Solution Review: Magnify a Slice

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

```go
package main
import "fmt"

var s []int

func main() {
        s = []int{1, 2, 3}
        fmt.Println("The length of s before enlarging is:", len(s))
        fmt.Println(s)
        s = enlarge(s, 5)       // calling function to magnify
        fmt.Println("The length of s after enlarging is:", len(s))
        fmt.Println(s)
}

func enlarge(s []int, factor int) []int {
        ns := make([]int, len(s) * factor)              // making a new slice of length len(s
        copy(ns, s)     // copying contents from s to new slice
        return ns
}
```

Magnify a Slice

In the code above, look at the header for function `enlarge` at **line 15**: `func enlarge(s []int, factor int) []int`. It takes a slice `s` that needs to be magnified, and a `factor` to decide the length of a magnified slice as **len(s)*factor**. In the next line, we make a new slice `ns` with the `make` function of length **len(s)*factor** as required. The slice `ns` will contain the **len(s)*factor** number of *zeros*. At **line 17**, we copy all the contents from the original slice `s` to `ns`. That means the first **len(s)** number of *zeros* in `ns` will be replaced by values in `s`, index by index. At last, we are returning `ns`, the magnified slice.

Now, look at the `main` function. At **line 7**, we declare a slice `s` with some values in it. At **line 8**, we print the length of `s` before magnifying it, and in the next line, we print `s`. Now, we call the `enlarge` function for `s` at **line 10**, and next line, we print `s`. Now, we call the `enlarge` function for `s` at **line 10**, and store the result in `s`. At **line 11**, we print the length of magnified `s`, and in the next line, we print magnified `s`, to verify that `s` is magnified by **len(s)*factor**.

That's it about the solution. In the next lesson, you'll attempt another challenge.