

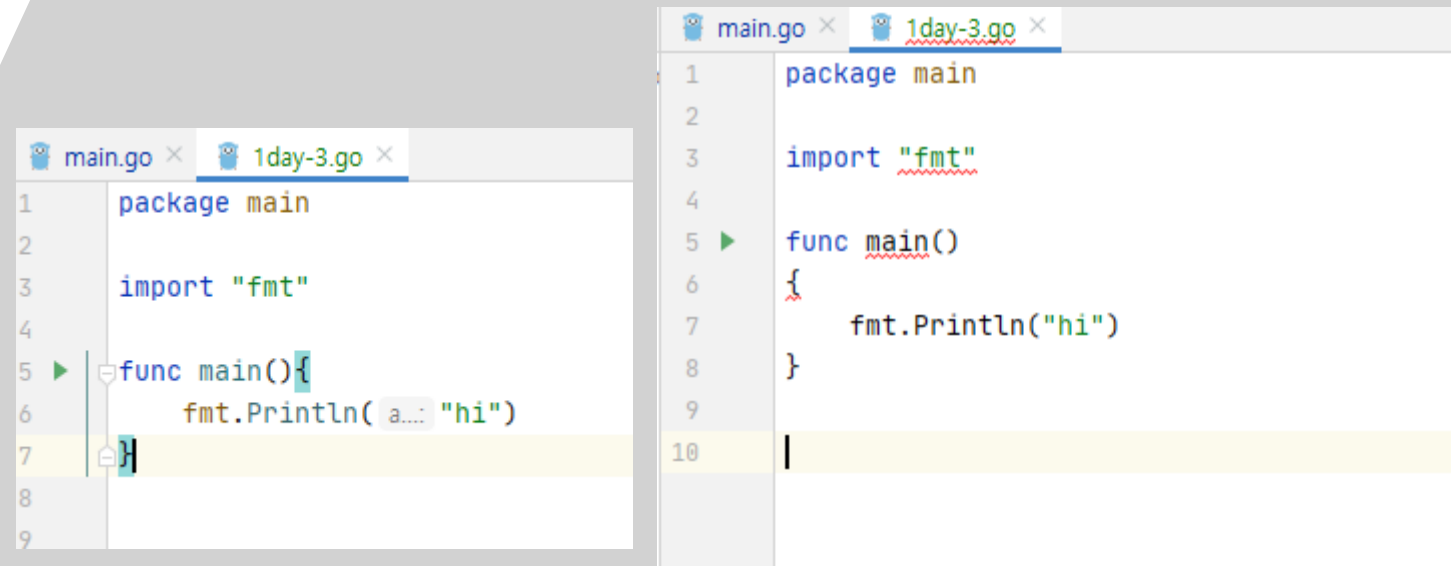


목차

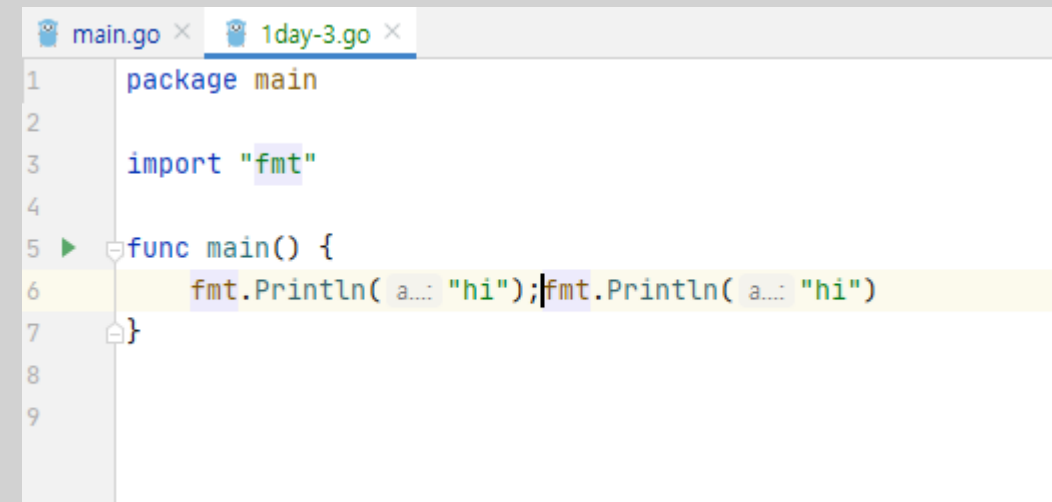
- 변수, 상수 선언
- 데이터 형
- 연산자
- 패키지
- 제어문
- 반복문

기본 문법

- 시작은 중괄호
- Tab을 활용한 들여쓰기
- 세미 콜론을 활용한 줄바꿈



```
1 package main
2
3 import "fmt"
4
5 func main(){
6     fmt.Println( a... "hi")
7 }
```



```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println( a... "hi");fmt.Println( a... "hi")
7 }
```

기본 문법

- 주석은 두가지
- // or /* */
- Eclipse Keymap 기준 control + /



The screenshot shows the Eclipse IDE with two tabs: 'main.go' and '1day-3.go'. The '1day-3.go' tab is active, displaying the following Go code:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     //fmt.Println("hi")
7     fmt.Println(a...: "hi")
8 }
9
10
```

Line 7 is highlighted in yellow. The code uses single-line comments with '//'.



The screenshot shows the Eclipse IDE with two tabs: 'main.go' and '1day-3.go'. The '1day-3.go' tab is active, displaying the following Go code:

```
1 package main
2
3 func main() {
4     /*
5         fmt.Println("hi")
6         fmt.Println("hi")
7     */
8 }
9
10
```

Line 10 is highlighted in yellow. The code uses multi-line comments with '/* */'.

변수와 상수

- 변수
 - 변하는 수
 - 변할 수 있는 Object
- 상수
 - 항상 같은 수
 - 항상 같은 Object

Example)

- $A+B = 10$ (A, B는 변수)
- $3+7 = 10$ (3, 7은 상수)

변수

- Var 키워드를 사용하는 방식
 - 자료형이 변수명 뒤에 있음 (가독성)
 - 문자와 숫자로 이뤄짐
 - 변수명의 시작은 _ or 알파벳으로 시작 (숫자, 특수문자 불가)

```
main.go x 1day-3.go x
1 package main
2
3 import "fmt"
4
5 func main() {
6     var i int
7     var s string
8
9     var age int = 10
10    var name string = "Maria"
11 }
12
```

```
main.go x 1day-3.go x
1 package main
2
3 import "fmt"
4
5 func main() {
6     var i = 10
7     var s = "test"
8
9 }
10
```

```
main.go x 1day-3.go x
1 package main
2
3 func main() {
4     i := 10
5     s := "test"
6 }
7
```

변수

- 변수 선언 및 초기화
 - 콤마로 구분
 - 변수 개수에 맞춰 초기화

```
main.go x 1day-3.go x
1 package main
2
3 ▶ func main() {
4     var x, y int
5     var a, b int = 1, 2
6
7     i, j, k := 10, 20, "test"
8 }
9
```

```
main.go x 1day-3.go x
1 package main
2
3 import "fmt"
4
5 ▶ func main() {
6     var (
7         x, y = 10, 20
8         age, name = 10, "taehoon"
9     )
10    fmt.Println(x, y, age, name)
11 }
12
```

변수

- 변수에 함수를 담아서 사용

```
main.go x 1day-3.go x
2
3 import (
4     "fmt"
5     "reflect"
6 )
7
8 func main() {
9     var f = ex
10    f()
11    f( temp...: "a", 10, "aa", "asdawe", 1.23231)
12    f( temp...: 1)
13
14    fmt.Println(reflect.TypeOf(f))
15
16 }
17
18 func ex(temp ...interface{}){
19     fmt.Println( a...: "example", temp)
20 }
21
```

```
Run: go build 1day-3.go x
<4 go setup calls>
example []
example [a 10 aa asdawe 1.23231]
example [1]
func(...interface {}))
Process finished with exit code 0
```


상수

- Const 키워드를 사용하는 방식

- 사용하는 방식은 변수와 같음
- 첫 초기화가 반드시 필요, 이후 값 변경 불가

```
6
7 ▶ func main() {
8
9     const a int = 10
10    const s string = "temp"
11
12    | fmt.Println(a, s)
13
14 }
```

```
6
7 ▶ func main() {
8
9     const a int = 10
10    const s string = "temp"
11
12    a = 1
13    s = "test"
14    fmt.Println(a, s)
15
16 }
```

데이터 형

Basic types

Go's basic types are

```
bool

string

int  int8  int16  int32  int64
uint uint8  uint16 uint32 uint64 uintptr

byte // alias for uint8

rune // alias for int32
    // represents a Unicode code point

float32 float64

complex64 complex128
```

The example shows variables of several types, and also that variable declarations may be "factored" into blocks, as with import statements.

The `int`, `uint`, and `uintptr` types are usually 32 bits wide on 32-bit systems and 64 bits wide on 64-bit systems. When you need an integer value you should use `int` unless you have a specific reason to use a sized or unsigned integer type.

출처:
<https://tour.golang.org/basics/11>

연산자

=	대입	변수나 상수에 값을 대입합니다. 변수는 변수끼리 대입할 수 있습니다. <pre>var a int = 1 var b int = 2 var c int = b const d string = "Hello, world!"</pre>
:=	변수 선언 및 대입	변수를 선언하는 동시에 값을 대입합니다. <pre>a := 1 // int b := 3.5 // float64 c := "Hello, world!" // string</pre>
+	덧셈	두 값을 더합니다. 사용할 수 있는 자료형은 정수, 실수, 복소수, 문자열입니다. <pre>a := 1 + 2 // 3: 두 정수 더하기 b := 2 + 3 // 5: 두 정수 더하기 c := a + b // 8: 두 변수 더하기 d := "Hello, " + "world!" // Hello, world!: 두 문자열 붙이기</pre>
-	뺄셈	두 값의 차이를 구합니다. 사용할 수 있는 자료형은 정수, 실수, 복소수입니다. <pre>a := 3 - 2 // 1: 두 정수 빼기 b := 4 - 5 // -1: 두 정수 빼기 c := a - b // 2: 두 변수 빼기</pre>
*	곱셈	두 값을 곱합니다. 사용할 수 있는 자료형은 정수, 실수, 복소수입니다. <pre>a := 2 * 3 // 6: 두 정수 곱하기 b := 9 * 21 // 189: 두 정수 곱하기 c := a * b // 1134: 두 변수 곱하기</pre>
/	나눗셈	두 값을 나눕니다. 사용할 수 있는 자료형은 정수, 실수, 복소수입니다. <pre>a := 5 / 2 // 2: 두 정수 나누기 b := 12 / 4 // 3: 두 정수 나누기 c := a / b // 0: 두 변수 나누기</pre>

%	나머지	두 값을 나눈 뒤 나머지를 구합니다. 사용할 수 있는 자료형은 정수입니다. <pre>a := 5 % 2 // 1: 5를 2로 나누었을 때 나머지 구하기</pre>
+=	덧셈 후 대입	현재 변수와 값을 더한 다음 다시 변수에 대입합니다. 문자열은 현재 변수에 문자열을 붙인 다음 다시 변수에 대입합니다. <pre>a := 5 // 5 a += 2 // 7: a에 2를 더한 후 대입 fmt.Println(a) // 7 a := "Hello, " // Hello, a += "world!" // Hello, world!: a에 world! 문자열을 붙인 후 대입 fmt.Println(a) // Hello, world!</pre>

출처:

<http://pyrasis.com/book/GoForTheReallyImpatient/Unit13>

연산자

== 같다

두 값이 같은지 비교합니다.

- 실수형은 값을 연산한 뒤에는 오차가 발생하므로 ==로 비교할 때 주의해야 합니다.
- 문자열은 내용이 같은지 비교합니다.
- 배열은 요소의 내용이 모두 같은지 비교합니다.
- 슬라이스와 얽은 배열과는 달리 내용을 비교할 수 없고, 메모리에 할당되어 있는지 확인합니다.
- 포인터는 주소가 같은지 비교합니다.

```
fmt.Println(1 == 1) // true: 두 정수가 같으므로 true
fmt.Println(3.5 == 3.5) // true: 두 실수가 같으므로 true
fmt.Println("Hello" == "Hello") // true: 두 문자열이 같으므로 true
```

```
a := [3]int{1, 2, 3}
b := [3]int{1, 2, 3}
fmt.Println(a == b) // true: 두 배열이 같으므로 true
```

```
c := [3]int{1, 2, 3}
fmt.Println(c == nil) // false: 슬라이스를 nil과 비교하여
// 메모리가 할당되었는지 확인
```

```
d := map[string]int{"Hello": 1}
fmt.Println(d == nil) // false: 맵을 nil과 비교하여
// 메모리가 할당되었는지 확인
```

```
e := 1
var p1 *int = &e
var p2 *int = &e
fmt.Println(p1 == p2) // true: 포인터에 저장된 메모리 주소가 같으므로 true
```

!= 같지 않다

두 값이 다른지 비교합니다.

```
fmt.Println(1 != 2) // true: 두 정수가 다르므로 true
fmt.Println(3.5 != 5.5) // true: 두 실수가 다르므로 true
fmt.Println("Hello" != "world") // true: 두 문자열이 다르므로 true
```

```
a := [3]int{1, 2, 3}
b := [3]int{3, 2, 1}
fmt.Println(a != b) // true: 두 배열이 다르므로 true
```

```
c := [3]int{1, 2, 3}
fmt.Println(c != nil) // true: 슬라이스를 nil과 비교하여
// 메모리가 할당되었는지 확인
```

```
d := map[string]int{"Hello": 1}
fmt.Println(d != nil) // true: 맵을 nil과 비교하여
// 메모리가 할당되었는지 확인
```

```
e := 1
f := 1
var p1 *int = &e
var p2 *int = &f
fmt.Println(p1 != p2) // true: 포인터에 저장된 메모리 주소가 다르므로 true
```

< 작다

앞의 값이 작은지 비교합니다. 문자열은 ASCII 코드 값을 기준으로 판단합니다. 또한, 첫 글자가 같다면 그 다음 글자부터 차례대로 비교하여 최종 값을 구합니다.

```
fmt.Println(1 < 2) // true: 1이 2보다 작으므로 true
fmt.Println(3.5 < 5.5) // true: 3.5가 5.5보다 작으므로 true
fmt.Println("Hello" < "world") // true: H가 w보다 ASCII 코드 값이
// 작으므로 true
```

<= 작거나 같다

앞의 값이 작거나 같은지 비교합니다.

```
fmt.Println(2 <= 2) // true: 2가 2보다 작거나 같으므로 true
fmt.Println(3.5 <= 5.5) // true: 3.5가 5.5보다 작거나 같으므로 true
fmt.Println("Hello" <= "world") // true: H가 w보다 ASCII 코드 값이
// 작거나 같으므로 true
```

> 크다

앞의 값이 큰지 비교합니다.

```
fmt.Println(2 > 1) // true: 2가 1보다 크므로 true
fmt.Println(5.5 > 3.5) // true: 5.5가 3.5보다 크므로 true
fmt.Println("world" > "Hello") // true: w가 H보다 ASCII 코드 값이 크므로 true
```

>= 크거나 같다

앞의 값이 크거나 같은지 비교합니다.

```
fmt.Println(2 >= 2) // true: 2가 2보다 크거나 같으므로 true
fmt.Println(5.5 >= 3.5) // true: 5.5가 3.5보다 크거나 같으므로 true
fmt.Println("world" >= "Hello") // true: w가 H보다 ASCII 코드 값이
// 크거나 같으므로 true
```

&& AND 논리 연산

두 볼 값이 모두 참인지 확인합니다.

```
fmt.Println(true && true) // true: 두 값이 모두 true이므로 true
fmt.Println(true && false) // false: 두 값 중 하나가 false이므로 false
fmt.Println(false && false) // false: 두 값이 모두 false이므로 false
```

|| OR 논리 연산

두 볼 값 중 한 개라도 참인지 확인합니다.

```
fmt.Println(true || true) // true: 두 값이 모두 true이므로 true
fmt.Println(true || false) // true: 두 값 중 하나가 true이므로 true
fmt.Println(false || false) // false: 두 값이 모두 false이므로 false
```

출처:

<http://pyrasis.com/book/GoForTheReallyImpatient/Unit13>

연산자

& 참조(레퍼런스) 연산 현재 변수의 메모리 주소를 구합니다.

```
a := 1
b := &a // a의 메모리 주소를 b에 대입
fmt.Println(b) // 0xc0820062d0 (메모리 주소)
```

* 역참조 연산 현재 포인터 변수에 저장된 메모리에 접근하여 값을 가져오거나 저장합니다.

```
a := new(int)
*a = 1 // a에 저장된 메모리에 접근하여 1을 저장
fmt.Println(*a) // 1: a에 저장된 메모리에 접근하여 값을 가져옴
```

<- 채널 수신 연산 채널에 값을 보내거나 값을 가져옵니다.

```
c := make(chan int)

go func() {
    c <- 1 // 채널 c에 1을 보냄
}()

a := <-c // 채널 c에서 값을 가져와서 a에 대입
fmt.Println(a) // 1
```

++ 증가

변수의 값을 1 증가시킵니다. 사용할 수 있는 자료형은 정수, 실수, 복소수입니다.

```
a := 1
a++ // 2: 정수 1을 1 증가시켜서 2

b := 1.5
b++ // 2.5: 실수 1.5를 1 증가시켜서 2.5

c := 1 + 2i
c++ // (2+2i): 복소수 1+2i를 1 증가시켜서 2+2i

fmt.Println(a) // 2
fmt.Println(b) // 2.5
fmt.Println(c) // (2+2i)
```

Go 언어에서는 ++ 연산자를 사용한 뒤 값을 대입할 수 없고, 변수 뒤에서만 사용할 수 있습니다. 따라서 ++ 연산자는 단독으로 사용하거나 if 조건문, for 반복문 안에서 주로 사용합니다.

```
a := 1
b := a++ // 컴파일 에러
c := ++a // 컴파일 에러
++a // 컴파일 에러
```

-- 감소

변수의 값을 1 감소시킵니다. 사용할 수 있는 자료형은 정수, 실수, 복소수입니다.

```
a := 1
a-- // 0: 정수 1을 1 감소시켜서 0

b := 1.5
b-- // 0.5: 실수 1.5를 1 감소시켜서 0.5

c := 1 + 2i
c-- // (0+2i): 복소수 1+2i를 1 감소시켜서 0+2i

fmt.Println(a) // 0
fmt.Println(b) // 0.5
fmt.Println(c) // (0+2i)
```

Go 언어에서는 -- 연산자를 사용한 뒤 값을 대입할 수 없고, 변수 뒤에서만 사용할 수 있습니다. 따라서 -- 연산자는 단독으로 사용하거나 if 조건문, for 반복문 안에서 주로 사용합니다.

```
a := 1
b := a-- // 컴파일 에러
c := --a // 컴파일 에러
--a // 컴파일 에러
```

출처:

<http://pyrasis.com/book/GoForTheReallyImpatient/Unit13>

연산자

- 연산자 우선순위
- 괄호()를 사용하는 습관을...

우선순위	연산자
5	* / % << >> & &^
4	+ - ^
3	== != < <= > >=
2	&&
1	

출처:

<http://pyrasis.com/book/GoForTheReallyImpatient/Unit13>

패키지

- 각종 기능과 라이브러리를 묶어서 제공
 - Import 키워드를 사용
 - OpenSource, CustomPackage 사용

```
main.go x 1day-3.go x
1 package main
2
3 import (
4     "fmt"
5     "reflect"
6 )
7
```

```
pkg
├── windows_amd64
│   └── github.com
│       └── go-sql-driver
│           ├── mysql.a
│           └── myLib.a
└── src
```

```
3
4 import "fmt"
5 import "reflect"
6 import "github.com/go-sql-driver/mysql"
7
```

```
4 import "fmt"
5 import "reflect"
6 import "github.com/go-sql-driver/mysql"
7 import "myLib"
8
```

```
4 import . "fmt"
5 import r "reflect"
6
7 func main() {
8
9     const a int = 10
10    const s string = "temp"
11
12    |
13    Println(a, s)
14    Println(r.TypeOf(a))
15
16 }
```

조건문 (if)

- 특정 조건을 설정해 프로그램의 흐름을 바꿀 때 사용
- If, else if , else 키워드 사용
- 조건 부분에는 True, False 형식만 사용 가능

```
8 ▶ func main() {  
9     var a int  
10    a = 10  
11  
12    if a == 10 {  
13        fmt.Println(a...: "10")  
14    } else if a > 11 {  
15        fmt.Println(a...: "up")  
16    } else {  
17        fmt.Println(a...: "down")  
18    }  
19 }
```

```
8 ▶ func main() {  
9     /.../  
11    /.../  
12    /.../  
19    |  
20    if temp := 0 ; temp < 10 {  
21        fmt.Println(a...: "hi")  
22    }  
23  
24 }  
25
```

```
24    if 1 {  
25  
26    }  
27    if true {  
28        |  
29    }  
30
```


조건문 (switch)

- 조건이 다양한 경우, 분기문을 활용해 간단히 표현 가능

```
8 ▶ func main() {  
9     i := 10  
10    switch i {  
11    case 10:  
12        fmt.Println(a... "i : 10")  
13    case 9:  
14        fmt.Println(a... "i : 9")  
15    case 11:  
16        fmt.Println(a... "i : 11")  
17    default:  
18        fmt.Println(a... "i : ?")  
19    }  
20 }
```

```
8 ▶ func main() {  
9     switch i:= 10; i {  
10    case 9, 10, 11:  
11        fmt.Println(a... "i : ", i)  
12    case 6,7,8:  
13        fmt.Println(a... "i : lower")  
14    case 12,13,14:  
15        fmt.Println(a... "i : upper")  
16    default:  
17        fmt.Println(a... "i : ?")  
18    }  
19 }
```

```
8 ▶ func main() {  
9     i := 12  
10    switch {  
11    case i == 10 || i == 11 || i == 12:  
12        fmt.Println(a... "i : ", i)  
13    case i < 10:  
14        fmt.Println(a... "i : lower")  
15    case i > 10:  
16        fmt.Println(a... "i : upper")  
17    default:  
18        fmt.Println(a... "i : ?")  
19    }  
20 }  
21 }
```

반복문

- 특정 조건에서 반복하는 일을 수행
- For 키워드 사용

```
8 ▶ func main() {  
9  
10     for i := 0 ; i < 10 ; i++){  
11         fmt.Println(i)  
12     }  
13 }  
14
```

```
8 ▶ func main() {  
9  
10     count := 0  
11     for true{  
12         if count == 10{  
13             break  
14         }  
15  
16         fmt.Printf(format: "%d ", count)  
17         count += 1  
18     }  
19 }
```

go build 1day-3.go ×

<4 go setup calls>
0 1 2 3 4 5 6 7 8 9
Process finished with exit code 0

```
8 ▶ func main() {  
9  
10     count := 0  
11     for true{  
12         if count == 10{  
13             break  
14         }  
15         if count == 5{  
16             count += 1  
17             continue  
18         }  
19         fmt.Printf(format: "%d ", count)  
20         count += 1  
21     }  
22 }  
23
```

go build 1day-3.go ×

<4 go setup calls>
0 1 2 3 4 6 7 8 9
Process finished with exit code 0