# G# PROGRAMMING LANGUAGE UNLOCKED

## FROM BEGINNER TO EXPERT

A COMPLETE GUIDE TO MODERN .NET DEVELOPMENT
WITH GO, KOTLIN, AND SWIFT ERGONOMICS

**LEARN FAST**
Clear, modern syntax
built for productivity

**WRITE BETTER CODE**
Expressive, concise,
and easy to maintain

**BUILD REAL SYSTEMS**
From simple apps to
scalable architectures

**POWERED BY THE CLR**
Leverage the full strength
of the .NET runtime

**MODERN & INTEROPERABLE**
Seamless .NET integration
and cross-platform ready


GO SIMPLICITY | EXPRESSIVENESS | ELEGANCE

- COMPLETE WORKING EXAMPLES
- BEST PRACTICES & PATTERNS
- ARCHITECTURAL GUIDANCE
- HONEST & PRACTICAL INSIGHTS

JONATHAN BRENNA

# G# Programming Language Unlocked

## From Beginner to Expert : A Complete Guide to Modern .NET Development with Go, Kotlin, and Swift Ergonomics

Steve Publications

This book is available at
https://leanpub.com/gprogramminglanguageunlocked

This version was published on 2026-08-21

# Contents

# From Beginner to Expert : A Complete Guide to Modern .NET Development with Go, Kotlin, and Swift Ergonomics

**About this book:** G# is a modern programming language for the .NET runtime that brings the clarity of Go, the expressiveness of Kotlin and Swift, and the full power of the CLR into one cohesive tool. This book takes you from your first program to production-ready systems, covering every major feature of the language with complete working code examples, best practices, architectural guidance, and honest assessment of where G# fits in today's ecosystem. Whether you are a C# developer seeking simplicity, a Go programmer wanting .NET access, or a beginner looking for an approachable entry point, this book provides the depth and rigor needed to write professional-quality G# software.

# Introduction : Why Another Language?

The story of modern programming is partly a story of fragmentation. Over the last decade alone, developers have watched new languages emerge with promises of simplicity, performance, safety, or developer happiness. Many deliver on their promises in narrow domains but leave engineers juggling multiple ecosystems, build systems, and deployment models just to ship one product.

G# arrives in this landscape with a different proposition. Instead of asking you to adopt an entirely new runtime or abandon your existing libraries, it compiles directly to the Common Language Runtime and speaks the same assembly format as C#, F#, and every .NET language that came before. The difference is what you write before it gets there: smaller syntax, clearer concurrency models, fewer footguns, and design choices borrowed from languages whose ergonomics have proven themselves in production.

If you already know .NET, G# gives you another way to build managed assemblies and call CLR APIs under the normal dotnet toolchain. If you come from Go, Kotlin, or Swift, you will recognize packages, functions, slices, channels, null safety, and structured concurrency without having to learn an entirely new mental model. And if you are new to programming altogether, G# keeps its surface intentionally small so that each concept builds on the last without overwhelming you with ceremony.

This book assumes only basic familiarity with programming ideas like variables, loops, and functions. It does not assume prior knowledge of .NET, C#, or any specific framework. Every chapter introduces concepts incrementally, demonstrates them with complete working code, and explains why the language works the way it does. By the time you reach the final chapters, you will have enough practical experience and architectural understanding to design, implement, test, and deploy substantial G# applications in production environments.

The journey ahead is organized deliberately. We begin by installing tools and writing your first program so that you can start experimenting immediately. We then walk through language fundamentals one layer at a time:

variables and types, control flow, functions, data structures, error handling, concurrency, interoperability with the .NET ecosystem, and advanced type system features. The later chapters shift from learning syntax to engineering software : testing strategies, debugging workflows, performance profiling, deployment patterns, and idiomatic design that distinguishes novice code from production-quality systems.

One note of honesty before we begin: G# is still a young language. As I write this book it sits at version 0.4, which means the surface area continues to evolve and some features documented under “next” have not yet shipped in stable releases. I will flag unreleased or experimental behavior clearly throughout the text so you can distinguish what works today from what is on the roadmap. Early-stage languages carry real risks in production, but they also offer rare opportunities: the chance to shape idioms before they harden, to contribute feedback directly to maintainers, and to adopt a tool whose design decisions still reflect thoughtful tradeoffs rather than decades of backward compatibility debt.

With that context set, let us begin by understanding exactly what G# is, where it came from, and why its creator built it the way he did.

# Chapter 1: What Is G#? : A Modern Language for the .NET Era

## The Problem G# Solves : Too Many Languages, Too Much Ceremony

Every technology decision carries tradeoffs. C# is undeniably powerful: a mature language with decades of ecosystem investment, first-class tooling from Microsoft, and near-universal support across cloud platforms, desktop environments, and mobile frameworks. Yet that power comes with complexity. The language has absorbed features over more than twenty years including properties, events, delegates, LINQ, async/await, pattern matching, records, spans and source generators. While each addition solved a real problem at the time, the cumulative effect is a language whose full surface area can take months or years to master.

For beginners this creates friction. A student trying to write their first .NET application must navigate namespaces, using directives, class wrappers around entry points, property syntax versus field syntax, and a type system with subtle rules about nullability, variance, and reference semantics. For experienced developers coming from other ecosystems the learning curve feels steep for what amounts to calling Console.WriteLine or reading a file.

At the same time, many engineers have grown accustomed to languages like Go that offer remarkably small surfaces. Go deliberately omits classes, inheritance, generics (until relatively late), and exception handling in favor of simplicity and predictability. The tradeoff is clear: you gain clarity and fast onboarding but lose access to powerful abstractions and, critically, to the vast .NET ecosystem if your work depends on it.

G# was designed to occupy the space between these two extremes. It keeps the runtime and tooling familiarity of .NET while offering a language surface that feels closer to Go in its compactness and intentionality. You get packages instead of sprawling namespace hierarchies, functions declared

with func rather than nested inside class scaffolding, slices and maps as first-class collection types, and structured concurrency built into the grammar. At the same time you retain full access to CLR features when you need them: inheritance for polymorphic hierarchies, interfaces for contracts, async/await for I/O-bound workloads, attributes for metadata, and every NuGet package ever published for .NET.

The result is a language that asks less of you in the common case while refusing to limit you in the uncommon one. If your problem can be solved with straightforward data structures and functions, G# gets out of your way. If you need sophisticated object hierarchies or deep integration with existing .NET frameworks, those capabilities are available without workarounds or escaping to C#.

## Origins and Vision : David Obando, CSharp Meets Go

G# was created by software engineer David Obando and first announced publicly in mid-2024 [1]. The name itself tells the story: it combines CSharp and Go, reflecting both its target platform and its stylistic inspiration. In his own words from the project repository, Obando wanted to enable developers to benefit from the .NET framework without having to learn all of C# [2]. He specifically called out new developers who find C#'s learning curve steep, VB.NET developers seeking a more modern syntax, Python developers who want strong typing without verbosity, and non-English speakers for whom English-heavy keywords can create cognitive load.

The vision is pragmatic rather than ideological. Obando is himself a Go developer who continues to use Go where it is appropriate; G# was never intended to steal users from Go but rather to offer a bridge for developers whose skills in Go could transfer productively into the .NET world [2]. If you understand packages, func declarations, defer, goroutines, channels, and select statements, much of that knowledge maps directly to G# constructs. The differences are deliberate adaptations to CLR realities: classes exist because .NET expects them, exceptions exist because the BCL throws them, and async/await exists because Task-based asynchrony is fundamental to modern .NET libraries.

The project carries a copyright date of 2026 and is published under the MIT license, making it freely available for commercial and open-source use

[3]. Development happens primarily on GitHub where the repository has attracted hundreds of stars and contributions from the community [4]. The documentation site at davidobando.github.io/gsharp provides a language tour, tutorials, a formal specification, tooling guides and an “Effective G#” idioms guide. All of these resources indicate serious intent to build a usable, well-documented language rather than a toy experiment.

## Design Principles : Small Surface, Full Power

Understanding why G# makes the design choices it does is essential for writing idiomatic code. The language follows several guiding principles that appear repeatedly in its specification and documentation.

**Minimize ceremony.** Entry points do not require wrapping functions inside class declarations. Variables are declared with var or let rather than verbose type annotations where inference suffices. Loops use simple for-in syntax instead of iterator protocol boilerplate. These choices reduce the cognitive overhead of writing basic programs without sacrificing expressiveness.

**Be explicit where ambiguity hurts.** Unlike Go which uses capitalization to control visibility, G# requires explicit public, private, or internal modifiers on declarations. Numeric types carry their width explicitly in canonical form (int32 rather than int) at API boundaries to prevent cross-library confusion. Nullable types are marked with ? suffixes that the binder checks statically. These decisions prioritize clarity and safety over convenience.

**Respect CLR realities.** G# compiles to managed assemblies using standard ECMA-335 metadata emission [5]. This means it must interoperate seamlessly with existing .NET code: C# libraries expect classes with properties and events, F# code uses discriminated unions represented as sealed class hierarchies, NuGet packages rely on specific attribute conventions. G# supports all of these patterns natively rather than trying to hide them behind abstractions that break compatibility.

**Provide multiple concurrency models.** The language recognizes that different teams have different preferences and expertise. The always-available surface includes structured concurrency via scope blocks, task-based async/await, and async streams. Go-style goroutines, channels, and select statements are available through an opt-in extension package. This flexibility

lets you choose the model that matches your team's mental model while keeping the core language simpler.

**Document decisions, not just features.** The project maintains a design decisions index alongside its specification [6], explaining not only what the language does but why it makes particular tradeoffs. This transparency helps users understand when to follow idioms and when deviations are justified.

## Where G# Fits : Compared to C#, F#, Go, Kotlin, Swift

Positioning a new language requires honest comparison with established alternatives. Let us examine how G# relates to the languages most likely to compete for your attention.

**G# versus C#.** Both target the CLR and share access to the same runtime, libraries, and tooling ecosystem. C# is more mature, has broader industry adoption, and offers a richer feature set including things like pattern matching on tuples, primary constructors, file-scoped namespaces, and extensive source generator support. G# trades some of that richness for simplicity: fewer keywords, less syntactic sugar layered on top of itself, and design choices that favor readability over cleverness. If your project demands cutting-edge C# features or heavy integration with frameworks optimized for C# conventions, C# remains the safer choice. If you want .NET power with a smaller learning curve and more Go-like ergonomics, G# becomes attractive.

**G# versus F#.** F# is also a CLR language but takes a fundamentally different approach: it is primarily functional-first with immutable data by default, computation expressions for monadic workflows, and a type system featuring discriminated unions, active patterns, and advanced inference. G# is more imperative and object-oriented in its defaults while still supporting functional patterns through higher-order functions, immutability via let bindings, and data types with structural equality. Choose F# if your team values functional programming as a first-class paradigm; choose G# if you prefer a more mainstream imperative style with optional functional features.

**G# versus Go.** Go influenced G#'s syntax heavily. Packages, func declarations, slices, maps, defer statements and the optional goroutine/channel/select model are all directly inspired by Go. The critical difference is runtime: Go compiles to native binaries with its own garbage collector and standard library, while G# runs on the CLR and depends on .NET's GC and BCL. This means G#

inherits .NET strengths (mature async libraries, rich data access stacks, cross-platform UI frameworks) but loses Go's single-binary deployment story and compile-time speed. Use Go for systems programming where control over binaries matters; use G# when you need those same ergonomics inside the .NET ecosystem.

**G# versus Kotlin.** Kotlin shares with G# an interest in developer ergonomics: null safety, data classes with structural equality, extension functions, and concise syntax are common to both. Kotlin targets the JVM primarily (with Kotlin/Native as a secondary target), while G# targets the CLR. The ecosystems differ substantially: Kotlin benefits from Android's massive adoption and JetBrains' tooling investment; G# benefits from .NET's cloud-native momentum and Microsoft's infrastructure investment. If your world is Java or Android, Kotlin is the natural choice. If your world is Azure, ASP.NET Core, or Windows development, G# positions itself as the modern alternative to C#.

**G# versus Swift.** Swift influenced aspects of G#'s design around value semantics, protocol-oriented programming concepts, and modern syntax aesthetics. Swift dominates iOS/macOS development while G# targets .NET's cross-platform story including Linux servers, Windows desktops, and WebAssembly. The overlap is more stylistic than practical: both languages care about readability and safety, but they serve different platforms.

The honest assessment is that G# does not attempt to beat any of these languages at their home game. Instead it occupies a niche: developers who want Go-like clarity and Kotlin-style ergonomics while working inside the .NET ecosystem. If that describes your situation, G# deserves serious consideration.

## Current Status and Ecosystem : Version 0.4, Tooling Landscape

As of this writing G# is at version 0.4, which places it firmly in pre-1.0 territory [7]. This has important implications for potential adopters.

**What works today.** The core language surface is functional and documented: packages, imports, functions, variables, control flow, structs, classes, data types, interfaces, generics, arrays, slices, maps, sequences, async/await, structured concurrency with scope, exception handling, attributes, and full CLR interoperability are all implemented [8]. The compiler emits valid managed

assemblies with Portable PDB debugging symbols. Projects build through the standard dotnet toolchain using the Gsharp.NET.Sdk MSBuild SDK published on NuGet [9]. A VS Code extension provides syntax highlighting, completion, diagnostics, formatting, and debugger configuration [10].

**What is maturing.** The Go-flavored concurrency layer (goroutines, channels, select) is available through the Gsharp.Extensions.Go package but marked as opt-in rather than core [11]. Source generator support via gsgen exists for integrating with Roslyn-based tooling. The cs2gs migration tool can translate C# projects into G# and run a verification pipeline to discover gaps in language coverage [12].

**What is unreleased or experimental.** Some documentation pages are marked as "next" indicating features under development that have not shipped in stable releases yet [13]. These may include additional standard library helpers, refined error messages, or expanded interop scenarios. I will flag such features throughout this book so you can distinguish production-ready capabilities from work-in-progress.

**Community and ecosystem size.** The GitHub repository shows active development with several hundred stars and regular commits [4]. NuGet packages for the SDK, templates, REPL, and migration tool are publicly available [9]. Discussions appear on Reddit's r/dotnet community and Hacker News where reception has been generally positive but realistically cautious. Readers recognize G# as interesting while noting that language adoption requires more than technical merit [14][15]. No major commercial organizations have publicly announced large-scale G# migrations yet, which is expected for a pre-1.0 project.

The bottom line: G# is usable today for learning, prototyping, and even production work in contexts where you can tolerate the risk of evolving APIs. It is not yet ready to be the sole language for mission-critical systems with long-term maintenance obligations unless your team is prepared to track upstream changes actively.

## Who Should Read This Book : Prerequisites and Roadmap

This book is designed for several audiences:

**Beginners new to programming.** If you have never written code before, G# offers a gentler introduction than many alternatives. Its small keyword set, explicit types, and straightforward concurrency model reduce confusion early on. You will still need to learn fundamental concepts like variables, loops, functions and data structures. This book teaches those using G# as the vehicle.

**C# developers seeking simplicity.** If you know C# but find its complexity burdensome for smaller projects or teams with mixed experience levels, this book shows you how G# expresses similar ideas with less ceremony while still accessing all your existing .NET libraries.

**Go developers wanting .NET access.** If you love Go's ergonomics but need to work inside the .NET ecosystem for business reasons, this book maps your existing mental model onto G#'s constructs and explains where CLR realities force adaptations.

**Kotlin or Swift developers exploring alternatives.** If those languages shaped your expectations about what modern syntax should look like, G# will feel familiar in many ways. This book highlights the parallels while teaching you the .NET-specific concepts you will need.

**Technical evaluators and architects.** If you are assessing whether G# is suitable for your organization's projects, this book provides enough technical depth covering type system behavior, runtime model, interop capabilities, deployment patterns and performance characteristics to make informed decisions.

The only prerequisite is basic familiarity with programming concepts. You should understand what a variable, loop, function, and data structure are at an abstract level. Prior exposure to any specific language helps but is not required. If you have used C#, Java, Python, JavaScript, Go, or similar languages before, you will find many concepts familiar even if the syntax differs.

Here is how the book is organized:

- **Chapters 2 through 6** cover language fundamentals from installation through control flow and functions. These chapters build incrementally so that each new concept rests on previously introduced material.
- **Chapters 7 through 10** explore data modeling, error handling, generics and concurrency. These are the features that distinguish competent code from robust systems.

- **Chapter 11** dives into .NET interoperability, showing how G# leverages the full ecosystem of libraries, frameworks, and tools available on the CLR.
- **Chapter 12** covers development workflows including testing, debugging, profiling, and migration from C#.
- **Chapter 13** addresses production engineering concerns: architecture patterns, performance optimization, security, deployment strategies, and idiomatic design principles.

By the end you will have written dozens of complete programs, understood every major language feature, and gained enough practical experience to evaluate G# honestly for your own use cases. Let us begin with the concrete step of getting tools installed and writing your first program.

# Chapter 2: Getting Started : Installation, Tooling, and Your First Program

## Prerequisites : .NET SDK Versions and Verification

G# does not ship with its own runtime. Instead it compiles to managed assemblies that run on the Common Language Runtime, which means your first step is ensuring you have a compatible .NET SDK installed.

G# version 0.4 supports targeting .NET 8, .NET 9, and .NET 10, with .NET 10 being the preferred and default target framework [16]. The tooling itself requires at minimum a .NET runtime capable of loading net10.0 assemblies for some components like the VS Code language server.

To verify your installation, open a terminal and run:

```
dotnet --info
```

Look for SDK listings that include version 8.0.x, 9.0.x, or 10.0.x. If you see at least one of these, you are ready to proceed. If dotnet is not recognized or no compatible SDK appears, install the .NET SDK from Microsoft's official site [17]: https://dotnet.microsoft.com/download.

On Linux systems using package managers, you can typically install via:

```
# Ubuntu/Debian
sudo apt update && sudo apt install dotnet-sdk-10.0

# Fedora/RHEL
sudo dnf install dotnet-sdk-10.0
```

On macOS with Homebrew:

```
brew install --cask dotnet-sdk
```

Windows users should download the installer from Microsoft's site or use winget:

```
winget install Microsoft.DotNet.SDK.10
```

After installation, verify again with dotnet –info before continuing. The rest of this chapter assumes you have a working SDK.

## Installing the MSBuild SDK and Templates

The recommended way to work with G# is through its MSBuild SDK integration, which lets you use familiar dotnet commands for building, running, and publishing projects. Two NuGet packages provide the foundation:

- **Gsharp.NET.Sdk**: The build system that compiles .gs files into managed assemblies [9].
- **Gsharp.Templates**: Project templates for dotnet new scaffolding [18].

Install the templates with this command:

```
dotnet new install Gsharp.Templates
```

You should see output confirming installation of templates including gsharp-console, which creates a basic console application project. If you encounter errors about package resolution, ensure your NuGet configuration includes the default public feed by running:

```
dotnet nuget list source
```

The output should include https://api.nuget.org/v3/index.json. If not, add it with:

```
dotnet nuget add source https://api.nuget.org/v3/index.json -n nuget.org
```

With templates installed, create your first project:

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

This generates a minimal project structure. Inspect the contents:

```
# On Linux/macOS
ls -la

# On Windows PowerShell
Get-ChildItem
```

You should see at least two files: Program.gs and HelloWorld.gsproj. The .gsproj file is your project definition, and it looks like this:

```
<Project Sdk="Gsharp.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net10.0</TargetFramework>
  </PropertyGroup>
</Project>
```

Notice the Sdk attribute points to Gsharp.NET.Sdk rather than the standard Microsoft.NET.Sdk. This SDK knows how to find and compile .gs files automatically without requiring explicit Compile item declarations. The OutputType of Exe means this project builds a runnable application rather than a library.

The default Program.gs contains a simple starting point:

```
package HelloWorld

import System

func Main() int {
    Console.WriteLine("Hello from G#!")
    return 0
}
```

Let us build and run it immediately to confirm everything works:

```
dotnet build
```

You should see compilation output ending with a success message. Then:

```
dotnet run
```

Output:

```
Hello from G#!
```

Congratulations : you have just built and executed your first G# program using standard .NET tooling. The same commands that work for C# projects (dotnet build, dotnet run, dotnet publish) work identically here because G# integrates at the MSBuild level rather than requiring separate toolchains.

## Setting Up Visual Studio Code with the G# Extension

While you can write G# in any text editor, Visual Studio Code provides excellent support through an official extension that adds syntax highlighting, IntelliSense completion, hover documentation, diagnostics, formatting, and integrated build/run commands.

Install the extension from within VS Code by opening the Extensions view (Ctrl+Shift+X or Cmd+Shift+X on macOS), searching for “G#”, and installing the extension published by gsharplang [10]. Alternatively, install from the command line:

```
code --install-extension gsharplang.vscode-gsharp
```

The extension requires a .NET 10 runtime to run its language server. If you do not have one installed, the extension can download it automatically via the .NET Install Tool when prompted [19].

After installation, open your HelloWorld project folder in VS Code:

```
code HelloWorld
```

You should immediately see syntax highlighting for your G# source file. The extension provides several useful commands accessible through the Command Palette (Ctrl+Shift+P or Cmd+Shift+P):

- **G#: Build** : Runs dotnet build on the current project
- **G#: Run** : Builds and runs the application
- **G#: Restart Language Server** : Useful if you encounter stale diagnostics

The extension also configures debugging out of the box. Set a breakpoint by clicking in the gutter next to a line number, then press F5 to start a debug session. You can inspect variables, step through code, and view call stacks using the standard VS Code debugging interface. This works because G# emits Portable PDB files that VS Code understands natively.

If you prefer other editors, any LSP-compatible editor (Neovim, Emacs, Sublime Text with plugins) can use the G# language server directly by installing it as a global tool:

```
dotnet tool install --global Gsharp.LanguageServer
```

Configure your editor to point at the installed binary following its LSP integration documentation. The core language features work identically regardless of editor choice; VS Code simply provides the most turnkey experience.

## The gsc Compiler : Direct Compilation Without Projects

While MSBuild integration is the recommended workflow for real projects, G# also ships a standalone compiler called gsc that lets you compile individual .gs

files without creating project structures. This is useful for quick experiments, scripting, or understanding what happens under the hood.

You can access gsc either by building it from source or by using the compiler that ships with the SDK. To build from source:

```
git clone https://github.com/DavidObando/gsharp.git
cd gsharp
dotnet build GSharp.sln
```

This produces the compiler binary in the output directories. For most users, however, the simpler approach is to use the compiler through the installed SDK infrastructure or to run single files directly with dotnet.

The basic gsc invocation looks like this:

```
dotnet path/to/gsc.dll Program.gs
```

Without an /out flag, gsc compiles and runs the program immediately. With /out you save the assembly:

```
dotnet path/to/gsc.dll Program.gs /out:Program.exe /target:exe /tfm:net10.0
```

This produces a managed executable at Program.exe that you can run with dotnet run or directly via dotnet Program.exe. The flags control behavior:

- **/out:path** : Specifies the output assembly path
- **/target:exe|library** : Chooses between executable and DLL output
- **/tfm:framework** : Sets the target framework (net8.0, net9.0, net10.0)
- **/r:assembly** : Adds a reference to another assembly
- **/debug** : Emits debug symbols (Portable PDB by default)
- **/doc:path** : Generates XML documentation file

Additional flags support response files (@file.rsp), warning suppression (/nowarn:codes), treating warnings as errors (/warnaserror+), deterministic builds (/deterministic+), and embedded sources (/embed+) [20]. These options mirror familiar C# compiler flags where practical, making the transition between languages smoother.

For day-to-day development you will mostly use dotnet build and dotnet run through MSBuild, but understanding gsc helps when troubleshooting build issues or working in constrained environments without full SDK tooling.

## The gsi REPL : Interactive Exploration and Scripting

G# includes an interactive read-eval-print loop called gsi that serves two purposes: exploring the language interactively and running single .gs files as scripts without project scaffolding. It ships as a .NET global tool [21].

Install it with:

```
dotnet tool install --global Gsharp.Repl
```

Run it with:

```
gsi
```

You enter a full-screen terminal user interface built on Spectre.Console that provides live syntax coloring, completion suggestions, hover information, and real-time diagnostics : all powered by the same compiler engine used in production builds [21]. The interface includes tabs for viewing declared variables and command history.

Try typing some expressions:

```
let x = 42
x + 8
"Hello, $x!"
```

The REPL evaluates each line and displays results immediately. This is invaluable for learning syntax, testing small snippets, or prototyping algorithms before committing them to files.

Key commands within the REPL (accessed by typing a period followed by the command):

- **.reset** : Clears all state and starts fresh
- **.load file.gs** : Loads and executes a source file
- **.theme name** : Changes the color theme

For scripting, gsi can run .gs files directly without requiring a project:

```
gsi myscript.gs
```

This compiles and executes the file on the fly. You can add assembly references with /r flags:

```
gsi myscript.gs /r:System.Text.Json.dll
```

The installed Gsharp.Repl tool bundles Gsharp.Extensions automatically, so scripts can use optional helpers like sequence utilities without adding explicit references [21]. This makes gsi excellent for quick automation tasks, data processing scripts, or learning exercises where creating a full project feels like overkill.

Note that gsi uses only the "emit" engine : the legacy tree-walking evaluator was removed in version 0.4 [22]. All code runs through proper CIL emission and execution on the CLR, so behavior matches production compilation rather than an interpreter approximation.

## Your First Program : Hello, World and Beyond

Let us now write a slightly more interesting program that demonstrates several basic G# concepts together: package declarations, imports, functions, variables, string interpolation, and console I/O. Create a new project:

```
dotnet new gsharp-console -n FirstSteps
cd FirstSteps
```

Replace the contents of Program.gs with this code:

```
package FirstSteps

import System
import System.IO

func Main() int {
    let programName = "FirstSteps"
    var count = 3

    Console.WriteLine("Welcome to G# programming!")
    Console.WriteLine("Program: $programName")

    for i in 1 .. count + 1 {
        Console.WriteLine("Countdown: $i")
    }

    let message = getGreeting("Reader")
    Console.WriteLine(message)

    return 0
}

func getGreeting(name string) string {
    return "Hello, $name! You are learning G#."
}
```

Build and run:

```
dotnet run
```

Output:

```
Welcome to G# programming!
Program: FirstSteps
Countdown: 1
Countdown: 2
Countdown: 3
Hello, Reader! You are learning G#.
```

Let us walk through what this program does:

- **package FirstSteps** declares the package name, which maps to a CLR namespace. Every G# source file begins with a package declaration [23].

- **import System** and **import System.IO** bring .NET namespaces into scope. The System namespace is implicitly imported unless you disable that behavior with compiler flags [24].
- **func Main() int** declares the entry point function. It takes no parameters and returns an int exit code. Zero conventionally indicates success.
- **let programName = "FirstSteps"** creates an immutable binding. The value cannot change after initialization. We will explore let versus var in detail in Chapter 4.
- **var count = 3** creates a mutable variable that can be reassigned later.
- **Console.WriteLine("Program: $programName")** demonstrates sigil-free string interpolation. The dollar sign followed by an identifier name inserts the value directly without requiring curly braces or format specifiers [25].
- **for i in 1 .. count + 1** iterates over a range from 1 up to but not including count + 1, yielding values 1, 2, and 3. Range syntax uses double dots (..) for half-open intervals.
- **getGreeting("Reader")** calls a user-defined function with a string parameter. Note that parameter types follow the parameter name rather than preceding it : this is a stylistic choice influenced by Go and Kotlin.

This small program touches on enough concepts to feel real while remaining simple enough to understand completely. As we proceed through subsequent chapters, each of these constructs will be explained in depth with more examples and edge cases. For now, the important takeaway is that G# programs look clean and readable even when combining multiple features, and they build using the same dotnet commands you already know from .NET development.

In the next chapter we will examine the structure of G# source files more carefully: how packages organize code, how imports resolve namespaces, how visibility modifiers control access, and what conventions keep larger codebases maintainable.

# Chapter 3: Language Foundations : Syntax, Packages, and Basic Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Source Files and the Compilation Unit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Package Declarations and Organization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Imports : Bringing Namespaces into Scope

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Comments : Line, Block, and Documentation Comments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Visibility and Access Modifiers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Coding Conventions and Style Guidelines

# Chapter 4: Variables, Constants, Types, and Values

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Declarations : var, let, const and When to Use Each

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Primitive Types : Numbers, Booleans, Characters, Strings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Numeric Widths and Aliases : int32 vs int and Why It Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Nullability : nil, T?, and Safe Access Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Operators and Precedence : Arithmetic, Comparison, Logical, Bitwise

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Expressions and Statement Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

# Chapter 5: Control Flow : Branching, Looping, and Pattern Matching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Conditional Statements and Expressions : if/else Rules and Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Nullable Unwrapping : if let, guard let, while let

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Loops : for, while, do-while, range Iteration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Switch Statements and Expressions : No Fallthrough Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Pattern Matching : Types, Lists, Properties, Relational, Combinators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Smart Casts and Type Narrowing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

# Chapter 6: Functions : The Building Blocks of Behavior

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Function Declarations : func Syntax and Signatures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Parameters : Named, Positional, Variadic, and Default Values

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Return Values and Expression Bodies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Closures and Captured Variables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Lambdas and Arrow Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Higher-Order Functions and Functional Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

# Chapter 7: Data Structures : Structs, Classes, and Collections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Structs : Value Types with Instance Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Classes : Reference Types, Identity, and Inheritance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Data Types : Structural Equality, Deconstruction, with-Copy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Interfaces : Contracts, Default Methods, Generic Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Arrays, Slices, and Rectangular Arrays

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Maps, Sequences, and CLR Collections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

# Chapter 8: Error Handling and Resource Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Exceptions in G# : CLR Model and throw as Expression

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Try/Catch/Finally : Catching Specific Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Custom Exception Types : When and How to Define Them

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Defer Statements : Guaranteed Scope Cleanup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Using Declarations : RAII for Disposable Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Error Handling Patterns : When to Throw vs Return nil

# Chapter 9: Generics, Type Parameters, and the Advanced Type System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Generic Functions and Types : Syntax and Instantiation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Constraints : any, comparable, Interface Bounds, Base Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Variance : in/out Covariance and Contravariance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Inline Structs : Single-Field Value Wrappers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Discriminated Unions via Payload-Bearing Enums

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Advanced Patterns : Type Tests, Conditional Logic

# Chapter 10: Concurrency : From Async/Await to Structured Concurrency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Asynchronous Functions : async func and Task[T]

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Await Expressions and Suspension Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Structured Concurrency with scope

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Async Streams : async sequence[T] and await for

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Go-Flavored Concurrency : go, chan T, select (Opt-In)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Concurrency Safety : Shared State, SyncMap, Locks

# Chapter 11: .NET Interoperability : Using the Full Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Calling CLR APIs : Types, Constructors, Properties, Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Static Imports and Namespace Aliases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Extension Functions : Go-Style Receivers and LINQ

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Events and Delegates : +=/-= Wiring Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Attributes : Kotlin-Style Syntax and Targets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Native Interop : P/Invoke, LibraryImport, Function Pointers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

# Chapter 12: Development Workflow : Building, Testing, Debugging, and Refactoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Project Structure and .gsproj Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Build Options : Targets, TFMs, References, Deterministic Builds

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Testing with xUnit/NUnit/MSTest from G#

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Debugging : Breakpoints, Watch Expressions, PDBs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Profiling and Performance Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## The cs2gs Migration Tool : Moving C# to G#

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

# Chapter 13: Architecture, Idioms, and Production Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Effective G# : Idiomatic Patterns from the Official Guide

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Package Design : Small Packages, Explicit APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Design Patterns in G# : Repository, Factory, Observer, Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Performance Characteristics : JIT, GC, IL Generation, Profiling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Security Considerations : Sandboxing, Input Validation, AOT

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Deployment : Single-File Publishing, WebAssembly, Containerization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

# Conclusion : The Future of G# and Your Next Steps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## What We Have Covered : Key Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## G# Today vs Tomorrow : Honest Assessment of Maturity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## When to Choose G# : Decision Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Contributing and Community Engagement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

## Where to Go From Here : Resources and Next Steps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.

# References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/gprogramminglanguageunlocked.