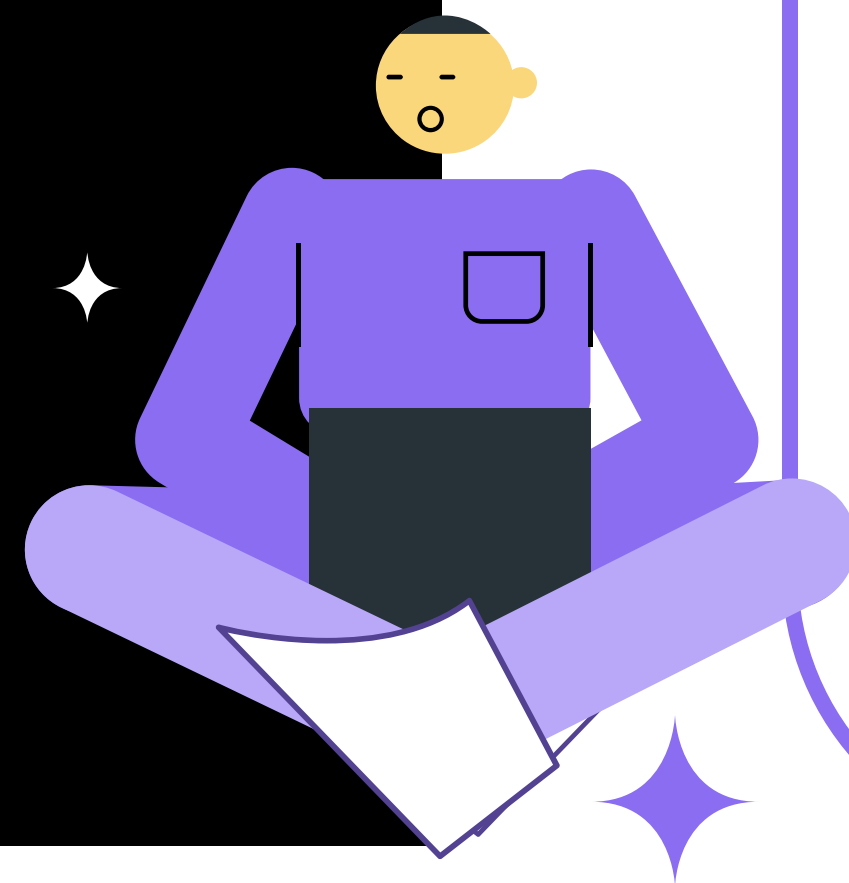


Coleções de dados



Os arrays e slices são **tipos de dados** que nos permitem **armazenar** um **conjunto homogêneo** de dados. Ou seja, dados que sejam do mesmo tipo.



Sumário

- 01 [Arrays](#)
- 02 [Slices](#)
- 03 [Maps](#)



01

Arrays

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var a [2]string
```

```
    a[0] = "Hello"
```

```
    a[1] = "World"
```

```
    fmt.Println(a[0], a[1])
```

```
    fmt.Println(a)
```

```
}
```

Declaração

Para declarar um array devemos definir um tamanho e tipo de dado.

Tamanho **Tipo de dado**

{ }

var a [2]string

Declara uma variável “a” como um array de duas strings.

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var a [2]string
```

```
    a[0] = "Hello"
```

```
    a[1] = "World"
```

```
    fmt.Println(a[0], a[1])
```

```
    fmt.Println(a)
```

```
}
```

Atribuir valores

Para atribuir um valor a um array, é preciso especificar a posição seguida do valor.

Posição

Valor

{ }

a[0] = "Hello"

a[1] = "World"

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var a [2]string
```

```
    a[0] = "Hello"
```

```
    a[1] = "World"
```

```
    fmt.Println(a[0], a[1])
```

```
    fmt.Println(a)
```

```
}
```

Obter valores

Para obter o valor de um array, basta especificar o nome da variável e a posição que você quer obter:

```
{}
```

```
fmt.Printf(a[0], a[1])
```

```
fmt.Println(a)
```

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var a [2]string
```

```
    a[0] = "Hello"
```

```
    a[1] = "World"
```

```
    fmt.Println(a[0], a[1])
```

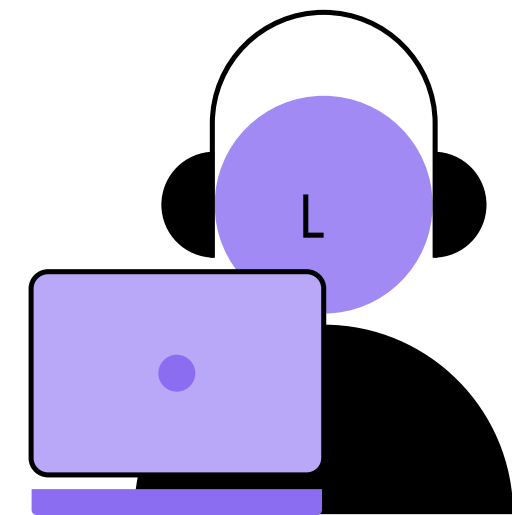
```
    fmt.Println(a)
```

```
}
```

O comprimento de um array faz parte do seu tipo, de modo que não é possível alterar o tamanho dos arrays.



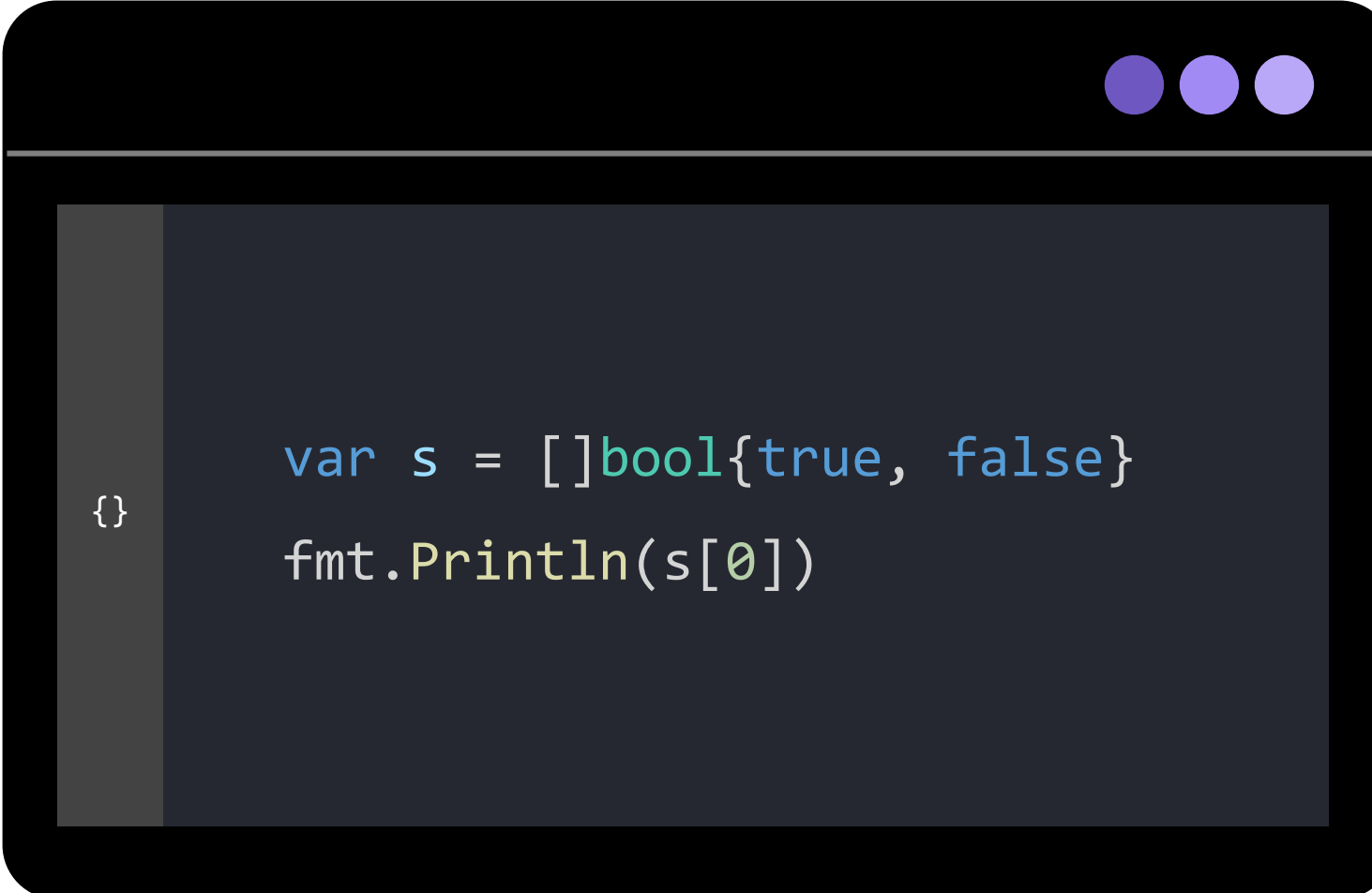
Go fornece uma forma confortável para se trabalhar com arrays.



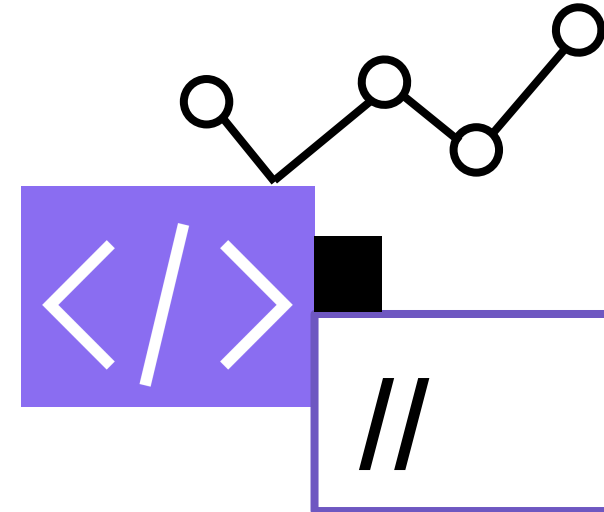
Slices

Um slice é declarado de forma parecida com o array mas, diferente do array, não é preciso especificar o seu tamanho, pois o Go é a responsável por gerenciá-lo dinamicamente.

Este é um slice com elementos do tipo bool. Vamos ver como obter um valor desse slice:



```
1  var s = []bool{true, false}
    fmt.Println(s[0])
```



Criar um slice com make()

Os slices também podem ser criados com a função **make()**. Essa função gera um array com os valores vazios e instanciados, e devolve um slice referente a esse array:

```
{} a := make([]int, 5) // len (a) = 5
```

Slices: selecionar fragmento

Outra forma de obter os valores de um slice é nos baseando em um intervalo que seja formado por dois índices, um de início e outro de fim (separados por dois pontos). Isso pode também ser realizado com os arrays.

Isto seleciona um intervalo semi-aberto que inclui o primeiro elemento, mas exclui o último.

{}

```
package main
import "fmt"

func main() {
    primes := []int{2, 3, 5, 7, 11, 13}
    fmt.Println(primes[1:4]) // Se não colocarmos
    um valor após os ":" ele retorna até o final do
    slice e viceversa.
}
```

Slices: exemplo de seleção de fragmento

No exemplo a seguir vamos realizar 3 leituras. A primeira vamos ler todos os valores do início até o primeiro elemento, a segunda do primeiro ao último e na terceira da segunda até a quarta.

Os valores resultantes serão respectivamente: "[2]", "[5 7 11 13]" "[5 7]".

{}

```
package main
import "fmt"

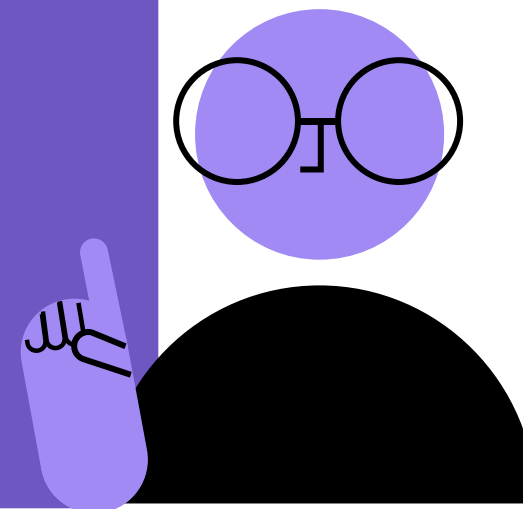
func main() {
    primes := []int{2, 3, 5, 7, 11, 13}
    fmt.Println(primes[:1]) // [2]
    fmt.Println(primes[2:]) // [5 7 11 13]
    fmt.Println(primes[2:4]) // [5 7]
}
```

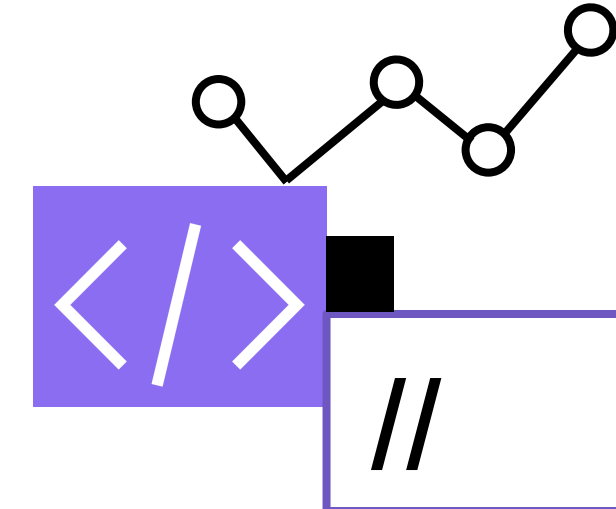
Slices: comprimento e capacidade

Vamos ver do que se tratam essas duas propriedades de um slice:

- O comprimento de um slice é o número de elementos que ele contém.
- A capacidade de um slice é o número de elementos do array subjacentes, contando a partir do primeiro elemento do fragmento.

O comprimento e a capacidade de um slice podem ser obtidas utilizando as funções `len()` e `cap()`.





Adicionar a um slice

É habitual ter que adicionar elementos a um slice. Para realizar essa tarefa, o Go nos fornece a função **append()**. Vamos ver como funciona:

```
{ }
```

```
func append(s []T, vs ...T) []T
```

Essa função recebe, como primeiro parâmetro, “s”, o slice do tipo “T (ao qual queremos adicionar um valor), e os demais parâmetros são valores do tipo “T” que queremos adicionar. Ele retorna um slice com todos os elementos anteriores mais os novos.

```
{ }
```

```
var s []int
```

```
s = append(s, 2, 3, 4)
```

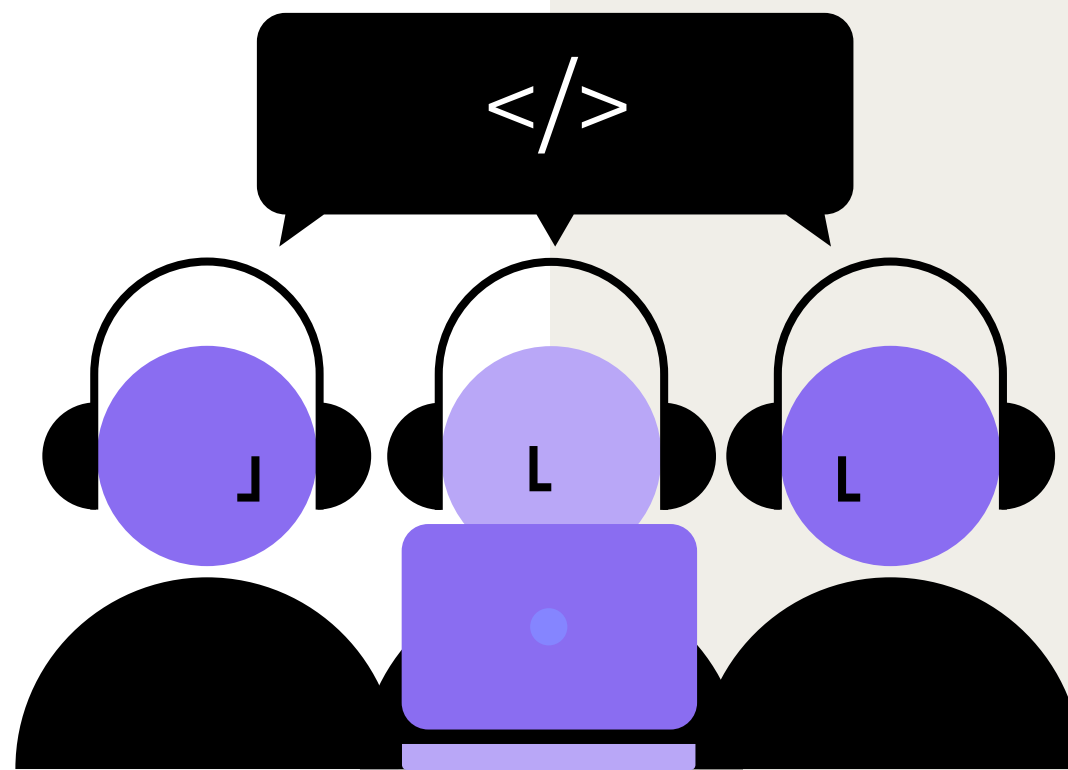
Arrays

vs.

Slices

- Os arrays têm um tamanho definido, que deveremos definir no momento de instanciá-lo.

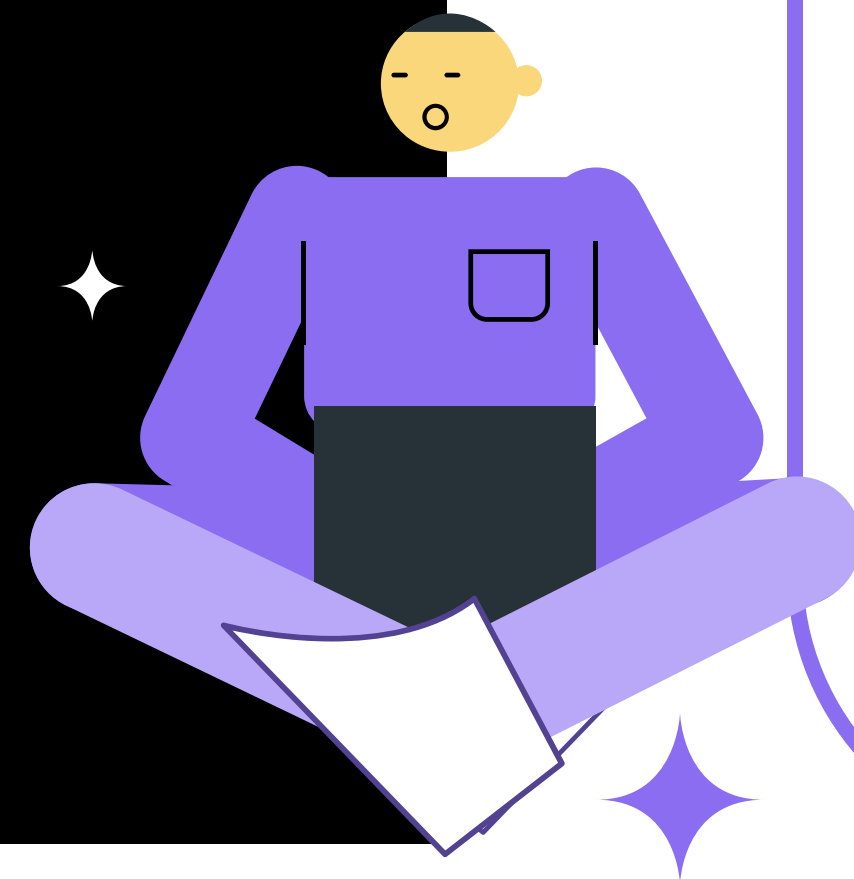
- Já os slices são dinâmicos, podendo aumentar ou diminuir em tempo de execução.



03

Maps

Os **maps** nos permitem criar variáveis do tipo **chave-valor**, definindo um tipo de dado para as chaves e outro para os valores. ✨



Declaração de um map

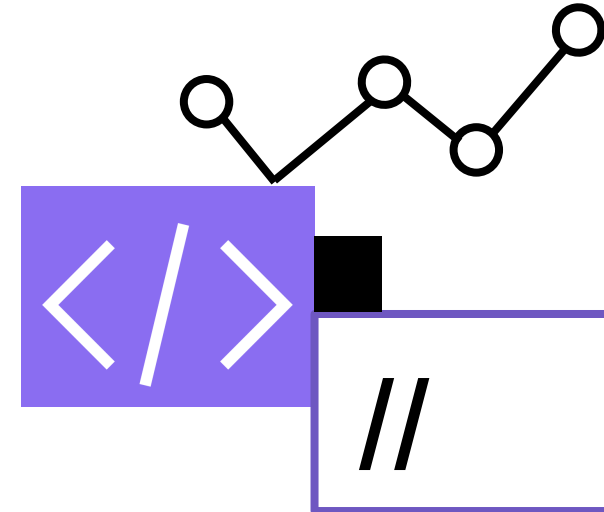
Um **map** pode ser instanciado de duas formas:

```
{}  
myMap := map[string]int{}  
myMap := make(map[string]string)
```

A função **make()** recebe como argumento o tipo de map e devolve um map inicializado.



A função **make** serve para inicializá-lo, mas não é possível introduzir dados na mesma sentença de inicialização.



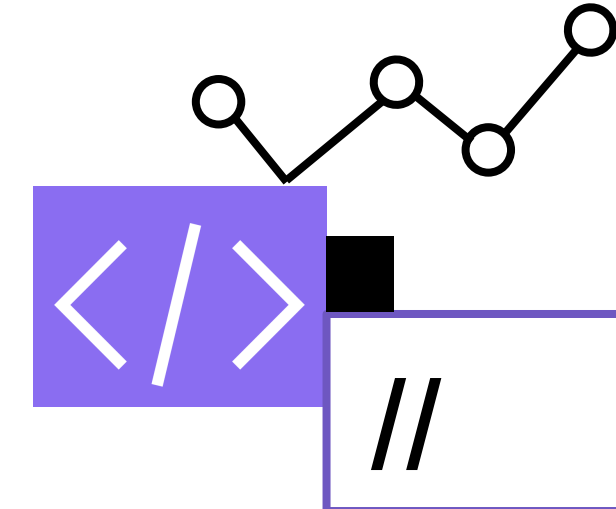
Comprimento de um map

A quantidade de elementos chave-valor de um map pode ser definida com a função **len()**.

{}

```
var myMap = map[string]int{}  
fmt.Println(len(myMap))
```

Essa função devolve zero para um mapa não inicializado.



Acessar elementos

Para acessar um elemento de um map, chamamos o seu nome, seguido do nome da chave que queremos acessar, entre colchetes.

A vantagem de um map é a sua capacidade para recuperar rapidamente os dados de um valor conforme a chave. Uma chave funciona como um índice, apontando para o valor associado a essa chave.

{ }

```
var students =  
map[string]int{"Benjamin": 20,  
               "Nahuel": 26}  
fmt.Println(students["Benjamin"])
```

Adicionar elementos

A adição de um elemento ao map é realizada utilizando uma nova chave de índice e atribuindo-lhe um valor.

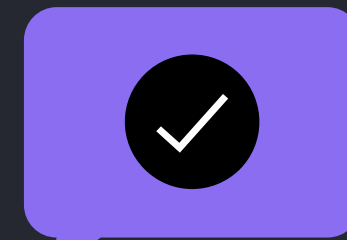
```
{}  
var students = map[string]int{"Benjamin": 20, "Nahuel": 26}  
fmt.Println(students)  
students["Brenda"] = 19  
students["Marcos"] = 22  
fmt.Println(students)
```

Atualizar valores

O valor de um elemento específico pode ser atualizado consultando o seu nome de chave:

```
var students = map[string]int{"Benjamin": 20, "Nahuel": 26}  
fmt.Println(students)  
students["Benjamin"] = 22  
fmt.Println(students)
```

{}

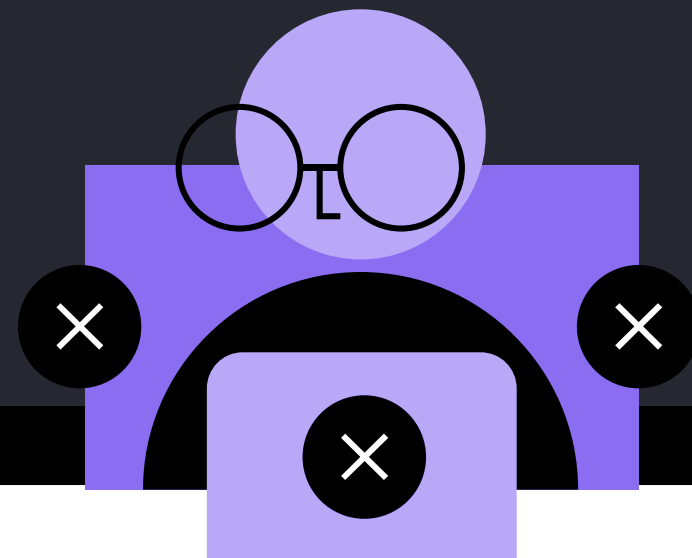


Remover elementos

Go nos fornece uma função para apagar elementos de um map:

{}

```
var students = make(map[string]int)
students["Benjamin"] = 20
fmt.Println(students)
delete(students, "Benjamin")
fmt.Println(students)
```



Muito obrigado!