type englishBot struct

func (englishBot) **getGreeting**() string

func **printGreeting**(eb englishBot)

type spanishBot struct

func (spanishBot) **getGreeting**() string

func **printGreeting**(sb spanishBot)

```go
 9        eb := englishBot{}
10        sb := spanishBot{}
11
12        printGreeting(eb)
13        // printGreeting(sb)
14  }
15
16  func printGreeting(eb englishBot) {
17        fmt.Println(eb.getGreeting())
18  }
19
20  // func printGreeting(sb spanishBot) {
21  //    fmt.Println(sb.getGreeting())
22  // }
23
```

And so to me, I really get the feeling that it would be nice to somehow reuse this function for both

```go
1  package main
2
3  import "fmt"
4
5  type bot interface {
6      getGreeting() string
7  }
8
9  type englishBot struct{}
10 type spanishBot struct{}
11
12 func main() {
13     eb := englishBot{}
14     sb := spanishBot{}
15
16     printGreeting(eb)
17     printGreeting(sb)
```

interface thing is,

Ln 6, Col 1 (20 selected)   Tab Size: 4   UTF-8   LF   Go

master*

125%

type englishBot struct

func (englishBot) **getGreeting**() string

type spanishBot struct

func (spanishBot) **getGreeting**() string

To whom it may concern...

type bot interface

Our program has a new type called 'bot'

getGreeting() string

If you are a type in this program with a function called 'getGreeting' and you return a string then you are now an honorary member of type 'bot'

Now that you're also an honorary member of type 'bot', you can now call this function called 'printGreeting'

func printGreeting(b bot)

has a type of English bot inside of it and it still has a type of