Advanced Go Programming (Go Language)

Prerequisites: Knowledge of Go Programming

Day 1: Advanced Go Concepts

Objective: Learn and apply advanced Go features like structs, methods, interfaces, and error handling patterns.

- **Morning Session (Theory)**
    1. **Structs and Methods**
        - Defining structs and their fields
        - Creating methods for structs (receiver functions)
        - Comparing values and pointers as receivers
    2. **Interfaces**
        - Understanding Go interfaces
        - Defining and implementing interfaces
        - Empty interfaces and type assertions
- **Lab 1: Structs and Methods**
    - Define a struct (Employee) with fields (name, age, salary)
    - Create methods to interact with the struct (e.g., increase salary, display information)
    - Experiment with pointer receivers and value receivers
- **Lab 2: Interfaces Implementation**
    - Create interfaces (e.g., Shape interface with Area() and Perimeter() methods)
    - Implement the interface in different structs (Circle, Rectangle)
    - Use type assertions with an empty interface to process multiple types
- **Afternoon Session (Theory)**
    1. **Error Handling and Custom Errors**
        - Best practices for error handling in Go
        - Creating custom error types
        - Wrapping errors with fmt.Errorf() and using the errors package

        - Using panic and recover for handling critical failures
    2. **Defer, Panic, and Recover**
        - The role of defer in resource management
        - Handling unexpected failures with panic
        - Using recover to gracefully handle panics
- **Lab 3: Custom Errors and Error Handling**
    - Create a program that handles errors gracefully using multiple return values
    - Create custom error types and handle them in different scenarios (e.g., file reading)
    - Experiment with defer, panic, and recover in handling critical failures

---

Day 2: Concurrency and Advanced Topics

Objective: Learn Go's concurrency model, channel communication, and explore testing and optimization techniques.

- **Morning Session (Theory)**
    1. **Goroutines**
        - Introduction to concurrency in Go
        - Creating and running goroutines
        - Synchronization and goroutine lifecycle
    2. **Channels**
        - Understanding unbuffered vs buffered channels
        - Sending and receiving values from channels
        - Channel synchronization patterns (e.g., worker pools)
        - Select statement and timeout with channels
- **Lab 4: Goroutines and Channels**
    - Write a program that runs multiple goroutines concurrently (e.g., a web scraper fetching multiple URLs)
    - Use channels for communication between goroutines

- Implement a worker pool using goroutines and channels

- **Afternoon Session (Theory)**

    1. **Concurrency Patterns**
        - Fan-in and fan-out patterns using channels
        - Using sync.WaitGroup to synchronize goroutines
        - Mutexes and atomic operations for shared state
    2. **Testing and Benchmarking in Go**
        - Writing test cases using Go's testing package
        - Writing benchmarks and profiling code
        - Running tests and understanding test coverage

- **Lab 5: Concurrency Patterns**
    - Implement a fan-in pattern to gather data from multiple goroutines into one channel
    - Use sync.WaitGroup to wait for goroutines to complete
    - Solve shared data access problems using mutexes and atomic operations
- **Lab 6: Writing Tests and Benchmarking**
    - Write unit tests for a function using the testing package
    - Create benchmark tests to measure the performance of a piece of code
    - Use Go's built-in tools (go test, go benchmark) to run tests and analyze performance

---

Wrap-Up & Final Lab

Lab 7: Mini Project – Concurrent Task Processor

- Build a concurrent task processor that uses goroutines and channels to process a series of tasks in parallel
- Implement interfaces to handle different types of tasks
- Add proper error handling, testing, and benchmarks to measure the system's performance