## Visual Basic.Net

(Introduction to VB.NET)

For

BCA Students

**ANANT KUMAR**

Faculty Member

Department of Computer Science

J. D. Women’s College, Patna

**Introduction to VB.NET**

## Identifiers

An identifier is a name used to identify a class, variable, function, or any other user-defined item. The basic rules for naming classes in VB.Net are as follows –

* A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore. The first character in an identifier cannot be a digit.
* It must not contain any embedded space or symbol like ? - +! @ # % ^ & \* ( ) [ ] { } . ; : " ' / and \. However, an underscore ( \_ ) can be used.
* It should not be a reserved keyword.
* It should not more than 51 characters.

## VB.Net Keywords

A **keyword** is a reserved word with special meanings in the compiler, whose meaning cannot be changed. Therefore, these keywords cannot be used as an identifier in **VB.NET** programming such as class name, variable, function, module, etc.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| AddHandler | AddressOf | Alias | And | AndAlso | As | Boolean |
| ByRef | Byte | ByVal | Call | Case | Catch | CBool |
| CByte | CChar | CDate | CDec | CDbl | Char | CInt |
| Class | CLng | CObj | Const | Continue | CSByte | CShort |
| CSng | CStr | CType | CUInt | CULng | CUShort | Date |
| Decimal | Declare | Default | Delegate | Dim | DirectCast | Do |
| Double | Each | Else | ElseIf | End | End If | Enum |
| Erase | Error | Event | Exit | False | Finally | For |
| Friend | Function | Get | GetType | GetXML Namespace | Global | GoTo |
| Handles | If | Implements | Imports | In | Inherits | Integer |
| Interface | Is | IsNot | Let | Lib | Like | Long |
| Loop | Me | Mod | Module | MustInherit | MustOverride | MyBase |
| MyClass | Namespace | Narrowing | New | Next | Not | Nothing |
| Not Inheritable | Not Overridable | Object | Of | On | Operator | Option |
| Optional | Or | OrElse | Overloads | Overridable | Overrides | ParamArray |
| Partial | Private | Property | Protected | Public | RaiseEvent | ReadOnly |
| ReDim | REM | Remove Handler | Resume | Return | SByte | Select |
| Set | Shadows | Shared | Short | Single | Static | Step |
| Stop | String | Structure | Sub | SyncLock | Then | Throw |
| To | True | Try | TryCast | TypeOf | UInteger | While |
| Widening | With | WithEvents | WriteOnly | Xor |  |  |

## VB.NET Comments

A comment is used to explain the various steps that we have taken in our programming. The compiler ignores these comment statements because the compiler is not executed or processed in VB.NET. Therefore, it does not take any place in your compilation code.

In VB.NET, we use ( **'** ) symbol to comment a statement.

## Data Types in VB.Net

## A Data Type refers to which type of data or value is assigning to a variable or function so that a variable can hold a defined data type value. For example, when we declare a variable, we have to tell the compiler what type of data or value is allocated to different kinds of variables to hold different amounts of space in computer memory.

**Different Data Types and their allocating spaces in VB.NET**

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Storage Allocation** | **Value Range** |
| Boolean | Depends on implementing platform | **True** or **False** |
| Byte | 1 byte | 0 through 255 (unsigned) |
| Char | 2 bytes | 0 through 65535 (unsigned) |
| Date | 8 bytes | 0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999 |
| Decimal | 16 bytes | 0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal |
| Double | 8 bytes | -1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values  4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values |
| Integer | 4 bytes | -2,147,483,648 through 2,147,483,647 (signed) |
| Long | 8 bytes | -9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed) |
| Object | 4 bytes on 32-bit platform  8 bytes on 64-bit platform | Any type can be stored in a variable of type Object |
| SByte | 1 byte | -128 through 127 (signed) |
| Short | 2 bytes | -32,768 through 32,767 (signed) |
| Single | 4 bytes | -3.4028235E+38 through -1.401298E-45 for negative values;  1.401298E-45 through 3.4028235E+38 for positive values |
| String | Depends on implementing platform | 0 to approximately 2 billion Unicode characters |
| UInteger | 4 bytes | 0 through 4,294,967,295 (unsigned) |
| ULong | 8 bytes | 0 through 18,446,744,073,709,551,615 (unsigned) |
| User-Defined | Depends on implementing platform | Each member of the structure has a range determined by its data type and independent of the ranges of the other members |
| UShort | 2 bytes | 0 through 65,535 (unsigned) |

## Example

Module DataTypes

Sub Main()

Dim b As Byte

Dim n As Integer

Dim si As Single

Dim d As Double

Dim da As Date

Dim c As Char

Dim s As String

Dim bl As Boolean

b = 1

n = 1234567

si = 0.12345678901234566

d = 0.12345678901234566

da = Today

c = "U"c

s = "Me"

If ScriptEngine = "VB" Then

bl = True

Else

bl = False

End If

If bl Then

'the oath taking

Console.Write(c & " and," & s & vbCrLf)

Console.WriteLine("declaring on the day of: {0}", da)

Console.WriteLine("We will learn VB.Net seriously")

Console.WriteLine("Lets see what happens to the floating point variables:")

Console.WriteLine("The Single: {0}, The Double: {1}", si, d)

End If

Console.ReadKey()

End Sub

End Module

## The Type Conversion Functions in VB.Net

VB.Net provides the following in-line type conversion functions –

|  |  |
| --- | --- |
| **Sr.No.** | **Functions & Description** |
| 1 | **CBool(expression)**  Converts the expression to Boolean data type. |
| 2 | **CByte(expression)**  Converts the expression to Byte data type. |
| 3 | **CChar(expression)**  Converts the expression to Char data type. |
| 4 | **CDate(expression)**  Converts the expression to Date data type |
| 5 | **CDbl(expression)**  Converts the expression to Double data type. |
| 6 | **CDec(expression)**  Converts the expression to Decimal data type. |
| 7 | **CInt(expression)**  Converts the expression to Integer data type. |
| 8 | **CLng(expression)**  Converts the expression to Long data type. |
| 9 | **CObj(expression)**  Converts the expression to Object type. |
| 10 | **CSByte(expression)**  Converts the expression to SByte data type. |
| 11 | **CShort(expression)**  Converts the expression to Short data type. |
| 12 | **CSng(expression)**  Converts the expression to Single data type. |
| 13 | **CStr(expression)**  Converts the expression to String data type. |
| 14 | **CUInt(expression)**  Converts the expression to UInt data type. |
| 15 | **CULng(expression)**  Converts the expression to ULng data type. |
| 16 | **CUShort(expression)**  Converts the expression to UShort data type. |

Option Strict On

Module DB\_Conversion

    Sub Main()

    'defining the Data type conversion

        Dim dblData As Double

        dblData = 5.78

        Dim A, B As Char

        Dim bool As Boolean = True

        Dim x, Z, B\_int As Integer

        A = "A"

        B = "B"

        B\_int = AscW(B)

        Console.WriteLine(" Ascii value of B is {0}", B\_int)

        x = 1

        Z = AscW(A)

        Z = Z + x

        Console.WriteLine("String to integer {0}", Z)

        Console.WriteLine("Boolean value is : {0}", CStr(bool))

        Dim num, intData As Integer

        num = CInt(dblData)

        intData = CType(dblData, Integer)

        Console.WriteLine(" Explicit conversion of Data type " & Str(intData))

        Console.WriteLine(" Value of Double is: {0}", dblData)

        Console.WriteLine("Double to Integer: {0}", num)

        Console.ReadKey()

    End Sub

End Module

**Output:**

Ascii value of B is 66

String to integer 66

Boolean value is: True

Explicit conversion of Data type 6

Value of Double is: 5.78

Double to Integer: 6

## 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**

Dim Variable\_Name as DataType

VariableName: It defines the name of the variable that you assign to store values.

DataType: It represents the name of the data type that you assign to a variable.

**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 −

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 (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is –

variable\_name = value;

**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

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.

Dim message As String

message = Console.ReadLine()

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

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 cannot appear on the left-hand side. Valid statement is −

Dim g As Integer = 20

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

20 = g