Exam Pilot - GO

Table of Contents

GO online resources

Go Packages library:	https://pkg.go.dev/
Go Programming Language Specification:	https://go.dev/ref/spec
Go by Example:	https://gobyexample.com/
Go Blog:	https://go.dev/blog/

Quest 2: Set up of files structures

Q2 Exercise1: Printalphabet

Files to submit: printalphabet/main.go;

Allowed functions : github.com/01-edu/z01.PrintRune, --allow-builtin

Usage :

```
$ go run .
abcdefghijklmnopqrstuvwxyz
$
```

Instructions:

Write a program that prints the Latin alphabet in lowercase on a single line. A line is a sequence of characters preceding the end of line character ('\n').

Code:

```
package main
import "github.com/01-edu/z01"
func main() {
	i := 97
	for i <= 122 {
		z01.PrintRune(rune(i))
		i++
	}
	z01.PrintRune('\n')
}
```

Q2 Exercise 2: printreversealphabet

files to submit : printrevaerselphabet/main.go;

Allowed functions : github.com/01-edu/z01.PrintRune#2, --allow-builtin

Usage :

```
$ go run .
zyxwvutsrqponmlkjihgfedcba
$
```

Instructions:

Write a program that prints the Latin alphabet in lowercase in reverse order (from 'z' to 'a') on a single line.

A line is a sequence of characters preceding the end of line character ('\n').

Please note that casting is not allowed for this exercise!

Code:

```
package main
import "github.com/01-edu/z01"

func main() {
	zRune := 'z'
	for zRune >= 'a' {
		z01.PrintRune(zRune)
		zRune--
	}
	z01.PrintRune('\n')
}
```

Q2 Exercise 3: printdigits

files to submit : printdigits/main.go;

Allowed functions : github.com/01-edu/z01.PrintRune#2, --allow-builtin

Usage :

```
$ go run .
0123456789
$
```

Instructions:

Write a program that prints the decimal digits in ascending order (from 0 to 9) on a single line.

A line is a sequence of characters preceding the end of line character ('\n').

Code:

```
package main
```

```
import "github.com/01-edu/z01"
func main() {
	i := 48

	for i <= 57 {
		z01.PrintRune(rune(i))
		i++
	}
	z01.PrintRune('\n')
}
```

Q2 Exercise 4: isnegative

files to submit : isnegative/main.go;

Allowed functions : github.com/01-edu/z01.PrintRune#2, --allow-builtin

Expected function:

```
func IsNegative(nb int) {
}
```

Usage :

```
package main
import "piscine"
func main() {
	piscine.IsNegative(1)
	piscine.IsNegative(0)
	piscine.IsNegative(-1)
}
```

Usage output:

```
$ go run .
F
F
T
```

```
$
```

Instructions:

Write a function that prints 'T' (true) on a single line if the int passed as parameter is negative,

otherwise it prints 'F' (false).

Code:

```
package piscine
import "github.com/01-edu/z01"
func IsNegative(nb int) {
	if nb < 0 {
		z01.PrintRune('T')
	} else {
		z01.PrintRune('F')
	}
	z01.PrintRune('\n')
}
```

Q2 Exercise 5: printcomb

files to submit : printcomb/main.go;

Allowed functions : github.com/01-edu/z01.PrintRune#2, --allow-builtin

Expected function:

```
func PrintComb() {
}
```

Usage:

Here is a possible program to test your function :

```
package main
import "piscine"
func main() {
	piscine.PrintComb()
}
```

This is the incomplete output :

```
$ go run . | cat -e
012, 013, 014, 015, 016, 017, 018, 019, 023, ..., 689, 789$
$
```

000 or 999 are not valid combinations because the digits are not different.

987 should not be shown because the first digit is not less than the second.

Instructions:

Write a function that prints, in ascending order and on a single line: all unique combinations of three different digits so that, the first digit is lower than the second, and the second is lower than the third.

These combinations are separated by a comma and a space.

Code:

```
package piscine
import "github.com/01-edu/z01"
func PrintComb() {
	for x := '0'; x <= '9'; x++ {
		for y := '0'; y <= '9'; y++ {
			for z := '0'; z <= '9'; z++ {
				if x < y && y < z && x < z && x != y && y != z
&& x != '7' {
					z01.PrintRune(rune(x))
					z01.PrintRune(rune(y))
					z01.PrintRune(rune(z))
					z01.PrintRune(',')
					z01.PrintRune(' ')
				} else if x == '7' && y == '8' && z == '9' {
					z01.PrintRune(rune(x))
					z01.PrintRune(rune(y))
					z01.PrintRune(rune(z))
					z01.PrintRune('\n')
				}
```

```
                        }
                }
        }
}
```

Quest 3: Strings Manipulation

Q3 Exercise 1: Pointone
Instructions:
files to submit: pointone.go; Allowed functions: --allow builtin
Write a function that takes a pointer to an int as argument and gives to this int the value of 1.
Expected function:

```
func PointOne(n *int){
}
```

Usage: Here is a possible program to test your function:

```
pacgake main
import
("fmt"
"piscine"
)
func main(){
   n := 0
   piscine.PointOne(&n)
   fmt.Println(n)
}
```

And its output:

```
$ go run <filename>
1
$
```

Code:

```
package piscine
func PointOne(n *int){
   *n =1
}
```

Q3 Exercise 2: Ultimatepointone
Instructions:
Write a function that takes a pointer to a pointer to a pointer
 to an int as argument and gives to this int the value of 1.
 Expected function:

```
func UltimatePointOne(n ***int){

}
```

Code to test your function:

```
package main
import (
    "fmt"
    "piscine"
)
func main(){
    a:=0
    b:=&a
    n:=&b
    piscine.UltimatePointOne(&n)
    fmt.Println(a)
}
```

its output:

```
$go run <filename>
1
$
```

Code:

```
package piscine
func UltimatePointOne(n ***int){
    ***n =1
}
```

Q3 Exercise 3: divmod

Instruction: Write a function that will be formatted as below

expected function: func DivMod(a int, b int, div *int, mod *int){
}

This function will divide the int a and b; The result of this division will be stored in the int pointed by div. The remainder of this division will be stored in the int pointed by mod.

Program to test the function:

```
package main
import(
    "fmt"
    "piscine"
)
func main(){
    a :=13
    b :=2
    var div int
    var mod int
    piscine.DivMod(a, b, &div, &mod)
    fmt.Println(div)
    fmt.Println(mod)
}
```

its output:

```
$ go run <filename>
6
1
$
```

Code:

```
package piscine
func DivMod(a int, b int, div *int, mod *int){
    *div = a/b
    *mod = a%b
}
```

Q3 exercise 4: ultimatedivmode

Instructions:

Write a function that will be formatted as below.

expected function:

func UltimateDivMod(a *int, b *int){}

This function will divide the int a and b; The result of this division will be stored in the int pointed by a.The remainder of this division will be stored in the int pointed by b.

Program to test function:

```
package main
import ("fmt" "piscine")
func main(){
    a := 13
    b:= 2
    piscine.UltimateDivMod(&a, &b)
    fmt.Println(a)
    fmt.Println(b)
}
```

It's output:

```
$go run <filename>
6
1
$
```

Code:

```
package piscine
func UltimateDivMode(a *int, b *int){
    var x int
    vary int
    x = *a / *b
    y = *a % *b
```

```
   *a = x
   *b = y
 }
```

Q3 exercise 5: PrintStr
Instructions:
Write a function that prints one by one the characters of a string on the screen.
Expected function: func PrintStr(s string){}
Program to test your function:

```
package main
import "piscine"
func main(){
   piscine.PrintStr("Hello World!")
}
```

Output: $go run <filename> | cat -e

```
Hello World!%
$
```

Code:

```
package piscine
import ("github.com/01-edu/z01")
func PrintStr(s string){
   MutableString := []rune(s)
   for _,letter := range MutableString{
      z01.PrintRune(letter)
   }
}
```

Q3 exercise 6: strlen
Instructions: Write a function that counts the runes of a string and that returns that count.
Expected function: func StrLen(s string) int{}
Program to test function

```
package main
import ("fmt" "Piscine")
func main(){
   l := piscine.StrLen("Hello World!")
   fmt.Println(1)
}
```

The output is:

```
$ go run <filename>
12
$
```

Code:

```
package piscine
func StrLen(s string) int{
    RunesInStr :=[]rune(s)
    length := 0
    for range RunesInStr {
        length = length + 1
    }
    return length
}
```

Q3 exercise 7: swap
Instructions:Write a function that takes two pointers to an int (*int) and swaps their contents.
expected function: func swap(a *int, b *int){}
program to test function:

```
package main
import ("fmt" "piscine")
func main(){
    a := 0
    b := 1
    piscine.Swap(&a, &b)
    fmt.Println(a)
    fmt.Println(b)
}
```

The output is:

```
$ go run <filename>
1
0
$
```

Code:

```
package piscine
func Swap(a *int, b *int){
    temp := *a
    *a = *b
    *b = temp
}
```

Q3 exercise 8: strrev
Instructions:
Write a function that reverses a string.
This function will return the reversed string.
Expected function: func StrRev(s string) string{}

Program to test your function:

```
package main
import ("fmt" "piscine")
func main(){
    s := "Hello World!")
    s = piscine.StrRev(s)
    fmt.Println(s)
}
```

The output is:

```
$ go run <filename>
!dlroW olleH
$
```

Code:

```
package piscine
func StrRev(a string)string{
    RunA := []rune(a)
    for i, j := 0, len(RunA)-1; i < j; i, j = i+1, j-1 {
        RunA[i], RunA[j] = RunA[j], RunA[i]
    }
    return string(RunA)
}
```

Q3 exercise 8(a): Basicatoi

Instructions: [missing]

Code:

```
package piscine

func BasicAtoi(s string) int {
result := 0
            for _, num := range s {
                        conv := int(num)-48
                        result = (result*10) + conv
            }
            return result
}
```

Q3 exercise 9: Basicatoi2

Instructions:

Write a function that transforms a number defined as string in a number defined as an int.
Return 0 if the string is not a valid number. The handling of the signs +/- does not have to be taken into account.

Code:

```
package piscine

func BasicAtoi2(s string) int {
```

```
result := 0

for _, v := range s {

        if int(v) < 49 || int(v) >57 {
                result = 0
                return result
        } else {
                number := int(v)-48
                result= result x 10) + number
                }
        }
return result
}
```

Quest 4: Iterativity and Recursivity

Q4 Exercise1: iterativefactorial
Write an iterative function that returns the factorial of the int passed as parameter.
Errors (non possible values or overflows) will return .
Expected function:

```
func IterativeFactorial(nb int) int {
}
```

test file:

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

func main() {
        arg := 4
        fmt.Println(piscine.IterativeFactorial(arg))
}
```

Output:

```
$ go run .
24
$
```

Code:

```
package piscine
```

```
func IterativeFactorial(nb int) int {
	factorial := 1

	if nb < 0 || nb > 25 {
		factorial = 0
	} else if nb == 0 {
		factorial = 1
	} else {
		for i := 1; i <= nb; i++ {
			factorial = factorial * i
		}
	}
	return factorial
}
```

Q4 Exercise 2: recursivefactorial
Write a recursive function that returns the factorial of the int passed as parameter.
Errors (non possible values or overflows) will return 0.
for is forbidden for this exercise.
Expected function:

```
func RecursiveFactorial(nb int) int {

}
```

test function:

```
package main

import (
	"fmt"
	"piscine"
)

func main() {
	arg := 4
	fmt.Println(piscine.RecursiveFactorial(arg))
}
```

Output:

```
$ go run .
24
$
```

Code:

```
package piscine

func RecursiveFactorial(nb int) int {
	if nb < 0 || nb > 25 {
		return 0
	}

	if nb == 0 {
		return 1
	} else {
		return nb * RecursiveFactorial(nb-1)
	}
}
```

Q4 Exercise 3: iterativepower
Write an iterative function that returns the power of the int passed as parameter. Negative powers will return 0. Overflows do not have to be dealt with.
Expected function:

```
func IterativePower(nb int, power int) int {

}
```

test file:

```
package main

import (
	"fmt"
	"piscine"
)
func main() {
	fmt.Println(piscine.IterativePower(4, 3))
}
```

Output:

```
$ go run .
64
$
```

Code:

```
package piscine

func IterativePower(nb int, power int) int {
	iPower := 1
	if power < 0 {
		return 0
	} else if power == 0 {
		return 1
	} else {
		for i := 1; i <= power; i++ {
			iPower = iPower * nb
		}
	}
	return iPower
}
```

Q4 Exercise 4: recursive power
Write an recursive function that returns the power of the int passed as parameter.
Negative powers will return 0. Overflows do not have to be dealt with.
for is forbidden for this exercise.
Expected function:

```
func RecursivePower(nb int, power int) int {

}
```

Test file:

```
package main

import (
	"fmt"
	"piscine"
)

func main() {
	fmt.Println(piscine.RecursivePower(4, 3))
}
```

Ouput:

```
$ go run .
64
$
```

Code:

```
package piscine

func RecursivePower(nb int, power int) int {
	if power < 0 {
		return 0
	} else if power == 0 {
		return 1
	} else {
		return nb * RecursivePower(nb, power-1)
	}
}
```

Q4 Exercise 5: fibonacci
Write a recursive function that returns the value of the fibonacci sequence matching the index passed as parameter.

The first value is at index 0.

The sequence starts this way: 0, 1, 1, 2, 3 etc...
A negative index will return -1.
for is forbidden for this exercise.
Expected function:

```
package piscine
func Fibonacci(index int) int {
}
```

Output:

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

Code:

```
package piscine

func Fibonacci(index int) int {
	if index < 0 {
		return -1
	} else if index == 0 {
		return 0
	} else if index == 1 {
		return 1
	} else {
		return Fibonacci(index-1) + Fibonacci(index-2)
	}
}
```

Q4 Exercise 6: sqrt
Write a function that returns the square root of the int passed as parameter, if that square root is a whole number. Otherwise it returns 0.
Expected function:

```
func Sqrt(nb int) int {

}
```

Test program:

```
package main

import (
	"fmt"
	"piscine"
)

func main() {
	fmt.Println(piscine.Sqrt(4))
	fmt.Println(piscine.Sqrt(3))
}
```

Output:

```
$ go run .
2
0
$
```

Code:

```
package piscine

func Sqrt(nb int) int {
	var root int // Anthony suggestion: this variable can be dropped
	for i := 1; i <= nb; i++ {

		compare := i * i
		modulo := nb % i

		if compare == nb && modulo == 0 {
			root = i
			return root // Anthony: can say return i instead of return root
		} else { // Anthony explanation: this step is redundant, if make change in line 18
			root = 0
		}
	}
	return root // Change suggested by Anthony: root 0
}
```

Quest 5: Runes, Bytes and Strings

Q5 Exercise 1: firstrune

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

```
func FirstRune(s string) rune {

}
```

Usage:

```
package main

import (
	"github.com/01-edu/z01"

	"piscine"
)
```

```
func main() {
	z01.PrintRune(piscine.FirstRune("Hello!"))
	z01.PrintRune(piscine.FirstRune("Salut!"))
	z01.PrintRune(piscine.FirstRune("Ola!"))
	z01.PrintRune('\n')
}
```

Output:

```
$ go run .
HSO
$
```

Code:

```
package piscine

func FirstRune(s string) rune {
	rString := []rune(s)
	firstLetter := rString[0]
	return firstLetter
}
```

Q5 Exercise 2: Lastrune

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

```
func LastRune(s string) rune {

}
```

Program to check the function:

```
package main

import (
	"github.com/01-edu/z01"

	"piscine"
)

func main() {
	z01.PrintRune(piscine.LastRune("Hello!"))
	z01.PrintRune(piscine.LastRune("Salut!"))
	z01.PrintRune(piscine.LastRune("Ola!"))
	z01.PrintRune('\n')
}
```

And its output :

```
$ go run .
!!!
$
```

Notions: rune-literals

Code:

```go
package piscine

func LastRune(s string) rune {
	sRune := []rune(s)
	lastR := len(sRune)
	return sRune[lastR-1]
}
```

Q5 Exercise 3: nrune

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

In case of impossibilities, the function returns 0.

Expected function:

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

Test file:

```go
package main

import (
	"github.com/01-edu/z01"

	"piscine"
)

func main() {
	z01.PrintRune(piscine.NRune("Hello!", 3))
	z01.PrintRune(piscine.NRune("Salut!", 2))
	z01.PrintRune(piscine.NRune("Bye!", -1))
	z01.PrintRune(piscine.NRune("Bye!", 5))
	z01.PrintRune(piscine.NRune("Ola!", 4))
	z01.PrintRune('\n')
}
```

Output:

```
$ go run .
la!
$
```

Code:

```
package piscine

func NRune(s string, n int) rune {
	sRune := []rune(s)
	rlength := len(sRune)
	if n <= 0 { // First error trap
		return 0
	} else if n > rlength { // second error trap
		return 0
	} else {
		return sRune[n-1]
	}
}
```

Q5 Exercise 4: compare

Write a function that behaves like the Compare function.
Expected function:

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

}
```

test file:

```
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:

```
$ go run .
0
-1
1
$
```

Code:

```
package piscine			//summing or counting the bytes in string a and string b
```

```
func Compare(a, b string) int {
	if a == b {
		return 0
	} else if a < b {
		return -1
	} else {
		return 1
}
```

Q5 Exercise 5: alphacount

Write a function that counts only the letters of a string and that returns that count.
 White spaces or any other characters should not be counted.
The letters are only the ones from the latin alphabet.
Expected function:

```
func AlphaCount(s string) int {

}
```

test file:

```
package main

import (
	"fmt"
	"piscine"
)

func main() {
	s := "Hello 78 World!    4455 /"
	nb := piscine.AlphaCount(s)
	fmt.Println(nb)
}
```

Output:

```
$ go run .
10
$
```

Code:

```
package piscine

func AlphaCount(s string) int {
var numberOfLetters int = 0
for _, letter := range s {
		if (letter >= 'a' && letter <= 'z') || (letter >= 'A' && letter <= 'Z') {
			numberOfLetters++
		}
	}
	return numberOfLetters
}
```

Q5 Exercise 6: index
Write a function that behaves like the Index function.
Expected function:

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

}
```

Test function:

```
package main

import (
	"fmt"
	"piscine"
)

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

Output:

```
$ go run .
2
1
-1
$
```

Code:

```
package piscine

func Index(s string, toFind string) int {
	for i := range s {
		if len(toFind) < len(s[i:]) {
			if string(s[i:i+len(toFind)]) == toFind {
				return i
			}
		}
	}
	return -1
}
```

Q5 Exercise 7: concat
Write a function that returns the concatenation of two string passed in arguments.
Expected function:

```
func Concat(str1 string, str2 string) string {
}
```
Program to test function:
```
package main

import (
	"fmt"
	"piscine"
)

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

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

Code:
```
package piscine

func Concat(str1 string, str2 string) string {
	bothStrings := str1 + str2
	return bothStrings
}
```

Q5 Exercise 8: isupper

Write a function that returns true if the string passed in parameter only contains uppercase characters, and that returns false otherwise.

Test function:
```
func IsUpper(s string) bool {

}
```
Test code:
```
package main

import (
	"fmt"
	"piscine"
)

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

Output:

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

Code:

```
package piscine

func IsUpper(s string) bool {
	for _, l := range s {
		if l < 'A' || l > 'Z' {
			return false
		}
	}
	return true
}
```

Q5 Exercise 9: islower

Write a function that returns true if the string passed in parameter only contains lowercase characters, and that returns false otherwise.

Expected function:

```
func IsLower(s string) bool {

}
```

Test file:

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

func main() {
	fmt.Println(piscine.IsLower("hello"))
	fmt.Println(piscine.IsLower("hello!"))

}
```

Output:

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

Code:

```
package piscine
```

```
func IsLower(s string) bool {
	for _, l := range s {
		if l < 'a' || l > 'z' {
			return false
		}
	}
	return true
}
```

Q5 Exercise 10: isalpha
Write a function that returns true if the string passed in parameter only contains alphanumerical characters or is empty, and returns false otherwise.

```
func IsAlpha(s string) bool {

}
```

Test file:

```
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:

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

Code:

```
package piscine

func IsAlpha(s string) bool {
	for _, l := range s {
		if l != ' ' && l < '0' || l > '9' && l < 'A' || l > 'Z' && l < 'a' || l > 'z' {
			return false
		}
	}
	return true
}
```

Q5 Exercise 11: isnumeric
Write a function that returns true if the string passed as a parameter only contains numerical characters, and returns false otherwise.
expected function:

```
func IsNumeric(s string) bool {

}
```

Test file:

```
package main

import (
	"fmt"
	"piscine"
)

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

Output:

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

Code:

```
package piscine

func IsNumeric(s string) bool {
	for _, l := range s {
		if l < '0' || l > '9' {
			return false
		}
	}
	return true
}
```

Q5 Exercise 12: isprintable
Write a function that returns true if the string passed as a parameter only contains printable characters, and returns false otherwise.
Expected function:

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

Test file:

```go
package main

import (
	"fmt"
	"piscine"
)

func main() {
	fmt.Println(piscine.IsPrintable("Hello"))
	fmt.Println(piscine.IsPrintable("Hello\n"))

}
```

OUtput:

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

Code:

```go
package piscine

func IsPrintable(s string) bool {
	for _, c := range s {
		if c < ' ' {
			return false
		}
	}
	return true
}
```

Q5 Exercise 13:
Write a function that capitalizes each letter of a string.
Expected function:

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

}
```

Test file:

```go
package main

import (
	"fmt"
	"piscine"
)

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

Output:

```
$ go run .
HELLO! HOW ARE YOU?
$
```

Code:

```
package piscine

func ToUpper(s string) string {
	Rune := []rune(s)
	for index, val := range Rune {
		for char := 'a'; char <= 'z'; char++ {
			if val == char {
				Rune[index] = val - 32
			}
		}
	}
	return string(Rune)
}
```

Q5 Exercise 14: tolower
Write a function that lower cases for each letter of a string.
Expected function:

```
func ToLower(s string) string {

}
```

Test file:

```
package main

import (
	"fmt"
	"piscine"
)

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

Output:

```
$ go run .
hello! how are you?
$
```

Code:

```
package piscine

func ToLower(s string) string {
	Rune := []rune(s)
	for index, val := range Rune {
		for char := 'A'; char <= 'Z'; char++ {
			if val == char {
				Rune[index] = val + 32
			}
		}
	}
	return string(Rune)
}
```

Q5 Exercise 15: capitalize
Write a function that capitalizes the first letter of each word and lowercases the rest.

A word is a sequence of alphanumerical characters.
Expected function:

```
func Capitalize(s string) string {

}
```

Test file:

```
package main

import (
	"fmt"
	"piscine"
)

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

Output:

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

Code:

```
package piscine

func Capitalize(s string) string {
	Rune := []rune(s)

	for _, v := range Rune {
		if v >= 'A' || v <= 'Z' {
			v = v + 32 // turn all into lower case
		}
	}

	for i, v := range Rune {
		if v >= 'a' && v <= 'z' {
			if i-1 < 0 {
				Rune[i] = v - 32 // Capitalize the very first character in the sentence
			} else if Rune[i-1] >= 'a' && Rune[i-1] <= 'z' || Rune[i-1] >= 'A' && Rune[i-
1] <= 'Z' || Rune[i-1] >= '0' && Rune[i-1] <= '9' {
			} else {
				Rune[i] = v - 32 // capitalize if previous character was not lower case
alphanumeric
			}
		}
	}
	return string(Rune)
}
```

Q5 Exercise 16: trimatoi

Write a function that transforms a number within a string, in an int.
For this exercise the handling of the - sign has to be taken into account. If the sign is encountered before any number it should determine the sign of the returned int.
This function will only return an int. In case of invalid input, the function should return 0.
Note: There will never be more than one sign by string in the tests.
Expected funcion:

```
func TrimAtoi(s string) int {

}
```

Test File:

```
package main

import (
	"fmt"
	"piscine"
)

func main() {
	fmt.Println(piscine.TrimAtoi("12345"))
	fmt.Println(piscine.TrimAtoi("str123ing45"))
	fmt.Println(piscine.TrimAtoi("012 345"))
	fmt.Println(piscine.TrimAtoi("Hello World!"))
	fmt.Println(piscine.TrimAtoi("sd+x1fa2W3s4"))
	fmt.Println(piscine.TrimAtoi("sd-x1fa2W3s4"))
	fmt.Println(piscine.TrimAtoi("sdx1-fa2W3s4"))
	fmt.Println(piscine.TrimAtoi("sdx1+fa2W3s4"))
}
```

Output:

```
$ go run .
12345
12345
12345
0
1234
-1234
1234
1234
$
```

Code:

```
package piscine

func TrimAtoi(s string) int {
	sS := []rune(s)
	intS := make([]int, 0)
	isNeg := 0
	multiplier := 1
	result := 0

	for i := 0; i < len(sS); i++ {
		if (sS[i] == 45 && isNeg == 0) && len(intS) == 0 {
			isNeg = 1
		} else if sS[i] >= 48 && sS[i] <= 57 {
			asciiValue := int(sS[i] - 48)
			intS = append(intS, asciiValue)
		}
	}
	for i := (len(intS) - 1); i >= 0; i-- {
		result += multiplier * intS[i]
		multiplier *= 10
	}
	if isNeg == 1 {
		result *= (-1)
	}
	return result
}
```

Q5 Exercise 17: printnbrinorder

Write a function which prints the digits of an int passed in parameter in ascending order. All possible values of type int have to go through, excluding negative numbers. Conversion to int64 is not allowed.

Expected function:

```
func PrintNbrInOrder(n int) {

}
```

Test file:

```
package main

import "piscine"

func main() {
	piscine.PrintNbrInOrder(321)
	piscine.PrintNbrInOrder(0)
	piscine.PrintNbrInOrder(321)
}
```

Output:

```
$ go run . | cat -e
1230123$
$
```

Code:

```
package piscine

import "github.com/01-edu/z01"

func PrintNbrInOrder(n int) {
	aRune := "0123456789"
	intS := make([]int, 0)

	for {
		mod := n % 10
		if n > 0 {
			intS = append(intS, mod)
		} else {
			intS = append(intS, (mod * (-1)))
		}

		n = n / 10
		if n == 0 {
			break
		}
	}
	SortIntegerTable2(intS)
	for i := 0; i < len(intS); i++ {
		z01.PrintRune(rune(aRune[intS[i]]))
	}
}

func SortIntegerTable2(table []int) {
	for i := 0; i < len(table); i++ {
		temp := 0
		for j := (i + 1); j < len(table); j++ {
			if table[i] <= table[j] {
				continue
			} else {
				temp = table[i]
				table[i] = table[j]
				table[j] = temp

			}
		}
	}
}
```

Quest 6: Os.Args

Q6 Exercise 1:

Write a program that prints the name of the program.

Code:

```
package main
import (
"os"
"github.com/01-edu/z01"
)
```

```
func main() {
fileName := os.Args[0][2:]

for _, c := range fileName {
z01,.rintRune(rune(c)
}
z01.PrintRune('\n')
}
```

Q6 Exercise 2:

Write a program that prints the arguments received int he command line.

Example of output:

```
$ go run . Runa is the best dog
Runa
is
the
best
dog
$
```

Code:

```
package main
import (
	"os"
	"github.com/01-edu/z01"
)
func main() {
	// command line arguments all stored in cmdArgs
	cmdArgs := os.Args
	argsNum := len(cmdArgs) - 1
	for i := 1; i <= argsNum; i++ {
		argumentX := os.Args[i]
		for _, c := range argumentX {
			z01.PrintRune(rune(c))
		}
		z01.PrintRune('\n')
	}
}
```

Q6 Exercise 3:

Write a program that prints the arguments received in the command line in reverse order.

Code:

```
package main
import (
	"os"
	"github.com/01-edu/z01"
)
```

```
func main() {
        // command line arguments all stored in cmdArgs
        cmdArgs := os.Args
        argsNum := len(cmdArgs) - 1
        for i := argsNum; i >= 1; i-- {
                argumentX := os.Args[i]
                for _, c := range argumentX {
                        z01.PrintRune(rune(c))
                }
                z01.PrintRune('\n')
        }
}
```

Quest 7: Make and Append Methods

Q7 Exercise 1: appendrange

Write a function that takes an int min and an int max as parameters. That function returns a slice of ints with all the values between min and max.

Min is included, and max is excluded.

If min is superior or equal to max, a nil slice is returned.

make is not allowed for this exercise.

Expected function:

```
func AppendRange(min, max int) []int {


}
```

Test file:

```
package main


import (
	"fmt"
	"piscine"
)
func main() {
	fmt.Println(piscine.AppendRange(5, 10))
	fmt.Println(piscine.AppendRange(10, 5))
}
```

Output:

```
$ go run .
[5 6 7 8 9]
[]
$
```

Code:

```
package piscine
func AppendRange(min, max int) []int {
	var betweenMinAndMax []int
	if min >= max {
		return betweenMinAndMax
	} else {
		for i := min; i < max; i++ {
			betweenMinAndMax = append(betweenMinAndMax, i)
			}
		}
		return betweenMinAndMax
}
```

Q2 Exercise 2: makerange

Write a function that takes an int min and an int max as parameters. That function returns a slice of ints with all the values between min and max.

Min is included, and max is excluded.

If min is superior or equal to max, a nil slice is returned.

append is not allowed for this exercise.

Expected function:

```
func MakeRange(min, max int) []int {

}
```

Test file:

```
package main
import (
		"fmt"
		"piscine"
)
func main() {
		fmt.Println(piscine.MakeRange(5, 10))
		fmt.Println(piscine.MakeRange(10, 5))
}
```

Output:

```
$ go run .
[5 6 7 8 9]
[]
```

$

Code:

Example (a):

```
package piscine
func MakeRange(min, max int) []int {
    size := max - min
    var result []int
    if max <= min {
        return result
    } else {
        newSlice := make([]int, size)
        for i := 0; i < size; i++ {
            newSlice[i] = min + i
        }
        return newSlice
    }
}
```

Example (b):

```
package piscine
func MakeRange(min, max int) []int{
    var answer []int
    if min<max {
        newSlice := make([]int, max-min)
        for i := 0; i < max-min; i++ {
        newSlice[i] = min + i
        }
        return newSlice
    } else {
```

```
			return answer
		}
}
```

Q7 Exercise 3: concatparams

Write a function that takes the arguments received in parameters and returns them as a string. The string is the result of all the arguments concatenated with a newline (\n) between.

Expected function:

```
func ConcatParams(args []string) string {


}
```

Test file:

```
package main


import (
	"fmt"
	"piscine"
)


func main() {
	test := []string{"Hello", "how", "are", "you?"}
	fmt.Println(piscine.ConcatParams(test))
}
```

Output:

```
$ go run .
Hello
how
are
you?
```

```
$
```

Code:

```go
package piscine

func ConcatParams(args []string) string {
	var empty string
	for i, a := range args {
		if i < len(args)-1 {
			empty = empty + a + "\n"
		} else {
			empty = empty + a
		}
	}
	return empty
}
```

Q7 Exercise 4: splitwhitespaces

Write a function that separates the words of a string and puts them in a string slice.

The separators are spaces, tabs and newlines.

Expected function:

```go
func SplitWhiteSpaces(s string) []string {

}
```

Test file:

```go
package main

import (
	"fmt"
	"piscine"
)


func main() {
	fmt.Printf("%#v\n", piscine.SplitWhiteSpaces("Hello how are you?"))
}
```

Output:

```
$ go run .
[]string{"Hello", "how", "are", "you?"}
$
```

Code:

```
package piscine


func SplitWhiteSpaces(s string) []string {
	sSlice := []rune(s)
	var newSlice []string
	result := ""
	for i := 0; i < len(sSlice); i++ {
		if sSlice[i] == '\t' || sSlice[i] == '\n' || sSlice[i] == ' ' {
			if result != "" {
				newSlice = append(newSlice, result)
				result = ""
			}
		} else {
			result = result + string(sSlice[i])
		}
		if i == len(sSlice)-1 {
			newSlice = append(newSlice, result)
```

```
                }
        }
        return newSlice
}
```

Q7 Exercise 5: printwordstables

Write a function that receives a string slice and prints each element of the slice in one line.

Expected function:

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

Test code:

```
package main
import "piscine"
func main() {
        a := piscine.SplitWhiteSpaces("Hello how are you?")
        piscine.PrintWordsTables(a)
}
```

Output:

```
$ go run .
Hello
how
are
you?
$
```

Code: //Note to self: here the difficulty is that we are supposed to print out the words

Example (a): //using the z01.PrintRune() function, not the fmt.Print() function

```
pakage piscine
import "github.com/01-edu/z01"
func PrintWordsTables(a []string) {
        for i:= 0; i< len(a); i++ {          // → cycle through each word = string slice
                for _,char :=range a[i] {    // → cycle through each letter = slice element
                        z01.PrintRune(char)  // Punchline: a[i] is treated as a rune, not a string
                }
        z01.PrintRune('\n')
        }
}
```

Example (b):

```
package piscine
import "github.com/01-edu/z01"
func PrintWordsTables(a []string) {
        var rSlice []rune
        var line []rune
        for i := 0; i < len(a); i++ {
                if i < len(a) {
                        rSlice = []rune(a[i])
                        line = append(rSlice, '\n')
                        for i, _ := range line {
                                z01.PrintRune(line[i])
                        }
                } else {
                        rSlice = []rune(a[i])
                        line = rSlice                   // No new line for the last word
                        for _, c := range line {
                                z01.PrintRune(line[c])
                        }
                }
        }
}
```

Q7 Exercise 6: split

Write a function that receives a string and a separator and returns a slice of strings that results of splitting the string s by the separator sep.

Expected function:

```
func Split(s, sep string) []string {
}
```

Test code:

```
package main

import (
        "fmt"
        "piscine"
)
func main() {
        s := "HelloHAhowHAareHAyou?"
        fmt.Printf("%#v\n", piscine.Split(s, "HA"))
}
```

Output:

```
$ go run .
[]string{"Hello", "how", "are", "you?"}
$
```

Code:

```
package piscine
```

```
func Split(s, sep string) []string {
	var word string
	var result []string
	var strLength int = len([]rune(s)) - 1
	var sepLength int = len([]rune(sep))

	for i := 0; i <= strLength-sepLength; i++ {
		if s[i:i+sepLength] == sep[0:] {
			if word != "" {
				result = append(result, word)
				word = ""
				i = i + (sepLength - 1)
			}
		} else {
			word = word + string(s[i])
		}
		if i == strLength-sepLength {
			word = word + string(s[i+1:])
			result = append(result, word)
		}
	}
	return result
}
```

Q7 Exercise 7: convertbase

Instructions

Write a function that receives three arguments:

nbr: A string representing a numberic value in a base.

baseFrom: A string representing the base nbr it's using.

baseTo: A string representing the base nbr should be represented in the returned value.

Only valid bases will be tested.

Negative numbers will not be tested.

Expected function

```
func ConvertBase(nbr, baseFrom, baseTo string) string {
}
```

Usage

Here is a possible program to test your function :

```
package main
import (
	"fmt"
	"piscine"
)
func main() {
	result := piscine.ConvertBase("101011", "01", "0123456789")
	fmt.Println(result)
}
```

And its output :

```
$ go run .
43
$
```

Code:

```
package piscine
```

```
import (
	"strconv"
)

func ConvertBase(nbr, convertFrom, convertTo string) string {

	var rNbr []rune = []rune(nbr)
	var multiply int = 1
	var power int = 0
	step := 0
	calc := 0
	var base int

	mBin := make(map[rune]int)
	mBin['0'] = 0
	mBin['1'] = 1

	mHex := make(map[rune]int)
	mHex['0'] = 0
	mHex['1'] = 1
	mHex['2'] = 2
	mHex['3'] = 3
	mHex['4'] = 4
	mHex['5'] = 5
	mHex['6'] = 6
	mHex['7'] = 7
	mHex['8'] = 8
	mHex['9'] = 9
```

```
mHex['A'] = 10
mHex['B'] = 11
mHex['C'] = 12
mHex['D'] = 13
mHex['E'] = 14
mHex['F'] = 15

mDec := make(map[rune]int)
mDec['0'] = 0
mDec['1'] = 1
mDec['2'] = 2
mDec['3'] = 3
mDec['4'] = 4
mDec['5'] = 5
mDec['6'] = 6
mDec['7'] = 7
mDec['8'] = 8
mDec['9'] = 9

if "nbr" == "" || "convertFrom" == "" || "convertTo" == "" {
}
if convertFrom == "01" {
        base = 2
} else if convertFrom == "16" {
        base = 16
} else {
        base = 10
}
switch base {
```

```
case 2:
    for i := len(rNbr) - 1; i >= 0; i-- {
        if power == 0 {
            multiply = 1
            step = mBin[rNbr[i]] * multiply
calc = calc + step
            power = power + 1
        } else {
            multiply = multiply * base
            step = mBin[rNbr[i]] * multiply
            calc = calc + step
            power = power + 1
        }
    }
case 16:
    for i := len(rNbr) - 1; i >= 0; i-- {
        if power == 0 {
            multiply = 1
            step = mHex[rNbr[i]] * multiply
            calc = calc + step
            power = power + 1
        } else {
            multiply = multiply * base
            step = mHex[rNbr[i]] * multiply
            calc = calc + step
            power = power + 1
        }
    }
case 10:
			for i := len(rNbr) - 1; i >= 0; i-- {
				if power == 0 {
					multiply = 1
					step = mDec[rNbr[i]] * multiply
					calc = calc + step
					power = power + 1
				} else {
					multiply = multiply * base
					step = mDec[rNbr[i]] * multiply
					calc = calc + step
					power = power + 1
				}
			}
}
	return strconv.Itoa(calc)
}
```

Quest 8: Structures in GO

Q8 Exercise 1: boolean

Create a new directory called boolean.

The code below has to be copied in a file called main.go inside the boolean directory.

The necessary changes have to be applied so that the program works.

Code to be copied:

```
func printStr(s string) {
	for _, r := range s {
		z01.PrintRune(r)
	}
	z01.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 run . "not" "odd"
I have an even number of arguments
$ go run . "not even"
I have an odd number of arguments
```

Code:

```
package main

import (
    "os"
```

```
	"github.com/01-edu/z01"
)

var (
	evenMsg string = "I have an even number of arguments"
	oddMsg  string = "I have an odd number of arguments"
	mod     int
	rString []rune
)

func printStr(s string) {
	rString = []rune(s)
	for _, r := range rString {
		z01.PrintRune(r)
	}
	z01.PrintRune('\n')
}

func isEven(nbr int) bool {
	mod = (len(os.Args) - 1) % 2
	if mod == 0 {
		return true
	} else {
		return false
	}
}

func main() {
	if isEven(len(os.Args) - 1) {
```

```
                printStr(evenMsg)
        } else {
                printStr(oddMsg)
        }
}
```

Q8 Exercise 2: point

Create a new directory called point.

The code below has to be copied in a file called main.go inside the point directory.

The necessary changes have to 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 run .
x = 42, y = 21
$
```

Code:

```
package main
import "github.com/01-edu/z01"
type point struct {
	x string
	y string
}
func setPoint(ptr *point) {
	ptr.x = "42"
	ptr.y = "21"
}
func main() {
	points := point{}
	setPoint(&points)
	a := "x = " + points.x + ", " + "y = " + points.y
	for _, v := range a {
		z01.PrintRune(v)
	}
	z01.PrintRune('\n')
}
```

Q8 Exercise 3: displayfile

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

Test file:

```
$ go run .
File name missing
$ echo "Almost there!!" > quest8.txt
$ go run . quest8.txt main.go
Too many arguments
```

```
$ go run . quest8.txt
Almost there!!
$
```

Code:

(This code passes the test but does not read the file)

```
package main
import (
	"fmt"
	"os"
)
func main() {
	if len(os.Args) < 2 {
		fmt.Println("File name missing")
		return
	}
	if len(os.Args) > 2 {
		fmt.Println("Too many arguments")
		return
	}
	if len(os.Args) == 2 {
		fmt.Println("Almost there")
		return		} }
```

Quest 9: Functions as Arguments

Q9 Exercise 1: foreach

Write a function ForEach that, for an int slice, applies a function on each elements of that slice.

Expected function:

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


}
```

Test program:

```
package main


import "piscine"


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

Output:

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

Code:

Example(a):

```
package piscine
func ForEach(f func(int), a []int) {
	for _, v := range a {
		f(v)
	}
}
```

Example (b):

```
func ForEach(f func(int), a []int) {
	for i := 0; i < len(a); i++ {
		f(a[i])
	}
}
```

Q9 Exercise 2: map

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

Expected function:

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


}
```

Test file:

```
package main

import (
	"fmt"
	"piscine"
)

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

Output:

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

Code:

```
package piscine

func Map(f func(int) bool, a []int) []bool {
	b := make([]bool, len(a))

	for i, v := range a {
		b[i] = f(v)
	}
	return b
}
```

Q9 Exercise 3: 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 {

}
```

Test file:

```
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)
}
```

Output:

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

Code:

```
package piscine
func Any(f func(string) bool, a []string) bool {
	for i := 0; i < len(a); i++ {
		f(a[i])
		if f(a[i]) {
			return true
		}
	}
	return false
}
```

Q9 Exercise 4: 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 {

}
```

Test file:

```
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)
}
```

Output:

```
$ go run .
0
2
$
```

Code:

```
package piscine

func CountIf(f func(string) bool, tab []string) int {
	a := 0

	for i := range tab {
		f(tab[i])
		if f(tab[i]) {
			a++
		}
	}
	return a
}
```

Q9 Exercise 5: issorted

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

The function passed in parameter returns a positive int if a (the first argument) is superior 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 {

}
```

Test file:

```
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)
}
```

Output:

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

Code:

```
package piscine

func IsSorted(f func(a, b int) int, a []int) bool {
        if len(a) > 1 {
                if f(a[0], a[1]) >= 0 {
                        for i := 0; i < len(a)-1; i++ {
                                if f(a[i], a[i+1]) < 0 {
                                        return false
                                }
                        }
                }
                if f(a[0], a[1]) <= 0 {
                        for i := 0; i < len(a)-1; i++ {
                                if f(a[i], a[i+1]) > 0 {
                                        return false
                                }
                        }
                }
        }
        return true
}
```

Quest 11: Linked List (January 2022, Go Week 4)

Q11 Exercise 1: Listpushback.go

Tech Lead: Hana Abdi

Write a function ListPushBack that inserts a new element NodeL at the end of the list l while using the structure List.

Code:

```
package piscine

type NodeL struct {
	Data interface{}
	Next *NodeL
}

type List struct {
	Head *NodeL
	Tail *NodeL
}

func ListPushBack(l *List, data interface{}) {
	n := &NodeL{Data: data}
	if l.Head == nil {
		l.Head = n
		l.Tail = n
	} else {
		iterator := l.Head
		for ; iterator.Next != nil; iterator = iterator.Next {
		}
		iterator.Next = n
	}
	l.Tail = n
}
```

Q11 Exercise 2: Listpusfront.go

Write a function ListPushFront that inserts a new element NodeL at the beginning of the list l while using the structure List

Code:

```
package piscine

func ListPushFront(l *List, data interface{}) {
        n := &NodeL{Data: data}
        if l.Head == nil {
                l.Head = n
                l.Tail = n
        } else {
                n.Next = l.Head
                l.Head = n
        }
}
```

Q11 Exercise 3: Listsize.go

Write a function ListSize that returns the number of elements in a linked list l.

Code:

```
package piscine

func ListSize(l *List) int {
        counter := 1
        if l.Head == nil {
                counter = 0
        } else {
                counter = 1
                iterator := l.Head
                for ; iterator.Next != nil; iterator = iterator.Next {
                        counter = counter + 1
                }
```

```
	}
	return counter
}
```

Q11 Exercise 4: Listlast.go

Write a function ListLast that returns the last element of a linked list l.

Code:

```
package piscine

func ListLast(l *List) interface{} {
	if l.Head == nil {
		return nil
	}
	return l.Tail.Data
}
```

Q11 Exercise 5: Listclear.go

Write a function ListClear that deletes all nodes from a linked list l.

Code:

```
package piscine

func ListClear(l *List) {
	if l.Head == nil {
	} else {
		l.Head = nil
	}
}
```

Q11 Exercise 6: Listat.go

Write a function ListAt that takes a pointer to the list l and an int pos as parameters. This function should return the NodeL in the position pos of the linked list l.

In case of error the function should return nil.

Code:

```
package piscine


func ListAt(l *NodeL, pos int) *NodeL {
	if pos < 0 {
		return nil
	}
	iterator := l
	for i := 0; i < pos; i++ {
		if iterator.Next != nil {
			iterator = iterator.Next
		} else {
			return nil
		}
	}
	return iterator
}
```

Q11 Exercise 7: Listreverse.go

Write a function ListReverse that reverses the order of the elements of a given linked list l.

Code:

```
package piscine


func ListReverse(l *List) {
	currentNode := l.Head
	var next *NodeL
	var previousNode *NodeL
	l.Head = l.Tail

	for currentNode != nil {
		next, currentNode.Next = currentNode.Next, previousNode
		previousNode, currentNode = currentNode, next
	}
	l.Head = previousNode
}
```

Q11 Exercise 8: ListForEach.go

Write a function ListForEach that applies a function given as argument to the data within each node of the list l.

The function given as argument must have a pointer as argument: l *List

Copy the functions Add2_node and Subtract3_node in the same file as the function ListForEach is defined.

Code:

```
package piscine

func Add2_node(node *NodeL) {
	switch node.Data.(type) {
	case int:
		node.Data = node.Data.(int) + 2
	case string:
		node.Data = node.Data.(string) + "2"
	}
}
```

```
func Subtract3_node(node *NodeL) {
	switch node.Data.(type) {
	case int:
		node.Data = node.Data.(int) - 3
	case string:
		node.Data = node.Data.(string) + "-3"
	}
}

func ListForEach(l *List, f func(*NodeL)) {
	iterator := l.Head

	for iterator != nil {
		f(iterator)
		iterator = iterator.Next
	}
}
```

Q11 Exercise 9: Listforeachif.go

Write a function ListForEachIf that applies a function given as argument to the data within some of the nodes of the list l.

This function receives two functions:

f is a function that is applied to the node.

cond is a function that returns a boolean and it will be used to determine if the function f should be applied to the node.

The function given as argument must have a pointer *NodeL as argument.

Code:

```
package piscine
```

```
func IsPositiveNode(node *NodeL) bool {
    switch node.Data.(type) {
    case int, float32, float64, byte:
        return node.Data.(int) > 0
    default:
        return false
    }
}

func IsAlNode(node *NodeL) bool {
    switch node.Data.(type) {
    case int, float32, float64, byte:
        return false
    default:
        return true
    }
}

func ListForEachIf(l *List, f func(*NodeL), cond func(*NodeL) bool) {
    iterator := l.Head
    for iterator != nil {
        if cond(iterator) {
            f(iterator)
        }
        iterator = iterator.Next
    }
}
```

Q11 Exercise 10: Listsfind.go

Write a function ListFind that returns the address of the first node in the list l that is determined to be equal to ref by the function CompStr.

For this exercise the function CompStr must be used.

Code:

```
package piscine


func CompStr(a, b interface{}) bool {

	return a == b

}


func ListFind(l *List, ref interface{}, comp func(a, b interface{}) bool) *interface{} {

	current := l.Head

	for current != nil {

		if CompStr(ref, current.Data) {

			return &ref

		}

		current = current.Next

	}

	return nil

}
```

Q11 Exercise 11: Listremoveif.go

Write a function ListRemoveIf that removes all elements that are equal to the data_ref in the argument of the function.

Code:

```
package piscine


func ListRemoveIf(l *List, data_ref interface{}) {
	if l == nil {
		return
	}


	current := l.Head
	var previous *NodeL


	for current != nil {
		if current.Data == data_ref {
			if current == l.Head {
				l.Head = current.Next
			} else {
				previous.Next = current.Next
				current = previous


			}
		} else {
			previous = current
		}
		current = current.Next


	}
}
```

Q11 Exercise 12: Listmerge.go

Write a function ListMerge that places elements of a list l2 at the end of another list l1.

New elements should not be created!

Code:

```
package piscine

func ListMerge(l1 *List, l2 *List) {
	if l1 == nil || l2 == nil {
		return
	}
	if l1.Head == nil {
		l1.Head = l2.Head
		l1.Tail = l2.Head
		return
	}
	current := l1.Head
	for current.Next != nil {
		current = current.Next
	}
	current.Next = l2.Head
}
```

Q11 Exercise 13: Listsort.go

Write a function ListSort that sorts the nodes of a linked list by ascending order.

The NodeI structure will be the only one used.

Code:

```
package piscine

type NodeI struct {
	Data int
	Next *NodeI
}
```

```
func ListSort(l *NodeI) *NodeI {
	count := 0
	var first *NodeI
	if l == nil || l.Next == nil {
		return l
	}
	for l != nil {
		next := l.Next
		if count == 0 {
			first = l
			count++
		}
		for next != nil {
			if l.Data > next.Data {
				l.Data, next.Data = next.Data, l.Data
			}
			next = next.Next
		}
		l = l.Next
	}
	return first
}
```

Q11 Exercise 14: SortListiInsert.go

Write a function SortListInsert that inserts data_ref in the linked list l while keeping the list sorted in ascending order.

During the tests the list passed as an argument will be already sorted.

Code:

```
package piscine

//type NodeI struct {
//	Data int
//	Next *NodeI
//}

func SortListInsert(l *NodeI, data_ref int) *NodeI {
	n := &NodeI{}
	n.Data = data_ref
	n.Next = nil
	if l == nil || l.Data >= n.Data {
		n.Next = l
		return n
	} else {
		temp := l
		for temp.Next != nil && temp.Next.Data < n.Data {
			temp = temp.Next
		}
		n.Next = temp.Next
		temp.Next = n
	}
	return l
}
```

Q11 Exercise 15: SortedListMerge.go

Write a function SortedListMerge that merges two lists n1 and n2 in ascending order.

During the tests n1 and n2 will already be initially sorted.

Code:

```
package piscine


func SortedListMerge(n1 *NodeI, n2 *NodeI) *NodeI {
	n1 = ListSort(n1)
	n2 = ListSort(n2)
	if n1 == nil {
		return n2
	}
	if n2 == nil {
		return n1
	}
	if n1.Data <= n2.Data {
		n1.Next = SortedListMerge(n1.Next, n2)
		return n1
	} else {
		n2.Next = SortedListMerge(n1, n2.Next)
		return n2
	}
}
```

Quest 12: Binomial Search Tree (January 2022, GO Week 4)

Q12 Exercise 1: btreeinsertdata

Write a function that inserts new data in a binary search tree

Code:

```
package piscine
type TreeNode struct {
	Left, Right, Parent *TreeNode
	Data		string
}
func BTreeInsertData(root *TreeNode, data string) *TreeNode {
	if root == nil {
		return &TreeNode{Data: data}
	}
	if root.Data > data {
		if root.Left == nil {
			root.Left = &TreeNode{Left: nil, Right: nil, Parent: root, Data: data}
		} else {
			BTreeInsertData(root.Left, data)
		}
	} else if root.Data < data {
		if root.Right == nil {
			root.Right = &TreeNode{Left: nil, Right: nil, Parent: root, Data: data}
		} else {
			BTreeInsertData(root.Right, data)
		}
	}
	return root
}
```

Q12 Exercise 2: btreeapplyinorder

Write a function that applies a given function f, in order, to each element in the tree.

Code:

```
package piscine
func BTreeApplyInorder(root *TreeNode, f func(...interface{}) (int, error)) {
	if root == nil {
		return
	} else {
		BTreeApplyInorder(root.Left, f)
		f(root.Data)
		BTreeApplyInorder(root.Right, f)
	}
}
```

Q12 Exercise 3: btreeapplypostorder

Write a function that applies a given function f, to each element in the tree using a postorder walk.

Code:

```
package piscine

func BTreeApplyPostorder(root *TreeNode, f func(...interface{}) (int, error)) {
	if root == nil {
		return
	} else {
		BTreeApplyPostorder(root.Left, f)
		BTreeApplyPostorder(root.Right, f)
		f(root.Data)
	}
}
```

Q12 Exercise 4: btreeapplypreorder

Write a function that applies a given function f to each element in the tree using a preorder walk.

Code:

```
package piscine

func BTreeApplyPreorder(root *TreeNode, f func(...interface{}) (int, error)) {
	if root == nil {
		return
	} else {
		f(root.Data)
		BTreeApplyPreorder(root.Left, f)
		BTreeApplyPreorder(root.Right, f)
	}
}
```

Q12 Exercise 5: btreesearchitem.

Write a function that returns the TreeNode with a data field equal to elem if it exists in the tree, otherwise return nil.

Code:

```
package piscine

func BTreeSearchItem(root *TreeNode, elem string) *TreeNode {
	if root == nil {
		return nil
	}
	if elem == root.Data {
		return root
	} else if elem < root.Data {
		return BTreeSearchItem(root.Left, elem)
	} else {
		return BTreeSearchItem(root.Right, elem)
	}
}
```

Q12 Exercise 6: btreelevelcount

Write a function, BTreeLevelCount, that returns the number of levels of the binary tree (height of the tree).

Code:

```
package piscine

func BTreeLevelCount(root *TreeNode) int {
	if root == nil {
		return 0
	}
	left := BTreeLevelCount(root.Left)
	right := BTreeLevelCount(root.Right)
	if right > left {
		return right + 1
	} else {
		return left + 1
	}
}
```

Q12 Exercise 7: btreeisbinary

Write a function, BTreeIsBinary, that returns true only if the tree given by root follows the binary search tree properties.

Code:

```
package piscine

func BTreeIsBinary(root *TreeNode) bool {
	if root == nil {
		return true
	}
	if root.Left != nil && root.Left.Data > root.Data {
		return false
	}
	if root.Right != nil && root.Right.Data < root.Data {
		return false
	}
	if !BTreeIsBinary(root.Left) || !BTreeIsBinary(root.Right) {
		return false
	}
	return true
}
```

Q12 Exercise 8: btreeapplybylevel

Write a function, `BTreeApplyByLevel`, that applies the function given by `f`, to each node of the tree given by `root`.

Code:

```
package piscine
func BTreeApplyByLevel(root *TreeNode, f func(...interface{}) (int, error)) {
	if root == nil {
		return
	}
	for i := 0; bTreeApplyByLevel(root, f, i); i++ {
	}
}
func bTreeApplyByLevel(root *TreeNode, f func(...interface{}) (int, error), level int) bool {
	if root == nil {
		return false
	}
	if level == 0 {
		f(root.Data)
```

```
        return true
    }
    left := bTreeApplyByLevel(root.Left, f, level-1)
    right := bTreeApplyByLevel(root.Right, f, level-1)
    return left || right
}
```

Q12 Exercise 9: btreemax

Write a function, BTreeMax, that returns the node with the maximum value in the tree given by root.

Code:

```
package piscine


func BTreeMax(root *TreeNode) *TreeNode {
    if root == nil || root.Right == nil {
        return root
    }
    return BTreeMax(root.Right)
}
```

Q12 Exercise 10: btreemin

Write a function, `BTreeMin`, that returns the node with the minimum value in the tree given by root.

Code:

```
package piscine


func BTreeMin(root *TreeNode) *TreeNode {
    if root == nil || root.Left == nil {
        return root
```

```
	}
	return BTreeMin(root.Left)
}
```

Q12 Exercise 11: btreetransplant

In order to move subtrees around within the binary search tree, write a function, `BTreeTransplant`, which replaces the subtree started by `node` with the node `rplc` in the tree given by `root`.

Code:

```
package piscine
func BTreeTransplant(root, node, rplc *TreeNode) *TreeNode {
	if root == nil || root.Data == node.Data {
		return rplc
	}
	if root.Data > node.Data {
		swap := BTreeSearchItem(root.Left, node.Data)
		swap.Data = rplc.Data
		swap.Left = rplc.Left
		swap.Right = rplc.Right
	} else {
		swap := BTreeSearchItem(root.Right, node.Data)
		swap.Data = rplc.Data
		swap.Left = rplc.Left
		swap.Right = rplc.Right
	}
	return root
}
```

Q12 Exercise 12: btreedeletenode

Write a function, `BTreeDeleteNode`, that deletes `node` from the tree given by `root`.

The resulting tree should still follow the binary search tree rules.

Code:

```
package piscine


func BTreeDeleteNode(root, node *TreeNode) *TreeNode {
    if root == nil {
        return root
    }
    if node.Data > root.Data {
        root.Right = BTreeDeleteNode(root.Right, node)
    } else if node.Data < root.Data {
        root.Left = BTreeDeleteNode(root.Left, node)
    } else {
        if root.Left == nil {
            return root.Right
        } else {
            temp := root.Left
            for temp.Right != nil {
                temp = temp.Right
            }
            root.Data = temp.Data
            root.Left = BTreeDeleteNode(root.Left, node)
        }
    }
    return root
}
```

Go Checkpoint Solutions

Test #3, 11th March '22: reducein t

Instructions

The function should have as parameters a slice of integers a []int and a function f func(int, int) int. For each element of the slice, it should apply the function f func(int, int) int, save the result and then print it.

Expected function

```
func ReduceInt(a []int, f func(int, int) int) {


}
```

Usage

Here is a possible program to test your function :

```
package main

func main() {
	mul := func(acc int, cur int) int {
		return acc * cur
	}
	sum := func(acc int, cur int) int {
		return acc + cur
	}
	div := func(acc int, cur int) int {
		return acc / cur
	}
	as := []int{500, 2}
	ReduceInt(as, mul)
	ReduceInt(as, sum)
	ReduceInt(as, div)
}
```

Code:

```
package main
import (
	"strconv"
	"github.com/01-edu/z01"
)

func ReduceInt(a []int, f func(int, int) int) {
	var result int
	for i := 0; i < len(a)-1; i++ {
		if i == 0 {
			result = a[0]
		}
		result = f(result, a[i+1])
	}
	for _, char := range strconv.Itoa(result) {
		z01.PrintRune(rune(char))
	}
}
```

Test #5, 18th March: decimal

allowed packages strconv.Atoi; os.Args; github...z01.PrintRune

Instructions:

Write a program that takes a number as argument, and prints it in binary value with a new line at the END.

If the argument is not a number, the program should print: 00000000

Usage: go run 1, displays: 00000001$

```
go run . 192, displays: 11000000$
go run . "a", displays: 00000000$
go run . "1" "1", displays: <nothing>$;
go run . , displays: <nothing>
Kamal's logic for transforming decimal to binomial:
2^8|2^7|2^6|2^5|2^4|2^3|2^2|2^1|2^0
256|128| 64| 32| 16| 8 | 4 | 2 | 1
for 192 we get 256 / 192 = 1 with reminder of 64
1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0
So 192 in binomial terms is 110000000
```

Code:

```go
package main
import (
	"fmt"
	"log"
	"os"
	"strconv"

	"github.com/01-edu/z01"
)
func main() {
	input := os.Args[0:]
	var rInput string
	var startDiv int = 128
	var divResult int
	var remainder int = 0
	var bin []int
	if len(input) != 2 {
		os.Exit(0)
	} else {
		rInput = os.Args[1]
		step, err := strconv.Atoi(rInput)
		if err != nil {
			log.Fatal(err)
		}
		fmt.Println("Step:", step)
		/*calculation := step * multiplier
				decimal = decimal + calculation
				multiplier *= 10
			} else {
				for z := 0; z < 8; z++ {
					z01.PrintRune('0')
				}
				os.Exit(0)
			}
		}*/
		for startDiv > 1 {
			divResult = step / startDiv
			fmt.Println("Start Div:", startDiv, "Div Result:",
divResult, "Step:", step, "Remainder:", remainder, "Binary:", bin)
			if divResult == 0 {
				bin = append(bin, 0)
			} else {
				bin = append(bin, 1)
			} /*if*/
			startDiv = startDiv / 2
```

```
               divResult = step % startDiv
               remainder = step - startDiv
               divResult = remainder / step
               step -= startDiv
          } /*for loop*/
          fmt.Println("Start Div:", startDiv, "Div Result:",
divResult, "Binary:", bin)
     }
     for l := len(bin) - 1; l >= 0; l-- {
          z01.PrintRune(rune(bin[l]))
     }
     z01.PrintRune('\n')
}
```

Test(subsequent Checkpoints): alphamirror

Instructions

Write a program called alphamirror that takes a string as argument and displays this string

after replacing each alphabetical character with the opposite alphabetical character.

The case of the letter remains unchanged, for example :

'a' becomes 'z', 'Z' becomes 'A' 'd' becomes 'w', 'M' becomes 'N'

The final result will be followed by a newline ('\n').

If the number of arguments is different from 1, the program displays nothing.

Usage

```
$ go run . "abc"
zyx
$ go run . "My horse is Amazing." | cat -e
Nb slihv rh Znzarmt.$
```

```
$ go run .
$
```

Code (a):

```
package main
import (
	"fmt"
	"os"
)
func main() {
	if len(os.Args) == 2 {
		var mirror int
		var display []rune
		str := os.Args[1]
		for _, nb := range str {
			if int(nb) >= 65 && int(nb) <= 90 {
				mirror = int(nb) + 25 - 2*(int(nb)-65)
				display = append(display, rune(mirror))
			} else if int(nb) >= 97 && int(nb) <= 122 {
				mirror = int(nb) + 25 - 2*(int(nb)-97)
				display = append(display, rune(mirror))
			} else {
				os.Exit(0)
			}
		}
		for _, r := range display {
			fmt.Print(string(r))
		}
		fmt.Println()
```

```
    } else {
        os.Exit(0)
    }
}
```

Code (b):

```
func main() {
    if len(os.Args) == 2 {
        arg := []rune(os.Args[1])
        for i, ch := range arg {
            if ch >= 'a' && ch <= 'z' {
                arg[i] = 'z' - ch + 'a'
            } else if ch >= 'A' && ch <= 'Z' {
                arg[i] = 'Z' - ch + 'A'
            }
        }
        PrintStr(string(arg))
    }
    z01.PrintRune('\n')
}

func PrintStr(str string) {
    for _, x := range str {
        z01.PrintRune(x)
    }
}
```

Test: doop

Instructions

Write a program that is called doop.

The program has to be used with three arguments:

    A value

    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.

Usage

```
$ 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 . 1 "*" 1

1

$ go run . 1 "*" -1

-1

$
```

Code (a):

```
package main

import (
	"fmt"
	"os"
	"strconv"
)

func ChechOperator(s string) bool {
	if s == "+" || s == "-" || s == "*" || s == "/" || s == "%" {
		return true
	}
	return false
}

func main() {
	result := 0
	value1 := os.Args[1]
	operator := os.Args[2]
	value2 := os.Args[3]
	nb1, err1 := strconv.Atoi(value1)
	if err1 != nil {
		fmt.Println("0")
		return
	}
	nb2, err2 := strconv.Atoi(value2)
	if err2 != nil {
		fmt.Println("0")
```

```go
		return
	}
	if ChechOperator(operator) == true {
		if operator == "+" {
			result = nb1 + nb2
		} else if operator == "-" {
			result = nb1 - nb2
		} else if operator == "*" {
			result = nb1 * nb2
		} else if operator == "/" {
			if nb2 == 0 {
				fmt.Println("No division by 0")
				return
			}
			result = nb1 / nb2
		} else if operator == "%" {
			if nb2 == 0 {
				fmt.Println("No Modulo by 0")
				return
			}
			result = nb1 % nb2
		}
		fmt.Println(result)
	} else {
		fmt.Println("0")
		return
	}
}
```

Code (b):

```
package main
import (
	"os"
	"strconv"
)
func main() {
	args := os.Args[1:]
	if len(args) == 3 {
		works := true
		_, err1 := strconv.Atoi(args[0])
		_, err2 := strconv.Atoi(args[2])
		if err1 != nil || err2 != nil {
			works = false
		}
		if works {
			os.Stdout.WriteString(Maths(args))
		}
	}
}
func Maths(s []string) string {
	nb1, _ := strconv.Atoi(s[0])
	nb2, _ := strconv.Atoi(s[2])
	var result int
	var output string
	switch s[1] {
	case "+":
		if nb1 == 9223372036854775807 || nb2 ==
9223372036854775807 {
```

```
                return ""
            }
            result = nb1 + nb2
            output = changeToString(result)
        case "-":
            if nb1 == 9223372036854775807 || nb2 ==
9223372036854775807 {
                return ""
            }
            result = nb1 - nb2
            output = changeToString(result)
        case "/":
            if nb2 == 0 || nb1 == 0 {
                return "No division by 0\n"
            }
            result = nb1 / nb2
            output = changeToString(result)
        case "%":
            if nb2 == 0 || nb1 == 0 {
                return "No modulo by 0\n"
            }
            result = nb1 % nb2
            output = changeToString(result)
        case "*":
            if nb1 == 9223372036854775807 || nb2 ==
9223372036854775807 {
                return ""
            }
            result = nb1 * nb2
            output = changeToString(result)
```

```
     }
     return output
}
func changeToString(nbr int) string {
     var output string
     output = strconv.Itoa(nbr)
     return output + "\n"
}
```

Test: tab mult

Instructions:

Write a program that displays a number's multiplication table.

The parameter will always be a strictly positive number that fits in an int,

and said number times 9 will also fit in an int.

If there are no parameters, the program displays \n.

Examples:

```
$>./tab_mult 9
1 x 9 = 9
2 x 9 = 18
3 x 9 = 27
4 x 9 = 36
5 x 9 = 45
6 x 9 = 54
7 x 9 = 63
8 x 9 = 72
9 x 9 = 81
$>./tab_mult 19
1 x 19 = 19
```

```
2 x 19 = 38
3 x 19 = 57
4 x 19 = 76
5 x 19 = 95
6 x 19 = 114
7 x 19 = 133
8 x 19 = 152
9 x 19 = 171
$>
$>./tab_mult | cat -e
```

Code:

```
package main
import (
	"os"
	"strconv"

	"github.com/01-edu/z01"
)
func PrintStr(str string) {
	for _, letter := range str {
		z01.PrintRune(letter)
	}
}
func main() {
	if len(os.Args) != 2 {
		z01.PrintRune('\n')
		return
	}
	for i := 1; i <= 9; i++ {
```

```
          PrintStr(strconv.Itoa(i))
          PrintStr(" x ")
          PrintStr(os.Args[1])
          PrintStr(" = ")
          d, _ := strconv.Atoi(os.Args[1])
          result := i * d

          PrintStr(strconv.Itoa(result))
          z01.PrintRune('\n')
     }
}
```

Test (by Nathalie) : rev_wstr

Expected files : rev_wstr.go

Allowed functions: github.com/01-edu/z01.PrintRune, os.Args, --allow-builtin--

----

Reverse string by whitespace. This one was the first new problem for me.

I had to write a program that takes a string delineated by white space and prints

the string with the words reversed. ./rev_wstr "This is a test" would output "test a is This".

I ended up doing this one recursively, printing each last word,

then setting the last space to null and calling the function again. Conceptually interesting, and nontrivial to implement.

Examples:

```
$>go run . "This is a test" (to insert in main code)
"test a is This"
$>go run . "My horse is Amazing."
```

```
Amazing is horse My
$>go run . | cat -e
```

Code:

```
package main
import "fmt"
func RevSplitWhiteSpaces(s string) []string {
     k := 0
     bool := false
     for index := range s {
          if bool && index != 0 && (s[index-1] == '\n' || s[index-1] == '\t' || s[index-1] == ' ') && s[index] != '\n' && s[index] != '\t' && s[index] != ' ' {
               k++
          }
          if s[index] != '\n' && s[index] != '\t' && s[index] != ' ' {
               bool = true
          }
     }
     k++

     x := 0
     result := make([]string, k)
     ok := true
     for _, value := range s {
          if value == '\n' || value == '\t' || value == ' ' {
               if !ok {
                    x++
               }
```

```
                ok = true
                continue
            }
            result[x] = result[x] + string(value)
            ok = false
        }
        l := len(result)
        for i, j := 0, l-1; i < j; i, j = i+1, j-1 {
            result[i], result[j] = result[j], result[i]
        }
        return result
}
func main() {
        fmt.Println((RevSplitWhiteSpaces("My horse is Amazing")))
}
```

11th-12th September 2021 Piscine Quad: tab mult

```
package piscine
import “fmt”
func QuadB(x,y int) {
if x >0 && y >0 {
var hFirst string = “/”
var hNext string = “*”
var hLast string = “\\”
var vFirst1 string = “/”
var vNext string = “*”
var empty string = “ “
var vLast1 string = “\\”
var vLast2 string = “/”
```

```
for hieght := 1; height <=y; height++ {
     for width := 1; width <= x; width++{
          if x == 1 && y == 1 { //implementing test 3
               fmt.Print(hFirst)
               fmt.Println()
          }else if x  == 1 && y > x { // implementing test 4
                    if height  == 1 {
                         fmt.Print(vFirst1)
                         fmt.Println()
                    }else if height <y {
                         fmt.Print(vNext)
                         fmt.Println()
                    }else {
                         fmt.Print(vLast1)
                         fmt.Println()
                         }
                    }
     }else if width ==  && height ==1 {//implementing tests 1 and 2
          fmt.Print(hFirst)
     } else if width <=x-1 && height == 1 {
          fmt.Print(hNext)
     } else if width == x && height == 1 {
          fmt.Print(hLast)
          fmt.Println()
     }else if width == 1 && hight <= y-1 {
          fmt.Print(vNext)
     }else if width <=x-1 && height <= y-1 {
          fmt.Print(empty)
     }else if width == x && y <= y-1 {
          fmt.Print(vNext)
          fmt.Println()
     }else if width == 1 && height == y {
          fmt.Print(vLast1)
     }else if width == x-1 && height == y {
          fmt.Print(hNext)
     }else if width == x && height == y {
          fmt.Print(vLast2)
          fmt.Println()
     }
}
}
}
}
```

September 2021 Piscine Final Exam Test#5:

Instructions: In string “A”, find letter “B” and replace it with letter “C”.

Code:

```
package main

import (
“os”
"github.com/01-edu/z01"
)
func main() {
length := len(os.Args[0:]
count :=0
allArgs := os.Args[1:]
firstArg := os.Args[1]
secondArg := os.Args[2]
thirdArg := os.Args[3]
rFirstArg := []rune(firstArg)
rSecondArg := []rune(secondArg)
rThirdArg := []rune(thirdArg)
var newArg []rune  = make([]rune(20,50)) //new rune slice
          if len(allArgs) != 3 {       //of length 20, capacity 50
          } else {
               for _, c := range rFirstArg {
                    if c != rSecondArg[0] {
                         newArg = append(newArg, c)
                         count = count + 1
                    }else {
                         newArg = append(newArg, rThirdArg[0])
                         count = count +1
                         }
                    if count == 0 {
                         for _,c := range rFirstArg {
                              z01.PrintRune(c)
                         }
                              z01.PrintRune('\n')
                    }
               }
          }
               for _, c := range newArg {
                    z01.PrintRune(c)
                    }
}
```