Go Training

Session 10

error

```go
package main
import "fmt"

// type error interface {
//  Error() string
// }

type errorString struct {
    str string
}

func (es *errorString) Error() string {
    return es.str
}

func NewErr(txt string) error {
    return &errorString{"error is " + txt}
}

func main() {

    var err error
    err = NewErr("check")
    fmt.Println(err)
}
```

error

```go
package main
import "fmt"

type errorString struct {
    str string
    negVal int
}

func (es *errorString) Error() string {
    return fmt.Sprintf("Error occurred %v denominator is %v", es.str, es.negVal)
}

func newErr(txt string, val int) error {
    return &errorString{str:txt, negVal:val}
}

func main() {

    res,err := divide(3,0)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(res)
}

func divide(x,y int) (result float32, err error) {
    if y==0 {
        err = newErr("zero denominator",y)
        return
    }
    result = float32(x)/float32(y)
    return
}
```

panic

- When Go breaks the program in runtime it is called runtime panics
- Unlike error, panic has the potential to break the program
- A panic can be triggered manually

```go
package main
import "fmt"

func main() {

    slc := []int{0,0,0,0}
    val := slc[4]
    fmt.Println(val)
}

```

```
G:\dsktp data\Go Training\examples>go run hello.go
panic: runtime error: index out of range [4] with length 4

goroutine 1 [running]:
main.main()
        G:/dsktp data/Go Training/examples/hello.go:7 +0x1b
exit status 2
```

panic

```go
package main
import "fmt"

func main() {

    slc := []int{0,0,0,0}

    fmt.Println(getValue(slc, 4))
}


func getValue(slc []int, index int) int{
    if(index >= len(slc)) {
        panic("not possible to access out of bound index")
    } else{
        return slc[index]
    }
}

```

Recovering from panic

```go
package main
import "fmt"

func main() {

    fmt.Println("main started")
    firstCall()
}


func firstCall() {
    fmt.Println("firstCall started")
    defer secondCall()
    panic("firstCall panicked")
    fmt.Println("firstCall done")
}

func secondCall() {
    fmt.Println("secondCall started")
    r := recover()
    if r!=nil {
        fmt.Println("recovering after panic value", r)
    }
    fmt.Println("secondCall done")
}

```

Revisiting Functions

- Function as type

```go
package main
import "fmt"


func main() {
    firstResult := thirdFunc("input", firstFunc)
    secondResult := thirdFunc("input", secondFunc)
    fmt.Println(firstResult)
    fmt.Println(secondResult)
}

func firstFunc(str string) string{
    return "first "+str
}

func secondFunc(str string) string{
    return "second "+str
}

func thirdFunc(str string, f func(string) string) string{
    output := f(str)
    return output
}
```

Revisiting Functions

- Function as value

```go
package main
import "fmt"

var firstFunc = func(str string) string{
    return "first "+str
}

func main() {
    output := firstFunc("input")
    fmt.Println(output)
}
```

Revisiting Functions

- Immediately invoked function

```go
package main
import "fmt"


func main() {
    output := func(str string) string{
        return "first "+str
    }("input")
    fmt.Println(output)
}

```

Closure

- Closure is a function value that references variables from outside its body

```go
package main
import "fmt"


func main() {
    innerFunc := outerFunc()
    x := innerFunc()
    fmt.Println(x)
    y := innerFunc()
    fmt.Println(y)
}

func outerFunc() func()int {
    initialValue := 0
    return func() int{
        initialValue++
        return initialValue
    }
}

```

Thank You