.NET & C# Fundamentals

Overview of .NET Ecosystem

What is .NET?

- A **free, open-source developer platform** for building many types of applications (web, desktop, mobile, gaming, cloud, IoT).
- Supports multiple languages: **C#, F#, VB.NET**.

Components:

- **.NET Framework** (Windows-only, legacy)
- **.NET Core** (Cross-platform, high performance)
- **.NET 5/6/7/8** → Unified, modern cross-platform framework (replaces .NET Core/.NET Framework)

LTS vs STS

Version	Type	Support Until
.NET 6	LTS	Nov 2024
.NET 7	STS	May 2024
.NET 8	LTS	Nov 2026

Cross-Platform Support

- Windows, Linux, macOS
- CLI & IDE support on all major platforms

CLR

CLR – Common Language Runtime

- Virtual machine component
- Handles:
  - **Memory management**
  - **Garbage collection**
  - **Security**
  - **Exception handling**
  - **Threading**

Setting up the IDE

Visual Studio

- Best for full-featured development (IntelliSense, Debugger, GUI Designer)
- Available in Community (free), Pro, and Enterprise

VS Code

- Lightweight, extensible, cross-platform
- Requires:
  - **C# extension (by Microsoft)**
  - **.NET SDK installed**

Anatomy of a Console App

Example: `Program.cs`

```
using System;

namespace HelloWorldApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, world!");
        }
    }
}
```

Structure:

- `using` : Imports namespaces
- `namespace` : Logical grouping
- `class` : Blueprint for objects
- `Main` : Entry point

Key Methods:

- `Console.Write()` – Outputs without newline
- `Console.WriteLine()` – Outputs with newline
- `Console.ReadLine()` – Reads user input

Live Walkthrough: Create & Run Project

CLI Version:

```
dotnet new console -n DemoApp
cd DemoApp
dotnet run
```

Visual Studio:

1. File → New → Project
2. Choose "Console App (.NET Core/.NET 6+)"
3. Write code in `Program.cs`
4. Run with **F5** or **Ctrl + F5**

Q & A

upGrad ENTERPRISE

C# Programming : Data Types & Variables

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer

Value vs Reference Types

Value Types

- Stored **on the stack**
- Hold actual data
- Examples: `int`, `float`, `bool`, `char`, `struct`

Reference Types

- Stored **on the heap**
- Hold **reference (address)** to the actual object
- Examples: `string`, arrays, `class`, `object`, delegates

Example:

```
int a = 5;
int b = a;  // value copied
b = 10;
// a is still 5

string s1 = "hello";
string s2 = s1;  // reference copied
s2 = "world";
// s1 may still point to "hello", but strings are immutable
```

Primitive Types

Integral & Floating Point:

```
int age = 25;
float temp = 36.6f;
double pi = 3.14159;
long population = 7000000000;
```

Character & Boolean:

```
char grade = 'A';
bool isLoggedIn = true;
```

String:

```
string result = String.Empty;
string name = "Narasimha";
string eamil = "tnrao.trainer@gmail.com";
string city = "Hyderabad";
```

`var` and Type Inference

- Let the **compiler infer the type** based on the value assigned.

```
var message = "Hello";    // string
var score = 89;           // int
```

> Must be initialized immediately
>
> Useful with LINQ or anonymous types

Constants

`const` (Compile-time constant)

```
const double Pi = 3.14;
```

5. Type Conversion

Implicit Conversion

```
int a = 100;
long b = a;  // safe
```

Explicit Conversion (Casting)

```
double pi = 3.14;
int approx = (int)pi;  // 3
```

Convert Class

```
string numStr = "123";
int num = Convert.ToInt32(numStr);
```

Parse & TryParse

```
int n = int.Parse("456");
bool isValid = int.TryParse("abc", out int result);  // false
```

> `Parse` throws exception if format is invalid; `TryParse` is safer.

Q & A