GO on the toilet

episode #15

Looping over channel

Operator **range** supports not only iterating over arrays, slices and maps but also over channels. Let's find out how it works under the hood.

```
func main() {
    source := make(chan int)
    go func() {
       for i := 0; i < 5; i++ {
             source <- i          // Produce values from 0 to 4.
       }
       close(source)              // Necessary for range to work.
    }()

    for v := range source {       // Range over all values from source.
       fmt.Println(v)
    }
}
```

Ranging over channel is merely syntactic sugar. In order to understand how this works underneath, one need to realize that channel receive operator **(<-chan)** when used in assignment yields two values:

```
v, ok := channel
```

First is value received from the channel itself. Second informs whether communication succeeded. This means that for not closed channel (values still to be received) **ok** will be **true**, and for closed channel it will be **false**. This is why closing a channel at the producer end is so important. Without it consumer doesn't know when to stop to receive from the channel.

```
for {                             // Known as "while true" in other languages.
    v, ok := <-source
    if !ok {                      // If there is no more values just stop.
        break
    }
    fmt.Println(v, ok)
}
```

As you can see looping is just receiving value after value and checking whether channel is closed or not. However **range** keyword is shorter, more verbose and preferable.