5-Day Go (Golang) Fresher Training Course

Course Overview: A hands-on introduction to Go programming designed for beginners. This course progresses from fundamentals to building a complete real-world application (capstone project).

Target Audience: Programmers new to Go

Duration: 5 days (6-8 hours per day)

Prerequisites: Basic programming knowledge (any language)

DAY 1 Go Fundamentals & Environment Setup

Morning (3-4 hours)

1.1 Introduction to Go

- What is Go and why use it?
    - Concurrency model
    - Performance characteristics
    - Use cases
- Installation & workspace setup
    - Installing Go SDK
    - Setting up GOPATH/GOMOD
    - IDE setup (VS Code with Go extensions)

1.2 Basic Syntax & Data Types

- Hello World - first program
- Variables and Constants
    - Declaration methods (var, :=, const)

    - Type system (static typing)
- Primitive Data Types
    - Integers (int, int64, uint32, etc.)
    - Floating-point (float32, float64)
    - Booleans
    - Strings and runes
- Type conversions and casting

1.3 Operators & Control Flow

- Arithmetic, logical, and comparison operators
- if/else statements
- switch statements
- for loops
    - Traditional loops
    - Range loops
    - Break and continue

Afternoon (3-4 hours)

1.4 Functions (Part 1)

- Function declaration and syntax
- Parameters and return types
- Multiple return values (error handling pattern)
- Named return values
- Variadic functions
- Defer statements

1.5 Hands-on Lab

Exercise 1.1: Calculator Program

- Build a command-line calculator

- Implement: add, subtract, multiply, divide
- Input validation
- Error handling basics

Exercise 1.2: Temperature Converter

- Convert between Celsius, Fahrenheit, Kelvin
- Multiple functions with different signatures

DAY 2 Data Structures & Methods

Morning (3-4 hours)

2.1 Composite Data Types

- Arrays
  - Fixed-length arrays
  - Indexing and iteration
- Slices
  - Dynamic arrays
  - Slice operations (append, slice, copy)
  - Slice internals (capacity vs length)
- Maps
  - Key-value pairs
  - Map operations (add, delete, lookup)
  - Checking key existence

2.2 Structs & Custom Types

- Struct definition and initialization
- Field access and modification
- Struct composition and embedding
- Type aliases

2.3 Pointers

- Pointer basics (*T and &x)
- Dereferencing
- Pointer receivers
- Nil pointers and safety

Afternoon (3-4 hours)

2.4 Methods & Receivers

- Method syntax
- Value receivers vs. pointer receivers
- Method chaining
- When to use pointers vs. values

2.5 Hands-on Lab

Exercise 2.1: Student Management System

- Create Student struct (name, ID, grades)
- Methods for:
  - Calculate average grade
  - Add new grade
  - Display student info
- Using slices to manage multiple students

Exercise 2.2: Bank Account System

- Account struct with balance
- Methods: Deposit, Withdraw, CheckBalance
- Validate transactions

DAY 3 Error Handling & Interfaces

Morning (3-4 hours)

3.1 Error Handling in Go

- The error interface
- Creating custom errors
- Wrapping errors
- Error handling patterns
- panic and recover

3.2 Interfaces

- Interface definition and implementation
- Implicit interface satisfaction
- Empty interface (interface{})
- Type assertions and type switches
- Interface composition

3.3 Packages & Modularity

- Package structure
- Public/private identifiers (capitalization)
- Package management (go mod)
- Importing packages
- Creating reusable packages

Afternoon (3-4 hours)

3.4 Concurrency Fundamentals

- Goroutines (lightweight threads)
- Creating and launching goroutines
- The `go` keyword
- Waitgroups for synchronization
- Goroutine communication patterns

3.5 Channels

- Channel basics
- Send and receive operations
- Channel buffering
- Range over channels
- Closing channels

3.6 Hands-on Lab

Exercise 3.1: Shape Area Calculator

- Define Shape interface with Area() method
- Implement: Rectangle, Circle, Triangle
- Create function that works with any Shape
- Practice interface-based design

Exercise 3.2: Concurrent File Processor

- Read multiple files concurrently using goroutines
- Use channels for communication
- Aggregate results from multiple goroutines
- Demonstrate synchronization with WaitGroup

DAY 4 Real-world Patterns & Web Basics

Morning (3-4 hours)

4.1 JSON & Data Serialization

- JSON marshaling and unmarshaling
- Struct tags for field mapping
- Custom JSON encoding/decoding
- YAML basics

4.2 File I/O & System Operations

- Reading files (simple and buffered)
- Writing files
- Directory operations
- Working with file paths

4.3 HTTP & REST Basics

- net/http package
- HTTP client basics
- Making HTTP requests
- Response handling and JSON parsing

Afternoon (3-4 hours)

4.4 Building Web Services

- HTTP server basics
- Handlers and routing
- Parsing query parameters and form data
- Returning JSON responses
- Middleware patterns

4.5 Testing & Debugging

- Testing conventions (*_test.go files)
- Writing unit tests (testing.T)
- Table-driven tests
- Debugging with delve
- Logging best practices

4.6 Hands-on Lab

Exercise 4.1: Weather API Client

- Fetch weather data from public API
- Parse JSON response
- Display formatted output
- Error handling for API calls

Exercise 4.2: Simple REST API

- Create a TODO API with:
  - GET /todos (list all)
  - POST /todos (create new)
  - GET /todos/{id} (get single)
  - DELETE /todos/{id} (delete)
- Use in-memory storage
- Return JSON responses
- Implement basic error handling

🎯 DAY 5: Capstone Project - Blog API Server

Morning (1-2 hours) - Project Introduction & Planning

5.1 Project Requirements

Students will build a RESTful API for a simple blog system with the following features:

Core Features:

- Create, read, update, delete blog posts
- Comment system on posts
- User authentication (basic)
- Search functionality
- Pagination

Technical Requirements:

- Use structs for data models

- Implement interfaces for extensibility
- Use goroutines for concurrent operations
- Proper error handling
- Unit tests for critical functions
- JSON serialization
- RESTful API design

Afternoon (4-6 hours) - Implementation & Presentation

5.2 Project Structure

```
blog-api/ ├── main.go # Entry point ├── models/ │ ├── post.go # Post struct and methods │ ├── comment.go # Comment struct │ └── user.go # User struct ├── handlers/ │ ├── post_handler.go # POST endpoints │ └── comment_handler.go ├── storage/ │ └── memory.go # In-memory storage ├── middleware/ │ └── auth.go # Authentication ├── tests/ │ └── *_test.go # Unit tests ├── go.mod # Module definition └── README.md # Documentation
```

5.3 Required Endpoints

```
Posts: POST /api/posts Create post GET /api/posts List posts (with pagination) GET /api/posts/{id} Get single post PUT /api/posts/{id} Update post DELETE /api/posts/{id} Delete post GET /api/posts/search Search posts Comments: POST /api/posts/{id}/comments Add comment GET /api/posts/{id}/comments List comments DELETE /api/comments/{id} Delete comment Authentication: POST /api/auth/register Register user POST /api/auth/login Login and get token
```

5.4 Learning Outcomes

Through this project, students will demonstrate:

1. Data Structures - Proper use of structs and slices
2. Methods & Receivers - Implementing behavior on data
3. Interfaces - Abstraction and extensibility
4. Error Handling - Robust error management
5. HTTP Handling - Building REST APIs
6. JSON Serialization - Working with JSON data
7. Concurrency Basics - Using goroutines appropriately

8. Testing - Writing unit tests
9. Code Organization - Logical package structure
10. Best Practices - Idiomatic Go code

5.5 Bonus Features (Optional Enhancements)

- Database integration (SQLite or PostgreSQL)
- Rate limiting
- CORS handling
- Request validation
- Comprehensive logging
- Docker containerization
- API documentation (Swagger/OpenAPI)
- User permissions system

5.6 Project Submission & Presentation

Deliverables:

- Complete source code on GitHub
- README with setup instructions
- Example API requests (Postman collection or cURL examples)
- At least 5 unit tests with >80% code coverage
- Brief presentation (10 minutes):
    - Architecture overview
    - Key implementation decisions
    - Challenges faced and solutions
    - Demo of the running API

Daily Schedule Template

Example Day Schedule (8 hours):

Time	Activity	Duration
09:00 - 10:30	Lecture & Concept Explanation	90 min
10:30 - 10:45	Break	15 min
10:45 - 12:30	Live Coding Demo & Discussion	105 min
12:30 - 13:30	Lunch	60 min
13:30 - 15:00	Hands-on Lab Exercise	90 min
15:00 - 15:15	Break	15 min
15:15 - 17:00	Continued Lab & Q&A	105 min

Resources & Tools

Essential Tools

- Go SDK (1.21+)
- VS Code with Go extension
- Git for version control
- Postman or curl for API testing

Recommended Packages

- Standard library (net, http, json, io, os)
- github.com/google/uuid (UUID generation)
- github.com/joho/godotenv (Environment variables)
- Testing: standard testing package

Learning Resources

- Effective Go
- Go Tour
- Go Documentation
- Go Code Review Comments

Assessment Criteria

Component	Weight
Daily Exercises	40%
Capstone Project	60%

Daily Exercises Evaluation

- Code correctness and functionality
- Error handling
- Code quality and style
- Test coverage

Capstone Project Evaluation

- Completeness of features
- Code organization and structure
- Error handling and edge cases
- API design and usability
- Documentation
- Presentation quality

Success Metrics

By the end of this course, freshers should be able to:

- Write basic to intermediate Go programs
- Use Go's type system effectively
- Build REST APIs with proper error handling
- Understand and use goroutines and channels
- Write unit tests for Go code
- Organize code into packages
- Read and understand Go documentation
- Build a complete web application from scratch

Notes for Instructors

- **Pace:** Adjust based on student understanding; Day 4-5 can run into Day 6 if needed
- **Hands-on Focus:** Coding exercises should take 50%+ of class time
- **Debugging:** Teach debugging early; use delve or IDE debugger
- **Best Practices:** Emphasize idiomatic Go from the start
- **Community:** Introduce Go community resources and best practices
- **Iteration:** Have students refactor exercises to improve code quality
- **Group Work:** Consider pairing for capstone project to enhance learning