Slip 1

Q.1]

```
package main
import(
 "fmt"
 )
 func main() {
 var a,b,z int
 fmt.Printf("enter two numbers:")
 fmt.Scanf("%d%d",&a,&b)
 fmt.Printf("\n1:addition \n2:subtraction \n3:multiplication \n4:division \n5:remainder")
 fmt.Printf("\nenter your choice:")
 fmt.Scanf("%d",&z)
 switch z {
 case 1:
  fmt.Printf("\n addition=%d",a+b)
 case 2:
  fmt.Printf("\n subtraction=%d",a-b)
 case 3:
  fmt.Printf("\n multiplication=%d",a*b)
 case 4:
  fmt.Printf("\n division=%d",a/b)
 case 5:
  fmt.Printf("\n remainder=%d",a%b)
 default:
  fmt.Printf("wrong choice")
 }
 }
```

PAYAL DANGAT

Q.2]

Slip 2

Q.1]

```
package main
import "fmt"
func fibonacci(num int) {
   var a,b int
   for a,b=0,1;a<=num;a,b=b,a+b {
      fmt.Printf("%d\t",a)
   }
   fmt.Println()
}

func main() {
   var num int
   fmt.Print("Enter number: ")
   fmt.Scanf("%d", &num)
   fibonacci(num)
}
```

Q.2]

```
package main

import (
	"fmt"
	"os"
	"log"
	"time"
)

func main() {
	filePath := "tybca.txt"
	fileInfo, err := os.Stat(filePath)
	if err != nil {
		log.Fatal(err)
	}
	fmt.Println("File Information:")
	fmt.Println("Name:", fileInfo.Name())
	fmt.Println("Size:", fileInfo.Size(), "bytes")
	fmt.Println("Mode:", fileInfo.Mode())
	fmt.Println("Is Directory:", fileInfo.IsDir())
	fmt.Println("Modification Time:", fileInfo.ModTime().Format(time.RFC3339))
}
```

Slip 3

Q.1]

```
package main

import "fmt"

var reverse int = 0

func revNumber(palNum int) int {
   var remainder int

   for ; palNum > 0; palNum = palNum / 10 {
       remainder = palNum % 10
       reverse = reverse*10 + remainder
   }
   return reverse
}
func main() {

   var palNum int

   fmt.Print("Enter the Number to check Palindrome = ")
   fmt.Scanln(&palNum)

   reverse = revNumber(palNum)
   fmt.Println("The Reverse of the Given Number = ", reverse)

   if palNum == reverse {
       fmt.Println(palNum, " is a Palindrome Number")
   } else {
       fmt.Println(palNum, " is Not a Palindrome Number")
   }
}
```

Slip 4

Q.1]

```
package main

import "fmt"

var sum int = 0
```

```
func SumOfDigits(num int) int {
        if num > 0 {
                sum += (num % 10) //add digit into sum
                SumOfDigits(num / 10)
        }
        return sum
}

func main() {
        var num int = 0
        var result int = 0

        fmt.Printf("Enter number: ")
        fmt.Scanf("%d", &num)

        result = SumOfDigits(num)

        fmt.Printf("Sum of digits is: %d\n", result)
}
```

Q.2]

```
package main
import "fmt"
func sortArray(arr [5]int, min int, temp int) [5]int {
  for i := 0; i <= 4; i++ {
    min = i
    for j := i + 1; j <= 4; j++ {
      if arr[j] < arr[min] {

        // changing the index to show the min value
    min = j
   }
  }
  temp = arr[i]
  arr[i] = arr[min]
  arr[min] = temp
 }
 return arr
}
func main() {
 arr := [5]int{50, 30, 20, 10, 40}
 fmt.Println("The unsorted array entered is:", arr)
 var min int = 0
 var temp int = 0
 array := sortArray(arr, min, temp)
 fmt.Println()
 fmt.Println("The final array obtained after sorting is:", array)
}
```

Slip 5

Q.1]

```
package main

import "fmt"
import "os"

func main() {
	file, err := os.Create("tybca.txt")
	if err != nil {
		fmt.Println("Unable to open file: %s", err)
	}

	len, err := file.WriteString("Hello World")

	if err != nil {
		fmt.Println("Unable to write data: %s", err)
	}
	file.Close()

	fmt.Printf("%d character written successfully into file", len)
}
```

Slip 6

Q.1]

```
package main

import "fmt"

func main() {
	var sum int = 0
	var matrix1 [2][2]int
	var matrix2 [2][2]int
	var matrix3 [2][2]int

	fmt.Printf("Enter matrix1 elements: \n")
	for i := 0; i < 2; i++ {
		for j := 0; j < 2; j++ {
			fmt.Printf("Elements: matrix1[%d][%d]: ", i, j)
			fmt.Scanf("%d", &matrix1[i][j])
		}
}

fmt.Printf("Enter matrix2 elements: \n")
for i := 0; i < 2; i++ {
    for j := 0; j < 2; j++ {
        fmt.Printf("Elements: matrix2[%d][%d]: ", i, j)
        fmt.Scanf("%d", &matrix2[i][j])
    }
}

//Multiplication of matrix1 and matrix2.
for i := 0; i < 2; i++ {
    for j := 0; j < 2; j++ {
        sum = 0
        for k := 0; k < 2; k++ {
            sum = sum + matrix1[i][k]*matrix2[k][j]
        }
        matrix3[i][j] = sum
    }
}

fmt.Printf("Matrix1: \n")
for i := 0; i < 2; i++ {
    for j := 0; j < 2; j++ {
        fmt.Printf("%d ", matrix1[i][j])
    }
    fmt.Printf("\n")
}
```

```
        fmt.Printf("Matrix2: \n")
        for i := 0; i < 2; i++ {
                for j := 0; j < 2; j++ {
                        fmt.Printf("%d ", matrix2[i][j])
                }
                fmt.Printf("\n")
        }

        fmt.Printf("Multiplication of matrix1 and matrix2: \n")
        for i := 0; i < 2; i++ {
                for j := 0; j < 2; j++ {
                        fmt.Printf("%d ", matrix3[i][j])
                }
                fmt.Printf("\n")
        }
}
```

SLIP 7

Q.1]

```
package main

import "fmt"

func main() {
   var i, j, rows, columns int

   var orgMat [10][10]int
   var transposeMat [10][10]int

   fmt.Print("Enter the Matrix rows and Columns = ")
```

```
    fmt.Scan(&rows, &columns)

    fmt.Println("Enter Matrix Items to Transpose = ")
    for i = 0; i < rows; i++ {
        for j = 0; j < columns; j++ {
            fmt.Scan(&orgMat[i][j])
        }
    }
    for i = 0; i < rows; i++ {
        for j = 0; j < columns; j++ {
            transposeMat[j][i] = orgMat[i][j]
        }
    }
    fmt.Println("*** The Transpose Matrix Items are ***")
    for i = 0; i < columns; i++ {
        for j = 0; j < rows; j++ {
            fmt.Print(transposeMat[i][j], "  ")
        }
        fmt.Println()
    }
}
```

Q.2]

```
package main

import (
        "fmt"
)

// Define the Student structure
```

```
type Student struct {
	ID    int
	Name  string
	Grade float64
}

// Define the Show method with a receiver of type pointer to Student
func (s *Student) Show() {
	fmt.Printf("Student ID: %d\n", s.ID)
	fmt.Printf("Name: %s\n", s.Name)
	fmt.Printf("Grade: %.2f\n", s.Grade)
}

func main() {
	// Create an instance of the Student struct
	student := Student{
		ID:    1,
		Name:  "John Doe",
		Grade: 85.5,
	}

	// Call the Show method on the Student instance
	student.Show()
}
```

SLIP 8

Q.1]

```
package main
```

```
import "fmt"

// Define the Book structure
type Book struct {
	ID     int
	Title  string
	Author string
	Price  float64
}

// Function to read book details
func readBookDetails() Book {
	var book Book

	fmt.Print("Enter Book ID: ")
	fmt.Scan(&book.ID)

	fmt.Print("Enter Book Title: ")
	fmt.Scanln()
	fmt.Scan(&book.Title)

	fmt.Print("Enter Author: ")
	fmt.Scanln()
	fmt.Scan(&book.Author)

	fmt.Print("Enter Price: ")
	fmt.Scan(&book.Price)

	return book
```

```
}

// Function to display book details
func displayBookDetails(book Book) {
	fmt.Printf("Book ID: %d\n", book.ID)
	fmt.Printf("Title: %s\n", book.Title)
	fmt.Printf("Author: %s\n", book.Author)
	fmt.Printf("Price: %.2f\n", book.Price)
	fmt.Println("---------------------------")
}

func main() {
	var n int

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

	// Slice to store book details
	books := make([]Book, n)

	// Reading book details
	for i := 0; i < n; i++ {
		fmt.Printf("\nEnter details for Book %d:\n", i+1)
		books[i] = readBookDetails()
	}

	// Displaying book details
	fmt.Println("\nBook Details:")
	for i := 0; i < n; i++ {
		displayBookDetails(books[i])
	}
}
```

Q.2]

```
package main

import (
	"fmt"
	"math"
)

// Define the Shape interface
type Shape interface {
	Area() float64
	Perimeter() float64
}

// Define the Circle type
type Circle struct {
	Radius float64
}

// Implement the Area method for Circle
func (c Circle) Area() float64 {
	return math.Pi * c.Radius * c.Radius
}

// Implement the Perimeter method for Circle
func (c Circle) Perimeter() float64 {
	return 2 * math.Pi * c.Radius
}

// Define the Rectangle type
type Rectangle struct {
	Width  float64
	Height float64
}

// Implement the Area method for Rectangle
func (r Rectangle) Area() float64 {
	return r.Width * r.Height
}

// Implement the Perimeter method for Rectangle
func (r Rectangle) Perimeter() float64 {
	return 2*r.Width + 2*r.Height
}

func main() {
	// Create instances of Circle and Rectangle
	circle := Circle{Radius: 5}
	rectangle := Rectangle{Width: 4, Height: 6}

	// Use the Shape interface to calculate and display area and perimeter
	printShapeDetails(circle, "Circle")
	printShapeDetails(rectangle, "Rectangle")
}
```

```
// Function to print shape details (area and perimeter)
func printShapeDetails(s Shape, shapeType string) {
	fmt.Printf("%s Details:\n", shapeType)
	fmt.Printf("Area: %.2f\n", s.Area())
	fmt.Printf("Perimeter: %.2f\n", s.Perimeter())
	fmt.Println("---------------------------------")
}
```

SLIP 9

Q.1]

```
package main

import (
	"fmt"
	"strconv"
)

// Function to check if a number is palindrome
func isPalindrome(num int) bool {
	// Convert the number to a string for easy comparison
	strNum := strconv.Itoa(num)

	// Compare characters from the beginning and end of the string
	for i, j := 0, len(strNum)-1; i < j; i, j = i+1, j-1 {
		if strNum[i] != strNum[j] {
			return false
		}
	}
	return true
}

func main() {
	var number int

	// Accept a number from the user
	fmt.Print("Enter a number: ")
	fmt.Scan(&number)

	// Check if the number is a palindrome
	if isPalindrome(number) {
		fmt.Println(number, "is a palindrome.")
	} else {
		fmt.Println(number, "is not a palindrome.")
	}
}
```

Q.2]

```
package main

import (
	"fmt"

)

// Define the Shape interface
type Shape interface {
	Area() float64
	Perimeter() float64
}

// Define the Circle type
type Square struct {
	side float64
}

// Implement the Area method for Circle
func (s Square) Area() float64 {
	return s.side * s.side
}

// Implement the Perimeter method for Circle
func (s Square) Perimeter() float64 {
	return 4 * s.side
}

// Define the Rectangle type
type Rectangle struct {
	Width  float64
	Height float64
}

// Implement the Area method for Rectangle
func (r Rectangle) Area() float64 {
	return r.Width * r.Height
}
```

```
// Implement the Perimeter method for Rectangle
func (r Rectangle) Perimeter() float64 {
	return 2*r.Width + 2*r.Height
}

func main() {
	// Create instances of Circle and Rectangle
	square:= Square{side: 5}
	rectangle := Rectangle{Width: 4, Height: 6}

	// Use the Shape interface to calculate and display area and perimeter
	printShapeDetails(square, "Square")
	printShapeDetails(rectangle, "Rectangle")
}

// Function to print shape details (area and perimeter)
func printShapeDetails(s Shape, shapeType string) {
	fmt.Printf("%s Details:\n", shapeType)
	fmt.Printf("Area: %.2f\n", s.Area())
	fmt.Printf("Perimeter: %.2f\n", s.Perimeter())
	fmt.Println("----------------------------")
}
```

SLIP 10

Q.1]

```
package main

import (
	"fmt"
```

```
)

// Define the custom interface
type MyInterface interface {
	display()
}

// Define a struct type implementing MyInterface
type MyStruct1 struct {
	Value string
}

func (ms MyStruct1) display() {
	fmt.Println("MyStruct1:", ms.Value)
}

// Define another struct type implementing MyInterface
type MyStruct2 struct {
	Value int
}

func (ms MyStruct2) display() {
	fmt.Println("MyStruct2:", ms.Value)
}

func main() {
	// Create instances of MyStruct1 and MyStruct2
	instance1 := MyStruct1{Value: "Hello"}
	instance2 := MyStruct2{Value: 42}
	// Use the interface type to store values of different types
	var myInterface MyInterface

	myInterface = instance1
	displayValue(myInterface)

	myInterface = instance2
	displayValue(myInterface)
}

// Function to display values using type assertion
func displayValue(i MyInterface) {
	// Type assertion to determine the actual underlying type
	switch v := i.(type) {
	case MyStruct1:
		v.display()
	case MyStruct2:
		v.display()
	default:
		fmt.Println("Unknown type")
	}
}
```

Q.2]

```
package main

import (
	"fmt"
	"sync"
)

// Function to generate Fibonacci series and send it to the channel
func generateFibonacci(n int, ch chan<- int, wg *sync.WaitGroup) {
	defer wg.Done()

	a, b := 0, 1
	for i := 0; i < n; i++ {
		ch <- a
		a, b = b, a+b
	}

	close(ch)
}

// Function to read from the channel and display the Fibonacci series
func displayFibonacci(ch <-chan int, wg *sync.WaitGroup) {
	defer wg.Done()

	for num := range ch {
		fmt.Print(num, " ")
	}

	fmt.Println()
}

func main() {
	// Create a channel to send and receive Fibonacci numbers
	fibonacciChannel := make(chan int, 10) // Adjust the buffer size as needed

	// Use WaitGroup to wait for goroutines to finish
	var wg sync.WaitGroup

	// Number of Fibonacci numbers to generate
	n := 10

	// Increment the WaitGroup counter for two goroutines
	wg.Add(2)

	// Launch goroutines to generate and display Fibonacci series
	go generateFibonacci(n, fibonacciChannel, &wg)
	go displayFibonacci(fibonacciChannel, &wg)

	// Wait for both goroutines to finish
	wg.Wait()
}
```

SLIP 11

Q.1]

```
package main

import (
	"fmt"
)

func main() {
	var num int
	// Accepting input from the user
	fmt.Print("Enter a number: ")
	fmt.Scan(&num)

	// Checking if the number is two-digit or not
	if num >= 10 && num <= 99 {
		fmt.Println("The entered number is a two-digit number.")
	} else {
		fmt.Println("The entered number is not a two-digit number.")
	}
}
```

Q.2]

```
package main

import (
	"fmt"
)

func main() {
	// Create a buffered channel with a capacity of 3
	bufferedChannel := make(chan int, 3)

	// Store values in the channel
	bufferedChannel <- 10
	bufferedChannel <- 20
	bufferedChannel <- 30
	// Find and print the channel capacity and length
	capacity := cap(bufferedChannel)
	length := len(bufferedChannel)
	fmt.Printf("Channel Capacity: %d\n", capacity)
	fmt.Printf("Initial Channel Length: %d\n", length)

	// Read values from the channel
	value1 := <-bufferedChannel
	value2 := <-bufferedChannel
	value3 := <-bufferedChannel

	// Find and print the modified length after reading
	length = len(bufferedChannel)
	fmt.Printf("Modified Channel Length: %d\n", length)

	// Print the values read from the channel
	fmt.Printf("Values read from the channel: %d, %d, %d\n", value1, value2, value3)
}
```

SLIP 12

Q.1]

```
package main

import (
	"fmt"
)

// swap function takes two pointers and swaps the values they point to
func swap(x *int, y *int) {
	temp := *x
	*x = *y
	*y = temp
}

func main() {
	// Declare and initialize two variables
	num1 := 5
	num2 := 10

	// Print the initial values
	fmt.Printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2)

	// Call the swap function with the addresses of num1 and num2
	swap(&num1, &num2)

	// Print the values after swapping
	fmt.Printf("After swapping: num1 = %d, num2 = %d\n", num1, num2)
}
```

Q.2]

```
package main

import (
	"fmt"
	"sync"
)

func checkEvenOdd(number int, evenChan chan int, oddChan chan int, wg *sync.WaitGroup) {
```

```go
		defer wg.Done()

		if number%2 == 0 {
				evenChan <- number
		} else {
				oddChan <- number
		}
}

func main() {
		// Create a slice of integers
		numbers := []int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

		// Create channels for even and odd numbers
		evenChan := make(chan int)
		oddChan := make(chan int)

		// Create a wait group to synchronize goroutines
		var wg sync.WaitGroup

		// Iterate over the slice and launch goroutines
		for _, num := range numbers {
				wg.Add(1)
				go checkEvenOdd(num, evenChan, oddChan, &wg)
		}

		// Close channels after all goroutines are done
		go func() {
				wg.Wait()
		close(evenChan)

		close(oddChan)

	}()


	// Receive and display even numbers

	fmt.Println("Even Numbers:")

	for even := range evenChan {

		fmt.Println(even)

	}


	// Receive and display odd numbers

	fmt.Println("\nOdd Numbers:")

	for odd := range oddChan {

		fmt.Println(odd)

	}

}
```

SLIP 13

Q.1]

```
package main


import "fmt"


func main() {

	// Initialize variables for sum of even and odd numbers

	sumEven := 0

	sumOdd := 0


	// Iterate through numbers from 1 to 100
	for i := 1; i <= 100; i++ {
		if i%2 == 0 {
			// Add even numbers to sumEven
			sumEven += i
		} else {
			// Add odd numbers to sumOdd
			sumOdd += i
		}
	}

	// Print the sum of even and odd numbers separately
	fmt.Printf("Sum of even numbers between 1 to 100: %d\n", sumEven)
	fmt.Printf("Sum of odd numbers between 1 to 100: %d\n", sumOdd)
}
```

Q.2]

```
package main

import (
	"fmt"
	"testing"
)

// Square function calculates the square of a number
func Square(x int) int {
	return x * x
}

func main() {
	// Test the Square function
	num := 5
	squareResult := Square(num)
	fmt.Printf("Square of %d is %d\n", num, squareResult)

	// Run the benchmark for the Square function
	fmt.Println("\nRunning Benchmark:")
	result := testing.Benchmark(benchmarkSquare)
	fmt.Println(result)
}

// benchmarkSquare is a benchmark function for the Square function
func benchmarkSquare(b *testing.B) {
	for i := 0; i < b.N; i++ {
		Square(5) // Square of 5 is calculated repeatedly for benchmarking
	}
}
```

SLIP 14

Q.1]

```
package main

import "fmt"

func main() {
	// Creating a slice with initial elements
	mySlice := []int{1, 2, 3, 4, 5}
	fmt.Println("Initial Slice:", mySlice)
```

```go
	// Appending elements to the slice
	mySlice = append(mySlice, 6, 7, 8)
	fmt.Println("After Appending:", mySlice)

	// Removing elements from the slice (removing the element at index 2)
	indexToRemove := 2
	mySlice = append(mySlice[:indexToRemove], mySlice[indexToRemove+1:]...)
	fmt.Println("After Removing at Index 2:", mySlice)

	// Copying the slice to a new slice
	copiedSlice := make([]int, len(mySlice))
	copy(copiedSlice, mySlice)
	fmt.Println("Copied Slice:", copiedSlice)
}
```

Q.2]

```go
package main

import (
	"fmt"
	"strconv"
	"sync"
)

func calculateSquareAndCubeSum(number int, squareSumChan chan int, cubeSumChan chan int, wg *sync.WaitGroup) {
	defer wg.Done()

	// Convert the number to a string to extract individual digits
	numStr := strconv.Itoa(number)
```

```
    // Initialize variables for sum of squares and cubes
    squareSum := 0
    cubeSum := 0

    // Iterate through each digit
    for _, digitStr := range numStr {
        digit, _ := strconv.Atoi(string(digitStr))

        // Calculate square and cube of the digit
        square := digit * digit
        cube := digit * digit * digit

        // Add to the sum of squares and cubes
        squareSum += square
        cubeSum += cube
    }

    // Send the sums to respective channels
    squareSumChan <- squareSum
    cubeSumChan <- cubeSum
}

func main() {
    // Input number
    num := 123

    // Create channels for sum of squares and cubes
    squareSumChan := make(chan int)
    cubeSumChan := make(chan int)

    // Create a wait group to synchronize goroutines
    var wg sync.WaitGroup

    // Launch goroutine to calculate sum of squares
    wg.Add(1)
    go calculateSquareAndCubeSum(num, squareSumChan, cubeSumChan, &wg)

    // Wait for goroutine to finish
    wg.Wait()

    // Receive results from channels
    squareSum := <-squareSumChan
    cubeSum := <-cubeSumChan

    // Print the results
    fmt.Printf("Number: %d\n", num)
    fmt.Printf("Sum of squares of individual digits: %d\n", squareSum)
    fmt.Printf("Sum of cubes of individual digits: %d\n", cubeSum)

    // Close channels
    close(squareSumChan)
    close(cubeSumChan)
}
```

SLIP 15

Q.1]

```
package main
```

```
import "fmt"

// addAndSubtract is a function that takes two integers
// and returns their sum and difference.
func addAndSubtract(a, b int) (int, int) {
	sum := a + b
	difference := a - b
	return sum, difference
}

func main() {
	// Call the function and receive multiple values
	resultSum, resultDiff := addAndSubtract(10, 5)

	// Print the results
	fmt.Printf("Sum: %d\n", resultSum)
	fmt.Printf("Difference: %d\n", resultDiff)
}
```

Q.2]

```
<?xml version="1.0" encoding="UTF-8"?>
<Person>
  <Name>John Doe</Name>
  <Age>30</Age>
  <City>New York</City>
</Person>
```

```
package main
```

```go
import (
	"encoding/xml"
	"fmt"
	"io/ioutil"
	"os"
)

// Person struct represents the structure of the XML data
type Person struct {
	XMLName xml.Name `xml:"Person"`
	Name    string   `xml:"Name"`
	Age     int      `xml:"Age"`
	City    string   `xml:"City"`
}

func main() {
	// Read XML file
	xmlFile, err := os.Open("data.xml")
	if err != nil {
		fmt.Println("Error opening XML file:", err)
		return
	}
	defer xmlFile.Close()

	// Read content of XML file
	xmlData, err := ioutil.ReadAll(xmlFile)
	if err != nil {
		fmt.Println("Error reading XML file:", err)
		return
	}

	// Create a Person structure to unmarshal the XML data
	var person Person

	// Unmarshal XML data into the structure
	err = xml.Unmarshal(xmlData, &person)
	if err != nil {
		fmt.Println("Error unmarshalling XML:", err)
		return
	}

	// Display the structure
	fmt.Println("Person Information:")
	fmt.Printf("Name: %s\n", person.Name)
	fmt.Printf("Age: %d\n", person.Age)
	fmt.Printf("City: %s\n", person.City)
}
```

SLIP 16

Q.1]

```
package main

import (
	"fmt"
	"rectangle" // Importing the user-defined package
)
```

```
func main() {
	// Input parameters for the rectangle
	length := 10.0
	width := 5.0

	// Calculate the area using the function from the user-defined package
	area := rectangle.Area(length, width)

	// Display the result
	fmt.Printf("Area of the rectangle with length %.2f and width %.2f is: %.2f\n",
length, width, area)
}
```

Q.2]

```
package main

import (
	"fmt"
	"time"
)

// delay function introduces a delay in milliseconds
func delay(ms time.Duration) {
	time.Sleep(ms * time.Millisecond)
}

func main() {
	for i := 0; i <= 10; i++ {
		fmt.Println(i)
		delay(250)
	}
}
```

SLIP 17

Q.1]

```
package main

import (
	"fmt"
)

// performOperations is a function that takes two numbers and returns their sum, difference, product, and quotient
func performOperations(a, b float64) (float64, float64, float64, float64) {
	sum := a + b
	difference := a - b
	product := a * b
	quotient := a / b

	return sum, difference, product, quotient
}

func main() {
	// Input numbers
	num1 := 10.0
	num2 := 5.0

	// Call the function to perform operations
	resultSum, resultDiff, resultProd, resultQuot := performOperations(num1, num2)
	// Display the results
	fmt.Printf("Sum: %.2f\n", resultSum)
	fmt.Printf("Difference: %.2f\n", resultDiff)
	fmt.Printf("Product: %.2f\n", resultProd)
	fmt.Printf("Quotient: %.2f\n", resultQuot)
}
```

Q.2]

```
package main

import (
	"fmt"
	"os"
)

func appendToFile(filename, content string) error {
	// Open the file in append mode, create it if it doesn't exist
	file, err := os.OpenFile(filename, os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0644)
	if err != nil {
		return err
	}
	defer file.Close()

	// Append the content to the file
	if _, err := file.WriteString(content); err != nil {
		return err
	}

	return nil
}
```

```
func main() {
	// File name and content to append
	filename := "tybca.txt"
	content := "This is additional content.\n"

	// Call the appendToFile function
	err := appendToFile(filename, content)
	if err != nil {
		fmt.Println("Error appending to file:", err)
		return
	}

	fmt.Println("Content appended to the file successfully.")
}
```

SLIP 18

Q.1]

```
package main

import "fmt"

// printMultiplicationTable function prints the multiplication table of a given number up to a specified limit
func printMultiplicationTable(number, limit int) {
	fmt.Printf("Multiplication Table of %d up to %d:\n", number, limit)

	for i := 1; i <= limit; i++ {
		result := number * i
		fmt.Printf("%d x %d = %d\n", number, i, result)
	}
}

func main() {
	// Input number and limit
	number := 5
	limit := 10

	// Call the function to print the multiplication table
	printMultiplicationTable(number, limit)
}
```

Q.2]

```
// calculator.go
package calculator

// Add performs addition of two numbers
func Add(a, b float64) float64 {
	return a + b
}

// Subtract performs subtraction of two numbers
func Subtract(a, b float64) float64 {
	return a - b
}

// Multiply performs multiplication of two numbers
func Multiply(a, b float64) float64 {
	return a * b
```

```
}

// Divide performs division of two numbers (returns 0 if division by zero)
func Divide(a, b float64) float64 {
	if b != 0 {
		return a / b
	}
	return 0
}
// main.go
package main

import (
	"fmt"
	"calculator" // Importing the user-defined package
)

func main() {
	var choice int
	var num1, num2 float64

	// Display menu
	fmt.Println("Choose operation:")
	fmt.Println("1. Addition")
	fmt.Println("2. Subtraction")
	fmt.Println("3. Multiplication")
	fmt.Println("4. Division")

	// Take user input for choice
```

```
}

    fmt.Print("Enter your choice (1-4): ")
    fmt.Scan(&choice)

    // Take user input for numbers
    fmt.Print("Enter first number: ")
    fmt.Scan(&num1)
    fmt.Print("Enter second number: ")
    fmt.Scan(&num2)

    // Perform the selected operation
    switch choice {
    case 1:
        result := calculator.Add(num1, num2)
        fmt.Printf("Result: %.2f\n", result)
    case 2:
        result := calculator.Subtract(num1, num2)
        fmt.Printf("Result: %.2f\n", result)
    case 3:
        result := calculator.Multiply(num1, num2)
        fmt.Printf("Result: %.2f\n", result)
    case 4:
        result := calculator.Divide(num1, num2)
        fmt.Printf("Result: %.2f\n", result)
    default:
        fmt.Println("Invalid choice")
    }
}
```

SLIP 19

Q.1]

```
package main


import "fmt"


// addAndSubtract is a function that takes two numbers
// and returns their sum and difference
func addAndSubtract(a, b float64) (float64, float64) {
	sum := a + b
	difference := a - b
	return sum, difference
}


func main() {
	// Input numbers
	num1 := 10.0
	num2 := 5.0

	// Call the function to perform addition and subtraction
	resultSum, resultDiff := addAndSubtract(num1, num2)

	// Display the results
	fmt.Printf("Sum: %.2f\n", resultSum)
	fmt.Printf("Difference: %.2f\n", resultDiff)
}
```

Q.2]

```
package main
```

```
import (
	"fmt"
	"os"
	"io/ioutil"
)

func main() {
	// Specify the file path
	filePath := "tybca.txt"

	// Open the file in read-only mode
	file, err := os.Open(filePath)
	if err != nil {
		fmt.Println("Error opening the file:", err)
		return
	}
	defer file.Close()

	// Read the content of the file
	content, err := ioutil.ReadAll(file)
	if err != nil {
		fmt.Println("Error reading the file:", err)
		return
	}

	// Display the content
	fmt.Printf("Content of %s:\n%s", filePath, content)
}
```

SLIP 20

Q.1]

```
package main

import (
	"fmt"
	"sync"
)

func produceNumbers(ch chan int, wg *sync.WaitGroup) {
	defer close(ch)
	for i := 1; i <= 5; i++ {
		ch <- i
	}
	wg.Done()
}

func main() {
	// Create a channel of integers
	numberChannel := make(chan int)

	// Create a wait group for synchronization
	var wg sync.WaitGroup
	wg.Add(1)

	// Start a goroutine to produce numbers and close the channel when done
	go produceNumbers(numberChannel, &wg)

	// Use a for range loop to receive values from the channel
```

```
    for num := range numberChannel {
        fmt.Println("Received:", num)
    }

    // Wait for the goroutine to finish
    wg.Wait()
}
```