

Go Piscine Go 07

Summary: THIS document is the subject for the Go 07 module of the Go Piscine @ 42Tokyo.

Contents

1	Instructions	2
II	Exercise 00 : foreach	3
III	Exercise 01 : map	4
IV	Exercise 02: any	5
V	Exercise 03: countif	7
VI	Exercice 04: issorted	9
VII	Exercise 05 : doop	11
VIII	Exercise 06 : sortwordarr	13
\mathbf{IX}	Exercise 07: advancedsortwordarr	14

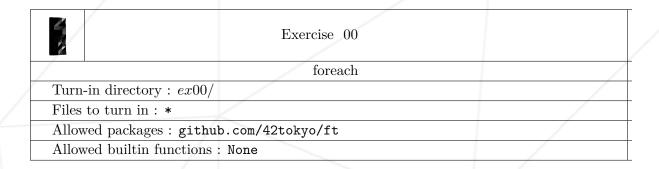
Chapter I

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 <u>cannot</u> leave <u>any</u> 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 use the latest versions of Go.

Chapter II

Exercise 00: foreach



Write a function For Each that, for an int slice, applies a function on each element of that slice.

• Expected function

```
func ForEach(f func(int), a []int) {
}
```

Usage

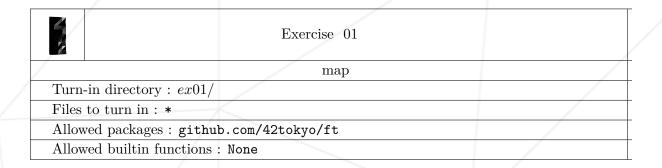
```
package main
import "piscine"

func main() {
    a := []int{1, 2, 3, 4, 5, 6}
    piscine.ForEach(piscine.PrintNbr, a)
}
```

```
$ go run .
123456
$
```

Chapter III

Exercise 01: map



Write a function Map that, for an int slice, applies a function of this type func(int) bool on each element of that slice and returns a slice of all the return values.

• Expected function

```
func Map(f func(int) bool, a []int) []bool {
}
```

Usage

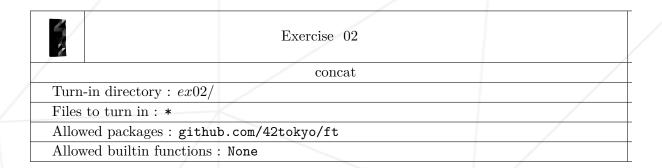
```
package main
import (
    "fmt"
    "piscine"
)

func main() {
    a := []int{1, 2, 3, 4, 5, 6}
    result := piscine.Map(piscine.IsPrime, a)
    fmt.Println(result)
}
```

```
$ go run .
[false true true false true false]
$
```

Chapter IV

Exercise 02: any



Write a function Any that returns true, for a string slice :, if when that string slice is passed through an f function, at least one element returns true.

• Expected function

```
func Any(f func(string) bool, a []string) bool {
}
```

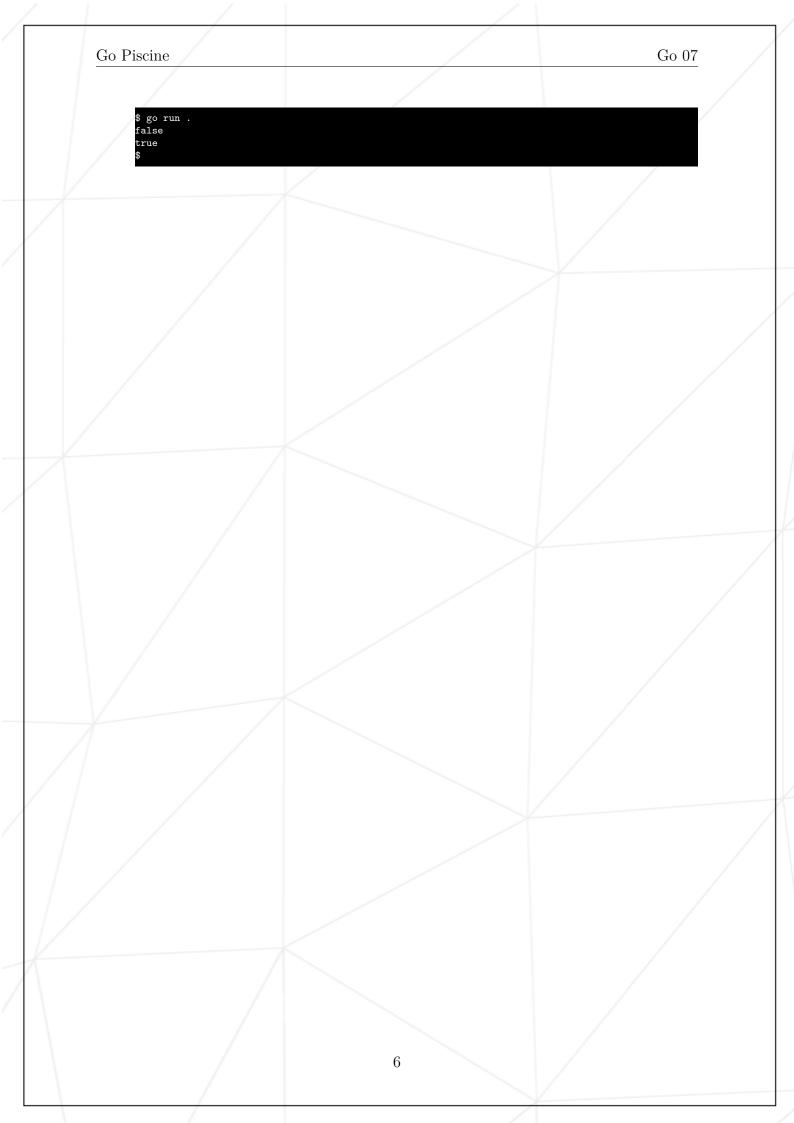
Usage

```
package main
import (
    "fmt"
    "piscine"
)

func main() {
    a1 := [] string{"Hello", "how", "are", "you"}
    a2 := [] string{"This", "is", "4", "you"}

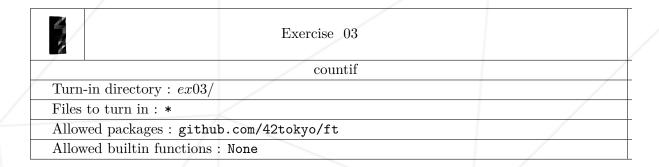
    result1 := piscine.Any(piscine.IsNumeric, a1)
    result2 := piscine.Any(piscine.IsNumeric, a2)

    fmt.Println(result1)
    fmt.Println(result2)
}
```



Chapter V

Exercise 03: countif



Write a function CountIf that returns, the number of elements of a string slice, for which the f function returns true.

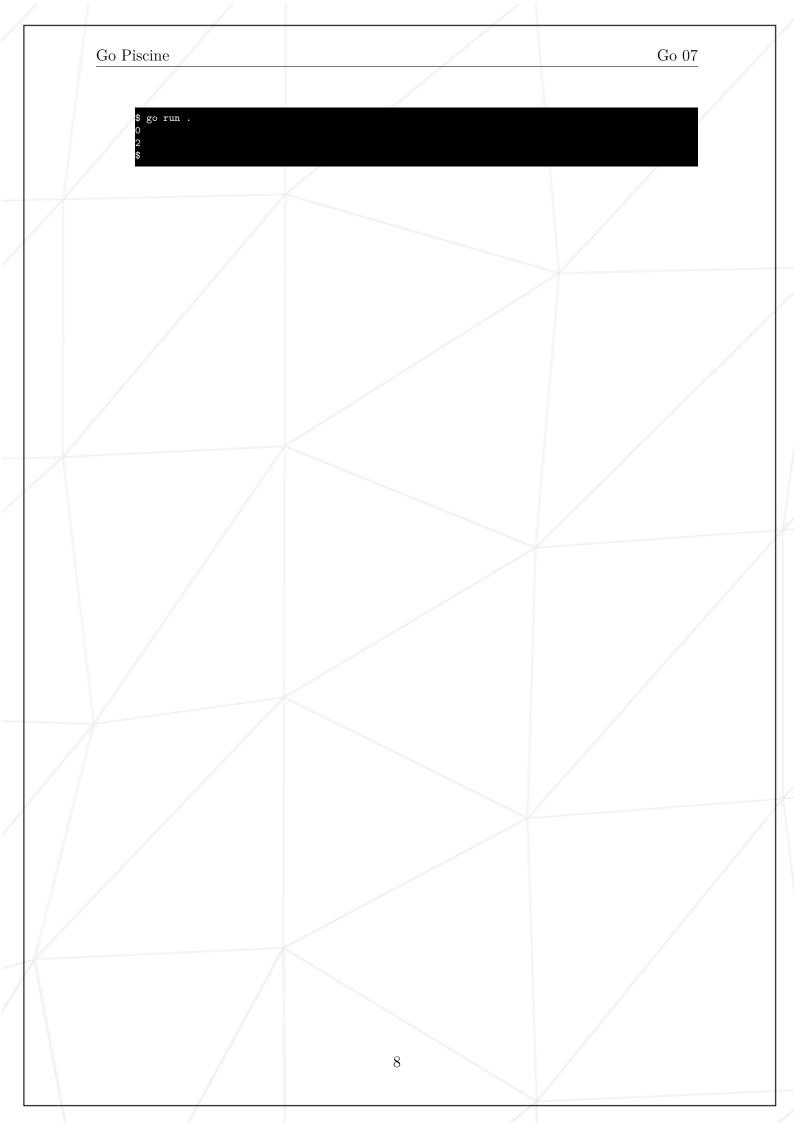
• Expected function

```
func CountIf(f func(string) bool, tab []string) int {
}
```

• Usage

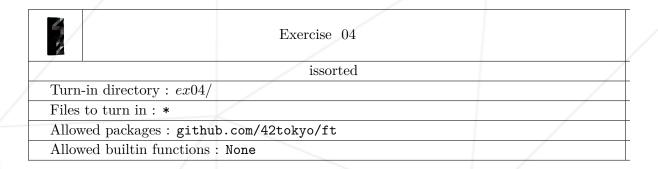
```
package main
import (
    "fmt"
    "piscine"
)

func main() {
    tab1 := []string{"Hello", "how", "are", "you"}
    tab2 := []string{"This","1", "is", "4", "you"}
    answer1 := piscine.CountIf(piscine.IsNumeric, tab1)
    answer2 := piscine.CountIf(piscine.IsNumeric, tab2)
    fmt.Println(answer1)
    fmt.Println(answer2)
}
```



Chapter VI

Exercice 04: issorted



Write a function IsSorted that returns true, if the slice of int is sorted, otherwise returns false.

- The function passed in the parameter returns a positive int if a (the first argument) is greater than to b (the second argument), it returns 0 if they are equal and it returns a negative int otherwise.
- To do your testing you have to write your own f function.
- Expected function

```
func IsSorted(f func(a, b int) int, a []int) bool {
}
```

- Usage
- Output of usage

```
$ go run .
true
false
```

Go Piscine Go 07

```
package main
import (
    "fmt"
    "piscine"
)

func main() {
    a1 := []int{0, 1, 2, 3, 4, 5}
    a2 := []int{0, 2, 1, 3}

    result1 := piscine.IsSorted(f, a1)
    result2 := piscine.IsSorted(f, a2)

    fmt.Println(result1)
    fmt.Println(result2)
}
```

Chapter VII

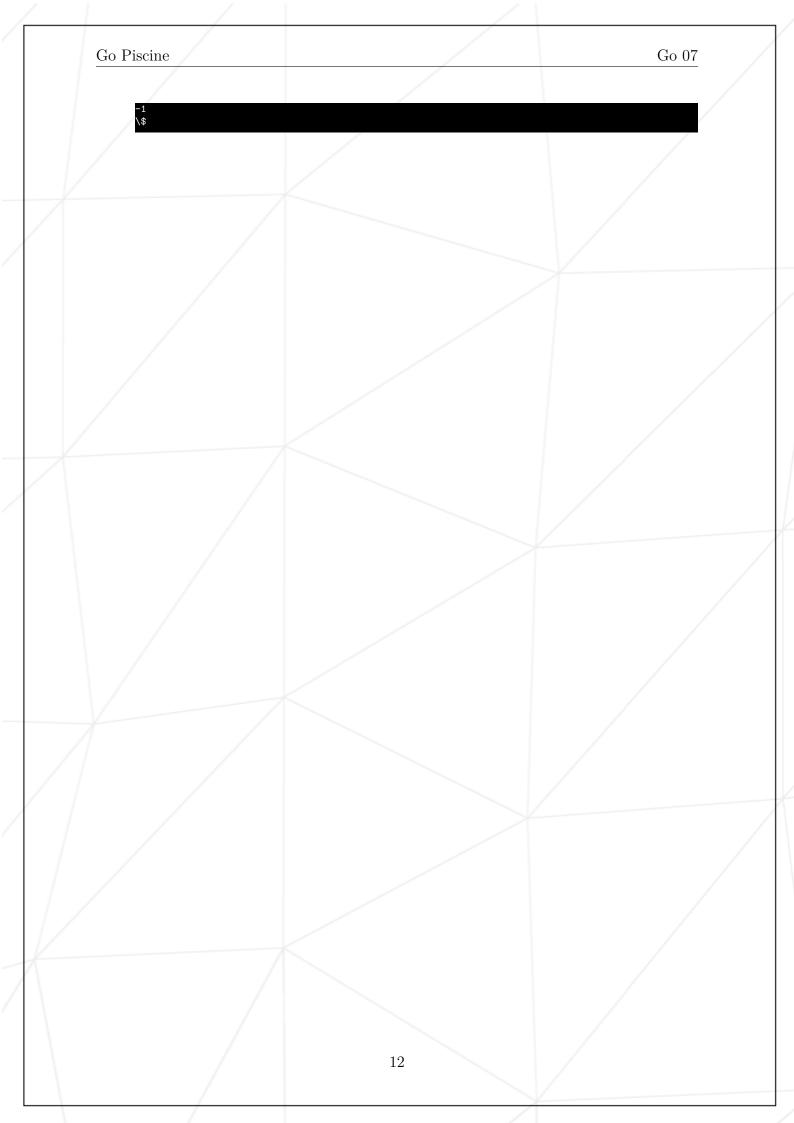
Exercise 05: doop

	Exercise 05	
/	doop	
Turn-in directory : $ex05/$		
Files to turn in: *		
Allowed packages: github.com/42tokyo/ft		
Allowed builtin functions : None		

Write a program that is called doop.

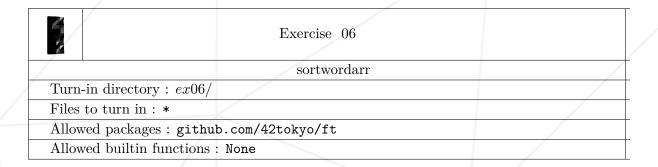
- The program has to be used with three arguments:
 - A value
 - \circ An operator, one of : +, -, /, *,
 - Another value
- In case of an invalid operator, value, number of arguments or an overflow, the programs prints nothing.
- The program has to handle the modulo and division operations by 0 as shown on the output examples below.
- Expected output

```
$ go run .
$ go run . 1 + 1 | cat -e
2$
$ go run . hello + 1
$ go run . 1 p 1
$ go run . 1 / 0 | cat -e
No division by 0$
$ go run . 1 \% 0 | cat -e
No modulo by 0$
$ go run . 9223372036854775807 + 1
$ go run . -9223372036854775809 - 3
$ go run . 9223372036854775807 "*" 3
$ go run . 9223372036854775807 "*" 3
$ go run . 1 "*" -1
```



Chapter VIII

Exercise 06: sortwordarr



Write a function SortWordArr that sorts by ascii (in ascending order) a string slice.

• Expected function

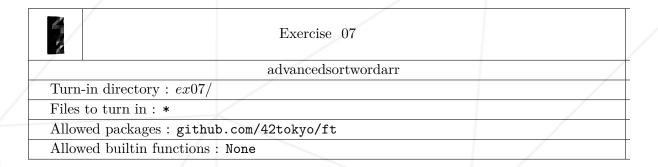
```
func SortWordArr(a []string) {
}
```

• Usage

```
$ go run .
[1 2 3 A B C a b c]
$
```

Chapter IX

Exercise 07: advancedsortwordarr



Write a function AdvancedSortWordArr that sorts a slice of string, based on the function f passed in parameter.

• Expected function

```
func AdvancedSortWordArr(a []string, f func(a, b string) int) {
}
```

• Usage

```
package main
import (
    "fmt"
    "piscine"
)
func main() {
    result := []string{"a", "A", "1", "b", "B", "2", "c", "C", "3"}
    piscine.AdvancedSortWordArr(result, piscine.Compare)
    fmt.Println(result)
}
```

```
$ go run .
[1 2 3 A B C a b c]
$
```