SIT 221: Event-Driven Programming

Lecture 3 :VB.Net Program Structure

VB.Net Program Structure

- Before we study basic building blocks of the VB.Net programming language, let us look a bare minimum VB.Net program structure.
- 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" −

VB.Net Hello World Example

```
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!
```

- 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 - Basic Syntax

- 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 instance 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 objects 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.
- **Instance Variables** – Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance 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 −

```
Imports System
Public Class Rectangle
   Private length As Double
   Private width As Double
   'Public methods
Public Sub AcceptDetails()
      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

- 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 –

  e.g. Dim r As New Rectangle()

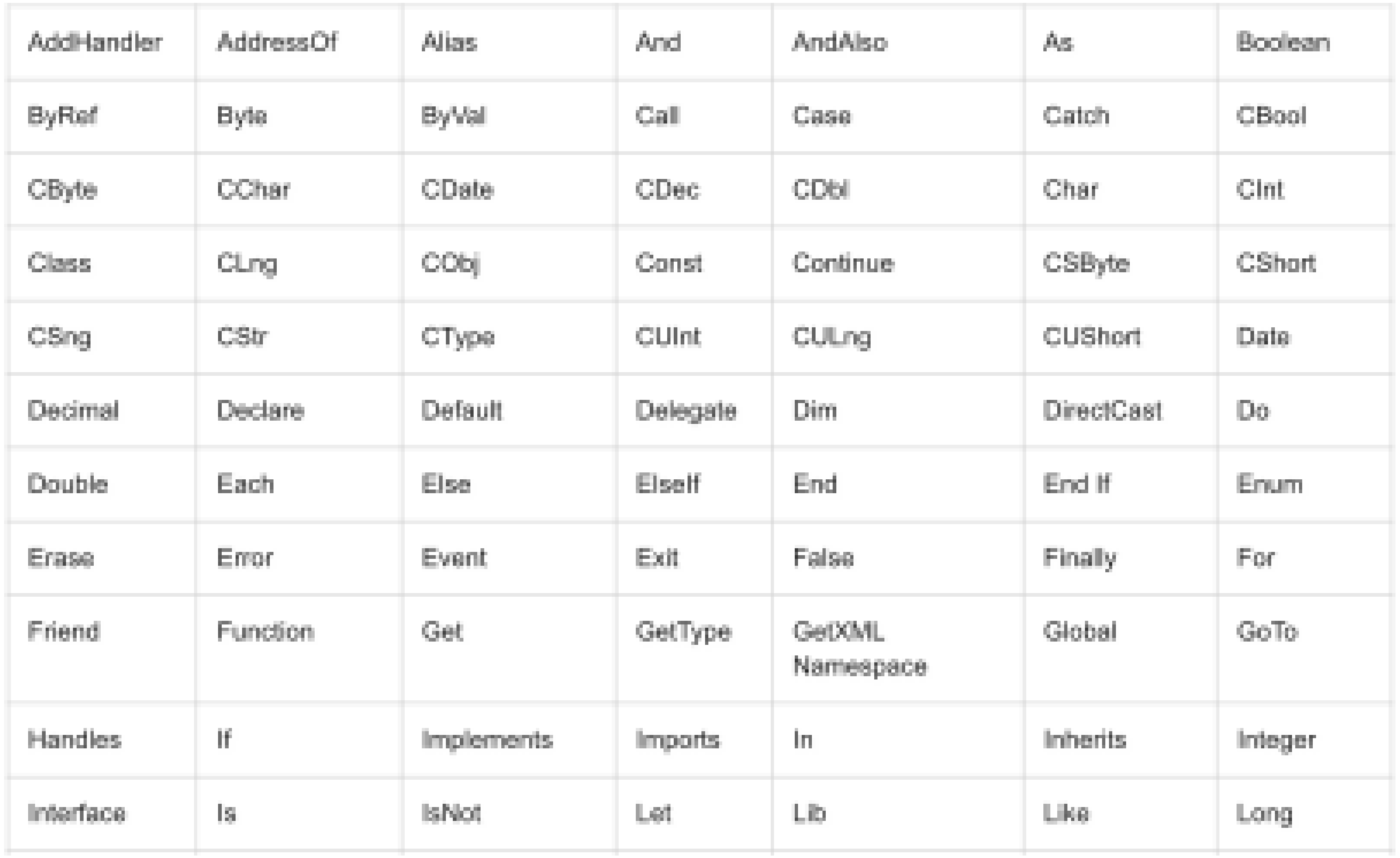
- A class may have members that can be accessible from outside class, if so specified. Data members are called fields and procedure members are called methods.
- **Shared** methods or **static** methods can be invoked without creating an object of the class. Instance methods are invoked through an object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

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

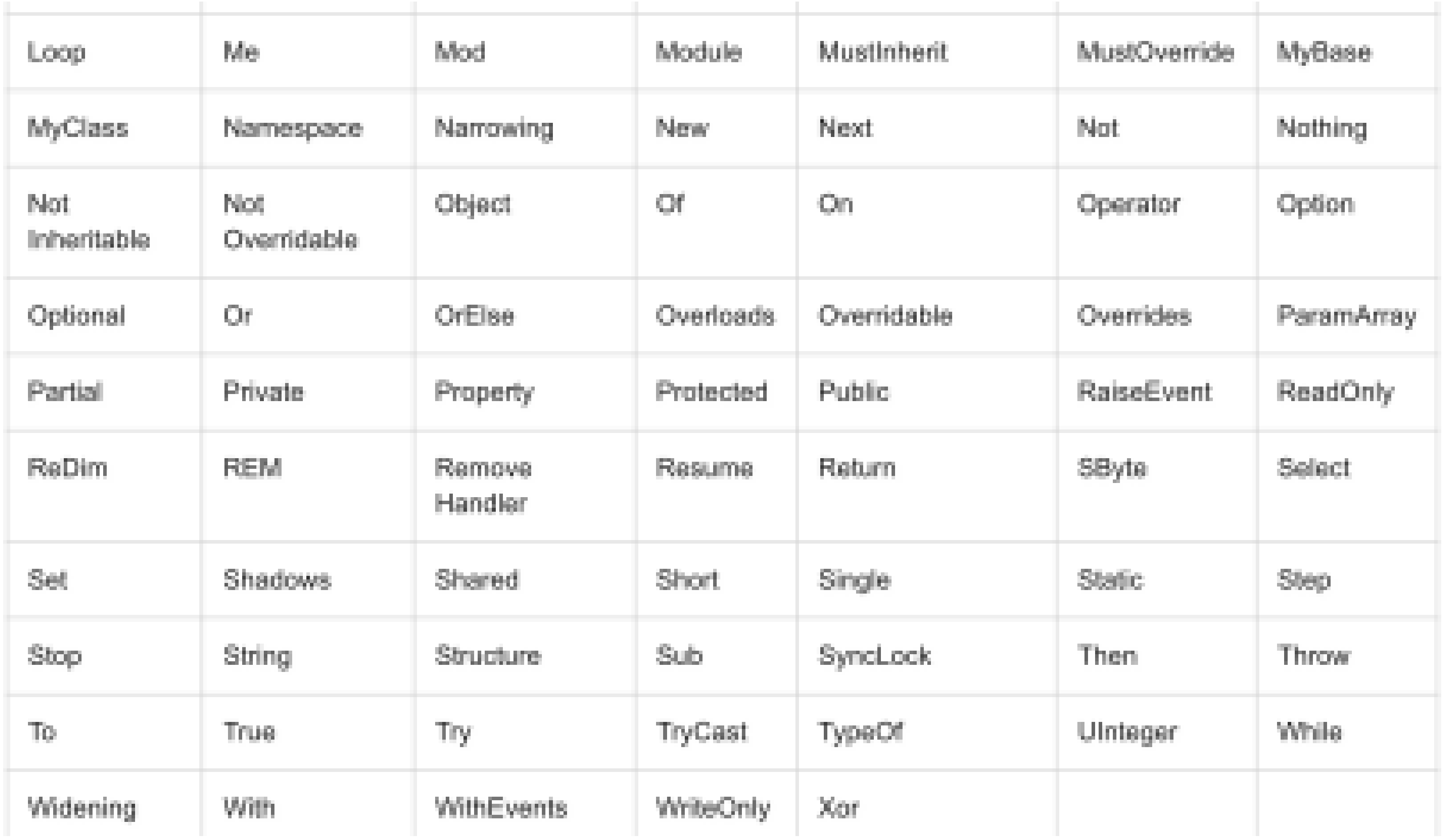
Identifiers

- 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 / Reserved Words



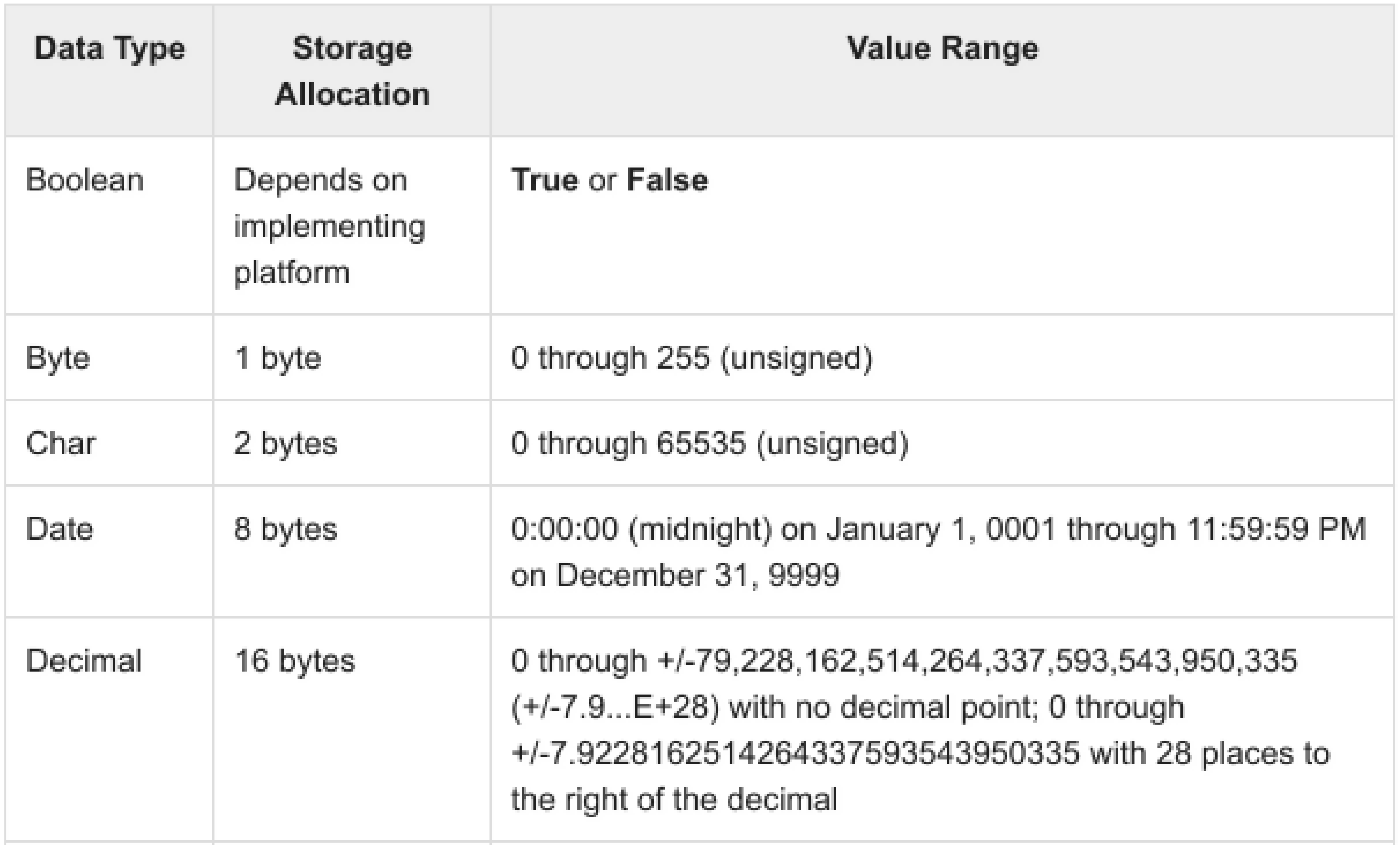
AddHandler	AddressOf	Alias	And	AndAlso	As	Boolean
ByRef	Byte	ByVal	Call	Case	Catch	CBool
CByte	CChar	CDate	CDec	CDbl	Char	CInt
Class	CLng	CObj	Const	Continue	CSByte	CShort
CSng	CStr	CType	CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate	Dim	DirectCast	Do
Double	Each	Else	ElseIf	End	End If	Enum
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



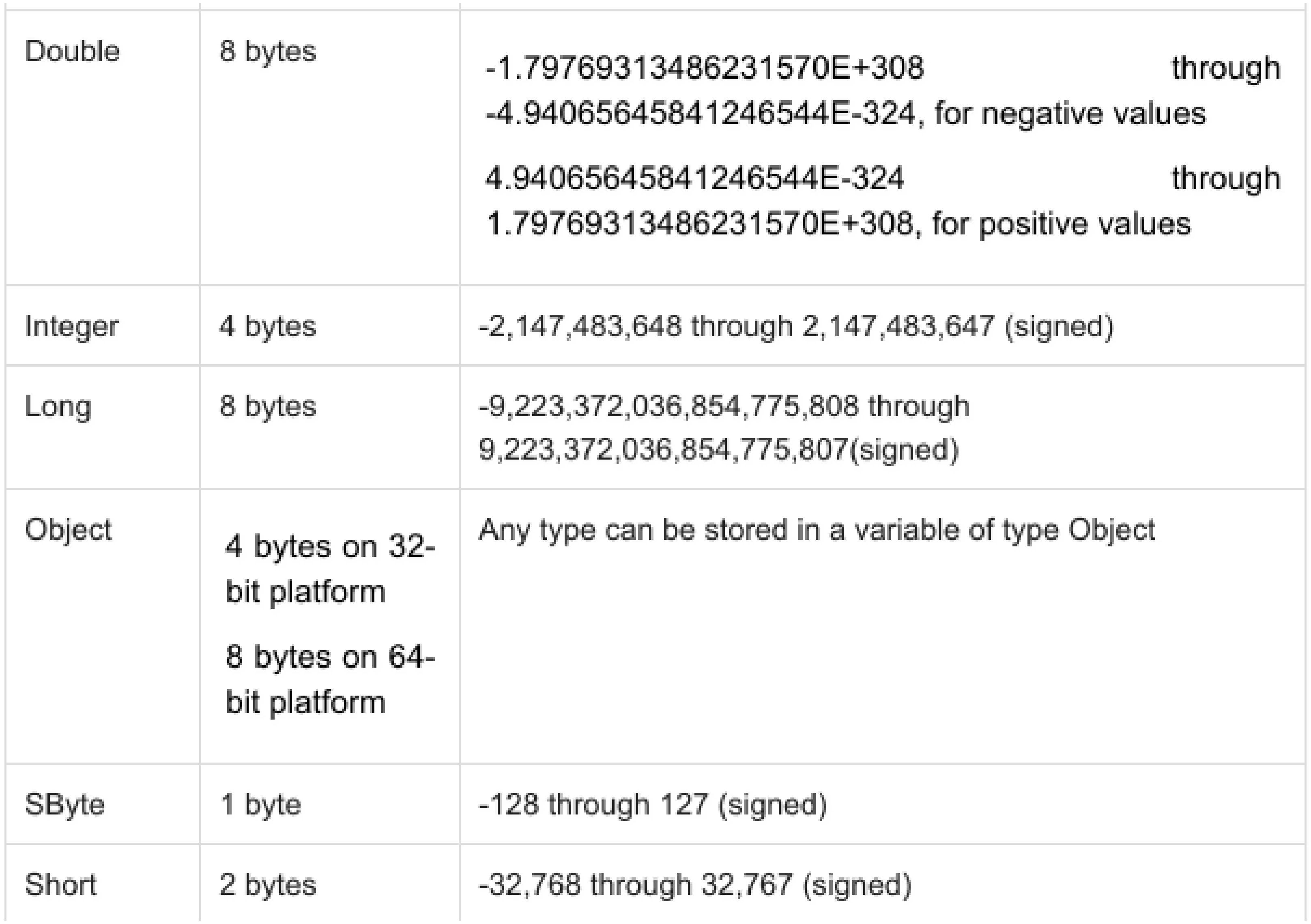
Loop	Me	Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	Narrowing	New	Next	Not	Nothing
Not Inheritable	Not Overridable	Object	Of	On	Operator	Option
Optional	Or	OrElse	Overloads	Overridable	Overrides	ParamArray
Partial	Private	Property	Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	Remove 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 - Data Types

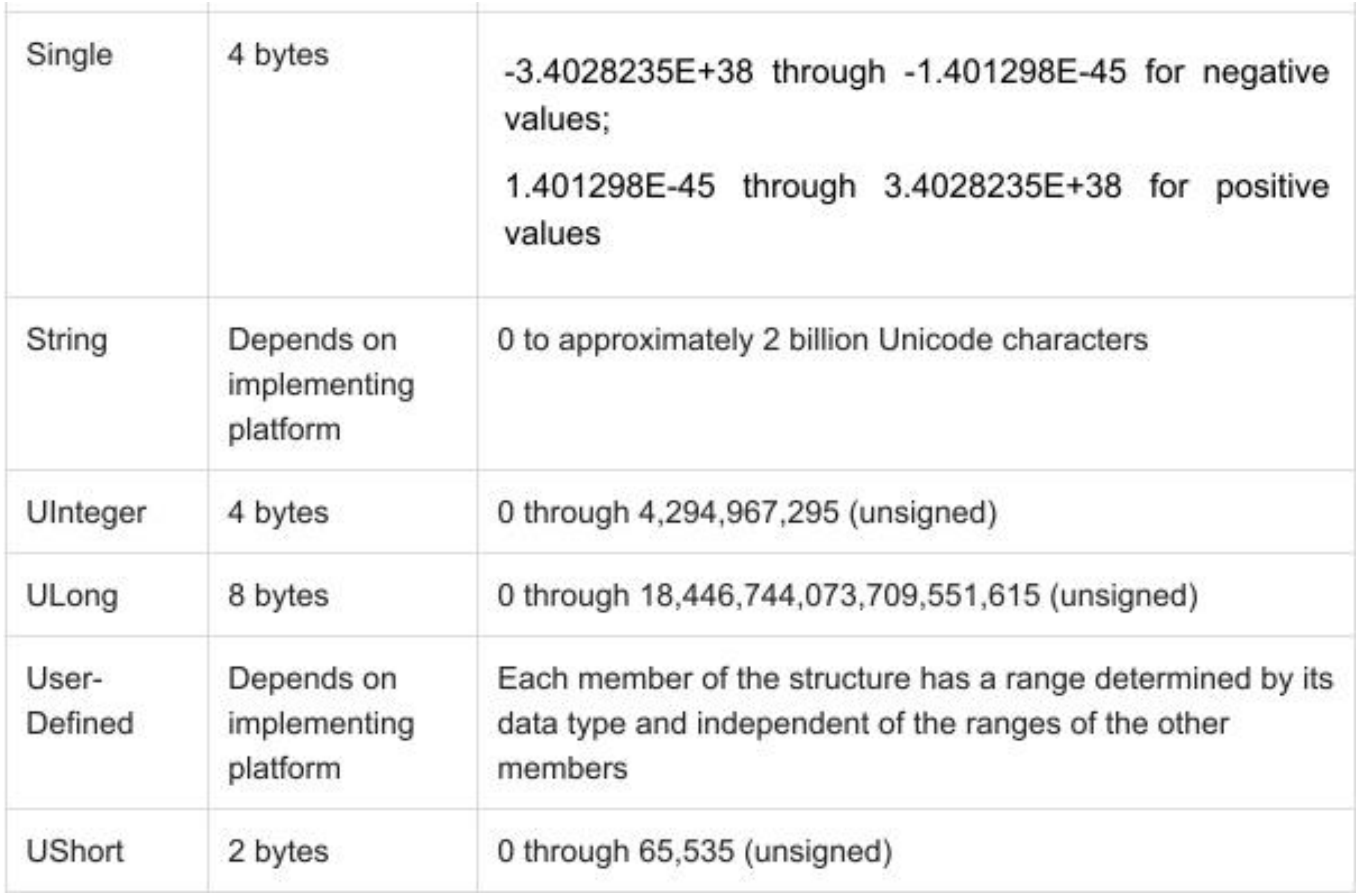
- Data types refer to an extensive system used for declaring variables or functions of different types.
- The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.
- VB.Net provides a wide range of data types. The following table shows all the data types available −



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)

```
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
```

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

```
U and, Me
declaring on the day of: 12/4/2020 3:50:00 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

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.

Sr.No.	Functions &	Description
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.

```
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
**Today's Date
1
1
```

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.
- The basic value types provided in VB.Net can be categorized as −

Type	**Example**
• Integral types	- SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong & 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**.

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.

Examples

```
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"
```

```
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
```

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
```

Example

```
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
```

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
```

VB.Net - Constants and Enumerations

- The **constants** refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.
- Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well.
- The constants are treated just like regular variables except that their values cannot be modified after their definition.
- An **enumeration** is a set of named integer constants.

Declaring Constants

- In VB.Net, constants are declared using the **Const** statement. The Const statement is used at module, class, structure, procedure, or block level for use in place of literal values.
- The syntax for the Const statement is −

  [< attributelist >] [accessmodifier] [Shadows]

  Const constantlist

Where,

- ***attributelist*** – specifies the list of attributes applied to the constants; you can provide multiple attributes separated by commas. Optional.
- ***accessmodifier*** – specifies which code can access these constants. Optional. Values can be either of the: Public, Protected, Friend, Protected Friend, or Private.
- ***Shadows*** – this makes the constant hide a programming element of identical name in a base class. Optional.
- ***Constantlist*** – gives the list of names of constants declared. Required.

Where, each constant name has the following syntax and parts −

constantname [As datatype] = initializer

- ***constantname*** − specifies the name of the constant
- ***datatype*** − specifies the data type of the constant
- ***initializer*** − specifies the value assigned to the constant

'The following statements declare constants.'

Const maxval As Long = 4999

Public Const message As String = "HELLO"

Private Const piValue As Double = 3.1415

Example

```
Module constantsNenum
   Sub Main()
      Const PI = 3.14149
      Dim radius, area As Single
      radius = 7
      area = PI * radius * radius
      Console.WriteLine("Area = " & Str(area))
      Console.ReadKey()
   End Sub
End Module
```

Declaring Enumerations

- An enumerated type is declared using the **Enum** statement. The Enum statement declares an enumeration and defines the values of its members. The Enum statement can be used at the module, class, structure, procedure, or block level.

  The syntax for the Enum statement is as follows −

  ```
  [ < attributelist > ] [ accessmodifier ]  [ Shadows ]
  Enum enumerationname [ As datatype ]
     memberlist
  End Enum
  ```

Where,

- ***attributelist*** – refers to the list of attributes applied to the variable. Optional.
- ***asscessmodifier*** – specifies which code can access these enumerations. Optional. Values can be either of the: Public, Protected, Friend or Private.
- ***Shadows*** – this makes the enumeration hide a programming element of identical name in a base class. Optional.
- ***enumerationname*** – name of the enumeration. Required
- ***datatype*** – specifies the data type of the enumeration and all its members.
- ***memberlist*** – specifies the list of member constants being declared in this statement. Required.

- Each member in the memberlist has the following syntax and parts:
  - [< attribute list >] member name [= initializer]

Where,

- ***name*** – specifies the name of the member. Required.
- ***initializer*** – value assigned to the enumeration member. Optional.

```
Enum Colors
  red = 1
  orange = 2
  yellow = 3
  green = 4
  azure = 5
  blue = 6
  violet = 7
End Enum
```

Example

```
Module constantsNenum
   Enum Colors
      red = 1
      orange = 2
      yellow = 3
      green = 4
      azure = 5
      blue = 6
      violet = 7
   End Enum
```

```
Sub Main()
      Console.WriteLine("The Color Red is : " & Colors.red)
      Console.WriteLine("The Color Yellow is : " & Colors.yellow)
      Console.WriteLine("The Color Blue is : " & Colors.blue)
      Console.WriteLine("The Color Green is : " & Colors.green)
      Console.ReadKey()
   End Sub
End Module
```