Chapter 1 Introduction to VBA

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 built-in Visual Basic Editor in Microsoft Excel, and you can use it to customize and extend capabilities of MS Excel. The applications you build with MS Excel is called Visual Basic for Applications, or simply VBA.

You can program VBA in every version of Microsoft Office, including MS Office 97, MS Office2000, MS Office2002, MS Office2003 and MS Office XP. The reason VBA is needed is due to the limitations in using the built-in functions of VB and macro recording. By using VBA, you can build some very powerful tools in MS Excel, including financial and scientific applications such as getting financial data from the Internet as well as linear programming.

There are two ways which you could program a 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 VB 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 toolbar and finally select the Control Toolbox after which the control toolbox bar will appear. Then you click on the command buttton and draw it on the spreadsheet.

Next, you click on the command button and the Visual Basic Editor will appear. Then you enter the statement as shown in the figure. The first statement will fill up cell A1 to cell A10 with the phrase "Visual Basic" while the second statement add the value in cell A11 and cell B11 and then show the sum in cell C11. It is that simple

Chapter 2 Working with Variables

2.1 The Concept of Variables

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 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 VBA, you have to follow a set of rules, as follows:

a) Variable Names

The following are the rules when naming the variables in VBA

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

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

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 VBA, one needs to declare the variables before using them by assigning names and data types. There are many 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 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

The nonnumeric 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 format 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, VB 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
```

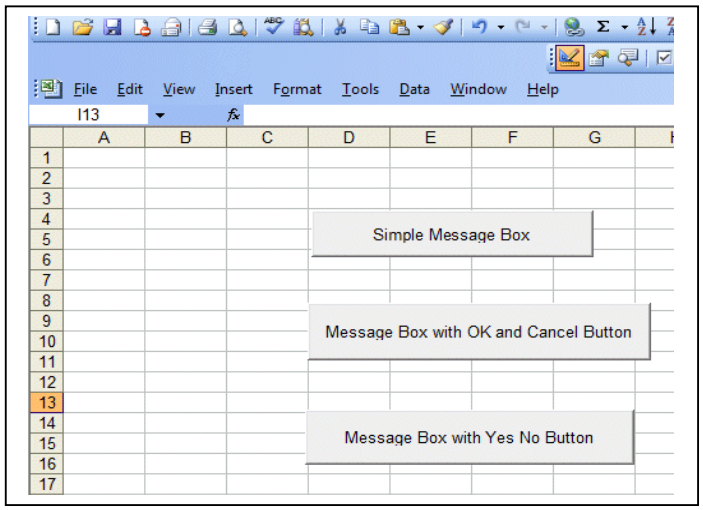
Chapter 3 Message Box

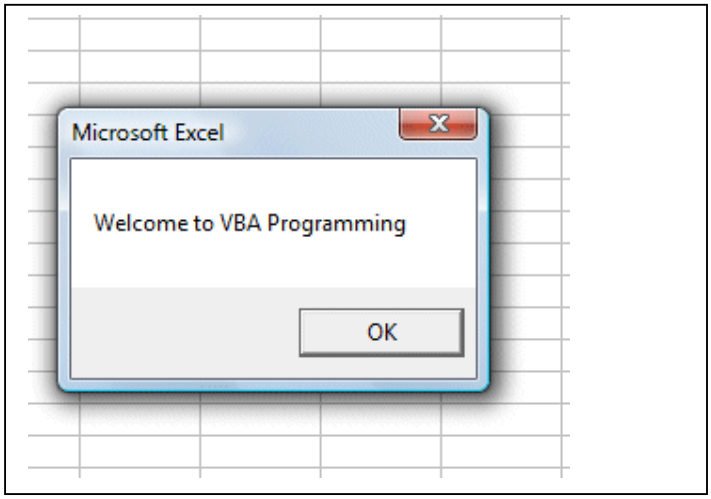
Yesterday I have shown that how we can display phrases in a range of cells and also perform arithmetic operations in MS Excel. Today, I shall demonstrate how we can display message boxes in a MS Excel worksheet . A message box normally act as a dialog box where users can interact with the computer, it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





This is the message box displayed by clicking the first message box
The codes are as follows:

```
Private Sub CommandButton1_Click()
MsgBox ("Welcome to VBA Programming")
End Sub
```

The code for the second button

```
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
```

The code for the third button:

```
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
```

The nessage box displayes Yes and No buttons

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

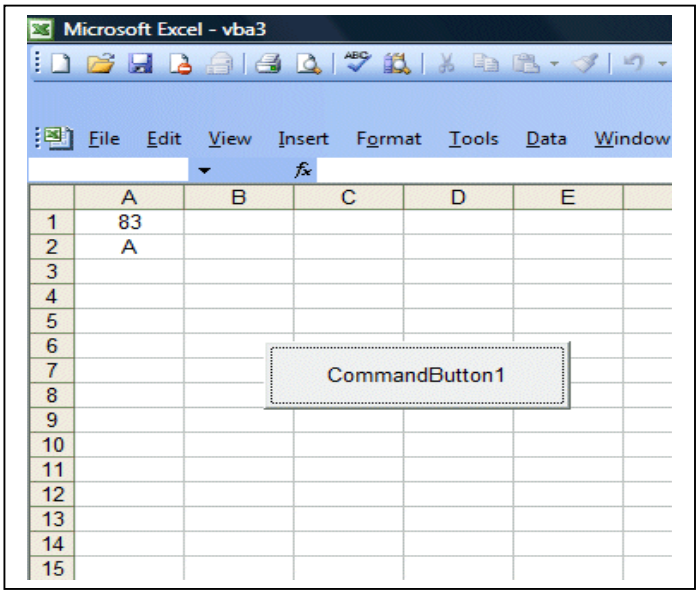
Visual Basic Editor in MS Excel is just as powerful as the stand alone Visual Basic compiler in the sense that you can use the same commands in programming. For example, you can use If..Then...Else to control program flow and display certain output based on certain conditions in MS Excel. Here, I am going to demonstrate the concept using one example.

In this program, you place the command button 1 on the MS Excel spreadsheet and go into the VB editor by clicking on the button. At the VB editor, key in the program codes as shown on the left.

I use randomize timer and the RND function to generate random numbers. In order to generate random integers between 0 and 100, I 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 INTERFACE



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
```

Chapter 5 For....Next Loop

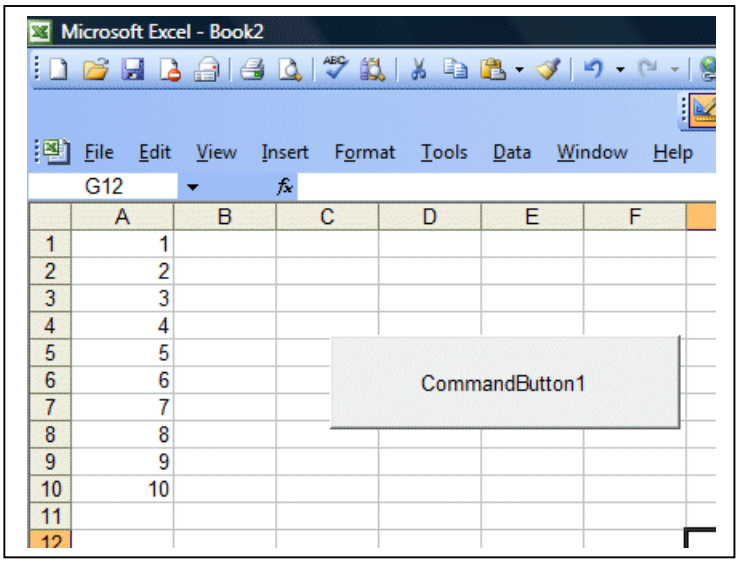
Looping is a very useful feature of Visual Basic because it makes repetitive works easier. There are two kinds of loops in Excel VBA, the For.....Next loop and the Do...Loop . To demonstrate the For....Next loop in Excel VBA, here are two examples.

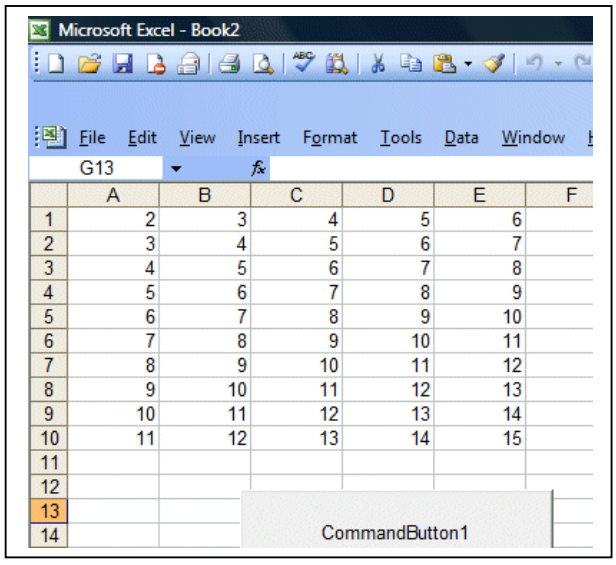
Example 1:

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

In this 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 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.

In example 2,we use the 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.





Codes of Example 2

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

```
Cells(i, j).Value = i + j
Next j
Next i
End Sub
```

Chapter 6 Do... Loop

In the previous chapter, you have learned to use the **For.....Next** loop to execute a repetitive process. In this chapter, you will learn about another looping method know as the **Do** Loop. There are four ways you can use the Do Loop as show below.

i) Do.....Loop While

(ii) Do until.....Loop

(iii) Do while.....Loop

(iv) Do.....Loop until

Example 1

```
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).

Example 2

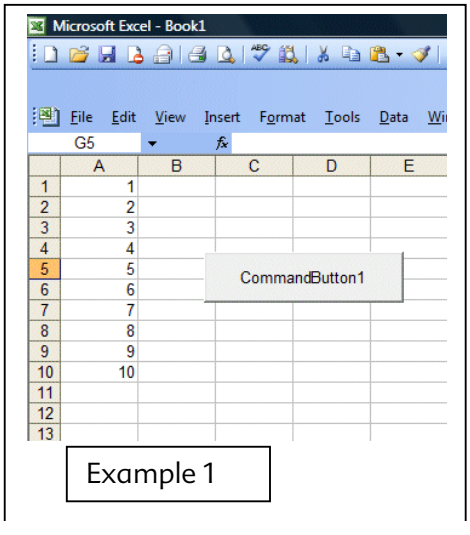
```
Private Sub CommandButton1_Click()
Dim counter As Integer
Do Until counter = 10
counter = counter + 1
Cells(counter, 1) = 11 - counter
Loop

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).

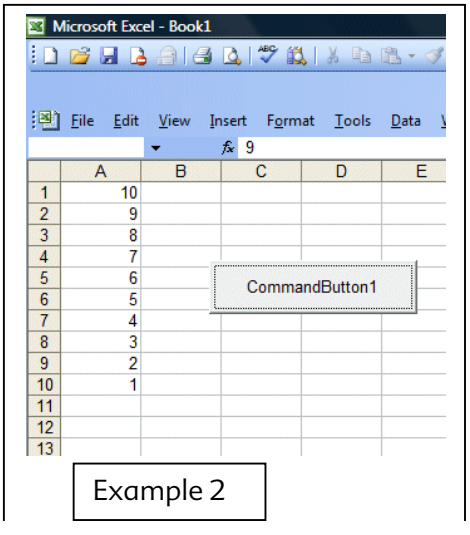
Example 3

```
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
```

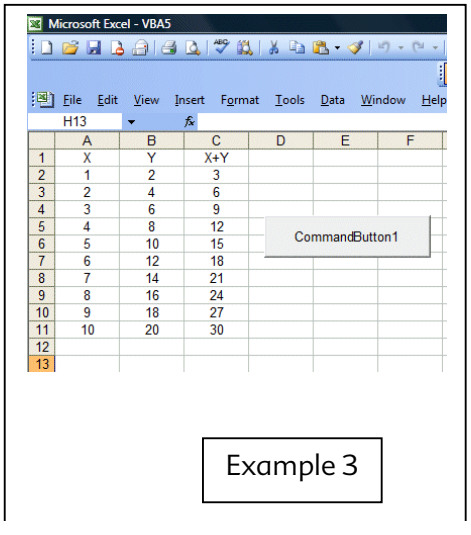
In this example, the program will display the values of X in cells(1,1) to cells(11,1). The value of Y is X^2 and the values are display in column 2, i.e. from cells(2,1) to cells(2,11). Finally, it shows the values of X+Y in column 3, i.e. from cells(3,1) to cells(3,11)



Example 1



Example 2



Example 3

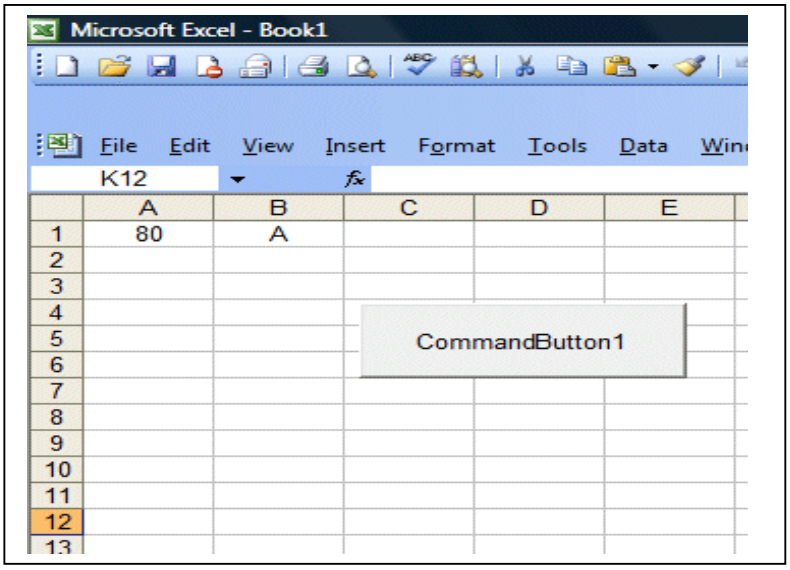
Chapter 7 Select Case.....End Select

Normally it is sufficient to use the conditional statement **If....Then....Else** for multiple options or selections programs. However, if there are too many different cases, the **If...Then...Else** structure could become too bulky and difficult to debug if problems arise. Fortunately, Visual Basic provides another way to handle complex multiple choice cases, that is, the **Select Case.....End Selec**t 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
```

In the following example, I will show you how to process the grades of students according to the marks given.

```
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.

The diagram on the lower left illustrates the output of this example.

Chapter 8 Font & Background Color

Today we will explore how to create VBA that can format the color of a MS Excel spreadsheet. Using Visual Basic codes, we can actually change the font color as well as the the background color of each cell effortlessly. Alright, I am going to creating a program that can create random font and background colors using a randomize process. Colors can be assigned using a number of methods in 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 format to set the font color is

cells(i,j).Font.Color=RGB(x,y,x), where x ,y , z can be any number between 1 and 255

For example

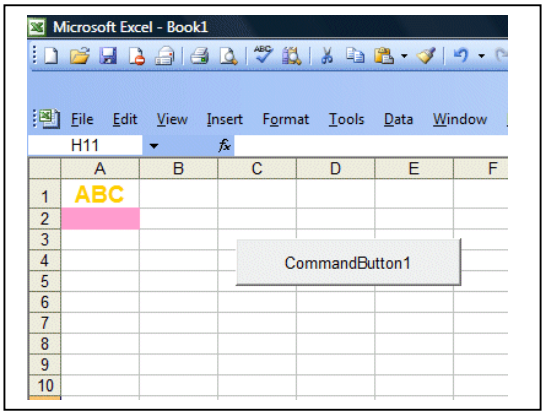
cells(1,1).Font.Color=RGB(255,255,0) will change the font color to yellow

The format 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

In the following 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.

```
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
```



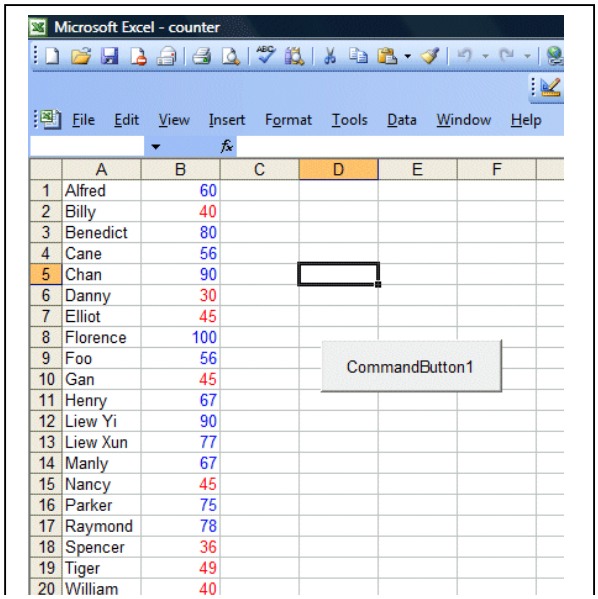
Explanation:
Rnd is a random number between 0 and 1
255* Rnd will produce a number between 0 and 255
Int(255*Rnd) will produce integers that take the values from 0 to 254
So we need to add 1 to get random integers from 0 to 255.
exampe;

Rnd=0.229
255*Rnd=58.395
Int(58.395)=58

Chapter 9 Creating a Counter

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

```
Private Sub CommandButton1_Click()
Dim i, counter As Integer
For i = 1 To 20

If Cells(i, 2).Value > 50 Then
counter = counter + 1
Cells(i, 2).Font.ColorIndex = 5
Else
'do nothing
Cells(i, 2).Font.ColorIndex = 3

End If
```



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

```
Next i
Cells(21, 2).Value = counter
Cells(22, 2).Value = 20 - counter

End Sub
```

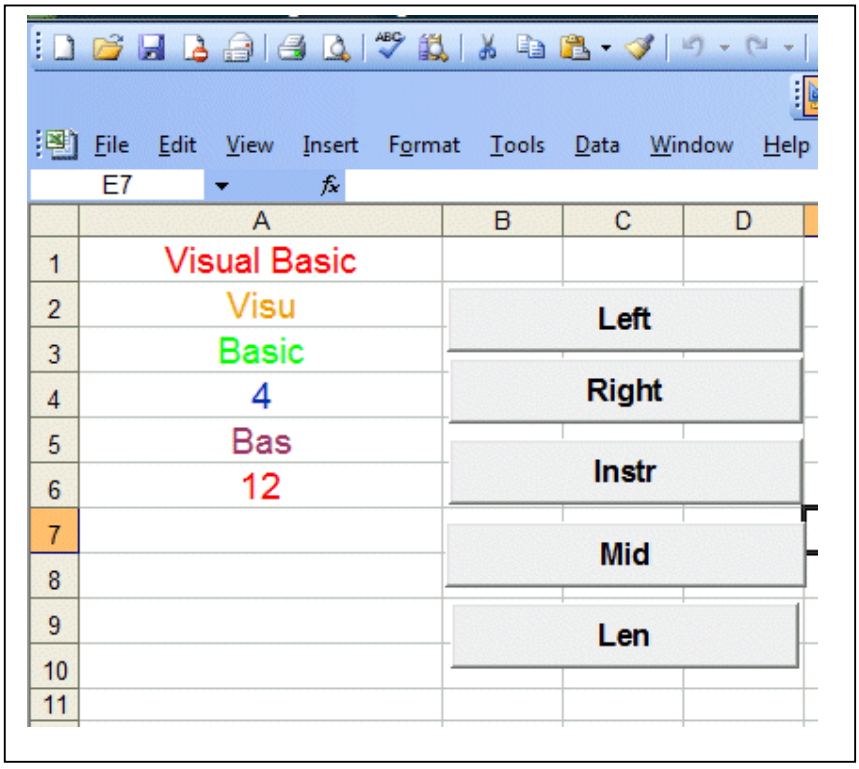
Explanation:

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

Chapter 10 String Handling

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 Visual Basic Editor. The following example illustrates the usage of all these functions

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

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

End Sub

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

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



End Sub

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

Explanation:

1. InStr is a function that looks for the position of a substring in a phrase.

InStr(phrase,"ual") will find the substring "ual" from "Visual Basic" and then return its position, in this case, it is 4 from the left.

2. Left is a function that extracts characters from a phrase, starting from the left. Left(phrase,4) means 4 characters are extracted from the phrase, starting from the leftmost position.

3. Right is a function that extracts characters from a phrase, starting from the Right. Right(phrase,5) means 5 characters are extracted from the phrase, starting from the rightmost position.

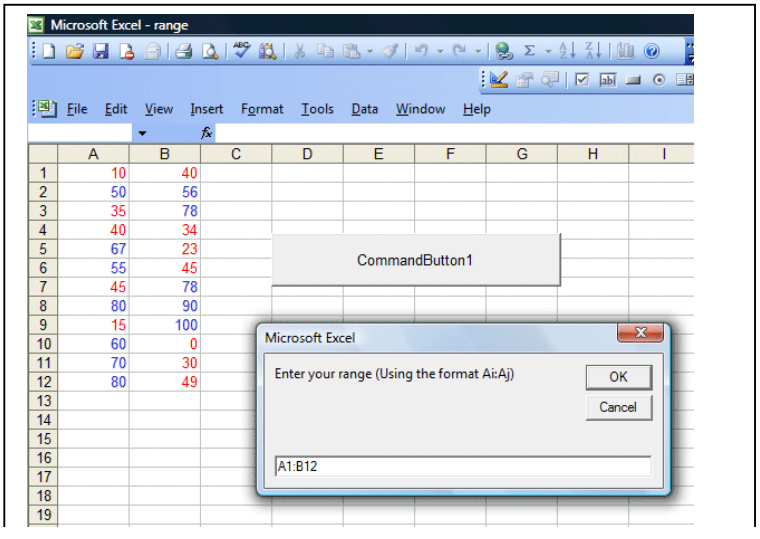
4. Mid is a function that extracts a substring from a phrase, starting from the posiiton specified by the second parameter in the bracket.Mid(phrase,8,3) means a substring of three characters are extracted from the phrase, starting from the 8th position from the left

5. Len is a function that return the length of a phrase.

Chapter 11 Range Selection & Manipulation

We can program a VBA that can select certain range of cells and at the same time perform certain tasks according to a set of conditions. In this example, I program the VBA 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.

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.

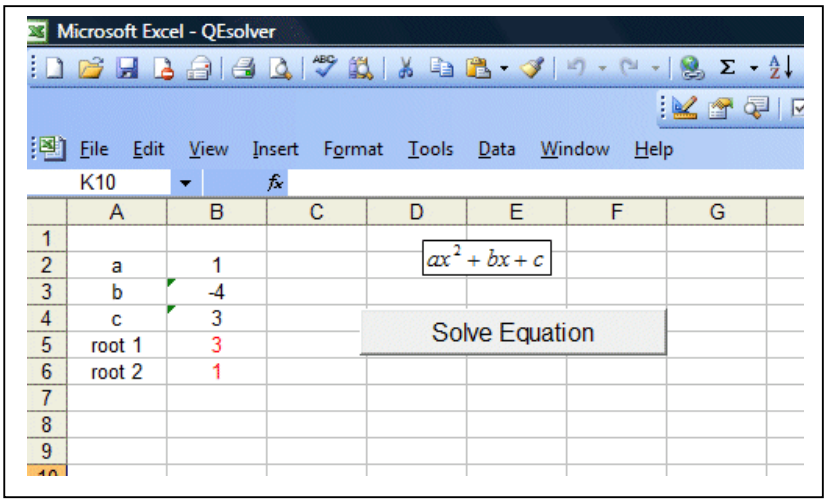
Once the input is obtained from the user, it is stored using the Set method and the Range function.

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.

Chapter 12 Creating a Quadratic Equation and Solver

I have presented to you a quadratic equation solver in standalone Visual Basic in December's VB Today. You can also create a similar program using MS Excel Editor. In fact, it is easier to do it in MS Excel as you just enter the values into the cells rather than having to create the text boxes. So for those of you who are without a copy of MS Visual Basic compiler, but you have MS Office, you can copy the codes and try this program out in your MS Excel




```
Private Sub CommandButton1_Click()

Dim a, b, c, det, root1, root2 As Single

a = Cells(2, 2)
b = Cells(3, 2)
c = Cells(4, 2)
det = (b ^ 2) - (4 * a * c)
If det > 0 Then
root1 = (-b + Sqr(det)) / (2 * a)
root2 = (-b - Sqr(det)) / (2 * a)
Cells(5, 2) = Round(root1, 2)
Cells(6, 2) = Round(root2, 2)

ElseIf det = 0 Then
root1 = (-b) / 2 * a
Cells(5, 2) = root1
Cells(6, 2) = root1
Else
Cells(5, 2) = "No root"
End If
End Sub
```

Explanation:

The format of the quadratic equation is as below:

ax^2+bx+c , where a,b,c are constants.

The number of roots depends on the determinant of b^2-4ac

If $b^2-4ac>0$ then there are two roots

If $b^2-4ac=0$ then there is only one root

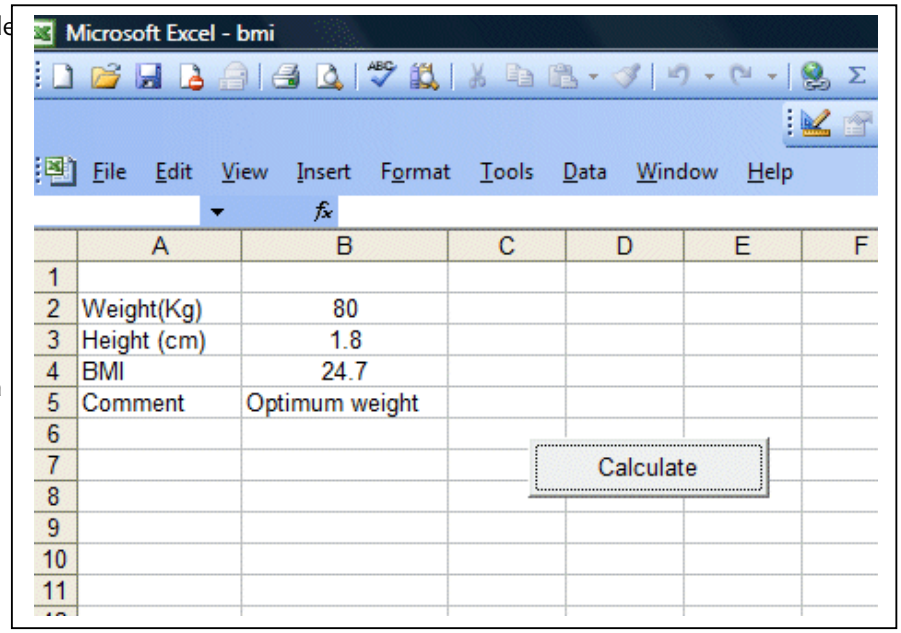
If $b^2-4ac<0$ then there is no root.

By making use the above conditions and employ the use of If....Then...Else statements, the program is able to solve the quadratic equation

Chapter 13 Creating a BMI Calculator

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. I have programmed a BMI calculator using VB6 professional, but now I will show you how to create a VBA BMI calculator in MS Excel.

```
Private Sub CommandButton1_Click()
Dim weight, height, bmi, x As Single

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



Explanation:

The formula for calculating BMI is

BMI=wieght(/(height^2)

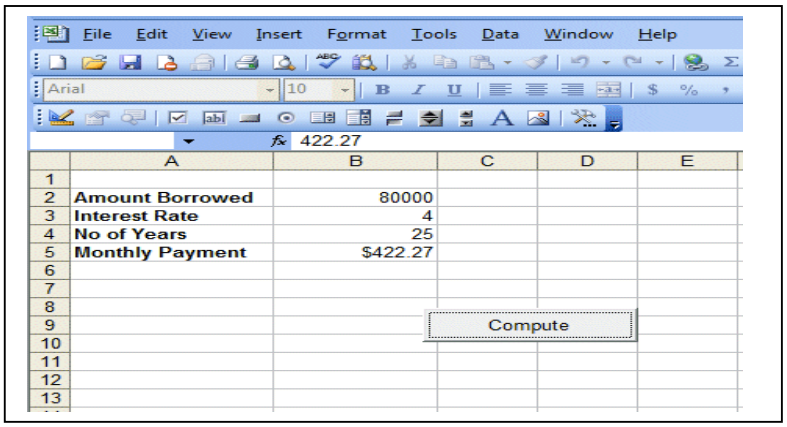
The function Round is to round the value to a certain decimal places. It takes the format Round(x,n), where n is the number to be rounded and n is the number of decimal places.

The second part of the program usees the If...Then..Else statement to evaluate the weight level.

Chapter 14 Creating a Financial Calculator

Excel VBA can be programmed to perform complex financial calculations. Here we have created a financial calculator to calculate the periodic payment for a loan taken from the bank. Below is the Excel VBA code for the financial calculator and its output interface.

```
Private Sub CommandButton1_Click()

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

End Sub
```



The formula to calculate periodic payment is payment=Initial Principal/PVIFA, where PVIFA is known as present value interest factor for an annuity . The formula to compute PVIFA is 1/i - 1/i(1+i)n where n is the number of payments. Normally you can check up a financial table for the value of PVIFA and then calculate the payments manually.

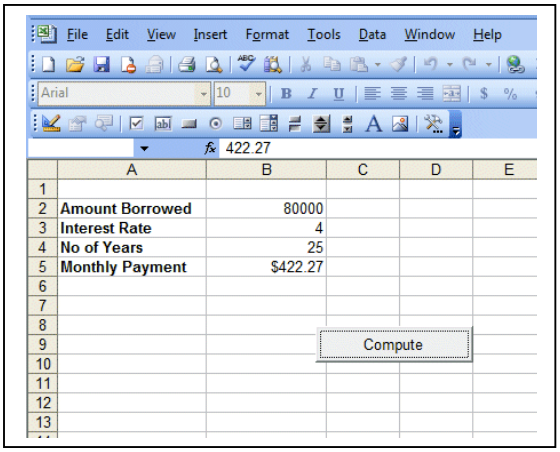
The function Format is to determine the number of decimal places and the use of the $ sign.

Chapter 15 Creating another Financial Calculator

In the previous chapter, we have shown you how to program a VBA program that can calculate monthly payment for a loan taken by a borrower. In this example, the financial VBA calculator is doing the same job but we use the built-in worksheet function, PMT. It is very much easier to program than the previous one. The format of this function is

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

Here is the VBA program:




```
Private Sub CommandButton1_Click()

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

End Sub
```

Explanation:

Normally people will key in the annual interest rate as an integer rather than in decimal form, so I need divide the rate by 100 and then divide again by 12 to get the monthly rate.

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

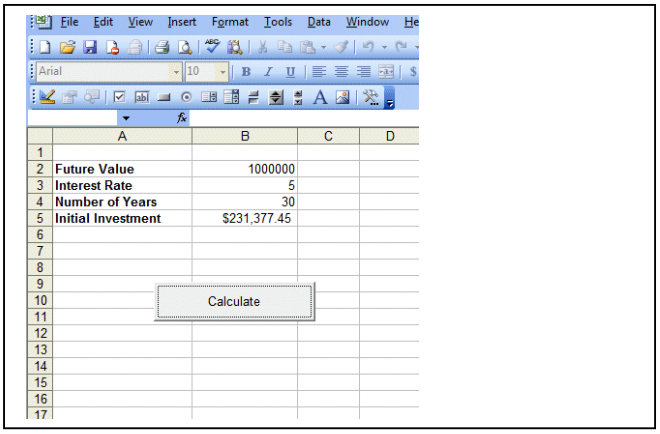
Chapter 16 Investment Calculation

This is a one-million-dollars puzzle In order to get one million dollars in the future, how much money do we need to invest now? To solve this puzzle, we need to calculate the initial investment based on the interest rate and the length of a period, usually in years. The formula is

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

where rate is the interest rate, N is the length of the period and amount is the amount borrowed.

Here is the VBA code:




```
Private Sub CommandButton1_Click()
Dim F_Money, Int_Rate, Investment As Double
Dim numYear As Single
F_Money = Cells(2, 2)
Int_Rate = (Cells(3, 2) / 100)
numYear = Cells(4, 2)
Investment = PV(Int_Rate, numYear, 0, F_Money, 1)
Cells(5, 2) = Format(-Investment, "$##,###,##0.00")

End Sub
```

Explanation:

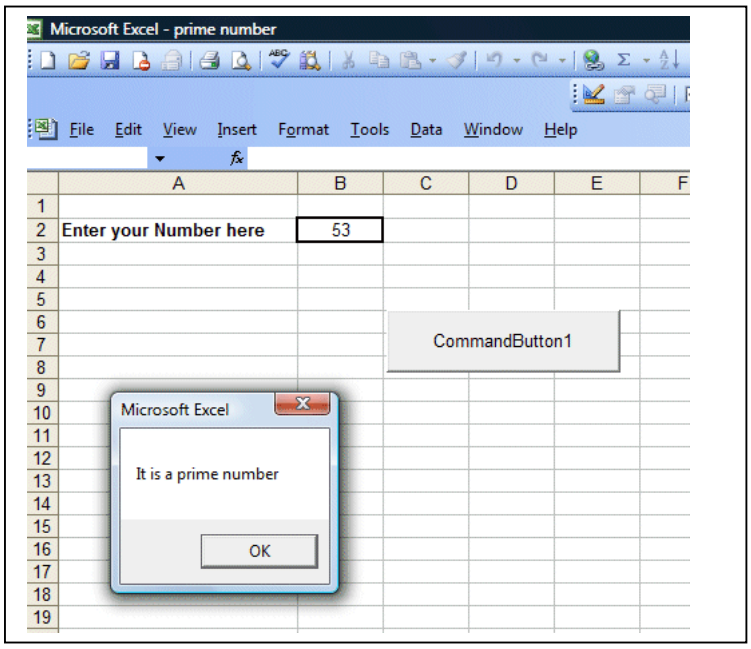
We have to divide the interest rate by 100 because it is expressed in percentage form so we have to convert it to decimal form. Annual payment is zero as we are interested only in the initial one-time payment in order to get one million dollars in the future.

We put a negative sign in front of the investment because this is the amount we have to pay.

Chapter 17 Prime Number Test

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

```
Private Sub CommandButton1_Click()

Dim N, D As Single
Dim tag As String
N = Cells(2, 2)

Select Case N
Case Is < 2
MsgBox "It is not a prime number"

Case Is = 2
MsgBox "It is a prime number"

Case Is > 2

D = 2
Do
If N / D = Int(N / D) Then
MsgBox "It is not a prime number"
tag = "Not Prime"
Exit Do
End If
D = D + 1
Loop While D <= N - 1

If tag <> "Not Prime" Then
MsgBox "It is a prime number"
End If
End Select
End Sub
```



Explanation:

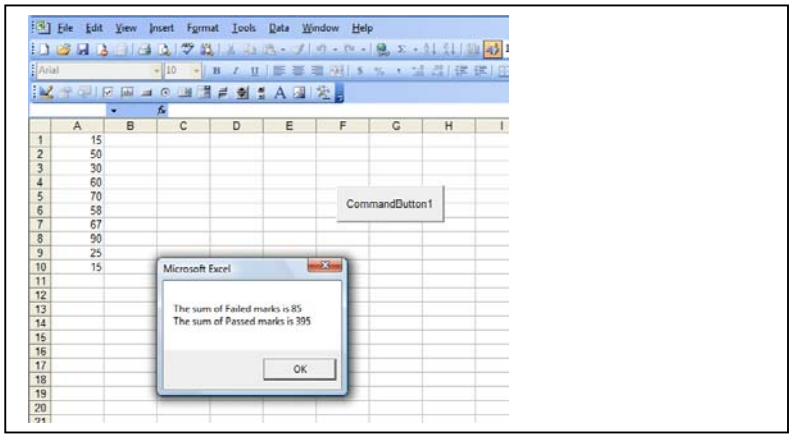
In this program, we use the Select Case .....End Select statement to determine whether a number entered by a user is a prime number or not. For case 1, all numbers that are less than 2 are not prime numbers. In Case 2, if the number is 2, it is a prime number. In the last case, if the number N is more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

Chapter 18 Selective Summation using VBA

In this lesson, we have created a VBA that can perform selective summation according to a set of conditions. For example, you might just want to sum up those figures that have achieved sales target and

vice versa. The VBA program I am showing you can sum up marks that are below 50 (which considered as failed) as well as those marks which are above 50 (which considered as passed). Here is the program

```
Private Sub CommandButton1_Click()
Dim rng As Range, i As Integer
Dim mark, sumFail, sumPass As Single
sumFail = 0
sumPass = 0
Set rng = Range("A1:A10")
For i = 1 To 10
mark = rng.Cells(i).Value
Select Case mark
Case Is < 50
sumFail = sumFail + mark
Case Is >= 50
sumPass = sumPass + mark
End Select
Next i
MsgBox "The sum of Failed marks is" & Str(sumFail) & vbCrLf & "The sum of Passed marks is" &
Str(sumPass)

End Sub
```



Explanation:

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

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

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

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

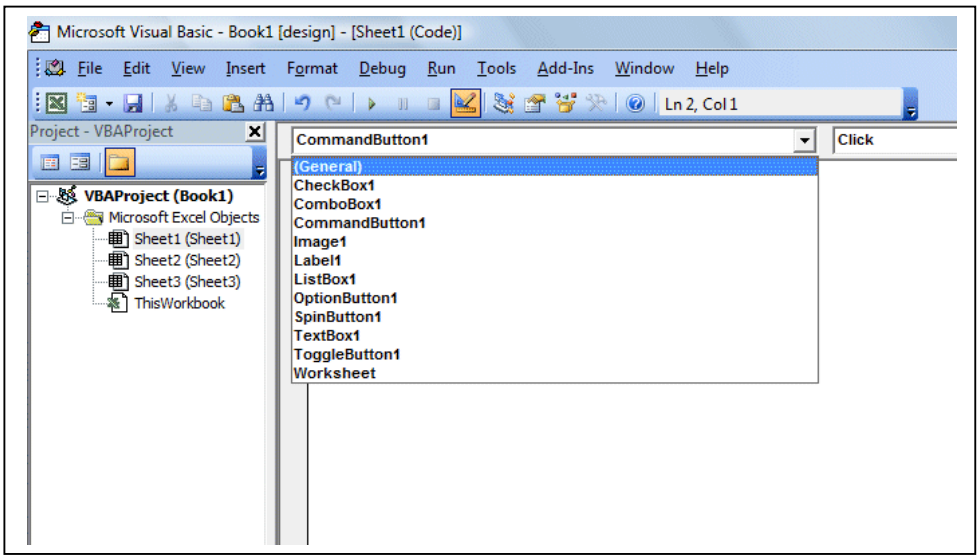
Finally, the results are shown in a message box

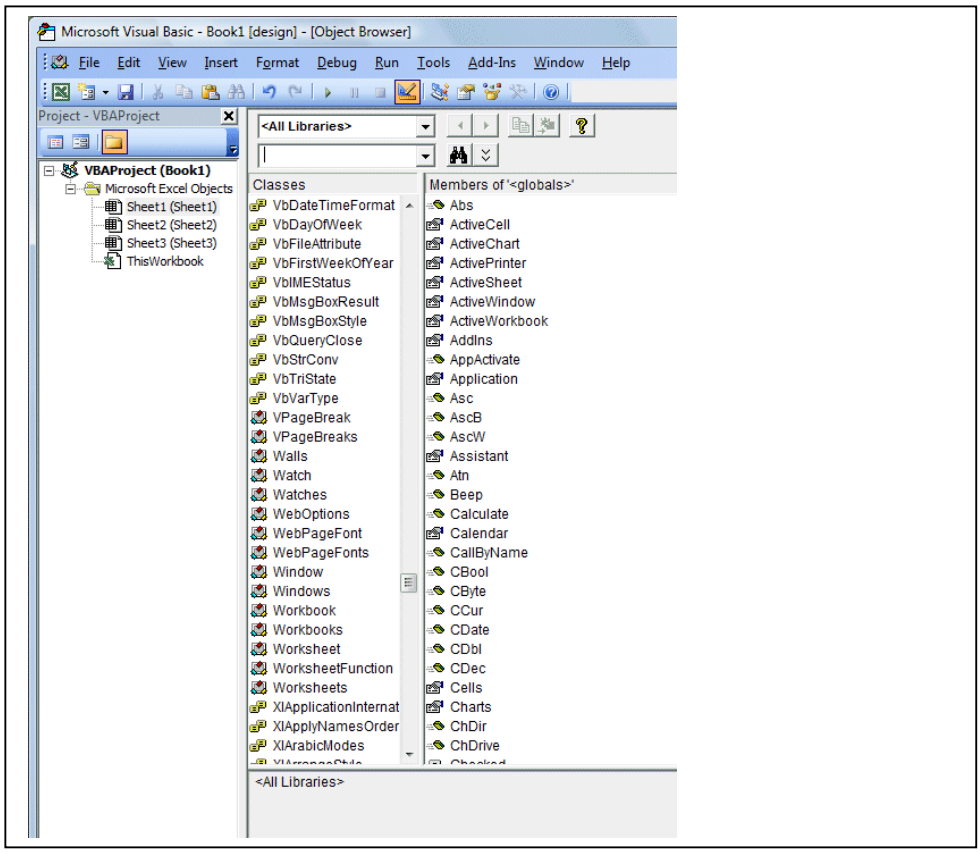
Chapter 19 Excel VBA Objects - I

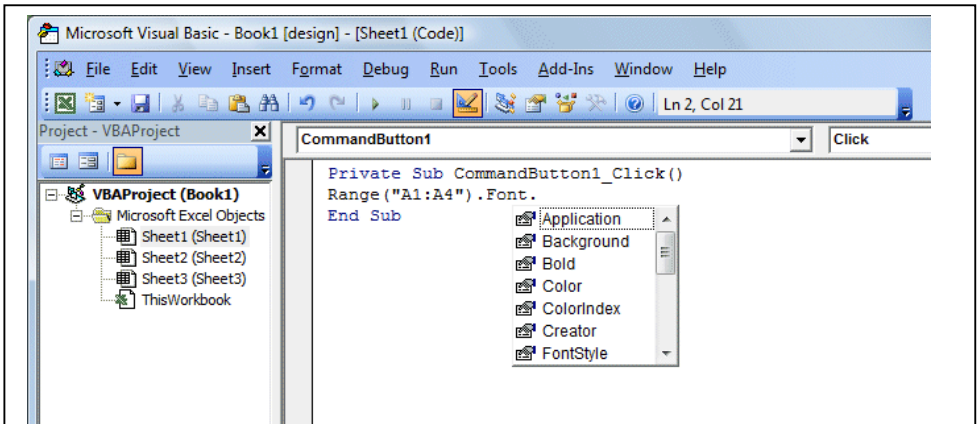
Objects

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. 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 VBA objects, you can insert a number of objects or controls into the worksheet, and click the command button to go into the code window. The upper left pane of the code window contains the list of objects you have inserted into the worksheet; you can view them in the dropdown dialog when you click the down arrow. The right pane represents the events associated with the objects, as shown in Figure 19.1 below.


Microsoft Visual Basic - Book1 [design] - [Sheet1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 2, Col 1
Project - VBAProject
CommandButton1
Click
(General)
CheckBox1
ComboBox1
CommandButton1
Image1
Label1
ListBox1
OptionButton1
SpinButton1
TextBox1
ToggleButton1
Worksheet
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook


Microsoft Visual Basic - Book1 [design] - [Object Browser]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Project - VBAProject
<All Libraries>
Classes
Members of '<globals>'
VbDateTimeFormat
VbDayOfWeek
VbFileAttribute
VbFirstWeekOfYear
VbIMEStatus
VbMsgBoxResult
VbMsgBoxStyle
VbQueryClose
VbStrConv
VbTriState
VbVarType
VPageBreak
VPageBreaks
Walls
Watch
Watches
WebOptions
WebPageFont
WebPageFonts
Window
Windows
Workbook
Workbooks
Worksheet
WorksheetFunction
Worksheets
XlApplicationInternat
XlApplyNamesOrder
XlArabicModes
Abs
ActiveCell
ActiveChart
ActivePrinter
ActiveSheet
ActiveWindow
ActiveWorkbook
AddIns
AppActivate
Application
Asc
AscB
AscW
Assistant
Atn
Beep
Calculate
Calendar
CallByName
CBool
CByte
CCur
CDate
CDbl
CDec
Cells
Charts
ChDir
ChDrive
<All Libraries>
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook


Microsoft Visual Basic - Book1 [design] - [Sheet1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 2, Col 21
Project - VBAProject
CommandButton1
Click
Private Sub CommandButton1_Click()
Range("A1:A4").Font.
End Sub
Application
Background
Bold
Color
ColorIndex
Creator
FontStyle
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook

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 19.1

```
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 19.2

```
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 sort of overwrite the range specified by the Range object.

Another object is font that belong 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 displayed 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 19.3

Chapter 20 Excel VBA Objects - II

Methods

Besides having properties, Excel VBA objects usually also have methods. Methods normally do something or perform certain operations. For example, ClearContents is a method of the range object. It clears the contents of a cell or a range of cells. You can write the following code to clear the contents:

Example 20.1

```
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 20.2

Example 20.2

```
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 20.3

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

Example 20.4

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 20.5

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 format 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 20.6

```
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 20.7

```
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 20.8

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
```

Chapter 21 VBA Procedures Part - 1

The Concept of Functions

A VBA procedure is a block of code that performs certain tasks. We have actually learned about VBA procedures in our previous chapters, but all of them are event procedures. Event procedures are VBA programs that are associated with VBA objects such as command buttons, checkboxes, and radio buttons. However, we can also create procedures that are independent from the event procedures. They are normally called into the event procedures to perform certain tasks. There are two types of the aforementioned procedures, namely Functions and Sub Procedures. In this chapter, we will discuss functions. We will deal with Sub Procedures in the next chapter.

Example 21.1

In this example, a function to calculate the area of a rectangle is created. There are two arguments needed, one to accept the value of width and the other is to accept the value of height. Note that the function Area_Rect is called from the event procedure (clicking the command button) and the values to be passed to the arguments are enclosed in the parentheses.

```
Private Sub CommandButton1_Click()

Dim a As Variant, b As Variant

a = InputBox("Enter Width of Rectangle")

b = InputBox("Enter Height of Rectangle")
```

```
MsgBox "The area of the rectangle is” & Area_Rect(a, b)

End Sub

Function Area_Rect(x As Variant, y As Variant) As Variant

Area_Rect = x * y

End Function
```

We can also create a user-defined function to be used just as the built-in functions by inserting a module in the Visual Basic Editor and enter the function code there. After creating the function, we can then return to the spreadsheet and use this function as any other built-in functions. To insert the module, click on Tool in the menu bar, select Macro and the click on Visual Basic Editor, as shown in Figure 21.1

21.2 Types of Functions

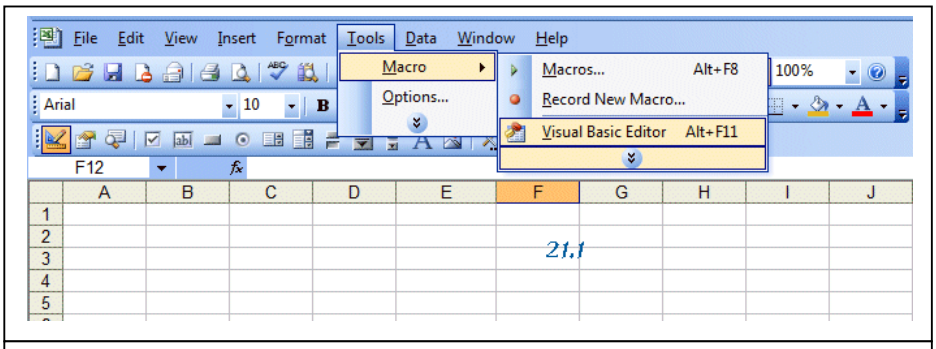
There are two types of Excel VBA functions; one is the built-in functions while the other one is the user-defined functions. We can use built-in functions in Excel for automatic calculations. Some of the Excel VBA built-in functions are Sum, Average, Min (to find the minimum value in a range), Max (To find the maximum value in a range), Mode, Median and more. However, built-in functions can only perform some basic calculations, for more complex calculations, user-defined functions are often required. User-defined functions are procedures created independently from the event procedures. A Function can receive arguments passed to it from the event procedure and then return a value in the function name. It is normally used to perform certain calculations.

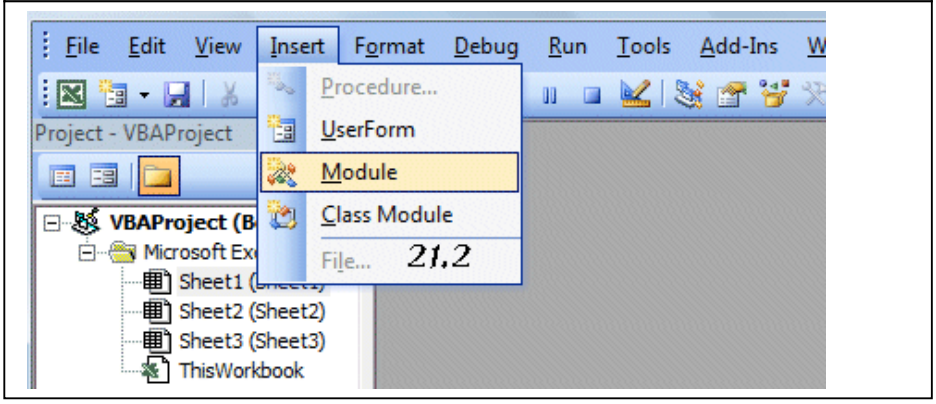
21.3 Writing Function Code

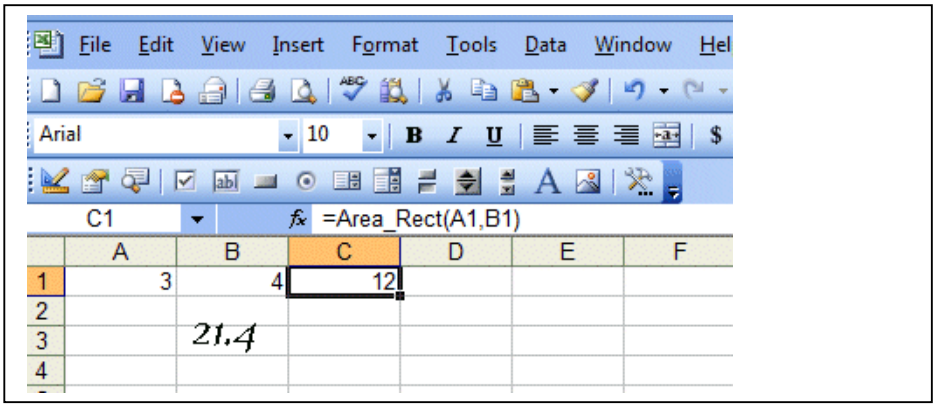
VBA Function begins with a Function statement and ends with an End Function statement. The program structure of a Function is as follows:

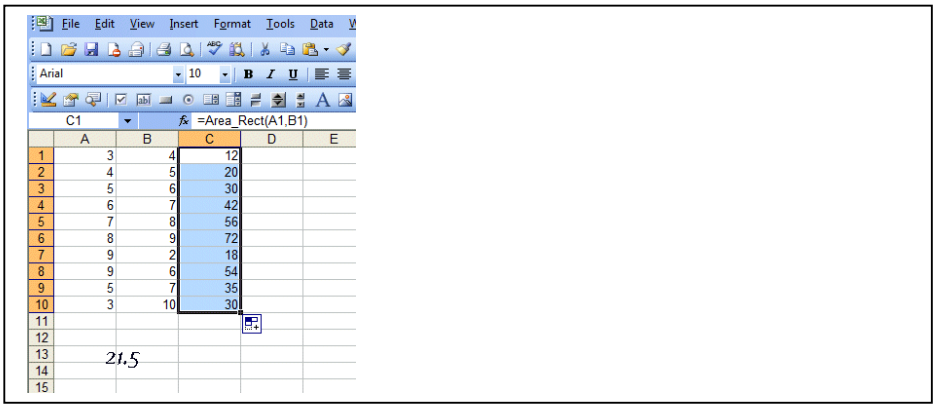
```
Function FunctionName (arguments) As DataType

Statements

End Function
```

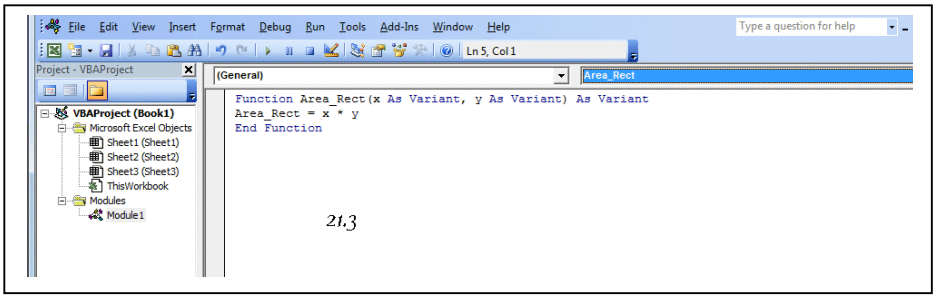
In Excel VBA, when you type the Function statement, the End Function statement will automatically appear.


File Edit View Insert Format Tools Data Window Help
Macro
Options...
Macros... Alt+F8
Record New Macro...
Visual Basic Editor Alt+F11
100%
Arial
10
F12
21.1


File Edit View Insert Format Debug Run Tools Add-Ins
Procedure...
UserForm
Module
Class Module
File...
Project - VBAProject
VBAProject (B
Microsoft Ex
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook
21.2


File Edit View Insert Format Tools Data Window Hel
Arial
10
C1
=Area_Rect(A1,B1)
3
4
12
21.4


File Edit View Insert Format Tools Data
Arial
10
C1
=Area_Rect(A1,B1)
21.5


File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Type a question for help
Ln 5, Col 1
Project - VBAProject
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook
Modules
Module1
(General)
Area_Rect
Function Area_Rect(x As Variant, y As Variant) As Variant
Area_Rect = x * y
End Function
21.3