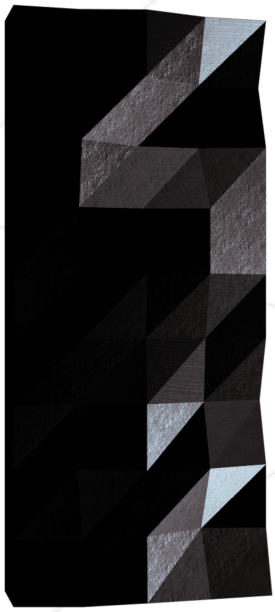

Go Piscine

Go 08

Summary: THIS document is the subject for the Go 08 module of the Go Piscine @ 42Tokyo.

Contents

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 use the latest versions of Go.

Chapter II

Exercise 00 : rot14

	Exercise 00
rot14	
Turn-in directory : $ex00/$	
Files to turn in : *	
Allowed packages : `github.com/42tokyo/ft`	
Allowed builtin functions : `None`	

Write a function `rot14` that returns the `string` within the parameter transformed into a `rot14 string`. Each letter will be replaced by the letter 14 spots ahead in the alphabetical order.

- 'z' becomes 'n' and 'Z' becomes 'N'. The case of the letter stays the same.
- Excepted function

```
func Rot14(s string) string {

}
```

- Usage

```
package main

import (
        "piscine"
        "github.com/42tokyo/ft"
)

func main() {
        result := piscine.Rot14("Hello! How are You?")

        for _, r := range result {
                ft.PrintRune(r)
        }
        ft.PrintRune('\n')
}
```

- And its output:

```
$ go run .
Vszzc! Vck ofs Mci?
$
```

Chapter III

Exercise 01 : Median

	Exercise 01
Median	
Turn-in directory : $ex01/$	
Files to turn in : *	
Allowed packages : None	
Allowed builtin functions : None	

Write a function that return the median of five `int` arguments.

- Excepted function

```
func Median(a, b, c, d, e int) int {

}
```

- Usage

```
package main

import (
        "piscine"
        "fmt"
)

func main() {
        median := piscine.Median(2, 3, 8, 5, 7)

        fmt.Println(median)
}
```

- And its output:

```
$ go run .
5
$
```

Chapter IV

Exercise 02 : comcheck

	Exercise 02
comcheck	
Turn-in directory : *ex*02/	
Files to turn in : *	
Allowed packages : github.com/42tokyo/ft	
Allowed builtin functions : None	

Write a program `comcheck` that displays on the standard output `Alert!!!` followed by newline (\n') if at least one of the arguments passed in parameter matches the `string`:

- `42`, `piscine` or `piscine 42`.
- If none of the parameters match, the program displays nothing.
- Usage

```
$ go run . "I" "Will" "Swim" "the" "piscine"
Alert!!!
$ go run . "piscine 42" "I" "am" "drowning"
Alert!!!
$
```

Chapter V

Exercise 03 : enigma

	Exercise 03
enigma	
Turn-in directory : $ex03/$	
Files to turn in : `*`	
Allowed packages : `None`	
Allowed builtin functions : `None`	

Write a function called `Enigma` that receives pointers as arguments and move its values around to hide them. This function will put:

- `a` into `c`
- `c` into `d`
- `d` into `b`
- `b` into `a`
- Excepted function

```
func Enigma(a ***int, b *int, c *******int, d ****int) {

}
```

- Usage

```
package main

import (
	"fmt"
	"piscine"
)

func main() {
	x := 5
	y := &x
	z := &y
	a := &z

	w := 2
	b := &w

	u := 7
	e := &u
	f := &e
	g := &f
	h := &g
	i := &h
	j := &i
	c := &j

	k := 6
	l := &k
	m := &l
	n := &m
	d := &n

	fmt.Println(***a)
	fmt.Println(*b)
	fmt.Println(*******c)
	fmt.Println(****d)

	piscine.Enigma(a, b, c, d)

	fmt.Println("After using Enigma")
	fmt.Println(***a)
	fmt.Println(*b)
	fmt.Println(*******c)
	fmt.Println(****d)

}
```

- And its output:

```
$ go run .
5
2
7
6
After using Enigma
2
6
5
7
$
```

Chapter VI

Exercise 04 : pilot

	Exercise 04
pilot	
Turn-in directory : $ex04/$	
Files to turn in : `*`	
Allowed packages : `None`	
Allowed builtin functions : `None`	

Fix the code.

- Create a directory called `pilot`.
- Inside the directory `pilot` create a file `main.go`
- Copy the code below to `main.go` and add the code needed so that the program compiles.
- Usage

```
package main

import "fmt"

func main() {
        var donnie Pilot
        donnie.Name = "Donnie"
        donnie.Life = 100.0
        donnie.Age = 24
        donnie.Aircraft = AIRCRAFT1

        fmt.Println(donnie)
}

const AIRCRAFT1 = 1
```

You can only add code, not delete

Chapter VII

Exercise 05 : Fix the Main

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

Fix the following program.

- Program to fix

```
package piscine

func PrintStr(s string) {
        for _, r := range s {
                ft.PrintRune(r)
        }
}

func CloseDoor(ptrDoor *Door) bool {
        PrintStr("Door Closing...")
        state = CLOSE
        return true
}

func IsDoorOpen(Door Door) {
        PrintStr("is the Door opened ?")
        return Door.state = OPEN
}

func IsDoorClose(ptrDoor *Door) bool {
        PrintStr("is the Door closed ?")
}

func main() {
        door := &Door{}

        OpenDoor(door)
        if IsDoorClose(door) {
                OpenDoor(door)
        }
        if IsDoorOpen(door) {
                CloseDoor(door)
        }
        if door.state == OPEN {
                CloseDoor(door)
        }
}
```

Chapter VIII

Exercise 06 : Compact

	Exercise 06
Compact	
Turn-in directory : $ex06/$	
Files to turn in : `*`	
Allowed packages : `None`	
Allowed builtin functions : `None`	

Write a function `Compact` that takes a pointer to a slice of `strings` as the argument. This function must:

- Return the number of elements with non-zero value.
- Compact, i.e., delete the elements with zero-values in the slice.
- Excepted function

```
func Compact(ptr *[]string) int {

}
```

- Usage

```
package main

import (
	"fmt"

	"piscine"
)

const N = 6

func main() {
	a := make([]string, N)
	a[0] = "a"
	a[2] = "b"
	a[4] = "c"

	for _, v := range a {
		fmt.Println(v)
	}

	fmt.Println("Size after compacting:", piscine.Compact(&a))

	for _, v := range a {
		fmt.Println(v)
	}
}
```

- And its output:

```
$ go run .
a

b

c

Size after compacting: 3
a
b
c
$
```

Chapter IX

Exercise 07 : activebits

	Exercise 07
activebits	
Turn-in directory : *ex*07/	
Files to turn in : *	
Allowed packages : None	
Allowed builtin functions : None	

Write a function, `ActiveBits`, that returns the number of active `bits` (bits with the value 1) in the binary representation of an integer number.

- Excepted function

```
func ActiveBits(n int) int {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        fmt.Println(piscine.ActiveBits(7))
}
```

- And its output:

```
$ go run .
3
$
```

Chapter X

Exercise 08 : max

	Exercise 08
max	
Turn-in directory : $ex08/$	
Files to turn in : *	
Allowed packages : None	
Allowed builtin functions : None	

Write a function `Max` that will return the maximum value in a slice of integers. If the slice is empty it will return 0.

- Excepted function

```
func Max(a []int) int {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        a := []int{23, 123, 1, 11, 55, 93}
        max := piscine.Max(a)
        fmt.Println(max)
}
```

- And its output:

```
$ go run .
123
$
```

Chapter XI

Exercise 09 : join

	Exercise 09
join	
Turn-in directory : *ex*09/	
Files to turn in : *	
Allowed packages : None	
Allowed builtin functions : None	

Write a function that returns the concatenation of all the strings of a slice of strings separated by the separator passed as the argument sep.

- Excepted function

```
func Join(strs []string, sep string) string {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        toConcat := []string{"Hello!", " How", " are", " you?"}
        fmt.Println(piscine.Join(toConcat, ":"))
}
```

- And its output:

```
$ go run .
Hello!: How: are: you?
$
```

Chapter XII

Exercise 10 : unmatch

	Exercise 10
unmatch	
Turn-in directory : $ex10/$	
Files to turn in : `*`	
Allowed packages : `None`	
Allowed builtin functions : `None`	

Write a function, `Unmatch`, that returns the element of the slice that does not have a correspondent pair.

- If all the number have a correspondent pair, it should return `-1`.
- Excepted function

```
func Unmatch(a []int) int {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        a := []int{1, 2, 3, 1, 2, 3, 4}
        unmatch := piscine.Unmatch(a)
        fmt.Println(unmatch)
}
```

- And its output:

```
$ go run .
4
$
```