

UNIT I- Dot Net Technology

Syllabus

- ☐ The .Net Framework: Introduction
- ☐ Features of .Net
- ☐ Common Language Runtime (CLR)
- ☐ Common Language Specification (CLS)
- ☐ Common Type System (CTS)
- ☐ Microsoft Intermediate Language (MSIL)
- ☐ Just-In-Time Compilation
- ☐ Framework Base Classes.
- ☐ Namespace
- ☐ ***Assemblies***
- ☐ .Net Assemblies
- ☐ Cross Language Interoperability
- ☐ Garbage Collection

Question Bank

Que 1:- Explain Architecture of .Net technology.

Que 2:- Describe feature of .net technology.

Que 3:- describe complete architecture of CLR with all its components.

Que 4 :- What is Garbage Collection? Explain all its advantages .

Que 5 :- Explain JIT and its working.

Que 6 :- Write short Note on a) Namespace

b) Assemblies

Que 7:- What is cross language and interoperability.

Que8:- Describe CTS(Common Type System) in brief.

Que 9:- Define managed code and unmanaged code.

Que10:- Write the steps for program execution in dot net .

Unit-1 Introduction to .Net framework

1. 1 Introduction to Dot Net Framework

- The .Net framework is a software development platform developed by Microsoft.
- The framework was meant to create applications, which would run on the Windows Platform.
- The first version of the .Net framework was released in the year 2002. The version was called .Net framework 1.0.
- The .Net framework can be used to create both - **Form-based** and **Web-based** applications.
- .NET Framework supports more than 60 programming languages in which 11 programming languages are designed and developed by Microsoft. The remaining Non-Microsoft Languages which are supported by .NET Framework but not designed and developed by Microsoft.

❖ Features of .NET Framework

- Interoperability
- Language Independence
- Portability
- High Performance
- Memory Management
- Security
- Performance
- Type safety

• **Interoperability**

It includes a large library and provides language interoperability (each language can use code written in other languages) across several programming languages).

• **Language Independence**

.NET Framework introduces a **Common Type System (CTS)** that defines all possible data types and programming constructs supported by CLR and how they may or may not interact conforming to CLI specification. **Because of this feature, .NET Framework supports the exchange of types and object instances between libraries and applications written using any conforming .NET language.**

• **Type Safety**

Type safety in .NET has been introduced **to prevent the objects of one type from peeking into the memory assigned for the other object.** Writing safe code also means **to prevent data loss during conversion of one type to another.**

• **Portability**

While Microsoft has never implemented the full framework on any system except Microsoft Windows, it has engineered the framework to be cross-platform and implementations are available for other operating systems.

This makes it possible for third parties to create compatible implementations of the framework and its languages on other platforms.

–

- **Security**

.NET Framework has its own security mechanism with two general features: **Code Access Security (CAS)**, and validation and verification.

Memory management

CLR frees the developer from the burden of managing memory (allocating and freeing up when done); it **handles memory management itself by detecting when memory can be safely freed**.

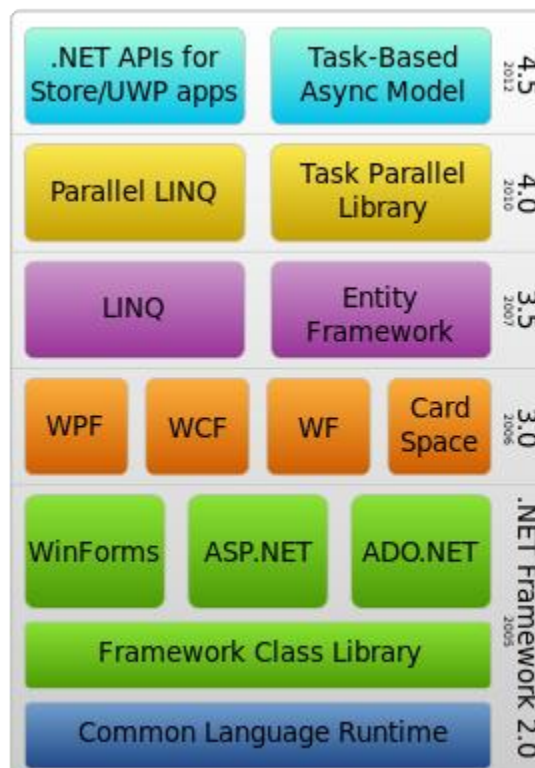
When no reference to an object exists, and it cannot be reached or used, it becomes garbage, eligible for collection.

- **Performance**

When an application is first launched, the .NET Framework compiles the CIL code into executable code using its just-in-time compiler, and caches the executable program into the .NET Native Image Cache. **Due to caching, the application launches faster for subsequent launches**, although the first launch is usually slower.

Major Components of .NET

The diagram given below describes various components of .NET Framework.



The bottom most layer is the CLR - the common runtime language.

Common Language Runtime (CLR)

The CLR is the heart of .NET framework. It is .NET equivalent of Java Virtual Machine (JVM). It is the runtime that converts a MSIL₃ (Micro Soft Intermediate Language) code into

the host machine language code, which is then executed appropriately.

The CLR provides a number of services that include:

- ☐ Loading and execution of codes
- ☐ Memory isolation for application
- ☐ Verification of type safety
- ☐ Compilation of IL into native executable code
- ☐ Providing metadata
- ☐ Automatic garbage collection
- ☐ Enforcement of Security
 - Interoperability with other systems
 - Managing exceptions and errors
 - Provide support for debugging and profiling

Main components of CLR:

- **Common Language Specification (CLS)**
- **Common Type System (CTS)**
- **Garbage Collection (GC)**
- **Just In – Time Compiler (JIT)**

Common Type System (CTS)

The language interoperability, in .NET Class Framework, are not possible without all the language sharing the same data types. What this means is that an -int should mean the same in VB, VC++, C# and all other .NET compliant languages. Same idea follows for all the other data types. This is achieved through introduction of Common Type System (CTS).

CTS, much like Java, defines every data type as a Class. Every .NET compliant language must stick to this definition. Since CTS defines every data type as a class; this means that only Object- Oriented (or Object-Based) languages can achieve .NET compliance. Given below is a list of CTS supported data types:

Data Type	Description
System.Byte	1-byte unsigned integer between 0-255
System.Int16	2-bytes signed integer in the following range: 32,678 to 32,767
System.Int32	4-byte signed integer containing a value in the following range: -2,147,483,648 to 2,147,483,647
System.Int64	8-byte signed integer containing a value from - 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

System.Single	4-byte floating point. The value limits are: for negative values: -3.402823E38 to -1.401298E-45 for positive values: 1.401298E-45 TO 3.402823E38
System.Double	8-bytes wide floating point. The value limits are: for negative values: -1.79769313486231E308 to -4.964065645841247E-324 for positive values: 4.964065645841247E-324 to 1.79769313486232E308
System.Object	4-bytes address reference to an object
System.Char	2-bytes single Unicode Character.
System.String	string of up to 2 billion Unicode characters.
System.Decimal	12-bytes signed integer that can have 28 digits on either side of decimal.
System.Boolean	4-Bytes number that contains true(1) or false (0)

Common Language Specification (CLS)

One of the obvious themes of .NET is unification and interoperability between various programming languages. In order to achieve this, certain rules must be laid and all the languages must follow these rules. In other words we can not have languages running around creating their own extensions and their own fancy new data types. **CLS is the collection of the rules and constraints that every language (that seeks to achieve .NET compatibility) must follow.** Microsoft has defined three level of CLS compatibility/compliance. The goals and objectives of each compliance level have been set aside. The three compliance levels with their brief description are given below:

Compliant producer

The component developed in this type of language can be used by any other language.

Consumer

The language in this category can use classes produced in any other language. In simple words this means that the language can instantiate classes developed in other language. This is similar to how COM components can be instantiated by your ASP code.

Extender

Languages in this category can not just use the classes as in CONSUMER category; but can also extend classes using inheritance.

Languages that come with Microsoft Visual Studio namely Visual C++, Visual Basic and C#; all satisfy the above three categories. Vendors can select any of the above categories as the targeted compliance level(s) for their languages.

Microsoft Intermediate Language (MSIL)

A .NET programming language (C#, VB.NET, J# etc.) does not compile into executable code; instead it compiles into an intermediate code called Microsoft Intermediate Language (MSIL). As a programmer one need not worry about the syntax of MSIL - since our source code is automatically converted to MSIL. The MSIL code is then sent to the CLR (Common Language Runtime) that converts the code to machine language which is then run on the host machine.

MSIL is similar to Java Byte code. A Java program is compiled into Java Byte code (the .class file) by a Java compiler, the class file is then sent to JVM which converts it into the host machine language.

- **Common Language Infrastructure**

It provides a virtual execution environment. It uses a compiler to process the language statement which is also known as source code into executable code. CLI programs can be written in various programming languages like C++, C#, ASP, etc. These all languages are compiled into CLI. CLR (by JIT) then compiles into machine code for further processing.

.NET also introduces Web Forms, Web Services and Windows Forms. The reason why they have been shown separately and not as a part of a particular language is that these technologies can be used by any .NET compliant language. For example Windows Forms is used by VC, VB.NET, C# all as a mode of providing GUI.

Garbage collection

NET's garbage collector manages the allocation and release of memory for your application. Each time you create a new object, the common language runtime allocates memory for the object from the managed heap. As long as address space is available in the managed heap, the runtime continues to allocate space for new objects. However, memory is not infinite. Eventually the garbage collector must perform a collection in order to free some memory. The garbage collector's optimizing engine determines the best time to perform a collection, based upon the allocations being made. When the **garbage collector performs a collection, it checks for objects in the managed heap that are no longer being used by the application and performs the necessary operations to reclaim their memory.**

Managed Code and Unmanaged code

The role of CLR doesn't end once we have compiled our code to MSIL and a JIT compiler has compiled this to native code. **Code written using the .NET framework, is managed code when it is executed under the control by CLR.** This means that the **CLR looks after our applications, by managing memory, handling security, allowing cross language debugging and so on.** By contrast, **applications that do not run under the control of the CLR are said to be unmanaged** and certain languages such as C++ can be used to write such applications, that for example, to access low level functions of the operating systems. However in C# we can only write code that runs in a managed environment.

Just In Time Compiler

- It is responsible for **converting the CIL (Common Intermediate Language) into machine code or native code** using the Common Language Runtime

Framework Base Classes

The next component of .NET is the **.NET Framework Base Classes**. These are the common class libraries (much like Java packages) that can be used by any .NET compliant language. These classes provide the programmers with a high degree of functionality that they can use in their programs. For example there are classes to handle reading, writing and manipulating XML documents, enhanced ADOs etc.

The .NET Framework has an extensive set of class libraries. This includes classes for:

- **Data Access:** High Performance data access classes for connecting to SQL Server or any other OLEDB provider.
- **XML Supports:** Next generation XML support that goes far beyond the functionality of MSXML.
- **Directory Services:** Support for accessing Active Directory/LDAP using ADSI.
- **Regular Expression :** Support for above and beyond that found in Perl 5.
- **Queuing Supports:** Provides a clean object-oriented set of classes for working with MSMQ.

These class libraries use the CLR base class libraries for common functionality.

Base Class Libraries

The Base class library in the .NET Framework is huge. It covers areas such as:

- **Collection :** The System.Collections namespace provides numerous collection classes.
- **Thread Support:** The System.Threading namespace provides support for creating fast, efficient, multi-threaded application.
- **Code Generation:** The System.CodeDOM namespace provides classes for generating source files in numerous language. ASP.NET uses these classes when converting ASP.NET pages into classes, which are subsequently compiled.
- **IO:** The System.IO provides extensive support for working with files and all other stream types.
- **Reflection:** The System.Reflection namespace provides support for loading assemblies, examining the type within assemblies, creating instances of types, etc.
- **Security:** The System.Security namespace provides support for services such as authentication, authorization, permission sets, policies, and cryptography. These base services are used by application development technologies like ASP.NET to build their security infrastructure.

Assemblies

What is an assembly?

- An Assembly is a logical unit of code
- Assembly physically exist as DLLs or EXEs
- One assembly can contain one or more files
- The constituent files can include any file types like image files, text files etc. alongwith DLLs or EXEs
- When you compile your source code by default the exe/dll generated is actually an assembly
- Unless your code is bundled as assembly it cannot be used in any other application
- When you talk about version of a component you are actually talking about version of the assembly to which the component belongs.
- Every assembly file contains information about itself. This information is called as **Assembly Manifest**.
-

What is assembly manifest?

- Assembly manifest is a data structure which stores information about an assembly
- This information is stored within the assembly file(DLL/EXE) itself.
- The information includes version information, list of constituent files etc.

What is private and shared assembly?

The assembly which is used only by a single application is called as private assembly. Suppose you created a DLL which encapsulates your business logic. This DLL will be used by your client application only and not by any other application. In order to run the application properly your DLL must reside in the same folder in which the client application is installed. Thus the assembly is private to your application.

Suppose that you are creating a general purpose DLL which provides functionality which will be used by variety of applications. Now, instead of each client application having its own copy of DLL you can place the DLL in 'global assembly cache'. Such assemblies are called as shared assemblies.

What is Global Assembly Cache?

Global assembly cache is nothing but a special disk folder where all the shared assemblies will be kept. It is located under <drive>:\WinNT\Assembly folder.

Managed Execution Process

The managed execution process includes the following steps;-

1. **Choosing a compiler.** To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime
2. **Compiling your code to MSIL.**

Compiling translates your source code into Microsoft intermediate language (MSIL) and generates the required metadata.

3. Compiling MSIL to native code.

At execution time, a just-in-time (JIT) compiler translates the MSIL into native code. During this compilation, code must pass a verification process that examines the MSIL and metadata to find out whether the code can be determined to be type safe.

4. Running code.

The common language runtime provides the infrastructure that enables execution to take place and services that can be used during execution.

Language Interoperability can be achieved in two ways:

- i. **Managed Code:** The MSIL code which is managed by the CLR is known as the Managed Code. For managed code CLR provides **three** .NET facilities:
 - **(Code Access Security)**
 - **Exception Handling**
 - **Automatic Memory Management.**
 -
- ii. **Unmanaged Code:** Before .NET development the programming language like .COM Components & Win32 API does not generate the MSIL code. So these are not managed by CLR rather managed by Operating System.

Programming Languages which are designed and developed by Microsoft are:

- C#.NET
- VB.NET
- C++.NET
- J#.NET
- F#.NET
- JSCRIPT.NET
- WINDOWS POWERSHELL
- IRON RUBY
- IRON PYTHON
- C OMEGA
- ASML(Abtract State Machine Language)

Unit-2

Programming with VB.net

Syllabus:-

- Data types with Programmes
- Variables, Constant ,Type Conversion
- Operators (all),Operator precedence Ex.
- Control Structure: Conditional statements (IF-Else, Select Case) with Programs
- Loops : Do Loops, For loops, While Loop, For eachNext Loop
- Various Programs based on Condition and Loops
- Array: Declaring arrays and Dynamic arrays
- Types, Structure, Enumeration
- Sub Procedure
- Functions, Passing argument types

Question Bank

1. Write control structure of VB.Net with example?
2. Differentiate between function and subroutine with example?
3. Explain primitive data types used in Vb.net with range and length?
4. Explain various conditional statements with example?
5. How to create enumeration in VB.Net? Explain with an example.
6. Describe Array and dynamic array and what are benefits of dynamic array over static array?
7. What is Procedure? Describe Types of procedure.
8. Explain function with example.
9. What is Control Loops and any 2 loop structure.
10. What is control structure explain any 3 structure.
11. Explain structure with example.
12. Explain the differences between a value type and a reference type with example.
13. Describe type conversion and its different types of types conversion function.
14. Write different types of operators with example.

❖ What are Data Types?

Data types determine the type of data that any variable can store. Variables belonging to different data types are allocated different amounts of space in the memory. There are various data types in VB.NET. They include:

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](#).

DataTypes	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 +/-7.92281625142264337593543950335 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.

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, Integer, 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
- 3.
4. Initialize the Emp_name
5. Dim Emp_name As String
6. Emp_name = "Vikas" 'Here Emp_name variable assigned a value of Vikas
- 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 = "Ram "

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

Variable1.vb

```
Imports System
Module Variable1
    Sub Main()
        'declaration of intData as Integer
        Dim intData As Integer
        'declaration of charData as Char
        Dim CharData As Char
        'declaration of strData as String
        Dim strData As String
        'declaration of dblData as Double
        Dim dblData As Double
        'declaration of single_data as Single
        Dim single_data As Single
        'Initialization of intData
        intData = 10
        'Initialization of CharData
        CharData = "A"
        'Initialization of strData
        strData = "VB.NET is a Programming Language."
        dblData = 4567.676
        'Initialization of dblData
        'Initialization of single_data
        single_data = 23.08
        Console.WriteLine(" Value of intData is: {0}", intData)
        Console.WriteLine(" Value of CharData is: {0}", CharData)
        Console.WriteLine(" Value of strData is: {0}", strData)
        Console.WriteLine(" Value of dblData is: {0}", dblData)
        Console.WriteLine(" Value of single_data is: {0}", single_data)

        Console.WriteLine("press any key to exit...")
        Console.ReadKey()
    End Sub

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:

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

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

User Data.vb

```
Imports System
Module User_Data
    Sub Main()
        Dim num As Integer
        Dim age As Double
        Dim name As String
        Console.WriteLine("Enter your favourite number")
        ' Console.ReadLine or Console.ReadLine() takes value from the user
        num = Console.ReadLine
        Console.WriteLine(" Enter Your Good name")
        'Read string data from the user
        name = Console.ReadLine
        Console.WriteLine(" Enter your Age")
        age = Console.ReadLine
        Console.WriteLine(" You have entered {0}", num)
        Console.WriteLine(" You have entered {0}", name)
        Console.WriteLine(" You have entered {0}", age)
        Console.ReadKey()

    End Sub
End Module
```

❖ 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

ItemName	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, Integer, 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:

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

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 = "Disha College"

Private Const PI As Double = 3.14

```
Module variablesNdatatypes
    Sub Main()
        Dim a As Short
        Dim b As Integer
        Dim c As Double

        a = 10
        b = 20
        c = a + b
        Console.WriteLine("a={0},b={1},c={2}", a, b, c)
        Console.ReadLine()
    End Sub
End Module
```

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

a = 10, b = 20, c = 30

Accepting Values from User

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


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

The following example demonstrates it –

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

When the above code is compiled and executed, it produces the following result (assume the user inputs Hello World) –

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

❖ Type Conversion Functions

There are functions that we can use to convert from one data type to another. They include:

- **CBool**(expression): converts the expression to a Boolean data type.
- **CDate**(expression): converts the expression to a Date data type.
- **CDBl**(expression): converts the expression to a Double data type.
- **CByte**(expression): converts the expression to a byte data type.
- **CChar**(expression): converts the expression to a Char data type.
- **CLng**(expression): converts the expression to a Long data type.
- **CDec**(expression): converts the expression to a Decimal data type.
- **CInt**(expression): converts the expression to an Integer data type.
- **CObj**(expression): converts the expression to an Object data type.
- **CStr**(expression): converts the expression to a String data type.
- **CSByte**(expression): converts the expression to a Byte data type.
- **CShort**(expression): converts the expression to a Short data type.

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.

1. 'Store only Integer values
2. Dim num As Integer() or Dim num() As Integer
3. 'Store only String values
4. Dim name As String() or Dim name(5) As String
5. ' Store only Double values
6. 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,

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

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

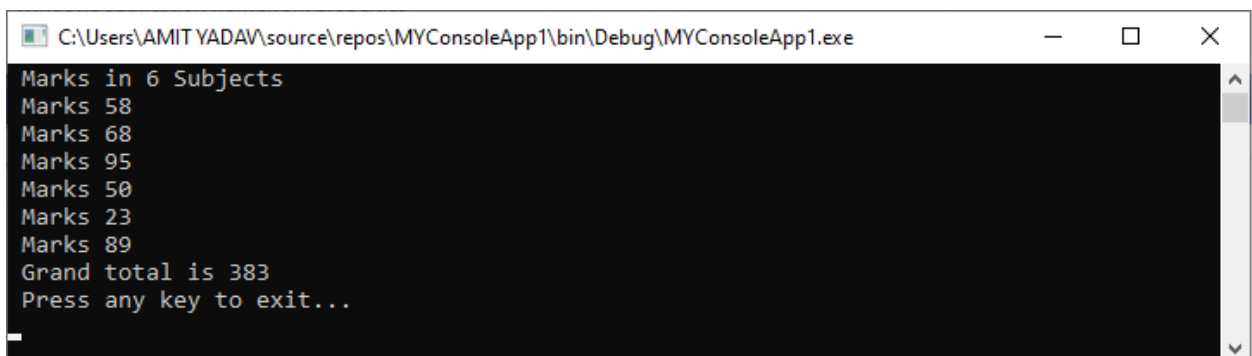
1. Dim intData() As Integer = {1, 2, 3, 4, 5}
2. Dim intData(5) As Integer
3. Dim array_name() As String = {"Peter", "John", "Brock", "James", "Maria"}
4. Dim misc() as Object = {"Hello friends", 16c, 12ui, "A"c}
5. Dim Emp(0 to 2) As String
6. Emp{0} = "Mathew"
7. Emp(1) = " Anthony"
8. Emp(2) = "Prince"

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. Dim marks() As Integer = {58, 68, 95, 50, 23, 89}
6. Console.WriteLine(" Marks in 6 Subjects")
- 7.
8. For i = 0 To marks.Length - 1
9. Console.WriteLine(" Marks {0}", marks(i))
10. Sum = Sum + marks(i)
11. Next
- 12.
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:



```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Marks in 6 Subjects
Marks 58
Marks 68
Marks 95
Marks 50
Marks 23
Marks 89
Grand total is 383
Press any key to exit...
```

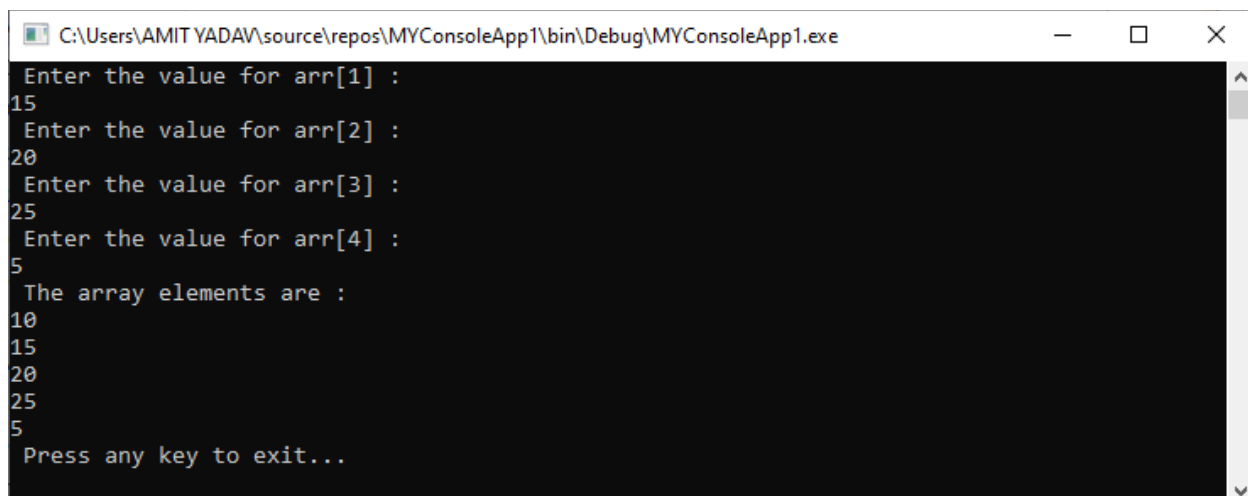
Input number in VB.NET Array

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

Input_array.vb

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

Output:



```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Enter the value for arr[1] :
15
Enter the value for arr[2] :
20
Enter the value for arr[3] :
25
Enter the value for arr[4] :
5
The array elements are :
10
15
20
25
5
Press any key to exit...
```

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"

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.

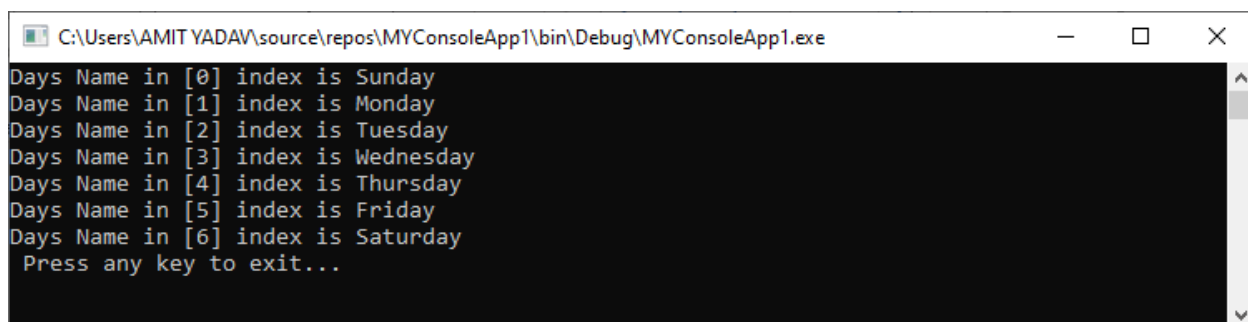
1. Dim myArr() As Variant
2. ReDim myArr(3)
3. myArr(0) = 10
4. myArr(0) = "String"
5. myArr(0) = false
6. myArr(0) = 4.6

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

Dynamic_Arr.vb

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

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.
12.        Console.WriteLine(" Before Preserving the Elements")
13.
14.        For i As Integer = 0 To Days.Length - 1
15.            Console.WriteLine("Days Name in [{0}] index is {1}", i, Days(i))
16.        Next
17.
18.
19.        Console.WriteLine(" After Preserving 0 to 2 index Elements")
20.        ReDim Preserve Days(6)
21.        Days(3) = "Wednesday"
22.        Days(4) = "Thursday"
23.        Days(5) = "Friday"
24.        Days(6) = "Saturday"
25.
26.        For i As Integer = 0 To Days.Length - 1
27.            Console.WriteLine("Days Name in [{0}] index is {1}", i, Days(i))
28.        Next
29.        Console.WriteLine(" Press any key to exit...")
30.        Console.ReadKey()
31.    End Sub
32. End Module
```

Output:

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Before Preserving the Elements
Days Name in [0] index is Sunday
Days Name in [1] index is Monday
Days Name in [2] index is Tuesday

After Preserving 0 to 2 index Elements
Days Name in [0] index is Sunday
Days Name in [1] index is Monday
Days Name in [2] index is Tuesday
Days Name in [3] index is Wednesday
Days Name in [4] index is Thursday
Days Name in [5] index is Friday
Days Name in [6] index is Saturday
Press any key to exit...
```

Program 1 :WAP to print marks and calculate grand total

Program 2 : WAP to print marks and grand total of a student which is given by user.

Structure

Structure is a user-defined datatype in computer language which allows us to combine data of different types together. Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.

For example: If I have to write a program to store Student information, which will have Student's name, age, branch, permanent address, father's name etc, which included string values, integer values etc, how can I use arrays for this problem, I will require something which can hold data of different types together.

In structure, data is stored in form of **records**.

in visual basic, **Structures** are same as classes but the only difference is classes are the reference types and structures are the value types. As a value type, the structures directly contain their value so their object or instance will be stored on the stack and structures are faster than classes.

In visual basic, the structures can contain fields, properties, member functions, operators, constructors, events, indexers, constants and even other structure types.

Create Structures in Visual Basic

In visual basic, structures are declared by using Structure keyword. Following is the declaration of structure in a visual basic programming language.

```
Public Structure Users
    ' Properties, Methods, Events, etc.
End Structure
```

If you observe the above syntax, we defined a structure “**users**” using Structure keyword with **Public** access modifier. Here, the **Public** access specifier will allow the users to create an object for this structure and inside of the body structure, we can create required fields, properties, methods and events to use it in our applications.

Following is the example of defining the structure in visual basic programming language.

```
Public Structure User
    Public name As String
    Public location As String
    Public age As Integer
End Structure
```

If you observe the above example, we defined a structure called “**User**” with the required fields and we can add required methods and properties based on our requirements.

Visual Basic Structure Initialization

In visual basic, structures can be instantiated with or without **New** keyword. Following is the example of assigning a values to the variables of structure.

```
Dim u As User = New User()
u.name = "Suresh Dasari"
u.location = "Hyderabad"
u.age = 32
```

To make use of **fields**, **methods** and events of structure, it’s mandatory to instantiate the structure with **New** keyword in a visual basic programming language.

Visual Basic Structure with Constructor

In visual basic, the structures won’t allow us to declare **default constructor** or **constructor** without parameters and it won’t allow us to initialize the **fields** with values unless they are declared as **const** or shared.

Following is the example of defining the structure with **parameterized constructor** and initializing the fields in a **constructor** in visual basic programming language.

```
Public Structure User
    Public name, location As String
    ' Parameterized Constructor
    Public Sub New(ByVal a As String, ByVal b As String)
        name = a
        location = b
    End Sub
End Structure
```

If you observe the above example, we defined the structure called “**User**” with required fields and **parameterized constructor**.

Visual Basic Structure with Default Constructor

As discussed, the structures will allow only [parameterized constructors](#) and fields cannot be initialized unless they are declared it as [const](#) or shared.

Following is the example of defining the structure with a [default constructor](#) and initializing fields with values.

```
Structure User
' Compiler Error
Public name As String = "Suresh Dasari"
Public location As String
Public age As Integer
' Compiler Error
Public Sub New()
    location = "Hyderabad"
    age = 32
End Sub
End Structure
```

When we execute the above code, we will get a compile-time error because we declared a structure with the [default constructor](#) (parameterless) and initialized fields without defining it as [const](#) or shared.

Now, we will see how to create a structure with fields and [parameterized constructor](#) in a visual basic programming language with example.

Visual Basic Structure Example

Following is the example of creating the structure with a different type of fields and [parameterized constructor](#) in a visual basic programming language with various data members and member functions.

```
Module Module1
    Structure User
        Public Const name As String = "Suresh Dasari"
        Public location As String
        Public age As Integer
        Public Sub New(ByVal a As String, ByVal b As Integer)
            location = a
            age = b
        End Sub
    End Structure
    Sub Main()
        ' Declare object with new keyword
        Dim u As User = New User("Hyderabad", 31)
        ' Declare object without new keyword
        Dim u1 As User
        Console.WriteLine("Name: {0}, Location: {1}, Age: {2}", User.name, u.location, u.age)
        ' Initialize Fields
    End Sub
End Module
```

```

    u1.location = "Guntur"
    u1.age = 32
    Console.WriteLine("Name: {0}, Location: {1}, Age: {2}", User.name, u1.location, u1.age)
    Console.WriteLine("Press Enter Key to Exit..")
    Console.ReadLine()
End Sub
End Module

```

If you observe the above example, we defined a structure (**User**) by including the required **fields**, **parameterized constructor** and created an instance of structure (**User**) with and without **New** keyword to initialize or get the field values based on our requirements.

VB.NET Enum

In VB.NET, **Enum** is a keyword known as Enumeration.

Enumeration is a user-defined data type used to define a related set of constants as a list using the keyword **enum** statement. It can be used with module, structure, class, and procedure. For example, month names can be grouped using Enumeration.

Syntax:

1. Enum enumeration name [As Data type]
2. ' enumeration data_list or Enum member list
3. End Enum

In the above syntax, the **enumeration name** is a valid identifier name, and the data type can be mentioned if necessary, the enumeration **data_list** or **Enum member list** represents the set of the related constant member of the Enumeration.

Declaration of Enumerated Data

Following is the declaration of enumerated data using the Enum keyword in [VB.NET Programming language](#):

1. Enum Fruits
2. Banana
3. Mango
4. Orange
5. Pineapple
6. Apple
7. End Enum

Here, **Fruits** is the name of enumerated data that contains a list of fruits called **enumeration** lists.

When we are creating an Enumerator constant data list, by default, it assigns a value 0 to the first index of the enumerator. Each successive item in the enumerator list is increased by 1. For example, in the above enumerator list, Banana value is 0, Mango value is 1, Orange value is 2, and so on.

Moreover, if we want to assign a new value to the default value of the enumerator items, set a new value to the list's first index, and then enumerator's values can automatically provide a new incremental value for successor objects in the enumerator list.

Override a default index value by setting a new value for the first element of the enumerator list.

1. Enum Fruits
2. Banana = 5
3. Mango
4. Orange
5. Pineapple
6. Apple
7. End Enum

Here, we have assigned 5 as the new index value of the first item in an enumerator list. So, the initial value of the list's Banana=5, Mango=6, orange=7, and so on. Furthermore, we can also provide new value in the middle of the enumerator list, and then it follows the sequence set by the previous item.

Example 1: Write a program to understand the uses of enumeration data in [VB.NET](#).

Enum_Data.vb

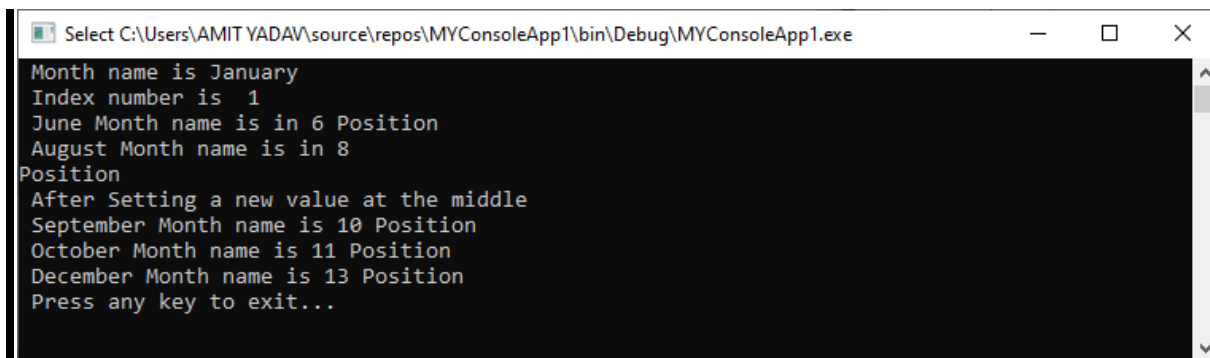
1. Imports System
2. Module Enum_Data
3. Enum Number
4. ' List of enumerated data
5. One
6. Two
7. Three
8. Four
9. Five
10. Six
11. End Enum
12. Sub Main()
13. Console.WriteLine(" Without the index number")
14. Console.WriteLine(" Number {0}", Number.One) 'call with enumeration and items name
15. Console.WriteLine(" Number {0}", Number.Two)
16. Console.WriteLine(" Number {0}", Number.Three)
17. Console.WriteLine(" Number {0}", Number.Four)
18. Console.WriteLine(" Number {0}", Number.Five)
19. Console.WriteLine(" Number {0}", Number.Six)
20. Console.WriteLine(" Press any key to exit...")
21. Console.ReadKey()
22. End Sub
23. End Module

Output:

In the above example, an enumerator "**Number**" that gets each value of the enumerated data such as "**Number.One**", etc. to print the list.

Enum_month.vb

```
1. Imports System
2. Module Enum_month
3.     Enum Monthname 'Enumeration name
4.         January = 1
5.         February
6.         march
7.         April
8.         May
9.         June
10.        July
11.        August
12.        September = 10 'Set value to 10
13.        October
14.        November
15.        December
16.    End Enum
17.    Sub Main()
18.        Dim x As Integer = CInt(Monthname.June)
19.        Dim y As Integer = CInt(Monthname.August)
20.        Dim p As Integer = CInt(Monthname.September)
21.        Dim q As Integer = CInt(Monthname.October)
22.        Dim r As Integer = CInt(Monthname.December)
23.
24.        Console.WriteLine(" Month name is {0}", Monthname.January)
25.        Console.WriteLine(" Index number is " & Monthname.January)
26.        Console.WriteLine(" June Month name is in {0}", x & " Position")
27.        Console.WriteLine(" August Month name is in {0}", y & vbCrLf & "Position")
28.
29.        Console.WriteLine(" After Setting a new value at the middle")
30.        Console.WriteLine(" September Month name is {0}", p & " Position")
31.        Console.WriteLine(" October Month name is {0}", q & " Position")
32.        Console.WriteLine(" December Month name is {0}", r & " Position")
33.        Console.WriteLine(" Press any key to exit...")
34.        Console.ReadKey()
35.
36.    End Sub
37. End Module
```



```
Select C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Month name is January
Index number is 1
June Month name is in 6 Position
August Month name is in 8 Position
After Setting a new value at the middle
September Month name is 10 Position
October Month name is 11 Position
December Month name is 13 Position
Press any key to exit...
```

Procedure

A procedure is a group of statements that together perform a task when called. After the procedure is executed, the control returns to the statement calling the procedure. VB.Net has two types of procedures –

- Functions
- Sub procedures or Subs

Functions return a value, whereas Subs do not return a value.

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.

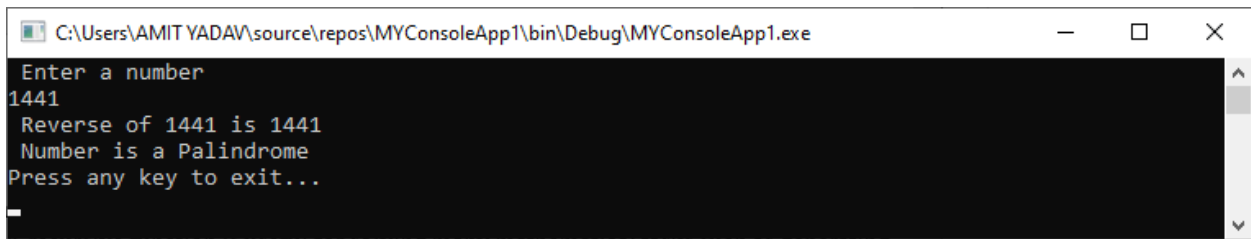
```
Public Function GetData( ByVal username As String, ByVal userId As Integer) As String
    ' Statement to be executed
End Function
```

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

Palindrome.vb

```
Imports System
Module Palindrome
    ' Define a reverse() function
    Function reverse(ByVal num As Integer) As Integer
        ' Define the local variable
        Dim remain As Integer
        Dim rev As Integer = 0
        While (num > 0)
            remain = num Mod 10
            rev = rev * 10 + remain
            num = num / 10
        End While
        Return rev
    End Function
    Sub Main()
        ' Define the local variable as integer
        Dim n, num2 As Integer
        Console.WriteLine(" Enter a number")
        n = Console.ReadLine() 'Accept the number
        num2 = reverse(n) ' call a function
        Console.WriteLine(" Reverse of {0} is {1}", n, num2)

        If (n = reverse(n)) Then
            Console.WriteLine(" Number is a Palindrome")
        Else
            Console.WriteLine(" Number is not a Palindrome")
        End If
        Console.WriteLine("Press any key to exit...")
        Console.ReadKey()
    End Sub
End Module
```



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.
- Public Function GetData1(ByRef username As String, ByRef userId As Integer)
- ' Statement to be executed
- End Sub

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

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

