Go Training

Session 8

if condition

- Not necessary to wrap condition inside parenthesis
- Curly braces {} are mandatory

```go
package main
import "fmt"

func main() {
    x := 5

    if x==5 {
        fmt.Println("input is 5")
    }

}
```

```go
package main
import "fmt"

func main() {

    if x:=5; x<10 {
        fmt.Println("x is less than 10")
    }

}

```

if-else condition

```go
package main
import "fmt"

func main() {

    x:= 5

    if x>10 {
        fmt.Println("x is more than 10")
    } else {
        fmt.Println("x is less than 10")
    }

}
```

```go
package main
import "fmt"

func main() {

    x:= 5

    if x>10 {
        fmt.Println("x is more than 10")
    } else if x==10{
        fmt.Println("x is 10")
    } else {
        fmt.Println("x is less than 10")
    }

}
```

switch condition

- Switch is generally used to replace multiple if-else conditions

```go
package main
import "fmt"

func main() {

    x:= 5

    switch x {
    case 2:
        x+=1
        fmt.Println(x)
    case 3:
        fmt.Println(x)
    case 4:
        x-=1
        fmt.Println(x)
    case 5:
        x*=2
        fmt.Println(x)
    }

}
```

```go
package main
import "fmt"

func main() {

    x:= 9
    switch x {
    case 2:
        x+=1
        fmt.Println(x)
    case 3:
        fmt.Println(x)
    case 4:
        x-=1
        fmt.Println(x)
    case 5:
        x*=2
        fmt.Println(x)
    default:
        fmt.Println("x not in scope")
    }
}
```

switch condition

- Multiple case values

```go
package main
import "fmt"

func main() {

    x:= 9
    switch x {
    case 3,5,7:
        fmt.Println("x is an odd prime")
    case 2:
        fmt.Println("x is an even prime")
    case 4,6,8,9:
        fmt.Println("x is composite")
    default:
        fmt.Println("x not in scope")
    }
}
```

switch condition

- Initial statement

```go
package main
import "fmt"

func main() {

    switch x:=9; x {
    case 3,5,7:
        fmt.Println("x is an odd prime")
    case 2:
        fmt.Println("x is an even prime")
    case 4,6,8,9:
        fmt.Println("x is composite")
    default:
        fmt.Println("x not in scope")
    }
}

```

switch condition

- Expressionless switch

```go
package main
import "fmt"

func main() {
    x:=9
    switch {
    case x==3,x==5,x==7:
        fmt.Println("x is an odd prime")
    case x==2:
        fmt.Println("x is an even prime")
    case x==4,x==6,x==8,x==9:
        fmt.Println("x is composite")
    default:
        fmt.Println("x not in scope")
    }
}

```

switch condition

- Fallthrough
  - Next case block is executed even if that condition is not matched

```go
package main
import "fmt"

func main() {
    x:=11
    switch {
    case x>5:
        fmt.Println("x is greater than 5")
    case x<10:
        fmt.Println("x is less than 10")
    case x%2==0:
        fmt.Println("x is even")
    default:
        fmt.Println("x not in scope")
    }
}
```

```go
package main
import "fmt"

func main() {
    x:=11
    switch {
    case x>5:
        fmt.Println("x is greater than 5")
        fallthrough
    case x<10:
        fmt.Println("x is less than 10")
    case x%2==0:
        fmt.Println("x is even")
    default:
        fmt.Println("x not in scope")
    }
}
```

for loop

```go
package main
import "fmt"

func main() {
    var x int
    for x=11; x>5; x-=2 {
        fmt.Println(x)
    }
}
```

```go
package main
import "fmt"

func main() {

    for x:=11; x>5; x-=2 {
        fmt.Println(x)
    }
}
```

```go
package main
import "fmt"

func main() {
    x:=11
    for x=8; x>5; x-=2 {
        fmt.Println(x)
    }
}
```

```go
package main
import "fmt"

func main() {
    x:=11
    for ; x>5; x-=2 {
        fmt.Println(x)
    }
}
```

for loop

```go
package main
import "fmt"

func main() {
    for i:=0; i<4; i++ {
        fmt.Println(i)
    }
}
```

```go
package main
import "fmt"

func main() {
    for i:=0; i<4; {
        fmt.Println(i)
        i++
    }
}
```

```go
package main
import "fmt"

func main() {
    x:=11
    for ; x>5; {
        fmt.Println(x)
        x--
    }
}
```

```go
package main
import "fmt"

func main() {
    x:=11

    for x>5 {
        fmt.Println(x)
        x--
    }
}
```

for loop

- Infinite loop
- break statement
- continue statement
- return statement

```go
package main
import "fmt"

func main() {
    x:=11

    for {
        fmt.Println(x)
        if x==5 {
            break
        }
        x--
    }
}
```

```go
package main
import "fmt"

func main() {
    x:=11

    for {
        if x<6 {
            break
        }
        x--
        if x%2==1 {
            continue
        }
        fmt.Println(x)

    }
}
```

```go
package main
import "fmt"

func main() {
    x:=11

    for {
        if x<6 {
            return
        }
        fmt.Println(x)
        x--

    }
}
```

defer statement

- Defers the execution of a function until the surrounding function returns
  - Last In First Out (LIFO)

```go
package main
import "fmt"

func main() {
    fmt.Println("first print statement")
    fmt.Println("second print statement")
}
```

```go
package main
import "fmt"

func main() {
    defer fmt.Println("first print statement")
    fmt.Println("second print statement")
}
```

```go
package main
import "fmt"

func main() {
    defer fmt.Println("first print statement")
    defer fmt.Println("second print statement")
    fmt.Println("third print statement")
}
```

```go
package main
import "fmt"

func main() {
    defer fmt.Println("first print statement")
    defer fmt.Println("second print statement")
    defer fmt.Println("third print statement")
}
```

defer statement

```go
package main
import "fmt"

func main() {
    fmt.Println("in main before first call")
    defer firstFunction()
    fmt.Println("in main after first call")

}

func firstFunction(){
    fmt.Println("first called")
    defer secondFunction()
    thirdFunction()
}

func secondFunction(){
    fmt.Println("second called")
}

func thirdFunction(){
    fmt.Println("third called")
}
```

defer statement

```go
package main
import "fmt"

func main() {
    x:=5
    fmt.Println("in main x value before defer first call", x)
    defer firstFunction(x)
    x *= 2
    fmt.Println("in main x value after defer first call", x)

}

func firstFunction(val int){
    fmt.Println("first called with x value", val)
}

```

Thank You