Chapter: C# Fundamentals —+

Section 1 — C# and .NET Introduction 🔧

Summary: What C# is, where it runs (CLR / .NET), and the developer tooling ecosystem.

- Key idea: C# is a language; .NET is the runtime & libraries (BCL). C# programs compile to IL and run on the CLR (or CoreCLR / .NET).

Helpful diagram:

- .NET platform overview (official docs) — https://learn.microsoft.com/dotnet/architecture/overview

Snippets:

```
using System;
class Program { static void Main() => Console.WriteLine("Hello, .NET!"); }
```

Best practice: Prefer cross-platform .NET (current LTS), use SDK-style projects.

Section 2 — C# Program Structure & Entry Points 🧭

Summary: Files, namespaces, classes, the `Main` entry point variants, top-level statements.

Diagram: Program flow / entry point illustration — https://upload.wikimedia.org/wikipedia/commons/3/33/Flowchart_for_program_execution.png

Snippets:

```
// Traditional Program class
namespace DemoApp {
  class Program {
    static void Main(string[] args) {
      Console.WriteLine(args.Length > 0 ? args[0] : "no args");
    }
  }
}
```

```
// Top-level statements (C# 9+)
using System;
Console.WriteLine("Top-level app started");
```

```
// Async Main
using System.Threading.Tasks;
class Program { static async Task Main() { await Task.Delay(10); }
}
```

Section 3 — Primitive Data Types (int, long, float, double, decimal, bool, char) 🔢

Summary: Fixed-size and value types; numeric precision and when to use `decimal` (money).

Snippets:

```
int a = 42;
long big = 3_000_000_000L;
float f = 3.14f;
double d = 2.71828;
decimal money = 19.99m;
bool ok = true;
char letter = 'A';
```

Tip: Use `var` for local inference when the type is obvious, otherwise explicit types aid readability.

Section 4 — String and StringBuilder 🧵

Summary: `string` is immutable; use `StringBuilder` for heavy mutations.

Snippets:

```
string s = "Hello" + " World";
string template = $"User: {Environment.UserName}, Date: {DateTime.Today:d}";
```

```
using System.Text;
var sb = new StringBuilder();
sb.Append("Line1").AppendLine();
sb.AppendFormat("{0} items", 5);
string result = sb.ToString();
```

Exercise: Time a concatenation loop vs. StringBuilder for 100k iterations and observe differences.

Section 5 — var and Type Inference 🧠

Summary: `var` infers compile-time type; not a dynamic type. Use where the type is clear.

Snippets:

```
var x = 10;            // int
var list = new List<string>(); // List<string>
```

Rule: Avoid `var` when it reduces clarity (e.g., `var data = GetSomething()` where GetSomething() return type isn't obvious).

Section 6 — Nullable Types ❓

Summary: Value types can be made nullable with `T?`. Use `?.` and `??` for safe access and defaults.

Snippets:

```
int? n = null;
int value = n ?? -1;          // coalesce
int? length = s?.Length;      // safe access
```

Diagram: Null-propagation and coalescing examples: https://learn.microsoft.com/dotnet/csharp/language-reference/operators/null-coalescing-operator

Section 7 — Control Flow — Conditional Statements 🧩

Summary: `if/else`, `switch` / `case`, modern `switch` expressions and pattern-rich switches.

Snippets:

```
if (x > 0) Console.WriteLine("pos");
else if (x < 0) Console.WriteLine("neg");
else Console.WriteLine("zero");
```

```
// switch expression
var result = x switch {
  0 => "zero",
  > 0 => "positive",
  < 0 => "negative"
};
```

Exercise: Convert nested ifs into a switch expression.

Section 8 — Pattern Matching 🎯

Summary: Powerful; matches shapes, types, and values. Useful in `switch` and `is` expressions.

Snippets:

```
object o = 5;
if (o is int i) Console.WriteLine(i + 1);

// property pattern
if (person is { Age: >= 18, Name: var n }) Console.WriteLine(n);
```

```
// switch with type patterns
string Describe(object? obj) => obj switch {
  null => "none",
  int i => $"int {i}",
  string s => $"str({s})",
  _ => "other"
};
```

Section 9 — Logical Operators & Comparison Operators ⚖️

Summary: `&&`, `||`, `!` are short-circuiting; equality uses `==` (value or overloaded). Be careful with reference vs. value equality.

Snippets:

```
if (x > 0 && x < 100) { /* in range */ }
if (!(isReady || hasData)) return;
```

Equality example:

```
string a = "hi";
bool eq = a == "hi"; // true
```

When overriding equality, also override GetHashCode.

Section 10 — Mathematical Operators ➗✖️

Summary: Basic arithmetic and modulo. Integer division truncates.

Snippets:

```
int sum = a + b;
double ratio = (double)numerator / denominator;
int r = 7 % 3; // 1
```

Edge: Watch overflow for integers — consider `checked`/`unchecked` or `BigInteger`.

Section 11 — Arrays and Collections 🗂️

Summary: Fixed-size arrays and resizable collections (`List<T>`). Prefer `List<T>` for dynamic data.

Snippets:

```
int[] arr = new int[3] {1,2,3};
var list = new List<string> {"a","b"};
list.Add("c");
```

Linq quick example:

```
using System.Linq;
var evens = listOfInts.Where(i => i % 2 == 0).ToArray();
```

Diagram: Collections overview (MS docs) — https://learn.microsoft.com/dotnet/standard/collections

---

Section 12 — Control Flow — Loops 🔁

Summary: `for`, `foreach`, `while`, `do-while`. `foreach` safest for IReadOnly patterns, `for` for index access.

Snippets:

```
for (int i = 0; i < arr.Length; i++) Console.WriteLine(arr[i]);
foreach (var item in list) Console.WriteLine(item);
int i = 0; while (i++ < 5) { /* ... */ }
```

Tip: Don't modify a collection while enumerating it; use a separate list or indexing.

Section 13 — Comments — Inline, Block, XML documentation 📝

Summary: Use `///` XML comments for public APIs; inline comments sparingly. Generate docs with `dotnet` toolchain.

Snippets:

```
// Inline comment
/* block comment */
/// <summary>Gets the name.</summary>
public string Name { get; }
```

Tooling: `dotnet build` + XML docs output in csproj to generate API docs.

Section 14 — Namespaces and Using Directives 📦

Summary: Group types with namespaces; use `using` for short names, avoid global using pollution.

Snippets:

```
namespace Company.Product.Module {
  public class Worker { }
}

using System.Text; // brings StringBuilder into scope
```

C# 10 feature: `global using` can reduce boilerplate for large solutions but use sparingly.

Section 15 — Debugging in Visual Studio 🐞

Summary: Breakpoints, watches, immediate window, Hot Reload, and step-through debugging. Use unit tests as fast feedback loops.

Short checklist:

- Set breakpoints, conditional breakpoints
- Inspect locals and watch expressions
- Use exception settings to break on thrown exceptions
- Use Live Unit Testing (if available)

Useful links:

- VS Debugging docs — https://learn.microsoft.com/visualstudio/debugger/debugger-feature-tour

---

Mini Exercises

- Implement a `Person` record, parse lines from a CSV into `List<Person>`, then filter adults using pattern matching and LINQ.
- Compare concatenation with `+` and `StringBuilder` using Stopwatch.

Final Notes & Further Reading 💡

- Patterns & Practices: Martin Fowler's refactorings and Mark Seemann on DI are excellent next steps.
- Microsoft C# guide: https://learn.microsoft.com/dotnet/csharp/

Suggested filename for a chapter: **CSharp-Fundamentals-Chapter.md**

> End of chapter — ready to use as a 2-hour workshop-ready reference. ✅