

# 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

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

### Valores Zero

<b>int, float</b>	<b>0</b>
<b>bool</b>	<b>false</b>
<b>string</b>	<b>""</b> (string vazia)
<b>pointer, slice, map</b>	<b>nil</b>

## 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

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

## 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

<b>sync.WaitGroup</b>	Aguardar múltiplas goroutines terminarem
<b>sync.Mutex</b>	Lock de exclusão mútua para estado compartilhado
<b>context.Context</b>	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

<b>errors.New(msg)</b>	Criar erro simples
<b>fmt.Errorf("%w", err)</b>	Encapsular erro com contexto
<b>errors.Is(err, target)</b>	Verificar cadeia de erros por correspondência
<b>errors.As(err, &amp;target)</b>	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

<b>len(s)</b>	Número de elementos
<b>cap(s)</b>	Capacidade do array subjacente
<b>append(s, elems...)</b>	Adicionar elementos, pode realocar
<b>copy(dst, src)</b>	Copiar elementos entre slices
<b>lices.Sort(s)</b>	Ordenar slice (pacote <b>lices</b> 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

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

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