



Road to Mercari

Gopher Dojo module 01

Summary: This document is the subject for the Gopher Dojo module 01 of the Road to Mercari @ 42 Tokyo.

Contents

I	Instructions	2
II	Foreword	3
III	Exercise 00 : cat	4
IV	Exercise 01 : Test	5

Chapter I

Instructions

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We **will not** take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet /`
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- If no other explicit information is displayed, you must assume the following versions of languages : `Go - 1.16.3`.


Chapter II

Foreword

<https://gopherdojo.org/>

Chapter III

Exercise 00 : cat

	Exercise 00
io.Reader and io.Writer	
Turn-in directory : <code>ex00/</code>	
Files to turn in : <code>go.mod</code> , <code>main.go</code>	
Allowed packages : <code>bufio</code> , <code>flag</code> , <code>io</code> , <code>os</code> , <code>strings</code>	

Find out `io.Reader` and `io.Writer`, and create a program called `ft_cat` which does the same thing as the system's `cat` command-line with the following specifications.

- You don't have to handle options.
- Create and use a function that takes `io.Reader` and `io.Writer` as arguments, one each.
- Use Go Modules.



Let's find out how `io.Reader` and `io.Writer` are used in the standard package.




Let's think about what the advantages of having `io.Reader` and `io.Writer`.



`man cat`

Chapter IV

Exercise 01 : Test

	Exercise 01
go test	
Turn-in directory : <i>ex01/</i>	
Files to turn in : <i>go.mod</i> , <i>main.go</i> , <i>imgconv/*</i> , <i>testdata/*</i>	
Allowed packages : self-created packages , standard packages , semi-standard packages	

Create tests for the image conversion command created in Exercise 00 of Road-to-Mercarie-Gopher-Dojo-00 that meets the following specifications.

- Refactoring to make it easier to test.
- The image conversion package should be separated from the main package.
- The test package should be separated from the image conversion package.
- Self-created packages should be placed in the *imgconv* directory.
- Test images should be placed in the *testdata* directory.
- The *ex01* directory structure should be as follows.

Example

```
?> ls -l .
total XX
-rw-r--r-- 1 XX XX      42 Aug 12 14:12 go.mod
drwxr-xr-x 6 XX XX    354 Aug 12 14:12 imgconv
-rw-r--r-- 1 XX XX      42 Aug 12 14:12 main.go
drwxr-xr-x 6 XX XX    354 Aug 12 14:12 testdata
```

- Create a user-defined type.
- The tests should be written in Go.
- The tests should be able to run in parallel.
- Measure test coverage.

- Use table-driven tests.
- Use test helper functions.
- Run the tests as follows.

Example

```
?> cd imgconv
?> go test -run ""
PASS
ok      imgconv 0.794s
```

- Use Go Modules.



Semi-standard packages are packages under golang.org/x