GO on the toilet

episode #5

Are there interfaces in go?

Sure, let's look at one:

```
type Writer interface {
  Write(p []byte) (n int, err error)
}
```

This is **`Writer`** type from standard library (from io package). If you want to implement such interface, you need to have struct with function **`Write`** defined on this struct.

Let's write some dummy implementation of this interface.

```
type DummyWriter struct {}

func (dw DummyWriter) Write (p []byte) (int, error) {
  return 0, nil
}
```

Here we have completely valid and extremely useless Writer implementation. I just wanted to show you that in golang, interface implementation is implicit. In order to satisfy interface, you need to have all of the method of the interface implemented on a struct. One struct can implement as much interfaces as it wants.

Why does it matter?

In golang there is a rule of thumb that *in function you should accept interfaces as parameters but return concrete implementations*. So for example if you want to have a function that writes something to file, write a function that accepts **io.Writer** interface instead of **os.File** struct (which is io.Writer implementation by the way). Then if you want to test this function, you don't need to give it real File, you can give it any mock/dummy struct that implements Write.

Most important thing - naming interfaces.

interface with function `Write` -> `Writer`
Interface with `Write` and `Close` -> `WriteCloser`
Interface with `Read`, `Write` and `Seek` -> `ReadWriteSeeker`
Interface with `Egnyte` -> `Egnyter`