



*Vishwendu Vidya Prasarak Mandal's*  
 (Regd. No: MAH 1906 / F-1614; Dt.5/3/1987)  
*Abhinav Vidyalay & Junior College*  
 (Govt. Regd. No. Prim.Edu. 6-970,90-91 Dtd:16-8-90; Index No: Sec:16.17.019 /  
 020; H.Sec: J16.17.24)  
 Contact: +91(0)251 2472232 [admin@abhinav.ac.in](mailto:admin@abhinav.ac.in); [www.abhinav.ac.in](http://www.abhinav.ac.in)

### **XI\_IT\_Visual-Basic.**

#### **Introduction to Visual Basic**

Microsoft Visual Basic is the fastest and easiest way to create applications for Microsoft Windows. The "Visual" part refers to the method used to create the graphical user interface (GUI).

The "Basic" part refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language, a language used by more programmers than any other language in the history of computing. Visual Basic has evolved from the original BASIC language and now contains several hundred statements, functions, and keywords, many of which relate directly to the Windows GUI. Beginners can create useful applications by learning just a few of the keywords, yet the power of the language allows professionals to accomplish anything that can be accomplished using any other Windows programming language.

#### **Elements of the Integrated Development Environment**

The working environment in Visual Basic is often referred to as the integrated development environment or IDE because it combines together (integrates) many different functions such as design, editing, compiling, and debugging within a common environment. More over it provides the tools through which you can connect to other applications, put together i.e. integrates and develops in the same VB environment.

For e.g. OLE is a control through which you can connect to the applications like Ms-Word, Ms Excel, Access, or any type of application or any of the existing files on your disk..

The Visual Basic integrated development environment (IDE) consists of the following elements.

Menu Bar : Displays the commands you use to work with Visual Basic.

Context Menus : Contain shortcuts to frequently performed actions.

Toolbars : Provide quick access to commonly used commands in the programming environment.

Toolbox : Provides a set of tools that you use at design time to place controls on a form.

List of standard toolbox controls :Standard Toolbox Controls

Pointer	The only item in the toolbox that doesn't draw a control. When you select the pointer, you can only resize or move a control that has already been drawn on a form.	
Label	Allow you to have text that you don't want the user to change, such as a caption under a graphic.	
Text Box	Holds text that the user can either enter or change.	
Frame	Allow you to create graphical or functional grouping for controls. To group controls, draw the frame first, and then draw controls inside the frame.	
Command Button	Creates a button so that the user can choose to carry out a command.	
Check Box	Creates a box that the user can easily choose to indicate if something is true or false, or to display multiple choices when the user can choose more than one.	
Option Button	Allow you to display multiple choices from which the user can choose only one.	
Combo Box	Allow you to draw a combination list box and text box. The user can either choose an item from the list of enter a value in the text box. Combo Box are of three types:	
	VB Combo Dropdown	It includes a dropdown list & a text box. The user can select from the list or type in the text box
	VB Combo Simple	Simple Combo box includes a textbox & a list that does not dropdown the user can just type in the text box.
	VB Combo Dropdown List	This style allows selection only from the dropdown list but we can't type in the text box.
List Box	Used to display a list of items from which the user can choose one. The list can be scrolled if it has more items that can be displayed at one time.	
HScroll Bar (Horizontal Scroll Bar)	Provides a graphical tool for quickly navigating through a long list of items or a large amount of information, for indicating the current position on a scale, or as an input device or indicator of speed or quantity.	
VScroll Bar (Vertical Scroll Bar)	Provides a graphical tool for quickly navigating through a long list of items or a large amount of information, for indicating the current position on a scale or as an input device or as an input device or indicator of speed or quantity.	
Shape	Allow you to draw a variety of shapes on your form at design time. You can choose a rectangle, rounded rectangle, square, rounded rectangle square, rounded square, oval, or circle.	
Line	Used to draw a variety of line styles on your form at design time	
Image	Displays a graphical image from a bitmap, icon, or metafile on your form. Images displayed in an image control can only be decorative & use fewer resources than a Picture Box.	
Picture Box	Displays graphical images (either decorative or active), as a container that receives output from graphics methods, or as a container that receives output from graphics methods, or as a container for other controls.	
Slider Control	Slider Control let the user specify a magnitude by scrolling a selector between its minimum and	

### The Structure of a Visual Basic Application

An application is really nothing more than a set of instructions directing the computer to perform a task or tasks. Because a Visual Basic application is based on objects (known as controls), the structure of its code closely models its physical representation on screen.

**Saving a Visual Basic Program :** A Visual basic Program always contains one or more form files, but how can you manage all of them? Visual Basic's solution is to store all the names of your form files in (yet) another file called a Project file. In other words, in project files all the form files are saved which are related to that project. A project file has .vbp extension and a form file has .frm extension.

**Variables:** A variable is a name or identifier, which is associated with some value or data. A variable is actually a location in the computers memory where data is stored. Variables are used for storing values temporarily in the memory.

e.g. `x=20` here x is a variable which has the value 20 or we can say that the value 20 is identified by the identifier or variable x.

In Visual Basic, you use variables to temporarily store values during the execution of an application. All variables have a name (the word you use to refer to the value) and a data type (which determines the kind of data the variable can store).

**Rules for naming the variable are as follow: -**

A variable name:

1. Must begin with a letter.
2. It should not exceed than 40 characters.
3. It should not contain any special characters like @, !, %, # etc.

**Declaring Variables:** To declare a variable is to tell the program about it in advance.

In most programming languages, variables must be declared in advance for the compiler. Historically, the reason for doing this has been to help the compiler. Every time a compiled application runs into a new variable, it has to create it. Doing so doesn't take a lot of statements, but it does produce a delay that could be avoided. If the compiler knows all the variables and their types that are going to be used in the application ahead of time, it can produce the most compact and efficient, or optimized, code.

For example, if the compiler knows the types of the variables, it will catch many typing errors at design or compile time (error that otherwise would surface at run time.)

**Implicit Declaration :** You can also choose not to declare variables. When Visual Basic meets an undeclared variable name, it creates a new variable on the spot and uses it. The new variable's type is variant, the generic data type that can accommodate all other data types. Using a new variable in your code is equivalent to declaring it with out type. Visual Basic adjusts its type according to the value you assign it.

If `x=2`, it will assume automatically that variable x will store integer values. And if `x="abc"` it will assume automatically that variable x will stored string values.

Visual Basic enforces variable declaration by the option explicit statement

### Explicit Declaration

A variable along with the data type is declared in advanced. In this type of declaration Dim statement is used A variable is provided with data type in advance.

Syntax:

`Dim <variable name> as <data type>`

e.g `Dim x as integer`

i.e. x will store only integer values only e.g. `x=2`, `x=100` etc

e.g `Dim x as string`

i.e. x will store only String values only e.g. `x="abc"`.

Explicit Declaration is more convenient than Implicit Declaration

Explicit Declaration is useful at the time of addition of numbers in textboxes

### Data types in Visual Basic

Variables are placeholders used to store values; they have names and data types. The data type of a variable determines how the bits representing those values are stored in the computer's memory. When you declare a variable, you can also supply a data type for it. All variables have a data type that determines what kind of data they can store.

By default, if you don't supply a data type, the variable is given the Variant data type. The Variant data type is like a chameleon — it can represent many different data types in different situations. You don't have to convert between these types of data when assigning them to a Variant variable: Visual Basic automatically performs any necessary conversion.

If you know that a variable will always store data of a particular type, however, Visual Basic can handle that data more efficiently if you declare a variable of that type Visual Basic recognizes following six types of data : -

1. Numeric
2. String
3. Boolean
4. Object
5. Date
6. Variant

1. Numeric Data Types: Numeric data types contains numbers, depending on the size, numeric data types can be a byte, integer, long, single, double, or Currency. Each of these has its own range of storage. Using a numeric data type generally uses less storage space than a variant.

Visual Basic supports following numeric data types

Data type	Storage size	Range
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767

Long (long integer)	4 bytes	-2,147,483,648 to 2,147,483,647
Single (single-precision)	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double (double-precision)	8 bytes	-1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807

2. String Data types : String data types stores alphanumeric values. (Alphanumeric: It contains alphabets and numbers both)

3. Boolean Data types :The Boolean data type stores True / False values. Although a single bit would be adequate, for efficiency reasons Visual Basic allocates two bytes to this data type. Boolean variables are, in essence, integers that take the value -1 (for True) and 0 (for False). Actually, any non-zero value is considered True.

Boolean variables are declared as:

Dim failure As Boolean

And they are initialized to False.

Boolean variables are used in testing conditions, such as the following:

If failure Then MsgBox "Couldn't complete the operation"

They are also combined with the logical operators AND, OR, NOT, and XOR. The NOT operator toggles the value of a Boolean variable. The following statement is a toggle:

Running = Not running

If the variable *running* is True, it's reset to False, and vice versa.

4. Date Data type :It stores double precision numbers; the integer part represents the date and the fractional part represents the time. A variable declared as Date can store both date and time values with a statement like the following:

Dim expiration As Date

The following are all valid assignments:

Expiration = "01/01/1997"

Expiration = "13:03:05 AM"

Expiration = "02/23/1995 13:03:05 AM"

5. Object Data type :An object data type refers to one of visual basics many objects. You can use a object variable to access an actual object. Object variables are stored as 32-bit (4-byte) addresses that refer to objects within an application or within some other application. A variable declared as Object is one that can subsequently be assigned (using the Set statement) to refer to any actual object recognized by the application.

Dim objDb As Object

Set objDb = OpenDatabase("c:\Vb5\Biblio.mdb")

Visual Basic can resolve references to the properties and methods of objects with specific types before you run an application. This allows the application to perform faster at run time.

6. Variant Data type :This is the most flexible data type because it can accommodate all other types. A variable declared as variant (or a variable that hasn't been declared at all ) is handled by Visual Basic according to the variable's current contents. If you assign an integer value to a variant, Visual Basic treats it as an integer. If you assign a string to a variant, Visual Basic treats it as a string. Variants can also hold different data types in the course of the same program. Visual Basic performs the necessary conversions for you.

To declare a variant, use the Dim statement without specifying a type, as follows:

Dim myVar

You can also specify the Variant type to make the code cleaner:

Dim myVar As Variant

User Defined Data Types

Creating Your Own Data Types

You can combine variables of several different types to create user-defined types (known as *structs* in the C programming language). User-defined types are useful when you want to create a single variable that records several related pieces of information.

You create a user-defined type with the Type statement, which must be placed in the Declarations section of a module. User-defined types can be declared as Private or Public with the appropriate keyword. For example:

Private Type MyDataType

-or-

Public Type MyDataType

For example, you could create a user-defined type that records information about a computer system:

' Declarations (of a standard module).

Private Type SystemInfo

CPU As Variant

Memory As Long

VideoColors As Integer

Cost As Currency

PurchaseDate As Variant

End Type

Note If declared using the Dim keyword, user-defined types will default to Public. If you intend a user-defined type to be private, make sure you declare it using the Private keyword.

## Assigning and Retrieving Values

Assigning and retrieving values from the elements of this variable is similar to setting and getting properties:

```
MySystem.CPU = "486"
```

```
If MySystem.PurchaseDate > #1/1/92# Then
```

You can also assign one variable to another if they are both of the same user-defined type. This assigns all the elements of one variable to the same elements in the other variable.

```
YourSystem = MySystem
```

**Operators**—Description: An operator is a symbol (+, =, <, &) or word (OR, NOT) that indicates an operation to be performed on one or more elements (number, expression, constants, strings, objects etc). Visual Basic has four kinds of operators:

Types of Operators	Purpose of Operators	Examples of Operators
Arithmetic Operators:	Operators used to perform mathematical calculations. E.g. +, -, *, /, ^.	X=4 Y=X+2 Ans: Y= 6 (In this case “+” is the Arithmetic Operator)
Comparison Operators/ Relational Operators	Operators used to perform comparisons. E.g. >, <, <=, >=, <>, +.	X=4 Y= X > 2 Ans: - True (In this case “>” is the Relational Operator)
Concatenation Operators	Operators used to combine strings. E.g. +, &.	Hello&Friends Ans: - HelloFriends (In this case “&” is the Concatenation Operator.)
Logical Operators	Operators used to perform logical operations. E.g. NOT, OR & AND.	X=4,Y=2 1) X > Y AND Y > X 2) X > Y OR Y > X 1) Ans: - False (In this case “AND “ is the Logical Operator.) 2) Ans: - True (In this case “OR” is the Logical Operator)

Boolean table for Logical\_Operators.

AND as the Logical Operator

1st Expression/ Variable	2 <sup>nd</sup> Expression/ Variable	Output
True	True	True
True	False	False
False	True	False
False	False	False

OR as the Logical Operator

1st Expression/ Variable	2nd Expression/ Variable	Output
True	True	True
True	False	True
False	True	True
False	False	False

**Control Structures** :Control structures allow you to control the flow of your program's execution. If left unchecked by control-flow statements, a program's logic will flow through statements from left to right, and top to bottom. While some very simple programs can be written with only this unidirectional flow, and while some flow can be controlled by using operators to regulate precedence of operations, most of the power and utility of any programming language comes from its ability to change statement order with structures and loops.

### Decision Structures & Loop Structures

Decision Structure: - Visual Basic procedures can test conditions and then, depending on the results of that test, perform different operations. The decision structures that Visual Basic supports include:

If...Then

If...Then...Else

Select Case

If...Then Use an If...Then structure to execute one or more statements conditionally. You can use either single-line syntax or multiple-line block syntax:

```
If condition Then statement
```

```
If condition Then  
statements
```

```
End If
```

The condition is usually a comparison, but it can be any expression that evaluates to a numeric value. Visual Basic interprets this value as True or False; a zero numeric value is False, and any nonzero numeric value is considered True. If condition is True, Visual Basic executes all the statements following the Then keyword. You can use either single-line or multiple-line syntax to execute just one statement conditionally (these two examples are equivalent):

```
If marks < 35 then Student=Fail
```

```
If marks<35then
```

```
Student=Fail
```

```
End if
```

Notice that the single-line form of If...Then does not use an End If statement. If you want to execute more than one line of code when condition is True, you must use the multiple-line block If...Then...End If syntax.

```
If marks < 35 Then
```

```
Student = Fail
```

```
Remark = improve
```

```
End If
```

```
If...Then...Else
```

Use an If...Then...Else block to define several blocks of statements, one of which will execute:

```
If condition1 Then [statementblock-1]
```

```
[Else
```

```
[statementblock-n]]
```

```
End If
```

Visual Basic first tests condition1. If it's False, Visual Basic proceeds to test the second condition when it finds a True condition; Visual Basic executes the corresponding statementblock and then executes the code following the End If.

```
Private Sub Command_click ()
```

```
If text1.text < 35 then
```

```
Msgbox "Fail"
```

```
Else
```

```
Msgbox "Pass"
```

```
End if
```

```
End Sub
```

Select Case :Visual Basic provides the Select Case structure as an alternative to If...Then...Else for selectively executing one block of statements from among multiple blocks of statements. A Select Case statement provides capability similar to the If...Then...Else statement, but it makes code more readable when there are several choices. A Select Case structure works with a single test expression that is evaluated once, at the top of the structure. Visual Basic then compares the result of this expression with the values for each Case in the structure. If there is a match, it executes the block of statements associated with that Case:

```
Select Case test expression
```

```
[Case expressionlist1
```

```
[statementblock-1]]
```

```
[Case expressionlist2
```

```
[statementblock-2]]
```

```
[Case else
```

```
[statement block-n]]
```

```
End Select
```

Each expression list is a list of one or more values. If there is more than one value in a single list, the values are separated by commas.

Each statementblock contains zero or more statements. If more than one Case matches the test expression, only the statement block associated with the first matching Case will execute. Visual Basic executes statements in the Case Else clause (which is optional) if none of the values in the expression lists matches the test expression.

For example, suppose you added another command to the Edit menu in the If...Then...Else example. You could add another ElseIf clause, or you could write the function with Select Case:

```
Private Sub Command_click ()
```

```
Select case text1.text
```

```
Case "Monday"
```

```
Msgbox "first day of the week"
```

```
Case "Tuesday"
```

```
Msgbox "second day of the week"
```

```
Case "Sunday"
```

```
Msgbox "it's a fun day"
```

```
End select
```

```
End Sub
```

Notice that the Select Case structure evaluates an expression once at the top of the structure. In contrast, the If...Then...Else structure can evaluate a different expression for each ElseIf statement. You can replace an If...Then...Else structure with a Select Case structure only if the If statement and each ElseIf statement evaluates the same expression.

Loop structures:- It allow you to execute one or more lines of code repetitively. The loop structures that Visual Basic supports includes:

```
For...Next
```

When you know you must execute the statements a specific number of times, however, a For...Next loop is a better choice. Unlike a Do loop, a For loop uses a variable called a counter that increases or decreases in value during each repetition of the loop. The syntax is:

```
For counter = start To end [Step increment]
```

```
statements
```

```
Next [counter]
```

The arguments counter, start, end, and increment are all numeric.

Note The increment argument can be either positive or negative. If increment is positive, start must be less than or equal to end or the statements in the loop will not execute. If increment is negative, start must be greater than or equal to end for the body of the loop to execute. If Step isn't set, then increment defaults to 1.

In executing the For loop, Visual Basic:

Sets counter equal to start.

Tests to see if counter is greater than end. If so, Visual Basic exits the loop. (If increment is negative, Visual Basic tests to see if counter is less than end.)

Executes the statements.

Increments counter by 1 or by increment, if it's specified.

Repeats steps 2 to 4.

```
Private Sub Command_Click ()
```

```
Dim i As Integer
```

```
For i = 1 To 5
```

```
Print i
```

```
Next i
```

End Sub

Working with Control Structure: - Nested Control Structures. You can place control structures inside other control structures (such as an If...Then block within a For...Next loop). A control structure placed inside another control structure is said to be nested.

Control structures in Visual Basic can be nested to as many levels as you want. It's common practice to make nested decision structures and loop structures more readable by indenting the body of the decision structure or loop.

### Unconditional Branching

goto name;

...

name: statement

The goto keyword transfers control directly to the statement specified by the label name.

### Event

An action, recognized by an object, for which you can write code to respond, is an event. Events can be generated by a user action—such as clicking the mouse or pressing a key—by program code, or by the system, as with timers.

The most common types of events can be classified into three categories:

**Keyboard Events** : Occur when the user presses a certain keys, such as tab, or a certain keystroke combination, such as ctrl + P

**Mouse Events**: Occur when the user moves the mouse, clicks or double-click the mouse button, or drags the mouse across the screen.

**Program Events**: Occur when a Visual Basic program loads, opens, or closes a form. Keyboard and mouse events occur when a user does something. Program events occur when Visual Basic code does something.

### Events Table

Name of the Event	Associated with.	Description
Form_Load()	Form object (program event)	Occurs when the form is loaded
Objectname_Change()	Textbox Control (Keyboard Event)	Occurs when any key on the keyboard is pressed in the textbox
Objectname_KeyPress()	Textbox Control (Keyboard Event)	Occurs when a specific key is pressed on the keyboard
Objectname_LostFocus()	All Controls (Keyboard Events)	Occurs when the cursor is shifted from the control
Objectname_GotFocus()	All Controls (Keyboard Events)	Occurs when the cursor is focused on the control.
Objectname_Click()	All Controls (Mouse Events)	Occurs when the control is clicked
Objectname_MouseMove()	All Controls (Mouse Events)	Occurs when the mouse is moved on the control
Objectname_MouseUp()	All controls (Mouse Events)	Occurs when the mouse button is released
Objectname_MouseDown()	All controls (Mouse Events)	Occurs when the mouse button is kept pressed.

**Load Event**: Occurs when a form is loaded. For a startup form, occurs when an application starts as the result of a **Load** statement or as the result of a reference to an unloaded form's properties or controls.

This example loads items into a **ComboBox** control when a form is loaded. To try this example, paste the code into the Declarations section of a form that contains a **ComboBox**, and then press F5.

```
Private Sub Form_Load ()
    Combo1.AddItem "Mozart" ' Add items to list.
    Combo1.AddItem "Beethoven"
    Combo1.AddItem "Rock 'n Roll"
    Combo1.AddItem "Reggae"
    Combo1.ListIndex = 2 ' Set default selection.
End Sub
```

**Unload Event**: Occurs when a form is about to be removed from the screen. When that form is reloaded, the contents of all its controls are reinitialized. This event is triggered by a user closing the form using the Close command on the Control menu or an **Unload** statement.

This example demonstrates a simple procedure to close a form while prompting the user with various message boxes.

```
Private Sub Form_Unload (Cancel As Integer)
    Dim Msg, Response ' Declare variables.
    Msg = "Save Data before closing?"
    Response = MsgBox(Msg, vbQuestion + vbYesNoCancel, "Save Dialog")
    Select Case Response
    Case vbCancel ' Don't allow close.
        Cancel = -1
        Msg = "Command has been canceled."
    Case vbYes
        ' Enter code to save data here.
        Msg = "Data saved."
    Case vbNo
        Msg = "Data not saved."
    End Select
    MsgBox Msg, vbOKOnly, "Confirm" ' Display message.
End Sub
```

## Functions

If you had to create everything from scratch every time you wrote a program, it would be tedious indeed. The same kind of functionality is often required in many programs – reading data from the keyboard, for example or displaying information on the screen.

Thus programming languages come supplied with considerable quantities of pre-written code called **functions** that provides standard facilities such as these, so you don't have to write code for them yourself. Thus a function is a self-contained block of code with a specific purpose.

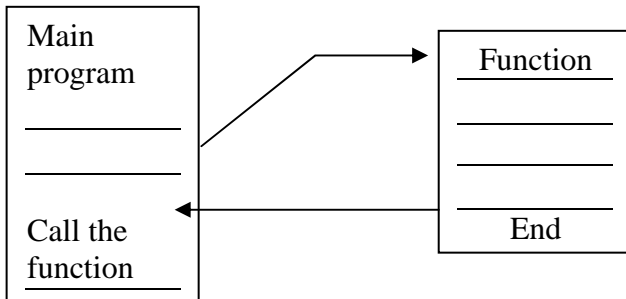
### Why use functions in a program?

1. Functions can also be reused. For example, suppose that you wrote a function to sort strings for use in one particular program. There is nothing to prevent you from using that same function in another context.
2. Breaking your program down into several functions can reduce the amount of memory required to run it. Most applications involve some calculations that is used repeatedly. If the code for such a calculation is contained in a function that you call when needed, then the code for that calculation is written just once. Without such a function, it would be necessary to repeat the code each time it was needed, and so the compiled program would be larger.

Visual Basic includes built-in, or intrinsic functions, like Sqr, Cos or Chr. In addition, you can use the Function statement to write your own Function

Usually the code in a function needs some information about the state of the program to do its job. This information consists of variables passed to the function when it is called. When a variable is passed to a function, it is called an *argument*.

1. Generally, you call a function by including the function name and arguments on the right side of a larger statement or expression (*returnvalue = function()*).
2. Functions have data types, just as variables do. This determines the type of the return value. (In the absence of an As clause, the type is the default Variant type.)
3. A function call passes execution control from the main program to the called function. The arguments, if any, are passed by value to the called function.



After executing the instructions in the function the control is again passed back to the calling program and program execution continues from that point onwards.

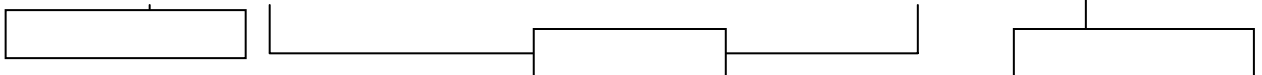
Example of a builtin function in visual basic

### InputBox function

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns a String containing the contents of the text box.

### Syntax

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



Arguments define type of values passed on to a function when it is called

```
Private Sub Command1_Click()
    primeno = InputBox("enter a prime number")
    MsgBox "you entered " & primeno
End Sub
```

### Arrays

We already know how to declare and initialize variables of the basic data types each variable can store a single data item of the specified type- we can have a variable that stores a integer or a variable that stores a string. An **array** can store several data items of the same type. You can have an array of integers or an array of strings – in fact an array of any type of data. Arrays allow you to refer to a series of variables by the same name and to use a number (an index) to tell them apart. This helps you create smaller and simpler code in many situations, because you can set up loops that deal efficiently with any number of cases by using the index number.

How arrays are stored in memory?

An array is simply a sequence of memory locations, each of which can store a single item of data of the same data type, and all of which are referenced through the same variable name. E.g

```
Dim temperatures(365) as double
```

This declares an array of the type double with the name temperature and with 366 elements. This means this array has 366 memory locations each of which can be used to hold a value of type double. The number of elements specified between brackets is called size of array.

You refer to individual elements in an array by using an integer referred to as index. The first element has an index of 0. e.g

```
temperatures(0)=99
```

The basic structure of an array is shown below

27	29	35	67	23	79
----	----	----	----	----	----

Height(0)          Height(1)          Height(2)          Height(3)          Height(4)          Height(5)

The array is of type integer and has six elements . each box represents the memory location holding the value of an array element. The array is declared as Height(5)

The compiler will allocate six contiguous storage locations for values of type integer . if an integer requires 2 bytes then this array will occupy 12 bytes in memory.

The type of array will determine the amount of memory required for storing each element. All elements of an array are stored in one , contiguous block of memory. Because Visual Basic allocates space for each index number, avoid declaring an array larger than necessary.

Array declarations

The following declarations are also valid

Dim Counters (14) As Integer

Dim Counters (1 To 15) As Integer

Dim Counters (100 To 120) As String

### Types of arrays

In Visual Basic there are two types of arrays: a **fixed-size array** which always remains the same size, and a **dynamic array** whose size can change at run-time.

### **Dynamic Arrays**

Sometimes you may not know exactly how large to make an array. You may want to have the capability of changing the size of the array at run time. A dynamic array can be resized at any time. Dynamic arrays are among the most flexible and convenient features in Visual Basic, and they help you to manage memory efficiently. For example, you can use a large array for a short time and then free up memory to the system when you're no longer using the array. The alternative is to declare an array with the largest possible size and then ignore array elements you don't need. However, this approach, if overused, might cause the operating environment to run low on memory.

### **To create a dynamic array**

1. Declare the array with a Dim statement in a procedure (if you want the array to be local). You declare the array as dynamic by giving it an empty dimension list.

```
Dim DynArray ()
```

2. Allocate the actual number of elements with a ReDim statement.

```
ReDim DynArray (X )
```

So The **ReDim** statement is used to size or resize a dynamic array that has already been formally declared. Eg:

```
Dim I As Integer, j As Integer
```

```
Dim a() As Integer ' Declared as Dynamic array
```

```
X = inputbox ("Enter number of elements in the array : ")
```

```
Redim a(1 To x)
```

Each time you execute the ReDim statement, all the values currently stored in the array are lost. Visual Basic resets the values to the Empty value (for Variant arrays), to zero (for numeric arrays), to a zero-length string (for string arrays), or to Nothing (for arrays of objects). This is useful when you want to prepare the array for new data.

### Multidimensional Arrays

Sometimes you need to keep track of related information in an array. For example, to keep track of each pixel on your computer screen, you need to refer to its X and Y coordinates. This can be done using a multidimensional array to store the values. With Visual Basic, you can declare arrays of multiple dimensions. For example, the following statement declares a two-dimensional 10-by-10 array within a procedure:

```
Dim MatrixA (9, 9) As Double
```

Either or both dimensions can be declared with explicit lower bounds:

```
Dim MatrixA (1 To 10, 1 To 10) As Double
```

You can extend this to more than two dimensions. For example:

```
Dim MultiD (3, 1 To 10, 1 To 15)
```

This declaration creates an array that has three dimensions with sizes 4 by 10 by 15. The total number of elements is the product of these three dimensions, or 600.

### Using Loops to Manipulate Arrays

You can efficiently process a multidimensional array by using nested For loops. For example, these statements initialize every element in MatrixA to a value based on its location in the array:

```
Dim I As Integer, J As Integer
```

```
Dim MatrixA(1 To 10, 1 To 10) As Double
```

```
For I = 1 To 10
```

```
    For J = 1 To 10
```

```
        MatrixA(I, J) = I * 10 + J
```

```
    Next J
```

```
Next I
```