Solution Review: Filter Even and Odd Numbers

This lesson discusses the solution to the challenge given in the previous lesson.

```
package main
import "fmt"

type flt func(int) bool // aliasing type

func isEven(n int) bool { // check if n is even or not
    if n % 2 == 0 {
        return true
    }
    return false
}

func filter(sl[] int, f flt)(yes, no[] int) { // split s into two slices: even and odd
    for _, val := range sl {
        if f(val) {
            yes = append(yes, val)
        } else {
            no = append(no, val)
        }
    }
    return
}

func main() {
    slice := [] int {1, 2, 3, 4, 5, 7}
    fmt.Println("slice = ", slice)
    even,odd := filter(slice, isEven)
    fmt.Println("The even elements of slice are: ", even)
    fmt.Println("The odd elements of slice are: ", odd)
}
```

Solution

The program above has one basic function. The function isEven takes `n` as a parameter and returns a boolean value (see its header at **line6**). If `n` is even, it will return **true**; otherwise, it will return **false**. At **line 4**, we are aliasing a type. A function that takes a single integer as a parameter and returns a single boolean value is given a type `flt`.

Now, we move towards a major part of the program: `filter` function. See its header at **line 13**. It takes a slice of integers (that are to be judged as even or odd) as a first parameter and function `f` of type flt (`isEven`). The function `filter` returns two slices of integers `yes` (integers that are even) and `no` (integers that are odd).

Let's see the `main` function now. At **line 25**, we declare a slice of integers named `slice`. Then, at **line27**, we call the `filter` function with `slice` as the first parameter and `isEven` as the second parameter and store the result in the `even` and `odd` slices. Printing `even` and `odd` slices at **line 28** and **line 29**, respectively, verifies the result.

That's it about the solution. In the next lesson, you'll study how to write a *closure* in a program.