Go Training

Session 8

if condition

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

if-else condition

```
package main
     import "fmt"
     func main() {
         x := 5
         if x>10 {
             fmt.Println("x is more than 10")
10
         } else {
             fmt.Println("x is less than 10")
11
12
13
```

```
package main
     import "fmt"
 3
 4
     func main() {
 5
         x := 5
 8
         if x>10 {
 9
              fmt.Println("x is more than 10")
         } else if x==10{
10
11
              fmt.Println("x is 10")
12
         } else {
13
14
              fmt.Println("x is less than 10")
15
```

Switch is generally used to replace multiple if-else conditions

```
package main
     import "fmt"
     func main() {
         x := 5
         switch x {
 8
         case 2:
             x+=1
10
             fmt.Println(x)
         case 3:
13
             fmt.Println(x)
         case 4:
15
             x-=1
16
             fmt.Println(x)
17
         case 5:
18
             x*=2
19
             fmt.Println(x)
20
21
```

```
package main
     import "fmt"
     func main() {
         x := 9
         switch x {
         case 2:
             x+=1
10
             fmt.Println(x)
11
         case 3:
12
             fmt.Println(x)
13
         case 4:
14
             x-=1
15
             fmt.Println(x)
16
         case 5:
17
             x*=2
18
             fmt.Println(x)
19
         default:
20
             fmt.Println("x not in scope")
21
```

Multiple case values

```
package main
     import "fmt"
 3
 4
     func main() {
 5
 6
         x := 9
         switch x {
 8
         case 3,5,7:
9
             fmt.Println("x is an odd prime")
10
         case 2:
11
             fmt.Println("x is an even prime")
12
         case 4,6,8,9:
13
             fmt.Println("x is composite")
14
         default:
15
             fmt.Println("x not in scope")
16
```

Initial statement

```
package main
     import "fmt"
     func main() {
         switch x:=9; x {
         case 3,5,7:
             fmt.Println("x is an odd prime")
         case 2:
10
             fmt.Println("x is an even prime")
11
         case 4,6,8,9:
12
             fmt.Println("x is composite")
13
         default:
14
             fmt.Println("x not in scope")
15
16
```

• Expressionless switch

```
package main
     import "fmt"
     func main() {
         x := 9
         switch {
         case x==3, x==5, x==7:
             fmt.Println("x is an odd prime")
 8
         case x==2:
10
             fmt.Println("x is an even prime")
         case x==4, x==6, x==8, x==9:
11
12
             fmt.Println("x is composite")
13
         default:
14
             fmt.Println("x not in scope")
15
16
```

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

```
package main
     import "fmt"
     func main() {
         x := 11
         switch {
         case x>5:
             fmt.Println("x is greater than 5")
         case x<10:
10
             fmt.Println("x is less than 10")
11
         case x\%2==0:
12
             fmt.Println("x is even")
13
         default:
14
             fmt.Println("x not in scope")
15
```

```
package main
     import "fmt"
     func main() {
         x:=11
 6
         switch {
 7 ▼
         case x>5:
             fmt.Println("x is greater than 5")
 9
             fallthrough
10
         case x<10:
11
             fmt.Println("x is less than 10")
12
         case x%2==0:
13
             fmt.Println("x is even")
14
         default:
15
             fmt.Println("x not in scope")
16
```

for loop

```
1  package main
2  import "fmt"
3
4  func main() {
    var x int
    for x=11; x>5; x-=2 {
       fmt.Println(x)
    }
9  }
10
```

```
1  package main
2  import "fmt"
3
4   func main() {
5          x:=11
6          for x=8; x>5; x-=2 {
7               fmt.Println(x)
8          }
9     }
10
```

```
1  package main
2  import "fmt"
3
4  func main() {
5
6    for x:=11; x>5; x-=2 {
7    fmt.Println(x)
8    }
9  }
```

```
1  package main
2  import "fmt"
3
4  func main() {
5     x:=11
6     for ; x>5; x-=2 {
7        fmt.Println(x)
8     }
9  }
10
```

for loop

```
1  package main
2  import "fmt"
3
4  func main() {
5    for i:=0; i<4; i++ {
6     fmt.Println(i)
7    }
8  }
9</pre>
```

```
1  package main
2  import "fmt"
3
4  func main() {
5     x:=11
6     for ; x>5; {
7        fmt.Println(x)
8          x--
9     }
10  }
11
```

```
1  package main
2  import "fmt"
3
4  func main() {
5    for i:=0; i<4; {
6    fmt.Println(i)
7    i++
8    }
9  }
10</pre>
```

for loop

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

```
package main
     import "fmt"
 3
     func main() {
 5
         x := 11
 6
         for {
              fmt.Println(x)
 8
              if x==5 {
10
                  break
11
12
              X---
13
```

```
package main
     import "fmt"
     func main() {
          x := 11
          for {
              if x<6 {
                  break
10
              X--
12
              if x%2==1 {
13
                  continue
14
15
              fmt.Println(x)
16
17
18
```

```
package main
      import "fmt"
     func main() {
          x := 11
 6
          for {
8
               if x<6 {
                    return
10
11
               fmt.Println(x)
12
               \mathbf{X} - -
13
14
```

defer statement

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

```
package main
import "fmt"

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

```
package main
import "fmt"

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

```
package main
import "fmt"

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

```
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

```
package main
     import "fmt"
     func main() {
         fmt.Println("in main before first call")
 6
         defer firstFunction()
         fmt.Println("in main after first call")
10
11
     func firstFunction(){
         fmt.Println("first called")
12
13
         defer secondFunction()
14
         thirdFunction()
15
16
17
     func secondFunction(){
18
         fmt.Println("second called")
19
20
21
     func thirdFunction(){
22
         fmt.Println("third called")
```

defer statement

```
package main
     import "fmt"
 3
 4
     func main() {
 5
         x := 5
         fmt.Println("in main x value before defer first call", x)
         defer firstFunction(x)
 8
         x *= 2
 9
         fmt.Println("in main x value after defer first call", x)
10
11
12
13
     func firstFunction(val int){
14
         fmt.Println("first called with x value", val)
15
16
```

Thank You