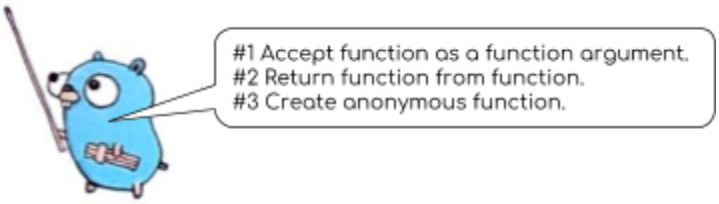
GO on the toilet

episode #6

Functions are first-class citizens

This property has some interesting consequences (we will show 4 of them). Thanks to it you can accept function as an argument to the function and return a function from a function. Because of this we can implement the *decorator pattern*. Here is one that logs the fact that function started and finished.

```
func logged(f func(int) int) func(int) int {
  return func(i int) int {
    fmt.Println("start")
    defer func() { fmt.Println("finish") }()
    return f(i)
  }
}
```



Another characteristic of being first-class citizen is also visible in the code above. The function that we return is anonymous function - it doesn't have a name and we don't care about it. Now, how would you use such decorator?

```
func square (int) int {
  fmt.Println("squaring: ", i)
  return i * i
}
var loggedSquare = logged(square)

func main () {
  loggedSquare(7)
}
// start
// squaring:  7
// finish
```

Decorator works just as expected - we are able to do something before and after function call. Example above seems irrelevant but it shows the mechanics of the pattern itself. Instead we could time how long function was executing or implement a backoff mechanism. Also we observed that function can be assigned to a variable.