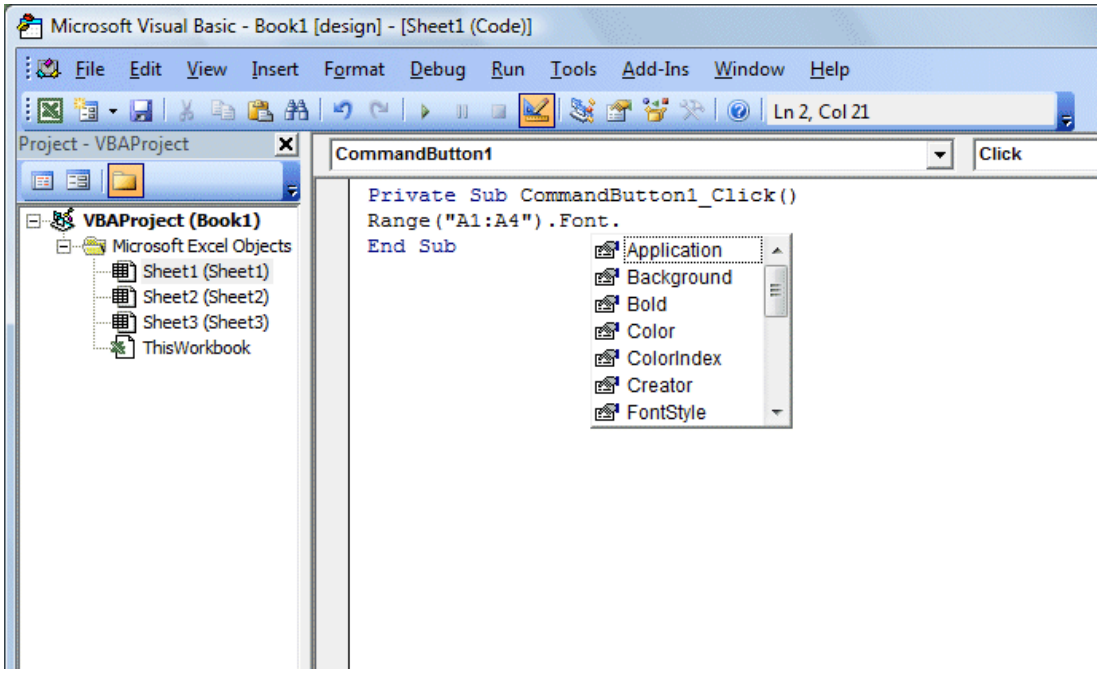


Figure 13.3

We shall discuss object methods in the next lesson

Excel VBA Lesson 14: Introduction to Excel VBA Objects Part 2

In lesson 13, we have learned the concepts of Excel VBA objects. You have also learned that an Excel VBA object has properties and methods. We have already dealth with properties and we shall leaarn about methods in this lesson.

14.1: Methods

A method of an Excel VBA object normally do something or perform certain operations. For example, ClearContents is a method of the range object that clears the contents of a cell or a range of cells. For example, You can write the following code to clear the contents of certain range:

Example 14.1

```
'Clear contents from cells A1 to A6
Private Sub CommandButton1_Click()
Range("A1:A6").ClearContents
End Sub
```

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 14.2

Example 14.2

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

In order to clear the contents of the entire worksheet, you can use the following code:

Sheet1.Cells.ClearContents

But if you only want to clear the formats of an entire worksheet, you can use the following syntax:

Sheet1.Cells.ClearFormats

To select a range of cells, you can use the **Select** method. This method selects a range of cells specified by the Range object. The syntax is Range("A1:A5").Select

Example 14.3

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

Example 14.4

This example allows the user to specifies the range of cells to be seleted.

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

To deselect the selected range, we can use the Clear method.

```
Range("CiRj:CmRn").Clear
```

Example 14.5

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:A5″).Select
End Sub

Private Sub CommandButton2_Click()
Range("A1:A5″).Clear
End Sub
```

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

Another very useful method is the Autofill method. This method performs an autofill on the cells in the specified range with a series of items including numbers, days of week, months of year and more. The syntax is

Expression.AutoFill(Destination, Type)

Where Expression can be an object or a variable that returns and object. Destination means the required Range object of the cells to be filled. The destination must include the source range. Type means type of series, such as days of week, month of year and more. The AutoFill type constant is something like xlFillWeekdays, XlFillDays, XlFillMonths and more.

Example 14.6

```
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.

Example 14.7

```
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
```

Example 14.8

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 the InputBox. Since each time we want to autofill a new range, we need to 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
```

Excel VBA Lesson 15: The Range Object

Range is one of the most important and most commonly used Excel VBA object. In fact, we have dealt with the Range object in previous lessons.

15.1 The Select Method

Range object contains two arguments that specifies a selected area on the spreadsheet. The syntax is

Range(starting_cell,Ending_ Cell)

For example, Range("A1:C6″) means the specified range is from cell A1 to C6.

To select the specified range, the syntax is

Range("A1:C6″).Select

where select is a method of the Range object

Example 15.1

```
Private Sub CommandButton1_Click()
Range("A1:C6″).Select
End Sub
```

15.2 The Columns Property

The columns property of the Range object is to select a certain columns in the particular range specified by the Range object.

The syntax is

Range(starting_cell,Ending_ Cell).Columns(i).Select

Example 15.2

This example select column C in the range A1 to C6

```
Private Sub CommandButton2_Click()
Range("A1:C6″).Columns(3).Select
End Sub
```

You can also use Cells(1,1) to Cells(6,3) instead of A1:C6, the syntax is

Range(Cells(1,1),Cells(6,3)).Columns(3).Select

* Notice that you don't use double inverted commas and colon.

The output is as shown in Figure 15.1

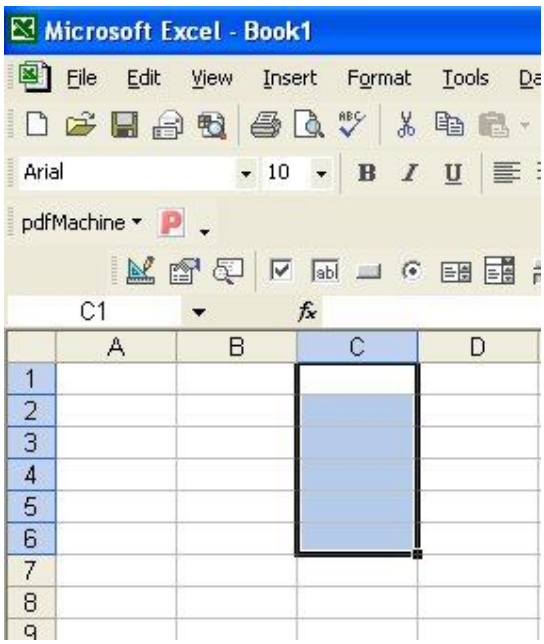


Figure 15.1

15.3 Using With Range……End With

You can also format font the cells in a particular column in terms of type, color, bold,italic, underlined and size using the With Range.....End With Structure. It can also be used to format other Range properties like background color. Using With Range....End With structure can save time and make the code leaner.

Example 15.3

```
Private Sub CommandButton1_Click()
With Range("A1:C6").Columns(2)
.Font.ColorIndex = 3
.Font.Bold = True
.Font.Italic = True
.Font.Underline = True
.Font.Name = "Times New Roman"
.Font.Size = 14
.Interior.Color = RGB(255, 255, 0)
```

End With

End Sub

* Without using With Range....End With, you need to write every line in full, like this

Range("A1:C6").Columns(2).Font.ColorIndex = 3

The output:

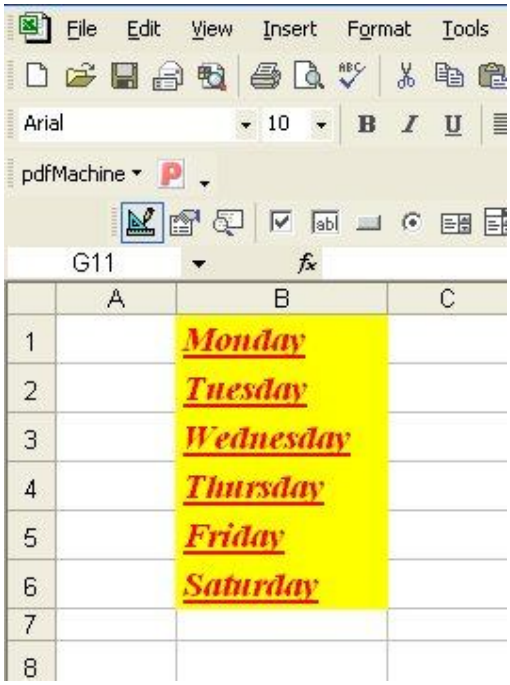


Figure 15.2

15.4 The Rows Property

Basically the syntax for the Rows property is similar to that of the Columns property, you just need to replace Columns with rows.

The syntax of selecting a row within a certain range is

Range(starting_cell,Ending_ Cell).Rows(i).Select

Example 15.4

This following code selects the third row within the range A1 to F3

```
Private Sub CommandButton2_Click()
Range("A1:F3").Rows(3).Select
End Sub
```

The output

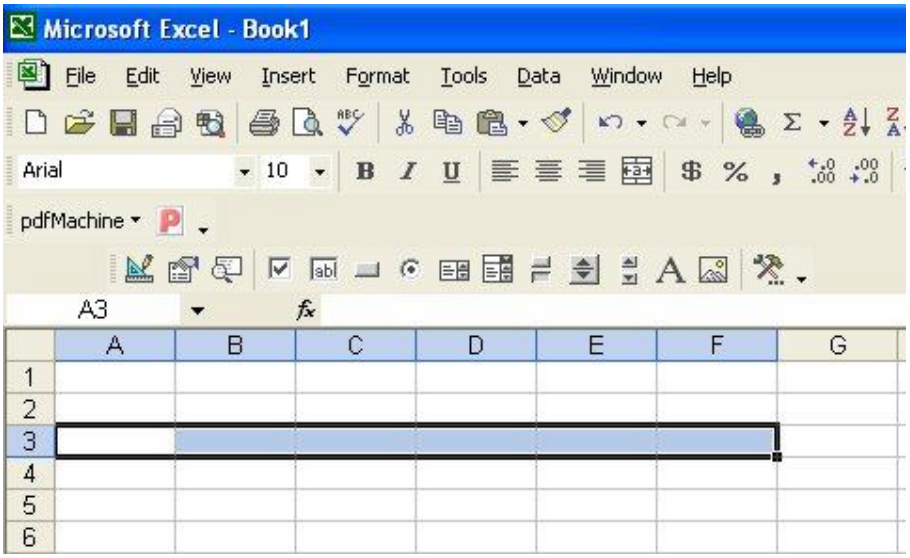


Figure 15.3

Example 15.5: Using With Range...End With for Rows

```
Private Sub CommandButton1_Click()
With Range("A1:F3").Rows(2)
.Font.ColorIndex = 3
.Font.Bold = True
.Font.Italic = True
.Font.Underline = True
.Font.Name = "Times New Roman"
.Font.Size = 14
.Interior.Color = RGB(255, 255, 0)

End With

End Sub
```

The Output

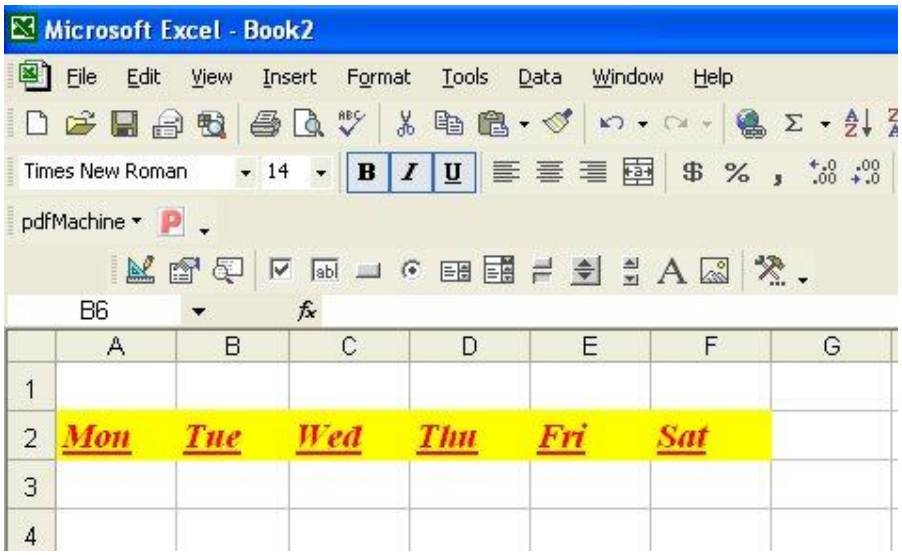


Figure 15.4

15.5 Using the Set keyword to Declare Range

We can write Excel VBA code that can specifies certain range of cells using the **Set** keyword and then perform certain tasks according to a set of conditions.

In Example 15.6, we shall write the ExcelVBA code such that it can accept range input from the user and then change the mark to blue if it is more than or equal to 50 and change it to red if the mark is less than 50.

Example 15.6

```
Private Sub CommandButton1_Click()
Dim rng, cell As Range, selectedRng As String
selectedRng = InputBox("Enter your range")
Set rng = Range(selectedRng)
For Each cell In rng
If cell.Value >= 50 Then
cell.Font.ColorIndex = 5
Else
cell.Font.ColorIndex = 3
End If
Next cell
End Sub
```

Explanation:

The InputBox function is used to accept value from the users.

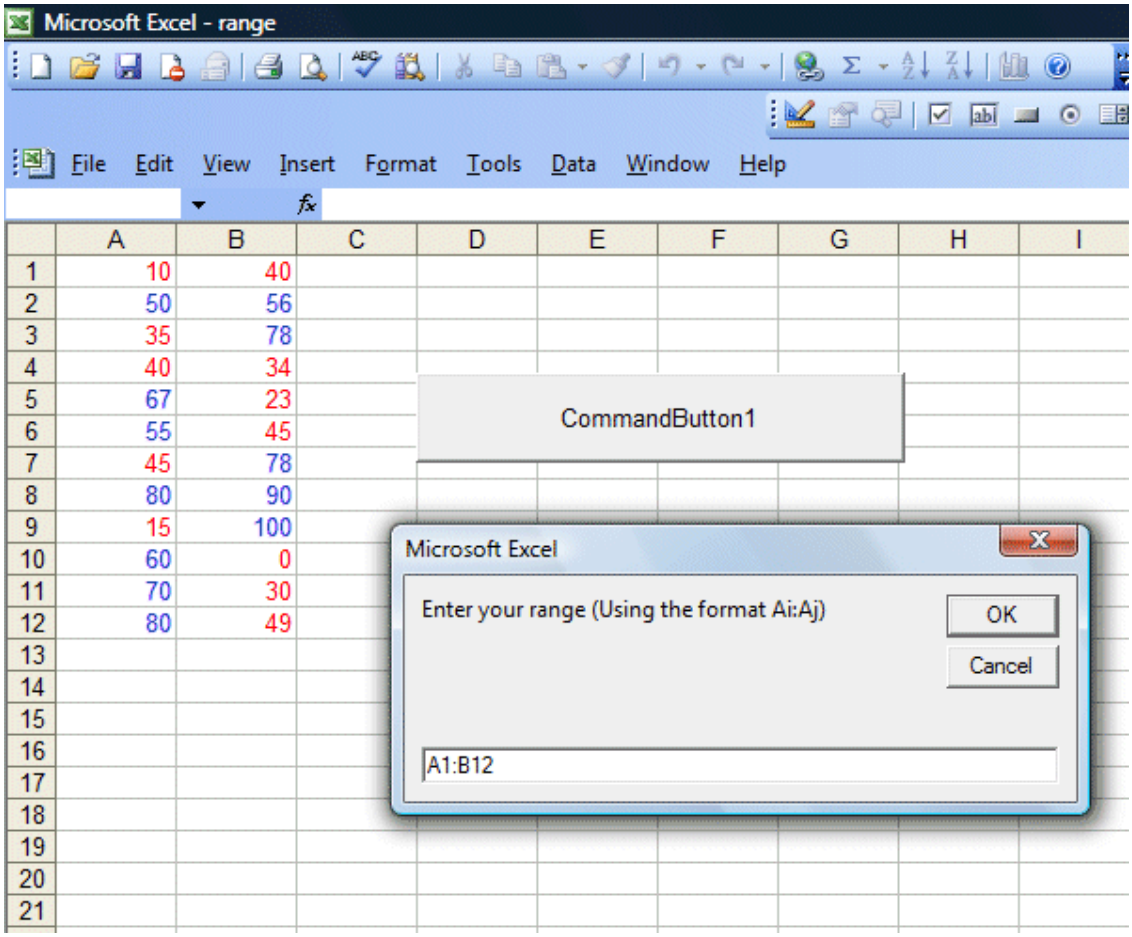
rng and cell are declared as a Range variable using the Dim statement while selectedRng is declared as a string that receive input from the user.

Once the input is obtained from the user, it is stored using the Set method and the Range function.

For Each cell In rng .....Net cell is a loop that can iterate through the selected range, one cell at a time.

The If...Then...Else statements are to specify the color of the font according to the range of values determined by the conditions.

The Output



	A	B	C	D	E	F	G	H	I
1	10	40							
2	50	56							
3	35	78							
4	40	34							
5	67	23							
6	55	45							
7	45	78							
8	80	90							
9	15	100							
10	60	0							
11	70	30							
12	80	49							
13									
14									
15									
16									
17									
18									
19									
20									
21									

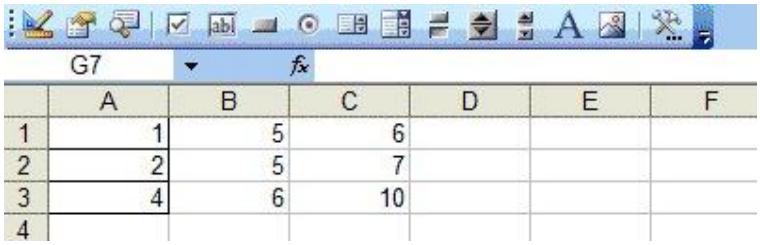
15.6 The Formula property

You can use the Formula property of the Range object to write your own customized formula.

Example 15.7

```
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 15.5



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

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

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

Example 15.8

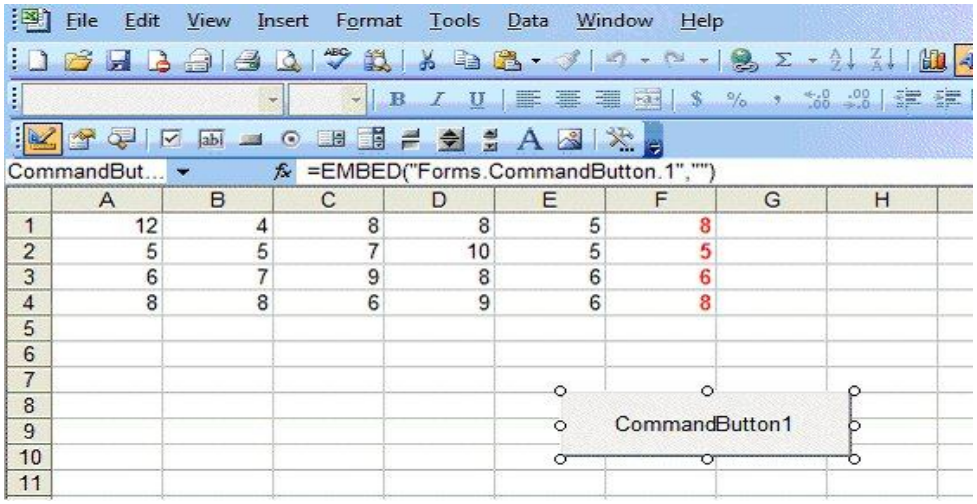
In this example, the program computes the average of the corresponding values in column A and column B and displays the results in column C. For example, the mean of values in cell A1 and Cell B1 is computed and displayed in Cell C1. Subsequent means are automatically copied down Column C until cell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

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

Example 15.9: 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 15.6.

```
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	H
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								

Excel VBA Lesson 16: The Worksheet Object

16.1 The Worksheet Properties in Excel VBA

Similar to the Range Object, the Worksheet has its own set of properties and methods. When we write Excel VBA code involving the Worksheet object, we use *Worksheets*. The reason is that we are dealing with a collection of worksheets most of the time, so using Worksheets enables us to manipulate multiple worksheets at the same time.

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

Example 16.1

```
Private Sub CommandButton1_Click()

MsgBox Worksheets(1).Name

End Sub
```

The above example will cause a pop-up dialog that displays the worksheet name as sheet 1, as shown below:

Figure 16.1

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

Example 16.2

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

The output is shown in Figure 16.2.

Figure 16.2

Example 16.3

The count property in this example will return the number of columns in the worksheet.

```
Private Sub CommandButton1_Click()

MsgBox Worksheets(1).Columns.Count

End Sub
```

The output is shown below:

Figure 16.3

Example 16.4

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 16.4

16.2 The Worksheet Methods

Some of the worksheet methods are *add, delete, select, SaveAs, copy, paste* and more.

Example 16.5

In this example, when the user clicks the first command button, it will add a new sheet to the workbook. When the user clicks the second command button, it will delete the new worksheet that has been added earlier.

```
Private Sub CommandButton1_Click()

Worksheets. Add

End Sub

Private Sub CommandButton2_Click()

Worksheets(1).Delete

End Sub
```

Example 16.6

The select method associated with worksheet lets the user select a particular worksheet. In this example, worksheet 2 will be selected.

```
Private Sub CommandButton1_Click()

'Worksheet 2 will be selected

Worksheets(2).Select

End Sub
```

The select method can also be used together with the Worksheet's properties Cells, Columns and Rows as shown in the following examples.

Example 16.7

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

Example 16.8

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

Example 16.9

```
Private Sub CommandButton1_Click()

'Row 1 will be selected

Worksheets (1).Rows (1).Select

End Sub
```

Excel VBA also allows us to write code for copy and paste. Let's look at the following Example:

Example 16.10

```
Private Sub CommandButton1_Click()
'To copy the content of a cell 1
Worksheets(1).Cells(1).Select
Selection.Copy
End Sub

Private Sub CommandButton2_Click()
'To paste the content of cell 1 to cell 2
Worksheets(1).Cells(2).Select
ActiveSheet.Paste
End Sub
```

Excel VBA Lesson 17: The Workbook Object

In previous lesson, we have learned write associated with the worksheet object. In this lesson, we shall learn about the Workbook object . The Workbook object at the top of the hierarchy of the Excel VBA objects. We will deal with properties and methods associated the Workbook object.

17.1 The Workbook Properties.

When we write Excel VBA code involving the Workbook object, we use Workbooks. The reason is that we are dealing with a collection of workbooks most of the time, so using Workbooks enables us to manipulate multiple workbooks at the same time.

When will deal with multiple workbooks, we can use indices to denote different workbooks that are open, using the syntax Workbooks (i), where i is an index. For example, Workbooks (1) denotes Workbook1, Workbooks (2) denotes Workbook2 and more.

Workbooks have a number of properties. Some of the common properties are Name, Path and FullName Let's look at the following example:

Example 17.1

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

The program will cause a message dialog box to pop up and displays the first workbook name, i.e. workbook_object1.xls as shown in Figure 17.1 below:

Figure 17.1

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
```

Example 17.2

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

Or you can use the following code:

```
Private Sub CommandButton1Click ()
MsgBox Workbooks ("workbook_object1.xls").Path
End Sub
```

The output is shown below:

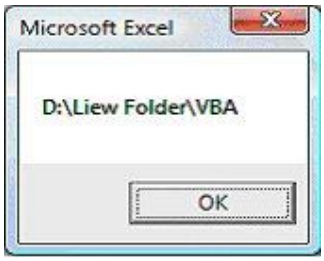


Figure 17.2

Example 17.3

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("workbook_object1.xls").Fullname
End Sub
```

The output is shown in Figure 17.3.

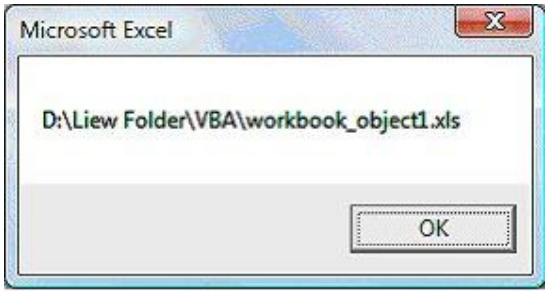


Figure 17.3

17.2 The Workbook Methods

There are a number of methods associated with the workbook object. These methods are Save, SaveAs, Open, Close and more.

Example 17.4

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 17.4.

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

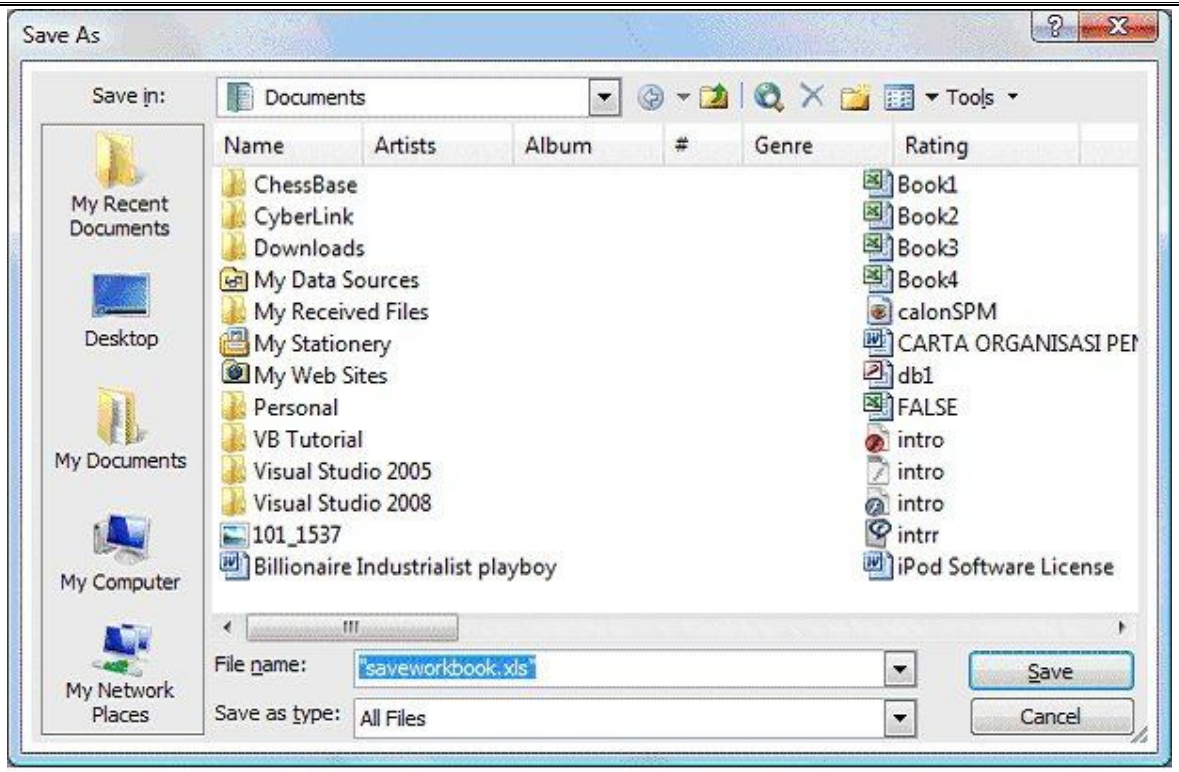


Figure 17.4

Another method associated with the workbook object is open. The syntax is Workbooks.Open ("File Name")

Example 17.5

In this example, when the user click on the command button, it wil open the file workbook_object1.xls under the path C:\Users\liewvk\Documents\

```
Private Sub CommandButton1_Click()
Workbooks.Open ("C:\Users\liewvk\Documents\workbook_object1.xls")
End Sub
```

The close method is the command that closes a workbook. The syntax is Workbooks (i).Close

Example 17.6

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

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

Excel VBA Lesson 18: Working with Excel VBA Controls Part 1

Excel VBA provides a number of controls that can be used to perform certain tasks by writing Excel VBA code for them. These controls are also known as Active-X controls. These controls are Excel VBA objects so they have their own properties, methods and events. They can be found on the Excel Control Toolbox, as shown in the Figure 18.1 :

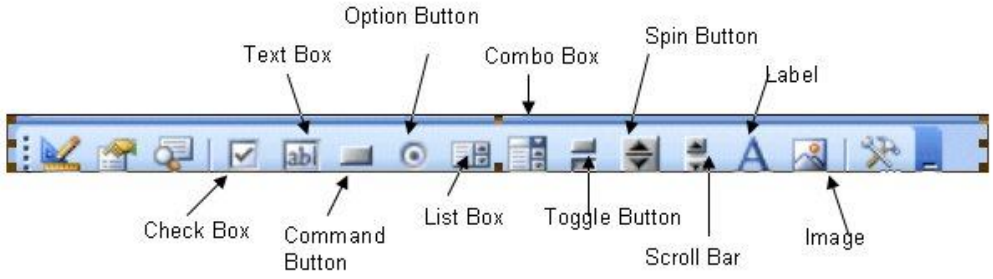


Figure 18.1

18.1 Check Box

The Check box is a very useful control in Excel VBA. It allows the user to select one or more items by checking the check box or check boxes concerned. For example, you may create a shopping cart where the user can click on check boxes that correspond to the items they intend to buy, and the total payment can be computed at the same time.

One of most important properties of the check box is Value. If the check box is selected or checked, the value is true, whilst if it is not selected or unchecked, the Value is False.
The usage of check box is illustrated in Example 18.1

Example 18.1

In this example, the user can choose to display the sale volume of one type of fruits sold or total sale volume. The code is shown below:

```
Private Sub CommandButton1_Click()
If CheckBox1.Value = True And CheckBox2.Value = False Then
MsgBox “Quantity of apple sold is” & Cells (2, 2).Value
ElseIf CheckBox2.Value = True And CheckBox1.Value = False Then
MsgBox “Quantity of orange sold is ” & Cells(2, 3).Value
Else
```

```
MsgBox “Quantity of Fruits sold is” & Cells (2, 4).Value
End If
End Sub
```

The Interface is shown in Figure 18.2

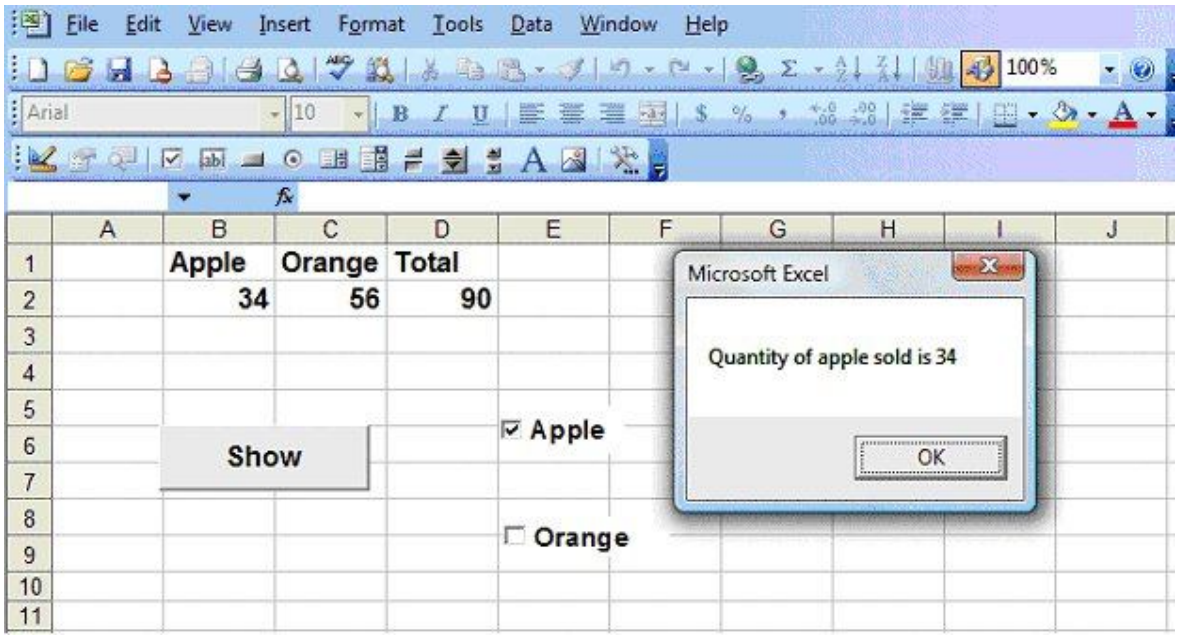


Figure 18.2

18.2 Text Box

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

Example 18.2

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

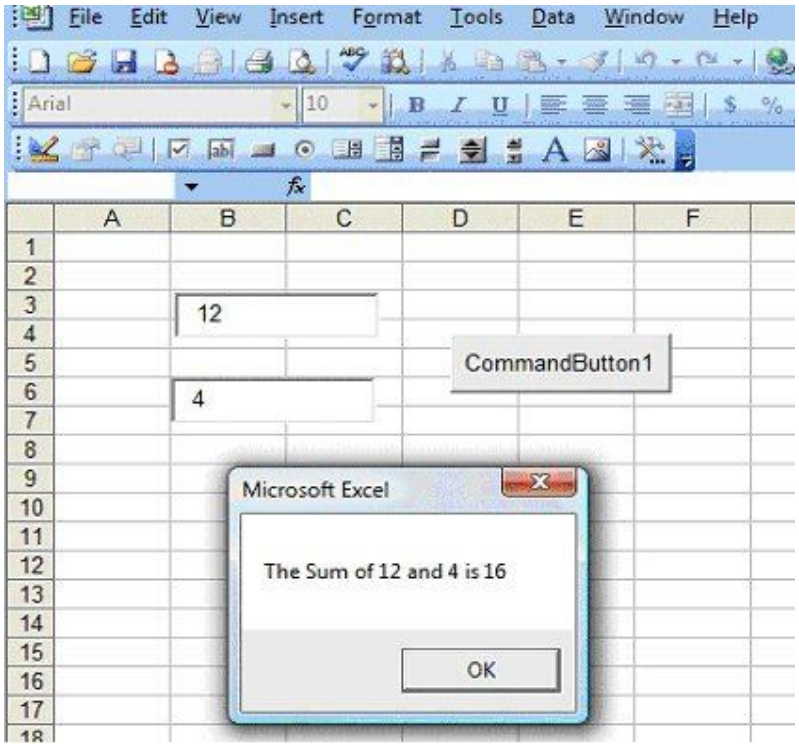


Figure 18.3

18.3 Option Button

The option button control also lets the user selects one of the choices. However, two or more option buttons must work together because as one of the option buttons is selected, the other option button will be deselected. 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 deselected; its value is set to “False”.

Example 18.3

This example demonstrates the usage of the option buttons. In this example, the Message box will display the option button selected by the user. The output interface is shown in Figure 18.4.

The code

Private Sub OptionButton1_Click ()

MsgBox “Option 1 is selected”

End Sub

```
Private Sub OptionButton2_Click()
MsgBox “Option 2 is selected”
End Sub
Private Sub OptionButton3_Click()
MsgBox “Option 3 is selected”
End Sub
```

The Output Interface

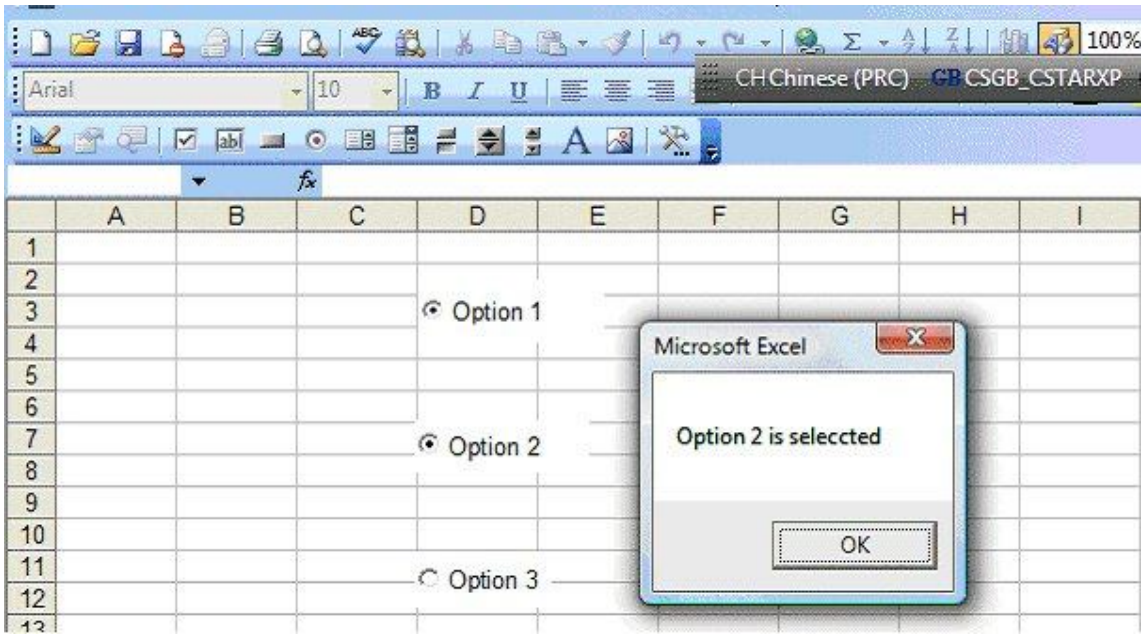


Figure 18.4

Excel VBA Lesson 19: Working with Excel VBA Controls Part 2

We have learned how to work with check boxes, option buttons and text boxes in Excel VBA in the previous lesson. We shall continue to learn how to manipulate other controls in Excel VBA in this lesson.

19.1 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 19.1.

Example 19.1

```
Private Sub CommandButton1_Click()
For x = 1 To 10
ListBox1.AddItem “Apple”
Next
End Sub
Private Sub CommandButton2_Click()
For x = 1 To 10
ListBox1.Clear
Next
End Sub
```

Figure 19.1

19.2 Combo Box

The function 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 which are presented in a drop-down list. In order to add items to the list, you can also use the Add Item method.

Example 19.2

```
Private Sub CommandButton1_Click()
ComboBox1.Text = “Apple”
For x = 1 To 10
ComboBox1.AddItem “Apple”
Next
End Sub
Private Sub CommandButton2_Click()
ComboBox1.Clear
End Sub
```

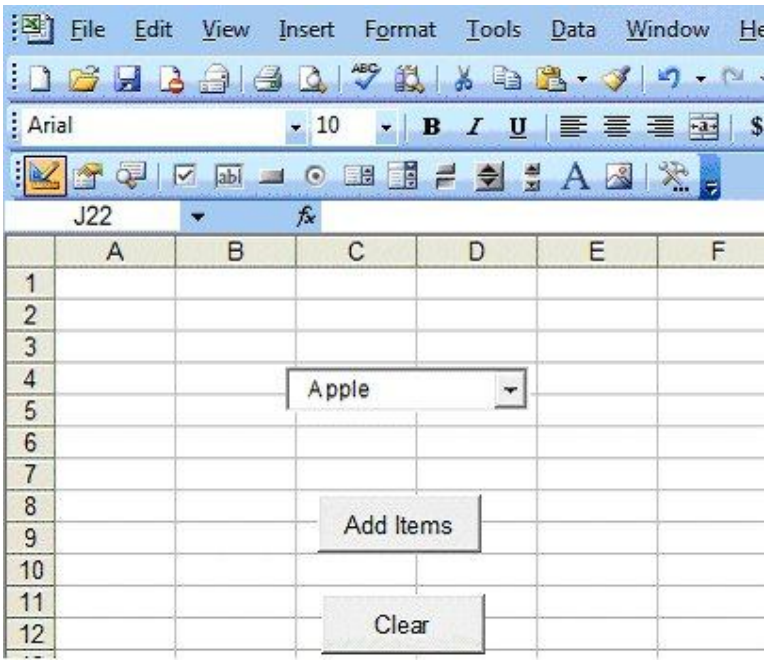


Figure 19.2

19.3 Toggle Button

Toggle button lets the user switches 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 code structure, we can thus switch from one action to another by pressing the toggle button repeatedly.

Example 19.3

In this example, the user can toggle between apple and orange as well as font colors.

```
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
```

View the animated image in Figure 19.3

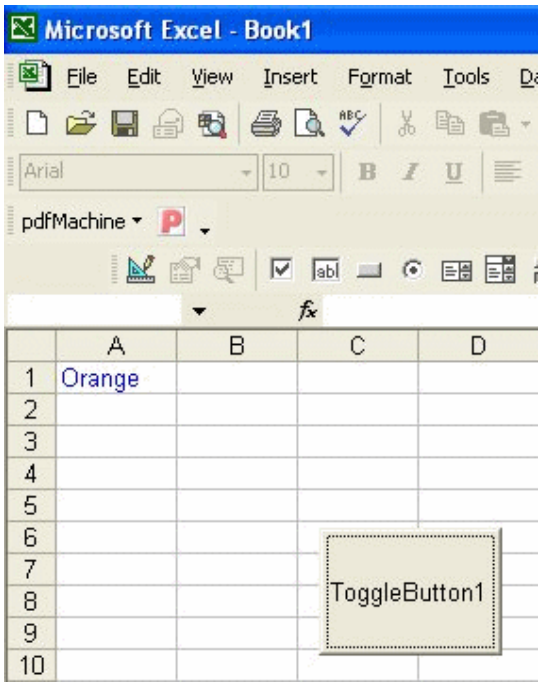

Microsoft Excel - Book1
File Edit View Insert Format Tools
Arial
10
pdfMachine
A B C D
1 Orange
2
3
4
5
6
7
8
9
10
ToggleButton1

Figure 19.3

Excel VBA Lesson 20: Sub Procedures

A sub procedure in Excel VBA is a procedure that performs a specific task and to return 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 an Excel VBA control such as a command button. It is called by the main program whenever it is required to perform a certain task. Sub procedures help to make programs smaller and easier to manage.

A Sub procedure begins with a Sub statement and ends with an End Sub statement. The program structure of a sub procedure is as follows:

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

Example 20.1

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 Rge 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 Rge (for range) to perform the task of resizing the font to 15 for values>40 in range A1 to A10.

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

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
```

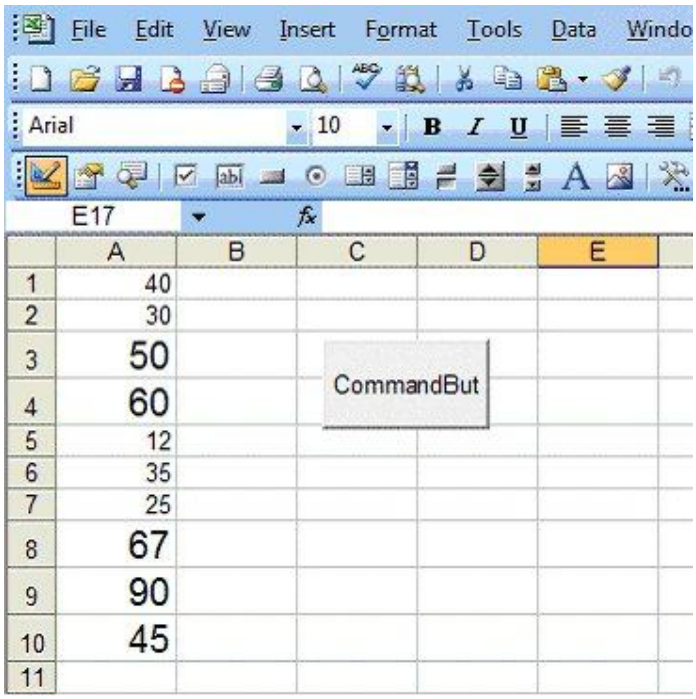


Figure 20.1

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 20.2.

Example 20.2

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

Excel VBA Lesson 21: Array in Excel VBA

21.1 Array in Excel VBA

When we work with a single item in Excel VBA, we only need to use one variable. However, if we have to deal with a list of items which are of similar type , we need to declare an array of variables instead of using a variable for each item. For example, if we need to enter one hundred names, instead of declaring one hundred different variables, we need to declare only one array. By definition, 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 name (1), name (2), name (3) .....etc.

21.2 Declaring Arrays in Excel VBA

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

21.2(a) One Dimensional Array The syntax to declare a one dimensional array in Excel VBA is as follows:
Dim arrayName(index) as dataType or
Dim arrayName(first index to last index) as dataTypeFor example,
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 21.1

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)
```

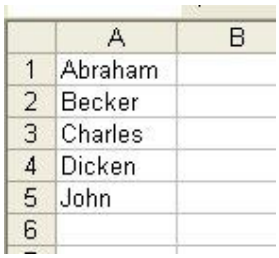
```
Next
End Sub
```

* 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.

Figure 21.1

The five names will be displayed in the spreadsheet as shown below:



	A	B
1	Abraham	
2	Becker	
3	Charles	
4	Dicken	
5	John	
6		

Figure 21.2

Example 21.2

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

```
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:

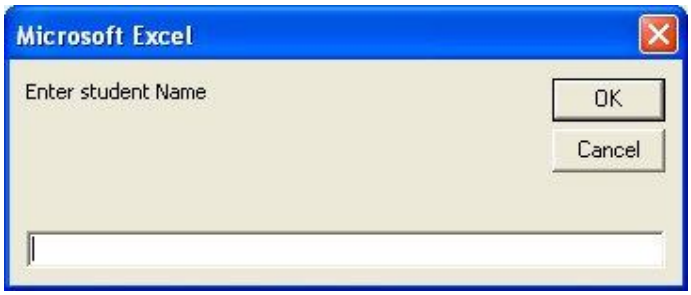


Figure 21.3

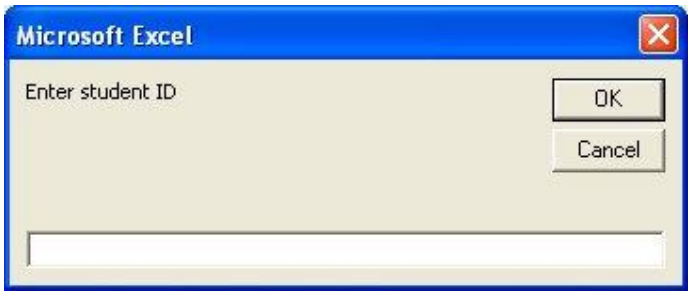


Figure 21.4

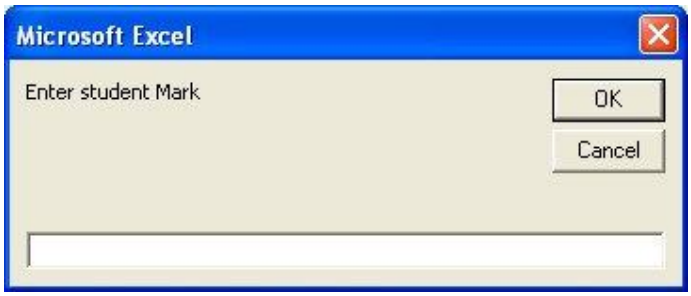


Figure 21.5

	A	B	C
1	Ali	A99	61
2	John	A100	75
3	Lee Dong	A101	90
4			
5			

Figure 21.6

21.2(b) Two Dimensional Array

Multidimensional arrays are often needed when we are dealing with more complex database, especially those that handle 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. 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 format 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 dimension array consists of 16 elements. These elements can be organized in a table form as shown in the table below:

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 21.3
If a company wants to track the performance of 5 salespersons over a period of 2 days, you can create a 5×2 array in Excel VBA, denoted by a 5X 2 table in a spreadsheet.
You can write the following VBA code:

```
Private Sub CommandButton1_Click()
Dim SalesVolume(2to 6, 2 to 3) as Single
Dim SalesPerson as Integer, Day as Integer
For SalesPerson=2 to 6
For Day=2 to3
SalesVolume(SalesPerson, Day)=inputbox("Enter Sales Volume")
Cells(SalesPerson, Day)=SalesVolume(SalesPerson,Day)
Next Day
Next SalesPerson
End Sub
```

When the user runs the program, the inputbox that will prompt the user to enter sales volume will appear 10 times, as shown in the Figure below :

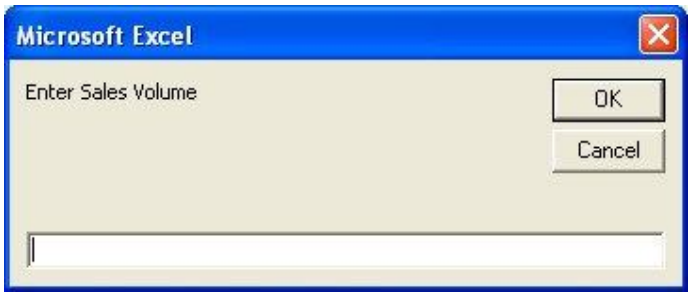


Figure 21.7

After all the sales Volumes are entered, the values in the spreadsheet are shown below:

	A	B	C	D
1	**Name**	**Day1**	**Day2**	
2	Abraham	100	110	
3	Becker	120	125	
4	Chan	90	85	
5	Elaine	150	160	
6	Florence	200	250	
7				

Figure 21.8

If you need to make sure the user enters the correct sales volume, you can change line 5 statement to

SalesVolume(SalesPerson, Day) = InputBox("Enter Sales Volume of " & "
SalesPerson " & (SalesPerson – 1) & " Day " & (Day – 1))

A clearer instruction will be shown as follows:

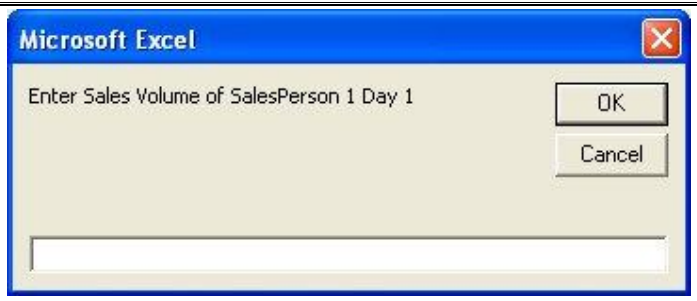

Microsoft Excel
Enter Sales Volume of SalesPerson 1 Day 1
OK
Cancel

Figure 21.9

Excel VBA Lesson 22: Creating Animation in Excel VBA

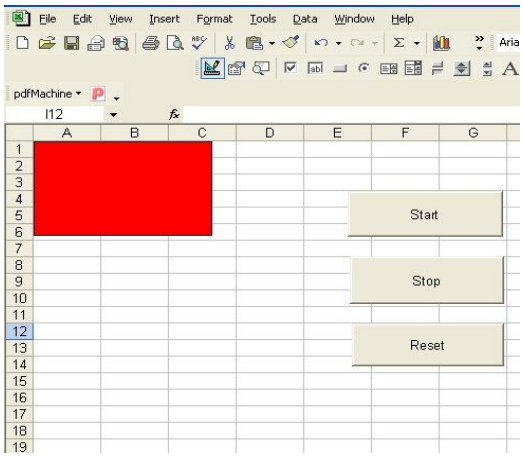
Beside creating Excel VBA code for mathematical and financial calculations, it is also possible to creating some fun applications in Excel VBA, including games and animation. Although professionals programmers might not be interested to write such applications, it is worth while trying them out as a hobby and for personal satisfaction.

Animation can be achieved by changing the position of an object continuously using a looping sub procedure. Two properties or functions that are required to change the positions or coordinates of the object are the Left and Top properties. The Left property specifies the distance of the left edge of the object in pixel from the left border of the screen and the Top property specifies the distance of the top edge of the object from the top border of the screen.

For for example, the following code makes the object move from left to right then back to left again repeatedly until the user press the stop button. The reset button move the object back to the starting position.

The code:

```
Private Sub StartButton_Click()
repeat:
With VBAProject.Sheet1.Image1
.Left = .Left + 1
DoEvents
If .Left > 200 Then .Left = 1
End With
GoTo repeat
End Sub
```



Please view the above animation live on YouTube

If you wish to move the object up and down, change the above code by replacing the property Left to Top, the code is as follows:

```
Private Sub StartButton_Click()
repeat:

With VBAProject.Sheet1.Image1
.Top= .Top+ 1
DoEvents
If .Top> 200 Then .Top = 1
End With

GoTo 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
.Left = .Left + 5

DoEvents
If .Top > 200 Then .Top = 1
If .Left > 200 Then .Left = 1

End With

GoTo repeat
End Sub
```