Pointers

This lesson discusses pointers and how to pass them as arguments in GO

WE'LL COVER THE FOLLOWING

Go and Pointers

Go has *pointers*, but no pointer arithmetic. *Struct* fields can be accessed through a struct pointer. The indirection through the pointer is **transparent** (you can directly call fields and methods on a pointer).

Note that by *default* Go passes arguments by value (copying the arguments), if you want to pass the arguments by reference, you need to pass pointers (or use a structure using reference values like slices and maps.

To get the pointer of a value, use the `&` symbol in front of the value; to dereference a pointer, use the `*` symbol.

Methods are often defined on pointers and not values (although they can be defined on both), so you will often store a pointer in a variable as in the example below:

Environment Variables

Key:	Value:
GOPATH	/go

```
client := &http.Client{}
resp, err := client.Get("http://gobootcamp.com")
```

Now that you're clear about *pointers* in GO. The next lesson will discuss the concept of *mutability*.