VISUAL BASIC

VISUAL BASIC is a high level programming language evolved from the earlier DOS version called BASIC. BASIC stands for **Beginners' All-purpose Symbolic Instruction Code**. The program codes in Visual Basic resemble the English language. Visual Basic is a fairly easy programming language to learn. VISUAL BASIC is a VISUAL and events driven Programming Language.

V.B is very powerful programming system that helps one develop sophisticated, graphical applications that can be run on Microsoft Windows environment. V.B is actually BASIC (Beginner's Allpurpose Symbolic Instruction Code) language, which is visual in its nature. i.e. it is used to create GUI applications i.e. visual Programming style involves a lot of illustrations (graphics entities), rather than writing numerous lines of code to describe the appearance, functioning etc, of the application's interface

Visual Basic is **event-driven** because users may click on a certain object randomly, so each object has to be programmed independently to be able to response to those actions (events). Examples of events are clicking a command button, entering text into a text box, selecting an item in a list box etc. Therefore, a VISUAL BASIC Program is made up of many subprograms; each with its own program code which can be executed independently and at the same time can be linked together in one way or another.

What is event driven programming?

The event-driven programming revolves around recognizing the occurrences of events and then responding to those events by taking appropriate actions. In event-driven programming an application is build up as a series of responses to user-events.

Why VB is called 'Event-Driven' programming language?

In traditional or procedural application, the application itself determines which portion of code is to be executed and in what sequence. Generally execution starts with the 1st line of code and follow the coding sequence define in the application. Where as application written in VB are 'Event-Driven'. In an event-driven application the code doesn't follow a pre determined path rather it execute different code sections in response to events. Event can be triggered by user's action, by message from system, other applications or even from the application itself. The sequences of these events determine the order in which the code execute and associated with the objects of application. They either act on an object or are triggered by an object to control the flow of execution when it is running. That is why VB called Event-Driven programming language.

Common Events of Visual Basic Controls

Events are what happen in and around your program. For example, when a user clicks a button, many events occur: The mouse button is pressed, the CommandButton in your program is clicked, and then the mouse button is released. These three things correspond to the MouseDown event, the Click event, and the MouseUp event. During this process, the GotFocus

event for the CommandButton and the LostFocus event for whichever object previously held the focus also occur.

Again, not all controls have the same events, but some events are shared by many controls. These events occur as a result of some specific user action, such as moving the mouse, pressing a key on the keyboard, or clicking a text box. These types of events are *user-initiated events* and are what you will write code for most often. Events common to most VB controls are described in the table below.

Event	*Occurs When ...*
Change	The user modifies text in a combo box or text box.
Click	The user clicks the primary mouse button on an object.
DblClick	The user double-clicks the primary mouse button on an object.
DragDrop	The user drags an object to another location.
DragOver	The user drags an object over another control.
GotFocus	An object receives focus.
KeyDown	The user presses a keyboard key while an object has focus.
KeyPress	The user presses and releases a keyboard key while an object has focus.
KeyUp	The user releases a keyboard key while an object has focus.
LostFocus	An object loses focus.
MouseDown	The user presses any mouse button while the mouse pointer is over an object.
MouseMove	The user moves the mouse pointer over an object.
MouseUp	The user releases any mouse button while the mouse pointer is over an object.

Managing Visual Basic Data

There are many types of data we come across in our daily life. For example, we need to handle data such as names, adresses, money, date, stock quotes, statistics and etc everyday. Similarly In Visual Basic, we are also going to deal with these kinds of data. However, to be more systematic, VB divides data into different types.

Types of Visual Basic Data

Numeric Data

Numeric data are data that consists of numbers, which can be computed mathematically with various standard operators such as add, minus, multiply, divide and so on. In Visual Basic, the numeric data are divided into 7 types, they are summarised as

Numeric Data Types

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

Non-numeric Data Types

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

Suffixes for Literals

Literals are values that you assign to a data. In some cases, we need to add a suffix behind a literal so that VB can handle the calculation more accurately. For example, we can use num=1.3089# for a Double type data.

Suffix	Data Type
&	Long
!	Single
#	Double
@	Currency

In additon, we need to enclose string literals within two quotations and date and time literals within two # sign. Strings can contain any characters, including numbers. The following are few examples:

memberName="Turban, John."

TelNumber="1800-900-888-777"

LastDay=#31-Dec-00#

ExpTime=#12:00 am#

Managing 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 term of VB, 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 Visual Basic, you have to follow a set of rules.

Variable Names

The following are the rules when naming the variables in Visual Basic

- 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

Table 5.4

Valid Name	Invalid Name
My_Car	My.Car
ThisYear	1NewBoy
Long_Name_Can_ beUSE	He&HisFather *& is not acceptable

Declaring Variables

In Visual Basic, one needs to declare the variables before using them by assigning names and data types. They are normally declared in the genaral section of the codes' windows using the Dim statement.

The format is as follows:

Dim variableNmae as DataType

Dim password As String

Dim yourName As String

Dim firstnum As Integer

Dim secondnum As Integer

Dim total As Integer

Dim doDate 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 data type is not specified, VB will automatically declares the variable as a Variant.

For string declaration, there are two possible format, 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 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.

Working with Variables

Assigning Values to Variables

After declaring various variables using the Dim statements, we can assign values to those variables. The general format of an assignment is

Variable=Expression

The variable can be a declared variable or a control property value. The expression could be a

mathematical expression, a number, a string, a boolean value(true or false) and etc. The following are some examples:

firstNumber=100

secondNumber=firstNumber-99

userName="John Lyan"

userpass.Text = password

Label1.Visible = True

Command1.Visible = false

Label4.Caption = textbox1.Text

ThirdNumber = Val(usernum1.Text)

total = firstNumber + secondNumber+ThirdNumber

Operators in Visual Basic

In order to compute inputs from users and to generate results, we need to use various mathematical operators. In Visual Basic, except for + and -, the symbols for the operators are different from normal mathematical operators as shown

Operator	Mathematical function	Example
^	Exponential	2^4=16
*	Multiplication	4*3=12
/	Division	12/4=3
Mod	Modulus(return the remainder from an integer division)	15 Mod 4=3
\	Integer Division(discards the decimal places)	19\4=4
+ or &	String concatenation	"Visual"&"Basic"="Visual Basic"

Example:

firstName=Text1.Text

secondName=Text2.Text

yourName=firstName+secondName

number1=val(Text3.Text)

number2=val(Text4.Text)

number3=num1*(num2^3)

number4=number3 Mod 2

number5=number4\number1

Total=number1+number2+number3+number4+number5

Average=Total/5

Conditional Operators

To control the VB program flow, we can use various conditional operators. Basically, they resemble mathematical operators. Conditional operators are very powerful tools, they let the VB program compare data values and then decide what action to take, whether to execute a program or terminate the program and etc.

Conditional Operators	Operator Meaning
=	Equal to
>	More than
<	Less Than
>=	More than and equal
<=	Less than and equal
<>	Not Equal to

* You can also compare strings with the above operators. However, there are certain rules to follows:

Upper case letters are less than lowercase letters, "A"<"B"<"C"<"D".....<"Z" and number are less than letters.

Logical Operators

In addition to conditional operators, there are a few logical operators which offer added power to the VB programs. There are shown in Table

Operator	Meaning
And	Both sides must be true

or	One side or other must be true
Xor	One side or other must be true but not both
Not Negates	truth

Error Types (Visual Basic)

In Visual Basic, errors fall into one of three categories: syntax errors, run-time errors, and logic errors.

Syntax Errors

Syntax errors are those that appear while you write code. If you make a mistake, such as misspelling a word or using a language element improperly. If you compile from the command line, Visual Basic displays a compiler error with information about the syntax error. Syntax errors are the most common type of errors. You can fix them easily in the coding environment as soon as they occur.

Note

The Option Explicit statement is one means of avoiding syntax errors. It forces you to declare, in advance, all the variables to be used in the application. Therefore, when those variables are used in the code, any typographic errors are caught immediately and can be fixed.

Run-Time Errors

Run-time errors are those that appear only after you compile and run your code. These involve code that may appear to be correct in that it has no syntax errors, but that will not execute. For example, you might correctly write a line of code to open a file. But if the file does not exist, the application cannot open the file, and it throws an exception. You can fix most run-time errors by rewriting the faulty code or by using exception handling, and then recompiling and rerunning it.

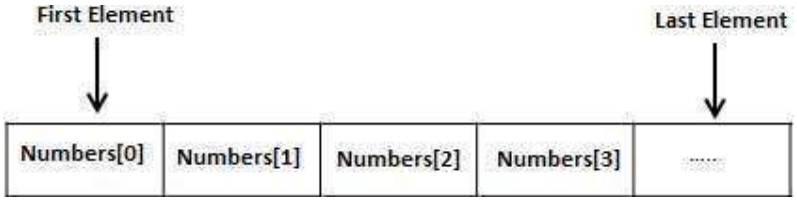
Logic Errors

Logic errors are those that appear once the application is in use. They are most often faulty assumptions made by the developer, or unwanted or unexpected results in response to user actions. For example, a mistyped key might provide incorrect information to a method, or you may assume that a valid value is always supplied to a method when that is not the case. Although logic errors can be handled by using exception handling most commonly they should be addressed by correcting the error in logic and recompiling the application.

ARRAYS IN VISUAL BASICS

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Creating Arrays in VB.Net

To declare an array in VB.Net, you use the Dim statement. For example,

```
Dim intData(30)                    ' an array of 31 elements
Dim strData(20) As String          ' an array of 21 strings
Dim twoDarray(10, 20) As Integer   'a two dimensional array of integers
Dim ranges(10, 100)          'a two dimensional array
```

You can also initialize the array elements while declaring the array. For example,

```
Dim intData() As Integer = {12, 16, 20, 24, 28, 32}
Dim names() As String = {"Karthik", "Sandhya", _
"Shivangi", "Ashwitha", "Somnath"}
Dim miscData() As Object = {"Hello World", 12d, 16ui, "A"c}
```

Dynamic Arrays

Dynamic arrays are arrays that can be dimensioned and re-dimensioned as par the need of the program. You can declare a dynamic array using the **ReDim** statement.

Syntax for ReDim statement −

```
ReDim [Preserve] arrayname(subscripts)
```

Where,

- The **Preserve** keyword helps to preserve the data in an existing array, when you resize it.
- **arrayname** is the name of the array to re-dimension.
- **subscripts** specifies the new dimension.

```
Module arrayApl
  Sub Main()
    Dim marks() As Integer
    ReDim marks(2)
    marks(0) = 85
    marks(1) = 75
    marks(2) = 90
```

```
    ReDim Preserve marks(10)
    marks(3) = 80
    marks(4) = 76
    marks(5) = 92
    marks(6) = 99
    marks(7) = 79
    marks(8) = 75

    For i = 0 To 10
      Console.WriteLine(i & vbTab & marks(i))
    Next i
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result −

```
0       85
1       75
2       90
3       80
4       76
5       92
6       99
7       79
8       75
9       0
10      0
```

Multi-Dimensional Arrays

VB.Net allows multidimensional arrays. Multidimensional arrays are also called rectangular arrays. You can declare a 2-dimensional array of strings as −

```
Dim twoDStringArray(10, 20) As String
```

or, a 3-dimensional array of Integer variables −

```
Dim threeDIntArray(10, 10, 10) As Integer
```

The following program demonstrates creating and using a 2-dimensional array −

```
Module arrayApl
  Sub Main()
    ' an array with 5 rows and 2 columns
    Dim a(,) As Integer = {{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4, 8}}
    Dim i, j As Integer
    ' output each array element's value '

    For i = 0 To 4
      For j = 0 To 1
        Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i, j))
      Next j
```

```
    Next i
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result −

```
a[0,0]: 0
a[0,1]: 0
a[1,0]: 1
a[1,1]: 2
a[2,0]: 2
a[2,1]: 4
a[3,0]: 3
a[3,1]: 6
a[4,0]: 4
a[4,1]: 8
```

What is Dynamic Array? In what situations is it used? Explain with an example.

Sometimes we may not know how large to make an array. Instead of making it large enough to hold the maximum of data (which means the most of the array may be empty) we can declare a dynamic array. The size of a dynamic array can vary during the course of the program, or we might need an array until the user has entered a bunch of data and the application has processed it and displayed the results. With a dynamic array, we can discard the data and return the resources if occupied the system.

Difference between Static Array and Dynamic array

There are two types of array in VBA, Static array and Dynamic array.

Static array:

- Array is dimenstioned during design time.
- Array size can not be changed.
- By using erase function, only array element values in the memory will be emptied, elements memory will not be deleted.
- Memory is fixed during design time.

Dynamic array:

- Array is dimenstioned during runtime
- Array size can be changed (Shrink or Enlarge) any number of times.
- By using erase function, both array element values and elements memory will be deleted.
- We can utilise the memory efficiently.