UNIT-1

.NET Framework

.NET is a framework to develop software applications. It is designed and developed by Microsoft and the first beta version released in 2000.

It is used to develop applications for web, Windows, phone. Moreover, it provides a broad range of functionalities and support.

This framework contains a large number of class libraries known as Framework Class Library (FCL). The software programs written in .NET are executed in the execution environment, which is called CLR (Common Language Runtime). These are the core and essential parts of the .NET framework.

This framework provides various services like Process management, networking, security, memory management, and type-safety.

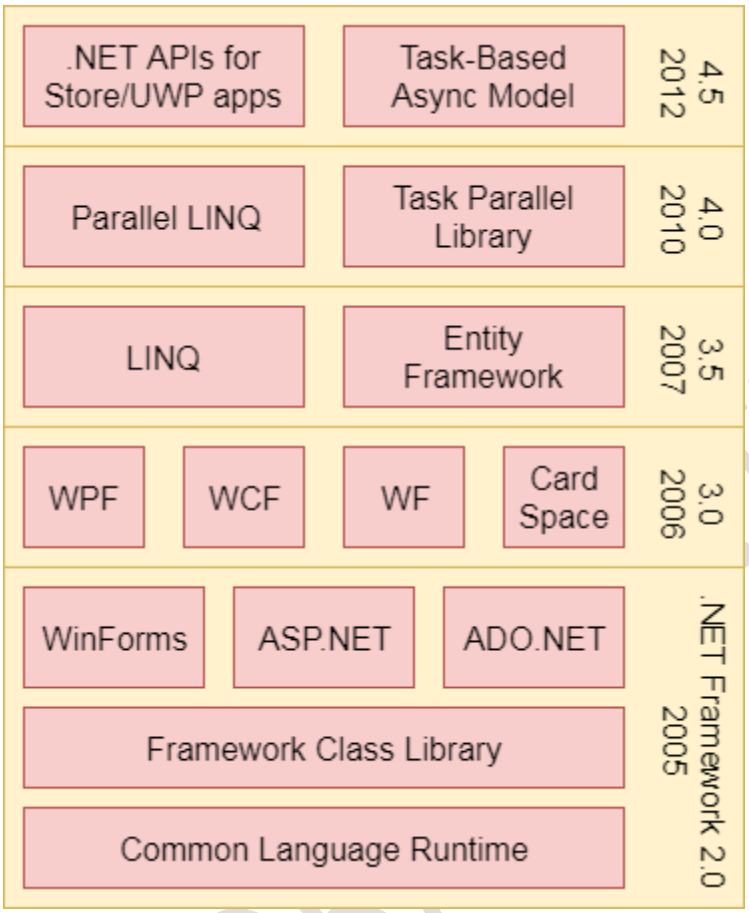
The .Net Framework supports more than 60 programming languages such as C#, F#, VB.NET, J#, VC++, JScript.NET, APL, COBOL, Perl, Oberon, ML, Pascal, Eiffel, Smalltalk, Python, Cobra, ADA, etc.

Following is the .NET framework Stack that shows the modules and components of the Framework.

The .NET Framework is composed of four main components:

1. Common Language Runtime (CLR)
2. Framework Class Library (FCL),
3. Core Languages (WinForms, ASP.NET, and ADO.NET), and
4. Other Modules (WCF, WPF, WF, Card Space, LINQ, Entity Framework, Parallel LINQ, Task Parallel Library, etc.)

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



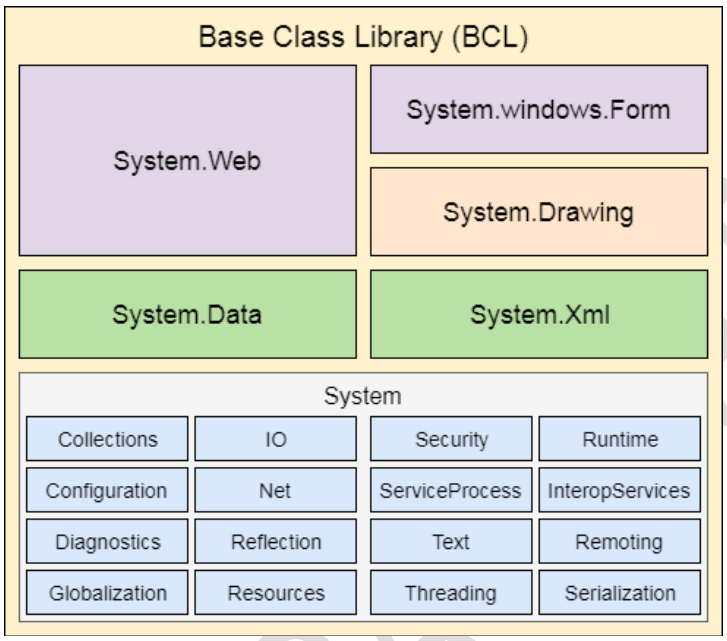
CLR (Common Language Runtime)

It is a program execution engine that loads and executes the program. It converts the program into native / byte code. It acts as an interface between the framework and operating system. It does exception handling, memory management, and garbage collection. Moreover, it provides security, type-safety, interoperability, and portablility. A list of CLR components are given below:

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

FCL (Framework Class Library)

It is a standard library that is a collection of thousands of classes and used to build an application. The BCL (Base Class Library) is the core of the FCL and provides basic functionalities.



WinForms

Windows Forms is a smart client technology for the .NET Framework, a set of managed libraries that simplify common application tasks such as reading and writing to the file system.

ASP.NET

ASP.NET is a web framework designed and developed by Microsoft. It is used to develop websites, web applications, and web services. It provides a fantastic integration of HTML, CSS, and JavaScript. It was first released in January 2002.

ADO.NET

ADO.NET is a module of .Net Framework, which is used to establish a connection between application and data sources. Data sources can be such as SQL Server and XML. ADO .NET consists of classes that can be used to connect, retrieve, insert, and delete data.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

WPF (Windows Presentation Foundation)

Windows Presentation Foundation (WPF) is a graphical subsystem by Microsoft for rendering user interfaces in Windows-based applications. WPF, previously known as "Avalon", was initially released as part of .NET Framework 3.0 in 2006. WPF uses DirectX.

WCF (Windows Communication Foundation)

It is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another.

WF (Workflow Foundation)

Windows Workflow Foundation (WF) is a Microsoft technology that provides an API, an in-process workflow engine, and a rehostable designer to implement long-running processes as workflows within .NET applications.

LINQ (Language Integrated Query)

It is a query language, introduced in .NET 3.5 framework. It is used to make the query for data sources with C# or Visual Basics programming languages.

Entity Framework

It is an ORM based open source framework which is used to work with a database using .NET objects. It eliminates a lot of developer's effort to handle the database. It is Microsoft's recommended technology to deal with the database.

Parallel LINQ

Parallel LINQ or PLINQ is a parallel implementation of LINQ to objects. It combines the simplicity and readability of LINQ and provides the power of parallel programming.

It can improve and provide fast speed to execute the LINQ query by using all available computer capabilities.

Apart from the above features and libraries, .NET includes other APIs and Model to improve and enhance the .NET framework.

In 2015, Task parallel and Task parallel libraries were added. In .NET 4.5, a task-based asynchronous model was added.

.NET Common Language Runtime (CLR)

.NET CLR is a run-time environment that manages and executes the code written in any .NET programming language.

It converts code into native code which further can be executed by the CPU.

.NET CLR Functions

Following are the functions of the CLR.

- It converts the program into native code.
- Handles Exceptions
- Provides type-safety
- Memory management
- Provides security
- Improved performance
- Language independent
- Platform independent
- Garbage collection
- Provides language features such as inheritance, interfaces, and overloading for object-oriented programmings.

.NET CLR Versions

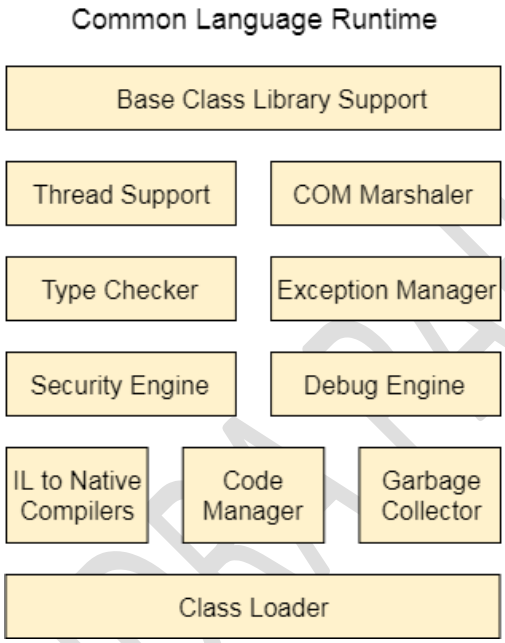
The CLR updates itself time to time to provide better performance.

.NET version	CLR version
1.0	1.0
1.1	1.1
2.0	2.0
3.0	2.0
3.5	2.0
4	4
4.5	4
4.6	4

4.8	4

.NET CLR Structure

Following is the component structure of Common Language Runtime.



Base Class Library Support

It is a class library that provides support of classes to the .NET application.

Thread Support

It manages the parallel execution of the multi-threaded application.

COM Marshaler

It provides communication between the COM objects and the application.

Type Checker

It checks types used in the application and verifies that they match to the standards provided by the CLR.

Code Manager

It manages code at execution run-time.

Garbage Collector

It releases the unused memory and allocates it to a new application.

Exception Handler

It handles the exception at runtime to avoid application failure.

ClassLoader

It is used to load all classes at run time.

.NET Framework Class Library

.NET Framework Class Library is the collection of classes, namespaces, interfaces and value types that are used for .NET applications.

It contains thousands of classes that supports the following functions.

- Base and user-defined data types
- Support for exceptions handling
- input/output and stream operations
- Communications with the underlying system
- Access to data
- Ability to create Windows-based GUI applications
- Ability to create web-client and server applications
- Support for creating web services

.NET Framework Class Library Namespaces

Following are the commonly used namespaces that contains useful classes and interfaces and defined in Framework Class Library.

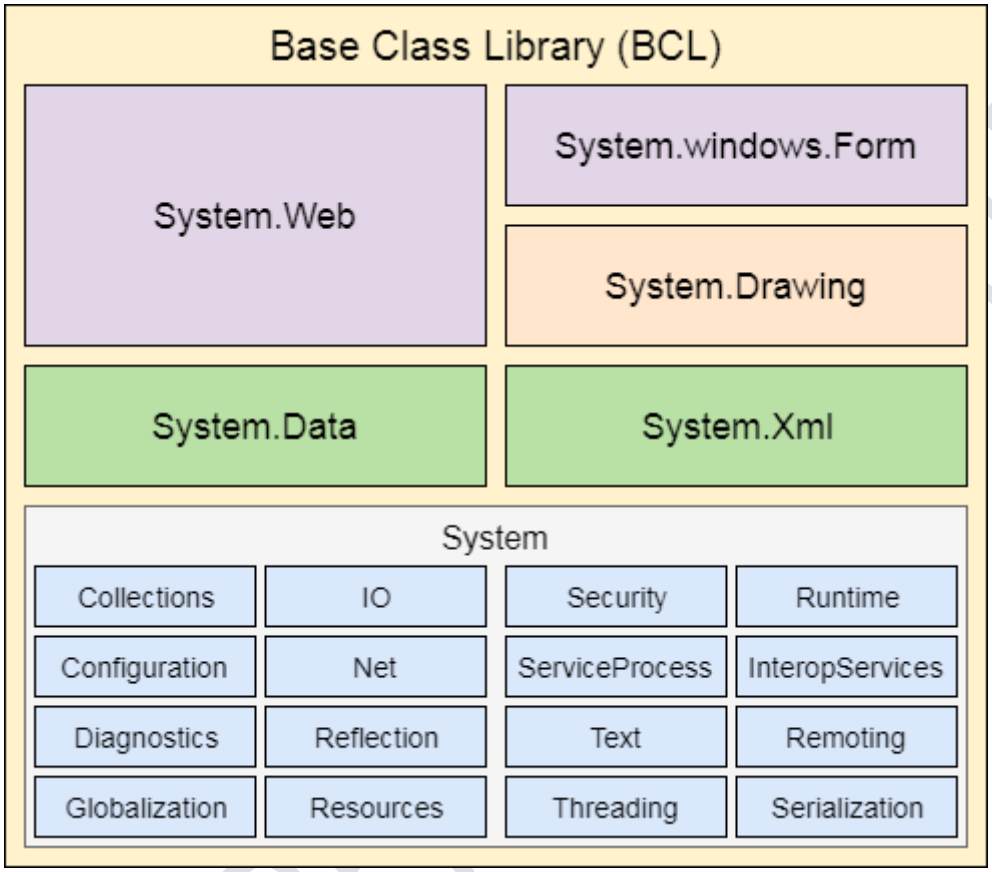
Namespaces	Description
System	It includes all common datatypes, string values, arrays and methods for data conversion.
System.Data, System.Data.Common, System.Data.OleDb, System.Data.SqlClient, System.Data.SqlTypes	These are used to access a database, perform commands on a database and retrieve database.
System.IO, System.DirectoryServices, System.IO.IsolatedStorage	These are used to access, read and write files.
System.Diagnostics	It is used to debug and trace the execution of an application.
System.Net, System.Net.Sockets	These are used to communicate over the Internet when creating peer-to-peer applications.
System.Windows.Forms, System.Windows.Forms.Design	These namespaces are used to create Windows-based applications using Windows user interface components.
System.Web, System.WebCaching, System.Web.UI, System.Web.UI.Design, System.Web.UI.WebControls, System.Web.UI.HtmlControls, System.Web.Configuration, System.Web.Hosting, System.Web.Mail, System.Web.SessionState	These are used to create ASP. NET Web applications that run over the web.
System.Web.Services,	These are used to create XML Web

System.Web.Services.Description, System.Web.Services.Configuration, System.Web.Services.Discovery, System.Web.Services.Protocols	services and components that can be published over the web.
System.Security, System.Security.Permissions, System.Security.Policy, System.WebSecurity, System.Security.Cryptography	These are used for authentication, authorization, and encryption purpose.
System.Xml, System.Xml.Schema, System.Xml.Serialization, System.Xml.XPath, System.Xml.Xsl	These namespaces are used to create and access XML files.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

.NET Framework Base Class Library

.NET Base Class Library is the sub part of the Framework that provides library support to Common Language Runtime to work properly. It includes the System namespace and core types of the .NET framework.

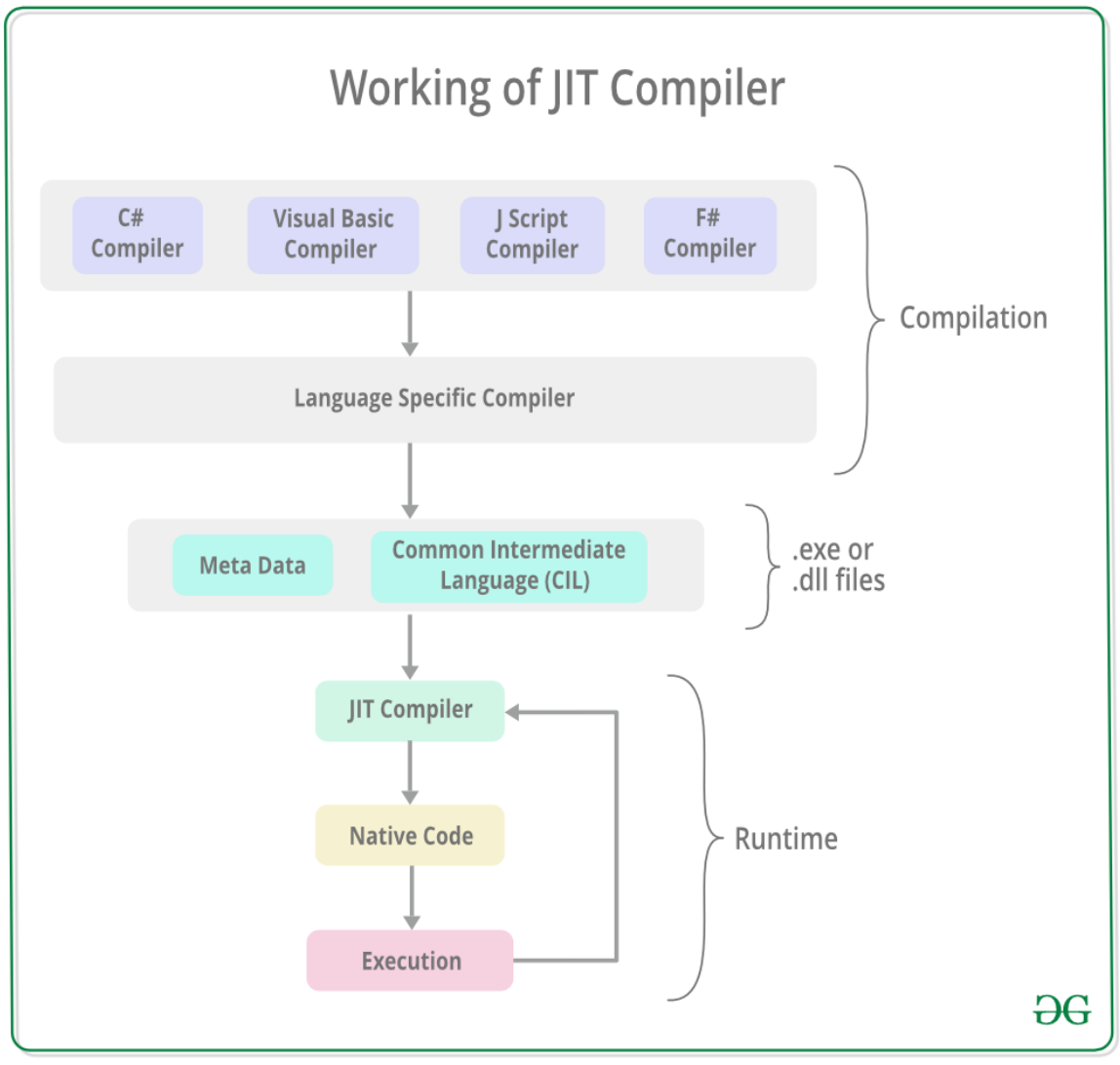


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Just-In-Time (JIT) Compiler in .NET

Just-In-Time compiler(JIT) is a part of **Common Language Runtime (CLR)** in *.NET* which is responsible for managing the execution of *.NET* programs regardless of any *.NET* programming language. A language-specific compiler converts the source code to the intermediate language. This intermediate language is then converted into the machine code by the Just-In-Time (JIT) compiler. This machine code is specific to the computer environment that the JIT compiler runs on.

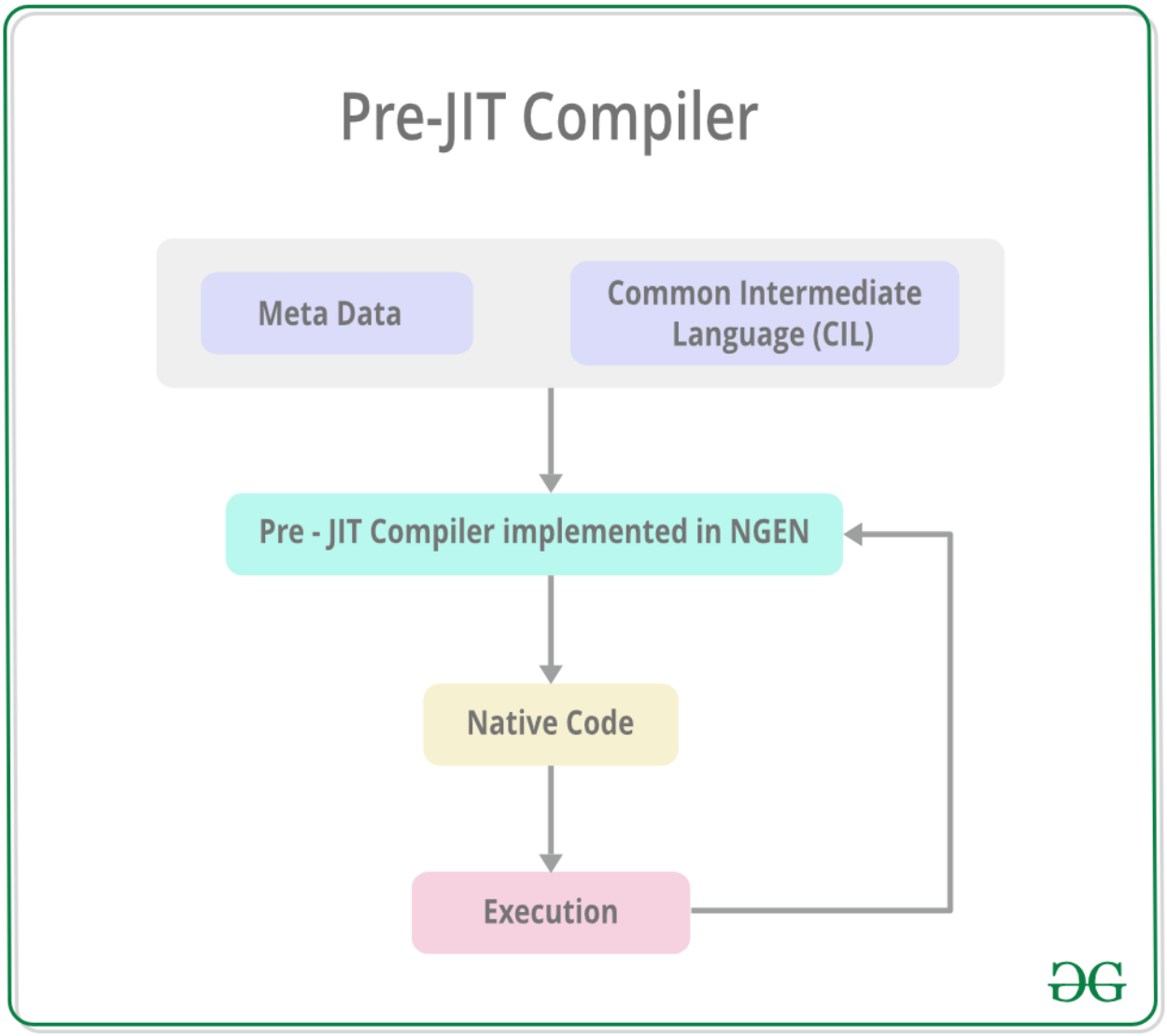
Working of JIT Compiler: The JIT compiler is required to speed up the code execution and provide support for multiple platforms. Its working is given as follows:



The JIT compiler converts the Microsoft Intermediate Language(MSIL) or Common Intermediate Language(CIL) into the machine code. This is done before the MSIL or CIL can be executed. The MSIL is converted into machine code on a requirement basis i.e. the JIT compiler compiles the MSIL or CIL as required rather than the whole of it. The compiled MSIL or CIL is stored so that it is available for subsequent calls if required.

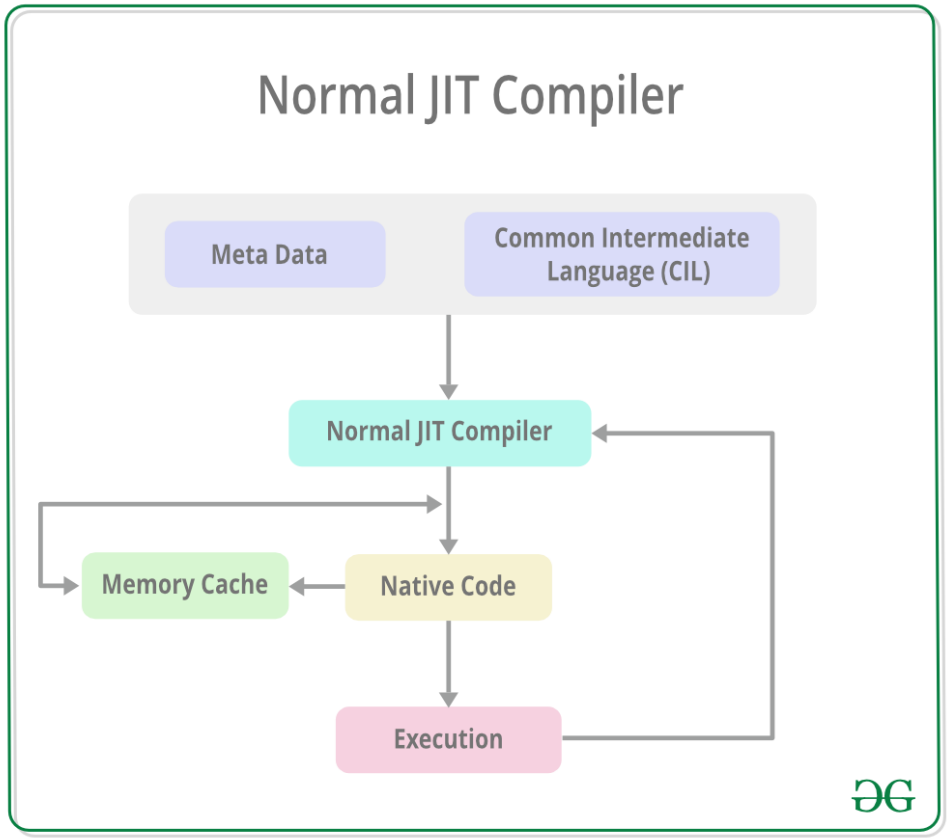
Types of Just-In-Time Compiler: There are **3** types of JIT compilers which are as follows:

- **Pre-JIT Compiler:** All the source code is compiled into the machine code at the same time in a single compilation cycle using the Pre-JIT Compiler. This compilation process is performed at application deployment time. And this compiler is always implemented in the ***Ngen.exe (Native Image Generator)***.



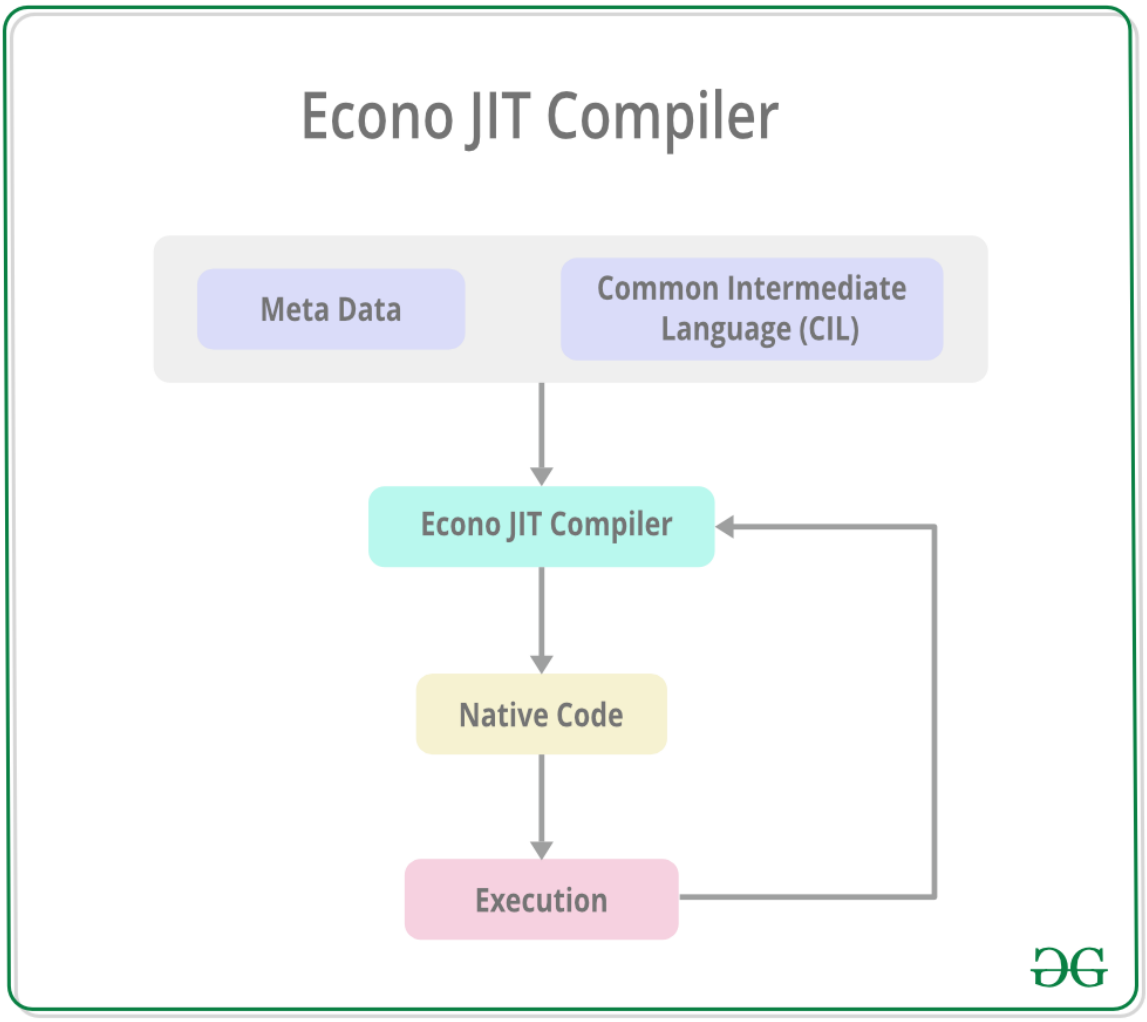
- **Normal JIT Compiler:** The source code methods that are required at run-time are compiled into machine code the first time they are called by the Normal JIT Compiler. After that, they are stored in the cache and used whenever they are called again.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



- **Econo JIT Compiler:** The source code methods that are required at run-time are compiled into machine code by the Econo JIT Compiler. After these methods are not required anymore, they are removed. This JIT compiler is obsolete starting from dotnet 2.0

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



Advantages of JIT Compiler:

- The JIT compiler requires less memory usage as only the methods that are required at run-time are compiled into machine code by the JIT Compiler.
- Page faults are reduced by using the JIT compiler as the methods required together are most probably in the same memory page.
- Code optimization based on statistical analysis can be performed by the JIT compiler while the code is running.

Disadvantages of JIT compiler:

- The JIT compiler requires more startup time while the application is executed initially.
- The cache memory is heavily used by the JIT compiler to store the source code methods that are required at run-time.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Microsoft intermediate language (MSIL)

Microsoft Intermediate Language (MSIL) is the assembly language output by .NET compilers-C#, VB.NET, etc.

.NET framework is shipped with compilers of all programming languages to develop programs. There are separate compilers for the visual basic, C#, and visual programming languages in .NET framework. Each .NET compiler produces an intermediate code after compiling the source code. The intermediate code after compiling the source code. The intermediate code is common for all environments. This intermediate code is known as Microsoft intermediate language (MSIL).

MSIL file is an executable file that can be transferred across various machines. Therefore, it is said that MSIL files are portable to any machine.

Thus the source code written by a programmer or user is compiled and converted into intermediate code. All the code of .NET compliant programming languages are converted into MSIL by their compilers like all managed code compilers for Microsoft .NET generate MSIL.

MSIL is machine independent and can be efficiently compiled into native code. The CLR has a just in time (JIT) compiler which converts the MSIL code into native machine code. Thus, before executing on CPU, MSIL must be translated by a JIT compiler. There is a JIT compiler for each machine architecture supported. The same MSIL will run on any supported machine.

So, .NET programs are compiled twice. During the first compilation process, the source code is converted into MSIL, which is machine-independent. It is stored in the files on the system. During the second compilation process, the MSIL file is executed and CLR transforms it into machine code.

Thus the Microsoft intermediate language (MSIL or simply IL) is the language into which code written with framework gets pre-compiled. The point of MSIL is to provide a CPU- independent instruction set. This way, code can be deployed on a varying number of platforms and efficiently converted to native code for a given platform by the runtime. Along with the MSIL, compilers create the metadata that describes the types, members, and code references for the given compiled code. This metadata helps the runtime do things like enforce security, locate and load class, and generally describe the code's interaction with the runtime

Role of MSIL in a .NET environment

1. Platform Independence:-

Platform independence means that the same file containing byte code instruction can be placed on any platform; at runtime, the final stage of compilation can then be easily accomplished so that the code can run on that particular platform. In other words, MSIL defines a set of portable instructions that are independent of any specific CPU.

2. Performance Improvement:-

Instead of compiling the entire application at once, the JIT compiler simply compiles each portion of code as it is called just in time. When code has been compiled once, the resultant native executable is stored until the application exists so that it does not need to be recompiled the next time that portion of the code is run. This process is more efficient than compiling the entire application code at the start. This shows that the execution of the MSIL code will be almost as fast as executing native machine code.

3. Language Interoperability:-

The use of MSIL facilitates language interoperability. One can compile to MSIL from one language, and this compiled code should then be interoperable with code that has been compiled to MSIL from another language. In other words, MSIL architecture enables the framework to be language neutral. To a large degree, language choice is no longer dictated by the preference of the developer or the team. One can even mix languages in a single application. A

class and an exception are thrown in a C#method that can be caught in a VB method.

4. Reducing maintenance headaches:-

MSIL code can be analyzed by the CLR to determine compliance with requirements such as type safety. Things like buffer overflows and unsafe casts can be caught at compile-time, greatly reducing maintenance headaches.

Garbage Collection

Garbage collection, in the context of .NET, is the process by which the common language runtime (CLR) of .NET framework manages the memory by allocating and releasing memory automatically.

Garbage collector of .NET tries to compact the memory in such a way as to increase the working space required for heap. The class GC of .NET class library controls the garbage collector. The core task of performing a collection is executed by the GC's optimizing engine that determines the best time to perform collection based on allocations being made. GC runs are non-deterministic since the call to GC cannot be guaranteed. However, explicit calls to the overloaded 'Collect()' method of the class GC can be used whenever needed.

The advantages that GC provides include:

- the elimination of memory de-allocation code in applications
- optimized memory usage in managed heap
- clearing up of memory of reclaimed objects that are no longer in use (which helps to initialize managed objects allocated in future and provision of memory safety of objects to avoid an object using content of another.)

Unit-2

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.

In this section, we will understand about the **VB.NET identifier, VB.NET comment,** and how we can use it in **VB.NET**.

There are following reserved keyword available in the VB.NET language.

AddHandler	AddressOf	Alias	And	AndAlso	As
Boolean	ByRef	Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar	CDate	CDbl
CDec	Char	Cint	Class	CLng	CObj
Const	Continue	CSByte	CShort	CSng	CStr
CType	CUnit	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	WhiteOnly

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Xor	#Else	IsReference	TimeOfDay	Append	Auto

Let's create a program to find the area and perimeter of a rectangle in VB.NET.

Rectangle.vb

```
1. Module Rectangle
2.     Public Length As Integer
3.
4.     Public Breadth As Integer
5.     Public Sub Dimension()
6.         Length = 5
7.         Breadth = 6
8.     End Sub
9.     Public Function Area() As Integer
10.        Area = Length * Breadth
11.    End Function
12.    Public Function Pera() As Integer
13.        Pera = 2 * (Length + Breadth)
14.    End Function
15.    Public Sub Display()
16.        Console.WriteLine(" Length is: {0}", Length)
17.        Console.WriteLine("Breadth is: {0}", Breadth)
18.        Console.WriteLine(" Area of Rectangle is: {0}", Area())
19.        Console.WriteLine(" Perimeter of Rectangle is: {0}", Pera())
20.    End Sub
21.    Sub Main()
22.        Dimension() ' directly call the function in main method
23.        Area()
24.        Pera()  ' directly call the function in main method
25.        Display()
26.        Console.WriteLine("Press any key to exit...")
27.        Console.ReadKey()
28.    End Sub
29. End Module
```

Output:

```
Length is: 5
Breadth is: 6
Area of Rectangle is: 30
Perimeter of Rectangle is: 22
Press any key to exit...
```

VB.NET Identifiers

As the name defines, an identifier is used to identify the name of variable, function, class, or any other user-defined elements in the program. An identifier should be the combination of letter, digit, and underscore. Still, the first character of the identifier or variable name should start with alphabet letter or underscore (_) of any length.

There are various rules for identifier in VB.NET, as follows:

1. The first character of an identifier must start with an alphabet or underscore, that could be followed by any sequence of digits (0-9), letter or underscore.
2. An identifier should not contain any reserved keyword.
3. It should not start with any digit.
4. It should not more than 51 characters.
5. An identifier can contain two underscores, but should not be consecutive.
6. It should not include any commas or white spaces in-between characters.

Generally, identifiers are meaningful names. Some valid identifiers are:

Value, a, rec1, my_data, Marks, num, etc.

Some invalid identifiers are:

5be : First character should be alphabets or underscore (_)

Class, Shared : Keyword are not allowed as identifier name.

A# - : Identifier does not contain any special symbol.

Avg marks : It should not contain any blank space.

Program.vb

```
1. Imports System
2. Module Program ' Identifier' name should be valid
3.
4.     Public Sub myfunc()  ' function name
```

```
5.          Console.WriteLine("Hello friends..")
6.      End Sub
7.
8.      Sub Main()
9.          myfunc()
10.         Console.WriteLine("Nice to meet you...")
11.         Console.WriteLine("press any key to exit...")
12.         Console.ReadKey()
13.     End Sub
14. End Module
```

Let's compile and execute the above program.

Output:

```
Hello friends...
Nice to meet you...
press any key to exit...
```

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.

```
1. Sub main()
2. 'Here Console.WriteLine() is used to print a statement.
3. Console.WriteLine(" Welcome to Sai College")
4. 'Above statement displays Welcome to Sai College
5. End Sub
```

Circle.vb

```
1. Imports System
2. Public Class Circle
3.   'define the variable
4.     Dim radius As Integer = 10
5.     Public Function SetCircle() As Double
```

```
6.              SetCircle = 2 * 3.14 * radius  ' Function is used to return some value
7.          End Function
8.          Public Sub display()        ' create display() sub function to print the message
9.              Console.Write(" Radius is: {0}", radius)
10.         Console.WriteLine(" Circumference of Circle: {0}", SetCircle())
11.     End Sub
12.     'Shared keyword can be used without creating an object
13.     Shared Sub Main()
14.         Dim obj As New Circle()
15.         obj.SetCircle() 'object reference
16.         obj.display()
17.         Console.ReadKey()
18.     End Sub
19. End Class
```

Now compile and execute the above program.

Output:

```
Radius is: 10 Circumference of Circle: 62.8
press any key to exit...
```

Hence, it is the best way to explain every step of VB.NET programming language.

VB.NET Data Type

In **VB.NET, data type** is used to define the type of a variable or function in a program. Furthermore, the conversion of one data type to another type using the data conversion function.

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.

Syntax:

```
1. 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.

Different Data Types and their allocating spaces in VB.NET

The following table shows the various data types list in the VB.NET programming language.

Data Types	Required Space	Value Range
Boolean	A Boolean type depends on the implementing platform	True or False
Byte	1 byte	Byte Range start from 0 to 255 (unsigned)
Char	2 bytes	Char Range start from 0 to 65535 (unsigned)
Date	8 bytes	Date range can be 0:00:0 (midnight) January 1, 0001 to 11:5959 PM of December 31, 9999.
Decimal	16 bytes	Range from 0 to +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) without any decimal point; And 0 to +/-7.9228162514226433759354395033 5 with 28 position to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 to -4.94-65645841246544E-324 for negative values; 4.94065645841246544E-324 to 1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 to 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (9.2...E + 18) (signed)
Object	Object size based on the platform such as 4 bytes in 32-bit and 8 bytes in 64-bit platform	It can store any type of data defined in a variable of type Object
SByte	1 byte	-128 to 127 (signed)
Short	2 bytes	-32,768 to 32,767 (signed)
Single	4 bytes	-3.4028235E + 38 to -1.401298E-45 for negative values; And for positive value: 1.401298E-45 to 3.4028235E + 38.
String	String Datatype depend on the implementing platform	It accepts Unicode character from 0 to approximately 2 billion characters.
UInteger	4 bytes	The range start from 0 to 4,294,967,295 (unsigned)
ULong	8 bytes	The range of ULong start from 0 to 18,446,744,073,709,551,615 (1.8...E + 19)

		(unsigned)
User-Defined (structure)	A user-defined data type depends on the implementing platform	Each member of the structure has its own data type and limits independent of the other members' ranges.
UShort	2 bytes	Range from 0 to 65,535 (unsigned)

Let's use the various data types in a VB.NET program.

Data_type.vb

```
1.  Module Data_type
2.      Sub Main()
3.  ' defining the Data Type to the variables
4.          Dim b As Byte = 1
5.          Dim num As Integer = 5
6.          Dim si As Single
7.          Dim db As Double
8.          Dim get_date As Date
9.          Dim c As Char
10.         Dim str As String
11.
12.         b = 1
13.         num = 20
14.         si = 0.12
15.         db = 2131.787
16.         get_date = Today
17.         c = "A"
18.         str = "Hello Friends..."
19.
20.         Console.WriteLine("Welcome to the Sai College")
21.         Console.WriteLine("Byte is: {0}", b)
22.         Console.WriteLine("Integer number is: {0}", num)
23.         Console.WriteLine("Single data type is: {0}", si)
24.         Console.WriteLine("Double data type is: {0}", db)
25.         Console.WriteLine("Today is: {0}", get_date)
26.         Console.WriteLine("Character is: {0}", b)
27.         Console.WriteLine("String message is: {0}", str)
28.         Console.ReadKey()
29.     End Sub
```

30. End Module

Output:

```
Welcome to the Sai College
Byte is: 1
Integer number is: 20
Single data type is: 0.12
Double data type is: 2131.787
Today is: 31-05-2020 00:00:00
Character is: 1
String message is: Hello Friends...
```

Type Conversion Functions in VB.NET

The following functions are available for conversion.

1. **CBool(expression):** It is used to convert an expression into a Boolean data type.
2. **CByte(expression):** It is used to convert an expression to a Byte data type.
3. **CChar(expression):** It is used to convert an expression to a Char data type.
4. **CDate(expression):** It is used to convert an expression to a Date data type.
5. **CDbl(expression):** It is used to convert an expression into a Double data type.
6. **CDec(expression):** It is used to convert an expression into a Decimal data type.
7. **CInt(expression):** It is used to convert an expression to an Integer data type.
8. **CLng(expression):** It is used to convert an expression to a Long data type.
9. **CObj(expression):** It is used to convert an expression to an Object data type.
10. **CSByte(expression):** It is used to convert an expression to an SByte data type.
11. **CShort(expression):** It is used to convert an expression to a Short data type.
12. **CSng(expression):** It is used to convert an expression into a Single data type.
13. **CStr(expression):** It is used to convert an expression into a String data type.
14. **CUInt(expression):** It is used to convert an expression to a UInt data type.
15. **CULng(expression**): It is used to convert an expression to a ULng data type.
16. **CUShort(expression)**: It is used to convert an expression into a UShort data type.

In the following, program we have performed different conversion.

DB_Conversion.vb

1. Option Strict On
2. Module DB_Conversion

```
3.      Sub Main()
4.      'defining the Data type conversion
5.          Dim dblData As Double
6.          dblData = 5.78
7.          Dim A, B As Char
8.          Dim bool As Boolean = True
9.          Dim x, Z, B_int As Integer
10.         A = "A"
11.         B = "B"
12.         B_int = AscW(B)
13.
14.         Console.WriteLine(" Ascii value of B is {0}", B_int)
15.
16.         x = 1
17.         Z = AscW(A)
18.         Z = Z + x
19.         Console.WriteLine("String to integer {0}", Z)
20.         Console.WriteLine("Boolean value is : {0}", CStr(bool))
21.         Dim num, intData As Integer
22.
23.         num = CInt(dblData)
24.         intData = CType(dblData, Integer)
25.         Console.WriteLine(" Explicit conversion of Data type " & Str(intData))
26.         Console.WriteLine(" Value of Double is: {0}", dblData)
27.         Console.WriteLine("Double to Integer: {0}", num)
28.         Console.ReadKey()
29.     End Sub
30. 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
```

Note: For data type conversion, the VB.NET provides Option Strict On that allows us to convert one data type to another. Some data types in VB.NET reject the conversion because of "Option Strict On". Remember that while performing conversion, turn off the Option Strict mode.

VB.NET Variable and Constant

In **VB.NET**, a **variable** is used to hold the value that can be used further in the programming. In this section, we will learn how to declare and initialize a **variable** and a **constant**.

What is a Variable?

A variable is a simple name used to store the value of a specific data type in computer memory. In VB.NET, each variable has a particular data type that determines the size, range, and fixed space in computer memory. With the help of variable, we can perform several operations and manipulate data values in any programming language.

VB.NET Variables Declaration

The declaration of a variable is simple that requires a variable name and data type followed by a Dim. A Dim is used in Class, Module, structure, Sub, procedure.

Syntax:

1. Dim [Variable_Name] As [Defined Data Type]

Name	Descriptions
Dim	It is used to declare and allocate the space for one or more variables in memory.
Variable_Name	It defines the name of the variable to store the values.

As	It is a keyword that allows you to define the data type in the declaration statement.
Data Type	It defines a data type that allows variables to store data types such as Char, String, Integ Decimal, Long, etc.
Value	Assign a value to the variable.

There are some valid declarations of variables along with their data type definition, as shown below:

1. Dim Roll_no As Integer
2. Dim Emp_name As String
3. Dim Salary As Double
4. Dim Emp_id, Stud_id As Integer
5. Dim result_status As Boolean

Further, if we want to **declare more than one variable** in the same line, we must separate each variable with a comma.

Syntax

1. Dim Variable_name1 As DataType1, variable_name2 As DataType2, Variable_name3 As DataType3

Note: The statements given below is also used to declare the variable with their data type:

1. Static name As String
2. Public bill As Decimal = 0

VB.NET Variable Initialization

After the declaration of a variable, we must assign a value to the variable. The following syntax describes the initialization of a variable:

Syntax:

1. Variable_name = value

For example:

1. Dim Roll_no As Integer 'declaration of Roll_no
2. Roll_no = 101 'initialization of Roll_no

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
3.
4.  Initialize the Emp_name
5.  Dim Emp_name As String
6.  Emp_name = "David" 'Here Emp_name variable assigned a value of David
7.
8.  Initialize a Boolean variable
9.  Dim status As Boolean 'Boolean value can be True or False.
10. status = True 'Initialize status value to True
```

We can also initialize a variable at the time of declaration:

```
1.  Dim Roll_no As Integer = 101
2.  Dim Emp_name As String = " Stephen Robert "
```

Let's create a program to use different types of variable declaration and initialization in VB.NET.

Variable1.vb

```
1.  Imports System
2.  Module Variable1
3.      Sub Main()
4.          'declaration of intData as Integer
5.          Dim intData As Integer
6.          'declaration of charData as Char
7.          Dim CharData As Char
8.          'declaration of strData as String
9.          Dim strData As String
10.         'declaration of dblData as Double
11.         Dim dblData As Double
12.         'declaration of single_data as Single
13.         Dim single_data As Single
14.         'Initialization of intData
15.         intData = 10
16.         'Initialization of CharData
17.         CharData = "A"
18.         'Initialization of strData
19.         strData = " VB.NET is a Programming Language."
20.         dblData = 4567.676
```

```
21.         'Initialization of dblData
22.         'Initialization of single_data
23.          single_data = 23.08
24.
25.         Console.WriteLine(" Value of intData is: {0}", intData)
26.         Console.WriteLine(" Value of CharData is: {0}", CharData)
27.         Console.WriteLine(" Value of strData is: {0}", strData)
28.         Console.WriteLine(" Value of dblData is: {0}", dblData)
29.         Console.WriteLine(" Value of single_data is: {0}", single_data)
30.
31.         Console.WriteLine("press any key to exit...")
32.         Console.ReadKey()
33.     End Sub
34.
35. End Module
```

Output:

```
Value of intData is: 10
 Value of CharData is: A
 Value of strData is:  VB.NET is a Programming Language.
 Value of dblData is: 4567.676
 Value of single_data is: 23.08
press any key to exit...
```

Getting Values from the User in VB.NET

In VB.NET, the Console class provides the Readline() function in the System namespace. It is used to take input from the user and assign a value to a variable. For example:

```
1. Dim name As String
2. name = Console.ReadLine()
3. Or name = Console.ReadLine
```

Let's create a program that takes input from the user.

User_Data.vb

```
1. Imports System
2. Module User_Data
3.     Sub Main()
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
4.      Dim num As Integer
5.      Dim age As Double
6.      Dim name As String
7.      Console.WriteLine("Enter your favourite number")
8.      ' Console.ReadLine or Console.ReadLine() takes value from the user
9.      num = Console.ReadLine
10.     Console.WriteLine(" Enter Your Good name")
11.     'Read string data from the user
12.     name = Console.ReadLine
13.     Console.WriteLine(" Enter your Age")
14.     age = Console.ReadLine
15.     Console.WriteLine(" You have entered {0}", num)
16.     Console.WriteLine(" You have entered {0}", name)
17.     Console.WriteLine(" You have entered {0}", age)
18.     Console.ReadKey()
19.
20.  End Sub
21. End Module
```

Output:

```
Enter your favourite number
7
 Enter Your Good name
Alexander
 Enter your Age
27.5
 You have entered 7
 You have entered Alexander
 You have entered 27.5
```

Note: Console.Read and Console.ReadKey() function is used to read a single character from the user.

Lvalues and Rvalues in VB.NET

There are two ways to express the expression value:

Lvalue: It is an lvalue expression that refers to a memory location for storing the address of a variable. An lvalue is a variable that can appear to the left or right of the assignment operator to hold values. Furthermore, in comparison to or swapping the variables' values, we can also define the variable on both sides (left or right-side) of the assignment operator.

Example:

```
1. Dim num As Integer
2. Num = 5
3. Or
4. Dim num As Integer = 5
```

But when we write the following statement, it generates a compile-time error because it is not a valid statement.

```
1. Dim x As Integer
2. 10 = x
```

Rvalue: It is an rvalue expression that is used to store a value in some address of memory. An rvalue can appear only on the right- hand side because it is a value of the variable that defines on the right-hand side.

```
1. Dim name As String
2. Name = "Peter" // rvalue define at right side of the assignment operator.
```

VB.NET Constants

As the name suggests, the name constant refers to a fixed value that cannot be changed during the execution of a program. It is also known as **literals**. These constants can be of any data type, such as Integer, Double, String, Decimal, Single, character, enum, etc.

Declaration of Constants

In VB.NET, **const** is a keyword that is used to declare a variable as constant. The Const statement can be used with module, structure, procedure, form, and class.

Syntax:

```
1. Const constname As datatype = value
```

Item Name	Descriptions
Const	It is a Const keyword to declare a variable as constant.
Constname	It defines the name of the constant variable to store the values.

As	It is a keyword that allows you to define the data type in the declaration statement.
Data Type	It defines a data type that allows variables to store data types such as Char, String, Inte Decimal, Long, etc.
Value	Assign a value to the variable as constant.

Further, if we want to **declare more than one variable** in the same line, we must separate each variable with a comma, as shown below. The Syntax for defining the multiple variables as constant is:

```
Dim Variable_name1 As DataType1, variable_name2 As DataType2, Variable_name3 As DataT
ype3
```

Note: The statements given below are also used to declare the variable with their data type:

```
Const num As Integer = 10
Static name As String
Public Const name As String = "Sai College"
Private Const PI As Double = 3.14
```

Example of Const keyword

Const1.vb

```
Module Const1
    Sub main()
        'declaration and initialization of Constant variable using Const keywords
        Const intData As Integer = 20
        Const name As String = "Sai College"
        Const topic As String = "VB.NET"
        Const PI = 3.14
        Dim radius, area As Integer

        Console.WriteLine(" Constant integer is {0}", intData)
        Console.WriteLine(" You have entered {0}", name)
        Console.WriteLine(" Your Topic is {0}", topic)
        Console.WriteLine("Enter the Radius")
        radius = Console.ReadLine()
```

```
15.         area = PI * radius * radius
16.         Console.WriteLine(" Area of Circle is {0}", area)
17.         Console.ReadKey()
18.
19.     End Sub
20. End Module
```

Output:

```
Constant integer is 20
 You have entered Sai College
 Your Topic is VB.NET
Enter the Radius
7
 Area of Circle is 154
```

Scope of Variable in VB.NET

The scope of a variable determines the accessible range of a defined variable at the time of declaration in any block, module, and class. You can access it if the variable is in a particular region or scope in the same block. And if the variable goes beyond the region, its scope expires.

The following are the methods to represent the scope of a variable in VB.NET.

1. Procedure Scope
2. Module Scope
3. Public Scope

Procedure (local) scope

A **local variable** is a type of variable defined within a procedure scope, block, or function. It is available with a code inside the procedure, and it can be declared using the **Dim or static** statement. These variables are not accessible from outside of the local method. However, the local variable can be easily accessed by the nested programming function in the same method.

1. Dim X As Integer

Local variables exist until the procedure in which they are declared is executed. Once a procedure is executed, the values of its local variables will be lost, and the resources used by these variables will be released. And when the block is executed again, all the local variables are rearranged.

Let's create a program that displays the local scope of a variable within a function.

Local_Scope.vb

```
1. Imports System
2. Module Local_scope
3.     Sub Main()
4.         Console.WriteLine(" Scope of local varibale within a function")
5.         local() ' call local() and local() function without any object reference
6.         local2()
7.         Console.WriteLine("press any key to exit...")
8.         Console.ReadKey()
9.     End Sub
10.    Sub local()
11.        Dim X As Integer
12.        ' declaration of local variable
13.        X = 50
14.        Console.WriteLine(" Value of Local value X is {0}", X)
15.
16.    End Sub
17.    Sub local2()
18.        Dim X As String
19.        ' scope of local variable within a function
20.        X = "Sai College"
21.        Console.WriteLine(" Value of X is {0}", X)
22.    End Sub
23. End Module
```

Output:

```
Scope of local variable within a function
 Value of Local value X is 50
 Value of X is Sai College
press any key to exit...
```

Module Scope

All existing procedures can easily identify a variable that is declared inside a module sheet is called a **module-level variable**. The defined module variable is **visible to all procedure**s within that module only, but it is not available for other module's procedures. The **Dim or private statement** at the top of the first procedure declaration can be declared the module-level variables. It means that these variables

cannot be declared inside any procedure block. Further, these variables are useful to share information between the procedures in the same module. And one more thing about the module-level variable is that these variables can remains existence as long as the module is executed.

```
' It is the declaration section of the module

Private num As Integer ' A private module-level variable
Dim name As String ' Another private module-level variable
```

Let's create a program that display the module level variable in VB.NET.

Module_scope.vb

```
Imports System
Module Module_scope
'module-level variable declaration
    Dim x As Integer
    Private y As Integer
    Private name As String = "Sai College"
    Sub example()
        x = 10
        y = x + 10
        Console.WriteLine(" Value of Y is {0}", y)
    End Sub
    Sub example2()
        Console.WriteLine(" Value of X is {0}", x)
        Console.WriteLine(" Value of Y is {0}", y)
        Console.WriteLine(" Name is {0}", name)
    End Sub
    Sub example3()
        Dim A As Integer  ' local variable or local scope
        A = x + y
        Console.WriteLine(" Local scope within a function of variable A {0}", A)
    End Sub
    Sub Main()
        Console.WriteLine(" Module scope of variable")
        example()
        example2()
```

```
26.         example3()
27.         Console.WriteLine("Press any key to exit...")
28.         Console.ReadKey()
29.     End Sub
30. End Module
```

Output:

```
Module scope of variable
 Value of Y is 20
 Value of X is 10
 Value of Y is 20
 Name is Sai College
 Local scope within a function of variable A 30
Press any key to exit...
```

Global (Public) Scope

As the name defines, a global variable is a variable that is used to access the variables **globally in a program**. It means these variables can be accessed by all the **procedures or modules** available in a program. To access the variables globally in a program, you need to use **the friend or public keyword** with a variable in a module or class at the top of the first procedure function. Global scope is also known as the **Namespace scope**.

Let's create a program that uses the global variable.

Global_scope1.vb

```
1.  Imports System
2.  Module Global_scope1
3.      'Global declaration of a variable
4.      Public str As String = "Hello, Programmer."
5.      Public topic As String
6.      Public exp As Integer
7.
8.      Sub Main()
9.          Console.WriteLine(" You have passed {0}", str)
10.         Console.WriteLine(" Enter the topic name")
11.         topic = Console.ReadLine
12.         Console.WriteLine(" Topic Name :{0}", topic)
13.         Console.WriteLine("How many years of experienced in {0}?", topic)
14.         exp = Console.ReadLine
```

```
15.         Console.WriteLine(" Your Experienced is {0} ", exp)
16.         Console.ReadKey()
17.     End Sub
18.
19. End Module
```

Output:

```
You have passed Hello, Programmer
 Enter the topic name
VB.NET
 Topic Name :VB.NET
How many years of experienced in VB.NET?
10
 Your Experienced is 10
```

VB.NET Operators

In **VB.NET** programming, the **Operator** is a symbol that is used to perform various operations on variables. **VB.NET** has different types of Operators that help in performing logical and mathematical operations on data values. The **Operator precedence** is used to determine the execution order of different Operators in the VB.NET programming language.

What is VB.NET Operator?

In VB.NET, **operator** is a special symbol that tells the compiler to perform the specific logical or mathematical operation on the data values. The data value itself (which can be either a variable or a constant) is called an **operand,** and the Operator performs various **operations** on the operand.

For example: In the expression,

```
3 + 2 - 1
```

The symbol **+** and **-** are the Operators, and the 3, 2, and 1 are operands.

Different Types of VB.NET Operators

Following are the different types of Operators available in VB.NET:

- Arithmetic Operators
- Comparison Operators

- Logical and Bitwise Operators
- Bit Shift Operators
- Assignment Operators
- Concatenation Operators
- Miscellaneous Operators

Arithmetic Operators

The Arithmetic Operators in VB.NET, used to perform mathematical operations such as **subtraction, addition, multiplication, division,** etc. on the operands in VB.NET. These are as follows:

Arithmetic Operators in VB.NET

Operators	Description	Example
^	It is an exponentiation Operator that is used to raises one operand to the power of another operand.	Y ^ X (X to the power Y)
+	The addition Operator is used to add numeric data, as well as concatenate two string variables.	X + Y
-	It is a subtraction Operator, which is used to subtract the second operand from the first operand.	X - Y
*****	The multiplication Operator is used to multiply the operands	X * Y
/	It is a division Operator used to divide one operand by another operand and returns a floating-point result.	X / Y
****	It is an integer division Operator, which is similar to division Operator, except that it returns an integer result while dividing one operand to another operand.	X \ Y
Mod	It is a modulo (Modulus) Operator, which is used to divide two operands and returns only a remainder.	X Mod Y

Example of **Arithmetic Operators in VB.NET:**

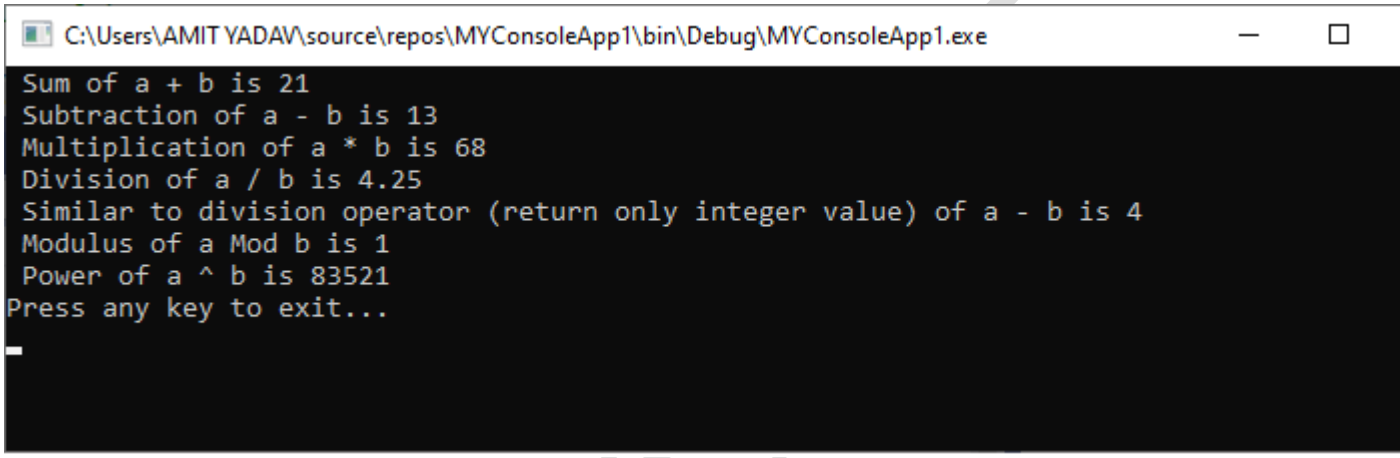
PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Arithmetic_Operator.vb

```
1. Imports System
2. Module Arithmetic_Operator
3.     Sub Main()
4.         'Declare a, b And c as integer Data Type()
5.         Dim a, b, c As Integer
6.         Dim d As Single
7.         a = 17
8.         b = 4
9.         ' Use of + Operator
10.        c = a + b
11.        Console.WriteLine(" Sum of a + b is {0}", c)
12.
13.        'Use of - Operator
14.        c = a - b
15.        Console.WriteLine(" Subtraction of a - b is {0}", c)
16.
17.        'Use of * Operator
18.        c = a * b
19.        Console.WriteLine(" Multiplication of a * b is {0}", c)
20.
21.        'Use of / Operator
22.        d = a / b
23.        Console.WriteLine(" Division of a / b is {0}", d)
24.
25.        'Use of \ Operator
26.        c = a \ b
27.        Console.WriteLine(" Similar to division Operator (return only integer value) of a -
    b is {0}", c)
28.
29.        'Use of Mod Operator
30.        c = a Mod b
31.        Console.WriteLine(" Modulus of a Mod b is {0}", c)
32.
33.        'Use of ^ Operator
34.        c = a ^ b
```

```
35.         Console.WriteLine(" Power of a ^ b is {0}", c)
36.         Console.WriteLine("Press any key to exit...")
37.         Console.ReadKey()
38.     End Sub
39. End Module
```

Now compile and execute the above program, by pressing the F5 button or Start button from the Visual Studio; then it shows the following result:



Comparison Operators

As the name suggests, the Comparison Operator is used to compare the value of two variables or operands for the various condition such as greater, less than or equal, etc. and returns a Boolean value either true or false based on the condition.

Operator	Description	Example
=	It checks whether the value of the two operands is equal; If yes, it returns a true value, otherwise it shows False.	(A = B)
<>	It is a Non-Equality Operator that checks whether the value of the two operands is not equal; it returns true; otherwise, it shows false.	(A <> B), check Non-Equality
>	A greater than symbol or Operator is used to determine whether the value of the left operand is greater than the value of the right operand; If the condition is true, it returns TRUE; otherwise, it shows	(A > B); if yes, TRUE, Else FALSE

	FALSE value.	
<	It is a less than symbol which checks whether the value of the left operand is less than the value of the right operand; If the condition is true, it returns TRUE; otherwise, it shows FALSE value.	(A < B); if the condition is true, returns TRUE else FALSE
>=	It is greater than equal to which checks two conditions whether the first operand is greater than or equal to the second operand; if yes, it returns TRUE; otherwise, it shows False.	A >= B
<=	This symbol represents less than equal to which determines the first operand is less than or equal to the second operand, and if the condition is true, it returns TRUE; otherwise, it shows FALSE.	A <= B
Is	The Is Operator is used to validate whether the two objects reference the same variable or object; If the test is true, it returns True; otherwise, the result is False. In short, it checks the equality of the objects. An Is Operator is also used to determine whether the object refers to a valid object.	result = obj1 Is obj2
IsNot	The IsNot Operator is similar to Is Operator, except that the two object references the different object; if yes, the result is True; otherwise, the result is False.	Result = obj1 IsNot obj2
Like	The Like Operator is used to check the pattern expression of string variable; And if the pattern matched, the result is True; otherwise, it returns False.	result = string Like the pattern, the pattern represents the series of characters used by Like Operator.

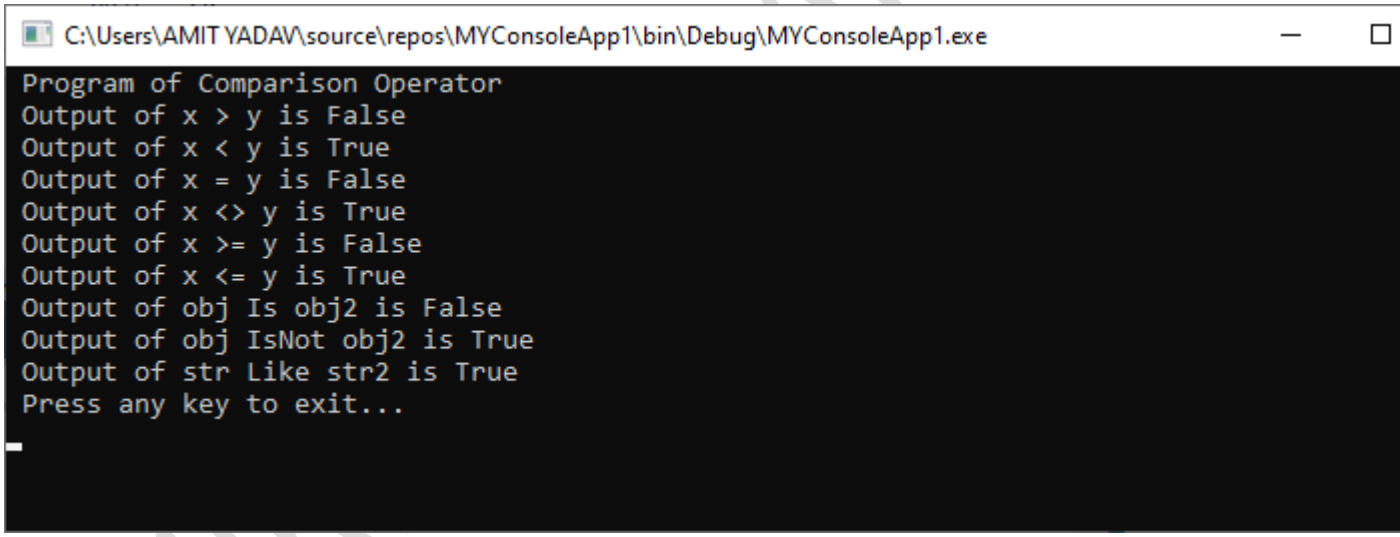
Example of **Comparison Operators in VB.NET**

Comparison_Operator.vb

```
Imports System
Module Comparison_Operator
    Sub Main()
        'declaration of Integer, Object and String Data Type variables
        Dim x As Integer = 5
        Dim y As Integer = 10
        Dim Result, obj, obj2 As Object
        Dim str, str2 As String
        str = "Apple12345"
        str2 = "Apple12345"
        obj = 10
        obj2 = 20

        Console.WriteLine(" Program of Comparison Operator")
        'Use of > Operator
        Console.WriteLine(" Output of x > y is {0}", x > y)

        'Use of < Operator
        Console.WriteLine(" Output of x < y is {0}", x < y)

        'Use of = Operator
        Console.WriteLine(" Output of x = y is {0}", x = y)

        'Use of <> Operator
        Console.WriteLine(" Output of x <> y is {0}", x <> y)

        'Use of >= Operator
        Console.WriteLine(" Output of x >= y is {0}", x >= y)

        'Use of <= Operator
        Console.WriteLine(" Output of x <= y is {0}", x <= y)

        'Use of Is Operator
        Result = obj Is obj2
        Console.WriteLine(" Output of obj Is obj2 is {0}", Result)

        'Use of Is Operator
```

```
        Result = obj IsNot obj2
        Console.WriteLine(" Output of obj IsNot obj2 is {0}", Result)

        'Use of Like Operator
        Result = str Like str2
        Console.WriteLine(" Output of str Like str2 is {0}", Result)

        Console.WriteLine(" Press any key to exit...")
        Console.ReadKey()
    End Sub
End Module
```

Now compile and execute the above code by pressing the F5 button or Start button in Visual studio, it returns the following output:



Logical and Bitwise Operators

The logical and bitwise Operators work with Boolean (true or false) conditions, and if the conditions become true, it returns a Boolean value. The following are the logical and bitwise Operators used to perform the various logical operations such as And, Or, Not, etc. on the operands (variables). Suppose there are two operand A and B, where A is True, and B is False.

Operator	Description	Example
And	The And Operator represents, whether both the operands are true; the result is True.	(A And B), result = False
Or	It is an Or Operator that returns a true value; if anyone operand is true from both the operands.	(A Or B), result = True
Not	The Not Operator is used to reverse the logical condition. For	Not A

	example, if the operand's logic is True, it reveres the condition and makes it False.	Or Not(A And B) is True
Xor	It is an Exclusive OR Operator that represents, whether both the expression is true or false, the result is True; otherwise, the result is False.	A Xor B is True
AndAlso	It is a logical AND Operator that performs short-circuit operation on the variables, and if both the operands are true, the result is True else the result is False.	A AndAlso B = False
OrElse	It is a logical OR Operator that perform short-circuit operation on Boolean data. If anyone of the operand is true, the result is True else the result is False.	A OrElse B = True
IsFalse	The IsFalse Operator is used to determine whether an expression is False.	
IsTrue	The IsTrue Operator is used to determine whether an expression is True.	

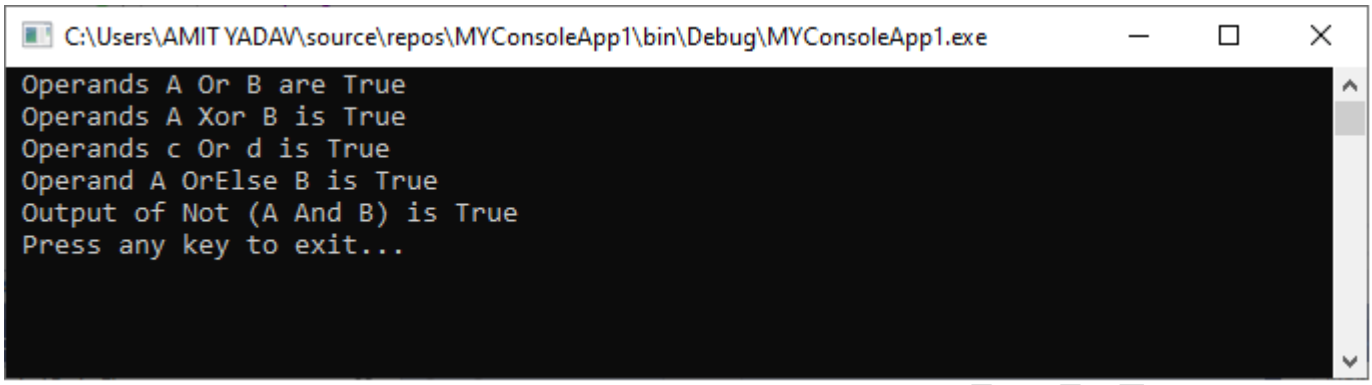
Example of **Logical and Bitwise Operator:**

Logic_Bitwise.vb

```
Imports System
Module Logic_Bitwise
    Sub Main()
        Dim A As Boolean = True
        Dim B As Boolean = False
        Dim c, d As Integer
        c = 10
        d = 20

        'Use of And Operator
        If A And B Then
            Console.WriteLine(" Operands A And B are True")
        End If

        'Use of Or Operator
        If A Or B Then
            Console.WriteLine(" Operands A Or B are True")
        End If
```

```
        'Use of Xor Operator
        If A Xor B Then
            Console.WriteLine(" Operands A Xor B is True")
        End If

        'Use of And Operator
        If c And d Then
            Console.WriteLine(" Operands c And d is True")
        End If

        'Use of Or Operator
        If c Or d Then
            Console.WriteLine(" Operands c Or d is True")
        End If

        'Use of AndAlso Operator
        If A AndAlso B Then
            Console.WriteLine(" Operand A AndAlso B is True")
        End If

        'Use of OrElse Operator
        If A OrElse B Then
            Console.WriteLine(" Operand A OrElse B is True")
        End If

        'Use of Not Operator
        If Not (A And B) Then
            Console.WriteLine(" Output of Not (A And B) is True")
        End If

        Console.WriteLine(" Press any key to exit?")
        Console.ReadKey()
    End Sub
End Module
```

Now compile and execute the above code by pressing the F5 button or Start button in Visual studio, it returns the following output:



Bit Shift Operators

The Bit Shit Operators are used to perform the bit shift operations on binary values either to the right or to the left.

Bit Shift operations in VB.NET

Operator	Description
AND	The Binary AND Operator are used to copy the common binary bit in the result if the bit exists in both operands.
OR	The Binary OR Operator is used to copy a common binary bit in the result if the bit found in either operand.
XOR	The Binary XOR Operator in VB.NET, used to determine whether a bit is available to copy in one operand instead of both.
Not	The binary NOT Operator is also known as the binary Ones' Compliment Operator, which is used to flip binary bits. This means it converts the bits from 0 to 1 or 1 to 0 binary bits.
<<	The Binary Left Shift Operator is used to shift the bit to the left side.
>>	The Binary Right Shift Operator is used to shift the bit to the right side.

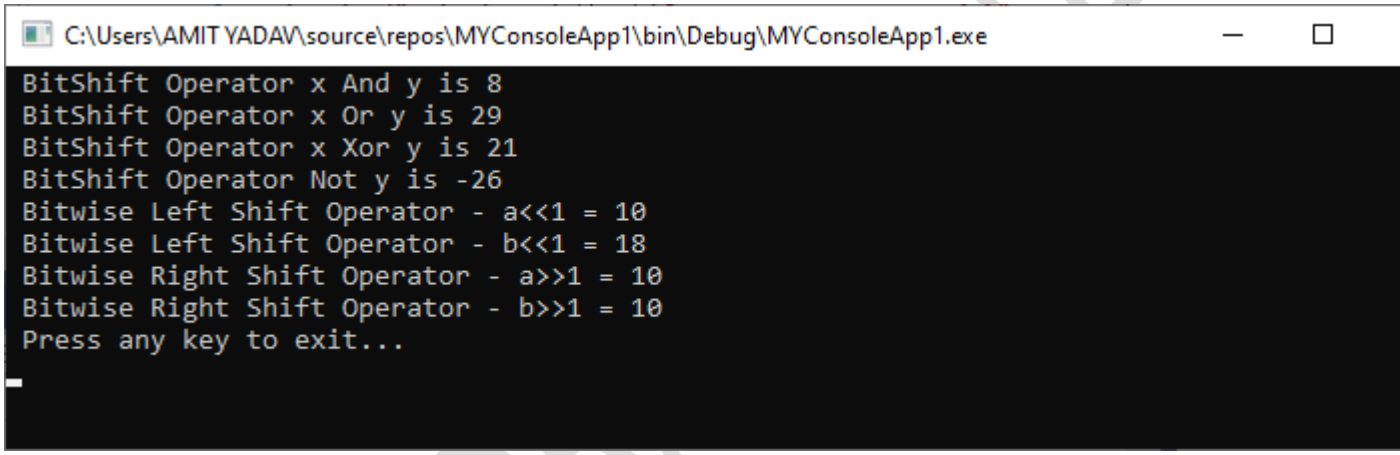
Example of **Bit Shift Operator in VB.NET:**

BitShift_Operator.vb

```
1.  Imports System
2.  Module Bitshift_Operator
3.      Sub Main()
4.          Dim x, y, z As Integer
5.          x = 12
6.          y = 25
7.          Dim a, b As Double
8.          a = 5 ' a = 5(00000101)
9.          b = 9 ' b = 9(00001001)
10.
11.         ' Use of And Operator
12.         z = x And y
13.         Console.WriteLine(" BitShift Operator x And y is {0}", z)
14.
15.         'Use of Or Operator
16.         z = x Or y
17.         Console.WriteLine(" BitShift Operator x Or y is {0}", z)
18.
19.         z = x Xor y
20.         Console.WriteLine(" BitShift Operator x Xor y is {0}", z)
21.
22.         z = Not y
23.         Console.WriteLine(" BitShift Operator Not y is {0}", z)
24.
25.         'Use of << Left-Shift Operator
26.         ' Output is 00001010
27.         Console.WriteLine(" Bitwise Left Shift Operator - a<<1 = {0}", a << 1)
28.
29.         'Output is 00010010
30.         Console.WriteLine(" Bitwise Left Shift Operator - b<<1 = {0}", b << 1)
31.
32.         'Use of >> Right-Shift Operator
33.         'Output is 00000010
34.         Console.WriteLine(" Bitwise Right Shift Operator - a>>1 = {0}", a << 1)
35.
```

```
36.         'Output is 00000100
37.         Console.WriteLine(" Bitwise Right Shift Operator - b>>1 = {0}", a << 1)
38.
39.         Console.WriteLine(" Press any key to exit...")
40.         Console.ReadKey()
41.     End Sub
42. End Module
```

Now compile and execute the above code by pressing the F5 button or Start button in Visual studio, it returns the following output:



Assignment Operators

The Assignment Operators are used to assign the value to variables in VB.NET.

Assignment Operators in VB.NET

Operator	Description	Example
=	It is a simple assignment Operator used to assign a right-side operand or value to a left side operand.	X = 5, X assign a value 5 X = P + Q, (P + Q) variables or value assign to X.
+=	An Add AND assignment Operator is used to add the value of the right operand to the left operand. And the result is assigned to the left operand.	X += 5, which means X= X+5 (5 will add and assign to X and then result saved to Left X operand)

-=	It is a Subtract AND assignment Operator, which subtracts the right operand or value from the left operand. And then, the result will be assigned to the left operand.	X -= P, which is same as X = X - P
***=**	It is a Multiply AND assignment Operator, which multiplies the right operand or value with the left operand. And then, the result will be assigned to the left operand.	X *= P, which is same as X = X * P
/=	It is a Divide AND assignment Operator, which divides the left operand or value with the right operand. And then, the result will be assigned to the left operand (in floating-point).	X /= P, which is same as X = X / P
\=	It is a Divide AND assignment Operator, which divides the left operand or value with the right operand. And then, the result will be assigned to the left operand (in integer-point division).	X \= P, which is same as X = X \ P
^=	It is an expression AND assignment Operator, which raises the left operand or value to the right operand's power. And then, the result will be assigned to the left operand.	X ^= P, which is same as X = X ^ P
&=	It is a concatenate string assignment Operator used to bind the right-hand string or variable with the left-hand string or variable. And then, the result will be assigned to the left operand.	Str &= name, which is same as Str = Str & name

Example of **Assignment Operator** in VB.NET:

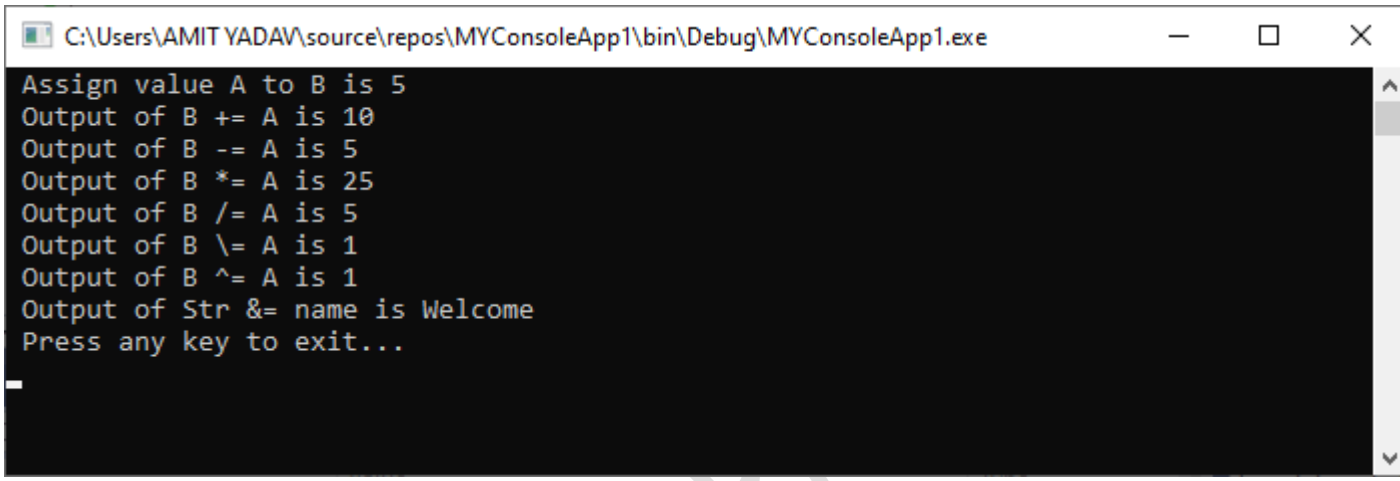
Assign_Operator.vb

```
Imports System
Module Assign_Operator
  Sub Main()
    'Declare variable and b As Integer
    Dim A As Integer = 5
    Dim B As Integer
```

```
Dim Str, name As String
name = "come"
Str = "Wel"

'Use of = Operator
B = A
Console.WriteLine(" Assign value A to B is {0}", B)

'Use of += Operator
B += A
Console.WriteLine(" Output of B += A is {0}", B)

'Use of -= Operator
B -= A
Console.WriteLine(" Output of B -= A is {0}", B)

'Use of *= Operator
B *= A
Console.WriteLine(" Output of B *= A is {0}", B)

'Use of /= Operator
B /= A
Console.WriteLine(" Output of B /= A is {0}", B)

'Use of = Operator
B \= A
Console.WriteLine(" Output of B \= A is {0}", B)

'Use of ^= Operator
B ^= A
Console.WriteLine(" Output of B ^= A is {0}", B)

'Use of &= Operator
Str &= name
Console.WriteLine(" Output of Str &= name is {0}", Str)

Console.WriteLine(" Press any key to exit...")
```

```
    Console.ReadKey()
  End Sub
End Module
```

Now compile and execute the above code by pressing the F5 button or Start button in Visual studio, it returns the following output:



Concatenation Operators

In VB.NET, there are two concatenation Operators to bind the operands:

Operator	Description	Example
&	It is an ampersand symbol that is used to bind two or more operand together. Furthermore, a nonstring operand can also be concatenated with a string variable (but in that case, Option Strict is on).	Result = Wel & come, Result = Welcome
+	It is also used to add or concatenate two number or string.	Result = Wel + come, Result = Welcome

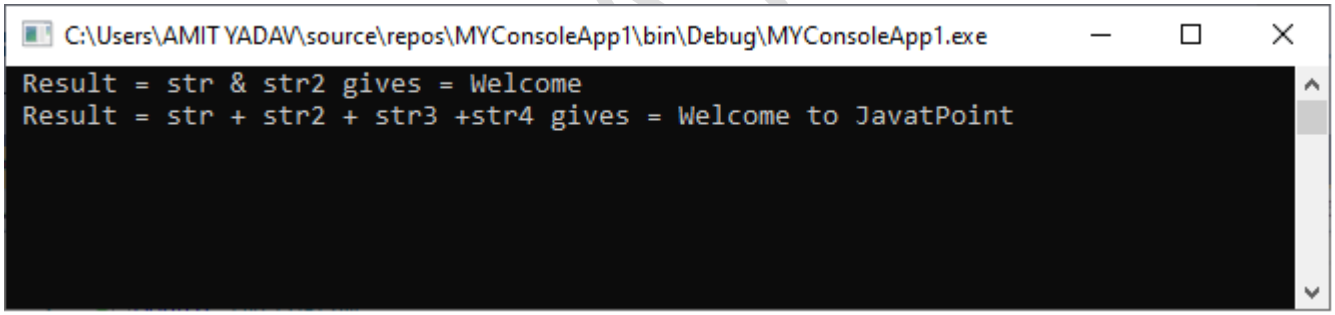
Example of Concatenation Operators in VB.NET.

MyProgram.vb

```
Imports System
Module MyProgram
```

```
Sub Main()
    Dim str As String = "Wel"
    Dim str2 As String = "come"
    Dim str3 As String = " "
    Dim str4 As String = "to Sai College"
    Dim result As String
    Dim result2 As String
    result = str & str2
    Console.WriteLine(" Result = & 'str' & str2 gives = {0}", result)
    result2 = str + str2 + str3 + str4
    Console.WriteLine(" Result = str + str2 + str3 +str4 gives = {0}", result2.ToString)
Console.ReadLine()
  End Sub
End Module
```

Now compile and execute the above code by pressing the F5 button or Start button in Visual studio, it returns the following output:



Miscellaneous Operators

There are some important Operator in VB.NET

Operator	Description	Example
Await	An Await Operator is used in an operand to suspend the execution of an asynchronous method or lambda expression until the awaited task completes.	Dim output as out = Await AsyncMethodThatReturnsResult() Await AsyncMethod()
AddressOf	The AddressOf Operator is used to	AddHandler Button2.Click, AddressOf

	provide a reference to the address of a procedure.	Button2_Click
GetType	A GetType Operator is used to retrieve the type of the specified object. In addition, the retrieved object type provides various information such as methods, properties, and events.	MsgBox(GetType("199.99").ToString())
Function Expression	It defines the lambda expression, which declares the parameter and code. A Lambda expression is a function that is used to calculate and return value without defining the name.	Dim mul2= Function(num As Integer) num * 4 Console.WriteLine(mul2(4))
If	The If Operator using short circuit evaluation to conditionally return a single object value from two defined object values. The If Operator can be used with two or three defined arguments.	Dim a = -4 Console.WriteLine(If (a >= 0, "Positive", "Negative"))

Example of **Miscellaneous Operator**s in VB.NET.

Misc_Operator.vb

```
1. Imports System
2. Module Misc_Operator
3.     Sub Main()
4.         ' Initialize a variable
5.         Dim a As Integer = 50
6.         ' GetType of the Defined Type
7.         Console.WriteLine(GetType(Double).ToString())
8.         Console.WriteLine(GetType(Integer).ToString())
9.         Console.WriteLine(GetType(String).ToString())
10.        Console.WriteLine(GetType(Single).ToString())
11.        Console.WriteLine(GetType(Decimal).ToString())
```

```
12.
13.         'Use of Function()
14.         Dim multiplywith10 = Function(sum As Integer) sum * 10
15.         Console.WriteLine(multiplywith10(10))
16.         Console.WriteLine(If(a >= 0, "Positive", "Negative"))
17.
18.         Console.WriteLine(" Press any key to exit...")
19.         Console.ReadLine()
20.
21.     End Sub
22. End Module
```

Now compile and execute the above code by pressing the F5 button or Start button in Visual studio, it returns the following output:

Operator Precedence in VB.NET

Operator precedence is used to determine the order in which different Operators in a complex expression are evaluated. There are distinct levels of precedence, and an Operator may belong to one of the levels. The Operators at a higher level of precedence are evaluated first. Operators of similar precedents are evaluated at either the left-to-right or the right-to-left level.

The Following table shows the operations, Operators and their precedence -

Operations	Operators	Precedence
Await		Highest

Exponential	**^**	
Unary identity and negation	**+, -**	
Multiplication and floating-point division	***, /**	
Integer division	****	
Modulus arithmetic	**Mod**	
Addition and Subtraction	**+, -**	
Arithmetic bit shift	**<<, >>**	
All comparison Operators	**=, <>, <, <=, >, >=, Is, IsNot, Like, TypeOf ...is**	
Negation	**Not**	
Conjunction	**And, AndAlso**	
Inclusive disjunction	**Or, Else**	
Exclusive disjunction	**Xor**	Lowest

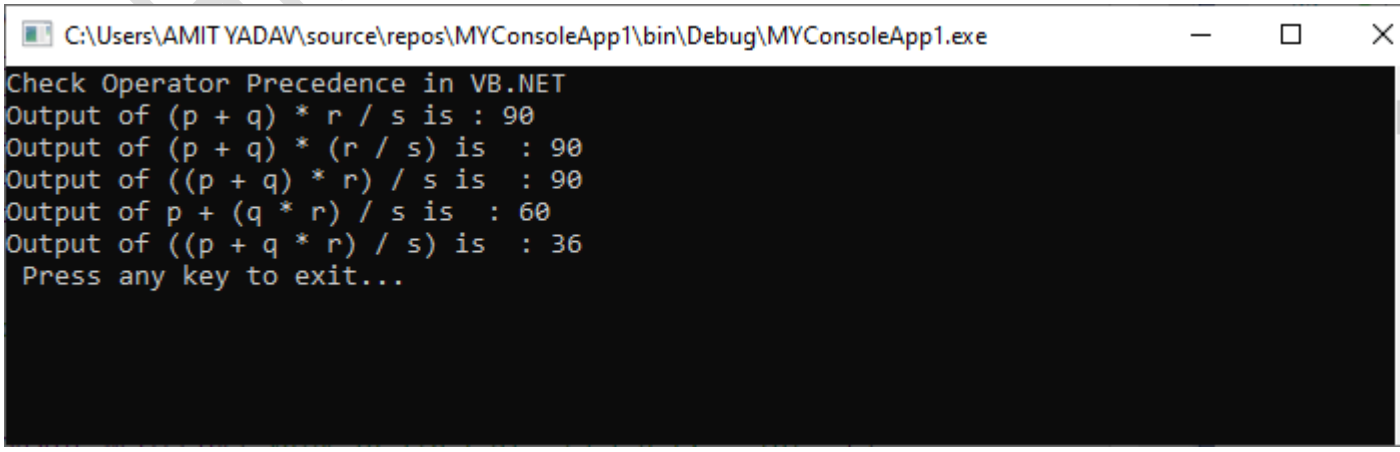
Example of **Operator Precedence in VB.NET.**

Operator_Precedence.vb

```
1. Imports System
2. Module Operator_Precedence
3.     Sub Main()
4.         'Declare and Initialize p, q, r, s variables
5.         Dim p As Integer = 30
6.         Dim q As Integer = 15
7.         Dim r As Integer = 10
8.         Dim s As Integer = 5
```

```
9.        Dim result As Integer
10.
11.       Console.WriteLine("Check Operator Precedence in VB.NET")
12.       'Check Operator Precedence
13.       result = (p + q) * r / s      '  45 * 10 / 5
14.       Console.WriteLine("Output of (p + q) * r / s is : {0}", result)
15.
16.       result = (p + q) * (r / s)   ' (45) * (10/5)
17.       Console.WriteLine("Output of (p + q) * (r / s) is  : {0}", result)
18.
19.       result = ((p + q) * r) / s    ' (45 * 10 ) / 5
20.       Console.WriteLine("Output of ((p + q) * r) / s is  : {0}", result)
21.
22.       result = p + (q * r) / s     '  30 + (150/5)
23.       Console.WriteLine("Output of p + (q * r) / s is  : {0}", result)
24.
25.       result = ((p + q * r) / s)     '  ((30 + 150) /5)
26.       Console.WriteLine("Output of ((p + q * r) / s) is  : {0}", result)
27.
28.       Console.WriteLine(" Press any key to exit...")
29.       Console.ReadKey()
30.    End Sub
31. End Module
```

Now compile and execute the above code by pressing the F5 button or Start button in Visual studio, it returns the following output:

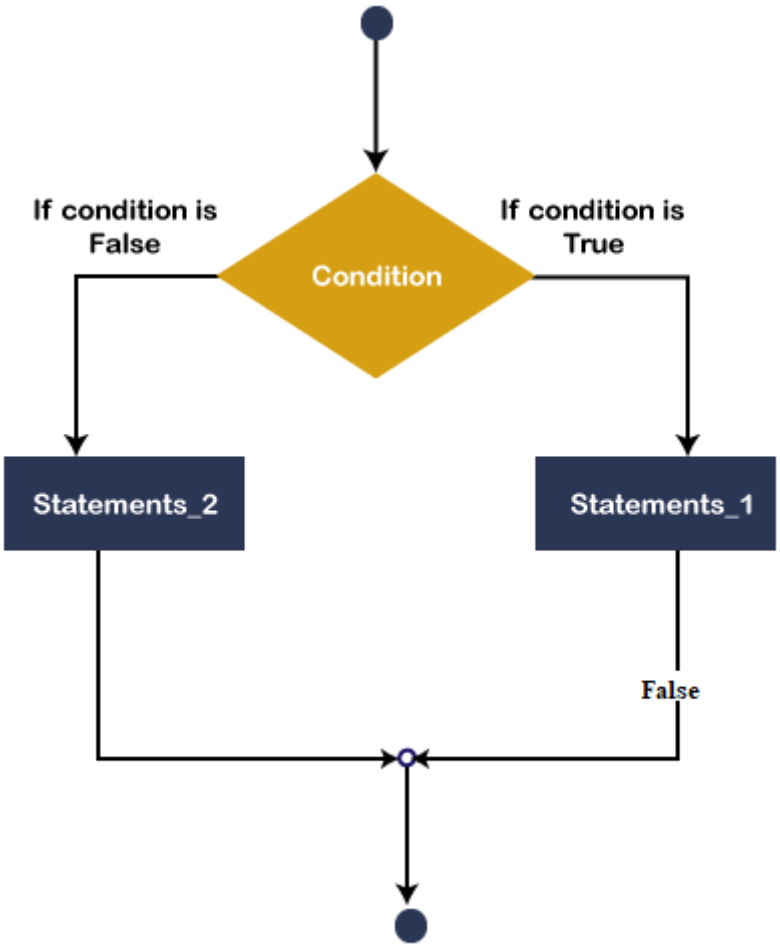


VB.NET Control Statements

In **VB.NET**, the **control statements** are the statements that controls the execution of the program on the basis of the specified condition. It is useful for determining whether a condition is true or not. If the condition is true, a single or block of statement is executed. In the control statement, we will use **if- Then, if Then Else, if Then ElseIf** and the **Select case** statement.

We can define more than one condition to be evaluated by the program with statements. If the defined condition is true, the statement or block executes according to the condition, and if the condition is false, another statement is executed.

The following figure shows a common format of the decision control statements to validate and execute a statement:



The above diagram shows that if the defined condition is true, statement_1 will be executed, and if the condition is false, statement_2 will be executed.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

VB.NET provides the following conditional or decision-making statements.

- If-Then Statement
- If-Then Else Statement
- If-Then ElseIf Statement
- Select Case Statement
- Nested Select Case Statements

If-Then Statement

The **If-Then** Statement is a control statement that defines one or more conditions, and if the particular condition is satisfied, it executes a piece of information or statements.

Syntax:

```
1. If condition Then
2. [Statement or block of Statement]
3. End If
```

In **If-Then** Statement, the **condition** can be a Boolean, logical, or relational condition, and the statement can be single or group of statements that will be executed when the condition is true.

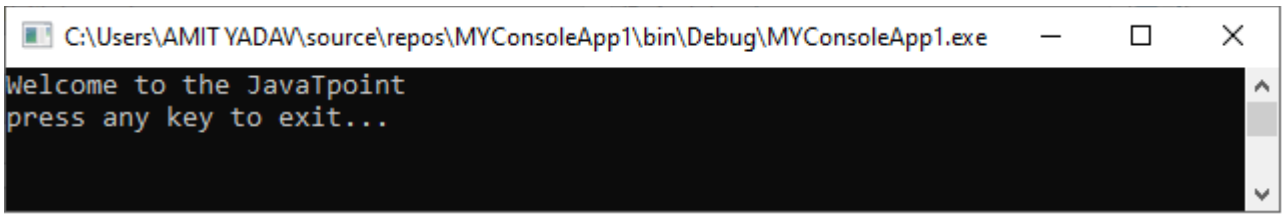
Example 1: Write a simple program to print a statement in VB.NET.

Module1.vb

```
1. Module Module1
2.     ' Declaration of variable str
3.     Dim str As String = "Sai College"
4.     Sub Main()
5.         'if str equal to "Sai College", below Statement will be executed.
6.         If str = "Sai College" Then
7.             Console.WriteLine("Welcome to the Sai College")
8.         End If
9.     Console.WritLine("press any key to exit?")
10.    Console.ReadKey()
```

```
11.     End Sub
12. End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:



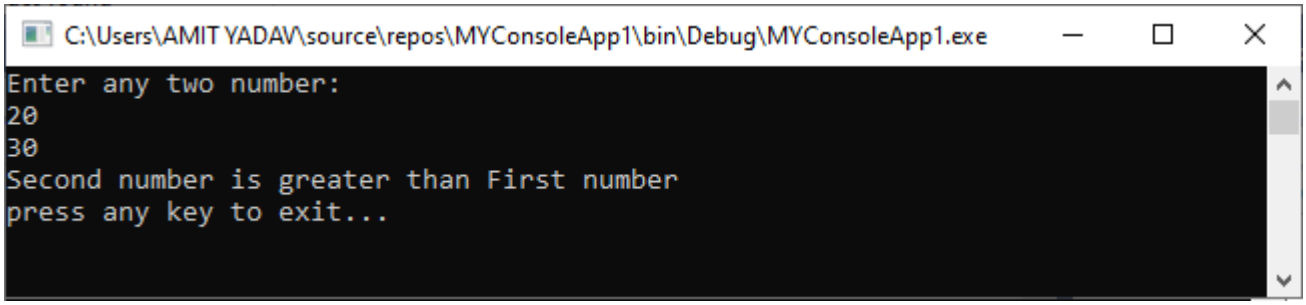
As we can see in the above example, if the value of **str** is equal to **Sai College**, the condition is **true**, and it prints the Statement.

Example 2: Write a program to print a number is greater than another number in VB.NET.

if_statment2.vb

```
1.  Module if_statement2
2.      Sub Main()
3.          ?Definition of variables
4.          Dim no1, no2 As Integer
5.          Console.WriteLine("Enter any two number:")
6.          no1 = Console.ReadLine() ?read no1 from user
7.          no2 = Console.ReadLine() ?read no2 from user
8.          If no1 > no2 Then
9.              Console.WriteLine("First number is greater than second number")
10.         End If
11.         If no1 < no2 Then
12.             Console.WriteLine("Second number is greater than First number")
13.         End If
14.         Console.WriteLine("press any key to exit...")
15.         Console.ReadKey()
16. End Sub
17. End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:



In the above program, we enter two numbers to find the greater number using the relational operator. And if the first number is greater than the other, the first statement is executed; otherwise, the second statement will be executed.

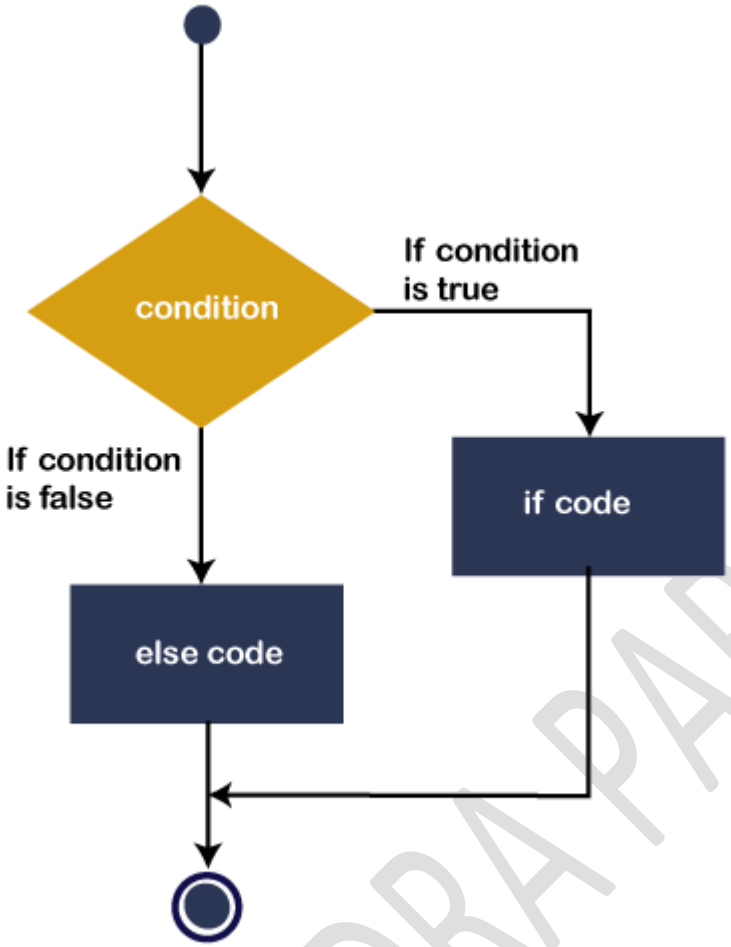
If-Then-Else Statement

The **If-Then** Statement can execute single or multiple statements when the condition is true, but when the expression evaluates to **false**, it does nothing. So, here comes the **If-Then-Else** Statement. The IF-Then-Else Statement is telling what **If** condition to do when if the statement is false, it executes the Else statement. Following is the If-Then-Else statement syntax in VB.NET as follows:

Syntax:

```
1. If (Boolean_expression) Then
2. 'This statement will execute if the Boolean condition is true
3. Else
4. 'Optional statement will execute if the Boolean condition is false
5. End If
```

Flow chart



The above diagram represents that if the Boolean expression (condition) is **true**, the if statement will execute, and if the Boolean expression is false, **Else code or statement** will be executed. After that, the control transfer to the next statement, which is immediately after the If-Then-Else control statement.

Example 1: Write a program to check whether the number is even or odd.

If_Else_statment.vb

```
1. Module If_Else_statement
2.     Sub Main()
3.         Dim num As Integer
4.         Console.WriteLine("Enter the Number")
5.         num = Console.ReadLine() 'read data from console
6.
7.         If (num Mod 2 = 0) Then ' if condition is true, print the if statement
8.             Console.WriteLine("It is an even number")
9.
```

```
10.       Else 'otherwise, Else statement is executed.
11.           Console.WriteLine("It is an odd number")
12.       End If
13.
14.    Console.WriteLine("press any key to exit...")
15.       Console.ReadKey()
16.    End Sub
17. End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:

Example 2: Write a program to print the larger and smaller of the two numbers.

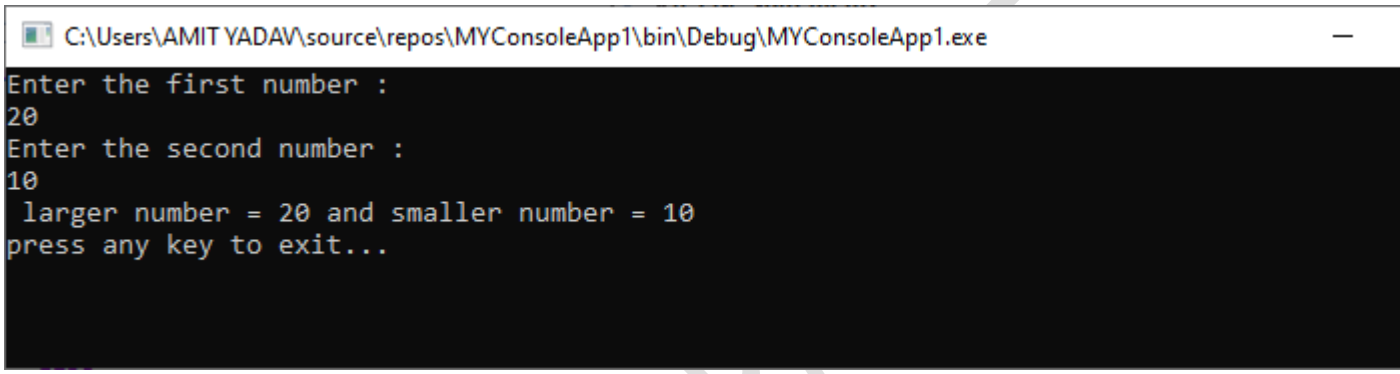
if_else_statment2.vb

```
1.  Module if_else_statement2
2.     Sub Main()
3.        Dim a As Integer
4.        Dim b As Integer
5.        Console.WriteLine("Enter the first number : ")
6.        a = Console.ReadLine()
7.
8.        Console.WriteLine("Enter the second number : ")
9.        b = Console.ReadLine()
10.
11.       If a > b Then
12.           Console.WriteLine(" larger number = {0} and smaller number = {1} ", a, b)
13.       Else
14.           Console.WriteLine(" larger number = {0} and smaller number = {1} ", b, a)
15.       End If
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
16.
17.         Console.WriteLine("press any key to exit...")
18.         Console.ReadKey()
19.     End Sub
20. End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:



VB.NET If-Then-ElseIf statement

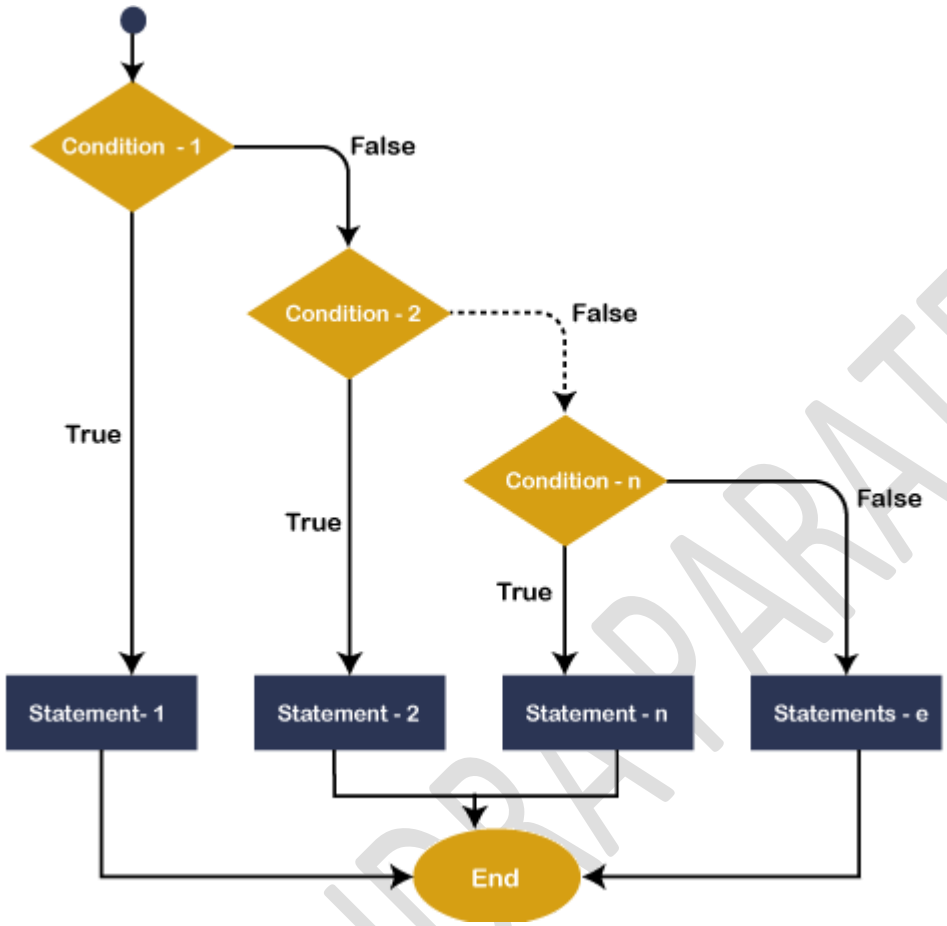
The **If-Then-ElseIf** Statement provides a choice to execute only one condition or statement from multiple statements. Execution starts from the top to bottom, and it checked for each If condition. And if the condition is met, the block of If the statement is executed. And **if none** of the conditions are true, the last **block** is executed. Following is the syntax of If-Then-ElseIf Statement in VB.NET as follows:

Syntax

```
1. If(condition 1)Then
2.     ' Executes when condition 1 is true
3. ElseIf( condition 2)Then
4.     ' Executes when condition 2 is true
5. ElseIf( boolean_expression 3)Then
6.     ' Executes when the condition 3 is true
7. Else
8.     ' executes the default statement when none of the above conditions is true.
9. End If
```

Flowchart

The following diagram represents the functioning of the If-Else-If Statement in the VB.NET programming language.



If this condition is true in the flowchart of the if-else-if statement, the statement is executed within the if block. If the condition is not true, it passes control to the next ElseIf condition to check whether the condition is matched. And if none of the conditions are matched, the else block is executed.

Example 1: Write a program to show the uses of If... ElseIf statements.

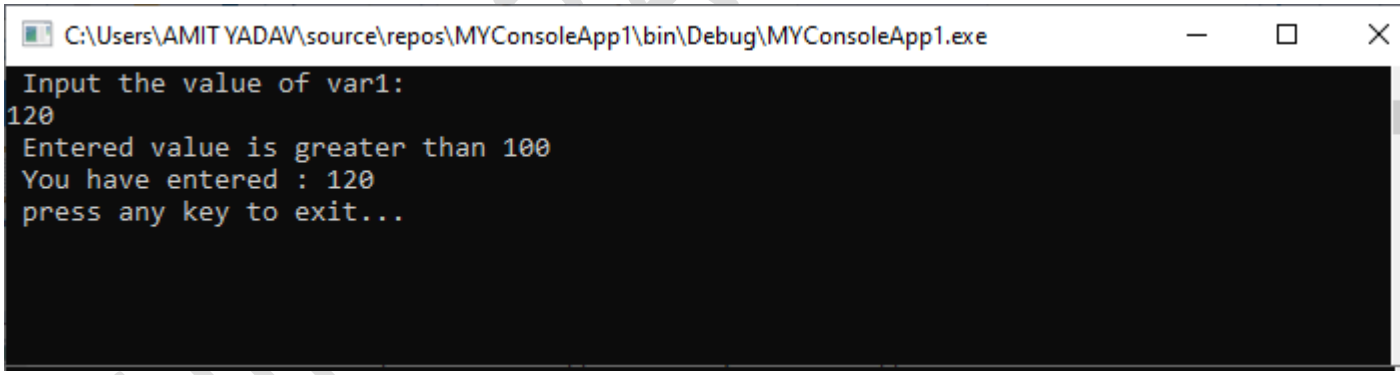
if_elseIf.vb

```
Module if_elseIf
  Sub Main()
    Dim var1 As Integer

    Console.WriteLine(" Input the value of var1: ")
    var1 = Console.ReadLine()
    If var1 = 20 Then
      'if condition is true then print the following statement'
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
        Console.WriteLine(" Entered value is equal to 20")
    ElseIf var1 < 50 Then
        Console.WriteLine(" Entered value is less than 50")

     ElseIf var1 >= 100 Then
        Console.WriteLine(" Entered value is greater than 100")
    Else
        'if none of the above condition is satisfied, print the following statement
        Console.WriteLine(" Value is not matched with above condition")
    End If
    Console.WriteLine(" You have entered : {0}", var1)
    Console.WriteLine(" press any key to exit...")
    Console.ReadKey()
  End Sub
End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:



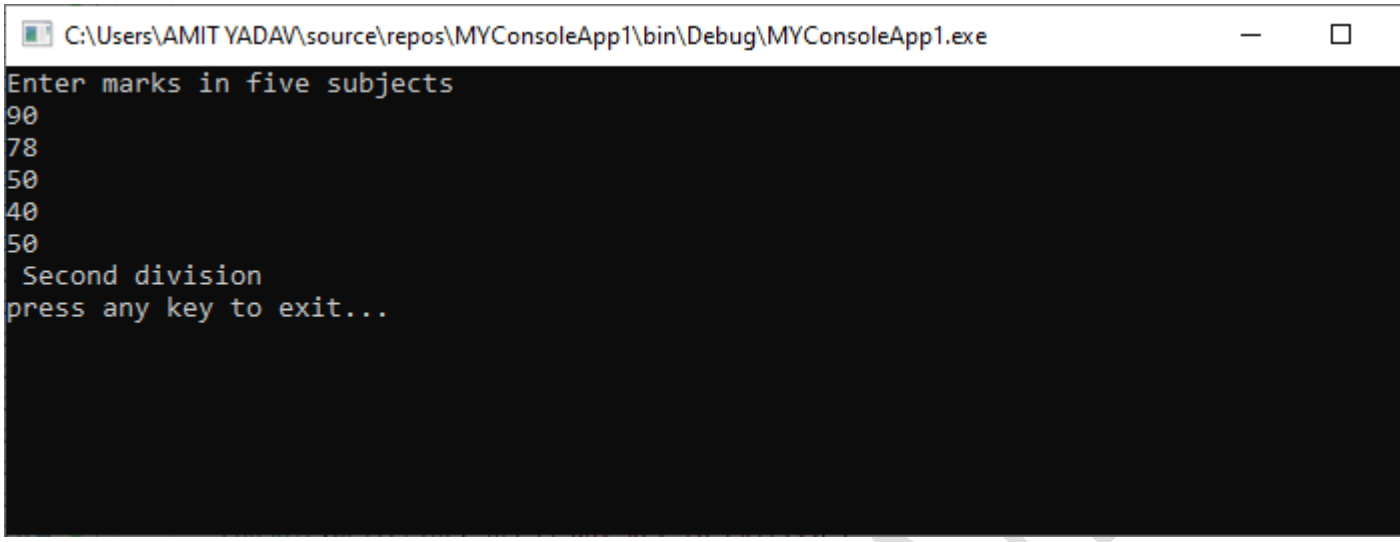
Example 2: Write a program to use the If-Then-ElseIf Statement for calculating the division obtained by the student. Also, take the marks obtained by the student in 5 different subjects from the keyboard.

if_elseIf2.vb

```
Module If_elseIf2
  Sub Main()      ' execution start from Main() method
    Dim m1, m2, m3, m4, m5, per As Integer
    Console.WriteLine("Enter marks in five subjects ")
    ' Read the marks of five subject
```

```
        m1 = Console.ReadLine()
        m2 = Console.ReadLine()
        m3 = Console.ReadLine()
        m4 = Console.ReadLine()
        m5 = Console.ReadLine()
        per = (m1 + m2 + m3 + m4 + m5) / 5
        If (per >= 70) Then
            'if condition is true, print the first division
            Console.WriteLine(" First division")
        ElseIf (per >= 60) Then
            'if ElseIf condition is true, print the second division
            Console.WriteLine(" Second division")
        ElseIf (per >= 50) Then
            'if ElseIf condition is true, print the third division
            Console.WriteLine(" Third division")
        ElseIf (per >= 40) Then
            'if ElseIf condition is true, print only pass with grace
            Console.WriteLine(" Only Pass with Grace")
        Else
            'if none of the condition is true, print the Failed
            Console.WriteLine(" Failed")
        End If
        Console.WriteLine("press any key to exit...")
        Console.ReadKey()
    End Sub

End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:



Select Case Statement

In VB.NET, the Select Case statement is a collection of multiple case statements, which allows executing a single case statement from the list of statements. A selected case statement uses a variable to test for **equality** against multiple cases or statements in a program. If the variable is matched with any test cases, that statement will be executed. And if the condition is not matched with any cases, it executes the default statement.

Using the select case statement in VB.NET programming, you can replace the uses of multiple If-Then-Else If statement from the program for better readability and easy to use.

Syntax

Following is the syntax of the Select Case statement in VB.NET, as follows:

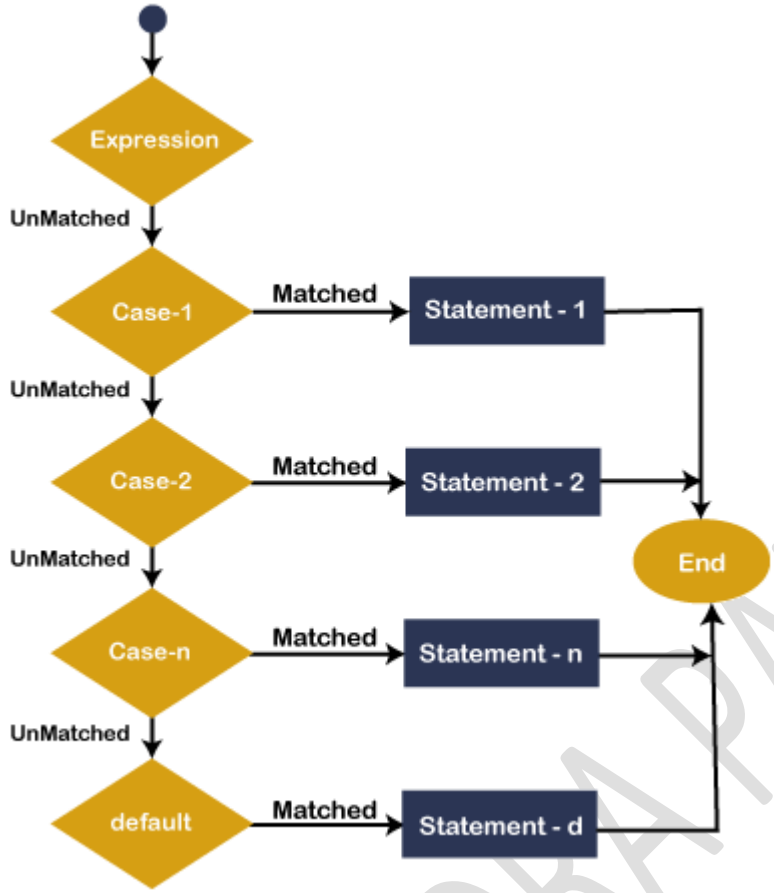
```
1. Select Case [variable or expression]
2. Case value1 'defines the item or value that you want to match.
3. // Define a statement to execute
4.
5. Case value2 'defines the item or value that you want to match.
6. // Define a statement to execute
7.
8. Case Else
9. // Define the default statement if none of the conditions is true.
10. End Select
```

Furthermore, you can also set more than one condition in a single case statement, such as:

```
1. Select Case Variable / expression
2. Case value1
3. Statement1
4.
5. Case value2, value3
6. Statement2
7.
8. Case Else
9. // define the default statement if none of the condition is true
10. End Select
```

Flowchart of Select Case Statement

The following flowchart represents the functioning of the Select case statement in the VB.NET programming language.



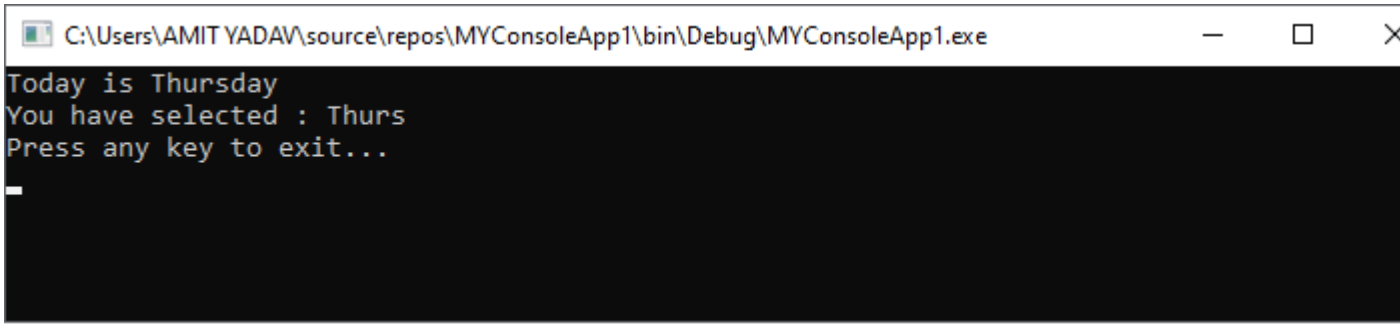
In Flowchart, the Select Case statement represents the evaluating of the process start from top to bottom. If the expression or value is matched with the first select case, statement -1 is executed else the control transfer to the next case for checking whether the expression is matching or not. Similarly, it checks all Select case statements for evaluating. If none of the cases are matched, the Else block statement will be executed, and finally, the Select Case Statement will come to an end.

Example 1: Write a program to display the Days name using the select case statement in VB.NET.

Select_case.vb

```
1. Imports System
2. Module Select_case
3.     Sub Main()
4.         'define a local variable.
5.         Dim Days As String
6.         Days = "Thurs"
7.         Select Case Days
8.             Case "Mon"
9.                 Console.WriteLine(" Today is Monday")
10.            Case "Tue"
11.                Console.WriteLine(" Today is Tuesday")
12.            Case "Wed"
13.                Console.WriteLine("Today is Wednesday")
14.            Case "Thurs"
15.                Console.WriteLine("Today is Thursday")
16.            Case "Fri"
17.                Console.WriteLine("Today is Friday")
18.            Case "Sat"
19.                Console.WriteLine("Today is Saturday")
20.            Case "Sun"
21.                Console.WriteLine("Today is Sunday")
22.            Case Else
23.                Console.WriteLine(" You have typed Something wrong")
24.
25.        End Select
26.        Console.WriteLine("You have selected : {0}", Days)
27.        Console.WriteLine("Press any key to exit...")
28.        Console.ReadLine()
29.    End Sub
30. End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:



In the select case statement, the value of Days **"Thurs"** will compare all the available **select cases**' values in a program. If a value matched with any condition, it prints the particular statement, and if the value is not matched with any select case statement, it prints the default message.

Example 2: Write a program to perform an arithmetic operation using the Select case statement in VB.NET.

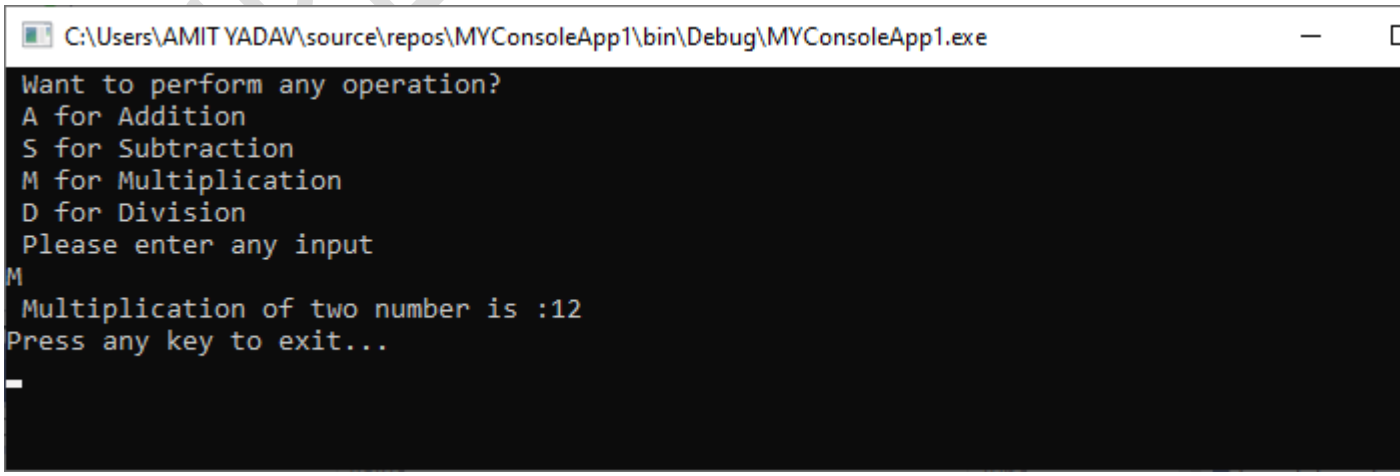
Operation.vb

Operation.vb

```
1.  Imports System
2.  Module Operation
3.      Sub main()
4.          'declaration of the variables
5.          Dim num1, num2, sum As Integer
6.          Dim def As Char
7.          'initialization of num1 and num2 variable
8.          num1 = 2
9.          num2 = 6
10.         Console.WriteLine(" Want to perform any operation?")
11.         Console.WriteLine(" A for Addition")
12.         Console.WriteLine(" S for Subtraction")
13.         Console.WriteLine(" M for Multiplication")
14.         Console.WriteLine(" D for Division")
15.         Console.WriteLine(" Please enter any input")
16.         def = Console.ReadLine()
17.         Select Case def
18.             Case "A"
19.                 'perform Addition
20.                 sum = num1 + num2
```

```
21.             Console.WriteLine(" Addition of two number is :{0}", sum)
22.         Case "S"
23.             'perform Subtraction
24.             sum = num2 - num1
25.             Console.WriteLine(" Subtraction of two number is :{0}", sum)
26.         Case "M"
27.             'perform Multiplication
28.             sum = num1 * num2
29.             Console.WriteLine(" Multiplication of two number is :{0}", sum)
30.         Case "D"
31.             'Peform Division
32.             sum = num2 / num1
33.             Console.WriteLine(" Division of two number is :{0}", sum)
34.         Case Else
35.             'If none of the operation matched, call default statement
36.             Console.WriteLine(" Please enter only define operation With Capital letter")
37.     End Select
38.     Console.WriteLine("Press any key to exit...")
39.     Console.ReadKey()
40. End Sub
41. End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:



In the above example, we defined Select with multiple case statements, and if the **user-defined** input is matched with any defined case statement, it executes that

statement. And if the condition is not matched with any case, it executes a default statement in VB.NET.

Here, we provide **'M'** as input, which checks all case statements, and if any case is matched with M, it executes the statement within the respective Case statement.

VB.NET Nested Select Case statements

When a **Select Case** statement is written inside the body of another **Select Case statement** is called a **nested Select Case statement**.

Syntax:

```
Select Case "num"
' code to be executed if num = 1
Case 1
' nested Select case
    Select Case n

                    ' code to be executed if n = 5
                     Case 5
                    Statement 1

                   ' code to be executed if n = 10
                    Case 10
                    Statement 2

                     ' code to be executed if n = 15
                      Case 15
                     Statement 3

                  ' code to be executed if n doesn't match with any cases.
                  Case Else
                   Statement

       ' code to be executed if num = 2
        Case 2
        Statement 2
```

```
27.          ' code to be executed if num = 3
28.          Case 3
29.          Statement 3
30.
31.          ' code to be executed if num doesn't match with any cases.
32.          Case Else
33.          Statement
```

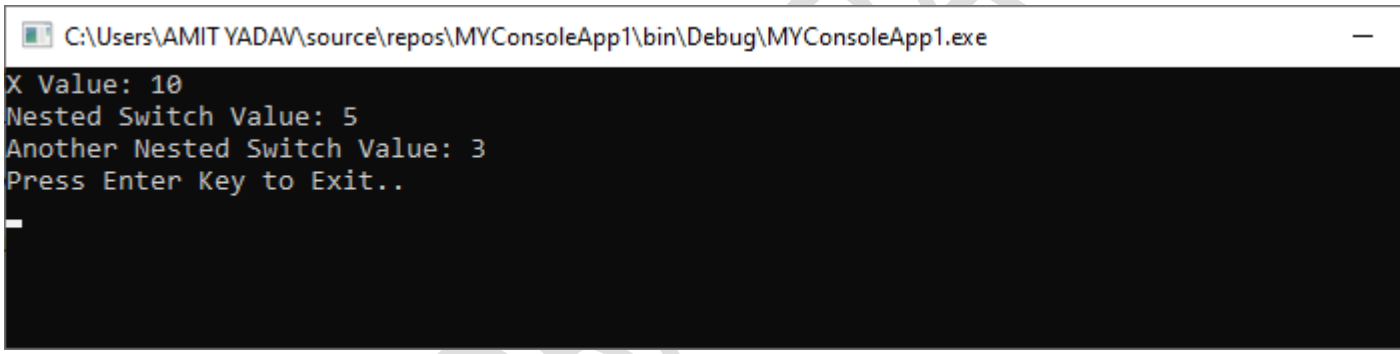
Example 1: Write a program to use a nested select case statement in VB.NET.

Module1.vb

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Dim x As Integer = 10, y As Integer = 5
6.          Select Case x
7.
8.              Case 10
9.                  Console.WriteLine("X Value: 10")
10.
11.                 Select Case y
12.                     Case 5
13.                         Console.WriteLine("Nested Switch Value: 5")
14.
15.                         Select Case y - 2
16.                             Case 3
17.                                 Console.WriteLine("Another Nested Switch Value: 3")
18.
19.                         End Select
20.                 End Select
21.
22.             Case 15
23.                 Console.WriteLine("X Value: 15")
24.
25.             Case 20
26.                 Console.WriteLine("X Value: 20")
```

```
27.
28.             Case Else
29.                 Console.WriteLine("Not Known")
30.
31.         End Select
32.         Console.WriteLine("Press Enter Key to Exit..")
33.         Console.ReadLine()
34.     End Sub
35. End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:



Example 2: Write a program to use the nested select case statement in VB.NET.

nested_selectcase.vb

```
1.  Imports System
2.  Module nested_selectcase
3.      Sub Main()
4.          Dim num As Integer
5.          Dim str As String
6.          str = "F"
7.
8.          Console.WriteLine(" Enter only First three number like 1, 2, 3")
9.          num = Console.ReadLine() 'take input from the user
10.         Select Case num
11.             Case 1
12.                 Console.WriteLine(" You are in block 1")
13.                 Console.WriteLine("Only First two letter such as A and B")
14.
```

```
15.             str = Console.ReadLine()
16.             Select Case str
17.                 Case "A", "a"
18.                     Console.WriteLine(" This is a VB.NET Tutorial")
19.                 Case "B", "b"
20.                     Console.WriteLine(" Welcome to the Sai College")
21.                 Case Else
22.                     Console.WriteLine(" Something is wrong")
23.             End Select
24.
25.         Case 2
26.             Console.WriteLine(" You are in block 2")
27.             Console.WriteLine("Only First two letter such as C and D")
28.             str = Console.ReadLine()
29.             Select Case str
30.                 Case "C", "c"
31.                     Console.WriteLine(" Welcome to the World!")
32.                 Case "D", "d"
33.                     Console.WriteLine(" Want to go in Heaven")
34.                 Case Else
35.                     Console.WriteLine(" Something is wrong")
36.             End Select
37.
38.         Case 3
39.             Console.WriteLine(" You are in block 3")
40.             Console.WriteLine("Only First two letter such as E and F")
41.             str = Console.ReadLine()
42.             Select Case str
43.                 Case "E", "e"
44.                     Console.WriteLine(" VB.NET is a programming language to develop web, wind
    ow, and console-based application. ")
45.                 Case "F", "f"
46.                     Console.WriteLine(" You have to basic knowledge of c and c++")
47.                 Case Else
48.                     Console.WriteLine(" Something is wrong")
49.             End Select
50.         Case Else
```

```
51.             Console.WriteLine(" Something is wrong")
52.         End Select
53.         Console.ReadLine()
54.     End Sub
55. End Module
```

Now compile and execute the above program by clicking on the Start or F5 button, it shows the following output:

VB.NET Do Loop

A **Loop** is used to repeat the same process multiple times until it meets the specified condition in a program. By using a loop in a program, a programmer can repeat any number of statements up to the desired number of repetitions. A loop also provides the suitability to a programmer to repeat the statement in a program according to the requirement. A loop is also used to reduce the program **complexity, easy to understand**, and easy to **debug**.

Advantages of VB.NET loop

- It provides code iteration functionality in a program.
- It executes the statement until the specified condition is true.
- It helps in reducing the size of the code.
- It reduces compile time.

Types of Loops

There are five types of loops available in VB.NET:

- Do While Loop
- For Next Loop
- For Each Loop
- While End Loop
- With End Loop

Do While Loop

In VB.NET, Do While loop is used to execute blocks of statements in the program, as long as the condition remains true. It is similar to the While End Loop, but there is slight difference between them. The **while** loop **initially checks** the defined condition, if the condition becomes true, the while loop's statement is executed. Whereas in the **Do** loop, is opposite of the while loop, it means that it executes the Do statements, and then it checks the condition.

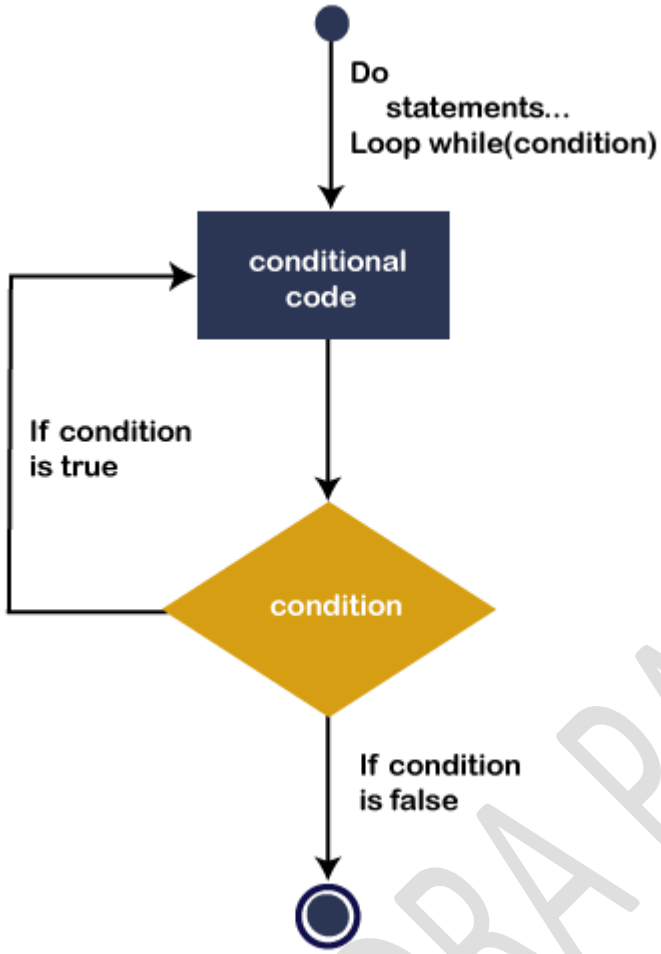
Syntax:

```
1. Do
2. [ Statements to be executed]
3. Loop While Boolean_expression
4. // or
5. Do
6. [Statement to be executed]
7. Loop Until Boolean_expression
```

In the above syntax, the **Do** keyword followed a block of statements, and **While** keyword checks **Boolean_expression** after the execution of the first Do statement.

Flowchart of Do loop

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



The above flow chart represents the flow of Do While loop. It is used to **control the flow of statements**, such that it executes the statement at least once before checking the While or Until condition. If the condition is true, the next iteration will be executed till the condition become false.

Example 1. Write a simple program to print a number from 1 to 10 using the Do While loop in VB.NET.

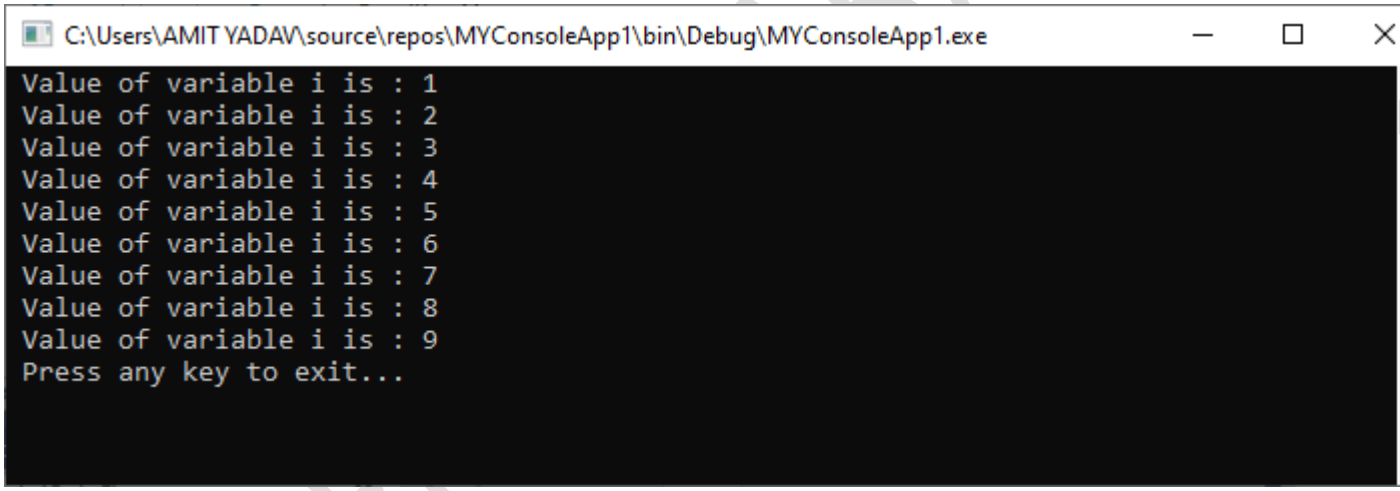
Do_loop.vb

```
1. Imports System
2. Module Do_loop
3.     Sub Main()
4.         ' Initializatio and Declaration of variable i
5.         Dim i As Integer = 1
6.
7.
8.
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
9.  Do
10.         ' Executes the following Statement
11.         Console.WriteLine(" Value of variable I is : {0}", i)
12.         i = i + 1 'Increment the variable i by 1
13.     Loop While i <= 10 ' Define the While Condition
14.
15.     Console.WriteLine(" Press any key to exit...")
16.     Console.ReadKey()
17.  End Sub
18. End Module
```

Now compile and execute the above program by clicking on the Start button, it shows the following output:



In the above program, the Do While loop executes the body until the given condition becomes **false**. When the condition becomes false the loop will be terminated.

Use of Until in Do Until Loop statement

In the VB.NET loop, there is a **Do Until loop** statement, which is similar to the **Do While loop.** The Do Statement executes as long as Until condition becomes true.

Example: Write a program to understand the uses of Do Until Loop in VB.NET.

Do_loop.vb

```
1. Imports System
2. Module Do_loop
3.     Sub Main()
```

```
4.          ' Initialization and Declaration of variable i
5.          Dim i As Integer = 1
6.          Do
7.              ' Executes the following Statement
8.              Console.WriteLine(" Value of variable i is : {0}", i)
9.              i = i + 1 'Increment variable i by 1
10.         Loop Until i = 10 ' Define the Until Condition
11.
12.         Console.WriteLine(" Press any key to exit...")
13.         Console.ReadKey()
14.     End Sub
15. End Module
```

Output:

In the above program, a Do Until loop is executed their statement until the given condition **Until (i =10)** is not meet. When the counter value of the **variable i** becomes 10, the defined statement will be false, and the loop will be terminated.

Nested Do While Loop Statement

In VB.NET, when we use one Do While loop inside the body of another Do While loop, it is called Nested Do While loop.

Syntax

```
1. Do
2. // Statement of the outer loop
3. Do
4.         // Statement of inner loop
5. While (condition -2)
6. // Statement of outer loop
7. While (condition -1)
```

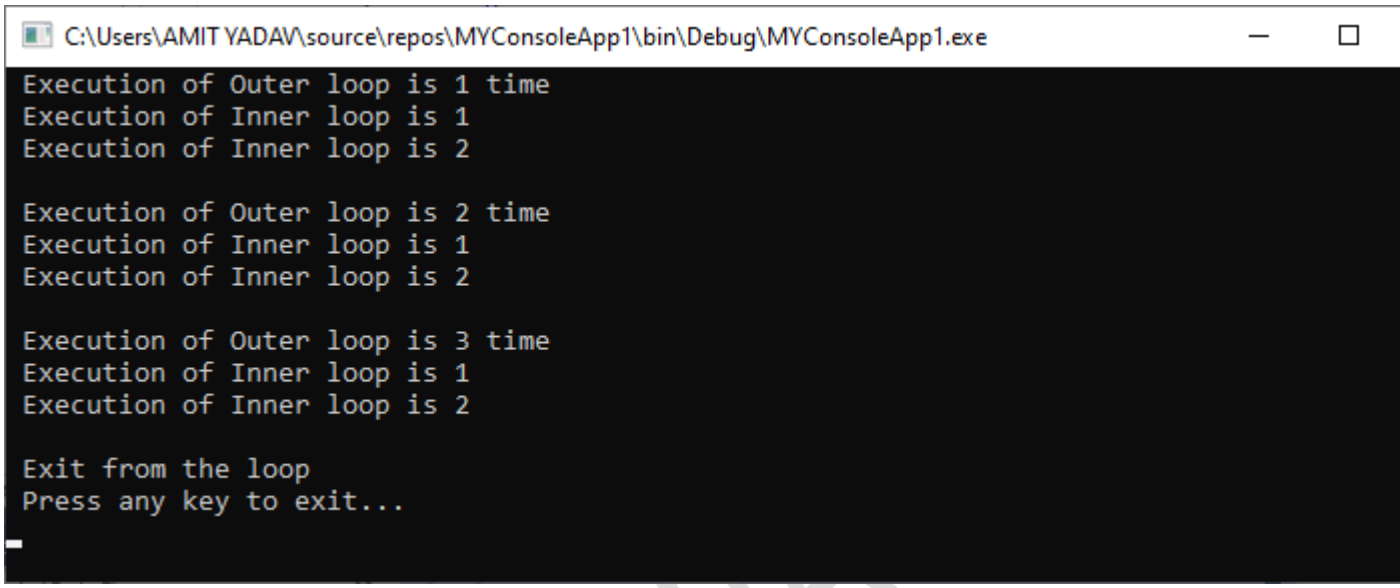
Example 2: Write a simple program to use the Do While loop Statement in VB.NET.

Nest_Do_While.vb

```
1. Imports System
2. Module Nest_Do_While
3.     Sub Main()
4.         ' Declare i and j as Integer variable
5.         Dim i As Integer = 1
6.         Do
7.             ' Outer loop statement
8.             Console.WriteLine(" Execution of Outer loop is {0}", i & " times")
9.
10.            Dim j As Integer = 1
11.
12.            Do
13.                'Inner loop statement
14.                Console.WriteLine(" Execution of Inner loop is {0}", j)
15.                j = j + 1 ' Increment Inner Counter variable by 1
16.            Loop While j < 3
17.
18.            Console.WriteLine()
19.            i = i + 1 ' Increment Outer Counter variable by 1
20.        Loop While i < 4
21.        Console.WriteLine(" Exit from the loop")
22.        Console.WriteLine(" Press any key to exit...")
23.        Console.ReadKey()
24.    End Sub
```

25. End Module

Output:



In the above example, each iteration of the outer loop also executes the inner loop repeatedly until the inner condition becomes false. When the condition of the outer loop becomes false, the execution of the outer and inner loop will be terminated.

For Next Loop

A **For Next loop** is used to repeatedly execute a sequence of code or a block of code until a given condition is satisfied. A For loop is useful in such a case when we know how many times a block of code has to be executed. In VB.NET, the For loop is also known as For Next Loop.

Syntax

1. For variable_name As [DataType] = start To end [Step step]
2. [Statements to be executed]
3. Next

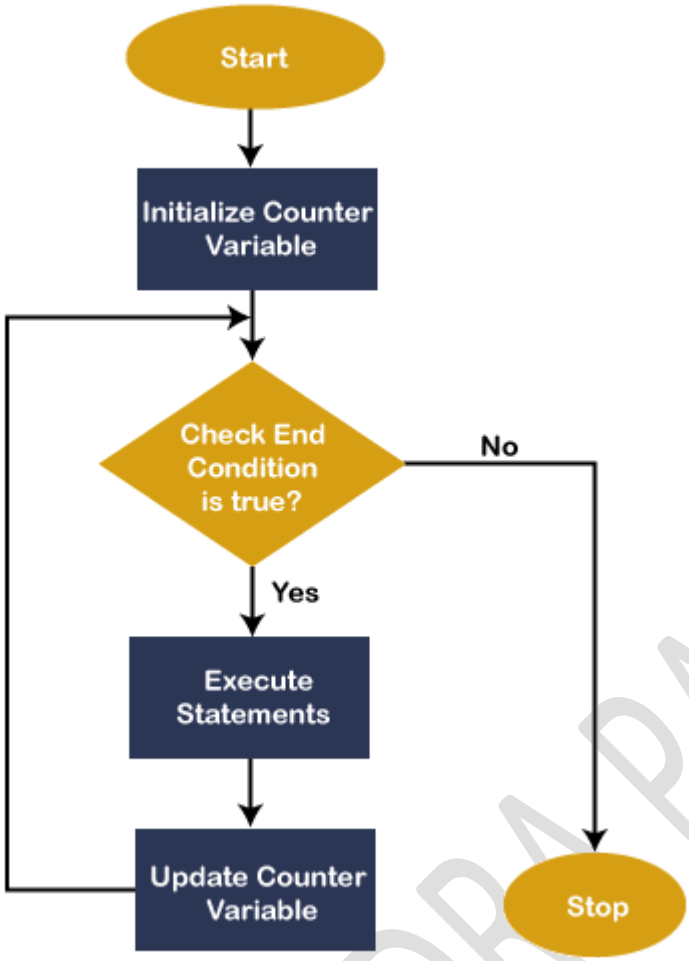
Let's understand the For Next loop in detail.

- **For**: It is the keyword that is present at the beginning of the definition.
- **variable_name**: It is a variable name, which is required in the For loop Statement. The value of the variable determines when to exit from the **For-Next loop,** and the value should only be a numeric.

- **[Data Type]**: It represents the Data Type of the **variable_name**.
- **start To end**: The **start** and **end** are the two important parameters representing the initial and final values of the **variable_name**. These parameters are helpful while the execution begins, the initial value of the variable is set by the start. Before the completion of each repetition, the variable's current value is compared with the end value. And if the value of the variable is less than the end value, the execution continues until the variable's current value is greater than the end value. And if the value is exceeded, the loop is terminated.
- **Step**: A step parameter is used to determine by which the **counter** value of a variable is increased or decreased after each iteration in a program. If the counter value is not specified; It uses 1 as the default value.
- **Statements**: A statement can be a single statement or group of statements that execute during the completion of each iteration in a loop.
- **Next**: In VB.NET a **Next** is a keyword that represents the end of the **For loop's**

Flowchart of For Next loop

The following flowchart represents the functioning of the For Next loop in the VB.NET programming language.



In the above flow chart, the first step is to initialize the variable name with the start value. And then, the value of the variable will be compared to the **end expression** or value. If the condition is true, the control enters the loop body and executes the statements. After that, the value of a variable will be **automatically incremented** by the compiler. Upon **completion** of each iteration, the current value of a variable will be **again compared** to the end expression. If the condition is not true, the controlled **exit** from the loop.

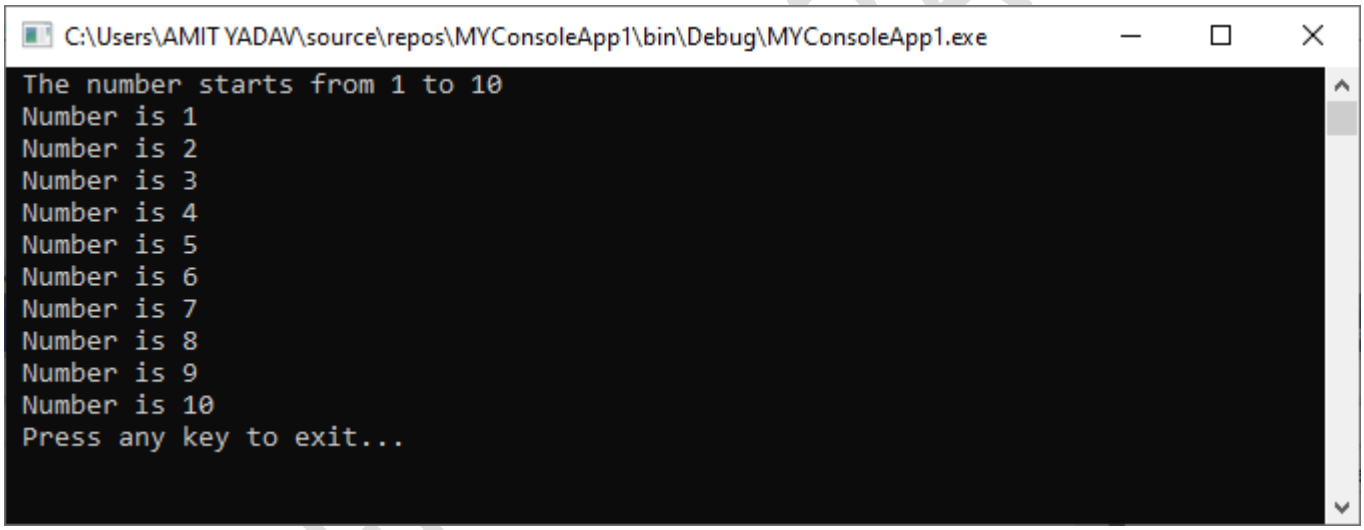
Example 1. Write a simple program to print the number from 1 to 10 using the For Next loop.

Number.vb

```
1. Imports System
2. Module Number
3.     Sub Main()
4.         ' It is a simple print statement, and 'vbCrLf' is used to jump in the next line.
5.         Console.Write(" The number starts from 1 to 10 " & vbCrLf)
```

```
6.          ' declare and initialize variable i
7.          For i As Integer = 1 To 10 Step 1
8.              ' if the condition is true, the following statement will be executed
9.              Console.WriteLine(" Number is {0} ", i)
10.             ' after completion of each iteration, next will update the variable counter
11.         Next
12.         Console.WriteLine(" Press any key to exit... ")
13.         Console.ReadKey()
14.     End Sub
15. End Module
```

Output:



In the above example, we have initialized an integer variable **i** with an initial value 1. The For loop will continuously execute its body until the value of **i** is smaller or equal to 10. After each iteration, the value of **i** is automatically increased with '**Step** 1'. If the value of i reached 10, the loop would be **terminated** and control transfer to the **Main**() function.

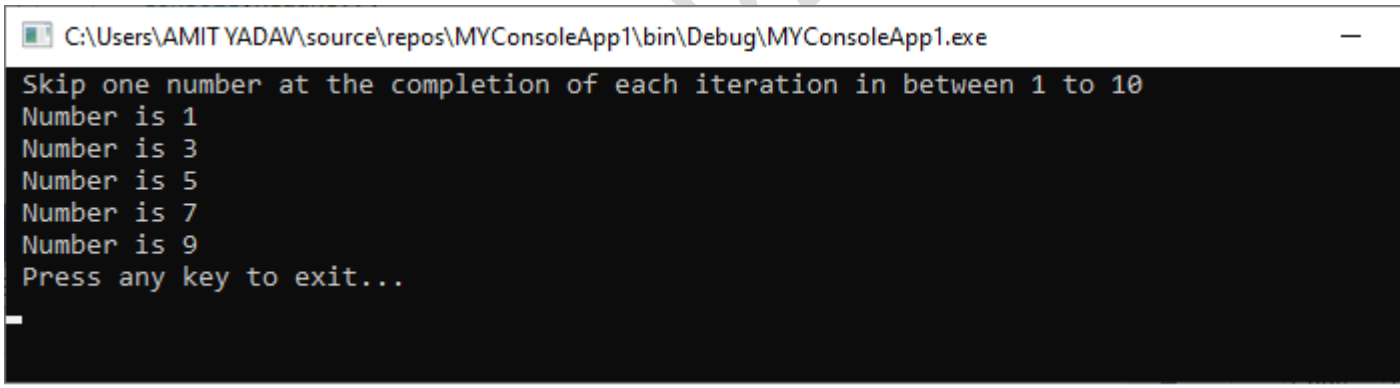
Further, we can change the **Step** in For Next loop. Write the following program to skip the number is 2.

Number.vb

```
1. Imports System
2. Module Number
3.     Sub Main()
4.         ' declaration of variable i
```

```
5.          Dim i As Integer
6.          Console.Write(" Skip one number at the completion of each iteration in between 1 to 10
    " & vbCrLf)
7.          ' initialize i to 1 and declare Step to 2 for skipping a number
8.          For i = 1 To 10 Step 2
9.              ' if condition is true, it skips one number
10.             Console.WriteLine(" Number is {0} ", i)
11.             ' after completion of each iteration, next will update the variable counter to step 2
12.         Next
13.         Console.WriteLine(" Press any key to exit... ")
14.         Console.ReadKey()
15.     End Sub
16. End Module
```

Output:



As we can see in the above output, the value of the variable **i is initialized with 1, and the value of i is skipped** by '**Step 2**' **in the loop for each iteration** to print the skipped number from 1 to 10.

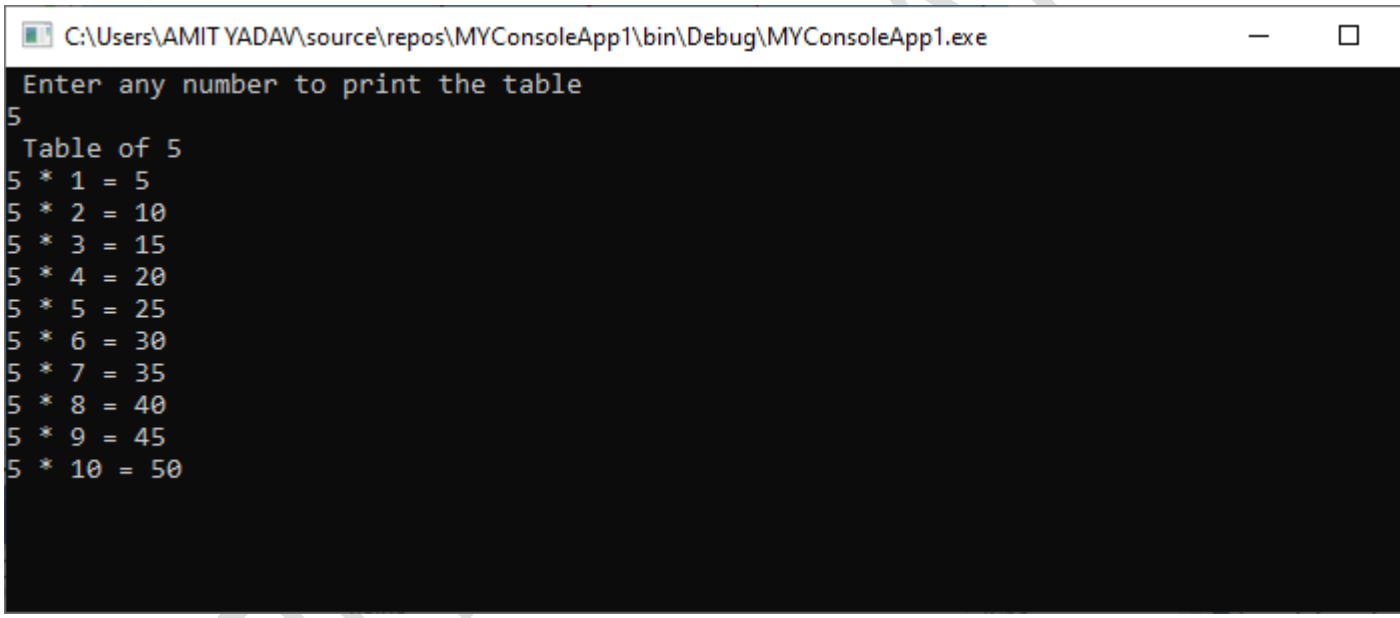
Example 2: Write a simple program to print a table in the VB.NET.

Table.vb

```
1.  Imports System
2.  Module Table
3.      Sub Main()
4.          'declaration of i and num variable
5.          Dim i, num As Integer
6.          Console.WriteLine(" Enter any number to print the table")
```

```
7.          num = Console.ReadLine() ' accept a number from the user
8.          Console.WriteLine(" Table of " & num)
9.          'define for loop condition, it automatically initialize step to 1
10.         For i = 1 To 10
11.             Console.WriteLine(num & " * " & i & " = " & i * num)
12.         Next
13.         Console.ReadKey()
14.     End Sub
15. End Module
```

Output:



Nested For Next Loop in VB.NET

In VB.NET, when we write one For loop inside the body of another For Next loop, it is called **Nested For Next** loop.

Syntax:

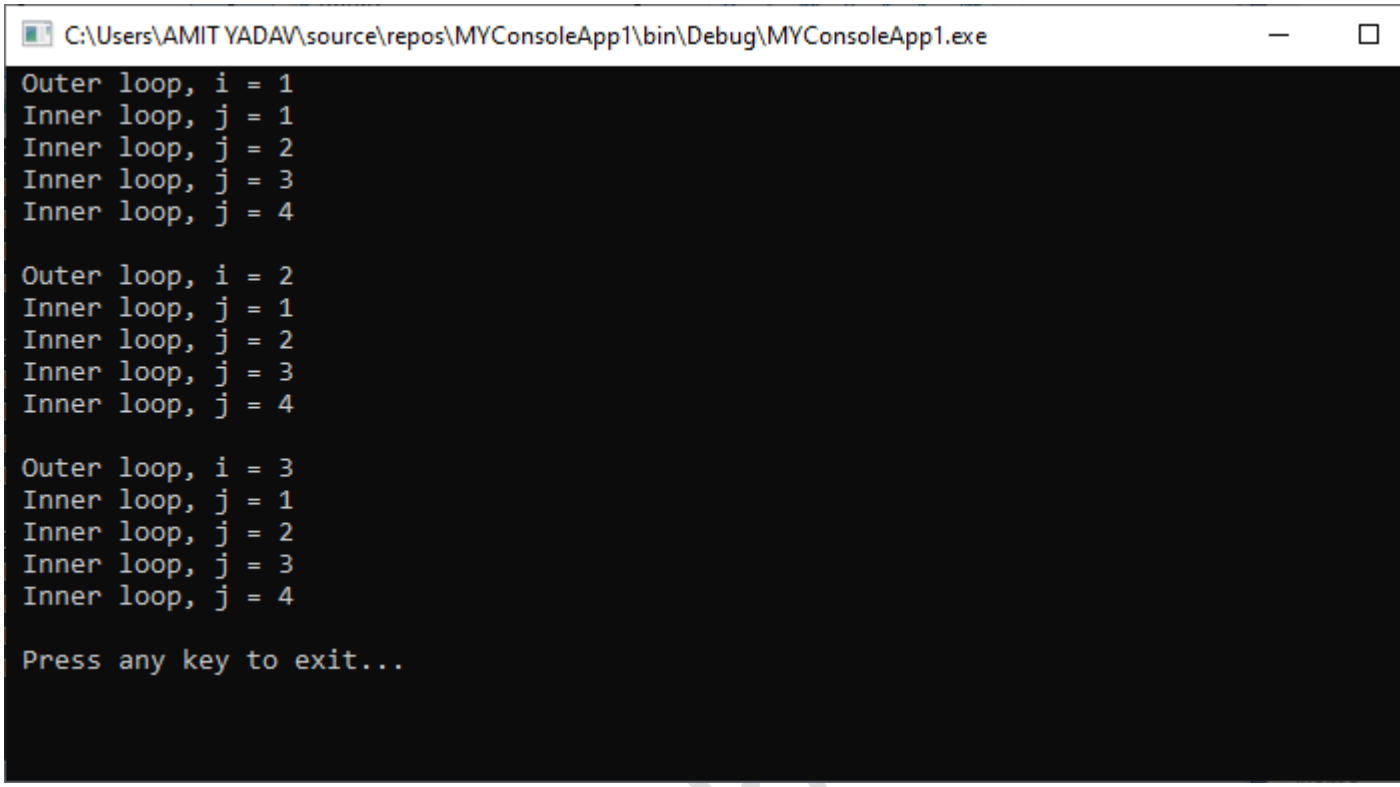
```
1. For variable_name As [Data Type] = start To end [ Step step ]
2.     For variable_name As [Data Type] = start To end [ Step step ]
3.         [ inner loop statements ]
4.     Next
5.     [ Outer loop statements ]
6. Next
```

Following is the example of Nested For Next loop in VB.NET.

Nested_loop.vb

```
1. Imports System
2. Module Nested_loop
3.     Sub Main()
4.         Dim i, j As Integer
5.         For i = 1 To 3
6.             'Outer loop
7.             Console.WriteLine(" Outer loop, i = {0}", i)
8.             'Console.WriteLine(vbCrLf)
9.
10.            'Inner loop
11.            For j = 1 To 4
12.                Console.WriteLine(" Inner loop, j = {0}", j)
13.            Next
14.            Console.WriteLine()
15.        Next
16.        Console.WriteLine(" Press any key to exit...")
17.        Console.ReadKey()
18.    End Sub
19. End Module
```

Output:



In the above example, at each iteration of the outer loop, the inner loop is repeatedly executed its entire cycles until the condition is not satisfied.

Example 2: Write a program to print a pattern in VB.NET.

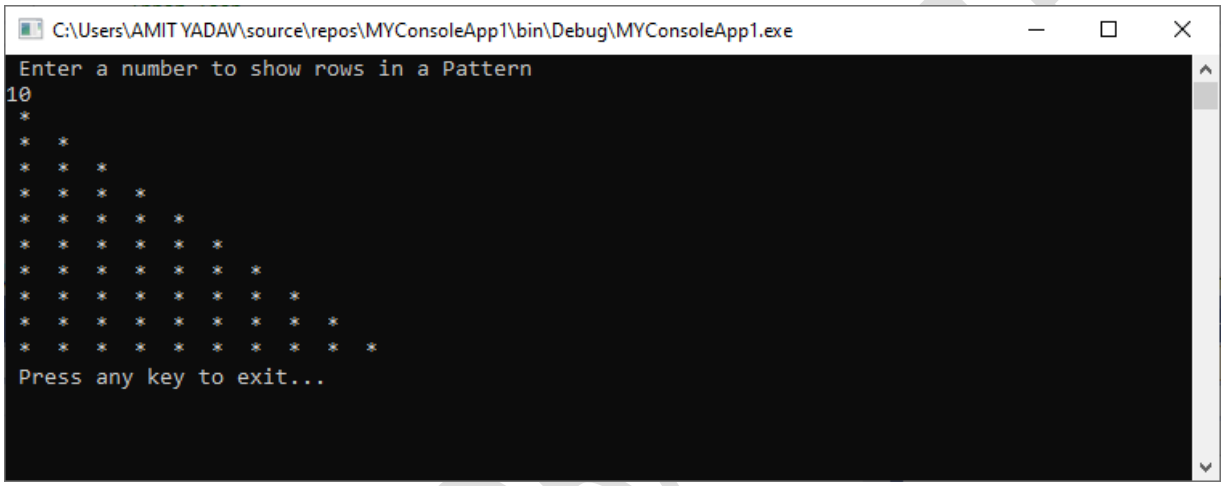
Pattern.vb

```
1.  Imports System
2.  Module Pattern
3.      Sub Main()
4.          Dim i, n, j As Integer
5.          Console.WriteLine(" Enter a number to show rows in a Pattern")
6.          ' take a number from user
7.          n = Console.ReadLine()
8.
9.          'Outer loop
10.         For i = 1 To n
11.             'Inner loop
12.             'value of j should be less than i
13.             For j = 1 To i
14.                 Console.Write(" * ")
15.             Next
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
16.             Console.WriteLine("")
17.         Next
18.         Console.WriteLine(" Press any key to exit...")
19.         Console.ReadKey()
20.     End Sub
21. End Module
```

Output:



VB.NET For Each Loop

In the VB.NET, **For Each loop** is used to iterate block of statements in an array or collection objects. Using For Each loop, we can easily work with collection objects such as lists, arrays, etc., to execute each element of an array or in a collection. And when iteration through each element in the array or collection is complete, the control transferred to the next statement to end the loop.

Syntax:

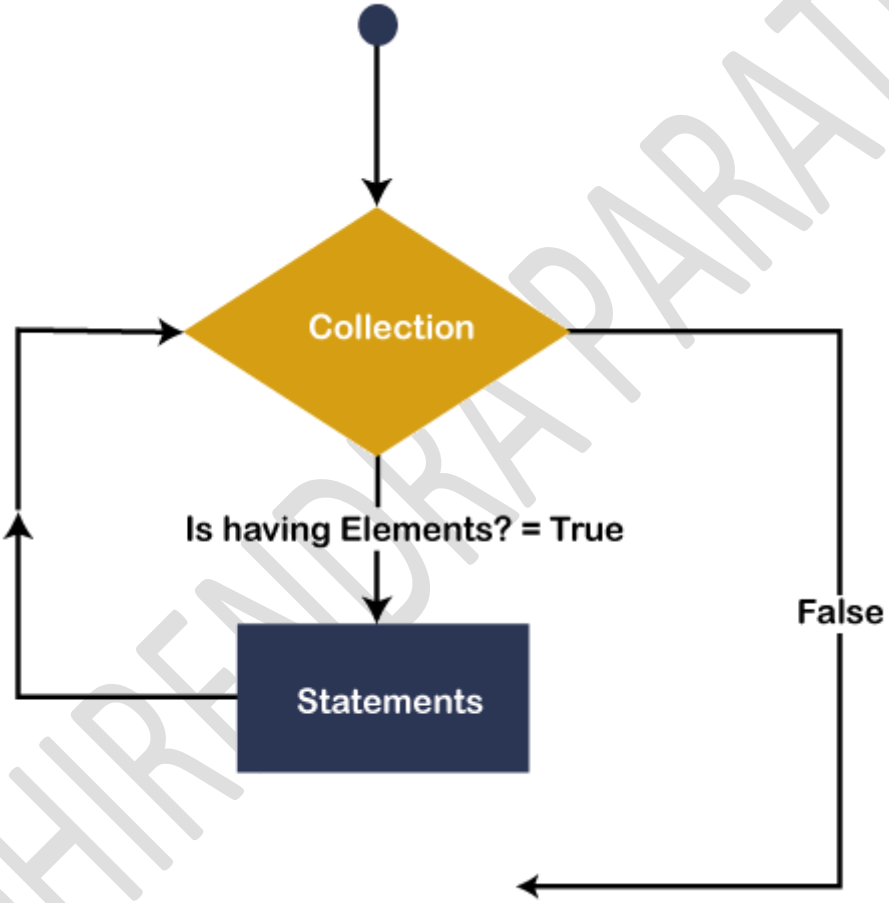
```
1. For Each var_name As [ DataType ] In Collection_Object
2. [ Statements to be executed]
3. Next
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

For Each loop is used to read each element from the collection object or an array. The **Data Type** represents the type of the variable, and **var_name** is the name of the variable to access elements from the **array** or **collection object** so that it can be used in the body of For Each loop.

Flowchart of For Each loop

The following flowchart represents the For Each Next loop's functioning to iterate through array elements in the VB.NET programming language.



The first step is to initialize an **array** or collection object to execute each element of the array with the help of **variables** in For Each loop. A variable is used in For Each loop to checks whether the **element** is available or not. If the element is available in the collection object, the For Each block will be executed until the **condition** remains true. After the execution of each element of an array, the control transfer to the end statement.

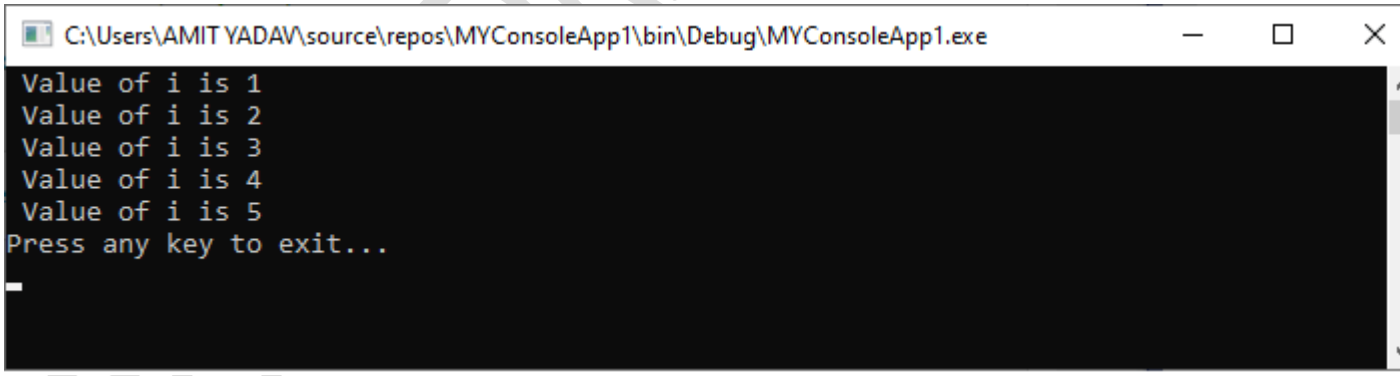
Examples of For Each Loop

Write a simple program to understand the uses of For Each Next loop in VB.NET.

For_Each_loop.vb

```
1. Imports System
2. Module For_Each_loop
3.     Sub Main()
4.         'declare and initialize an array as integer
5.         Dim An_array() As Integer = {1, 2, 3, 4, 5}
6.         Dim i As Integer 'Declare i as Integer
7.
8.         For Each i In An_array
9.             Console.WriteLine(" Value of i is {0}", i)
10.        Next
11.        Console.WriteLine("Press any key to exit...")
12.        Console.ReadLine()
13.    End Sub
14. End Module
```

Output:



In the above example, we create an integer array with the name **An_array ()**, and For Each loop is used to iterate each element of the array with the help of defined **variable 'i'**.

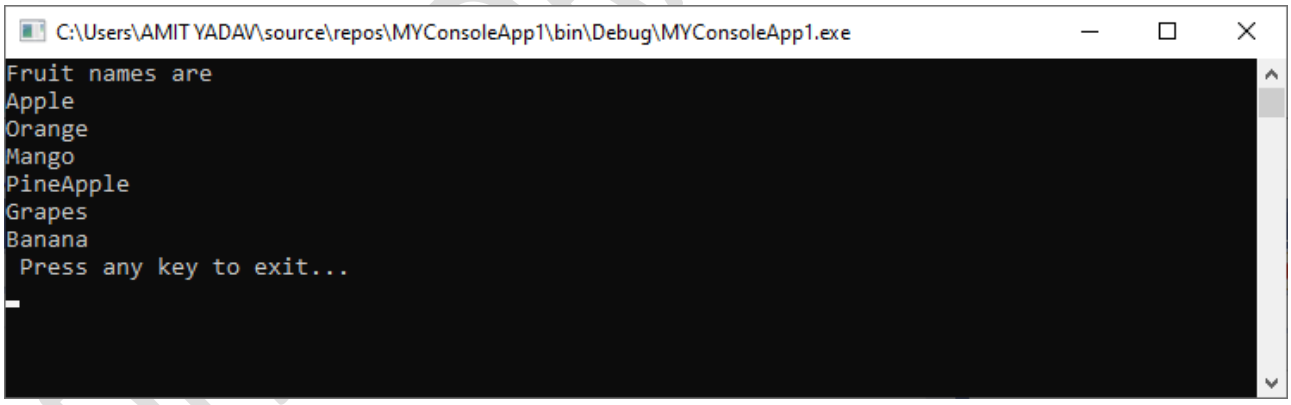
Example 2: Write a simple program to print fruit names using For Each loop in VB.NET.

For_each.vb

```
1. Imports System
2. Module For_each
```

```
3.      Sub Main()
4.          'Define a String array
5.          Dim str() As String
6.
7.          'Initialize all element of str() array
8.          str = {"Apple", "Orange", "Mango", "PineApple", "Grapes", "Banana"}
9.
10.         Console.WriteLine("Fruit names are")
11.
12.         'Declare variable name as fruit
13.         For Each fruit As String In str
14.             Console.WriteLine(fruit)
15.         Next
16.         Console.WriteLine(" Press any key to exit...")
17.         Console.ReadKey()
18.     End Sub
19. End Module
```

Output:



In this example, **str()** is a String type array that defines different fruits names. And **fruit** is the name of a variable that is used to iterate each element of the **str()** array using **For Each** loop in the program. If all the element is read, control passes to the **Main()** function to terminate the program.

VB.NET While End Loop

The **While End loop** is used to execute blocks of code or statements in a program, as long as the given **condition** is true. It is useful when the number of executions of

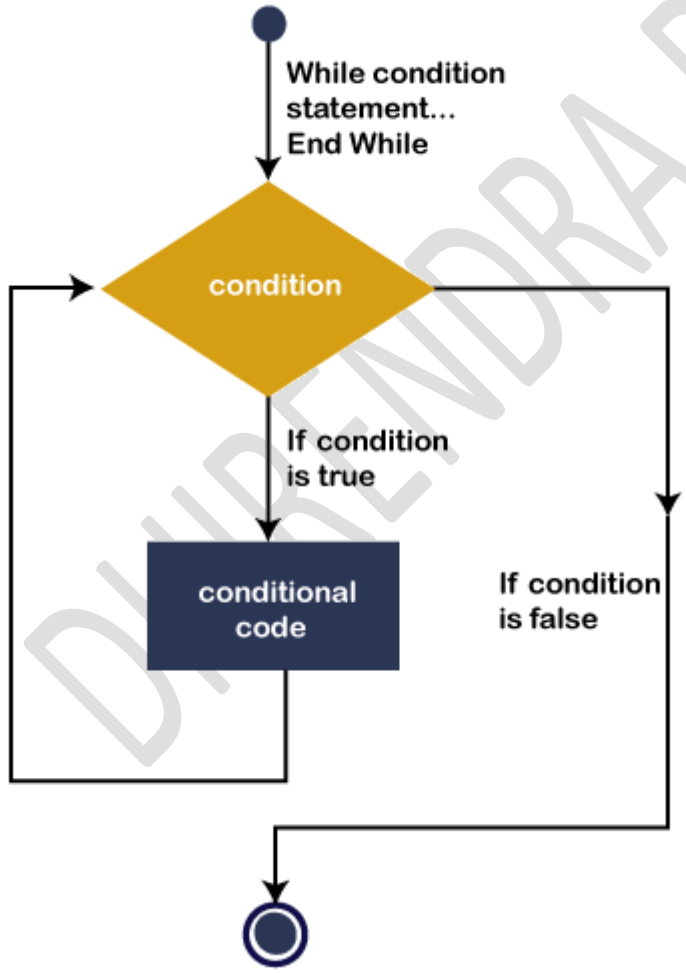
a block is not known. It is also known as an **entry-controlled loop** statement, which means it initially checks all loop conditions. If the condition is true, the body of the while loop is executed. This process of repeated execution of the body continues until the condition is not false. And if the condition is false, control is transferred out of the loop.

Syntax:

1. While [condition]
2. [Statement to be executed]
3. End While

Here, **condition** represents any **Boolean condition,** and if the logical condition is true, the **single or block of statements** define inside the body of the while loop is executed.

Flow Diagram of the While End Loop in VB.NET



As we know, the **While End loop** is an **entry-controlled** loop used to determine if the condition is true, the statements defined in the body of the loop are executed,

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

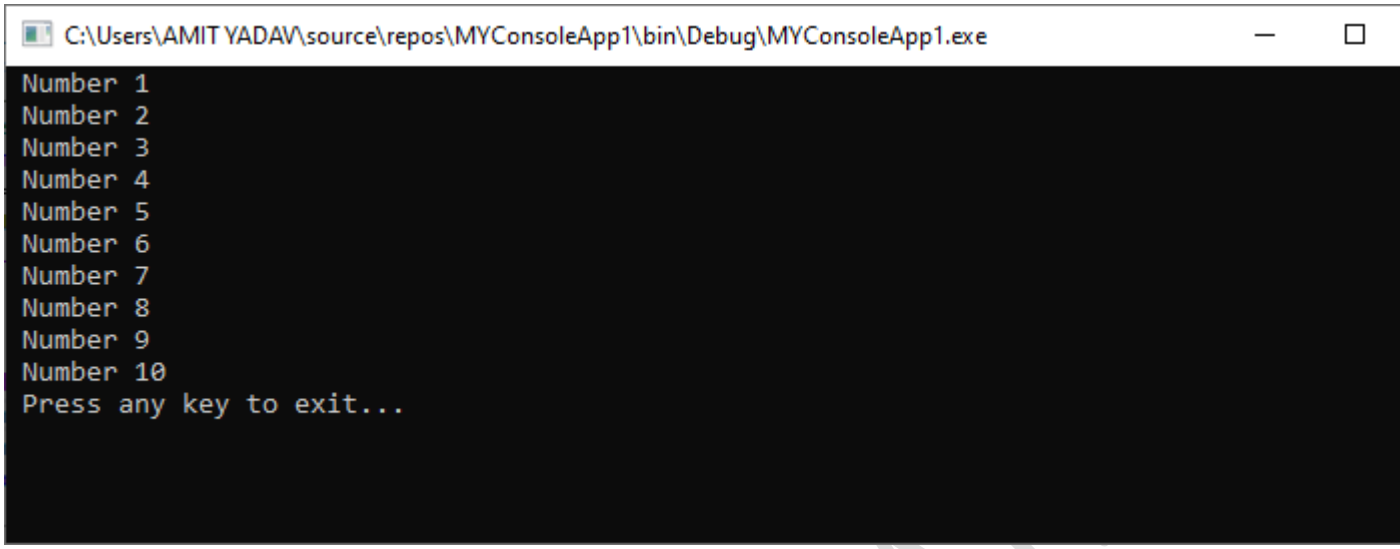
and the execution process continues till the **condition** is satisfied. Furthermore, after each iteration, the value of the counter variable is incremented. It again checks whether the defined condition is true; And if the condition is again true, the body of the While loop is executed. And when the condition is not true, the control transferred to the end of the loop.

Example: Write a simple program to print the number from 1 to 10 using while End loop in VB.NET.

while_number.vb

```
1. Imports System
2. Module while_number
3.     Sub Main()
4.         'declare x as an integer variable
5.         Dim x As Integer
6.         x = 1
7.         ' Use While End condition
8.         While x <= 10
9.             'If the condition is true, the statement will be executed.
10.            Console.WriteLine(" Number {0}", x)
11.            x = x + 1 ' Statement that change the value of the condition
12.        End While
13.        Console.WriteLine(" Press any key to exit...")
14.        Console.ReadKey()
15.    End Sub
16. End Module
```

Output:



In the above example, while loop executes its body or statement up to the defined state (i <= 10). And when the value of the variable i is 11, the defined condition will be false; the loop will be terminated.

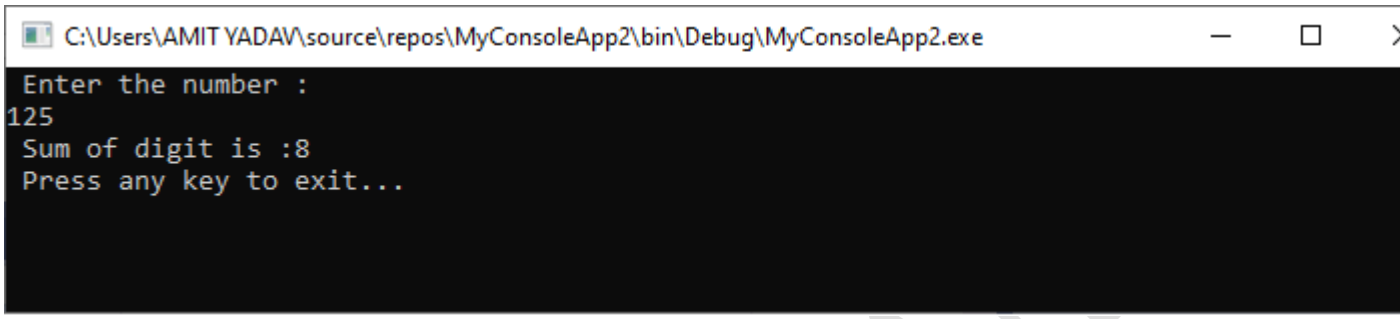
Example 2: Write a program to print the sum of digits of any number using while End loop in VB.NET.

Total_Sum.vb

```
1.  Public Class Total_Sum ' Create a Class Total_sum
2.      Shared Sub Main()
3.          'Declare an Integer variable
4.          Dim n, remainder, sum As Integer
5.          sum = 0
6.
7.          Console.WriteLine(" Enter the number :")
8.          n = Console.ReadLine()  ' Accept a number from the user
9.
10.         ' Use While loop and write given below condition
11.         While (n > 0)
12.             remainder = n Mod 10
13.             sum += remainder
14.             n = n / 10
15.         End While
16.         Console.WriteLine(" Sum of digit is :{0}", sum)
17.         Console.WriteLine(" Press any key to exit...")
18.         Console.ReadKey()
```

```
19.     End Sub
20. End Class
```

Output:



Nested While End Loop

In VB.NET, when we write a While End loop inside the body of another While End loop, it is called Nested While End loop.

Syntax

```
1. While (condition -1)
2. // body of the outer while loop
3. While (condition -2)
4.                 // body of inner while loop
5. End While
6. // body of the outer loop
7. End While
```

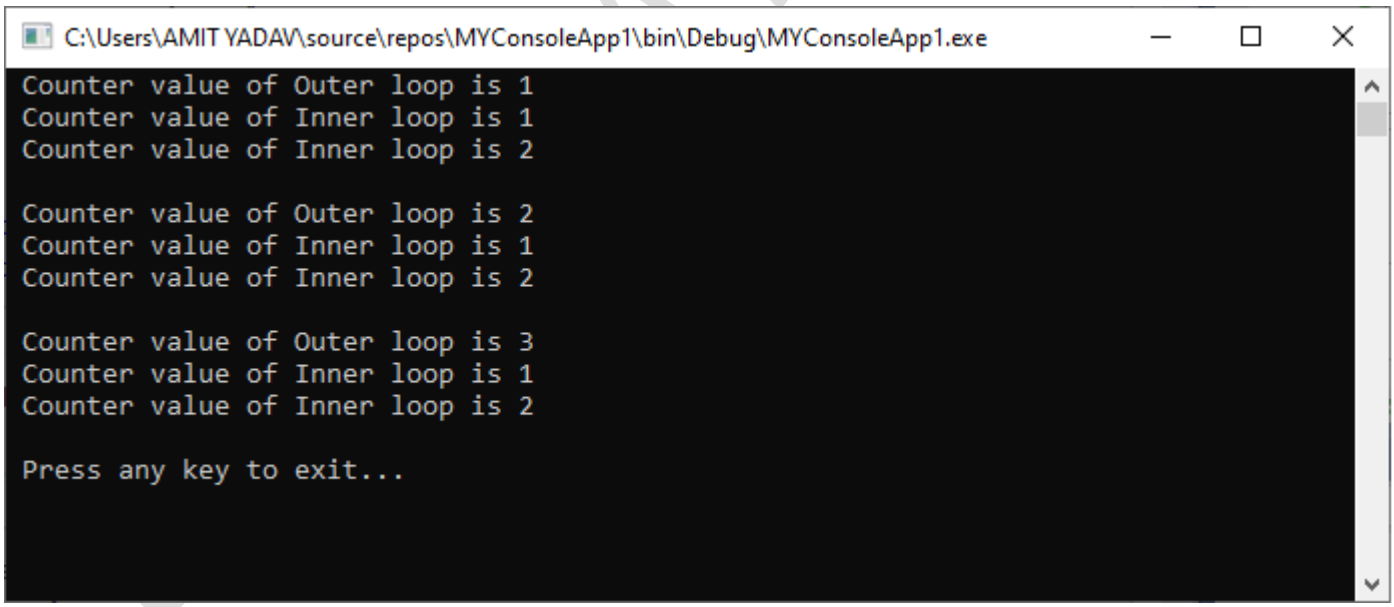
Write a program to understand the Nested While End loop in VB.NET programming.

Nest_While.vb

```
1. Imports System
2. Module Nest_While
3.     Sub Main()
4.         ' Declare i and j as Integer variable
5.         Dim i As Integer = 1
6.
7.
8.         While i < 4
9.             ' Outer loop statement
```

```
10.         Console.WriteLine(" Counter value of Outer loop is {0}", i)
11.         Dim j As Integer = 1
12.
13.         While j < 3
14.             'Inner loop statement
15.             Console.WriteLine(" Counter value of Inner loop is {0}", j)
16.             j = j + 1 ' Increment Inner Counter variable by 1
17.         End While
18.         Console.WriteLine() 'print space
19.         i = i + 1   ' Increment Outer Counter variable by 1
20.     End While
21.     Console.WriteLine(" Press any key to exit...")
22.     Console.ReadKey()
23.  End Sub
24. End Module
```

Output:



In the above example, at each iteration of the outer loop, the inner loop is repeatedly executed its entire cycles until the inner condition is not satisfied. And when the condition of the outer loop is false, the execution of the outer and inner loop is terminated.

VB.NET Exit Statement

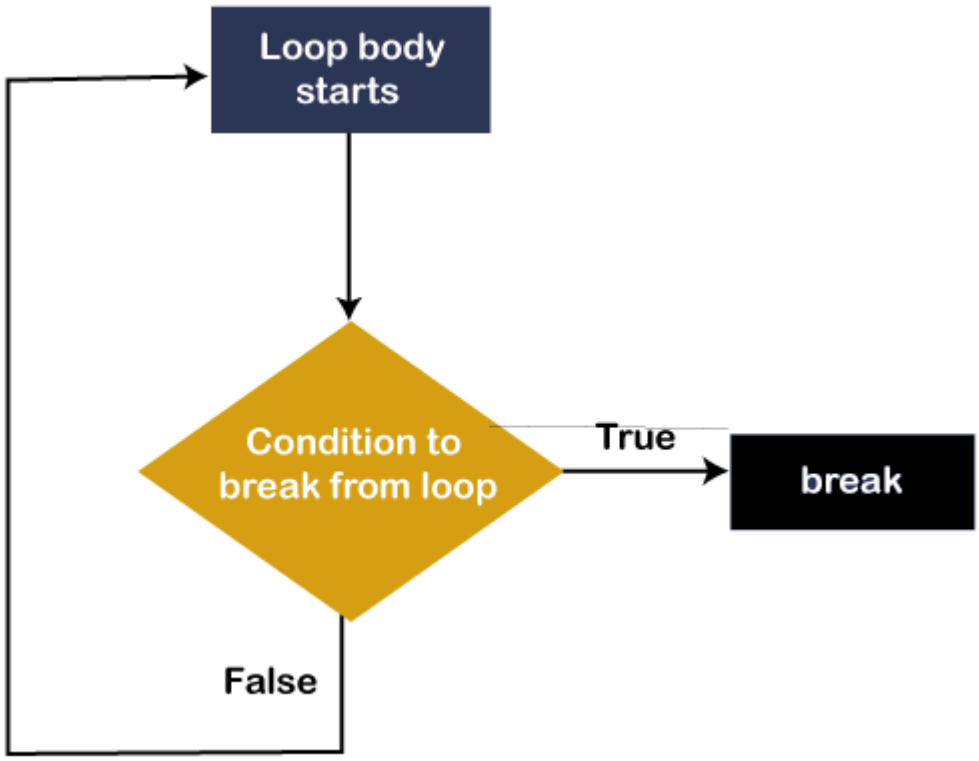
In VB.NET, the **Exit statement** is used to terminate the loop (**for, while, do, select case, etc.**) or exit the loop and pass control immediately to the next statement of the termination loop. Furthermore, the Exit statement can also be used in the **nested loop** to stop or terminate the execution of the inner or outer loop at any time, depending on our requirements.

Syntax

1. Exit { Do | For | Function | Property | Select | Sub | Try | While }

The flow of Exit Statement

Following is the diagrammatical representation of Exit Statement in VB.NET programming language.



Generally, the Exit statement is written along with a condition. If the **Exit** condition is **true** inside the loop, it exits from the loop and control transfer to the next statement, followed by the loop. And if the **Exit** condition fails **for the first time**, it will not check any statement inside the loop and **terminates** the program.

We will now see how to use the Exit statement in loops and Select case statements to finish the program's execution in the VB.NET programming language.

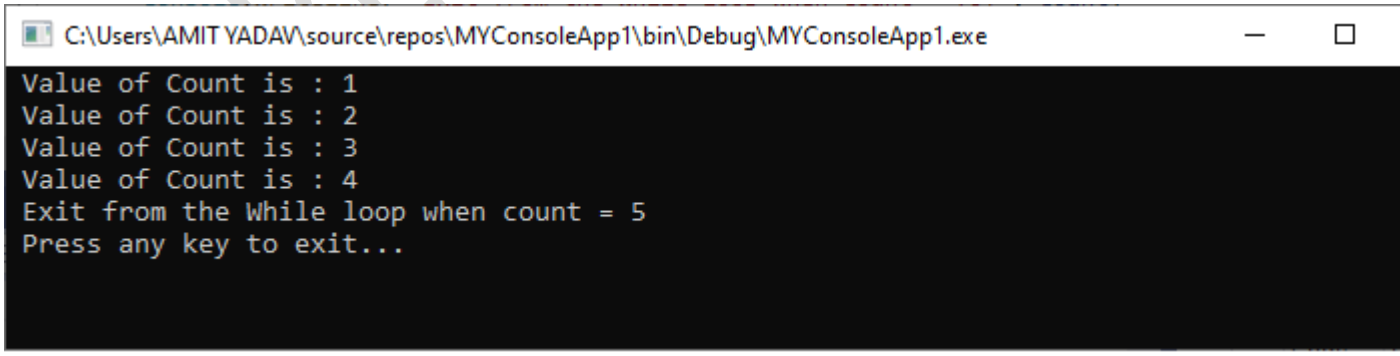
Use of Exit statement in While End loop

Example 1: Write a simple program to use the Exit Statement in While End loop.

Exit_While.vb

```
1. Imports System
2. Module Exit_While
3.     Sub Main()
4.         ' Definition of count variable
5.         Dim count As Integer = 1
6.
7.         ' Execution of While loop
8.         While (count < 10)
9.             ' Define the Exit condition using If statement
10.            If count = 5 Then
11.                Exit While ' terminate the While loop
12.            End If
13.            Console.WriteLine(" Value of Count is : {0}", count)
14.            count = count + 1
15.        End While
16.        Console.WriteLine(" Exit from the While loop when count = {0}", count)
17.        Console.WriteLine(" Press any key to exit...")
18.        Console.ReadKey()
19.    End Sub
20. End Module
```

Output:



In the above example, the While End loop is continuously executed, its body until the given condition **While (count < 10)** is not satisfied. But when the **Exit** condition **(count = 5)** falls inside a while loop, the execution of the loop is automatically terminated, and control moves to the next part of the loop statement.

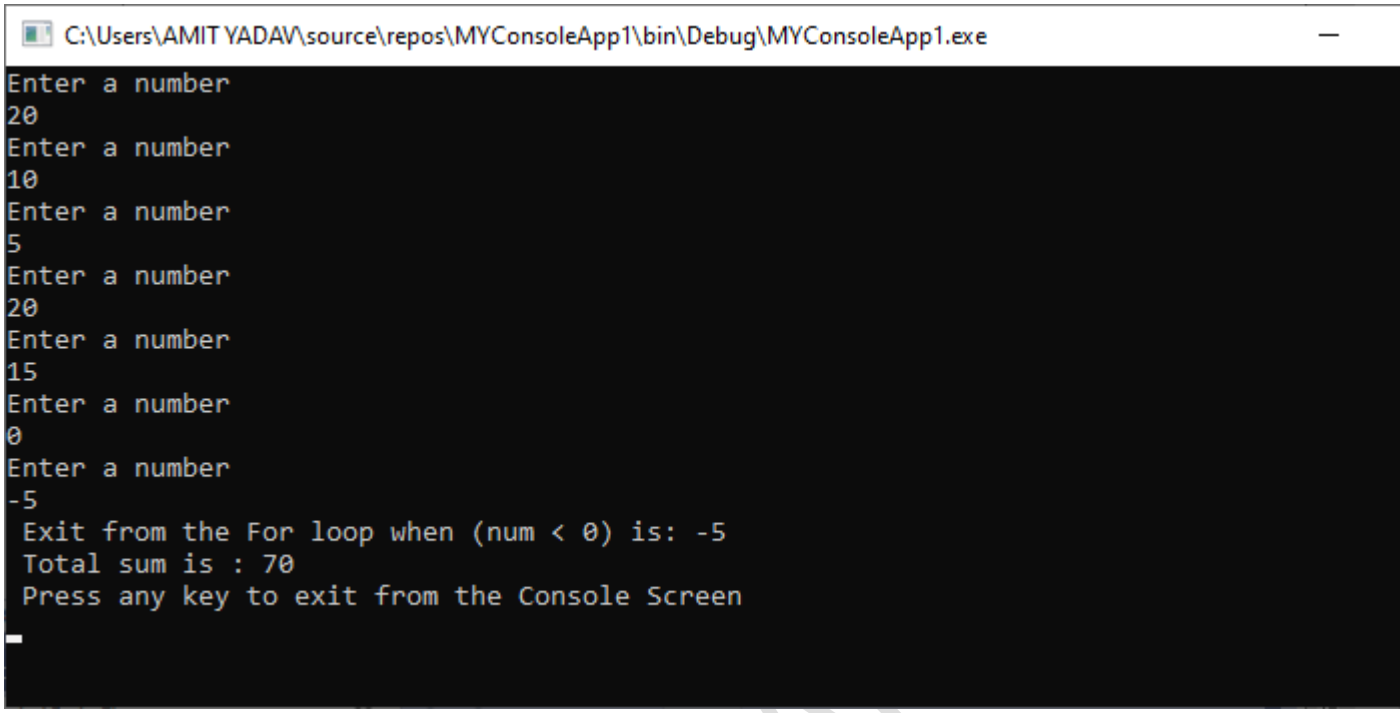
Use of Exit statement in For Next loop

Example 2: Write a program to calculate the sum of 10 numbers, and if a negative number is entered, the For Each loop ends.

```
1.  Imports System
2.  Module Exit_For
3.      Sub Main()
4.          'Definition of num variable
5.          Dim num As Integer
6.          Dim sum As Double = 0.0
7.
8.          'Execution of For loop
9.          For i As Integer = 1 To 10
10.             'Accept a number from the user
11.             Console.WriteLine("Enter a number : ")
12.             num = Console.ReadLine()
13.             ' If the user enters a negative number, the loop terminates
14.             If num < 0 Then
15.                 Exit For ' terminate the For loop
16.             End If
17.             sum += num
18.         Next
19.
20.         Console.WriteLine(" Exit from the For loop when (num < 0) is: {0}", num)
21.         Console.WriteLine(" Total sum is : {0}", sum)
22.         Console.WriteLine(" Press any key to exit from the Console Screen")
23.         Console.ReadKey()
24.     End Sub
25. End Module
```

Output:

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



The above program accepts a number from the user until it encounters a negative or less than 0 **(num < 0)**. And when there is a negative number, the **Exit** statement terminates the loop, and the control transfer to the next part of the loop.

Use of Exit statement in Do While loop

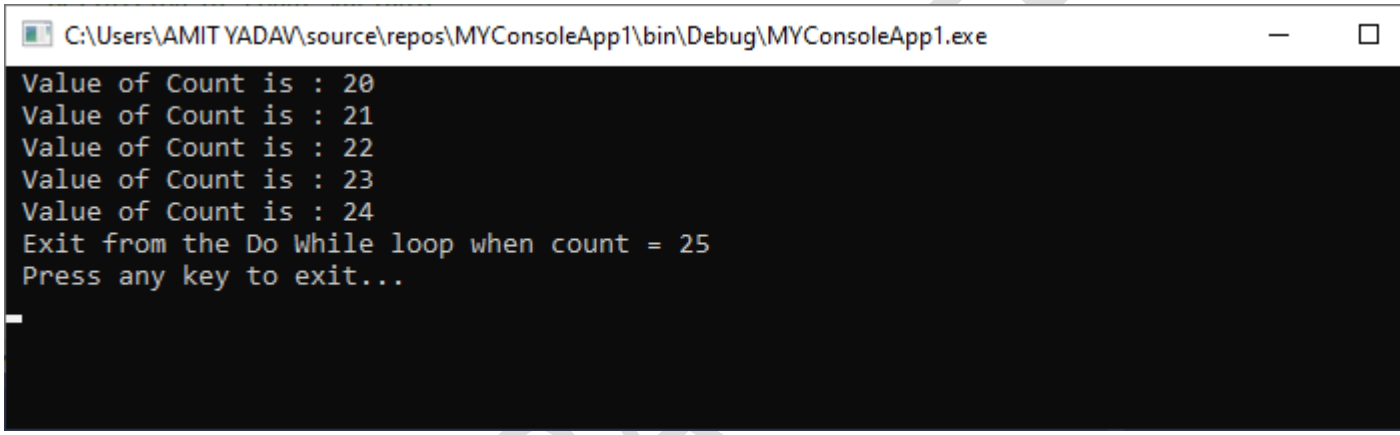
Example 3: Write a simple program to use the Exit Statement in Do While Loop.

Exit_Do_While.vb

```
1.  Imports System
2.  Module Exit_Do_While
3.      Sub Main()
4.          'Definition of the count variable
5.          Dim count As Integer = 20
6.
7.          ' Definition of Do While loop
8.          Do
9.              'Define the Exit condition using If statement.
10.
11.             If count = 25 Then
12.                 Exit Do ' terminate the Do While loop
13.             End If
14.             Console.WriteLine(" Value of Count is : {0}", count)
```

```
15.             count = count + 1
16.         Loop While (count < 50)
17.         Console.WriteLine(" Exit from the Do While loop when count = {0}", count)
18.         Console.WriteLine(" Press any key to exit...")
19.         Console.ReadKey()
20.     End Sub
21. End Module
```

Output:



In the above example, the Do While loop is continuously executed, its body until the given condition **While (count < 50)** is not satisfied. But when the **Exit** condition **(count = 25)** is encountered, the Do While loop is automatically terminated. The control is immediately transferred to the next statement, followed by the loop statements.

VB.NET Continue Statement

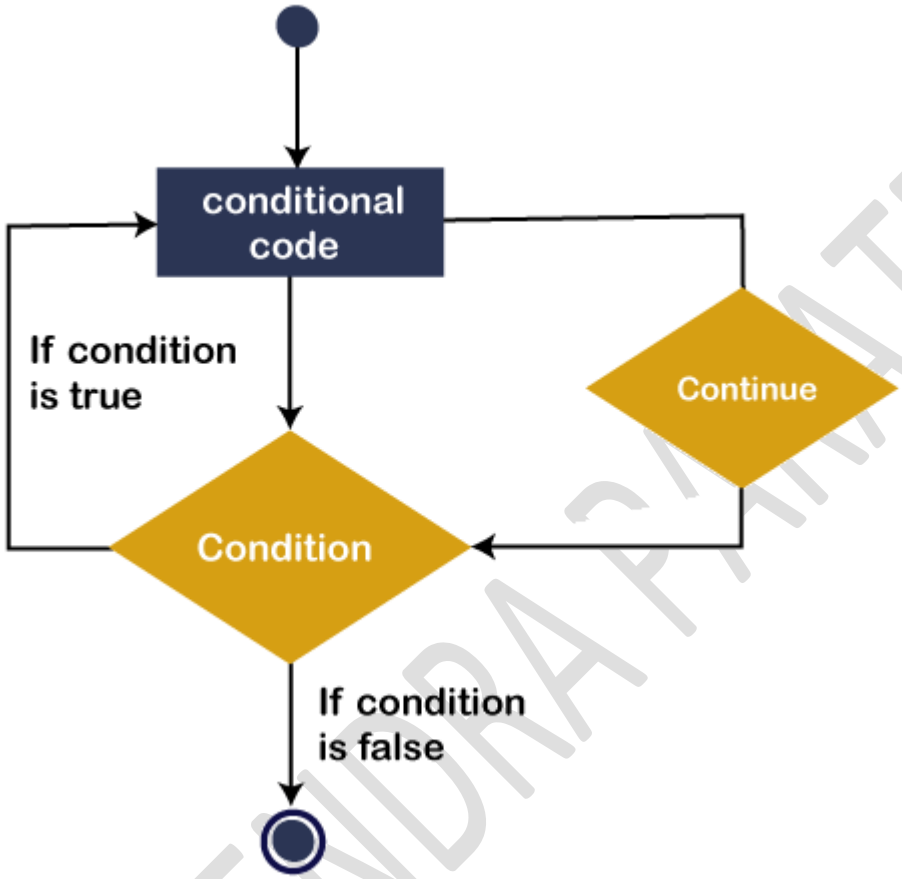
In **VB.NET,** the **continue** statement is used to skip the particular iteration of the loop and continue with the next iteration. Generally, the **continue** Statement is written inside the body of the For, While and Do While loop with a condition. In the previous section, we learned about the Exit Statement. The main difference between the **Exit** and a **Continue** Statement is that the Exit Statement is used to exit or terminate the loop's execution process. In contrast, the Continue Statement is used to **Skip** the particular iteration and **continue** with the next iteration **without** ending the loop.

Syntax:

```
1. Continue { Do | For | While }
```

Flow Diagram of Continue Statement

Following is the pictorial representation of the Continue Statement in VB.NET programming language.



In the above diagram, a **Continue** Statement is placed inside the loop to skip a particular statement or iteration. Generally, a continue statement is used with a **condition**. If the condition is **true**, it skips the particular iteration and immediately transfers the control to the beginning of the loop for the execution of the next iteration.

Now we will see how to use the Continue statement in the loop to skip the execution of the code and send the control to the beginning of the loop to execute further statements in the VB.NET programming language.

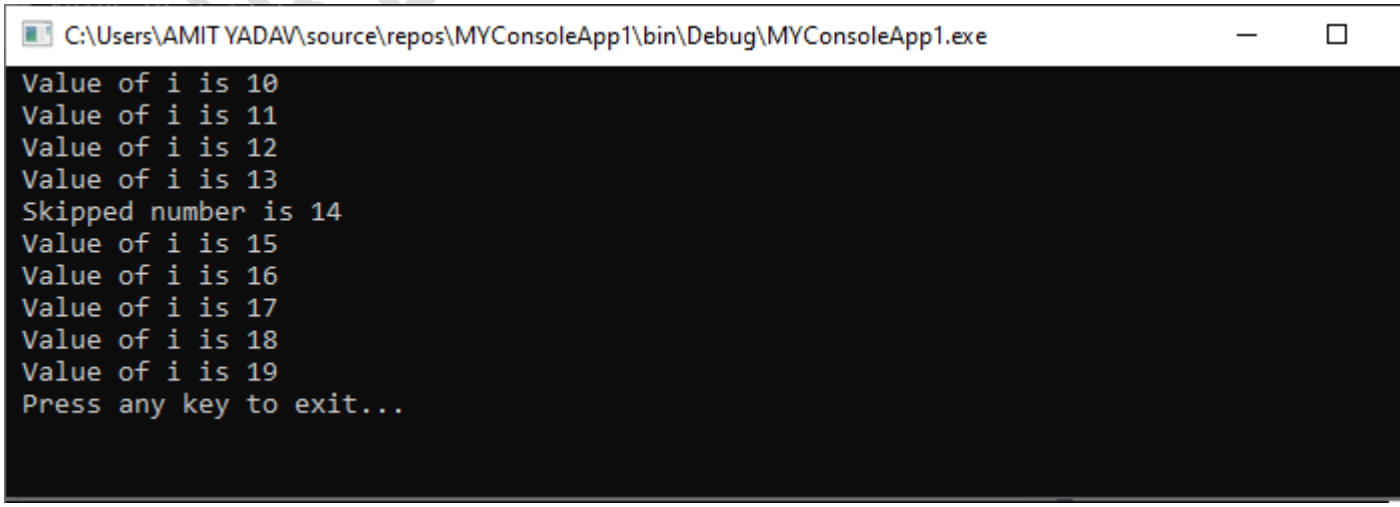
Use of Continue statement in While End loop

Example 1: Write a simple program to use the Continue Statement in While End loop.

Continue_While.vb

```
1. Imports System
2. Module Continue_While
3.     Sub Main()
4.         'Declaration and initialization of variable i
5.         Dim i As Integer = 10
6.
7.         'Define the While Loop Condition
8.         While i < 20
9.
10.            If i = 14 Then
11.                Console.WriteLine(" Skipped number is {0}", i)
12.                i += 1 ' skip the define iteration
13.                Continue While
14.            End If
15.            Console.WriteLine(" Value of i is {0}", i)
16.
17.            i += 1 ' Update the variable i by 1
18.        End While
19.        Console.WriteLine(" Press any key to exit...")
20.        Console.ReadKey()
21.    End Sub
22. End Module
```

Output:



In the above program, the While loop is continuously executed, their body until the given condition **(i < 20)** is met. But when the value of i is equal to 15, the Continue

statement is encountered. It skips the current execution and transfers the control to the beginning of the loop to display the next iteration.

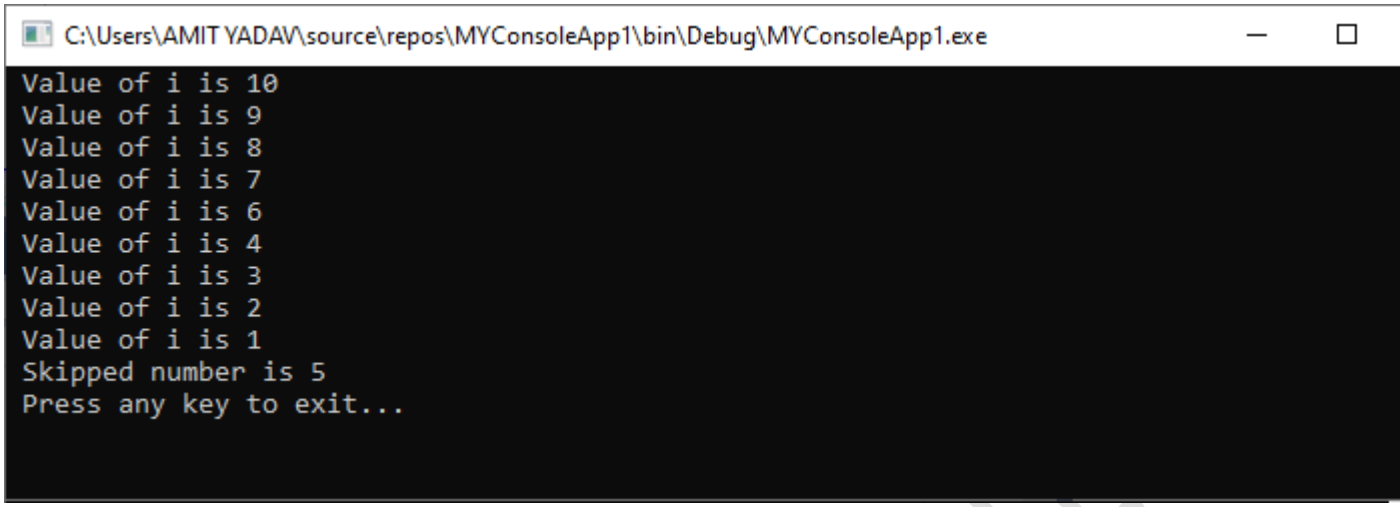
Use of Continue statement in For Next loop

Example 2: Write a simple program to print the number from 10 to 1 with Continue Statement in For Next loop.

Continue_For.vb

```
1. Imports System
2. Module Continue_For
3.     Sub Main()
4.         'Declaration and initialization of variable i, num
5.         Dim i As Integer = 10
6.         Dim num As Integer
7.         'Define the For Loop Condition
8.         For i = 10 To 1 Step -1
9.
10.            If i = 5 Then ' if i = 5, it skips the iteration
11.                num = i 'Assign the skip value to num variable
12.                Continue For ' Continue with Next Iteration
13.            End If
14.            Console.WriteLine(" Value of i is {0}", i)
15.
16.        Next
17.        Console.WriteLine(" Skipped number is {0}", num)
18.        Console.WriteLine(" Press any key to exit...")
19.        Console.ReadKey()
20.    End Sub
21. End Module
```

Output:



In the above program, the For loop is continuously decremented their body until the variable i is **1**. But when the value of **i** is equal to **5**, the Continue Statement is encountered. Then it skips the **current** execution and transfers control to the beginning of the loop to display the next iteration.

Use of Continue statement in Do While loop

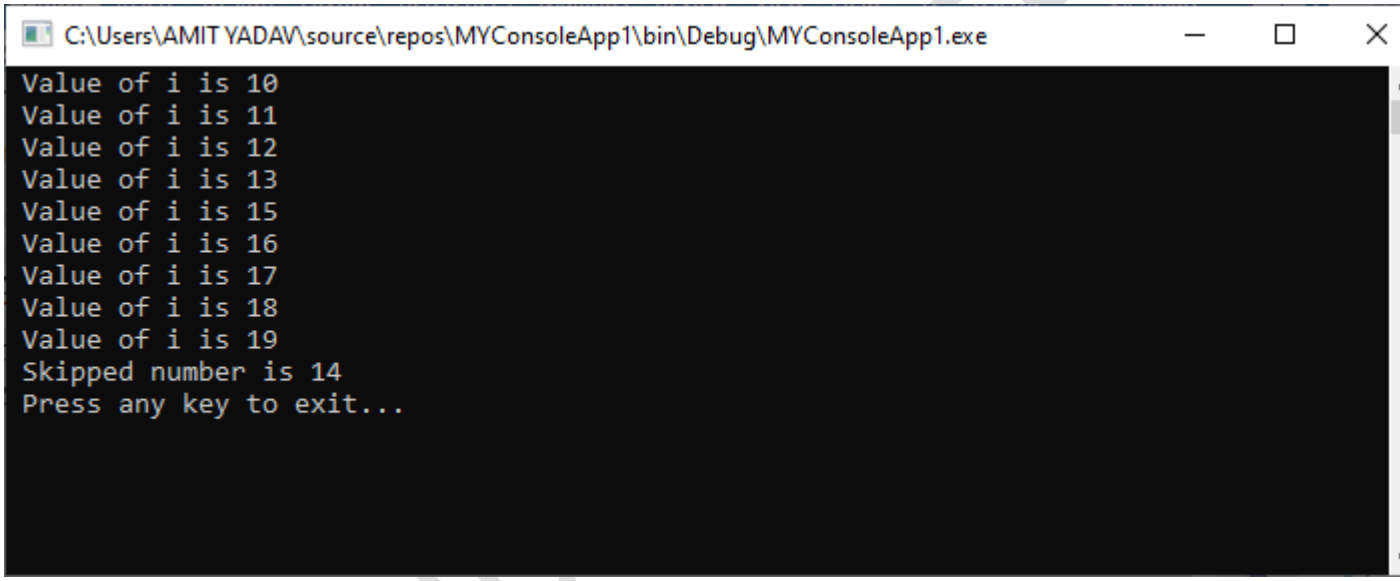
Example 3: Write a simple program to Understand the use of Continue Statement in Do While loop.

Continue_Do_While.vb

```
1.  Imports System
2.  Module Continue_Do_While
3.      Sub Main()
4.          'Declaration and initialization of local variable
5.          Dim i As Integer = 10, num As Integer
6.
7.          'Definition the Do While Loop
8.          Do
9.              If i = 14 Then
10.                 num = i
11.                 i += 1 ' skip the define iteration
12.                 Continue Do
13.             End If
14.             Console.WriteLine(" Value of i is {0}", i)
15.
16.             i += 1 ' Update the variable i by 1
```

```
17.         Loop While i < 20
18.         Console.WriteLine(" Skipped number is {0}", num)
19.
20.         Console.WriteLine(" Press any key to exit...")
21.         Console.ReadKey()
22.     End Sub
23. End Module
```

Output:



In the above program, the Do loop is continuously executed their body until the given condition **(i < 20)** is met. But when the value of i is equal to 15, the Continue Statement is encountered. It skips the current execution and transfers the control to the beginning of the loop to display the next iteration.

VB.NET GoTo Statement

In **VB.NET**, the GoTo statement is known as a jump statement. It is a control statement that transfers the flow of control to the specified label within the procedure. The GoTo statement uses labels that must be a valid identifier. The GoTo statement can be used in Select case, decision control statements, and loops.

Syntax:

```
1. GoTo label_1
```

Here, **GoTo** is a keyword, and **label_1** is a label used to transfer control to a specified label statement in a program.

Now we will see how to use the **GoTo** statement in the loop, select case, or decision-making statement to transfer control to the specified label statement in the VB.NET program.

Use of GoTo statement in If Else

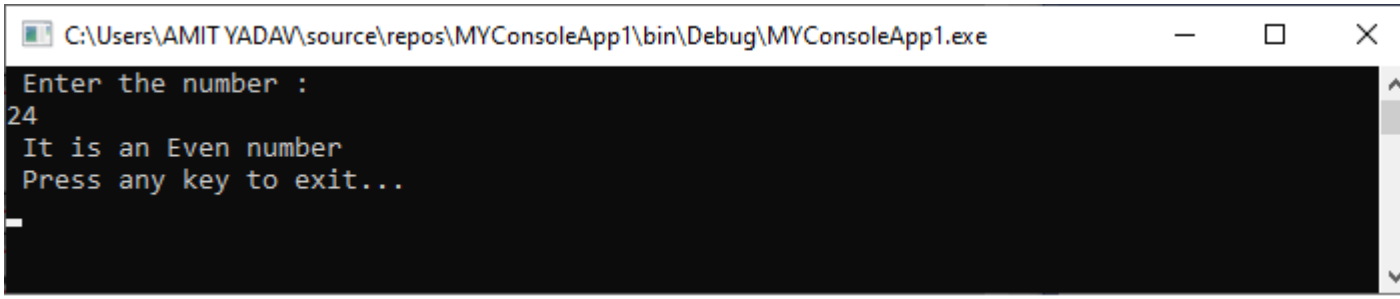
Example 1: Write a simple program to print whether the number is even or odd in GoTo Statement.

Goto_Statement.vb

```
1.  Imports System
2.  Module Goto_Statement
3.      Sub Main()
4.          'Declaration of local variable
5.          Dim num As Integer
6.          Console.WriteLine(" Enter the number :")
7.          num = Console.ReadLine ' Accept a number from the user
8.          If (num Mod 2 = 0) Then
9.              GoTo even ' Jump to even label
10.         Else
11.             GoTo odd   ' Jump to odd label
12.         End If
13. odd:
14.         Console.WriteLine(" It is an Odd number")
15.
16. even:
17.         Console.WriteLine(" It is an Even number ")
18.         Console.WriteLine(" Press any key to exit...")
19.         Console.ReadKey()
20.     End Sub
21. End Module
```

Output:

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



In the above program, when we enter a number, it checks whether the number is even or odd. And if the number is odd, the **GoTo statement** will be encountered, and the control transfers to the **odd statement** Else the control transfer to the **even statement**.

Use of GoTo Statement in For Next loop

Example 2: Write a program to print the sum of the First ten numbers using the Goto Statement in VB.NET.

Goto_For.vb

```
1.  Imports System
2.  Module Goto_For
3.      Sub Main()
4.          'Declaration of local variable
5.          Dim i, n As Integer, Sum As Integer = 0
6.
7.          ' Define For Loop Statement
8.          For i = 1 To 10
9.              Console.WriteLine(" Value of i is {0}", i)
10.             Sum = Sum + i 'At each iteration the value of i added to Sum
11.             If i = 5 Then
12.                 n = i 'Assign i to n
13.
14.                 GoTo total_sum  ' Jump to total_sum
15.             End If
16.         Next
17. total_sum:
18.         Console.WriteLine(" Total sum is {0}", Sum)
19.         Console.WriteLine(" GoTo Encounter, when value of i = {0}", n)
20.         Console.WriteLine(" Press any key to exit...")
```

```
21.         Console.ReadKey()
22.     End Sub
23. End Module
```

Output:

In the above program, the For loop is executed till the given condition **(i = 1 To 10)**. And when the value of **i is equal to 5**, the **GoTo** statement will be encountered, and it transfers the control to **total_sum** so that the total sum of each iteration can be printed in the For loop.

Use of GoTo statement in Select Case Statement

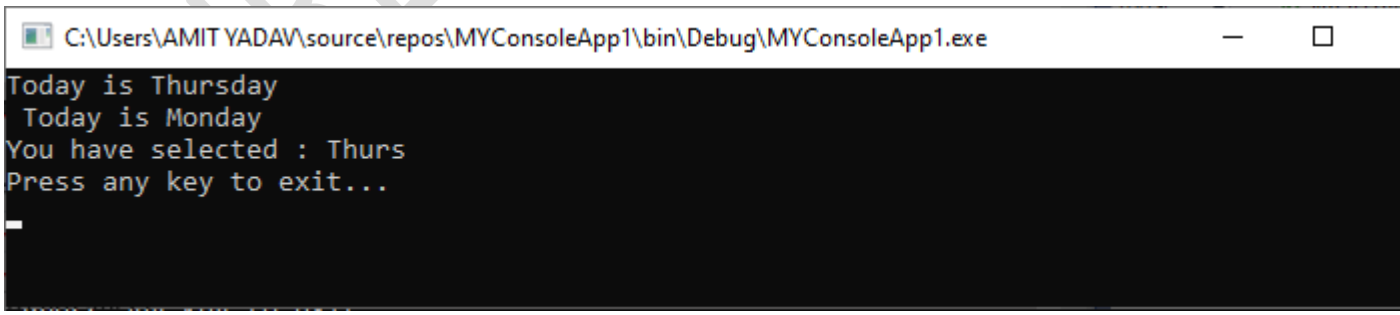
Example 3: Write a simple program to print the days' names in select cases using the GoTo Statement.

Goto_Select.vb

```
1.  Imports System
2.  Module Goto_Select
3.      Sub Main()
4.          'Definition of local variable
5.          Dim Days As String
6.          Days = "Thurs"
7.          Select Case Days
8.
9.              Case "Mon"
10. Case1:
11.
12.                 Console.WriteLine(" Today is Monday")
```

```
13.          Case "Tue"
14.             Console.WriteLine(" Today is Tuesday")
15.          Case "Wed"
16.             Console.WriteLine("Today is Wednesday")
17.          Case "Thurs"
18.             Console.WriteLine("Today is Thursday")
19.             GoTo Case1
20.          Case "Fri"
21.             Console.WriteLine("Today is Friday")
22.          Case "Sat"
23.             Console.WriteLine("Today is Saturday")
24.          Case "Sun"
25.             Console.WriteLine("Today is Sunday")
26.          Case Else
27.             Console.WriteLine(" Something wrong")
28.
29.       End Select
30.       Console.WriteLine("You have selected : {0}", Days)
31.       Console.WriteLine("Press any key to exit...")
32.       Console.ReadLine()
33.    End Sub
34. End Module
```

Output:



In the Select case statement, the value of Days **Thurs** will compare the values of all selected cases available in a program. If it matches any statement, it prints the particular statement and contains the GoTo statement that transfers control to defined case1: and then the following statement is executed.

Use of GoTo statement in While End Loop Statement

Example 4: Write a simple program to understand the use of GoTo statement in the While loop.

Goto_While.vb

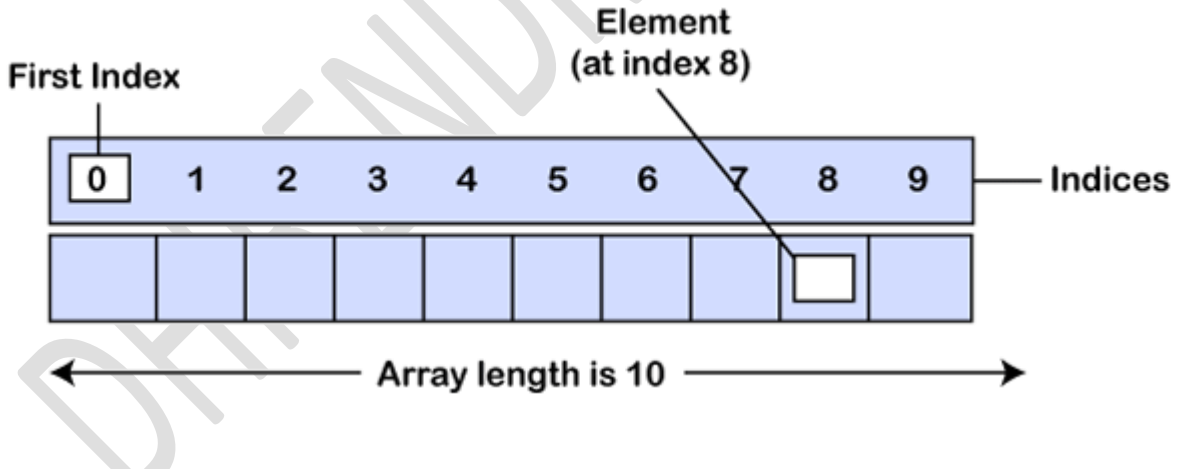
```
1. Imports System
2. Module GoTo_While
3.     Sub Main()
4.         'Declaration and initialization of the local variable
5.         Dim i As Integer = 1, num As Integer
6. start1:
7.         'Definition of While Loop
8.         While i < 10
9.             If i = 6 Then
10.                num = i
11.                i += 1
12.                GoTo start1 'transfer control at start1
13.            End If
14.            Console.WriteLine(" Value of i is {0}", i)
15.
16.            i += 1 ' Update the variable i by 1
17.        End While
18.        Console.WriteLine(" Control transfer at number {0}", num)
19.
20.        Console.WriteLine(" Press any key to exit...")
21.        Console.ReadKey()
22.    End Sub
23. End Module
```

Output:

VB.NET Arrays

An **array** is a linear data structure that is a collection of data elements of the same type stored on a **contiguous memory** location. Each data item is called an element of the array. It is a fixed size of sequentially arranged elements in computer memory with the first element being at **index 0** and the last element at index **n - 1,** where **n** represents the total number of elements in the array.

The following is an illustrated representation of similar data type elements defined in the VB.NET array data structure.



In the above diagram, we store the Integer type data elements in an array starting at index 0. It will continue to store data elements up to a defined number of elements.

Declaration of VB.NET Array

We can declare an array by specifying the data of the elements followed by parentheses () in the VB.NET.

1. Dim array_name As [Data_Type] ()

In the above declaration, **array_name** is the name of an array, and the **Data_Type** represents the type of element (Integer, char, String, Decimal) that will to store contiguous data elements in the VB.NET array.

Now, let us see the example to declare an array.

```
'Store only Integer values
Dim num As Integer() or Dim num(5) As Integer
'Store only String values
Dim name As String() or Dim name(5) As String
' Store only Double values
Dim marks As Double()
```

Initialization of VB.NET Array

In VB.NET, we can initialize an array with **New** keyword at the time of declaration. For example,

```
'Declaration and Initialization of an array elements with size 6
Dim num As Integer() = New Integer(5) { }
Dim num As Integer() = New Integer(5) {1, 2, 3, 4, 5, 6}
Initialize an array with 5 elements that indicates the size of an array
Dim arr_name As Integer() = New Integer() {5, 10, 5, 20, 15}
Declare an array
Dim array1 As Char()
array1 = New Char() {'A', 'B', 'C', 'D', 'E'}
```

Furthermore, we can also initialize and declare an array using the following ways, as shown below.

```
Dim intData() As Integer = {1, 2, 3, 4, 5}
Dim intData(5) As Integer
Dim array_name() As String = {"Peter", "John", "Brock", "James", "Maria"}
Dim misc() as Object = {"Hello friends", 16c, 12ui, "A"c}
Dim Emp(0 to 2) As String
Emp{0} = "Mathew"
Emp(1) = " Anthony"
Emp(2) = "Prince"
```

Let's create a program to add the elements of an array in VB.NET programming language.

num_Array.vb

```
1. Imports System
2. Module num_Array
3.     Sub Main()
4.         Dim i As Integer, Sum As Integer = 0
5.         'In VB.NET the size of an array is n+1
6.         'Declaration and Initialization of marks() array
7.         Dim marks() As Integer = {58, 68, 95, 50, 23, 89}
8.         Console.WriteLine(" Marks in 6 Subjects")
9.         For i = 0 To marks.Length - 1
10.            Console.WriteLine(" Marks {0}", marks(i))
11.            Sum = Sum + marks(i)
12.        Next
13.        Console.WriteLine(" Grand total is {0}", Sum)
14.
15.        Console.WriteLine(" Press any key to exit...")
16.        Console.ReadKey()
17.    End Sub
18. End Module
```

Output:

In the above program, we create an integer array with name **marks()** and define a For loop to access each item of the array marks.

Input number in VB.NET Array

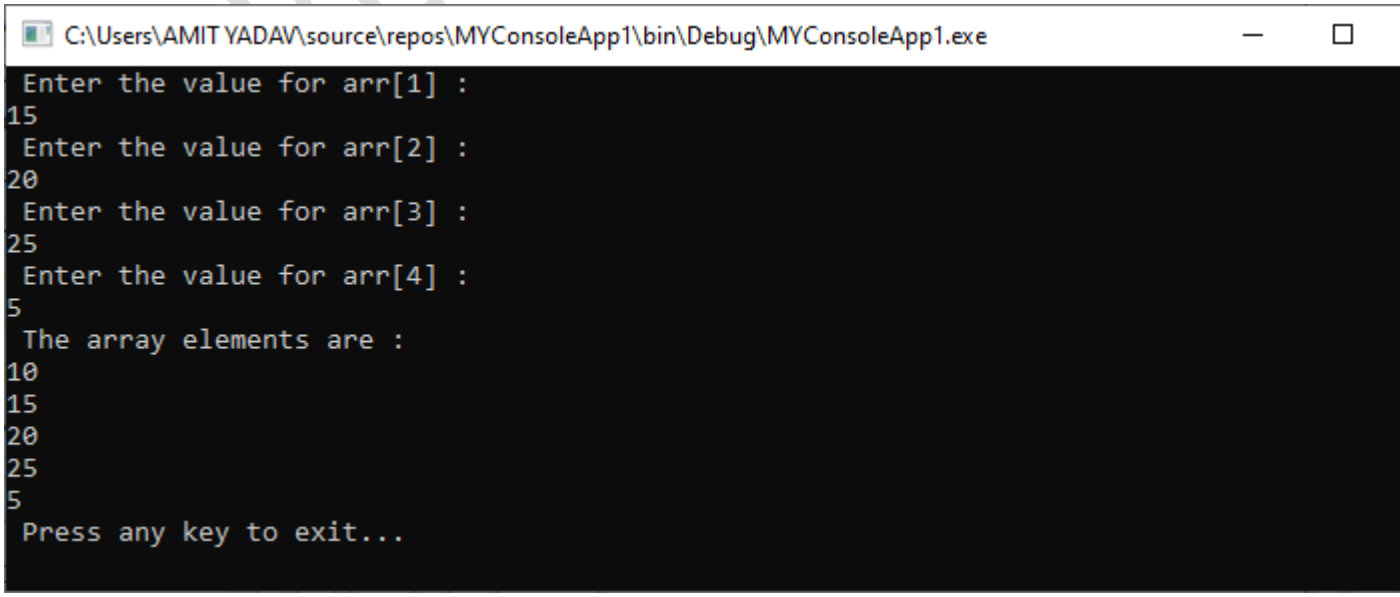
Let's create a program to take input values from the user and display them in VB.NET programming language.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Input_array.vb

```
1. Imports System
2. Module Input_array
3.     Sub Main()
4.         'Definition of array
5.         Dim arr As Integer() = New Integer(5) {}
6.         For i As Integer = 0 To 5
7.             Console.WriteLine(" Enter the value for arr[{0}] : ", i)
8.             arr(i) = Console.ReadLine()  ' Accept the number in array
9.         Next
10.        Console.WriteLine(" The array elements are : ")
11. ' Definition of For loop
12.        For j As Integer = 0 To 5
13.
14.            Console.WriteLine("{0}", arr(j))
15.        Next
16.
17.        Console.WriteLine(" Press any key to exit...")
18.        Console.ReadKey()
19.    End Sub
20. End Module
```

Output:



Multidimensional Array

In VB.NET, a multidimensional array is useful for storing more than one dimension in a tabular form, such as rows and columns. The multidimensional array support two or three dimensional in VB.NET.

Declaration of Multidimensional Array

```
1. Declaration of two-dimensional array
2. Dim twoDimenArray As Integer( , ) = New Integer(3, 2) {}
3. Or Dim arr(5, 3) As Integer
4. Representation of Three Dimensional array
5. Dim arrThree(2, 4, 3) As Integer
6. Or Dim arr1 As Integer( , , ) = New Integer(5, 5, 5) { }
```

In the above representation of multidimensional, we have created 2-dimensional array **twoDimenArray with 3 rows and 2 columns and** 3-dimensional array with three dimensions 2, 4, and 3.

Initialization of Multidimensional Array

The following ways to initialize the multidimensional array:

```
1. ' Initialization of Two Dimensional Array
2. Dim intArray As Integer( , ) = New Integer( 3, 2) { {4, 5}, {2, 3}, {6, 7} }
3. Dim intArray( , ) As Integer = {  {5, 4}, {3, 2}, {4, 7} }
4. ' Initialization of Three Dimensional Array
5. Dim threeDimen(3, 3, 2 ) As Integer = { {{1, 3, 2}, {2, 3, 4}}, {{5, 3, 6}, {3, 4, 5}}, {{1, 2, 2}, {5, 2, 3}
   }}
```

Multidimensional Array Example

Let's create a program to understand the multidimensional array.

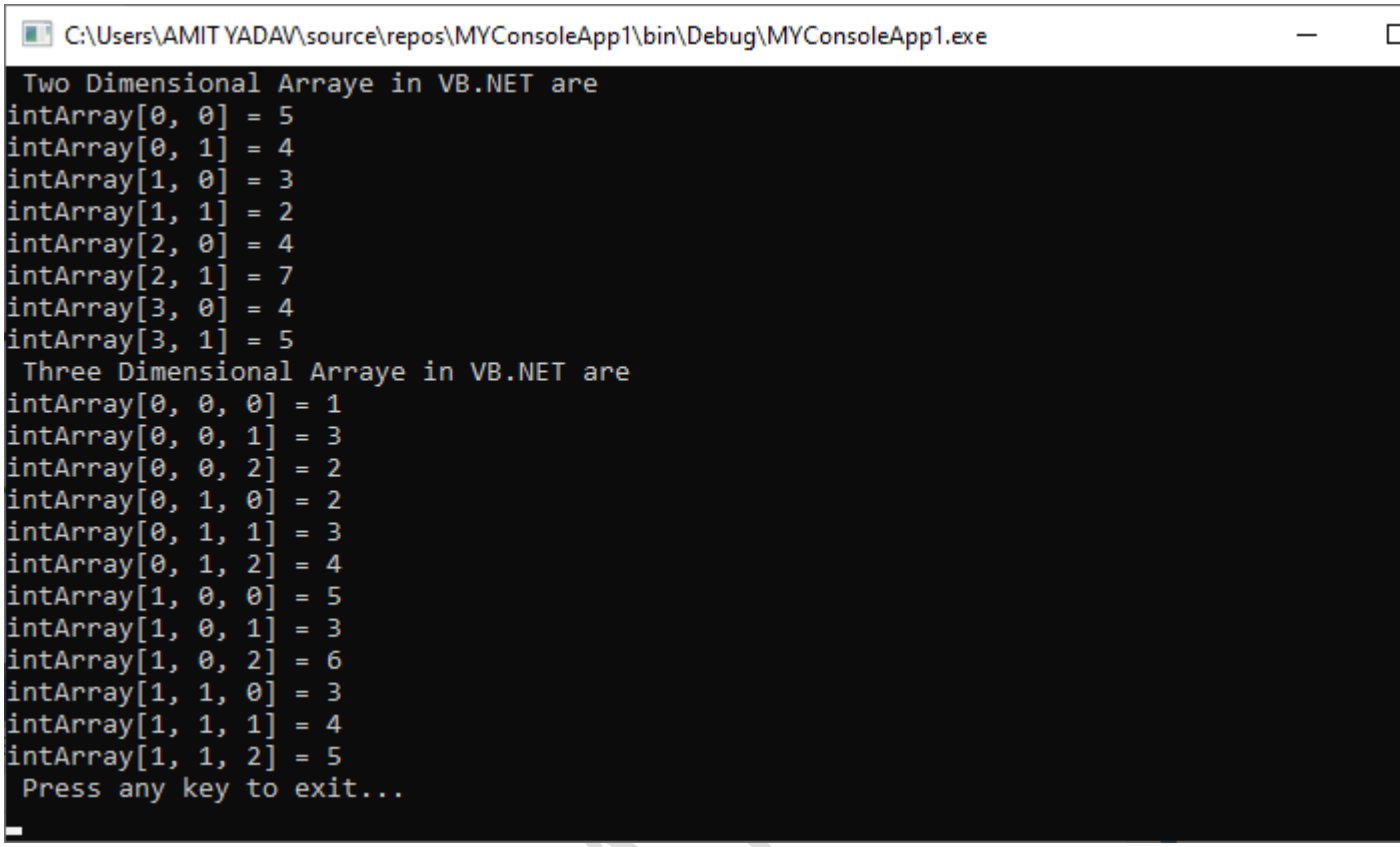
MultidimenArray.vb

```
1. Imports System
2. Module MultidimenArray
3.     Sub Main()
4.         ' Definition of 2 Dimensional Array
5.         Dim intArray(,) As Integer = {{5, 4}, {3, 2}, {4, 7}, {4, 5}}
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
6.
7.          ' Definition of 3 Dimensional Array
8.          Dim threeDimen(,,) As Integer =
9.              {{{1, 3, 2}, {2, 3, 4}},
10.             {{5, 3, 6}, {3, 4, 5}},
11.         {{1, 2, 2}, {5, 2, 3}}}
12.
13.         Console.WriteLine(" Two Dimensional Arraye in VB.NET are")
14.         For i As Integer = 0 To 3
15.             For j As Integer = 0 To 1
16.                 Console.WriteLine("intArray[{0}, {1}] = {2}", i, j, intArray(i, j))
17.             Next j
18.         Next i
19.
20.         Console.WriteLine(" Three Dimensional Arraye in VB.NET are")
21.         For i As Integer = 0 To 2 - 1
22.             For j As Integer = 0 To 2 - 1
23.                 For k As Integer = 0 To 4
24.                     Console.WriteLine("intArray[{0}, {1}, {2}] = {3}", i, j, k, threeDimen(i, j, k))
25.                 Next k
26.             Next j
27.         Next i
28.
29.         Console.WriteLine(" Press any key to exit...")
30.         Console.ReadKey()
31.     End Sub
32. End Module
```

Output:



Fixed Size Array

In VB.NET, a fixed- size array is used to hold a fixed number of elements in memory. It means that we have defined the number of elements in the array declaration that will remain the same during the definition of the elements, and its size cannot be changed. For example, we need to hold only 5 names in an array; it can be defined and initialized in the array such as,

```
1. Dim names( 0 to 4) As String
2. names(0) = "Robert"
3. names(1) = "Henry"
4. names(2) = "Rock"
5. names(3) = "James"
6. names(4) = "John"
```

The above representation of the fixed array is that we have defined a string array **names 0** to **4**, which stores all the elements in the array from 0 to index 4.

VB.NET Dynamic Array

A Dynamic array is used when we do not know how many items or elements to be inserted in an array. To resolve this problem, we use the dynamic array. It allows us to insert or store the number of elements at runtime in sequentially manner. A Dynamic Array can be resized according to the program's requirements at run time using the **"ReDim"** statement.

Initial Declaration of Array

Syntax:

```
1. Dim array_name() As Integer
```

Runtime Declaration of the VB.NET Dynamic array (Resizing)

Syntax:

```
1. ReDim {Preserve] array_name(subscripts)
```

The **ReDim** statement is used to declare a dynamic array. To resize an array, we have used a **Preserve** keyword that preserve the existing item in the array. The **array_name** represents the name of the array to be re-dimensioned. A **subscript** represents the new dimension of the array.

Initialization of Dynamic Array

```
1. Dim myArr() As String
2. ReDim myArr(3)
3. myArr(0) = "One"
4. myArr(1) = "Two"
5. myArr(2) = "Three"
6. myArr(3) = "Four"
```

To initialize a Dynamic Array, we have used create a string array named **myArr()** that uses the Dim statement in which we do not know the array's actual size. The **ReDim** statement is used to resize the existing array by defining the subscript (**3**). If we want to store one more element in index 4 while preserving three elements in an array, use the following statements.

```
1. ReDim Preserve myArr(4)
2. myArr(4) = "Five"
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

the above array has four elements.

Also, if we want to store multiple data types in an array, we have to use a **Variant** data type.

```
1. Dim myArr() As Variant
2. ReDim myArr(3)
3. myArr(0) = 10
4. myArr(0) = "String"
5. myArr(0) = false
6. myArr(0) = 4.6
```

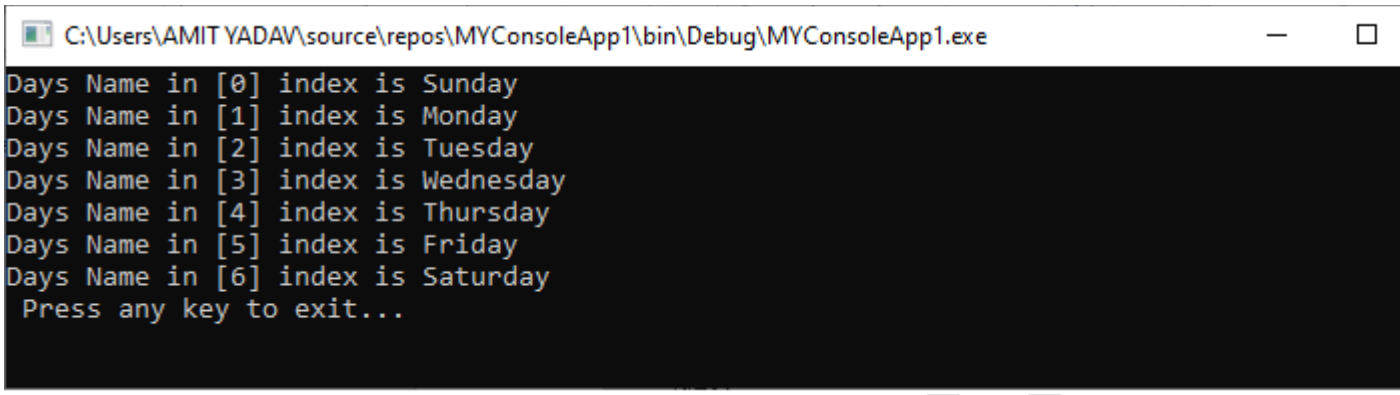
Let's create a program to understand the dynamic array.

Dynamic_Arr.vb

```
1. Imports System
2. Module Dynamic_Arr
3.     Sub Main()
4.         'Declaration and Initialization of String Array Days()
5.         Dim Days(20) As String
6.         ' Resize an Array using the ReDim Statement
7.         ReDim Days(6)
8.         Days(0) = "Sunday"
9.         Days(1) = "Monday"
10.        Days(2) = "Tuesday"
11.        Days(3) = "Wednesday"
12.        Days(4) = "Thursday"
13.        Days(5) = "Friday"
14.        Days(6) = "Saturday"
15.
16.        For i As Integer = 0 To Days.Length - 1
17.            Console.WriteLine("Days Name in [{0}] index is {1}", i, Days(i))
18.        Next
19.        Console.WriteLine(" Press any key to exit...")
20.        Console.ReadKey()
21.    End Sub
22. End Module
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Output:



Adding New Element to an Array

When we want to insert some new elements into an array of fixed size that is already filled with old array elements. So, in this case, we can use a dynamic array to add new elements to the existing array.

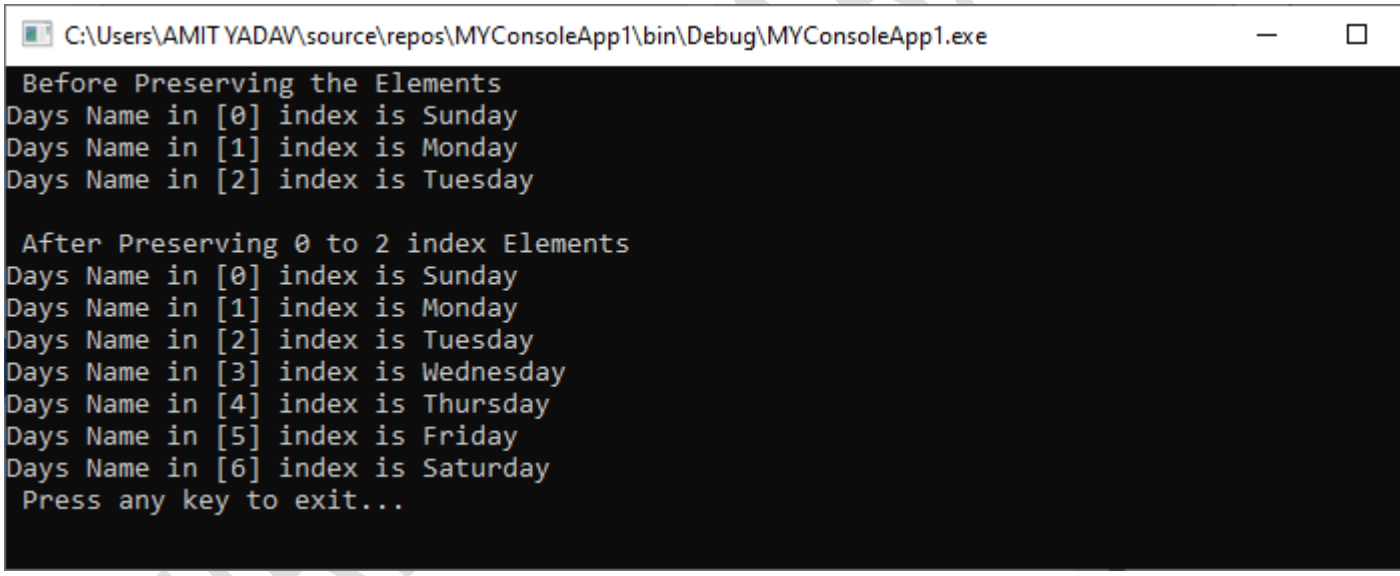
Let us create a program to understand how we can add new elements to a dynamic array.

Dynamic_Arr1.vb

```
1.  Imports System
2.  Module Dynamic_arr1
3.      Sub Main()
4.          'Declaration and Initialization of String Array Days()
5.          Dim Days() As String
6.          ' Resize an Array using the ReDim Statement
7.          ReDim Days(2)
8.          Days(0) = "Sunday"
9.          Days(1) = "Monday"
10.         Days(2) = "Tuesday"
11.         Console.WriteLine(" Before Preserving the Elements")
12.         For i As Integer = 0 To Days.Length - 1
13.             Console.WriteLine("Days Name in [{0}] index is {1}", i, Days(i))
14.         Next
15.         Console.WriteLine()
16.
17.         Console.WriteLine(" After Preserving 0 to 2 index Elements")
18.         ReDim Preserve Days(6)
```

```
19.         Days(3) = "Wednesday"
20.         Days(4) = "Thursday"
21.         Days(5) = "Friday"
22.         Days(6) = "Saturday"
23.         For i As Integer = 0 To Days.Length - 1
24.             Console.WriteLine("Days Name in [{0}] index is {1}", i, Days(i))
25.         Next
26.         Console.WriteLine(" Press any key to exit...")
27.         Console.ReadKey()
28.     End Sub
29. End Module
```

Output:



In the above program, we have created a dynamic array **Days as a String** that executes the first three elements of Days such as **Sunday, Monday,** and **Tuesday**. we have also used a Preserve Keyword to keep the existing elements of an array with new elements in dynamic array Days.

VB.NET Functions

In VB.NET, the function is a separate group of codes that are used to perform a specific task when the defined function is called in a program. After the execution of a function, control transfer to the **main()** method for further execution. It returns a value. In VB.NET, we can create more than one function in a program to perform various functionalities. The function is also useful to code reusability by reducing the duplicity of the code. For example, if we need to use the same functionality at multiple places in a program, we can simply create a function and call it whenever required.

Defining a Function

The syntax to define a function is:

```
1. [Access_specifier ] Function Function_Name [ (ParameterList) ] As Return_Type
2. [ Block of Statement ]
3. Return return_val
4. End Function
```

Where,

- **Access_Specifier:** It defines the access level of the function such as public, private, or friend, Protected function to access the method.
- **Function_Name:** The function_name indicate the name of the function that should be unique.
- **ParameterList:** It defines the list of the parameters to send or retrieve data from a method.
- **Return_Type:** It defines the data type of the variable that returns by the function.

The following are the various ways to define the function in a VB.NET.

```
1. Public Function add() As Integer
2. ' Statement to be executed
3. End Function
4.
5. Private Function GetData( ByVal username As String) As String
6. ' Statement to be executed
7. End Function
8.
```

```
Public Function GetData( ByVal username As String, ByVal userId As Integer) As String
' Statement to be executed
End Function
```

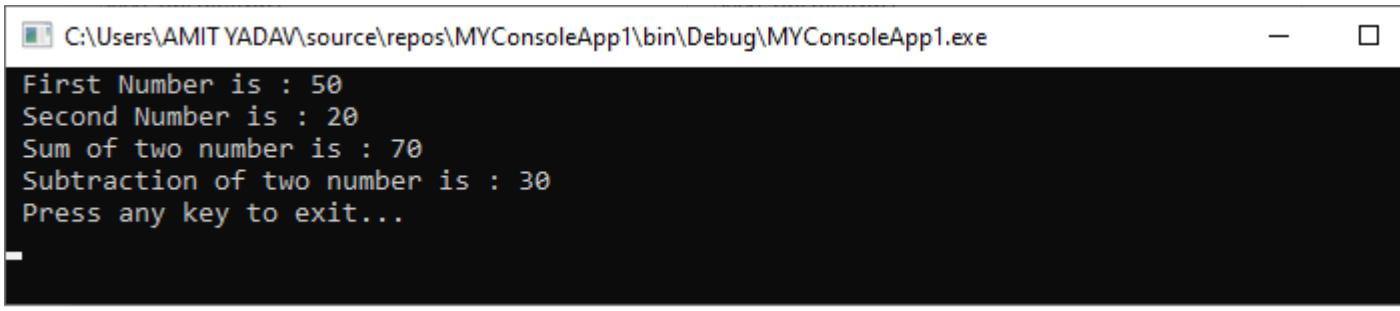
Example: Write a program to find the sum and subtraction of two numbers using the function.

Find_Sum.vb

```
Imports System
Module Find_Sum
    ' Create the SumOfTwo() Function and pass the parameters.
    Function SumOfTwo(ByVal n1 As Integer, ByVal n2 As Integer) As Integer
        ' Define the local variable.
        Dim sum As Integer = 0
        sum = n1 + n2
        Return sum
    End Function
    Function SubtractionOfTwo(ByVal n1 As Integer, ByVal n2 As Integer) As Integer
        ' Define the local variable.
        Dim subtract As Integer
        subtract = n1 - n2
        Return subtract
    End Function
    Sub Main()
        ' Define the local variable a and b.
        Dim a As Integer
        Dim b As Integer
        Dim total, total1 As Integer
        Console.WriteLine(" First Number is : {0}", a)
        Console.WriteLine(" Second Number is : {0}", b)
        total = SumOfTwo(a, b) 'call SumOfTwo() Function
        total1 = SubtractionOfTwo(a, b) 'call SubtractionOfTwo() Function
        Console.WriteLine(" Sum of two number is : {0}", total)
        Console.WriteLine(" Subtraction of two number is : {0}", total1)
        Console.WriteLine(" Press any key to exit...")
        Console.ReadKey()
    End Sub
```

30. End Module

Output:



In the above example, we have defined a **SumOfTwo() and SubtractionOfTwo()** function to add and subtract two predefined numbers. When the functions are called in the main() method, each function is executed and returns the sum and subtraction of two numbers, respectively.

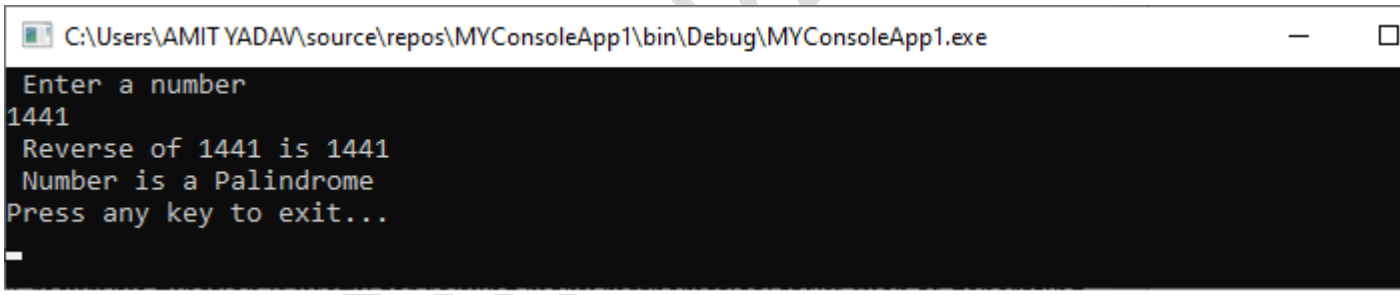
Example: Write a program to reverse a number and check whether the given number is palindrome or not.

Palindrome.vb

```
1.  Imports System
2.  Module Palindrome
3.      ' Define a reverse() function
4.      Function reverse(ByVal num As Integer) As Integer
5.          ' Define the local variable
6.          Dim remain As Integer
7.          Dim rev As Integer = 0
8.          While (num > 0)
9.              remain = num Mod 10
10.             rev = rev * 10 + remain
11.             num = num / 10
12.         End While
13.         Return rev
14.     End Function
15.     Sub Main()
16.         ' Define the local variable as integer
17.         Dim n, num2 As Integer
18.         Console.WriteLine(" Enter a number")
```

```
19.       n = Console.ReadLine() 'Accept the number
20.       num2 = reverse(n) ' call a function
21.       Console.WriteLine(" Reverse of {0} is {1}", n, num2)
22.
23.       If (n = reverse(n)) Then
24.           Console.WriteLine(" Number is a Palindrome")
25.       Else
26.           Console.WriteLine(" Number is not a Palindrome")
27.       End If
28.       Console.WriteLine("Press any key to exit...")
29.       Console.ReadKey()
30.   End Sub
31. End Module
```

Output:



Recursive Function

When a function calls itself until the defined condition is satisfied, it is known as a **recursive function**. A recursive function is useful for solving many mathematical tasks such as generating the Fibonacci series, the factorial of a number, etc.

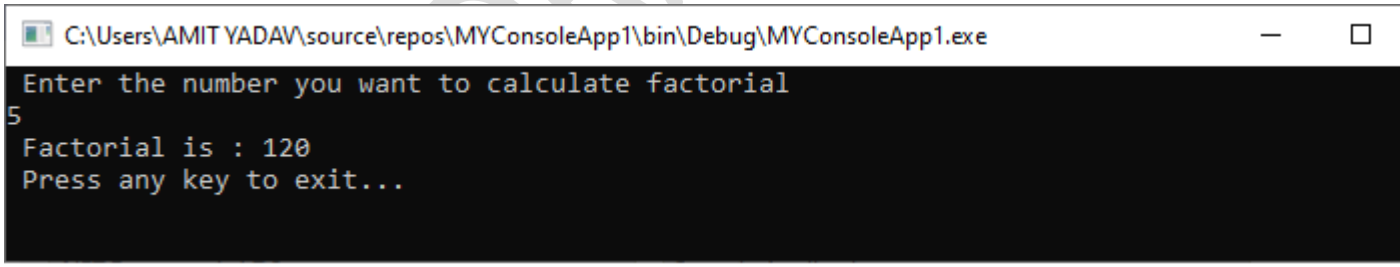
Let's create a program to calculate the factorial of a number using the recursive function.

Factorial_function.vb

```
1. Imports System
2. Module Factorial_function
3.     ' Create a Fact() function
4.     Function Fact(ByVal num As Integer) As Integer
5.         If (num = 0) Then
6.             Return 0
```

```
7.          ElseIf (num = 1) Then
8.              Return 1
9.          Else
10.             Return num * Fact(num - 1)
11.         End If
12.     End Function
13.     Sub Main()
14.         ' Define the local variable as integer
15.         Dim n, f As Integer
16.         Console.WriteLine(" Enter the number you want to calculate factorial")
17.         n = Console.ReadLine() 'Accept a number
18.         f = Fact(n) 'call Fact() function
19.         Console.WriteLine(" Factorial is : {0}", f)
20.         Console.WriteLine(" Press any key to exit...")
21.         Console.ReadKey()
22.     End Sub
23. End Module
```

Output:



Passing Array as a Parameter

Let's create a program that parses an array as a pass parameters in a function.

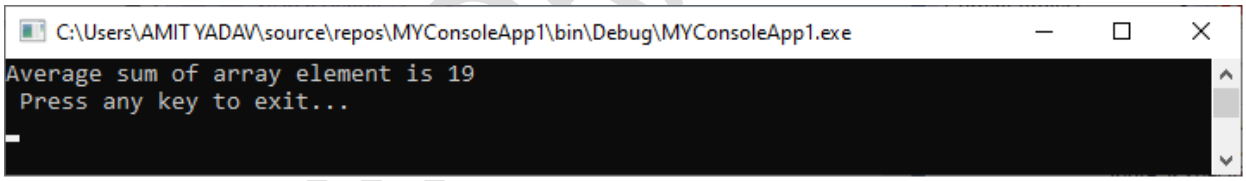
Array_Parameter.vb

```
1. Imports System
2. Module array_Parameter
3.     Function AddPara(ByVal Arr As Integer(), ByVal size As Integer) As Double
4.         'Declare a local variable.
5.
6.         Dim i As Integer
7.         Dim avg As Double
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
8.          Dim Sum As Integer = 0
9.
10.         For I = 0 To size - 1
11.             Sum = Sum + Arr(i)
12.         Next
13.         avg = Sum / size
14.         Return avg
15.     End Function
16.     Sub Main()
17.         Dim arrays As Integer() = {50, 10, 20, 5, 4, 25}
18.         Dim getAvg As Double
19.         getAvg = AddPara(arrays, 6)
20.         Console.WriteLine("Average sum of array element is {0}", getAvg)
21.         Console.WriteLine(" Press any key to exit...")
22.         Console.ReadKey()
23.     End Sub
24. End Module
```

Output:



VB.NET Sub

A Sub procedure is a separate set of codes that are used in VB.NET programming to execute a specific task, and it does not return any values. The Sub procedure is enclosed by the Sub and End Sub statement. The Sub procedure is similar to the function procedure for executing a specific task except that it does not return any value, while the function procedure returns a value.

Defining the Sub procedure

Following is the syntax of the Sub procedure:

```
1. [Access_Specifier ] Sub Sub_name [ (parameterList) ]
2. [ Block of Statement to be executed ]
```

```
3. End Sub
```

Where,

- **Access_Specifier:** It defines the access level of the procedure such as public, private or friend, Protected, etc. and information about the overloading, overriding, shadowing to access the method.
- **Sub_name:** The Sub_name indicates the name of the Sub that should be unique.
- **ParameterList:** It defines the list of the parameters to send or retrieve data from a method.

The following are the different ways to define the types of Sub method.

```
1. Public Sub getDetails()
2. ' Statement to be executed
3. End Sub
4.
5. Private Sub GetData( ByVal username As String) As String
6. ' Statement to be executed
7. End Sub
8.
9. Public Function GetData1( ByRef username As String, ByRef userId As Integer)
10. ' Statement to be executed
11. End Sub
```

Example: Write a simple program to pass the empty, a single or double parameter of Sub procedure in the VB.NET.

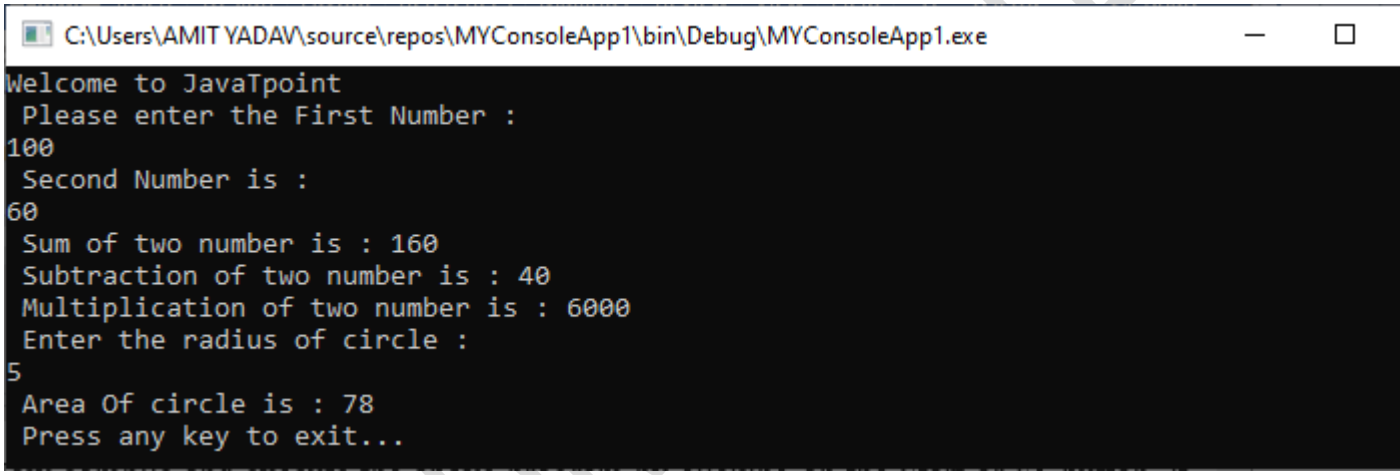
Sub_Program.vb

```
1. Module Sub_Program
2.     Sub sample()
3.         Console.WriteLine("Welcome to Sai College")
4.     End Sub
5.
6.     Sub circle(ByVal r As Integer)
7.         Dim Area As Integer
8.         Const PI = 3.14
9.         Area = PI * r * r
```

```
10.       Console.WriteLine(" Area Of circle is : {0}", Area)
11.   End Sub
12.
13.   ' Create the SumOfTwo() Function and pass the parameters.
14.   Sub SumOfTwo(ByVal n1 As Integer, ByVal n2 As Integer)
15.       ' Define the local variable.
16.       Dim sum As Integer = 0
17.       sum = n1 + n2
18.       Console.WriteLine(" Sum of two number is : {0}", sum)
19.   End Sub
20.
21.   Sub SubtractionOfTwo(ByVal n1 As Integer, ByVal n2 As Integer)
22.       ' Define the local variable.
23.       Dim subtract As Integer
24.       subtract = n1 - n2
25.       Console.WriteLine(" Subtraction of two number is : {0}", subtract)
26.   End Sub
27.
28.   Sub MultiplicationOfTwo(ByVal n1 As Integer, ByVal n2 As Integer)
29.       ' Define the local variable.
30.       Dim multiply As Integer
31.       multiply = n1 * n2
32.       Console.WriteLine(" Multiplication of two number is : {0}", multiply)
33.   End Sub
34.
35.   Sub Main()
36.       ' Define the local variable a, b and rad.
37.       Dim a, b, rad As Integer
38.       sample() ' call sample() procedure
39.       Console.WriteLine(" Please enter the First Number : ")
40.       a = Console.ReadLine()
41.       Console.WriteLine(" Second Number is : ")
42.       b = Console.ReadLine()
43.
44.       SumOfTwo(a, b) 'call SumOfTwo() Function
45.       SubtractionOfTwo(a, b) 'call SubtractionOfTwo() Function
46.       MultiplicationOfTwo(a, b) 'call MultiplicationOfTwo() Function
```

```
47.
48.         Console.WriteLine(" Enter the radius of circle : ")
49.         rad = Console.ReadLine()
50.         circle(rad)
51.         Console.WriteLine(" Press any key to exit...")
52.         Console.ReadKey()
53.     End Sub
54. End Module
```

Output:



In the VB.NET programming language, we can pass parameters in two different ways:

- Passing parameter by Value
- Passing parameter by Reference

Passing parameter by Value

In the VB.NET, passing parameter by value is the default mechanism to pass a value in the Sub method. When the method is called, it simply copies the actual value of an argument into the formal method of Sub procedure for creating a new storage location for each parameter. Therefore, the changes made to the main function's actual parameter that do not affect the Sub procedure's formal argument.

Syntax:

```
1. Sub Sub_method( ByVal parameter_name As datatype )
2. [ Statement to be executed]
3. End Sub
```

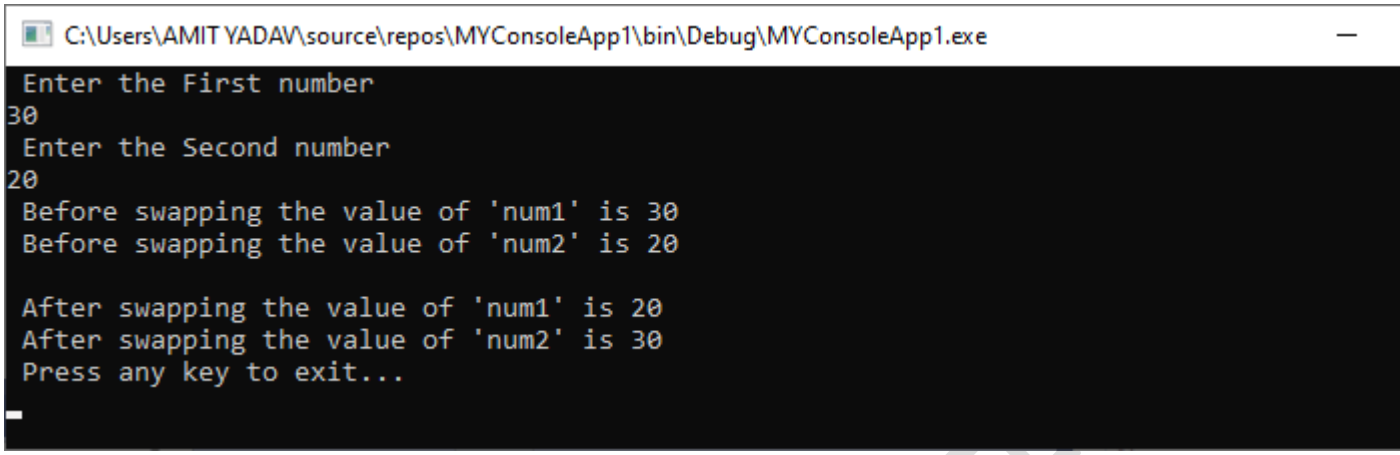
In the above syntax, the **ByVal** is used to declare parameters in a Sub procedure.

Let's create a program to understand the concept of passing parameter by value.

Passing_value.vb

```
1. Imports System
2. Module Passing_value
3.     Sub Main()
4.         ' declaration of local variable
5.         Dim num1, num2 As Integer
6.         Console.WriteLine(" Enter the First number")
7.         num1 = Console.ReadLine()
8.         Console.WriteLine(" Enter the Second number")
9.         num2 = Console.ReadLine()
10.        Console.WriteLine(" Before swapping the value of 'num1' is {0}", num1)
11.        Console.WriteLine(" Before swapping the value of 'num2' is {0}", num2)
12.        Console.WriteLine()
13.
14.        'Call a function to pass the parameter for swapping the numbers.
15.        swap_value(num1, num2)
16.        Console.WriteLine(" After swapping the value of 'num1' is {0}", num1)
17.        Console.WriteLine(" After swapping the value of 'num2' is {0}", num2)
18.        Console.WriteLine(" Press any key to exit...")
19.        Console.ReadKey()
20.    End Sub
21.
22.    ' Create a swap_value() method
23.    Sub swap_value(ByVal a As Integer, ByVal b As Integer)
24.        ' Declare a temp variable
25.        Dim temp As Integer
26.        temp = a ' save the value of a to temp
27.        b = a   ' put the value of b into a
28.        b = temp ' put the value of temp into b
29.    End Sub
30. End Module
```

Output:



Passing parameter by Reference

A Reference parameter is a reference of a variable in the memory location. The reference parameter is used to pass a reference of a variable with **ByRef** in the Sub procedure. When we pass a reference parameter, it does not create a new storage location for the sub method's formal parameter. Furthermore, the reference parameters represent the same memory location as the actual parameters supplied to the method. So, when we changed the value of the formal parameter, the actual parameter value is automatically changed in the memory.

The syntax for the passing parameter by Reference:

```
1. Sub Sub_method( ByRef parameter_name, ByRef Parameter_name2 )
2. [ Statement to be executed]
3. End Sub
```

In the above syntax, the **ByRef** keyword is used to pass the Sub procedure's reference parameters.

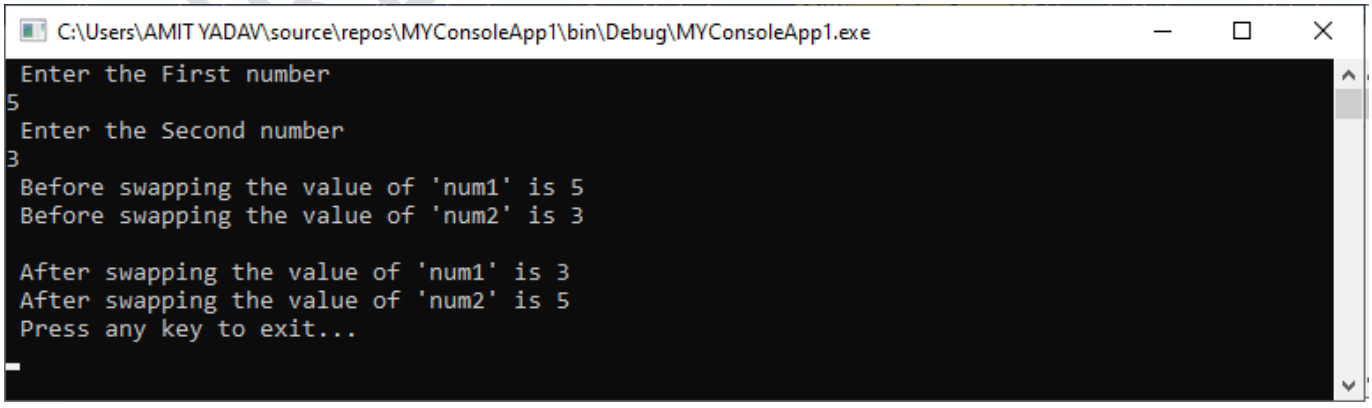
Let's create a program to swap the values of two variables using the ByRef keyword.

Passing_ByRef.vb

```
1. Imports System
2. Module Passing_ByRef
3.     Sub Main()
4.         ' declaration of local variable
5.         Dim num1, num2 As Integer
6.         Console.WriteLine(" Enter the First number")
7.         num1 = Console.ReadLine()
8.         Console.WriteLine(" Enter the Second number")
```

```
9.          num2 = Console.ReadLine()
10.         Console.WriteLine(" Before swapping the value of 'num1' is {0}", num1)
11.         Console.WriteLine(" Before swapping the value of 'num2' is {0}", num2)
12.         Console.WriteLine()
13.
14.         'Call a function to pass the parameter for swapping the numbers.
15.         swap_Ref(num1, num2)
16.         Console.WriteLine(" After swapping the value of 'num1' is {0}", num1)
17.         Console.WriteLine(" After swapping the value of 'num2' is {0}", num2)
18.         Console.WriteLine(" Press any key to exit...")
19.         Console.ReadKey()
20.     End Sub
21.
22.     ' Create a swap_Ref() method
23.     Sub swap_Ref(ByRef a As Integer, ByRef b As Integer)
24.         ' Declare a temp variable
25.         Dim temp As Integer
26.         temp = a ' save the value of a to temp
27.         a = b   ' put the value of b into a
28.         b = temp ' put the value of temp into b
29.     End Sub
30. End Module
```

Output:



Unit-3

VB.NET Form Controls

A **Form** is used in VB.NET to create a form-based or window-based application. Using the form, we can build a attractive user interface. It is like a container for holding different control that allows the user to interact with an application. The controls are an object in a form such as buttons, Textboxes, Textarea, labels, etc. to perform some action. However, we can add any control to the runtime by creating an instance of it.

A Form uses a **System.Windows.Form** namespace, and it has a wide family of controls that add both forms and functions in a Window-based user interface.

VB.NET Form Properties

The following are the most important list of properties related to a form. And these properties can be set or read while the application is being executed.

Properties	Description
BackColor	It is used to set the background color for the form.
BackgroundImage	It is used to set the background image of the form.
Cursor	It is used to set the cursor image when it hovers over the form.
AllowDrop	Using the AllowDrop control in a form, it allows whether to drag and drop on the form.
Font	It is used to get or set the font used in a form.
Locked	It determines whether the form is locked or not.
FormBorderStyle	It is used to set or get border style in a form.
Text	It is used to set the title for a form window.
MinimizeBox	MinimizeBox It is used to display the minimum option on the title bar of the form.
IsMDIChild	It is used to authenticate whether a form is a container of a Multiple Document Interface (MDI) child form.
Autoscroll	It allows whether to enable auto-scrolling in a form.
MaximizeBox	It is used to display the maximum option on the title bar of the form.
MaximumSize	It is used to set the maximum height and width of the form.
Language	It is used to specifies the localized language in a form.
AcceptButton	It is used to set the form button if the enter key is pressed.
Top, Left	It is used to set the top-left corner coordinates of the form in pixel.
Name	It is used to define the name of the form.
MinimumSize	It is used to set the minimum height and width of the form.
Enabled	It uses the True or False value to enable mouse or keyboard events in the form.
TopMost	It uses a Boolean value that represents whether you want to put the

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

	window form on top of the other form. By default, it is False.

Form Events

The following are the most important list of events related to a form.

Events	Description
Activated	An activated event is found when the user or program activates the form.
Click	A click event is active when the form is clicked.
Closed	A closed event is found before closing the form.
Closing	It exists when a form is closing.
DoubleClick	DoubleClick The DoubleClick event is activated when a user double clicks on the form.
DragDrop	A DragDrop event is activated when a drag and drop operation is performed.
MouseDown	A MouseDown event is activated when the mouse pointer is on the form, and the mouse button is pressed.
GotFocus	A GotFocus event is activated when the form control receives a focus.
HelpButtonClicked	It is activated when a user clicked on the help button.
KeyDown	A KeyDown event is activated when a key is pressed while focussing on the form.
KeyUp	A KeyUp event is activated when a key is released while focusing on the form.
Load	The load event is used to load a form before it is first displayed.

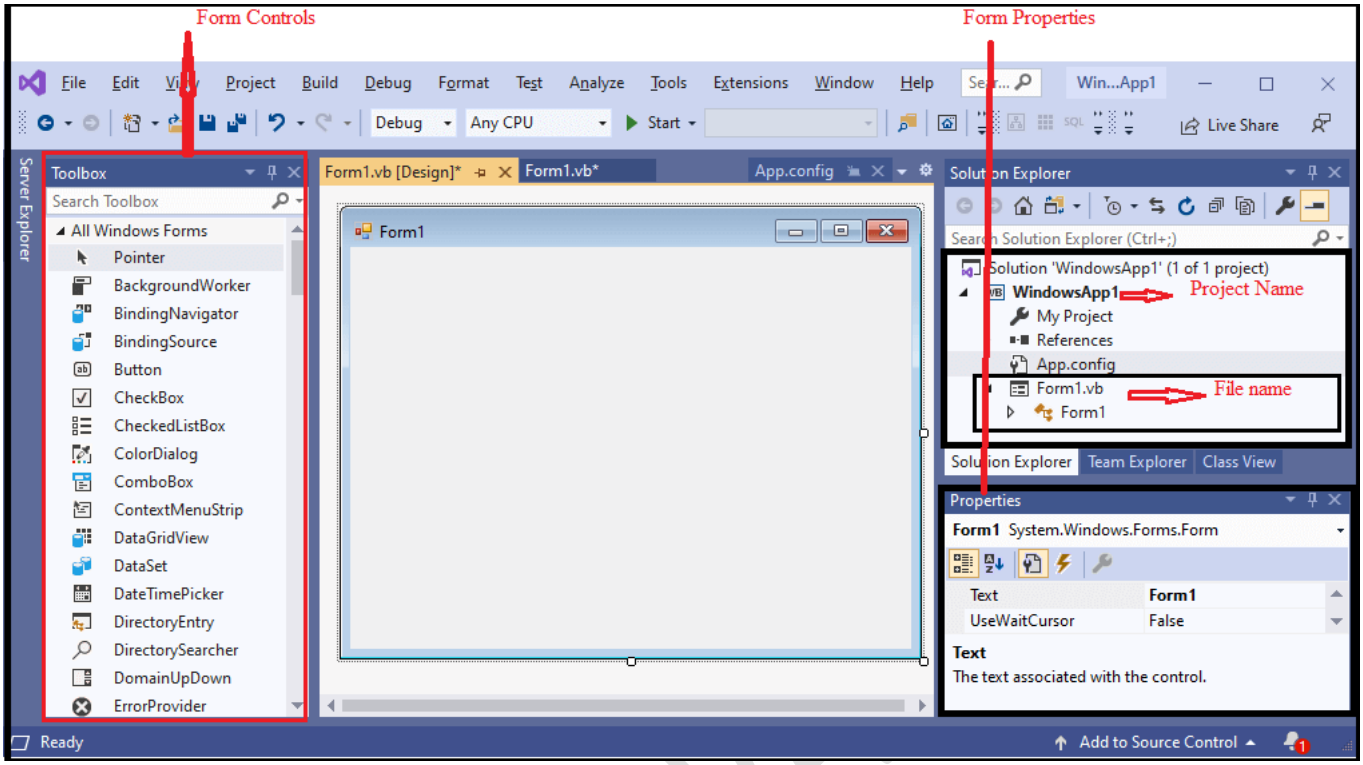
LostFocus	It is activated when the form loses focus.
MouseEnter	A MouseEnter event is activated when the mouse pointer enters the form.
MouseHover	A MouseHover event is activated when the mouse pointer put on the form.
MouseLeave	A MouseLeave event is activated when the mouse pointer leaves the form surface.
Shown	It is activated whenever the form is displayed for the first time.
Scroll	A Scroll event is activated when a form is scrolled through a user or code.
Resize	A Resize event is activated when a form is resized.
Move	A Move event is activated when a form is moved.

For creating a Windows Forms application in VB.NET, we need to follow the following steps in Microsoft Visual Studio.

1. GoTo File Menu.
2. Click on New Project.
3. Click on Windows Forms App or Application

And finally, click on the 'Create' button to create your project, and then, it displays the following window form with a name Form1.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



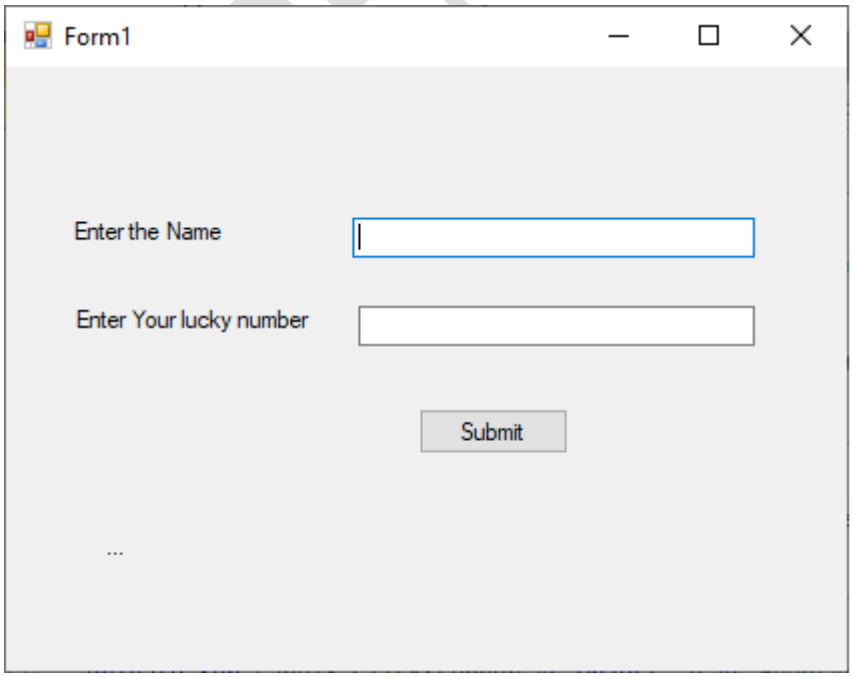
Now create a simple program of Windows form control in VB.NET.

Form1.vb

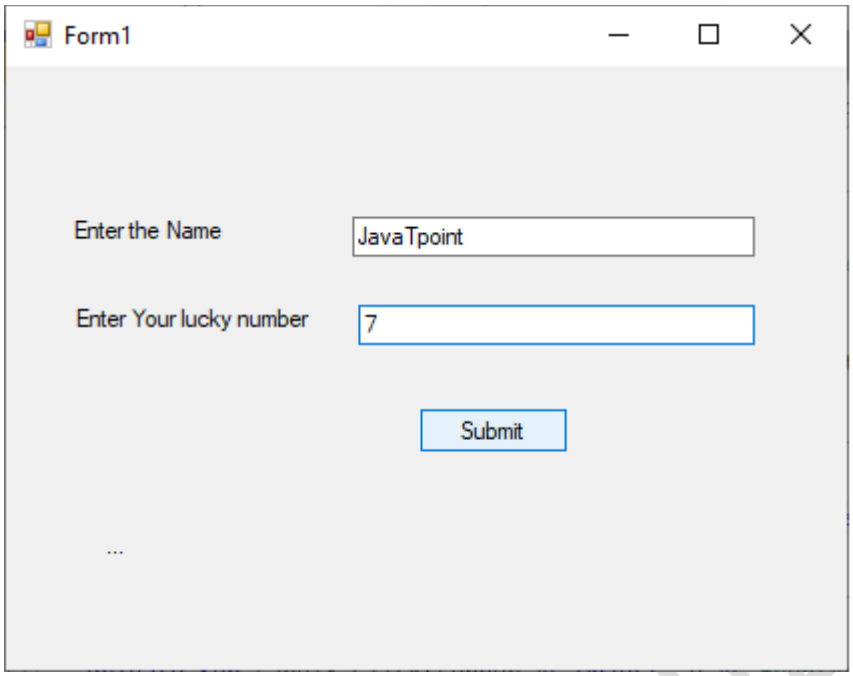
```
1.  Public Class Form1
2.  ' Create nameStr and num variables
3.      Dim nameStr As String
4.      Dim num As Integer
5.      Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
6.
7.      End Sub
8.     ' It is Label1
9.      Private Sub Label1_Click(sender As Object, e As EventArgs) Handles Label1.Click
10.
11.     End Sub
12.   ' It is TextBox1 for inserting the value.
13.     Private Sub TextBox1_TextChanged(sender As Object, e As EventArgs) Handles TextBox1.T
    extChanged
14.
15.     End Sub
16.     ' It is Label2
17.     Private Sub Label2_Click(sender As Object, e As EventArgs) Handles Label2.Click
```

```
18.
19.     End Sub
20.
21. ' It is a Button1 for transferring the control.
22.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
23.         nameStr = TextBox1.Text
24.         num = TextBox2.Text
25.         Label3.Text = "You have entered the Name " & nameStr + " Number " & num
26.
27.     End Sub
28.
29.     ' It is TextBox2 for inserting the value.
30.     Private Sub TextBox2_TextChanged(sender As Object, e As EventArgs) Handles TextBox2.T
    extChanged
31.
32.     End Sub
33. ' It is label3
34.     Private Sub Label3_Click(sender As Object, e As EventArgs) Handles Label3.Click
35.
36.     End Sub
37. End Class
```

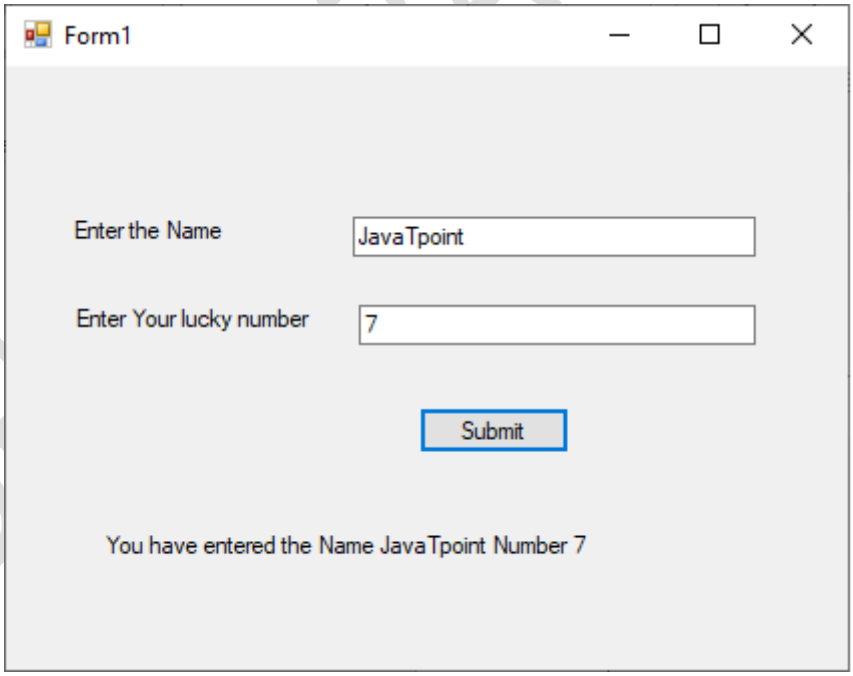
Output:



Now enter the following details in the form:



After filling all the details, click on the **Submit** button. After that, it shows the following Output:



VB.NET Menu Control

A menu is used as a menu bar in the Windows form that contains a list of related commands, and it is implemented through MenuStrip Control. The Menu control is also known as the VB.NET **MenuStrip** Control. The menu items are created with

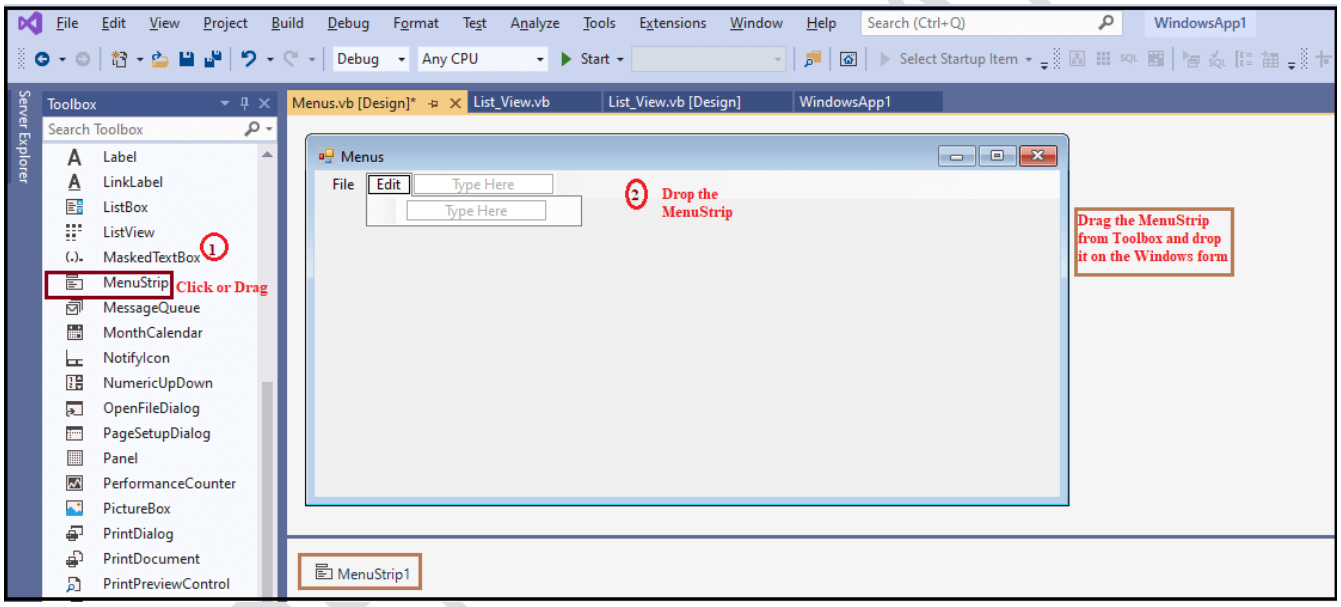
ToolStripMenuItem Objects. Furthermore, the **ToolStripDropDownMenu** and **ToolStripMenuItem** objects enable full control over the structure, appearance, functionalities to create menu items, submenus, and drop-down menus in a VB.NET

application.

Let's create a MenuBar by dragging a MenuStrip control from the toolbox and dropping it to the Windows

form.

Step 1. Drag the MenuStrip control from the toolbox and drop it on to the Form.



Step 2: Once the MenuStrip is added to the form, we can set various properties of the Menu by clicking on the MenuStrip control.

Properties of the MenuStrip Control

There are following properties of the VB.NET MenuStrip control.

Properties	Description
CanOverflow	The CanOverflow property is used to authenticate whether the control supports overflow functionality by setting values in the MenuStrip control.

Stretch	The Stretch property is used to obtain a value that specifies whether the menustrip stretches from end to end in the MenuStrip control.
GripStyle	The GripStyle property obtains or sets the visibility of the grip that uses the reposition of the menu strip control.
ShowItemToolTips	It is used to obtain or set the value that determines if the ToolTips are displayed for the MenuStrip Control.
DefaultSize	The DefaultSize property is used to get the default horizontal and vertical dimension of the MenuStrip in pixel when it is first created.

Methods of the MenuStrip Control

Method	Description
CreateAccessibilityInstance()	It is used to create a new accessibility instance for the MenuStrip Control.
ProcessCmdKey()	The ProcessCmdKey method is used to process the command key in the MenuStrip Control.
CreateDefaultItem()	The CreateDefaultItem method is used to create a ToolStripMenuItem with the specified text, image, and event handlers for the new MenuStrip.
OnMenuActivate()	It is used to initiate the MenuActivate event in the MenuStrip control.
OnMenuDeactivate()	It is used to start the MenuDeactivate event in the MenuStrip control.

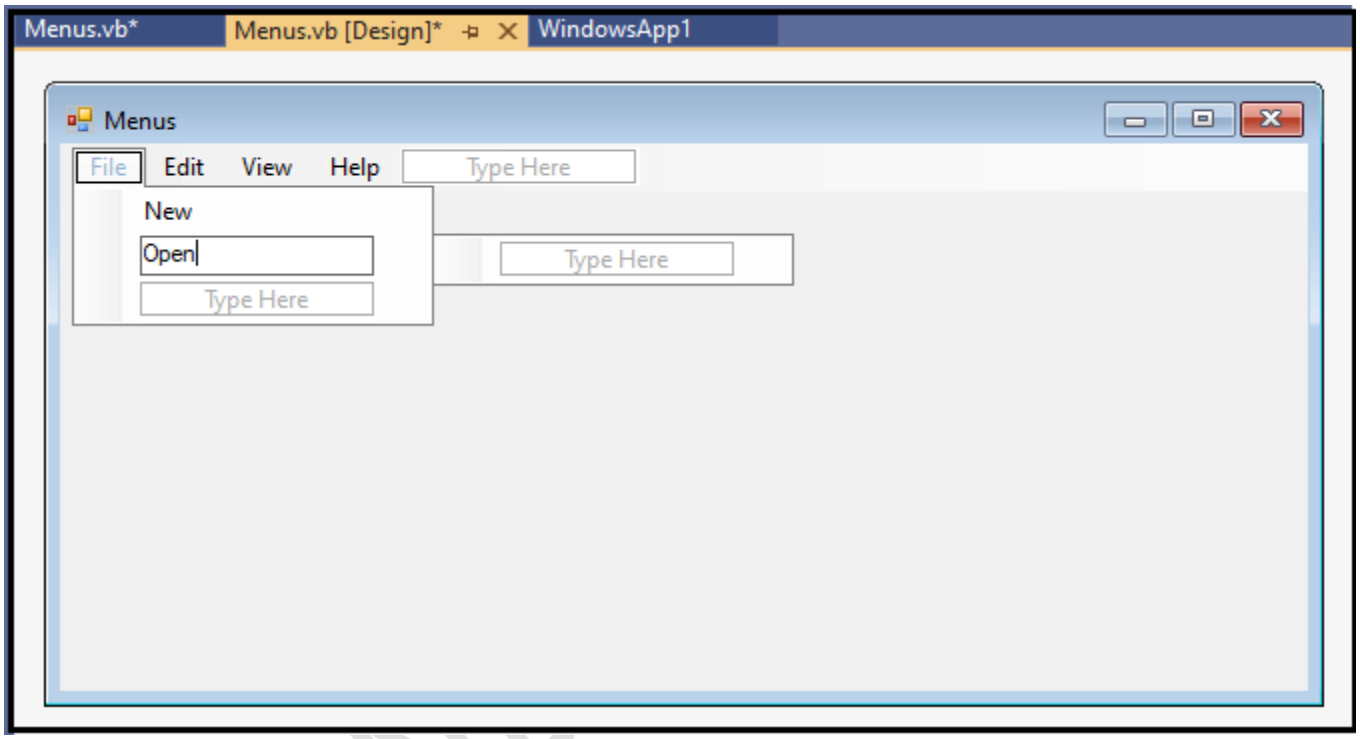
Events of the MenuStrip Control

Events	Description
MenuActivate	When a user uses a menu bar control with a mouse or keyboard, a MenuActivate event occurs.

MenuDeactivate	The MenuDeactivate event occurs when the MenuStrip control is deactivated in the Windows form.

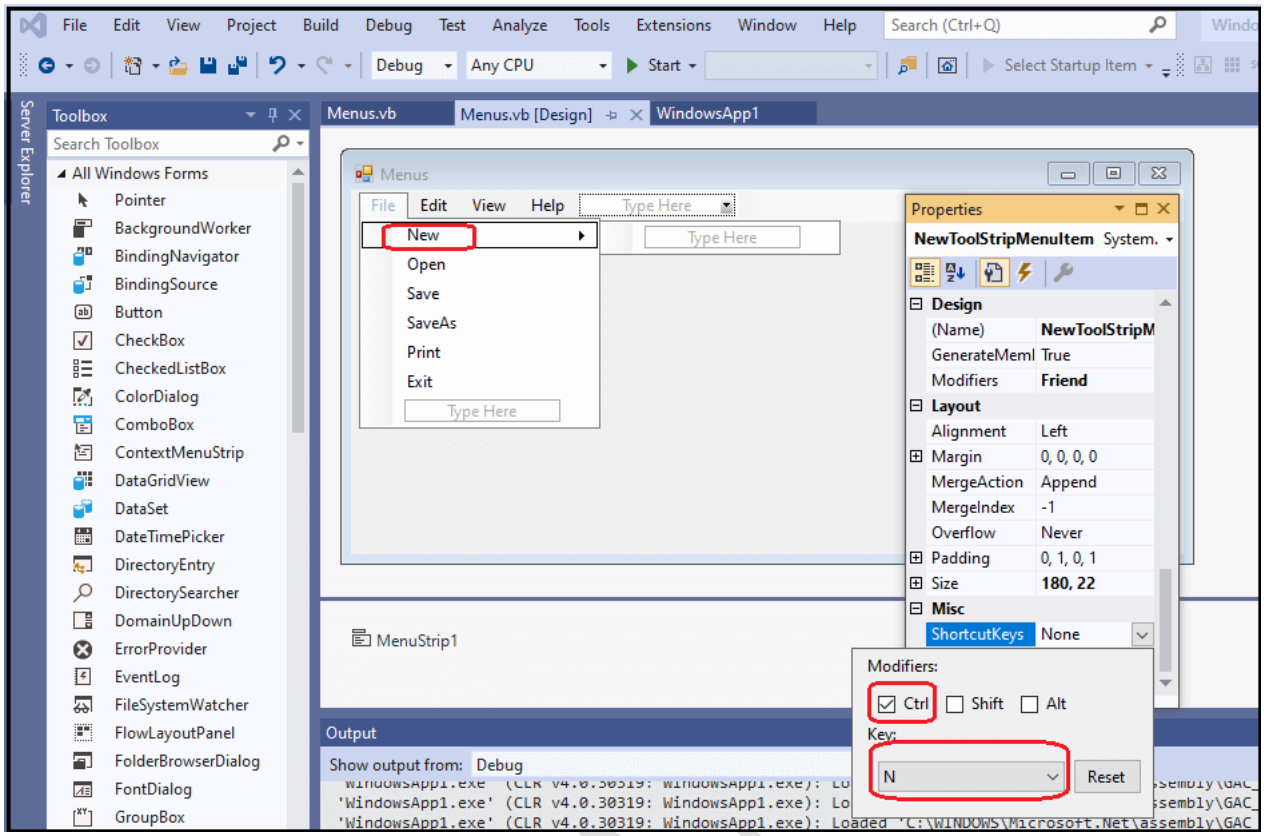
Let's create a program to display the menu bar in the Windows form.

In this image, we have created the menu and sub-items of the menu bar in the form.

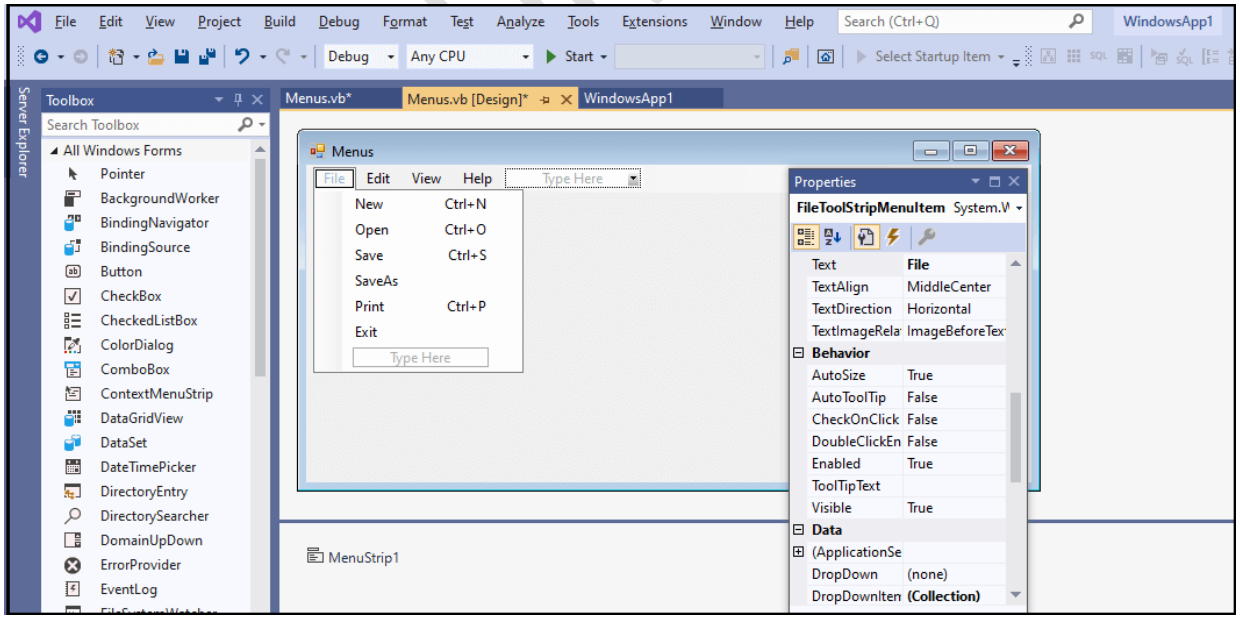


Now, we write the Shortcut keys for the File subitems, such as **New -> Ctrl + N, Open -> Ctrl + O**, etc.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



After that, we can see the subitems of the Files with their Shortcut keys, as shown below.



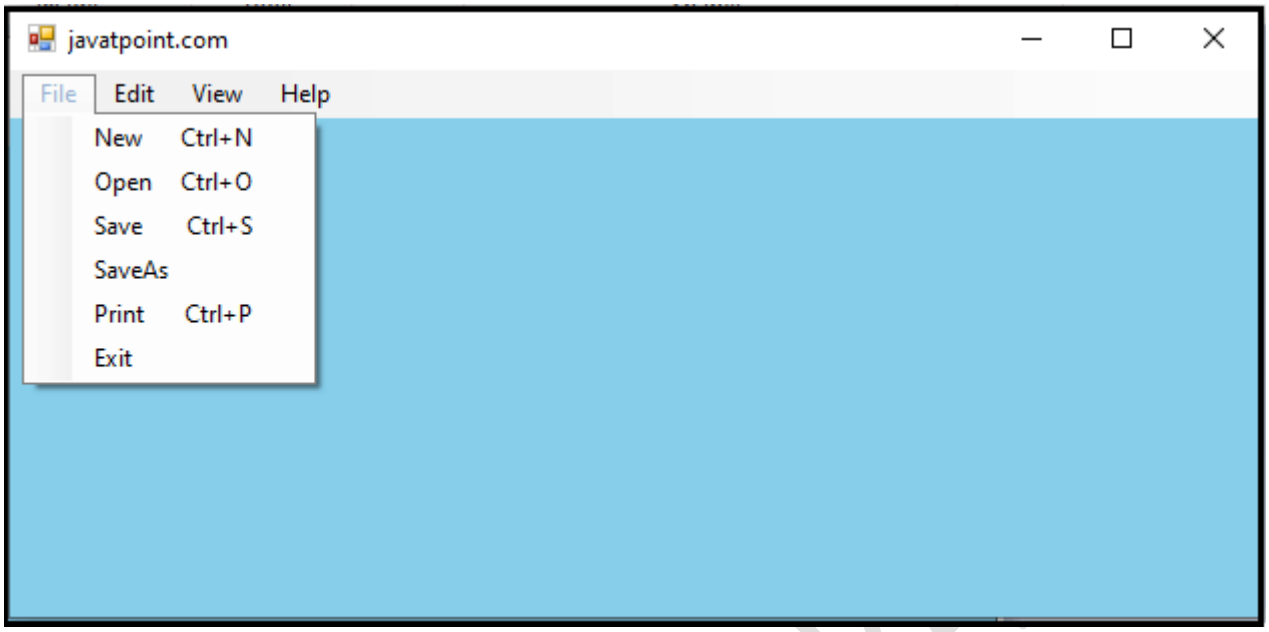
PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Menus.vb

```
1. Public Class Menus
2.     Private Sub Menus_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.         Me.Text = "javatpoint.com" 'set the title of the bar
4.         BackColor = Color.SkyBlue
5.     End Sub
6.
7.     Private Sub ExitToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles ExitTo
olStripMenuItem.Click
8.         Me.Dispose() ' exit from the form
9.     End Sub
10. End Class
```

Output:

Click on the File menu that shows the multiple options related to files.



VB.NET MDI Form

MDI stands for **Multiple Document Interface** applications that allow users to work with multiple documents by opening more than one document at a time. Whereas, a **Single Document Interface (SDI)** application can manipulate only one document at a time.

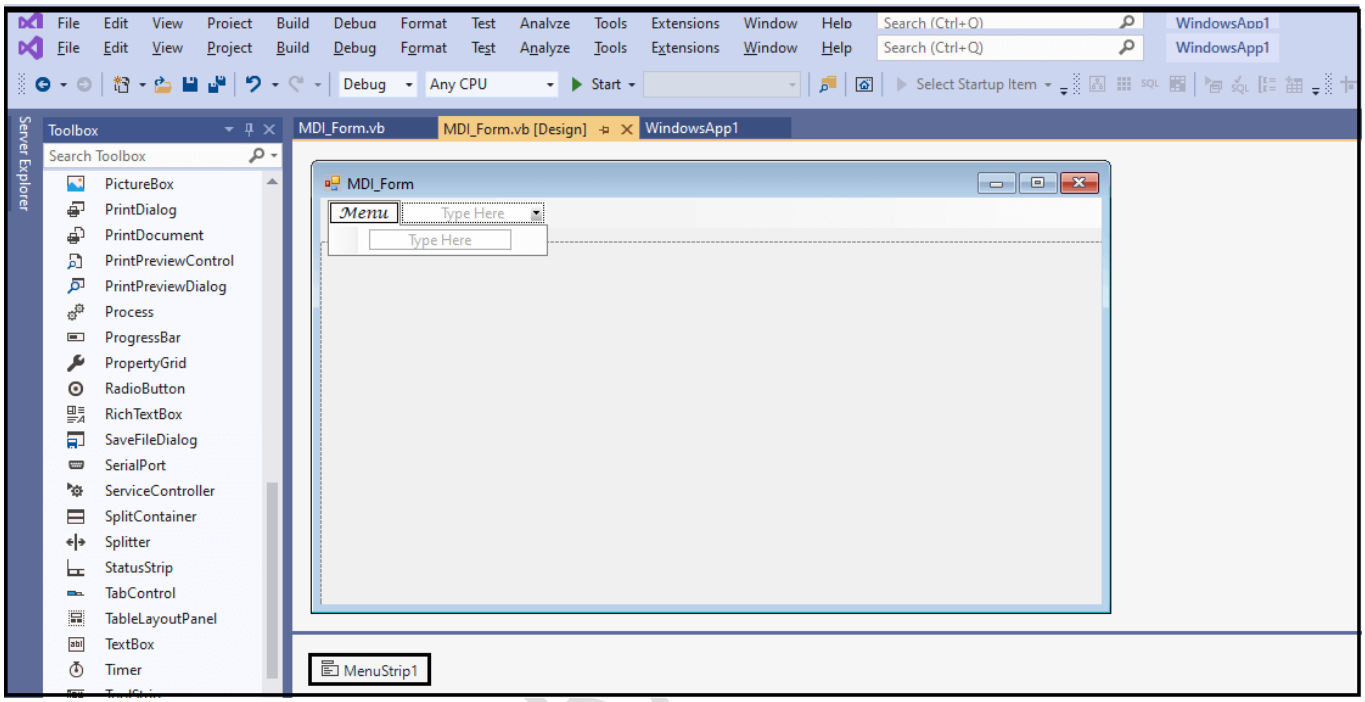
The MDI applications act as the parent and child relationship in a form. A parent form is a container that contains child forms, while child forms can be multiple to display different modules in a parent form.

VB.NET has folowing rules for creating a form as an MDI form.

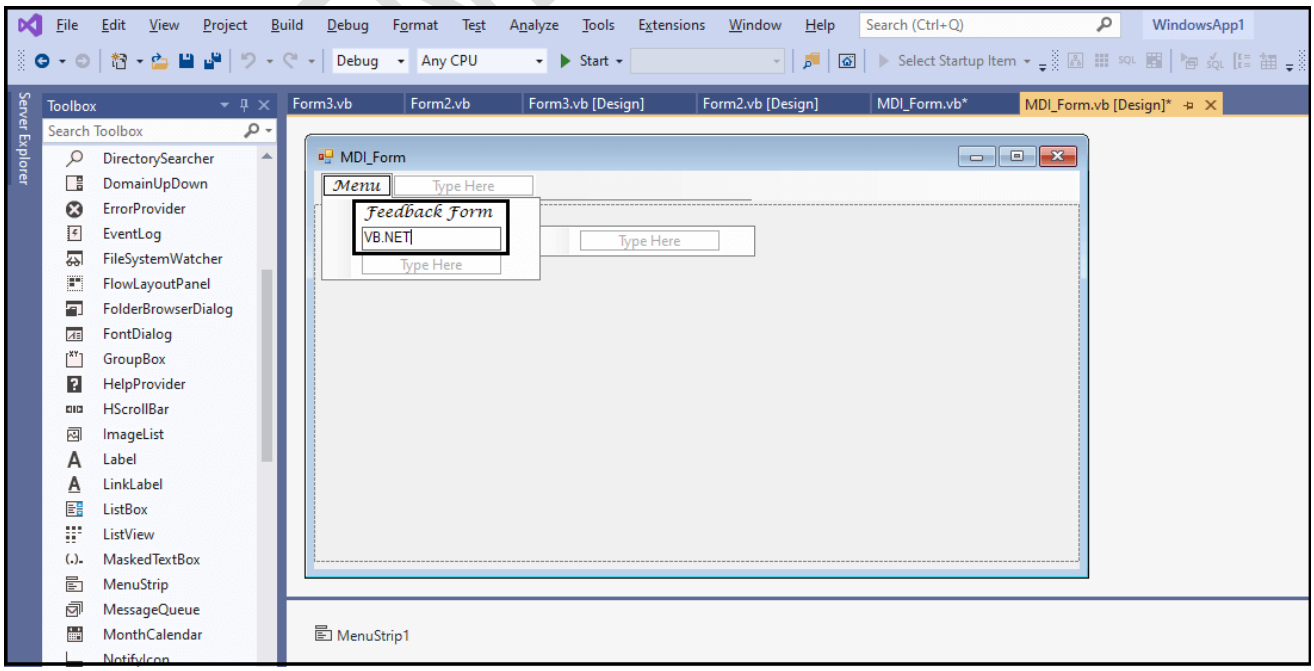
1. **MidParent:** The MidParent property is used to set a parent form to a child form.
2. **ActiveMdiChild:** The ActiveMdiChild property is used to get the reference of the current child form.
3. **IsMdiContainer:** The IsMdiContainer property set a Boolean value to True that represents the creation of a form as an MDI form.
4. **LayoutMdi():** The LayoutMdi() method is used to arrange the child forms in the parent or main form.
5. **Controls:** It is used to get the reference of control from the child form.

Let's create a program to display the multiple windows in the VB.NET Windows Forms.

Step 1: First, we have to open the Windows form and create the Menu bar with the use of MenuStrip control, as shown below.



Step 2: After creating the Menu, add the Subitems into the Menu bar, as shown below.



PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

In the above image, we have defined two Subitems, First is the **Feedback Form**, and the Second is VB.NET.

Step 3: In the third step, we will create two Forms: The Child Form of the **Main Form** or **Parent Form**.

Here, we have created the first Child Form with the name Form2.

Form2.vb

```
1. Public Class Form2
2.     Private Sub Form2_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.         Me.Text = "Feedback form" ' Set the title of the form
4.         Label1.Text = " Fill the Feedback form"
5.         Button1.Text = "Submit"
6.         Button1.BackColor = Color.SkyBlue
7.         Button2.Text = "Cancel"
8.         Button2.BackColor = Color.Red
9.     End Sub
10.    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
11.        MsgBox(" Successfully submit the feedback form")
12.    End Sub
13.    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
14.        Me.Dispose() ' end the form2
15.    End Sub
16. End Class
```

Another **Child Form** with the name **Form3**.

Form3.vb

```
1. Public Class Form3
2.     Private Sub Form3_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.         Label1.Text = "Welcome to JavaTpoint Tutorial Site"
4.         Label.BackColor = Color.Green
5.         Label2.Text = "This is the VB.NET Tutorial and we are learning the VB.NET MDI Form"
6.         Label2.BackColor = Color.SkyBlue
7.     End Sub
8. End Class
```

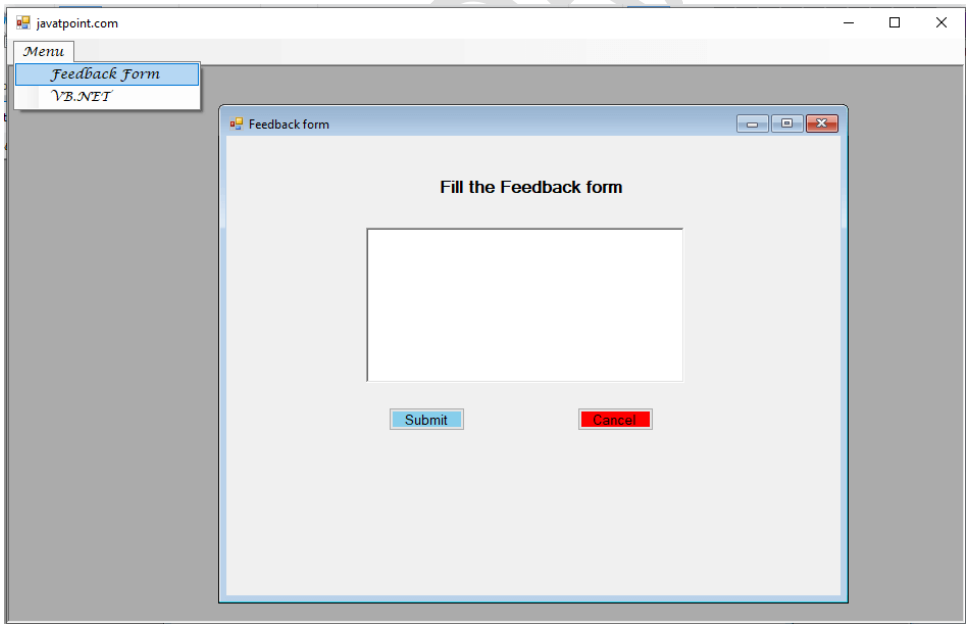
Step 4: Now we write the programming code for the Main or Parent Form, and here is the code for our Main Form.

MDI_form.vb

```
1. Public Class MDI_Form
2.     Private Sub MDI_Form_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.         IsMdiContainer = True 'Set the Boolean value to true to create the form as an MDI form
   .
4.         Me.Text = "javatpoint.com" 'set the title of the form
5.         PictureBox1.Image = Image.FromFile("C:\Users\AMIT YADAV\Desktop\jtp2.png")
6.         PictureBox1.Height = 550
7.         PictureBox1.Width = 750
8.     End Sub
9.     Private Sub FeedbackFormToolStripMenuItem_Click(sender As Object, e As EventArgs) Han
   dles FeedbackFormToolStripMenuItem.Click
10.        PictureBox1.Visible = False
11.        Dim fm2 As New Form2
12.        fm2.MdiParent = Me 'define the parent of form3, where Me represents the same form
13.        fm2.Show() 'Display the form3
14.    End Sub
15.    Private Sub VBNETToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles VB
   NETToolStripMenuItem.Click
16.        PictureBox1.Visible = False
17.        Dim fm3 As New Form3
18.        fm3.MdiParent = Me 'define the parent of form3, where Me represent the same form
19.        fm3.Show() 'Display the form3
20.    End Sub
21. End Class
```

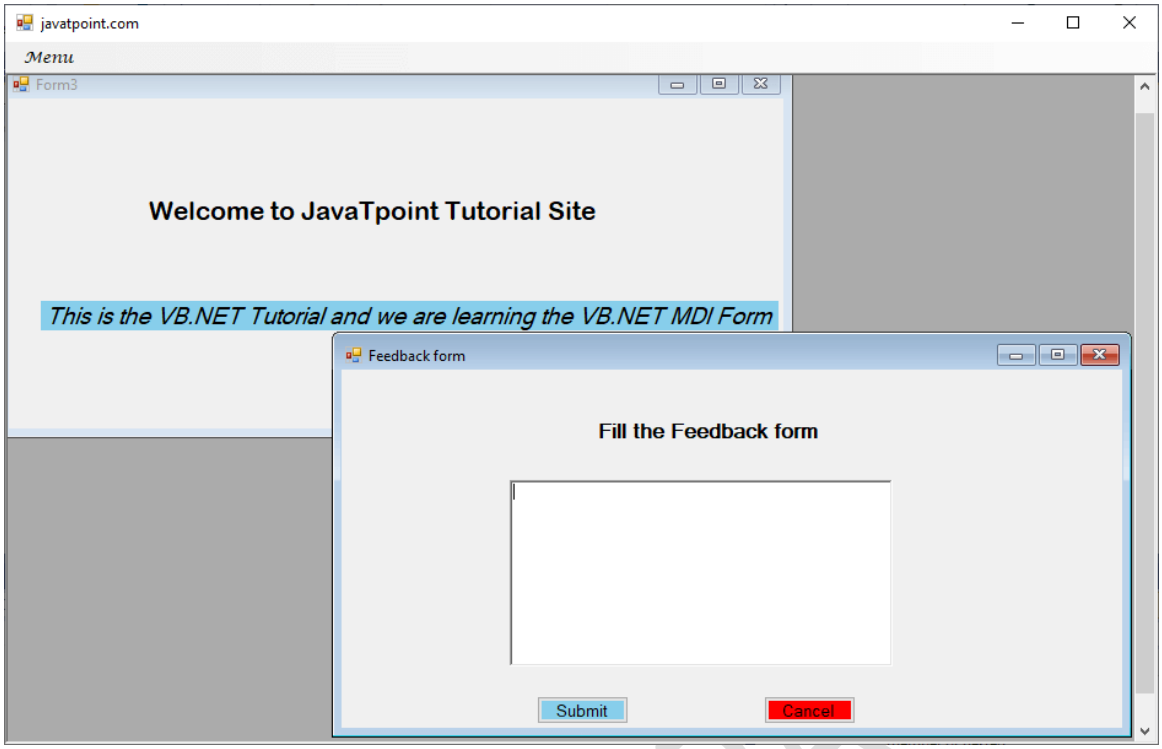
Output:

After that, click on the Menu button, it shows two sub-items of the Menu as **Feedback Form** and **VB.NET**. We have clicked on the Feedback Form that displays the following form on the window.



When we click on the Menu item, it shows the following image on the screen.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



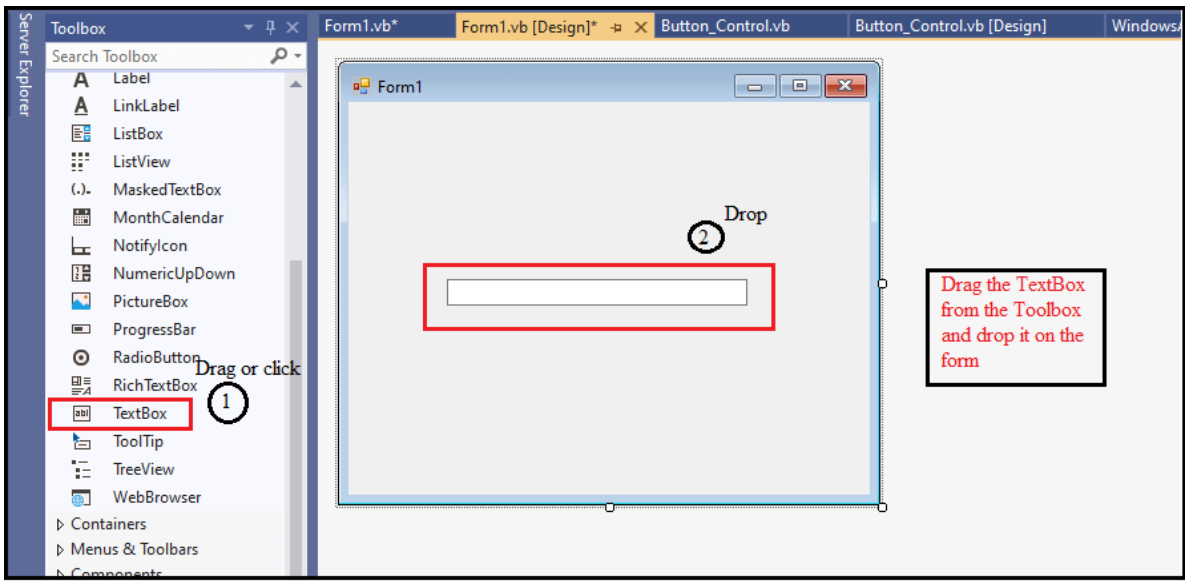
VB.NET TextBox Control

A TextBox control is used to display, accept the text from the user as an input, or a single line of text on a VB.NET Windows form at runtime. Furthermore, we can add multiple text and scroll bars in textbox control. However, we can set the text on the textbox that displays on the form.

Let's create a TextBox control in the VB.NET Windows form by using the following steps:

Step 1: We have to drag the TextBox control from the Toolbox and drop it on the Windows form, as shown below.

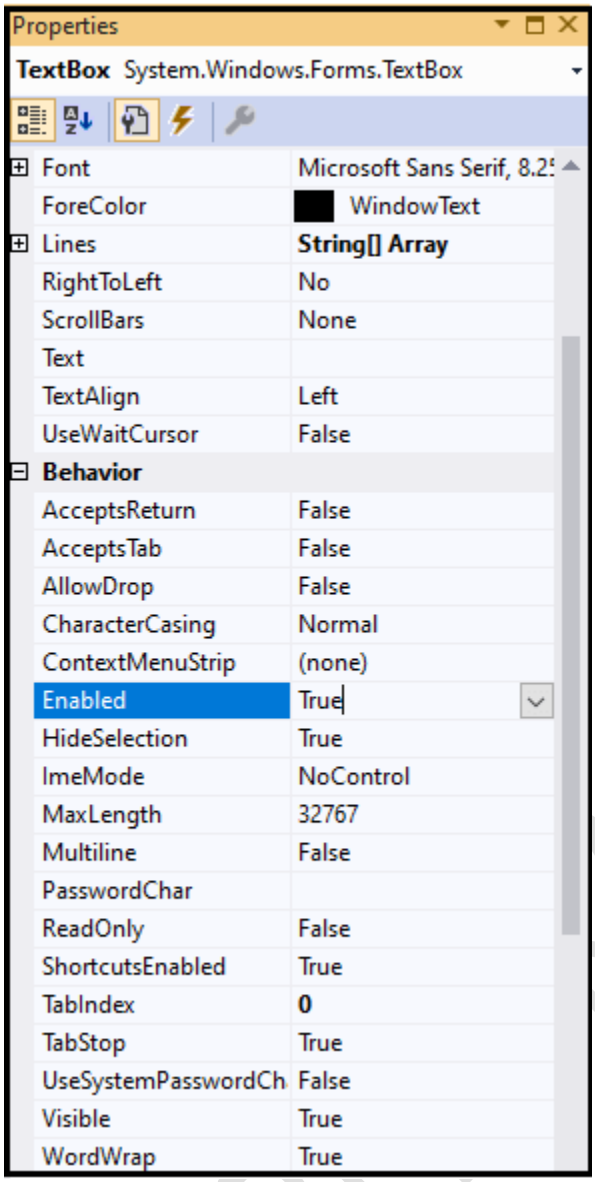
PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



Step 2: Once the TextBox is added to the form, we can set various properties of the TextBox by clicking on the TextBox control.

VB.NET TextBox Properties

There are following properties of the TextBox control.



Properties	Description
AutoCompleteMode	It is used to get or set a value that indicates how the automatic completion works for the textbox control.
Font	It is used to set the font style of the text displayed on a Windows form.
Lines	It is used to set the number of lines in a TextBox control.
CharacterCasing	It is used to get or set a value representing whether the TextBox control can modify the character's case as they typed.

Multiline	It is used to enter more than one line in a TextBox control, by changing the Multiline property value from False to True.
AcceptsReturn	It is used to get or set a value that indicates whether pressing the enter button in a multiline textbox; it creates a new line of text in control.
PasswordChar	It is used to set the password character that can be a mask in a single line of a TextBox control.
PreferredHeight	It is used to set the preferred height of the textbox control in the window form.
ScrollBars	It is used to display a scrollbar on a multiline textbox by setting a value for a Textbox control.
Text	It is used to get or set the text associated with the textbox control.
Visible	The Visible property sets a value that indicates whether the textbox should be displayed on a Windows Form.
WordWrap	The WordWrap properties validate whether the multiline Textbox control automatically wraps words to the beginning of the next line when necessary.

VB.NET TextBox Events

Events	Description
Click	When a textbox is clicked, a click event is called in the textbox control.
CausesValidationChanged	It occurs in the TextBox Control when the value of CauseValidation property is changed.
AcceptTabsChanged	It is found in the TextBox control when the property value of the AcceptTab is changed.
BackColorChanged	It is found in the TextBox Control when the property value of

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

	the BackColor is changed.
BorderStyleChanged	It is found in the TextBox Control when the value of the BorderStyle is changed.
ControlAdded	It is found when the new control is added to the Control.ControlCollection.
CursorChanged	It is found in TextBox, when the textbox control is removed from the Control.ControlCollection.
FontChanged	It occurs when the property of the Font is changed.
GetFocus	It is found in TextBox control to get the focus.
MouseClick	A MouseClick event occurs when the mouse clicks the control.
MultilineChanged	It is found in a textbox control when the value of multiline changes.

Furthermore, we can also refer to the VB.NET Microsoft documentation to get a complete list of TextBox properties and events.

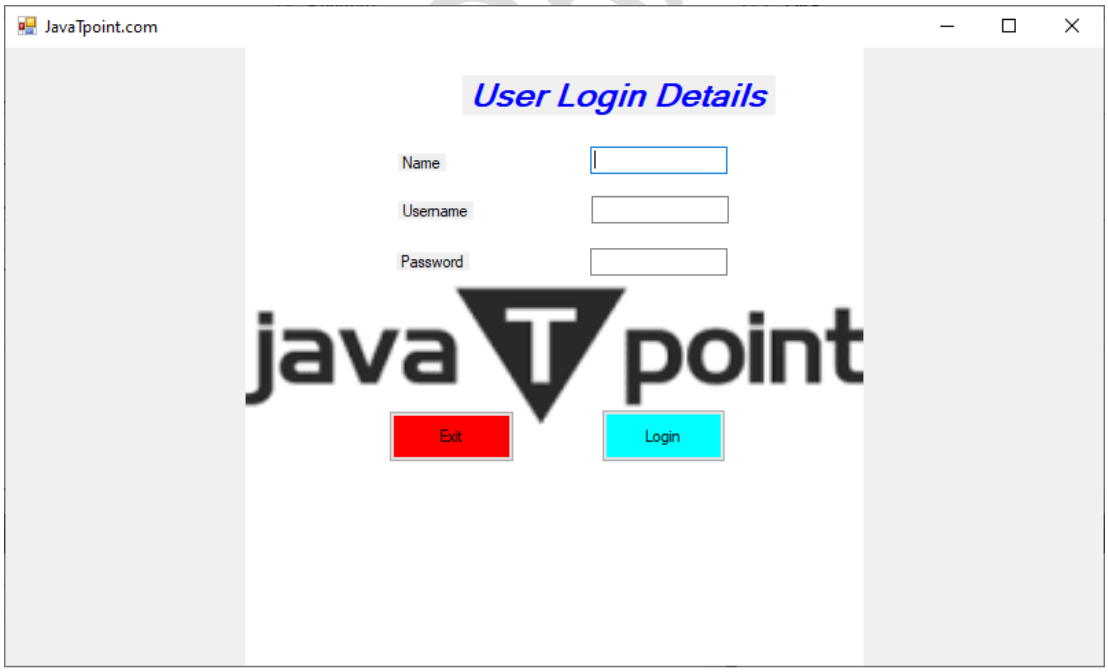
Let's create a program that displays the login details.

JavatPoint1.vb

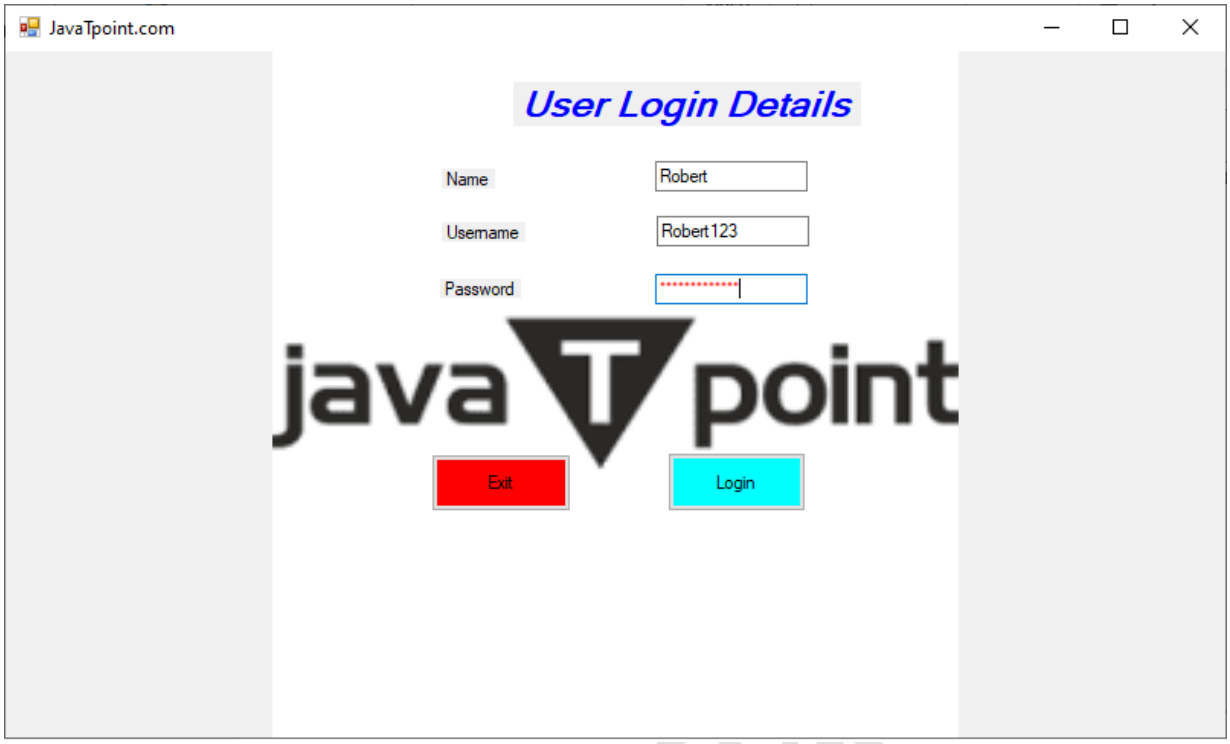
```
1.  Public Class JavatPoint1
2.      Private Sub JavatPoint1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.          Me.Text = "JavaTpoint.com" ' title name
4.          Label1.Text = "User Login Details" ' Set the title name for Label1
5.          Label2.Text = "Name"             ' Set the name for label2
6.          Label3.Text = "Username"  ' Set the username for label2
7.          Label4.Text = "Password"     ' Set the label name Passowrd
8.          Text3.PasswordChar = "*"
9.          Button1.Text = "Login"  ' Set the name of Button1 as Login
10.         Button2.Text = "Exit"       ' Set the name of Button2 As Exit
11.     End Sub
```

```
12.
13.     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
14.         End ' terminate the program when the user clinks button 2
15.     End Sub
16.
17.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
18.         Dim name As String
19.         Dim Uname As String
20.         Dim pass As String
21.         name = text1.Text
22.         Uname = Text2.Text
23.         pass = Text3.Text
24.         ' Display the user details, when the Button1 is clicked
25.         MsgBox(" Your Name: " & name + vbCrLf + "Your UserName: " & Uname + vbCrLf + "Y
    our Password: " & pass)
26.     End Sub
27. End Class
```

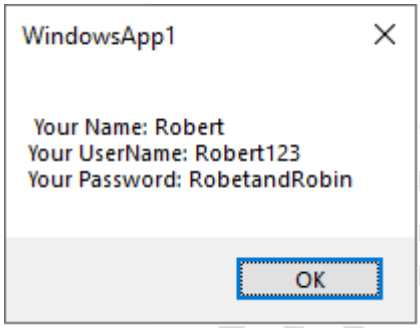
Output:



Now enter all the details of the User Login form, it shows the following image, as shown below.



Now, click on the **Login** button. It shows all the details filled by the user in the form.



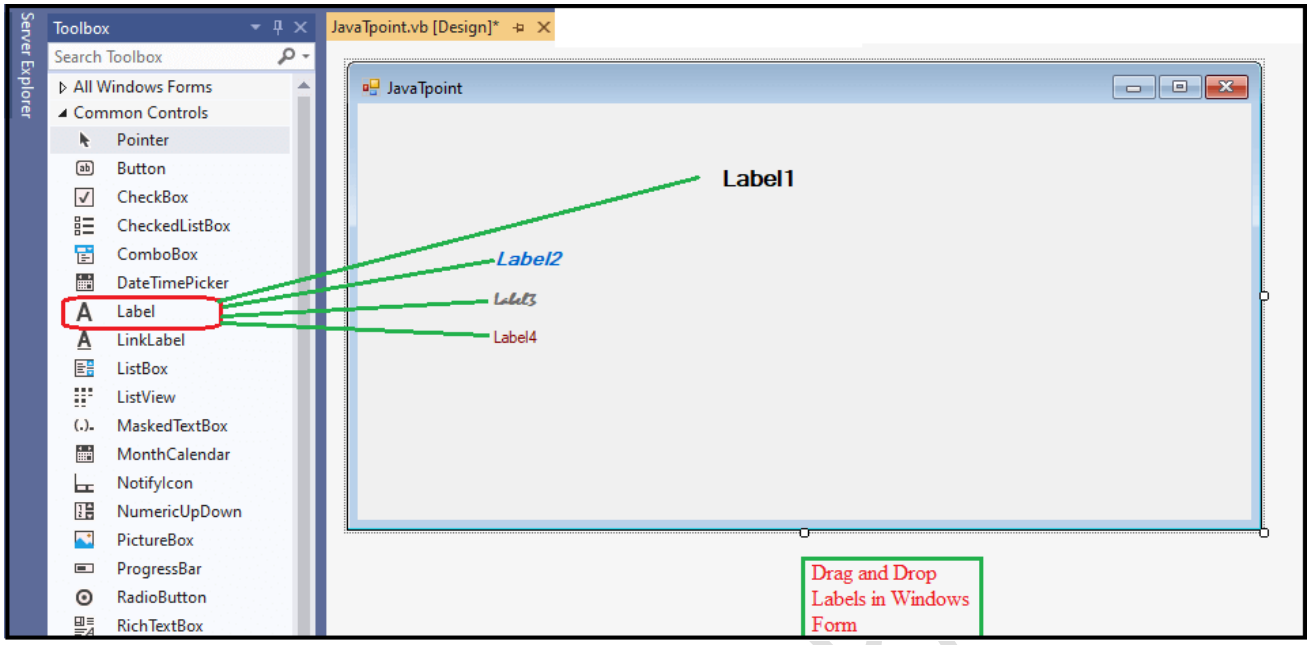
The Exit button in the form used to terminate the program.

VB.NET Label Control

In VB.NET, a label control is used to display descriptive text for the form in control. It does not participate in user input or keyboard or mouse events. Also, we cannot rename labels at runtime. The labels are defined in the class System.Windows.Forms namespace.

Let's create a label in the VB.NET Windows by using the following steps:

Step 1: We have to drag the Label control from the Toolbox and drop it on the Windows form, as shown below.



Step 2: Once the Label is added to the form, we can set various properties to the Label by clicking on the Label control.

VB.NET Label Properties

Properties	Description
AutoSize	As the name defines, an AutoSize property of label control is used to set or get a value if it is automatically resized to display all its contents.
Border Style	It is used to set the style of the border in the Windows form.
PreferredWidth	It is used to set or get the preferred width for the Label control.
Font	It is used to get or set the font of the text displayed on a Windows form.
PreferredHeight	It is used to set the height for the Label Control.
TextAlign	It is used to set the alignment of text such as centre, bottom, top, left, or right.
ForeColor	It is used to set the color of the text.
Text	It is used to set the name of a label in the Windows Form.

ContextMenu	It is used to get or sets the shortcut menu associated with the Label control.
DefaultSize	It is used to get the default size of the Label control.
Image	It is used to set the image to a label in Windows Form.
ImageIndex	It is used to set the index value to a label control displayed on the Windows form.

VB.NET Label Events

Events	Description
AutoSizeChanged	An AutoSizeChanged event occurs in the Label control when the value of AutoSize property is changed.
Click	Click event is occurring in the Label Control to perform a click.
DoubleClick	When a user performs a double-clicked in the Label control, the DoubleClick event occurs.
GotFocus	It occurs when the Label Control receives focus on the Window Form.
Leave	The Leave event is found when the input focus leaves the Label Control.
TabIndexChanged	It occurs when the value of Tabindex property is changed in the Label control.
ControlRemoved	When the control is removed from the Control.ControlCollection, a ControlRemoved event, occurs.
TabStopChanged	It occurs when the property of TabStop is changed in the Label Control.
BackColorChanged	A BackColorChanged event occurs in the Label control when the value of the BackColor property is changed.
ControlAdded	When a new control is added to the Control.ControlCollection, a

	ControlAdded event occurs.
DragDrop	A DragDrop event occurs in the Label control when a drag and drop operation is completed.

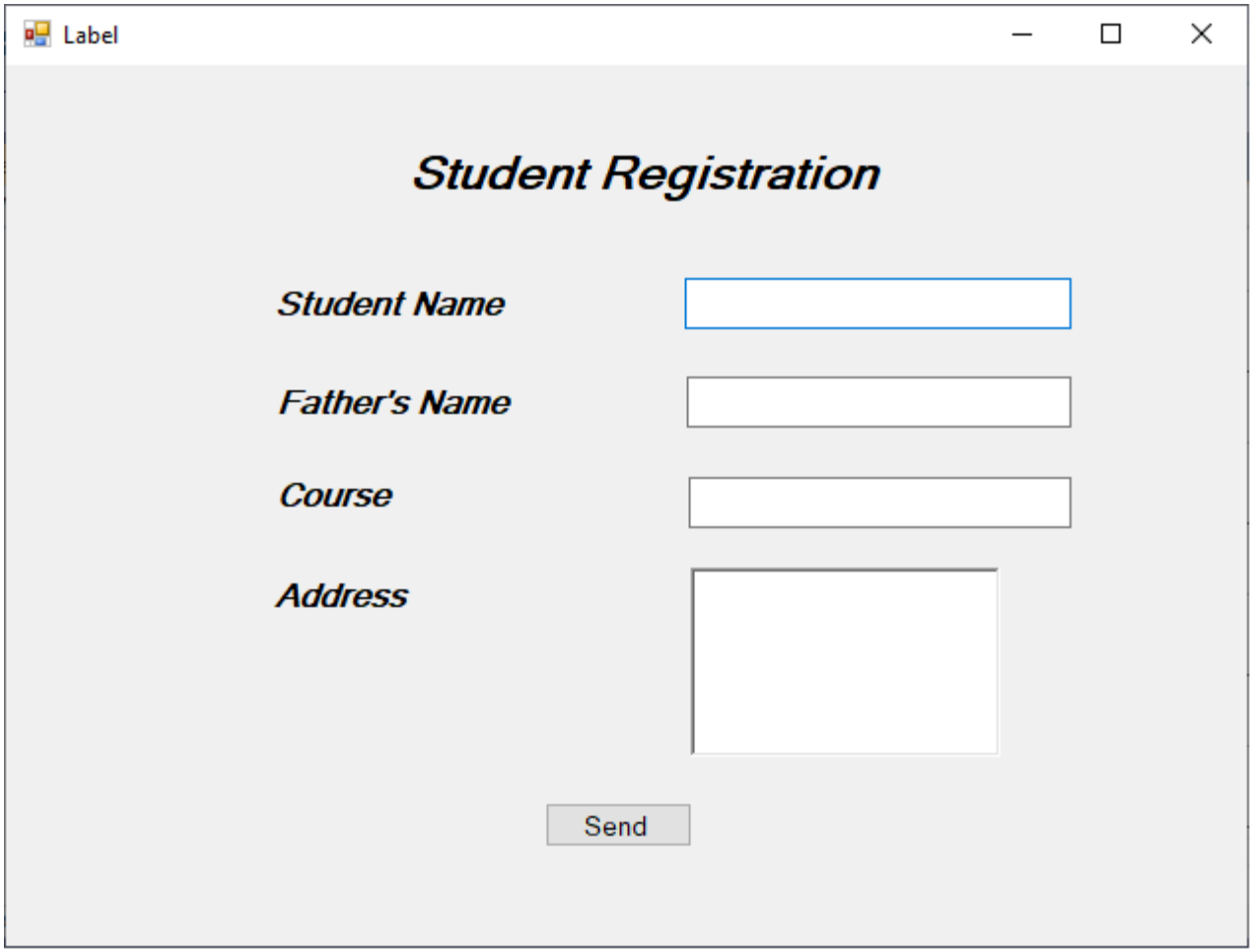
Furthermore, we can also refer to the VB.NET Microsoft documentation to get a complete list of Label properties and events.

Let's create a program to display the Label controls in VB.NET.

Label.vb

```
1. Public Class Label
2.     Private Sub Label_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.         Me.Text = "javatpoint.com" 'Set the title for a Windows Form
4.         Label1.Text = "Student Registration"
5.         Label1.Font = New Font("Microsoft Sans Serif", "style = Bold", "Italic", 18)  ' Set Font styl
   e
6.         Label2.Text = "Student Name"
7.         Label2.Font = New Font("Microsoft Sans Serif", "style = Bold", "Italic", 12)
8.         Label3.Text = "Father's Name"
9.         Label3.Font = New Font("Microsoft Sans Serif", "style = Bold", "Italic", 12)
10.        Label4.Text = "Course "
11.        Label4.Font = New Font("Microsoft Sans Serif", "style = Bold", "Italic", 12)
12.        Label5.Text = "Address"
13.        Label5.Font = New Font("Microsoft Sans Serif", "style = Bold", "Italic", 12)
14.        Button1.Text = "Send"
15.        TextBox1.Text = " "
16.        TextBox2.Text = " "
17.        TextBox3.Text = " "
18.        RichTextBox1.Text = " "
19.    End Sub
20. End Class
```

Output:



We have created 5 Labels on the Windows Form by using drag and drop operation in the above output.

Example2: Write a program to display only Labels on Windows forms.

Form1.vb

```
1. Public Class Form1
2.     Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.         'Set Text for Label1 and their property
4.         Label1.Text = ?Welcome to JavaTpoint?
5.         Label1.BorderStyle = BorderStyle.FixedSingle
6.         Label1.TextAlign = ContentAlignment.MiddleCenter
7.         'Set Text for Label2
8.         Label2.Text = ? VB.NET Label Control?
9.     End Sub
10. End Class
```

Output:

We can also load an image in the Label Control by using the following statement in the program.

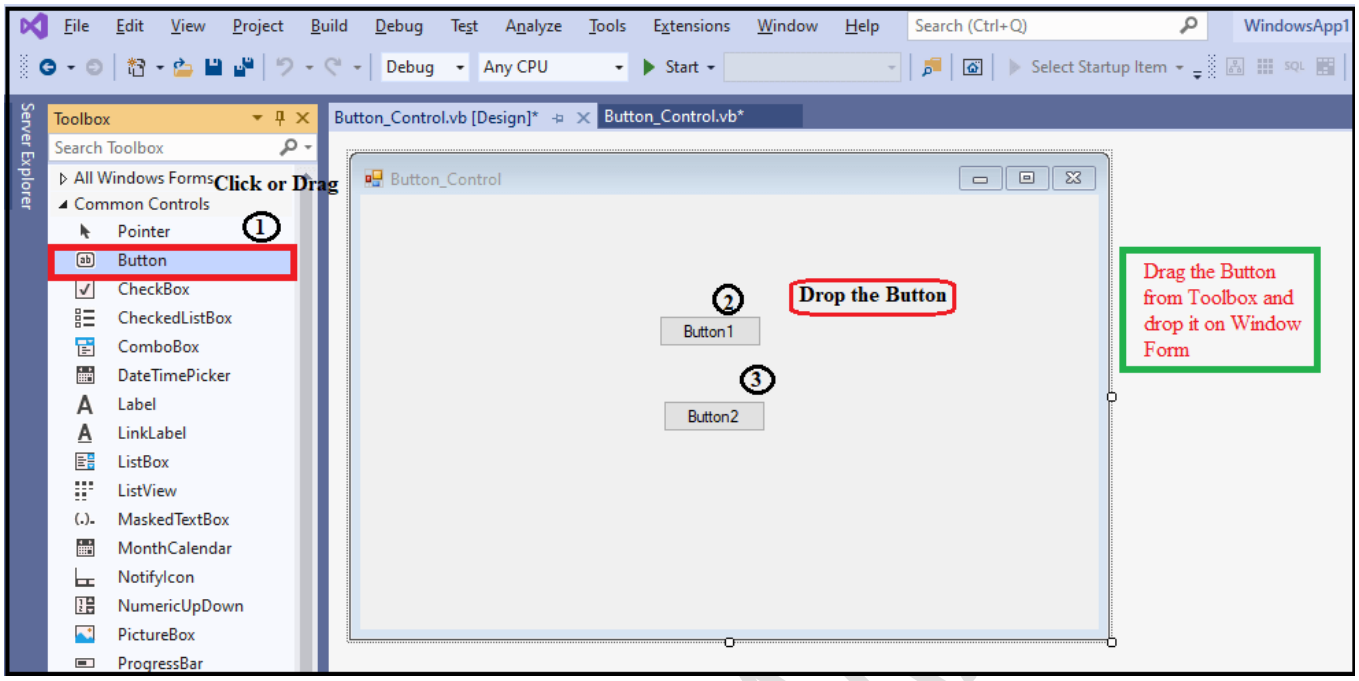
```
1. Label1.Image = Image.FromFile(/"C:\Desktop\mypic.jpg")
```

VB.NET Button Control

Button control is used to perform a click event in Windows Forms, and it can be clicked by a mouse or by pressing Enter keys. It is used to submit all queries of the form by clicking the submit button or transfer control to the next form. However, we can set the buttons on the form by using drag and drop operation.

Let's create a Button in VB.NET Windows form by using the following steps:

Step 1: We have to drag the Button control from the Toolbox and drop it on the Windows form, as shown below.



Step 2: Once the button is added to the form, we can set various properties of the Button by clicking on the Button control.

VB.NET Button Properties

Properties	Description
AutoSizeMode	It is used to get or set the auto mode value through which the button can automatically resize in the Windows form.
BackColor	It is used to set the background color of the Button in the Windows form.
BackgroundImage	It is used to set the background image of the button control.
ForeColor	It is used to set or get the foreground color of the button control.
Image	It is used to set or gets the image on the button control that is displayed.
Location	It is used to set the upper-left of the button control's coordinates relative to the upper-left corner in the windows form.
Text	It is used to set the name of the button control in the windows form.

AllowDrop	It is used to set or get a value representing whether the button control can accept data that can be dragged by the user on the form.
TabIndex	It is used to set or get the tab order of the button control within the form.

VB.NET Button Events

BackColorChanged	A BackColorChaged event is found in button control when the Background property is changed.
BackgroundImageChanged	A BackgoundImageChanged event is found in button control when the value of the BackgoundImage property is changed.
Click	A Click event is found in the button control when the control is clicked.
ContextManuChanged	It is found in button control when the value of the ContextMenu property is changed.
ControlAdded	A ControlAdded event is found in button control when a new control is added to the Control.ControlCollection.
CursorChanged	A CursorChanged event is found in button control when the value of the control is changed.
DoubleClick	When the user makes a double click on the button, a double click event is found in the button control.
TextChanged	It is found in the button control when the value of the text property is changed.
DragDrop	The DragDrop event is found in the button control when the drag and drop operation is completed in the Form.

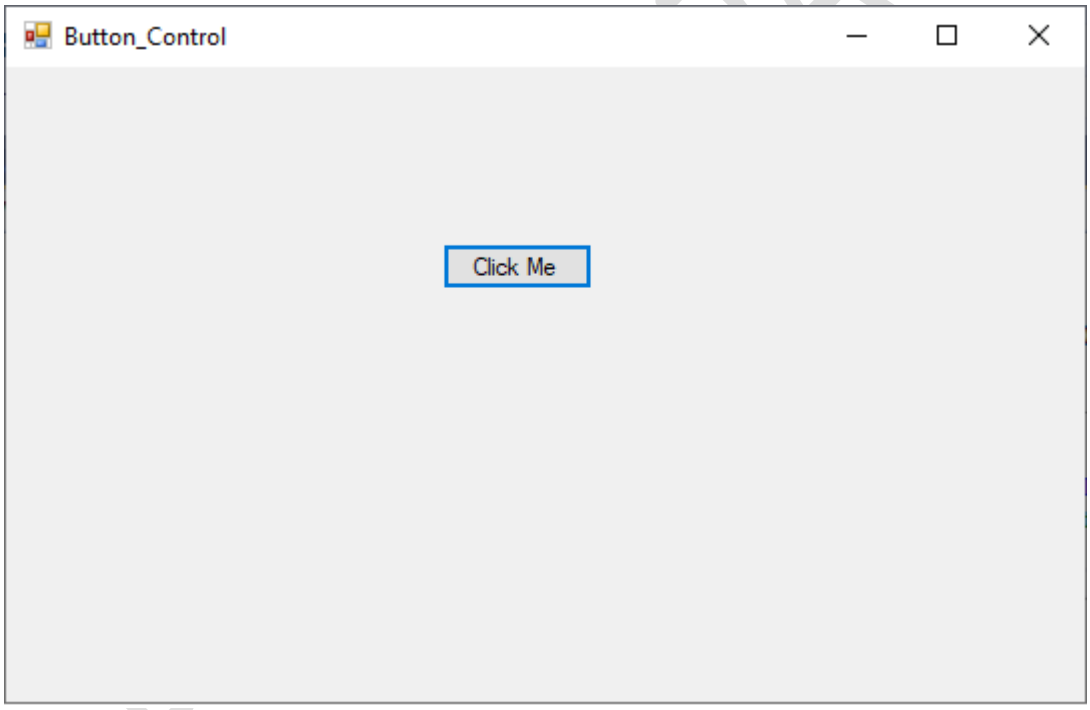
Furthermore, we can also refer to the VB.NET Microsoft documentation to get a complete list of Button properties and events.

Let's create a program to display a message on the Windows Forms using the button control in VB.NET.

Button_Control.vb

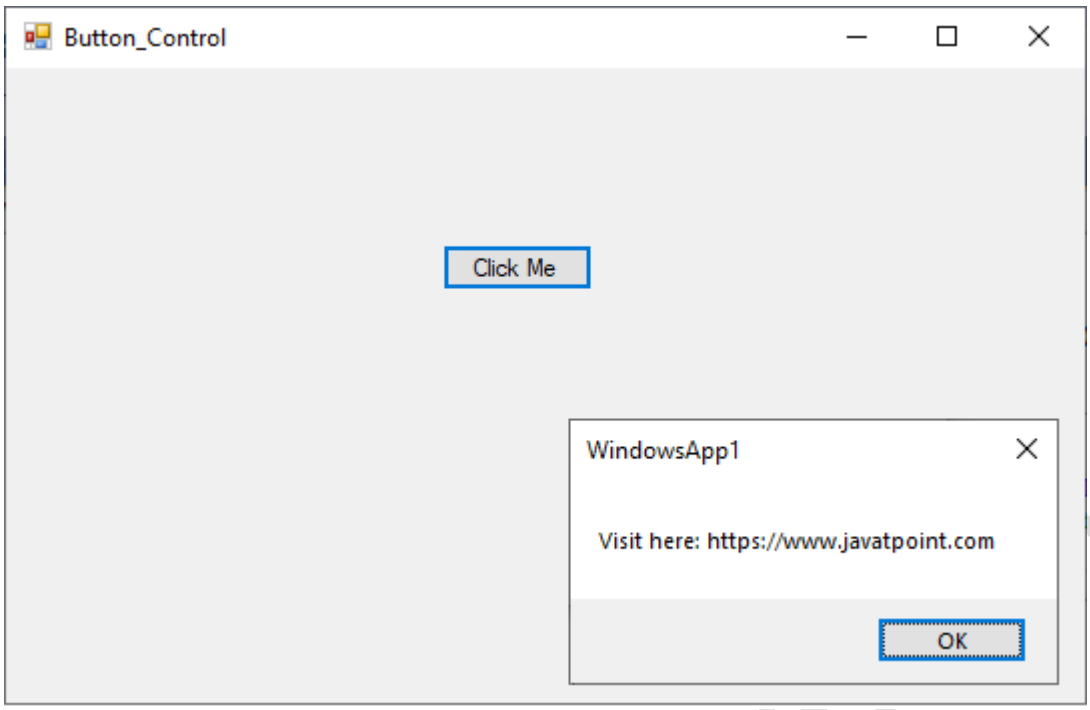
```
1. Public Class Button_Control
2.     Private Sub Button_Control_Load(sender As Object, e As EventArgs) Handles MyBase.Load

3.         Button1.Text = "Click Me" ' Set the name of button
4.     End Sub
5.
6.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
7.         MsgBox(" Visit here: https://www.javatpoint.com")
8.  ' Display the message, when a user clicks on Click me button
9.     End Sub
10. End Class
```

Output:



Now click on the **'Click Me'** button, it shows the following message on the form.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



Let's create another program that displays separate buttons on the form to perform different tasks.

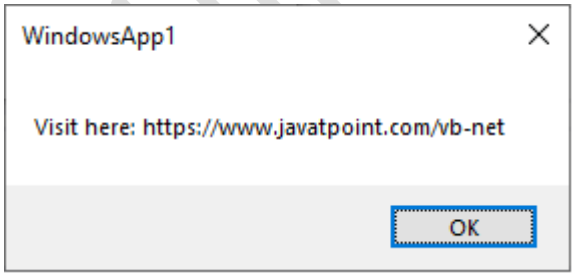
Button_Control.vb

```
1.  Public Class Button_Control
2.      Private Sub Button_Control_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.          ' Button1.Text = "Submit" ' Set the name of button
4.          Button2.Text = "Exit"
5.          Button3.Text = "Show Image"
6.      End Sub
7.
8.      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
9.          MsgBox(" Visit here: https://www.javatpoint.com/vb-
    net") ' Display the message, when a user click on Click me button
10.     End Sub
11.
12.     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
13.         End ' It is used for terminating the program.
14.     End Sub
15.
16.     Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
```

```
17.         PictureBox1.Visible = True
18.     End Sub
19.
20.     Private Sub PictureBox1_Click(sender As Object, e As EventArgs) Handles PictureBox1.Click

21.     End Sub
22. End Class
```

Output:

Now Click on the **Click Me** button, it shows the following message on the screen.



Now click on the **Show Image** button, it shows the following image on the screen.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

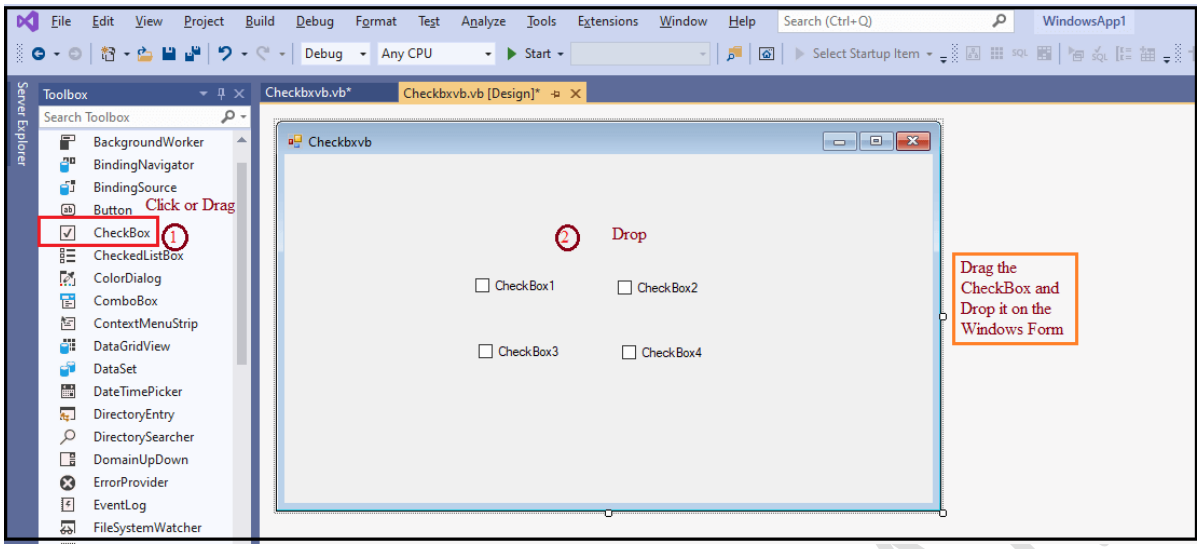
Click on the Exit button to terminate the program.

VB.NET CheckBox Control

The CheckBox control is a control that allows the user to select or deselect options from the available options. When a checkbox is selected, a tick or checkmark will appear on the Windows form.

Let's create a CheckBox control in the VB.NET Windows form using the following steps.

Step 1: We need to drag the CheckBox control from the toolbox and drop it to the Windows form, as shown below.



Step 2: Once the CheckBox is added to the form, we can set various properties of the checkbox by clicking on the CheckBox control.

CheckBox Properties

There are some properties of the VB.NET CheckBox control.

Property	Description
Default	It is used to get the default size of the checkbox.
AutoCheck	The AutoCheck property is used to check whether the checked value or appearance of control can be automatically changed when the user clicked on the CheckBox control.
CheckAlign	It is used to set the checkmark's alignment, such as horizontal or vertical on the checkbox.
Appearance	The Appearance property is used to display the appearance of a checkbox control by setting a value.
CheckState	The CheckState property is used to verify whether the checkbox status is checked in the window form.
ThreeState	The ThreeState property is used to check whether the control allows one to set three check positions instead of two by setting values.

FlatStyle	It is used to obtain or set the flat appearance of a checkbox.

CheckBox Methods

There are some Methods of the VB.NET CheckBox control.

Method	Description
OnClick	The OnClick method is used to fetch the Click event in the CheckBox control.
OnCheckStateChanged	It is used to call the CheckStateChanged event in the CheckBox control.
ToString	The ToString method is used to return the current string of the CheckBox control.
OnCheckedChanged	When the Checked property is changed in the CheckBox control, the OnCheckedChanged events occur.
OnMouseUp	It is used when it receives the OnMouseUp event in the CheckBox control.

CheckBox Events

There are some Events of the VB.NET CheckBox control.

Event	Description
CheckedChanged	The CheckedChanged event is found when the value of the checked property is changed to CheckBox.
DoubleClick	It occurs when the user performs a double click on the CheckBox control.
CheckStateChanged	It occurs when the value of the CheckState property changes to the CheckBox control.

AppearanceChanged	It occurs when the property of the Appearance is changed to the CheckBox control.

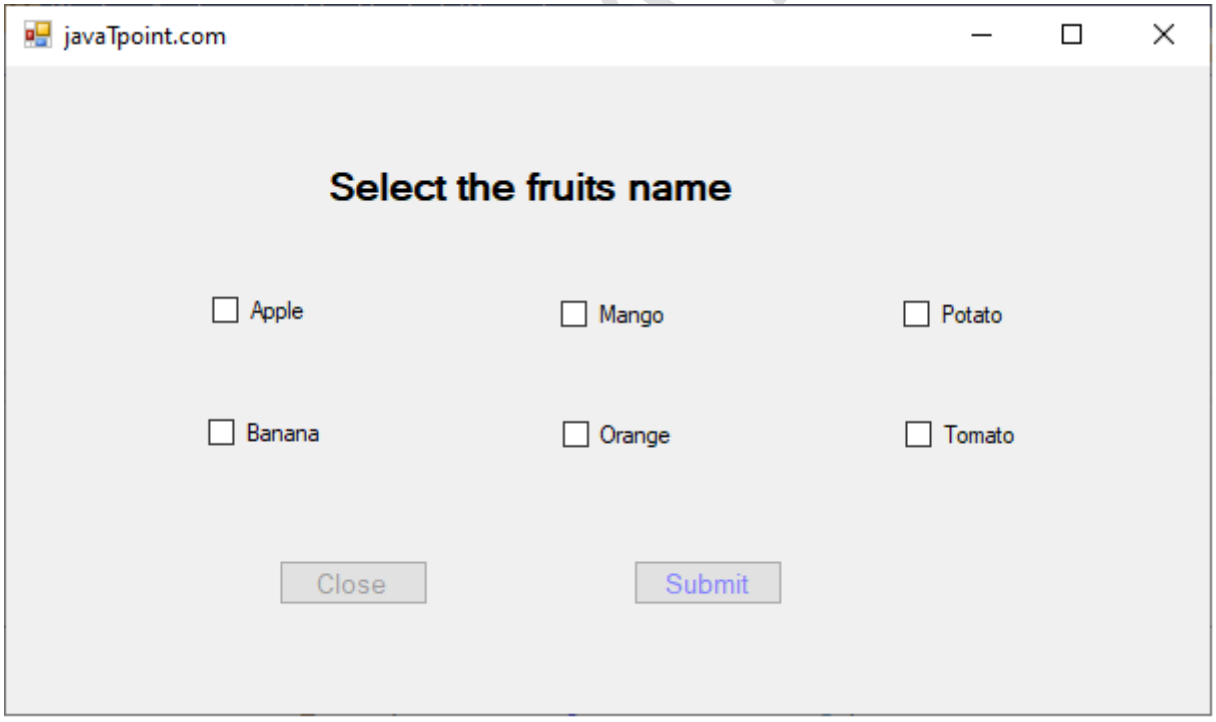
Let's create a program to understand the uses of CheckBox control in the VB.NET form.

Checkbx.vb

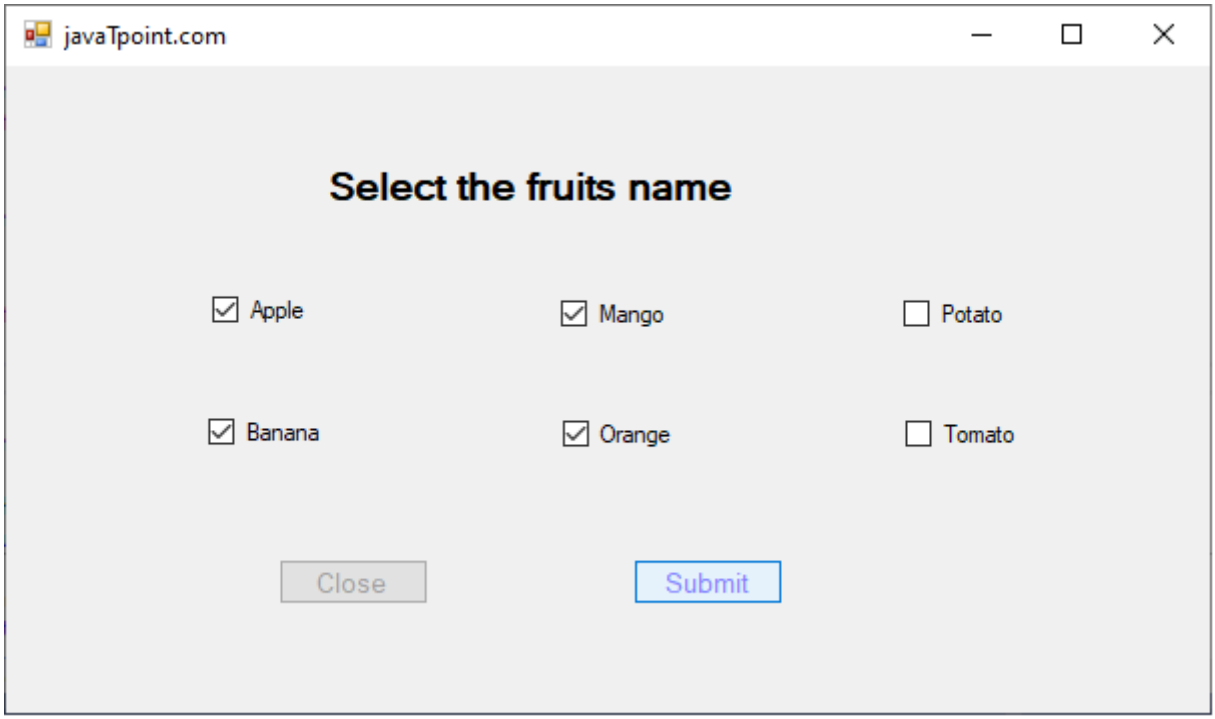
```
1.  Public Class Checkbxvb
2.      Private Sub Checkbxvb_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.          Me.Text = "saicollege.com" ' Set the title name of the form
4.          Label1.Text = "Select the fruits name"
5.          CheckBox1.Text = "Apple"
6.          CheckBox2.Text = "Mango"
7.          CheckBox3.Text = "Banana"
8.          CheckBox4.Text = "Orange"
9.          CheckBox5.Text = "Potato"
10.         CheckBox6.Text = "Tomato"
11.         Button1.Text = "Submit"
12.         Button2.Text = "Close"
13.     End Sub
14.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
15.         Dim fruit As String
16.         fruit = " "
17.         If CheckBox1.Checked = True Then
18.             fruit = "Apple"
19.         End If
20.         If CheckBox2.Checked = True Then
21.             'fruit = CheckBox2.Text
22.             fruit = fruit & " Mango"
23.         End If
24.         If CheckBox3.Checked = True Then
25.             fruit = fruit & " Banana"
26.         End If
27.         If CheckBox4.Checked = True Then
28.             fruit = fruit & " Orange"
29.         End If
```

```
30.         If CheckBox5.Checked = True Then
31.             fruit = fruit & " Potato"
32.         End If
33.         If CheckBox6.Checked = True Then
34.             fruit = fruit & " Tomato"
35.         End If
36.         If fruit.Length <> 0 Then
37.             MsgBox(" Selected items " & fruit)
38.         End If
39.         CheckBox1.ThreeState = True
40.     End Sub
41.     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
42.         End 'terminate the program
43.     End Sub
44. End Class
```

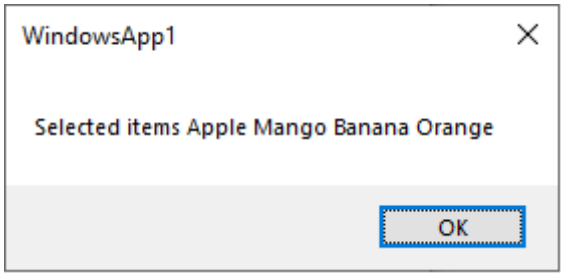
Output:



Now select the fruit name by clicking on the items.



Now click on the **Submit** button, it displays the following output on your screen.

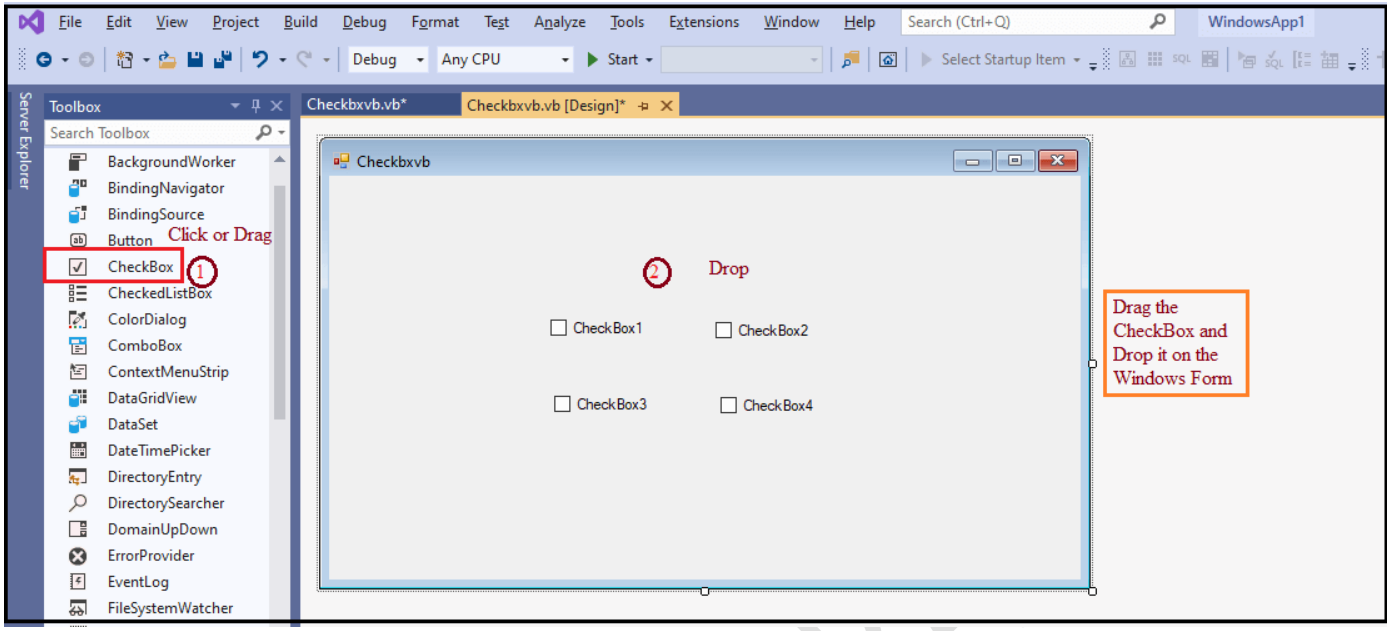


VB.NET CheckBox Control

The CheckBox control is a control that allows the user to select or deselect options from the available options. When a checkbox is selected, a tick or checkmark will appear on the Windows form.

Let's create a CheckBox control in the VB.NET Windows form using the following steps.

Step 1: We need to drag the CheckBox control from the toolbox and drop it to the Windows form, as shown below.



Step 2: Once the CheckBox is added to the form, we can set various properties of the checkbox by clicking on the CheckBox control.

CheckBox Properties

There are some properties of the VB.NET CheckBox control.

Property	Description
Default	It is used to get the default size of the checkbox.
AutoCheck	The AutoCheck property is used to check whether the checked value or appearance of control can be automatically changed when the user clicked on the CheckBox control.
CheckAlign	It is used to set the checkmark's alignment, such as horizontal or vertical on the checkbox.
Appearance	The Appearance property is used to display the appearance of a checkbox control by setting a value.
CheckState	The CheckState property is used to verify whether the checkbox status is checked in the window form.

ThreeState	The ThreeState property is used to check whether the control allows one to set three check positions instead of two by setting values.
FlatStyle	It is used to obtain or set the flat appearance of a checkbox.

CheckBox Methods

There are some Methods of the VB.NET CheckBox control.

Method	Description
OnClick	The OnClick method is used to fetch the Click event in the CheckBox control.
OnCheckStateChanged	It is used to call the CheckStateChanged event in the CheckBox control.
ToString	The ToString method is used to return the current string of the CheckBox control.
OnCheckedChanged	When the Checked property is changed in the CheckBox control, the OnCheckedChanged events occur.
OnMouseUp	It is used when it receives the OnMouseUp event in the CheckBox control.

CheckBox Events

There are some Events of the VB.NET CheckBox control.

Event	Description
CheckedChanged	The CheckedChanged event is found when the value of the checked property is changed to CheckBox.
DoubleClick	It occurs when the user performs a double click on the CheckBox control.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

CheckStateChanged	It occurs when the value of the CheckState property changes to the CheckBox control.
AppearanceChanged	It occurs when the property of the Appearance is changed to the CheckBox control.

Let's create a program to understand the uses of CheckBox control in the VB.NET form.

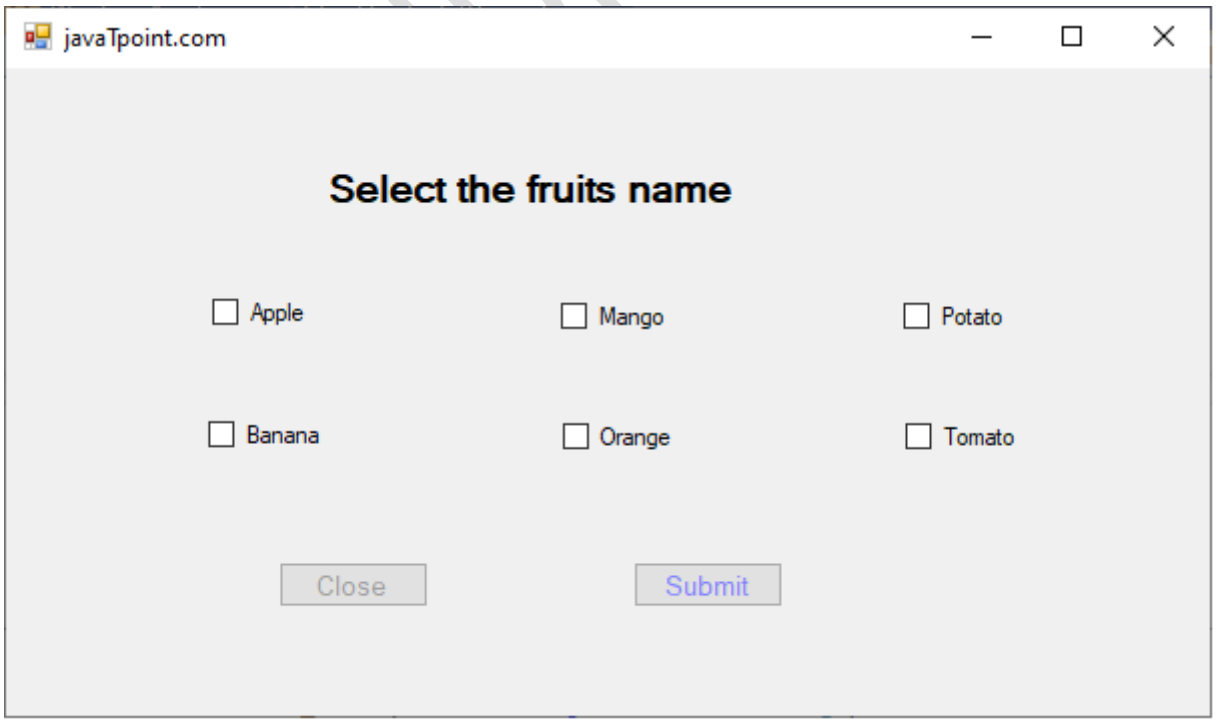
Checkbx.vb

```
1.  Public Class Checkbxvb
2.      Private Sub Checkbxvb_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.          Me.Text = "javaTpoint.com" ' Set the title name of the form
4.          Label1.Text = "Select the fruits name"
5.          CheckBox1.Text = "Apple"
6.          CheckBox2.Text = "Mango"
7.          CheckBox3.Text = "Banana"
8.          CheckBox4.Text = "Orange"
9.          CheckBox5.Text = "Potato"
10.         CheckBox6.Text = "Tomato"
11.         Button1.Text = "Submit"
12.         Button2.Text = "Close"
13.     End Sub
14.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
15.         Dim fruit As String
16.         fruit = " "
17.         If CheckBox1.Checked = True Then
18.             fruit = "Apple"
19.         End If
20.         If CheckBox2.Checked = True Then
21.             'fruit = CheckBox2.Text
22.             fruit = fruit & " Mango"
23.         End If
24.         If CheckBox3.Checked = True Then
25.             fruit = fruit & " Banana"
26.         End If
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

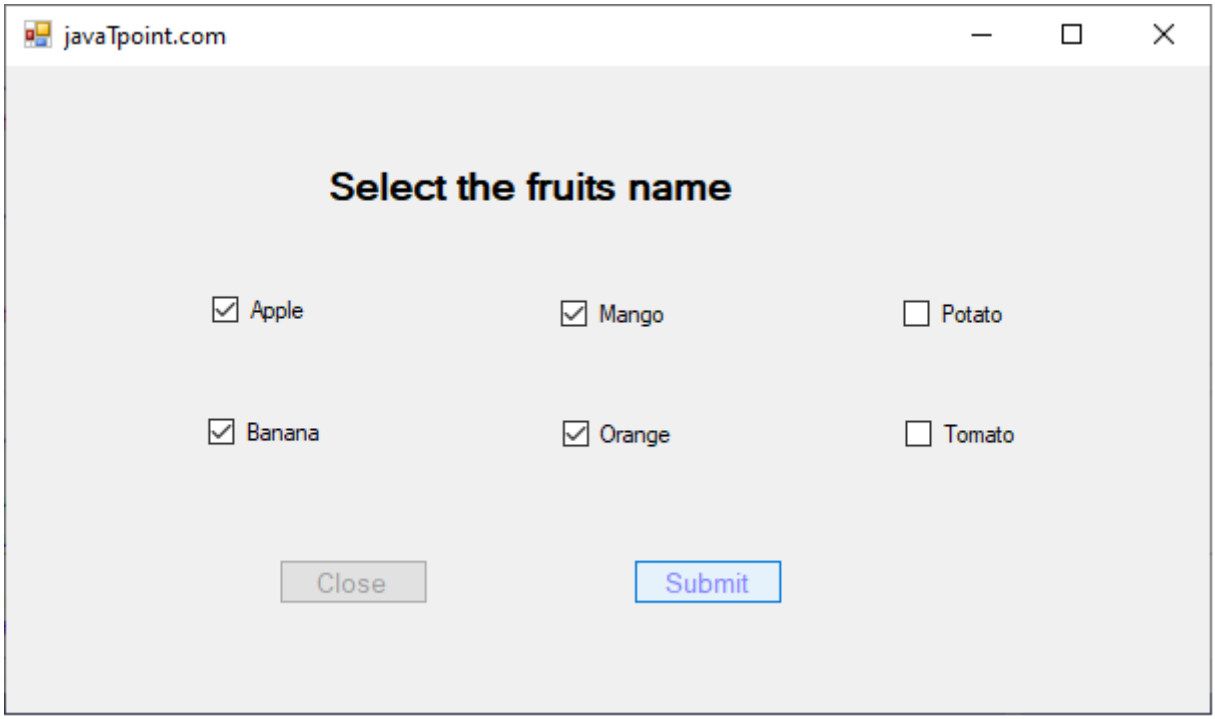
```
27.         If CheckBox4.Checked = True Then
28.             fruit = fruit & " Orange"
29.         End If
30.         If CheckBox5.Checked = True Then
31.             fruit = fruit & " Potato"
32.         End If
33.         If CheckBox6.Checked = True Then
34.             fruit = fruit & " Tomato"
35.         End If
36.         If fruit.Length <> 0 Then
37.             MsgBox(" Selected items " & fruit)
38.         End If
39.         CheckBox1.ThreeState = True
40.     End Sub
41.     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
42.         End 'terminate the program
43.     End Sub
44. End Class
```

Output:

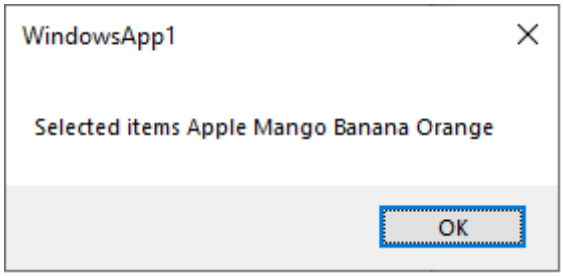


Now select the fruit name by clicking on the items.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



Now click on the **Submit** button, it displays the following output on your screen.



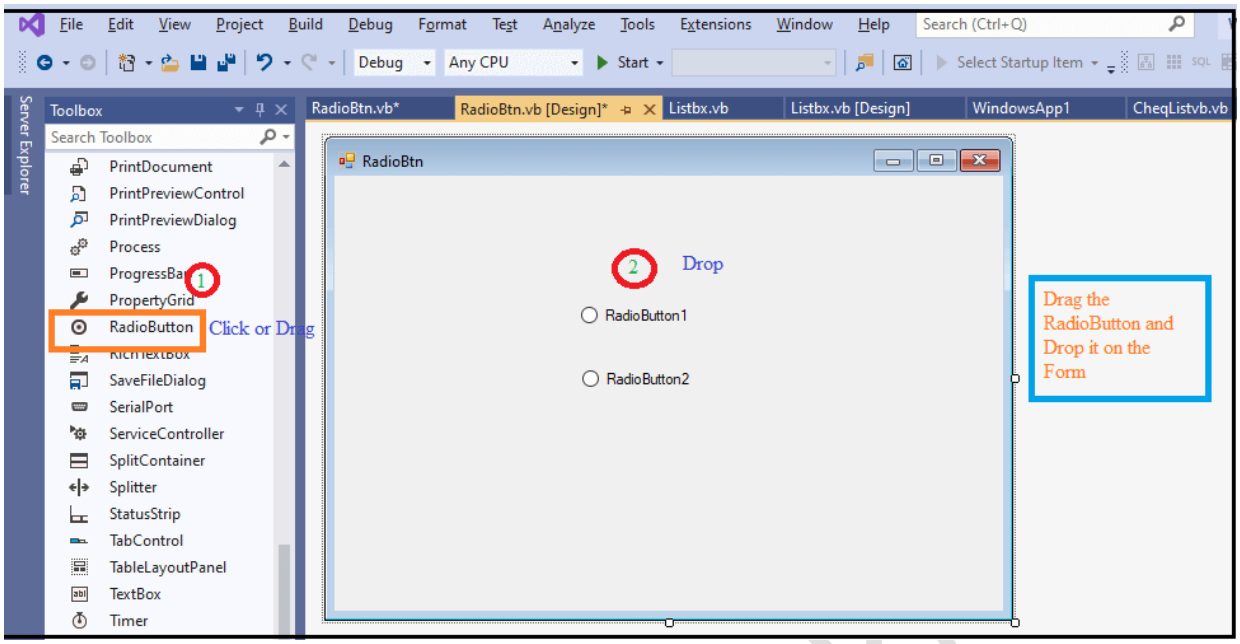
RadioButton Control

The **RadioButton** is used to select one option from the number of choices. If we want to select only one item from a related or group of items in the windows forms, we can use the radio button. The **RadioButton** is mutually exclusive that represents only one item is active and the remains unchecked in the form.

Let's create a **RadioButton** control in the VB.NET Windows by using the following steps.

Step 1: Drag the **RadioButton** control from the toolbox and drop it to the Windows form, as shown below.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



Step 2: Once the RadioButton is added to the form, we can set various properties of the RadioButton by clicking on the Radio control.

RadioButton Properties

There are following properties of the VB.NET RadioButton control.

Property	Description
AllowDrop	It is used to set or get a value representing whether the RadioButton allows the user to drag on the form.
Appearance	It is used to get or set a value that represents the appearance of the RadioButton.
AutoScrollOffset	It is used to get or set the radio control in ScrollControlIntoView(Control).
AutoCheck	The AutoCheck property is used to check whether the checked value or appearance of control can be automatically changed when the user clicked on the RadioButton control.
AutoSize	The AutoSize property is used to check whether the radio control can be automatically resized by setting a value in the RadioButton control.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

CanSelect	A CanSelect property is used to validate whether a radio control can be selected by setting a value in the RadioButton control.
CheckAlign	It is used to obtain or set a value that indicates the location of the check portion in the radioButton control.
Text	The Text property is used to set the name of the RadioButton control.

RadioButton Methods

Method Name	Description
Contains(Control)	The Contains() method is used to check if the defined control is available in the RadioButton control.
DefWndProc(Message)	It is used to send the specified message to the Window procedure.
DestroHandle()	It is used to destroy the handle associated with the RadioButton Control.
Focus()	The Focus() method is used to set the input focus to the window form's RadioButton control.
GetAutoSizeMode()	It is used to return a value that represents how the control will operate when the AutoSize property is enabled in the RadioButton control of the Window form.
ResetText()	As the name suggests, a ResetText() method is used to reset the property of text to its default value or empty.
Update()	It is used to reroute an invalid field, which causes control in the client region.

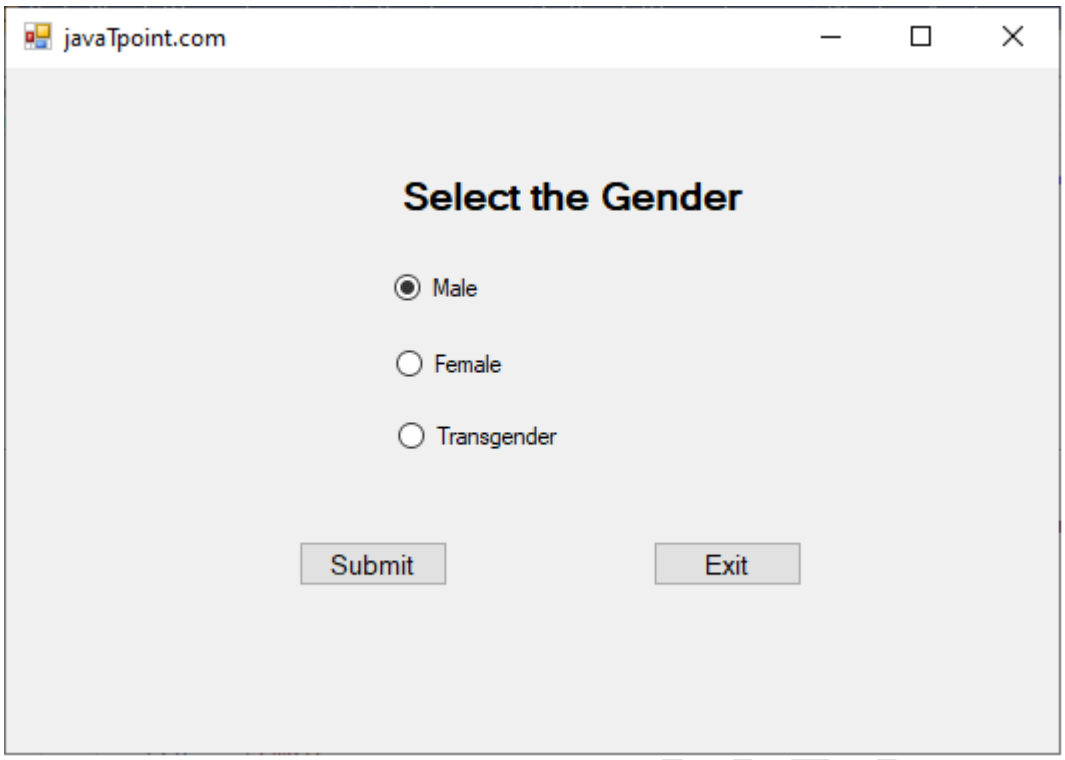
We can also refer to Microsoft documentation to get a complete list of RadioButton Control properties and methods in the VB .NET.

Let's create a program to understand the uses of Radio button control in the VB.NET form.

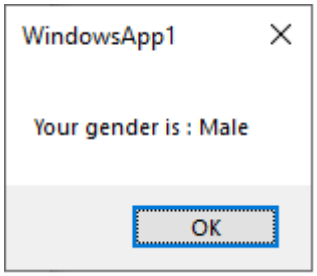
RadioBtn.vb

```
1.  Public Class RadioBtn
2.      Private Sub RadioBtn_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.          Me.Text = "saicollege.com" ' Set the title of the form
4.          Label1.Text = "Select the Gender"
5.          RadioButton1.Text = "Male"  ' Set the radiobutton1 and radiobutton2
6.          RadioButton2.Text = "Female"
7.          RadioButton3.Text = "Transgender"
8.          Button1.Text = "Submit"    ' Set the button name
9.          Button2.Text = "Exit"
10.     End Sub
11.
12.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
13.         Dim gen As String
14.         If RadioButton1.Checked = True Then
15.             gen = "Male"
16.             MsgBox(" Your gender is : " & gen)
17.
18.         ElseIf RadioButton2.Checked = True Then
19.             gen = "Female"
20.             MsgBox(" Your gender is : " & gen)
21.         Else
22.             gen = "Transgender"
23.             MsgBox(" You have Selected the gender : " & gen)
24.         End If
25.
26.     End Sub
27.
28.     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
29.         End 'Terminate the program
30.     End Sub
31. End Class
```

Output:



Click on the **Submit** button. It shows the following message on the screen.

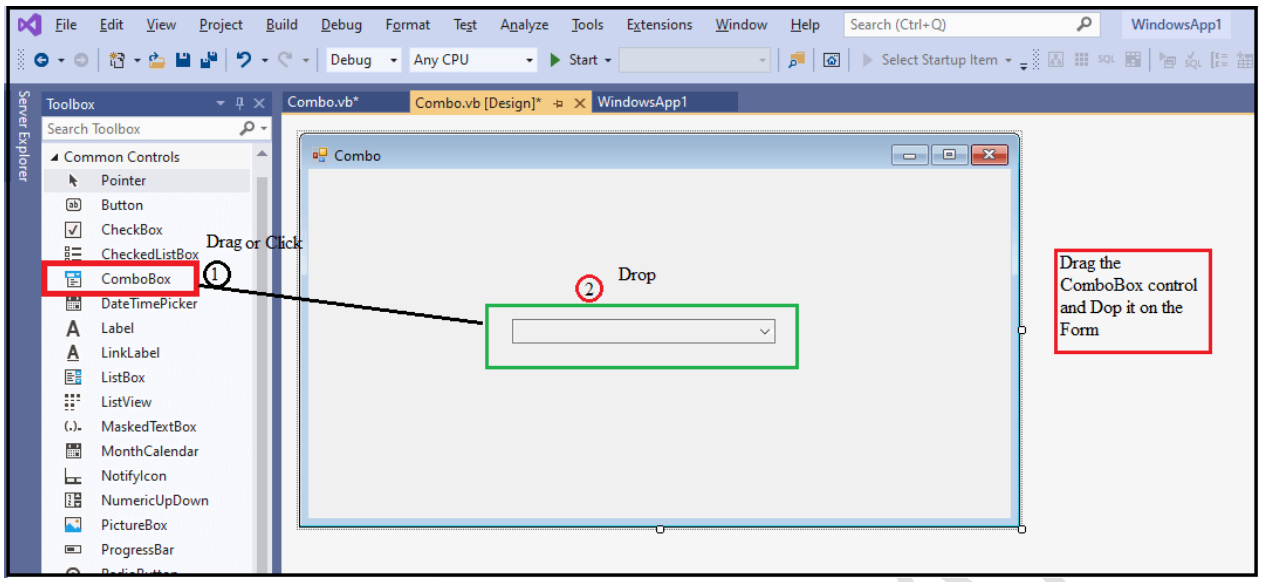


VB.NET ComboBox Control

The **ComboBox** control is used to display more than one item in a drop-down list. It is a combination of **Listbox** and **Textbox** in which the user can input only one item. Furthermore, it also allows a user to select an item from a drop-down list.

Let's create a ComboBox control in the VB.NET Windows by using the following steps.

Step 1: We need to drag the combo box control from the toolbox and drop it to the Windows form, as shown below.

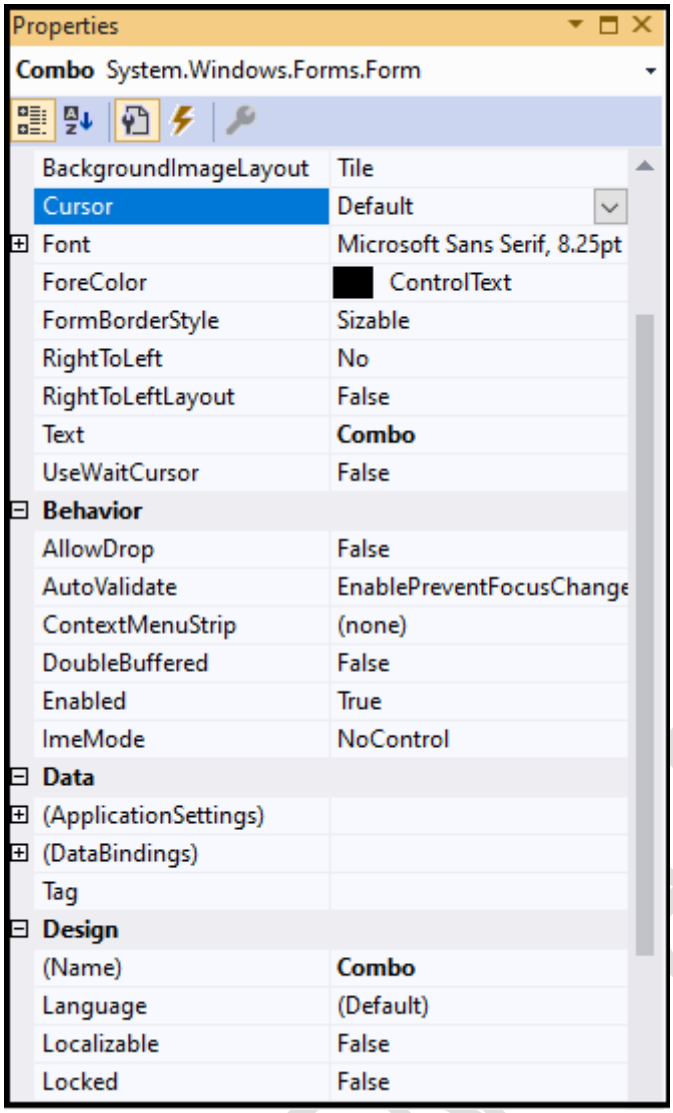


Step 2: Once the ComboBox is added to the form, we can set various properties of the ComboBox by clicking on the ComboBox control.

ComboBox Properties

There are following properties of the ComboBox control.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



Property	Description
AllowSelection	The AllowSelection property takes the value that indicates whether the list allows selecting the list item.
AutoCompleteMode	It takes a value that represents how automatic completion work for the ComboBox.
Created	It takes a value that determines whether the control is created or not.
DataBinding	It is used to bind the data with a ComboBox Control.
BackColor	The BackColor property is used to set the background color of the

	combo box control.
DataSource	It is used to get or set the data source for a ComboBox Control.
FlatStyle	It is used to set the style or appearance for the ComboBox Control.
MaxDropDownItems	The MaxDropDownItems property is used in the combo box control to display the maximum number of items by setting a value.
MaxLength	It is used by the user to enter maximum characters in the editable area of the combo box.
SelectedItem	It is used to set or get the selected item in the ComboBox Control.
Sorted	The Sorted property is used to sort all the items in the ComboBox by setting the value.

ComboBox Events

Events	Description
FontChanged	It occurs when the property of the font value is changed.
Format	When the data is bound with a combo box control, a format event is called.
SelectIndexChanged	It occurs when the property value of SelectIndexChanged is changed.
HelpRequested	When the user requests for help in control, the HelpRequested event is called.
Leave	It occurs when the user leaves the focus on the ComboBox Control.
MarginChanged	It occurs when the property of margin is changed in the ComboBox control.

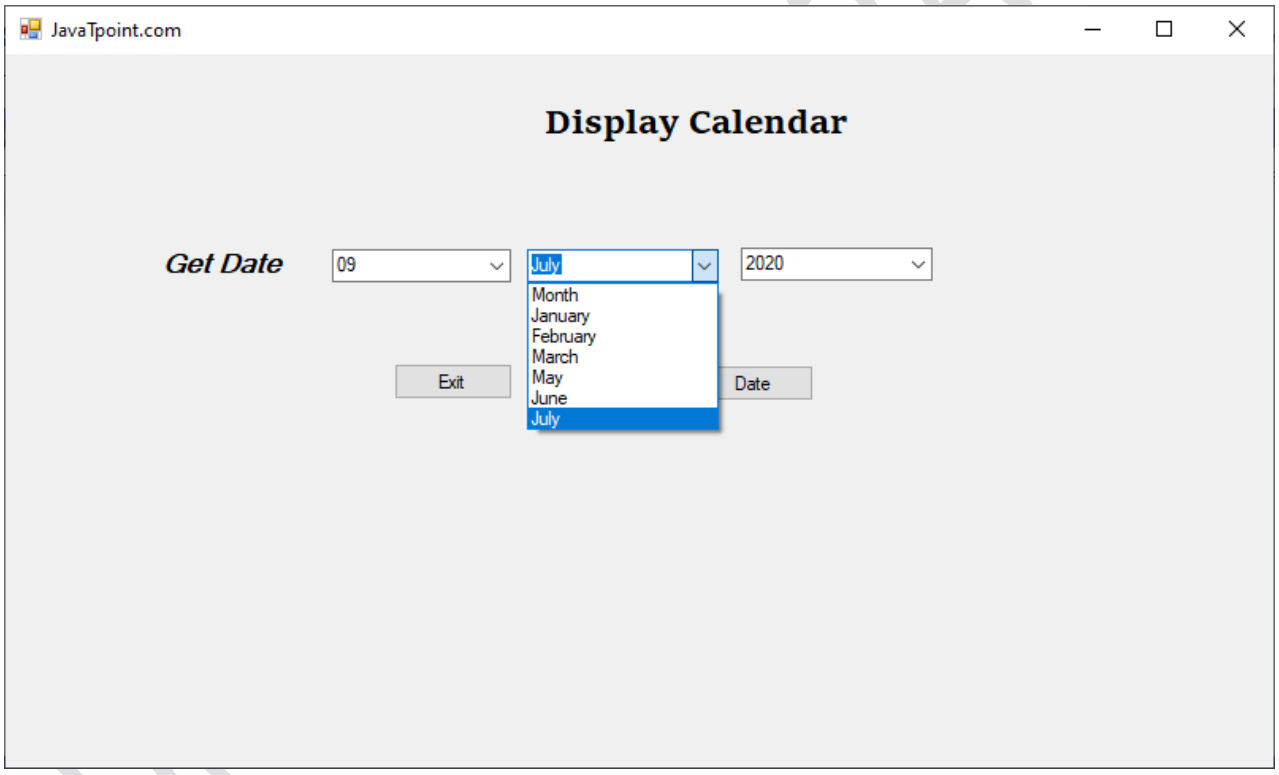
Let's create a program to display the Calendar in the VB.NET Windows Form.

ComboBox_Control.vb

```
1.  Public Class ComboBox_Control
2.      Dim DT As Integer
3.      Dim MM As String
4.      Dim YY As Integer
5.      Private Sub ComboBox_Control_Load(sender As Object, e As EventArgs) Handles MyBase.L
    oad
6.          Me.Text = "saicollege.com"
7.          Label1.Text = "Display Calendar"
8.          Label2.Text = "Get Date"
9.          Button1.Text = "Date"
10.         Button2.Text = "Exit"
11.         ComboBox1.Items.Add("Date")
12.         ComboBox1.Items.Add("01")
13.         ComboBox1.Items.Add("02")
14.         ComboBox1.Items.Add("03")
15.         ComboBox1.Items.Add("04")
16.         ComboBox1.Items.Add("05")
17.         ComboBox1.Items.Add("06")
18.         ComboBox1.Items.Add("07")
19.         ComboBox1.Items.Add("08")
20.         ComboBox1.Items.Add("09")
21.         ComboBox2.Items.Add("Month")
22.         ComboBox2.Items.Add("January")
23.         ComboBox2.Items.Add("February")
24.         ComboBox2.Items.Add("March")
25.         ComboBox2.Items.Add("May")
26.         ComboBox2.Items.Add("June")
27.         ComboBox2.Items.Add("July")
28.         ComboBox3.Items.Add("Year")
29.         ComboBox3.Items.Add("2016")
30.         ComboBox3.Items.Add("2017")
31.         ComboBox3.Items.Add("2018")
32.         ComboBox3.Items.Add("2019")
33.         ComboBox3.Items.Add("2020")
34.     End Sub
35.
36.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

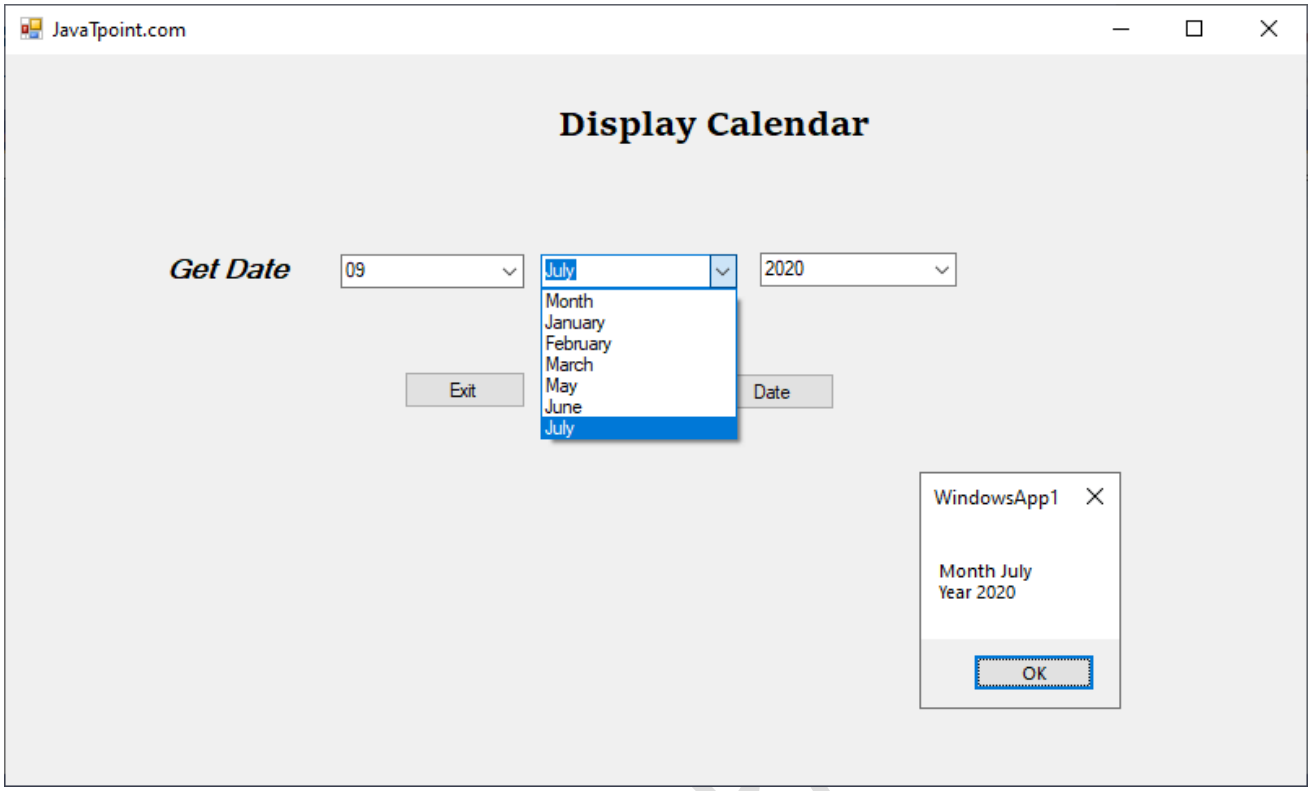
```
37.         DT = ComboBox1.Text
38.         MM = ComboBox2.Text
39.         YY = ComboBox3.Text
40.         MsgBox("Month " & MM + vbCrLf + "Year " & YY)
41.     End Sub
42.
43.     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
44.         End
45.     End Sub
46. End Class
```

Output:



Now select the day, month, and year from dropdown box and then click on the Date button to display the date in the form.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

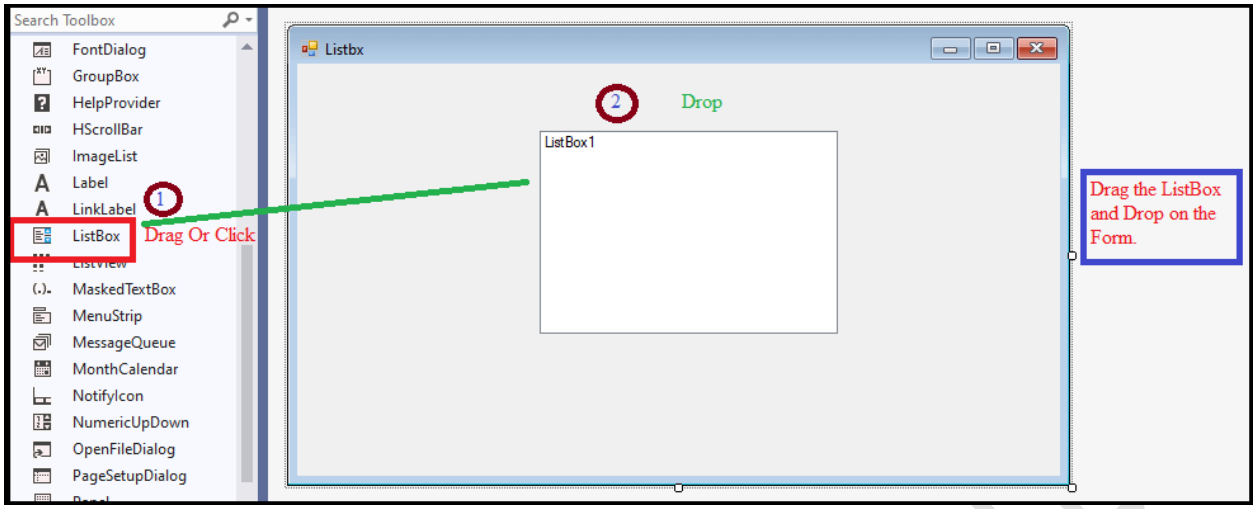


VB.NET ListBox Control

The ListBox control is used to display a list of items in Windows form. It allows the user to select one or more items from the ListBox Control. Furthermore, we can add or design the list box by using the properties and events window at runtime.

Let's create a ListBox control in the VB.NET Windows by using the following steps.

Step 1: Drag the ListBox control from the Toolbox and drop it to the Windows form, as shown below.



Step 2: Once the ListBox is added to the Form, we can set various properties of the Listbox by clicking on the ListBox control.

ListBox Properties

There are following properties of the ListBox control.

Properties Name	Description
AllowSelection	It takes a value that defines whether the list box allows the user to select the item from the list.
CanSelect	It obtains a value that determines whether the Listbox control can be selected.
ColumnWidth	It is used to get or set the width of the columns in a multicolumn Listbox.
Container	As the name defines, a container gets the IContainer that stores the component of ListBox control.
Controls	It is used to get the collection of controls contained within the control.
Created	It takes a value that determines whether the control is created or not.
Width	It is used to set the width of the ListBox control.

Visible	It takes a value that determines whether the ListBox control and all its child are displayed on the Windows Form.
SelectionMode	It is used to get or set the method that determines which items are selected in the ListBox.
MultiColumn	It allows multiple columns of the item to be displayed by setting the True value in the Listbox.

ListBox Methods

Method Name	Description
Add()	The Add() method is used to add items to an item collection.
Remove	It is used to remove an item from an item collection. However, we can remove items using the item name.
Clear	It is used to remove all items from the item collection at the same time.
Contains	It is used to check whether the particular item exists in the ListBox or not.
Show()	It is used to display the control to the user.
Sort()	As the name suggests, a Sort() method is used to arrange or sort the elements in the ListBox.
ResetText()	A ResetText() method is used to reset ListBox's text property and set the default value.
ResetBackColor()	It is used to reset the backColor property of the ListBox and set the default value.
OnNotifyMessage	It is used to notify the message of the ListBox to Windows.
GetSelected	The GetSelected method is used to validate whether the specified item is selected.

Furthermore, we can also refer to VB.NET Microsoft documentation to get a complete list of ListBox properties, and methods.

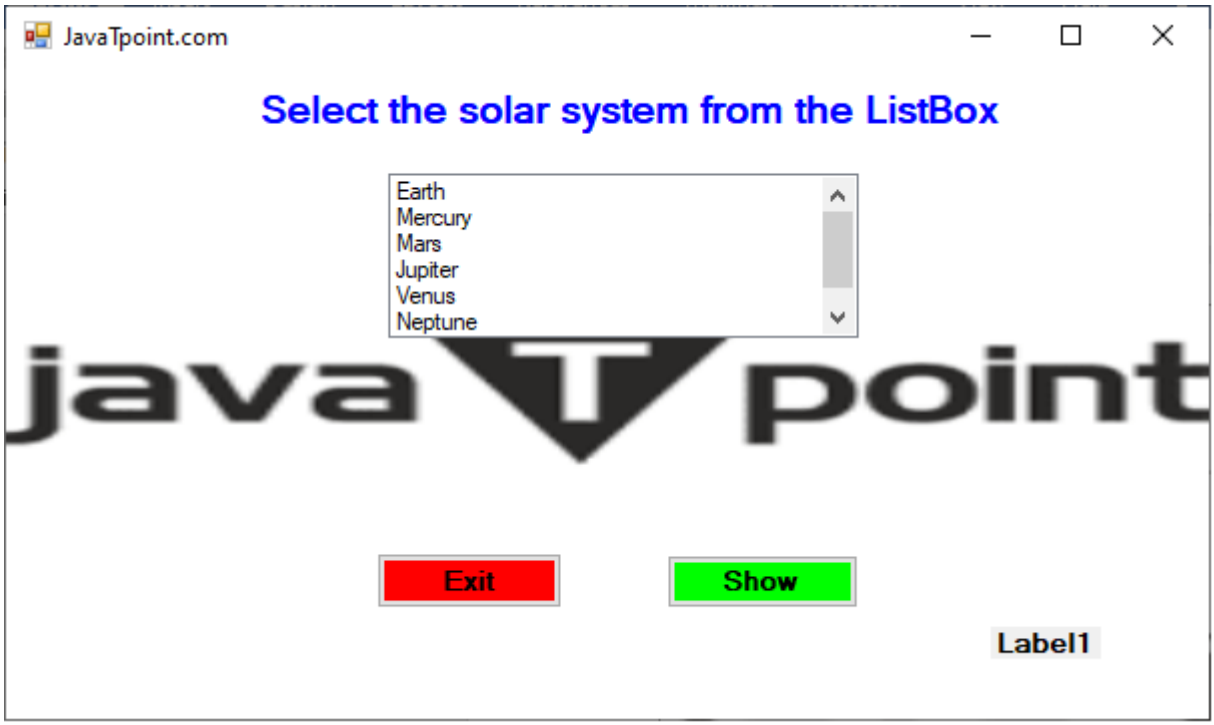
Let's create a program to select an item from the ListBox in the VB.NET form.

Listbx.vb

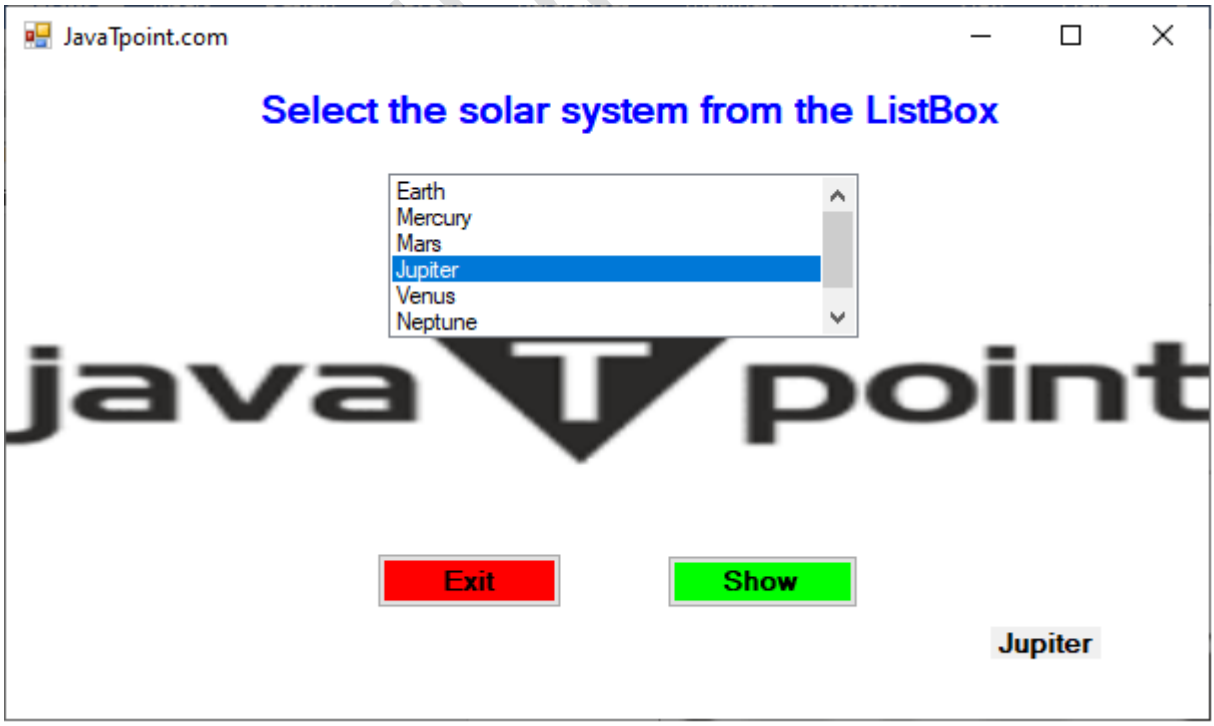
```
1.  Public Class Listbx
2.      Private Sub Listbx_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.          ' set the title of the Form.
4.          Me.Text = "saicollege.com"
5.          ' Add the items into the ListBox
6.          ListBox1.Items.Add("Earth")
7.          ListBox1.Items.Add("Mercury")
8.          ListBox1.Items.Add("Mars")
9.          ListBox1.Items.Add("Jupiter")
10.         ListBox1.Items.Add("Venus")
11.         ListBox1.Items.Add("Neptune")
12.         ListBox1.Items.Add("Uranus")
13.         ' Set the name of the Button1 and Button2
14.         Button1.Text = "Show"
15.         Button2.Text = "Exit"
16.         Label2.Text = "Select the solar system from the ListBox"
17.     End Sub
18.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
19.         Dim lt As String  ' define a local variable.
20.         lt = ListBox1.Text  'accept the data from the ListBox1
21.         MsgBox(" Selected Solar System is " & lt)  ' Display the selected item
22.     End Sub
23.     Private Sub ListBox1_SelectedIndexChanged(sender As Object, e As EventArgs) Handles Lis
    tBox1.SelectedIndexChanged
24.         Label1.Text = ListBox1.SelectedItem.ToString()  'When a user clicks on an item, it display
    s the item name.
25.     End Sub
26.
27.     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
28.         End 'End or exit an application
29.     End Sub
```

30. End Class

Output:

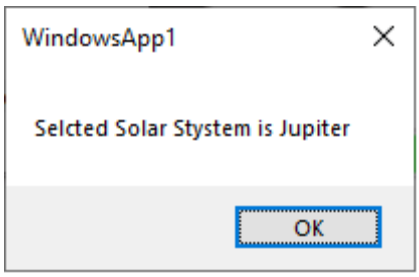


Now select an item from the list. We have selected Jupiter.



Click on the **Show** button to display the selected item in Windows Form, as follows.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

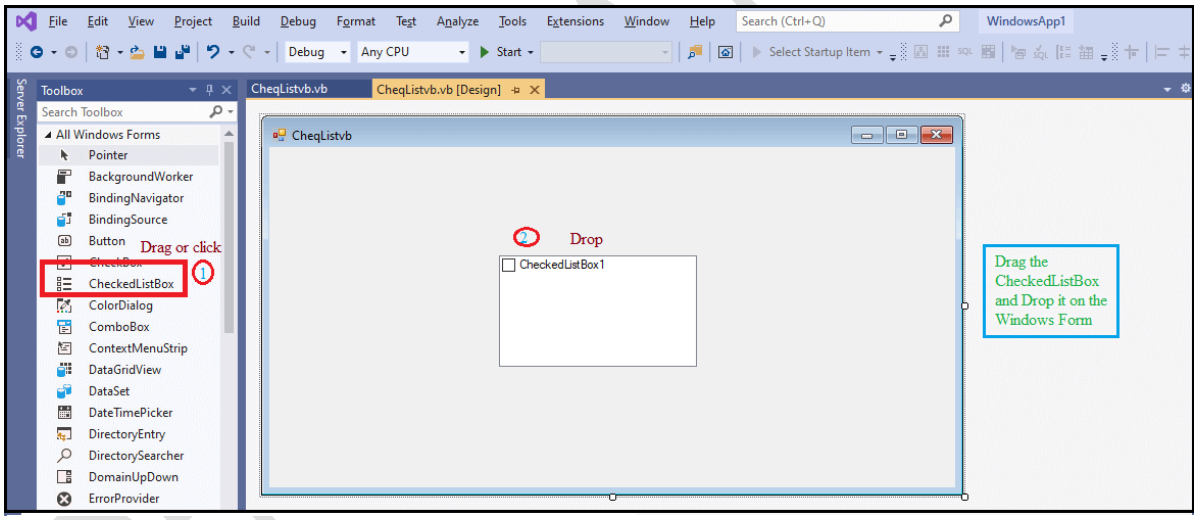


VB.NET CheckedListBox Control

The **CheckedListBox** is similar to **Listbox** except that it displays all items in the list with a checkbox that allows users to check or uncheck single or multiple items.

Let's create a CheckedListBox control in the VB.NET Windows form using the following steps.

Step 1: Drag the **CheckedListBox** control from the Toolbox and drop it to the Windows form, as shown below.



Step 2: Once the CheckedListBox is added to the Form, we can set various properties of the CheckedListbox by clicking on the CheckedListBox control.

CheckedListBox Properties

There are following properties of the CheckedListBox control.

Properties	Description

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

AccessibilityObject	It obtains a value that determines whether the AccessibilityObject is assigned to the CheckedListBox control.
AccessibleName	It gets or sets a value that tells if the accessible client application has used the name of the checkedlistbox control.
AllowSelection	It gets a value that indicates whether the ListBox allows for the item to be selected from the list.
AllowScollOffset	It gets or sets a value representing whether the CheckedListBox control is scrolled in ScrollControlIntoView(Control).
BorderStyle	It is used to set the type of border around the CheckedListBox by getting or setting a value.
CheckedItems	It is used to store a collection of checked items in the CheckedListBox.
ScrollAlwaysVisible	It is used to set or get a value that indicates if the vertical scroll bar appears in Windows Forms at all times.
SelectedItems	It is used to get all selected items from CheckedListBox.
SelectionMode	It is used to get or set a value representing the items' selection mode in the CheckedListBox.
TopIndex	It is used to set the first visible item at the top of the index in the CheckedListBox.

CheckedListBox Methods

Methods	Description
ClearSelected()	It is used to unselect all the selected items in the CheckedListBox.
CreateAccessibilityInstance()	It is used to create new accessibility of the object in the CheckedListBox Control.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

CreateItemCollection()	It is used to create a new instance for the collected item in the list box.
DestroyHandle()	It is used to destroy the handle associated with CheckedListBox.
Equals(Object)	It is used to validate whether the specified object is equal to the current object in the CheckedListBox or not.
FindForm()	It is used to obtain the form in which CheckedListBox has control.
GetItemText(Object)	It is used to get the text of the specified item in the CheckedListBox.
GetType()	It is used to get the current item type in the CheckedListBox.
Show()	A Show() method is used to display the CheckedListBox Control to the user.
Sort()	A Sort() method is used to sort or organize all the items available in CheckedListBox.

Furthermore, we can also refer to VB.NET Microsoft documentation to get a complete list of **CheckedListBox** properties and methods.

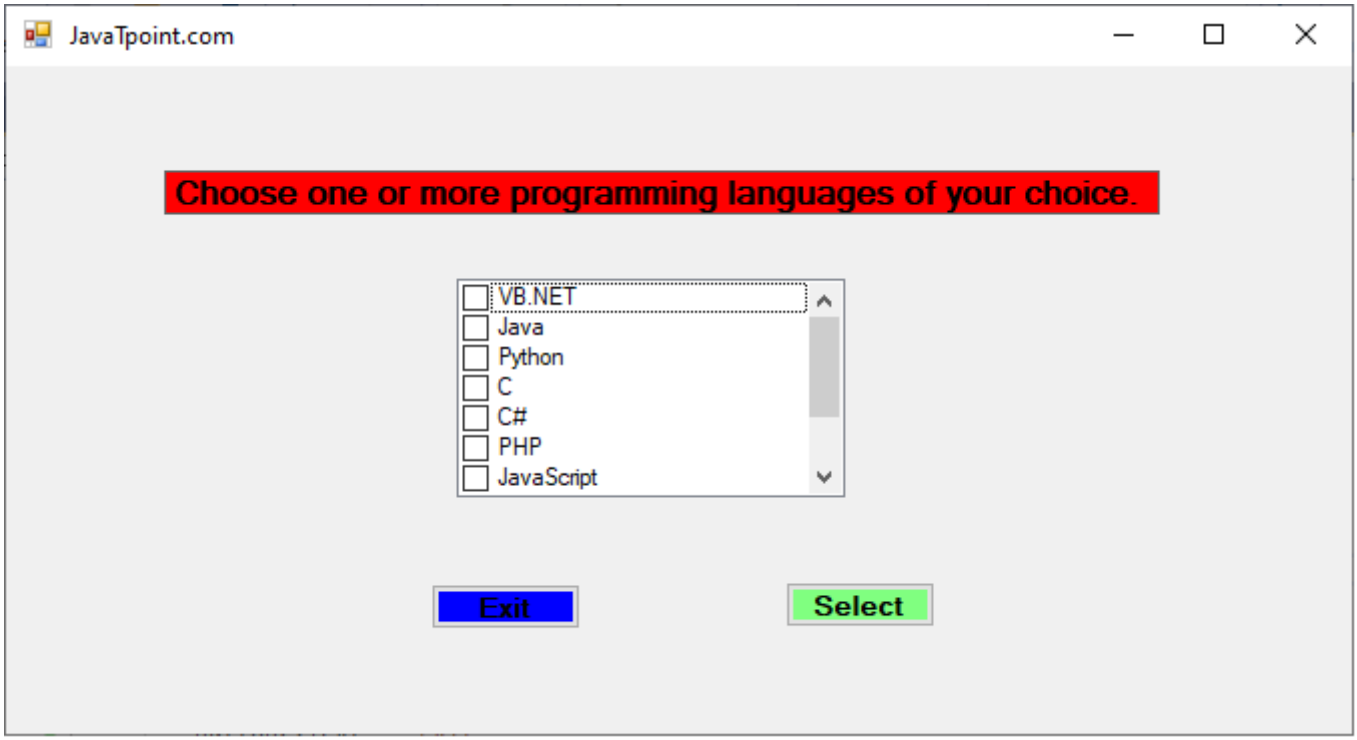
Let's create a program to select or check more than one item from the CheckedListBox in the VB .NET form.

CheqListvb.vb

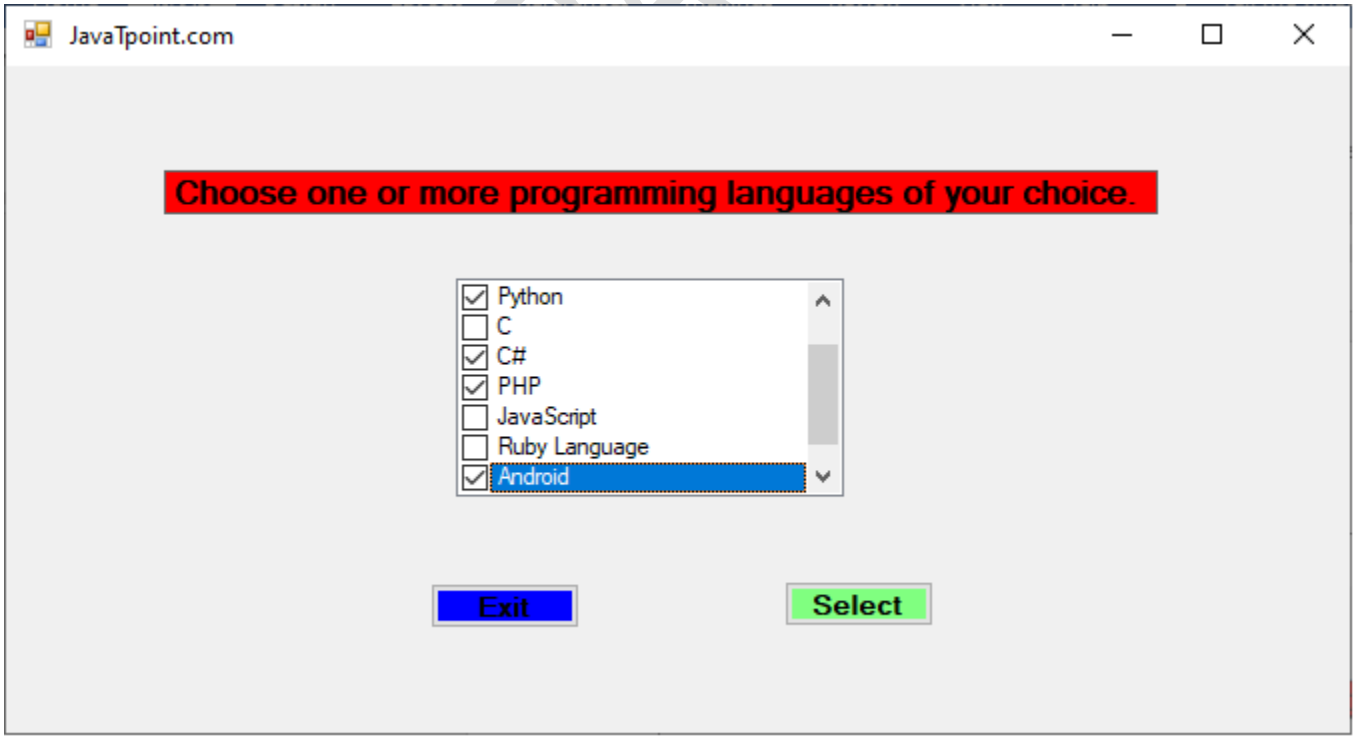
```
1. Public Class CheqListvb
2.     Private Sub CheqListvb_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.         Me.Text = " saicollege.com"
4.         CheckedListBox1.Items.Add("VB.NET")
5.         CheckedListBox1.Items.Add("Java")
6.         CheckedListBox1.Items.Add("Python")
7.         CheckedListBox1.Items.Add("C")
8.         CheckedListBox1.Items.Add("C#")
```

```
9.        CheckedListBox1.Items.Add("PHP")
10.       CheckedListBox1.Items.Add("JavaScript")
11.       CheckedListBox1.Items.Add("Ruby Language")
12.       CheckedListBox1.Items.Add("Android")
13.       CheckedListBox1.Items.Add("Perl")
14.       Label1.Text = "Choose one or more programming languages of your choice. "
15.       Button1.Text = "Select"
16.       Button2.Text = "Exit"
17.    End Sub
18.
19.    ' To submit the checked items, click on Button1 or Select
20.    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
21.       Dim cheq As New System.Text.StringBuilder
22.       For Each item In CheckedListBox1.CheckedItems
23.          cheq.Append(item)
24.          cheq.Append(" ")
25.       Next
26.       MessageBox.Show(" Your Checked Items are : " & cheq.ToString())
27.
28.    End Sub
29.
30.    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
31.       End 'terminate the program
32.    End Sub
```

Output:

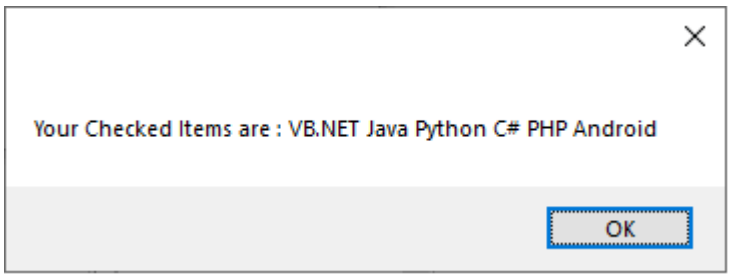


In the above list, we can select more than one item from the CheckedListBox in Windows Form.



After that, click on the **Select** button to display the selected item in Windows Form.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

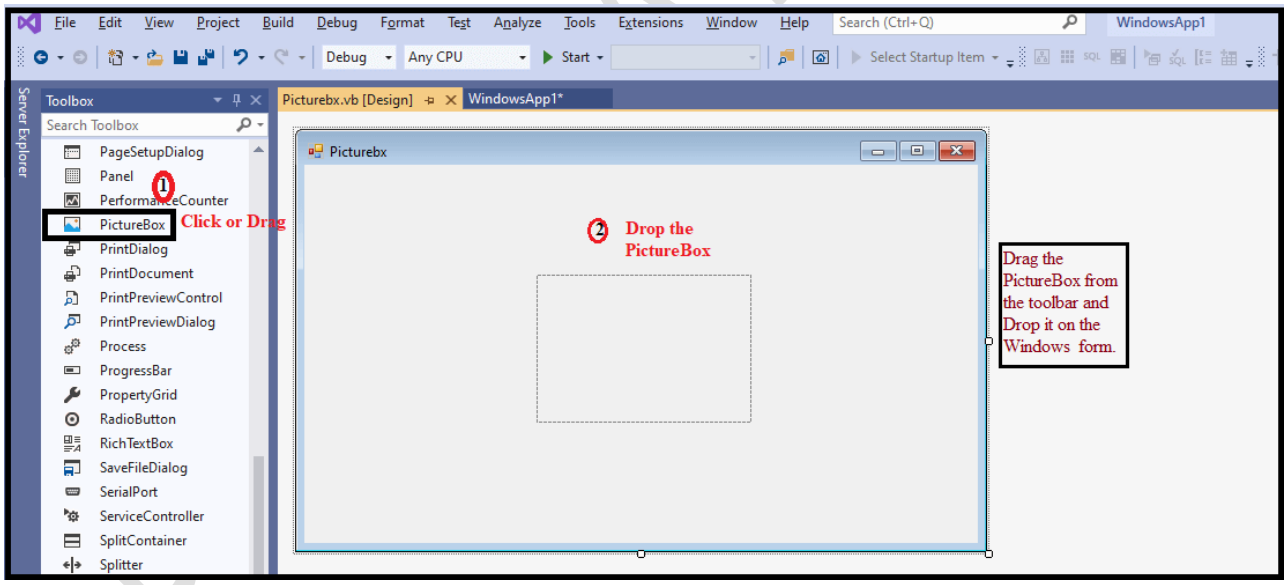


VB.NET PictureBox Control

PictureBox control is used to display the images on Windows Form. The PictureBox control has an image property that allows the user to set the image at runtime or design time.

Let's create a PictureBox control in the VB.NET Windows form using the following steps.

Step 1: We have to find the PictureBox control from the toolbox and then drag and drop the PictureBox control onto the window form, as shown below.



Step 2: Once the PictureBox is added to the form, we can set various properties of the image by clicking on the PictureBox control.

Properties of the PictureBox

There are following properties of the VB.NET PictureBox control.

Property	Description
BackColor	It is used to set the background color for the PictureBox in the window form.
BackgroundImage	It is used to set the background image of a window form by setting or getting value in the picture box.
ErrorImage	The ErrorImage property is used to display an image if an error occurs while loading an image on a window form.
InitialImage	The initial image is used to display an image on the PictureBox when the main image is loaded onto a window form by setting a value in the PictureBox control.
WaitOnLoad	It represents whether the particular image is synchronized or not in the PictureBox control.
Text	It is used to set text for the picture box controls in the window form.
Image	The image property is used to display the image on the PictureBox of a Windows form.
BorderStyle	It is used to set the border style for the picture box in the windows form.
ImageLocation	It is used to set or get the path or URL of the image displayed on the picture box of the window form.
IsMirrored	It obtains a value that determines whether the picture box control is mirrored.

Methods of the PictureBox Control

Method	Description
CancelAysnc()	The CancelAsync method is used to cancel an asynchronous image load in a PictureBox control.

CreateHandle()	It is used to create handles for the picture box controls in window form.
DestroyHandle()	It is used to destroy all the handles that are associated with the picture box control.
GetStyle()	The GetStyle() method is used to get values for the specified bit style in the PictureBox control.
Load()	The Load() method is used to load the specified image from the control using the ImageLocation property.
LoadAsync(String)	It is used to asynchronous load the image at the specified position of the picture box control.

Events of the PictureBox Control

There are some Events of the VB.NET PictureBox control.

Events	Description
BackColorChanged	It occurs when the property of the backcolor is changed in the PictureBox control.
BackgroundImageLayoutChanged	It occurs when the property value of the BackgroundImage is changed in the PictureBox control.
ContextMenuChanged	It occurs when the property of the ContextMenu is changed in the PictureBox control.
Resize	The resize event occurs when the picture box control is changed.

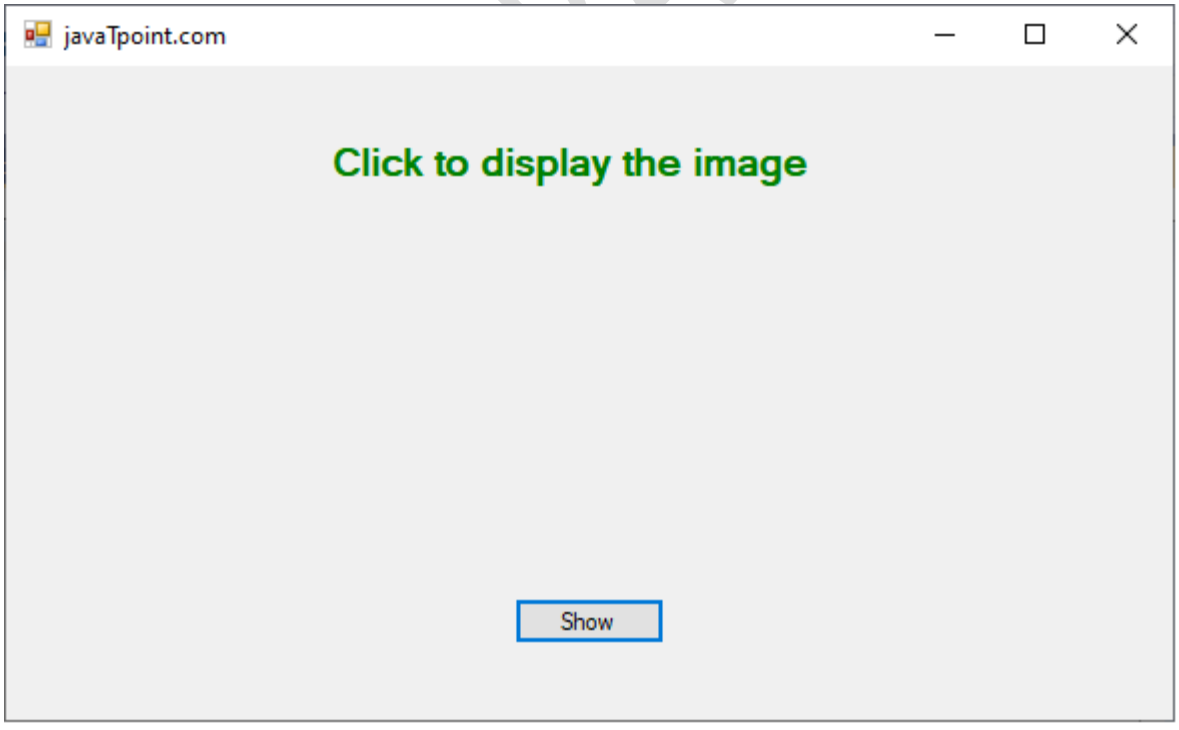
Furthermore, we can also refer to VB.NET Microsoft documentation to get a complete list of **PictureBox** control properties, methods, and events in the VB.NET.

Let's create a program to display an image in the VB.NET form.

Picturebx.vb

```
1.  Public Class Picturebx
2.      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3.          'Dim Str As String = "C:\Users\AMIT YADAV\Desktop\"
4.          PictureBox1.Image = Image.FromFile("C:\Users\AMIT YADAV\Desktop\jtp2.png")
5.          PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
6.          PictureBox1.Height = 250
7.          PictureBox1.Width = 400
8.          Label1.Visible = False
9.      End Sub
10.     Private Sub Picturebx_Load(sender As Object, e As EventArgs) Handles MyBase.Load
11.         Me.Text = "saicollege.com" 'Set the title name for the form
12.         Button1.Text = "Show"
13.         Label1.Text = "Click to display the image"
14.         Label1.ForeColor = ForeColor.Green
15.     End Sub
16. End Class
```

Output:



Now click on the **Show** button to display an image in the windows form.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

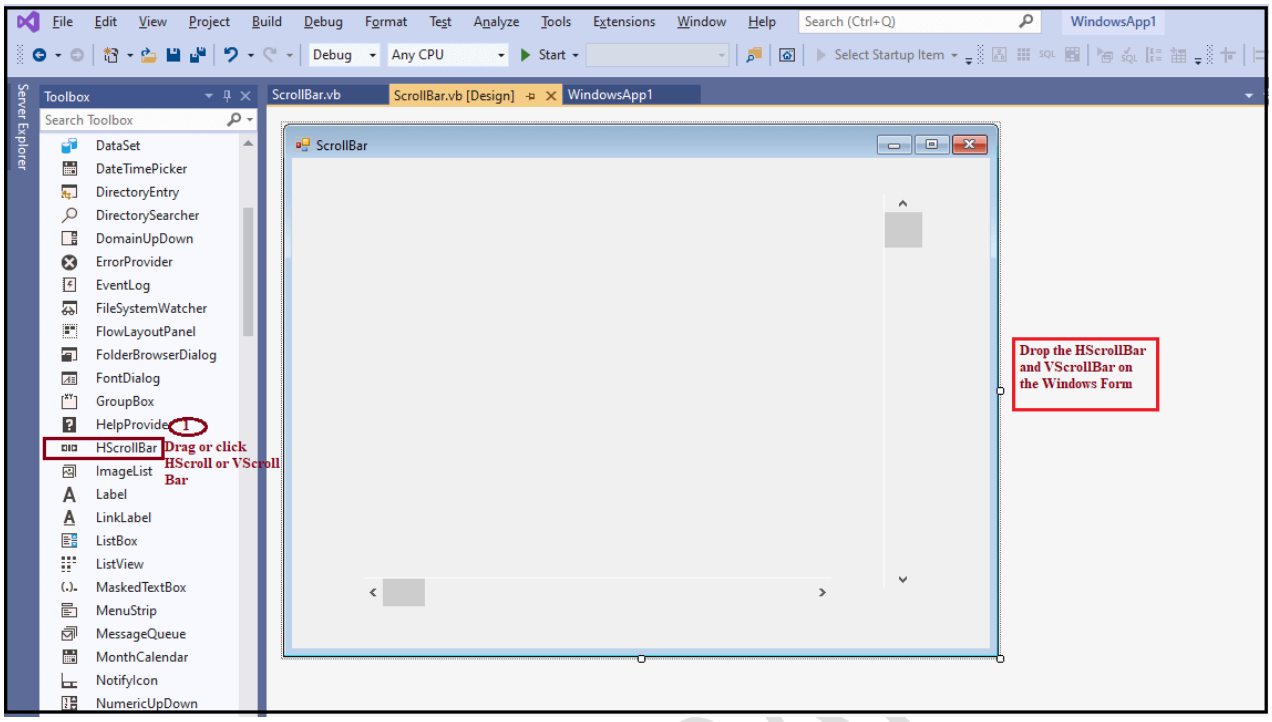
VB.NET ScrollBars Control

A ScrollBar control is used to create and display vertical and horizontal scroll bars on the Windows form. It is used when we have large information in a form, and we are unable to see all the data. Therefore, we used VB.NET ScrollBar control. Generally, ScrollBar is of two types: HScrollBar for displaying scroll bars and VScrollBar for displaying Vertical Scroll bars.

Let's create a ScrollBar control in the VB.NET Windows form using the following steps.

Step 1: The first step is to drag the HScrollBar and VScrollBar control from the toolbox and drop it on to the form.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



Step 2: Once the ScrollBar is added to the form, we can set various properties of the ScrollBar by clicking on the HScrollBar and VScrollBar control.

Properties of the ScrollBar Control

There are following properties of the VB.NET ScrollBar control.

Property	Description
BackColor	The BackColor property is used to set the back color of the scroll bar.
Maximum	It is used to set or get the maximum value of the Scroll Bar control. By default, it is 100.
Minimum	It is used to get or set the minimum value of the Scroll bar control. By default, it is 0.
SmallChange	It is used to obtain or set a value that will be added or subtracted from the property of the scroll bar control when the scroll bar is moved a short distance.
AutoSize	As the name suggests, the AutoSize property is used to get or set a value representing whether the scroll bar can be resized automatically or not

	with its contents.
LargeChange	It is used to obtain or set a value that will be added or subtracted from the property of the scroll bar control when the scroll bar is moved a large distance.
Value	It is used to obtain or set a value in a scroll bar control that indicates a scroll box's current position.
DefaultImeMode	It is used to get the default input method Editor (IME) that are supported by ScrollBar controls in the Windows Form.

Methods of the ScrollBar Control

Method	Description
UpdateScrollInfo	It is used to update the ScrollBar control using the Minimum, maximum, and the value of LargeChange properties.
OnScroll(ScrollEventArgs)	It is used to raise the Scroll event in the ScrollBar Control.
OnEnabledChanged	It is used to raise the EnabledChanged event in the ScrollBar control.
Select	It is used to activate or start the ScrollBar control.
OnValueChanged(EventArgs)	It is used to raise the ValueChanged event in the ScrollBar control.

Events of the ScrollBar Control

Event	Description
AutoSizeChanged	The AutoSizeChanged event is found in the ScrollBar control when the value of the AutoSize property changes.
Scroll	The Scroll event is found when the Scroll control is moved.
TextChangedEvent	It occurs in the ScrollBar control when the value of the text property

	changes.
ValueChanged	A ValueChanged event occurs when the property of the value is changed programmatically or by a scroll event in the Scrollbar Control.

Furthermore, we can also refer to VB.NET Microsoft documentation to get a complete list of **ScrollBar** control properties, methods, and events in the VB.NET.

Let's create a simple program to understand the use of **ScrollBar** Control in the VB.NET Windows Forms.

ScrollBar.vb

```
1. Public Class ScrollBar
2.     Private Sub ScrollBar_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.         Me.Text = "saicollege.com"  'Set the title for a Windows Form
4.         Label1.Text = "Use of ScrollBar in Windows Form"
5.         Label1.ForeColor = Color.Blue
6.         Me.AutoScroll = True
7.         Me.VScrollBar1.Minimum = 0
8.         Me.VScrollBar1.Maximum = 100
9.         Me.VScrollBar1.Value = 0
10.        Me.VScrollBar1.BackColor = Color.Blue
11.        Me.HScrollBar1.Minimum = 0
12.        Me.HScrollBar1.Maximum = 100
13.        Me.HScrollBar1.Value = 35
14.    End Sub
15. End Class
```

Output:

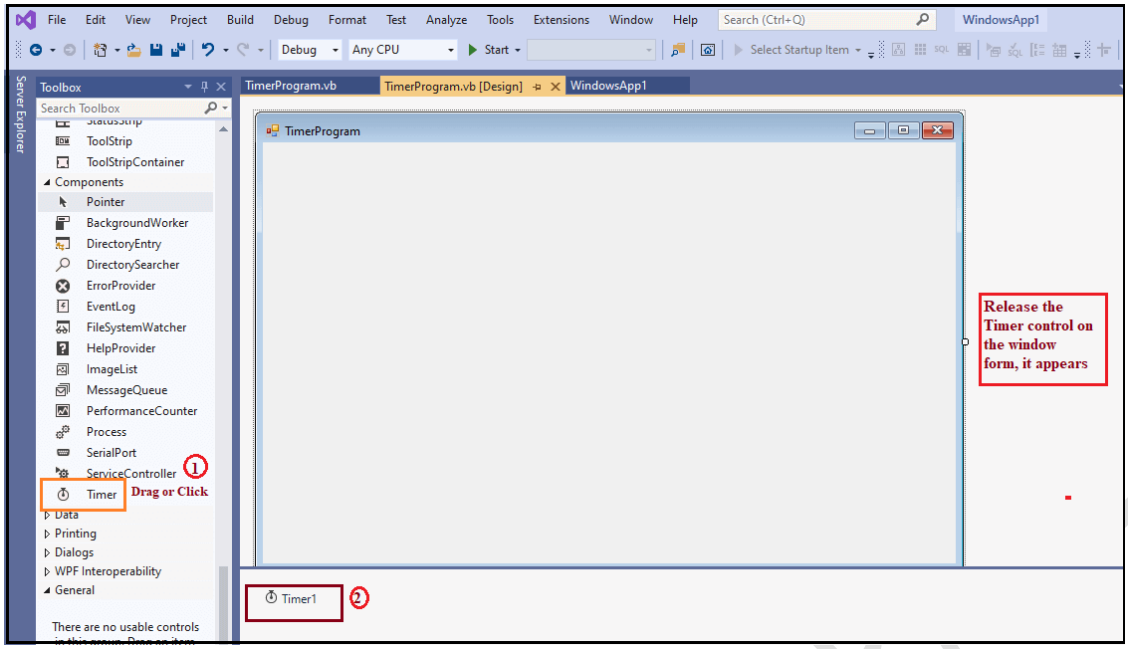
VB.NET Timer Control

The timer control is a looping control used to repeat any task in a given time interval. It is an important control used in Client-side and Server-side programming, also in Windows Services.

Furthermore, if we want to execute an application after a specific amount of time, we can use the **Timer Control**. Once the timer is enabled, it generates a tick event handler to perform any defined task in its time interval property. It starts when the start() method of timer control is called, and it repeats the defined task continuously until the timer stops.

Let's create a Timer control in the VB.NET Windows form by using the following steps.

Step 1: Drag and drop the Timer control onto the window form, as shown below.



Step 2: Once the Timer is added to the form, we can set various properties of the Timer by clicking on the Timer control.

Timer Control Properties

There are following properties of the VB.NET Timer control.

Properties	Description
Name	The Name property is used to set the name of the control.
Enabled	The Enables property is used to enable or disable the timer control. By default, it is True.
Interval	An Interval property is used to set or obtain the iteration interval in milliseconds to raise the timer control's elapsed event. According to the interval, a timer repeats the task.
AutoReset	The AutoReset property is used to obtain or set a Boolean value that determines whether the timer raises the elapsed event only once.
Events	Events property are used to get the list of event handler that is associated with Event Component.

CanRaiseEvents	It is used to get a value that represents whether the component can raise an event.

Events of Timer Control

Events	Description
Disposed	When control or component is terminated by calling the Dispose method, a Dispose event occurs.
Elapsed	When the interval elapses in timer control, the Elapsed event has occurred.
Tick	A tick event is used to repeat the task according to the time set in the Interval property. It is the default event of a timer control that repeats the task between the Start() and Stop() methods.

Methods of Timer Control

Methods	Description
BeginInt()	The BeginInt() method is used to start run time initialization of a timer control used on a form or by another component.
Dispose()	The Dispose() method is used to free all resources used by the Timer Control or component.
Dispose(Boolean)	It is used to release all resources used by the current Timer control.
Close()	The Close() method is used to release the resource used by the Timer Control.
Start()	The Start() method is used to begin the Timer control's elapsed event by setting the Enabled property to true.
EndInt()	The EndInt() method is used to end the run time initialization of timer control that is used on a form or by another component.
Stop()	The Stop() method is used to stop the timer control's elapsed event by setting Enabled property to false.

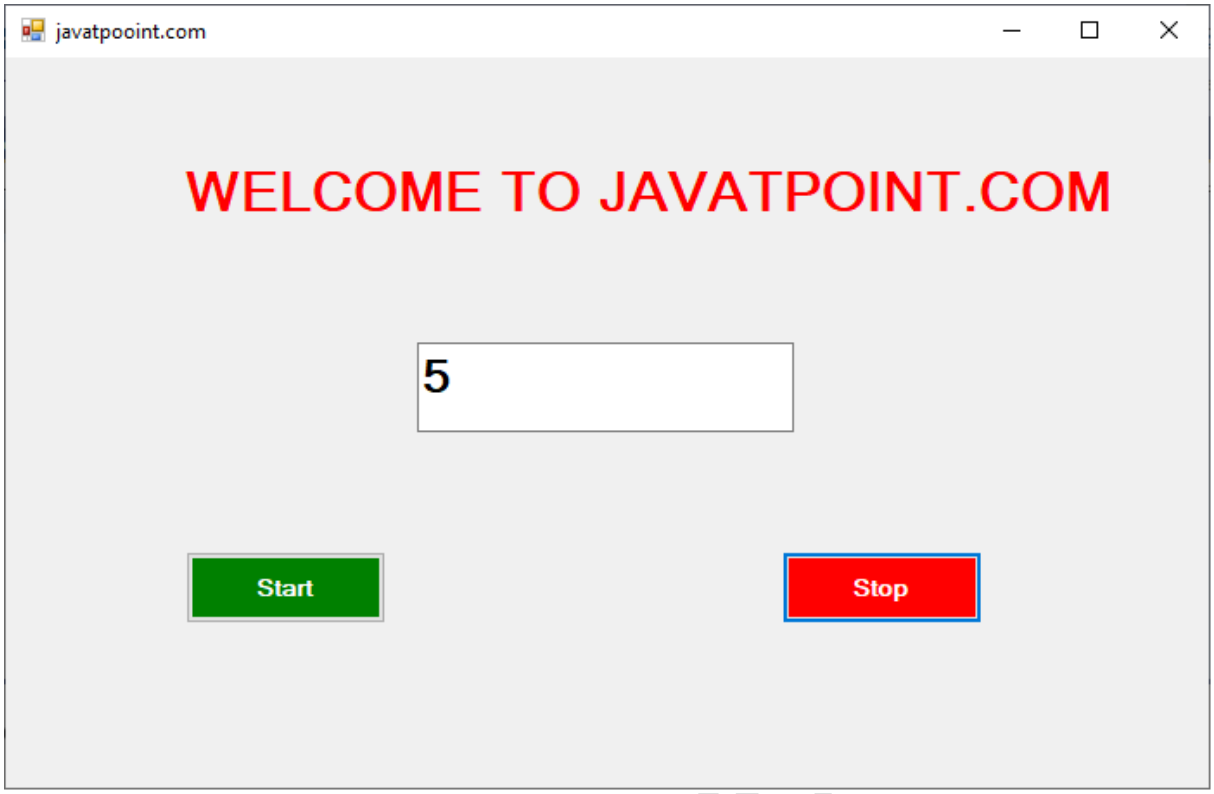
PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Let's create a simple program to understand the use of Timer Control in the VB.NET Windows Forms.

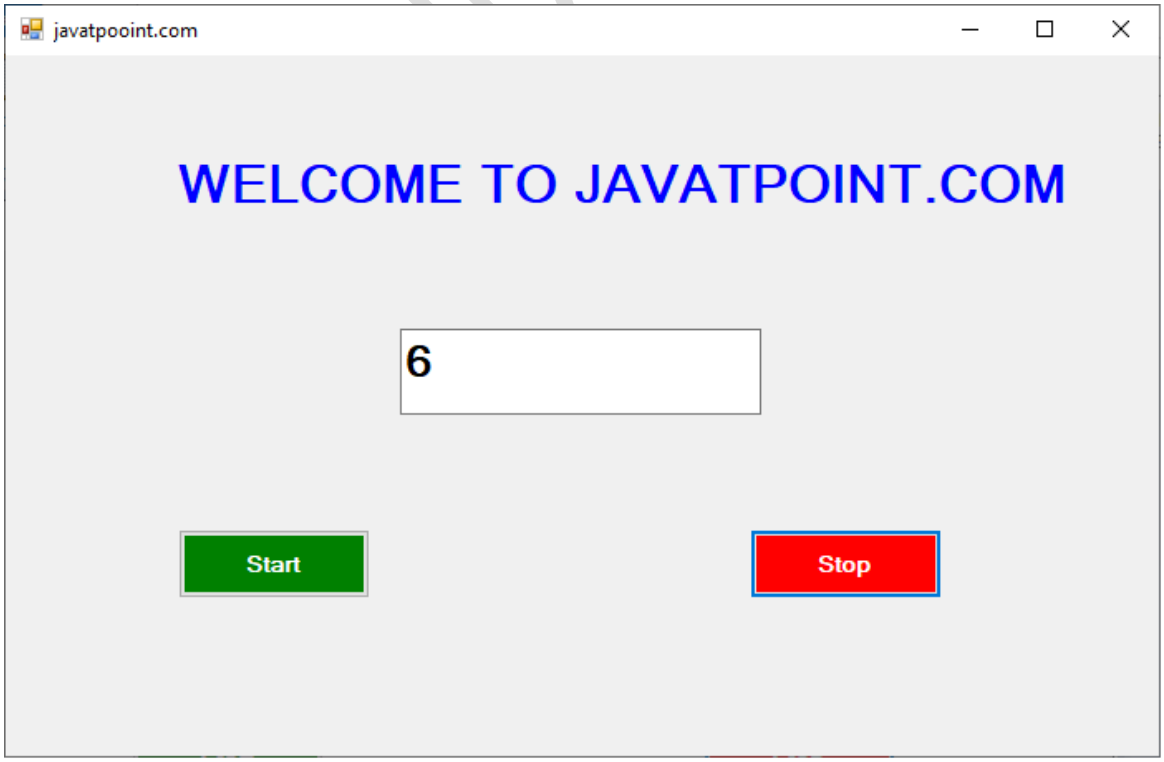
TimerProgram.vb

```
1.  Public Class TimerProgram
2.      Private Sub TimerProgram_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.          Me.Text = "saicollege.com" 'Set the title for a Windows Form
4.          Label1.Text = "WELCOME TO saicollege.COM"
5.          TextBox1.Text = 1
6.          Timer1.Enabled = True
7.          Button1.Text = "Start"
8.          Button1.BackColor = Color.Green
9.          Button1.ForeColor = Color.White
10.         Button2.Text = "Stop"
11.         Button2.BackColor = Color.Red
12.         Button2.ForeColor = Color.White
13.         Timer1.Start()
14.         Timer1.Interval = 600 'set the time interval
15.     End Sub
16.     Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
17.         If Label1.ForeColor = Color.Red Then
18.             Label1.ForeColor = Color.Blue
19.         ElseIf Label1.ForeColor = Color.Blue Then
20.             Label1.ForeColor = Color.Red
21.         End If
22.         TextBox1.Text = TextBox1.Text + 1 'Incremenet the TextBox1 by 1
23.     End Sub
24.     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
25.         Timer1.Stop() ' Stop the timer
26.     End Sub
27.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
28.         Timer1.Start() 'Start the timer
29.     End Sub
30. End Class
```

Output:



When the program executes, it starts blinking the **WELCOME TO JAVATPOINT.COM** statement and counting the number to 1, as shown above. When the number is odd, the color of the statement is **Red**, and when the number is **even**, the color of the statement is **Blue**, as shown below.



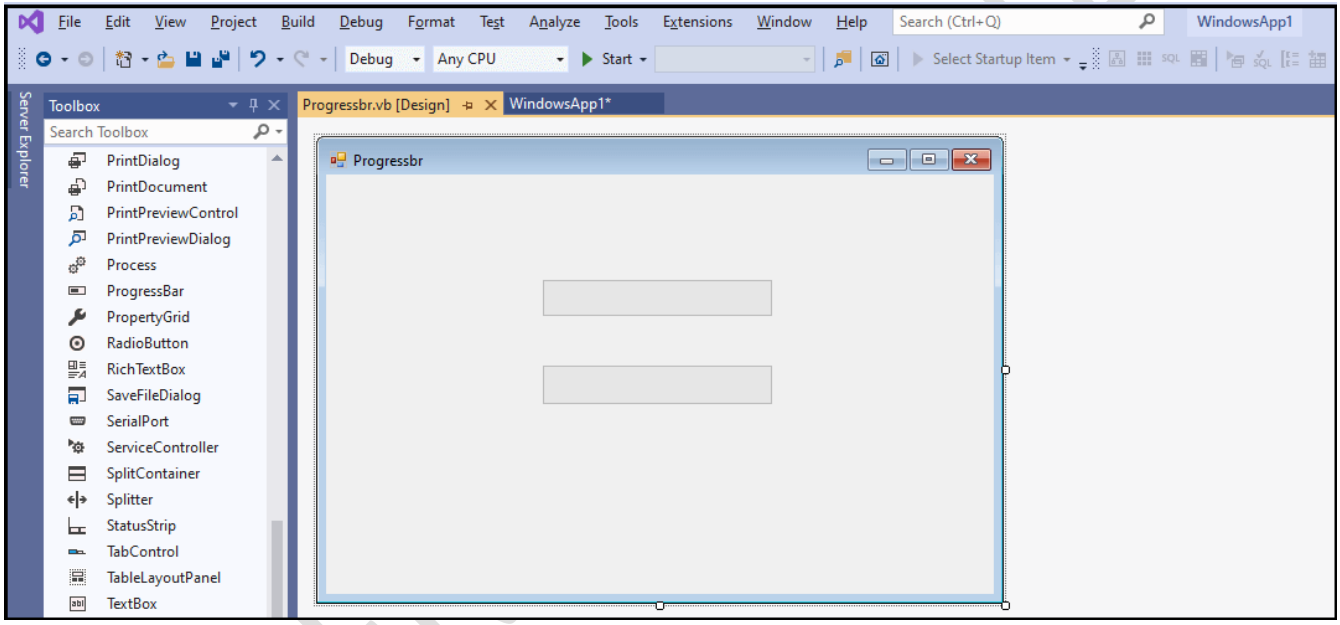
PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

VB.NET ProgressBar Control

The Window ProgressBar control is used by the user to acknowledge the progress status of some defined tasks, such as downloading a large file from the web, copying files, installing software, calculating complex results, and more.

Let's create a ProgressBar by dragging a ProgressBar control from the toolbox and dropping it to the Windows form.

Step 1: The first step is to drag the ProgressBar control from the toolbox and drop it on to the Form.



Step 2: Once the ProgressBar is added to the Form, we can set various properties of the ProgressBar by clicking on the ProgressBar control.

Properties of the ProgressBar Control

There are following properties of the VB.NET ProgressBar control.

Property	Description
BackgroundImage	The BackgroundImage property is used to set the background image in the progressbar control.
MarqueeAnimationSpeed	It is used to determine the progress status for a progress bar in milliseconds.

Padding	The padding property is used to create a space between the edges of the progress bar by setting the value in the progressbar control.
Step	It is used to get or set a value in control that calls the PerformStep method to increase the current state of the progress bar by adding a defined step.
Maximum	It is used to set the maximum length of the progress bar control in the windows form.
Minimum	It is used to set or get the minimum value of the progress bar control in the windows form.
AllowDrop	It obtains a value representing whether the progress bar control enables the user to be dragged onto the form.
Style	It is used to set a value that represents how types the progress bar should be displayed on the Windows Form.

Methods of the ProgressBar Control

Event	Description
ForeColor	The ForeColor method is used to reset the forecolor to its default value.
ToString	The ToString method is used to display the progress bar control by returning the string.
Increment	It is used to increase the current state of the progress bar control by defining the specified time.
PerformStep	The PerformStep method is used to increase the progress bar by setting the step specified in the ProgressBar property.

Events of the ProgressBar Control

Events	Description
Leave	The Leave event occurs when the focus leaves the progress bar control.
MouseClick	A MouseClick event occurred when the user clicked on the progress bar control by the mouse.
BackgroundImageChanged	When the background property changes to the progress bar control, the BackgroundImageChanged event changes.
TextChanged	It occurs when the property of the text is changed in the progress bar control.
PaddingChanged	It occurs when the padding property is changed in the progress bar control.

Furthermore, we can also refer to VB.NET Microsoft documentation to get a complete list of **ProgressBar** control properties, methods, and events in the VB.NET.

Let's create a program to display the progress status in the VB.NET Windows form.

Progressbr.vb

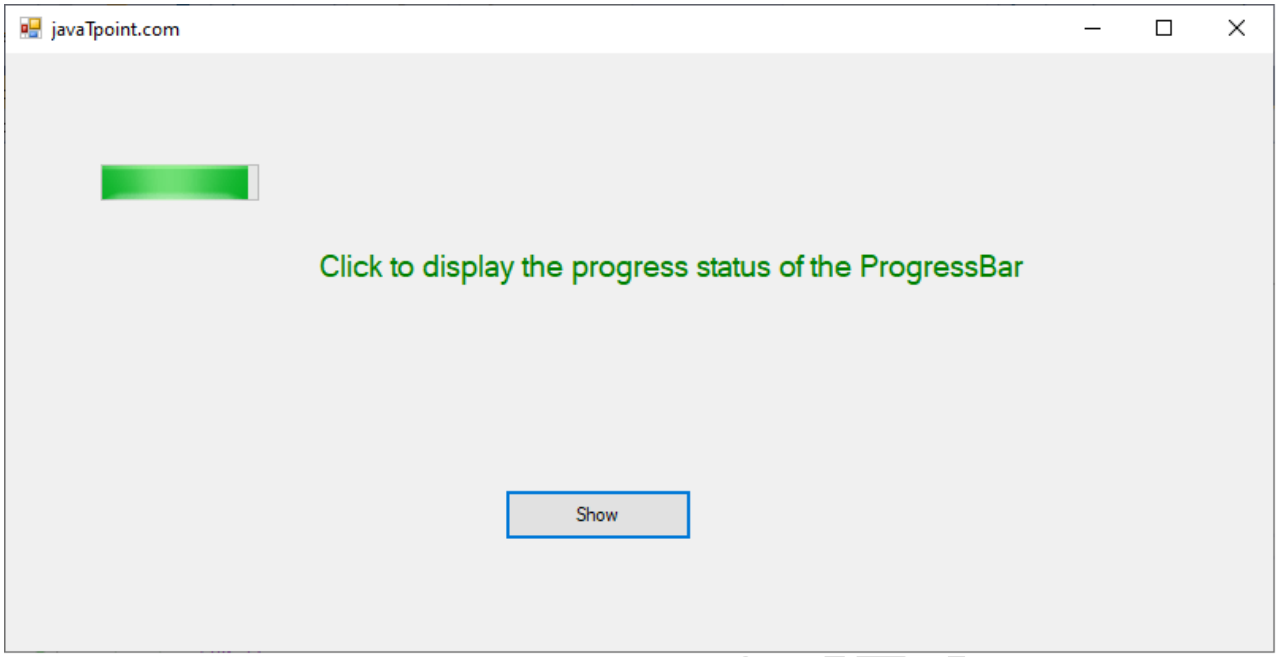
```
1.  Public Class Progressbr
2.      Private Sub Progressbr_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3.          Me.Text = "saicollege.com" 'Set the title name for the form
4.          Button1.Text = "Show"
5.          Label1.Text = "Click to display the progress status of the ProgressBar"
6.          Label1.ForeColor = ForeColor.Green
7.          ProgressBar1.Visible = False
8.
9.          'create a progress bars
10.         Dim Progressbr2 As ProgressBar = New ProgressBar()
11.
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

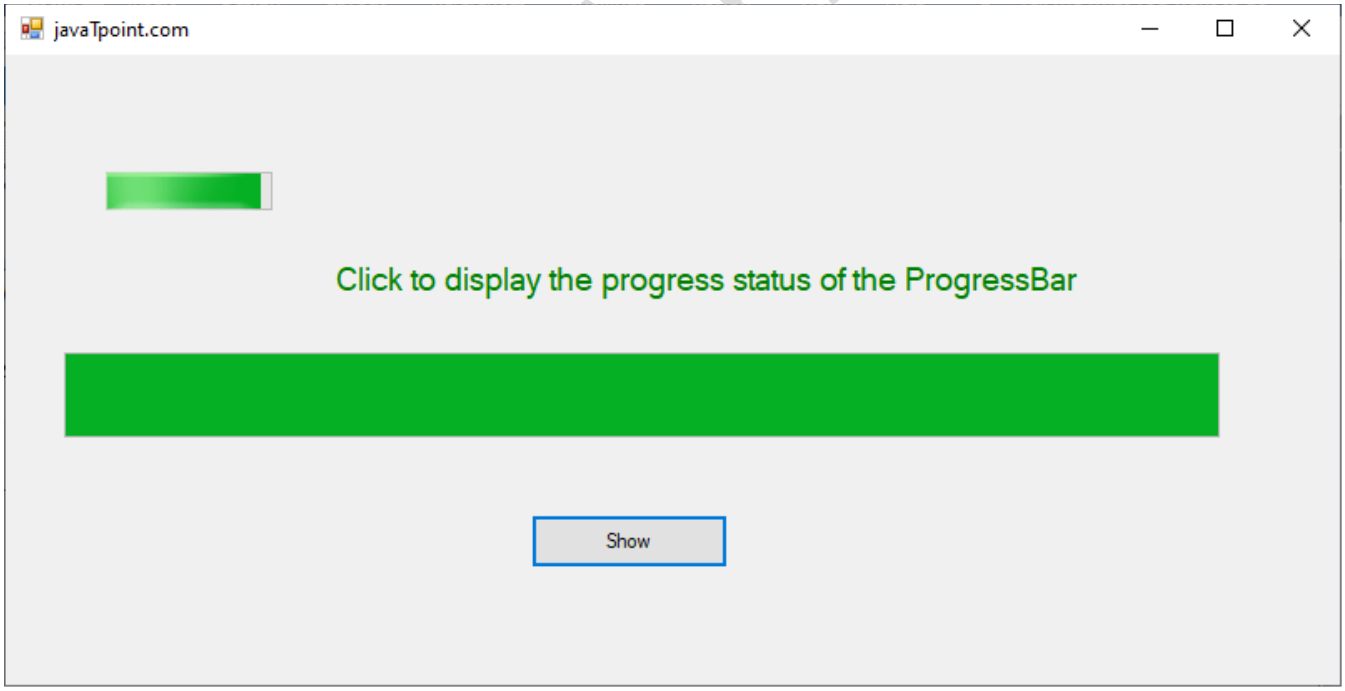
```
12.         'set the progressbar position
13.         Progressbr2.Location = New Point(60, 70)
14.         'set the values for Progressbr2
15.         Progressbr2.Minimum = 0
16.         Progressbr2.Maximum = 500
17.         Progressbr2.Value = 470
18.
19.         'add the progress bar status to the form.
20.         Me.Controls.Add(Progressbr2)
21.
22.     End Sub
23.
24.     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        ProgressBar1.Visible = True

        Dim i As Integer
        ProgressBar1.Minimum = 0
        ProgressBar1.Maximum = 300

        For i = 0 To 300 Step 1
            ProgressBar1.Value = i
            If i > ProgressBar1.Maximum Then
                i = ProgressBar1.Maximum
            End If
        Next
        MsgBox("Successfully Completed")
25.     End Sub
26. End Class
```

Output:

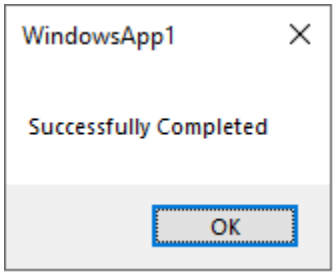
PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



Click on the **Show** button to display the progress status at run time in Windows Form.



And when the progress status reaches the progress bar's maximum value, it displays the following message.



Message Box

The **MsgBox function** displays a message box and waits for the user to click a button and then an action is performed based on the button clicked by the user.

Syntax

```
MsgBox(prompt[,buttons][,title][,helpfile,context])
```

Parameter Description

- **Prompt** − A Required Parameter. A String that is displayed as a message in the dialog box. The maximum length of prompt is approximately 1024 characters. If the message extends to more than a line, then the lines can be separated using a carriage return character (Chr(13)) or a linefeed character (Chr(10)) between each line.
- **Buttons** − An Optional Parameter. A Numeric expression that specifies the type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If left blank, the default value for buttons is 0.
- **Title** − An Optional Parameter. A String expression displayed in the title bar of the dialog box. If the title is left blank, the application name is placed in the title bar.
- **Helpfile** − An Optional Parameter. A String expression that identifies the Help file to use for providing context-sensitive help for the dialog box.
- **Context** − An Optional Parameter. A Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If context is provided, helpfile must also be provided.

The **Buttons** parameter can take any of the following values −

- 0 vbOKOnly - Displays OK button only.
- 1 vbOKCancel - Displays OK and Cancel buttons.
- 2 vbAbortRetryIgnore - Displays Abort, Retry, and Ignore buttons.
- 3 vbYesNoCancel - Displays Yes, No, and Cancel buttons.
- 4 vbYesNo - Displays Yes and No buttons.

- 5 vbRetryCancel - Displays Retry and Cancel buttons.
- 16 vbCritical - Displays Critical Message icon.
- 32 vbQuestion - Displays Warning Query icon.
- 48 vbExclamation - Displays Warning Message icon.
- 64 vbInformation - Displays Information Message icon.
- 0 vbDefaultButton1 - First button is default.
- 256 vbDefaultButton2 - Second button is default.
- 512 vbDefaultButton3 - Third button is default.
- 768 vbDefaultButton4 - Fourth button is default.
- 0 vbApplicationModal Application modal - The current application will not work until the user responds to the message box.
- 4096 vbSystemModal System modal - All applications will not work until the user responds to the message box.

The above values are logically divided into four groups: The **first group** (0 to 5) indicates the buttons to be displayed in the message box. The **second group** (16, 32, 48, 64) describes the style of the icon to be displayed, the **third group** (0, 256, 512, 768) indicates which button must be the default, and the **fourth group** (0, 4096) determines the modality of the message box.

Return Values

The MsgBox function can return one of the following values which can be used to identify the button the user has clicked in the message box.

- 1 - vbOK - OK was clicked
- 2 - vbCancel - Cancel was clicked
- 3 - vbAbort - Abort was clicked
- 4 - vbRetry - Retry was clicked
- 5 - vbIgnore - Ignore was clicked
- 6 - vbYes - Yes was clicked
- 7 - vbNo - No was clicked

Example

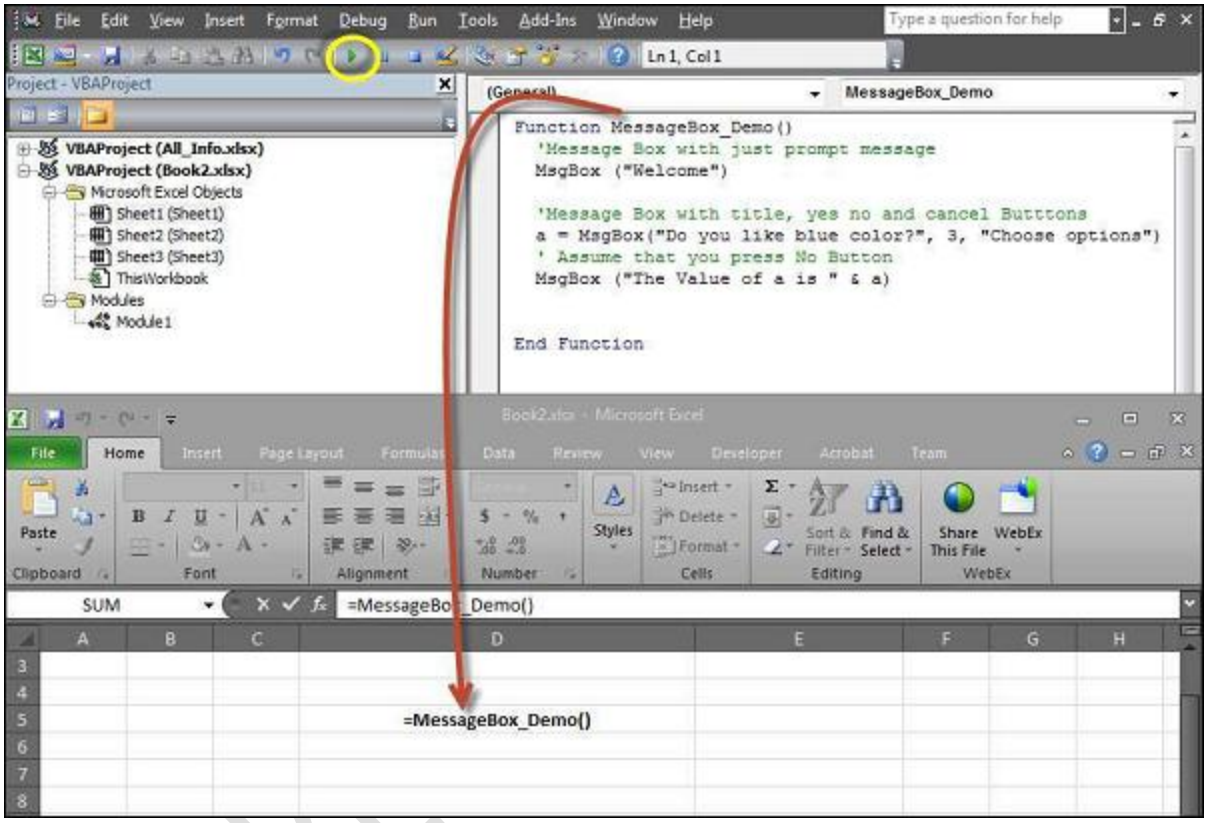
```
FunctionMessageBox_Demo()
'Message Box with just prompt message
   MsgBox("Welcome")

   'MessageBoxwith title, yes noand cancel Butttons
int a =MsgBox("Do you like blue color?",3,"Choose options")
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
' Assume that you press No Button
   msgbox ("The Value of a is " & a)
End Function
```

Output

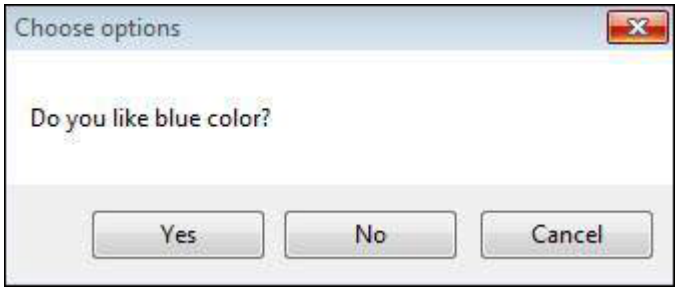
Step 1 − The above Function can be executed either by clicking the "Run" button on VBA Window or by calling the function from Excel Worksheet as shown in the following screenshot.



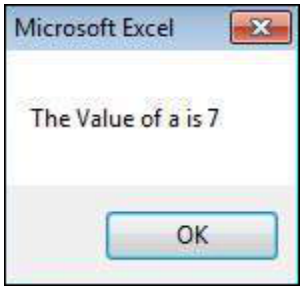
Step 2 − A Simple Message box is displayed with a message "Welcome" and an "OK" Button

Step 3 − After Clicking OK, yet another dialog box is displayed with a message along with "yes, no, and cancel" buttons.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



Step 4 − After clicking the 'No' button, the value of that button (7) is stored as an integer and displayed as a message box to the user as shown in the following screenshot. Using this value, it can be understood which button the user has clicked.



-
- +1

Syntax

[*object*].**Show** *modal*

The **Show** method syntax has these parts:

SYNTAX

Part	Description
object	Optional. An object expression that evaluates to an object in the **Applies To** list. If *object* is omitted, the **UserForm** associated with the active **UserForm** module is assumed to be *object.*
modal	Optional. Variant value that determines if the **UserForm** is modal or modeless.

Settings

The settings for *modal* are:

SETTINGS

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Constant	Value	Description
vbModal	1	**UserForm** is modal. Default.
vbModeless	0	**UserForm** is modeless.

Remarks

If the specified object isn't loaded when the **Show** method is invoked, Visual Basic automatically loads it.

Note

In Microsoft Office 97, if a **UserForm** is set to display as modeless, it causes a run-time error; Office 97 **UserForms** are always modal.

When a **UserForm** is modeless, subsequent code is executed as it is encountered. Modeless forms do not appear in the task bar and are not in the window tab order.

Note

You may lose data associated with a modeless **UserForm** if you make a change to the **UserForm** project that causes it to recompile, for example, removing a code module.

When a **UserForm** is modal, the user must respond before using any other part of the application. No subsequent code is executed until the **UserForm** is hidden or unloaded. Although other forms in the application are disabled when a **UserForm** is displayed, other applications are not.

Example

The following example assumes two **UserForms** in a program. In UserForm1's **Initialize** event, UserForm2 is loaded and shown. When the user clicks UserForm2, it is hidden and UserForm1 appears. When UserForm1 is clicked, UserForm2 is shown again.

VBCopy

```
' This is the Initialize event procedure for UserForm1
PrivateSub UserForm_Initialize()
    Load UserForm2
    UserForm2.Show
EndSub
' This is the Click event for UserForm2
PrivateSub UserForm_Click()
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
    UserForm2.Hide
EndSub

' This is the click event for UserForm1
PrivateSub UserForm_Click()
    UserForm2.Show
EndSub
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

UNIT-4

VB.NET Classes and Object

A class is a group of different data members or objects with the same properties, processes, events of an object, and general relationships to other member functions. Furthermore, we can say that it is like a template or architect that tells what data and function will appear when it is included in a class object. For example, it represents the method and variable that will work on the object of the class.

Objects are the basic run-time units of a class. Once a class is defined, we can create any number of objects related to the class to access the defined properties and methods. For example, the Car is the Class name, and the speed, mileage, and wheels are attributes of the Class that can be accessed by the Object.

Creating the Class

We can create a class using the Class keyword, followed by the class name. And the body of the class ended with the statement End Class. Following is the general syntax for creating classes and objects in the VB.NET programming language

1. [Access_Specifier] [Shadows] [MustInherit | NotInheritable] [Partial] Class ClassName
2. ' Data Members or Variable Declaration
3. ' Methods Name
4. ' Statement to be executed
5. End Class

Where,

- **Access_Specifier:** It defines the access levels of the class, such as Public, Private or Friend, Protected, Protected Friend, etc. to use the method. (It is an optional parameter).
- **Shadows:** It is an optional parameter. It represents the re-declaration of variables and hides an identical element name or set of overloaded elements in a base class.
- **MustInherit:** It is an optional parameter that specifies that the class can only be used as a base class, and the object will not directly access the base class or the abstract class.
- **NotInheritable:** It is also an optional parameter that representing the class not being used as a base class.

- **Partial:** As the name defines, a Partial represents the partial definition of the class (optional).
- **Implements:** It is used to specify interfaces from which the class inherits (optional).

My_program.vb

```
1. Public Class My_program
2. ' properties, method name, etc
3. ' Statement to be executed
4. End Class
```

In the above syntax, we have created a class with the name 'My_program' using the Class keyword.

The Syntax for creating an object

```
1. Dim Obj_Name As Class_Name = New Class_Name() ' Declaration of object
2. Obj_Name.Method_Name() ' Access a method using the object
```

In the above syntax, we have created an instance **(Obj_Name)** for the class **Class_Name**. By using the object name **'Obj_Name'** to access all the data members and the method name of **Class_Name**.

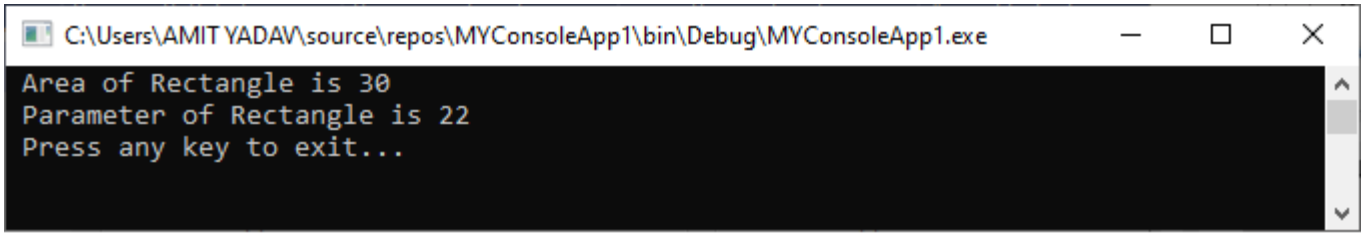
Let's create a program to find the Area and Parameter of a rectangle using the class and object in VB.NET.

My_program.vb

```
1. Imports System
2. Module My_program
3.     Sub Main()
4.         Dim rect As Rectangle = New Rectangle() 'create an object
5.         Dim rect2 As Rectangle = New Rectangle() 'create an object
6.         Dim area, para As Integer
7.
8.         'rect specification
9.         rect.setLength = (5)
10.        rect.setBreadth= (6)
11.
12.        'rect2 specification
```

```
13.         rect2.setLength = (5)
14.         rect2.setBreadth = (10)
15.
16.         'Area of rectangle
17.         area = rect.length * rect.Breadth
18.         'area = rect.GetArea()
19.         Console.WriteLine(" Area of Rectangle is {0}", area)
20.
21.         'Parameter of rectangle
22.         'para = rect.GetParameter()
23.          para = 2 (rect2.length + rect.Breadth)
24.         Console.WriteLine(" Parameter of Rectangle is {0}", para)
25.         Console.WriteLine(" Press any key to exit...")
26.         Console.ReadKey()
27.     End Sub
28.     Public Class Rectangle
29.         Public length As Integer
30.         Public Breadth As Integer
31.
32.         Public Sub setLength(ByVal len As Integer)
33.             length = len
34.         End Sub
35.
36.         Public Sub setBreadth(ByVal bre As Integer)
37.             Breadth = bre
38.         End Sub
39.         Public Function GetArea() As Integer
40.             Return length * Breadth
41.         End Function
42.
43.         Public Function GetParameter() As Integer
44.             Return 2 * (length + Breadth)
45.         End Function
46.     End Class
47. End Module
```

Output:



Member Function

The member function of a class is used to define the structure of member inside the definition of the class. It can be accessed by all defined objects of the class and operated on the data member. Furthermore, member variables are the attributes of an object to be implemented to a member function. And we can access member variables using the public member function.

Constructor and Destructor

In VB.NET, the constructor is a special method that is implemented when an object of a particular class is created. Constructor is also useful for creating and setting default values for a new object of a data member.

When we create a class without defining a constructor, the compiler automatically creates a default constructor. In this way, there is a constructor that is always available in every class. Furthermore, we can create more than one constructor in a class with the use of **New** keyword but with the different parameters, and it does not return anything.

Default Constructor: In VB.NET, when we create a constructor without defining an argument, it is called a default constructor.

VB.NET Default Constructor Syntax

The following is the syntax for creating a constructor using the New keyword in VB.NET.

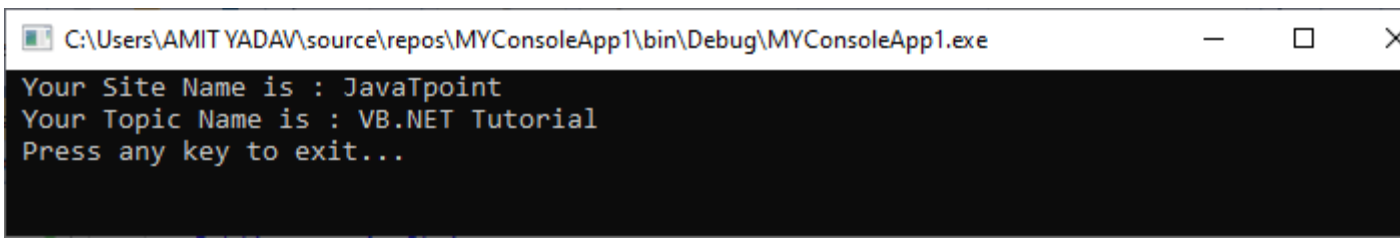
```
1. Public Class MyClass
2. ' Creates a Constructor using the New
3. Public Sub New()
4.     'Statement to be executed
5.         End Sub
6. End Class
```

Let's create a program to define the default constructor in a VB.NET programming language.

Default_Const.vb

```
1.  Imports System
2.  Module Default_Const
3.      Class Tutor
4.          Public name As String
5.          Public topic As String
6.          ' Create a default constructor
7.          Public Sub New()
8.              name = "Sai College"
9.              topic = "VB.NET Tutorial"
10.         End Sub
11.     End Class
12.     Sub Main()
13.         ' The constructor will automatically call when the instance of the class is created
14.         Dim tutor As Tutor = New Tutor() ' Create an object as a tutor
15.         Console.WriteLine(" Your Site Name is : {0}", tutor.name)
16.         Console.WriteLine(" Your Topic Name is : {0}", tutor.topic)
17.         Console.WriteLine(" Press any key to exit...")
18.         Console.ReadKey()
19.     End Sub
20. End Module
```

Output:



In the above example, we created a class **'Tutor'** and defined a default constructor method with **New()** keyword without passing any arguments. When the object (tutor) is created, the default constructor is called into the class.

Parameterized Constructor

In VB.NET, when we pass one or more arguments to a constructor, the constructor is known as a parameterized constructor. And the object of the class should be initialized with arguments when it is created.

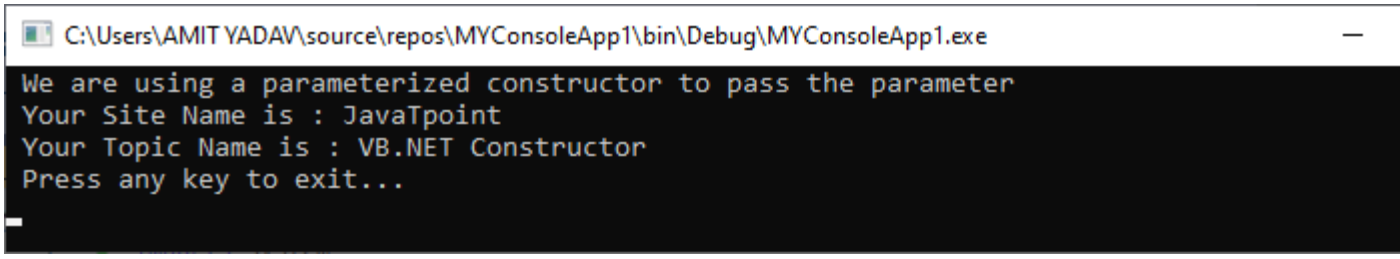
Let's create a program to use the parameterized constructor to pass the argument in a class.

Para_Const.vb

```
1. Imports System
2. Module Para_Const
3.     Class Tutor
4.         Public name As String
5.         Public topic As String
6.         ' Create a parameterized constructor to pass parameter
7.         Public Sub New(ByVal a As String, ByVal b As String)
8.             name = a
9.             topic = b
10.            Console.WriteLine(" We are using a parameterized constructor to pass the parameter"
    )
11.        End Sub
12.    End Class
13.    Sub Main()
14.        ' The constructor will automatically call when the instance of the class is created
15.        Dim tutor As Tutor = New Tutor("saicollege", "VB.NET Constructor")
16.        Console.WriteLine(" Your Site Name is : {0}", tutor.name)
17.        Console.WriteLine(" Your Topic Name is : {0}", tutor.topic)
18.        Console.WriteLine(" Press any key to exit...")
19.        Console.ReadKey()
20.    End Sub
21. End Module
```

Output:

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



VB.NET Destructor

In VB.NET, Destructor is a special function that is used to destroy a class object when the object of the class goes out of scope. It can be represented as the **Finalize()** method and does not accept any parameter nor return any value. Furthermore, it can be called automatically when a class object is not needed.

VB.NET Destructor Syntax

Destructor using the Finalize() method in VB.NET.

```
Class My_Destructor
 ' define the destructor
    Protected Overrides Sub Finalize()
' Statement or code to be executed
End Sub
End Class
```

Let's create a program to use the Finalize() method in VB.NET Destructor.

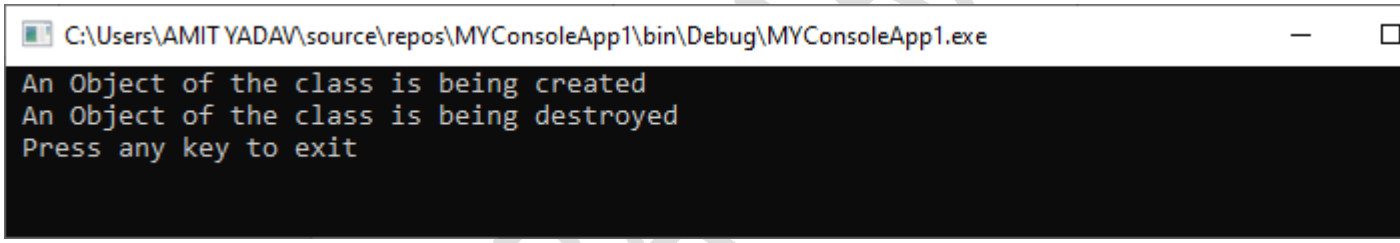
Destruct.vb

```
Imports System
Module Destruct
    Class Get_Destroy
        Public Sub New()
            Console.WriteLine(" An Object of the class is being created")
        End Sub
        ' Use Finalize() method of Destructor
        Protected Overrides Sub Finalize()
            Console.WriteLine(" An Object of the class is being destroyed")
            Console.WriteLine(" Press any key to exit")
        End Sub
    End Class
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
13.
14.     Sub Main()
15.         Get_Details() ' After invoking the Get_Details() method, garbage collector is called auto
    matically
16.         GC.Collect()
17.         Console.ReadKey()
18.         End Sub
19.     Public Sub Get_Details()
20.         Dim dest As Get_Destroy = New Get_Destroy()
21.     End Sub
22. End Module
```

Output:



Constructor Overloading

In VB.NET, the overloading of constructors is a concept in which we can overload constructors by defining more than one constructor with the same name but with different parameter lists to perform different tasks.

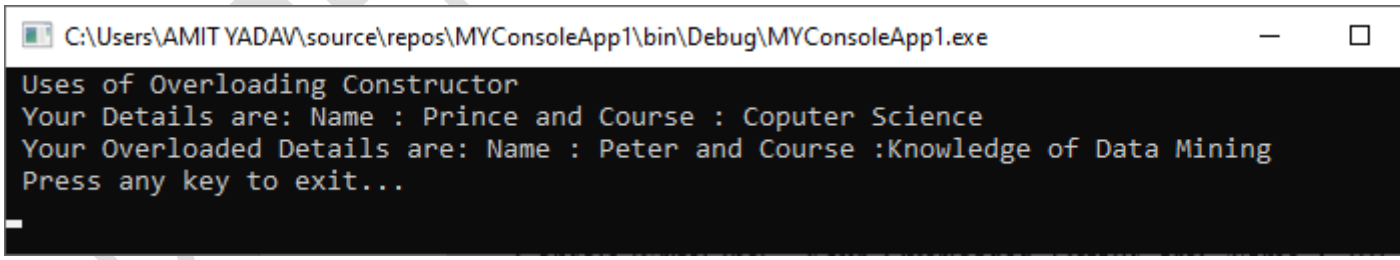
Let's create a program to use the Constructor Overloading in a class.

Const_Over.vb

```
1. Imports System
2. Module Const_Over
3.     Class Details
4.         Public name As String
5.         Public course As String
6.         ' Define Default Constructor
7.         Public Sub New()
8.             name = "Prince"
9.             course = "Computer Science"
```

```
10.             Console.WriteLine(" Uses of Overloading Constructor")
11.         End Sub
12.         ' Create a parametrized constructor
13.         Public Sub New(ByVal a As String, ByVal b As String)
14.             name = a
15.             course = b
16.         End Sub
17.     End Class
18.     Sub Main()
19.         ' Called default constructor
20.         Dim det As Details = New Details()
21.         ' Called the parametrized constructor
22.         Dim det1 As Details = New Details("Peter", "Knowledge of Data Mining")
23.         Console.WriteLine(" Your Details are: Name : {0} and Course : {1}", det.name, det.course)

24.         Console.WriteLine(" Your Overloaded Details are: Name : {0} and Course :{1}", det1.name
    , det1.course)
25.         Console.WriteLine(" Press any key to exit...")
26.         Console.ReadKey()
27.     End Sub
28. End Module
```

Output:



Inheritance

In VB.NET, **Inheritance** is the process of creating a new class by inheriting properties and functions from the old class and extending the functionality of the existing class. It also provides a reusable and faster implementation time of the code. When we create a derived or inherited class, inheritance allows us to inherit all the properties of the existing class. The old class is known as the **base class**, and the inherited class is known as the **derived class**.

Inheritance Syntax

The following is the syntax of Inheritance in VB.NET.

```
<access_modifier> Class <Name_of_baseClass>
' Implementation of base class
End Class
<access_modifier> Class <Name_of_derivedClass>
Inherits Name_of_baseClass
' Implementation of derived class
End Class
```

Let's create a program to understand the concept of Inheritance in VB.NET

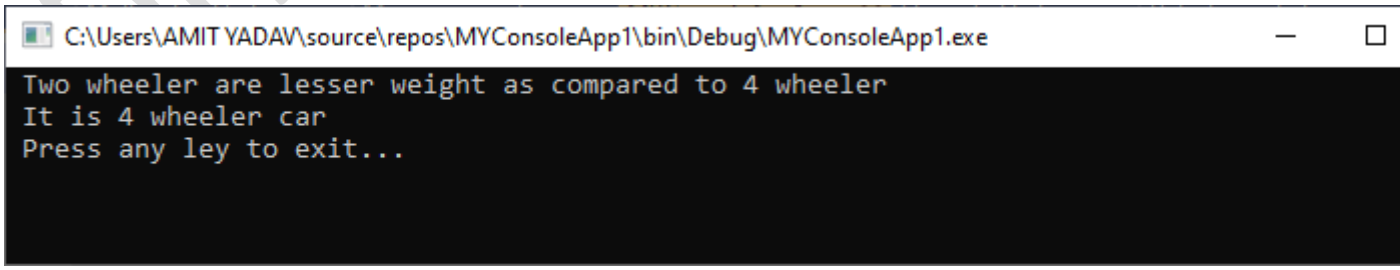
Simple_Inherit.vb

```
Imports System
Module Simple_Inherit
    Public Class Vehicle
        Public Sub speed()
            Console.WriteLine(" 4 Wheeler cars are more luxurious than 2 Wheeler")
        End Sub

        Public Overridable Sub perform()
            Console.WriteLine(" Both Vehicles are good, but in a narrow lane, two-
wheeler are more comfortable than 4-Wheeler")
        End Sub
    End Class
```


PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

```
Class Bike
    Inherits Vehicle
    Public Overrides Sub perform()
        Console.WriteLine(" Two-wheeler are lesser weight as compared to 4 wheeler")
    End Sub
End Class
Class Car
    Inherits Vehicle
    Public Overrides Sub perform()
        Console.WriteLine(" It is 4 wheelar car")
    End Sub
End Class

Sub Main()
    Dim vehicle As Vehicle = New Vehicle()
    Dim bike As Bike = New Bike()
    Dim car As Car = New Car()
    vehicle = bike
    vehicle.perform()
    vehicle = car
    vehicle.perform()
    Console.WriteLine(" Press any ley to exit...")
    Console.ReadKey()
End Sub
End Module
```

Output:



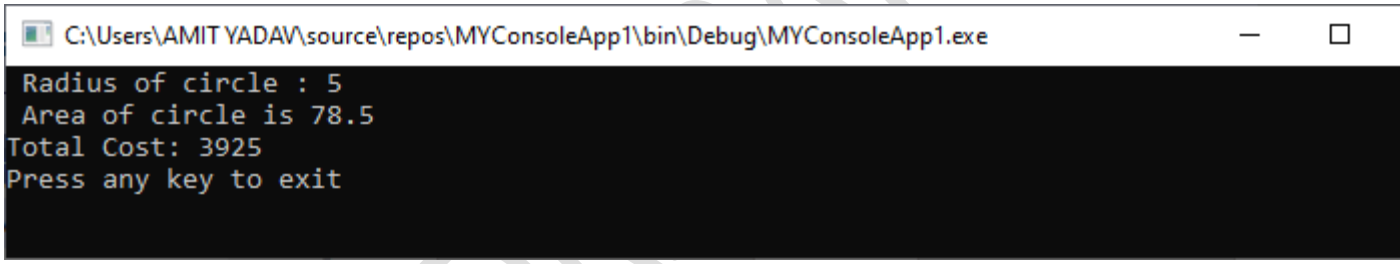
Let's create a program of inheritance using the MyBase in VB.NET.

Inherit_class.vb

```
Module Inherit_class
    Class Circle   'base class
        Protected radius As Integer
        Public Const PI As Double = 3.14
        Public Sub New(ByVal r As Integer)
            radius = r
        End Sub
        Public Function GetAreaCircle() As Double
            Return (PI * radius * radius)
        End Function
        Public Overridable Sub Show()
            Console.WriteLine(" Radius of circle : {0}", radius)
            Console.WriteLine(" Area of circle is {0}", GetAreaCircle())
        End Sub
    End Class

    'Derived Class
    Class MyDeriveClass : Inherits Circle
        Private dimen As Double
        Public Sub New(ByVal r As Integer)
            MyBase.New(r)
        End Sub
        Public Function Getdimen() As Double
            Dim dimen As Double
            dimen = GetArea() * 50
            Return dimen
        End Function
        Public Overrides Sub Show()
            MyBase.Show()
            Console.WriteLine("Total Cost: {0}", Getdimen())
```

```
    End Sub
End Class

Class Circle_Class
    Shared Sub Main()
        Dim c As MyDeriveClass = New MyDeriveClass(5)
        c.Show()
        Console.WriteLine("Press any key to exit")
        Console.ReadKey()
    End Sub
End Class
End Module
```

Output:



Multi-Level Inheritance

VB.NET only supports single inheritance that means a class can be inherited from the base class. However, VB.NET uses hierarchical inheritance to extend one class to another class, which is called **Multi-Level Inheritance**. For example, Class **C** extends Class **B**, and Class **B** extends Class **A**, Class **C** will inherit the member of both class **B** and Class **A**. The process of extending one class to another is known as multilevel inheritance.

```
1. Public Class A
2.     ' implementation of code
3. End Class
4.
5. Public Class B
6.     Inherits A
7.     ' Implementation of Code
8. End Class
9.
```

```
10. Public Class C
11.     Inherits B
12.     ' Implementation of code
13. End Class
```

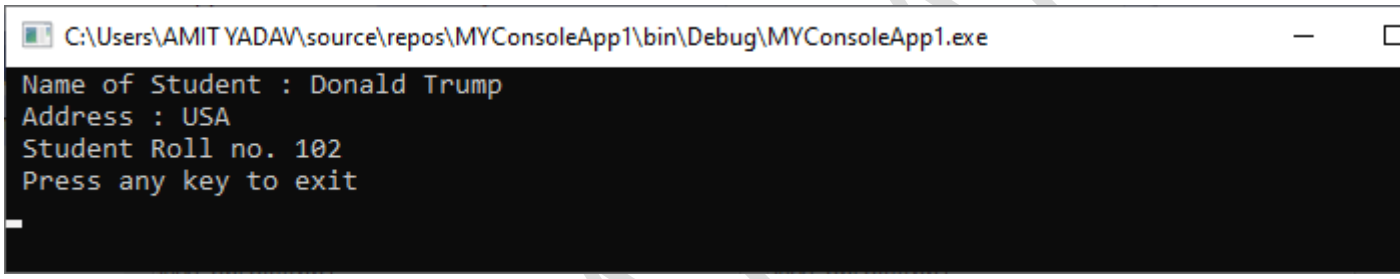
Let's create a program to understand the concept of Multi-Level Inheritance in VB.NET

Multi_inherit1.vb

```
1.  Module Multi_inherit1
2.      Public Class A
3.          Public SName As String
4.          Public Sub Display()
5.              Console.WriteLine(" Name of Student : {0}", SName)
6.          End Sub
7.      End Class
8.      Public Class B
9.          Inherits A
10.         Public place As String
11.         Public Sub GetPlace()
12.             Console.WriteLine(" Address : {0}", place)
13.         End Sub
14.     End Class
15.
16.     Public Class C
17.         Inherits B
18.         Public rollno As Integer
19.         Public Sub GetRollno()
20.             Console.WriteLine(" Student Roll no. {0}", rollno)
21.         End Sub
22.     End Class
23.
24.     Class Profile
25.         Public Sub Main(ByVal args As String())
26.             Dim c As C = New C()
27.             c.SName = "Donald Trump"
28.             c.place = "USA"
```

```
29.             c.rollno = 102
30.             c.Display()
31.             c.GetPlace()
32.             c.GetRollno()
33.             Console.WriteLine(" Press any key to exit")
34.             Console.ReadKey()
35.         End Sub
36.     End Class
37. End Module
```

Output:



Interface

In VB.NET, the interface is similar to a class for inheriting all properties, methods, and events that a class or structure can implement. Using the interface in VB.NET, we can use multiple inheritances of classes. It uses the **Implements** keyword to implement the interfaces, and the Interface keyword is used to create the interface. All interfaces in VB.NET starts with interface.

The Syntax for implementing an interface in a class.

```
1. Class MyClass
2.     Inherits IClass
3.     Private Sub FetchDetails() Implements IClass.FetchDetails
4.     ' Method Implementation
5.     End Sub
6. End Class
```

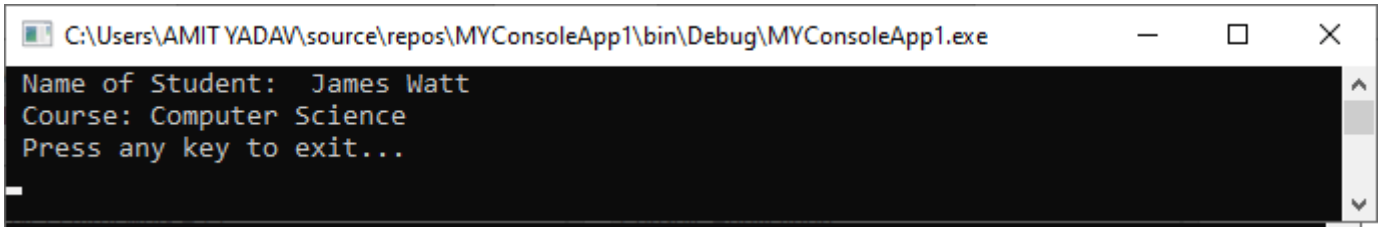
In the above snippets, we inherited an interface (**IClass**) in the Class **MyClass** that implemented to the interface method defined in a class.

Let's create and implement an instance using a class in VB.NET.

Get_Interface.vb

```
Module Get_Interface
    Interface IStudent
        Sub Details(ByVal y As String)
    End Interface
    Class Student
        Implements IStudent
        Public Sub Details(ByVal a As String) Implements IStudent.Details
            '       Throw New NotImplementedException()
            Console.WriteLine(" Name of Student: {0}", a)
        End Sub
    End Class

    Class Student1
        Implements IStudent
        Public Sub Details(ByVal a As String) Implements IStudent.Details
            'Throw New NotImplementedException()
            Console.WriteLine(" Course: {0}", a)
        End Sub
    End Class
    Sub Main(ByVal args As String())
        Dim stud As IStudent = New Student()
        stud.Details(" James Watt")
        Dim stud1 As IStudent = New Student1()
        stud1.Details("Computer Science")
        Console.WriteLine(" Press any key to exit...")
        Console.ReadKey()
    End Sub
End Module
```

Output:



UNIT-5

A database management system typically deals with storing, modifying, and extracting information from a database. It can also add, edit and delete records from the database. However, a DBMS can be very difficult to handle by ordinary people or business men who have no technological backgrounds. Fortunately, we can create user-friendly database applications to handle the aforementioned jobs with the DBMS running in the background. One of the best programs that can create such database application is none other than Visual Basic 2010.

Introduction to ADO.NET

The .NET Framework includes its own data access technology i.e. **ADO.NET.** ADO.NET is the latest implementation of Microsoft's Universal Data Access strategy. ADO.NET consists of managed classes that allows .NET applications to connect to data sources such as Microsoft SQL Server, Microsoft Access, Oracle, XML, etc., execute commands and manage disconnected data.

Microsoft ADO.NET is the latest improvement after ADO. Firstly, ADO.NET was introduced in the 10th version of the .NET framework, which helps in providing an extensive array of various features to handle data in different modes, such as connected mode and disconnected mode. In connected mode, we are dealing with live data and in disconnected mode, data is provided from the data store.

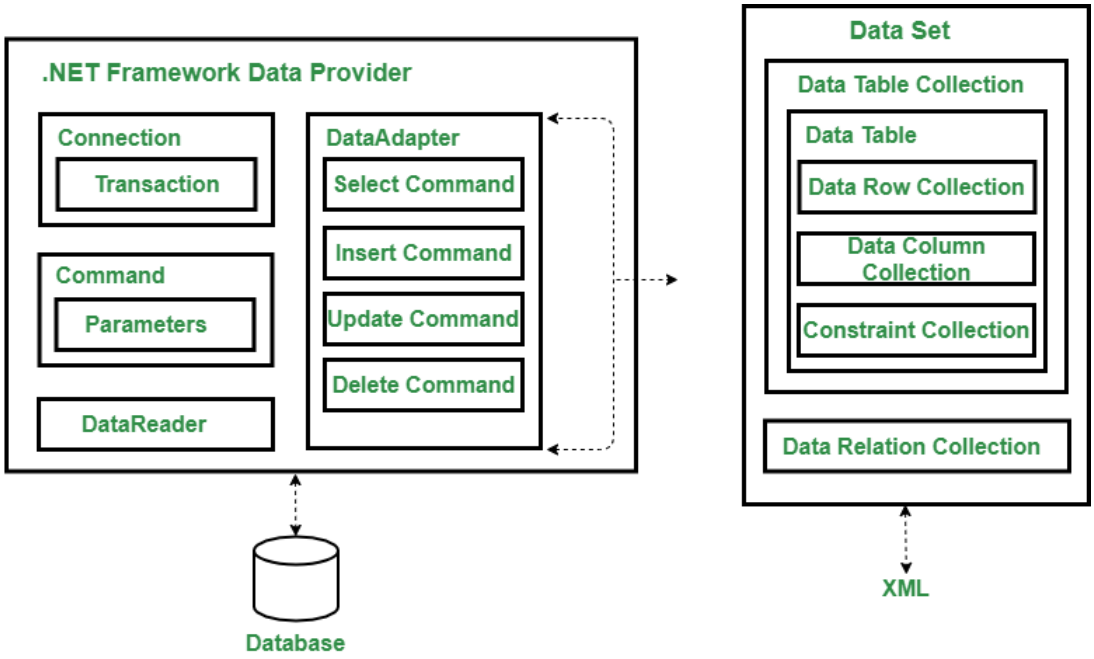
ADO.NET was primarily developed to address two ways to work with data that we are getting from data sources. The two ways are as follows :

1. The first is to do with the user's need to access data once and to iterate through a collection of data in a single instance.
2. The second way to work with data is disconnected architecture mode, in which we have to grab a collection of data and we use this data separately from the data store itself.

Architecture of ADO.NET :
ADO.NET uses a multi-layered architecture that revolves around a few key concepts as –

- asConnection
- Command
- DataSet objects

The ADO.NET architecture is a little bit different from the ADO, which can be shown from the following figure of the architecture of ADO.NET.



Architecture of ADO.NET

One of the key differences between ADO and ADO.NET is how they deal with the challenge of different data sources. In ADO.NET, programmers always use a generic set of objects, no matter what the underlying data source is. **For example,** if we want to retrieve a record from an Oracle Database, we use the same connection class that we use to tackle the same task with SQL Server. This is not the case with ADO.NET, which uses a data provider model and the DataSet.

Features of ADO.NET :
The following are the features of ADO.NET –

- **Interoperability-**
  We know that XML documents are text-based formats. So, one can edit and edit XML documents using standard text-editing tools. ADO.NET uses XML in all data exchanges and for internal representation of data.
- **Maintainability –**
  ADO.NET is built around the idea of separation of data logic and user interface. It means that we can create our application in independent layers.

- **Programmability (Typed Programming) –**
  It is a programming style in which user words are used to construct statements or evaluate expressions. For example: If we want to select the "Marks" column from "Kawal" from the "Student" table, the following is the way to do so:

```
DataSet.Student("Kawal").Marks;
```

- **Performance –**
  It uses disconnected data architecture which is easy to scale as it reduces the load on the database. Everything is handled on the client-side, so it improves performance.
- **Scalability –**
  It means meeting the needs of the growing number of clients, which degrading performance. As it uses disconnected data access, applications do not retain database lock connections for a longer time. Thus, it accommodates scalability by encouraging programmers to conserve limited resources and allow users to access data simultaneously.

ADO.NET Framework Data Providers

Data provider is used to connect to the database, execute commands and retrieve the record. It is lightweight component with better performance. It also allows us to place the data into DataSet to use it further in our application.

The .NET Framework provides the following data providers that we can use in our application.

.NET Framework data provider	Description
.NET Framework Data Provider for SQL Server	It provides data access for Microsoft SQL Server. It requires the **System.Data.SqlClient** namespace.
.NET Framework Data Provider for OLE DB	It is used to connect with OLE DB. It requires the **System.Data.OleDb** namespace.
.NET Framework Data Provider for ODBC	It is used to connect to data sources by using ODBC. It requires the **System.Data.Odbc** namespace.
.NET Framework Data Provider for Oracle	It is used for Oracle data sources. It uses the **System.Data.OracleClient** namespace.
EntityClient Provider	It provides data access for Entity Data Model applications. It requires the **System.Data.EntityClient** namespace.
.NET Framework Data Provider for SQL Server Compact 4.0.	It provides data access for Microsoft SQL Server Compact 4.0. It requires the **System.Data.SqlServerCe** namespace.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

.NET Framework Data Providers Objects

Following are the core object of Data Providers.

Object	Description
Connection	It is used to establish a connection to a specific data source.
Command	It is used to execute queries to perform database operations.
DataReader	It is used to read data from data source. The DbDataReader is a base class for all DataReader objects.
DataAdapter	It populates a DataSet and resolves updates with the data source. The base class for all DataAdapter objects is the DbDataAdapter class.

.NET Framework Data Provider for SQL Server

Data provider for SQL Server is a lightweight component. It provides better performance because it directly access SQL Server without any middle connectivity layer. In early versions, it interacts with ODBC layer before connecting to the SQL Server that created performance issues.

The .NET Framework Data Provider for SQL Server classes is located in the **System.Data.SqlClient** namespace. We can include this namespace in our C# application by using the following syntax.

1. using System.Data.SqlClient;

This namespace contains the following important classes.

Class	Description
SqlConnection	It is used to create SQL Server connection. This class cannot be inherited.
SqlCommand	It is used to execute database queries. This class cannot be inherited.
SqlDataAdapter	It represents a set of data commands and a database connection that are used to fill the DataSet. This class cannot be inherited.
SqlDataReader	It is used to read rows from a SQL Server database. This class cannot be inherited.
SqlException	This class is used to throw SQL exceptions. It throws an exception when an error is occurred. This class cannot be inherited.

.NET Framework Data Provider for Oracle

It is used to connect with Oracle database through Oracle client. The data provider supports Oracle client software version 8.1.7 or a later version. This data provider supports both local and distributed transactions.

Oracle Data Provider classes are located into **System.Data.OracleClient** namespace. We must use both **System.Data.OracleClient** and **System.data** to connect our application with the Oracle database.

```
1. using System.Data;
2. using System.Data.OracleClient;
```

Which .NET Framework Data Provider is better

Selection of data provider is depends on the design and data source of our application. Choice of optimum .NET Framework data provider can improve the performance, capability and integrity of our application. The following table demonstrates advantages and disadvantages of data provider.

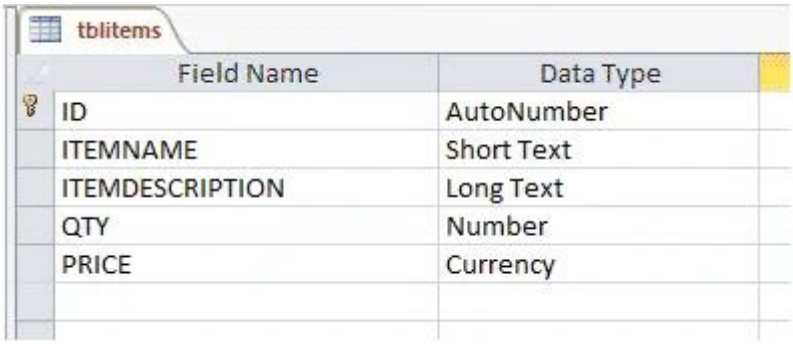
Data Provider	Note
.NET Framework Data Provider for SQL Server	It is good for middle-tier applications, single-tier applications that use Microsoft SQL Server.
.NET Framework Data Provider for OLE DB	It is good for single-tier applications that use Microsoft Access databases.
.NET Framework Data Provider for ODBC	It is good for middle and single-tier applications that use ODBC data sources.
.NET Framework Data Provider for Oracle	It is good for middle and single-tier applications that use Oracle data sources.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

Steps How to Connect Access Database in VB.Net

1. **Step 1: Create an MS Access Database.**
   Open an **MS Access Database** in your Computer and Create a **Blank Database** and Save it as ***"inventorydb.accdb"***.

2. **Step 2: Create a Database Table.**
   To create a table, follow the image below and save it as ***"tblitems"***.



tblitems

Field Name	Data Type
ID	AutoNumber
ITEMNAME	Short Text
ITEMDESCRIPTION	Long Text
QTY	Number
PRICE	Currency

3. **Step 3: Populate the table.**
   Add sample records in the table. follow the sample records in the image below.

Sort & Filter | Records | Find

tblitems

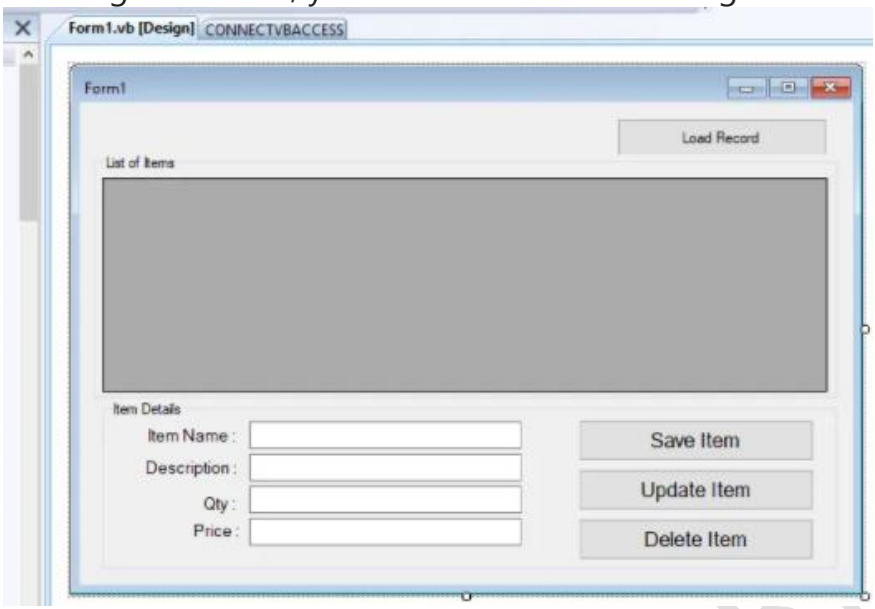
ID	ITEMNAME	ITEMDESCRI	QTY	PRICE	Click to Add
1	MONITOR	COMPUTER MC	5	$60.00	
2	MONTHERBOA	COMPUTER MC	10	$200.00	
3	KEYBOARD	KEYBOARD COL	10	$150.00	
5	laptop	sony laptop	10	$1,500.00	
(New)			0	$0.00	

4. **Step 4: Create a VB.Net Application.**
   Open Visual Studio and Create a Visual Basic Application project and Save it as ***"connectvbaccess"***.

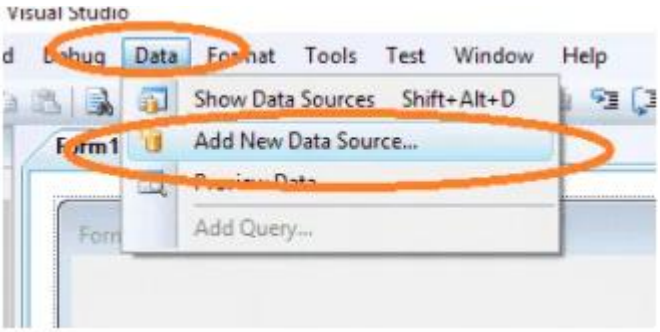
5. **Step 5: Design the user interface.**

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

To design the form, you need to follow the image below.

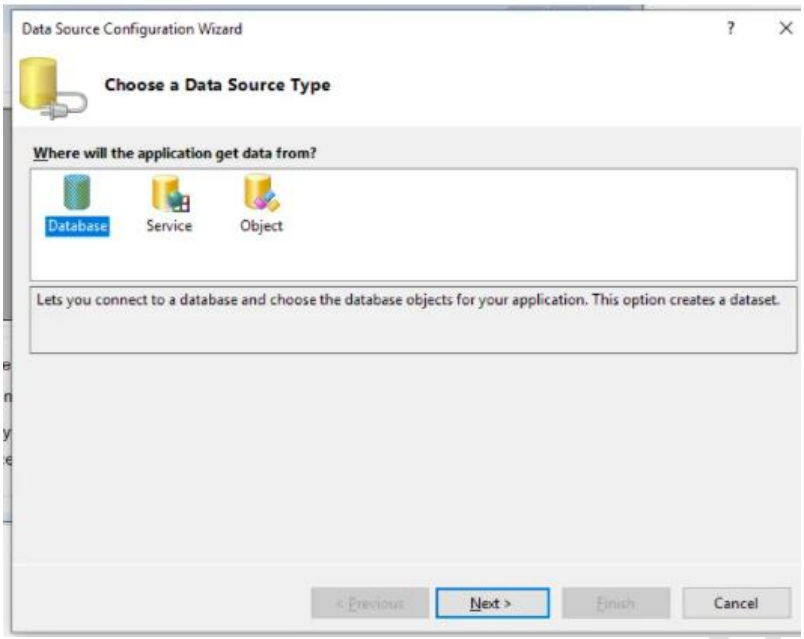


6. **Step 6: Add New Data Source.**
   On the menu, click data and select **"Add new Data Source.."**

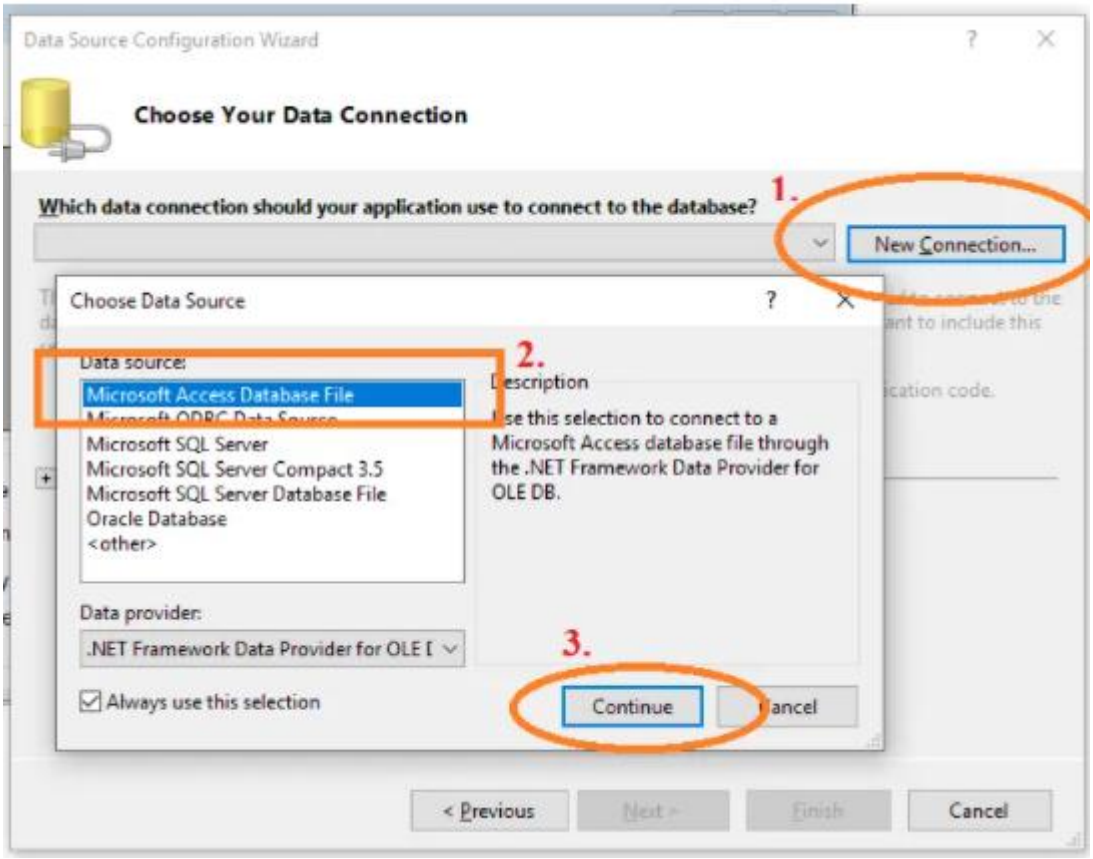


7. **Step 7: Choose a Data Source Type**
   Select **Database** and **click Next.**

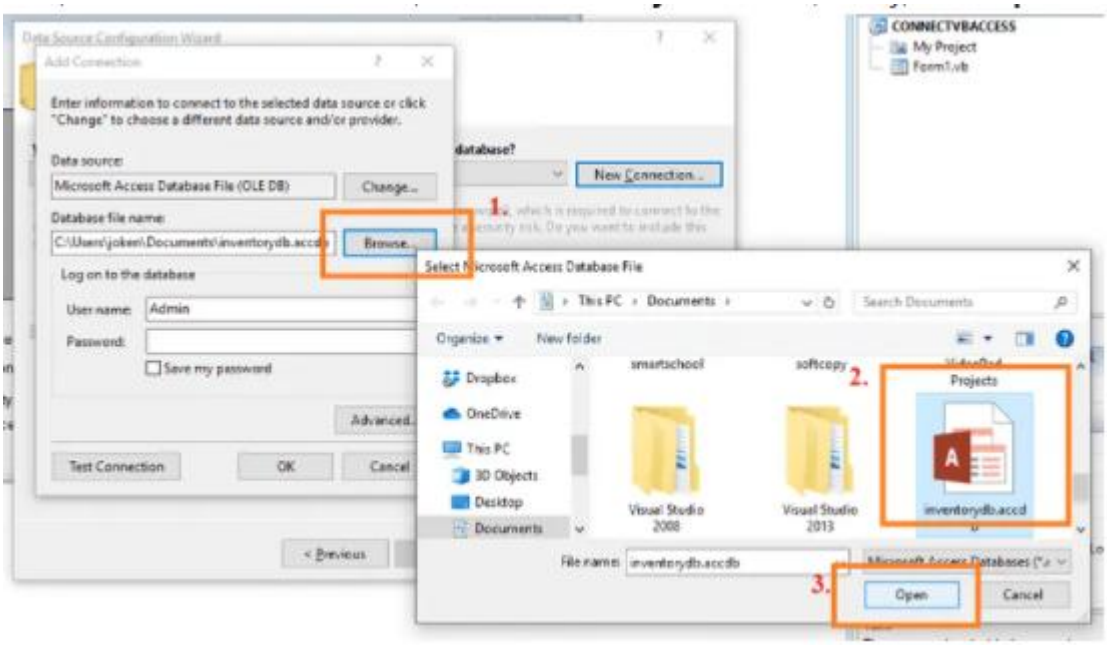
PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI



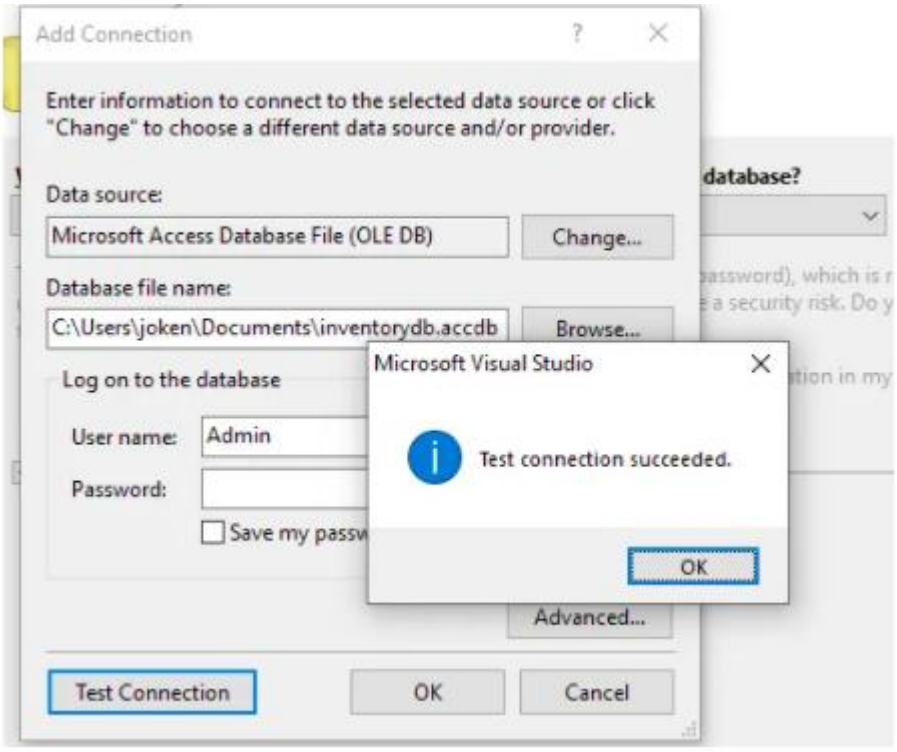
8. **Step 7: Choose a Data Source**
   Click **new Connection** then, Select **Microsoft AccessDatabase file** and, Click **Continue**. *Follow the image below.*



9. **Step 9: Add Connection**
   First, click **Browse** Button then, Select "**inventorydb.accdb"**, Lastly, Click **Open**.



10. **Step 10: Test Connection**
    To test the connection, click the **"Test Connection"** button, finally click **"OK"** button at the side of the **"Cancel"** button.



PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI

11. **Step 11: Copy the Connection String.**
    Copy the connection string so that we can use this in our next step.

12. **Step 12: Start Coding.**
    In this final step, we will now start adding functionality to our vb.net program by adding some functional codes.

Code To Connect Access Database in VB.Net

Double the **"Form1"** and add the following code under **"Public Class Form1"**.

```
Dim con As New OleDb.OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:\Users\joken\Documents\inventorydb.accdb")
```

The Code above started with a **Declaration of Variable** name ***"con"*** with an **Ole Object Type OledbConnection**.

Inside **OledbConnection**, we pasted the connection string we copied from the **"Step 11"** instructions.

Test the Connection of Access database in VB.Net

To test the **Connection between ms access database and VB.Net**, Double click the **form1** add the following code under "Form1_Load" events.

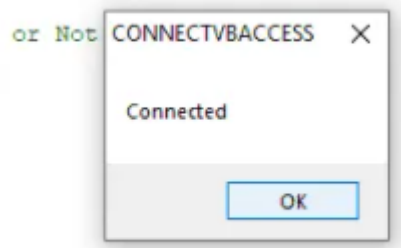
```
Try
  con.Open()

  If con.State = ConnectionState.Open Then
    MsgBox("Connected")
  Else
    MsgBox("Not Connected!")
```

```
    End If
  Catch ex As Exception
    MsgBox(ex.Message)
  Finally
    con.Close()

  End Try
```

- We use try-catch to the exceptions that may occur during runtime.
- open the connection
- check using if statement if the connection is open
- 'Display a message box if successfully connected or Not
- close the connection

Press **"F5"** to run the Project.

When you run the project it will give you this message.



How to Load Record from Access Database to Datagridview In VB.Net

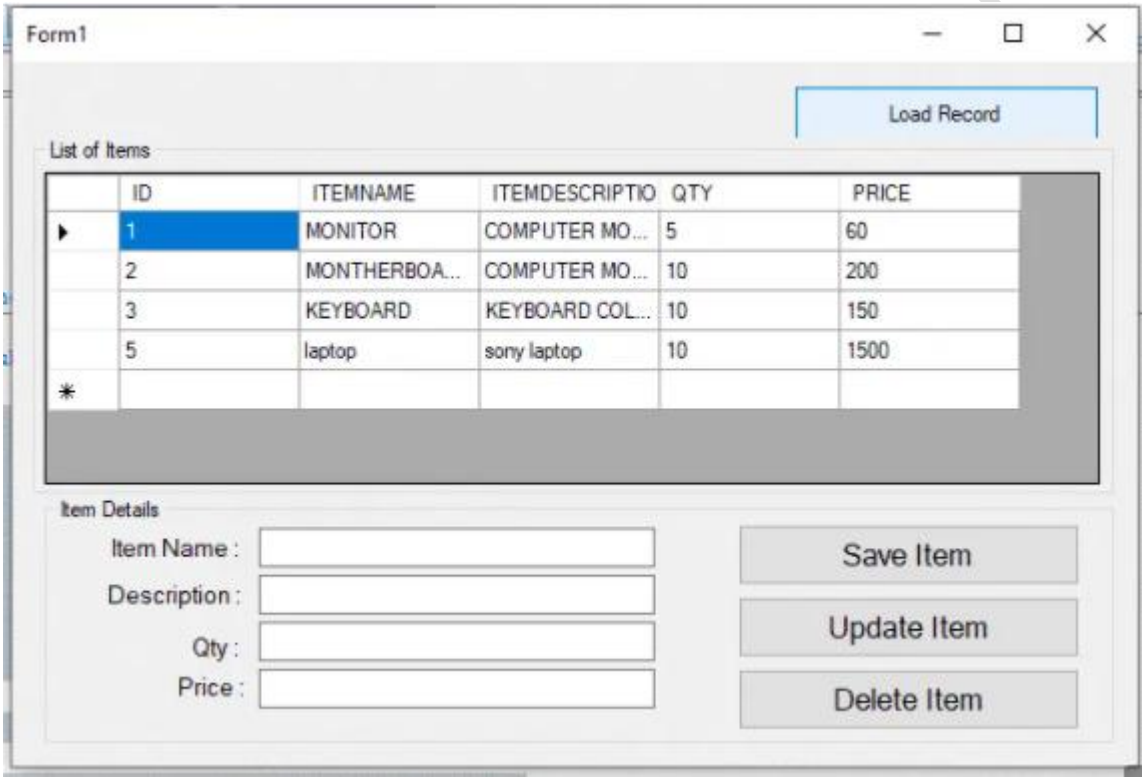
In this section, we will learn **how to load a record from the Access database to Datagridview using vb.net**. To start with, double click the "**Load record"** button and add the following code.

```
Try
    Dim sql As String
    Dim cmd As New OleDb.OleDbCommand
    Dim dt As New DataTable
    Dim da As New OleDb.OleDbDataAdapter
    con.Open()
    sql = "Select * from tblitems"
    cmd.Connection = con
    cmd.CommandText = sql
    da.SelectCommand = cmd
     da.Fill(dt)
```

```
    DataGridView1.DataSource = dt
Catch ex As Exception
    MsgBox(ex.Message)
Finally
    con.Close()

End Try
```

After adding the code, you may press **F5** or click the Start debugging button to test the code. The output should look like as shown below.

How to Load Record from Access Database to Datagridview In VB.Net



Save Record in Access Database using VB.net

In this section, we learn **how to save the record in the access database using vb.net.** To do this, double click the **"Save Item"** button and add the following code.

```
Try
        Dim sql As String
        Dim cmd As New OleDb.OleDbCommand
        con.Open()
```

```
        sql = "INSERT INTO tblitems (ITEMNAME,ITEMDESCRIPTION,QTY,PRICE)
values ('" &amp;amp; txtitemname.Text &amp;amp; "', '" &amp;amp; txtdescription.Text &amp;amp; "'," &amp;amp;
Val(txtqty.Text) &amp;amp; "," &amp;amp; Val(txtprice.Text) &amp;amp; ");"
        cmd.Connection = con
        cmd.CommandText = sql
        i = cmd.ExecuteNonQuery
        If i &gt; 0 Then
           MsgBox("New record has been inserted successfully!")
        Else
           MsgBox("No record has been inserted successfully!")
        End If

      Catch ex As Exception
        MsgBox(ex.Message)
      Finally
        con.Close()

      End Try
```

Code explanation

- We start the code by adding try-catch
- declare variable "sql" as a string this will hold the INSERT STATEMENT.
- Declare variable **"cmd"** as oledbCommand it represents an SQL Statement or Store procedure to execute against a data source.
- next, we open the connection
- Then, we assign a query command to **"sql"** variable.
- **Execute** the ***Command***
- Then in we assign the result of "**ExecuteNonQuery**" to **"i"** variable
- Next, we check if the value of the variable "**i**" is greater than 0
- we use **if condition statement** if the result is true then will display a message box **"New record has been inserted successfully!"**.
- else if the result is false, the display is "**No record has been inserted successfully!"**

At this time, you can now press **"F5"** to test code.

Updating Records from Access Database in VB.Net

In this section, we will learn **how to update records from an access database using vb.net**. In order for us to proceed in **updating the record**, we will add first a code to pass value from datagridview to textboxes.

To start with, go back to form design and double click the datagridview. And Change the Event to "**CellClick"** from **"CellContentClick"**. It means that every time the user clicks the selected data in the Datagrid view, the value will automatically pass to the textboxes. So here's the following code.

```
Me.Text = DataGridView1.CurrentRow.Cells(0).Value
txtitemname.Text = DataGridView1.CurrentRow.Cells(1).Value
txtdescription.Text = DataGridView1.CurrentRow.Cells(2).Value
txtqty.Text = DataGridView1.CurrentRow.Cells(3).Value
txtprice.Text = DataGridView1.CurrentRow.Cells(4).Value
```

And here's the Following code for Updating the record from access database using vb.net.

```
Try
  Dim sql As String
  Dim cmd As New OleDb.OleDbCommand
  con.Open()
  sql = "UPDATE tblitems SET ITEMNAME='" &amp; txtitemname.Text &amp; "',
ITEMDESCRIPTION='" &amp; txtdescription.Text &amp; "', " &amp; _
   " QTY=" &amp; Val(txtqty.Text) &amp; ", PRICE=" &amp; Val(txtprice.Text)
&amp; " WHERE ID=" &amp; Val(Me.Text) &amp; ""
  cmd.Connection = con
  cmd.CommandText = sql

  i = cmd.ExecuteNonQuery
  If i &gt; 0 Then
    MsgBox("Record has been UPDATED successfully!")

  Else
    MsgBox("No record has been UPDATED!")
  End If

Catch ex As Exception
  MsgBox(ex.Message)
Finally
  con.Close()

End Try
```

The code we use for **updating the record from ms access database using vb.net** is almost similar to the code above for **inserting a new record to ms access database in vb.net.**

Only the Query is different.

Deleting of Records from Access Database In VB.Net

For **deleting of records from the access database in vb.net,** we will still use the same code in **inserting** and **updating** the record from **access using vb.net.**

Go back to **Form design** and double click the ***"Delete Item"*** button. Then add the following code.

```
Try
        Dim sql As String
        Dim cmd As New OleDb.OleDbCommand
        con.Open()
        sql = "Delete * from tblitems WHERE ID=" &amp;amp; Val(Me.Text) &amp;amp; ""
        cmd.Connection = con
        cmd.CommandText = sql

        i = cmd.ExecuteNonQuery
        If i &gt; 0 Then
           MsgBox("Record has been deleted successfully!")

        Else
           MsgBox("No record has been deleted!")
        End If

    Catch ex As Exception
       MsgBox(ex.Message)
    Finally
       con.Close()

    End Try
```

After this process, you can now run the program to test if all the codes in this tutorial are working well.

What is Data provider?

The data provider is used to connecting to a database. Executing commands and retrieving data, storing it in a **Dataset**, reading the retrieved data and updating the database.

What is database Connection?

This component is used to set up a connection with a data source

What is Command?

A command is a SQL statement or a stored procedure used to retrieve, insert, delete or modify data in a data source.

What is DataAdapter?
This is integral to the working of ADO.Net since data is transferred to and from a database through a data adapter. It retrieves data from a database into a dataset and updates the database. When changes are made to the dataset, the changes in the database are actually done by the data adapter.

What is DataSet?
Dataset is an in-memory representation of data. It is a disconnected, cached set of records that are retrieved from a database. When a connection is established with the database, the data adapter creates a dataset and stores data in it.

What is DataTable?
Datatable consists of the DataRow and DataColumn objects. The DataTable objects are case-sensitive.

What is OledbConnection?
OleDbConnection is designed for connecting to a wide range of databases, like Microsoft Access and Oracle.

What is SqlConnection?
SqlConnection is designed for connecting to Microsoft SQL Server.

PREPARED BY – MR. DHIRENDRA PARATE
ASST. PROF, SAI COLLEGE, BHILAI