Referência Rápida Go

Sintaxe, tipos, concorrência, tratamento de erros essencial

Básico

Hello World

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

Executar e Compilar

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

Inicializar Módulo

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

Variáveis e Tipos

Declaração

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

Tipos Básicos

`bool`	**`true`**, **`false`**
`string`	Sequência de bytes UTF-8 imutável
`int, int8..int64`	Inteiros com sinal (plataforma / largura fixa)
`uint, uint8..uint64`	Inteiros sem sinal
`float32, float64`	Ponto flutuante IEEE-754
`byte`	Alias para **`uint8`**
`rune`	Alias para **`int32`** (código Unicode)

Valores Zero

`int, float`	**`0`**
`bool`	**`false`**
`string`	**`""`** (string vazia)
`pointer, slice, map`	**`nil`**

Funções

Função Básica

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

Múltiplos 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 e Anônima

```
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
}
```

Controle de Fluxo

If / Else

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

Laço 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")
}
```

Structs e Métodos

Definição 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   // 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
```

Interfaces

Definindo e Implementando

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

Interfaces Comuns

`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)
}
```

Goroutines e Channels

Goroutines

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

Channels

```
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")
}
```

Padrões

`sync.WaitGroup`	Aguardar múltiplas goroutines terminarem
`sync.Mutex`	Lock de exclusão mútua para estado compartilhado
`context.Context`	Cancelamento, prazos, valores com escopo de requisição

Tratamento de Erros

Padrão Básico

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

Erros Personalizados

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

Pacote errors

`errors.New(msg)`	Criar erro simples
`fmt.Errorf("%w", err)`	Encapsular erro com contexto
`errors.Is(err, target)`	Verificar cadeia de erros por correspondência
`errors.As(err, &target)`	Extrair erro tipado da cadeia

Referência Rápida Go

Slices e 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 { }
```

Operações com Slice

`len(s)`	Número de elementos
`cap(s)`	Capacidade do array subjacente
`append(s, elems...)`	Adicionar elementos, pode realocar
`copy(dst, src)`	Copiar elementos entre slices
`slices.Sort(s)`	Ordenar slice (pacote **`slices`** Go 1.21+)

Pacotes e Imports

Estilos de Import

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

Visibilidade

Primeira letra maiúscula = exportado (público).
Primeira letra minúscula = não exportado (privado ao pacote).
Sem palavras-chave como public/private.

Biblioteca Padrão Comum

`fmt`	I/O formatado (Print, Sprintf, Errorf)
`os`	Funções do SO (arquivos, env, args)
`io`	Primitivos de I/O (Reader, Writer)
`net/http`	Cliente e servidor HTTP
`encoding/json`	Codificar/decodificar JSON
`strings`	Funções de manipulação de string
`strconv`	Conversões string ↔ número
`testing`	Framework de testes unitários

Generics

Parâmetros 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
}
```

Constraints

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

Testes

Teste Básico

```
// 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)
    }
}
```

Comandos de Teste

`go test`	Executar testes no pacote atual
`go test ./...`	Executar todos os testes recursivamente
`go test -v`	Saída verbose
`go test -run TestAdd`	Executar teste específico por nome
`go test -bench .`	Executar benchmarks
`go test -cover`	Mostrar porcentagem de cobertura