VB.NET
programming language reference

www.tutorialspoint.com
About the Tutorial

VB.Net is a simple, modern, object-oriented computer programming language developed by Microsoft to combine the power of .NET Framework and the common language runtime with the productivity benefits that are the hallmark of Visual Basic.

This tutorial will teach you basic VB.Net programming and will also take you through various advanced concepts related to VB.Net programming language.

Audience

This tutorial has been prepared for the beginners to help them understand basic VB.Net programming. After completing this tutorial, you will find yourself at a moderate level of expertise in VB.Net programming from where you can take yourself to next levels.

Prerequisites

VB.Net programming is very much based on BASIC and Visual Basic programming languages, so if you have basic understanding on these programming languages, then it will be a fun for you to learn VB.Net programming language.

Copyright & Disclaimer

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book can retain a copy for future reference but commercial use of this data is not allowed. Distribution or republishing any content or a part of the content of this e-book in any manner is also not allowed without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial..............................................................................................................................................................1
1. VB.NET – OVERVIEW .................................................................................. 8
   Strong Programming Features VB.Net .......................................................... 8

2. VB.NET – ENVIRONMENT SETUP ............................................................... 10
   The .Net Framework ................................................................................... 10
   Integrated Development Environment (IDE) For VB.Net ............................. 11
   Writing VB.Net Programs on Linux or Mac OS .......................................... 11

3. VB.NET – PROGRAM STRUCTURE ............................................................. 12
   VB.Net Hello World Example ...................................................................... 12
   Compile & Execute VB.Net Program .......................................................... 13

4. VB.NET – BASIC SYNTAX .......................................................................... 15
   A Rectangle Class in VB.Net ........................................................................ 15
   Identifiers .................................................................................................... 17
   VB.Net Keywords ....................................................................................... 17

5. VB.NET – DATA TYPES ............................................................................ 19
   Data Types Available in VB.Net .................................................................. 19
   Example ...................................................................................................... 21
   The Type Conversion Functions in VB.Net ................................................. 22
   Example ...................................................................................................... 24

6. VB.NET – VARIABLES .............................................................................. 25
Variable Declaration in VB.Net

Variable Initialization in VB.Net

Example

Accepting Values from User

Lvalues and Rvalues

7. VB.NET – CONSTANTS AND ENUMERATIONS

Declaring Constants

Example

Print and Display Constants in VB.Net

Declaring Enumerations

Example

8. VB.NET – MODIFIERS

List of Available Modifiers in VB.Net

9. VB.NET – STATEMENTS

Declaration Statements

Executable Statements

10. VB.NET – DIRECTIVES

Compiler Directives in VB.Net

11. VB.NET – OPERATORS

Arithmetic Operators

Example

Comparison Operators

Logical/Bitwise Operators
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit Statement</td>
<td>..........................................................</td>
<td>92</td>
</tr>
<tr>
<td>Continue Statement</td>
<td>..........................................................</td>
<td>94</td>
</tr>
<tr>
<td>GoTo Statement</td>
<td>..........................................................</td>
<td>95</td>
</tr>
<tr>
<td>14. VB.NET – STRINGS</td>
<td>..........................................................</td>
<td>98</td>
</tr>
<tr>
<td>Creating a String Object</td>
<td>..........................................................</td>
<td>98</td>
</tr>
<tr>
<td>Properties of the String Class</td>
<td>..........................................................</td>
<td>99</td>
</tr>
<tr>
<td>Methods of the String Class</td>
<td>..........................................................</td>
<td>99</td>
</tr>
<tr>
<td>Examples</td>
<td>..........................................................</td>
<td>105</td>
</tr>
<tr>
<td>15. VB.NET – DATE &amp; TIME</td>
<td>..........................................................</td>
<td>108</td>
</tr>
<tr>
<td>Properties and Methods of the DateTime Structure</td>
<td>..........................................................</td>
<td>109</td>
</tr>
<tr>
<td>Creating a DateTime Object</td>
<td>..........................................................</td>
<td>112</td>
</tr>
<tr>
<td>Getting the Current Date and Time</td>
<td>..........................................................</td>
<td>113</td>
</tr>
<tr>
<td>Formatting Date</td>
<td>..........................................................</td>
<td>114</td>
</tr>
<tr>
<td>Predefined Date/Time Formats</td>
<td>..........................................................</td>
<td>115</td>
</tr>
<tr>
<td>Properties and Methods of the DateAndTime Class</td>
<td>..........................................................</td>
<td>117</td>
</tr>
<tr>
<td>16. ARRAYS</td>
<td>..........................................................</td>
<td>121</td>
</tr>
<tr>
<td>Creating Arrays in VB.Net</td>
<td>..........................................................</td>
<td>121</td>
</tr>
<tr>
<td>Dynamic Arrays</td>
<td>..........................................................</td>
<td>122</td>
</tr>
<tr>
<td>Multi-Dimensional Arrays</td>
<td>..........................................................</td>
<td>124</td>
</tr>
<tr>
<td>Jagged Array</td>
<td>..........................................................</td>
<td>125</td>
</tr>
<tr>
<td>The Array Class</td>
<td>..........................................................</td>
<td>126</td>
</tr>
<tr>
<td>17. VB.NET – COLLECTIONS</td>
<td>..........................................................</td>
<td>131</td>
</tr>
<tr>
<td>Various Collection Classes and Their Usage</td>
<td>..........................................................</td>
<td>131</td>
</tr>
<tr>
<td>ArrayList</td>
<td>Hashtable</td>
<td>SortedList</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VB.NET – SUB PROCEDURES</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining Sub Procedures</td>
<td>155</td>
</tr>
<tr>
<td>Example</td>
<td>155</td>
</tr>
<tr>
<td>Passing Parameters by Value</td>
<td>156</td>
</tr>
<tr>
<td>Passing Parameters by Reference</td>
<td>157</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VB.NET – CLASSES &amp; OBJECTS</th>
<th>159</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Definition</td>
<td>159</td>
</tr>
<tr>
<td>Member Functions and Encapsulation</td>
<td>161</td>
</tr>
<tr>
<td>Constructors and Destructors</td>
<td>162</td>
</tr>
<tr>
<td>Shared Members of a VB.Net Class</td>
<td>165</td>
</tr>
<tr>
<td>Inheritance</td>
<td>166</td>
</tr>
</tbody>
</table>
Base & Derived Classes......................................................................................................................166
Base Class Initialization ....................................................................................................................168

21. VB.NET – EXCEPTION HANDLING ........................................................................................................170
   Syntax ................................................................................................................................................170
   Exception Classes in .Net Framework ........................................................................................171
   Handling Exceptions ......................................................................................................................172
   Creating User-Defined Exceptions .............................................................................................173
   Throwing Objects ..........................................................................................................................174

22. VB.NET – FILE HANDLING ..................................................................................................................175
   Binary Files ....................................................................................................................................182

23. VB.NET – BASIC CONTROLS ...............................................................................................................192

24. VB.NET – DIALOG BOXES ....................................................................................................................285

25. VB.NET – ADVANCED FORM ..............................................................................................................307

26. VB.NET – EVENT HANDLING ............................................................................................................330

27. VB.NET – REGULAR EXPRESSIONS ..................................................................................................336

28. VB.NET – DATABASE ACCESS ..........................................................................................................350

29. VB.NET – EXCEL SHEET ....................................................................................................................365

30. VB.NET – SEND EMAIL ......................................................................................................................370

31. VB.NET – XML PROCESSING ............................................................................................................376

32. VB.NET – WEB PROGRAMMING ......................................................................................................391
1. VB.NET – Overview

Visual Basic .NET (VB.NET) is an object-oriented computer programming language implemented on the .NET Framework. Although it is an evolution of classic Visual Basic language, it is not backwards-compatible with VB6, and any code written in the old version does not compile under VB.NET.

Like all other .NET languages, VB.NET has complete support for object-oriented concepts. Everything in VB.NET is an object, including all of the primitive types (Short, Integer, Long, String, Boolean, etc.) and user-defined types, events, and even assemblies. All objects inherits from the base class Object.

VB.NET is implemented by Microsoft’s .NET framework. Therefore, it has full access to all the libraries in the .Net Framework. It's also possible to run VB.NET programs on Mono, the open-source alternative to .NET, not only under Windows, but even Linux or Mac OSX.

The following reasons make VB.Net a widely used professional language:

- Modern, general purpose.
- Object oriented.
- Component oriented.
- Easy to learn.
- Structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.

Strong Programming Features VB.Net

VB.Net has numerous strong programming features that make it endearing to multitude of programmers worldwide. Let us mention some of these features:

- Boolean Conditions
- Automatic Garbage Collection
- Standard Library
• Assembly Versioning
• Properties and Events

• Delegates and Events Management
• Easy-to-use Generics
• Indexers
• Conditional Compilation
• Simple Multithreading
In this chapter, we will discuss the tools available for creating VB.Net applications. We have already mentioned that VB.Net is part of .Net framework and used for writing .Net applications. Therefore before discussing the available tools for running a VB.Net program, let us understand how VB.Net relates to the .Net framework.

The .Net Framework

The .Net framework is a revolutionary platform that helps you to write the following types of applications:

- Windows applications
- Web applications
- Web services

The .Net framework applications are multi-platform applications. The framework has been designed in such a way that it can be used from any of the following languages: Visual Basic, C#, C++, Jscript, and COBOL, etc.

All these languages can access the framework as well as communicate with each other.

The .Net framework consists of an enormous library of codes used by the client languages like VB.Net. These languages use object-oriented methodology.

Following are some of the components of the .Net framework:

- Common Language Runtime (CLR)
- The .Net Framework Class Library
- Common Language Specification
- Common Type System
- Metadata and Assemblies
- Windows Forms
- ASP.Net and ASP.Net AJAX
- ADO.Net
• Windows Workflow Foundation (WF)
• Windows Presentation Foundation
• Windows Communication Foundation (WCF)
• LINQ

For the jobs each of these components perform, please see ASP.Net - Introduction, and for details of each component, please consult Microsoft's documentation.

Integrated Development Environment (IDE) For VB.Net

Microsoft provides the following development tools for VB.Net programming:

• Visual Studio 2010 (VS)
• Visual Basic 2010 Express (VBE)
• Visual Web Developer

The last two are free. Using these tools, you can write all kinds of VB.Net programs from simple command-line applications to more complex applications. Visual Basic Express and Visual Web Developer Express edition are trimmed down versions of Visual Studio and have the same look and feel. They retain most features of Visual Studio. In this tutorial, we have used Visual Basic 2010 Express and Visual Web Developer (for the web programming chapter).

You can download it from here. It gets automatically installed in your machine. Please note that you need an active internet connection for installing the express edition.

Writing VB.Net Programs on Linux or Mac OS

Although the .NET Framework runs on the Windows operating system, there are some alternative versions that work on other operating systems. Mono is an open-source version of the .NET Framework which includes a Visual Basic compiler and runs on several operating systems, including various flavors of Linux and Mac OS. The most recent version is VB 2012.

The stated purpose of Mono is not only to be able to run Microsoft .NET applications cross-platform, but also to bring better development tools to Linux developers. Mono can be run on many operating systems including Android, BSD, iOS, Linux, OS X, Windows, Solaris and UNIX.
Before we study basic building blocks of the VB.Net programming language, let us look a bare minimum VB.Net program structure so that we can take it as a reference in upcoming chapters.

**VB.Net Hello World Example**

A VB.Net program basically consists of the following parts:

- Namespace declaration
- A class or module
- One or more procedures
- Variables
- The Main procedure
- Statements & Expressions
- Comments

Let us look at a simple code that would print the words "Hello World":

```vbnet
Imports System
Module Module1
    'This program will display Hello World
    Sub Main()
        Console.WriteLine("Hello World")
        Console.ReadKey()
    End Sub
End Module
```

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

```
Hello, World!
```
Let us look various parts of the above program:

- The first line of the program **Imports System** is used to include the System namespace in the program.
- The next line has a **Module** declaration, the module **Module1**. VB.Net is completely object oriented, so every program must contain a module of a class that contains the data and procedures that your program uses.

- Classes or Modules generally would contain more than one procedure. Procedures contain the executable code, or in other words, they define the behavior of the class. A procedure could be any of the following:
  - Function
  - Sub
  - Operator
  - Get
  - Set
  - AddHandler
  - RemoveHandler
  - RaiseEvent
- The next line (*This program*) will be ignored by the compiler and it has been put to add additional comments in the program.

- The next line defines the Main procedure, which is the entry point for all VB.Net programs. The Main procedure states what the module or class will do when executed.

- The Main procedure specifies its behavior with the statement **Console.WriteLine** (*"Hello World"*) *WriteLine* is a method of the **Console** class defined in the **System** namespace. This statement causes the message "Hello, World!" to be displayed on the screen.

- The last line **Console.ReadKey()** is for the VS.NET Users. This will prevent the screen from running and closing quickly when the program is launched from Visual Studio .NET.

**Compile & Execute VB.Net Program**

If you are using Visual Studio.Net IDE, take the following steps:

- Start Visual Studio.
• On the menu bar, choose File → New → Project.

• Choose Visual Basic from templates

• Choose Console Application.

• Specify a name and location for your project using the Browse button, and then choose the OK button.

• The new project appears in Solution Explorer.

• Write code in the Code Editor.

• Click the Run button or the F5 key to run the project. A Command Prompt window appears that contains the line Hello World.

You can compile a VB.Net program by using the command line instead of the Visual Studio IDE:

• Open a text editor and add the above mentioned code.

• Save the file as **helloworld.vb**

• Open the command prompt tool and go to the directory where you saved the file.

• Type **vbc helloworld.vb** and press enter to compile your code.

• If there are no errors in your code the command prompt will take you to the next line and would generate **helloworld.exe** executable file.

• Next, type **helloworld** to execute your program.

• You will be able to see "Hello World" printed on the screen.
VB.Net is an object-oriented programming language. In Object-Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions. The actions that an object may take are called methods. Objects of the same kind are said to have the same type or, more often, are said to be in the same class.

When we consider a VB.Net program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods, and instant variables mean.

- **Object** - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating, etc. An object is an instance of a class.

- **Class** - A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.

- **Methods** - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

- **Instant Variables** - Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.

### A Rectangle Class in VB.Net

For example, let us consider a Rectangle object. It has attributes like length and width. Depending upon the design, it may need ways for accepting the values of these attributes, calculating area and displaying details.

Let us look at an implementation of a Rectangle class and discuss VB.Net basic syntax on the basis of our observations in it:

```vbnet
Imports System
Public Class Rectangle
    Private length As Double
    Private width As Double

    'Public methods
    Public Sub AcceptDetails()
```

---

4. VB.NET – Basic Syntax
length = 4.5
width = 3.5
End Sub

Public Function GetArea() As Double
    GetArea = length * width
End Function

Public Sub Display()
    Console.WriteLine("Length: {0}", length)
    Console.WriteLine("Width: {0}", width)
    Console.WriteLine("Area: {0}", GetArea())
End Sub

Shared Sub Main()
    Dim r As New Rectangle()
    r.Acceptdetails()
    r.Display()
    Console.ReadLine()
End Sub
End Class

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

| Length: 4.5 |
| Width: 3.5 |
| Area: 15.75 |

In previous chapter, we created a Visual Basic module that held the code. Sub Main indicates the entry point of VB.Net program. Here, we are using Class that contains both code and data. You use classes to create objects. For example, in the code, r is a Rectangle object.

An object is an instance of a class:

Dim r As New Rectangle()

A class may have members that can be accessible from outside class, if so specified. Data members are called fields and procedure members are called methods.
**Shared** methods or **static** methods can be invoked without creating an object of the class. Instance methods are invoked through an object of the class:

```vbnet
Shared Sub Main()
    Dim r As New Rectangle()
    r.AcceptDetails()
    r.Display()
    Console.ReadLine()
End Sub
```

**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.

**VB.Net Keywords**

The following table lists the VB.Net reserved keywords:

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

**tutorialsPoint**

SIMPLY EASY LEARNING
5. VB.NET – Data Types

Data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

Data Types Available in VB.Net

VB.Net provides a wide range of data types. The following table shows all the data types available:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Storage Allocation</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Depends on implementing platform</td>
<td>True or False</td>
</tr>
<tr>
<td>Byte</td>
<td>1 byte</td>
<td>0 through 255 (unsigned)</td>
</tr>
<tr>
<td>Char</td>
<td>2 bytes</td>
<td>0 through 65535 (unsigned)</td>
</tr>
<tr>
<td>Date</td>
<td>8 bytes</td>
<td>0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999</td>
</tr>
<tr>
<td>Decimal</td>
<td>16 bytes</td>
<td>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</td>
</tr>
<tr>
<td>Double</td>
<td>8 bytes</td>
<td>-1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values; 4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values</td>
</tr>
<tr>
<td>Type</td>
<td>Size</td>
<td>Range/Values</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Integer</td>
<td>4 bytes</td>
<td>-2,147,483,648 through 2,147,483,647 (signed)</td>
</tr>
<tr>
<td>Object</td>
<td>4 bytes on 32-bit platform; 8 bytes on 64-bit platform</td>
<td>Any type can be stored in a variable of type Object</td>
</tr>
<tr>
<td>SByte</td>
<td>1 byte</td>
<td>-128 through 127 (signed)</td>
</tr>
<tr>
<td>Short</td>
<td>2 bytes</td>
<td>-32,768 through 32,767 (signed)</td>
</tr>
<tr>
<td>Single</td>
<td>4 bytes</td>
<td>-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values</td>
</tr>
<tr>
<td>String</td>
<td>Depends on implementing platform</td>
<td>0 to approximately 2 billion Unicode characters</td>
</tr>
<tr>
<td>UInteger</td>
<td>4 bytes</td>
<td>0 through 4,294,967,295 (unsigned)</td>
</tr>
<tr>
<td>ULong</td>
<td>8 bytes</td>
<td>0 through 18,446,744,073,709,551,615 (unsigned)</td>
</tr>
<tr>
<td>User-Defined</td>
<td>Depends on implementing platform</td>
<td>Each member of the structure has a range determined by its data type and independent of the ranges of the other members</td>
</tr>
<tr>
<td>UShort</td>
<td>2 bytes</td>
<td>0 through 65,535 (unsigned)</td>
</tr>
</tbody>
</table>
Example

The following example demonstrates use of some of the types:

```vbcnet
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
    End Sub
End Module
```
When the above code is compiled and executed, it produces the following result:

U and, Me
declaring on the day of: 12/4/2012 12:00:00 PM
We will learn VB.Net seriously
Lets see what happens to the floating point variables:
The Single:0.1234568, The Double: 0.123456789012346

The Type Conversion Functions in VB.Net

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

<table>
<thead>
<tr>
<th>S.N</th>
<th>Functions &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>CBool(expression)</strong></td>
</tr>
<tr>
<td></td>
<td>Converts the expression to Boolean data type.</td>
</tr>
<tr>
<td>2</td>
<td><strong>CByte(expression)</strong></td>
</tr>
<tr>
<td></td>
<td>Converts the expression to Byte data type.</td>
</tr>
<tr>
<td>3</td>
<td><strong>CChar(expression)</strong></td>
</tr>
<tr>
<td></td>
<td>Converts the expression to Char data type.</td>
</tr>
<tr>
<td>4</td>
<td><strong>CDate(expression)</strong></td>
</tr>
<tr>
<td></td>
<td>Converts the expression to Date data type.</td>
</tr>
<tr>
<td>5</td>
<td><strong>CDbl(expression)</strong></td>
</tr>
<tr>
<td></td>
<td>Converts the expression to Double data type.</td>
</tr>
</tbody>
</table>

```csharp
Console.ReadKey()
End Sub
End Module
```
<table>
<thead>
<tr>
<th></th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td><code>CDec(expression)</code></td>
<td>Converts the expression to Decimal data type.</td>
</tr>
<tr>
<td>7</td>
<td><code>CInt(expression)</code></td>
<td>Converts the expression to Integer data type.</td>
</tr>
<tr>
<td>8</td>
<td><code>CLng(expression)</code></td>
<td>Converts the expression to Long data type.</td>
</tr>
<tr>
<td>9</td>
<td><code>CObj(expression)</code></td>
<td>Converts the expression to Object type.</td>
</tr>
<tr>
<td>10</td>
<td><code>CSByte(expression)</code></td>
<td>Converts the expression to SByte data type.</td>
</tr>
<tr>
<td>11</td>
<td><code>CShort(expression)</code></td>
<td>Converts the expression to Short data type.</td>
</tr>
<tr>
<td>12</td>
<td><code>CSng(expression)</code></td>
<td>Converts the expression to Single data type.</td>
</tr>
<tr>
<td>13</td>
<td><code>CStr(expression)</code></td>
<td>Converts the expression to String data type.</td>
</tr>
<tr>
<td>14</td>
<td><code>CUInt(expression)</code></td>
<td>Converts the expression to UInt data type.</td>
</tr>
<tr>
<td>15</td>
<td><code>CULng(expression)</code></td>
<td>Converts the expression to ULng data type.</td>
</tr>
</tbody>
</table>
**CUShort(expression)**
Converts the expression to UShort data type.

**Example**
The following example demonstrates some of these functions:

```vbnet
Module DataTypes
    Sub Main()
        Dim n As Integer
        Dim da As Date
        Dim bl As Boolean = True
        n = 1234567
        da = Today
        Console.WriteLine(bl)
        Console.WriteLine(CSByte(bl))
        Console.WriteLine(CStr(bl))
        Console.WriteLine(CStr(da))
        Console.WriteLine(CChar(CChar(CStr(n))))
        Console.WriteLine(CChar(CStr(da)))
        Console.ReadKey()
    End Sub
End Module
```

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

```
True
-1
True
12/4/2012
1
1
```
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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integral types</td>
<td>SByte, Byte, Short, UShort, Integer,UInteger, Long, ULong and Char</td>
</tr>
<tr>
<td>Floating point types</td>
<td>Single and Double</td>
</tr>
<tr>
<td>Decimal types</td>
<td>Decimal</td>
</tr>
<tr>
<td>Boolean types</td>
<td>True or False values, as assigned</td>
</tr>
<tr>
<td>Date types</td>
<td>Date</td>
</tr>
</tbody>
</table>

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:

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

Where,

- **variblename**: 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 (assigned a value) 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 variablesNdatatypes
    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,

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

The following example demonstrates it:

```vbnet
Module variablesNdatatypes

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 (assume the user inputs Hello World):

```
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:

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

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

```vbnet
20 = g
```
End of ebook preview
If you liked what you saw...
Buy it from our store @ https://store.tutorialspoint.com