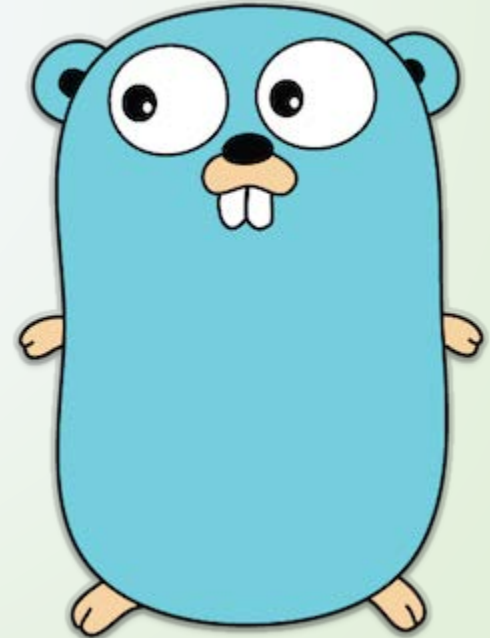


5일차 실습



- 체인코드 개발 기초

- ① Go 언어



① Go 언어

Go 언어 소개



- 구글이 2009년에 만든 C, JAVA와 같은 범용 프로그래밍 언어로 시스템 프로그래밍 및 네트워크 프로그램의 개발을 목표로 만들어진 언어
- 컴파일 시간에 타입을 체크하는 정적 타입 언어
- 다른 언어들보다 적은 25개의 키워드를 사용하여 비교적 쉽게 프로그래밍 가능
- 병행성(concurrency)을 잘 지원해주는 언어

Go 언어 소개



- 범용 프로그래밍 언어
- 깔끔하고 간결하게 생산성 높은 프로그램 작성 가능
- 생산성이 높은 이유
 - 부분적이나 **자료형 추론을 제공**하여 반복하여 자료형 이름을 작성할 필요 없음
 - 소스 코드 형식을 자동으로 맞춰주는 도구 및 여러 **편리한 도구** 기본 제공
 - Example 테스트를 이용하여 **쉽게 테스트 코드 작성**하면서 **코드 문서화** 가능
 - **함수 리터럴 및 클로저**를 자유자재로 사용 가능
 - 명시적으로 인터페이스를 지정하지 않아도 **인터페이스의 구현**이 가능하여 기존에 있던 코드를 고치지 않고도 유연한 구현이 가능

Go 언어 소개



- 생산성이 높은 이유
 - 채널을 이용하여 동시성 구현을 락 등을 이용하지 않고 간편하게 할 수 있으며, 자체적 지원으로 교착 상태나 경쟁 상태 파악이 쉬움
 - 컴파일 속도가 빨라서 컴파일 및 테스트를 반복적으로 수행하면서 코드 작성하기에 용이
 - 가비지 컬렉션 지원으로 메모리 관리에 대한 부담 저하
 - 자료형 리터럴을 쉽게 사용 가능
- Go언어를 배우는 이유
 - 풍부한 라이브러리 기반의 언어
 - 웹서버, 웹브라우저, 봇, 검색엔진, 컴파일러, OS 개발에 사용
 - Hyperledger Fabric 환경구축 강의 시 실시 사용한 Docker의 개발 언어도 Go언어

Go 언어 간단한 프로그램 돌려보기



- Go 놀이터
 - Go언어를 컴파일, 링크하여 실행한 뒤 결과를 확인할 수 있는 웹서비스
 - 샌드박스 내에서 실행되므로 Go언어의 컴파일러를 설치할 필요 없음
 - <https://play.golang.org/>

The screenshot shows the 'The Go Playground' interface. At the top, there are buttons for 'Run', 'Format', 'Imports', 'Share', and 'About'. The main area contains a Go code snippet:

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     fmt.Println("Hello, playground")
9 }
10
```

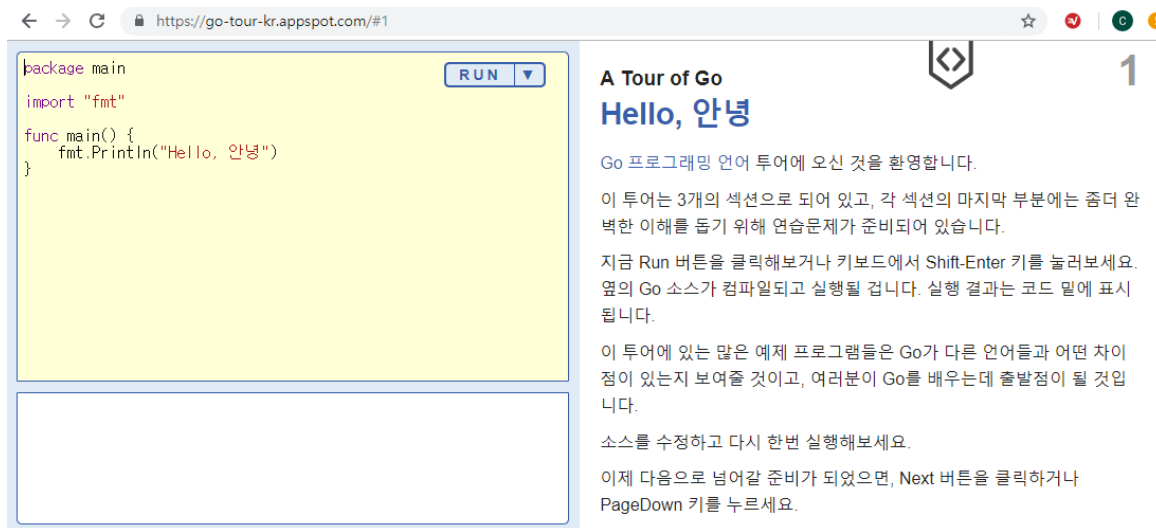
Below the code, the output 'Hello, playground' and 'Program exited.' are displayed. Annotations with callouts explain the interface:

- Run**: 코드 작성창에 작성한 코드를 컴파일, 링크, 실행과정을 거쳐 실행
- Format**: 코드 작성창에 작성한 코드의 형식을 맞춰주는 작업 수행
- Imports**: 체크 시 Format 작업 수행할 때, 코드를 보고 필요한 import 경로를 자동으로 추가
- Share**: 작성한 코드를 URL로 만들어 공유할 수 있도록 함
- 코드작성 창**: The code editor area.
- 코드 실행결과 출력 창**: The output area showing 'Hello, playground' and 'Program exited.'

Go 언어 간단한 프로그램 돌려보기



- Go 여행
 - Go언어를 컴파일, 링크하여 실행한 뒤 결과를 확인할 수 있는 웹서비스
 - 샌드박스 내에서 실행되므로 Go언어의 컴파일러를 설치할 필요 없음
 - Go언어 학습을 위한 예제와 설명 제공
 - <https://go-tour-kr.appspot.com/>

A screenshot of the Go Tour web application. The browser address bar shows 'https://go-tour-kr.appspot.com/#1'. The main content area is divided into two panels. The left panel is a code editor with a yellow background, containing the following Go code:

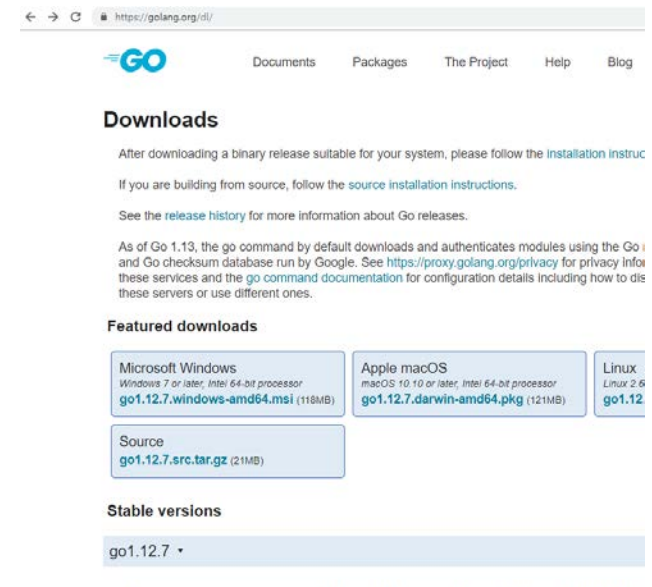
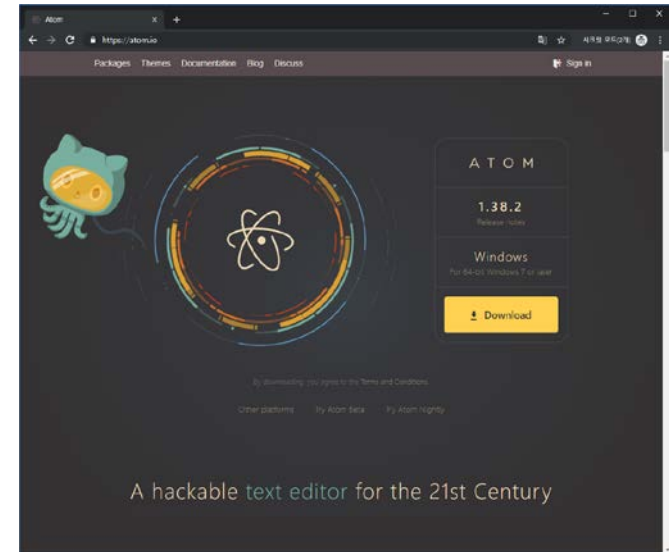
```
package main
import "fmt"
func main() {
    fmt.Println("Hello, 안녕")
}
```

A 'RUN' button is visible in the top right corner of the code editor. The right panel has a white background and contains the title 'A Tour of Go' followed by 'Hello, 안녕' in blue. Below the title, there is Korean text explaining the tour and providing instructions on how to run the code. The text includes: '이 투어는 3개의 섹션으로 되어 있고, 각 섹션의 마지막 부분에는 좀더 완벽한 이해를 돕기 위해 연습문제가 준비되어 있습니다.', '지금 Run 버튼을 클릭해보거나 키보드에서 Shift-Enter 키를 눌러보세요. 옆의 Go 소스가 컴파일되고 실행될 겁니다. 실행 결과는 코드 밑에 표시 됩니다.', '이 투어에 있는 많은 예제 프로그램들은 Go가 다른 언어들과 어떤 차이점이 있는지 보여줄 것이고, 여러분이 Go를 배우는데 출발점이 될 것입니다.', '소스를 수정하고 다시 한번 실행해보세요.', and '이제 다음으로 넘어갈 준비가 되었으면, Next 버튼을 클릭하거나 PageDown 키를 누르세요.'

GO 언어 개발 IDE설치 및 설정



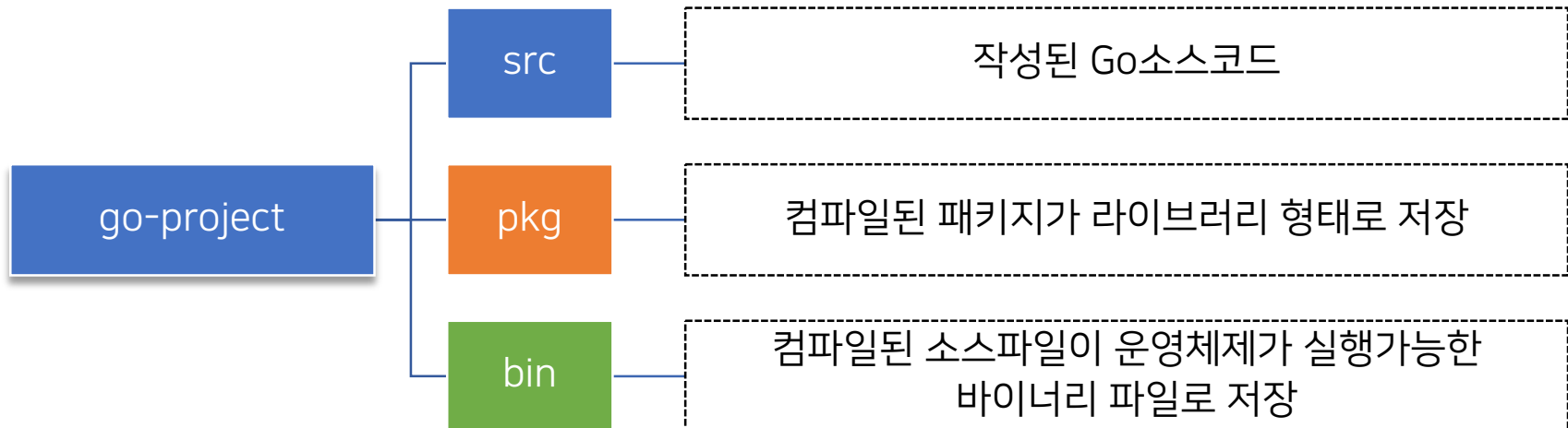
- 비주얼 스튜디오 코드 설치
 - <https://code.visualstudio.com/>
- Go language
 - <https://golang.org/dl/>
- Git 설치
 - <https://git-scm.com/downloads>



프로젝트 추가



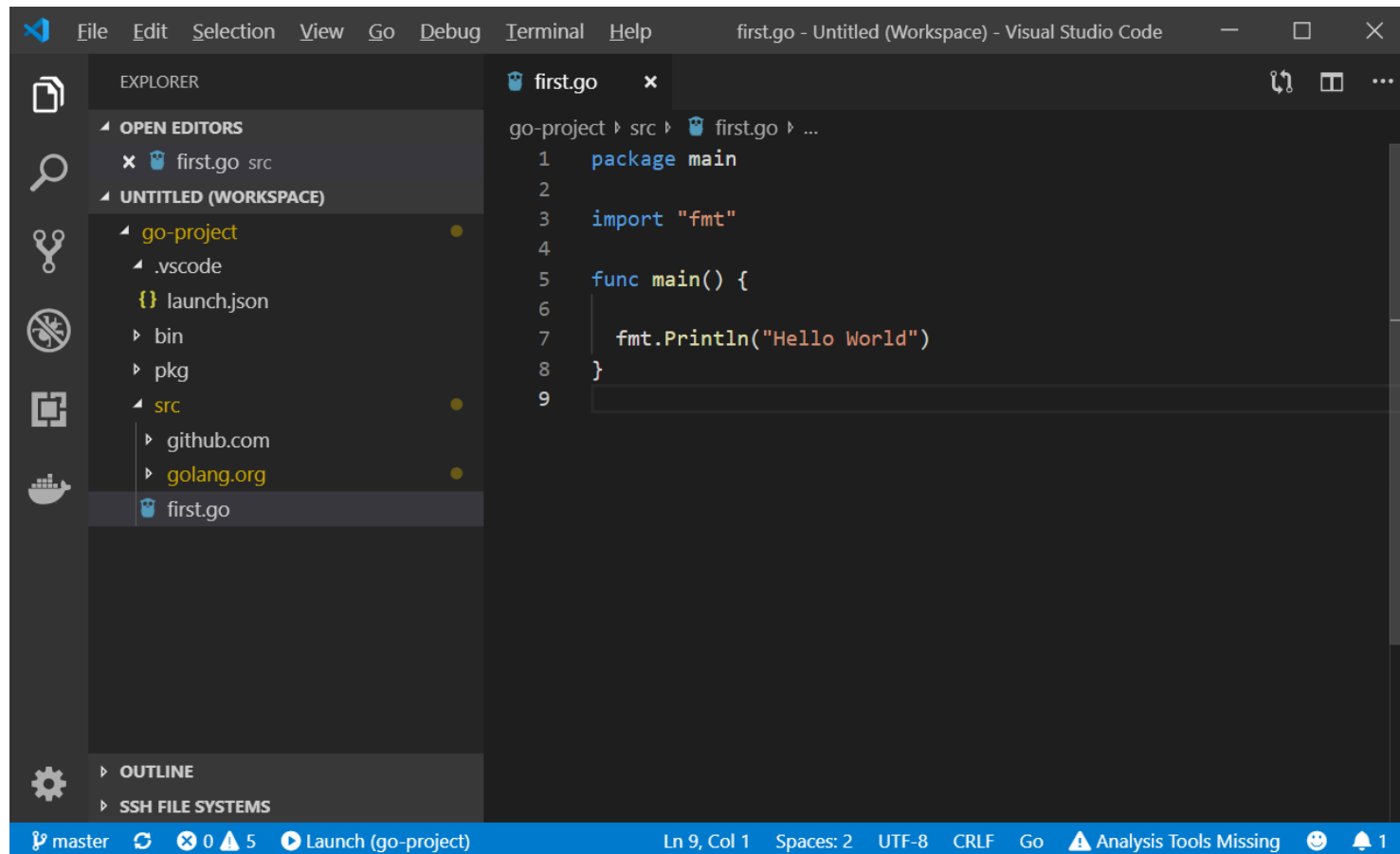
- File - Add Folder to workspace
- 바탕화면에 go-project 폴더생성 -> ADD 버튼 클릭
- go-project 폴더 선택
 - 왼쪽 트리 go-project에서 오른쪽클릭
 - new-folder -> src생성
 - new-folder -> bin생성
 - new-folder -> pkg생성



소스파일 추가



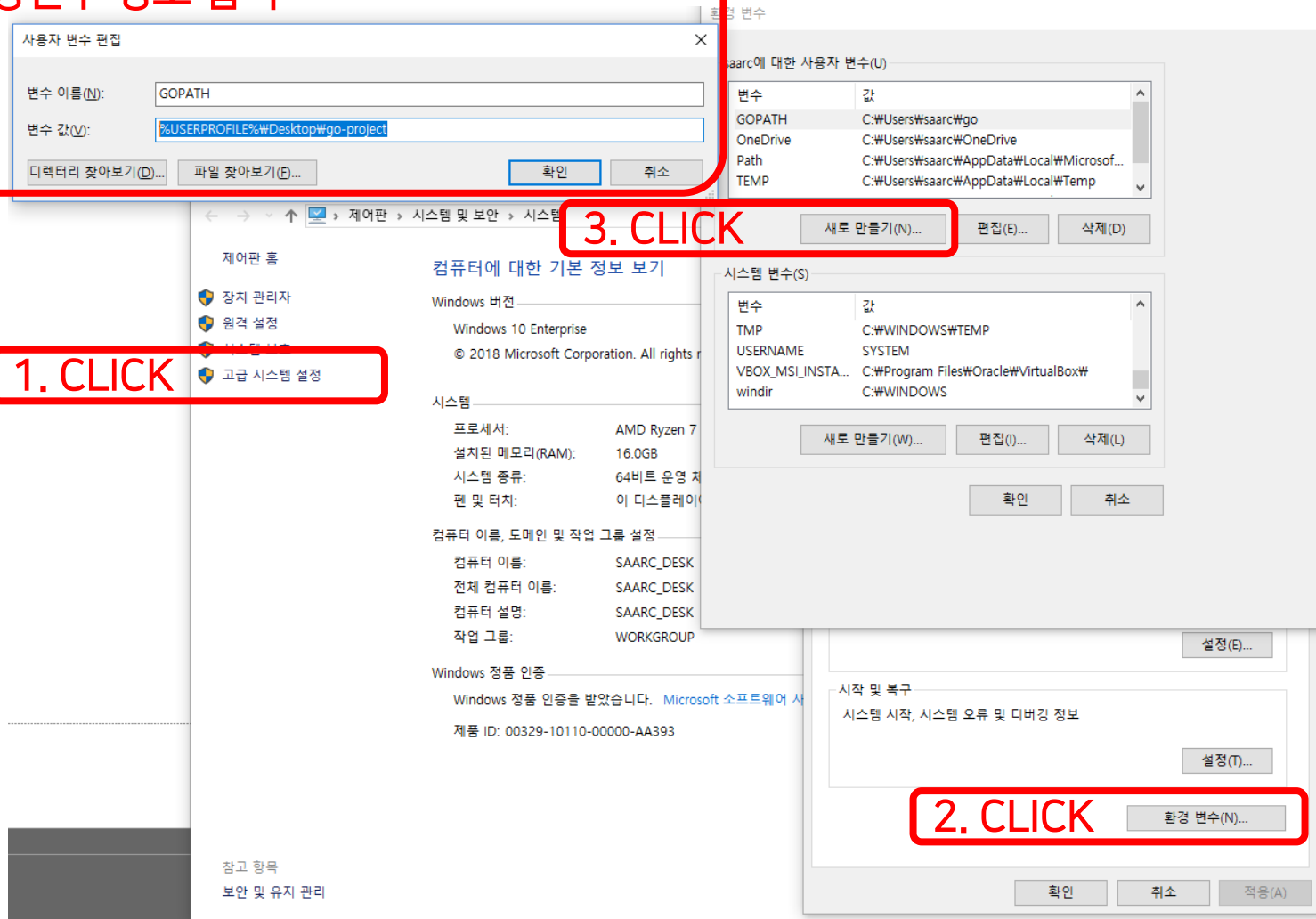
- src에서 오른쪽 클릭
 - new-file -> first.go 생성
 - first.go 더블클릭
 - 소스코드 작성



GOPATH 환경변수 등록

- windows key + Break key

- 4. GOPATH 환경변수 정보 입력
- 5. 확인클릭



The screenshot illustrates the process of registering the GOPATH environment variable in Windows. The steps are numbered 1 through 5:

- 1. CLICK**: Click on '고급 시스템 설정' (Advanced system settings) in the '제어판' (Control Panel) window.
- 2. CLICK**: Click on '환경 변수(N)...' (Environment variables...) in the '시스템 및 보안' (System and Security) window.
- 3. CLICK**: Click on '새로 만들기(N)...' (New...) in the '시스템 변수(S)' (System variables) window.
- 4. GOPATH 환경변수 정보 입력**: Enter the variable name 'GOPATH' and the value '%USERPROFILE%\Desktop\go-project' in the '사용자 변수 편집' (User variable edit) window.
- 5. 확인클릭**: Click on '확인' (OK) in the '사용자 변수 편집' window.

설치할 extention



- 메뉴 -> Preference -> Extention
 - go extention 설치
- F1->
 - go:install/update tool
 - dlv 체크박스 선택
- F1->
 - Debug:open launch.json
- ctrl+`
 - go get github.com/derekparker/delve/cmd/dlv

File Edit Selection View Go Debug Terminal Help launch.json - Untitled (Workspace) - Visual Studio Code

EXPLORER

1 UNSAVED

first.go src

launch.json .vscode

UNTITLED (WORKSPACE)

go-project

.vscode

launch.json

bin

pkg

src

github.com

golang.org

first.go

OUTLINE

SSH FILE SYSTEMS

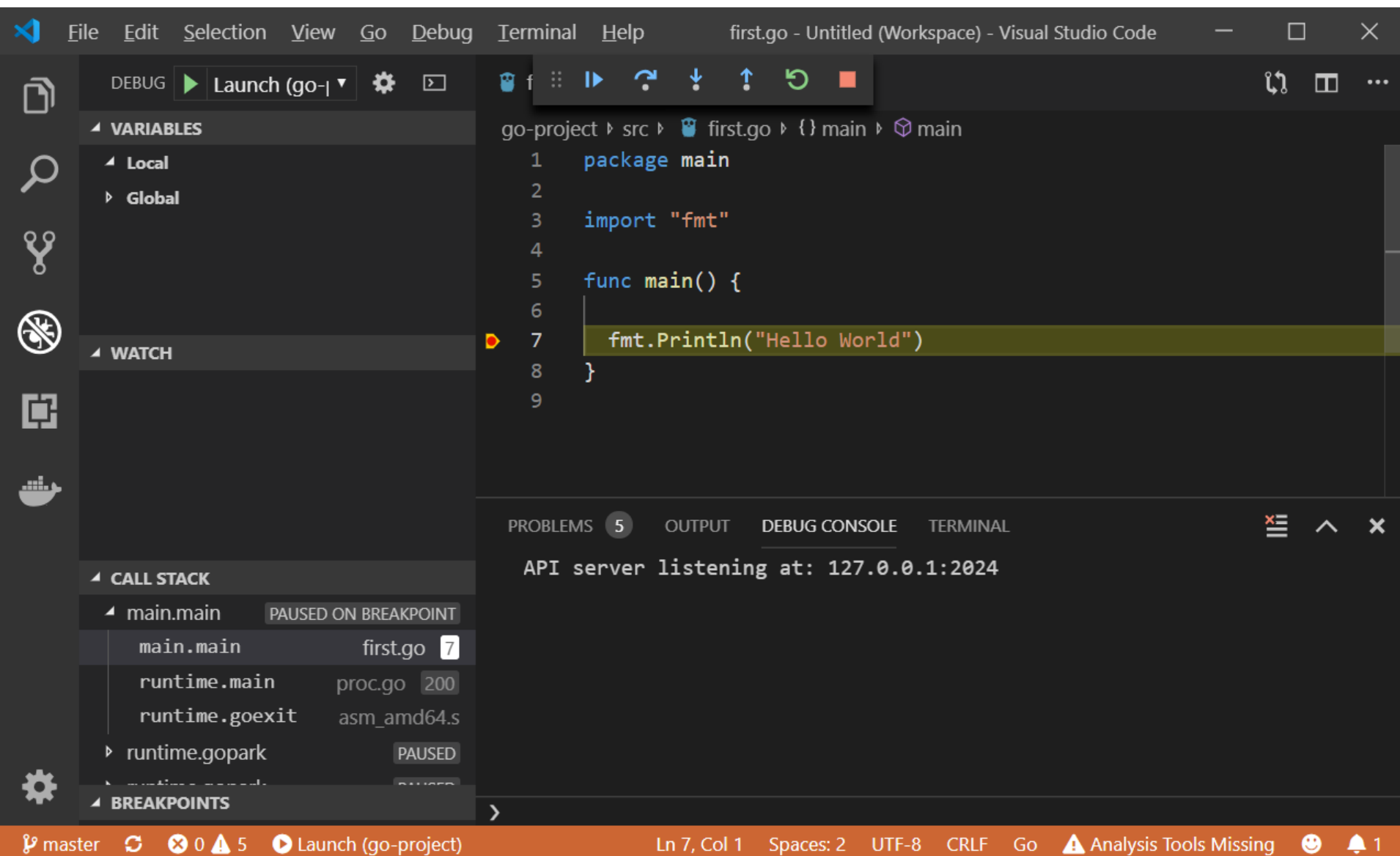
go-project .vscode launch.json Launch Targets { } Launch

```
1 {
2     // Use IntelliSense to learn about possible attributes.
3     // Hover to view descriptions of existing attributes.
4     // For more information, visit: https://go.microsoft.com/fwlink/
5     "version": "0.2.0",
6     "configurations": [
7         {
8             "name": "Launch",
9             "type": "go",
10            "request": "launch",
11            "mode": "auto",
12            "program": "${workspaceRoot}\\src",
13            "env": {},
14            "args": []
15        }
16    ]
17 }
18
```

Add Configuration...

Ln 12, Col 33 Spaces: 4 UTF-8 CRLF JSON with Comments 1

Hello World 프로그램 디버깅



Visual Studio Code interface showing a Go program being debugged. The program is named `first.go` and is located in the `src` directory of a `go-project`.

```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     fmt.Println("Hello World")
8 }
9
```

The program is paused at line 7, where the `fmt.Println("Hello World")` statement is executed. The output of the program is visible in the **DEBUG CONSOLE** panel:

```
API server listening at: 127.0.0.1:2024
```

The **CALL STACK** panel shows the execution path:

- `main.main` (first.go:7) - PAUSED ON BREAKPOINT
- `runtime.main` (proc.go:200)
- `runtime.goexit` (asm_amd64.s)

The **WATCH** panel is empty. The **PROBLEMS** panel shows 5 errors. The **TERMINAL** panel is also empty.

package, import



```
package main
```

모든 Go 프로그램은 패키지로 구성

```
import (  
    "fmt"  
    "math"  
)
```

모든 Go 프로그램은 패키지로 구성

```
func main() {  
    fmt.Println("Happy", math.Pi, "Day")  
}
```

모든 Go 프로그램은 main함수 포함

- 개발환경 설정
 - vscode 설치하기
 - go언어 설치하기
 - git 설치하기
 - GO-PROJECT디렉토리 만들기 (src, pkg, bin)
 - 환경변수 GOPATH설정
 - vscode extention설정
 - go extention
 - vscode go debug설정
 - dlv설치
 - launch.json에 program 수정
- Hello World 프로그램 작성 및 디버깅



체인코드 작성을 위한 Go 문법, 예제 그리고 실습

목차



자료형 및 변수

var (int, string , pointer)

함수와 제어구조

func / if-else, switch / for, while /

문자열 및 자료구조

string, array, slice, map

함수상세

func (input, output), closure

구조체 및 인터페이스

struct, json(marshall, unmarshall), method, interface

Chaincode GO SDK

자료형 및 변수

- 변수란 데이터가 저장된 메모리의 공간을 가리키는 이름
- 변수를 선언을 위해 var 을 사용
- 타입은 문장 끝에 명시

```
package main
```

```
import "fmt"
```

```
var x, y, z int  
var c, python, java bool
```

```
func main() {  
    fmt.Println(x, y, z, c, python, java)  
}
```

변수의 이름

변수의 타입

자료형



- 상수: 정해진 값 - 한번 정해지면 값을 변경할 수 없음
 - 문자, 문자열, boolean, 숫자타입 중 하나

숫자 상수

```
const Pi = 3.14
```

문자열 상수

```
func main() {  
    const World = "안녕"  
    fmt.Println("Hello", World)  
    fmt.Println("Happy", Pi, "Day")  
  
    const Truth = true  
    fmt.Println("Go rules?", Truth)  
}
```

Go의 자료형



자료형	저장범위	설명
uint8	0 ~ 255	부호 없는 8비트 정수형
uint16	0 ~ 65,535	부호 없는 16비트 정수형
uint32	0 ~ 4,294,967,295	부호 없는 32비트 정수형
uint64	0 ~ 18,446,744,073,709,551,615	부호 없는 64비트 정수형
uint		32비트 시스템에서는 uint32, 64비트 시스템에서는 uint64
int8	-128 ~ 127	부호 있는 8비트 정수형
int16	-32,768 ~ 32,767	부호 있는 16비트 정수형
int32	-2,147,483,648 ~ 2,147,483,647	부호 있는 32비트 정수형
int64	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	부호 있는 64비트 정수형
int		32비트 시스템에서는 int32, 64비트 시스템에서는 int64
float32		IEEE-754 32비트 부동소수점, 7자리 정밀도
float64		IEEE-754 64비트 부동소수점, 12자리 정밀도
complex64		float32 크기의 실수부와 허수부로 구성된 복소수
complex128		float64 크기의 실수부와 허수부로 구성된 복소수
uintptr		uint와 같은 크기를 갖는 포인터형
bool		참, 거짓을 표현하기 위한 8비트 자료형
byte		8비트 자료형
rune		유니코드 저장을 위한 자료형, 크기는 int32와 동일
string		문자열을 저장하기 위한 자료형

포인터변수

- 변수의 주소를 저장하는 변수

숫자 변수

```
var num int = 1
```

변수의 주소를 저장하는 변수

```
var numPtr *int = &num
```

```
fmt.Println(numPtr)  
fmt.Println(*numPtr)  
fmt.Println(&num)
```

변수의 주소를 이용하여 변수접근

```
// 실행 결과  
0xc0820062d0  
1  
0xc0820062d0
```

추가 자료형

- 배열/슬라이스

- 하나의 변수가 같은타입의 여러개의 값을 가지는 자료형

```
p := []int{2, 3, 5, 7, 11, 13}
```

- 구조체

- struct 는 필드(데이터)들의 조합
- 여러개의 자료형을 묶어 하나의 변수로 만들어주는 자료형

```
type Vertex struct {  
    X int  
    Y int  
}
```

- 맵

- 키와 값을 가지는 자료형

- etc....

```
var m map[string]Vertex  
m["Bell Labs"] = Vertex{ 40.68433, -74.39967, }
```

- 여러 자료형을 포함하는 프로그램 작성, 컴파일, 디버깅

```
// 파일이름:data\data.go
package main

import (
    "fmt"
    "math/cmplx"
)

var (
    ToBe bool = false
    MaxInt uint64 = 1<<64 - 1
    z complex128 = cmplx.Sqrt(-5 + 12i)
)

func main() {
    const f = "%T(%v)\n"
    fmt.Printf(f, ToBe, ToBe)
    fmt.Printf(f, MaxInt, MaxInt)
    fmt.Printf(f, z, z)
}
```



함수와 제어구조



- 함수
 - 함수 정의 (함수 이름, 함수 입력, 함수 출력)
 - 함수 호출

```
package main
```

```
import "fmt"
```

```
func add(x int, y int) int {  
    return x + y  
}
```

```
func main() {  
    fmt.Println(add(42, 13))  
}
```

함수는 매개변수(인자)를 입력으로 사용

함수는 하나 이상의 결과값을 반환

return 키워드를 사용하여 반환

함수는 다른 함수에서 호출될 수 있음

여러개의 결과를 가지는 함수

```
package main
```

```
import "fmt"
```

```
func swap(x, y string) (string, string) {  
    return y, x  
}
```

함수정의에 여러 개의 반환형 지정

함수내에서 여러 개의 값을 한꺼번에 반환

```
func main() {  
    a, b := swap("hello", "world")  
    fmt.Println(a, b)  
}
```

함수호출 후 여러 개의 값을 저장

- 여러 개의 함수반환값을 가지는 함수 작성

```
// 파일이름:func\multi_return_func.go
package main

import "fmt"

func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    a, b := swap("hello", "world")
    fmt.Println(a, b)
}
```



조건구조



- 조건수식에 따라 수행 여부를 결정짓는 구조

```
func pow(x, n, lim float64) float64 {
```

```
    if {v := math.Pow(x, n); v < lim} {
```

```
        return v
```

```
    } else {
```

```
        fmt.Printf("%g >= %g\n", v, lim)
```

```
    }
```

```
// can't use v here, though
```

```
    return lim
```

```
}
```

```
func main() {
```

```
    fmt.Println(
```

```
        pow(3, 2, 10),
```

```
        pow(3, 3, 20),
```

```
    )
```

```
}
```

조건 수식

조건에 따르는 실행문

조건에 따르지 않는 실행문

반복구조

- 조건수식에 따라 반복여부를 결정짓는 구조

```
package main
```

```
import "fmt"
```

```
func main() {  
    sum := 0
```

```
    for i := 0 ; i < 10 ; i++ {
```

```
        sum += i  
    }
```

```
    fmt.Println(sum)
```

```
}
```

초기 수식

조건 수식

증감 수식

조건에 따르는 실행문

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    sum := 1
```

```
    for {sum < 1000} {
```

```
        sum += sum
```

```
    }
```

```
    fmt.Println(sum)
```

```
}
```

조건 수식만 포함 할 수도 있음

- 조건, 반복을 포함한 프로그램 작성
- Hello World를 5번 화면에 출력하는 프로그램을 작성하시오

```
// 파일이름:control\if_for.go
package main

import "fmt"

func printhello(x int) (int) {
    for i := 0 ; i < 10 ; i++ {
        if i%2 == 0 {
            fmt.Println("Hello World")
        }
    }
    return x;
}

func main() {
    printhello(5)
}
```



문자열 및 자료구조

- 문자열
 - 바이트들의 연속적 나열 -> 텍스트
- string: 읽기 전용
 - 값 재할당 가능 / 값 편집 불가

```
var s string
s = "abc"
//or
s := "abc"
```

```
fmt.Println(s)
```


문자열 연결

- 문자열은 읽기 전용으로 수정이 아닌 두 문자열을 이어붙인 문자열을 새로 만드는 것
- “abc”와 “def” 문자열을 이어붙이려면 ‘+’ 연산을 이용

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
s := "abc"
```

```
ps := &s
```

```
s += "def"
```

```
fmt.Println(s)
```

```
fmt.Println(*ps)
```

```
}
```

문자열 선언 및 초기화

문자열 붙이기

• 문자열 함수 실행해보기

```
//파일명:str/str_func.go
package main

import (
    "fmt"
    s "strings"
)

var p = fmt.Println

func main() {
    p("Contains: ", s.Contains("test", "es"))
    p("Count: ", s.Count("test", "t"))
    p("HasPrefix: ", s.HasPrefix("test", "te"))
    p("HasSuffix: ", s.HasSuffix("test", "st"))
    p("Index: ", s.Index("test", "e"))
    p("Join: ", s.Join([]string{"a", "b"}, "-"))
    p("Repeat: ", s.Repeat("a", 5))
    p("Replace: ", s.Replace("foo", "o", "0", -1))
    p("Replace: ", s.Replace("foo", "o", "0", 1))
    p("Split: ", s.Split("a-b-c-d-e", "-"))
    p("ToLower: ", s.ToLower("TEST"))
    p("ToUpper: ", s.ToUpper("test"))
    p()
    p("Len: ", len("hello"))
    p("Char:", "hello"[1])
}
```

```
Contains: true
Count: 2
HasPrefix: true
HasSuffix: true
Index: 1
Join: a-b
Repeat: aaaaa
Replace: f00
Replace: f0o
Split: [a b c d e]
ToLower: test
ToUpper: TEST

Len: 5
Char: 101
```



배열



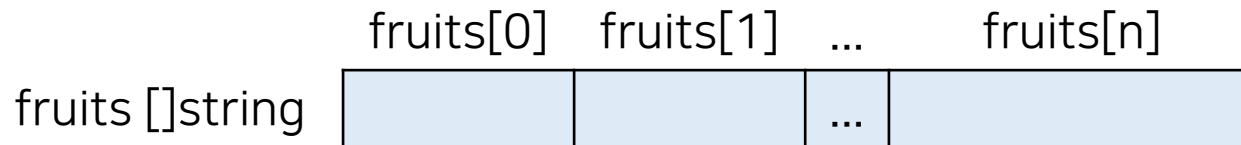
- 연속된 메모리 공간을 순차적으로 이용하는 자료구조
- 사용방법
 - `var ar [3]int`
 - 3개의 정수로 되어 있는 'ar'이라는 이름의 배열을 구성
 - `fruits := [3]string{"사과", "바나나", "토마토"}`
 - 3개의 문자열 "사과", "바나나", "토마토" 로 구성된 'fruits'라는 배열 구성
 - `fruits := [...]string{"사과", "바나나", "토마토"}`
 - 컴파일러가 배열의 개수를 알아내어 '...'을 '3'으로 생성

	fruits[0]	fruits[1]	fruits[2]
fruits [3]string	"사과 "	"바나나"	"토마토"

슬라이스



- 배열에 기초하여 만들어짐
- 배열에 비해 유연한 구조를 가지고 있어 배열보다 자주 사용됨
- 슬라이스는 길이와 용량을 갖고 길이가 변할 수 있는 구조
- 배열 사용 방법
 - `var ar []int` → 크기가 없는 배열로 보임
 - `fruits := make([]string, n)`
→ `n`개의 문자열 공간을 할당한 'fruits'라는 슬라이스 생성



슬라이스



- 슬라이스에 자료 넣기

```
fruits := make([]string, 3)
```

```
fruits[0] = “사과”
```

```
fruits[1] = “바나나”
```

```
fruits[2] = “토마토”
```

```
fruits := []string{“apple”, “banana”, “tomato”}
```

- 슬라이스 잘라내기

```
fruits[:2]
```

```
fruits = fruits[:2]
```

→ 2개를 잘라서 다시 fruits에 할당

슬라이스의 컨셉



Conceptually:

```
type Slice struct {  
    base *elemType // pointer to 0th element  
    len int        // num. of elems in slice  
    cap int        // num. of elems available  
}
```

Array:

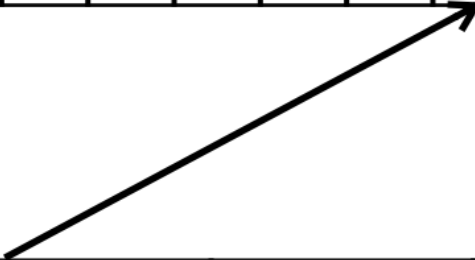
ar:

7	1	5	4	3	8	7	2	11	5	3
---	---	---	---	---	---	---	---	----	---	---

Slice:

a=ar[7:9]:

base=&ar[7]	len=2	cap=4
-------------	-------	-------



슬라이스 용량

- 슬라이스는 연속된 메모리 공간을 사용하기 때문에 용량에 제한이 있음
- 남은 용량이 없는데 덧붙이고자 하면 더 넓은 메모리 공간으로 이전
- 배열이나 슬라이스의 길이 확인 시 `len(x)` 함수 사용
- 슬라이스의 용량 확인 시 `cap(x)` 함수 사용
`fmt.Println(len(fruits))` // fruits 배열 또는 슬라이스의 길이
`fmt.Println(cap(fruits))` // fruits 슬라이스의 용량
- 용량을 미리 지정하여 슬라이스 생성하기
`nums := make([]int, 3, 5)`

슬라이스의 용량



```
package main
```

```
import "fmt"
```

```
func main() {  
    a := make([]int, 5)  
    printSlice("a", a)  
    b := make([]int, 0, 5)  
    printSlice("b", b)  
    c := b[:2]  
    printSlice("c", c)  
    d := c[2:5]  
    printSlice("d", d)  
}
```

```
func printSlice(s string, x []int) {  
    fmt.Printf("%s len=%d cap=%d %v\n",  
        s, len(x), cap(x), x)  
}
```

```
a len=5 cap=5 [0 0 0 0 0]  
b len=0 cap=5 []  
c len=2 cap=5 [0 0]  
d len=3 cap=3 [0 0 0]
```


- 슬라이스 붙이기

```
//파일:slice\slice_append.go
```

```
package main
```

```
import "fmt"
```

```
func main() {  
    f1 := []string{"사과", "바나나", "토마토"}  
    f2 := []string{"포도", "딸기"}  
    f3 := append(f1, f2...) // 이어붙이기  
    // 토마토를 제외하고 이어붙이기  
    f4 := append(f1[:2], f2...)
```

```
    fmt.Println(f1)  
    fmt.Println(f2)  
    fmt.Println(f3)  
    fmt.Println(f4)  
}
```

```
[사과 바나나 토마토]  
[포도 딸기]  
[사과 바나나 토마토 포도 딸기]  
[사과 바나나 포도 딸기]
```



- 키(Key)에 대응하는 값(Value)을 신속히 찾는 해시테이블 (Hash table)을 구현한 자료구조

- 선언

```
var m map[keyType]valueType
var m map[string]string
```

- 맵에 자료 넣기

- make()함수 사용

- 리터럴(literal)을 사용

리터럴(literal)이란 소스
코드의 고정된 값을
대표하는 용어

```
idMap = make(map[int]string)
```

```
tickers := map[string]string{
    "GOOG": "Google Inc",
    "MSFT": "Microsoft",
    "FB":   "FaceBook",
}
```

맵 - 예제



- for를 사용한 맵 열거
- Map은 unordered 이므로 출력 순서는 무작위

```
var m map[int]string
```

```
m = make(map[int]string)
```

```
//추가 혹은 갱신
```

```
m[901] = "Apple"
```

```
m[134] = "Grape"
```

```
m[777] = "Tomato"
```

```
// 키에 대한 값 읽기
```

```
str := m[134]
```

```
fmt.Println(str)
```

```
noData := m[999] // 값이 없으면 nil 혹은 zero 리턴
```

```
fmt.Println(noData)
```

```
// 삭제
```

```
delete(m, 777)
```

- 맵 정보 모두 출력하기

```
//파일:map\map.go
package main

import "fmt"

func main() {
    myMap := map[string]string{
        "A": "Apple",
        "B": "Banana",
        "C": "Charlie",
    }
    // for range 문을 사용하여 모든 맵 요소 출력
    for key, val := range myMap {
        fmt.Println(key, val)
    }
}
```

A Apple
B Banana
C Charlie



함수 (상세)



- 여러 문장을 묶어서 실행하는 코드 블록의 단위
- Go에서 함수는 **func 키워드**를 사용하여 정의
- 함수 선언 방법

```
func square(f float64) float64 { return f*f }
```

- 다수의 값 반환 가능

```
func MySqrt(f float64) (float64, bool) {  
    if f >= 0 {  
        return math.Sqrt(f), true  
    }  
    return 0, false  
}
```

함수 반환 형식 정의

함수 반환

- 반환된 값 중 일부만 사용한다면 → `_` (빈 식별자)

```
val, _ = MySqrt(foo())
```

함수 - 예제



- 반환 값 없는 함수

```
func square(f float64) {
```

- square()함수는 float형 인자 f를 가지고 있음
- 반환 값이 없으므로 별도의 반환 타입을 정의하지 않음
- return명령은 단순히 **제어권**을 함수호출한곳으로 반환

함수 - 예제



- 가변인자함수

```
func say(msg ...string) {
```

- 함수에 고정된 수의 인자를 전달하지 않고 다양한 개수의 인자를 전달하고자 할 때 사용
- 가변 인자를 나타내는 ... (3개의 마침표) 사용
- 가변 인자를 갖는 함수 호출 시 n개의 동일 타입 인자 전달

```
func main() {  
    say("This", "is", "a", "book")  
    say("Hi")  
}
```

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

```
This  
is  
a  
book  
Hi
```

range: for 반복문에서 슬라이스나 맵을 순회(iterates)할 때 사용

함수 - 예제

- 반환값 있는 함수

```
func sum(nums ...int) int { return s }
```

- 복수 개의 값을 반환 함수

- 가변 인자를 나타내는 ... (3개의 마침표) 사용
- 가변 인자를 갖는 함수 호출 시 n개의 동일 타입 인자 전달

```
func main() {  
    count, total := sum(1, 7, 3, 5, 9)
```

```
    fmt.Println(count, total)
```

```
}
```

```
func sum(nums ...int) (int, int) {
```

```
    s := 0 // 합계
```

```
    count := 0 // 요소 갯수
```

```
    for _, n := range nums {
```

```
        s += n
```

```
        count++ }
```

```
    return count, s }
```

복수개의 함수 반환값을 저장

가변인자값을 입력으로 정의

가변인자값을 함수내에서 사용

익명함수

- 함수명을 갖지 않는 함수
- 일반적으로 함수 전체를 변수에 할당하거나
다른 함수의 인자에 직접 정의되어 사용
 - 변수명이 함수명과 같이 취급
 - "변수명(인자들)" 형식으로 함수 호출

```
func main() {
```

```
    sum := func(n ...int) int {      s := 0
        for _, i := range n {
            s += i
        }
    }
```

```
    return s
```

```
}
```

```
result := sum(1, 2, 3, 4, 5)
```

```
fmt.Println(result)
```

```
}
```

익명함수 변수에 할당

익명함수 정의

익명함수 사용

일급함수



- 다른 함수의 인자로 전달
- 다른 함수의 반환 값으로 이용

```
func main() { // 변수 add 에 익명함수 할당
    add := func(i int, j int) int {
        return i + j
    }
}
```

```
r1 := calc(add, 10, 20) // add 함수 전달
fmt.Println(r1)
```

```
// 직접 첫번째 파라미터에 익명함수 정의
```

```
r2 := calc(func(x int, y int) int { return x - y }, 10, 20)
fmt.Println(r2)
}
```

```
func calc(f func(int, int) int, a int, b int) int {
    result := f(a, b)
    return result
}
```

함수 원형 정의

- type문
 - 구조체(struct), 인터페이스 등 Custom Type (혹은 User Defined Type) 정의
 - 또한 함수 원형을 정의하는데 사용 가능
 - 변수명이 함수명과 같이 취급
 - "변수명(인자들)" 형식으로 함수 호출

// 원형 정의

```
type calculator func(int, int) int
```

// calculator 원형 사용

```
func calc(f calculator, a int, b int) int {  
    result := f(a, b)  
    return result  
}
```

클로저 (closure)



- 함수 바깥에 있는 변수를 참조하는 함수값(function value)을 의미
- 외부의 변수를 마치 함수 안으로 끌어들이는 듯이 그 변수를 읽거나 쓸 수 있게 함

```
func main() {  
    f := adder()  
    fmt.Print(f(1))  
    fmt.Print(f(20))  
    fmt.Print(f(300))  
}
```

- 함수는 클로저(full closures)
- 코드에서 adder 함수는 클로저(closure)를 반환
- 각각의 클로저는 자신만의 x변수를 가짐

```
func adder() func(int) int {  
    var x int  
    return func(delta int) int {  
        x += delta  
        return x  
    }  
}
```

- 함수 코드 실행해 보기

```
//파일명:func/func.go
package main
import "fmt"
// return이 없는 경우
func printAdd(x int, y int) {
    fmt.Println(x + y) }

// return 값이 하나인 경우
func add(x int, y int) int {
    return x + y }

// return 값이 두 개 이상인 경우
func addAndMultiply(x int, y int) (int, int) {
    return x + y, x * y }

func main() {
    fmt.Println(add(42, 13))
    printAdd(42, 13)
    fmt.Println(addAndMultiply(42, 13)) }
```



구조체



- Custom Data Type을 표현하는데 사용
- 필드들의 모음 또는 묶음
- 배열과의 비교
 - 배열 : 서로 같은 자료형의 자료를 묶어놓은 것
 - 구조체 : 서로 다른 자료형의 자료들도 묶을 수 있는 것
- 구조체 사용법

```
type person struct {  
    name string  
    age  int  
}
```

구조체 객체 생성

- 빈 객체 생성 후 필드 값 채우기

```
p := person{}  
p.name = "Lee"  
p.age = 10
```

- 객체 생성 시 필드 값 할당

```
p2 := person{name: "Sean", age: 50}
```

- new() 사용
 - new()를 사용하면 모든 필드를 Zero value로 초기화
 - person 객체의 포인터(*person) 리턴

```
p := new(person)  
p.name = "Lee"
```

• 구조체 예제 작성

```
//파일명:struct/struct.go
package main
import "fmt"
// struct 정의
type person struct {
    name string
    age int
}
func main() {
    // 빈 객체 생성 후 필드값 채우기
    p := person{}
    p.name = "Lee"
    p.age = 10

    // 객체 생성 시 필드값 할당
    p2 := person{name:"Sean", age: 50}

    // new() 사용하여 객체 만들기
    p3 := new(person)
    p3.name = "Lee" // p3가 포인터라도 . 을 사용한다

    fmt.Println(p)
    fmt.Println(p2)
    fmt.Println(p3)
}
```



JSON



- 직렬화란
 - 객체의 상태를 보관이나 전송 가능한 상태로 변환 하는 것
 - 직렬화 가능한 요소
 - 구조체,
 - 숫자, 문자열과 같은 다양한 데이터 타입의 변수 또는 상수
 - 배열, 맵
 - 사용 분야
 - 저장 및 불러오기
 - 네트워크를 통한 메시지 전송
(ex. RPC(Remote Procedure Call))

JSON



- (JavaScript Object Notation)
- 자료 교환 형식 중 하나
- 자바스크립트에서 객체를 표현하는 방식과 비슷함
- XML에 비하여 **사람이 읽기 쉽고 간단하기 때문에** 현재 많이 사용 됨

```
{  
  "Title": "Laundry",  
  "Status": 2,  
}
```

‘Fabcar’ 예제에서 JSON

- import ("encoding/json")
 - JSON을 읽고 쓰기 때문에 import 시킴
- json.Marshal()
 - `carAsBytes, _ = json.Marshal(car)`
 - car 구조체의 데이터를 JSON 형식으로 저장
 - ‘_’으로 에러는 처리하지 않음
- json.Unmarshal()
 - `json.Unmarshal(carAsBytes, &car)`
 - 전달받은 ‘carAsBytes’라는 JSON형식의 데이터를 car 구조체 안의 값으로 채움

JSON 태그



- 기존 구조체 필드 이름을 JSON에서 다른 이름으로 사용하고 싶을 때 사용
- JSON에서 필드를 나열하고 싶지 않을 때
 - ex. Internal string 'json:"-'"

```
// Define the car structure, with 4 properties.  
Structure tags are used by encoding/json library  
type Car struct {  
    Make string `json:"make"`  
    Model string `json:"model"`  
    Colour string `json:"colour"`  
    Owner string `json:"owner"`  
}
```

- JSON 예제 작성

```
//파일명:json/json.go
package main

import (
    "encoding/json"
    "fmt"
    "log"
)

type Task struct {
    Title string
    Status status
}

type status int

const (
    UNKNOWN status = iota
    TODO
    DONE
)
```

```
func main() {
    ExampleTask_marshalJSON()
    ExampleTask_unmarshalJSON()
}

func ExampleTask_marshalJSON() {
    t := Task{
        "Laundry",
        DONE,
    }
    b, err := json.Marshal(t)
    if err != nil {
        log.Println(err)
        return
    }
    fmt.Println(string(b))
}

func ExampleTask_unmarshalJSON() {
    b := []byte(`{"Title":"Buy Milk","Status":2}`)
    t := Task{}
    err := json.Unmarshal(b, &t)
    if err != nil {
        log.Println(err)
        return
    }
    fmt.Println(t.Title)
    fmt.Println(t.Status)
}
```

메서드



- 객체 지향 프로그래밍 언어(OOP)
 - 클래스 내부에서 정의된 함수 (객체의 함수)를 메소드
 - 타 언어 OOP의 클래스가 필드와 메서드를 함께 가짐
- Go에서의 메서드
 - 구조체가 필드만을 가지며, 메서드는 별도로 분리되어 정의
 - 메서드는 함수 정의에서 func 키워드와 함수명 사이에 **그 함수가 어떤 구조체를 위한 메서드인지** 표시
- 메서드 사용법

```
type Point struct { x, y float64 }  
  
func (p *Point) Abs() float64 {  
    return math.Sqrt(p.x*p.x + p.y*p.y)  
}
```

Value vs 포인터 receiver

- Value receiver

- struct의 데이터를 복사(copy)하여 전달
- 메서드 내에서 그 구조체의 필드값이 변경되더라도 호출자의 데이터는 변경되지 않음

```
func (p Point) Abs() float64 {
```

- 포인터 receiver

- 구조체의 포인터만 전달
- 메서드 내의 필드값 변경이 그대로 호출자에서 반영

```
func (p *Point) Abs() float64 {
```

- fabcar예제

```
type SmartContract struct {}
```

```
func (s *SmartContract) queryAllCars(APIStub  
    shim.ChaincodeStubInterface) sc.Response {
```

- JSON 예제 작성

```
//파일명:method/method.go
package main

import "fmt"

type Rect struct { //Rect - struct 정의
    width, height int
}
func (r Rect) area() int { //Rect의 area() 메소드
    return r.width * r.height
}
func (r *Rect) area2() int { // 포인터 Receiver
    r.width++
    return r.width * r.height
}
func main() {
    rect := Rect{10, 20}
    area1 := rect.area() //메서드 호출
    area2 := rect.area2() //메서드 호출
    fmt.Println(area1)
    fmt.Println(rect.width, area2)
}
```



인터페이스



- 구조체 : 필드들의 집합체
- 인터페이스 : 메서드들의 집합체
- 인터페이스는 type이 구현해야 하는 메서드 원형들 정의
- 인터페이스 정의
 - 구조체와 유사한 구조

```
type Shape interface {  
    area() float64  
    perimeter() float64  
}
```

인터페이스 구현



- 해당 타입이 그 인터페이스의 메서드들을 모두 구현
- Shape 인터페이스를 구현 하기 위하여 구현 할 메서드

`area()`

`perimeter()`

```
type Rect struct { //Rect 정의
    width, height float64
}
type Circle struct { //Circle 정의
    radius float64
}
//Rect 타입에 대한 Shape 인터페이스 구현
func (r Rect) area() float64 {
    return r.width * r.height
}
func (r Rect) perimeter() float64 {
    return 2 * (r.width + r.height)
}
//Circle 타입에 대한 Shape 인터페이스 구현
func (c Circle) area() float64 {
    return math.Pi * c.radius * c.radius
}
func (c Circle) perimeter() float64 {
    return 2 * math.Pi * c.radius
}
```

인터페이스 사용



- 함수가 인자로 인터페이스 사용

```
func main() {  
    r := Rect{10., 20.}  
    c := Circle{10}  
  
    showArea(r, c)  
}  
  
func showArea(shapes ...Shape) {  
    for _, s := range shapes {  
        a := s.area() //인터페이스 메서드 호출  
        fmt.Println(a)  
    }  
}
```

인터페이스 타입



- 빈 인터페이스(empty interface)

- 표준패키지들의 함수 Prototype

```
func Marshal(v interface{}) ([]byte, error);  
func Println(a ...interface{}) (n int, err error);
```

- 메서드를 전혀 갖지 않는 빈 인터페이스
- 어떠한 타입도 담을 수 있는 컨테이너 (Dynamic Type)

```
func main() {  
var x interface{}  
    x = 1  
    x = "Tom"  
    printIt(x)  
}
```

```
func printIt(v interface{}) {  
    fmt.Println(v) //Tom  
}
```

- 체인코드 ChaincodeStubInterface 이해하기
 - fabcar 의 체인코드 fabcar.go 열어보기
 - fabcar구조체 살펴보기
 - fabcar인터페이스 살펴보기

```
// SimpleAsset implements a simple chaincode to manage an asset
```

```
type SimpleAsset struct {  
}
```

```
// Init is called during chaincode instantiation to initialize any data.
```

```
func (t *SimpleAsset) Init(stub  
shim.ChaincodeStubInterface) peer.Response {  
  
}
```



에러 처리

- 내장 타입으로 error 라는 인터페이스가 있어 커스텀 에러 타입을 구현할 수 있음
- log 패키지의 Fatal, Panic, Print 등의 함수를 이용해 에러 처리 가능

```
package main
```

```
import (  
    "log"  
    "os"  
)
```

```
func main() {
```

```
    // os.Open 함수는 첫번째로 파일 포인터, 두번째로 error 인터페이스를 리턴  
    f, err := os.Open("1.txt")
```

```
    //에러 체크
```

```
    if err != nil {  
        //Fatal 함수는 메시지를 출력후 os.Exit(1)를 호출하여 프로그램 종료  
        log.Fatal(err.Error())  
    }
```

```
    //에러 발생시 실행되지 않음
```

```
    println(f.Name())  
    println("end")  
}
```

```
comrry3@i-1:~/goex$ go run err.go  
2019/07/04 12:11:43 open 1.txt: no such file or directory  
exit status 1
```

Chaincode GO SDK



- <https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim>
- <https://godoc.org/github.com/hyperledger/fabric/protos/peer>

shim

ChaincodeStub

GetFunctionAndParameters()

GetArgs()

GetState()/PutState()/DelState()

GetPrivateData()/PutPrivateData()

DelPrivateData()

GetTransient()

Error()/Sucess()

peer

type Response