

# GO 빠른 참조

문법, 타입, 동시성, 에러 처리 핵심

## 기본

### Hello World

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

### 실행 & 빌드

```
go run main.go           # compile and run
go build -o app .       # compile to binary
go test ./...          # run all tests
```

### 모듈 초기화

```
go mod init github.com/user/project
go mod tidy                # sync dependencies
```

## 변수 & 타입

### 선언

```
var name string = "Go"
age := 15 // short declaration
var x, y int = 1, 2
const Pi = 3.14159
```

### 기본 타입

```
bool           `true`, `false`
string         UTF-8 불변 바이트 시퀀스
int, int8..int64  부호 있는 정수 (플랫폼 / 고정 너비)
uint, uint8..uint64 부호 없는 정수
float32, float64 IEEE-754 부동소수점
byte           `uint8`의 별칭
rune          `int32`의 별칭 (유니코드 코드 포인트)
```

### 제로값

```
int, float     `0`
bool          `false`
string        `""` (빈 문자열)
pointer, slice, map `nil`
```

## 함수

### 기본 함수

```
func add(a, b int) int {
    return a + b
}
```

### 다중 반환값

```
func divide(a, b float64) (float64, error) {
    if b == 0 {
        return 0, errors.New("division by zero")
    }
    return a / b, nil
}
```

### 가변인자 & 익명 함수

```
func sum(nums ...int) int {
    total := 0
    for _, n := range nums { total += n }
    return total
}
double := func(x int) int { return x * 2 }
```

## Defer

```
func readFile(path string) {
    f, _ := os.Open(path)
    defer f.Close() // runs when function returns
}
```

## 제어 흐름

### If / Else

```
if x > 0 {
    fmt.Println("positive")
} else if x == 0 {
    fmt.Println("zero")
} else {
    fmt.Println("negative")
}
```

### For 루프

```
for i := 0; i < 10; i++ { // classic
    for x < 100 { x *= 2 } // while-style
    for { break } // infinite
    for i, v := range slice { // range
}
```

### Switch

```
switch day {
case "Mon", "Tue":
    fmt.Println("early week")
case "Fri":
    fmt.Println("TGIF")
default:
    fmt.Println("other")
}
```

## 구조체 & 메서드

### 구조체 정의

```
type User struct {
    Name string
    Email string
    Age int
}
u := User{Name: "Alice", Email: "a@b.com", Age: 30}
```

### 메서드

```
func (u User) Greeting() string {
    return "Hi, " + u.Name
}
func (u *User) SetAge(age int) {
    u.Age = age // pointer receiver mutates
}
```

### 임베딩

```
type Admin struct {
    // embedded struct
    User
    Level string
}
a := Admin{User: User{Name: "Bob"}, Level: "super"}
fmt.Println(a.Name) // promoted field
```

## 인터페이스

### 정의 & 구현

```
type Stringer interface {
    String() string
}
// implicit implementation - no "implements" keyword
func (u User) String() string {
    return u.Name
}
```

### 주요 인터페이스

```
io.Reader `Read(p []byte) (n int, err error)`
io.Writer `Write(p []byte) (n int, err error)`
fmt.Stringer `String() string`
error `Error() string`
```

### 타입 어실선

```
var i interface{} = "hello"
s, ok := i.(string) // ok == true
switch v := s; (type) {
case string: fmt.Println(v)
case int:    fmt.Println(v * 2)
}
```

## 고루틴 & 채널

### 고루틴

```
go func() {
    fmt.Println("running concurrently")
}()
time.Sleep(time.Second)
```

### 채널

```
ch := make(chan int) // unbuffered
buf := make(chan int, 5) // buffered
ch <- 42 // send
val := <- ch // receive
```

### Select

```
select {
case msg := <- ch1:
    fmt.Println(msg)
case ch2 <- 42:
    fmt.Println("sent")
case <- time.After(time.Second):
    fmt.Println("timeout")
}
```

## 패턴

```
sync.WaitGroup 여러 고루틴이 끝날 때까지 대기
sync.Mutex 공유 상태를 위한 상호 배제 잠금
context.Context 취소, 마감 시간, 요청 범위 값
```

## 에러 처리

### 기본 패턴

```
result, err := doSomething()
if err != nil {
    return fmt.Errorf("failed: %w", err)
}
```

### 커스텀 에러

```
type NotFoundError struct {
    ID string
}
func (e *NotFoundError) Error() string {
    return "not found: " + e.ID
}
```

## errors 패키지

```
errors.New(msg) 단순 에러 생성
fmt.Errorf("%w", err) 컨텍스트와 함께 에러 래핑
errors.Is(err, target) 에러 체인에서 일치 확인
errors.As(err, &target) 체인에서 타입 에러 추출
```

## 슬라이스 & 맵

### 슬라이스

```
s := []int{1, 2, 3}
s = append(s, 4, 5)
sub := s[1:3] // [2, 3]
cp := make([]int, len(s))
copy(cp, s)
```

### 맵

```
m := map[string]int{"a": 1, "b": 2}
m["c"] = 3
val, ok := m["a"] // ok == true
delete(m, "b")
for k, v := range m { }
```

### 슬라이스 작업

```
len(s) 요소 수
cap(s) 기저 배열 용량
append(s, elems...) 요소 추가, 재할당 가능
copy(dst, src) 슬라이스 키 요소 복사
slices.Sort(s) 슬라이스 정렬 (Go 1.21+ `slices` 패키지)
```

## 패키지 & 임포트

### 임포트 방식

```
import "fmt"
import (
    "os"
    "strings"
    "github.com/user/pkg"
)
```

### 가시성

대문자로 시작 = 내보내기 (공개).  
소문자로 시작 = 내보내기 안 함 (패키지 전용).  
public/private 키워드 불필요.

## 주요 표준 라이브러리

```
fmt 서식 있는 I/O (Print, Printf, Errorf)
os OS 함수 (파일, 환경, 인수)
io I/O 기본 요소 (Reader, Writer)
net/http HTTP 클라이언트 및 서버
encoding/json JSON 인코딩/디코딩
strings 문자열 조작 함수
strconv 문자열 ↔ 숫자 변환
testing 단위 테스트 프레임워크
```

## 제네릭

### 타입 파라미터

```
func Map[T, U any](s []T, f func(T) U) []U {
    r := make([]U, len(s))
    for i, v := range s { r[i] = f(v) }
    return r
}
```

### 제약 조건

```
type Number interface {
    ~int | ~float64
}
func Sum[T Number](nums []T) T {
    var total T
    for _, n := range nums { total += n }
    return total
}
```

## 테스트

### 기본 테스트

```
// file: math_test.go
func TestAdd(t *testing.T) {
    got := Add(2, 3)
    if got != 5 {
        t.Errorf("Add(2,3) = %d, want 5", got)
    }
}
```

### 테스트 명령

```
go test 현재 패키지의 테스트 실행
go test ./... 모든 테스트 재귀 실행
go test -v 상세 출력
go test -run TestAdd 이름으로 특정 테스트 실행
go test -bench . 벤치마크 실행
go test -cover 커버리지 비율 표시
```