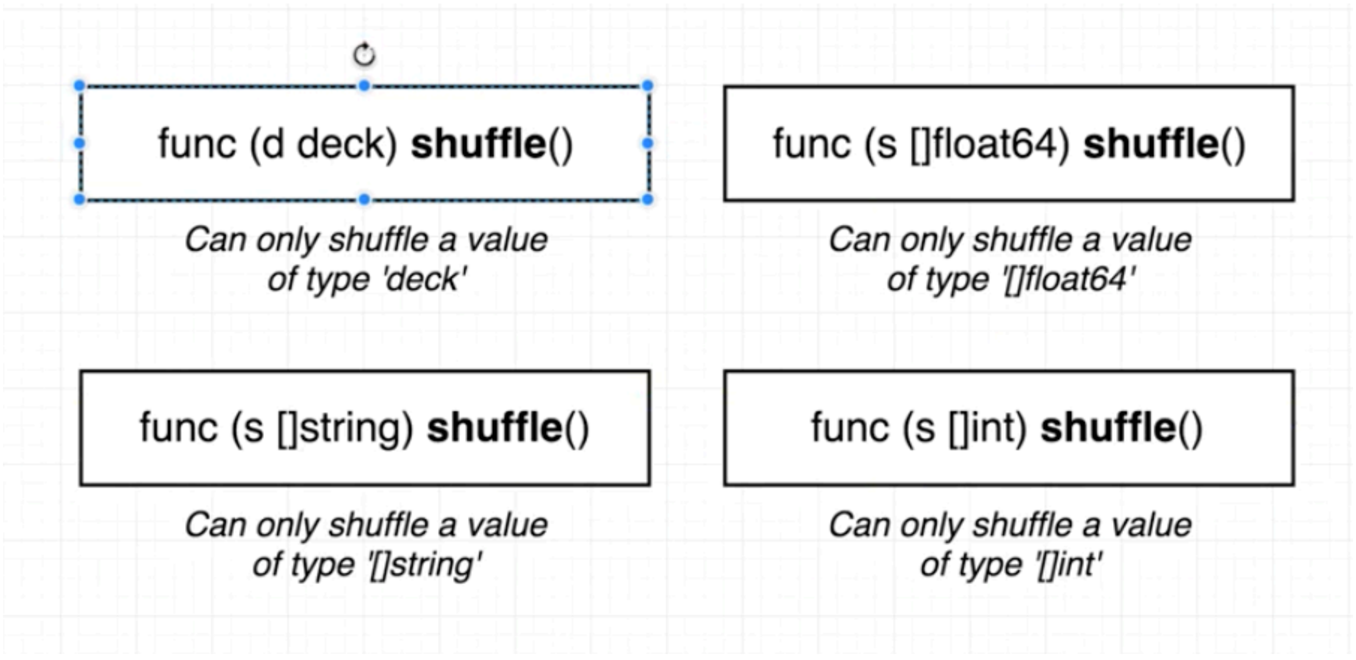
6 - Interfaces

Purpose of Interfaces

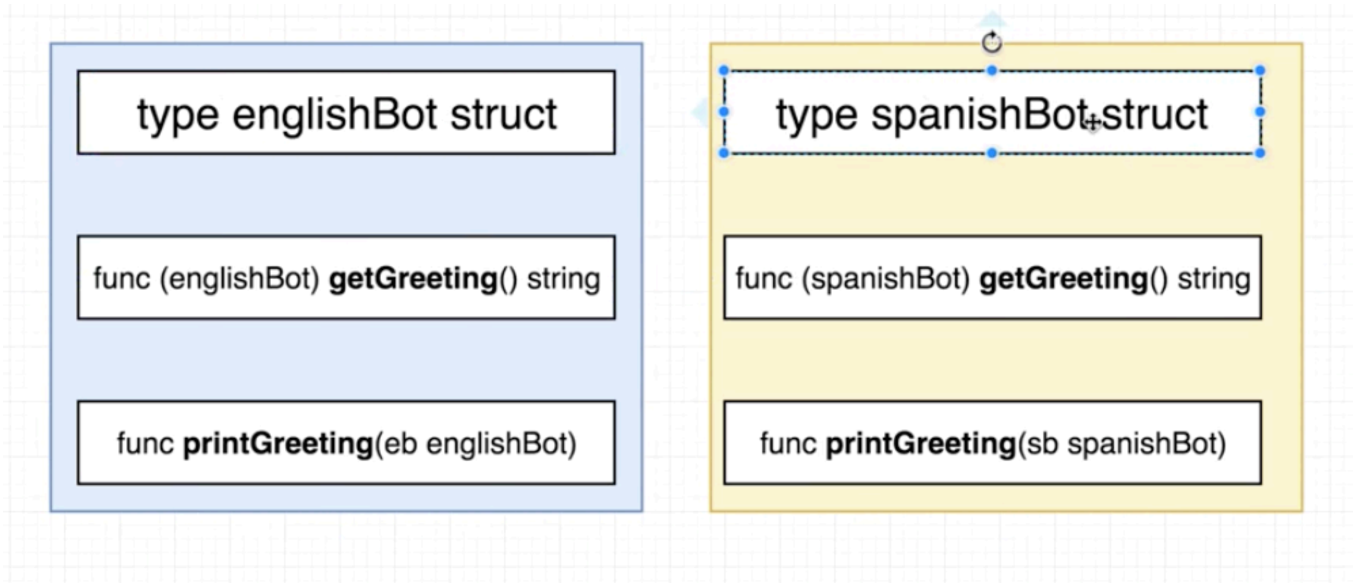


```
func (d deck) shuffle()
```
Can only shuffle a value of type 'deck'

```
func (s []float64) shuffle()
```
Can only shuffle a value of type '[]float64'

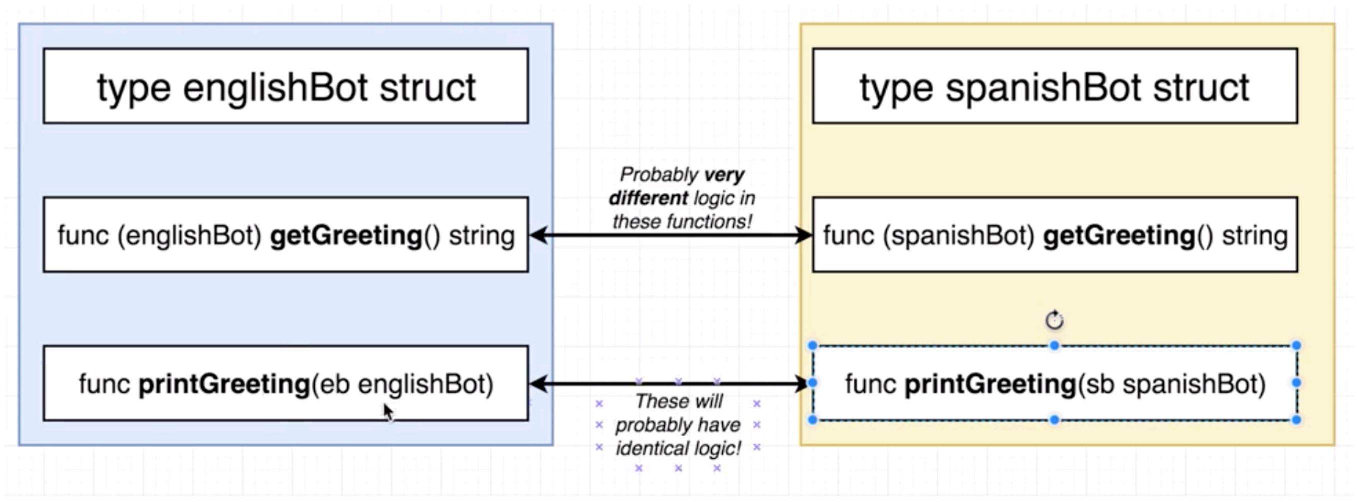
```
func (s []string) shuffle()
```
Can only shuffle a value of type '[]string'

```
func (s []int) shuffle()
```
Can only shuffle a value of type '[]int'



```
type englishBot struct

func (englishBot) getGreeting() string

func printGreeting(eb englishBot)
```

```
type spanishBot struct

func (spanishBot) getGreeting() string

func printGreeting(sb spanishBot)
```

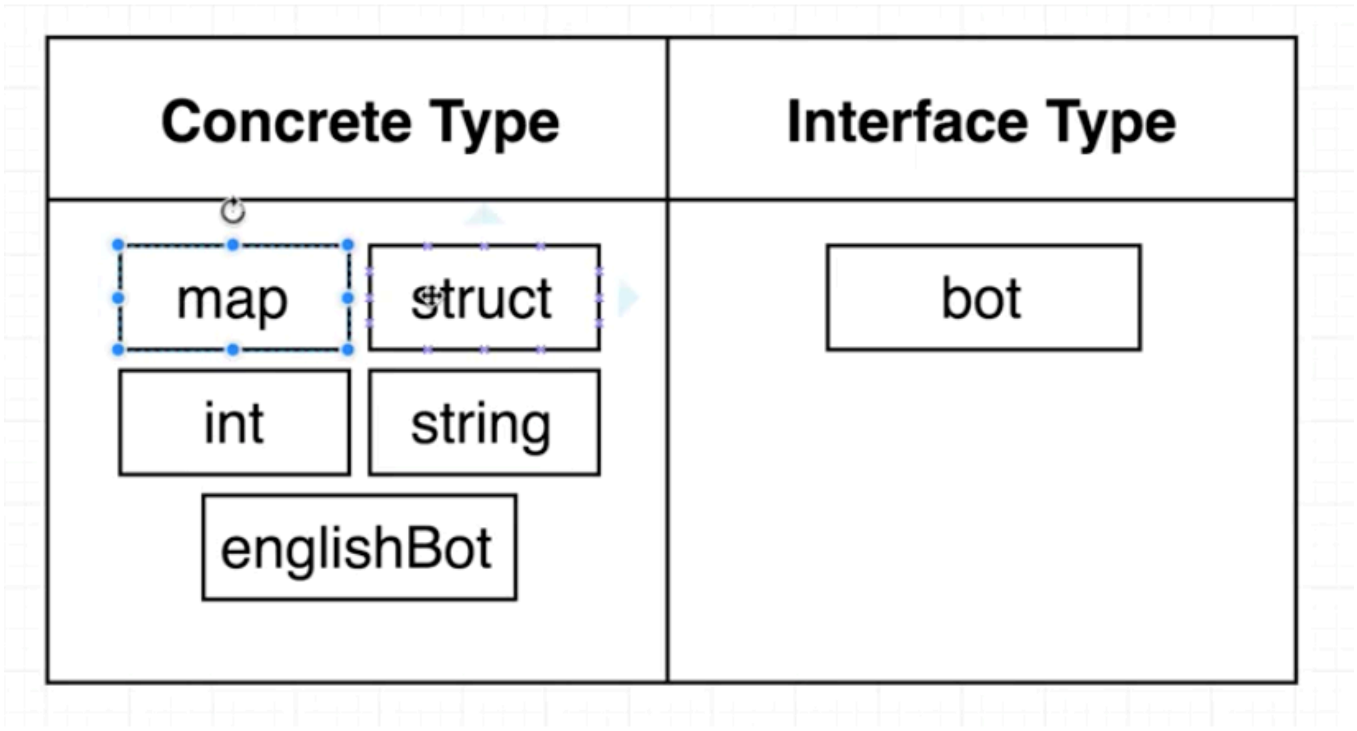

type englishBot struct
func (englishBot) getGreeting() string
func printGreeting(eb englishBot)
type spanishBot struct
func (spanishBot) getGreeting() string
func printGreeting(sb spanishBot)
Probably very different logic in these functions!
These will probably have identical logic!

Interfaces in Practice

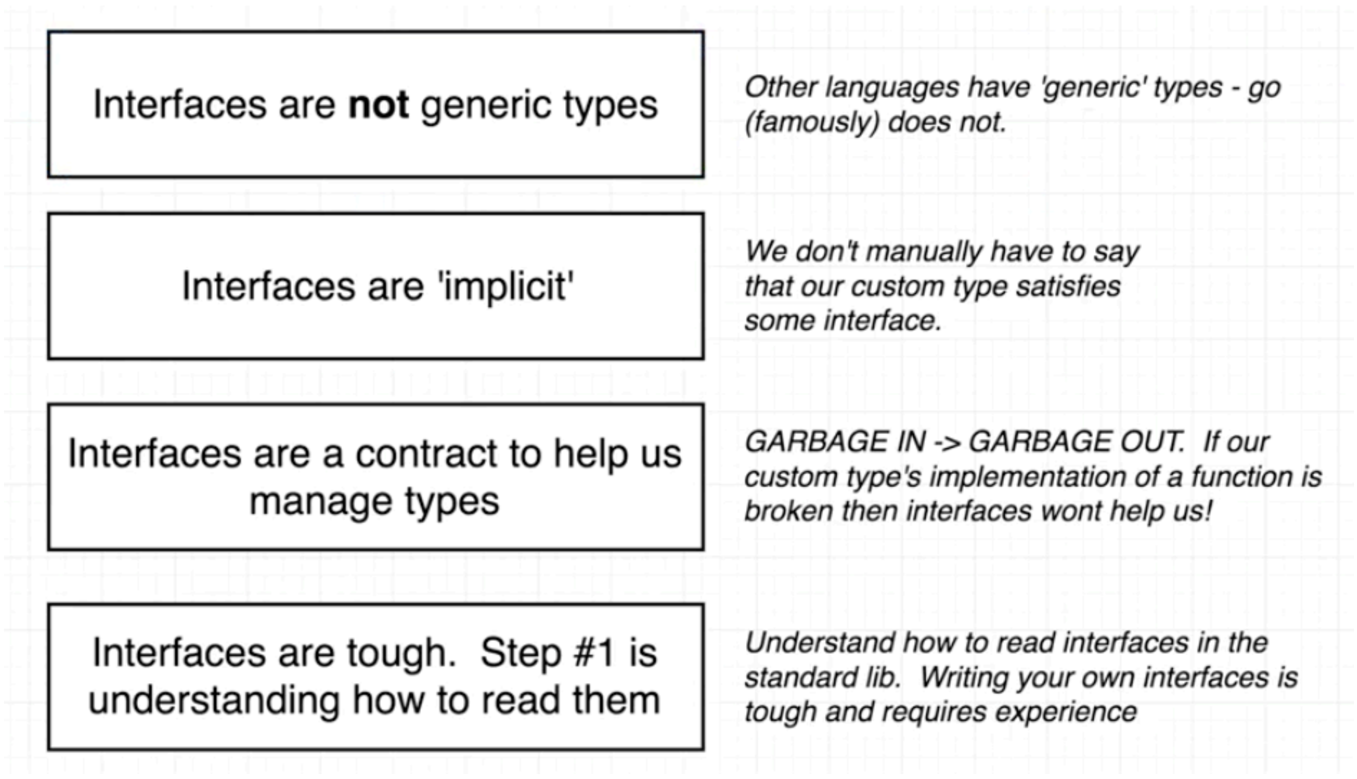
main.go

```go
type bot interface {
    getGreeting() string
}

type englishBot struct{}
type spanishBot struct{}

func main() {
    eb := englishBot{}
    sb := spanishBot{}

    printGreeting(eb)
    printGreeting(sb)

}

func printGreeting(b bot) {
    fmt.Println(b.getGreeting())
}

func (englishBot) getGreeting() string {
    // VERY custom logic for generating an english greeting
    return "Hi There!"
}

func (spanishBot) getGreeting() string {
    // VERY custom logic for generating an spanish greeting
    return "Hola!"
}
```

Rules of Interfaces



Extra Interface Notes



Interfaces are **not** generic types

Other languages have 'generic' types - go (famously) does not.

Interfaces are 'implicit'

We don't manually have to say that our custom type satisfies some interface.

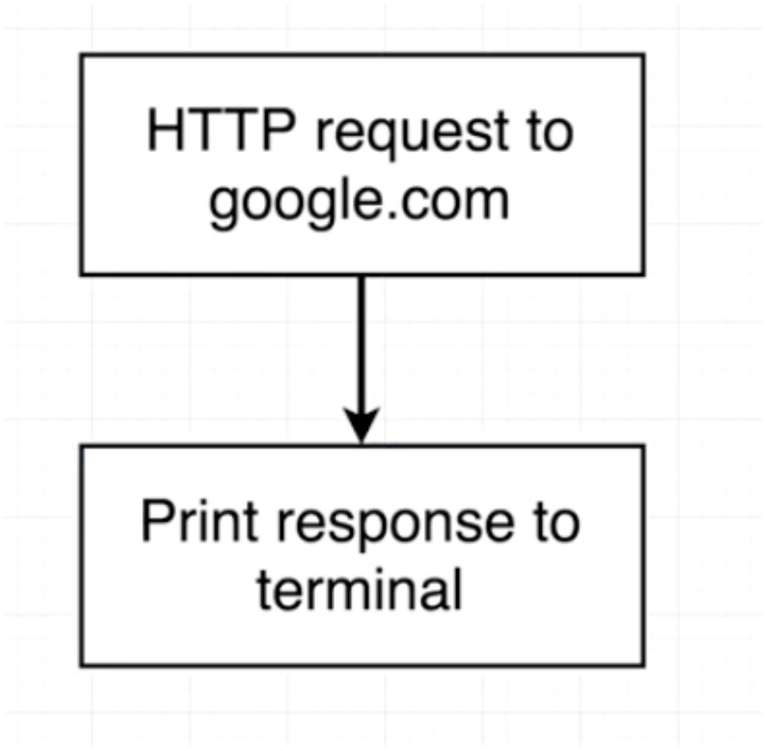
Interfaces are a contract to help us manage types

GARBAGE IN -> GARBAGE OUT. If our custom type's implementation of a function is broken then interfaces wont help us!

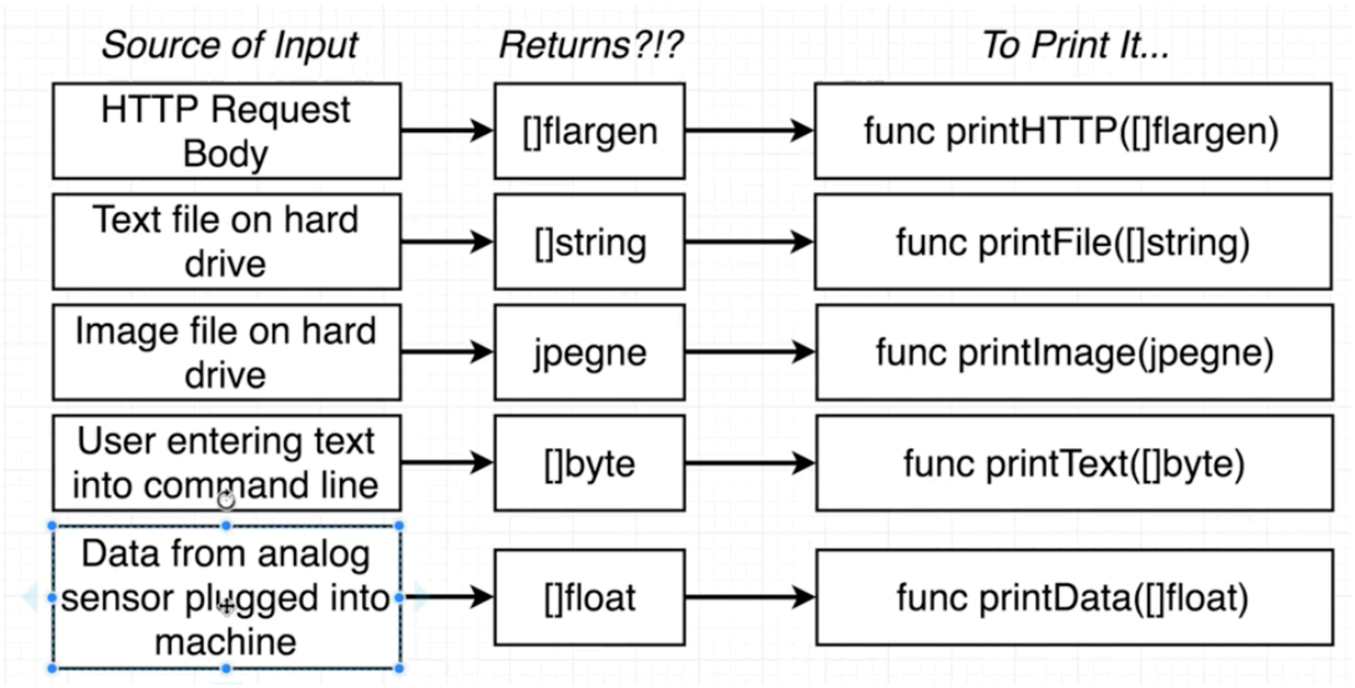
Interfaces are tough. Step #1 is understanding how to read them

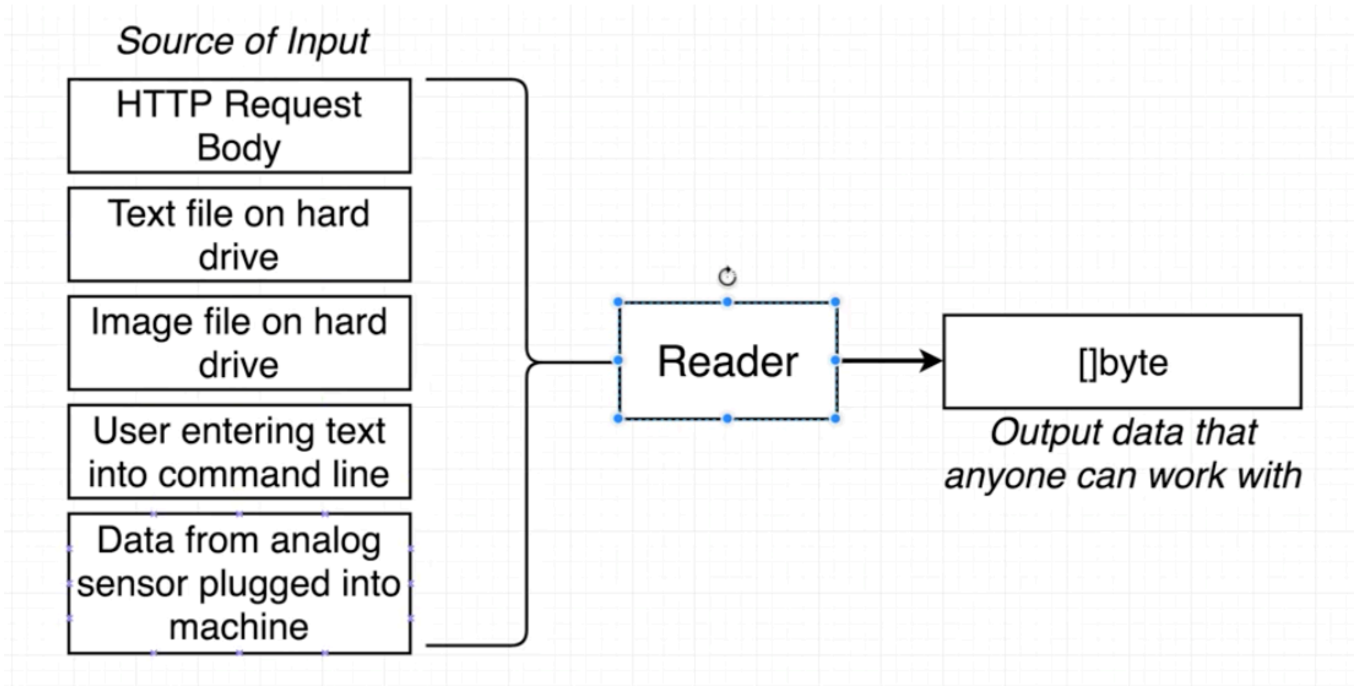
Understand how to read interfaces in the standard lib. Writing your own interfaces is tough and requires experience

The HTTP Package

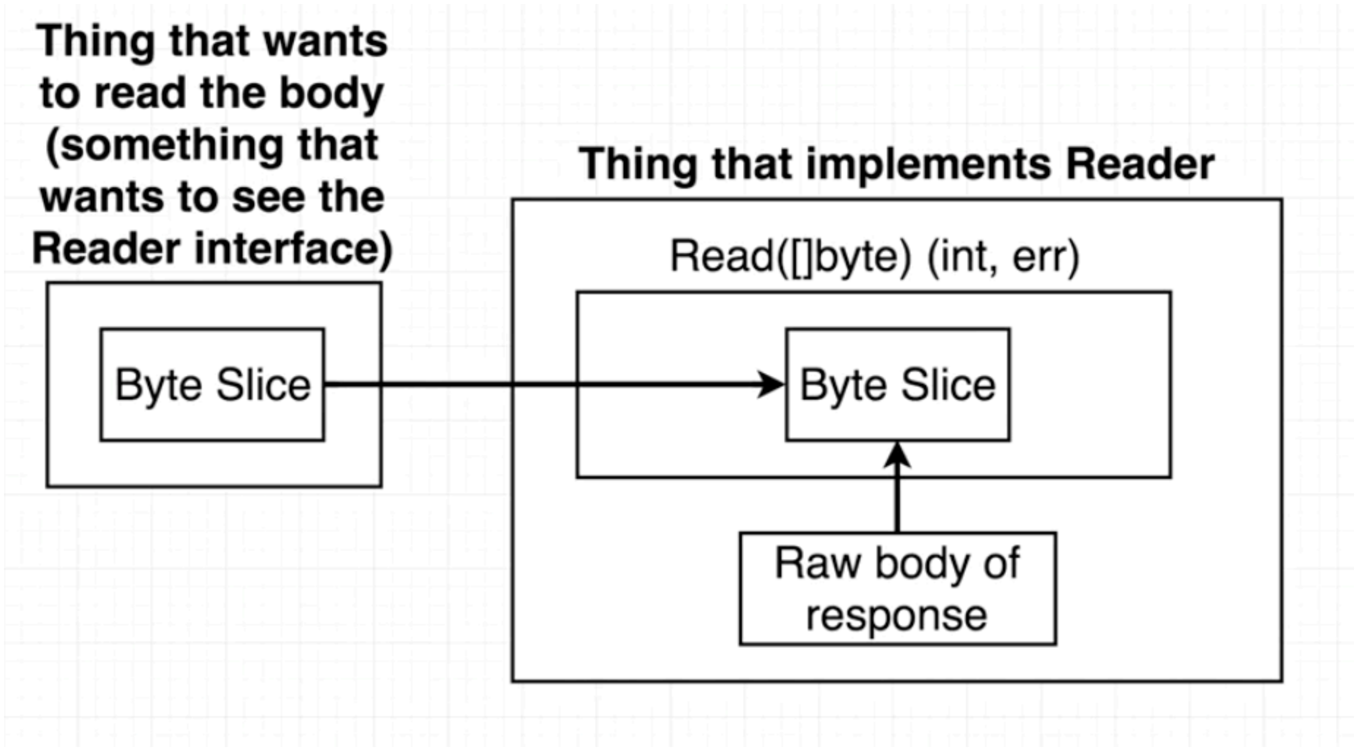


The Reader Interface




Source of Input
HTTP Request Body
Text file on hard drive
Image file on hard drive
User entering text into command line
Data from analog sensor plugged into machine
Reader
[]byte
Output data that anyone can work with

More on the Reader Interface


Thing that wants to read the body (something that wants to see the Reader interface)
Byte Slice
Thing that implements Reader
Read([]byte) (int, err)
Byte Slice
Raw body of response

Working with the Read Function

main.go

```go
package main

import (
    "fmt"
    "net/http"
    "os"
)

func main() {
    resp, err := http.Get("https://www.google.com/")
    if err != nil {
        fmt.Println("Error: ", err)
        os.Exit(1)
    }

    bs := make([]byte, 99999)
    resp.Body.Read(bs)
    fmt.Println(string(bs))
}
```

The Writer Interface

main.go

```go
package main

import (
    "fmt"
    "io"
    "net/http"
    "os"
)

func main() {
    resp, err := http.Get("https://www.google.com/")
    if err != nil {
        fmt.Println("Error: ", err)
        os.Exit(1)
    }

    io.Copy(os.Stdout, resp.Body)
}
```

A Custom Writer

main.go

```go
package main

import (
    "fmt"
    "io"
    "net/http"
    "os"
)

type logWriter struct{}

func main() {
    resp, err := http.Get("https://www.google.com/")
```

```go
	if err != nil {
		fmt.Println("Error: ", err)
		os.Exit(1)
	}

	lw := logWriter{}

	io.Copy(lw, resp.Body)
}

func (logWriter) Write(bs []byte) (int, error) {
	fmt.Println(string(bs))
	fmt.Println("Just wrote this many bytes:", len(bs))
	return len(bs), nil
}
```