

# ThoughtWorks®

uma pegadinha em #golang

---

# 3 SABORES DE VARIÁVEIS

---

*Valores, ponteiros, e "referências" na linguagem Go.*



**ThoughtWorks®**

# **ALÉM DA IMAGINAÇÃO**

---

*Fenômenos na 5<sup>a</sup> dimensão*

# UM ÔNIBUS PARANORMAL

---

```
5 // TwilightBus drops passengers to the 5th dimension
6 type TwilightBus struct {
7     passengers []string
8 }
9
10 // New builds and returns a pointer to a TwilightBus
11 func New(passengers []string) *TwilightBus {
12     bus := TwilightBus{}
13     bus.passengers = passengers
14     return &bus
15 }
16
17 // Pick picks a named passenger
18 func (bus *TwilightBus) Pick(name string) {
19     bus.passengers = append(bus.passengers, name)
20 }
21
22
23 // find needle in slice, returns index, found bool
24 func find(slice []string, needle string) (int, bool) {
25     for i, item := range slice {
26         if item == needle {
27             return i, true
28         }
29     }
30     return -1, false
31 }
32
33 // remove item from slice, preserving order
34 // from GOPL, sec. 4.2.2, p. 93
35 func remove(slice []string, i int) []string {
36     copy(slice[i:], slice[i+1:])
37     return slice[:len(slice)-1]
38 }
39
40 // Drop drops a named passenger
41 func (bus *TwilightBus) Drop(name string) {
42     i, found := find(bus.passengers, name)
43     if found {
44         bus.passengers = remove(bus.passengers, i)
45     }
46 }
```

# UM ÔNIBUS PARANORMAL

---

```
5 // TwilightBus drops passengers to the 5th dimension
6 type TwilightBus struct {
7     passengers []string
8 }
9
10 // New builds and returns a pointer to a TwilightBus
11 func New(passengers []string) *TwilightBus {
12     bus := TwilightBus{}
13     bus.passengers = passengers
14     return &bus
15 }
16
17 // Pick picks a named passenger
18 func (bus *TwilightBus) Pick(name string) {
19     bus.passengers = append(bus.passengers, name)
20 }
```

# UM ÔNIBUS PARANORMAL

---

```
23 // find needle in slice, returns index, found bool
24 func find(slice []string, needle string) (int, bool) {
25     for i, item := range slice {
26         if item == needle {
27             return i, true
28         }
29     }
30     return -1, false
31 }
32
33 // remove item from slice, preserving order
34 // from GOPL, sec. 4.2.2, p. 93
35 func remove(slice []string, i int) []string {
36     copy(slice[i:], slice[i+1:])
37     return slice[:len(slice)-1]
38 }
```

# UM ÔNIBUS PARANORMAL

---

```
33 // remove item from slice, preserving order
34 // from GOPL, sec. 4.2.2, p. 93
35 func remove(slice []string, i int) []string {
36     copy(slice[i:], slice[i+1:])
37     return slice[:len(slice)-1]
38 }
39
40 // Drop drops a named passenger
41 func (bus *TwilightBus) Drop(name string) {
42     i, found := find(bus.passengers, name)
43     if found {
44         bus.passengers = remove(bus.passengers, i)
45     }
46 }
```

# EXEMPLO DE USO DO ÔNIBUS PARANORMAL

---

```
5 func Example() {
6     team := []string{"Su", "Tina", "Mia", "Dea", "Pri"}
7     fmt.Printf("team:\n%#v\n", team)
8
9
10
11
12
13
14 // Output:
15 // team:
16 // []string{"Su", "Tina", "Mia", "Dea", "Pri"}
```

# EXEMPLO DE USO DO ÔNIBUS PARANORMAL

---

```
5 func Example() {
6     team := []string{"Su", "Tina", "Mia", "Dea", "Pri"}
7     fmt.Printf("team:\n %#v\n", team)
8     bus := New(team)
9     fmt.Printf("bus:\n %#v\n", bus)

14    // Output:
15    // team:
16    //     []string{"Su", "Tina", "Mia", "Dea", "Pri"}
17    // bus:
18    //     &main.TwilightBus{passengers:[]string{"Su", "Tina", "Mia", "Dea", "Pri"}}
```

# EXEMPLO DE USO DO ÔNIBUS PARANORMAL

---

```
5 func Example() {
6     team := []string{"Su", "Tina", "Mia", "Dea", "Pri"}
7     fmt.Printf("team:\n %#v\n", team)
8     bus := New(team)
9     fmt.Printf("bus:\n %#v\n", bus)
10    fmt.Println("# bus drops Tina")
11    bus.Drop("Tina")
12    fmt.Printf("bus:\n %#v\n", bus)

14 // Output:
15 // team:
16 //     []string{"Su", "Tina", "Mia", "Dea", "Pri"}
17 // bus:
18 //     &main.TwilightBus{passengers:[]string{"Su", "Tina", "Mia", "Dea", "Pri"}}
19 // # bus drops Tina
20 // bus:
21 //     &main.TwilightBus{passengers:[]string{"Su", "Mia", "Dea", "Pri"}}
```

# EXEMPLO DE USO DO ÔNIBUS PARANORMAL

---

```
5 func Example() {
6     team := []string{"Su", "Tina", "Mia", "Dea", "Pri"}
7     fmt.Printf("team:\n %#v\n", team)
8     bus := New(team)
9     fmt.Printf("bus:\n %#v\n", bus)
10    fmt.Println("# bus drops Tina")
11    bus.Drop("Tina")
12    fmt.Printf("bus:\n %#v\n", bus)
13    fmt.Printf("team:\n %#v\n", team)
14    // Output:
15    // team:
16    //   []string{"Su", "Tina", "Mia", "Dea", "Pri"}
17    // bus:
18    //   &main.TwilightBus{passengers:[]string{"Su", "Tina", "Mia", "Dea", "Pri"}}
19    // # bus drops Tina
20    // bus:
21    //   &main.TwilightBus{passengers:[]string{"Su", "Mia", "Dea", "Pri"}}
22    // team:
23    //   []string{"Su", "Mia", "Dea", "Pri", "Pri"}
24 }
```

YOU HAVE NOW  
CROSSED  
OVER INTO...

THE  
TWILIGHT  
ZONE



# DUAS SURPRESAS

---

Ônibus deixou Tina, e ela sumiu do time!

Apareceu uma clone de Pri no time!!

```
# bus drops Tina
bus:
  &main.TwilightBus{passengers:[ ]string{"Su", "Mia", "Dea", "Pri"}}
team:
  [ ]string{"Su", "Mia", "Dea", "Pri", "Pri"}
```



# PRINCÍPIO DA MÍNIMA SURPRESA

---

*"If a feature is accidentally misapplied by the user and causes what appears to him to be an unpredictable result, that feature has a high astonishment factor and is therefore undesirable. If a necessary feature has a high astonishment factor, it may be necessary to redesign the feature."*

— Cowlishaw, M. F. (1984). "The design of the REXX language"

Citado na Wikipédia, artigo Principle of least astonishment

# PRINCÍPIO DA MÍNIMA SURPRESA

---

*“Se um recurso é acidentalmente aplicado incorretamente pelo usuário e causa um resultado não previsto, o recurso apresenta um alto fator de surpresa e, portanto, é indesejável. Se um recurso necessário tiver um alto fator de surpresa, pode ser necessário redesenhar o recurso.”*

— Cowlishaw, M. F. (1984). "The design of the REXX language"

Citado na Wikipédia, artigo Principle of least astonishment

# SOLUÇÃO PARA ESSE CASO: COPIAR A SLICE DE NOMES

```
1 package main
2
3 import "fmt"
4
5 // TwilightBus drops passengers to the 5th dimension
6 type TwilightBus struct {
7     passengers []string
8 }
9
10 // New builds and returns a pointer to a TwilightBus
11 func New(passengers []string) *TwilightBus {
12     bus := TwilightBus{}
13     bus.passengers = passengers
14
15     return &bus
16 }
17
18 // Pick picks a named passenger
19 func (bus *TwilightBus) Pick(name string) {
20     bus.passengers = append(bus.passengers, name)
21 }
22
23 // find needle in slice, returns index, found bool
24 func find(slice []string, needle string) (int, bool) {
25     for i, item := range slice {
26         if item == needle {
27             return i, true
28     }
29 }
```

```
1 package main
2
3 import "fmt"
4
5 // Bus picks and drops passengers
6 type Bus struct {
7     passengers []string
8 }
9
10 // New builds and returns a pointer to a Bus
11 func New(passengers []string) *Bus {
12     bus := Bus{}
13     bus.passengers = make([]string, len(passengers))
14     copy(bus.passengers, passengers)
15
16     return &bus
17 }
18
19 // Pick picks a named passenger
20 func (bus *Bus) Pick(name string) {
21     bus.passengers = append(bus.passengers, name)
22 }
23
24 // find needle in slice, returns index, found bool
25 func find(slice []string, needle string) (int, bool) {
26     for i, item := range slice {
27         if item == needle {
28             return i, true
29     }
30 }
```

# MODELOS DE VARIÁVEIS EM LINGUAGENS

---

*O que a gente vê por aí*

# MODELOS DE VARIÁVEIS EM ALGUMAS LINGUAGENS

---

	valores	ponteiros	referências
C	✓	✓	
C++	✓	✓	✓
Java	✓		✓
JavaScript			✓
Python			✓
Go	✓	✓	“✓”



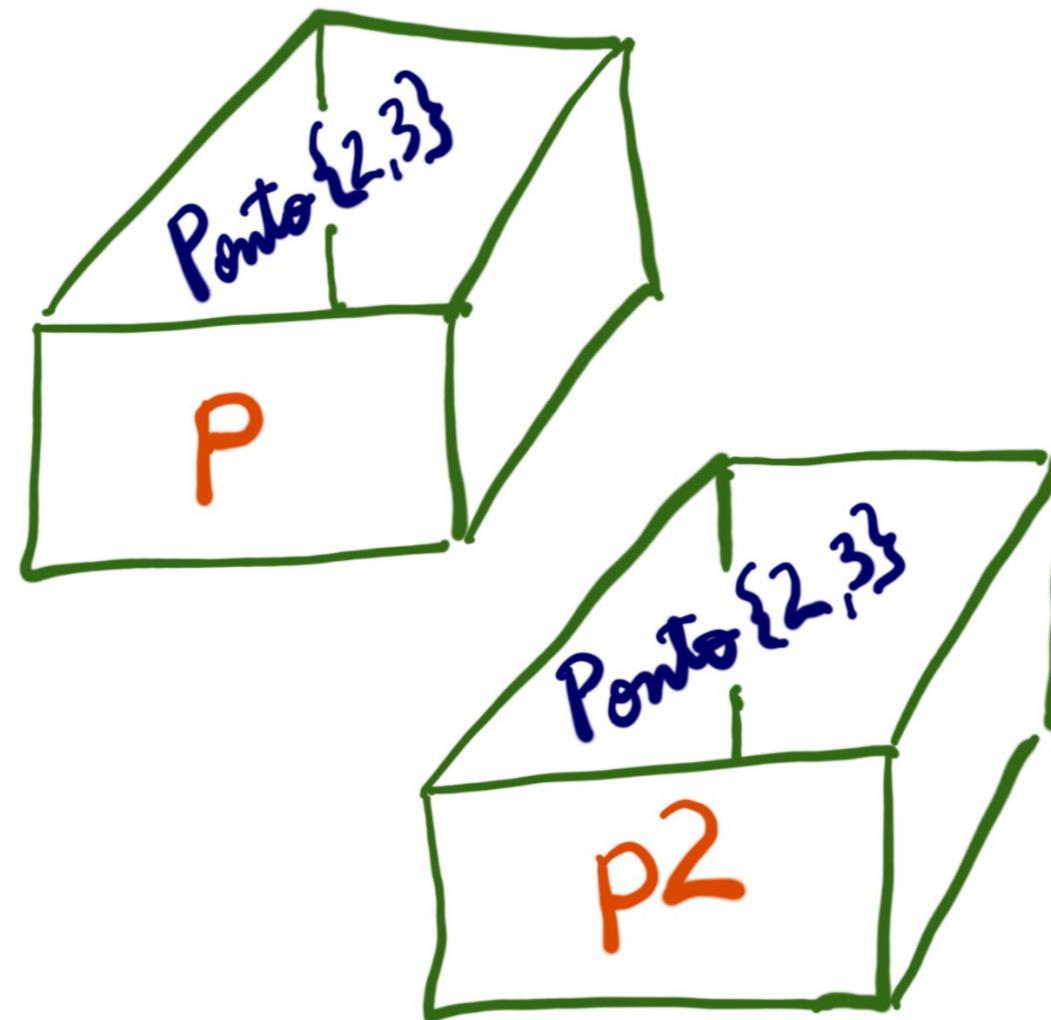
# VARIÁVEIS EM GO

---

*Comportamentos diferentes*

# EM GO, VARIÁVEIS SÃO “CAIXAS”

---



```
p := Ponto{2, 3}  
p2 := p  
p2.y++  
fmt.Printf("p\t%#v\np2\t%#v\n", p, p2)
```

```
p    main.Ponto{x:2, y:3}  
p2   main.Ponto{x:2, y:4}
```

# VARIÁVEIS STRUCT, INT E ARRAY SÃO “CAIXAS”

---

```
p := Ponto{2, 3}  
p2 := p  
p2.y++  
fmt.Printf("p\t%#v\np2\t%#v\n", p, p2)
```

```
p    main.Ponto{x:2, y:3}  
p2   main.Ponto{x:2, y:4}
```

# VARIÁVEIS STRUCT, INT E ARRAY SÃO “CAIXAS”

---

```
p := Ponto{2, 3}  
p2 := p  
p2.y++  
fmt.Printf("p\t%#v\np2\t%#v\n", p, p2)
```

```
p    main.Ponto{x:2, y:3}  
p2   main.Ponto{x:2, y:4}
```

```
i := 3  
i2 := i  
i2++  
fmt.Printf("i\t%#v\ni2\t%#v\n", i, i2)
```

```
i    3  
i2   4
```

# VARIÁVEIS STRUCT, INT E ARRAY SÃO “CAIXAS”

---

```
p := Ponto{2, 3}  
p2 := p  
p2.y++  
fmt.Printf("p\t%v\np2\t%v\n", p, p2)
```

```
p    main.Ponto{x:2, y:3}  
p2   main.Ponto{x:2, y:4}
```

```
i := 3  
i2 := i  
i2++  
fmt.Printf("i\t%v\ni2\t%v\n", i, i2)
```

```
i    3  
i2   4
```

```
a := [...]int{1, 2, 3}  
a2 := a  
a2[0]++  
fmt.Printf("a\t%v\na2\t%v\n", a, a2)
```

```
a    [3]int{1, 2, 3}  
a2   [3]int{2, 2, 3}
```

# **"GO TEM SEMÂNTICA DE VALORES"**

---

- Variáveis são áreas de memória que contém os bytes representando os dados em si.
- Não ocorre *aliasing* (apelidamento)
- Atribuição faz cópia dos dados.
- Parâmetros recebidos por funções são cópias dos argumentos passados.
- A função pode alterar sua cópia, mas não tem como alterar os dados do cliente (quem a invocou).

**ThoughtWorks®**

# PONTEIROS

---

*Uma rápida introdução*

# O OPERADOR & (ENDEREÇO) DEVOLVE UM PONTEIRO

---

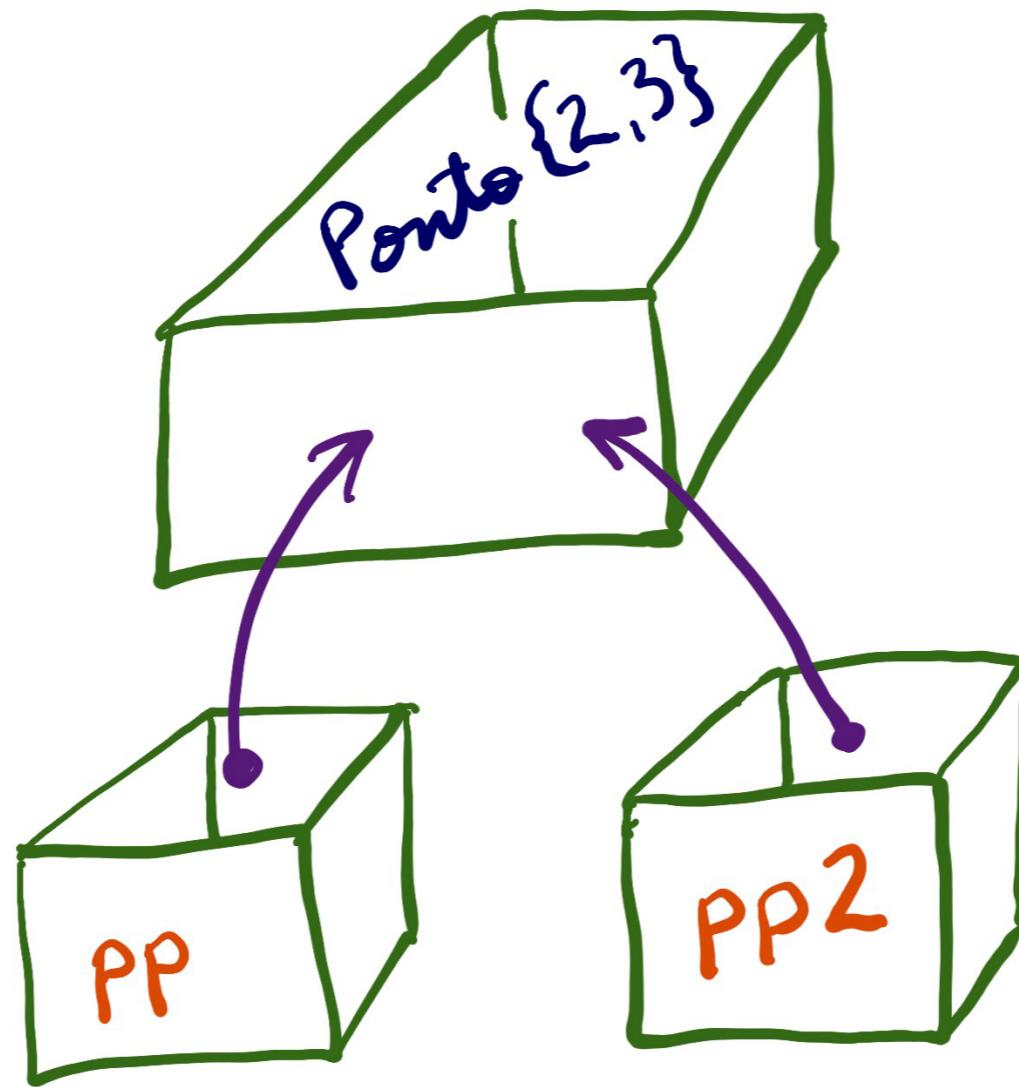
```
pp := &Ponto{2, 3}
pp2 := pp
pp2.y++
fmt.Printf("pp\t%v\npp2\t%v\n\n", pp, pp2)
```

```
pp &main.Ponto{x:2, y:4}
pp2 &main.Ponto{x:2, y:4}
```

# CAIXAS PP E PP2 TÊM PONTEIROS PARA A MESMA CAIXA

```
pp1 := &Ponto{2, 3}  
pp2 := pp1  
pp2.y++  
fmt.Printf("pp\t%#v\npp2\t%#v\n\n", pp1, pp2)
```

```
pp1 &main.Ponto{x:2, y:4}  
pp2 &main.Ponto{x:2, y:4}
```



# SINTAXE DE PONTEIROS: &X, PX, \*PX

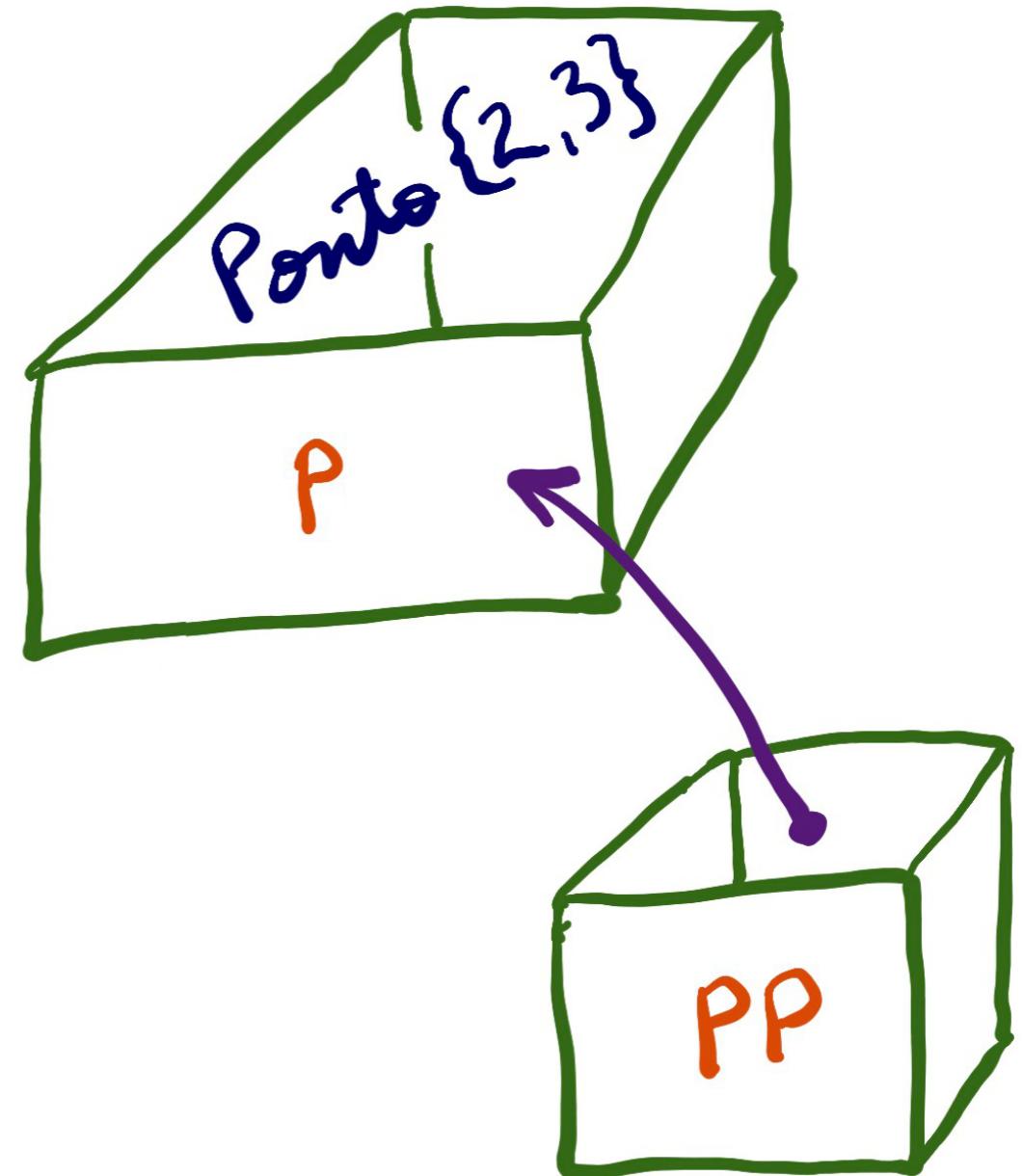
```
type Ponto struct {
    x, y float64
}

func main() {
    p := Ponto{2, 3}
    fmt.Printf("p\t%#v\n\n", p)

    → var pp *Ponto
    → pp = new(Ponto)
        fmt.Printf("pp\t%#v\n", pp)
        fmt.Printf("\t(%p)\n\n", pp)

    → pp = &p
        fmt.Printf("pp\t%#v\n", pp)
        fmt.Printf("\t(%p)\n\n", pp)

    fmt.Printf("*pp\t%#v\n\n", *pp)
}
```



# FORMATO %P MOSTRA O PONTEIRO EM SI, NÃO SEU ALVO

```
type Ponto struct {
    x, y float64
}

func main() {
    p := Ponto{2, 3}
    fmt.Printf("p\t%#v\n\n", p)

    var pp *Ponto

    pp = new(Ponto)
    fmt.Printf("pp\t%#v\n", pp)
    fmt.Printf("\t(%p)\n\n", pp)

    pp = &p
    fmt.Printf("pp\t%#v\n", pp)
    fmt.Printf("\t(%p)\n\n", pp)

    fmt.Printf("*pp\t%#v\n\n", *pp)
}
```

p	main.Ponto{x:2, y:3}
pp	&main.Ponto{x:0, y:0} (0xc0000140c0)
pp	&main.Ponto{x:2, y:3} (0xc000014080)
*pp	main.Ponto{x:2, y:3}

# EU LEIO \*P ASSIM: "A COISA APONTADA POR P" (O ALVO)

```
type Ponto struct {
    x, y float64
}

func main() {
    p := Ponto{2, 3}
    fmt.Printf("p\t%#v\n\n", p)

    var pp *Ponto

    pp = new(Ponto)
    fmt.Printf("pp\t%#v\n", pp)
    fmt.Printf("\t(%p)\n\n", pp)

    pp = &p
    fmt.Printf("pp\t%#v\n", pp)
    fmt.Printf("\t(%p)\n\n", pp)

    fmt.Printf("*pp\t%#v\n\n", *pp)
}
```

```
p    main.Ponto{x:2, y:3}

pp    &main.Ponto{x:0, y:0}
      (0xc0000140c0)

pp    &main.Ponto{x:2, y:3}
      (0xc000014080)

*pp   main.Ponto{x:2, y:3}
```



## PONTEIROS EM GO

---

Ao contrário de C, C++, e Pascal, Go tem ponteiros mas também tem um GC (garbage collector).

A pessoa que programa em Go não precisa manualmente alocar e liberar memória.

O compilador gera código de apoio que supervisiona o uso de ponteiros para saber quais estruturas de dados podem ser descartadas.

O valor de um ponteiro não é fixo: o alvo pode ser realocado e o valor do ponteiro será atualizado automaticamente.

## **“REFERÊNCIAS”**

---

*Aliasing em Go*

# O QUE HÁ NESSAS “CAIXAS”?

---

```
s := []int{1, 2, 3}
s2 := s
s2[0]++
fmt.Printf("s\t%#v\ns2\t%#v\n\n", s, s2)
```

```
s    []int{2, 2, 3}
s2   []int{2, 2, 3}
```

```
m := map[byte]int{1: 1, 2: 2, 3: 3}
m2 := m
m2[3]++
fmt.Printf("m\t%#v\nm2\t%#v\n\n", m, m2)
```

```
m    map[uint8]int{0x2:2, 0x3:4, 0x1:1}
m2   map[uint8]int{0x1:1, 0x2:2, 0x3:4}
```

## ALGUNS EXEMPLOS DE “ALIASING”

---

Aliasing é literalmente “apelidamento”: ocorre quando vários nomes ou apelidos referem-se à mesma coisa.

```
pp := &Ponto{2, 3}  
pp2 := pp  
pp2.y++
```

```
pp  &main.Ponto{x:2, y:4}  
pp2 &main.Ponto{x:2, y:4}
```

```
s := []int{1, 2, 3}  
s2 := s  
s2[0]++
```

```
s  []int{2, 2, 3}  
s2 []int{2, 2, 3}
```

```
m := map[byte]int{1: 1, 2: 2, 3: 3}  
m2 := m  
m2[3]++
```

```
m  map[uint8]int{0x2:2, 0x3:4, 0x1:1}  
m2 map[uint8]int{0x1:1, 0x2:2, 0x3:4}
```

## AS PEGADINHAS

---

"Referências" em Go são implícitas.

Ponteiros têm sintaxe explícita (`&x`, `*p`) mas valores com referências **não têm sintaxe explícita**.

Somente 3 tipos nativos mutáveis usam referências:

- slice
  - map
  - channel
- } As únicas estruturas de dados construídas com **make()**



Strings também usam referências, mas são imutáveis.

Você não pode criar seus próprios tipos com referências.

# SEMÂNTICA DE VALORES × SEMÂNTICA DE PONTEIROS

---

*“Value semantics keep values on the stack, which reduces pressure on the Garbage Collector (GC). However, value semantics require various copies of any given value to be stored, tracked and maintained. Pointer semantics place values on the heap, which can put pressure on the GC. However, pointer semantics are efficient because only one value needs to be stored, tracked and maintained.”*

— Bill Kennedy, *Design Philosophy On Data And Semantics*

# SEMÂNTICA DE VALORES × SEMÂNTICA DE PONTEIROS

---

*“A semântica de valores mantém os valores na pilha, reduzindo a pressão no Garbage Collector (GC). No entanto, a semântica de valores exige que várias cópias de cada valor sejam armazenadas, rastreadas e mantidas. A semântica do ponteiro coloca valores no heap, o que pode pressionar o GC. No entanto, a semântica do ponteiro é eficiente porque apenas um valor precisa ser armazenado, rastreado e mantido.”*

*— Bill Kennedy, Design Philosophy On Data And Semantics*

# USO DE CÓPIAS OU PONTEIROS

---

*“Most of the time your ability to use value semantics is limiting. It isn't correct or reasonable to make copies of the data as it passes from function to function. Changes to the data need to be isolated to a single value and shared. This is when pointer semantics need to be used. If you are not 100% sure it is correct or reasonable to make copies, then use pointer semantics.”*

— Bill Kennedy, Design Philosophy On Data And Semantics

## USO DE CÓPIAS OU PONTEIROS

---

*“Na maioria das vezes, sua capacidade de usar a semântica de valores é limitada. Não é correto ou razoável fazer cópias dos dados à medida que passam de uma função para outra. Alterações nos dados precisam ser isoladas em um único valor e compartilhadas. É quando a semântica do ponteiro precisa ser usada. Se você não tiver 100% de certeza de que é correto ou razoável fazer cópias, use a semântica do ponteiro.”*

*— Bill Kennedy, Design Philosophy On Data And Semantics*

## USO DE CÓPIAS OU PONTEIROS

---

*“Na maioria das vezes, sua capacidade de usar a semântica de valores é limitada. ~~Não é correto ou razoável fazer cópias dos dados à medida que passam de uma função para outra.~~ Alterações nos dados precisam ser isoladas em um único valor e compartilhadas. É quando a semântica do ponteiro precisa ser usada. ~~Se você não tiver 100% de certeza de que é correto ou razoável fazer cópias, use a semântica do ponteiro.”~~*

— Bill Kennedy, Design Philosophy On Data And Semantics

## USO DE CÓPIAS OU PONTEIROS

---

“Na maioria das vezes, sua capacidade de usar a semântica de valores é limitada. ~~Não é correto ou razoável fazer cópias dos dados à medida que passam de uma função para outra.~~  
Alterações nos dados precisam ser isoladas em um único valor e compartilhadas. É quando a semântica do ponteiro precisa ser usada. ~~Se você não tiver 100% de certeza de que é correto ou razoável fazer cópias, use a semântica do ponteiro.”~~

— Bill Kennedy, Design Philosophy On Data And Semantics

Otimização prematura!

A photograph showing four people in an office environment. A woman with glasses and a dark top is on the left, looking down. In the center, a man is leaning forward, also looking down. To his right, another man is smiling broadly. On the far right, a fourth person's face is partially visible, looking towards the camera. They appear to be working on a computer screen, with papers and a keyboard visible on the desk.

ThoughtWorks®

# EXEMPLOS SIMPLES

---

O que acontece na prática

# TRIPLICADOR DE VALORES (SUPER ÚTIL ;-)

---

```
package main

import "fmt"

func triInt(x int) int {
    x *= 3
    return x
}

func triIntUpdate(x *int) int {
    *x *= 3
    return *x
}

func triArray(x [5]int) [5]int {
    for i := range(x) {
        x[i] *= 3
    }
    return x
}

func triSliceUpdate(x []int) []int {
    for i := range(x) {
        x[i] *= 3
    }
    return x
}

func triArrayUpdate(x *[5]int) [5]int {
    for i := range(x) {
        x[i] *= 3
    }
    return *x
}

func triIntVariadic(x ...int) []int {
    for i := range(x) {
        x[i] *= 3
    }
    return x
}
```

Código-fonte deste exemplo: [tgo.li/2UtD7Xe](https://tgo.li/2UtD7Xe)

# TRIPLICADOR DE VALORES (SUPER ÚTIL ;-)

---

```
func main() {
    x1 := 2
    fmt.Printf("triInt\t\t%v\t", x1)
    fmt.Printf("%v\t%v\n", triInt(x1), x1)
    x2 := [...]int{10, 20, 30, 40, 50}
    fmt.Printf("triArray\t%v\t", x2)
    fmt.Printf("%v\t%v\n", triArray(x2), x2)
    x3 := []int{10, 20, 30, 40, 50}
    fmt.Printf("triSliceUpdate\t%v\t", x3)
    fmt.Printf("%v\t%v\n", triSliceUpdate(x3), x3)
    x4 := 4
    x4ptr := &x4
    fmt.Printf("triIntUpdate\t%v\t", x4)
    fmt.Printf("%v\t%v\n", triIntUpdate(x4ptr), x4)
    x5 := [...]int{10, 20, 30, 40, 50}
    x5ptr := &x5
    fmt.Printf("triArrayUpdate\t%v\t", x5)
    fmt.Printf("%v\t%v\n", triArrayUpdate(x5ptr), x5)
    x6, x7, x8 := 100, 200, 300
    fmt.Printf("triIntVariadic\t%v, %v, %v\t", x6, x7, x8)
    fmt.Printf("%v\t%v, %v, %v\n", triIntVariadic(x6, x7, x8), x6, x7, x8)
    x9 := []int{10, 20, 30, 40, 50}
    fmt.Printf("triIntVariadic\t%v\t", x9)
    fmt.Printf("%v\t%v\n", triIntVariadic(x9...), x9)
}
```

# TRIPLICADOR DE VALORES (SUPER ÚTIL ;-)

```
package main

import "fmt"

func triInt(x int) int {
    x *= 3
    return x
}

func triIntUpdate(x *int) int {
    *x *= 3
    return *x
}

func triArray(x [5]int) [5]int {
    for i := range(x) {
        x[i] *= 3
    }
    return x
}

func triSliceUpdate(x []int) []int {
    for i := range(x) {
        x[i] *= 3
    }
    return x
}

func triArrayUpdate(x *[5]int) [5]int {
    for i := range(x) {
        x[i] *= 3
    }
    return *x
}

func triIntVariadic(x ...int) []int {
    for i := range(x) {
        x[i] *= 3
    }
    return x
}
```

triInt	2	6	2
triArray	[10 20 30 40 50]	[30 60 90 120 150]	[10 20 30 40 50]
triSliceUpdate	[10 20 30 40 50]	[30 60 90 120 150]	[30 60 90 120 150]
triIntUpdate	4	12	12
triArrayUpdate	[10 20 30 40 50]	[30 60 90 120 150]	[30 60 90 120 150]
triIntVariadic	100, 200, 300	[300 600 900]	100, 200, 300
triIntVariadic	[10 20 30 40 50]	[30 60 90 120 150]	[30 60 90 120 150]

ThoughtWorks®

# ANATOMIA DE SLICES

Examinando um tipo de referência por dentro.

# ANALISADOR DE SLICE

Inspirado em  
post de Bill  
Kennedy:  
“Understanding  
slices”

[tgo.li/2QjwoR3](https://tgo.li/2QjwoR3)

Código-fonte  
deste exemplo:

[tgo.li/2L7xNEQ](https://tgo.li/2L7xNEQ)

```
package main

import (
    "fmt"
    "unsafe"
)

func InspectSlice(intSlice []int) {

    fmt.Println("intSlice:")
    fmt.Printf("\t%#v\n\n", intSlice)

    // Get slicePtr of slice structure
    slicePtr := unsafe.Pointer(&intSlice)
    ptrSize := unsafe.Sizeof(slicePtr)

    // Compute addresses of len and cap
    lenAddr := uintptr(slicePtr) + ptrSize
    capAddr := uintptr(slicePtr) + (ptrSize * 2)

    // Create pointers to len and cap
    lenPtr := (*int)(unsafe.Pointer(lenAddr))
    capPtr := (*int)(unsafe.Pointer(capAddr))

    // Get pointer to underlying array
    // How to do this without hardcoding the array size?
    arrayPtr := (*[100]int)(unsafe.Pointer(*(*uintptr)(slicePtr)))

    fmt.Println("intSlice:")

    // Not using %T on next line to show expected data array size
    // fmt.Printf("\t@%p: data %T = %p\n", slicePtr, arrayPtr, arrayPtr)
    fmt.Printf("\t@%p: data *[%d]int = %p\n", slicePtr, *capPtr, arrayPtr)

    fmt.Printf("\t@%p: len %T = %d\n", lenPtr, *lenPtr, *lenPtr)
    fmt.Printf("\t@%p: cap %T = %d\n", capPtr, *capPtr, *capPtr)

    fmt.Println("data:")

    for index := 0; index < *capPtr; index++ {
        fmt.Printf("\t@%p: [%d] %T = %d\n",
            &(*arrayPtr)[index], index, (*arrayPtr)[index], (*arrayPtr)[index])
    }
}

func main() {
    intSlice := make([]int, 3, 5)
    intSlice[0] = 11
    intSlice[1] = 12
    intSlice[2] = 13

    InspectSlice(intSlice)

    for _, n := range []int{140, 150, 160} {
        intSlice = append(intSlice, n)
        InspectSlice(intSlice)
    }
}
```

```
intSlice:
[]int{11, 12, 13}

intSlice:
@0xc00000a060: data *[5]int = 0xc000072030
@0xc00000a068: len int = 3
@0xc00000a070: cap int = 5
data:
@0xc000072030: [0] int = 11
@0xc000072038: [1] int = 12
@0xc000072040: [2] int = 13
@0xc000072048: [3] int = 0
@0xc000072050: [4] int = 0

intSlice:
[]int{11, 12, 13, 140}

intSlice:
@0xc00000a0a0: data *[5]int = 0xc000072030
@0xc00000a0a8: len int = 4
@0xc00000a0b0: cap int = 5
data:
@0xc000072030: [0] int = 11
@0xc000072038: [1] int = 12
@0xc000072040: [2] int = 13
@0xc000072048: [3] int = 140
@0xc000072050: [4] int = 0

intSlice:
[]int{11, 12, 13, 140, 150}

intSlice:
@0xc00000a0e0: data *[5]int = 0xc000072030
@0xc00000a0e8: len int = 5
@0xc00000a0f0: cap int = 5
data:
@0xc000072030: [0] int = 11
@0xc000072038: [1] int = 12
@0xc000072040: [2] int = 13
@0xc000072048: [3] int = 140
@0xc000072050: [4] int = 150

intSlice:
[]int{11, 12, 13, 140, 150, 160}

intSlice:
@0xc00000a120: data *[10]int = 0xc0000180f0
@0xc00000a128: len int = 6
@0xc00000a130: cap int = 10
data:
@0xc0000180f0: [0] int = 11
@0xc0000180f8: [1] int = 12
@0xc000018100: [2] int = 13
@0xc000018108: [3] int = 140
@0xc000018110: [4] int = 150
@0xc000018118: [5] int = 160
@0xc000018120: [6] int = 0
@0xc000018128: [7] int = 0
@0xc000018130: [8] int = 0
@0xc000018138: [9] int = 0

intSlice:
[]int{11, 12, 13, 140, 150, 160, 170}

intSlice:
@0xc00000a160: data *[10]int = 0xc0000180f0
@0xc00000a168: len int = 7
@0xc00000a170: cap int = 10
data:
@0xc0000180f0: [0] int = 11
@0xc0000180f8: [1] int = 12
@0xc000018100: [2] int = 13
@0xc000018108: [3] int = 140
@0xc000018110: [4] int = 150
@0xc000018118: [5] int = 160
@0xc000018120: [6] int = 170
@0xc000018128: [7] int = 0
@0xc000018130: [8] int = 0
@0xc000018138: [9] int = 0
```

# ANALISADOR DE SLICE: MAIN

---

```
func main() {
    intSlice := make([]int, 3, 5)
    intSlice[0] = 11
    intSlice[1] = 12
    intSlice[2] = 13

    InspectSlice(intSlice)

    for _, n := range []int{140, 150, 160} {
        intSlice = append(intSlice, n)
        InspectSlice(intSlice)
    }
}
```

Código-fonte  
deste exemplo:

[tgo.li/2L7xNEQ](https://tgo.li/2L7xNEQ)

# ANALISADOR DE SLICE: INSPECT SLICE

---

```
func InspectSlice(intSlice []int) {  
  
    fmt.Println("intSlice:")  
    fmt.Printf("\t %#v\n\n", intSlice)  
  
    // Get slicePtr of slice structure  
    slicePtr := unsafe.Pointer(&intSlice)  
    ptrSize := unsafe.Sizeof(slicePtr)  
  
    // Compute addresses of len and cap  
    lenAddr := uintptr(slicePtr) + ptrSize  
    capAddr := uintptr(slicePtr) + (ptrSize * 2)  
  
    // Create pointers to len and cap  
    lenPtr := (*int)(unsafe.Pointer(lenAddr))  
    capPtr := (*int)(unsafe.Pointer(capAddr))  
  
    // Get pointer to underlying array  
    arrayPtr := (*[100]int)(unsafe.Pointer(*(*uintptr)(slicePtr)))  
  
    fmt.Println("intSlice:")  
  
    fmt.Printf("\t@%p: data *[%d]int = %p\n", slicePtr, *capPtr, arrayPtr)  
    fmt.Printf("\t@%p: len %T = %d\n", lenPtr, *lenPtr, *lenPtr)  
    fmt.Printf("\t@%p: cap %T = %d\n", capPtr, *capPtr, *capPtr)  
  
    fmt.Println("data:")  
    for index := 0; index < *capPtr; index++ {  
        fmt.Printf("\t@%p: [%d] %T = %d\n",  
            &(*arrayPtr)[index], index, (*arrayPtr)[index], (*arrayPtr)[index])  
    }  
}
```

Código-fonte  
deste exemplo:  
[tgo.li/2L7xNEQ](https://tgo.li/2L7xNEQ)

# ANALISADOR DE SLICE: MAIN

```
func main() {  
    intSlice := make([]int, 3, 5)  
    intSlice[0] = 11  
    intSlice[1] = 12  
    intSlice[2] = 13  
  
    → InspectSlice(intSlice)  
  
    for _, n := range []int{140, 150, 160} {  
        intSlice = append(intSlice, n)  
        InspectSlice(intSlice)  
    }  
}
```

*Array subjacente  
(underlying array)* →

[tgo.li/2L7xNEQ](https://tgo.li/2L7xNEQ)

Slice é um struct com três campos:  
**data, len, cap**

```
intSlice:  
[]int{11, 12, 13}  
  
intSlice:  
@0xc00000a060: data *[5]int = 0xc000072030  
@0xc00000a068: len int = 3  
@0xc00000a070: cap int = 5  
data:  
@0xc000072030: [0] int = 11  
@0xc000072038: [1] int = 12  
@0xc000072040: [2] int = 13  
@0xc000072048: [3] int = 0  
@0xc000072050: [4] int = 0
```

# ANALISADOR DE SLICE: MAIN

```
func main() {
    intSlice := make([]int, 3, 5)
    intSlice[0] = 11
    intSlice[1] = 12
    intSlice[2] = 13

    InspectSlice(intSlice)

    for _, n := range []int{140, 150, 160} {
        intSlice = append(intSlice, n)
    }
}
```



```
intSlice:
    []int{11, 12, 13, 140}

intSlice:
    @0xc00000a0a0: data *[5]int = 0xc000072030
    @0xc00000a0a8: len int = 4
    @0xc00000a0b0: cap int = 5
data:
    @0xc000072030: [0] int = 11
    @0xc000072038: [1] int = 12
    @0xc000072040: [2] int = 13
    @0xc000072048: [3] int = 140
    @0xc000072050: [4] int = 0
```

# ANALISADOR DE SLICE: MAIN

```
func main() {  
    intSlice := make([]int, 3, 5)  
    intSlice[0] = 11  
    intSlice[1] = 12  
    intSlice[2] = 13  
  
    → InspectSlice(intSlice)  
  
    for _, n := range []int{140, 150}  
        intSlice = append(intSlice, n)  
    → InspectSlice(intSlice)  
}  
}
```

```
intSlice:  
    []int{11, 12, 13}  
  
intSlice:  
    @0xc00000a060: data *[5]int = 0xc000072030  
    @0xc00000a068: len int = 3  
    @0xc00000a070: cap int = 5  
data:  
    @0xc000072030: [0] int = 11  
    @0xc000072038: [1] int = 12  
    @0xc000072040: [2] int = 13  
    @0xc000072048: [3] int = 0  
    @0xc000072050: [4] int = 0  
  
intSlice:  
    []int{11, 12, 13, 140}  
  
intSlice:  
    @0xc00000a0a0: data *[5]int = 0xc000072030  
    @0xc00000a0a8: len int = 4  
    @0xc00000a0b0: cap int = 5  
data:  
    @0xc000072030: [0] int = 11  
    @0xc000072038: [1] int = 12  
    @0xc000072040: [2] int = 13  
    @0xc000072048: [3] int = 140  
    @0xc000072050: [4] int = 0
```

# ANALISADOR DE SLICE

Ao fazer **append**, quando a capacidade inicial é ultrapassada, um novo array subjacente é criado com o dobro da capacidade, e o conteúdo anterior é copiado para lá.

Código-fonte deste exemplo:

[tgo.li/2L7xNEQ](https://tgo.li/2L7xNEQ)

```
intSlice:  
    []int{11, 12, 13, 140, 150}  
  
intSlice:  
    @0xc00000a0e0: data *[5]int = 0xc000072030  
    @0xc00000a0e8: len int = 5  
    @0xc00000a0f0: cap int = 5 ←  
data:  
    @0xc000072030: [0] int = 11  
    @0xc000072038: [1] int = 12  
    @0xc000072040: [2] int = 13  
    @0xc000072048: [3] int = 140  
    @0xc000072050: [4] int = 150  
  
intSlice:  
    []int{11, 12, 13, 140, 150, 160}  
  
intSlice:  
    @0xc00000a120: data *[10]int = 0xc0000180f0  
    @0xc00000a128: len int = 6  
    @0xc00000a130: cap int = 10 ←  
data:  
    @0xc0000180f0: [0] int = 11  
    @0xc0000180f8: [1] int = 12  
    @0xc000018100: [2] int = 13  
    @0xc000018108: [3] int = 140  
    @0xc000018110: [4] int = 150  
    @0xc000018118: [5] int = 160  
    @0xc000018120: [6] int = 0  
    @0xc000018128: [7] int = 0  
    @0xc000018130: [8] int = 0  
    @0xc000018138: [9] int = 0
```

**ThoughtWorks®**

# **CONCLUSÃO**

---

# SEMÂNTICA DE VALORES X SEMÂNTICA DE REFERÊNCIAS

## Semântica de valores

Variáveis são áreas de memória que contém os bits representando os dados em si.

Não ocorre aliasing.

Atribuição faz cópia dos dados.

Parâmetros recebidos por funções são cópias dos argumentos passados: a função pode alterar sua cópia, mas não tem como alterar os dados do código cliente.

## Semântica de referências

Variáveis contém apenas referências ou ponteiros que apontam para os dados alocados em outra parte da memória.

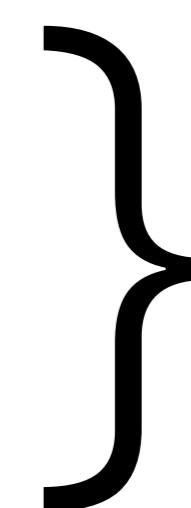
Pode ocorrer aliasing: mais de uma referência ao mesmo dado.

Atribuição faz cópia da referência ou ponteiro; os dados são compartilhados.

Parâmetros recebidos por funções são referências para os dados do cliente: a função pode alterar os dados do código cliente.

# TAMANHOS EM BYTES E VALORES ZERO

Tipo	unsafe.Sizeof()	Valor “zero”
string	16 $\rightarrow$	""
int	8	0
float32	4	0
[3]float32	12	[3]float32{0, 0, 0}
*[3]float32	8	(*[3]float32)(nil)
[]float32	24 $\rightarrow$	[]float32(nil)
map[string]int	8 $\rightarrow$	map[string]int(nil)
chan uint8	8 $\rightarrow$	(chan uint8)(nil)



**nil** é o valor zero dos tipos que têm ponteiros

- Essa tabela é verdadeira para uma CPU de 64 bits, com ponteiros de 8 bytes.
- Os tamanhos com  $\rightarrow$  incluem um ponteiro oculto, mas não incluem os dados referenciados na string, slice, map e channel.

## UMA FORMA DE ENTENDER

---

Go adota a semântica de valores por padrão, mas em alguns casos o valor é uma "referência" a uma estrutura que têm um ponteiro oculto.

## DICAS FINAIS

---

É praticamente impossível programar em Go sem usar slices, mas as slices são o tipo de referência mais traiçoeiro da linguagem. Entenda a fundo como elas funcionam. Saiba que o array subjacente pode ser compartilhado e pode mudar a qualquer momento.

Cuidado ao passar ou receber qualquer tipo de referência mutável como argumento (slice, map, channel): a função pode mudar a estrutura de dados sem você saber. Se você é a autora da função, considere usar um nome que deixe isso explícito, por exemplo: **ReverseInPlace**, **RankUpdate**.

# MAIS REFERÊNCIAS

---

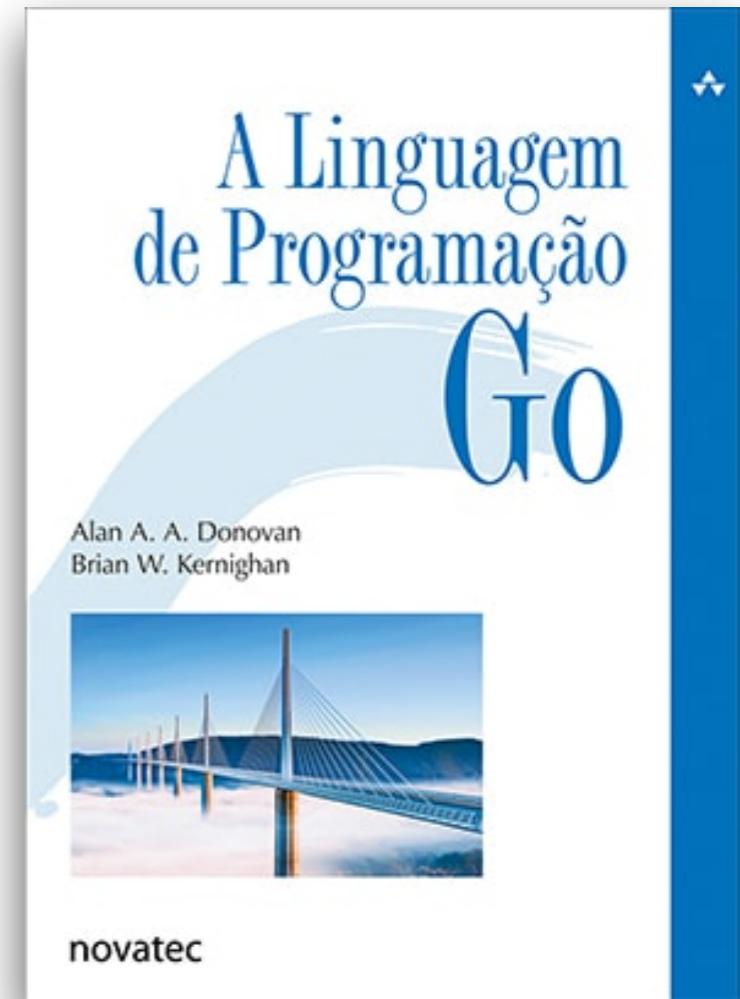
## GOPL: The Go Programming Language

— Donovan & Kernighan  
(A Linguagem de Programação Go,  
Ed. Novatec)

## Golang Wiki: SliceTricks

## Dave Cheney (<https://dave.cheney.net/>)

- Pointers in Go
- There is no pass-by-reference in Go
- If a map isn't a reference variable, what is it?
- Should methods be declared on T or \*T
- Slices from the ground up



# MUITO GRATO!

*Dúvidas ou sugestões?*

*Twitter: @ramalhoorg*

*E-mail: [luciano.ramalho@thoughtworks.com](mailto:luciano.ramalho@thoughtworks.com)*

**ThoughtWorks®**