VB.NET - VARIABLES

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in VB.Net has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

We have already discussed various data types. The basic value types provided in VB.Net can be categorized as:

Type	Example
Integral types	SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong and Char
Floating point types	Single and Double
Decimal types	Decimal
Boolean types	True or False values, as assigned
Date types	Date

VB.Net also allows defining other value types of variable like **Enum** and reference types of variables like **Class**. We will discuss date types and Classes in subsequent chapters.

Variable Declaration in VB.Net

The **Dim** statement is used for variable declaration and storage allocation for one or more variables. The Dim statement is used at module, class, structure, procedure or block level.

Syntax for variable declaration in VB.Net is:

```
[ < attributelist> ] [ accessmodifier ] [[ Shared ] [ Shadows ] | [ Static ]]
[ ReadOnly ] Dim [ WithEvents ] variablelist
```

Where,

- ***attributelist*** is a list of attributes that apply to the variable. Optional.
- ***accessmodifier*** defines the access levels of the variables, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
- ***Shared*** declares a shared variable, which is not associated with any specific instance of a class or structure, rather available to all the instances of the class or structure. Optional.
- ***Shadows*** indicate that the variable re-declares and hides an identically named element, or set of overloaded elements, in a base class. Optional.
- ***Static*** indicates that the variable will retain its value, even when the after termination of the procedure in which it is declared. Optional.
- ***ReadOnly*** means the variable can be read, but not written. Optional.
- ***WithEvents*** specifies that the variable is used to respond to events raised by the instance assigned to the variable. Optional.
- ***Variablelist*** provides the list of variables declared.

Each variable in the variable list has the following syntax and parts:

```
variablename[ ( [ boundslist ] ) ] [ As [ New ] datatype ] [ = initializer ]
```

Where,

- ***variablename***: is the name of the variable
- ***boundslist***: optional. It provides list of bounds of each dimension of an array variable.
- ***New***: optional. It creates a new instance of the class when the Dim statement runs.
- ***datatype***: Required if Option Strict is On. It specifies the data type of the variable.
- ***initializer***: Optional if New is not specified. Expression that is evaluated and assigned to the variable when it is created.

Some valid variable declarations along with their definition are shown here:

```
Dim StudentID As Integer
Dim StudentName As String
Dim Salary As Double
Dim count1, count2 As Integer
Dim status As Boolean
Dim exitButton As New System.Windows.Forms.Button
Dim lastTime, nextTime As Date
```

Variable Initialization in VB.Net

Variables are initialized *assignedavalue* with an equal sign followed by a constant expression. The general form of initialization is:

```
variable_name = value;
```

for example,

```
Dim pi As Double
pi = 3.14159
```

You can initialize a variable at the time of declaration as follows:

```
Dim StudentID As Integer = 100
Dim StudentName As String = "Bill Smith"
```

Example

Try the following example which makes use of various types of variables:

```
Module variablesNdataypes
   Sub Main()
      Dim a As Short
      Dim b As Integer
      Dim c As Double
      a = 10
      b = 20
      c = a + b
      Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c)
      Console.ReadLine()
   End Sub
End Module
```

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

```
a = 10, b = 20, c = 30
```

Accepting Values from User

The Console class in the System namespace provides a function **ReadLine** for accepting input from the user and store it into a variable. For example,

```
Dim message As String
message = Console.ReadLine
```

The following example demonstrates it:

```
Module variablesNdataypes
   Sub Main()
      Dim message As String
      Console.Write("Enter message: ")
      message = Console.ReadLine
      Console.WriteLine()
      Console.WriteLine("Your Message: {0}", message)
      Console.ReadLine()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result *assumetheuserinputsHelloWorld*:

```
Enter message: Hello World
Your Message: Hello World
```

Lvalues and Rvalues

There are two kinds of expressions:

- **lvalue :** An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue :** An expression that is an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side. Following is a valid statement:

```
Dim g As Integer = 20
```

But following is not a valid statement and would generate compile-time error:

```
20 = g
```