**Go Concurrency**

Go routine and channels are a lightweight built in features for managing concurrency and communication between several function executing at a same time.

This way once can execute the code that outside if the main program.

Go has below keywords like

go

chan

Concurrency with go routine

package main

import (

    "fmt"

    "time"

)

func timesThree(number int) {

    fmt.Println(number \* 3)

}

func main() {

    fmt.Println("We are executing a go routine")

    go timesThree(3)

    fmt.Println("Done!")

    time.Sleep(time.Second)

}

PS F:\Training\Golang\Program> go run concurrency.go

We are executing a go routine

Done!

9

We have successfully run the concurrency execution

Main program will creates go routine for executing timesThree function

There for fmt.Println(“Done!”) will executes before go routine

But, what if we need some value returning from that function to continue with our main function.

Thats where channel comes and save the day.

package main

import (

    "fmt"

)

func timesThree(number int, ch chan int) {

    result := number \* 3

    fmt.Println(number \* 3)

    ch <- result

}

func main() {

    fmt.Println("We are executing a goroutine")

    ch := make(chan int)

    go timesThree(3, ch)

    result := <-ch

    fmt.Printf("The result is: %v", result)

}

We are executing a goroutine

9

The result is: 9

Once the main program executes the goroutines, it waits for the channel to get some data before continuing, therefore fmt.Println("The result is: %v", result) is executed after the goroutine returns the result. This doesn’t mean that the main program will wait for the full goroutine to execute, just until the data is served to the channel.