VBA Message boxes and Input boxes

Objectives: At the end of this lesson you shall be able to
- **state the uses of message boxes and input boxes in VBA**
- **describe the msgbox method and msgbox function**
- **describe the inputbox method and inputbox function.**

Introduction

Many applications depend on data input from users to take the necessary action. Excel VBA has very useful functions that allow you to gather user input for your applications. VBA allows you to create message boxes, user input forms and input boxes to get user input.VBA message boxes provide a way to give information to a user and get information from a user while the program is running. The input Box function can be used to prompt the user to enter a value.

Message Box

In VBA Message Boxes fall into two basic categories, the MsgBox method and the MsgBox function.

The MsgBox Method

The message box method is used to display a pre- defined message to the user. It also contains a single command button "OK" to allow the user to dismiss the message and they must do so before they can continue working in the program.

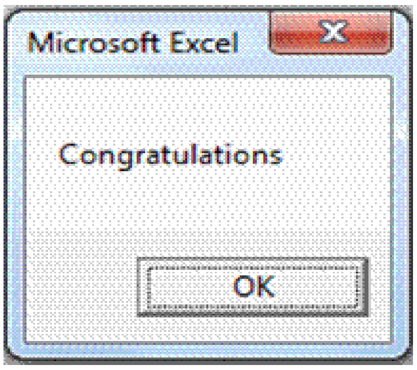
The basic form of the Message Box (msgbox) in VBAis :Msgbox("message")

Example:

```
Sub result()

Msgbox("congratulations")

End sub
```

This displays a message box as shown in Fig 1

Fig 1


Microsoft Excel

Congratulations

Customize the buttons in a VBA message box

The Msgbox() can be customized by changing the buttons and icons placed on it.

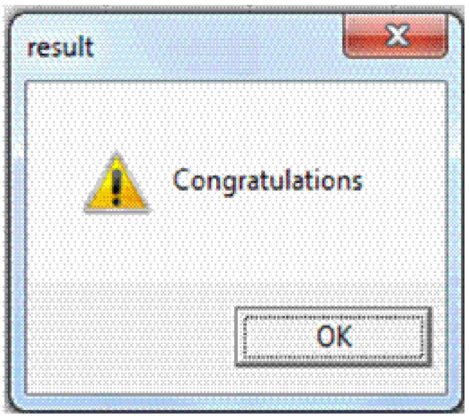
A list of various buttons and icons that can be used in the VBA message box is shown in the Table 1.

For ex. to add an icon and a title to the Msgbox() we can write the following code

```
Sub test()

Dim n As Integer

n = MsgBox("Congratulations", vbExclamation, "result")

End Sub
```

This will produce the following result as in Fig 2.

Fig 2


result

Congratulations

The MsgBox Function

The MsgBox Function displays a message in a dialog box, waits for the user to click a button, and then returns an integer indicating which button was clicked by the user.The syntax of the Msgbox() function is :

Return value = MsgBox(Prompt, Button and Icon types, Title, Help File, Help File Context)

Table 1

Constant	Description
vbOKOnly	It displays a single OK button
vbOKCancel	It displays two buttons OK and Cancel.
vbAbortRetryIgnore	It displays three buttons Abort, Retry, and Ignore.
vbYesNoCancel	It displays three buttons Yes, No, and Cancel.
vbYesNo	It displays two buttons Yes and No.
vbRetryCancel	It displays two buttons Retry and Cancel.
vbCritical	It displays a Critical Message icon.
vbQuestion	It displays a Query icon.
vbExclamation	It displays a Warning Message icon.
vbInformation	It displays an Information Message icon.
vbDefaultButton1	First button is treated as default.
vbDefaultButton2	Second button is treated as default.
vbDefaultButton3	Third button is treated as default.
vbDefaultButton4	Fourth button is treated as default.
vbApplicationModal	This suspends the current application till the user responds to the message box.
vbSystemModal	This suspends all the applications till the user responds to the message box.
vbMsgBoxHelpButton	This adds a Help button to the message box.
VbMsgBoxSetForeground	Ensures that message box window is foreground.
vbMsgBoxRight	This sets the Text to right aligned
vbMsgBoxRtlReading	This option specifies that text should appear as right-to-left.

Where:

Return Value: Indicates the action the user took when the message box was shown to him/her.

Prompt : It is the message contained in the main body of the message box.

Button and Icon Types : This specifies the set of buttons & Icons and their placement as they would appear to the user.

Help File : This is the path to a help file that the user can refer to on this topic.

Help File Context : This is the pointer to that part of the help file that specifically deals with this message.

Values returned by MsgBox Function:

VBA MsgBox function returns a value based on the user input. These values can be anyone of the ones shown in Table 2.

A Msgbox function example is shown in the code mentioned below.

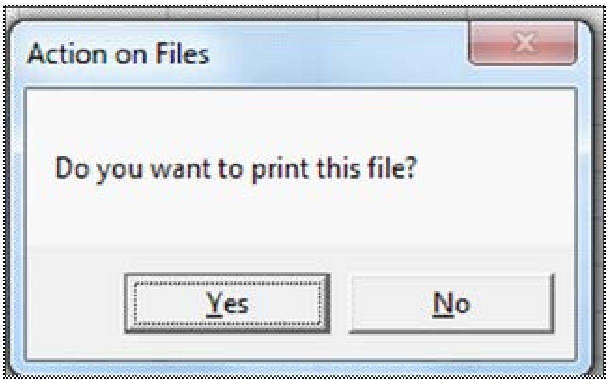
```
Sub test()

Dim n As Integer

 n = MsgBox("Do you want to print this file?", vbYesNo,
"Action on Files")

End Sub
```

Table 2

Value	Description
1	Specifies that OK button is clicked.
2	Specifies that Cancel button is clicked.
3	Specifies that Abort button is clicked.
4	Specifies that Retry button is clicked.
5	Specifies that Ignore button is clicked.
6	Specifies that Yes button is clicked.
7	Specifies that No button is clicked.

This will produce the result as in Fig 3.

Fig 3


Action on Files

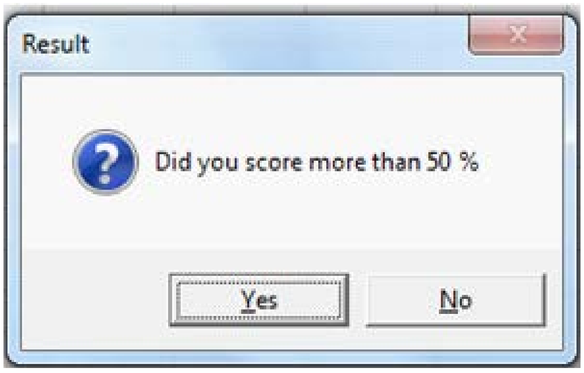
Do you want to print this file?

Reading the Msgbox() return values

Based on the value returned by the MsgBox(), decisions can be made.

For ex, the code mentioned here will display the message box, and when the user clicks "Yes" it will display a congratulatory message. If the user clicks "No" another message "Better Luck Next time" will appear as shown in Fig 4.

Fig 4


Result

Did you score more than 50 %

```
Sub test()

Dim n As Integer

n = MsgBox("Did you score more than 50 % ", vbYesNo + vbQuestion, "Result")

If n = 6 Then

MsgBox ("Congratulations")

Else

MsgBox ("Better Luck Next Time")

 End If

End Sub
```

Input box

For accepting the input from the user the Input box is used in two ways- The Input Box Function and the Input Box Method.The InputBox method differs from the InputBox function in that it allows selective validation of the user's input, and it can be used with Microsoft Excel objects, error values, and formulas.

Note that Application.Input Box calls the Input Box method; Input Box with no object qualifier calls the InputBox function.

Input Box Function

The Input Box Function displays a dialog box for user input. It returns the information entered in the dialog box. The syntax for the InputBox function is:

InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])

In its simplest form , the input box function looks like:n = Inputbox("Enter your Age")

The InputBox Method

When we precede the Input Box Function with "Application" we get an InputBox Method that will allow us to specify the type of info that we can collect. Ie. Application.InputBox

Its Syntax is :Input Box(Prompt, Title, Default, Left, Top, HelpFile, HelpContextId, Type)

The Prompt, Title and Default are the same as in the InputBox Function. However, it is the last argument "Type" that allows us to specify the type of data we are going to collect. These are as shown below.

Type:=0 A formula

Type:=1 A number

Type:=2 Text (a string)

Type: = 4 A logical value (True or False)

Type: = 8 A cell reference, as a Range object

Type: = 16 An error value, such as #N/A

Type := 64 An array of values

The following is an example of an InputBox method

```
Sub test()

Dim n As Integer

n = Application.InputBox("Enter you age", "Personal Details", , , , , , 1)

 'Exit sub if Cancel button used

If n > 60 Then

MsgBox "You are eligible for senior citizen's concession"

Else

MsgBox ("No concession")

End If

End Sub
```

Decision making statements in VBA

Objectives: At the end of this lesson you shall be able to
- **describe the decision making process using the “if... Then” statement**
- **describe the use of “ladder off” and “nested if” statement**
- **explain the use of the “select ....case” statements.**

Introduction

In a program a set of statements are normally executed sequentially in the order in which they appear. This happens when no decision making or repetitions are involved. But in reality, there may be a number of situations where we may have to change the order of execution of statements based on certain conditions being true or false. Some of the examples may be:

a To decide if a trainee is to be declared "Passed" or "Failed".

b To display the Grade achieved by a student.

c To accept input only of a particular data type like numbers.

d To decide if a number is prime or not.

e To decide if a string is a palindrome or not.

f To calculate pay, tax, commission etc. based on certain conditions etc.

g To repeat an action a certain number of times or till a certain limit is reached.

Decision making process can solve practical problems intelligently and provide useful output or feedback to the user. In order to control the program flow and to make decisions, we need to use the conditional operators and the logical operators together with the If control structure.

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is found to be true, and other statements to be executed if the condition is found to be false.Table 1 shows the commonly used decision making statements in VBA.

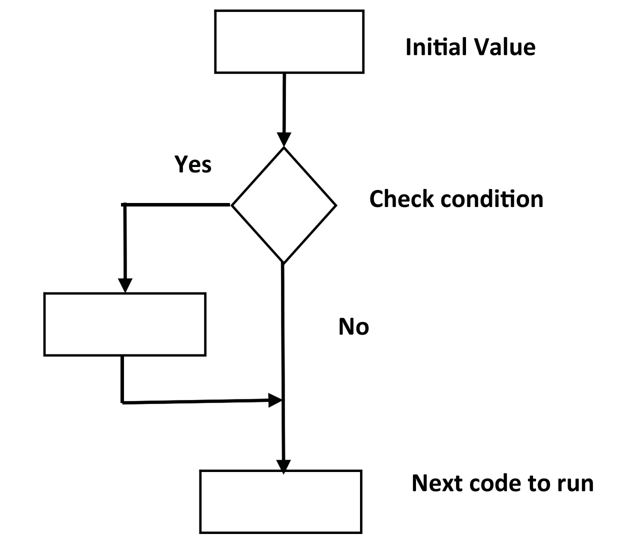
The If … Then Statement

It is the simplest form of control statement, frequently used in decision making and changing the control flow of the program execution. Syntax for if-then statement is:

```
If CONDITION Then

' code if the condition is met

End If
```

The flow chart for a typical If statement is shown in Fig 1.

Fig 1



Here condition refers to an expression which results in a Boolean type result, ie. True or False.For ex. the statement "if age <18" will test if the value of the variable "age" is less than 18 or not. If the condition evaluates to true, then the block of code inside the If statement will be executed. For example:

```
If (age < 18) Then

debug.print "Not Eligible"

End If
```

The following example tests the value of the number in the textbox and takes a decision.

```
Private Sub Button1_Click()

Dim n As Integer

'Enter the number of items sold by the agent

n = val(TextBox1.Text)

If n> 100 Then

Label1.Caption = " You are entitled for a commission of Rs. 10000"
```

End If

End Sub

The If Then…. Else Statements

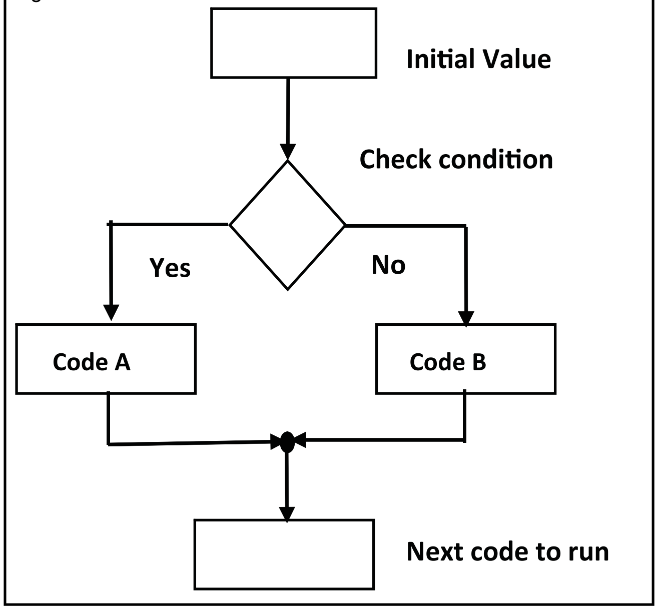
When an action has to be taken if the condition returns true and another action if the condition returns false, then we use the If Then…. Else Statements.

The syntax for the If Then … Else statements is as follows

```
If CONDITION Then

' code if the condition is met

Else

' code if the condition is not met

End If
```

The flow chart for a typical If Then … Else structure is as shown in Fig 2.

Fig 2



The example of an If Then … Else structure is shown below. This program tests if the Taxable Income entered by the user is less than 250000 or not. If yes, a message box appears stating that the user need not pay Income Tax. Else, another message box tells the user to pay the tax.

```
Sub test()

Dim income As Long

income = Application.InputBox("Enter you Taxable income")

If income< 250000 Then

MsgBox "You need not pay any income tax"

Else

Msg Box ("You must pay income tax")

End If

End Sub
```

Using Multiple If Statements

Sometimes the condition being tested is to be evaluated not just for returning "True" or "False" based on one condition, but for multiple conditions too. In such cases the multiple If Then …. Else statements can be used. They can be used in two ways:

1 Ladder If and

2 Nested if

Ladder If statements

The ladder if statements can be used to test if a condition1, condition2 … etc is met, and decision be taken based on which condition is met. The typical syntax of a Ladder If structure is:

```
if(boolean_expression 1)

{

/* Executes when the boolean expression 1 is true */

}

else if( boolean_expression 2)

{

 /* Executes when the boolean expression 2 is true */

}

else if( boolean_expression 3)

{

 /* Executes when the boolean expression 3 is true */

}

else

{

 /* executes when the none of the above condition is true */

}
```

The following is an example of a ladder if structure.

```
Sub grades()

Dim marks As Integer

marks = InputBox("Enter you marks")

If marks >= 80 Then

MsgBox "Distinction"

ElseIf marks >= 70 Then

MsgBox "A Grade"

ElseIf marks >= 60 Then

MsgBox "B Grade"

ElseIf marks >= 40 Then

MsgBox "C Grade"

Else

MsgBox "Failed"

End If

End Sub
```

This program would display the grade based on the marks entered by the user.

Nested If statements

Sometimes it is required to evaluate one condition only if an earlier condition is met. In such cases an If Then statement can be placed inside an outer If Then statement. This type of structure is also called a Nested If structure.The syntax of a nested if structure is as follows:

```
If(Boolean_expression 1)

{

//Executes when the Boolean expression 1 is true

If(Boolean_expression 2)

{

//Executes when the Boolean expression 2 is true

}

}
```

For ex. A certain recruitment condition states that a candidate to be declared eligible must have a minimum of 5 years' experience **and also** must have scored atleast 75% marks in the exam. In such a case, the first condition to be tested is for experience >= 5 years andonly if this condition is met, the second condition is to be evaluated.

If the first condition is not met, the control jumps to the statement after the End if statement. The following code is an example for the mentioned example.

```
Sub job_test()

Dim experience, marks As Integer

experience = InputBox("Enter your work experience in years")

If experience >= 5 Then

marks = InputBox("Enter you marks percentage")

If marks >= 75 Then

MsgBox (" You are eligible for the post")

Else

MsgBox (" You are NOT eligible for the post")

End If

Else

MsgBox (" You are NOT eligible for the post")

End If

End Sub
```

Using Logical operators in If Structure

The Logical operators And, Or and Not can be used in If structure and produce the same results as those produced in Nested If Structures.

For ex. the above mentioned condition can be evaluated using the And operator in the conditional statement.

```
Sub job_test()

Dim experience, marks As Integer

experience = InputBox("Enter your work experience in years")

marks = InputBox("Enter you marks percentage")

If experience >= 5 And marks >= 75 Then

MsgBox (" You are eligible for the post")

Else

MsgBox (" You are NOT eligible for the post")

End If

End Sub
```

Select...Case

Another way to implement decision making in your VBA code is to use a Select...Case statement. Select...Case statements can be used to easily evaluate the same variable multiple times and then take a particular action depending on the evaluation.

It is always a good practice to use Select Case Statement when multiple If-Else conditions are involved. As the number of If-Else conditions increases, debugging and understanding all the flow becomes a tedious job.

The syntax for a Select...Case statement is:

```
Select Case VARIABLE

Case VALUE1

' code to run if VARIABLE equals Value1

Case VALUE2

' code to run if VARIABLE equals Value2

Case Else

' code to run for remaining cases
```

End Select

For Ex. This program asks the user to type the name of the game and displays the number of players for the game.

```
Sub players()
Dim game As String
game = InputBox("enter the name of the game")
game = LCase(game)
Select Case game
Case "tennis"
Debug.Print "2 Players."
Case "cricket"
Debug.Print "11 Players."
Case "volleyball"
Debug.Print "5 Players."
Case "baseball"
Debug.Print "9 Players."
Case Else
Debug.Print "I have no idea."
End Select
End Sub
```

IIF Function

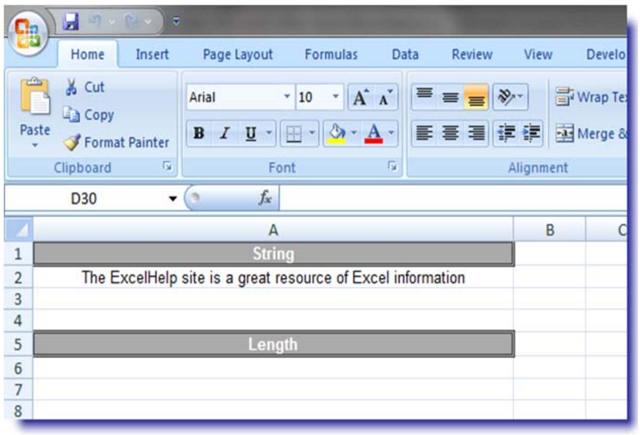
IIF function is used to evaluate an expression and perform one of two actions based on the outcome of the evaluation. For example:

IIF (Value > 10, Perform this action if Value is <= 10, Perform this action is Value is > 10)

This function is available within VBA code and also as an Excel function. Usually the IIF function is used to perform quick logical assessments and can be nested to perform more complicated evaluations. It is however important to remember that nested IF statements can become very complicated and difficult to support and maintain.

Now let's look at an example. Let's assume that we want to calculate the length of the string only if it contains the value Excel Help and Excel. (Fig 3)

Fig 3



D30

	A	B
1	String	
2	The ExcelHelp site is a great resource of Excel information	
3		
4		
5	Length	
6		
7		
8		

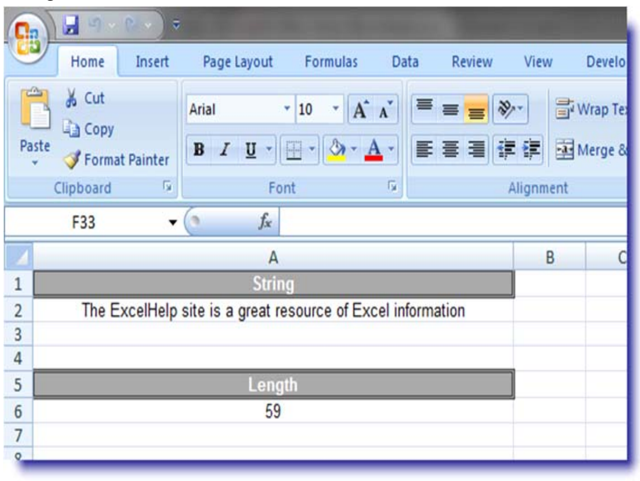
It is important to note that we could have used the IIF statement in one of our For Next loops to run through all the rows on a worksheet.

Code

```
Dim StringToProcess As String'Variable to hold the string to be processed

StringToProcess = ActiveSheet.Cells(2, 1).Value

ActiveSheet.Cells(6, 1).Value = IIf(InStr(StringToProcess, "ExcelHelp") > 0, IIf(InStr(StringToProcess, " Excel ") > 0, Len(StringToProcess), 0), 0)
```

Output (Fig 4)

Fig 4



F33

	A	B
1	String	
2	The ExcelHelp site is a great resource of Excel information	
3		
4		
5	Length	
6	59	
7		