Anonymous Closure

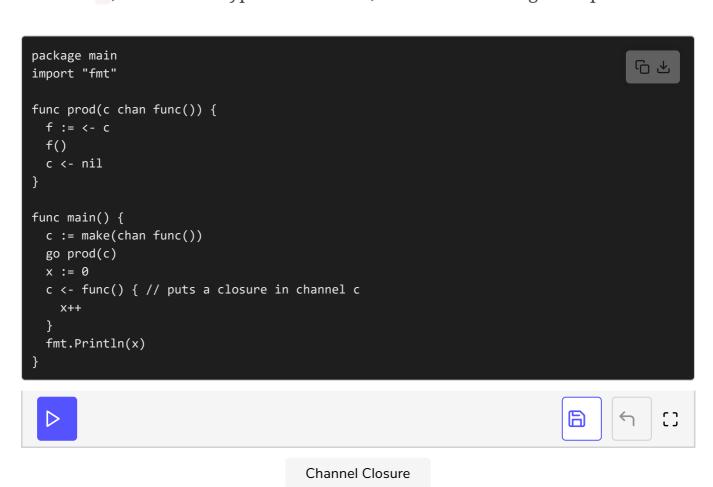
In this lesson there is a detailed description of how to use closures along with a channel.

WE'LL COVER THE FOLLOWING ^

Passing anonymous closure through a channel

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:



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 \times is defined. In the commented line 14, an *anonymous* closure, capturing the value of \times , is sent to the channel \circ . The closure increments \times .

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.