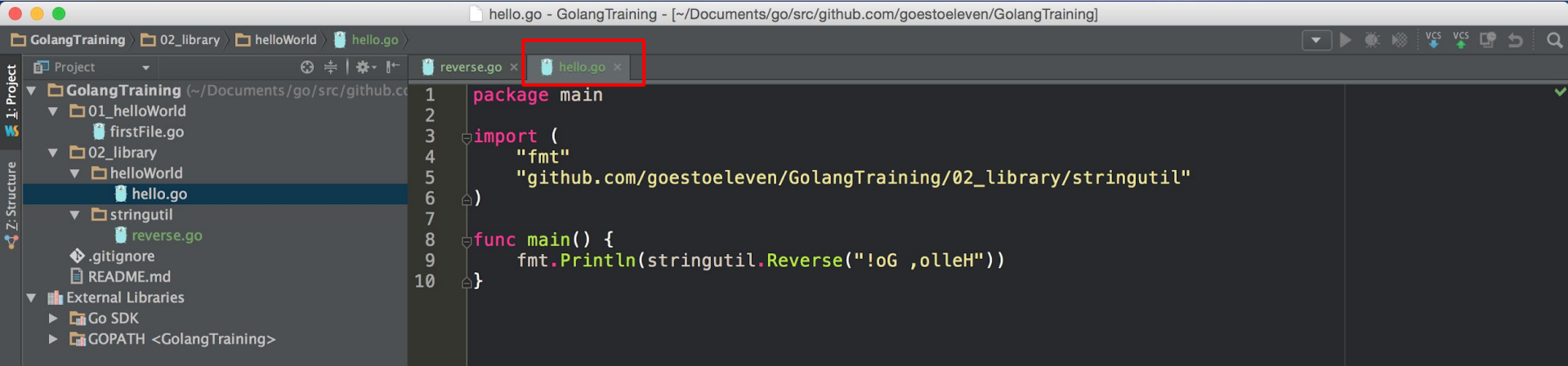


# **go build & go install**

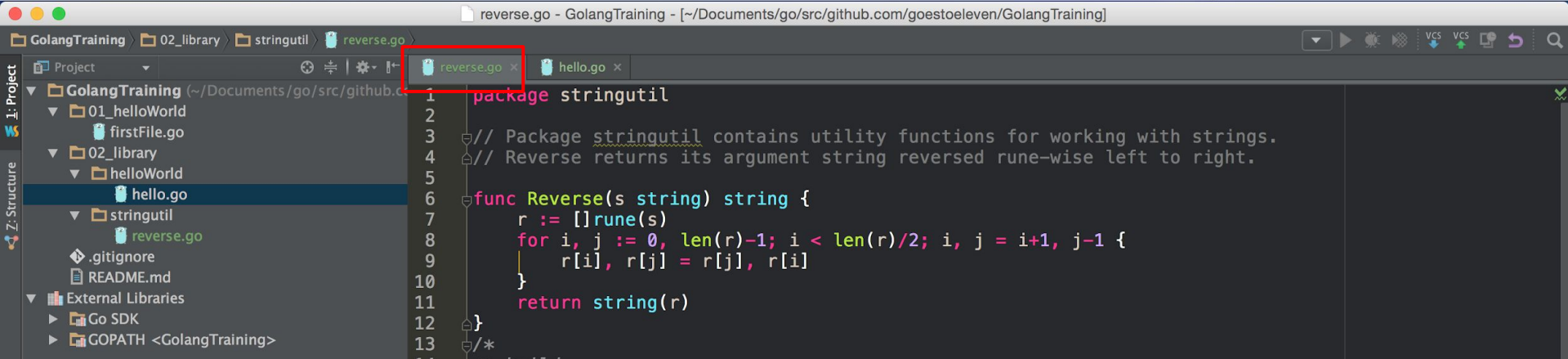
review

# two packages and two files from before ...



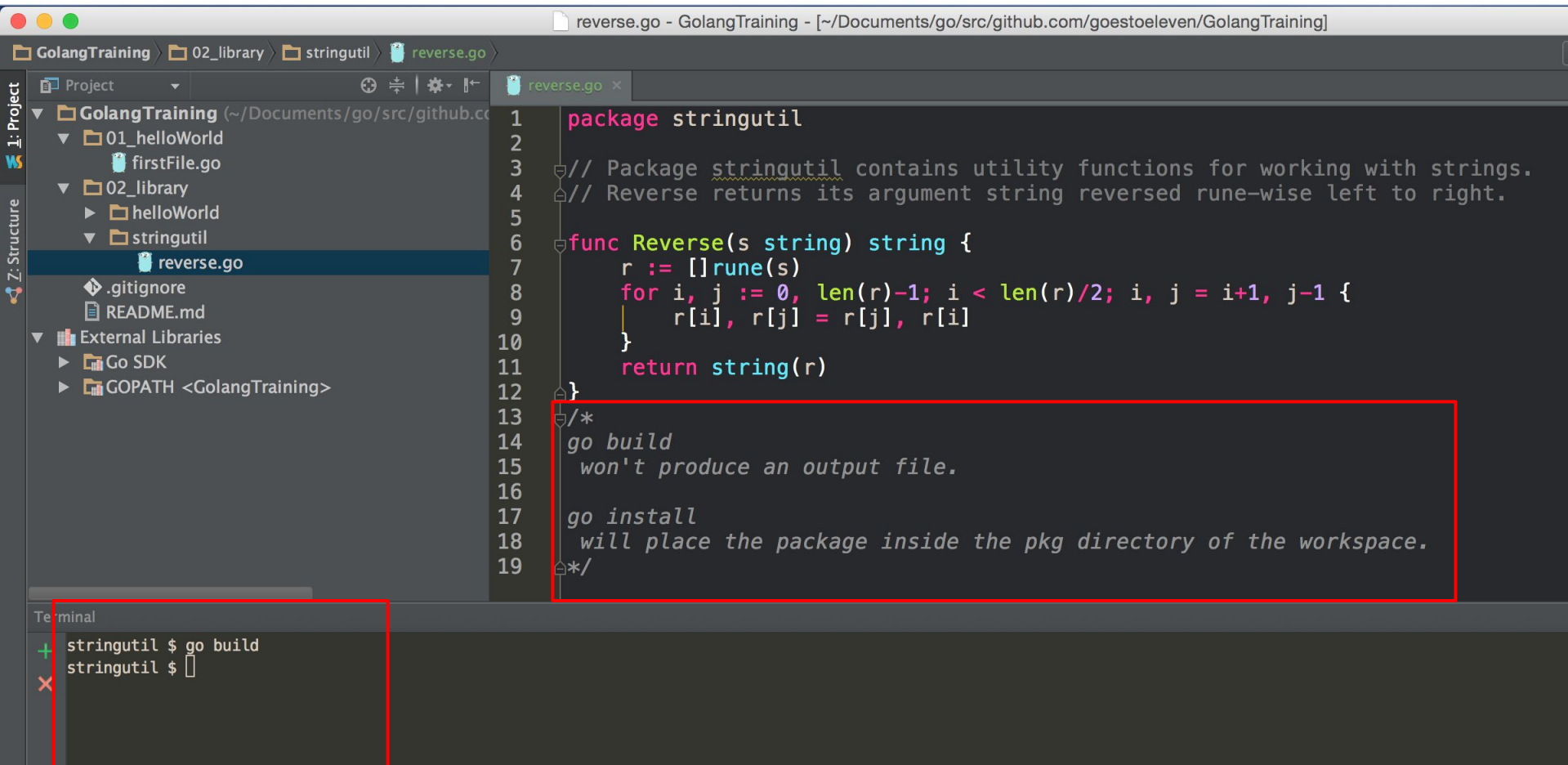
hello.go - GolangTraining - [~/Documents/go/src/github.com/goestoeleven/GolangTraining]

```
1 package main
2
3 import (
4     "fmt"
5     "github.com/goestoeleven/GolangTraining/02_library/stringutil"
6 )
7
8 func main() {
9     fmt.Println(stringutil.Reverse("!oG ,olleH"))
10 }
```



reverse.go - GolangTraining - [~/Documents/go/src/github.com/goestoeleven/GolangTraining]

```
1 package stringutil
2
3 // Package stringutil contains utility functions for working with strings.
4 // Reverse returns its argument string reversed rune-wise left to right.
5
6 func Reverse(s string) string {
7     r := []rune(s)
8     for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
9         r[i], r[j] = r[j], r[i]
10    }
11    return string(r)
12 }
13 /*
```





GolangTraining &gt; 02\_library &gt; stringutil &gt; reverse.go

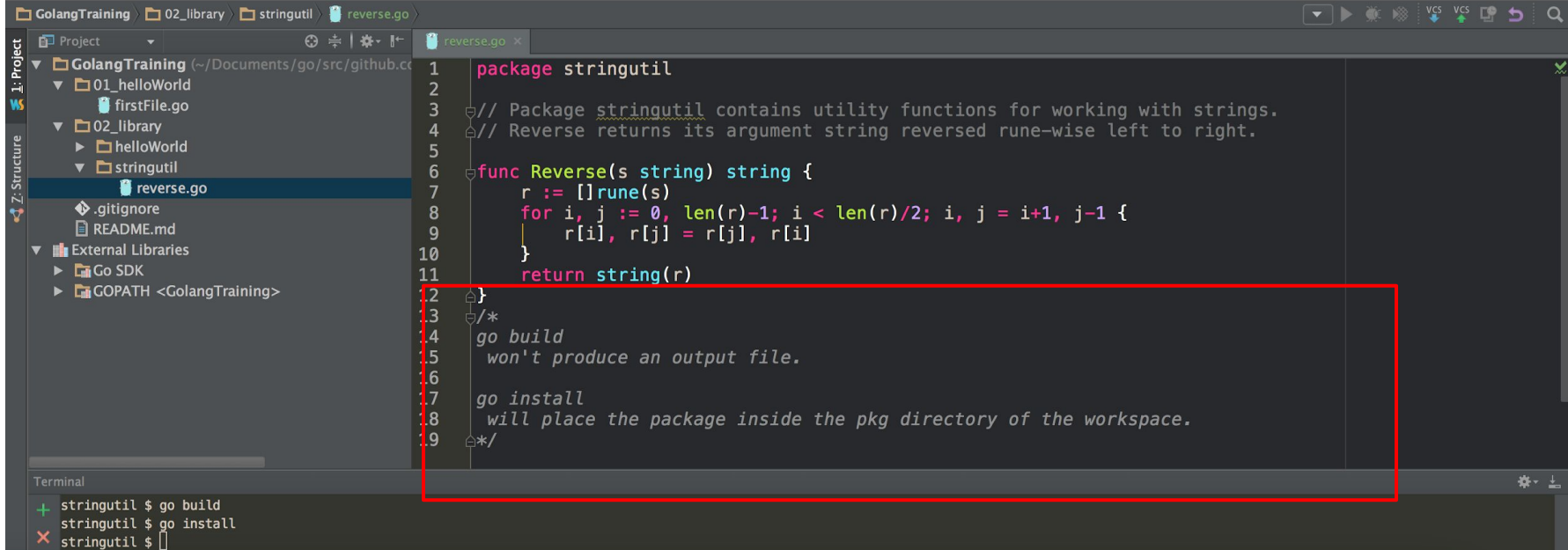
Project

- GolangTraining (~/Documents/go/src/github.com/goestoeleven/GolangTraining)
  - 01\_helloWorld
    - firstFile.go
  - 02\_library
    - helloWorld
    - stringutil
      - reverse.go
  - External Libraries
    - Go SDK
    - GOPATH <GolangTraining>

```
1 package stringutil
2
3 // Package stringutil contains utility functions for working with strings.
4 // Reverse returns its argument string reversed rune-wise left to right.
5
6 func Reverse(s string) string {
7     r := []rune(s)
8     for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
9         r[i], r[j] = r[j], r[i]
10    }
11    return string(r)
12 }
13 /*
14 go build
15 won't produce an output file.
16
17 go install
18 will place the package inside the pkg directory of the workspace.
19 */
```

Terminal

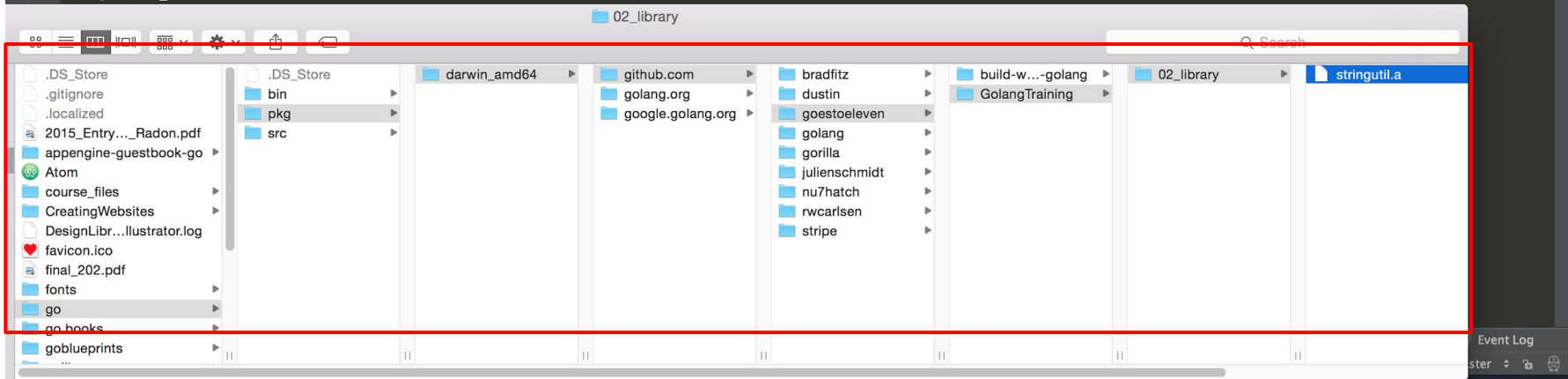
```
+ stringutil $ go build
+ stringutil $ go install
- stringutil $
```



```
1 package stringutil
2
3 // Package stringutil contains utility functions for working with strings.
4 // Reverse returns its argument string reversed rune-wise left to right.
5
6 func Reverse(s string) string {
7     r := []rune(s)
8     for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
9         r[i], r[j] = r[j], r[i]
10    }
11    return string(r)
12}
13
14/*
15go build
16won't produce an output file.
17
18go install
19will place the package inside the pkg directory of the workspace.
20*/
```

Terminal

```
+ stringutil $ go build
+ stringutil $ go install
- stringutil $
```



02\_library

Search

darwin\_amd64

github.com

golang.org

google.golang.org

bradfitz

dustin

goestoeleven

golang

gorilla

julienschmidt

nu7hatch

rwcarlsen

stripe

build-w...-golang

GolangTraining

02\_library

stringutil.a

.DS\_Store

.gitignore

.localized

2015\_Entry...Radon.pdf

appengine-guestbook-go

Atom

course\_files

CreatingWebsites

DesignLibr...llustrator.log

favicon.ico

final\_202.pdf

fonts

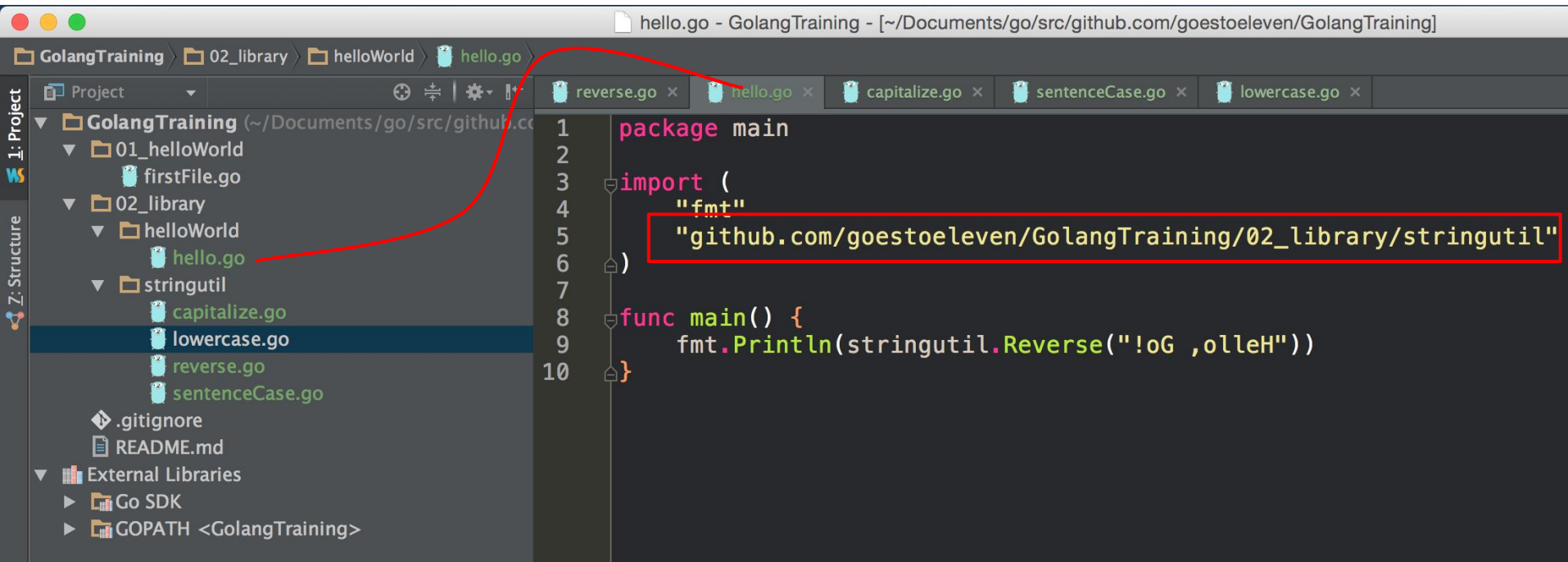
go

go books

goblueprints

**import**

Notice how we import a package ...



The screenshot shows an IDE window titled "hello.go - GolangTraining - [~/Documents/go/src/github.com/goestoeleven/GolangTraining]". The left sidebar displays the project structure:

- GolangTraining (~/.Documents/go/src/github.com/goestoeleven/GolangTraining)
  - 01\_helloWorld
    - firstFile.go
  - 02\_library
    - helloWorld
      - hello.go
    - stringutil
      - capitalize.go
      - lowercase.go
      - reverse.go
      - sentenceCase.go
  - .gitignore
  - README.md
  - External Libraries
    - Go SDK
    - GOPATH <GolangTraining>

The main editor shows the code in `hello.go`:

```
1 package main
2
3 import (
4     "fmt"
5     "github.com/goestoeleven/GolangTraining/02_library/stringutil"
6 )
7
8 func main() {
9     fmt.Println(stringutil.Reverse("!oG ,olleH"))
10 }
```

A red box highlights the import path `"github.com/goestoeleven/GolangTraining/02_library/stringutil"`. A red arrow points from the `hello.go` file in the project structure to the highlighted import statement.

Notice how we import a package ...

The screenshot shows an IDE window titled "hello.go - GolangTraining - [~/Documents/go/src/github.com/goestoeleven/GolangTraining]". The left sidebar displays the project structure:

- Project
  - GolangTraining (~/Documents/go/src/github.com/goestoeleven/GolangTraining)
    - 01\_helloWorld
      - firstFile.go
    - 02\_library
      - helloWorld
        - hello.go
      - stringutil
        - capitalize.go
        - lowercase.go
        - reverse.go
        - sentenceCase.go
    - .gitignore
    - README.md
    - External Libraries
      - Go SDK
      - GOPATH <GolangTraining>

The main editor shows the code in `hello.go`:

```
1 package main
2
3 import (
4     "fmt"
5     "github.com/goestoeleven/GolangTraining/02_library/stringutil"
6 )
7
8 func main() {
9     fmt.Println(stringutil.Reverse("!oG ,olleH"))
10 }
```

A red box highlights the import statement on line 5: `"github.com/goestoeleven/GolangTraining/02_library/stringutil"`. A red arrow points from this box to the `hello.go` file in the project structure. A white box highlights the usage of `stringutil.Reverse` on line 9: `stringutil.Reverse("!oG ,olleH")`. Below this box, the text "... and then use one of its public funcs." is displayed.



# Aliasing Imports

```
import (  
    "math/rand"  
    crand "crypto/rand"  
    "net/mail"  
    aemail "appengine/mail"  
)
```

# just for fun

```
bin — bash — 84x28
~ $ cd Documents/go/bin/
bin $ ls
01_helloWorld      gocode             golint
06_challenge_HTML  goimports          oracle
bin $ hexdump 01_helloWorld
00000000 cf fa ed fe 07 00 00 01 03 00 00 00 02 00 00 00
00000100 09 00 00 00 48 08 00 00 01 00 00 00 00 00 00 00
00000200 19 00 00 00 48 00 00 00 5f 5f 50 41 47 45 5a 45
00000300 52 4f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000400 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000500 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000600 00 00 00 00 00 00 00 00 00 19 00 00 00 28 02 00 00
00000700 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
00000800 00 10 00 00 00 00 00 00 00 00 70 14 00 00 00 00 00
00000900 00 00 00 00 00 00 00 00 00 00 70 14 00 00 00 00 00
00000a00 07 00 00 00 05 00 00 00 06 00 00 00 00 00 00 00 00
```

# Review

- `go build`
  - builds go code; if code is package main, it creates a binary executable and drops it in the package main's folder; if code is just a package, it builds it then throws away the binary.
- `go install`
  - builds and installs go code; if code is package main, it creates a binary executable and drops it in the workspace's bin folder; if code is a package, it builds it then drops it in the pkg folder (a file with a .a extension)
- `import`
  - import path is everything after the "src" folder in your workspace
  - use the last folder in the import path to reference the package in your code
  - you can alias packages in your imports

# Review Questions

# import

- Search for uuid at godoc.org
- use **go get** to get the code for **nu7hatch/gouuid**
- open up finder
- locate this code
  - take a screenshot of this
- Create a file that imports this code
- alias the package as just **uuid**
  - take a screenshot of this