Go Lang

**1. Channels code:**

package main

import (

"fmt"

"sync"

)

func add(numbers [] int, resultChan chan int, wg\*sync.WaitGroup){

defer wg.Done()

sum:=0

for \_,num:= range numbers{

sum+=num

}

resultChan<-sum

}

func main() {

numbers:=[]int{1,2,3,4,5,6,7,8,9,10}

resultChan:= make(chan int)

var wg sync.WaitGroup

wg.Add(2)

go add(numbers[:len(numbers)/2],resultChan,&wg)

go add(numbers[len(numbers)/2:],resultChan,&wg)

go func() {

wg.Wait()

close(resultChan)

}()

total:=0

for result:=range resultChan{

total+=result

}

fmt.Println("Total sum:",total)

}

**2. Defer statement:**

package main

import (

"fmt"

)

func testDefer(){

fmt.Println("Starting statement")

defer fmt.Println("Second statement")

fmt.Println("End statement")

}

func testDefer2(val int){

fmt.Println("Function 2 called",val)

}

func testDefer3(name string){

fmt.Println("Function 3 called",name)

}

func main() {

fmt.Println("Main function")

var value int

var name string

fmt.Println("Enter an int value:")

fmt.Scan(&value)

fmt.Println("Enter a name:")

fmt.Scan(&name)

defer testDefer3(name)

testDefer2(value)

testDefer()

}

**3. Buffered Channel**:

package main

import (

"fmt"

"time"

)

func main() {

//fmt.Println("Main function")

messageChannel:=make(chan string,2)

go func(){

for i:=0;i<=4;i++{

message:=fmt.Sprintf("Message %d",i)

messageChannel <-message

fmt.Printf("sent",message)

}

close(messageChannel)

}()

go func(){

for message:=range messageChannel{

fmt.Printf("Received:",message)

time.Sleep(time.Second)

}

}()

time.Sleep(5\*time.Second)

}

**4. Error Handling**

package main

import(

"fmt"

"errors"

)

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

if b==0{

return 0,errors.New("division by 0 not possible")

}

return a/b,nil

}

func main() {

result,err:=divide(100,2)

if err!=nil{

fmt.Println("Error")

fmt.Println(err)

}else{

fmt.Println("Result:" )

fmt.Println(result)

}

result1,err1:=divide(100,0)

if err!=nil{

fmt.Println("Error")

fmt.Println(err1)

}else{

fmt.Println("Result:" ,result1)

fmt.Println(result1)

}

fmt.Println("Hello World!")

}

**5.Panic function:**

package main

import(

"fmt"

)

func main() {

fmt.Println("Start of main")

panic("Something went wrong")

defer fmt.Println("Hello World!")

fmt.Println("End of main!")

}

**6.Recover function:**

package main

import(

"fmt"

)

func main() {

fmt.Println("Start of main")

defer func(){

r:=recover()

if r!=nil{

fmt.Println("Recovered from panic",r)

}

} ()

fmt.Println("Hello World!")

panic("Something went wrong")

fmt.Println("End of main!")

}

**7. Interface:**

package main

import(

"fmt"

)

type Shape interface{

Area() float64

}

type Rectangle struct{

width float64

height float64

}

func (r Rectangle) Area() float64{

return r.width\*r.height

}

type Circle struct{

Radius float64

}

func (c Circle) Area() float64{

return 3.14\*c.Radius\*c.Radius

}

func PrintArea(s Shape){

fmt.Printf("Area of shape: %f",s.Area())

}

func main() {

fmt.Println("Start of main")

rectangle:=Rectangle{width:3,height:5}

circle:=Circle{Radius:2}

PrintArea(rectangle)

PrintArea(circle)

}

**8. Implementation of multiple interfaces:**

package main

import "fmt"

type Shape interface {

Area() float64

}

type Colour interface{

GetColour() string

}

type Rectangle struct{

Width float64

Height float64

}

type ColouredRectangle struct{

Rectangle

ColourName string

}

func (r Rectangle) Area() float64{

return r.Width \* r.Height

}

func (c ColouredRectangle) GetColour() string{

return c.ColourName

}

func main() {

colouredRectangle:=ColouredRectangle{Rectangle:Rectangle{Width:4,Height:2},ColourName:"BLue"}

area:=colouredRectangle.Area()

fmt.Printf("Are of the coloured rectangle is: %f\t",area)

colour:=colouredRectangle.GetColour()

fmt.Printf("\n Colour is: \t",colour)

fmt.Println("Hello World!")

}

**9.Slices and Arrays:**

package main

import "fmt"

func main() {

slice:=[]int{1,2,}

length:=len(slice)

capacity:=cap(slice)

fmt.Println("Slice 1:\t",slice)

fmt.Println("\n length od slice: %d",length)

fmt.Println("\n capacity of slice:%d",capacity)

arr:=[5]int{6,7,8,9,10}

slice1:=arr[1:4]

fmt.Println("Slice 2:\t",slice1)

slice=append(slice,4,5)

fmt.Println("Slice 1:\t",slice)

fmt.Println("Hello World!")

}