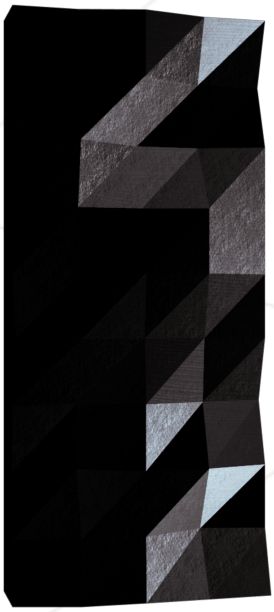

Go Piscine

Go 09

Summary: THIS document is the subject for the Go 09 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.
- Your turn-in directory for each exercise should look something like this:

```
ex[XX]
|-- main.go
|-- vendor
   |-- ft
      |-- printrune.go
   |-- piscine
      |-- [excercisename].go
```

Chapter II

Exercise 00 : rot14

	Exercise 00
rot14	
Turn-in directory : $ex00/$	
Files to turn in : *	
Allowed packages : None	
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.
- Expected function

```
func Rot14(s string) string {

}
```

- Usage

```
package main

import (
        "piscine"
        "ft"
)

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

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

- Output of usage

```
$ go mod init ex00
$ go run .
Vszzc! Vck ofs Mci?
$
```

Chapter III

Exercise 01 : abort

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

Write a function that returns the median of five int arguments.

- Expected function

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

        }
```

- Usage

```go
package main

import (
	"fmt"

	"piscine"
)

func main() {

	link := &piscine.List{}

	piscine.ListPushFront(link, "Hello")
	piscine.ListPushFront(link, "there")
	piscine.ListPushFront(link, "how are you")

	it := link.Head
	for it != nil {
		fmt.Print(it.Data, " ")
		it = it.Next
	}
	fmt.Println()
}
```

- Output of usage

```
$ go mod init ex01
$ go run .
5
$
```

Chapter IV

Exercise 02 : collatzcountdown

	Exercise 02
	collatzcountdown
Turn-in directory : *ex02/*	
Files to turn in : *	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function, CollatzCountdown, that returns the number of steps necessary to reach 1 using the collatz countdown.

- It must return -1 if start is equal to 0 or negative.
- Expected function

```go
func CollatzCountdown(start int) int {

        }
```

- Usage

```go
package main

import (
        "fmt"
        "piscine"
)

func main() {
        steps := piscine.CollatzCountdown(12)
        fmt.Println(steps)
}
```

- Output of usage

```
$ go mod init ex02
$ go run .
9
$
```

Chapter V

Exercise 03 : comcheck

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

- 01, galaxy or galaxy 01.
- If none of the parameters match, the program displays nothing.
- Example output

```
$ go mod init ex03
$ go run . "I" "Will" "Enter" "the" "galaxy"
Alert!!!
$ go run . "galaxy 01" "do" "you" "hear" "me"
Alert!!!
\$
```

Chapter VI

Exercise 04 : enigma

	Exercise 04
	enigma
Turn-in directory : $ex04/$	
Files to turn in : `*`	
Allowed packages : `fmt`	
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.
- Expected function

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

}
```

- Usage

```go
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)

}
```

- Output of usage

```
$ go mod init ex04
$ go run .
5
2
7
6
After using Enigma
2
6
5
7
$
```

Chapter VII

Exercice 05 : pilot

	Exercise 05
pilot	
Turn-in directory : $ex05/$	
Files to turn in : `*`	
Allowed packages : `None`	
Allowed builtin functions : `None`	

- 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.
- You can only copy code, not delete.

```go
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 : Fix the Main

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

- Fix the following program:

```go
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 IX

Exercise 07 : Compact

	Exercise 07
	Compact
Turn-in directory : $ex07/$	
Files to turn in : `*`	
Allowed packages : `fmt`	
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.
- Expected function

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

}
```

- Usage

```go
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)
	}
}
```

- Output of usage

```
$ go mod init ex07
$ go run .
a

b

c

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

Chapter X

Exercise 08 : activebits

	Exercise 08
	activebits
Turn-in directory : *ex08/*	
Files to turn in : *	
Allowed packages : `fmt`	
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.

- Expected function

```go
func ActiveBits(n int) int {

}
```

- Usage

```go
package main

import (
	"fmt"
	"piscine"
)

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

- Output of usage

```
$ go mod init ex08
$ go run .
3
$
```

Chapter XI

Exercise 09 : max

	Exercise 09
	max
Turn-in directory : *ex09/*	
Files to turn in : *	
Allowed packages : `fmt`	
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.

- Expected function

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

}
```

- Usage

```go
package main

import (
	"fmt"
	"piscine"
)

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

- Output of usage

```
$ go mod init ex09
$ go run .
123
$
```

Chapter XII

Exercise 10 : join

	Exercise 10
	join
Turn-in directory : $ex10/$	
Files to turn in : *	
Allowed packages : `fmt`	
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.

- Expected 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, ":"))
}
```

- Output of usage

```
$ go mod init ex10
$ go run .
Hello!: How: are: you?
$
```

Chapter XIII

Exercise 11 : unmatch

	Exercise 11
	unmatch
Turn-in directory : $ex11/$	
Files to turn in : *	
Allowed packages : `fmt`	
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.
- Expected 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)
}
```

- Output of usage

```
$ go mod init ex11
$ $ go run .
4
$
```