**Go Programming – Detailed Course Outline (With Copilot Integration)**

**⏳ Duration:** 5 Days (40 hours)  
**👩‍🏫 Mode:** Hands-on, Instructor-led  
**🎯 Audience:** Developers aiming to learn Go from basic to advanced, including concurrency, cryptography, and microservices

**✅ Course Prerequisites**

* Basic knowledge of programming concepts (variables, loops, functions)
* Familiarity with terminal/CLI usage
* Optional: Exposure to C, Python, or Java
* GitHub account to use **GitHub Copilot**

**🛠️ Lab Requirements**

* **Software:**
  + Go (latest stable version)
  + Git & GitHub CLI
  + Visual Studio Code with Go extension
  + **GitHub Copilot plugin for VS Code**
  + Docker (for Kafka, Vault labs)
  + Postman for API testing
* **Environment:**
  + Internet access to fetch Go modules & use Copilot
  + Kafka and Vault setup via Docker Compose
  + Test database (SQLite or PostgreSQL) for GORM labs

**🎓 Learning Outcomes**

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

* Understand Go architecture and memory model
* Write idiomatic, concurrent Go programs
* Build modular and testable packages
* Use Go for systems, networking, and microservice development
* Implement secure, resilient microservices using external packages and patterns
* Use **GitHub Copilot** effectively for code suggestions, testing, refactoring, and documentation

**📅 Day-wise Course Breakdown with TOC and Copilot Integration**

**Day 0 Go Environment Setup Checklist**

**1. ✅ Install Go (Golang)**

* **Official site: https://go.dev/dl/**
* **Download and install the latest stable version for your OS.**

**Verify installation:**

**bash**

**go version**

**# Output example: go version go1.22.0 linux/amd64**

**Check environment variables:**

**bash**

**go env**

**Ensure:**

* **GOPATH is set (default is $HOME/go)**
* **GOROOT is set correctly (optional for most setups)**
* **PATH includes $GOROOT/bin and $GOPATH/bin**

**2. ✅ Folder Structure and Workspace Setup**

**Create Go workspace folder:**

**bash**

**mkdir -p $HOME/go/src/github.com/<your-username>**

**Set GOPATH in .bashrc, .zshrc, or environment:**

**bash**

**export GOPATH=$HOME/go**

**export PATH=$PATH:$GOPATH/bin**

**3. ✅ Install a Code Editor / IDE**

**✅ Recommended IDEs:**

| **IDE** | **Download Link** |
| --- | --- |
| **VS Code** | **https://code.visualstudio.com/** |
| **GoLand** | [**https://www.jetbrains.com/go/**](https://www.jetbrains.com/go/) |

**4. ✅ Install VS Code Extensions (if using VS Code)**

**text**

**- Go (by Go Team at Google) [golang.go]**

**- Code Runner (optional)**

**- Delve Debugger (included in Go extension)**

**- Go Test Explorer (optional)**

**- GitLens (for Git features)**

**- REST Client (optional for testing APIs)**

**Install via command:**

**bash**

**code --install-extension golang.go**

**5. ✅ Initialize a Go Module**

**bash**

**go mod init github.com/<your-username>/<project-name>**

**6. ✅ Install Go Tools**

**Run the following after installing the Go extension (VS Code prompts this as well):**

**bash**

**go install golang.org/x/tools/gopls@latest # Language server**

**go install github.com/go-delve/delve/cmd/dlv@latest # Debugger**

**go install honnef.co/go/tools/cmd/staticcheck@latest # Linter**

**go install github.com/golangci/golangci-lint/cmd/golangci-lint@latest**

**7. ✅ Test Go Commands**

**Create a test file:**

**go**

**// hello.go**

**package main**

**import "fmt"**

**func main() {**

**fmt.Println("Hello, Go World!")**

**}**

**Run:**

**bash**

**go run hello.go**

**Build:**

**bash**

**go build hello.go**

**./hello**

**Test:**

**Create test file:**

**go**

**// hello\_test.go**

**package main**

**import "testing"**

**func TestMainOutput(t \*testing.T) {**

**expected := "Hello, Go World!"**

**if expected != "Hello, Go World!" {**

**t.Errorf("Expected %s", expected)**

**}**

**}**

**Run tests:**

**bash**

**go test -v**

**8. ✅ Format & Lint**

**bash**

**go fmt ./... # Format all files**

**go vet ./... # Find suspicious constructs**

**staticcheck ./... # Advanced linting**

**9. ✅ (Optional) Git Setup**

**bash**

**git init**

**touch .gitignore**

**# .gitignore example**

**\*.exe**

**\*.out**

**\*.log**

**vendor/**

**10. ✅ (Optional) Go Project Tools**

* **Gin (Web framework): go get -u github.com/gin-gonic/gin**
* **Cobra (CLI apps): go install github.com/spf13/cobra-cli@latest**
* **Testify (testing): go get github.com/stretchr/testify**

**✅ Summary: Final Verification Checklist**

| **Item** | **Command** | **Should Output** |
| --- | --- | --- |
| **Go installed** | **go version** | **go version go1.xx.xx** |
| **Environment configured** | **go env** | **Shows GOPATH, GOROOT** |
| **Run a Go file** | **go run hello.go** | **Hello, Go World!** |
| **Build a binary** | **go build hello.go && ./hello** | **Hello, Go World!** |
| **Run tests** | **go test -v** | **PASS** |
| **VS Code extension installed** | **Go Extension** | **IntelliSense, Debug, Linting** |
| **Format and Vet** | **go fmt, go vet** | **No output = success** |

**🟩 Day 1: Go Fundamentals**

**Topics Covered:**

* **Getting Started with Go**
  + Compilation model, runtime
  + Identifiers, Constants, Variables
  + Operators, Assignments, Pointers, Strings
* **Flow Control**
  + if-else, switch, loops (for, range), break/continue
* **Functions**
  + Parameters, returns, defer, closures, recursion
* **Working with Data**
  + Arrays, slices, maps, structs (including embedded, tagged, recursive)

**Copilot Usage:**

* Generate control flow logic
* Auto-complete slice/map manipulations
* Assist in struct field access & initialization
* Suggest recursive functions and closures

**Labs:**

* Implement custom struct-based JSON marshaling
* Use Copilot to auto-generate boilerplate functions for slice/map operations

**Outcome:**  
Understand Go's core syntax, data structures, and control constructs, and use Copilot to write cleaner and faster code.

**🟨 Day 2: Interfaces, Methods, and Concurrency**

**Topics Covered:**

**Quiz (15 min)**

* **Methods and Interfaces**
  + Method declarations, receivers, method sets
  + Interfaces (empty, embedded), type assertions, type switches
* **Concurrency with Goroutines & Channels**
  + Parallelism vs concurrency
  + Goroutines, Lambdas, Channels (buffered/unbuffered)
  + Select, WaitGroups, timers, tickers

**Copilot Usage:**

* Generate interface implementations
* Suggest common goroutine/channel patterns
* Fix race conditions with WaitGroups and locks

**Labs:**

* Create interfaces and implement polymorphic behavior
* Build goroutine-based concurrent job runner with Copilot suggestions

**Outcome:**  
Gain hands-on experience with Go’s interface system and concurrency model using channels and goroutines, guided by Copilot where appropriate.

**🟧 Day 3: Testing, Packages, File I/O, Synchronization**

**Topics Covered:**

* **Packages & Testing**
  + Package layout, exports/imports
  + Unit testing, table-driven tests, benchmarking
* **File I/O**
  + Read/write/copy files, directories
  + JSON/XML/HTML file parsing
* **Error Handling**
  + Panic/recover, error propagation strategies
* **Synchronization**
  + Atomicity, cancellation, work stealing, advanced channels

**Copilot Usage:**

* Auto-generate test cases and table-driven test templates
* Suggest benchmarks and assertions (with testify)
* Recommend best practices for error propagation

**Labs:**

* Write testable Go modules using table tests
* Process file contents (CSV/XML/JSON) using Go I/O
* Synchronize multiple goroutines using advanced channel patterns

**Outcome:**  
Learn modular code development, testing strategies, and robust file/error handling with the help of Copilot to generate tests and refactor code.

**🟦 Day 4: Networking, Cryptography, Memory, Profiling**

**Topics Covered:**

**Quiz (15 min)**

* **Networking**
  + Sockets, TCP, HTTP clients/servers
  + Data serialization (JSON/XML/protobuf)
* **Cryptography**
  + Public/private keys, encryption (AES/RSA), hashing, digital signatures
* **Memory Management**
  + make vs new, unsafe pointers, alignment
* **Tips and Tricks**
  + Profiling, mocking, anonymous structures, method expressions

**Copilot Usage:**

* Suggest client-server code for HTTP/TCP
* Generate encryption/decryption logic
* Guide memory-safe refactors

**Labs:**

* Build TCP chat server and HTTP API with Copilot
* Implement AES encryption/decryption and hashing
* Profile a Go program for memory/cpu using pprof

**Outcome:**  
Build secure and network-aware applications in Go with proper memory profiling and use Copilot to assist in building systems code.

**🟥 Day 5: Advanced Go, Microservices, Kafka, Resilience, Vault**

**Topics Covered:**

* **Advanced Topics**
  + Go idioms, code smells, workspace setup
  + Unicode/text processing, compression, encoding
  + GORM for SQL/NoSQL DBs
* **External Libraries**
  + Testing: testify, ginkgo
  + Functionality: go-set, go-ethereum
  + Web/Microservices: gin, traefik, etv
* **Microservices & Resilience**
  + Kafka (event choreography, command orchestration)
  + Retry logic, circuit breakers, HA with libraries
  + Vault configuration and secrets management
  + Copilot Code Review to understand best practices
* **Communication Formats**
  + JSON, XML, CSV, binary

**Copilot Usage:**

* Scaffold microservice APIs using gin + gorm
* Generate Kafka producer/consumer code
* Recommend retry/circuit breaker wrappers
* Assist in secure secret loading from Vault

**Labs:**

* Implement a microservice using gin + GORM + Kafka + Vault
* Apply SAGA pattern (event-based + command orchestration)
* Add fallback and resiliency using Copilot-aided libraries

**Outcome:**  
Master end-to-end Go microservices development with real-world tools. Use Copilot as a development partner to scaffold, refactor, and secure your production-grade services.

**📦 Deliverables**

* Complete slide deck (PPT)
* Daily lab workbook with Copilot usage
* Sample Go projects per day with README
* Docker Compose setup for Kafka/Vault
* GitHub Copilot usage cheatsheet
* Quiz & evaluation per day

**🎯 Final Course Summary**

| **Day** | **Focus Area** | **Copilot Use** |
| --- | --- | --- |
| 1 | Go basics, functions, structs | Code generation, recursive logic |
| 2 | Interfaces and concurrency | Interface impl, goroutines |
| 3 | Testing, File I/O, Error handling | Table tests, benchmarks, I/O |
| 4 | Networking, cryptography, memory | TCP/HTTP, encryption, profiling |
| 5 | Microservices, Kafka, Vault, resilience | API scaffolding, retry logic |