Method Receivers

This lesson explains with examples method receivers in Go

WE'LL COVER THE FOLLOWING

- Example

Methods can be associated with a *named* type (`User` for instance) or a *pointer* to a named type (`*User`). In the two type aliasing examples in previous lesson, methods were defined on the value types (`MyStr` and `MyFloat`).

Example

There are two reasons to use a *pointer receiver*. First, to avoid copying the value on each method call (more efficient if the value type is a large struct). The previous `User` example would have been better written as follows:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import (
        "fmt"
)

type User struct {
        FirstName, LastName string
}

func (u *User) Greeting() string { //pointers
        return fmt.Sprintf("Dear %s %s", u.FirstName, u.LastName)
}

func main() {
        u := &User{"Matt", "Aimonetti"}
        fmt.Println(u.Greeting())
}
```

Remember that Go passes everything by value, meaning that when `Greeting()` is defined on the value type, every time you call `Greeting()`, you are copying the `User` struct. Instead when using a pointer, only the pointer is copied (cheap).

The other reason why you might want to use a pointer is so that the method can modify the value that its receiver points to.

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import (
        "fmt"
        "math"
)

type Vertex struct {
        X, Y float64
}

func (v *Vertex) Scale(f float64) {
        v.X = v.X * f //v will be modified directly here
        v.Y = v.Y * f
}

func (v *Vertex) Abs() float64 {
        return math.Sqrt(v.X*v.X + v.Y*v.Y)
}

func main() {
        v := &Vertex{3, 4}
        v.Scale(5)
        fmt.Println(v, v.Abs())
}
```

In the example above, `Abs()` could be defined on the value type or the pointer since the method doesn't modify the receiver value (the vertex). However `Scale()` has to be defined on a pointer since it does modify the receiver. `Scale()` resets the values of the `X` and `Y` fields.

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import (
        "fmt"
        "math"
)

type MyFloat 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())
}
```

This marks the end of this chapter. In the next chapter, we will discuss *interfaces*. Read on to find out more!