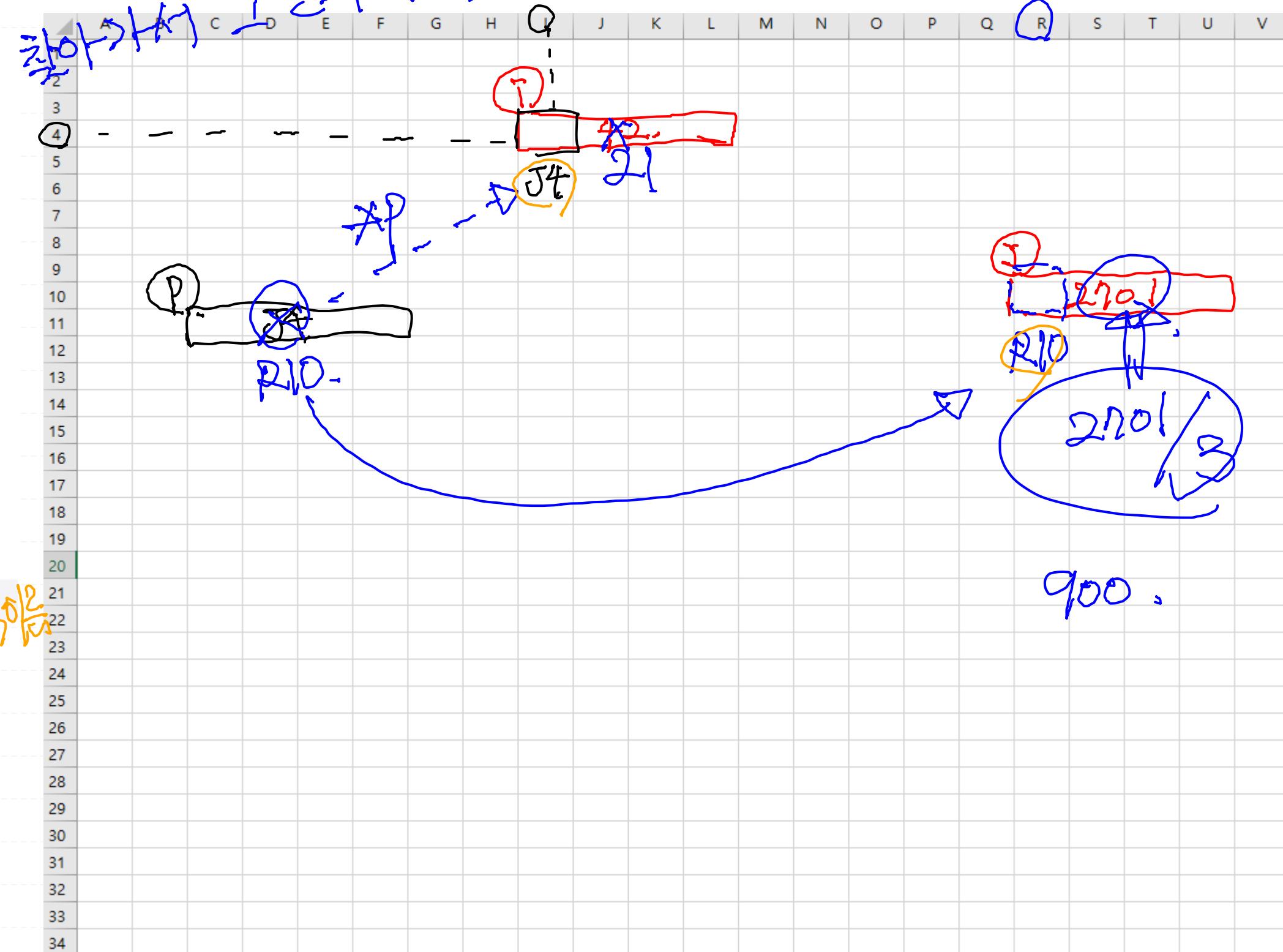


11. 포인터

```
1 import "fmt"
2
3 func main() {
4     i, j := 42, 2701
5
6     p := &i // i를 가리키는 포인터
7     fmt.Println(*p) // 포인터를 통해 i 값을 읽습니다.
8     *p = 21 // 포인터를 통해 i 값을 설정합니다
9     fmt.Println(i)
10
11    p = &j // j를 가리킵니다.
12    *p = *p / 37 // 포인터를 통해 j를 나눕니다.
13    fmt.Println(j)
14 }
```

4. 4. $\star p \rightarrow J4$ 를
4. 4. 주5. T의 대상 주5.

P → J4
*P → J4 을 찾아가기 고민의 경지,



var a int

a=10

- - - - -

var a int =10

a := 10

var a, b int

a=10

b=20.

- - - - -

var a,b int = 10, 20

a, b := 10, 20

[데이터]의 표현 \Rightarrow 변수.

ex) 기. 물류제 등...

[데이터] 그룹 \Rightarrow 배열

ex) 학급 성적표 \Rightarrow 같은 타입.

다른 종류의 [데이터] 그룹 \Rightarrow 구조체. \Rightarrow ex) 상품.

12. 구조체

```
1 package main
2
3 import "fmt"
4
5 type Marine struct {
6     name string
7     hp    int
8     speed int
9     dps   int
10 }
11
12 func main() {
13     // 구조체 변수 선언 및 할당
14     var m1 Marine
15     fmt.Println("이름: ", m1.name, " 체력: ", m1.hp, " 이동속도: ", m1.speed, " 공격력: ", m1.dps)
16
17     m1.name = "해병1"
18     m1.hp = 40
19     m1.speed = 1
20     m1.dps = 5
21     fmt.Println("이름: ", m1.name, " 체력: ", m1.hp, " 이동속도: ", m1.speed, " 공격력: ", m1.dps)
22
23     // 구조체 변수 선언 할당 동시 처리 --> 할당되지 않는 값은 0으로 초기화
24     //var m2 Marine = Marine{"해병2", 40, 1, 5}
25     m2 := Marine{"해병2", 40, 1, 5}
26     fmt.Println("이름: ", m2.name, " 체력: ", m2.hp, " 이동속도: ", m2.speed, " 공격력: ", m2.dps)
27 }
```

구조체 선언 ⇒ 복합적인 데이터 타입

구조체 초기화

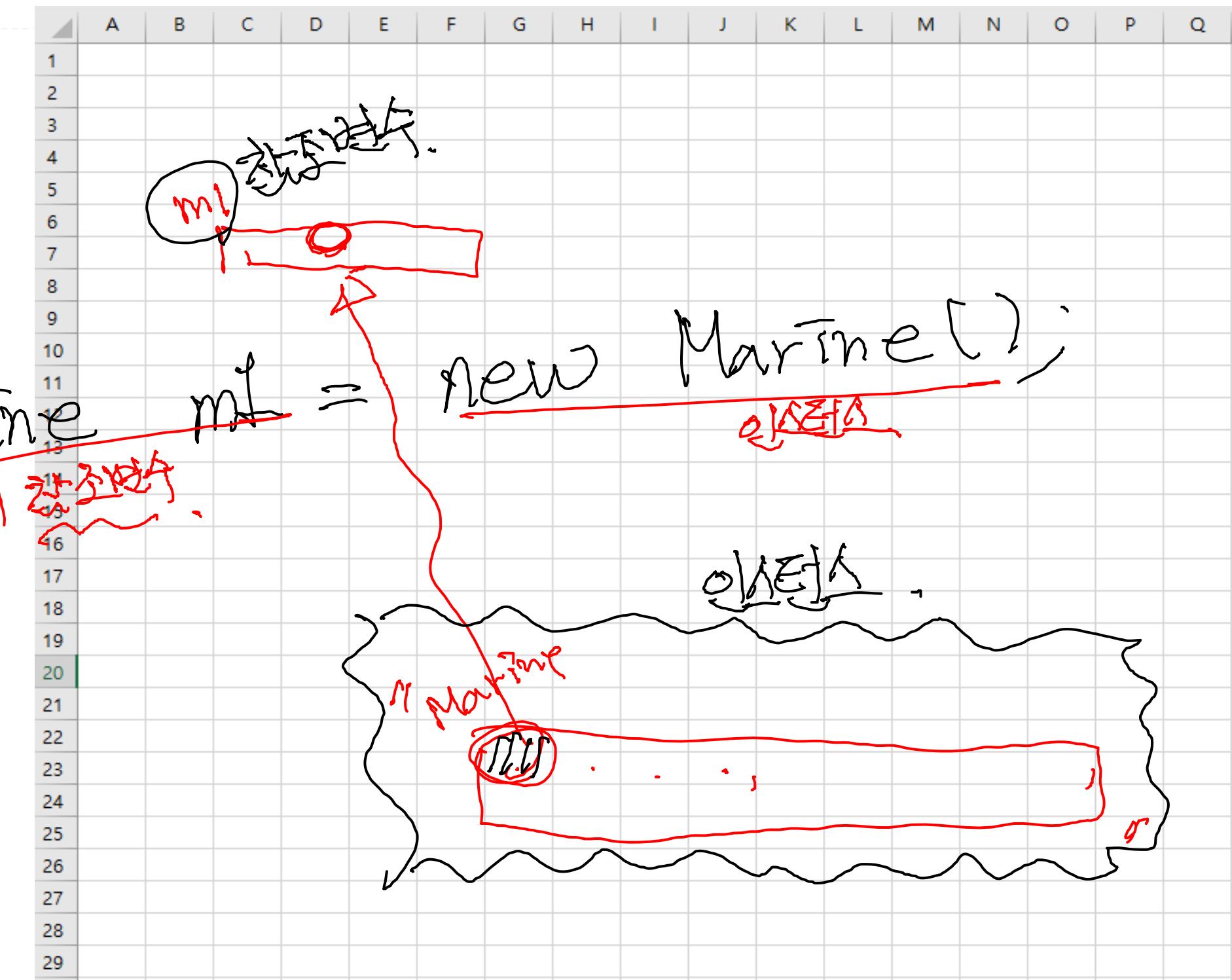
선언 ⇒ 기본값을 갖고 있는 생성이 완료됨
⇒ 멤버변수에 초기값 부여

실제 사용 예제

```

1 package main
2
3 import "fmt"
4
5 type Marine struct {
6     name string  $\Rightarrow$  8byte.
7     hp int  $\Rightarrow$  4
8     speed int  $\Rightarrow$  4
9     dps int  $\Rightarrow$  4
10 }
11
12 func main() {
13     // 구조체 변수에 대한 포인터 선언
14     var m1 *Marine
15
16     // 구조체 변수에 대한 포인터 할당
17     m1 = &Marine{"마린", 40, 1, 5}
18
19     // 출력
20     fmt.Println("마린 이름:", m1.name, "HP:", m1.hp, "속도:", m1.speed, "DPS:", m1.dps)
21 }

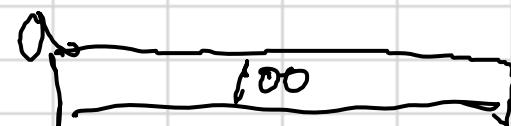
```



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1																										
2																										
3																										
4																										
5																										
6																										
7																										
8																										
9																										
10																										
11																										
12																										
13																										
14																										
15																										
16																										
17																										
18																										
19																										
20																										
21																										
22																										
23																										
24																										
25																										
26																										
27																										
28																										
29																										
30																										
31																										
32																										
33																										

int a = 100

Java, C, Go ...



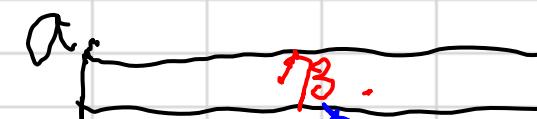
int b = a



b++

let a = 100 / a = 100

JS - Python.



let b = a



b++

13. 슬라이스 \Rightarrow 동적배열 \Rightarrow ArrayList.

문[문의 일부를 잘라서 창조시킨.]

```

3 func main() {
4     names := [4]string{
5         "John", 0
6         "Paul", 1
7         "George", 2
8         "Ringo", 3
9     }
10    fmt.Println("배열 names:", names)
11    fmt.Println("①슬라이스 선언")
12    // 슬라이스 선언방법
13    // ① 일반적인 선언방법 : 변수 선언과 비슷합니다. 슬라이스타입은 []type입니다.
14    var s1 []string = names[0:3]
15    // ② 슬라이스도 var 키워드와 타입 명시를 생략할 수 있습니다.
16    s2 := names[0:2]
17
18    fmt.Println("names[0:3]:", s1) [John Paul George]
19    fmt.Println("names[0:2]:", s2) [John Paul]
20
21    // s1에서 값을 바꾸면 names, s1에서도 같은 값을 볼 수 있습니다.
22    fmt.Println("②슬라이스로 값 변경")
23    fmt.Println("s1[0]", s1[0])
24    s1[0] = "XXX"
25    fmt.Println("s1[0] = XXX 실행 후 s1:", s1)
26    fmt.Println("s1[0] = XXX 실행 후 s2:", s2)
27    fmt.Println("s1[0] = XXX 실행 후 names:", names)
28
29    s2 = s1[0:2]
30    fmt.Println("s2 = s1[0:2] 실행 후 s2:", s2)
31
32 }
```

{ 일반적인 문자열 배열.

배열 names: [John Paul George Ringo]
xxx

// ① 일반적인 선언방법 : 변수 선언과 비슷합니다. 슬라이스타입은 []type입니다.

// ② 슬라이스도 var 키워드와 타입 명시를 생략할 수 있습니다.

names[0:3]: [John Paul George]
names[0:2]: [John Paul]

// s1에서 값을 바꾸면 names, s1에서도 같은 값을 볼 수 있습니다.

②

③

④

⑤

⑥

⑦

⑧

⑨

⑩

⑪

⑫

⑬

⑭

⑮

⑯

⑰

⑱

⑲

⑳

㉑

㉒

㉓

㉔

㉕

㉖

㉗

㉘

㉙

㉚

㉛

㉜

㉝

㉞

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

㉟

var a [3]bool { true, true, false } \Rightarrow 비[3]

var ~~b~~ []bool { true, true, false } \Rightarrow 숫자[3]

가짜[3]

b = append(b, false)

\Rightarrow { true, true, false, false }

Java ArrayList & Python

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 class Product {
5     String name;
6     int price;
7     String color;
8     String size;
9 }
10
11 public class Hello {
12     public static void main(String[] args) {
13         List<Product> products = new ArrayList<Product>();
14
15         Product product1 = new Product();
16         product1.name = "T-shirt";
17         product1.price = 100;
18         product1.color = "Red";
19         product1.size = "M";
20         products.add(product1);
21
22         Product product2 = new Product();
23         product2.name = "Pants";
24         product2.price = 200;
25         product2.color = "Blue";
26         product2.size = "L";
27         products.add(product2);
28     }
29 }
```

```
1 import "fmt"
2
3 type Product struct {
4     name string
5     price float64
6     color string
7     size string
8 }
9
10 func main() {
11     p := []Product{
12         {"Shoes", 100, "Red", "M"},
13         {"Shirt", 50, "Blue", "S"},
14         {"Pants", 70, "Black", "L"},
15     }
16
17     p = append(p, Product{"Hat", 20, "White", "S"})
18     p = append(p, Product{"Socks", 5, "Green", "M"})
19     p = append(p, Product{"Gloves", 10, "Yellow", "L"})
20 }
```

→
Shop 스토어 품목 } ↗ (20)
len() 품목 수
size := len(p)

13. range

슬라이스나 배열을 탐색하는 기법 --> 반복문에서 사용함

```
1 import "fmt"
2
3 var pow = []int{1, 2, 4, 8, 16, 32, 64, 128} ← 슬라이스
4
5 func main() {
6     //① 일반적인 range
7     fmt.Println("① 일반적인 range")
8     for i, v := range pow {
9         fmt.Printf("2**%d = %d\n", i, v)
10    }
11
12     //② 인덱스가 필요없는 경우 _로 비워둘 수 있습니다.
13     fmt.Println("② 인덱스가 필요없는 경우")
14     for _, v := range pow {
15         fmt.Println(v)
16    }
17 }
```

size := len(pow)

for i:=0; i<size; i++ {
 pow[i]

3

14. Map

```
1 import "fmt"
2
3 type Vertex struct {
4     Lat, Long float64
5 }
6
7 func main() {
8     //① map 사용
9     //map[string] 타입 변수 선언
10    var mymap map[string]Vertex
11    //make()로 맵 생성
12    mymap = make(map[string]Vertex) ← 빨강줄
13    mymap["Bell Labs"] = Vertex{
14        40.68433, -74.39967,
15    }
16    fmt.Println("① mymap[\"Bell Labs\"]: ", mymap["Bell Labs"])
17    //② map literal 사용
18    var mymap_literal = map[string]Vertex{
19        "Bell Labs": Vertex{
20            40.68433, -74.39967,
21        },
22        "Google": Vertex{
23            37.42202, -122.08408,
24        },
25    }
26    fmt.Println("② mymap_literal[\"Bell Labs\"], mymap_literal[\"Bell Labs\"]")
27
28 }
```

key . value.

Map [key] type] valuetype

{ 40.68433, -74.39967 }

mymap["Bell Labs"]

```
1 import "fmt"
2
3 func main() {
4     m := make(map[string]int)
5
6     //① key-value 지정하기
7     m["Answer"] = 42
8     fmt.Println("m[\"Answer\"]값은:", m["Answer"])
9
10    //② key-value 삭제하기
11    delete(m, "Answer")
12    fmt.Println("m[\"Answer\"]값은", m["Answer"])
13
14    //③ key존재 확인하기
15    v, ok := m["Answer"]
16    fmt.Println("m[\"Answer\"]값은", v, "존재하나요?", ok)
17 }
```

$m \leftarrow \{ \cancel{\text{Answer}} : \cancel{42} \}$

42

~~0~~

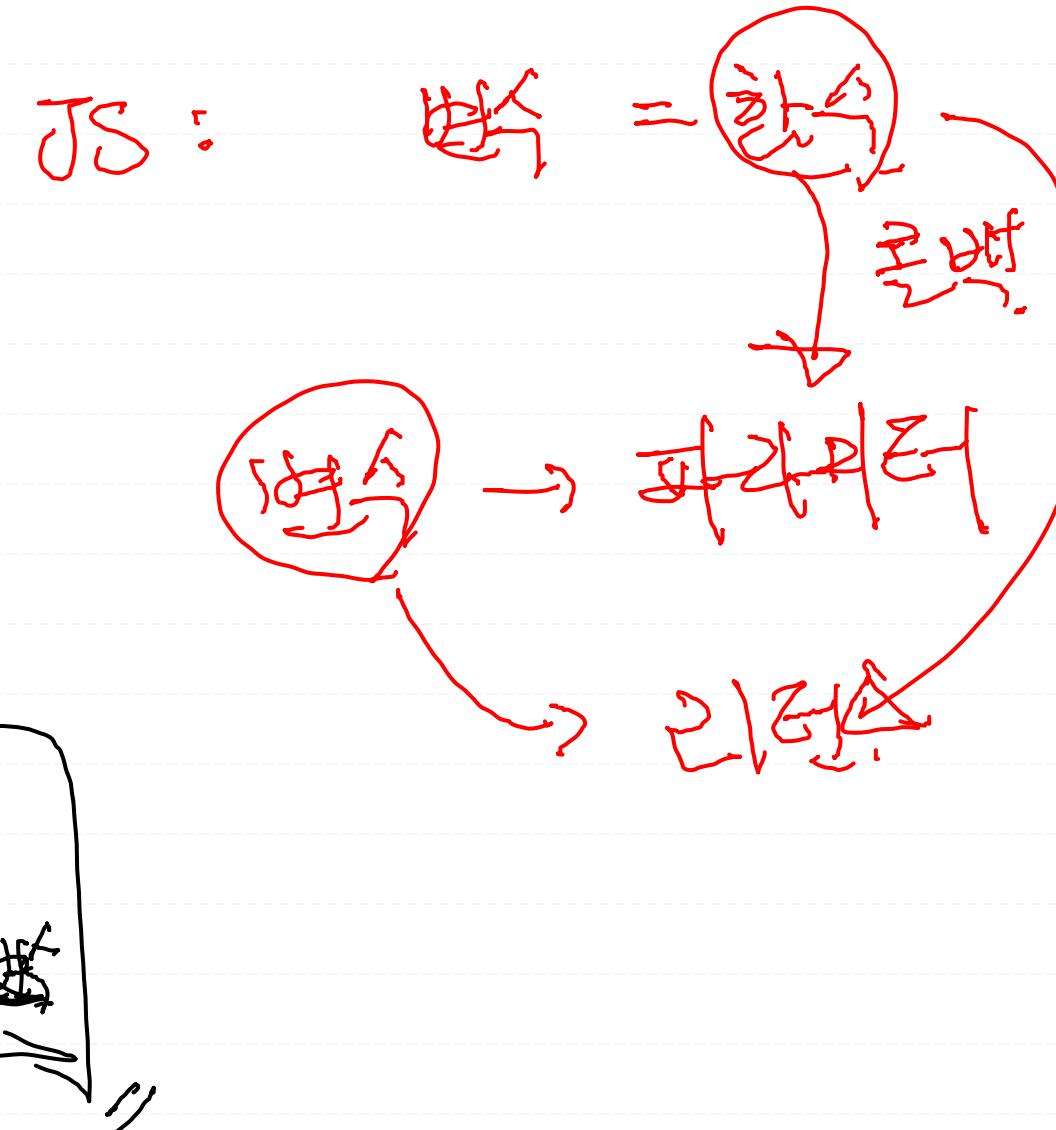
,

~~0~~

false.

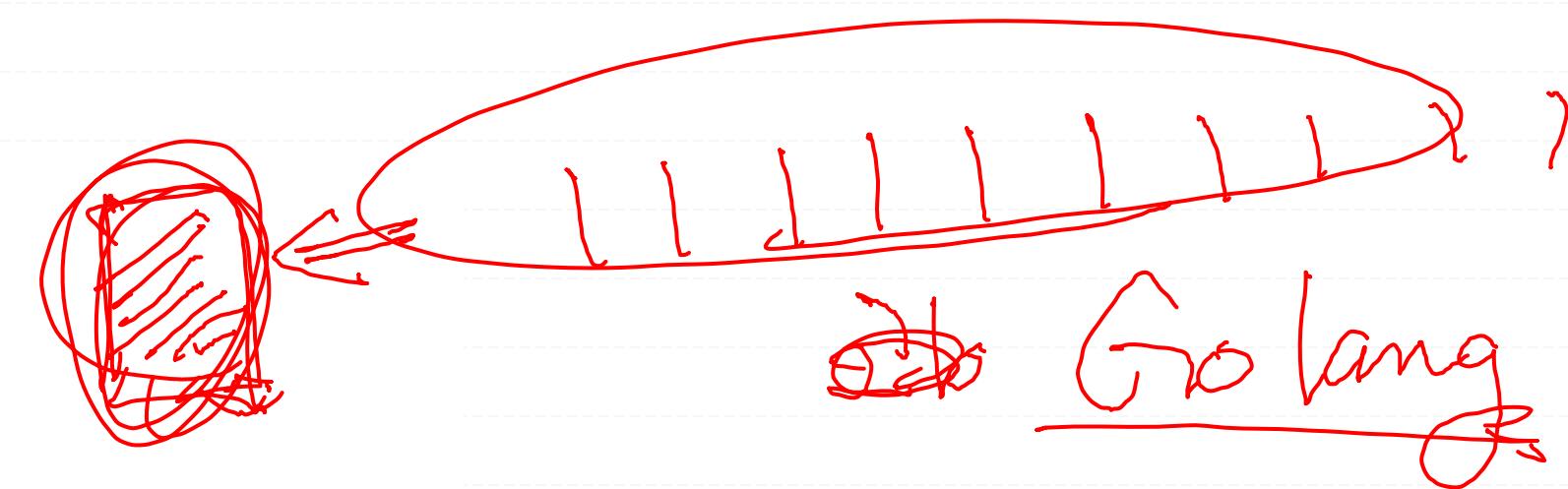
15. 클로저 --> 리턴값으로 사용되는 함수

```
1 import (
2     "fmt"
3 )
4
5 func adder() func(int) int {
6     sum := 0
7     return func(x int) int {
8         sum += x
9         return sum
10    }
11 }
12
13 func main() {
14     // pos, neg는 서로 다른 변수 sum을 가집니다.
15     pos, neg := adder(), adder()
16     for i := 0; i < 10; i++ {
17         fmt.Println(
18             i, ":",
19             pos(i),
20             neg(-2*i),
21         )
22     }
23 }
```



16. 고루틴 (비동기처리)

```
1 import (
2     "fmt"
3     "time"
4 )
5
6 func say(s string) {
7     for i := 0; i < 5; i++ {
8         time.Sleep(100 * time.Millisecond) ←
9         fmt.Println(s)
10    }
11 }
12
13 func main() {
14     go say("② 다른 루틴")
15     say("① 이 루틴")
16 }
```



동시성 프로그래밍

Java \Rightarrow Thread

JS \Rightarrow Promise
async await