UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Array e Lista

QXD0001 - Fundamentos de Programação

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Agenda

- Introdução
- Array em Go
- Percorrendo vetores
- Erros comuns
- Lista em Go

Introdução

Introdução

Por que precisamos de array?

- Imagine que você precisa escrever um programa para indicar a maior nota, a menor nota e a média dos alunos de um turma de tamanho qualquer ?
- E se você precisasse saber quantos alunos tiraram a maior nota?

Precisamos utilizar um Array

Introdução

O que é um Array?

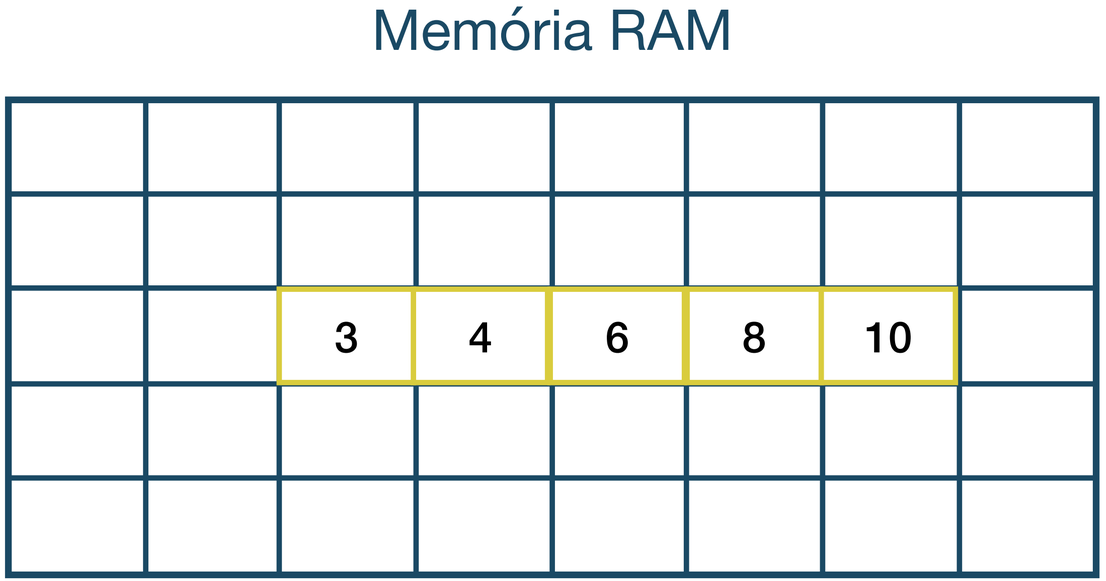
- Uma coleção de valores do mesmo tipo
- Permite guardar vários valores em uma única variável
- Os valores/elementos são armazenados de maneira ordenada
- Cada valor é identificado pelo seu índice

Introdução

Armazenamento em memória

- Ao criar um vetor, o computador reserva um bloco de endereços vizinhos

```
n := [5]float64{3, 4, 6, 8, 10}
```



Array em Go

Array em Go

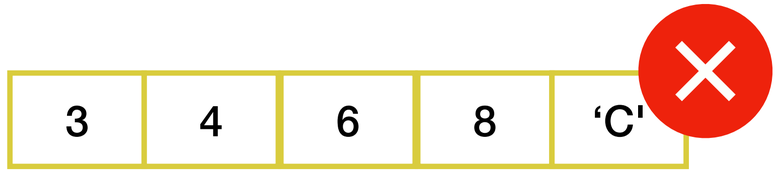
- Homogêneo
  - Todos os elementos devem ser do mesmo tipo

Array em Go

- Homogêneo
  - Todos os elementos devem ser do mesmo tipo

Array em Go

- Homogêneo
  - Todos os elementos devem ser do mesmo tipo



Array em Go

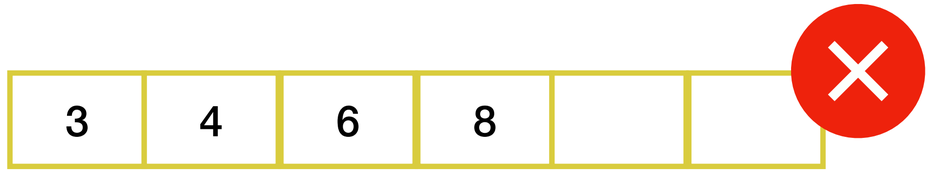
- Homogêneo
  - Todos os elementos devem ser do mesmo tipo
- Tamanho Fixo

Array em Go

- Homogêneo
  - Todos os elementos devem ser do mesmo tipo
- Tamanho Fixo
  - Uma vez criado com 10 posições, ele sempre terá 10 posições

Array em Go

- Homogêneo
  - Todos os elementos devem ser do mesmo tipo
- Tamanho Fixo
  - Uma vez criado com 10 posições, ele sempre terá 10 posições

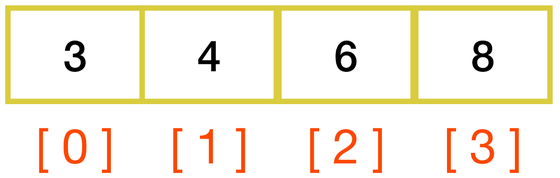


Array em Go

- Homogêneo
  - Todos os elementos devem ser do mesmo tipo
- Tamanho Fixo
  - Uma vez criado com 10 posições, ele sempre terá 10 posições

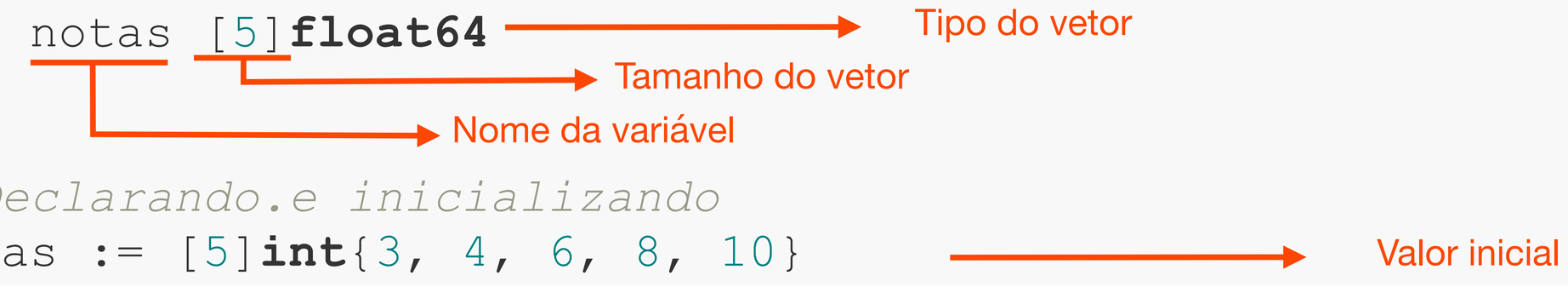
Array em Go

- Indexação
  - O primeiro índice é sempre a de número 0
  - O último é n-1
- Acesso Direto
  - Você pode acessar qualquer posição instantaneamente se souber o índice



Array em Go

```go
// Declaração
var notas [5]float64

//Declarando.e inicializando
notas := [5]int{3, 4, 6, 8, 10}
```

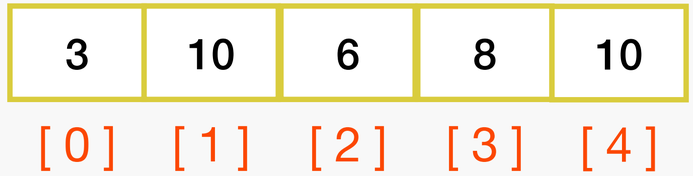


Array em Go

Acessando e alterando elementos

```go
notas := [5]int{3, 4, 6, 8, 10}

// Acessando elementos - leitura
fmt.Println(notas[0])
fmt.Println(notas[2])

// Modificando um elemento - escrita
notas[1] = 10.0
```

Índice



Percorrendo vetores

Percorrendo vetores

A forma tradicional

```
notas := [3]float64{10, 8.5, 7.0}

for i := 0; i < len(notas); i++ {
    fmt.Println("Nota do aluno", i, "é", notas[i])
}
```

Retorna o tamanho do vetor

Vai receber o valor dos índices do elementos do vetor

Percorrendo vetores

Usando o operador range

```
notas := [3]float64{10, 8.5, 7.0}

for indice, valor := range notas {
    fmt.Println("Nota do aluno", indice, "é", valor)
}

for _, valor := range notas {
    fmt.Println(valor)
}
```

O valor do elemento atual, ou seja, aquele armazenado na posição indica

Operador range

Vai receber o valor dos índices do elementos do vetor

Caso o índice não seja útil

Em Go, quando passamos um Array para uma função, o Go cria uma cópia (passagem por valor) de todo o array. Se o array for gigantesco, isso pode ser lento.

Erros comuns

Erros comuns

❌ Índice fora do limite

```
notas := [5]int{3, 4, 6, 8, 10}

fmt.Println(notas[10])❌
```

Erros comuns

❌ Erro 2— Tipos diferentes

```
prova1 := [5]int{3, 4, 6, 8, 10}
prova2 := [10]int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

var notas [5]int = prova2 ❌
```

Lista em Go

Lista em Go

Diferença Visual na Sintaxe:

```go
// Declarando um vetor
var notas [5]float64

//Declarando uma lista
var notas []float64
// Forma literal
numeros := []int{10, 20, 30}
```

Tipo da lista

Nome da variável

Por que usar Slices?

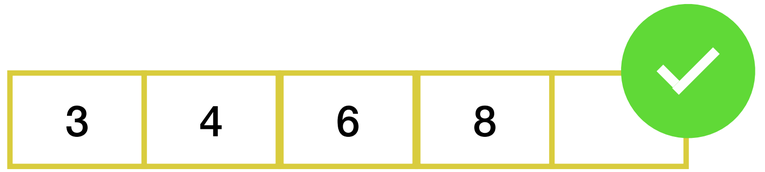
No Array clássico, precisamos definir o tamanho na hora de compilar (ex: [5]int). E se não tivermos como prever?

Lista em Go

- São chamadas de Slices
  - Todos os elementos devem ser do mesmo tipo
  - Tamanho flexível

Lista em Go

- São chamadas de Slices
  - Todos os elementos devem ser do mesmo tipo
  - Tamanho flexível
    - Pode ser alterado ao longo da execução



Lista em Go

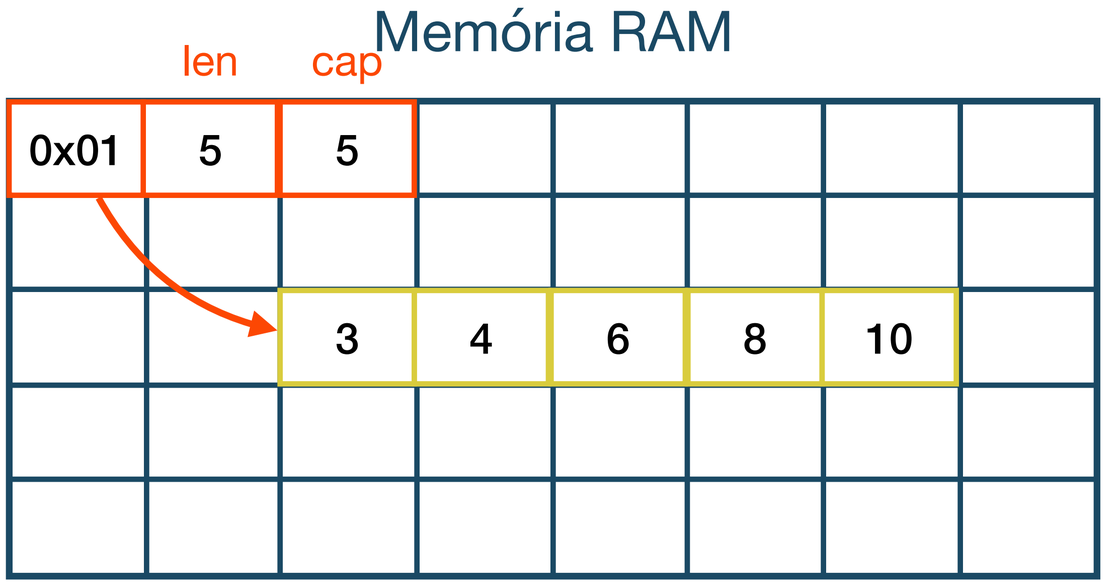
Anatomia de um slice

- É uma estrutura que guarda 3 coisas:
  - Ponteiro:
    - Aponta para o primeiro elemento de um Array oculto na memória
  - Tamanho (Length):
    - Quantos elementos estão no Slice atualmente
  - Capacidade (Capacity):
    - O tamanho máximo que o Array oculto suporta antes de precisar crescer

Lista em Go

Anatomia de um slice

```go
n := [ ]int{3, 4, 6, 8, 10}

// Tamanho de um slice
fmt.Println(len(n))

// Capacidade de um slice
fmt.Println(cap(n))
```



Como o Slice guarda o ponteiro, tamanho e capacidade, passá-lo para uma função é extremamente rápido e eficiente. Se a função alterarmos um elemento do Slice, essa alteração reflete no Slice original (Passagem por Referência implícita).

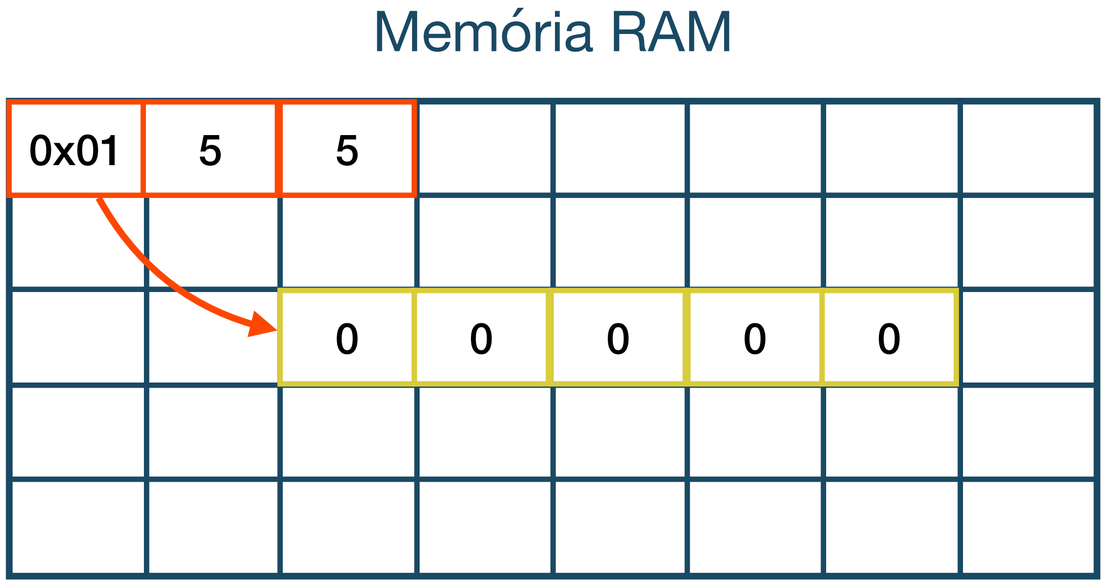
Lista em Go

Usando a função make

Lista em Go

Usando a função make

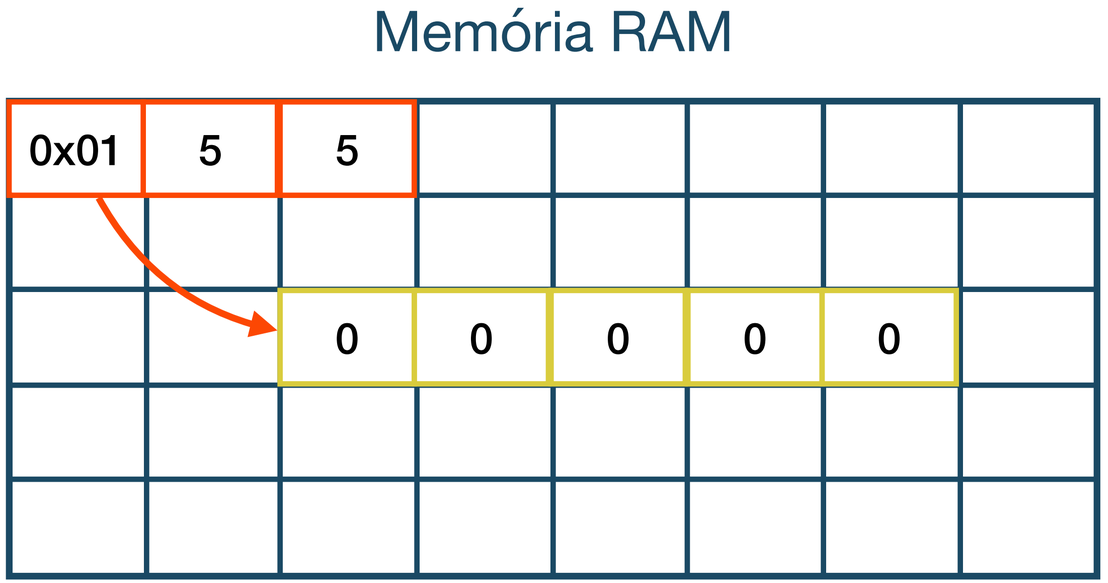
```
n := make([]int, 5)
```



Lista em Go

Usando a função make

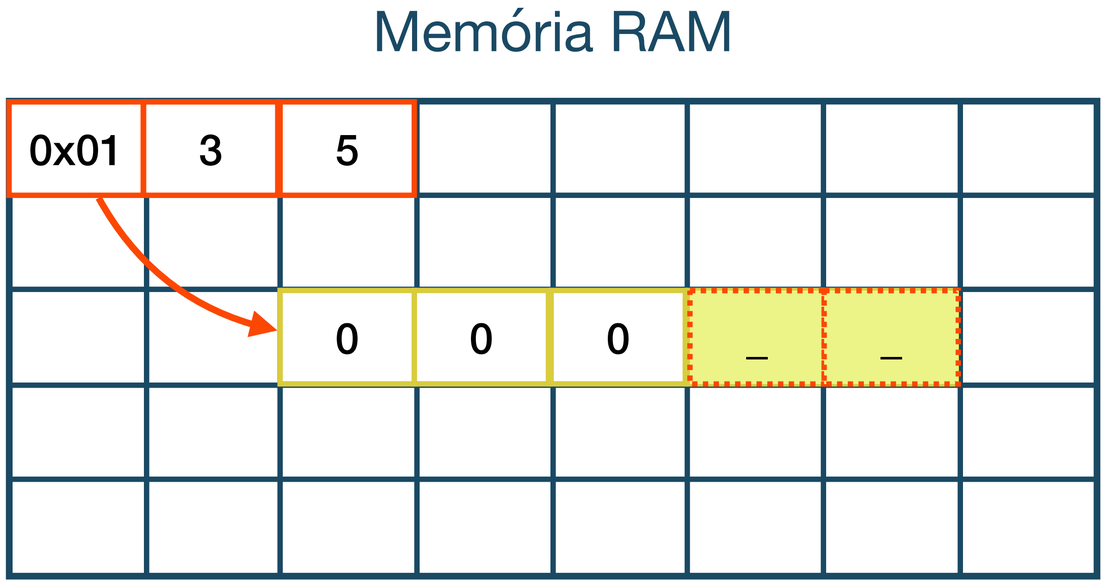
```go
n := make([]int, 5)

n := make([]int, 3, 5)
```



Lista em Go

Usando a função make

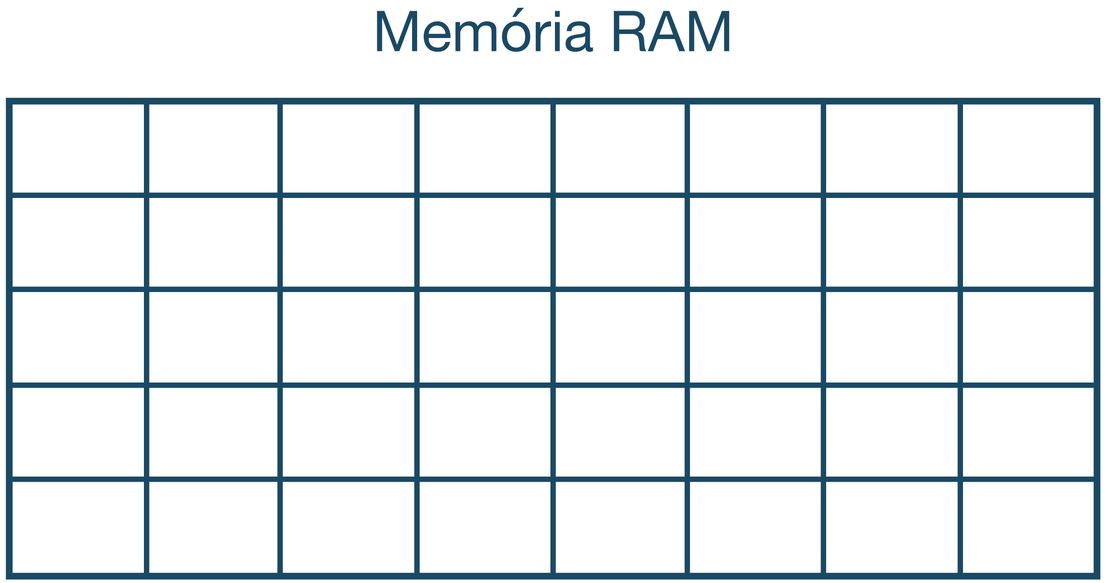
```go
n := make([]int, 5)

n := make([]int, 3, 5)
```



Lista em Go

Alterando o tamanho de um slice

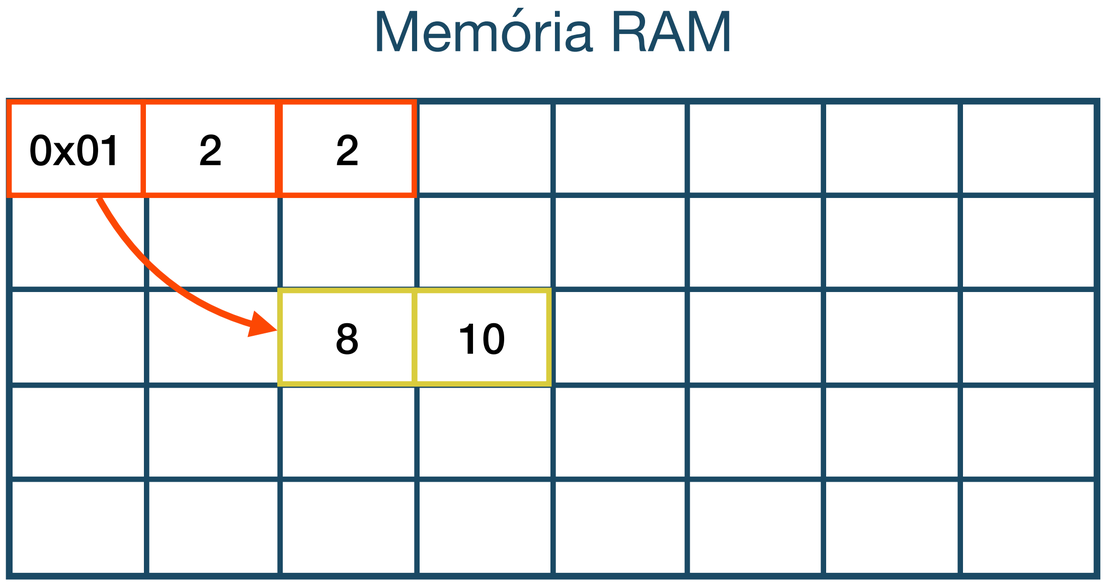
```
n := []int{8, 10}
```



Lista em Go

Alterando o tamanho de um slice

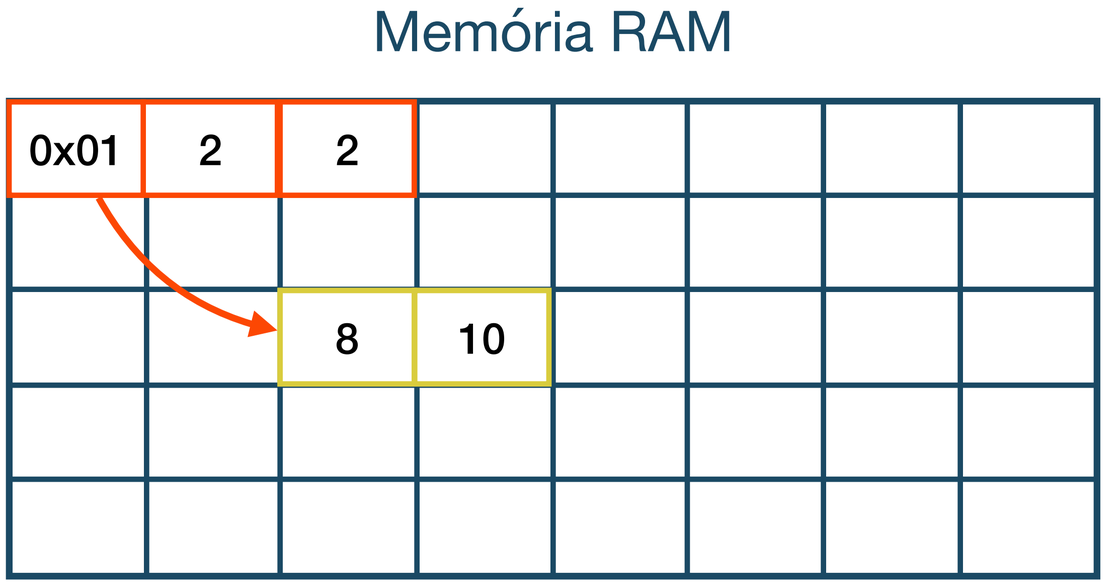
```
n := []int{8, 10}
```



Lista em Go

Alterando o tamanho de um slice

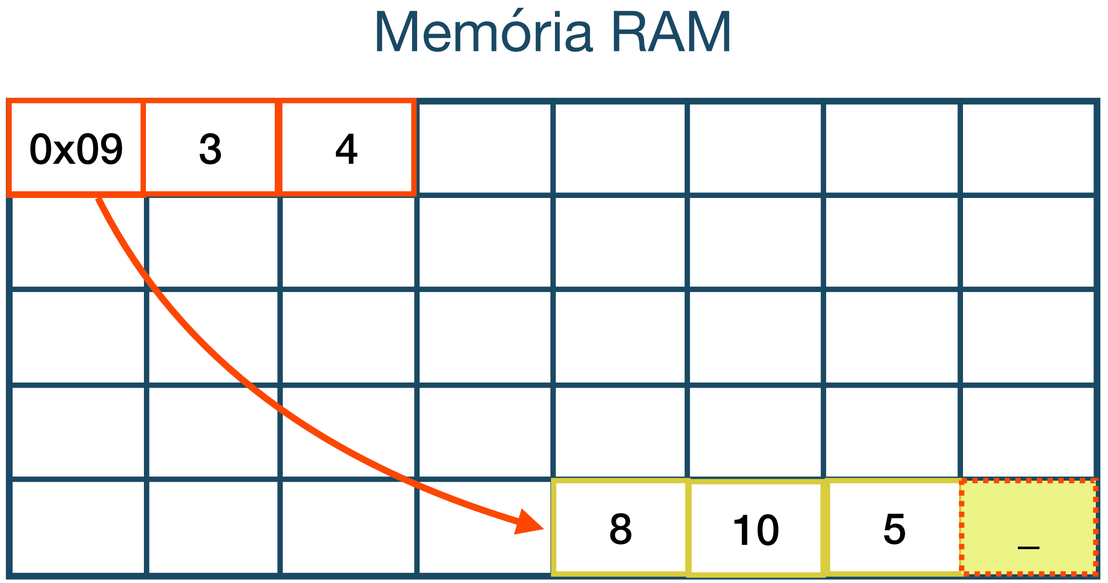
```
n := []int{8, 10}

n = append(n, 5)
```



Lista em Go

Alterando o tamanho de um slice

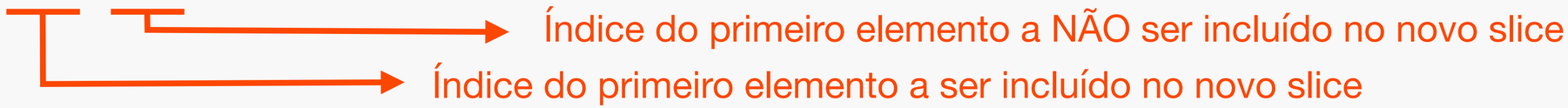
```
n := []int{8, 10}

n = append(n, 5)
```



Lista em Go

Fatiamento

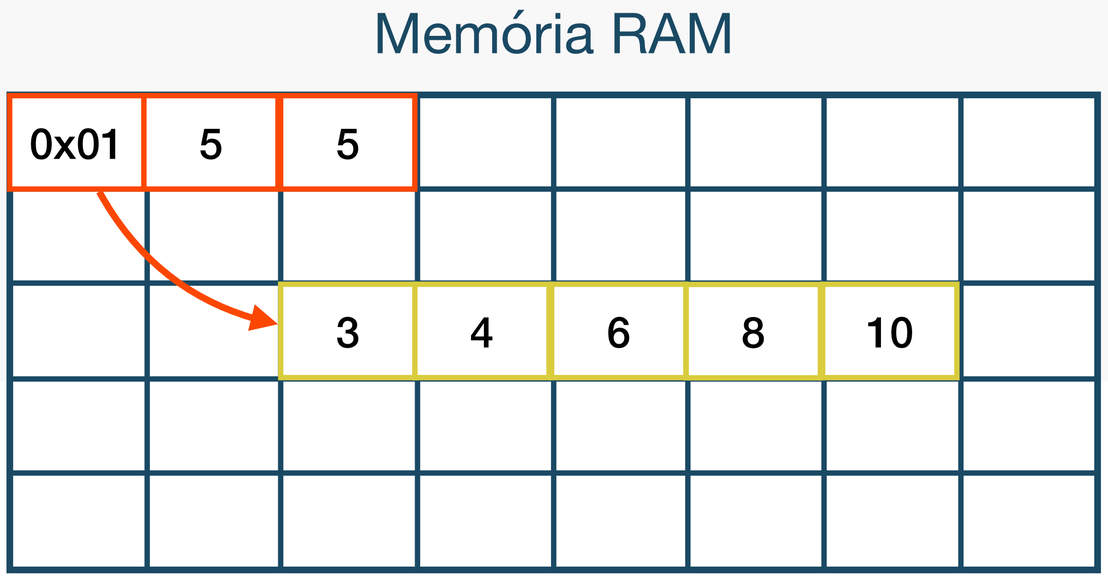
```
n := []int{3, 4, 6, 8, 10}

//Fatiamento - sintaxe
n[a : b]

//Podemos omitir valores
n[0:5]
n[:5]
n[0:]
n[:]
```



Lista em Go

Fatiamento

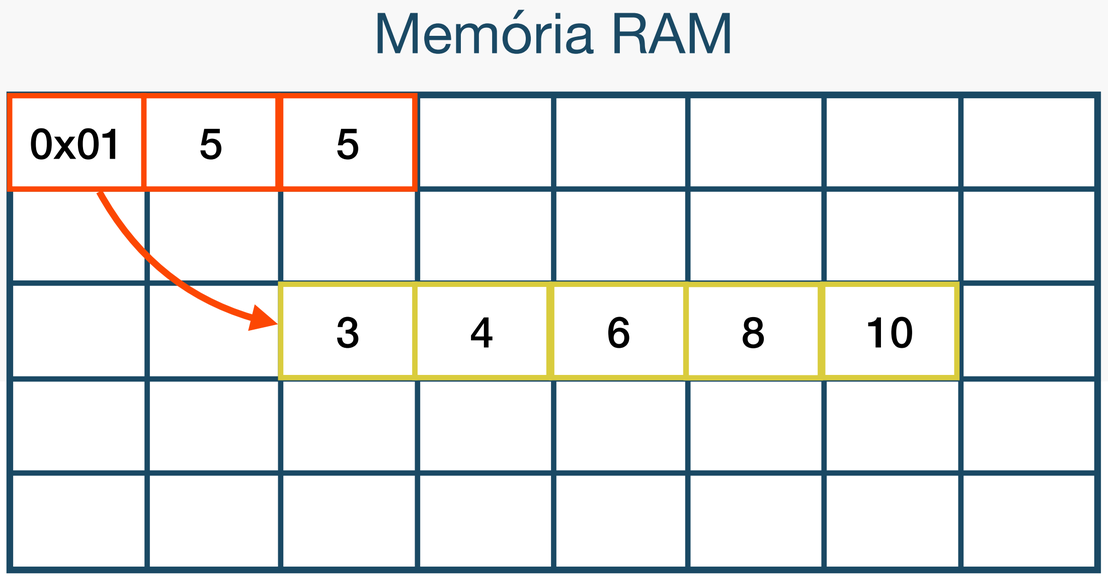
```
n := []int{3, 4, 6, 8, 10}
```



Lista em Go

Fatiamento

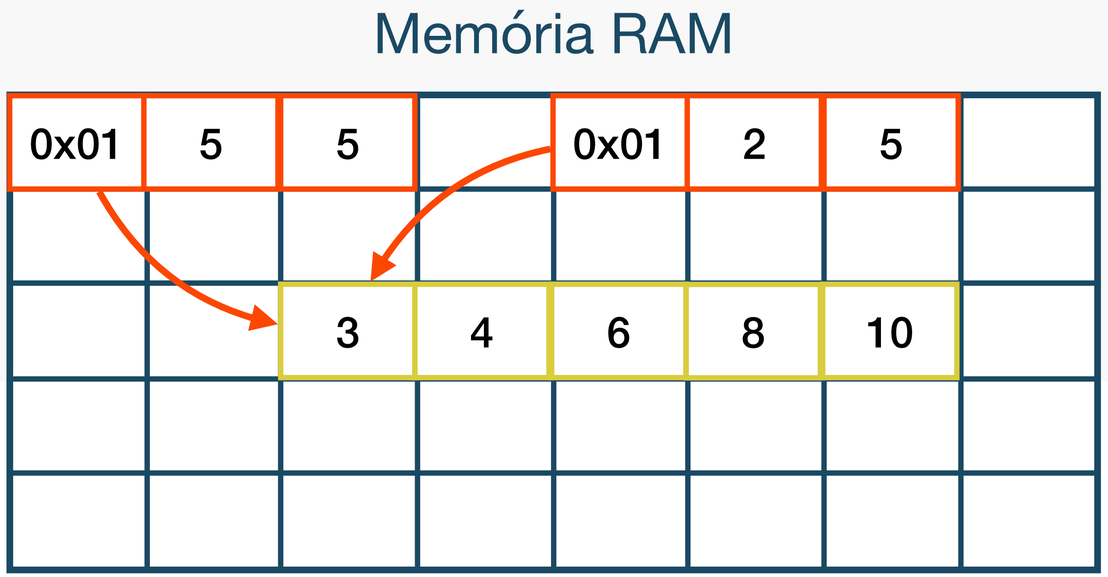
```go
n := []int{3, 4, 6, 8, 10}

menores := n[:2] // n[0:2]
```



Lista em Go

Fatiamento

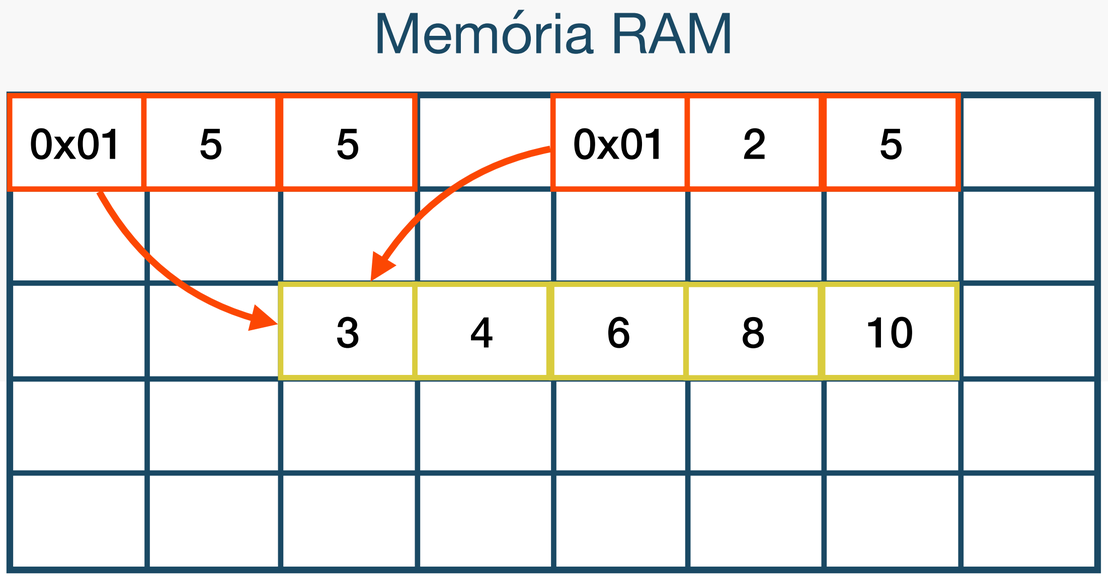
```go
n := []int{3, 4, 6, 8, 10}

menores := n[:2] // n[0:2]
```



Lista em Go

Fatiamento

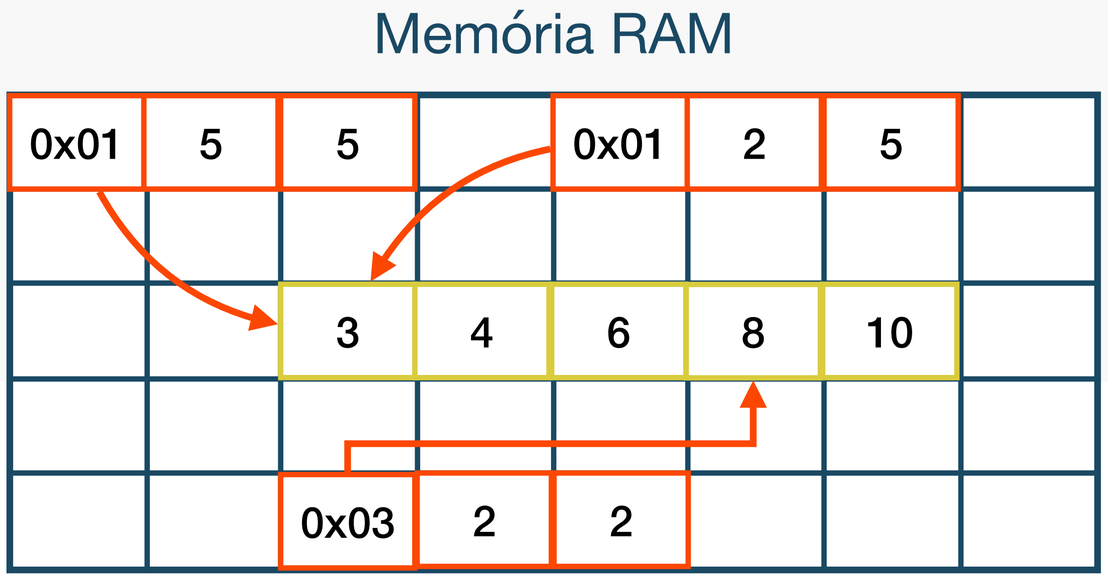
```go
n := []int{3, 4, 6, 8, 10}

menores := n[:2] // n[0:2]
maiores := n[3:] // n[3:5]
```



Lista em Go

Fatiamento

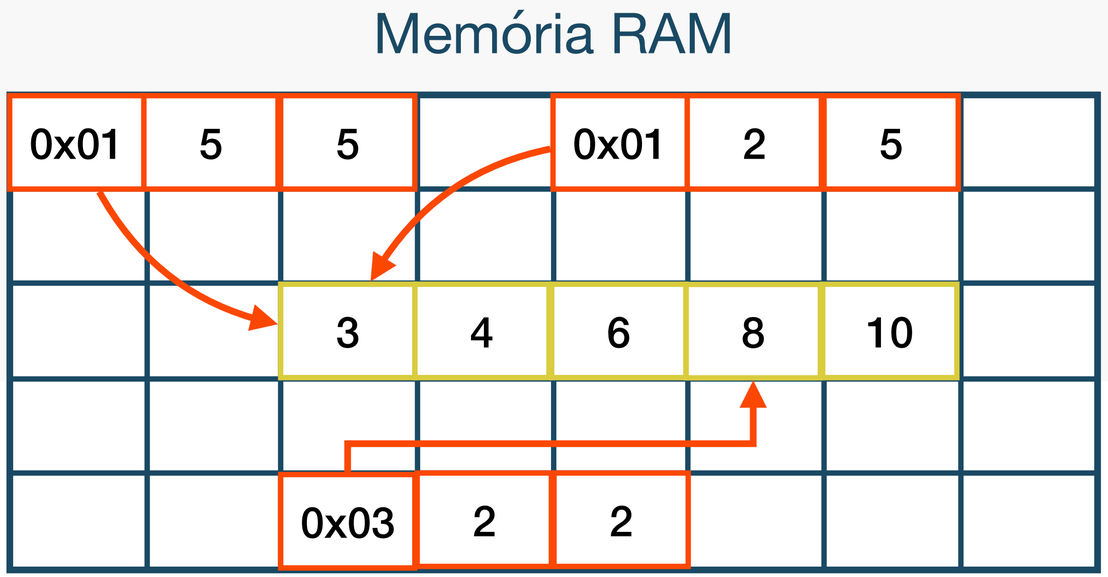
```go
n := []int{3, 4, 6, 8, 10}

menores := n[:2] // n[0:2]
maiores := n[3:] // n[3:5]
```



Lista em Go

Fatiamento

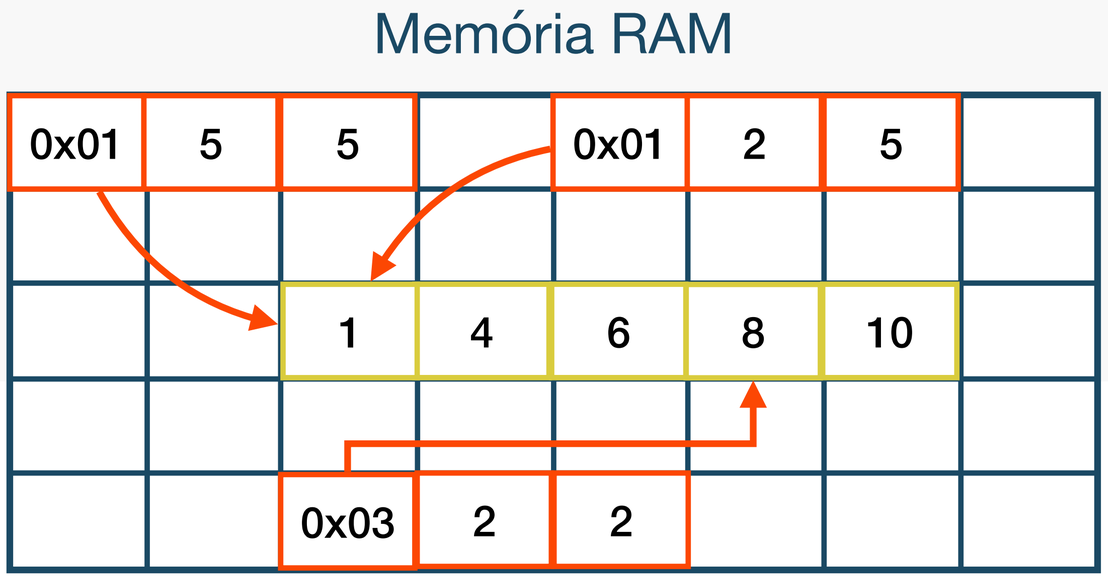
```go
n := []int{3, 4, 6, 8, 10}

menores := n[:2] // n[0:2]
maiores := n[3:] // n[3:5]

//⚠️ Cuidado com slice de slice
menores[0] = 1
```



Lista em Go

Fatiamento

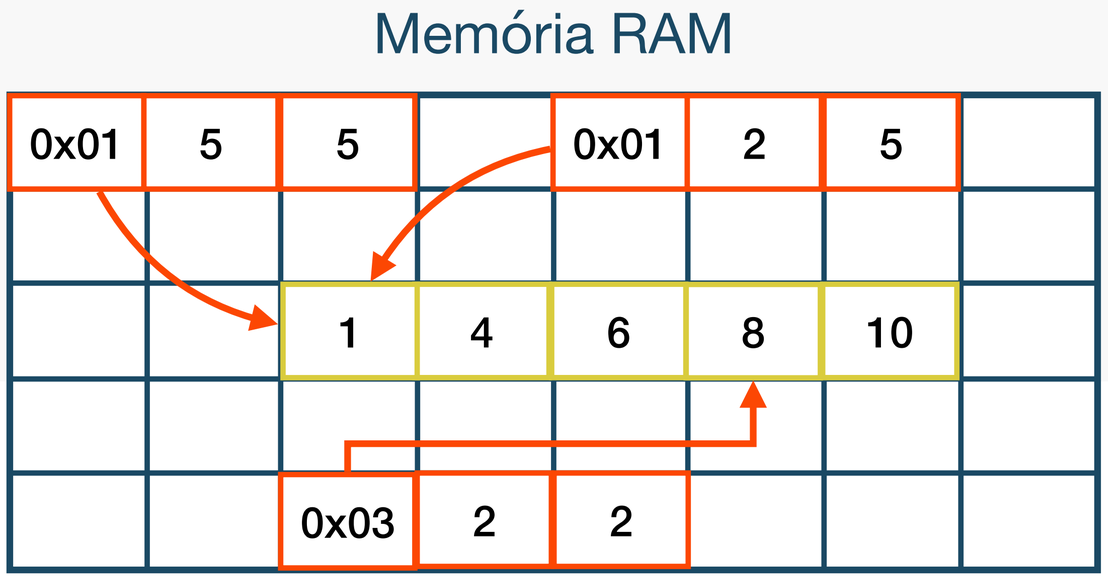
```go
n := []int{3, 4, 6, 8, 10}

menores := n[:2] // n[0:2]
maiores := n[3:] // n[3:5]

//⚠️ Cuidado com slice de slice
menores[0] = 1
```



Lista em Go

Fatiamento

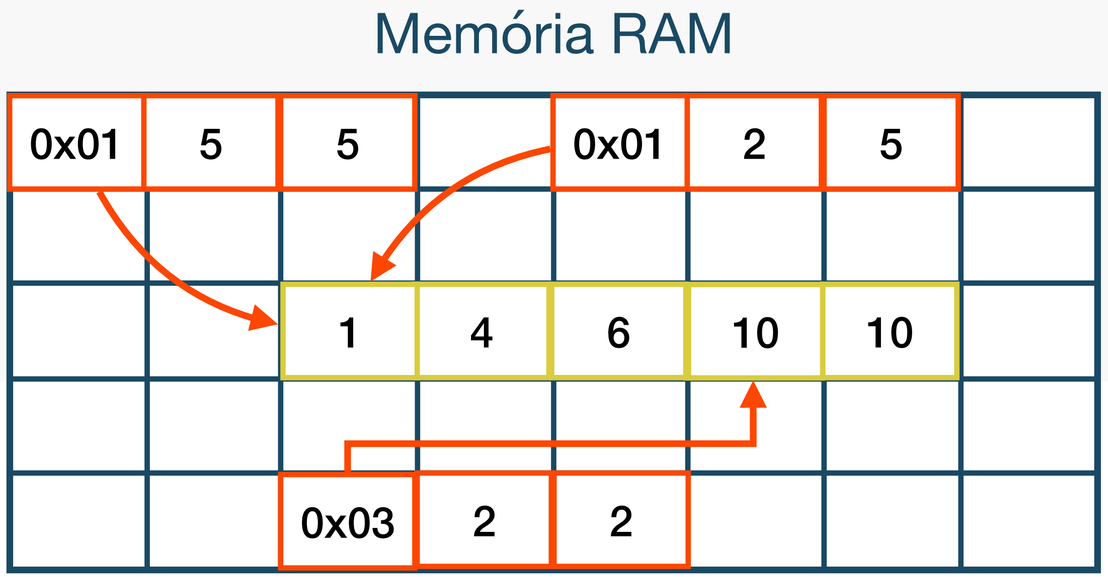
```go
n := []int{3, 4, 6, 8, 10}

menores := n[:2] // n[0:2]
maiores := n[3:] // n[3:5]

//⚠️ Cuidado com slice de slice
menores[0] = 1
maiores[0] = 10
```



Lista em Go

Fatiamento

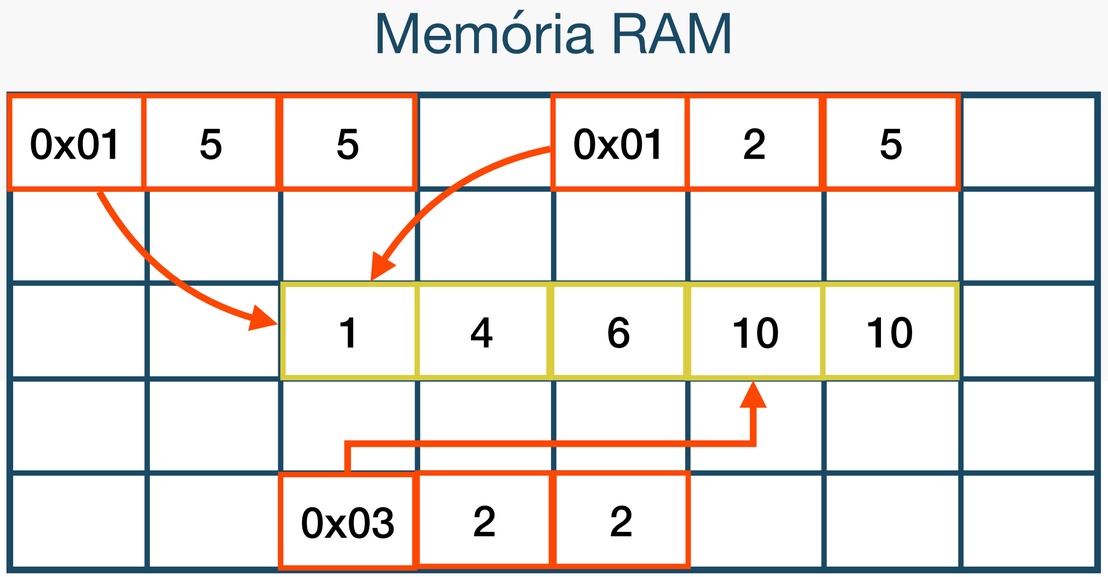
```go
n := []int{3, 4, 6, 8, 10}

menores := n[:2] // n[0:2]
maiores := n[3:] // n[3:5]

//⚠️ Cuidado com slice de slice
menores[0] = 1
maiores[0] = 10
```



Lista em Go

Fatiamento

```go
n := []int{3, 4, 6, 8, 10}

menores := n[:2] // n[0:2]
maiores := n[3:] // n[3:5]

//⚠️ Cuidado com slice de slice
menores[0] = 1
maiores[0] = 10

fmt.Println(n[0])
```



Cuidado ao Compartilhar Memória: Se você cria um sliceB fatiando um sliceA, ambos apontam para o mesmo Array na memória. Alterar sliceB altera sliceA! A Solução: Se quiser uma cópia independente, use a função copy().

Lista em Go

Principais funções do pacote slices

Função	Descrição
`slices.Sort(s)`	Ordena os elementos do slice in-place
`slices.Contains(s, v)`	Determina se v está contido no slice
`slices.Index(s, v)`	Retorna o índice da primeira ocorrência de v, caso contrário -1
`slices.Insert(s, i, v)`	Insere v na posição i. Se necessário o tamanho do slice é alterado.
`slices.Delete(s, i, j)`	Remove os elementos de i até j (exclusivo) e retorna o slice modificado
`slices.Reverse(s)`	Inverte a ordem do slice in-place
`slices.Clone(s)`	Retorna um cópia do slice

Lista em Go

Principais funções do pacote slices

Lista em Go

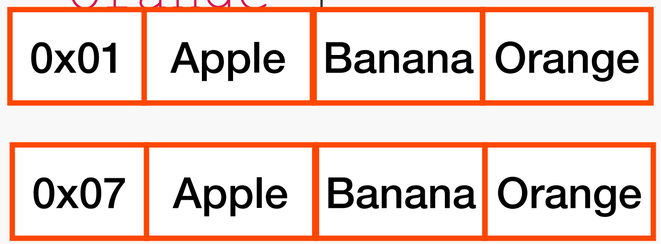
Principais funções do pacote slices

```
original := []string{"Apple", "Banana", "Orange"}
```

Lista em Go

Principais funções do pacote slices

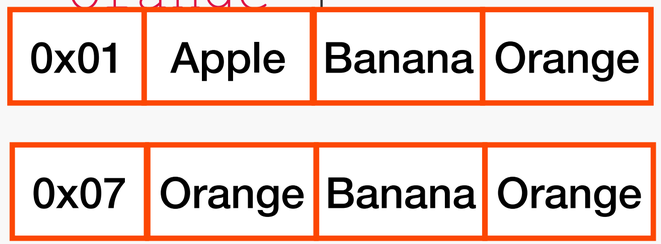
```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
```



Lista em Go

Principais funções do pacote slices

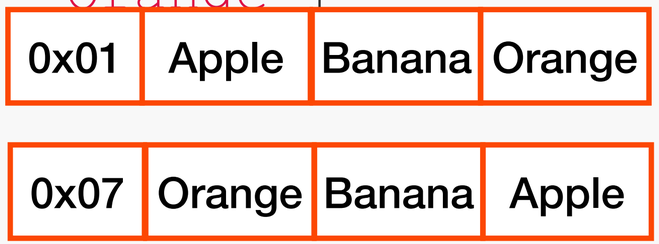
```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
```



Lista em Go

Principais funções do pacote slices

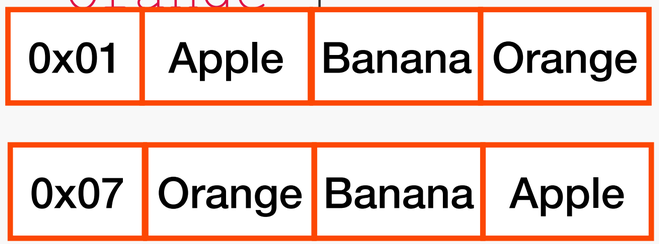
```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"
```



Lista em Go

Principais funções do pacote slices

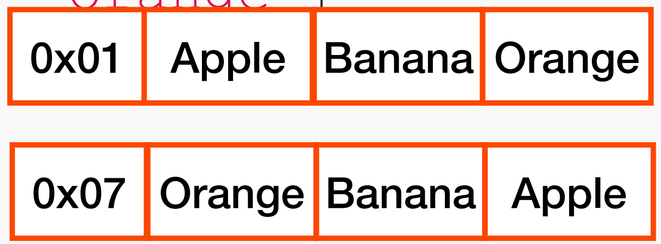
```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"

fmt.Println("Found Apple?", slices.Contains(s, "Apple"))
fmt.Println("Found Strawberry?", slices.Contains(s, "Strawberry"))
fmt.Println("Index for Banana:", slices.Index(s, "Banana"))
fmt.Println("Index for Strawberry:", slices.Index(s, "Strawberry"))
```



Lista em Go

Principais funções do pacote slices

```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"

fmt.Println("Found Apple?", slices.Contains(s, "Apple"))
fmt.Println("Found Strawberry?", slices.Contains(s, "Strawberry"))
fmt.Println("Index for Banana:", slices.Index(s, "Banana"))
fmt.Println("Index for Strawberry:", slices.Index(s, "Strawberry"))
s = slices.Insert(s, 1, "Grape")
```



Lista em Go

Principais funções do pacote slices

```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"

fmt.Println("Found Apple?", slices.Contains(s, "Apple"))
fmt.Println("Found Strawberry?", slices.Contains(s, "Strawberry"))
fmt.Println("Index for Banana:", slices.Index(s, "Banana"))
fmt.Println("Index for Strawberry:", slices.Index(s, "Strawberry"))
s = slices.Insert(s, 1, "Grape")
```

Lista em Go

Principais funções do pacote slices

```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"

fmt.Println("Found Apple?", slices.Contains(s, "Apple"))
fmt.Println("Found Strawberry?", slices.Contains(s, "Strawberry"))
fmt.Println("Index for Banana:", slices.Index(s, "Banana"))
fmt.Println("Index for Strawberry:", slices.Index(s, "Strawberry"))
s = slices.Insert(s, 1, "Grape")
slices.Sort(s)
```

Lista em Go

Principais funções do pacote slices

```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"

fmt.Println("Found Apple?", slices.Contains(s, "Apple"))
fmt.Println("Found Strawberry?", slices.Contains(s, "Strawberry"))
fmt.Println("Index for Banana:", slices.Index(s, "Banana"))
fmt.Println("Index for Strawberry:", slices.Index(s, "Strawberry"))
s = slices.Insert(s, 1, "Grape")
slices.Sort(s)
```

Lista em Go

Principais funções do pacote slices

```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"

fmt.Println("Found Apple?", slices.Contains(s, "Apple"))
fmt.Println("Found Strawberry?", slices.Contains(s, "Strawberry"))
fmt.Println("Index for Banana:", slices.Index(s, "Banana"))
fmt.Println("Index for Strawberry:", slices.Index(s, "Strawberry"))
s = slices.Insert(s, 1, "Grape")
slices.Sort(s)
slices.Reverse(s)
```

Lista em Go

Principais funções do pacote slices

```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"

fmt.Println("Found Apple?", slices.Contains(s, "Apple"))
fmt.Println("Found Strawberry?", slices.Contains(s, "Strawberry"))
fmt.Println("Index for Banana:", slices.Index(s, "Banana"))
fmt.Println("Index for Strawberry:", slices.Index(s, "Strawberry"))
s = slices.Insert(s, 1, "Grape")
slices.Sort(s)
slices.Reverse(s)
```

Lista em Go

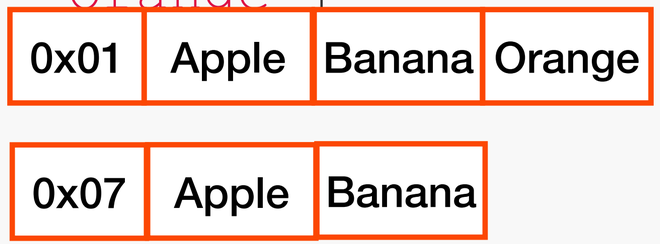
Principais funções do pacote slices

```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"

fmt.Println("Found Apple?", slices.Contains(s, "Apple"))
fmt.Println("Found Strawberry?", slices.Contains(s, "Strawberry"))
fmt.Println("Index for Banana:", slices.Index(s, "Banana"))
fmt.Println("Index for Strawberry:", slices.Index(s, "Strawberry"))
s = slices.Insert(s, 1, "Grape")
slices.Sort(s)
slices.Reverse(s)
s = slices.Delete(s, 0, 2)
```

Lista em Go

Principais funções do pacote slices

```
original := []string{"Apple", "Banana", "Orange"}
s := slices.Clone(original)
s[0] = "Orange"
s[2] = "Apple"

fmt.Println("Found Apple?", slices.Contains(s, "Apple"))
fmt.Println("Found Strawberry?", slices.Contains(s, "Strawberry"))
fmt.Println("Index for Banana:", slices.Index(s, "Banana"))
fmt.Println("Index for Strawberry:", slices.Index(s, "Strawberry"))
s = slices.Insert(s, 1, "Grape")
slices.Sort(s)
slices.Reverse(s)
s = slices.Delete(s, 0, 2)
```



Referências

- Go 101
- The Slices Package