Slip 1

package main

import (

"fmt"

)

func main() {

var num1, num2 float64

var operator string

fmt.Print("Enter first number: ")

fmt.Scanln(&num1)

fmt.Print("Enter second number: ")

fmt.Scanln(&num2)

fmt.Print("Enter operator (+, -, \*, /): ")

fmt.Scanln(&operator)

var result float64

switch operator {

case "+":

result = num1 + num2

case "-":

result = num1 - num2

case "\*":

result = num1 \* num2

case "/":

if num2 != 0 {

result = num1 / num2

} else {

fmt.Println("Error: Cannot divide by zero!")

return

}

default:

fmt.Println("Error: Invalid operator!")

return

}

fmt.Printf("Result: %.2f %s %.2f = %.2f\n", num1, operator, num2, result)

}

Slip 2

package main

import "fmt"

func fibonacci(n int) {

if n <= 0 {

fmt.Println("Invalid input. Please enter a positive integer.")

return

}

var a, b int = 0, 1

fmt.Print("Fibonacci Series up to ", n, " terms:")

for i := 0; i < n; i++ {

fmt.Print(" ", a)

temp := a

a = b

b = temp + b

}

fmt.Println()

}

func main() {

var n int

fmt.Print("Enter the number of terms for Fibonacci series: ")

fmt.Scanln(&n)

fibonacci(n)

}

Slip 3

package main

import (

"fmt"

)

func isPalindrome(num int) bool {

original := num

reversed := 0

// Reverse the number

for num > 0 {

digit := num % 10

reversed = reversed\*10 + digit

num /= 10

}

// Check if the original number is equal to its reverse

return original == reversed

}

func main() {

var num int

fmt.Print("Enter a number to check if it's a palindrome: ")

fmt.Scanln(&num)

if isPalindrome(num) {

fmt.Println(num, "is a palindrome.")

} else {

fmt.Println(num, "is not a palindrome.")

}

}

Slip 4

package main

import (

"fmt"

)

func recursiveDigitSum(num int) int {

if num < 10 {

return num

}

return num%10 + recursiveDigitSum(num/10)

}

func main() {

var num int

fmt.Print("Enter a number: ")

fmt.Scanln(&num)

sum := recursiveDigitSum(num)

fmt.Println("Recursive sum of digits of", num, "is:", sum)

}

Slip 5 q2

package main

import (

"fmt"

)

type Employee struct {

Eno int

Ename string

Salary float64

}

func main() {

var n int

fmt.Print("Enter the number of employees: ")

fmt.Scanln(&n)

employees := make([]Employee, n)

// Input employee details

for i := 0; i < n; i++ {

fmt.Printf("Enter details for employee %d:\n", i+1)

fmt.Print("Employee Number: ")

fmt.Scanln(&employees[i].Eno)

fmt.Print("Employee Name: ")

fmt.Scanln(&employees[i].Ename)

fmt.Print("Salary: ")

fmt.Scanln(&employees[i].Salary)

}

// Find minimum salary

minSalary := employees[0].Salary

for \_, emp := range employees {

if emp.Salary < minSalary {

minSalary = emp.Salary

}

}

// Display employees with minimum salary

fmt.Println("\nEmployees with minimum salary:")

for \_, emp := range employees {

if emp.Salary == minSalary {

fmt.Printf("Eno: %d, Ename: %s, Salary: %.2f\n", emp.Eno, emp.Ename, emp.Salary)

}

}

}

Slip 6 q2

package main

import (

"fmt"

)

func copyArray(source []int, destination []int) {

for i, num := range source {

destination[i] = num

}

}

func main() {

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

destinationArray := make([]int, len(sourceArray))

copyArray(sourceArray, destinationArray)

fmt.Println("Source Array:", sourceArray)

fmt.Println("Destination Array:", destinationArray)

}

Slip 7 q2

package main

import "fmt"

// Define the Student structure

type Student struct {

ID int

Name string

Age int

}

// Method show() with receiver of type pointer to Student

func (s \*Student) show() {

fmt.Printf("Student ID: %d\n", s.ID)

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

fmt.Printf("Student Age: %d\n", s.Age)

}

func main() {

// Create a new Student instance

student := &Student{

ID: 1,

Name: "John Doe",

Age: 20,

}

// Call the show() method

student.show()

}

Slip 8

package main

import (

"fmt"

)

// Define the Book structure

type Book struct {

BookID int

Title string

Author string

Price float64

}

func main() {

var n int

fmt.Print("Enter the number of books: ")

fmt.Scanln(&n)

books := make([]Book, n)

// Input book details

for i := 0; i < n; i++ {

fmt.Printf("Enter details for book %d:\n", i+1)

fmt.Print("Book ID: ")

fmt.Scanln(&books[i].BookID)

fmt.Print("Title: ")

fmt.Scanln(&books[i].Title)

fmt.Print("Author: ")

fmt.Scanln(&books[i].Author)

fmt.Print("Price: ")

fmt.Scanln(&books[i].Price)

}

// Display book details

fmt.Println("\nDetails of", n, "books:")

for i, book := range books {

fmt.Printf("Book %d:\n", i+1)

fmt.Println("Book ID:", book.BookID)

fmt.Println("Title:", book.Title)

fmt.Println("Author:", book.Author)

fmt.Println("Price:", book.Price)

fmt.Println()

}

}

Slip 9

package main

import (

"fmt"

)

func isPalindrome(num int) bool {

reversed := 0

original := num

for num > 0 {

digit := num % 10

reversed = reversed\*10 + digit

num /= 10

}

return original == reversed

}

func main() {

var num int

fmt.Print("Enter a number to check if it's a palindrome: ")

fmt.Scanln(&num)

if isPalindrome(num) {

fmt.Println(num, "is a palindrome.")

} else {

fmt.Println(num, "is not a palindrome.")

}

}

Slip 10

package main

import "fmt"

// Define an interface

type Printer interface {

Print()

}

// Define a struct type

type Integer int

// Implement the Printer interface for Integer

func (i Integer) Print() {

fmt.Println("Integer:", i)

}

// Define another struct type

type String string

// Implement the Printer interface for String

func (s String) Print() {

fmt.Println("String:", s)

}

func main() {

// Create values of different types

var values = []Printer{

Integer(10),

String("Hello, World!"),

Integer(20),

String("Goodbye!"),

}

// Iterate through the values and display them using type assertion

for \_, val := range values {

switch v := val.(type) {

case Integer:

v.Print()

case String:

v.Print()

default:

fmt.Println("Unknown type")

}

}

}

Slip 11

package main

import (

"fmt"

)

func main() {

var num int

fmt.Print("Enter a number: ")

fmt.Scanln(&num)

if num >= 10 && num <= 99 {

fmt.Println(num, "is a two-digit number.")

} else {

fmt.Println(num, "is not a two-digit number.")

}

}

Slip 12

package main

import (

"fmt"

)

// Function to swap two numbers using call by reference

func swapByReference(a \*int, b \*int) {

temp := \*a

\*a = \*b

\*b = temp

}

func main() {

var num1, num2 int

// Input two numbers

fmt.Print("Enter the first number: ")

fmt.Scanln(&num1)

fmt.Print("Enter the second number: ")

fmt.Scanln(&num2)

// Display the original numbers

fmt.Println("Before swapping:")

fmt.Println("First number:", num1)

fmt.Println("Second number:", num2)

// Swap the numbers using call by reference

swapByReference(&num1, &num2)

// Display the swapped numbers

fmt.Println("\nAfter swapping:")

fmt.Println("First number:", num1)

fmt.Println("Second number:", num2)

}

Slip 13

package main

import "fmt"

func main() {

evenSum, oddSum := 0, 0

for i := 1; i <= 100; i++ {

if i%2 == 0 {

evenSum += i

} else {

oddSum += i

}

}

fmt.Println("Sum of even numbers between 1 and 100:", evenSum)

fmt.Println("Sum of odd numbers between 1 and 100:", oddSum)

}

Slip 14

package main

import (

"fmt"

)

func main() {

// Creating a slice

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

fmt.Println("Original Slice:", slice)

// Appending elements to the slice

slice = append(slice, 6, 7, 8)

fmt.Println("After Appending:", slice)

// Removing an element at index 2

indexToRemove := 2

slice = append(slice[:indexToRemove], slice[indexToRemove+1:]...)

fmt.Println("After Removing Element at Index 2:", slice)

// Copying a slice

copySlice := make([]int, len(slice))

copy(copySlice, slice)

fmt.Println("Copied Slice:", copySlice)

// Modifying an element in the copied slice

copySlice[0] = 100

fmt.Println("Modified Copied Slice:", copySlice)

// Original slice remains unchanged

fmt.Println("Original Slice:", slice)

}

Slip 15

package main

import "fmt"

// Function that returns multiple values

func calculate(x, y int) (int, int) {

sum := x + y

difference := x - y

return sum, difference

}

func main() {

// Call the function and receive multiple return values

sum, difference := calculate(10, 5)

fmt.Println("Sum:", sum)

fmt.Println("Difference:", difference)

}

Slip 16 q2

package main

import (

"fmt"

"time"

)

func main() {

for i := 0; i <= 10; i++ {

fmt.Println(i)

time.Sleep(250 \* time.Millisecond)

}

}

Slip 17

package main

import "fmt"

// Function to perform addition and return the result

func add(a, b float64) float64 {

return a + b

}

// Function to perform subtraction and return the result

func subtract(a, b float64) float64 {

return a - b

}

// Function to perform multiplication and return the result

func multiply(a, b float64) float64 {

return a \* b

}

// Function to perform division and return the result

// Returns an error if dividing by zero

func divide(a, b float64) (float64, error) {

if b == 0 {

return 0, fmt.Errorf("cannot divide by zero")

}

return a / b, nil

}

func main() {

// Define two numbers

num1 := 10.0

num2 := 5.0

// Perform addition

sum := add(num1, num2)

fmt.Printf("Addition: %.2f + %.2f = %.2f\n", num1, num2, sum)

// Perform subtraction

difference := subtract(num1, num2)

fmt.Printf("Subtraction: %.2f - %.2f = %.2f\n", num1, num2, difference)

// Perform multiplication

product := multiply(num1, num2)

fmt.Printf("Multiplication: %.2f \* %.2f = %.2f\n", num1, num2, product)

// Perform division

quotient, err := divide(num1, num2)

if err != nil {

fmt.Println("Error:", err)

} else {

fmt.Printf("Division: %.2f / %.2f = %.2f\n", num1, num2, quotient)

}

}

Slip 18

package main

import "fmt"

// Function to print the multiplication table of a number up to a specified limit

func multiplicationTable(number, limit int) {

fmt.Printf("Multiplication Table of %d:\n", number)

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

result := number \* i

fmt.Printf("%d x %d = %d\n", number, i, result)

}

}

func main() {

var number, limit int

// Input the number and limit from the user

fmt.Print("Enter the number for multiplication table: ")

fmt.Scanln(&number)

fmt.Print("Enter the limit for multiplication table: ")

fmt.Scanln(&limit)

// Call the function to print the multiplication table

multiplicationTable(number, limit)

}

Slip 19

package main

import (

"fmt"

"os"

)

func main() {

// Specify the file path

filePath := "example.txt"

// Open the file in READ only mode

file, err := os.Open(filePath)

if err != nil {

fmt.Println("Error:", err)

return

}

defer file.Close()

fmt.Println("File opened successfully in READ only mode.")

// You can now read from the file using file.Read() or other methods

}

Slip 20

package main

import (

"fmt"

)

func main() {

// Create a channel of type int

ch := make(chan int)

// Start a goroutine to send numbers to the channel

go func() {

for i := 0; i < 5; i++ {

ch <- i

}

// Close the channel after sending all numbers

close(ch)

}()

// Read values from the channel using a for range loop

for num := range ch {

fmt.Println("Received:", num)

}

// Check if the channel is closed

\_, ok := <-ch

if ok {

fmt.Println("Channel is not closed")

} else {

fmt.Println("Channel is closed")

}

}