



목차

- 함수
- 매개변수와 리턴값 사용하기
- 익명함수
- 클로저

함수

- 서브루틴
 - 코드의 덩어리를 만든 다음, 그것을 호출하고 귀한할 수 있는 구조
 - 코드 재사용 -> 중복 코드 감소
 - 내부와 외부 분리 -> 코드 추상화 및 단순화
 - Golang에서는 이러한 서브루틴이 함수
 - 서브프로그램, 프로시저, 메서드, 호출 가능 객체 라고도 불림

함수

- 함수의 기본 구조
 - 스택 구조로 이뤄져 있음
 - 인자를 활용해 값을 넘겨줄 수 있고, 돌려받을 수 있음
 - 이를 활용해 기본적인 모듈화 가능
 - 내부 구조를 몰라도, 입력 파라미터와, 출력 값만으로 프로그래밍 가능
- Golang은 Call by Value형식의 함수를 지원 (함수만 지원)
 - 함수 내에서 받은 값을 변경하더라도, 기존 값에 영향이 없음
 - 포인터를 사용해 값 변경 가능

함수

- 주의 키워드
 - 중괄호 위치
- 함수의 위치는 어디에 있어도 상관 없음
- 함수 타입 지정할 필요 없음
 - Ex) int, void, etc...

```
myFunc.go X
workspace_go > src > second_day > myFunc.go > ...
1 package main
2
3 import "fmt"
4
5 func test() {
6     fmt.Println("hi")
7 }
8
9 func main() {
10     test()
11 }
12

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
API server listening at: 127.0.0.1:9405
hi
```

함수

```
workspace_go > src > second_day > -go myFunc.go > {} main > main
1  package main
2
3  import "fmt"
4
5  func sum(a, b int) int {
6      return a + b
7  }
8
9  func main() {
10     fmt.Println(sum(1, 2))
11 }
12
```

```
workspace_go > src > second_day > -go myFunc.go > ...
1  package main
2
3  import "fmt"
4
5  func sum(a, b int) (result int) {
6      result = a + b
7      return
8  }
9
10 func main() {
11     fmt.Println(sum(2, 4))
12 }
13
```

함수

```
1 package main
2
3 import "fmt"
4
5 func cal(a, b int) (x, y, z int) {
6     x = a + b
7     y = a - b
8     z = a * b
9     return
10 }
11
12 func main() {
13     result1, result2, result3 := cal(4, 5)
14     fmt.Println(result1, result2, result3)
15 }
16
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL



API server listening at: 127.0.0.1:13864
9 -1 20
Process exiting with code: 0

```
workspace-go / src / second_day / myfunc.go (1) main
1 package main
2
3 import "fmt"
4
5 func cal(a, b int) (x, y, z int) {
6     x = a + b
7     y = a - b
8     z = a * b
9     return
10 }
11
12 func main() {
13     result1, _, result3 := cal(4, 5)
14     fmt.Println(result1, result3)
15 }
16
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

API server listening at: 127.0.0.1:35182
9 20
Process exiting with code: 0

함수

```
workspace_go > src > second_day > -go myFunc.go > {} main >   
1 package main  
2  
3 import "fmt"  
4  
5 ▾ func cal(a, b int) (int, int, int)   
6 |     x := a + b  
7 |     y := a - b  
8 |     z := a * b  
9 |     return x, y, z  
10 |  
11  
12 ▾ func main() {  
13 |     result1, _, result3 := cal(4, 5)  
14 |     fmt.Println(result1, result3)  
15 | }  
16
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

API server listening at: 127.0.0.1:31477

9 20

Process exiting with code: 0

```
workspace_go > src > second_day > -go myFunc.go > {} m  
1 package main  
2  
3 import "fmt"  
4  
5 ▾ func cal(temp ...int) int {  
6 |     result := 0  
7 |     for _, val := range temp {  
8 |         result += val  
9 |     }  
10 |     return result  
11 | }  
12  
13 ▾ func main() {  
14 |     result1 := cal(4, 5, 1, 3)  
15 |     fmt.Println(result1)  
16 | }  
17
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

API server listening at: 127.0.0.1:8031

13

Failed to get state - Process 5124 has exited with

함수

- Type 앞의 ... (가변인자)는 순차적으로 데이터를 받음
- 변수 뒤 ... (가변인자)는 순차적으로 데이터를 보냄

```
1 package main
2
3 import "fmt"
4
5 func cal(temp ...int) int {
6     result := 0
7     for _, val := range temp {
8         result += val
9     }
10    return result
11 }
12
13 func main() {
14     buff := []int{4, 3, 2, 5, 6}
15     result1 := cal(buff...)
16     fmt.Println(result1)
17 }
18
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

API server listening at: 127.0.0.1:20514
20
Process exiting with code: 0

함수

- 재귀함수
- 조심히 사용해야 함
- Stack, Memory 문제

```
workspace_go > src > second_day > -go myFunc.go > {  
1  package main  
2  
3  import "fmt"  
4  
5  func factorial(n int) int {  
6      if n == 0 {  
7          return 1  
8      }  
9      return n * factorial(n-1)  
10 }  
11  
12 func main() {  
13     fmt.Println(factorial(5))  
14 }  
15  
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
API server listening at: 127.0.0.1:35118  
120  
Process exiting with code: 0
```

익명 함수

- 이름이 없는 함수
- Golang에서는 Goroutine을 사용하기 위해 자주 사용됨
- 보통 변수에 넣어서 초기화 후 사용
- 그렇다면, 왜? 익명함수를 사용하는가

익명 함수

- 함수 작성의 단점
 - 함수 선언 자체가 프로그래밍 전역으로 초기화 되면서 메모리를 할당함
 - 기능을 수행할 때마다 함수를 찾아 호출해야 함
 - 모든 기능을 함수로 구현하면, 프로그램이 복잡해짐
- 이러한 단점들을 보완하기 위해 익명함수 사용
 - 특정 기능만 간단히 수행할 수 있는 기능 구현
 - 반복하지 않는 기능을 구현

익명 함수

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     sum := func(n ...int) int {
7         s := 0
8         for _, i := range n {
9             s += i
10        }
11        return s
12    }
13
14    result := sum(1, 2, 3, 4, 5)
15    fmt.Println(result)
16 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
API server listening at: 127.0.0.1:12200
15
Process exiting with code: 0
```

workspace_go > src > second_day > myFunc.go > ...

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     func() {
7         fmt.Println("hi")
8     }()
9
10    func(s string) {
11        fmt.Println(s)
12    }("test")
13
14    r := func(a, b int) (result int) {
15        result = a + b
16        return
17    }(1, 5)
18
19    fmt.Println(r)
20 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
API server listening at: 127.0.0.1:16202
hi
test
6
Process exiting with code: 0
```

익명 함수

```
workspace_go > src > second_day > go myFunc.go > {} main > main
1  package main
2
3  import "fmt"
4
5  func main() {
6      add := func(i int, j int) int {
7          return i + j
8      }
9
10     r1 := calc(add, 10, 20)
11     fmt.Println(r1)
12
13     r2 := calc(func(x int, y int) int { return x - y }, 10, 20)
14     fmt.Println(r2)
15
16 }
17
18 func calc(f func(int, int) int, a int, b int) int {
19     result := f(a, b)
20     return result
21 }
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
API server listening at: 127.0.0.1:35046
30
-10
Process exiting with code: 0
```

클로저

- 바깥 함수가 변수와 클로저 함수를 에워싸고(close over) 있다고 해서 Closure라고 명칭 됨
- 함수 안에서 함수를 선언 및 정의
- 함수 밖에 있는 변수 접근 가능
- 클로저에서 사용 된 지역변수는 메모리를 유지할 수 있음

클로저

```
workspace_go > src > second_day > -go myFunc.go > {} main > main
1  package main
2
3  import "fmt"
4
5  func main() {
6      a, b := 3, 5
7      f := func(x int) int {
8          return a*x + b
9      }
10
11     y := f(5)
12     fmt.Println(y)
13 }
14
```

DEBUG CONSOLE ... Filter (e.g. text, !exclude)

API server listening at: 127.0.0.1:2865
20
Process exiting with code: 0

```
workspace_go > src > second_day > -go myFunc.go > ...
1  package main
2
3  import "fmt"
4
5  func test() func() int {
6      i := 0
7      return func() int {
8          i++
9          return i
10     }
11 }
12 func main() {
13     var f = test()
14     fmt.Println(f())
15     fmt.Println(f())
16     fmt.Println(f())
17 }
```

DEBUG CONSOLE ... Filter (e.g. text, !exclude)

API server listening at: 127.0.0.1:25401

1
2
3

지연 호출

- 현재 함수가 끝나기 직전 실행
- 다른 언어의 finally 구문과 비슷함
- Defer라는 키워드를 사용
- 보통 익명 함수와 같이 사용
- Goroutine, Database, Io 환경에서 필수적으로 사용 됨

지연 호출

```
workspace_go > src > second_day > go myFunc.go > {}  
1 package main  
2  
3 import "fmt"  
4  
5 func main() {  
6     defer func() {  
7         fmt.Println("main end")  
8     }()  
9  
10    fmt.Println("main start")  
11 }  
12
```

DEBUG CONSOLE ... Filter (e.g. text, !exclude)

```
API server listening at: 127.0.0.1:20054  
main start  
main end  
Process exiting with code: 0
```

```
workspace_go > src > second_day > go myFunc.go > {}  
1 package main  
2  
3 import "fmt"  
4  
5 func main() {  
6     defer func() {  
7         fmt.Println("main end")  
8     }()  
9  
10    fmt.Println("main start")  
11  
12    test()  
13 }  
14  
15 func test() {  
16     defer fmt.Println("test end")  
17     fmt.Println("test start")  
18 }  
19
```

DEBUG CONSOLE ... Filter (e.g. text, !exclude)

```
API server listening at: 127.0.0.1:29290  
main start  
test start  
test end  
main end  
Process exiting with code: 0
```

패닉 함수

- 일반적으로 무언가 예상치 못하게 잘못되었을 경우, 패닉 함수가 실행됨
- 오류를 빠르게 인식하고, 프로그램 실행을 멈춤
- 타 언어의 exception 기능과 비슷하게 사용됨

패닉 함수

```
workspace_go > src > second_day > go myFunc.go > {} main > test
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     a := []int{1, 2, 3, 4, 5}
8     for i := range a {
9         fmt.Println(a[i+1])
10    }
11 }
12
```

```
PS C:\Users\taehoon\Desktop\workspace_go\src\second_day> go run .\myFunc.go
2
3
4
5
panic: runtime error: index out of range [5] with length 5

goroutine 1 [running]:
main.main()
    C:/Users/taehoon/Desktop/workspace_go/src/second_day/myFunc.go:9 +0x129
exit status 2
```

```
workspace_go > src > second_day > go myFunc.go > ...
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     defer func() {
7         r := recover()
8         fmt.Println(r)
9     }()
10
11     a := []int{1, 2, 3, 4, 5}
12     for i := range a {
13         fmt.Println(a[i+1])
14     }
15 }
16
```

DEBUG CONSOLE

Filter (e.g. text, !exclude)

```
API server listening at: 127.0.0.1:2222
2
3
4
5
runtime error: index out of range [5] with length 5
Process exiting with code: 0
```

패닉 함수

```
2
3 import "fmt"
4
5 func main() {
6     a := []int{1, 2, 3, 4, 5}
7     for i := range a {
8         fmt.Println(a[i])
9     }
10    panic("panic")
11 }
```

PS C:\Users\taehoon\Desktop\workspace_go\src\second_day> go run .\myFunc.go

```
1
2
3
4
5
panic: panic
```

```
goroutine 1 [running]:
main.main()
```

```
    C:/Users/taehoon/Desktop/workspace_go/src/second_day/myFunc.go:10 +0x113
exit status 2
```

```
2
3 import "fmt"
4
5 func main() {
6     defer func() {
7         r := recover()
8         fmt.Println(r)
9     }()
10
11    a := []int{1, 2, 3, 4, 5}
12    for i := range a {
13        fmt.Println(a[i])
14    }
15    panic("panic")
16 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

API server listening at: 127.0.0.1:19424

```
1
2
3
4
5
panic
```