**Golang Intermediate Challenge**

1. **Please provide correct code and run it**

package main

import "fmt"

func main() {

fmt.println("Hello, World!")

}

1. **Please provide correct code and run it**

mport "fmt"

func main() {

var age int

age = 30

name := "Alice"

fmt.Printf("Name:, Age: ", name, age)

}

1. **Please provide correct code and run it**

package main

import "fmt"

func main() {

// String Data Type

var username string = "Shadab"

fmt.Println(username)

fmt.Printf("Variables Type is: %T \n", username)

// Boolean Data Type

var isloggedin bool = true

fmt.Println(isloggedin)

fmt.Printf("Variables Type is: %T \n", isloggedin)

// uint8 Data Type

var smallValue uint8 = 255

fmt.Println(smallValue)

fmt.Printf("Variables Type is: %T \n", smallValue)

// float32 Data Type

var smallValueFloat float = 255.3273839736837837

fmt.Println(smallValueFloat)

fmt.Printf("Variables Type is: %T \n", smallValueFloat)

// float64 Data Type

var bigValueFloat float = 255.3273839736837837

fmt.Println(bigValueFloat)

fmt.Printf("Variables Type is: %T \n", bigValueFloat)

// Default values and some aliases

var anotherVariable int

fmt.Println(anotherVariable)

fmt.Printf("Variables Type is: %T \n", anotherVariable)

var anotherVariableString string

fmt.Println(anotherVariableString)

fmt.Printf("Variables Type is: %T \n", anotherVariableString)

// implicit Type

var website = "skillwise.com"

fmt.Println(website)

fmt.Printf("Variables Type is: %T \n", website)

// no var keyword declaration

totalUsers := 49736

fmt.Println(totalUsers)

fmt.Printf("Variables Type is: %T \n", totalUsers)

fmt.Println(loginToken)

fmt.Printf("Variables Type is: %T \n", loginToken)

}

1. **Print Output without infinite iterations**

import "fmt"

func main() {

i := 0

for {

fmt.Println(i)

i++

}

}

1. **Print Output without infinite iterations**

package main

import (

"bufio"

"fmt"

"os"

)

func main() {

fmt.Println("Input Varialbe:")

reader := bufio.NewReader()

// comma ok // comma err

fmt.Println("Enter the rating for travel")

input, \_ := reader.ReadString('\n')

fmt.Println("Thanks for rating, ", input)

}

1. **Create a Go program that does the following:**

* Producer: Generates numbers from 1 to 100 and sends them to a channel.
* Consumer: Reads from the channel, squares each number, and sends the results to another channel.
* Aggregator: Reads the squared numbers from the second channel, sums them up, and prints the final sum.
* **Requirements**
* Use goroutines for the producer, consumer, and aggregator.
* Use channels to communicate between goroutines.
* Ensure that the program waits for all goroutines to finish before exiting

1. **Create a Go program that does the following:**
2. Task Generator: Generates a series of tasks, where each task is a function that sleeps for a random amount of time (between 100ms and 500ms) and then returns a result (the sleep duration in milliseconds).
3. Worker Pool: Uses a pool of worker goroutines to process these tasks concurrently.
4. Aggregator: Collects the results of all tasks and prints the total time taken to complete all tasks.
5. **Requirements**
6. Use goroutines and channels to handle task generation, processing, and result aggregation.
7. Ensure that the program waits for all tasks to complete before printing the total time taken.

**8. Please complete this program**

// Golang program to illustrate the working of the slice components

package main

import "fmt"

func main() {

// Creating an array

// Display array

// Creating a slice

// Display slice

// Display length of the slice

// Display the capacity of the slice

}

**9. Please complete this program**

// This program demonstrates nested structures by representing a company with employees.

package main

import "fmt"

// Define a structure 'Employee'

type Employee struct {

// Employee's first name

// Employee's last name

// Employee's age

}

// Define a structure 'Company' with a slice of employees

type Company struct {

// Company's name

// Slice of Employee structures

}

func main() {

// Create instances of 'Employee'

emp1 := Employee{"John", "Doe", 30} // First employee

emp2 := Employee{"Alice", "Smith", 25} // Second employee

// Create an instance of 'Company' with employees

company := Company{

Name: "Acme Inc", // Company name

// List of employees

}

// Access and print company name and employee details

fmt.Println("Company Name:", company.Name) // Print company name

fmt.Println("Employees:") // Print employees header

for \_, emp := range company.Employees { // Iterate over employees

fmt.Printf(" %s %s, Age: %d\n", emp.FirstName, emp.LastName, emp.Age) // Print employee details

}

}

**END**