GO on the toilet

episode #13

About functions implementing interfaces

It is commonly known that struct types can implement interfaces, but did you know that any type can implement an interface?

An interface used when working with HTTP servers is **Handler**. It has one method **ServeHTTP** and every struct implementing this interface can be registered as a handler for incoming HTTP requests. This is simple implementation of **Handler** interface.

```go
type helloServer struct{}

func (s helloServer) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("hello")) // usually something useful is going on here...
}

//registration
http.Handle("/hello", helloHandler)
```

Designers of the standard library also thought of a use case where function is register as a handler that will handle HTTP requests and the way they did it is quite astonishing.

```go
type HandlerFunc func(ResponseWriter, *Request)

func (f HandlerFunc) ServeHTTP(w ResponseWriter, r *Request) {
    f(w, r)
}
```

When I first saw it, I couldn't understand it. Now it makes perfect sense and I find it clever. First of all there is a **HandlerFunc** type that is a proxy to the function which accepts **ResponseWriter** and ***Request**. Every function with that signature can be converted to the type **HandlerFunc**.

In the next line magic happens. There is **ServeHTTP** method defined on the **HandlerFunc** type and the only thing it does, it kind of calls itself. Since every type with **ServeHTTP** defined on it implements **Handler** interface therefore **HandlerFunc** type implements **Handler** interface.

```go
func hiHandler(w http.ResponseWriter, r *http.Request) { w.Write([]byte("hi")) }

//registration
http.Handle("/hi", http.HandlerFunc(hiHandler))
```

http.Handle accepts **Handler** as a second parameter, but since **hiHandler** can be converted to **HandlerFunc** which implements **Handler** - we can just use the function after converting it to **HandlerFunc**.