Introduction to Programming in Go

7. Advanced Functions in Go

Module Topics

1. Functions as first class objects
2. Function Literals
3. Anonymous Functions
4. Closures

Functions as First Class Objects

Basic Function Syntax

1. Functions are first class objects in Go.

2. Functions can be assigned to variables.

3. Functions can be passed to and returned from other functions.

4. A function type is number and types of parameters and return values:

   Eg. func f(x, y, string)(a , b int){...} has type

   func(string,string)(int, int)

Function Variables

```
// Example 07-01 Function variables
...
func f1() string {
   return "I'm f1"
}
func f2() string {
   return "and I'm f2"
}

func main() {
   f := f1
   fmt.Printf("f is of type '%T' \n", f)
   fmt.Println(f())
   f = f2
   fmt.Println(f())
}
```

```
[example 07] go run ex07-01.go
f is of type 'func() string'
I'm f1
and I'm f2
```

Function as Parameter

```
// Example 07-02  Functions as parameters
...

func f1() string {
   return "I'm f1"
}
func f2(fparam func() string) {
   fmt.Println(fparam())
}

func main() {
   f := f1
   f2(f)
}
```

```
[Module07]$ go run ex07-02.go
I'm f1
```

Function as Return Value

```
// Example 07-03  Function as return value
...

func f1() string {
   return "I'm f1"
}
func f2() func() string {
   return f1
}

func main() {
   f := f2()
   fmt.Println(f())
}
```

```
[Module07]$ go run ex07-03.go
I'm f1
```

Function Literals

Function Literal

1. We can assign a function body to a variable.
2. In this case the function body is called a "function literal."
3. Acts like any other type of literal (first class object).
4, Can be passed as parameters, etc.
5. Function literals are identified by the func keyword.

Function Literal

```
// Example 07-04  Function literal

...

func main() {
   f := func(i int) bool { return i == 0 }
   fmt.Println("2 == 0 is", f(2))
   fmt.Println("0 == 0 is", f(0))
}
```

```
[Module07]$ go run ex07-04.go
2 == 0 is false
0 == 0 is true
```

Function Literal as Parameter

```
// Example 07-05  Function parameter

...

func f2(p func(int) bool) {
   fmt.Println("2 == 0 is", p(2))
   fmt.Println("0 == 0 is", p(0))
}

func main() {
   f2(func(i int) bool { return i == 0 })
}
```

```
[Module07]$ go run ex07-05.go
2 == 0 is false
0 == 0 is true
```

Anonymous Functions

Anonymous Functions

1. Anonymous functions are function literals that are executed without being assigned to a variable.

   func() { ...} is a function literal

   func() {...}() executes the function literal (anon function)

2. The following assigns the function literal to the variable f

   f: = func() {...}

4. The following assigns the result of executing the function to g

   g : = func() {...} ()

5 The second of these is an anonymous function since there no way to reference the function itself.

Inner Function

```
// Example 07-06  Executing a literal directly

...

func main() {
   z := func(x int) (y int) {
      y = x + 1
      return
   }(0)
   fmt.Println(z)
}
```

```
[Module07]$ go run ex07-06.go
1
```

Anonymous Inner Function

```
// Example 07-07  Anonymous inner function

...

func main() {
   defer func(name string) {
       fmt.Println("Hello ", name)
       return
   }("World")
fmt.Println("Main Function")
}
```

```
[Module07]$ go run ex07-07.go
Main Function
Hello  World
```

Closures

Closure

1. Functions can have “free variables” – variables used in the scope of the function that are not defined in the function.

2. When we create an instance of the function, we also create copies of any of the free variables referenced in the function body.

3. The combination of function instance plus its copies of the free variables is called a closure.

```
a:= 1
f: = func () int{
   return a + 1
}
```

4. In the example above, "a" is a free variable and we will need to remember its value to execute f()

Simple Closure

```
// Example 07-08  Simple Closure

...

func f(p func(string)) {
   p("Mars") // "a" is out of scope here
}

func main() {
   a := "again "
   z := func(name string) {
      fmt.Println("Hello ", a, name)
      return
   }
   f(z)
}
```

```
[Module07]$ go run ex07-08.go
Hello  again  Mars
```

Another Closure Example

```
// Example 07-09 Closure
// From the go tour on the golang website
...

func intSeq() func() int {
   i := 0
   return func() int {
      i += 1
      return i
   }
}

func main() {
   nextInt := intSeq()
   fmt.Println(nextInt())
   fmt.Println(nextInt())
   newInts := intSeq()
   fmt.Println(newInts())
}
```

```
[Module07]$ go run ex07-09.go
1
2
1
```

Lab 7: Advanced Functions