Go Training

Session 7

Methods

- When a method has a pointer receiver

```go
package main

import "fmt"

type Student struct {
    firstName string
    rollNo int
}

func (s Student) chnageFirstName(newName string) {
    s.firstName=newName
    return
}

func main() {

    aks := Student{"Akanksha", 2}
    aks.chnageFirstName("Ashlesha")

    fmt.Println(aks) //{Akanksha 2}
}
```

```go
package main

import "fmt"

type Student struct {
    firstName string
    rollNo int
}

func (sp *Student) chnageFirstName(newName string) {
    (*sp).firstName=newName
    return
}

func main() {

    aks := Student{"Akanksha", 2}
    (&aks).chnageFirstName("Ashlesha")

    fmt.Println(aks) //{Ashlesha 2}
}
```

Methods

- When a method has a pointer receiver

```go
package main

import "fmt"

type Student struct {
    firstName string
    rollNo int
}

func (sp *Student) chnageFirstName(newName string) {
    (*sp).firstName=newName
    return
}

func main() {

    aks := Student{"Akanksha", 2}
    (&aks).chnageFirstName("Ashlesha")

    fmt.Println(aks) //{Ashlesha 2}
}
```

```go
package main

import "fmt"

type Student struct {
    firstName string
    rollNo int
}

func (sp *Student) chnageFirstName(newName string) {
    sp.firstName=newName
    return
}

func main() {

    aks := Student{"Akanksha", 2}
    aks.chnageFirstName("Ashlesha")

    fmt.Println(aks) //{Ashlesha 2}
}
```

Methods

- Methods can accept both pointer and values
  - You can define a method on value receiver and call it on pointer instead
  - You can define a method on pointer receiver and call it on value instead

```go
package main

import "fmt"

type Student struct {
    firstName string
    rollNo int
}

func (s Student) chnageFirstName(newName string) {
    s.firstName=newName
    return
}

func main() {

    aks := Student{"Akanksha", 2}
    (&aks).chnageFirstName("Ashlesha")

    fmt.Println(aks) //{Akanksha 2}
}

```

```go
package main

import "fmt"

type Student struct {
    firstName string
    rollNo int
}

func (sp *Student) chnageFirstName(newName string) {
    sp.firstName=newName
    return
}

func main() {

    aks := Student{"Akanksha", 2}
    aks.chnageFirstName("Ashlesha")

    fmt.Println(aks) //{Ashlesha 2}
}

```

Interfaces

- An interface is a collection of method signatures that a Type can implement using methods.
  - If a Type implements all the methods with names and signatures defined in an Interface then that Type implements that Interface.

- How to declare an interface?

```go
package main

import "fmt"

type mathOps interface{
    mult() float32
    add() float32
}

type mathValues struct{
    val1 float32
    val2 float32
}

func (mv mathValues) mult() float32{
    return mv.val1 * mv.val2
}

func (mv mathValues) add() float32{
    return mv.val1 + mv.val2
}

func main() {
    var mo mathOps
    fmt.Println(mo) // <nil>

    mo = mathValues{4.0, 5.2}
    fmt.Println(mo.mult())  // 20.8

    mv := mathValues{4.0, 5.2}

    fmt.Println(mo==mv) // true
}
```

Interfaces

```go
package main
import "fmt"

type figure interface{
    area() float32
}

type rectangle struct{
    length float32
    width float32
}
type circle struct{
    radius float32
}
func (r rectangle) area() float32{
    return r.length * r.width
}

func (c circle) area() float32{
    return 3.14 * c.radius * c.radius
}

func main() {
    var s figure

    s = rectangle{4.0, 3}
    fmt.Println(s.area())  // 12

    s = circle{2}

    fmt.Println(s.area())  // 12.56
}
```

Interfaces

- Empty Interface
  - Empty interface has zero methods
  - Represented by interface{}
  - All Types implement empty interface implicitly

```go
package main
import "fmt"


type rectangle struct{
    length float32
    width float32
}
type circle struct{
    radius float32
}
func (r rectangle) area() float32{
    return r.length * r.width
}

func main() {
    flexibleFunc(rectangle{1,2})
    fmt.Println()
    flexibleFunc(circle{3})
}

func flexibleFunc(i interface{}) {
    fmt.Printf("you passed type %T and with value %v", i, i)
}
```

Interfaces

```go
package main
import "fmt"


type rectangle struct{
    length float32
    width float32
}
type circle struct{
    radius float32
}
func (r rectangle) area() float32{
    return r.length * r.width
}

func main() {
    flexibleFunc(rectangle{1,2})
    fmt.Println()
    flexibleFunc(circle{3})
}

func flexibleFunc(i interface{}) {
    fmt.Printf("you passed type %T and with value %v", i, i)
}
```

Interfaces

```go
package main
import "fmt"


type rectangle struct{
    length float32
    width float32
}
type circle struct{
    radius float32
}
func (r rectangle) area() float32{
    return r.length * r.width
}

func main() {
    flexibleFunc(rectangle{1,2})
    fmt.Println()
    flexibleFunc(3)
}

func flexibleFunc(i interface{}) {
    switch i.(type) {
    case rectangle:
        fmt.Println(i.(rectangle).area())
    case int:
        fmt.Println(i.(int))
    default:
        fmt.Println("input type not recognized")
    }
}
```

Assignment

1. Read loop structure in Go: https://www.geeksforgeeks.org/loops-in-go-language/
2. Read if-else in Go: https://www.geeksforgeeks.org/go-decision-making-if-if-else-nested-if-if-else-if/
3. Write a program to output each character of the string “ Hellő There ”
4. Write a program to implement a function uniqueFunc() which takes variable number of arguments and types of arguments can also vary. It should print square for int argument, perimeter for rectangle struct argument and some error message for other types.

   uniqueFunc(3, 4, rectangle{2,5}) => 9, 16, 14

   uniqueFunc(rectangle{2.5,4}, 9, 10 ,11, 12, rectangle{1,2}) => 13, 81, 100, 121, 144, 6

Thank You