

Function Literals



01

Anonymous Function

02

Closures

03

Aliases

Function Literals

- | **Function literals** provide a way to define a function within a function
- | Possible to assign function literals to variables
- | They can be passed to a function as parameters
 - | More dynamic code
- | Also known as **closures** or **anonymous functions**
 - | Closures allow data to be encapsulated within

Anonymous Function

```
func helloWorld() {  
    fmt.Printf("Hello, ")  
    world := func() {  
        fmt.Printf("World!\n")  
    }  
    world()  
    world()  
    world()  
    world()  
}
```

```
> go run ./lecture.go  
Hello, World!  
World!  
World!  
World!
```

As Function Parameter

```
func customMsg(fn func(m string), msg string) {  
    msg = strings.ToUpper(msg)  
    fn(msg)  
}  
  
func surround() func(msg string) {  
    return func(msg string) {  
        fmt.Printf("%.5s\n", len(msg), "-----")  
        fmt.Println(msg)  
        fmt.Printf("%.5s\n", len(msg), "-----")  
    }  
}  
  
customMsg(surround(), "hello")
```

```
> go run ./lecture.go  
-----  
HELLO  
-----
```


Closure

```
discount := 0.1
discountFn := func(subTotal float64) float64 {
    // Buy more save more!
    if subTotal > 100.0 {
        discount += 0.1
    }
    // Max discount
    if discount > 0.3 {
        discount = 0.3
    }
    return discount
}
```

Closure

```
func calculatePrice(  
    subTotal float64,  
    discountFn func(subTotal float64) float64) float64 {  
    return subTotal - (subTotal * discountFn(subTotal))  
}  
  
discount := 0.1  
discountFn := func(subTotal float64) float64 {  
    // Buy more save more!  
    if subTotal > 100.0 {  
        discount += 0.1  
    }  
    // Max discount  
    if discount > 0.3 {  
        discount = 0.3  
    }  
    return discount  
}  
  
totalPrice := calculatePrice(20.0, discountFn)
```

Type Alias

```
func calculatePrice(  
    subTotal float64,  
    discountFn func(subTotal float64) float64) float64 {  
    return subTotal - (subTotal * discountFn(subTotal))  
}
```



```
type DiscountFunc func(subTotal float64) float64  
  
func calculatePrice(  
    subTotal float64,  
    discountFn DiscountFunc) float64 {  
    return subTotal - (subTotal * discountFn(subTotal))  
}
```

Recap

- | **Function literals** can be passed to other functions as arguments
- | They can capture surrounding variables
- | Type aliases are helpful when passing function literals to other functions
- | Function literals can be returned from a function directly, or assigned to a variable
- | **Closure and anonymous functions** are other terms for **function literal**