

Introduction to .NET

What is .NET?

.NET is a **developer platform** created by **Microsoft** used to build different types of applications using multiple programming languages.

Applications built using .NET:

- Windows Desktop Applications (Console, WinForms, WPF)
- Web Applications (ASP.NET)
- Web APIs
- Cloud Applications

In simple words:

.NET gives us ready-made tools, libraries, and runtime so we can focus on logic instead of low-level details.

Why do we use .NET?

Problems without .NET:

- Writing large code from scratch
- Manual memory handling
- Platform dependency

How .NET helps:

- Automatic memory management
- Huge class library
- Secure execution
- Object Oriented
- Industry accepted

Real-life example:

Instead of building a house brick-by-brick, .NET gives you ready-made blocks.

NET Framework vs .NET Core vs .NET (Modern)

Feature	.NET Framework	.NET Core	.NET
Platform	Windows only	Cross-platform	Cross-platform
Performance	Medium	Fast	Faster
Cloud	Poor	Excellent	Excellent
Status	Old	Deprecated	Current

Note:

Today we use .NET 6 / .NET 7 / .NET 8.

Software Installation Process

Step-by-step Installation

Download **Visual Studio 2022**

Select Workloads:

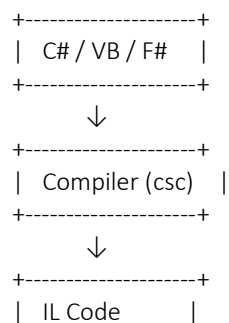
- .NET Desktop Development
- ASP.NET & Web Development

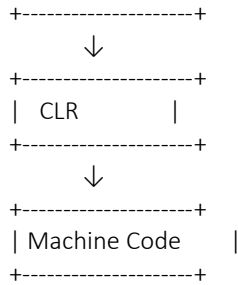
Install → Finish

Ready to code!

NET Architecture (With Diagram ☆☆☆)

Overall Architecture





CLR (Common Language Runtime)

CLR is the **execution engine** of .NET.

Functions of CLR:

- Memory Management
- Garbage Collection
- Security
- Code Execution
- Thread Handling

Example:

You don't free memory manually like C/C++ – CLR handles it.

IL (Intermediate Language)

- Language-independent code
- Generated after compilation

Why IL is important?

- Same IL works for all .NET languages
- Platform independence

Diagram:

C# → IL ← VB.NET

JIT Compiler (Just-In-Time)

- Converts IL to machine code
- Executes method-by-method

Advantages:

- Faster execution
 - Platform optimization
-

Garbage Collection (GC)

GC automatically removes unused objects.

Benefits:

- No memory leak
- Better performance

Example:

Unused objects are cleaned without developer effort.

C# Programming Basics

Structure of C# Program

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello .NET");
    }
}
```

Explanation:

- using → Import namespace
 - class → Blueprint
 - Main() → Entry point
-

Data Types in C#

Value Types

- int, float, double, bool, char

Reference Types

- string, array, class, object

```
int a = 10;  
string name = "Student";
```

Control Statements

Loop Example

```
for(int i = 1; i <= 5; i++)  
{  
    Console.WriteLine(i);  
}
```

Methods & Parameters

```
static int Add(int a, int b)  
{  
    return a + b;  
}
```

Object-Oriented Programming (OOP)

1. Class & Object

```
class Student  
{  
    public string Name;  
}
```

```
Student s = new Student();  
s.Name = "Amit";
```

2. Encapsulation

```
class Bank  
{  
    private int balance = 5000;  
    public int GetBalance() => balance;  
}
```

3. Inheritance

```
class Person
{
    public void Speak()
    {
        Console.WriteLine("Speaking");
    }
}

class Teacher : Person { }
```

4. Polymorphism

```
class A
{
    public virtual void Show() { }
}

class B : A
{
    public override void Show()
    {
        Console.WriteLine("Override");
    }
}
```

5. Abstraction

```
abstract class Shape
{
    public abstract void Draw();
}
```

Exception Handling

```
try
{
    int a = 10 / 0;
}
catch(Exception e)
{
    Console.WriteLine(e.Message);
}
```

Collections in C#

- List
- Dictionary
- Stack
- Queue

```
List<int> numbers = new List<int>() {1,2,3};
```

Final Summary

- ✓ ☐ .NET is powerful and secure
 - ✓ ☐ CLR, IL, JIT form execution backbone
 - ✓ ☐ C# supports full OOP
 - ✓ ☐ Industry-ready framework
-