RIFERIMENTO RAPIDO GO

Sintassi, tipi, concorrenza, gestione errori

Basi

Hello World

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

Esecuzione e Build

```
go run main.go        # compila ed esegui
go build -o app .     # compila in binario
go test ./...         # esegui tutti i test
```

Inizializzazione Modulo

```
go mod init github.com/user/project
go mod tidy           # sincronizza dipendenze
```

Variabili e Tipi

Dichiarazione

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

Tipi Base

`bool`	`true`, `false`
`string`	Sequenza di byte UTF-8 immutabile
`int, int8..int64`	Interi con segno (piattaforma / larghezza fissa)
`uint, uint8..uint64`	Interi senza segno
`float32, float64`	Virgola mobile IEEE-754
`byte`	Alias per `uint8`
`rune`	Alias per `int32` (code point Unicode)

Valori Zero

`int, float`	`0`
`bool`	`false`
`string`	`""` (stringa vuota)
`pointer, slice, map`	`nil`

Funzioni

Funzione Base

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

Valori di Ritorno Multipli

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

Variadic e Anonime

```
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()   // eseguito al ritorno dalla funzione
}
```

Controllo del Flusso

If / Else

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

Ciclo For

```
for i := 0; i < 10; i++ { }  // classico
for x < 100 { x *= 2 }       // stile while
for { break }                // infinito
for i, v := range slice { }   // range
```

Switch

```
switch day {
case "Mon", "Tue":
    fmt.Println("inizio settimana")
case "Fri":
    fmt.Println("TGIF")
default:
    fmt.Println("altro")
}
```

Struct e Metodi

Definizione Struct

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

Metodi

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

Embedding

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

Interfacce

Definizione e Implementazione

```
type Stringer interface {
    String() string
}
// implementazione implicita — nessuna parola chiave "implements"
func (u User) String() string {
    return u.Name
}
```

Interfacce Comuni

`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 e Channel

Goroutine

```
go func() {
    fmt.Println("esecuzione concorrente")
}()
time.Sleep(time.Second)
```

Channel

```
ch := make(chan int)       // non bufferizzato
buf := make(chan int, 5)   // bufferizzato
ch <- 42                   // invia
val := <-ch                // ricevi
```

Select

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

Pattern

`sync.WaitGroup`	Attende il completamento di più goroutine
`sync.Mutex`	Lock di esclusione reciproca per stato condiviso
`context.Context`	Cancellazione, scadenze, valori con scope alla richiesta

Gestione Errori

Pattern Base

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

Errori Personalizzati

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

Pacchetto errors

`errors.New(msg)`	Crea errore semplice
`fmt.Errorf("%w", err)`	Avvolge l'errore con contesto
`errors.Is(err, target)`	Controlla la catena errori per corrispondenza
`errors.As(err, &target)`	Estrae errore tipizzato dalla catena

Slice e 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 { }
```

Operazioni Slice

`len(s)`	Numero di elementi
`cap(s)`	Capacità dell'array sottostante
`append(s, elems...)`	Aggiunge elementi, può riallocare
`copy(dst, src)`	Copia elementi tra slice
`slices.Sort(s)`	Ordina la slice (Go 1.21+ pacchetto `slices`)

Pacchetti e Import

Stili di Import

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

Visibilità

Prima lettera maiuscola = esportato (pubblico).
Prima lettera minuscola = non esportato (privato al pacchetto).
Nessuna parola chiave come public/private necessaria.

Libreria Standard Comune

`fmt`	I/O formattato (Print, Sprintf, Errorf)
`os`	Funzioni OS (file, env, args)
`io`	Primitive I/O (Reader, Writer)
`net/http`	Client e server HTTP
`encoding/json`	Codifica/decodifica JSON
`strings`	Funzioni di manipolazione stringhe
`strconv`	Conversioni stringa ↔ numero
`testing`	Framework test unitari

Generics

Parametri di Tipo

```
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
}
```

Vincoli

```
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 Base

```
// 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)
    }
}
```

Comandi Test

`go test`	Esegui test nel pacchetto corrente
`go test ./...`	Esegui tutti i test ricorsivamente
`go test -v`	Output verbose
`go test -run TestAdd`	Esegui test specifico per nome
`go test -bench .`	Esegui benchmark
`go test -cover`	Mostra percentuale di copertura