Type Aliasing

This lesson takes a look at the concept of aliasing using an example

WE'LL COVER THE FOLLOWING

- Example

To define methods on a type you don't "own", you need to define an *alias* for the type you want to extend.

Example

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import (
        "fmt"
        "strings"
)

type MyStr string //using MyStr as an alias for type string

func (s MyStr) Uppercase() string {
        return strings.ToUpper(string(s))
}

func main() {
        fmt.Println(MyStr("test").Uppercase())
}
```

Down below is another example explaining the concept of aliasing for better understanding.

Environment Variables

Key:	Value:
GOPATH	/go

```go
package main

import (
    "fmt"
    "math"
)

type MyFloat float64   //using MyFloat as an alias for type float64

func (f MyFloat) Abs() float64 {
    if f < 0 {
        return float64(-f)
    }
    return float64(f)
}

func main() {
    f := MyFloat(-math.Sqrt2)
    fmt.Println(f.Abs())
}
```

As you can see above, type aliasing declares a new name to be used as a substitute for a previously known type. It does not introduce a new type, neither does it change the meaning of the existing type name.

In the next lesson, we will discuss *method receivers*.