Faysal Aziz

Foundational C#

From Syntax to LINQ and Tooling in 3 days (or 6 x half days)

FA-0208 · 3 days

Software Development · Automation, RPA & Power Platform

Introduction

Rapid, industry-focused C# training with real-world tools and practices

---

Understanding and using the C# language effectively remains a key skill in the modern .NET ecosystem. This intensive three-day course is designed to help you move from the fundamentals of C# syntax and programming through object-oriented design, tooling (Git, Makefile, NuGet) and modern features such as LINQ and the standard library. The instructor brings over 30 years of industry experience and will share how C# is used in real software projects (rather than purely academic examples). You'll gain hands-on experience building simple programs in Visual Studio, working with version control, dependencies and queries, and using modern APIs and error-handling.

Learning outcomes

By the end of the course participants will be able to:

- Write C# programs using the core syntax: variables, control flow (if/else, switch, loops), methods/functions and basic I/O.
- Use Visual Studio (or equivalent) to create, compile and run simple C# console or Windows apps, and explore the IDE features.
- Understand and apply object-oriented principles in C#: classes, objects, inheritance, interfaces, encapsulation, polymorphism.
- Use version control with Git, build/manage dependencies using tools such as Makefile (or equivalent build file) and NuGet for package management.
- Employ the C# standard library: collections, generics, LINQ (Language Integrated Query) for data querying and transformation, and handle exceptions effectively.

- Independently build simple C# applications, manage tooling and dependencies, and lay a foundation for further .NET development.

Prerequisites

- Basic familiarity with programming (variables, loops, functions) in any programming language.
- A computer/laptop with Visual Studio (or Visual Studio Code + .NET SDK) installed and ability to install or manage packages.
- Internet connection for downloading packages and dependencies.
- A Git-capable environment (Git client installed) and permissions to create repositories and commits.

Training outline

1. Fundamentals and Tooling

Setting up the C# environment

- Install .NET SDK (e.g., .NET 7/8+), Visual Studio or Visual Studio Code configured for C#.
- Create a new console application, understand project/solution structure, namespace and entry point (static void Main).

Core C# syntax

- Data types: primitive types (int, double, bool, char), reference types (string, object), value types and reference types.
- Variables, constants (const), readonly, type inference (var).
- Operators: arithmetic, relational, logical, assignment, coalescing (??, ?:).
- Control flow: if/else, switch (including modern switch patterns), loops (for, foreach, while, do-while).
- Methods: definition, parameters (including optional/default parameters), return values, overloading.
- Namespaces, using directives, assemblies.

Using Visual Studio and tooling workflow

- Create, build, run and debug a simple “Hello World” or basic calculator console app.
- IDE features: breakpoints, stepping through code, inspection of variables, watch expressions.

Version control and dependencies

- Git basics: initialize repository, stage, commit, push, branch, merge. Best practices (commit messages, branching model).
- Build file/automation: introduction to Makefile (or equivalent script) for building .NET projects, running tests, cleaning build artifacts.
- NuGet package manager: adding external libraries, managing versions, restoring packages.
- Project setup: .csproj, PackageReference, build and run from CLI (dotnet build, dotnet run).

2. Object-Oriented Programming and Standard Libraries

Deep dive into OOP in C#

- Classes and objects: fields, properties, methods, constructors, destructors/finalizers.
- Encapsulation: access modifiers (public, private, protected, internal), automatic properties, expression-bodied members.
- Inheritance: base and derived classes, virtual, override, sealed, abstract classes.
- Interfaces: defining contracts, implementing multiple interfaces, explicit interface implementation.
- Polymorphism: dynamic dispatch, interface polymorphism, design considerations (composition vs inheritance).
- Structs vs classes: value types vs reference types, when to use each.

Using the standard library and modern language features

- Collections and generics: List<T>, Dictionary<TKey,TValue>, IEnumerable<T>, IQueryable<T>.
- Generics constraints, variance, type inference.
- Exception handling: try/catch/finally, throw, custom exceptions, best practices (avoid swallowing exceptions, use meaningful error messages).
- Newer C# language enhancements (C#9-C#14, such as records, pattern matching, global usings, file-scoped namespaces).

LINQ (Language Integrated Query)

- Concept of LINQ: queries against collections, uniform syntax for data sources.
- Two forms: query expression syntax (from ... in ... where ... select ...) and method syntax (extension methods + lambdas).
- LINQ operators: Where, Select, OrderBy, GroupBy, Join, Aggregate.
- LINQ to Objects vs LINQ to other sources (e.g., LINQ to XML, LINQ to Entities).

- Hands-on: build a console app filtering and projecting lists of objects, then extend to e.g., reading data from a file or database and using LINQ to transform.

Practical project and build integration

- Build a console (or small GUI) application that leverages OOP, uses standard library collections, handles exceptions robustly, and uses LINQ to process data.
- Integrate with Git: track code versions, branch for features, perhaps merge simple changes.
- Use NuGet to add a library (e.g., JSON parsing library) and handle dependencies. Use Makefile/CLI script for build/run tasks.

3. Tooling, Advanced Topics and Wrap-Up

Build and automation advanced practices

- Extend Makefile or CLI script: run unit tests, build debug vs release configurations, packaging executable or library.
- Dependency management: NuGet versioning, package updates, transitive dependencies.
- Git workflows: feature branches, pull requests, merge conflict resolution, tagging releases.

Advanced C# language and library features (overview)

- Latest features in C# (e.g., records, pattern matching enhancements, file-scoped namespace, global usings) and how they impact writing cleaner code.
- Threading and asynchronous programming (introductory): Task, async/await, Parallel class, concurrency concerns (for future extension).

Final hands-on: build a small end-to-end application

- Example: Data processing application that reads a dataset (e.g., CSV/JSON), uses LINQ to filter/transform, uses classes/interfaces for modular design, handles errors, builds via script/Makefile, uses Git, packages via NuGet (if applicable) and runs.
- Demonstrate full flow: edit code → build → test → commit → package.

Review, next steps and resource pointers

- Discussion: how C# fits into enterprise/back-end/cloud development (.NET Core, ASP.NET, microservices).
- Guidance on how to continue learning: e.g., ASP.NET Core, Entity Framework, unit testing frameworks, containerization.

- Q&A, wrap-up course deliverables, feedback and certifications.

This three-day intensive course will give you a practical, solid foundation in C#, using tools and workflows that matter in industry today. The instructor’s decades of experience ensure that this is not just academic theory, but skills you can immediately apply in real-world projects.

© 2026 Faysal Aziz · Kuala Lumpur