Share on your Social Media

Dot Net Tutorial

Published On: September 10, 2024

Dot Net Tutorial

Dot Net is an open-source framework that allows developers to create native desktop, web, and mobile apps that work with any operating system. Explore our Dot Net tutorial, which covers various tools, libraries, and programming languages for developing scalable applications.

Download DOTNET Tutorial PDF

Introduction to Dot Net

Dot Net is a popular and frequently used framework, especially when developing web services and cross-platform applications. In this Dot Net tutorial, we cover the following:

- Overview of Dot Net
- Dot Net Architecture
- Garbage Collection in Dot Net
- Code Execution in Dot Net
- Dot Net Core: Standard Library
- Dot Net Core: Portable Class Library
- C# Fundamentals
- Design Patterns in C#

Overview of Dot Net

Dot Net is used in the development of web services, web-based applications, and form-based applications. Applications for Windows, smartphones, the web, etc. are developed with it. In addition to offering many features, it complies with industry requirements.

The evolution of .NET technology can be divided into three key stages.

- **OLE Technology:** One of Microsoft's component document technologies is OLE (Object Linking and Embedding). In essence, connecting components from various applications is its primary goal.
- **COM Technology:** The Microsoft Windows operating system series includes Microsoft COM (Common Object Model) technology, which enables communication between various software components. There are numerous programming languages available for use in creating COM objects.
- **.NET Technology:** The.NET technology is a group of technologies used in the development of Windows and web applications. NGWS (Next Generation Web Services) was its initial name. Among the most potent, well-liked, and practical Internet technologies on the market right now is this one.

More than 60 programming languages are supported by the.NET Framework, of which 11 are created and maintained by Microsoft. Although not created and maintained by Microsoft, the remaining non-Microsoft languages are supported by the.NET Framework.

Microsoft has designed and developed 11 programming languages. These are:

- C#.NET
- VB.NET
- C++.NET
- J#.NET
- F#.NET
- JSCRIPT.NET

- Windows PowerShell
- IRON RUBY
- IRON PYTHON
- C OMEGA
- ASML (Abstract State Machine Language)

Computing Aspirants VMware software allows you to run a virtual machine...

DOTNET Interview Questions and Answers

Key Features of Dot Net

Many features are available on Dot Net, some of which are listed below.

Easy: In the sense that it offers an organized method and a wide range of library functions, data types, etc., **C#** is a straightforward language.

Rich Library: C# has a large number of built-in functions that speed up development.

High Speed: The compilation and execution times of the C# language are quite short.

Updateable and Scalable: The programming language C# is automatically scalable and updateable. We replace the outdated files with new ones when we update our application.

Modern Programming Language: Based on the current trend, .Net is incredibly strong and easy to use for creating robust, scalable, and interoperable systems.

Structured Programming Language: Because functions allow us to divide a program into manageable chunks, C# is a structured programming language.

Object-Oriented: The language for object-oriented programming is C#. OOPs simplify development and maintenance, but procedure-oriented programming languages can be challenging to manage when code develops in proportion to the size of the project.

Component-oriented: It is the most widely used software development process for creating programs that are more reliable and easily scalable.

Type Safe: Code that is safe in C# can only access memory locations that are authorized for its execution. As a result, the program's security is enhanced.

Interoperability: The C# programs can perform nearly all of the functions of a native **C++ application** due to the interoperability procedure.

Dot Net Architecture

A software architecture called Net Architecture is used to create apps that function on Microsoft's .NET platform. By offering a standardized set of APIs and a consistent programming architecture, it offers a collection of libraries and tools that make the building of complicated applications easier.

Major Components of Dot Net

The .NET architecture consists of four fundamental components:

Common Language Runtime (CLR): The.NET programming language's execution engine is the Common Language Runtime (CLR). Along with security features, it offers memory management and thread management. It also provides a standard-type system that enables seamless interoperability between objects written in different programming languages.

Base Class Library (BCL): A collection of reusable classes, interfaces, and value types that provide the fundamental functionality of .NET programs is known as the Base Class Library (BCL). Included are classes covering networking, input/output, threading, collections, and other related subjects.

JIT (Just-In-Time) Compiler: The Intermediate Language (IL) code from .NET applications is translated into native machine code by this compiler so that the CPU can execute it. It improves performance by compiling code when needed.

Visual Studio: Known for its suite of tools for developing .NET applications, **Visual Studio** is one of the most popular Integrated Development Environments (IDEs). Along with other capabilities, it offers a code editor, debugger, and project management tools.

DOTNET Syllabus PDF

C# Fundamentals

The most potent programming language offered by the.NET Framework is C#.NET. Along with a few extra features, it has all the features of Java, C++, and VB.NET. The C#.NET programming language is intended to be object-oriented, modern, easy to use, and general-purpose. Pronounce C# like C-Sharp.

Types of Applications Developed using C#:

We can develop a variety of safe and reliable applications using the C# programming language:

- Window Applications
- Web Applications
- Web Service Applications
- Mobile Applications
- Database Applications
- IoT Applications
- Game Applications

Sample C# Program

using System;

namespace namespace_name

```
{
class Class_Name
{
int n1;
float n2;
static void block()
{
console.WriteLine(“Hello World”);
}
}
}
```

Data Types of C#

A variable’s data type, integer, floating-point, character, etc., indicates the kind of data it may hold.

- Value Data Type: short, int, char, float, double, etc.
- Reference Data Type: string, class, object, and interface.
- Pointer Data Type: Pointers

Value Data Type

Both floating-point and integer-based value data formats are available. Both signed and unsigned literals are supported in C#.

The C# language has two different value data types.

- **Predefined data types** are float, boolean, and integer.
- **User-defined data types** include enumerations, structures, and more.

Data types' memory sizes can vary depending on whether they run on a 32- or 64-bit operating system.

Reference Data Type

The reference data types include a reference to the variables rather than the actual data that is stored in a variable.

When one of the variables modifies the data, the other variable immediately updates to reflect the new value.

In the C# language, reference data types come in two varieties.

- **Predefined types,** such as strings and objects.
- **User-defined Types:** Classes, Interfaces, etc.

Pointer Data Type

In the C# programming language, a pointer is a variable that points to a value's address. It is sometimes referred to as a locator or indicator.

- **& (ampersand sign):** Address operator: Ascertain a variable's address.
- *** (asterisk sign):** Indirection operator: Retrieve an address's value.

Declaring a Pointer

In the C# language, the pointer can be specified using the * (asterisk) symbol.

*int * a;*

*char * c;*

Operators in C#

A symbol that is utilized to carry out operations is called an operator. Numerous operations are possible, including bitwise, arithmetic, and logical ones.

The C# language has the following sorts of operators to carry out various operations.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Unary Operators
- Ternary Operators
- Misc Operators

Precedence of Operators in C#

The operator that will be evaluated first and next is indicated by its precedence. The operator direction to be evaluated, left to right or right to left, is specified by the associativity.

Let's use the following example to better grasp precedence:

*int data= 10+ 5*5*

Because the multiplication operator * is evaluated before the additive operator +, the "data" variable will contain 35.

The operators in C# are listed in the following order and associativity:

Category	**Operators**
Unary	+, −, ~, ++, −, &, (type)*
Additive	+, −
Multiplicative	%, /, *
Relational	<, >, <=, >=

Shift	<<, >>
Equality	==, !=
Logical AND	&
Logical OR	|
Logical XOR	^
Conditional OR	||
Conditional AND	&&
Null	??
Ternary	>:
Assignment	= *= /= %= += − = <<= >>= &= ^= |= =>

DOTNET Developer Salary

Keywords in C#

A reserved term is a keyword. It cannot be used as a constant name, variable name, etc.

Keywords cannot serve as identifiers in C#. Nonetheless, we can prefix the keywords with the @ character if we wish to utilize them as identifiers. Some of the popular keywords in C#:

abstract	base	as	bool
break	catch	case	byte
char	checked	class	const
continue	decimal	private	protected
public	return	explicit	extern
finally	float	fixed	false

Control Statements in C#

The if statement is used in C# programming to test the condition. In C#, there are several kinds of if statements.

- if statement
- if-else statement
- nested if statement
- if-else-if ladder

If Statement

The condition is tested by the C# if statement. If the condition is true, it is carried out.

> **Syntax**

```
if(condition){

//code to be executed

}
```

Example:

using System;

```
public class IfExample

   {

      public static void Main(string[] args)

        {

           int num = 10;

           if (num % 2 == 0)

           {

              Console.WriteLine(“It is even
number”);

           }

        }

    }
```

Output

It is even number

If Else Statement

The if-else statement likewise tests the condition in C#. If the condition is true, the if block is executed; if not, the else block is run.

Syntax

```
if(condition){

//code if condition is true

}else{

//code if condition is false

}
```

Example

using System;

```
public class IfExample

    {

       public static void Main(string[]
args)

       {
            int num = 11;

            if (num % 2 == 0)

            {

                Console.WriteLine("It is even number");

            }

            else

            {

                Console.WriteLine("It is odd number");

            }

        }

    }
```

Output

It is odd number

IF-else-if ladder statement

One condition from several lines is carried out by the C# if-else-if ladder statement.

Syntax

```
if(condition1){

//code to be executed if condition1 is true
}else if(condition2){

//code to be executed if condition2 is
true

}

else if(condition3){

//code to be executed if condition3 is
true

}

…

else{

//code to be executed if all the conditions
are false

}
```

Example

```
using System;

public class IfExample

   {

       public static void Main(string[]
args)

       {

           Console.WriteLine("Enter a
number to check grade:”);

           int num =
Convert.ToInt32(Console.ReadLine());

           if (num <0 || num >100)

           {

              Console.WriteLine(“wrong
number”);

           }

           else if(num >= 0 && num < 50){

              Console.WriteLine(“Fail”);

           }

           else if (num >= 50 && num < 60)

           {

              Console.WriteLine(“D Grade”);

           }

           else if (num >= 60 && num < 70)

           {

              Console.WriteLine(“C Grade”);

           }

           else if (num >= 70 && num < 80)
           {
              Console.WriteLine("B Grade");
           }
           else if (num >= 80 && num < 90)
           {
              Console.WriteLine("A Grade");
           }
           else if (num >= 90 && num <=
100)
           {
              Console.WriteLine("A+
Grade");
           }
         }
       }
```

Example

Enter a number to check grade: 66

C Grade

DOTNET Training

Garbage Collections in Dot Net

The allocated objects are stored on separate heaps for each of the three generations of the.NET Garbage Collector.

Most objects have one of two basic lifespans: short life or lengthy life.

Generation First (0)

First, objects are assigned in Generation 0.

- Since they are no longer in use (out of scope) by the time the next garbage collection takes place, items in this generation frequently don't survive past the first generation.
- Because Generation 0's related heap is small, it collects quickly.

Generation Second (1)

- Objects have a second chance space in Generation 1.
- Short-lived items move to Generation 1 if they make it through the Generation 0 collection (often due to coincidence in timing).
- Because the related heap of Generation 1 is likewise small, Generation 1 collections are also fast.
- Since objects are either promoted to the next generation heap or collected, the first two heaps stay tiny.

Generation Third (2)

- All lengthy objects in Generation 2 have a life cycle, and their heap has the potential to expand greatly.
- There is no next-generation heap to promote objects further, and the items in this generation can last for a very long time.
- The Large Object Heap (LOH) is an extra heap the Garbage Collector maintains for huge items.
- It is only applicable to items larger than 85,000 bytes.
- Large objects are allocated straight to the LOH rather than to the generational heaps.

- Programs that have been running for a long time or that operate over vast volumes of data may find that Generation 2 and LOH collections take a noteworthy amount of time.
- In Generation 2, every long object has a life cycle, and its heap might grow very large.
-
- The things in this generation can last for a very long time, and there is no next-generation heap to promote objects further.
- The Large Object Heap (LOH) is an additional heap kept up to date for large things by the Garbage Collector.
- Only objects greater than 85,000 bytes are eligible for it.
- Rather than being placed in the generational heaps, large objects are directly assigned to the LOH.
- Generation 2 and LOH collections may take a significant amount of time for programs that operate over large amounts of data or that have been running for a long time.

Code Execution in Dot Net

Here the .NET Core execution process provides a comparison with the .NET Framework. The subsequent phases are part of the controlled execution process.

- Selecting a compiler
- Assembling your code for the MSIL
- MSIL compilation to native code
- Executing code

Selecting a Compiler

- There are many different data kinds and linguistic aspects supported by the runtime, which is a multilingual execution environment.
- You must utilize one or more language compilers that target the runtime to profit from the common language runtime.

Assembling your code for the MSIL

- Compiling creates the necessary metadata and converts your source code into Microsoft Intermediate Language (MSIL).
- The types in your code are described by metadata, which also includes other information that the runtime utilizes during execution, each type's description, member signatures, and members that your code references.
- When necessary for execution, the runtime finds and retrieves the metadata from the file and framework class libraries (FCL).

MSIL to Native Code Compilation

- A just-in-time (JIT) compiler converts the MSIL into native code during runtime.
- Code must complete this compilation process by passing through a verification procedure that looks through the MSIL and metadata to see if the code may be deemed type-safe.

Executing Code

- The infrastructure needed for execution to occur as well as services that can be utilized while execution is underway are provided by the common language runtime.
- Services like garbage collection, security, cross-language debugging support, interoperability with unmanaged code, and improved deployment and versioning support are provided to managed code while it is being executed.

Dot Net Core: Standard Library

The types and methods that can be invoked from any application are defined by a class library.

- Any .NET platform that supports that version of the .NET Standard Library can call a class library created with .NET Core since it is compatible with the .NET Standard Library.
- Once your class library is complete, you can

choose to include it as a component that comes with one or more apps, or you can publish it as a third-party component.

Example

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace UtilityLibrary {

  public static class StringLib {

    public static bool
StartsWithUpper(this String str) {

      if (String.IsNullOrWhiteSpace(str))

      return false;

      Char ch = str[0];

      return Char.IsUpper(ch);

    }

    public static bool
StartsWithLower(this String str) {

      if (String.IsNullOrWhiteSpace(str))

      return false;

      Char ch = str[0];
      return Char.IsLower(ch);

    }

    public static bool
StartsWithNumber(this String str) {

      if (String.IsNullOrWhiteSpace(str))

      return false;

      Char ch = str[0];

      return Char.IsNumber(ch);

    }

  }

}
```

Dot Net Core: Portable Class Library

Using the Portable Class Library project, you can create managed assemblies that are compatible with multiple .NET Framework platforms.

- You can construct classes that hold code that you want to reuse in several projects, like shared business logic, and then use those classes as references in other kinds of projects.
- Additionally, it can make cross-platform applications and libraries for Microsoft platforms simple and quick to develop.
- You can cut down on the time and expenses associated with writing and **testing code** using portable class libraries.
- Write and build portable.NET Framework assemblies with this project type, then use those assemblies as references in programs

that run on Windows, Windows Phone, and other platforms.

Example

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace StringLibrary {

  public static class StringLib {

    public static bool
StartsWithUpper(this String str) {

      if (String.IsNullOrWhiteSpace(str))

        return false;

      Char ch = str[0];

      return Char.IsUpper(ch);

    }

    public static bool
StartsWithLower(this String str) {

      if (String.IsNullOrWhiteSpace(str))

        return false;
```

```
            Char ch = str[0];

            return Char.IsLower(ch);

        }

        public static bool
    StartsWithNumber(this String str) {

            if (String.IsNullOrWhiteSpace(str))

                return false;

            Char ch = str[0];

            return Char.IsNumber(ch);

        }

    }

}
```

DOTNET Project Ideas

Design Patterns in C#

When it comes to finding gaps and possibilities to address this kind of problem, design patterns are crucial. Design patterns serve as general solution suppliers or as adaptable solution builders.

Within the field of object-oriented programming, it is a source of reusable solutions that are typically encountered regularly during the **development of applications.** A design pattern offers a model or descriptive method for resolving issues that crop up in various contexts.

Importance of Design Patterns

The goal of design patterns is to solve these problems by identifying the complexity, agility, and simplicity in generating an optimal solution that can solve millions of problems with a single solution. More specifically, it’s a pattern language for all these kinds of issues.

Creational Design Patterns

Creational Design Patterns provide a powerful and dynamic technique that improves flexibility and harnesses the closeness of code reusability to solve problems in a broader context. The easier it is to locate and resolve, the larger the arena.

- **Factory Method:** The primary purpose of the factory method is to supply creational objects to the subclass; these subclasses are permitted to modify the type of objects that are created.
- **Abstract Factory:** The abstract factory design allows us to create families of linked objects without having to identify the classes to which they belong. They stay completely concealed.
- **Builder Pattern:** By using the builder pattern, we may create families of linked objects without having to identify the classes to which they belong. They stay completely concealed.
- **Prototype Design Pattern:** We won’t have to rely solely on the defined classes while implementing the existing objects thanks to the prototype design pattern.
- **Singleton:** This pattern guarantees that a class has just one instance, provided that the instance has a global access point.

Structural Design Patterns

With the help of structural design patterns, developers may build objects and classes with greater flexibility, enabling them to handle more complex issues as they arise.

This design pattern makes sure that all the classes and objects come together to create a substantial structure while maintaining optimal flexibility and efficiency.

It is divided into many subcategories, as indicated below.

- **Adapter:** This makes it possible to work together with any interface that is determined to be incompatible with the solution-based methodology.
- **Bridge:** Based on hierarchies, it enables the division of a sizable class or collection into multiple smaller pieces or closely related classes. These little classes are handled logically and independently in what is known as an abstract implementation.
- **Composite:** By breaking the items up into individual objects and treating each node separately to power a mechanism based on a solution, the composite structure enables us to work with the objects in a way that resembles a tree structure.
- **Decorator:** Attaching different additional behaviors to the objects is made possible by the decorator. It accomplishes this by encasing these things in unique wrapper objects and then exposing their behavior.
- **Facade:** Regardless of the framework we use, the facade manages to portray efficient solutions with interface enhancements while providing a straightforward user interface and exposure to the library or framework.
- **Flyweight:** We can fit more objects into the RAM through the flyweight. Instead of having to save all the data for each item, this is accomplished by sharing a common portion of the state that is present across several objects.
- **Proxy:** It offers an alternative method or an equivalent for anything else. It provides a stand-in for the proxy that manages the original objects' access. As a result, the requests are handled either before or after the

initial object is approved.

Behavioral Design Pattern

Behavioral design patterns are limited to algorithmic or strategic methods only. This design pattern primarily addresses how to divide or assign duties among several objects.

There are several subcategories into which this pattern might be divided.

- **Chain of Responsibility:** It forwards the requests to the handlers in a chain. Regardless of the circumstance, these handlers determine whether to process the request that is now being received or forward it to another handler in the chain.
- **Command:** It transforms the requests into independent objects that hold all the information about the request under consideration. This request is later supplied as a method argument to support and process queued or undoable operations.
- **Iterator:** This will enable you to revisit the set or catalog of troublesome patterns repeatedly without disclosing the underlying representation.
- **Mediator:** By preventing the items from communicating directly with one another and forcing them to cooperate through a mediator object, it only lessens the dependencies that are producing turmoil.
- **Memento:** All it does is conceal the object's implementation details while preserving operations like saving and reverting to the initial state.
- **Observer:** Defining a subscription system that eventually alerts the objects to potential events while they are out on observation is helpful.
- **State:** Defining a subscription system that eventually alerts the objects to potential events while they are out on observation is helpful.
- **Strategy:** It is beneficial to design a

subscription system that will eventually notify the objects of possible events while they are being observed.
- **Template Method:** It allows the subclass to override certain established stages of an algorithm without requiring changes to the superclass, which goes against the basic structure of the suggested algorithm.
- **Visitor:** It does nothing more than divide the objects that use the designated algorithm.

To solve challenges that demand maximum efficiency and advancement at every stage, design patterns are essential. Not only may they address problems about a certain level of the problem that arises periodically, but they also assist developers in approaching problems by thinking like a computer.

Conclusion

We have covered the basics of Dot Net with C# fundamentals in this Dot Net tutorial. Learn comprehensively in our **Dot Net training in Chennai.**

No.10, PT Rajan Salai, K.K. Nagar, Chennai – 600 078.
Landmark: Karnataka Bank Building
Phone: +91 86818 84318
Email: enquiry@softlogicsys.in
Map: Google Maps Link

OMR

No. E1-A10, RTS Food Street
92, Rajiv Gandhi Salai (OMR),
Navalur, Chennai - 600 130.
Landmark: Adj. to AGS Cinemas
Phone: +91 89256 88858
Email: info@softlogicsys.in
Map: Google Maps Link

SLA's Recently Placed Students

Reviews

Sitemap

Important Links

Disclaimer

Privacy Policy

Terms and Conditions

Courses

Python

Software Testing

Full Stack Developer

Java

Power BI

Clinical SAS

Data Science

Embedded

Cloud Computing

Hardware and Networking

VBA Macros

Mobile App Development

DevOps

Social Media Links

Review Sources

Google

Trustpilot

Glassdoor

Mouthshut

Sulekha

Justdial

Ambitionbox

Indeed

Software Suggest

Sitejabber