

# 핀테크 산업 응용

엄현상 교수님 강의 실습 1차시  
서윤형, 반지훈  
2022/05/10

# Contents

1. Go install

2. Hello Go

3. 자료형

4. 함수

5. 동시성

6. 디비 연동

7. 서버 통신

8. 테스트

오늘은 앞으로의 수업 실습 진행에 무리가 없도록,  
프로그래밍 언어 Go 에 대해 학습하겠습니다

# 0 Go 란?



# 0 Go 란?

<https://www.youtube.com/watch?v=LJvEljRBSDA>



Write in Go (Fall 2014)

조회수 193,032회 • 2014. 10. 25.

👍 3.8천    🗑️ 싫어요    ➦ 공유    ✂️ 클립    ➕ 저장    ...

## 0 Go 란?

- C++와 같이 Go는 컴파일러를 통해 컴파일
- 정적 타입(Statically Typed)의 언어
- Java와 같이 Go는 Garbage Collection 기능을 제공
- Go는 단순하고 간결한 프로그래밍 언어를 지향
- 인터프리터 언어(파이썬)와 다르게 사전 컴파일 가능

## 0 Go 란?

- Java의 절반에 해당하는 25개의 키워드(예약어)만으로 프로그래밍이 가능

1	<code>break</code>	<code>default</code>	<code>func</code>	<code>interface</code>	<code>select</code>
2	<code>case</code>	<code>defer</code>	<code>go</code>	<code>map</code>	<code>struct</code>
3	<code>chan</code>	<code>else</code>	<code>goto</code>	<code>package</code>	<code>switch</code>
4	<code>const</code>	<code>fallthrough</code>	<code>if</code>	<code>range</code>	<code>type</code>
5	<code>continue</code>	<code>for</code>	<code>import</code>	<code>return</code>	<code>var</code>

- Communicating Sequential Processes(CSP) 스타일의 Concurrent 프로그래밍을 지원

# 1 Go 설치하기

## 1) 최신버전 확인 및 다운로드

> <https://go.dev/dl/>

## 2) 설치 확인

> Go version

## 3) 환경 변수 확인

> Go env



# 1 Go 설치하기

go1.18.1 ▾

File name	Kind	OS	Arch	Size
<a href="#">go1.18.1.src.tar.gz</a>	Source			22MB
<a href="#">go1.18.1.darwin-amd64.tar.gz</a>	Archive	macOS	x86-64	137MB
<a href="#">go1.18.1.darwin-amd64.pkg</a>	Installer	macOS	x86-64	138MB
<a href="#">go1.18.1.darwin-arm64.tar.gz</a>	Archive	macOS	ARM64	132MB
<a href="#">go1.18.1.darwin-arm64.pkg</a>	Installer	macOS	ARM64	132MB
<a href="#">go1.18.1.linux-386.tar.gz</a>	Archive	Linux	x86	108MB
<a href="#">go1.18.1.linux-amd64.tar.gz</a>	Archive	Linux	x86-64	135MB
<a href="#">go1.18.1.linux-arm64.tar.gz</a>	Archive	Linux	ARM64	104MB
<a href="#">go1.18.1.linux-armv6l.tar.gz</a>	Archive	Linux	ARMv6	105MB
<a href="#">go1.18.1.windows-386.zip</a>	Archive	Windows	x86	123MB
<a href="#">go1.18.1.windows-386.msi</a>	Installer	Windows	x86	108MB
<a href="#">go1.18.1.windows-amd64.zip</a>	Archive	Windows	x86-64	151MB
<a href="#">go1.18.1.windows-amd64.msi</a>	Installer	Windows	x86-64	132MB

1) 자신의 OS 에 맞는 버전  
installer 클릭

2) Go 설치 경로 확인  
> Which go

# 1 Go 설치하기

## 1) GOROOT

- > Go 가 설치된 디렉토리
- > GOROOT/bin : 실행 파일
- > GOROOT/pkg : 표준 패키지

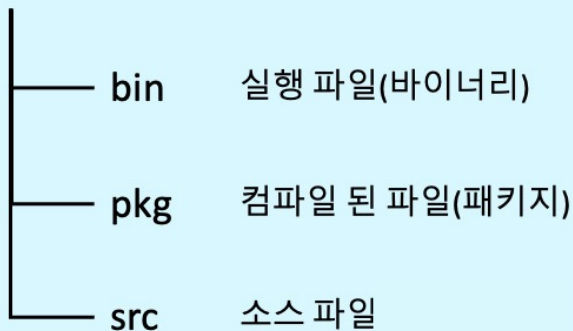
## 2) GOPATH

- > 표준 패키지 이외 패키지
- > Go get [github.com/hello/package](https://github.com/hello/package)

# 1 Go 설치하기



## GOPATH



## 1) GOPATH 확인

> Echo \$GOPATH

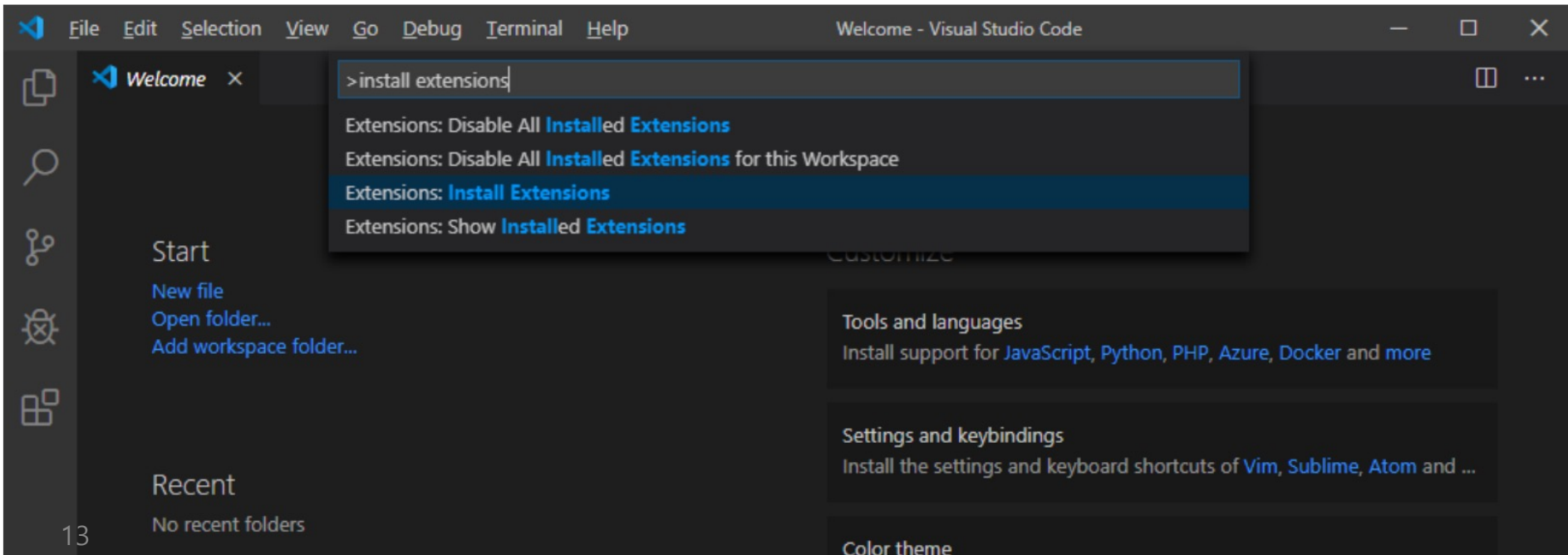
## 2) GOPATH/src 에 모든 소스 코드가 존재해야한다

# 1 Go 설치하기

- Go 지원 통합 개발 환경
  - GoLand
  - VSCode <https://code.visualstudio.com/download>
- 온라인 실행 환경
  - <https://replit.com/languages/go>

# 1 Go 설치하기

- VSCode
  - Ctrl + Shift + p



# 1 Go 설치하기

- VSCode
  - go

The screenshot shows the Visual Studio Code interface with the Go extension marketplace. The left sidebar displays a list of extensions, with 'Go' by Microsoft selected. The main panel shows the details for the 'Go' extension, including the Go logo, version 0.12.0, and a green 'Install' button. The extension is described as 'Rich Go language support for Visual Studio Code'. The interface also shows the 'Extensions: Marketplace' view, the 'Go' extension's details, and the 'Go for Visual Studio Code' section with links to 'gitter', 'join chat', 'build', and 'failing'.

File Edit Selection View Go Debug Terminal Help Extension: Go - Visual Studio Code

EXTENSIONS: MARKETPLACE

go

**Go** 0.12.0  
Rich Go language support for Visual S...  
Microsoft [Install](#)

**Go Outliner** 0.1.20  
Go code outline explorer  
766b [Install](#)

**Go Critic** 0.1.0  
Integration for the go-critic golang lin...  
neverik [Install](#)

**Go Doc** 0.1.1  
Show documentation of go symbols a...  
Minhaz Ahmed Syrus [Install](#)

**Go Autotest** 1.6.0

**Go** ms-vscode.go

Microsoft | 2,709,448 | ★★★★★ | Repository | License

Rich Go language support for Visual Studio Code

[Install](#)

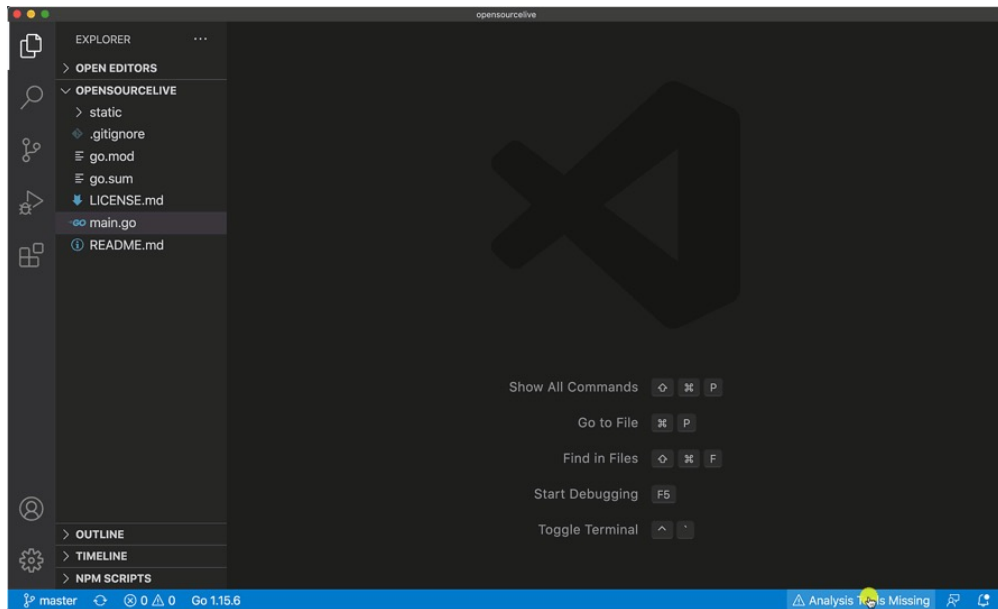
[Details](#) [Contributions](#) [Changelog](#)

Go for Visual Studio Code

[gitter](#) [join chat](#) [build](#) [failing](#)

# 1 Go 설치하기

- `$GOPATH/src/` 아래에 `main.go`
- VSCode 열면 missing tools 를 설치하라는 메시지가 나옴



# 1 Hello Go

- 컴파일과 동시에 실행
  - `go run main.go`
- 실행파일.exe 생성
  - `go build main.go`



# 1 Hello Go

- Hello go!

```
✓ FIRSTGO
-GO main.go

-GO main.go > ...
1  package main
2
3  import "fmt"
4
5  func main() {
6      fmt.Println("hello go")
7  }
```

10분뒤 다시 시작합니다

## 3 자료형

### 1) 변수의 선언 :=

- > 자동 타입 지정됨
- > 함수 안에서만 동작
- > 전역변수는 불가능

```
a := 1           // int
b := 3.5         // float64
c := "Hello, world!" // string
```

### 3 자료형

#### 1) 문자열 - ``

- > 복수 라인의 문자열을 표현
- > Raw String 그대로의 값

#### 2) 문자열 - " "

- > 복수 라인에 걸쳐 쓸 수 없으며, 인용부호 안의 Escape(\n,\t...) 문자열들은 특별한 의미로 해석
- > 문자열을 여러 라인에 걸쳐 쓰기 위해서는 + 연산자를 이용해 결합하여 사용

### 3 자료형

```
GO main.go > ...
1  package main
2
3  import "fmt"
4
5  func main() {
6      rawLiteral := `아리랑\n
7      아리랑\n
8      아라리요`
9
10     interLiteral := "아리랑아리랑\n아리리요"
11     // 아래와 같이 +를 사용하여 두 라인에 걸쳐 사용할 수도 있다.
12     // interLiteral := "아리랑아리랑\n" +
13     //                  "아리리요"
14
15     fmt.Println(rawLiteral)
16     fmt.Println()
17     fmt.Println(interLiteral)
18 }
```

## 3 자료형

### 1) 변수의 선언 =

- > 타입 명시적으로 선언
- > 변수를 선언하면서 초기값을 지정하지 않으면, Go는 Zero Value를 기본적으로 할당
- > 즉, 숫자형에는 0, bool 타입에는 false, 그리고 string 형에는 "" (빈문자열)을 할당
- > 선언 후 값 변경 가능 여부에 따라 const 혹은 var 사용

### 3 자료형

#### 1) constant 는 변경 불가

```
const a string = "hi"  
a = "bye" // 불가능
```

#### 2) var 는 변경 가능

```
var name string = "y"  
name = "z" // 가능
```

## 3 자료형

### 1) 타입 변환

- > 암묵적으로 일어나지 않음
- > 명시적으로 변환 필수

```
func main() {  
    var i int = 100  
    var u uint = uint(i)  
    var f float32 = float32(i)  
    println(f, u)  
  
    str := "ABC"  
    bytes := []byte(str)  
    str2 := string(bytes)  
    println(bytes, str2)  
}
```



## 3 자료형

### 1) 포인터

> C 와 동일하게 사용

```
func main(){  
    a := 2  
    b := &a  
    *b = 20  
    fmt.Println(a)  
    a = 15  
    fmt.Println(*b) //a의 주소값을 참조하기 때문에 a가 바뀔때 같이 바뀐다  
}
```

## 3 자료형

### 1) 배열

- > 연속적인 메모리 공간에 동일한 타입의 데이터를 순서적으로 저장하는 자료구조

```
func main() {  
    var a [3]int //정수형 3개 요소를 갖는 배열 a 선언  
    a[0] = 1  
    a[1] = 2  
    a[2] = 3  
    fmt.Println(a[1]) // 2 출력  
}
```

## 3 자료형

### 1) Slice

#### > 가변형 데이터 배열

```
slice1 := make([]int, 10)
slice2 := []int{1,2,3}
```

## 3 자료형

### 1) Maps

- > 키(Key)에 대응하는 값(Value)을 신속히 찾는 해시테이블(Hash table)을 구현한 자료구조
- > make() 함수를 써서 초기화 가능

```
func main(){  
    infos := map[string]string{"name":"lemon","age":"12"}  
    //    map[key_type]value_type  
    for keys, value := range infos {  
        fmt.Println(value)  
    }  
}
```

### 3 자료형

#### 1) 쓰지 않는 변수는 에러가 났음

> \_로 선언해줘야함

```
-GO main.go > ...
1  package main
2
3  import "fmt"
4
5  func main() {
6      infos := map[string]string{"name": "lemon", "age": "12"}
7      //    map[key_type]value_type
8      for _, value := range infos {
9          fmt.Println(value)
10     }
11 }
12
```

## 3 자료형

### 1) 구조체

> 이름이 지정된 필드가 포함된 타입

```
type Circle struct {  
    x, y, z float64  
}  
  
var c Circle // 또는  
c := new(Circle)  
  
c := Circle{x: 0, y: 0, r: 5} // 초기 변수 대입  
c := Circle{0, 0, 5} // 멤버 변수 순서 알고 있으면  
  
// 접근방법  
c.x = 10
```

## 4 Go package

### 1) 표준 라이브러리 패키지

- > 표준 라이브러리 패키지들은 GOROOT/pkg 안에 존재

### 2) main package

- > "main" 이라고 명명된 패키지는 Go Compiler에 의해 공유 라이브러리가 아닌 실행(executable) 프로그램이 됨
- > main 패키지 안의 main() 함수가 프로그램의 시작점 즉, Entry Point임
- > 패키지를 공유 라이브러리로 만들 때에는, main 패키지나 main 함수를 사용해서는 안 됨

## 4 Go package

### 1) Import

- > 표준 패키지는 GOROOT/pkg 에서 찾음
- > 사용자 패키지나 3rd Party 패키지의 경우 GOPATH/pkg 에서 패키지를 찾음
- > Relative import 가 없음
  - Import “../p/test”



## 4 Go package

### 1) 사용자 정의 패키지 생성

- > 하나의 서브 폴더안에 있는 .go 파일들은 동일한 패키지명을 가짐
- > 패키지명은 해당 폴더의 이름과 같게 함
- > 즉, 해당 폴더에 있는 여러 \*.go 파일들의 (컴파일된 결과인) 해당 obj 파일들이 하나의 패키지로 묶임

## 4 Go package

### 1) Scope

- > 이름(Identifier)이 첫문자를 대문자로 시작하면 이는 public
- > 즉, 패키지 외부에서 이들을 호출 및 사용 가능
- > 반면, 이름이 소문자로 시작하면 이는 non-public 으로 패키지 내부에서만 사용

## 4 함수와 메소드

### 1) 함수의 선언

- > func 함수명(param)(return){...}
- > 다양한 수의 파라미터 전달 가능 (가변 인자)

```
func say(msg ...string){  
    for _,s :=range msg{  
        println(s)  
    }  
}
```

## 4 함수와 메소드

### 1) 함수의 리턴

> Return 을 여러개 해줄 수도 있음

```
func sum(nums ...int) (int, int) {  
    s := 0      // 합계  
    count := 0  // 요소 갯수  
    for _, n := range nums {  
        s += n  
        count++  
    }  
    return count, s  
}
```

## 4 함수와 메소드

### 1) 함수의 종료

- > defer ...
- > Close 호출을 Open 호출 가까이에 뒤서 이해하기가 쉬움
- > 지연된 함수는 런타임 패닉이 일어나더라도 실행됨

```
go main.go > ...  
1  package main  
2  
3  import "fmt"  
4  
5  func main() {  
6      defer fmt.Print("end")  
7      fmt.Print("start")  
8  }
```

## 4 함수와 메소드

### 1) 익명함수

- > 함수 전체를 변수에 할당하거나 다른함수 파라미터에 직접 정의되어 사용됨
- > 변수명이 함수명과 같이 취급되어 변수명으로 함수 호출
- > 바로 실행을 원한다면 func(){...}()

```
func main() {  
    func() {  
        fmt.Printf("hi")  
    }()  
}
```

## 1) 새 디렉토리에 Go 파일 만들기

> mypackage/mylib.go

## 2) 외부 파일의 함수 호출하기

> Public 선언은 대문자면 충분함

## 3) Import?

> Relative import 없음

## 1) 새 Go 파일에 함수 작성

- > Main 이 아닌 패키지
- > 함수의 인풋값
  - 하나의 정수
- > 짝수면 true, 홀수면 false 출력하는 함수 작성
- > 함수의 리턴값
  - 짝수면 true, 홀수면 false
  - 입력받은 수
- > Main.go 에서 함수를 호출해서 테스트 해보기



10분뒤 다시 시작합니다

## 4 함수와 메소드

### 1) 메소드

- > 객체에 연결된 함수
- > 함수명 앞에 Receiver라고 부르는 타입과 변수명을 붙임

```
func (p Point) printInfo() {  
    p.X += 2  
}  
func (p *Point) printInfo() {  
    p.X += 2  
}
```

## 1) 구조체와 함수, 메소드를 이용하여 구현해보자

- > Rectangle 구조체를 선언하고
- > 구조체에 대한 함수와 메소드를 작성하자
- > 어떤 것이 메소드가 될 수 있을까?
  - 넓이 리턴
  - 둘레 리턴
  - 가로 수정
  - 세로 수정
  - 정사각형 판별

## 5 Go 동시성

### 1) Go Routine

- > 고 런타임이 관리
- > Lightweight 논리적(가상) 스레드
- > 현재 수행 흐름과 별개의 흐름을 만들어줌

### 2) 고루틴 생성

- > 함수 앞에 go 를 붙여 호출
- > → 논리적으로 별개의 흐름이 되어 진행됨

## 5 Go 동시성

### 1) Concurrency

- > 커피마시면서 신문보기 어느 것을 먼저해야하는지는 상관없음
- > → 동시성이 있는 두 루틴은 서로 의존 관계가 없음

```
func main() {  
    go fmt.Print("go routine")  
    fmt.Print("main")  
    time.Sleep(time.Second * 3)  
}
```

## 5 Go 동시성

### 1) Runtime 패키지

- > 동시에 몇개의 OS 스레드에서 GO 코드를 수행할지 결정
- > default 는 1개

### 2) runtime.NumCPU()

- > 컴퓨터의 CPU 수

### 3) runtime.GOMAXPROCS()

- > 파라미터로 CPU 개수 설정

## 1) Go 동시성을 이용한 실습을 해보자 1

- > 기기의 CPU 수를 구한 후 OS 스레드를 최대한 돌리기
- > 이전에 만들었던 홀짝 판별 함수를 수정해 Go routine 로 실행하자
- > 전역변수를 함수의 인풋으로 넣는다
- > 메인에서 전역변수를 ++한다
- > 비동기적으로 실행되는 Go routine 을 어떻게 제어할까?

## 1) 왜 함수가 작동이 되지 않을까?

> `Time.Sleep()`



### 1) 멀티스레드 프로그래밍의 어려움



## 5 Go 동시성

### 1) Go channel

- > routine 끼리 정보를 공유하는 방식이 아니라 메시지를 주고받는 방식을 쓴다면 lock으로 공유 메모리를 관리할 필요가 없고 구현도 어렵지 않음
- > go routine에서 데이터를 주고 받는 통로
- > 넣은 데이터를 뽑아낼 수 있는 마치 파이프같은 형태

## 5 Go 동시성

### 1) 동기적 Go channel 생성

- > map이나 slice처럼 채널도 쓰기 전에 채널임을 선언해줘야 함
- > 동기적으로 동작하는 기본 채널

```
c1 := make(chan int)
var chan int c2=c1
```

- > Default 크기가 0 -> 받은 데이터를 보관할 공간 없음
- > 채널의 양쪽이 다른 쪽이 준비될 때까지 기다림

## 5 Go 동시성

### 1) 비동기적 Go channel 생성

- > map이나 slice처럼 채널도 쓰기 전에 채널임을 선언
- > 비동기적으로 동작하는 버퍼 채널

```
c := make(chan int, 1)
```

- > 두번째 파라미터에 사용할 버퍼 갯수
- > 메시지를 보내거나 받을 때 채널이 이미 꽉 차 있지 않는 이상 기다리지 않음

## 5 Go 동시성

### 1) Go channel 데이터 송수신

- > 채널에서 데이터를 꺼내, 특정 변수에 담는 것을 수신
- > 채널로 데이터를 넣는 것을 송신
- > 채널로 송신/수신이 되는 순간 해당 go routine은 정지하고, 다시 송수신을 받으면 다시 lock이 풀리고 다음 작업을 수행
- > 채널 오퍼레이터 <-를 통해 값을 주고 받을 수 있음

## 5 Go 동시성

### 1) Go channel 데이터 송수신

```
func main() {  
    c1 := make(chan int)  
  
    go func() {  
        c1 <- 123  
    }()  
  
    go func() {  
        temp := <-c1  
        fmt.Printf("%d", temp)  
    }()  
    time.Sleep(time.Second * 3)  
}
```

## 5 Go 동시성

### 1) Go channel 닫기

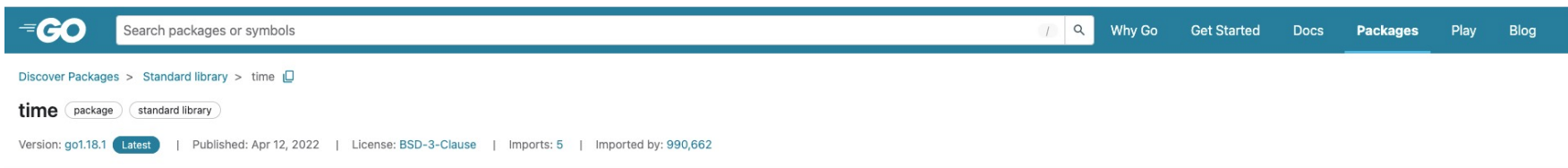
- > 채널을 닫지 않으면 리소스 부하
- > `close(mychan)`

Go ▾

```
chEven := make(chan bool)
defer close(chEven)
```

## 1) Golang 라이브러리 활용하기

> <https://pkg.go.dev/time>



The screenshot shows the Go Packages website interface. At the top, there is a navigation bar with the Go logo and a search bar. Below the search bar, the breadcrumb path "Discover Packages > Standard library > time" is visible. The main content area displays the "time" package, which is part of the "standard library". It includes metadata such as "Version: go1.18.1", "Published: Apr 12, 2022", "License: BSD-3-Clause", "Imports: 5", and "Imported by: 990,662".

### func Now ¶

```
func Now() Time
```

Now returns the current local time.



# \* Go 동시성 실습

## 1) Go 동시성을 이용한 실습을 해보자 2

- > Go channel 을 이용해보자
- > 동기 채널로 홀짝 판별을 구현해보자
  - Input 1~100

## 2) 소요 시간 재기

- > Time 패키지를 이용

# \* Go 동시성 실습

## 1) Go 동시성을 이용한 실습을 해보자 3

- > Go channel 을 이용해보자
- > 비동기 채널(용량 10)로 홀짝 판별을 구현해보자
  - Input 1~100

## 2) 소요 시간 재기

- > Time 패키지를 이용

10분뒤 다시 시작합니다

## 6 Go 디비 연동

### 1) mySql 기기에 설치

- > <https://shanepark.tistory.com/41> (Mac)
- > <https://goddaehee.tistory.com/277> (Window)

## 6 Go 디비 연동

### 1) mySql 에 testdb 만들기

- > `mysql.server start`
- > `mysql -u root -p`
- > `create database `testdb`;`
- > `show databases;`

## 6 Go 디비 연동

### 1) Go 와 mySql 연동

```
db, err := sql.Open("mysql", "root:password@/testdb")
if err != nil {
    panic(err)
}
defer db.Close()
```

## 6 Go 디비 연동

### 1) Table 생성 및 insert

```
// CREATE TABLE
_, err = db.Exec("CREATE TABLE test(id INT, name VARCHAR(20))")
if err != nil {
    log.Fatal(err)
}
```

```
// INSERT INTO
_, err = db.Exec("INSERT INTO test(id, name) VALUES (1, 'Alex')")
if err != nil {
    log.Fatal(err)
}
```

## 6 Go 디비 연동

### 1) Select 쿼리 실행

```
rows, err := db.Query("SELECT id, name FROM test")
if err != nil {
    log.Fatal(err)
}
defer rows.Close() //반드시 닫는다 (지연하여 닫기)
```



## 7 Go 서버 통신

### 1) Go 를 이용해 간단한 웹사이트 구동

```
func index(w http.ResponseWriter, r *http.Request) {  
    t, _ := template.ParseFiles("index.html")  
    t.Execute(w, "hello world!")  
}  
  
func main() {  
    http.HandleFunc("/", index)  
    log.Fatal(http.ListenAndServe(":8080", nil))  
}
```

## 7 Go 서버 통신

### 1) Go 를 이용해 간단한 웹사이트 구동

```
3 func index(w http.ResponseWriter, r *http.Request) {  
9     switch r.Method {  
0     case "GET":  
1         http.ServeFile(w, r, "index.html")  
2     case "POST":  
3     }  
4 }
```

## 7 Go 서버 통신

### 1) 웹사이트에 버튼 만들기

#### > Html 파일 수정

```
≡ index.html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset = "UTF-8">
5  |   <title>GO WEB</title>
6  </head>
7  <body>
8  |   <form method="POST" action="/">
9  |   |   <input type="submit" value="submit" />
10 |   </form>
11 </body>
12 </html>
13
```

## 1) Go 를 이용해 로그인 기능이 있는 웹사이트 구현

- > Id, password 를 db 에 저장
- > Db 에 일치하는 정보가 있다면 로그인 가능하도록

10분뒤 다시 시작합니다

## 8 Go 테스트

### 1) 테스트 실행

- > go test 명령 실행해 실행 가능
- > \*\_test.go를 테스트 코드로 인식하고 일괄적으로 실행함

```
~/goapp/src/calc$ ls
calc.go          calc_test.go

~/goapp/src/calc$ go test
PASS
ok      calc      0.007s
```

## 8 Go 테스트

### 1) 테스트 코드 작성

- > 테스트 프레임워크 내장
- > 테스트 메소드명은 특별히 TestXxx 형태
- > 테스트 파라미터는 t \*testing.T

```
func TestSum(t *testing.T) {  
    s := calc.Sum(1, 2, 3)  
  
    if s != 6 {  
        t.Error("Wrong result")  
    }  
}
```

## 1) 오늘 수행했던 과제코드 중 하나에 대한 테스트 함수를 작성

- > 테스트 함수를 작성하고
- > 테스트 결과값을 캡처하여 제출



긴 수업 잘 들어주셔서 감사합니다  
다음 수업에 만나요😊