VBA Data types, Variables and Constants

Objectives: At the end of this lesson you shall be able to
- **list the data types in VBA**
- **declare variables and assign values**
- **describe the option explicit statement.**

Introduction

Variables are entities that hold data. In VBA, variables are areas allocated by the computer memory to hold data. The following are the variable naming rules in VBA:

a Variable Names

The following are the rules when naming the variables in VBA

1 A Variable name must start with a letter and not a number. Numbers can be included within the name, but not as the first character.

2 A Variable name can be no longer than 250 characters.

3 A Variable name cannot be the same as any one of Excel's key words. For ex. you cannot name a Variable with such names as Sheet, Worksheet etc.

4 All Variables must consist of one continuous string of characters only. You can separate words by using the underscore.

b Declaring Variables

In VBA, the variables are declared before using them by assigning names and data types. Declaring variables before use tells the computer to allocate a certain memory for the variable data to be placed. Though it is a good practice to declare variables before use, in Visual Basic it is not actually compulsory to specifically declare a variable before it is used. If a variable isn't declared, VB will automatically declare the variable as a Variant. A variant is data type that can hold any type of data.

To declare a variable we use the word "Dim" (short for Dimension) followed by our chosen variable name then the word "As" followed by the variable type. For ex. Dim n as Integer.

You may also combine more variables to be declared in one line, separating each variable with a comma, as follows:

Dim first_name As String, joining_date As Date, Pay As Integer.

Declaring a variable before use is a good programming practice for the following reasons:

1 **Memory & Calculation Speed:** If you do not declare a variable to have a data type, it will, by default, have the Variant type. This takes up more memory than many of the other data types. Sometimes, Variant data types also take more time to process and at times may slow down the process.

2 **Prevention of typing errors:** If you always declare your variables, then you can use a VBA option to force you to declare variables. This will prevent you from introducing errors in your code by accidentally typing a variable name incorrectly.

3 **Highlighting wrong Data Values:** If you declare a variable to have a specific data type, and you attempt to assign the wrong type of data to it, this will generate an error in your program.

The Option Explicit Statement

The option 'Explicit' forces you to declare all variables that you use in your VBA code, by highlighting any undeclared variables as errors during compilation (before the code will run). To use this option, simply type the line as the very first line of the program (In the General Declarations section).

If you select the 'Require Variable Declaration' option of your VBA editor, the statement'Option Explicit' is always automatically included at the top of every new VBA module that is created.

To do this:

- In the Visual Basic Editor, select **Tools → Options**...
- Ensure the **Editor** tab is selected
- Check the box next to the option **Require Variable Declaration** and click OK

Keywords

Keywords in Excel VBA are words that Excel has set aside to use in the execution of code. This means we cannot use them for any other purpose. For example, Select, Active, Sub, End, Function etc are all Keywords that we can only use for their intended purpose.

Some of the reserved keywords are shown in Table 1.

Table 1

ByVal	Call	Case	CBool	CByte	CDate
CDbl	CInt	CLng	Const	CSng	CStr
Date	Dim	Do	Double	Each	Else
ElseIf	End	EndIf	Error	False	For
Function	Get	GoTo	If	Integer	Let
Lib	Long	Loop	Me	Mid	Mod
New	Next	Not	Nothing	Option	Or (Bitwise)
Or (Condition)	Private	Public	ReDim	REM	Resume
Select	Set	Single	Static	Step	String
Sub	Then	To	True	Until	vbCrLf
vbTab	With	While	Xor		

Data Types:

Visual Basic classifies data into two major categories, the numeric data types and the non-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 +, -, x, / and more. Examples of numeric data types are examination marks, height, weight, the number of students in a class, share values, price of goods, monthly bills, fees and others.

In VBA, numeric data are divided into 7 types, depending on the range of values they can store. Calculations that only involve round figures or data that does not need precision can use Integer or Long integer in the computation. Programs that require high precision calculation need to use Single and Double decision data types, they are also called floating point numbers. For currency calculation, you can use the currency data types. Lastly, if even more precision is required to perform calculations that involve many decimal points, we can use the decimal data types. These numeric data types summarized in Table 2.

Table 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).

Non Numeric Data Types

Non-numeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type.The non numeric data types are summarized in Table 3.

Table 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

Enumerated Data Types

If you have a set of unchanging values that are logically related to each other, you can define them together in an enumeration. This provides meaningful names for the enumeration and its members, which are easier to remember than their values. You can then use the enumeration members in many places in your code.

An enumeration has a name, an underlying data type, and a set of members. Each member represents a constant.

The Enum statement can declare the data type of an enumeration. Each member takes the enumeration's data type. You can specify Byte, Integer, Long etc..If you do not specify datatype for the enumeration, each member takes the data type of its initializer. If you specify both datatype and initializer, the data type of initializer must be convertible to data type. If neither datatype nor initializer is present, the data type defaults to Integer.

Ex.

```
Public Enum OS
Windows
Linux
Unix
DOS
MAC
End Enum
```

Suffixes for Literals

Literals are values that you assign to data. In some cases, we need to add a suffix to a literal so that VB can handle the calculation more accurately. For example, we can use pay=12000@ for a Currency type data. Some of the suffixes are displayed in Table 4.

Table 4

Suffix	Data Type
&	Long
!	Single
#	Double
@	Currency

Note: Enclose string literals within two quotations

Enclose date and time literals within two # symbols. Ex:

```
Dim marks%          'integer
Dim gorss_pay&      'long
Dim Average!        'single
Dim Total#          'double
Dim Profit@         'currency
Dim FirstName$      'string
```

If the data type is not specified, VB will automatically declare the variable as a Variant.

Named Constants

If you have a value that never changes in your application, you can define a named constant and use it in place of a literal value. A name is easier to remember than a value. You can define the constant just once and use it in many places in your code. If in a later version you need to redefine the value, the Const statement is the only place you need to make a change.

You can use Const only at module or procedure level. This means the declaration context for a variable must be a class, structure, module, procedure, or block, and cannot be a source file, namespace, or interface

Example:Const Pi As Single=3.142

Assigning Values to Variables

After declaring various variables using the Dim keywords or other keywords, we need to assign values or information to those variables. Assigning a value to a variable means storing the value in that variable. The form of an assignment statement is as follows:

Variable = Expression

The variable can be a declared variable or a control's property value. The expression could be a mathematical expression, a number, a literal value, a string, a Boolean value (true or false) , a combination of other variables and constants, a function and more. The following are some examples:

Basic = 10000

DA = Basic * 0.9

First Name = "Uma"

Label1.Caption = "Enter your age"

Command 1 Visible = false

Textbox.multiline = True

Label 1 Caption = textbox1.Text

A type mismatch error occurs when you try to assign a value to a variable of incompatible data type.

Operators in VBA and operator precedence

Objectives : At the end of this lesson you shall be able to
• **explain the various operators and their precedence in VBA.**

Operators in VBA

An **Operator** can be defined using a simple expression - 4 + 5 is equal to 9. Here, 4 and 5 are called **operands** and + is called **operator**. VBA supports following types of operators "

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Concatenation Operators

The Arithmetic Operators

Following arithmetic operators are supported by VBA.

Assume variable A holds 5 and variable B holds 10, the results of the various operators as shown in Table 1.

Table 1

Operator	Description	Example
+	Adds the two operands	A + B will give 15
-	Subtracts the second operand from the first	A - B will give -5
*	Multiplies both the operands	A * B will give 50
/	Divides the numerator by the denominator	B / A will give 2
%	Modulus operator and the remainder after an integer division	B % A will give 0
^	Exponentiation operator	B ^ A will give 100000

The Comparison Operators

There are following comparison operators supported by VBA.

Table 1 Assume variable A holds 10 and variable B holds 20, the results of various comparison operators as shown

Table 2

Operator	Description	Example
=	Checks if the value of the two operands are equal or not. If yes, then the condition is true	(A = B) is False
<>	Checks if the value of the two operands are equal or not. If the values are not equal, then the condition is true	(A <> B) is True
>	Checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition is true	(A > B) is False
<	Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition is true	(A < B) is True
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition is true	(A >= B) is False
<=	Checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition is true	(A <= B) is True

The Logical Operators

Following logical operators are supported by VBA.

Assume variable A holds 10 and variable B holds 0, the results on the various logical operators shown in Table 3

Table 3

Operator	Description	Example
AND	Called Logical AND operator. If both the conditions are True, then the Expression is true	a<>0 AND b<>0 is False
OR	Called Logical OR Operator. If any of the two conditions are True, then the condition is true	a<>0 OR b<>0 is true
NOT	Called Logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make false	NOT(a<>0 OR b<>0) is false
XOR	Called Logical Exclusion. It is the combination of NOT and OR Operator. If one, and only one, of the expressions evaluates to be True, the result is True.	(a<>0 XOR b<>0) is true

The Concatenation Operators

Following Concatenation operators are supported by VBA.

Assume variable A holds 5 and variable B holds 10, the result of various concatenation operators shown in Table 4

Table 4

Operator	Description	Example
+	Adds two Values as Variable. Values are Numeric	A + B will give 15
&	Concatenates two Values	A & B will give 510

Assume variable A = “Microsoft” and variable B = “VBScript”, the result of the various concatenation shown in Table 5

Table 5

Operator	Description	Example
+	Concatenates two Values	A + B will give MicrosoftVBScript
&	Concatenates two Values	A & B will give MicrosoftVBScript

Note: Concatenation Operators can be used for both numbers and strings. The output depends on the context, if the variables hold numeric value or string value.

Precedence

When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence.

When operators have the same precedence they are evaluated from left-to-right.Parentheses can be used to override the order and to evaluate certain parts of the expression. Operations inside parentheses are always performed before those outside.

When a series of operators appear in the same expression there is a strict order in which they will be evaluated.

The rules of precedence tell the compiler which operators to evaluate first.

Parentheses can obviously be used to change the order of precedence.

Operators are evaluated in the following order: Mathematical, Concatenation, Relational, Logical.

The table 6 shows the precedence order of operators.

Table 6

Order	Operator	Symbol
1	Exponentiation	^
2	Negation	-
3	Multiplication	*
3	Division	/
4	Division with Integer result	\
5	Modulo	Mod
6	Addition	+
6	Subtraction	-
7	String Concatenation	&
8	Equal or Assignment	=
8	Not Equal To	<>
8	Less Than	<
8	Greater Than	>
8	Less Than or Equal To	<=
8	Greater Than or Equal To	>=
9	Not	NOT
10	And	AND
11	Or	OR
12	Exclusive OR	XOR
13	Equivalence	EQV
14	Implication	IMP

The table 7 shows the expression, steps to evaluate and the result.

Expression	First Step	Second Step	Third Step	Result
3^(15/5)*2-5	3^3*2-5	27*2-5	54-5	49
3^((15/5)*2-5)	3^(3*2-5)	3^(6-5)	3^1	3
3^(15/(5*2-5))	3^(15/(10-5))	3^(15/5)	3^3	27

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

Customize the buttons in a VBA message box

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

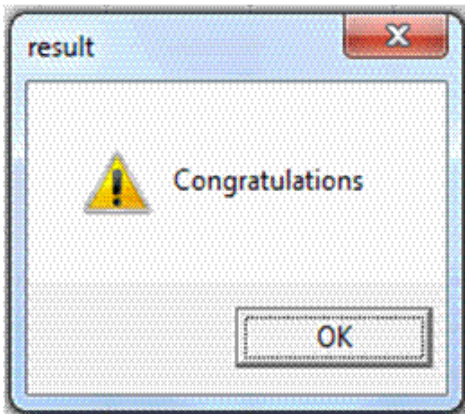
A list of various buttons and icons that can be used in the VBA message box is shown in the Table 1.

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



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.

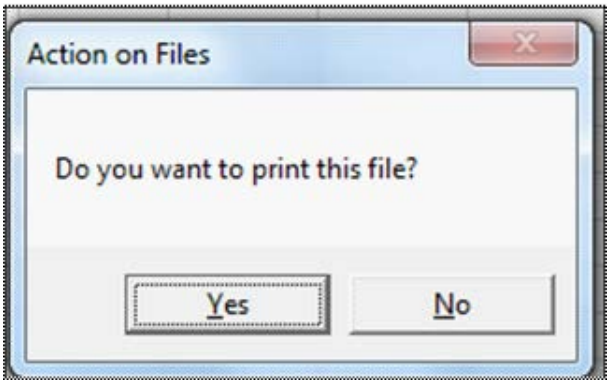
```
Sub test()

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

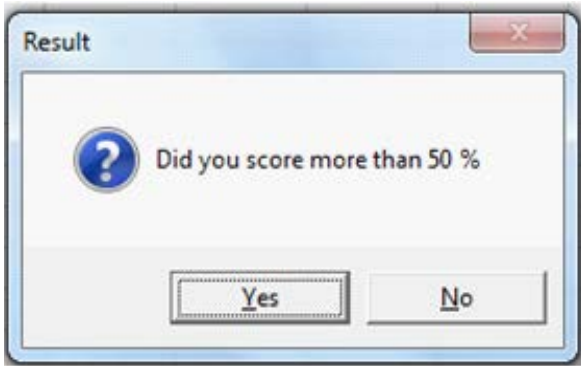


Reading the Msgbox() return values

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




```
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.

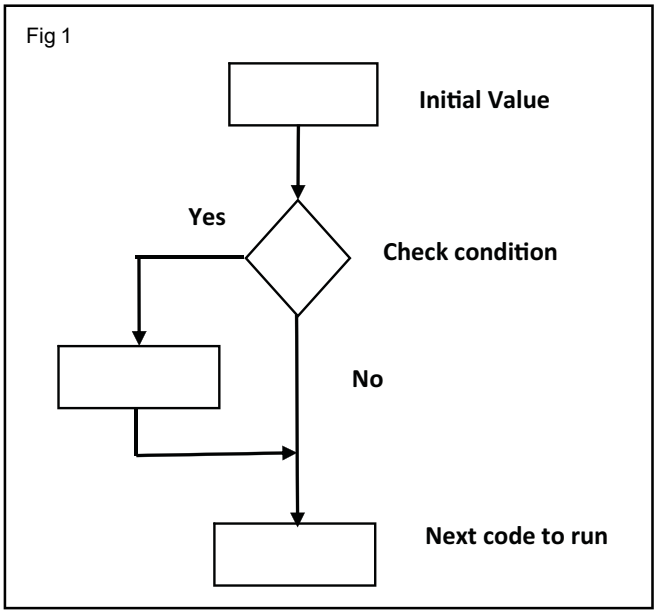
The If … Then Statement

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

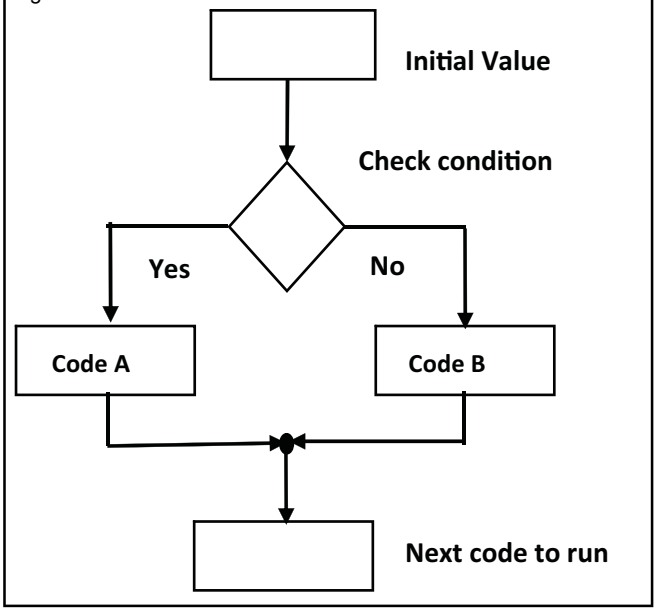
Else

' 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

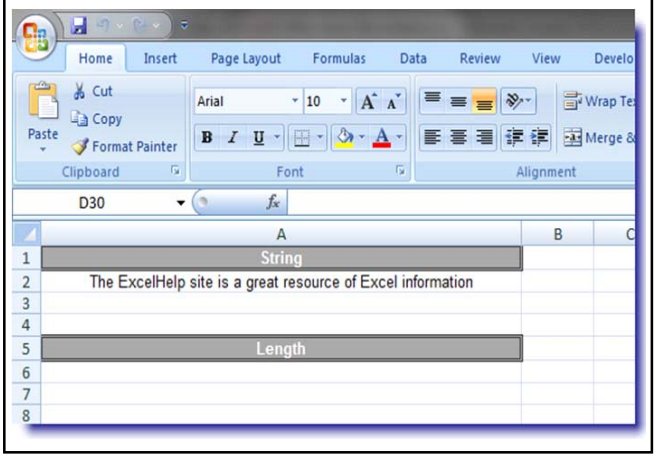
IIF function is used to evaluate an expression and perform one of two actions based on the outcome of the evaluation. For example:

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



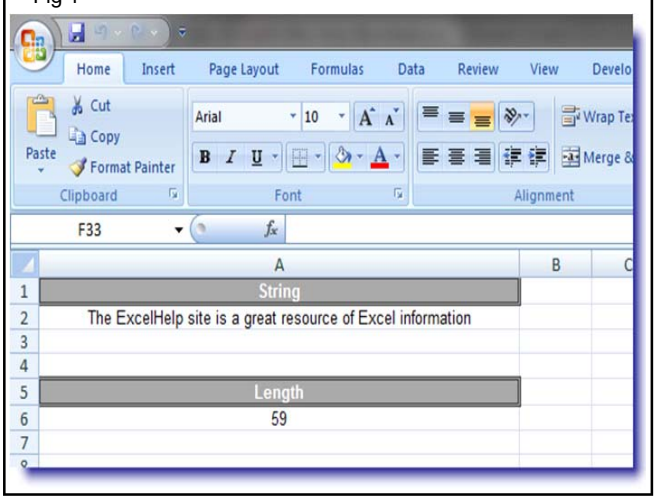
It is important to note that we could have used the IIF statement in one of our For Next loops to run through all the rows on a worksheet.

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



Looping statements in VBA

Objectives: At the end of this lesson you shall be able to
- **describe the "for" loops in VBA**
- **describe the "do" loops in VBA**
- **explain the use of the "exit" statement in VBA loops**
- **write appropriate code to perform repetitive tasks.**

Introduction

There may be many situations where you need to perform a task repeatedly / a certain number of times. In such cases the code for the task is placed inside a loop and the program iterates or repeats through the loop a certain number of times ie. till a certain condition is met. Some examples of such repetitive tasks are:

a Printing a text or number n number of times.

b Generating a sequence or series of numbers.

c Generating a table of certain calculations.

d Searching / Re arranging a set of numbers etc.

VBA provides the following types of loops to handle looping requirements (Refer Table 1)

Table 1

Loop Type	Description
for next loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do....until loop	Repeats a statement or group of statements until a condition is met.
do….while loop	Repeats a statement or group of statements as long as the condition is true.

The For Loop

The For ... next loop sets a variable to a specified set of values, and for each value, runs the VBA code inside the loop. For Ex.

```
For n = 1 To 10

debug.print  n

Next n
```

In this example, the initial value of n is set to 1, and the loop code, ie. printing the value of n is performed.The value of n is set to the next value which is by default an increment of 1. Thus this loop is executed 10 times and would print the numbers 1 to 10. The for statement in the above code is the same as For n = 1 To 10 Step 1 since the default increment is 1

The same code will print numbers from 10 to 1 if the step is changed to a negative value as shown below.

```
For n = 10 To 1 Step -1

debug. print  n

Next n
```

Similarly, the following Ex. would add all the numbers from 1 to 10 and print the sum.

```
Dim n, sum as integer

Sum=0

For n = 1 To 10

sum=sum + n

debug. print  sum

Next n
```

The For Each Loop

The For Each loop is similar to the For ... Next loop but, instead of looping through a set of values for a variable, it loops through every object within a set of objects. The following example would print the names of all the worksheets.

```
Dim ws As Worksheet

For each ws in Worksheets

debug. print ws.name

Next ws
```

The Exit For Statement

If you need to end the For loop before the end condition is reached or met, simply use the END FOR in conjunction with the IF statement. In the example given below, we exit the for loop prematurely and before the end condition is

met. The for example given below, the loop exits when n reaches a value of 5.

```
For n = 0 To 10

debug.print n

If n=5 Then Exit For

Next n
```

The Do ….Until Loop repeats a statement or group of statements until a condition is met.

There are 2 ways a Do Until loop can be used in Excel VBA Macro code.

a Test the condition before executing the code in the loop

b Execute the code in the loop and then test for the condition.

Do Until….. Loop

In this example, the value of n is tested before going into the loop.

If the condition n=10 is not met right at the beginning itself, the code inside the loop is not executed at all. The control then jumps to the statements appearing after the Loop statement.

```
Do Until n=10

Debug.print n

n=n+1

Loop
```

Do ….. Loop Until

In this example, the code in the loop is executed at least once before testing the condition. If the condition is true, the looping stops, else the loop is executed again.

```
Do

Debug. print n

n=n+1

Loop Until n=10
```

The Do While … Loop repeats a statement or group of statements as long as the condition is true.

Like the Do until loop, a Do While loop can be also be used in two ways.

a Test the condition before executing the code in the loop

b Execute the code in the loop and then test for the condition.

Do While ….Loop

In this example, the condition ie. num<10 is checked before entering the loop. Only if the condition is met, the code in the loop is executed, otherwise it is skipped entirely. This example will print a table as shown in Fig 1.

Fig 1

number	spc(3)
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

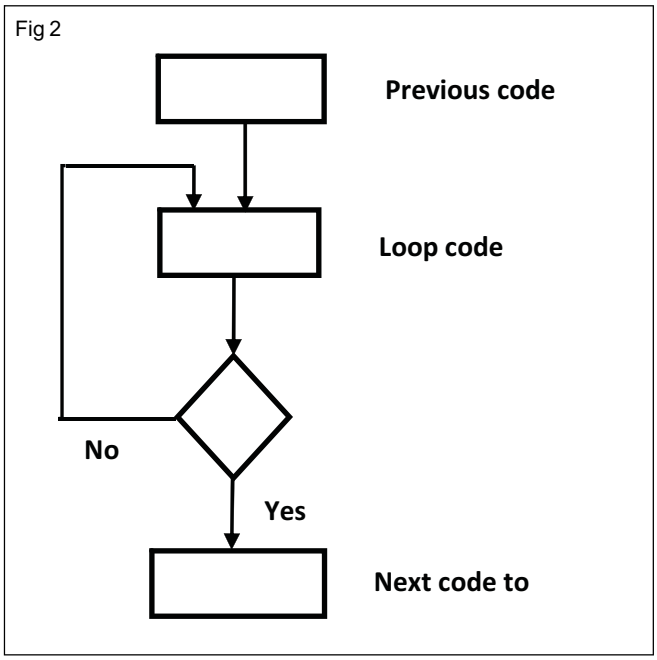
```
Dim num As Integer

Debug.Print "number"; Spc(2); "square"

Do While num < 10

num = num + 1

Debug.Print num; Spc(5); num * num

Loop
```

Do…. Loop While

In a Do…. Loop While , a set of statements in the loop are executed once, then the condition is checked. The code in the loop is executed only if the condition is met. (Refer Fig 2 for the flow chart)

In this example, the value 1 is placed in cell (1,1). The row value is incremented each time the loop code is executed. The incremented value is placed in the cell (row,1). The loop is executed as long as the row value is less than 10 after which the iterations stop. The condition checking is done after executing the loop code at least once. (Fig 2)

Fig 2




```
Dim row As Integer

row = 0

Do

row = row + 1

Cells(row, 1) = row

Loop While row < 10
```

The While ….. Wend loop

The While ….. Wend loop executes a series of statements as long as a given condition is True.

In this example the condition checking is done at the beginning of the loop. This code prints hello 5 times and then prints the value of the counter, ie. 5 at the end of the program.

```
Dim Counter

Counter = 0

While Counter < 5

Counter = Counter + 1

Debug.Print "hello"

Wend

Debug.Print Counter
```

The Exit Statement

The Exit Statement exits a procedure or block and transfers control immediately to the statement following the procedure call or the block definition. It may be in the form of Exit Do, Exit For, Exit While, Exit Select etc. depending on where it is being used. An example of an Exit statement is as follows:

```
Do While True

Count = Count + 1

Debug.Print Count

If Count = 5 Then

Debug.Print "stop at 5"

Exit Do

End If

Loop
```

In this example, the loop condition stops the loop when count=5.