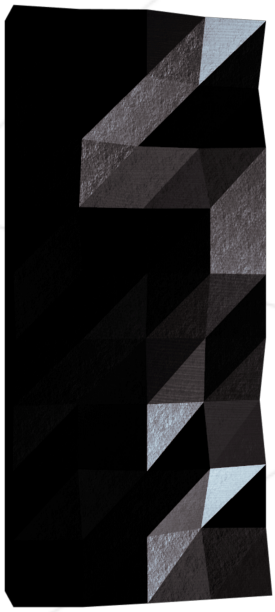

Go picine

Go 00

Summary: This document is the subject for the module Go 00 of the Go picine @ 42Tokyo.

Contents

I	Instructions	2
II	Foreword	3
III	Exercise 00 : Hello World	4
IV	Exercise 01 : Variables	5
V	Exercise 02 : isEmailAddress	6
VI	Exercise 03 : Create stairs	7
VII	Exercise 04 : Create function	8
VIII	Exercise 05 : a slice of ...	10
IX	Exercise 06 : echo!!!	12

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potencially change up to an hour before submission
- 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 grader by your fellow classmates.
- Assignments are also checked and grader by Go's official code formatter called-gofmt.

```
$ gofmt -d test.go
$
```

- Remember that if you run the above code and see the code change output, you will receive a score of 0 for not following the code rules.
- If it fails to compile, the evaluation is zero.
- **All package name is main.**
- **Using of builtin functions is allowed.**
- **Your Go version should be == 1.16.3.**
- Your reference guide is called **Google** / man /The Internet / ....
- Go is a language developed by Google, so ask **Google**!!
- If you research Gopher, you might get a clue.
- Evaluation will be done on 42's iMac(Guacamole).

golang.org

Chapter II

Foreword

> "Go is an open source programming language that makes it easy to build simple, reliable, and efficient software." (From the Go web site at golang.org)

Go was conceived in September 2007 by Robert Griesemer, Rob Pike,
and Ken Thompson, all at Google, and was announced in November 2009.

The goals of the language and its accompanying tools were to be expressive,
efficient in both compilation and execution,
and effective in writing reliable and robust programs.

Good luck.

Thank you for your help.

@42Tokyo

khiroshi
tayamamo
syudai
ryhosoka
ymiyata
dnakano
diuchi

Chapter III

Exercise 00 : Hello World

	Exercise 00
Hello World	
Turn-in directory : $ex00/$	
Files to turn in : `hello-world.go`	
Allowed packages : `fmt`	

- Write a program that points the text "Hello World!" to the console(stdout).

```
$ go run hello-world.go | cat -e
Hello World!$
$
```

Chapter IV

Exercise 01 : Variables

	Exercise 01
What kind of variables are there?	
Turn-in directory : $ex01/$	
Files to turn in : `vars.go`	
Allowed packages : `fmt`	

- Write a program that prints the same output as below to the console(stdout).

```
$ go run vars.go | cat -e
42 is a string.$
42 is a uint.$
42 is an int.$
42 is a uint8.$
42 is an int16.$
42 is a uint32.$
42 is an int64.$
false is a bool.$
42 is a float32.$
42 is a float64.$
(42+0i) is a complex64.$
(42+21i) is a complex128.$
{} is a main.FortyTwo.$
[42] is a [1]int.$
map[42:42] is a map[string]int.$
0x0 is an *int.$
[] is a []int.$
<nil> is a chan int.$
<nil> is a <nil>.$
$
```

Chapter V

Exercise 02 : isEmailAddress

	Exercise 02
isEmailAddress	
Turn-in directory : $ex02/$	
Files to turn in : `isEmailAddress.go`	
Allowed packages : `All`	

- Receive command line arguments more than zero.If zero, then

```
Invalid argument
```

  will be printed with a newline.

- If the string is a valid email address, then

```
<argument> is a valid email address.
```

  will be printed with a newline (of course, replacing the <argument>), otherwise

```
<argument> is NOT a valid email address.
```

  will be printed and again with a newline. (Replace <argument>!)

- We assume that a valid email address is a string that can be expressed using the following regex and is within 256 characters long or less.

  `"[\w\-\._]+@[\w\-\._]+\.[A-Za-z]+"`

- examples

```
$ go build isEmailAddress.go
$ ./isEmailAddress marvin@student.42tokyo.jp abc@def.123 | cat -e
marvin@student.42tokyo.jp is a valid email address.$
abc@def.123 is NOT a valid email address.$
```

```
$ ./isEmailAddress | cat -e
Invalid argument$
$
```

Chapter VI

Exercise 03 : Create stairs

	Exercise 03
Create stairsX	
Turn-in directory : $ex03/$	
Files to turn in : `createStairs.go`	
Allowed packages : `fmt, os, strconv`	

- Create a program with "*" to draw a staircase as shown in the example.
- The maximum number of times that "*" can be used is passed from the command line arguments.
- One valid command line argument will always be passed. There is no need to check the argument.
- The value given by the command line argument is an integer between 0 and 10000.
- Example

```
$ go build createStairs.go
$ ./createStairs 0 | cat -e
$ ./createStairs 1 | cat -e
$ *$
$ ./createStairs 4 | cat -e
$ *$
$ **$
$ ./createStairs 10 | cat -e
$ *$
$ **$
$ ***$
$ ****$
$ ./createStairs 12 | cat -e
$ *$
$ **$
$ ***$
$ ****$
$
```

Chapter VII

Exercise 04 : Create function

	Exercise 04
Create function	
Turn-in directory : $ex04/$	
Files to turn in : `calculation.go`	
Allowed packages : `fmt, os, strconv`	

- Create a function calculationStr that satisfies the following main function and standard output.

```
const ERROR_MSG string = "Arguments is invalid."

func main() {
    s, ok := calculationStr(os.Args)
    if !ok {
        fmt.Println(ERROR_MSG)
        os.Exit(1)
    }
    fmt.Print(s)
}
```

```
$ go build calculation.go
$ /calculation 12 4 | cat -e
sum: 16$
difference: 8$
product: 48$
quotient: 3$
$
```

```
$ ./calculation a 4 | cat -e
Arguments is invalid.$
$
```

Make sure that all error messages are output whenever an error is expected.

Chapter VIII

Exercise 05 : a slice of ...

	Exercise 05
a slice of ...	
Turn-in directory : *ex*05/	
Files to turn in : `subSlice.go`	
Allowed packages : `fmt`	

- Create a function subSlice takes a slice of int and 3 integer numbers (*begin*, *length* and *capacity*) and return a new slice of int.

```
func subSlice(slice []int, begin int, length int, capacity int) []int {
    [...]
}
```

- The elememts of new slice will be the portion of the original slice that starts at position *begin* and contains *length* of elememts. If there are not enough elements in the original slice, the rest will be filled with value 0.
- The capacity of the new slice will be desginated by *capacity*, but *length* will be used when *capacity* is smaller than *length*.
- Followings are a sample of main function and its standard output.

```
func main() {
    var orig = []int{0, 1, 2, 3, 4, 5}
    var ret []int

    ret = subSlice(orig, 0, 3, 3)
    fmt.Printf("ret = %v, len = %d, cap = %d\n", ret, len(ret), cap(ret))

    ret = subSlice(orig, 2, 7, 10)
    fmt.Printf("ret = %v, len = %d, cap = %d\n", ret, len(ret), cap(ret))
```

```
    ret = subSlice(orig, 2, 7, 3)
    fmt.Printf("ret = %v, len = %d, cap = %d\n", ret, len(ret), cap(ret))
}
```

```
$ go build subSlice.go
$ ./subSlice | cat -e
ret = [0 1 2], len = 3, cap = 3$
ret = [2 3 4 5 0 0 0], len = 7, cap = 10$
ret = [2 3 4 5 0 0 0], len = 7, cap = 7$
$
```

Chapter IX

Exercise 06 : echo!!!

	Exercise 06
echo!!!	
Turn-in directory : *ex*06/	
Files to turn in : `echo42.go`	
Allowed packages : `flag, fmt, strings`	

- Implement the echo42 command, which takes an option representing n and s. All output is standard output.
- The -n option omits the trailing newline.
- The -s option separates the output arguments by the contents of the string- passed after -s, not by spaces.

```
$ go build echo42.go
$ ./echo42 12 34 555 | cat -e
12 34 555$
$ ./echo42 -s / a bc def | cat -e
a/bc/def$
$ ./echo42 -s XXXXXXX 12 34 56789 | cat -e
12XXXXXXX34XXXXXXX56789$
$ ./echo42 -n 12 34 555 | cat -e
12 34 555
$ ./echo42 -help
Usage of ./echo42:
  -n    omit trailing newline
  -s string
        separator (default " ")
$ ./echo42 | cat -e
$ $
$
```

- Note that with the -n option, the last newline is omitted.

`flag??`