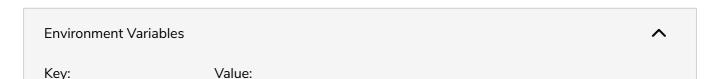
Mutability

This lesson discusses mutability and how to use it with help of pointers in GO

In Go, only *constants* are **immutable**. However, because arguments are passed by value, a function receiving a value argument and mutating it, won't mutate the original value.



As you can see the total amount of songs on the me variable's value wasn't changed. To mutate the passed value, we need to pass it by reference, using a pointer.



GOPATH /go

```
package main
import "fmt"
type Artist struct {
        Name, Genre string
        Songs
                    int
}
func newRelease(a *Artist) int { //passing an Artist by reference
        a.Songs++
        return a.Songs
}
func main() {
        me := &Artist{Name: "Matt", Genre: "Electro", Songs: 42}
        fmt.Printf("%s released their %dth song\n", me.Name, newRelease(me))
        fmt.Printf("%s has a total of %d songs", me.Name, me.Songs)
}
```

The only change between the two versions is that newRelease takes a pointer
to an Artist value and when I initialize our me variable, I used the & symbol
to get a pointer to the value.

Another place where you need to be careful is when calling methods on values as explained a bit later.

This marks the end of this chapter. In the next chapter, we will discuss basic types and conversions.