Table of Contents

1. Introduction to Golang

- Go (Golang) is an open-source programming language developed by Google.
- Known for simplicity, efficiency, and concurrency support.
- Designed for scalable and high-performance applications.

---

2. Setup and Running Go

1. Download and Install Go:
    - Download Go.
2. Verify Installation:

   ```
   go version
   ```

3. Run Go Program:

   ```
   go run main.go
   ```

4. Build Executable:

   ```
   go build main.go
   ```

---

3. Basic Go Program Structure

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

- package main – Defines the entry point.
- import “fmt” – Imports the fmt package for I/O.
- func main() – Main function, program execution starts here.

---

4. Variables and Data Types

Variable Declaration

```
var name string = "GoLang"
var age int = 25
isTrue := true  // Short-hand declaration
```

Data Types

Type	Example
string	var s string = “Hello”
int	var i int = 10
float64	var f float64 = 9.8
bool	var b bool = true
byte	var b byte = ‘A’
rune	var r rune = ‘♥’

---

5. Constants

```
const Pi = 3.14
const Greeting = "Welcome to Go"
```

---

6. Control Structures

If-Else

```
if age > 18 {
    fmt.Println("Adult")
} else {
    fmt.Println("Minor")
```

```
}
```

Switch Statement

```
switch day := 3; day {
case 1:
    fmt.Println("Monday")
case 2, 3:
    fmt.Println("Midweek")
default:
    fmt.Println("Other day")
}
```

---

7. Loops

For Loop (No While in Go)

```
for i := 1; i <= 5; i++ {
    fmt.Println(i)
}
```

Infinite Loop

```
for {
    fmt.Println("Running...")
    break
}
```

Range Loop (Iterate Over Arrays/Maps)

```
nums := []int{1, 2, 3}
for index, value := range nums {
    fmt.Println(index, value)
}
```

---

8. Arrays and Slices

Arrays

```
var arr [3]int = [3]int{10, 20, 30}
fmt.Println(arr[0])  // Access element
```

Slices (Dynamic Arrays)

```
nums := []int{1, 2, 3}
nums = append(nums, 4)
fmt.Println(nums)
```

---

9. Maps (Dictionaries)

```
person := map[string]string{
    "name": "John",
    "city": "New York",
}
fmt.Println(person["name"])  // Access value
```

---

10. Functions

```
func add(a int, b int) int {
    return a + b
}

func main() {
    result := add(5, 3)
    fmt.Println(result)
}
```

- Multiple Return Values

```
func divide(x, y int) (int, int) {
```

```
    return x / y, x % y
}
```

---

11. Pointers

```
x := 10
p := &x  // Pointer to x
fmt.Println(*p)  // Dereference to get value
```

---

12. Structs (Custom Types)

```
type Car struct {
    Brand string
    Speed int
}

func main() {
    myCar := Car{"Toyota", 120}
    fmt.Println(myCar.Brand)
}
```

---

13. Methods (Struct Functions)

```
func (c Car) Drive() {
    fmt.Println(c.Brand, "is driving at", c.Speed, "km/h")
}
```

---

14. Interfaces (Abstraction)

```
type Vehicle interface {
    Drive()
}

type Bike struct{}
func (b Bike) Drive() {
    fmt.Println("Bike is driving")
}
```

---

15. Error Handling

```
func divide(x, y int) (int, error) {
    if y == 0 {
        return 0, fmt.Errorf("division by zero")
    }
    return x / y, nil
}
```

---

16. Goroutines (Concurrency)

```
func sayHello() {
    fmt.Println("Hello")
}

func main() {
    go sayHello()  // Run concurrently
    fmt.Println("Main function")
}
```

---

17. Channels (Goroutine Communication)

```
ch := make(chan int)

go func() {
    ch <- 10  // Send data
}()

x := <-ch  // Receive data
fmt.Println(x)
```

---

18. File I/O

```
import (
    "os"
)

func main() {
    file, _ := os.Create("example.txt")
    file.WriteString("Hello, Go!")
    file.Close()
}
```

---

19. Packages

```
import "math"

func main() {
    result := math.Sqrt(16)
    fmt.Println(result)
}
```

---

20. Useful Go Commands

Command	Description
go run file.go	Run Go code
go build file.go	Compile Go code
go fmt	Format Go code
go mod init module_name	Initialize Go module
go test	Run tests
go doc fmt.Println	Show documentation

Tips for Learning Go

- Focus on Simplicity – Go emphasizes readability and simplicity.
- Use Goroutines – Concurrency is Go’s strength.
- Practice Error Handling – Explicit error checking is encouraged.
- Leverage Interfaces – They provide flexibility without deep inheritance.
- Explore Standard Library – Go’s standard library is powerful and covers most tasks.