



Go Piscine

Go 03

Summary: THIS document is the subject for the Go 03 module of the Go Piscine @ 42Tokyo.

Contents

| | | |
|-------|----------------------------|----|
| I | Instructions | 2 |
| II | Exercise 00 : first rune | 3 |
| III | Exercise 01 : nrune | 4 |
| IV | Exercise 02 : lastrune | 5 |
| V | Exercise 03 : index | 6 |
| VI | Exercise 04 : compare | 7 |
| VII | Exercise 05 : toupper | 8 |
| VIII | Exercise 06 : tolower | 9 |
| IX | Exercise 07 : capitalize | 10 |
| X | Exercise 08 : isalpha | 11 |
| XI | Exercise 09 : isnumeric | 13 |
| XII | Exercise 10 : isupper | 14 |
| XIII | Exercise 11 : islower | 15 |
| XIV | Exercise 12 : isprintable | 16 |
| XV | Exercise 13 : concat | 17 |
| XVI | Exercise 14 : basicjoin | 18 |
| XVII | Exercise 15 : join | 19 |
| XVIII | Exercise 16 : printnbrbase | 20 |
| XIX | Exercise 17 : atoi base | 22 |

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 : first rune

| | |
|---|-------------|
|  | Exercise 00 |
| first rune | |
| Turn-in directory : <i>ex00/</i> | |
| Files to turn in : * | |
| Allowed packages : None | |
| Allowed builtin functions : None | |

Write a function that returns the first rune of a string.

- Expected function

```
func FirstRune(s string) rune {  
}
```

- Usage


```
package main  
  
import (  
    "ft"  
    "piscine"  
)  
  
func main() {  
    ft.PrintRune(piscine.FirstRune("Hello!"))  
    ft.PrintRune(piscine.FirstRune("Salut!"))  
    ft.PrintRune(piscine.FirstRune("Ola!"))  
    ft.PrintRune('\n')  
}
```

- Output of usage

```
$ go mod init ex00  
$ go run .  
HS0  
$
```

Chapter III

Exercise 01 : nrune

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

Write a function that returns the nth rune of a string. If not possible, it returns 0.

- Expected function

```
func NRune(s string, n int) rune {  
}
```

- Usage


```
package main  
  
import (  
    "ft"  
    "piscine"  
)  
  
func main() {  
    ft.PrintRune(piscine.NRune("Hello!", 3))  
    ft.PrintRune(piscine.NRune("Salut!", 2))  
    ft.PrintRune(piscine.NRune("Bye!", -1))  
    ft.PrintRune(piscine.NRune("Bye!", 5))  
    ft.PrintRune(piscine.NRune("Ola!", 4))  
    ft.PrintRune('\n')  
}
```

- Output of usage

```
$ go mod init ex01  
$ go run .  
la!  
$
```

Chapter IV

Exercise 02 : lastrune

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

Write a function that returns the last rune of a string.

- Expected function

```
func LastRune(s string) rune {  
}
```

- Usage


```
package main  
  
import (  
    "ft"  
    "piscine"  
)  
  
func main() {  
    ft.PrintRune(piscine.LastRune("Hello!"))  
    ft.PrintRune(piscine.LastRune("Salut!"))  
    ft.PrintRune(piscine.LastRune("Ola!"))  
    ft.PrintRune('\n')  
}
```

- Output of usage

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

Chapter V

Exercice 03 : index

| | |
|---|-------------|
|  | Exercise 03 |
| index | |
| Turn-in directory : <i>ex03/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that behaves like the Index function.

- Expected function

```
func Index(s string, toFind string) int {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.Index("Hello!", "l"))  
    fmt.Println(piscine.Index("Salut!", "alu"))  
    fmt.Println(piscine.Index("Ola!", "h0l"))  
}
```

- Output of usage

```
$ go mod init ex03  
$ go run .  
2  
1  
-1  
$
```

Chapter VI

Exercise 04 : compare

| | |
|---|-------------|
|  | Exercise 04 |
| compare | |
| Turn-in directory : <i>ex04/</i> | |
| Files to turn in : * | |
| Allowed packages : <code>fmt</code> | |
| Allowed builtin functions : <code>None</code> | |

Write a function that behaves like the Compare function.

- Expected function

```
func Compare(a, b string) int {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.Compare("Hello!", "Hello!"))  
    fmt.Println(piscine.Compare("Salut!", "lut!"))  
    fmt.Println(piscine.Compare("Ola!", "Ol"))  
}
```

- Output of usage

```
$ go mod init ex04  
$ go run .  
0  
-1  
1  
$
```


Chapter VII

Exercise 05 : toupper

| | |
|---|-------------|
|  | Exercise 05 |
| toupper | |
| Turn-in directory : <i>ex05/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that capitalizes each letter in a string.

- Expected function

```
func ToUpper(s string) string {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.ToUpper("Hello! How are you?"))  
}
```

- Output of usage

```
$ go mod init ex05  
$ go run .  
HELLO! HOW ARE YOU?  
$
```

Chapter VIII

Exercise 06 : tolower

| | |
|---|-------------|
|  | Exercise 06 |
| tolower | |
| Turn-in directory : <i>ex06/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that lower cases for each letter in a string.

- Expected function

```
func ToLower(s string) string {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.ToLower("Hello! How are you?"))  
}
```

- Output of usage

```
$ go mod init ex06  
$ go run .  
hello! how are you?  
$
```

Chapter IX

Exercise 07 : capitalize

| | |
|---|-------------|
|  | Exercise 07 |
| capitalize | |
| Turn-in directory : <i>ex07/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that capitalizes the first letter of each word and lowercases the rest.

- A word is a sequence of alphanumeric characters.
- Expected function

```
func Capitalize(s string) string {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.Capitalize("Hello! How are you? How+are+things+4you?"))  
}
```

- Output of usage

```
$ go mod init ex07  
$ go run .  
Hello! How Are You? How+Are+Things+4you?  
$
```

Chapter X

Exercise 08 : isalpha

| | |
|---|-------------|
|  | Exercise 08 |
| isalpha | |
| Turn-in directory : <i>ex08/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that returns true if the string passed as the parameter only contains alphanumerical characters or is empty, otherwise, and returns false.

- Expected function

```
func IsAlpha(s string) bool {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.IsAlpha("Hello! How are you?"))  
    fmt.Println(piscine.IsAlpha("HelloHowareyou"))  
    fmt.Println(piscine.IsAlpha("What's this 4?"))  
    fmt.Println(piscine.IsAlpha("Whatsthis4"))  
}
```

- Output of usage

```
$ go mod init ex08
$ go run .
false
true
false
true
$
```

Chapter XI

Exercise 09 : isnumeric

| | |
|---|-------------|
|  | Exercise 09 |
| isnumeric | |
| Turn-in directory : <i>ex09/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that returns true if the string passed as a parameter contains only numerical characters or is empty, otherwise, returns false.

- Expected function

```
func IsNumeric(s string) bool {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.IsNumeric("010203"))  
    fmt.Println(piscine.IsNumeric("01,02,03"))  
}
```

- Output of usage

```
$ go mod init ex09  
$ go run .  
true  
false  
$
```

Chapter XII

Exercise 10 : isupper

| | |
|---|-------------|
|  | Exercise 10 |
| isupper | |
| Turn-in directory : <i>ex10/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that returns true if the string passed as parameter contains only uppercase characters or is empty, otherwise, returns false.

- Expected function

```
func IsUpper(s string) bool {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.IsUpper("HELLO"))  
    fmt.Println(piscine.IsUpper("HELLO!"))  
}
```

- Output of usage

```
$ go mod init ex10  
$ go run .  
true  
false  
$
```

Chapter XIII

Exercise 11 : islower

| | |
|---|-------------|
|  | Exercise 11 |
| islower | |
| Turn-in directory : <i>ex11/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that returns true if the string passed as the parameter contains only lowercase characters or is empty, otherwise, returns false.

- Expected function

```
func IsLower(s string) bool {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.IsLower("hello"))  
    fmt.Println(piscine.IsLower("hello!"))  
}
```

- Output of usage

```
$ go mod init ex11  
$ go run .  
true  
false  
$
```


Chapter XIV

Exercise 12 : isprintable

| | |
|---|-------------|
|  | Exercise 12 |
| isprintable | |
| Turn-in directory : <i>ex12/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that returns true if the string passed as the parameter contains only printable characters or is empty, otherwise returns false.

- Expected function

```
func IsPrintable(s string) bool {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.IsPrintable("Hello"))  
    fmt.Println(piscine.IsPrintable("Hello\n"))  
}
```

- Output of usage

```
$ go mod init ex12  
$ go run .  
true  
false  
$
```

Chapter XV

Exercise 13 : concat

| | |
|---|---|
|  | Exercise 13 |
| | concat |
| | Turn-in directory : <i>ex13/</i> |
| | Files to turn in : * |
| | Allowed packages : fmt |
| | Allowed builtin functions : None |

Write a function that returns the concatenation of two string passed in arguments.

- Expected function

```
func Concat(str1 string, str2 string) string {  
}
```

- Usage


```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.Concat("Hello!", " How are you?"))  
}
```

- Output of usage

```
$ go mod init ex13  
$ go run .  
Hello! How are you?  
$
```

Chapter XVI

Exercise 14 : basicjoin

| | |
|---|-------------|
|  | Exercise 14 |
| basicjoin | |
| Turn-in directory : <i>ex14/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that returns a concatenated string from the 'strings' passed as arguments.

- Expected function

```
func BasicJoin(elems []string) string {  
}
```

- Usage


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

- Output of usage

```
$ go mod init ex14  
$ go run .  
Hello! How are you?  
$
```

Chapter XVII

Exercise 15 : join

| | |
|---|-------------|
|  | Exercise 15 |
| join | |
| Turn-in directory : <i>ex15/</i> | |
| Files to turn in : * | |
| Allowed packages : fmt | |
| Allowed builtin functions : None | |

Write a function that returns a concatenated string from the 'strings' passed as arguments.

- 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 ex15  
$ go run .  
Hello!: How: are: you?  
$
```

Chapter XVIII

Exercise 16 : printnbrbase

| | |
|---|---|
|  | Exercise 16 |
| | printnbrbase |
| | Turn-in directory : <i>ex16/</i> |
| | Files to turn in : * |
| | Allowed packages : None |
| | Allowed builtin functions : None |

Write a function that prints an int in a string base passed as parameters.

- If the base is not valid, the function prints NV (Not Valid):
- Validity rules for a base :
 - A base must contain at least 2 characters.
 - Each character of a base must be unique.
 - A base should not contain + or - characters.
- The function has to manage negative numbers. (as shown in the example)
- Expected function

```
func PrintNbrBase(nbr int, base string) {  
}
```

- Usage
- Output of usage

```
$ go mod init ex16  
$ go run .  
125  
-1111101  
7D  
~uoi  
NV  
$
```

```
package main

import (
    "fmt"


    "github.com/42tokyo/ft"

    "piscine"
)

func main() {
    piscine.PrintNbrBase(125, "0123456789")
    ft.PrintRune('\n')
    piscine.PrintNbrBase(-125, "01")
    ft.PrintRune('\n')
    piscine.PrintNbrBase(125, "0123456789ABCDEF")
    ft.PrintRune('\n')
    piscine.PrintNbrBase(-125, "choumi")
    ft.PrintRune('\n')
    piscine.PrintNbrBase(125, "aa")
    ft.PrintRune('\n')
}
```

Chapter XIX

Exercise 17 : atoibase

| | |
|---|---|
|  | Exercise 17 |
| | atoibase |
| | Turn-in directory : <i>ex17/</i> |
| | Files to turn in : * |
| | Allowed packages : fmt |
| | Allowed builtin functions : None |

Write a function that does the following:

- The function that takes two arguments:
 - *s*: a numeric string in a given base.
 - *base*: a string representing all the different digits that can represent a numeric value.
- And returns the integer value of *s* in the given base.
- If the base is not valid it returns 0.
- Validity rules for a base :
 - A base must contain at least 2 characters.
 - Each character of a base must be unique.
 - A base should not contain + or - characters.
- String number must contain only elements that are in base.
- Only valid string numbers will be tested.
- The function does not have to manage negative numbers.
- Expected function
- Usage

```
func AtoiBase(s string, base string) int {  
}
```

```
package main  
  
import (  
    "fmt"  
    "piscine"  
)  
  
func main() {  
    fmt.Println(piscine.AtoiBase("125", "0123456789"))  
    fmt.Println(piscine.AtoiBase("1111101", "01"))  
    fmt.Println(piscine.AtoiBase("7D", "0123456789ABCDEF"))  
    fmt.Println(piscine.AtoiBase("uoi", "choumi"))  
    fmt.Println(piscine.AtoiBase("bbbbbab", "-ab"))  
}
```

- Output of usage

```
$ go mod init ex17  
$ go run .  
125  
125  
125  
125  
0  
$
```