Anonymous Closure

In this lesson there is a detailed description of how to use closures along with a channel.

WE'LL COVER THE FOLLOWING

Passing anonymous closure through a channel

Functions are values in Go, and so are closures. So, we can construct a channel `c`, whose data type is a function, as in the following example:

```
package main
import "fmt"

func prod(c chan func()) {
  f := <- c
  f()
  c <- nil
}

func main() {
  c := make(chan func())
  go prod(c)
  x := 0
  c <- func() { // puts a closure in channel c
    x++
  }
  fmt.Println(x)
}
```

Channel Closure

The output of this program is **1**. At **line 11**, you can see how to make channel `c` with lambdas: `c := make(chan func())`. Then, at **line 12**, we start a goroutine by passing `c` to it. Executing the `prod()` function, this is implemented from **line 4** to **line 8**.

Back to `main()` at **line 13**, an integer `x` is defined. In the commented **line 14**, an *anonymous* closure, capturing the value of `x`, is sent to the channel `c`. The closure increments `x`.

The goroutine that executes the function `prod` reads that closure `f` from the channel at **line 5** and calls it at **line 6**. This increments the value of `x` from **0** to **1**. Then at **line 7**, we put `nil` on the channel. The net result is that `x` gets the value **1**.

Now that you're familiar with how anonymous closures work with channels, in the next lesson, you'll learn another combined concept that involves closures and lazy generators.