High-Performance Go Training - Highly Customized

As part of our DNA, we strive to provide the highest quality training available in the marketplace for Go. Ardan Labs instructors are thought leaders in their fields of expertise with decades of field experience working with some of the major corporations in the world.

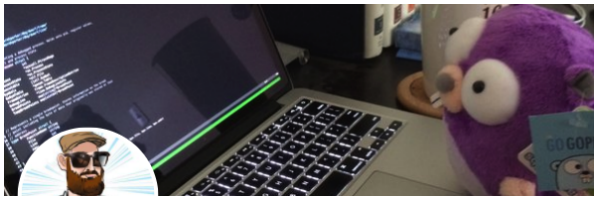
William “Bill” Kennedy

Bill has been developing software professionally for more than 30 years. In 2013 he became a pioneer using Go and now has trained over 25,000 engineers worldwide. He also is the author of the book Go in Action, Ultimate Go Notebook, and is the main contributor to our blog.

Miki Tebeka

Miki is a thought leader and developer with 25 + years of cutting edge software development experience. He’s taught many workshops on various technical subjects all over the world at companies such as AT&T, Oracle, Dropbox, J.P. Morgan, and others. Miki is involved in open source, both in the Go and Python worlds. He has several open source projects of his own and contributed to many others including Go & Python. Miki wrote "Forging Python", "Go Brain Teasers" and "Python Brain Teasers", he's a LinkedIn Learning author, and speaks at various conferences.

Derek Parker



Derek is the creator of Delve, the Go programming language debugger. He has been developing software for over a decade and has traveled the world speaking at conferences and teaching workshops about debugging. After years of experience, Derek has put together the Ultimate Go Debugging workshop to level up developers of all skill levels in the art of debugging code.

Go Training Curriculum

- Bill Kennedy
  - Ultimate Go
  - Ultimate Go: Service with Kubernetes
  - Ultimate Go: Advanced Engineering
- Miki Tebeka
  - Practical Go for Developers
  - Performance Go for Developers
  - Practical Go Services
  - Custom Practical Go Training
- Derek Parker
  - Ultimate Go: Debugging

Ultimate Go: Foundations

The **Ultimate Go Foundations** class has been designed over the past 8 years and goes beyond just being a Go language class. There will be very little time spent on specific Go syntax. Our time will be spent learning how to read and comprehend Go code with a big focus on “if performance matters” then these things matter.

Course Outline

Part 1: Language Mechanics / Semantics - We will talk about micro-level code design, semantics, guidelines, and idioms.

- Data-Oriented Design
- Memory Semantics and Allocations
- Mechanical Sympathy with Hardware and Runtime
- Data Semantics

Part 2: Polymorphism - We will talk about the decoupling mechanics and semantics of the language. This includes interfaces and compositional design.

- Polymorphism with Interfaces
- Decoupling Mechanics / Semantics
- Compositional Design

Part 3: Multi-Threaded Programming - We will talk about all the foundational considerations required for writing multi-threaded software. We will explore how to manage concurrency and parallelism.

- Scheduler Semantics
- Goroutines
- Data Races
- Synchronization / Orchestration

Part 4: Tooling - We will talk about how the profiler and trace tooling work in Go by writing and exploring multi-threaded programs.

- CPU and Memory Profiling
- Tracing Programs
- Micro/Macro level profiling

Ultimate Go: Service with Kubernetes

The **Ultimate Go Service with Kubernetes** class has been designed over the past 5 years and goes beyond just being a Go service class. This course teaches you how to build production-level services in Go, leveraging the power of Kubernetes.

From the beginning of the course, you will pair-program with your instructor Bill Kennedy as he walks you through the **design philosophies, architectural decisions**, and **best practices** as they apply to engineering a production-ready Go service.

With each new feature that is added to the service, you will learn how to deploy and manage the Kubernetes environment used to run the service. Throughout the class, the code being worked on is pushed to a repository for personal access and review.

Course Outline

- Introduction
- Modules
- Kurbernetes
- Initial Service Design
- HTTP Routing Basics
- Web Framework
- Middleware
- JSON Web Tokens (JWT)
- Authentication / Authorization
- Database Support
- Database Migrations and Seeding
- Business Packages
- Testing Data Business Packages
- REST API
- Open Telemetry
- Review Service Project
- Beyond The Branch

Ultimate Go: Advanced Engineering

The Ultimate Go Advanced Engineering class has been designed over the past year and will teach advanced Go concepts by building a reference implementation of a blockchain in Go! The goal of this class is to share how to code complex engineering tasks required to build a blockchain technology.

From the beginning, you will pair-program with the instructor, walking through the design philosophies and guidelines used to engineer the code. Throughout the class, you will learn more about Go and the advanced engineering features of the language.

Course Outline

- Introduction
- Blockchain Fundamentals
- Genesis
- Digital Signatures
- Database
- Cryptographic Audit Trails
- Memory Pools
- Accepting Signed Transactions
- Mining
- Storage
- Peer to Peer Networking
- Wallets

Practical Go for Developers

The **Practical Go for Developers** training class helps experienced engineers get effective with the Go programming language and learn by writing code to solve common programming tasks.

Syllabus

Each session is 4 hours long.

Session 1: Getting Started

- Strings & formatted output
  - What is a string?
  - Unicode basics
  - Using fmt package for formatted output
- Calling REST APIs
  - Making HTTP calls with net/http
  - Defining structs
  - Serializing JSON
- Working with files
  - Handling errors
  - Using defer to manage resources
  - Working with io.Reader & io.Writer interfaces

Session 2: Interfaces & Panics

- Sorting
  - Working with slices
  - Writing methods
  - Understanding interfaces
- Catching panics
  - The built-in recover function
  - Named return values
- Processing text
  - Reading line by line with bufio.Scanner
  - Using regular expressions

Session 3: Concurrency

- Distributing work
  - Using goroutines & channels
  - Using the sync package to coordinate work
- Timeouts & cancellation
  - Working with multiple channels using select
  - Using context for timeouts & cancellations
  - Standard library support for context

Session 4: Project Engineering

- Testing your code
  - Working with the testing package
  - Using testify for easier testing
  - Managing dependencies with go mod
- Structuring your code
  - Writing & using sub-packages
- Writing an HTTP server
  - Writing handlers
  - Fancier routing with gorilla/mux
- Adding metrics & logging
  - Using expvar for metrics
  - Using the log package and a look at user/zap
- Configuration patterns
  - Reading environment variables and a look at external packages
  - Using the flag package for command line processing

What You'll Need

You need to know how to program and core concepts of working with computers (such as files, HTTP, memory management…)

You should have the following installed on your computer prior to the workshop:

- Go SDK from https://golang.org/dl/
- An IDE, either VSCode with Go extension or GoLand
- Git

Performance Go for Developers

The **Performance Go for Developers** training class is aimed for experienced Go developers who'd like to extend their knowledge.

Syllabus

Each session is 4 hours long.

Session 1: Interfaces

- Writing custom errors and the case of the nil error
- Stronger abstractions with small interfaces and using type assertions
- Customizing behavior by implementing interfaces in the standard library
- Reducing code size by using generics

Session 2: Performance Optimization

- Should you be optimizing? Questions you should ask yourself before you start
- Going faster with CPU optimization
    - Writing benchmarks
    - Profiling
    - Tips & Tricks
- Reducing memory usage
    - Benchmarking memory & tracing GC
    - Writing GC friendly code
    - A look at the execution tracer

Session 3: HTTP Servers

- Writing middleware for common tasks (such as authentication)
- Advanced JSON serialization
    - Working with custom types
    - Missing vs zero values
- Streaming responses for large dynamic data
- Securing your server

Session 4: Project Engineering

- Utilizing workspaces to developing two dependant modules (such as application & logging library)
- Using build tags for platform dependent code
- Task automation with "go generate", code that writes code
- Avoiding mocks by spinning services
    - A look at testcontainers

What You'll Need

You need to have working knowledge of Go and HTTP. You should be able to work in the terminal (command line).

You should have the following installed on your computer prior to the workshop:

- Go SDK from https://golang.org/dl/ or via your package manager
- An IDE, either VSCode with the Go extension or GoLand
- Git
- Docker

Practical Go Services

This workshop is for Go developers who improve writing services and learn industry best practices. During this workshop we'll develop a web service and implement the topics we discuss.

Syllabus

Each session is 4 hours long.

Session 1: Writing a Web Service

We'll write a REST web service and discuss routing and middleware. We'll also discuss serialization and see examples with JSON, including custom serialization.

Session 2: Storage & Health

We see how to work with relational databases, and key value databases. We'll discuss and implement effective logging & monitoring.

Session 3: Testing & Performance

We'll test our web handlers both internally and by spinning a web server. We'll also look at how to measure performance, find bottlenecks and fix them.

Session 4: Securing Your Application

We'll cover security best practices, look at OWASP top 10 and harden our web service.

What You'll Need

You need a working knowledge of Go, core concepts of working with computers (such as files, HTTP, memory management…). You should be familiar with the command line and how to use it.

You should have the following installed on your computer prior to the workshop:

- Go SDK from https://golang.org/dl/
- An IDE, either VSCode with Go extension or GoLand
- Git
- Docker

We also recommend that you'll read the following before the workshop:

- A Tour of Go
- Effective Go

Ultimate Go Debugging

The **Ultimate Go Debugging** course is designed for developers who want to learn how to be proficient debugging Go software using the Delve debugger. Developers who have never used a debugger before to those who have experience with Delve, GDB or LLDB, will gain important and practical skills and knowledge. Everyone who takes this class will walk away with practical information, tips, and tricks which can be used in their day-to-day development workflow.

Your team can take the full course or just the Bootcamp versions:

- Ultimate Go Debugging Bootcamp (Days 1-2-3)
- Advanced Ultimate Go Debugging Bootcamp (Days 4-5)
- Ultimate Go Debugging Bootcamp Training (All 5 Days)

Course Outline

Day 1: Introduction, Getting Started, Navigating your program, Inspecting program state, Changing program state.

Day 2: Advanced program navigation, Tracing programs, Examining core dumps (post mortem debugging), Record and Replay debugging.

Day 3: Scripting Delve, Using the JSON-RPC API, Remote Debugging.

Day 4: Debugging containerized app, Debugging on Kubernetes

Day 5: Using pprof profiling tools, Using perf on Go binaries, Deep dive into Delve and Go internals

Custom Practical Go Training and Boot Camps By Miki Tebeka

Our custom hands-on Go training contains several modules that you can choose from to create a syllabus tailored to your needs. Feel free to reach out if you have a need that is not converted by the following topics - we have a lot of experience with Go and development at large.

These classes are led by Miki Tebeka. Miki has more than 25 years of experience developing software for clients, with 11 of those years writing code in Go. Miki is the author of several books, a LinkedIn Learning author and a very busy teacher. Miki teaching style is “hands on” - you’re going to write a lot of coding during these sessions.

- Each module takes between 3.5 and 4 hours to complete
- Some modules can be turned into 2 hour blocks
- Custom Training is taught Monday through Thursday for 4 hours each day
- Boot Camps are 4 2-hour sessions
- Boot Camp 1: Data Oriented - Structs & Methods + Interfaces
- Boot Camp 2: Concurrency: Goroutines & channels + sync & sync/atomic + timeout & cancellation
- Boot Camp 3: Testing
- Boot Camp 4: HTTP: HTTP Clients + HTTP Servers
- Boot Camp 5: gRPC: Protocol Buffers + gRPC Basics + gRPC Topics
- Boot Camp 6: Security: Writing Secure Go Code + Static Analysis

Custom Practical Go Training and Boot Camps

Modules

1. Variables, Conditions & Loops

a. Variable declaration & scope
b. Numerical types and conversion
c. Loops & conditionals

2. Collections Types

a. Strings & Unicode
b. Slices, uses and internals
c. Maps & zero values

3. Functions

a. Defining functions
b. Calling convention
c. Functions as first-class objects
d. Named return argument

4. Error Handling

a. Using defer to manage resources
b. Error handling
c. Defining your own errors
d. Handling panics

5. Structs & Methods

a. Define your own types with structs
b. Methods & receivers
c. Embedding structs

6. Interfaces

a. Defining interfaces
b. Implementing interfaces
c. A tour of interfaces in the standard library

7. Goroutines & Channels

a. Concurrency with goroutines
b. Communication with channels

8. The “sync” and “sync/atomic” packages

a. sync.Mutex
b. sync.WaitGroup and errgroup
c. “sync/atomic”

Custom Practical Go Training and Boot Camps

Modules

9. Logging & Metrics

a. Metrics & the “expvar” package
b. Using the built-in logger
c. Using the “zap” logger

10. Timeouts & Cancellations

a. The “select” statement
b. Using context.Context for timeout & cancellation
c. Avoiding goroutine leaks
d. Built-in support for context.Context

11. Project Management

a. Dependency management with “go mod”
b. Project structure
c. Documenting your code

12. Testing

a. Writing tests
b. Table driven tests
c. Fuzzing

13. HTTP Clients

a. GET requests
b. JSON encoding & decoding
c. POST requests with data

14. HTTP Servers

a. Writing handlers & testing them
b. Routing
c. Writing middleware

15. Protocol Buffers

a. Writing definition file
b. Generating code
c. Serialization
d. Time & enums

16. gRPC Basics

a. Defining services
b. Generating code
c. Writing a server
d. Writing a client

Custom Practical Go Training and Boot Camps

Modules

17. gRPC Topics *

a. Streaming
b. Using TLS
c. Writing interceptors (middleware)

18. Writing Secure Go Code

a. The security mindset
b. A look at OWASP top 10
c. Using “gosec”
d. Safely handling input & output
e. Authentication
f. Security configuration

19. Static Analysis

a. Using linters and look at common ones
b. Writing your own tools with x/tools/go/analysis

20. The Empty Interface

a. The empty interface - interface{}
b. Uses for the empty interface
c. Using reflection

21. Iota & Enums

a. Understanding iota
b. Enums
c. String representation for enums
d. Bitmasks

22. Working with Files

a. The os.File type
b. The io.Reader & io.Writer interface
c. In memory readers & writers
d. Iterating over lines
e. The “encoding/csv” package

23. Working with Relational Databases

a. The “database/sql” package
b. Writing queries
c. Reading results
d. Timeouts
e. Embedding SQL with the embed package

Custom Practical Go Training and Boot Camps

Modules

24. Calling non-Go Code

a. Using os/exec to call external commands
b. Using cgo to interface C code and shared libraries

25. Working with Time

a. The built-in “time” package
b. Formatting & parsing time
c. Time zone conversion
d. JSON encoding

26. Configuration

a. Parsing command line arguments
b. Reading environment variables
c. A discussion on configuration file format
d. Using “viper” to read configuration

27. Networking

a. Writing a TCP client
b. Writing a TCP server
c. UDP client & server
d. Using unix domain sockets

28. Sorting

a. The “sort.Interface” interface
b. Sorting slices
c. Sorting by keys

29. Building

a. Using “go generate” to generate code
b. Injecting version into executables
c. Conditional building with build tags
d. Cross compiling
e. Generating static executables

30. Regular Expressions

a. The “regexp” package
b. strings vs []byte
c. Regular expressions syntax
d. Parsing text