REFERENCIA RÁPIDA DE GO

Sintaxis, tipos, concurrencia, manejo de errores esenciales

Básico

Hola Mundo

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

Ejecutar y compilar

```
go run main.go        # compilar y ejecutar
go build -o app .     # compilar a binario
go test ./...         # ejecutar todas las pruebas
```

Inicializar módulo

```
go mod init github.com/user/project
go mod tidy           # sincronizar dependencias
```

Variables y tipos

Declaración

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

Tipos básicos

`bool`	`true`, `false`
`string`	Secuencia de bytes UTF-8 inmutable
`int, int8..int64`	Enteros con signo (plataforma / ancho fijo)
`uint, uint8..uint64`	Enteros sin signo
`float32, float64`	Punto flotante IEEE-754
`byte`	Alias de `uint8`
`rune`	Alias de `int32` (punto de código Unicode)

Valores cero

`int, float`	`0`
`bool`	`false`
`string`	`""` (cadena vacía)
`pointer, slice, map`	`nil`

Funciones

Función básica

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

Múltiples valores de retorno

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

Variádica y anónima

```
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()   // se ejecuta cuando la función retorna
}
```

Flujo de control

If / Else

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

Bucle For

```
for i := 0; i < 10; i++ { }  // clásico
for x < 100 { x *= 2 }       // estilo while
for { break }                // infinito
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")
}
```

Structs y métodos

Definición de struct

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

Métodos

```
func (u User) Greeting() string {
    return "Hi, " + u.Name
}
func (u *User) SetAge(age int) {
    u.Age = age   // receptor de puntero muta
}
```

Embedding

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

Interfaces

Definir e implementar

```
type Stringer interface {
    String() string
}
// implementación implícita — sin palabra clave "implements"
func (u User) String() string {
    return u.Name
}
```

Interfaces comunes

`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`

Aserción de tipo

```
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)
}
```

Goroutines y canales

Goroutines

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

Canales

```
ch := make(chan int)       // sin buffer
buf := make(chan int, 5)   // con buffer
ch <- 42                   // enviar
val := <-ch                // recibir
```

Select

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

Patrones

`sync.WaitGroup`	Esperar que terminen múltiples goroutines
`sync.Mutex`	Bloqueo de exclusión mutua para estado compartido
`context.Context`	Cancelación, plazos, valores con scope de solicitud

Manejo de errores

Patrón básico

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

Errores personalizados

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

Paquete errors

`errors.New(msg)`	Crear error simple
`fmt.Errorf("%w", err)`	Envolver error con contexto
`errors.Is(err, target)`	Buscar coincidencia en la cadena de errores
`errors.As(err, &target)`	Extraer error tipado de la cadena

Slices y Maps

Slices

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

Maps

```
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 { }
```

Operaciones con slices

`len(s)`	Número de elementos
`cap(s)`	Capacidad del array subyacente
`append(s, elems...)`	Agregar elementos, puede reasignar
`copy(dst, src)`	Copiar elementos entre slices
`slices.Sort(s)`	Ordenar slice (paquete `slices` Go 1.21+)

Paquetes e importaciones

Estilos de importación

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

Visibilidad

Primera letra en mayúscula = exportado (público).
Primera letra en minúscula = no exportado (privado al paquete).
No se necesitan palabras clave como public/private.

Biblioteca estándar común

`fmt`	E/S formateada (Print, Sprintf, Errorf)
`os`	Funciones del SO (archivos, env, args)
`io`	Primitivas de E/S (Reader, Writer)
`net/http`	Cliente y servidor HTTP
`encoding/json`	Codificar/decodificar JSON
`strings`	Funciones de manipulación de cadenas
`strconv`	Conversiones cadena ↔ número
`testing`	Framework de pruebas unitarias

Genéricos

Parámetros de 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
}
```

Restricciones

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

Pruebas

Prueba básica

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

Comandos de prueba

`go test`	Ejecutar pruebas en el paquete actual
`go test ./...`	Ejecutar todas las pruebas recursivamente
`go test -v`	Salida detallada
`go test -run TestAdd`	Ejecutar prueba específica por nombre
`go test -bench .`	Ejecutar benchmarks
`go test -cover`	Mostrar porcentaje de cobertura