

# Tipagem em Go

# A Confusão Comum



```
got := Hello() // "Não declarei tipo, então é dinâmica?"
```

Às vezes pode-se pensar que funciona assim:

- Tipagem estática = "preciso escrever tipos sempre"
- Tipagem dinâmica = "não preciso escrever tipos"

**Não é bem assim, como vamos ver a seguir em exemplos**

# **Dois Eixos Independentes**

São **dois conceitos separados** que se combinam:

- 1. Estática vs Dinâmica → QUANDO** verificar tipos
- 2. Forte vs Fraca → RIGOR** nas verificações

# Tipagem

	FORTE	FRACA
ESTÁTICA	Go, Rust, Java, C#	C, C++
DINÂMICA	Python, Ruby	JavaScript, PHP

## **Eixo 1: Estática vs Dinâmica**

**Significa QUANDO os tipos são verificados**

# Tipagem estática

## Tipos verificados em **compilação**

```
// Go – verificação em COMPILAÇÃO
got := "hello"
got = 42 // ERRO antes de executar
          // cannot use 42 (type int) as type string
```

- Erros são detectados **antes de rodar**
- Não compila se houver conflito de tipo

# Tipagem dinâmica

## Tipos verificados em execução

```
# Python – verificação em execução pelo interpretador
got = "hello"
got = 42 # tipo da variável pode ser modificado em runtime
```

- Mais flexível, mas menos seguro

## **Eixo 2: Forte vs Fraca**

**Rigor do sistema de tipos**

# Tipagem forte

## Sem conversões implícitas

```
// Go - FORTE
var x int = 10
var y float64 = 3.14
z := x + y
// ERROR! invalid operation
// (mismatched types int and float64)

// Precisa converter explicitamente:
z := float64(x) + y // OK
```

# Tipagem FRACA

## Com conversões implícitas

```
// JavaScript – FRACA
var x = 10;
var y = "5";

console.log(x + y); // "105" – virou string!
console.log(x - y); // 5      – virou número!
```

⚠️ Conversões "mágicas" podem gerar bugs se não houver o devido cuidado. Além disso, é um alento poder contar com a checagem desse tipo de erro em tempo de compilação, pois problemas podem ser evitados antes do código entrar em produção.

**Go = Tipagem Estática + Forte + Inferência**

# Inferência de Tipos em Go

```
// Três formas EQUIVALENTES:
```

```
var got string = Hello()    // 1. Explícita
var got = Hello()           // 2. Infere pelo valor
got := Hello()              // 3. Declaração da variável + atribuição do valor
```

Em todas as três:

- Tipo é determinado em **tempo de compilação**
- O tipo de uma variável é **permanente**
- Ou seja, **não se muda** o tipo durante execução

# Inferência ≠ Dinâmica

A inferência facilita as coisas para o programador, enquanto mantém a checagem estática da tipagem.

```
// O compilador deduz o tipo...
got := Hello() // compilador sabe que Hello() retorna string

// ...mas o tipo é FIXO após compilação
got = 42 // erro de compilação
          // não pode mudar tipo!
```

**Tipo inferido** = tipo conhecido em compilação

**Tipo dinâmico** = tipo só conhecido em execução

# Python (tipagem dinâmica por padrão)

```
got = "hello"      # tipo: string (por enquanto)
got = 42          # isso funciona – tipo mudou em runtime
```

## Características:

- Tipo decidido na execução
- Por padrão pode mudar a qualquer momento
- Máxima flexibilidade

**Go é:**

**Estaticamente tipada** - tipos em compilação

**Fortemente tipada** - sem conversões implícitas

**Inferência de tipos** - compilador deduz pra você

# Vantagens do Go

1. **Segurança** - erros pegos cedo (compilação)
2. **Performance** - compilador conhece os tipos
3. **Clareza** - sem conversões mágicas
4. **Ergonomia** - menos verbosidade (inferência)

# Exemplo de conversão em Go (tipagem forte)

```
var x int = 10
var y float64 = 3.14

// ✗ Não compila:
z := x + y

// ✓ Precisa converter explicitamente:
z := float64(x) + y

// Ou:
z := x + int(y) // perde precisão
```

# Inferência em Go

```
// Compilador infere automaticamente:  
x := 42                      // int  
y := 3.14                     // float64  
nome := "Web II"               // string  
ok := true                     // bool  
  
// Mas pode ser explícito quando quiser:  
var x int = 42  
var y float64 = 3.14
```

# Conclusão

## Go Combina:

Característica	Benefício
Estática	Erros cedo, performance
Forte	Sem conversões estranhas
Inferência	Código conciso, mais ergonomia

# Vamos Codar!



Agora que entendemos a tipagem do Go, vamos evoluir nosso Olá, Mundo!

## Próximos passos:

- Adicionar parâmetros
- Validar entrada
- Usar TDD