



Golang Programming

Methods. Composing Structs by Type Embedding

Where to Find The Code and Materials?

<https://github.com/iproduct/coursego>

Methods

```
type Vertex struct {  
    X, Y float64  
}
```

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

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

Methods on Non Structs

```
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())  
}
```

Methods on Non Structs

```
type Role int

const (
    User Role = 1 << iota
    Manager
    Admin
    RoleMask = (1 << (iota)) - 1
)

func (r Role) String() string {
    switch r {
    case User:
        return "User"
    case Manager:
        return "Manager"
    case Admin:
        return "Admin"
    default:
        return "Invalid role"
    }
}
```

```
// Status type
type Status int
```

```
// User statuses enum
const (
    Registered Status = iota
    Active
    Disabled
)
```

```
// Returns string representation of the Role
func (r Status) String() string {
    switch r {
    case Registered:
        return "Registered"
    case Active:
        return "Active"
    case Disabled:
        return "Disabled"
    default:
        return "Invalid status"
    }
}
```

Value and Pointer Receivers

```
type Vertex struct {  
    X, Y float64  
}  
  
func (v Vertex) Abs() float64 {  
    return math.Sqrt(v.X*v.X + v.Y*v.Y)  
}  
  
func (v *Vertex) Scale(f float64) {  
    v.X = v.X * f  
    v.Y = v.Y * f  
}  
  
func main() {  
    v := Vertex{3, 4}  
    v.Scale(10)  
    fmt.Println(v.Abs())  
}
```

Methods Are Just Like Functions

```
type Vertex struct {  
    X, Y float64  
}  
  
func Abs(v Vertex) float64 {  
    return math.Sqrt(v.X*v.X + v.Y*v.Y)  
}  
  
func Scale(v *Vertex, f float64) {  
    v.X = v.X * f  
    v.Y = v.Y * f  
}  
  
func main() {  
    v := Vertex{3, 4}  
    Scale(&v, 10)  
    fmt.Println(Abs(v))  
}
```

Methods and Pointer Indirection

```
type Vertex struct {
    X, Y float64
}
func (v *Vertex) Scale(f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}
func ScaleFunc(v *Vertex, f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}
func (v Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
func AbsFunc(v Vertex) float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

```
func main() {
    // Pointer receiver methods
    v := Vertex{3, 4}
    v.Scale(2)
    ScaleFunc(&v, 5)

    p := &Vertex{4, 3}
    p.Scale(5)
    ScaleFunc(p, 2)
    fmt.Println(v, p)

    // Value receiver methods
    fmt.Println(v.Abs())
    fmt.Println(AbsFunc(v))

    fmt.Println(p.Abs())
    fmt.Println(AbsFunc(*p))
}
```


Methods: Value and Pointer Receivers

```
type ByteSlice []byte
```

```
func (slice ByteSlice) Append(data []byte) []byte {  
    return append([]byte(slice), data...)  
}
```

```
func (slice *ByteSlice) AppendPointer(data []byte) {  
    *slice = append([]byte(*slice), data...)  
}
```

```
func (slice *ByteSlice) Write(data []byte) (n int, err error) {  
    *slice = append([]byte(*slice), data...)  
    return len(data), nil  
}
```

```
func main() {  
    var b ByteSlice  
    fmt.Fprintf(&b, "This hour has %d days\n", 7)  
    fmt.Printf("%v", b)  
}
```

Choosing Value or Pointer Receiver

There are two reasons to use a pointer receiver:

- The first is so that the **method can modify the value** that its receiver points to.
- The second is to **avoid copying the value** on each method call. This can be more efficient if the receiver is a large **struct**, for example.
- In general, **all methods** on a given type **should have either value or pointer receivers**, but not a mixture of both.
- More about selectors and method expressions:
<https://golang.org/ref/spec#Selectors>

Method Receivers and Interfaces

```
type Abser interface {
    Abs() float64
}

func main() {
    var a Abser
    f := MyFloat(-math.Sqrt2)
    v := Vertex{3, 4}

    a = f // MyFloat implements Abser
    fmt.Println(a.Abs())
    a = &v // *Vertex implements Abser

    // Vertex do not implement Abser
    //a = v

    fmt.Println(a.Abs())
}
```

```
type MyFloat float64

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

type Vertex struct {
    X, Y float64
}

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

Methods with Nil Receivers

// Path represents a sequence of Vertices. A nil Path represents empty sequence.

```
type Path []Vertex
func (p *Path) Distance() (dist float64) {
    dist = 0
    if *p == nil || len(*p) == 0 {
        return 0
    }
    v1 := (*p)[0]
    var v2 Vertex
    for i := 1; i < len(*p); i++ {
        v2 = (*p)[i]
        dist += v1.Distance(v2)
        v1 = v2
    }
    return
}

func main() {
    var path Path
    path = Path{{1, 1}, {4, 5}, {4, 1}, {1, 1}}
    fmt.Println("Perimeter = ", path.Distance())
}
```

Composing structs by Type Embedding

```
type ColorVertex struct {  
    Vertex  
    Color color.RGBA  
}  
  
func main() {  
    green := color.RGBA{0, 255, 0, 255}  
    yellow := color.RGBA{255, 255, 0, 255}  
    cv1 := ColorVertex{Vertex{2, 3}, green}  
    cv2 := ColorVertex{Vertex{6, 6}, yellow}  
    fmt.Println(cv1.Distance(cv2.Vertex)) // 5  
    cv1.Scale(4)  
    cv2.Scale(4)  
    fmt.Println(cv1.Distance(cv2.Vertex)) // 20  
    // cv1.Distance(cv2) // no cv1.Distance(ColorVertex)  
}
```

```
type Vertex struct {  
    X, Y float64  
}  
  
func (v Vertex) Distance(o Vertex) float64 {  
    return math.Hypot(o.X-v.X, o.Y-v.Y)  
}  
  
func (v *Vertex) Scale(f float64) {  
    v.X = v.X * f  
    v.Y = v.Y * f  
}
```

Composing structs by Pointer Type Embedding

```
type ColorVertexP struct {  
    *Vertex  
    Color color.RGBA  
}  
  
cvp1 := ColorVertexP{&Vertex{2, 7}, green}  
cvp2 := ColorVertexP{&Vertex{5, 3}, yellow}  
fmt.Println(cvp1.Distance(*cvp2.Vertex)) // "5"  
cvp1.Vertex = cvp2.Vertex  
cvp1.Scale(3)  
fmt.Println(*cvp1.Vertex, *cvp2.Vertex) // {15 9} {15 9}
```

Rules for Method Promotion – Value vs. Pointer Fields

Given a **struct** type S and a **defined type** T , promoted methods are included in the **method set** of the **struct** as follows:

- If S contains an embedded field T , the method sets of S and $*S$ both include promoted methods with **receiver** T . The method set of $*S$ also includes promoted methods with **receiver** $*T$.
- If S contains an embedded field $*T$, the method sets of S and $*S$ both include promoted methods with **receiver** T or $*T$.

Method Values and Expressions

```
a := Vertex{2, 7}
b := Vertex{5, 3}
```

```
distance := Vertex.Distance // method expression
fmt.Println(distance(a, b)) // 5
fmt.Printf("%T\n", distance) // func(main.Vertex, main.Vertex) float64
```

```
scale := (*Vertex).Scale // method expression
scale(&a, 2)
fmt.Println(a) // {4 14}
fmt.Printf("%T\n", scale) // func(*main.Vertex, float64)
```

```
scaleB := (&b).Scale // method value
fmt.Printf("%T\n", scaleB) // func(float64)
scaleB(2)
fmt.Printf("Scaling b with factor 2: b now is %f\n", b) //{10 6}
```

```
distanceFromA := a.Distance // method value
fmt.Printf("%T\n", distanceFromA) // func(*Vertex, float64)
fmt.Printf("Distance from A of B is %f\n", distanceFromA(b)) //10
```


Encapsulation

- State encapsulation:

```
type IntSet struct {  
    words []uint64  
}
```

- No state encapsulation:

```
type IntSet []uint64
```

Examples

- IntBitSet
- PriorityQueue
- HttpServer

Interface http.Handler

```
type Handler interface {  
    ServeHTTP(ResponseWriter, *Request)  
}
```

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

```
func hello(w http.ResponseWriter, req *http.Request) {  
    fmt.Fprintf(w, "hello\n")  
}  
func headers(w http.ResponseWriter, r *http.Request) {  
    fmt.Fprintf(w, "%s %s %s\n", r.Method, r.URL, r.Proto)  
    fmt.Fprintf(w, "Host = %q\nRemoteAddr = %q\n\n ", r.Host, r.RemoteAddr)  
    for name, headers := range r.Header {  
        for _, h := range headers {  
            fmt.Fprintf(w, "%v: %v\n", name, h)  
        }  
    }  
}  
func main() {  
    http.HandleFunc("/hello", hello)  
    http.HandleFunc("/headers", headers)  
    http.ListenAndServe(":8080", nil)  
}
```

Example: HTTP QR Link Generator



localhost:8080/?s=https%3A%2F%2Fgithub.com%2Fiproduct%2Fcoursego&q=Show+QR



<https://github.com/iproduct/coursego>

Show QR

Example: HTTP - QR Link Generator - Code

```
import ( "flag"; "html/template"; "log"; "net/http")

var addr = flag.String("addr", ":8080", "http service address")
var templ = template.Must(template.New("qr").Parse(templateStr))

func main() {
    flag.Parse()
    http.Handle("/", http.HandlerFunc(QR))
    err := http.ListenAndServe(*addr, nil)
    if err != nil {
        log.Fatal("ListenAndServe:", err)
    }
}

// QR http handler function return the HTML template with QR code for the input link
func QR(w http.ResponseWriter, req *http.Request) {
    templ.Execute(w, req.FormValue("s"))
}
```

HTML Template [\[https://golang.org/pkg/html/template/\]](https://golang.org/pkg/html/template/)

```
const templateStr = `  
<html>  
<head>  
<title>QR Link Generator</title>  
</head>  
<body>  
{{if .}}  
  
<br>  
{{.}}  
<br>  
<br>  
{{end}}  
<form action="/" name=f method="GET"><input maxLength=1024 size=70  
name=s value="" title="Text to QR Encode"><input type=submit  
value="Show QR" name=qr>  
</form>  
</body>  
</html>  
`
```

Text Templates [\[https://golang.org/pkg/text/template/\]](https://golang.org/pkg/text/template/)

```
import ("html/template"; "log"; "os")
const textTempl =
`{{len .}} issues:
{{range .}}-----
ID: {{.ID}}
Title: {{.Title | printf "%.64s"}}
{{end}}`

func main() {
    tmpl := template.New("report")
    tmpl, err := tmpl.Parse(textTempl)
    if err != nil {
        log.Fatal("Error Parsing template: ", err)
        return
    }
    err1 := tmpl.Execute(os.Stdout, goBooks)
    if err1 != nil {
        log.Fatal("Error executing template: ", err1)
    }
}
```

3 books:

ID: fmd-DwAAQBAJ

Title: Hands-On Software Architecture with Golang

ID: o86PDwAAQBAJ

Title: Learn Data Structures with Golang

ID: xfPEDwAAQBAJ

Title: From Ruby to Golang

JSON Marshalling and Unmarshalling

// Structs --> JSON

```
data, err := json.Marshal(goBooks)
if err != nil {
    log.Fatalf("JSON marshaling failed: %s", err)
}
fmt.Printf("%s\n", data)
```

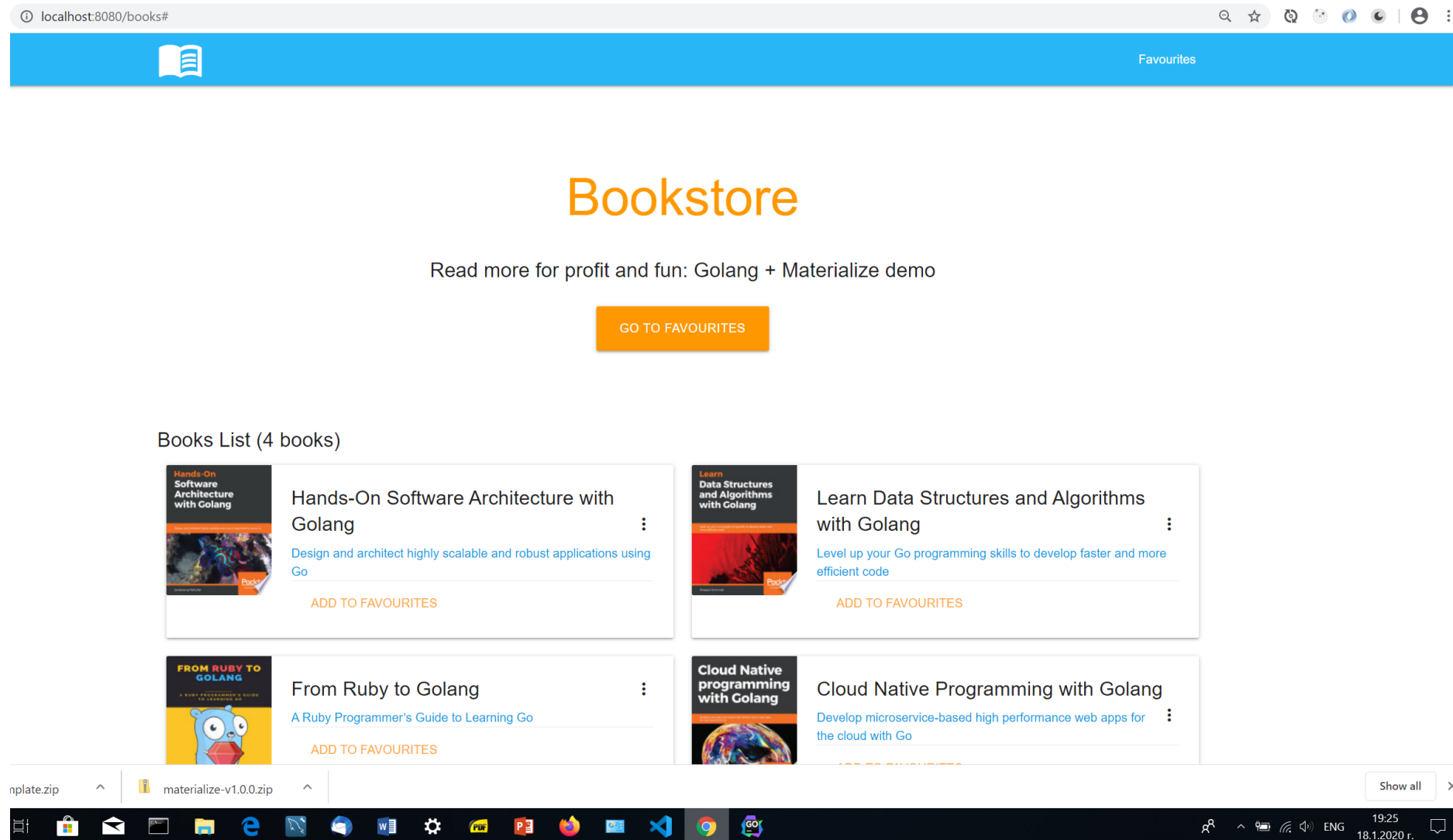
// Prettier formatting

```
data, err = json.MarshalIndent(goBooks, "", "    ")
if err != nil {
    log.Fatalf("JSON marshaling failed: %s", err)
}
fmt.Printf("%s\n", data)
```

// JSON -> structs

```
var books []Book
if err := json.Unmarshal(data, &books); err != nil {
    log.Fatalf("JSON unmarshaling failed: %s", err)
}
fmt.Println("AFTER UNMARSHAL:\n", books)
```


Example: Bookstore Web App



<https://github.com/iproduct/coursego/tree/master/http/bookstore>

Routing with http.ServeMux (1)

```
type database map[string]Book
```

```
func (db database) list(w http.ResponseWriter, req *http.Request) {  
    for _, book := range db {  
        fmt.Fprintf(w, "%s: %s - $%6.2f\n", book.ID, book.Title, book.RetailPrice)  
    }  
}
```

```
func (db database) price(w http.ResponseWriter, req *http.Request) {  
    id := req.URL.Query().Get("id")  
    if book, ok := db[id]; ok {  
        fmt.Fprintf(w, "$%6.2f\n", book.RetailPrice)  
    } else {  
        w.WriteHeader(http.StatusNotFound) // 404  
        fmt.Fprintf(w, "no book with ID: %q\n", id)  
    }  
}
```

Routing with http.ServeMux (2)

```
var db database = make(map[string]Book, 10)

func init() {
    for _, book := range goBooks {
        db[book.ID] = book
    }
}

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/list", db.list)
    mux.HandleFunc("/price", db.price)
    log.Fatal(http.ListenAndServe("localhost:8080", mux))
}
```

Homework 2 (GitHub API Client)

Implement GitHub API HTTP client that will:

- Read a text file given as command line argument to the program and parse different Github usernames – each username on separate line in the file
- Fetch GitHub users data in JSON format using public GitHub API:
[https://api.github.com/users/\\${username}](https://api.github.com/users/${username})
- Fetch GitHub user repositories data in JSON format from:
[https://api.github.com/users/\\${username}/repos](https://api.github.com/users/${username}/repos)
- Parse the JSON data using [json.Unmarshal](#) into appropriate data structures in Go.
- Print a statistics report containing the information about the user and the languages used and number of files in different repositories.

Recommended Literature

- The Go Documentation - <https://golang.org/doc/>
- The Go Bible: Effective Go - https://golang.org/doc/effective_go.html
- David Chisnall, *The Go Programming Language Phrasebook*, Addison Wesley, 2012
- Alan A. A. Donovan, Brian W. Kernighan, *The Go Programming Language*, Addison Wesley, 2016
- Nathan Youngman, Roger Peppé, *Get Programming with Go*, Manning, 2018
- Naren Yellavula, *Building RESTful Web Services with Go*, Packt, 2017

Thank's for Your Attention!



Trayan Iliev

IPT – Intellectual Products & Technologies

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>