**SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING**

**ADDIS ABABA INSTITUTE OF TECHNOLOGY, ADDIS ABABA UNIVERSITY**

Topic – Golang Basic Programming

**Activity  1: Setting Up and Running a Go Program**  
  
**Objective:** Set up Go on your local machine and write a simple Go program to print "Hello, World!".

**Instructions:**

1. **Install Golang**

* Download and install Go from the official website: https://golang.org/dl/.
* Verify installation by running go version in your terminal.

1. **Write and Run "Hello World":**

* Create a new file called main.go.
* Write the following

**Code:**

package main  
  
import "fmt"  
  
func main() {  
    fmt.Println("Hello, World!")  
}  
  
**Explanation:**

* package main: Defines this program as part of the main package, which is required for standalone executables in Go.
* import "fmt": Imports the fmt package, which provides formatting utilities like Println for output.
* func main(): The main function is the entry point of every Go program. Code execution starts here.
* fmt.Println("Hello, World!"): Prints "Hello, World!" to the console using the Println function from the fmt package.

1. **Run the Program**

* Open your terminal and run:

                      go run main.go

1. **Build the Program**

* Compile the program into an executable:

                     go build main.go  
  
  
**Activity 2: Variables and Data Types**  
  
**Objective:** Learn how to declare and use variables in Go.

**Instructions**:

1. **Create a new Go file** called variables.go.
2. **Write code** to declare and initialize variables, then print their values.

**Code:**

package main  
import "fmt"  
func main() {  
    // Declaring variables with explicit types  
    var a int = 10  
    var b float64 = 20.5  
    var c string = "Golang"  
    var d bool = true  
  
    // Short variable declaration (type inferred)  
    e := "Short Declaration"  
  
    // Print the values of the variables  
    fmt.Println("Integer:", a)  
    fmt.Println("Float:", b)  
    fmt.Println("String:", c)  
    fmt.Println("Boolean:", d)  
    fmt.Println("Short Declaration:", e)  
}  
  
**Explanation:**

* var a int = 10: Declares an integer variable a with an explicit type int and assigns it the value 10.
* var b float64 = 20.5: Declares a floating-point variable b with type float64.
* var c string = "Golang": Declares a string variable c initialized with the value "Golang".
* var d bool = true: Declares a boolean variable d with the value true.
* e := "Short Declaration": Uses the shorthand := to declare and initialize the variable e without specifying a type (Go infers the type as string).
* fmt.Println: Prints the variable values to the console.

**Activity 3: Functions and Control Flow**  
  
**Objective**: Learn how to define functions, use if statements, and implement loops.

**Instructions**:

1. **Create a new Go file** called functions.go.
2. **Write code** to define a function that adds two numbers, and demonstrate control flow.

Code :

package main  
  
import "fmt"  
  
// Function to add two numbers  
func add(a int, b int) int {  
    return a + b  
}  
  
func main() {  
    result := add(5, 3)  
    fmt.Println("Sum:", result)  
  
    // Control flow example: if-else statement  
    if result > 5 {  
        fmt.Println("The result is greater than 5")  
    } else {  
        fmt.Println("The result is 5 or less")  
    }  
  
    // Loop example: for loop  
    for i := 0; i < 5; i++ {  
        fmt.Println("Loop iteration:", i)  
    }  
}  
  
**Explanation:**

* func add(a int, b int) int: Defines a function add that takes two integers (a and b) and returns their sum.
* result := add(5, 3): Calls the add function with arguments 5 and 3, storing the result in the variable result.
* if result > 5: An if statement checks if the result is greater than 5 and prints different messages based on the condition.
* for i := 0; i < 5; i++: A simple for loop that iterates 5 times, printing the value of i in each iteration.

**Activity 4: Arrays, Slices, and Maps**  
  
**Objective**: Work with arrays, slices, and maps, which are common collections in Go.

**Instructions:**

1. **Create a new Go file** called collections.go.
2. **Write code** to declare and manipulate arrays, slices, and maps.

**Code:**  
package main  
  
import "fmt"  
  
func main() {  
    // Array  
    arr := [3]int{1, 2, 3}  
    fmt.Println("Array:", arr)  
  
    // Slice (dynamic array)  
    slice := []int{4, 5, 6}  
    slice = append(slice, 7) // Add element to slice  
    fmt.Println("Slice:", slice)  
  
    // Map (key-value pairs)  
    myMap := make(map[string]int)  
    myMap["Alice"] = 25  
    myMap["Bob"] = 30  
    fmt.Println("Map:", myMap)  
    fmt.Println("Alice's age:", myMap["Alice"])  
  
    // Looping over a slice  
    for i, v := range slice {  
        fmt.Printf("Index: %d, Value: %d\n", i, v)  
    }  
  
    // Looping over a map  
    for key, value := range myMap {  
        fmt.Printf("%s is %d years old\n", key, value)  
    }  
}  
  
**Explanation:**

* arr := [3]int{1, 2, 3}: Declares an array of three integers.
* slice := []int{4, 5, 6}: Declares a slice (dynamic array) and then appends a new element (7).
* myMap := make(map[string]int): Declares a map with string keys and integer values.
* myMap["Alice"] = 25: Adds a key-value pair to the map.
* for i, v := range slice: A for loop that iterates over the elements of the slice, where i is the index and v is the value.
* for key, value := range myMap: Iterates over the map's key-value pairs.

**Activity 5: Goroutines (Concurrency Basics)**  
  
**Objective:** Understand basic concurrency by using Goroutines.

**Instructions:**

1. **Create a new Go file** called goroutines.go.
2. **Write code** that demonstrates how to run a function concurrently with a goroutine.

**Code:**  
package main  
  
import (  
    "fmt"  
    "time"  
)  
  
func printNumbers() {  
    for i := 1; i <= 5; i++ {  
        fmt.Println(i)  
        time.Sleep(1 \* time.Second)  
    }  
}  
  
func printLetters() {  
    for i := 'A'; i <= 'E'; i++ {  
        fmt.Printf("%c\n", i)  
        time.Sleep(1 \* time.Second)  
    }  
}  
  
func main() {  
    // Run printNumbers as a goroutine (concurrently)  
    go printNumbers()  
  
    // Run printLetters in the main goroutine  
    printLetters()  
  
    // Allow time for the printNumbers goroutine to finish  
    time.Sleep(6 \* time.Second)  
    fmt.Println("Main function finished")  
}  
  
Explanation:

* go printNumbers(): Launches the printNumbers function as a goroutine, allowing it to run concurrently.
* printLetters(): Runs the printLetters function in the main goroutine (main execution thread).
* time.Sleep(6 \* time.Second): Ensures that the program waits long enough for the printNumbers goroutine to finish before exiting.

**Activity 6: Channels (Communicating Between Goroutines)**

**Objective:** Learn how to use channels to communicate between goroutines.

**Instructions:**

1. **Create a new Go file** called channels.go.
2. **Write code** that demonstrates sending and receiving data via channels.

**Code:**

package main  
  
import "fmt"  
  
// Function that sends numbers to the channel  
func sendData(ch chan int) {  
    for i := 0; i < 5; i++ {  
        ch <- i // Send data to channel  
    }  
    close(ch) // Close the channel when done  
}  
  
func main() {  
    // Create a channel  
    ch := make(chan int)  
  
    // Start a goroutine that sends data  
    go sendData(ch)  
  
    // Receive data from the channel  
    for val := range ch {  
        fmt.Println("Received:", val)  
    }  
  
    fmt.Println("Channel closed, program finished.")  
}  
  
Explanation:

* ch := make(chan int): Creates a channel that passes integer values between goroutines.
* ch <- i: Sends data to the channel.
* for val := range ch: Receives data from the channel until it is closed.
* close(ch): Closes the channel to signal that no more data will be sent.

**Activity 7: Error Handling and Defer**  
  
**Objective**: Learn how to handle errors and use defer to ensure resources are properly released.

**Instructions**:

1. **Create a new Go file** called channels.go.
2. **Write code** that demonstrates sending and receiving data via channels.

**Code:**  
package main  
  
import (  
    "fmt"  
    "os"  
)  
  
// Function that opens a file and reads it (simplified for this example)  
func readFile(filename string) error {  
    file, err := os.Open(filename)  
    if err != nil {  
        return err // Return the error if file opening fails  
    }  
    defer file.Close() // Ensure the file is closed after function returns  
  
    fmt.Println("File opened successfully:", filename)  
    return nil  
}  
  
func main() {  
    err := readFile("test.txt")  
    if err != nil {  
        fmt.Println("Error:", err)  
    } else {  
        fmt.Println("File read successfully.")  
    }  
}  
  
Explanation:

* file, err := os.Open(filename): Tries to open a file and returns an error if the file doesn’t exist or can't be opened.
* defer file.Close(): Ensures the file is closed after the function returns, even if there’s an error.
* if err != nil: Checks if there’s an error and handles it by printing the error message.

**What to submit**

* Submit the code file containing the program for each activity
* A screenshot showing the terminal output for each activity and the output showing the result of each activities