



Go Piscine

Go 06

*Summary: THIS document is the subject for the Go 06 module of the Go Piscine @ 42Tokyo.*

# Contents

<b>I</b>	<b>Instructions</b>	<b>2</b>
<b>II</b>	<b>Exercise 00 : boolean</b>	<b>3</b>
<b>III</b>	<b>Exercise 01 : point</b>	<b>5</b>
<b>IV</b>	<b>Exercise 02 : displayfile</b>	<b>6</b>
<b>V</b>	<b>Exercise 03 : cat</b>	<b>7</b>
<b>VI</b>	<b>Exercise 04 : ztail</b>	<b>8</b>

# 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
|   |-- [exercisename].go
```

# Chapter II

## Exercise 00 : boolean

	Exercise 00
boolean	
Turn-in directory : <i>ex00/</i>	
Files to turn in : *	
Allowed packages : <b>os</b>	
Allowed builtin functions : <b>None</b>	

- The code below must be copied into a file called `main.go`.
- The necessary changes must be applied for the program to work.
- Code to be copied

```
func printStr(s string) {
    for _, r := range s {
        ft.PrintRune(r)
    }
    ft.PrintRune('\n')
}

func isEven(nbr int) boolean {
    if even(nbr) == 1 {
        return yes
    } else {
        return no
    }
}


func main() {
    if isEven(lengthOfArg) == 1 {
        printStr(EvenMsg)
    } else {
        printStr(OddMsg)
    }
}
```

- Usage

```
❏ go mod init ex00
❏ go run . "not" "odd"
I have an even number of arguments
❏ go run . "not even"
I have an odd number of arguments
```

# Chapter III

## Exercise 01 : point

	Exercise 01
point	
Turn-in directory : <i>ex01/</i>	
Files to turn in : *	
Allowed packages : <code>fmt</code>	
Allowed builtin functions : <code>None</code>	

- The code below must be copied into a file called `main.go`.
- The necessary changes must be applied so that the program works.
- Code to be copied


```
func setPoint(ptr *point) {  
    ptr.x = 42  
    ptr.y = 21  
}  
  
func main() {  
    points := &point{}  
  
    setPoint(points)  
  
    fmt.Printf("x = %d, y = %d\n",points.x, points.y)  
}
```

- Usage

```
❏ go mod init ex01  
❏ go run .  
x = 42, y = 21  
❏
```

# Chapter IV

## Exercise 02 : displayfile

	Exercise 02
displayfile	
Turn-in directory : <i>ex02/</i>	
Files to turn in : *	
Allowed packages : <i>os</i>	
Allowed builtin functions : <i>None</i>	


Write a program that displays, on the standard output, the content of a file given as argument.

- Usage

```
❏ go mod init ex02
❏ go run .
File name missing
❏ echo "Ultimate Question of Life, the Universe, and Everything" > 42.txt
❏ go run . 42.txt main.go
Too many arguments
❏ go run . 42.txt
Ultimate Question of Life, the Universe, and Everything
❏
```

# Chapter V

## Exercise 03 : cat

	Exercise 03
cat	
Turn-in directory : <i>ex03/</i>	
Files to turn in : *	
Allowed packages : <b>os</b>	
Allowed builtin functions : <b>None</b>	

Write a program that behaves like a simplified cat command.


- The options do not have to be handled.
- If the program is called without arguments it should take the standard input (stdin) and print it back on the standard output (stdout).
- Usage

```
❏ echo "Born2Code" > quote.txt
❏ cat <<EOF> 42.txt
"Ultimate Question of Life, the Universe, and Everything"
EOF
❏ go mod init ex03
❏ go run . abc
ERROR: open abc: no such file or directory
exit status 1
❏ go run . quote.txt
Born2Code
❏ go run . quote.txt abc
Born2Code
ERROR: abc: No such file or directory
❏ cat quote.txt | ./cat
Born2Code
❏ go run .
Hello
Hello
^C
❏ go run . quote.txt alan_turing.txt
Born2Code
"Ultimate Question of Life, the Universe, and Everything"
❏
```



# Chapter VI

## Exercise 04 : ztail

	Exercise 04
ztail	
Turn-in directory : <i>ex04/</i>	
Files to turn in : *	
Allowed packages : <b>os</b>	
Allowed builtin functions : <b>make</b> , <b>append</b>	

Write a program that behaves like a simplified `tail` command that takes at least one file as an argument.

- The only option to be handled is `-c` and will be used in all the tests as the first argument, with positive values.
- Handle the errors by returning a non-zero exit status but process all the files.
- If several files are given, print a newline and the file name between each one of them (see below).
- Usage
- If `file1.txt` & `file2.txt` contains :

```
abcdefghijklmnopqrstuvwxyz
```

- Normal cases :

```
❯ go mod init ex04
❯ go run . -c 4 file1.txt
xyz
❯ go run . -c 4 file1.txt file2.txt
==> file1.txt <==
xyz

==> file2.txt <==
xyz
❯
```

- Error cases :

```
❯ go run . -c 4 file1.txt nonexistent1.txt file2.txt nonexistent2.txt
==> file1.txt <==
xyz
open nonexistent1.txt: no such file or directory

==> file2.txt <==
xyz
open nonexistent2.txt: no such file or directory
❯ echo ?
?
❯
```