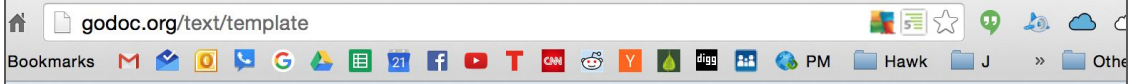
Templates

text templates



GoDoc Home Index About Search

Go: text/template Index | Examples | Files | Directories

package template

```
import "text/template"
```

Package template implements data-driven templates for generating textual output.

To generate HTML output, see package html/template, which has the same interface as this package but automatically secures HTML output against certain attacks.

Templates are executed by applying them to a data structure. Annotations in the template refer to elements of the data structure (typically a field of a struct or a key in a map) to control execution and derive values to be displayed. Execution of the template walks the structure and sets the cursor, represented by a period '.' and called "dot", to the value at the current location in the structure as execution proceeds.

The input text for a template is UTF-8-encoded text in any format. "Actions"--data evaluations or control structures--are delimited by "{{" and "}}"; all text outside actions is copied to the output unchanged. Except for raw strings, actions may not span newlines, although comments can.

Once parsed, a template may be executed safely in parallel.

Here is a trivial example that prints "17 items are made of wool".

```
type Inventory struct {
        Material string
        Count    uint
}
sweaters := Inventory{"wool", 17}
tmpl, err := template.New("test").Parse("{{.Count}} items are made of {{.Material}}")
if err != nil { panic(err) }
err = tmpl.Execute(os.Stdout, sweaters)
if err != nil { panic(err) }
```

main.go × tpl.gohtml ×

```go
package main

import (
    "log"
    "os"
    "text/template"
)

func main() {
    tpl, err := template.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, nil)
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go × tpl.gohtml ×

```gohtml
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>Hello</h1>
</body>
</html>
```

Terminal

```
01 $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>Hello</h1>
</body>
</html>01 $
```

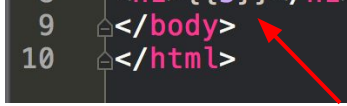
template actions

main.go × tpl.gohtml ×

```go
package main

import (
    "log"
    "os"
    "text/template"
)

func main() {
    tpl, err := template.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, nil)
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go × tpl.gohtml ×

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>{{5}}</h1>
</body>
</html>
```



Terminal

```
02 $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>5</h1>
</body>
</html>02 $
```

main.go × tpl.gohtml ×

```go
package main

import (
    "log"
    "os"
    "text/template"
)

func main() {
    tpl, err := template.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, 10)
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go × tpl.gohtml ×

```gohtml
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>{{.}}</h1>
</body>
</html>
```

Terminal

```
03 $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>10</h1>
</body>
</html>03 $
```

Think of the “.” like the “.” in Unix. Just like in Unix the “.” means the current directory, here the “.” means the current data the “.” is known as the pipeline

main.go × tpl.gohtml ×

```go
package main

import (
    "log"
    "os"
    "text/template"
)

func main() {
    tpl, err := template.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, 5*5)
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go × tpl.gohtml ×

```gohtml
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>{{.}}</h1>
</body>
</html>
```

Terminal

```
04 $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>25</h1>
</body>
</html>04 $
```

Think of the “.” like the “.” in Unix. Just like in Unix the “.” means the current directory, here the “.” means the current data

main.go

```go
package main

import (
    "log"
    "os"
    "text/template"
)

func main() {
    tpl, err := template.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, []int{1,2,3,4,5,})
    if err != nil {
        log.Fatalln(err)
    }
}
```

tpl.gohtml

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<ul>
    {{ range . }}
    <li>{{ . }}</li>
    {{ end }}
</ul>
</body>
</html>
```

Terminal

```
<!doctype html>n.go
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<ul>

    <li>1</li>

    <li>2</li>

    <li>3</li>

    <li>4</li>

    <li>5</li>

</ul>
</body>
05 $
```

Think of the “.” like the “.” in Unix. Just like in Unix the “.” means the current directory, here the “.” means the current data

Go's templates are similar to other templating

Logic-less templates.

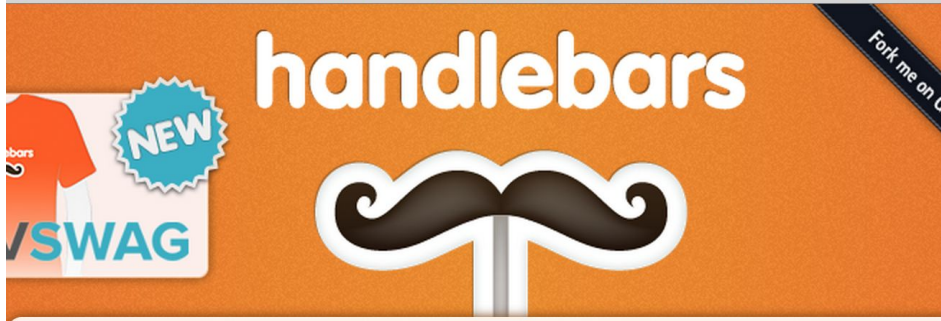
Available in Ruby, JavaScript, Python, Erlang, node.js, PHP, Perl, Perl6, Objective-C, Java, C#/.NET, Android, C++, Go, Lua, ooc, ActionScript, ColdFusion, Scala, Clojure, Fantom, CoffeeScript, D, Haskell, XQuery, ASP, Io, Dart, Haxe, Delphi, Racket, Rust, OCaml, Swift, Bash, Julia, R, Crystal, and for Common Lisp

Works great with TextMate, Vim, Emacs, Coda, and Atom

The Manual: mustache(5) and mustache(1)

handlebarsjs.com

Apps Bookmarks Other Bo



Handlebars provides the power necessary to let you build **semantic templates** effectively with no frustration.

Handlebars is largely compatible with Mustache templates. In most cases it is to swap out Mustache with Handlebars and continue using your current templa Complete details can be found here.

Installation

Getting Started

Handlebars templates look like regular HTML, with embedded handlebars expressions.

```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

main.go

```go
package main

import (
    "log"
    "os"
    "text/template"
)

func main() {
    tpl, err := template.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, []int{})
    if err != nil {
        log.Fatalln(err)
    }
}
```

tpl.gohtml

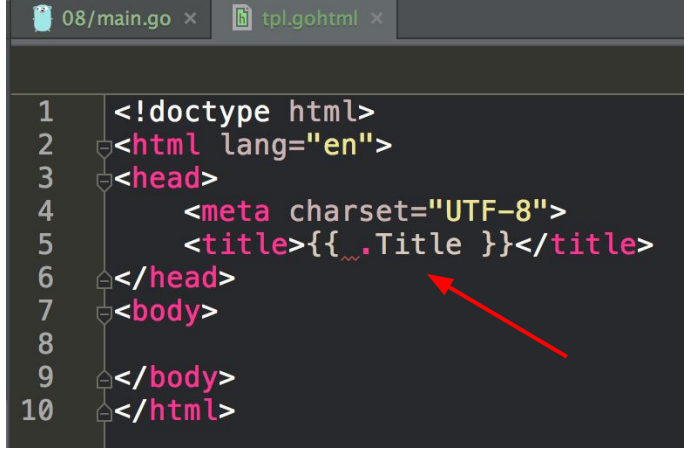
```gohtml
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<ul>
    {{ range . }}
    <li>{{ . }}</li>
    {{ else }}
    <li>NO ITEMS</li>
    {{ end }}
</ul>
</body>
</html>
```

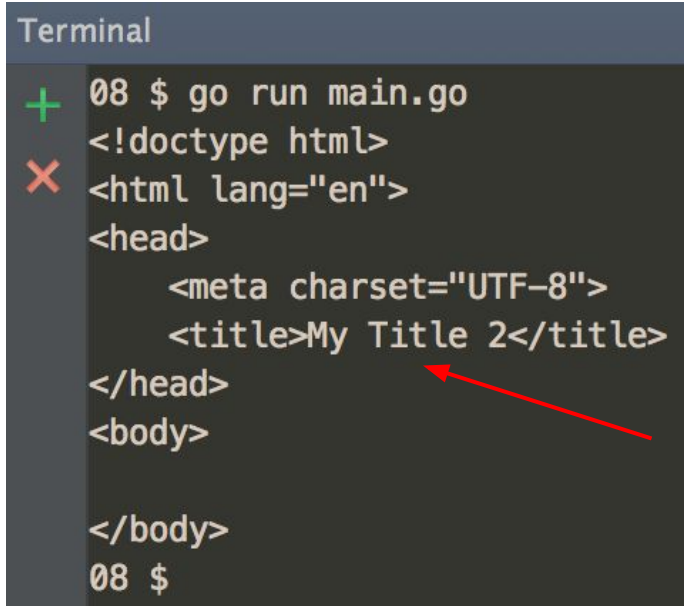
Terminal

```
06 $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<ul>

    <li>NO ITEMS</li>

</ul>
</body>
06 $
```

08/main.go

```go
package main

import (
    "log"
    "os"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    tpl, err := template.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title 2",
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```





main.go × tpl.gohtml ×

```go
package main

import (
    "log"
    "os"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "mycustomfunc": func() string {
            return "This should work"
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title",
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go × tpl.gohtml ×

```gohtml
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ .Title }}</title>
</head>
<body>
{{mycustomfunc}}
</body>
</html>
```

Terminal

```
09_function $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My Title</title>
</head>
<body>
This should work
</body>
</html>09_function $
```

main.go × tpl.gohtml ×

```go
package main

import (
    "log"
    "os"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "mycustomfunc": func() string {
            return "This should work"
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title",
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

godoc.org/text/template

type Template

- func Must(t *Template, err error) *Template
- func New(name string) *Template
- func ParseFiles(filenames ...string) (*Template, error)
- func ParseGlob(pattern string) (*Template, error)
- func (t *Template) AddParseTree(name string, tree *parse.Tree) (*Template, error)
- func (t *Template) Clone() (*Template, error)
- func (t *Template) DefinedTemplates() string
- func (t *Template) Delims(left, right string) *Template
- func (t *Template) Execute(wr io.Writer, data interface{}) (err error)
- func (t *Template) ExecuteTemplate(wr io.Writer, name string, data interface{}) error
- func (t *Template) Funcs(funcMap FuncMap) *Template
- func (t *Template) Lookup(name string) *Template
- func (t *Template) Name() string
- func (t *Template) New(name string) *Template
- func (t *Template) Option(opt ...string) *Template
- func (t *Template) Parse(text string) (*Template, error)
- func (t *Template) ParseFiles(filenames ...string) (*Template, error)
- func (t *Template) ParseGlob(pattern string) (*Template, error)
- func (t *Template) Templates() []*Template

main.go × tpl.gohtml ×

```go
package main

import (
    "log"
    "os"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "mycustomfunc": func() string {
            return "This should work"
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title",
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

godoc.org/text/template

type Template

- func Must(t *Template, err error) *Template
- func New(name string) *Template
- func ParseFiles(filenames ...string) (*Template, error)
- func P… Glob(pattern string) (*Template, error)
- func (… ParseTree(name string, tree *parse.Tree) (*Template, error)
- func … error)
- func …
- fun…
- fu…
- f…
- func (t *Template) …
- func (t *Template) Name() string
- func (t *Template) New(name string) *Templ…
- func (t *Template) Option(opt ...string) *Template
- func (t *Template) Parse(text string) (*Template, error)
- func (t *Template) ParseFiles(filenames ...string) (*Template, error)
- func (t *Template) ParseGlob(pattern string) (*Template, error)
- func (t *Template) Templates() []*Template

func New

```
func New(name string) *Template
```

New allocates a new, undefined template with the given name.

main.go | tpl.gohtml

```go
package main

import (
    "log"
    "os"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "mycustomfunc": func() string {
            return "This should work"
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title",
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

godoc.org/text/template

type Template

- func Must(t *Template, err error) *Template
- func New(name string) *Template
- func ParseFiles(filenames ...string) (*Template, error)
- func P…Glob(pattern string) (*Template, error)
- func (…ParseTree(name string, tree *parse.Tree) (*Template, error)
- func …error)
- func …
- func …
- func …
- func (t *Templ…
- …

func New

func New(name string) *T…

New allocates a new…

func (*Template) Funcs

func (t *Template) Funcs(funcMap FuncMap) *Template

Funcs adds the elements of the argument map to the template's function map. It panics if a value in the map is not a function with appropriate return type. However, it is legal to overwrite elements of the map. The return value is the template, so calls can be chained.

…string) (*Template, error)

…(pattern string) (*Template, error)

…Templates() []*Template

main.go × tpl.gohtml ×

```go
package main

import (
    "log"
    "os"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "mycustomfunc": func() string {
            return "This should work"
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title",
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

godoc.org/text/template

type Template

- func Must(t *Template, err error) *Template
- func New(name string) *Template
- func ParseFiles(filenames ...string) (*Template, error)

type FuncMap

```
type FuncMap map[string]interface{}
```

FuncMap is the type of the map defining the mapping from names to functions. Each function must have either a single return value, or two return values of which the second has type error. In that case, if the second (error) return value evaluates to non-nil during execution, execution terminates and Execute returns that error.

func (*Template) Funcs

```
func (t *Template) Funcs(funcMap FuncMap) *Template
```

Funcs adds the elements of the argument map to the template's function map. It panics if a value in the map is not a function with appropriate return type. However, it is legal to overwrite elements of the map. The return value is the template, so calls can be chained.

main.go | tpl.gohtml

```go
package main

import (
    "log"
    "os"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "mycustomfunc": func() string {
            return "This should work"
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title",
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

godoc.org/text/template

type Template

- func Must(t *Template, err error) *Template
- func New(name string) *Template
- func ParseFiles(filenames ...string) (*Template, error)

type FuncMap

```
type FuncMap map[string]interface{}
```

FuncMap is the type of the map defining the mapping from names to functions. Each function must have either value, or two return values of which the second has type error. In that case, if the second (error) non-nil during execution, execution terminates and Execute returns that error.

func (*Template) ParseFiles

```
func (t *Template) ParseFiles(filenames ...string) (*Template, error)
```

ParseFiles parses the named files and associates the resulting templates with t. If an error occurs, parsing stops and the returned template is nil; otherwise it is t. There must be at least one file. Since the templates created by ParseFiles are named by the base names of the argument files, t should usually have the name of one of the (base) names of the files. If it does not, depending on t's contents before calling ParseFiles, t.Execute may fail. In that case use t.ExecuteTemplate to execute a valid template.

It panics if a value in the map is of the map. The return

Funcs adds not a function wi value is the template,

main.go | tpl.gohtml

```go
package main

import (
    "log"
    "os"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "mycustomfunc": func() string {
            return "This should work"
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title",
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go

```go
package main

import (
    "log"
    "os"
    "strings"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "uppercase": func(str string) string {
            return strings.ToUpper(str)
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title 2",
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go × tpl.gohtml ×

```gohtml
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{.Title }}</title>
</head>
<body>
{{uppercase .Title}}
</body>
</html>
```

Terminal

```
10_function $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My Title 2</title>
</head>
<body>
MY TITLE 2
</body>
</html>10_function $
```

main.go

```go
package main

import (
    "log"
    "os"
    "strings"
    "text/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "uppercase": func(str string) string {
            return strings.ToUpper(str)
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title 2",
        Body:  `hello world <script>alert("Yow!");</script>`,
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

tpl.gohtml

```gohtml
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ .Title }}</title>
</head>
<body>
{{uppercase .Title}}
{{ .Body }}
</body>
</html>
```

Terminal

```
11 $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My Title 2</title>
</head>
<body>
MY TITLE 2
hello world <script>alert("Yow!");</script>
</body>
</html>11 $
```

html templates

main.go

```go
package main

import (
    "log"
    "os"
    "strings"
    "html/template"
)

type Page struct {
    Title string
    Body  string
}

func main() {
    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "uppercase": func(str string) string {
            return strings.ToUpper(str)
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title 2",
        Body:  `hello world <script>alert("Yow!");<
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go | tpl.gohtml

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ .Title }}</title>
</head>
<body>
{{uppercase .Title}}
{{ .Body }}
</body>
</html>
```

Terminal

```
<!doctype html>n.go
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My Title 2</title>
</head>
<body>
MY TITLE 2
hello world &lt;script&gt;alert(&#34;Yow!&#34;);&lt;/script&gt;
</body>
</html>01 $
```

main.go

```go
package main

import (
    "html/template"
    "log"
    "os"
    "strings"
)

type Page struct {
    Title string
    Body  template.HTML
}

func main() {
    log.SetFlags(0)

    var err error

    tpl := template.New("tpl.gohtml")
    tpl = tpl.Funcs(template.FuncMap{
        "uppercase": func(str string) string {
            return strings.ToUpper(str)
        },
    })
    tpl, err = tpl.ParseFiles("tpl.gohtml")
    if err != nil {
        log.Fatalln(err)
    }
    err = tpl.Execute(os.Stdout, Page{
        Title: "My Title 2",
        Body:  template.HTML(`hello world <script>alert("Yow!");</script>`),
    })
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go | tpl.gohtml

```gohtml
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ .Title }}</title>
</head>
<body>
{{uppercase .Title}}
{{ .Body }}
</body>
</html>
```

Terminal

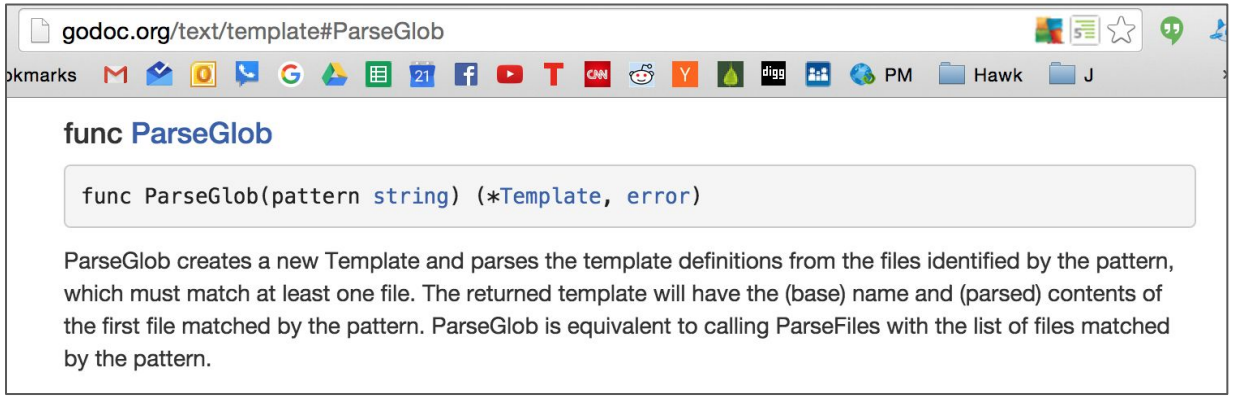
```
02 $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My Title 2</title>
</head>
<body>
MY TITLE 2
hello world <script>alert("Yow!");</script>
</body>
</html>02 $
```

ParseFiles
ParseGlob
Execute
ExecuteTemplate

func ParseFiles

```
func ParseFiles(filenames ...string) (*Template, error)
```

ParseFiles creates a new Template and parses the template definitions from the named files. The returned template's name will have the base name and parsed contents of the first file. There must be at least one file. If an error occurs, parsing stops and the returned *Template is nil.



func (*Template) Execute

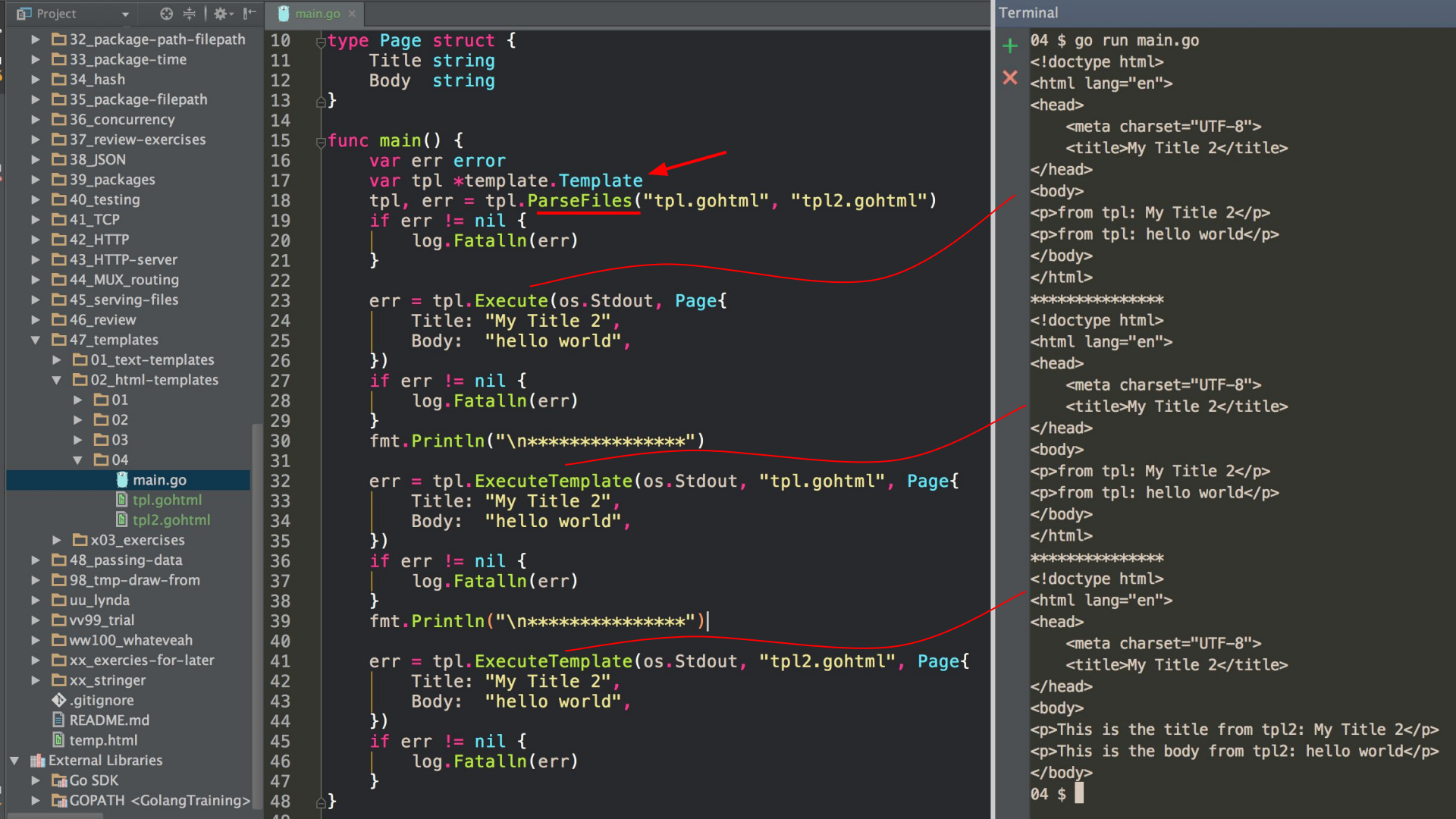
```
func (t *Template) Execute(wr io.Writer, data interface{}) (err error)
```

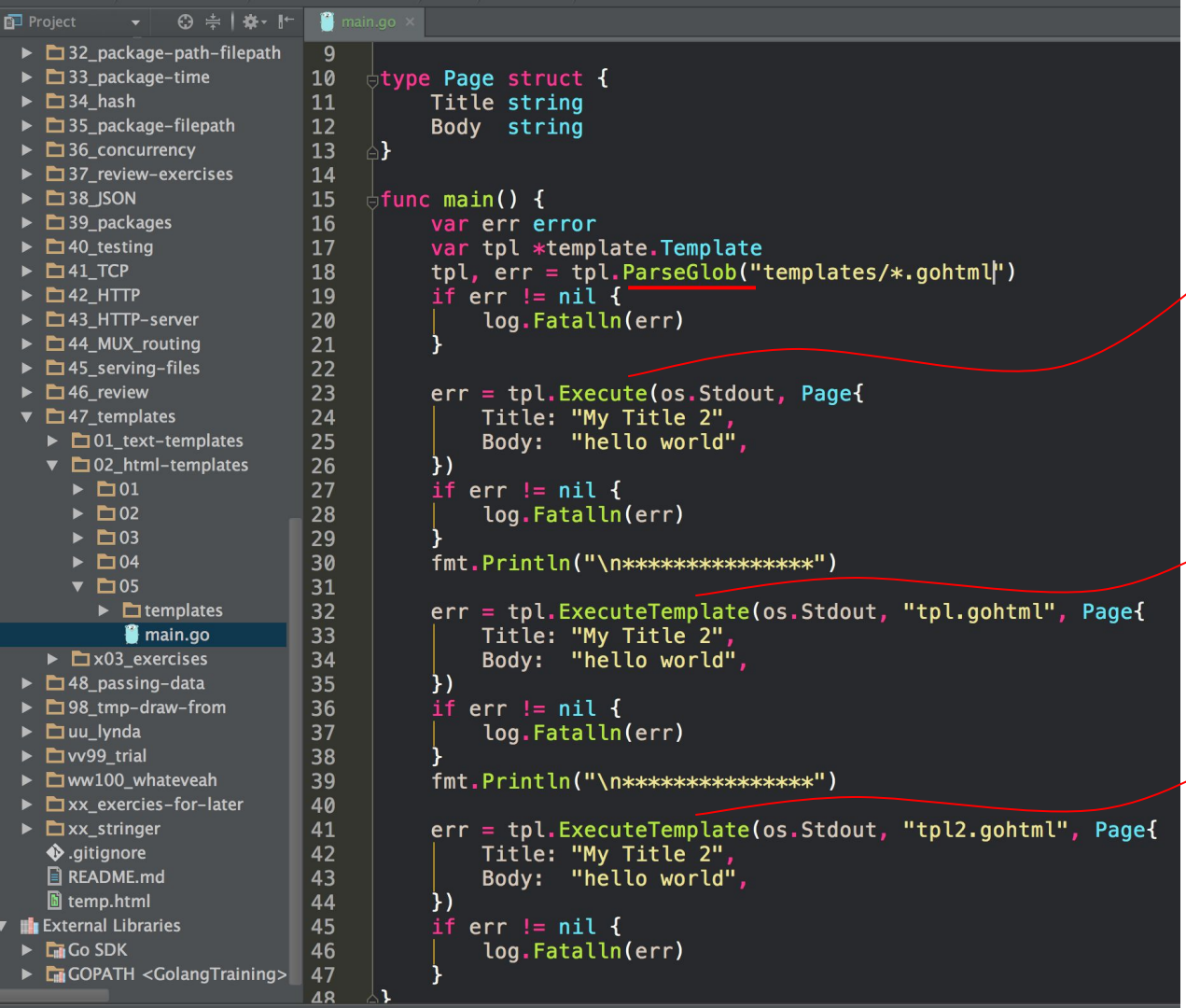
Execute applies a parsed template to the specified data object, and writes the output to wr. If an error occurs executing the template or writing its output, execution stops, but partial results may already have been written to the output writer. A template may be executed safely in parallel.

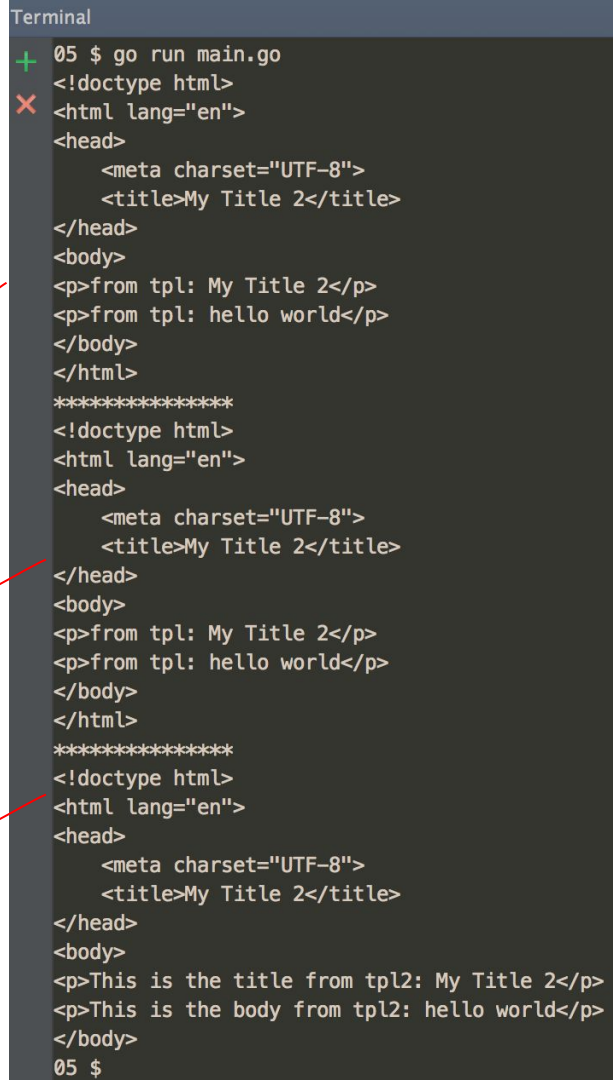
func (*Template) ExecuteTemplate

```
func (t *Template) ExecuteTemplate(wr io.Writer, name string, data interface{}) error
```

ExecuteTemplate applies the template associated with t that has the given name to the specified data object and writes the output to wr. If an error occurs executing the template or writing its output, execution stops, but partial results may already have been written to the output writer. A template may be executed safely in parallel.


Project
32_package-path-filepath
33_package-time
34_hash
35_package-filepath
36_concurrency
37_review-exercises
38_JSON
39_packages
40_testing
41_TCP
42_HTTP
43_HTTP-server
44_MUX_routing
45_serving-files
46_review
47_templates
01_text-templates
02_html-templates
01
02
03
04
main.go
tpl.gohtml
tpl2.gohtml
x03_exercises
48_passing-data
98_tmp-draw-from
uu_lynda
vv99_trial
ww100_whateveah
xx_exercies-for-later
xx_stringer
.gitignore
README.md
temp.html
External Libraries
Go SDK
GOPATH <GolangTraining>
main.go
10 type Page struct {
11 Title string
12 Body string
13 }
14
15 func main() {
16 var err error
17 var tpl *template.Template
18 tpl, err = tpl.ParseFiles("tpl.gohtml", "tpl2.gohtml")
19 if err != nil {
20 log.Fatalln(err)
21 }
22
23 err = tpl.Execute(os.Stdout, Page{
24 Title: "My Title 2",
25 Body: "hello world",
26 })
27 if err != nil {
28 log.Fatalln(err)
29 }
30 fmt.Println("\n***************")
31
32 err = tpl.ExecuteTemplate(os.Stdout, "tpl.gohtml", Page{
33 Title: "My Title 2",
34 Body: "hello world",
35 })
36 if err != nil {
37 log.Fatalln(err)
38 }
39 fmt.Println("\n***************")
40
41 err = tpl.ExecuteTemplate(os.Stdout, "tpl2.gohtml", Page{
42 Title: "My Title 2",
43 Body: "hello world",
44 })
45 if err != nil {
46 log.Fatalln(err)
47 }
48 }
Terminal
04 $ go run main.go
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>My Title 2</title>
</head>
<body>
from tpl: My Title 2
from tpl: hello world
</body>
</html>
****
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>My Title 2</title>
</head>
<body>
from tpl: My Title 2
from tpl: hello world
</body>
</html>
****
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>My Title 2</title>
</head>
<body>
This is the title from tpl2: My Title 2
This is the body from tpl2: hello world
</body>
04 $


Project
main.go
32_package-path-filepath
33_package-time
34_hash
35_package-filepath
36_concurrency
37_review-exercises
38_JSON
39_packages
40_testing
41_TCP
42_HTTP
43_HTTP-server
44_MUX_routing
45_serving-files
46_review
47_templates
01_text-templates
02_html-templates
01
02
03
04
05
templates
main.go
x03_exercises
48_passing-data
98_tmp-draw-from
uu_lynda
vv99_trial
ww100_whateveah
xx_exercies-for-later
xx_stringer
.gitignore
README.md
temp.html
External Libraries
Go SDK
GOPATH <GolangTraining>
type Page struct {
Title string
Body string
}
func main() {
var err error
var tpl *template.Template
tpl, err = tpl.ParseGlob("templates/*.gohtml")
if err != nil {
log.Fatalln(err)
}
err = tpl.Execute(os.Stdout, Page{
Title: "My Title 2",
Body: "hello world",
})
if err != nil {
log.Fatalln(err)
}
fmt.Println("\n***************")
err = tpl.ExecuteTemplate(os.Stdout, "tpl.gohtml", Page{
Title: "My Title 2",
Body: "hello world",
})
if err != nil {
log.Fatalln(err)
}
fmt.Println("\n***************")
err = tpl.ExecuteTemplate(os.Stdout, "tpl2.gohtml", Page{
Title: "My Title 2",
Body: "hello world",
})
if err != nil {
log.Fatalln(err)
}
}


Terminal
05 $ go run main.go
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>My Title 2</title>
</head>
<body>
from tpl: My Title 2
from tpl: hello world
</body>
</html>
****
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>My Title 2</title>
</head>
<body>
from tpl: My Title 2
from tpl: hello world
</body>
</html>
****
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>My Title 2</title>
</head>
<body>
This is the title from tpl2: My Title 2
This is the body from tpl2: hello world
</body>
05 $

exercises

Hello World

- Create an http application that returns Hello <Your Name> using a template.

main.go

```go
package main

import (
    "html/template"
    "log"
    "os"
)

func main() {
    
    name := "Todd"

    // parse template
    tpl, err := template.ParseFiles("hw.gohtml")
    if err != nil {
        log.Fatalln(err)
    }

    // execute template
    err = tpl.Execute(os.Stdout, name)
    if err != nil {
        log.Fatalln(err)
    }
}
```

main.go | hw.gohtml

```gohtml
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
Hello {{. }}
</body>
</html>
```

```
01 $ go run main.go
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
Hello Todd
</body>
```

Hello URL

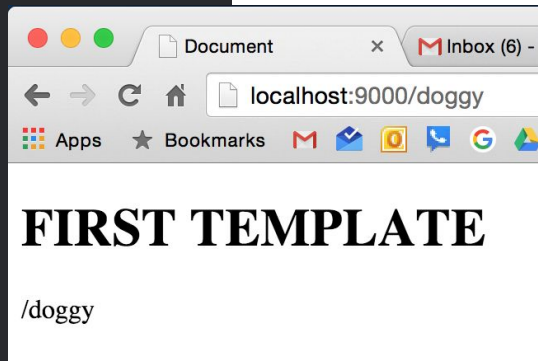
- Create an http application that returns the currently selected URL (use a variable in your template)

hw.gohtml × main.go ×

```go
package main

import (
    "html/template"
    "log"
    "net/http"
)

func main() {
    // parse template
    tpl, err := template.ParseFiles("hw.gohtml")
    if err != nil {
        log.Fatalln(err)
    }

    // function
    http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
        // execute template
        err = tpl.Execute(res, req.RequestURI)
        if err != nil {
            log.Fatalln(err)
        }
    })

    // create server
    http.ListenAndServe(":9000", nil)
}
```

hw.gohtml × main.go ×

```html
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>FIRST TEMPLATE</h1>
{{ . }}
</body>
</html>
```



Parse CSV

- Parse a CSV file then, using templates, send the results as HTML to a web browser.

table.csv × main.go ×

```
1  Date,Open,High,Low,Close,Volume,Adj Close
2  2015-07-09,523.119995,523.77002,520.349976,520.679993,1839400,520.679993
3  2015-07-08,521.049988,522.734009,516.109985,516.830017,1264600,516.830017
4  2015-07-07,523.130005,526.179993,515.179993,525.02002,1595700,525.02002
5  2015-07-06,519.50,525.25,519.00,522.859985,1276500,522.859985
6  2015-07-02,521.080017,524.650024,521.080017,523.400024,1234000,523.400024
```

Take this,
and make it look like this

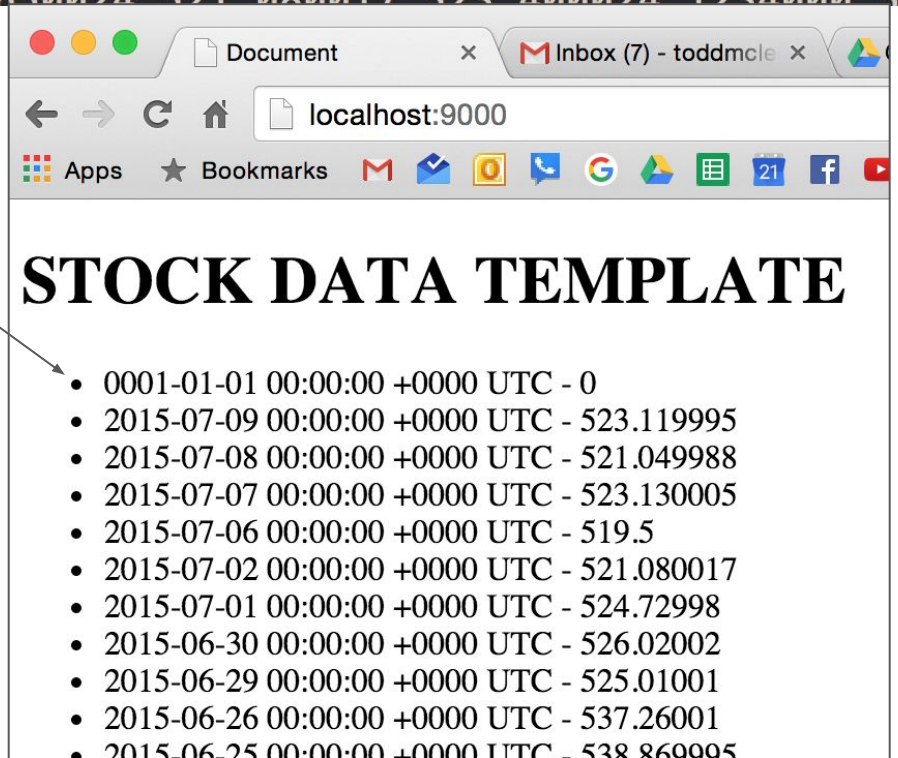


table.csv × main.go ×

```go
package main

import (
    "html/template"
    "log"
    "net/http"
    "github.com/goestoeleven/GolangTraining/47_templates/04_template_csv-parse/parse"
)

func main() {
    // parse csv
    records := parse.Parse("table.csv")

    // parse template
    tpl, err := template.ParseFiles("hw.gohtml")
    if err != nil {
        log.Fatalln(err)
    }

    // function
    http.HandleFunc("/", func(res http.ResponseWriter, req *http.Request) {
        // execute template
        err = tpl.Execute(res, records)
        if err != nil {
            log.Fatalln(err)
        }
    })

    // create server
    http.ListenAndServe(":9000", nil)
}
```

table.csv × main.go × parse.go ×

```go
import (
    "encoding/csv"
    "os"
    "strconv"
    "time"
)

type Record struct {
    Date time.Time
    Open float64
}

func Parse(filePath string) []Record {
    src, err := os.Open(filePath)
    if err != nil {
        panic(err)
    }
    defer src.Close()

    rows, err := csv.NewReader(src).ReadAll()
    if err != nil {
        panic(err)
    }

    records := make([]Record, 0, len(rows))

    for _, row := range rows {
        date, _ := time.Parse("2006-01-02", row[0])
        open, _ := strconv.ParseFloat(row[1], 64)

        records = append(records, Record{
            Date: date,
            Open: open,
        })
    }

    return records

}
```

table.csv × main.go × parse.go × hw.gohtml ×

```html
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<h1>STOCK DATA TEMPLATE</h1>
<ul>
    {{ range . }}
    <li>{{ .Date }} - {{ .Open }}</li>
    {{ end }}
</ul>

</body>
</html>
```