Piscine Golang - Day01

Created @October 24, 2021 11:27 PM

요약 : 이 문서는 Golang의 Slice에 대한 내용을 포함하고 있습니다.

서브젝트 오류 문의 : bigpel66@icloud.com

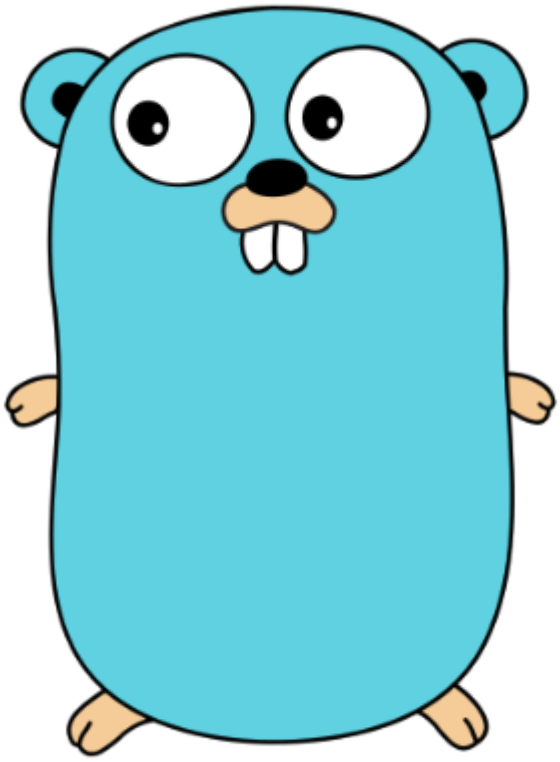
Contents

Chapter I - General Rules

- 모든 Exercise들은 Golang 1.17로 진행하며 테스트 코드와 함께 작성되어야 합니다. 작성된 테스트 코드들은 Exercise와 함께 제출되어야 합니다.
- 제출된 테스트 코드 내에 테스트 케이스는 반드시 존재해야 합니다.
- 적극적인 테스트 케이스는 권장되며, 특히 예외 케이스에 신경써야 합니다.
- 평가자는 자신이 생각했던 테스트 케이스가 존재하지 않는 경우, 피평가자에게 테스트 케이스를 요청할 수 있습니다. 이 때 피평가자의 코드가 동작하지 않으면 평가는 종료됩니다.
- 테스트 진행은 "testing" 패키지를 이용해야 하며, "testing"과 "github.com/stretchr/testify/assert" 둘 중 하나를 일관성 있게 사용해야 합니다.
- 테스트 진행 시 testing 객체를 이용한 벤치마킹 역시 적극 권장됩니다.
- Exercise의 제출은 동료 평가 및 Moulinette 채점으로 **PASS** 혹은 **FAIL**이 결정됩니다.
- Exercise들은 gofmt를 이용하여 포맷팅이 되어 있어야 합니다. 동료 평가 진행 시, 제출된 Exercise가 포맷팅이 되어 있지 않다면 **FAIL**입니다.
- Exercise들은 golangci-lint를 이용했을 때, 별다른 문제가 없다는 것이 검증되어야 합니다. golangci-lint에서 문제가 있는 부분이 발견되면 **FAIL**입니다.
- 동료 평가 진행을 위한 주석은 적극 허용됩니다. 각 함수 및 변수들에게 적용된 주석은 Golang에서 정식으로 지원하는 godoc을 이용하여 Document로 만들 수 있습니다.
- 컴파일은 **go build**를 이용하며, 이를 위해 **go mod**를 먼저 이해해야 합니다. **go mod**를 이용하지 않고 제출된 Exercise 역시 **FAIL**입니다.
- 제공된 서브젝트를 철저히 파악하여 Golang의 기초 지식을 모두 얻을 수 있도록 하십시오.
- 서브젝트에서 요구하는 내용들이 짧더라도, 이 내용들을 이해하고 예상한대로 작동되도록 시간을 쏟는 것은 굉장히 가치 있는 행동입니다. 다양한 시도를 권장합니다.
- *Go ? Ahead ! Write in Go.*

go mod를 이용하기 전에 GOPATH, GOROOT를 알아보십시오.

Chapter II - Preamble



Golang의 마스코트이다. 정확하게 정해진 이름은 없지만 통칭으로 Gopher라고 부른다.

우리말로 Gopher는 땅다람쥐(땅속 굴에 사는 북미산 동물)를 의미하기도 한다.

Golang을 만든 개발자들은 간단하고, 간결하고, 직관적인 언어를 지향하고자 했다. Golang은 공식적으로 C 계열로 분류하는데, 이는 어찌보면 당연한 것이 B 언어와 C 언어를 만들었던 Ken Thompson이 Golang을 만들었기 때문이다. 특히 현대 대부분의 언어는 C 언어를 기반으로 만들어졌기 때문에, Golang 개발진들은 기존의 형태를 아예 바꿔 새로운 패러다임을 내는 것이 어렵기도 하고 효용도 역시 적을 것이라고 생각했다. 이에 따라 C 언어를 기반으로 하되, 현존하는 프로그래밍 언어의 장점들을 의미있게 접목하는 쪽으로 방향을 잡았다.

Golang의 일부 문법은 Pascal에서 가져왔고, Golang의 중요한 특징인 동시성은 Rob Pike가 만든 Newsqueak의 영향을 적지 않게 받았다. 이외에도 Java, Python 등의 장점도 뽑아내어, 결과적으로 C 언어 문법에서 동시성을 지원하는 언어를 만들려는 Rob Pike의 오랜 염

원에 결과를 낼 수 있었다. 특히 C 언어의 키워드 수가 32개인 것에 비해, Golang이 갖는 키워드는 오로지 25개임을 감안하면 얼마나 적은 키워드를 두려고 노력했는지 엿 볼 수 있다.

여담으로 Golang에는 클래스, 상속, 생성자, final, 제네릭 (Golang 1.17 버전 기준)이 존재하지 않는다. 이 때문에 Golang이 OOP를 지원하지 않는 것이라고 생각할 수 있는데, Gopher(Golang을 이용하는 사람)들은 그렇게 생각하지 않는다. OOP는 클래스의 유무가 아니라 객체간 관계성을 만들 수 있냐는 것이 포인트인데, 그런 관점에서 Golang은 기존 언어들의 클래스와 상속이 갖는 문제점을 보완하기 위해 메서드와 인터페이스와 같은 새로운 패러다임을 만듦으로써 OOP가 가능하도록 만들어졌다.

이와 같은 멋진 기능들을 모두 모아둔 Awesome-Go라는 저장소도 있으니, 자주 탐색하여 원하는 기능들을 만들 수 있도록 해보자.

- https://awesome-go.com

Chapter III

Exercise 00 : Finding a smaller number than

Aa Name	Tags
Turn-in directory	ex00
Files to turn in	find/find.go, find/find_test.go main.go go.mod
Module name	ex00
Allowed packages	"fmt"

특정 수 x와 숫자들을 입력받아 x보다 작은 수들을 반환하는 함수를 만드십시오. 함수의 형태는 다음과 같습니다.

```
func Find(int, ...int) []int
```

첫번째 인자는 기준이 되는 x가 들어오며 두번째 인자부터 비교할 수들을 받습니다. 단, 해당 수들은 오름차순으로 정렬된 상태여야 합니다.

Chapter IV

Exercise 01 : deeeeepcopy

Aa Name	Tags
Turn-in directory	ex01
Files to turn in	deepcopy/deepcopy.go deepcopy_test.go go.mod
Module name	ex01
Allowed packages	None

yongckim은 원본 슬라이스의 일부분을 사용하고 싶어서 슬라이싱을 통해 복사본을 만드려고 합니다. 그런데 막상 슬라이싱을 통해 슬라이스를 만드니 만든 슬라이스의 값을 변경하면 원본의 값도 같이 변경이 문제가 발생했습니다.

```
func main() {
  nums := []int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  numSlice := nums[1:4]
  for i, value := range numSlice {
    numSlice[i] += value
  }
  fmt.Println(nums)
  fmt.Println(numSlice)
}
```

```
▸go run main.go
[0 2 4 6 4 5 6 7 8 9]
[2 4 6]
```

이런 상황이 왜 일어나는지 한 번 알아보세요

Go Slices: usage and internals

Andrew Gerrand 5 January 2011 Go's slice type provides a convenient and efficient means of working with sequences of typed data. Slices are analogous to arrays in other languages,

https://go.dev/blog/slices-intro

yongckim을 위해 이런 상황을 해결할 수 있는 슬라이스를 만들어 줍시다. 정의하는 함수는 다음과 같은 형태로 이뤄져 있습니다.

```
func Deepcopy([]int, int, int) ([]int, error)
```

Deepcopy의 매개변수는 원본 슬라이스, 시작 인덱스, 끝 인덱스를 받습니다.

Deepcopy는 슬라이싱한 슬라이스와 error를 반환합니다. 만약, 슬라이싱 도중 Error가 발생했다면 error가 갖는 내용을 반환해야 하고, 문제가 없다면 nil을 반환합니다.

> 💡 error 타입과 nil에 대해 알아보십시오.

error가 갖는 내용은 자유롭게 작성하되, 반드시 어떤 문제인지 알 수 있도록 작성해야 합니다.

해당 함수로 반환받은 슬라이스는 값을 변경했을 때 원본 슬라이스의 값이 변경 되어서는 안됩니다.

Chapter V

Exercise 02: How many palindrome strings can you make?

Aa Name	Tags
Turn-in directory	ex02
Files to turn in	palindrome.go, palindrome_test.go, main.go go.mod
Module name	ex02
Allowed packages	"fmt", "os", "unicode/utf8"

팰린드롬 문자열은 왼쪽에서 오른쪽으로 읽었을 때와 오른쪽에서 왼쪽으로 읽었을 때 똑같은 문자열을 의미합니다.

예를들어 level 이라는 문자열은 왼쪽부터 읽든, 오른쪽부터 읽든 똑같은 문자열이라서 팰린드롬 문자열이라고 할 수 있습니다.

문자열이 주어지면 오른쪽에 문자들을 추가할 수 있습니다. 이 때 가장 짧은 팰린드롬 문자열을 구하는 함수를 만드십시오. 함수의 원형은 다음과 같습니다.

```
Panlidrome(...string)
```

Palindrome 함수는 만들어진 문자열과 문자열의 길이를 출력합니다.

main 함수는 다음과 같이 Palindrome 함수를 테스트할 수 있어야 합니다.

```go
func main() {
    palindrom.Palindrom("foo", "level", "", "안녕")
}
```

```
▸go run main.go
foof 4
level 5
안녕안 3
```

main 함수는 다양한 케이스를 테스트 하도록 작성되어야 하며, 들어오는 문자열은 알파벳이 될 수도 있고 숫자와 공백이 들어올 수도 있으며, 심지어는 한글도 들어올 수 있습니다.

단, 빈문자열이 들어올 경우에는 아무것도 출력하지 않아야 합니다.

Chapter VI

exercise 03 : Let's make a hackathon team!

Aa Name	Tags
Turn-in directory	ex03
Files to turn in	hackathon/hackathon.go hackathon/hackathon_test.go main.go go.mod
Module name	ex03
Allowed packages	"fmt" "sort"

42Seoul에서 해커톤을 진행하려고 합니다.

해커톤 진행 전, 참가자는 팀이 되고 싶은 인원 한 명을 적어서 운영진에게 알려주어야 합니다. (자기 자신을 적는 것도 가능합니다.)

참가자는 이름대신 참가번호가 주어지며, 참가번호는 0번부터 시작합니다.

운영진은 전달받은 내용을 토대로 팀을 맺고 싶어하는 인원끼리 먼저 팀을 만들어 주려고 합니다.

팀이 구성되는 경우는 다음과 같습니다.

- 자기 자신을 지목한 경우 (자신 혼자 팀)
- 서로를 지목한 경우 (지목한 인원끼리 팀)
- 서로를 지목해 사이클을 그리는 경우

예를들어, 1, 2, 3이 있을 때 1번이 2번을 지목하고, 2번이 3번을 지목하고, 3번이 1번을 지목하는 경우 한 팀이 될 수 있습니다.

팀을 구성하는 데 실패한 경우에는 실패한 팀원들로 팀이 구성됩니다.

참여 인원이 지목한 인원에 대한 정보가 담긴 슬라이스가 매개변수로 주어지며 반환 값으로 팀이 된 번호들을 받는 다음과 같은 함수를 작성해봅시다.

```
func Match([]int) [][]int
```

예를들어, 다음과 같은 슬라이스가 매개변수로 주어질 경우

```
[2, 1, 3, 0, 5, 4, 0, 1]
```

0번 참가자는 2번 참가자를 가리키고, 2번 참가자는 3번 참가자를 가리키고, 3번 참가자는 0번 참가자를 가리키기 때문에 한 팀을 이룰 수 있습니다.

1번 참가자는 자기 자신을 지목하기 때문에 팀을 이룰 수 있습니다.

4번 참가자는 5번 참가자와 서로 지목하고 있기 때문에 팀을 이룰 수 있습니다.

6번 참가자와 7번참가자는 지목한 대상과 연결되지 않기 때문에 팀 매칭에 실패하고 두 인원으로 팀이 새로 만들어지게 됩니다.

따라서 함수의 반환 값은 다음과 같아야 합니다.

```
[[0, 2, 3], [1], [4, 5], [6,7]]
```

각 팀은 오름차순으로 정렬되어 있어야 하며, main 함수에는 다양하게 상황을 예시로 작성되어 있어야 합니다.