1. Our program began with package main. What would the files in the fmt package begin with?
2. Given the array: x := [6]string{"a","b","c","d","e","f"} what would x[2:5] give you?

[c d e]

1. Write a function with one variadic parameter that finds the greatest number in a list of numbers.

package main

import "fmt"

func greatestNumber(nums ... int) int {

max := int(0)

for \_, num := range nums {

if num > max {

max = num

}

}

return max

}

func main() {

fmt.Println(greatestNumber(10,30,24,52,72,205,3000))

}

1. What is iota used for ? Write a small code defining constants first, second, third with values 0, 1 and 2. Now print them.

**Iota.** This is an enumerator for const creation. The Go compiler starts iota at 0 and increments it by one for each following constant. We can use it in expressions.We cannot use iota in an expression that must be evaluated at runtime—a const is determined at compile-time.

package main

import "fmt"

const (

num1 = iota

num2

num3

)

func main() {

// Use our iota constants.

fmt.Println(num1)

fmt.Println(num2)

fmt.Println(num3)

}

1. What is an equivalent one liner code for following- myArray := [3]int{} myArray[0] = 23 myArray[1] = 47 myArray[2] = 61

myArray:= []int{23,47, 61}

1. Write a function, that takes integers as its argument and returns the number of arguments and its sum-eg. addAll(2,5,3) should return 3 and 10 . Similarily, addAll(5,6,7,10) should return 4 and 28

package main

import "fmt"

func sum(nums ...int) {

fmt.Print(nums, " ")

total := 0

for \_, num := range nums {

total += num

}

fmt.Println(total)

}

func main() {

nums1 := []int{1, 2, 3, 4}

sum(nums1...)

fmt.Println("lenght=",len (nums1))

nums2 := []int{1, 2, 3}

sum(nums2...)

fmt.Println("lenght=",len (nums2))

}

1. What are defer, panic and recover? How do you recover from a run-time panic?

**Panic** is a built-in function that stops the ordinary flow of control and begins *panicking*. When the function F calls panic, execution of F stops, any deferred functions in F are executed normally, and then F returns to its caller. To the caller, F then behaves like a call to panic. The process continues up the stack until all functions in the current goroutine have returned, at which point the program crashes. Panics can be initiated by invoking panic directly. They can also be caused by runtime errors, such as out-of-bounds array accesses.

**Recover** is a built-in function that regains control of a panicking goroutine. Recover is only useful inside deferred functions. During normal execution, a call to recover will return nil and have no other effect. If the current goroutine is panicking, a call to recover will capture the value given to panic and resume normal execution.

A **defer statement** pushes a function call onto a list. The list of saved calls is executed after the surrounding function returns. Defer is commonly used to simplify functions that perform various clean-up actions.

1. Add a new method to the Shape interface called perimeter which calculates the perimeter of a shape. Implement the method for Circle and Rectangle.

package main

import "fmt"

type Shape interface {

Perimeter() int

}

type Rectangle struct {

length, width int

}

type Circle struct{

radius float64

}

func (r Rectangle) Perimeter() int {

return 2\*(r.length + r.width)}

func(c Circle) Perimeter() float64{

return 2 \* 3.14 \* c.radius

}

func main() {

r := Rectangle{length:5, width:3}

fmt.Println("Rectangle details are: ", r)

fmt.Println("Rectangle Perimeter is: ", r.Perimeter())

c:= Circle{radius:3}

fmt.Println("Circle details are: ", c)

fmt.Println("Circle Perimeter is: ", c.Perimeter())

}