**Java Assessment -7**

**Practical Questions:**

**Q1. How to swap two numbers without using a third variable?**

**Ans:** **Step 1**: Define a function that accepts two numbers, type is int.

**Step 2**: Find b = a + b;

**Step 3**: Then a = b – a and b = b – a

**Program:**

Package main

Import “fmt”

Func swap (a, b int ){

fmt.Println(“Before swapping, numbers are %d and %d\n”, a, b)

b = a + b

a = b – a

b = b – a

fmt.Println(“After swapping, numbers are %d and %d\n”, a, b)

}

Func main(){

Swap (23, 45)

Swap (56, 100)

}

**Q2. Golang Program to check if a vowel is present in the string?**

**Ans: Program:**

package main

import "fmt"

func main() {

    var user\_string string

    var check\_for\_vowel bool

    fmt.Print("Enter a String to check if vowel is present or not:")

    fmt.Scanf("%s\n", &user\_string)

    for \_, char := range user\_string {

        if char == 'a' || char == 'e' || char == 'i' || char == 'o' || char == 'u' {

            check\_for\_vowel = true

        }

    }

    if check\_for\_vowel {

        fmt.Println("The String Contains vowel.")

    } else {

        fmt.Println("There are No vowel in the String.")

    }

}

**Q3. Golang program to check if the given number is Prime?**

**Ans: Program:**

package main

import "fmt"

func CheckPrime(number int) {

    isPrime := true

    if number == 0 || number == 1 {

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

    } else {

        for i := 2; i <= number/2; i++ {

            if number%i == 0 {

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

                isPrime = false

                break

            }

        }

        if isPrime == true {

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

        }

    }

}

func main() {

    var user\_input int

    fmt.Print("Enter a String to check if it is prime a number or not:")

    fmt.Scanf("%s\n", &user\_input)

    CheckPrime(user\_input)

}

}

**Q4. Please write a Golang program to add tab, new line and print "Mr. ABC" along with double quotes from a string?**

**Ans: Program:**

package main

import "fmt"

func main() {

    fmt.Println("\n\t\"Mr. ABC\"")

}

**Q5. Please write a Golang program to find sum and average from the given array [2,4,6,8,34,5,7,3,5,67,4]**

**Ans: Program:**

package main

import "fmt"

func main() {

// declaring an array of values

array := []int{2, 4, 6, 8, 34, 5, 7, 3, 5, 67, 4}

// size of the array

n := 11

// declaring a variable

// to store the sum

sum := 0

// traversing through the

// array using for loop

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

// adding the values of

// array to the variable sum

sum += (array[i])

}

// declaring a variable

// avg to find the average

avg := (float64(sum)) / (float64(n))

// typecast all values to float

// to get the correct result

fmt.Println("Sum = ", sum, "\nAverage = ", avg)

}

**Q6. Please write a program to find factorial of the number input from the keyboard?**

**Ans: Program:**

package main

import "fmt"

func factorial(x uint) uint {

    if x == 0 {

        return 1

    }

    return x \* factorial(x-1)

}

func main() {

    var user\_input uint

    fmt.Print("Enter a Number to know its Factorial:")

    fmt.Scanf("%s\n", &user\_input)

    println(factorial(user\_input))

}

**Q7. Create a Pyramid of Characters in in Golang?**

**Ans: Program:**

package main

import "fmt"

func main() {

    var rows int = 5

    var k int

    for i := 1; i <= rows; i++ {

        k = 0

        for space := 1; space <= rows-i; space++ {

            fmt.Print("  ")

        }

        for {

            fmt.Print("\* ")

            k++

            if k == 2\*i-1 {

                break

            }

        }

        fmt.Println("")

    }

}

**Q8. Please write a program to concatenate two given arrays [3,5,76,3,6,3,5,6,3] and [2,3,65,7,4,3,6,3,56,3]?**

**Ans: Program:**

package main

import "fmt"

func main() {

    var slice\_1 = []int{3, 5, 76, 3, 6, 3, 5, 6, 3}

    var slice\_2 = []int{2, 3, 65, 7, 4, 3, 6, 3, 56, 3}

    slice\_3 := append(slice\_1, slice\_2...)

    fmt.Printf("slice\_1: %v\n", slice\_1)

    fmt.Printf("slice\_2: %v\n", slice\_2)

    fmt.Printf("Concantination on slice\_1 and slice\_2: %v\n", slice\_3)

}

**Q9. Find second largest number in an array [3,5,76,3,6,3,5,6,3]?**

**Ans: Program:**

package main

import "fmt"

func main() {

    var large1 int = 0

    var large2 int = 0

    arr := [...]int{3, 6, 76, 7, 6, 55, 55, 6, 77}

    large1 = arr[0]

    for i := 1; i <= 4; i++ {

        if large1 < arr[i] {

            large2 = large1

            large1 = arr[i]

        } else if large2 < arr[i] {

            large2 = arr[i]

        }

    }

    fmt.Println("Second largest element is: ", large2)

}

**Q10. Please write a Golang program to print your name in the text file?**

**Ans: Program:**

package main

import (

    "fmt"

    "io/ioutil"

    "log"

    "os"

)

func CreateFile() {

    file, err := os.Create("test.txt") // Truncates if file already exists, be careful!

    if err != nil {

        log.Fatalf("failed creating file: %s", err)

    }

    defer file.Close() // Make sure to close the file when you're done

    len, err := file.WriteString("Sudeb Dolui")

    if err != nil {

        log.Fatalf("failed writing to file: %s", err)

    }

    fmt.Printf("\nLength: %d bytes", len)

    fmt.Printf("\nFile Name: %s", file.Name())

}

func main() {

    fmt.Printf("########Create a file and Write the content #########\n")

    CreateFile()

}

**Theoretical Questions:**

**Q1. What’s the difference between a goroutine and an operating system thread?**

**Ans:**

|  |  |
| --- | --- |
| **Goroutine** | **Thread** |
| Goroutines are managed by the go runtime. | Operating system threads are managed by kernel. |
| Goroutine are not hardware dependent. | Threads are hardware dependent. |
| Goroutines have easy communication medium known as channel. | Thread does not have easy communication medium. |
| Goroutine does not have ID because go does not have Thread Local Storage. | Threads have their own unique ID because they have Thread Local Storage. |
| Goroutines are cheaper than threads. | The cost of threads is higher than goroutine. |
| They are cooperatively scheduled. | They are preemptively scheduled. |
| They have fasted startup time than threads. | They have slow startup time than goroutines. |
| Goroutine has growable segmented stacks. | Threads does not have growable segmented stacks. |

**Q2. Can constants be computed in Go?**

**Ans:** **Constants in Go:**

* Must be able to be assigned at compile time. The value of a const can’t be the result of a runtime calculation
* Run faster because the compiler can make specific optimizations
* Cannot change. The compiler will not allow them to be re-assigned

In Go, constants provide complete safety in regards to the value they hold. They cannot be computed (making them used less often), but are guaranteed to always reference the same value.

**Q3. What does the Go ecosystem use for package and dependency management?**

**Ans:**  Go modules or go.mod is used which is one of many package managers to deal with dependencies in Go.

**Q4. How would you succinctly swap the values of two variables in Go?**

**Ans:** In the program if we add the line

x, y := y, x  
 Thinking that x and y are our declared variables. Then we can succinctly swap the values

**Q5. Do you have any preferences for error handling methodologies in Go?**

**Ans:** Defer, panic and recover are the methods used to handle errors in GoLang.

**Q6. What is a pointer and when would you use it?**

**Ans:** Pointers are used to store and manage the addresses of [dynamically allocated](https://en.wikipedia.org/wiki/Dynamic_memory_allocation) blocks of memory. Such blocks are used to store data objects or arrays of objects. Most structured and object-oriented languages provide an area of memory, called the *heap* or *free store*, from which objects are dynamically allocated.

**Q7. Describe the difference between sync.Mutex and sync.RWMutex?**

**Ans: sync.Mutex and sync.RWMutex:**

Mutex and RWMutex are not associated with goroutines, but RWMutex is obviously more suitable for scenarios with more reads and less writes. For read performance only, RWMutex is higher than Mutex, because multiple reads of rwmutex can coexist.

**sync.Mutex:**

A Mutex is a mutual exclusion lock. The zero value for a Mutex is an unlocked mutex.

A Mutex must not be copied after first use.

A Mutex is a method used as a locking mechanism to ensure that only one Goroutine is accessing the critical section of code at any point of time. This is done to prevent race conditions from happening. **sync** package contains the Mutex. Two methods defined on Mutex:

* func (m \*Mutex) Lock()

Lock locks m. If the lock is already in use, the calling goroutine blocks until the mutex is available.

* func (m \*Mutex) Unlock()

Unlock unlocks m. It is a run-time error if m is not locked on entry to Unlock.

A locked Mutex is not associated with a particular goroutine. It is allowed for one goroutine to lock a Mutex and then arrange for another goroutine to unlock it.

**sync.RWMutex:**

Using RWMutex, a reader/writer mutual exclusion lock which allows any number of readers to hold the lock or one writer. This tends to be more efficient than using a full mutex in situations where you have a high ratio of reads to writes.

A RWMutex must not be copied after first use. RWMutex is based on Mutex. On the basis of Mutex, read and write semaphores are added, and the number of read locks similar to reference counting is used.

There are methods defined on RWMutex :

* func (rw \*RWMutex) Lock()

Lock locks rw for writing. If the lock is already locked for reading or writing, Lock blocks until the lock is available.

* func (rw \*RWMutex) RLock()

RLock locks rw for reading.

It should not be used for recursive read locking; a blocked Lock call excludes new readers from acquiring the lock. See the documentation on the RWMutex type.

* func (rw \*RWMutex) RUnlock()

RUnlock undoes a single RLock call; it does not affect other simultaneous readers. It is a run-time error if rw is not locked for reading on entry to RUnlock.

* func (rw \*RWMutex) Unlock()

Unlock unlocks rw for writing. It is a run-time error if rw is not locked for writing on entry to Unlock.

As with Mutexes, a locked RWMutex is not associated with a particular goroutine. One goroutine may RLock (Lock) a RWMutex and then arrange for another goroutine to RUnlock (Unlock) it

**Q8. Consider the following code. What will be the value of s1?**

**Ans:** primes : = [6] int {2, 3, 5, 7, 11, 13 }

S1 : = primes [1:4]

S1 will be : [ 3 5 7 ]

When slicing an existing array or slice the first index is inclusive while the last index is exclusive. If an index is omitted on one side of the colon, then all values until the edge of the original slice are included in the result.

**Q9. Are channels and maps safe for concurrent access?**

**Ans:** Maps are not safe for concurrent use: it’s not defined what happens when you read and write to them simultaneously. If you need to read from and write to a map from concurrently executing goroutines, the accesses must be mediated by some kind of synchronization mechanism.

**Q10. How would you sort a slice of custom structs?**

**Ans:** GO has a **sort** package that provides utility primitives for the sorting of slices and user-defined types. Any collection can be sorted by the **Sort** function of **sort** package of GO it if implements the **sort.Interface**.