**National University of Technology**



**Computer Science Department**

Semester Spring– 2025

**Program:** Artificial intelligence

**Course:** Programming for AI Lab

**Course Code:** CS283

**Lab Report- 09**

| Submitted To: | Submitted By: |
| --- | --- |
| Umar Aftab | Muhammad Ahad Imran |
|  | F23607034 |

Go Programming

# 1. Hello, World!

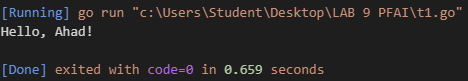
package main

import "fmt"

func main() {

fmt.Println("Hello, Ahad!")

}



# 2. Variables & Constants

package main

import "fmt"

func main() {

var x int = 5

const y = 10

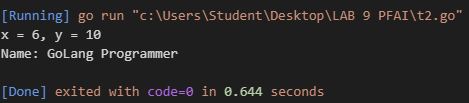
x += 1

var name string = "GoLang Programmer" // Added string variable

fmt.Printf("x = %d, y = %d\n", x, y)

fmt.Printf("Name: %s\n", name) // Printing the string

}



# 3. Arithmetic Operations

package main

import "fmt"

func main() {

a := 10

b := 3

fmt.Printf("Sum: %d\n", a+b)

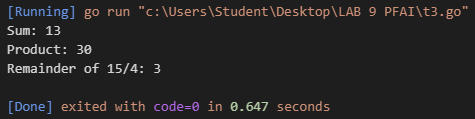
fmt.Printf("Product: %d\n", a\*b)

// Calculate remainder of 15 divided by 4

remainder := 15 % 4

fmt.Printf("Remainder of 15/4: %d\n", remainder)

}



# 4. Conditionals (if-else)

package main

import "fmt"

func main() {

n := -5

if n > 0 {

fmt.Println("Positive")

} else if n < 0 {

fmt.Println("Negative")

} else {

fmt.Println("Zero")

}

// Check if a number is even or odd

num := 7

if num%2 == 0 {

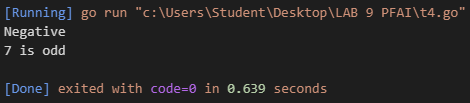
fmt.Printf("%d is even\n", num)

} else {

fmt.Printf("%d is odd\n", num)

}

}



# 5. Loops (for)

package main

import "fmt"

func main() {

for i := 1; i <= 3; i++ {

fmt.Printf("Iteration %d\n", i)

}

counter := 1

for counter <= 3 {

fmt.Printf("Counter: %d\n", counter)

counter++

}

// Print numbers from 10 to 1 in reverse

fmt.Println("Counting backward:")

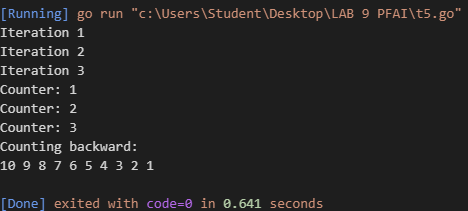
for i := 10; i >= 1; i-- {

fmt.Printf("%d ", i)

}

fmt.Println() // New line after the countdown

}



# 6. Basic Functions

package main

import "fmt"

func square(x int) int {

return x \* x

}

// Calculate factorial

func factorial(n int) int {

if n <= 1 {

return 1

}

return n \* factorial(n-1)

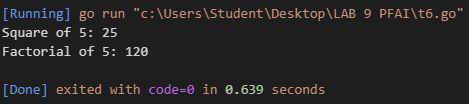
}

func main() {

fmt.Printf("Square of 5: %d\n", square(5))

fmt.Printf("Factorial of 5: %d\n", factorial(5))

}



# 7. Multiple Return Values

package main

import "fmt"

func swap(a, b string) (string, string) {

return b, a

}

// Return both sum and product

func sumAndProduct(a, b int) (int, int) {

return a + b, a \* b

}

func main() {

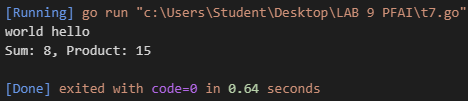
x, y := swap("hello", "world")

fmt.Println(x, y)

sum, product := sumAndProduct(5, 3)

fmt.Printf("Sum: %d, Product: %d\n", sum, product)

}



# 8. Arrays & Slices

package main

import "fmt"

func main() {

arr := [3]int{10, 20, 30}

slice := append(arr[:], 40)

fmt.Printf("First element: %d\n", arr[0])

fmt.Println("Slice:", slice)

// Create and iterate over a slice of strings

fruits := []string{"Apple", "Orange", "Banana", "Mango"}

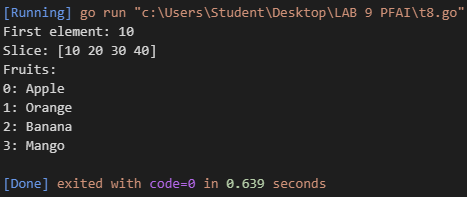
fmt.Println("Fruits:")

for i, fruit := range fruits {

fmt.Printf("%d: %s\n", i, fruit)

}

}



# 9. Maps & Structs

package main

import "fmt"

type Person struct {

Name string

Age int

}

func main() {

dict := map[string]string{"name": "Alice", "job": "Engineer"}

fmt.Println("Job:", dict["job"])

// Add a new key-value pair

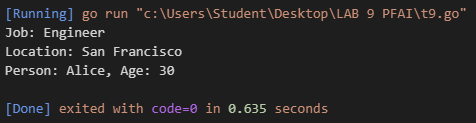
dict["location"] = "San Francisco"

fmt.Println("Location:", dict["location"])

p := Person{Name: "Alice", Age: 30}

fmt.Printf("Person: %s, Age: %d\n", p.Name, p.Age)

}



# 10. Error Checking

package main

import (

"fmt"

"strconv"

)

func parseNumber(s string) (int, error) {

return strconv.Atoi(s)

}

func main() {

if num, err := parseNumber("123"); err == nil {

fmt.Println("Number:", num)

} else {

fmt.Println("Error:", err)

}

// Handle invalid input

if num, err := parseNumber("abc"); err == nil {

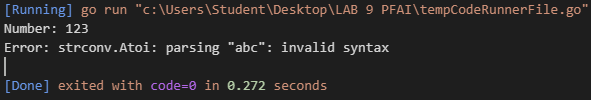
fmt.Println("Number:", num)

} else {

fmt.Println("Error:", err)

}

}



# 11. Read/Write Files

package main

import (

"fmt"

"os"

)

func main() {

os.WriteFile("data.txt", []byte("Go is efficient!"), 0644)

data, \_ := os.ReadFile("data.txt")

fmt.Println("File content:", string(data))

// Append a new line to the file

file, \_ := os.OpenFile("data.txt", os.O\_APPEND|os.O\_WRONLY, 0644)

defer file.Close()

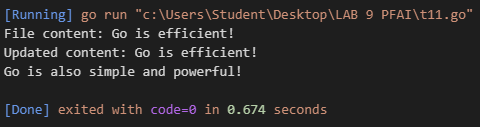
file.WriteString("\nGo is also simple and powerful!")

// Read updated content

updatedData, \_ := os.ReadFile("data.txt")

fmt.Println("Updated content:", string(updatedData))

}



# 12. Goroutines & Channels

package main

import (

"fmt"

"time"

)

func printNumbers(ch chan int) {

for i := 1; i <= 3; i++ {

ch <- i

time.Sleep(time.Second)

}

close(ch)

}

func sumArray(arr []int, ch chan int) {

sum := 0

for \_, num := range arr {

sum += num

}

ch <- sum

}

func main() {

ch := make(chan int)

go printNumbers(ch)

for num := range ch {

fmt.Println("Received:", num)

}

// Create two goroutines to calculate sum of two arrays

arr1 := []int{1, 2, 3, 4, 5}

arr2 := []int{6, 7, 8, 9, 10}

ch1 := make(chan int)

ch2 := make(chan int)

go sumArray(arr1, ch1)

go sumArray(arr2, ch2)

sum1 := <-ch1

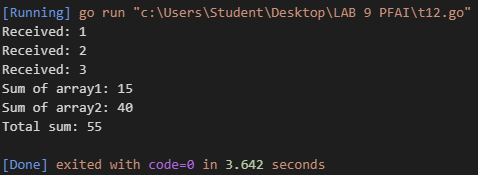
sum2 := <-ch2

fmt.Printf("Sum of array1: %d\n", sum1)

fmt.Printf("Sum of array2: %d\n", sum2)

fmt.Printf("Total sum: %d\n", sum1+sum2)

}



# 13. Using External Packages (e.g., Gin)

package main

import "github.com/gin-gonic/gin"

func main() {

r := gin.Default()

r.GET("/", func(c \*gin.Context) {

c.String(200, "Hello from Gin!")

})

// Create a GET endpoint that returns JSON data

r.GET("/api/data", func(c \*gin.Context) {

c.JSON(200, gin.H{

"status": "success",

"message": "Data retrieved successfully",

"data": gin.H{

"id": 123,

"name": "Product",

"price": 29.99,

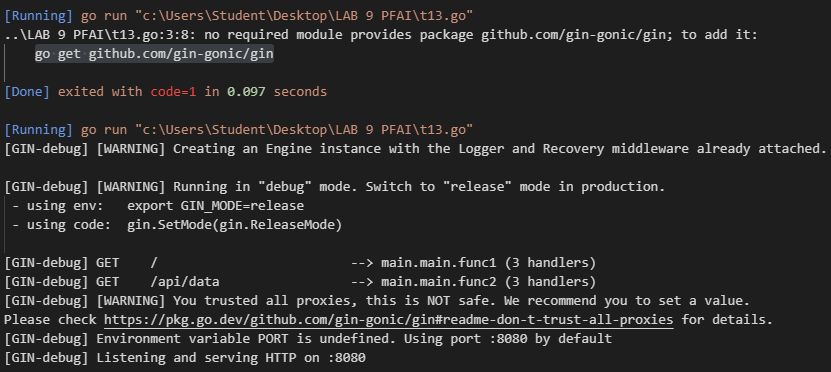
},

})

})

r.Run()

}



# 14. Prime Number Checker

package main

import (

"fmt"

"math"

)

// isPrime checks if a number is prime

func isPrime(n int) bool {

// 0 and 1 are not prime numbers

if n <= 1 {

return false

}

// 2 is prime

if n == 2 {

return true

}

// Even numbers (except 2) are not prime

if n%2 == 0 {

return false

}

// Check odd divisors up to square root of n

sqrtN := int(math.Sqrt(float64(n)))

for i := 3; i <= sqrtN; i += 2 {

if n%i == 0 {

return false

}

}

return true

}

func main() {

testCases := []int{0, 1, 2, 13, 27}

fmt.Println("Prime Number Checker:")

for \_, num := range testCases {

if isPrime(num) {

fmt.Printf("%d is prime\n", num)

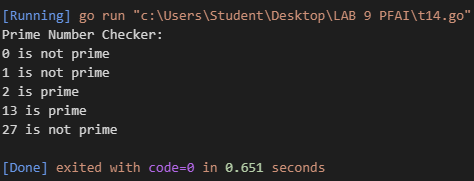
} else {

fmt.Printf("%d is not prime\n", num)

}

}

}



# 15. Slice Average Calculator

package main

import (

"errors"

"fmt"

)

// average calculates the mean of a slice of float64 values

func average(numbers []float64) (float64, error) {

if len(numbers) == 0 {

return 0, errors.New("cannot calculate average of empty slice")

}

sum := 0.0

for \_, num := range numbers {

sum += num

}

return sum / float64(len(numbers)), nil

}

func main() {

fmt.Println("Slice Average Calculator:")

// Test with non-empty slice

nums := []float64{5.2, 6.8, 9.1}

avg, err := average(nums)

if err == nil {

fmt.Printf("Average of %v: %.2f\n", nums, avg)

} else {

fmt.Printf("Error: %s\n", err)

}

// Test with empty slice

emptySlice := []float64{}

avg, err = average(emptySlice)

if err == nil {

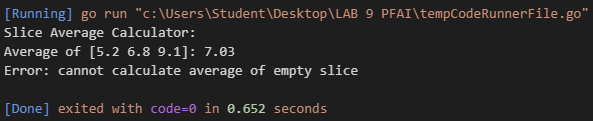
fmt.Printf("Average of %v: %.2f\n", emptySlice, avg)

} else {

fmt.Printf("Error: %s\n", err)

}

}



# 16. Employee Management System

package main

import "fmt"

// Employee struct with Name, Position, Salary

type Employee struct {

Name string

Position string

Salary float64

}

// highestPaid returns the employee with the highest salary

func highestPaid(employees []Employee) Employee {

if len(employees) == 0 {

return Employee{}

}

highest := employees[0]

for \_, emp := range employees {

if emp.Salary > highest.Salary {

highest = emp

}

}

return highest

}

func main() {

// Create a slice of 5 employees

employees := []Employee{

{Name: "Alice Johnson", Position: "Software Engineer", Salary: 85000},

{Name: "Bob Smith", Position: "Project Manager", Salary: 92000},

{Name: "Carol Davis", Position: "CTO", Salary: 120000},

{Name: "David Wilson", Position: "UI Designer", Salary: 78000},

{Name: "Eva Brown", Position: "DevOps Engineer", Salary: 90000},

}

// Find and print the highest paid employee

topEarner := highestPaid(employees)

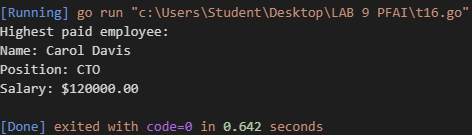
fmt.Printf("Highest paid employee:\n")

fmt.Printf("Name: %s\n", topEarner.Name)

fmt.Printf("Position: %s\n", topEarner.Position)

fmt.Printf("Salary: $%.2f\n", topEarner.Salary)

}



# 17. Robust File Appender

package main

import (

"fmt"

"os"

)

func appendToFile(filename, content string) error {

// Check if file exists

\_, err := os.Stat(filename)

if os.IsNotExist(err) {

// Create file if it doesn't exist

return os.WriteFile(filename, []byte(content), 0644)

}

// Open file for appending

file, err := os.OpenFile(filename, os.O\_APPEND|os.O\_WRONLY, 0644)

if err != nil {

return err

}

defer file.Close()

// Append content

\_, err = file.WriteString(content)

return err

}

func main() {

filename := "data.txt"

content := "\nThis is new content appended to the file."

err := appendToFile(filename, content)

if err != nil {

fmt.Printf("Failed to append to file: %s\n", err)

} else {

fmt.Printf("Successfully appended content to %s\n", filename)

// Read and print the file content

data, err := os.ReadFile(filename)

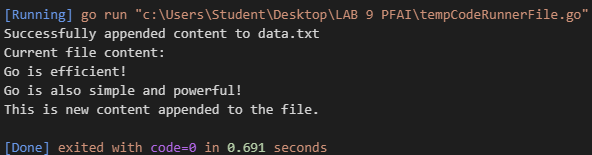
if err == nil {

fmt.Printf("Current file content:\n%s\n", string(data))

}

}

}



# 18. Concurrent Array Sum

package main

import "fmt"

func sumArray(arr []int, ch chan int) {

sum := 0

for \_, num := range arr {

sum += num

}

ch <- sum

}

func main() {

// Create two arrays

array1 := []int{1, 2, 3, 4, 5}

array2 := []int{6, 7, 8, 9, 10}

// Create channels for results

ch1 := make(chan int)

ch2 := make(chan int)

// Calculate sum concurrently

go sumArray(array1, ch1)

go sumArray(array2, ch2)

// Receive results

sum1 := <-ch1

sum2 := <-ch2

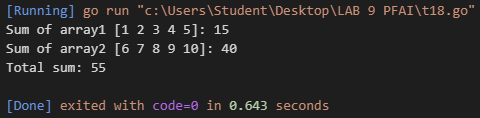
// Print results

fmt.Printf("Sum of array1 %v: %d\n", array1, sum1)

fmt.Printf("Sum of array2 %v: %d\n", array2, sum2)

fmt.Printf("Total sum: %d\n", sum1+sum2)

}



# 19. JSON API Endpoint

package main

import "github.com/gin-gonic/gin"

func main() {

r := gin.Default()

// Create /user endpoint returning JSON

r.GET("/user", func(c \*gin.Context) {

c.JSON(200, gin.H{

"id": 123,

"name": "Alice",

"email": "alice@example.com",

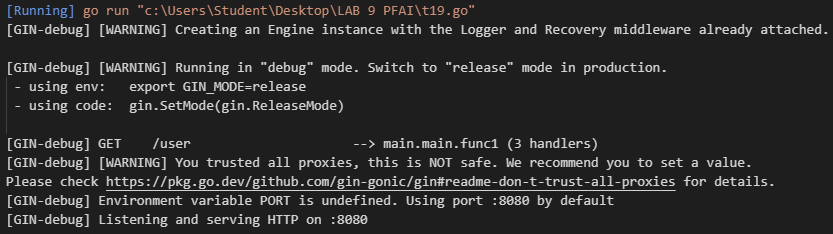
})

})

// Start the server

r.Run() // Listens on 0.0.0.0:8080

}



# 20. Bank Account Manager

package main

import (

"errors"

"fmt"

)

// BankAccount struct with balance

type BankAccount struct {

balance float64

}

// Deposit adds amount to balance

func (a \*BankAccount) Deposit(amount float64) error {

if amount <= 0 {

return errors.New("deposit amount must be positive")

}

a.balance += amount

return nil

}

// Withdraw subtracts amount from balance

func (a \*BankAccount) Withdraw(amount float64) error {

if amount <= 0 {

return errors.New("withdrawal amount must be positive")

}

if amount > a.balance {

return errors.New("insufficient funds")

}

a.balance -= amount

return nil

}

// Balance returns the current balance

func (a \*BankAccount) Balance() float64 {

return a.balance

}

func main() {

// Create a new account

account := BankAccount{balance: 100}

fmt.Printf("Initial balance: $%.2f\n", account.Balance())

// Test deposit

err := account.Deposit(50)

if err != nil {

fmt.Printf("Deposit error: %s\n", err)

} else {

fmt.Printf("After deposit: $%.2f\n", account.Balance())

}

// Test successful withdrawal

err = account.Withdraw(30)

if err != nil {

fmt.Printf("Withdrawal error: %s\n", err)

} else {

fmt.Printf("After withdrawal: $%.2f\n", account.Balance())

}

// Test overdraft prevention

err = account.Withdraw(200)

if err != nil {

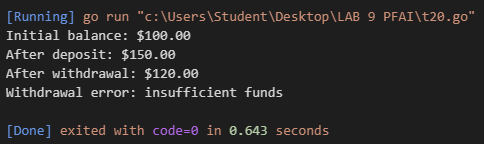
fmt.Printf("Withdrawal error: %s\n", err)

} else {

fmt.Printf("After withdrawal: $%.2f\n", account.Balance())

}

}



# 21. Shape Interface

package main

import (

"fmt"

"math"

)

// Shape interface with Area method

type Shape interface {

Area() float64

}

// Circle implements Shape

type Circle struct {

radius float64

}

func (c Circle) Area() float64 {

return math.Pi \* c.radius \* c.radius

}

// Rectangle implements Shape

type Rectangle struct {

width float64

height float64

}

func (r Rectangle) Area() float64 {

return r.width \* r.height

}

func main() {

// Create a slice of different shapes

shapes := []Shape{

Circle{radius: 5},

Rectangle{width: 4, height: 6},

Circle{radius: 3},

Rectangle{width: 10, height: 2},

}

// Calculate and print each shape's area

for i, shape := range shapes {

fmt.Printf("Shape %d area: %.2f\n", i+1, shape.Area())

}

}

