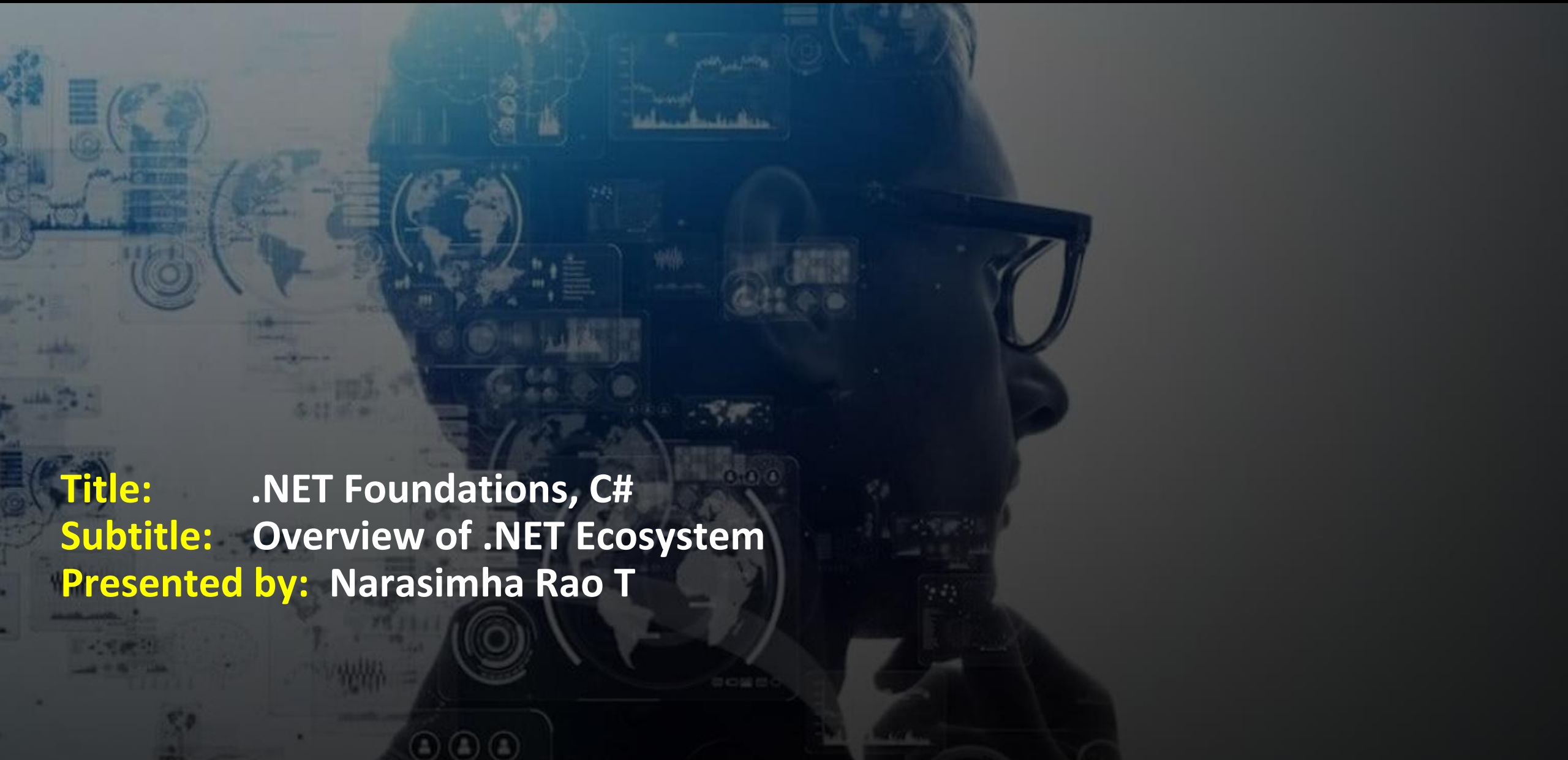
upGrad ENTERPRISE

.NET FSD Bootcamp

upGrad ENTERPRISE



Title: .NET Foundations, C#

Subtitle: Overview of .NET Ecosystem

Presented by: Narasimha Rao T

upGrad ENTERPRISE

.NET FSD Bootcamp

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

Index

1. Overview of .NET ecosystem (.NET Core, .NET 6/8, .NET Framework)
2. Setting up IDE (VS Code, Visual Studio)
3. Namespaces and using directives
4. Anatomy of a console app
5. Writing and running first C# programs
6. Console I/O (Console.ReadLine(), WriteLine())

.NET & C# Fundamentals

1. 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)

	.NET Core / .NET (5/6/7/8+)	.NET Framework
Platform	Cross-platform (Windows, Linux, macOS)	Windows only
Use Case	New apps, cloud, containers, microservices	Legacy enterprise apps
Status	Actively developed	Maintenance mode

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

2. CLR, FCL/BCL, and JIT Explained

CLR – Common Language Runtime

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

FCL/BCL

- **FCL (Framework Class Library)**: Huge set of reusable libraries
- **BCL (Base Class Library)**: Core subset (System, IO, Collections, etc.)

JIT – Just-In-Time Compiler

- Converts **CIL (Common Intermediate Language)** into native code **at runtime**
- Variants:
  - **JIT**
  - **Tiered JIT**
  - **AOT (Ahead-of-Time)** in .NET Native/Blazor

Diagram (text-based):

```
YourApp.cs → C# Code
     ↓
C# Compiler (Roslyn)
     ↓
YourApp.dll (IL/CIL)
     ↓
CLR loads and JIT compiles to native code at runtime
```

3. 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**

CLI Tools:

```
dotnet --version
dotnet new --list
dotnet new console -n MyApp
dotnet run
```

4. 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

5. Namespaces and Using Directives

Namespaces:

- Prevent naming conflicts
- Example:

```
namespace MyCompany.Utilities
{
    class Logger { ... }
}
```

Using Directives:

```
using System;
using MyCompany.Utilities;
```

- Brings namespace members into scope

6. Writing and Running First C# Programs

Hello World:

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Welcome to .NET!");
    }
}
```

Compile & Run (CLI):

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

7. Console I/O

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Enter your name: ");
        string name = Console.ReadLine();
        Console.WriteLine($"Hello, {name}!");
    }
}
```

Key Methods:

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

8. 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**

VS Code:

1. Open terminal → `dotnet new console`
2. Open folder in VS Code
3. Install **C# extension**
4. Add `launch.json` for debugging
5. Run via **Terminal** or **Run > Start Debugging**

9. VS/VS Code Tips & Shortcut Contest

Visual Studio Tips:

- `Ctrl + .` → Quick Actions (rename, import)
- `F12` → Go to Definition
- `Ctrl + Shift + B` → Build
- `Ctrl + K + D` → Format Document

VS Code Tips:

- `Ctrl + P` → Quick file search
- `Ctrl + Shift + ` (backtick) → New terminal
- `Ctrl + Space` → IntelliSense
- `F5` → Debug

Summary

What We've Learned:

- .NET architecture & tooling
- IDE setup & CLI usage
- Writing, compiling, and running C# code
- Key components: CLR, JIT, BCL
- Hands-on with console apps

Q & A