Go Tham Khảo Nhanh

Cú pháp, kiểu dữ liệu, concurrency, xử lý lỗi

Cơ Bản

Hello World

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

Chạy & Build

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

Khởi Tạo Module

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

Biến & Kiểu

Khai Báo

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

Kiểu Cơ Bản

bool	**true**, **false**
string	Chuỗi byte UTF-8 bất biến
int, int8..int64	Số nguyên có dấu (platform / chiều rộng cố định)
uint, uint8..uint64	Số nguyên không dấu
float32, float64	Số thực IEEE-754
byte	Alias cho **uint8**
rune	Alias cho **int32** (Unicode code point)

Giá Trị Zero

int, float	**0**
bool	**false**
string	**""** (chuỗi rỗng)
pointer, slice, map	**nil**

Hàm

Hàm Cơ Bản

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

Nhiều Giá Trị Trả Về

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

Variadic & Ẩn Danh

```
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
}
```

Luồng Điều Khiển

If / Else

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

Vòng Lặp For

```
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")
}
```

Struct & Method

Định Nghĩa Struct

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

Method

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

Embedding

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

Interface

Định Nghĩa & Triển Khai

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

Interface Phổ Biến

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**

Type Assertion

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

Goroutine & Channel

Goroutine

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

Channel

```
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")
}
```

Pattern

sync.WaitGroup	Chờ nhiều goroutine hoàn thành
sync.Mutex	Khóa loại trừ tương hỗ cho trạng thái dùng chung
context.Context	Hủy, deadline, giá trị theo phạm vi request

Xử Lý Lỗi

Pattern Cơ Bản

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

Lỗi Tùy Chỉnh

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

Package errors

errors.New(msg)	Tạo lỗi đơn giản
fmt.Errorf("%w", err)	Wrap lỗi với context
errors.Is(err, target)	Kiểm tra chuỗi lỗi
errors.As(err, &target)	Trích xuất lỗi có kiểu từ chuỗi

Go Tham Khảo Nhanh

Slice & Map

Slice

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

Map

```
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 { }
```

Thao Tác Slice

`len(s)`	Số phần tử
`cap(s)`	Dung lượng mảng nền
`append(s, elems...)`	Thêm phần tử, có thể cấp phát lại
`copy(dst, src)`	Sao chép phần tử giữa các slice
`slices.Sort(s)`	Sắp xếp slice (Go 1.21+ **`slices`** pkg)

Package & Import

Cách Import

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

Phạm Vi Hiển Thị

Chữ cái đầu hoa = exported (public).
Chữ cái đầu thường = unexported (package-private).
Không cần từ khóa public/private.

Thư Viện Chuẩn Phổ Biến

`fmt`	I/O có định dạng (Print, Sprintf, Errorf)
`os`	Hàm OS (file, env, args)
`io`	Primitive I/O (Reader, Writer)
`net/http`	HTTP client và server
`encoding/json`	Mã hóa/giải mã JSON
`strings`	Hàm xử lý chuỗi
`strconv`	Chuyển đổi chuỗi ↔ số
`testing`	Framework unit test

Generics

Tham Số Kiểu

```
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
}
```

Ràng Buộc

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

Testing

Test Cơ Bản

```
// 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)
    }
}
```

Lệnh Test

`go test`	Chạy test trong package hiện tại
`go test ./...`	Chạy tất cả test đệ quy
`go test -v`	Output chi tiết
`go test -run TestAdd`	Chạy test cụ thể theo tên
`go test -bench .`	Chạy benchmark
`go test -cover`	Hiển thị phần trăm coverage