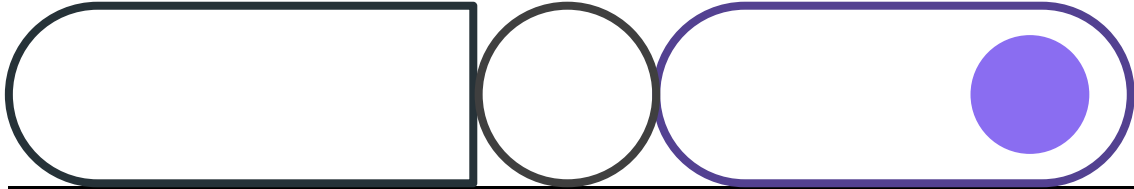
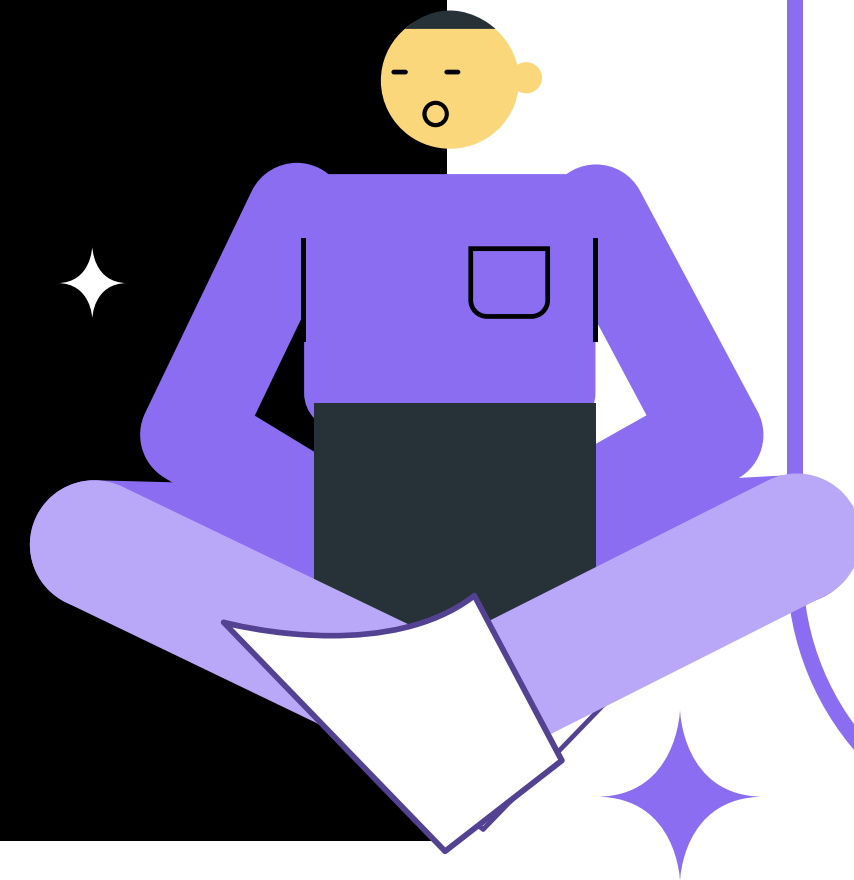


# Colecciones de datos



Los arrays y slices son **tipos de datos** que nos permiten **almacenar** un **conjunto** de datos **homogéneo**. Es decir, datos que sean del mismo tipo.



# Índice

- 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)
```

```
}
```

# Declaración

Para declarar un array debemos definir un tamaño y un tipo de dato.

**Tamaño**    **Tipo de dato**

{ }

var a [2]string

**Declara una variable “a” como un array de dos 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)
```

```
}
```

# Asignar valores

Para asignar un valor a un array hay que especificar la posición seguida por el valor.

**Posición**

**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)
}
```

## Obtener valores

Para obtener el valor de un array solo hace falta especificar el nombre de la variable y la posición que deseas obtener:

```
{}
```

```
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])
```

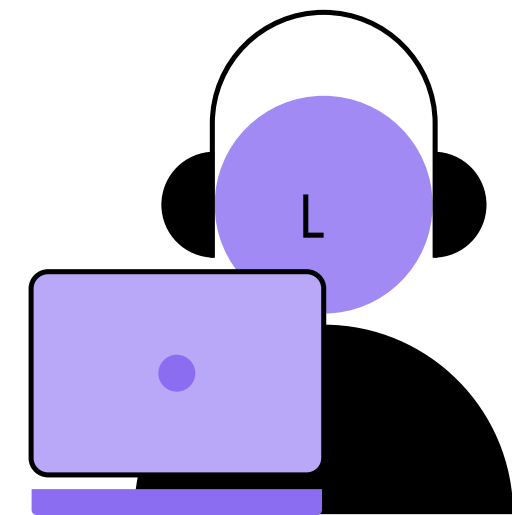
```
    fmt.Println(a)
```

```
}
```

La longitud de un array es parte de su tipo, por lo que no se puede cambiar el tamaño de los arrays.



Go proporciona una forma cómoda de trabajar con arrays.





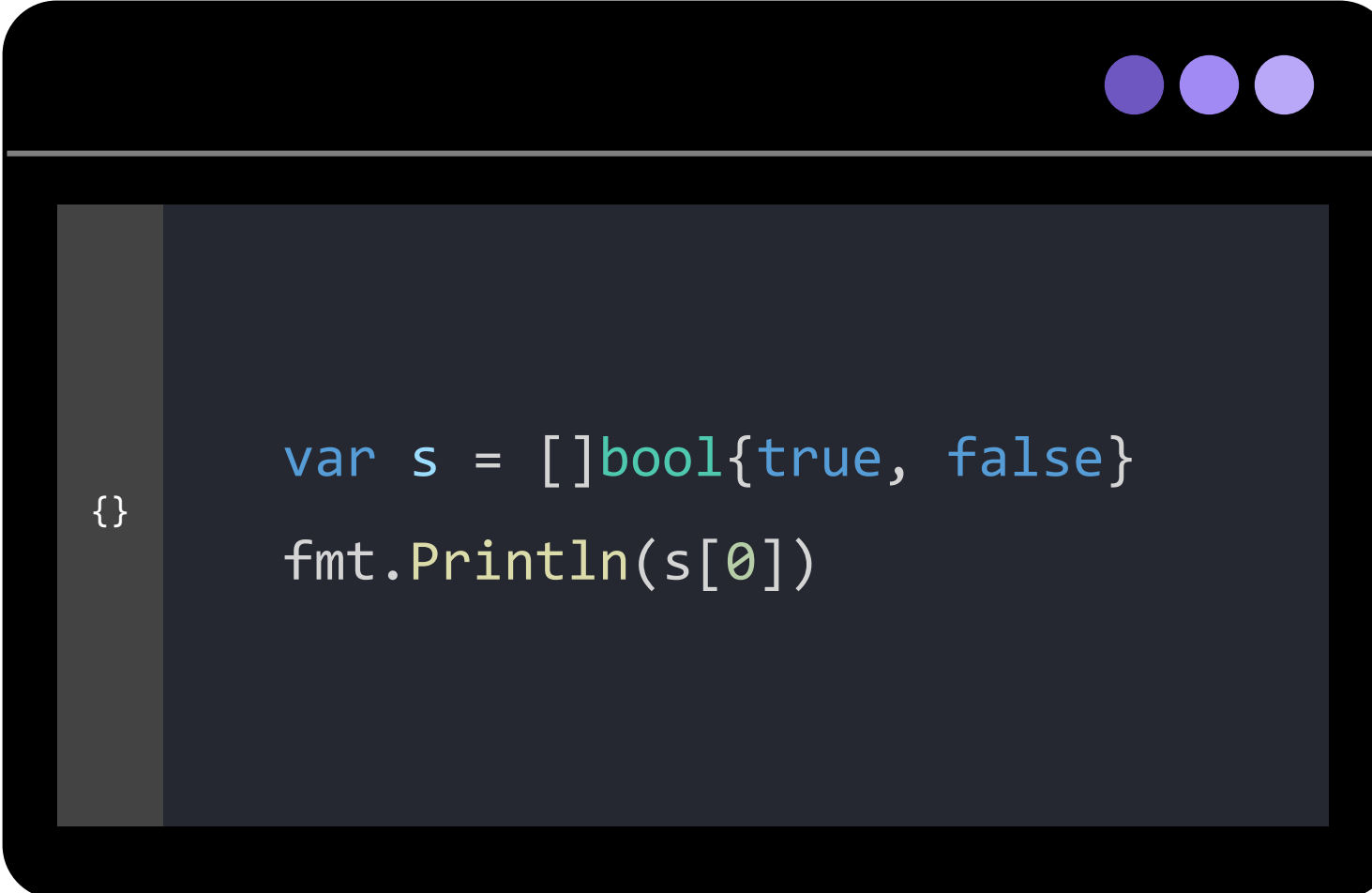
02

Slices

# Slices

Un slice se declara similar a un array, pero —a diferencia del array— no le tenemos que especificar el tamaño, ya que Go se encarga de manejarlo dinámicamente.

El siguiente es un slice con elementos de tipo bool. Veamos cómo obtener un valor de dicho slice:



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



# Crear un slice con make()

También los slices se pueden crear con la función **make()**. Esta función genera un array con los valores en 0 y devuelve un slice que hace **referencia** a ese array:

```
{ }
```

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

# Slices: obtener rango

Otra forma de obtener los valores de un slice es basándonos en un rango que esté formado por dos índices, uno de inicio y otro de fin (separados con dos puntos). Esto también se puede hacer con los arrays.

Esto selecciona un rango semiabierto que incluye el primer elemento, pero excluye el último.

{ }

```
package main
import "fmt"

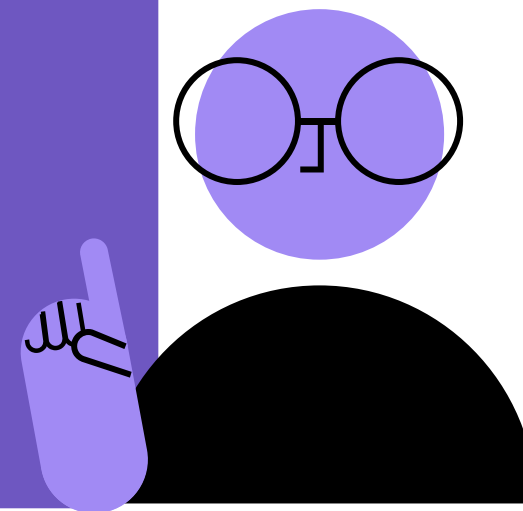
func main() {
    primes := []int{2, 3, 5, 7, 11, 13}
    fmt.Println(primes[1:4]) // Si no ponemos un
                             // valor después de los ":" toma hasta el fin de
                             // elementos del slice y viceversa.
}
```

# Slices: longitud y capacidad

Veamos de qué se tratan estas dos propiedades de un slice:

- La longitud de un slice es el número de elementos que contiene.
- La capacidad de un slice es el número de elementos del array subyacente, contando desde el primer elemento del segmento.

La longitud y la capacidad de un slice se pueden obtener utilizando las funciones **len()** y **cap()**.





# Agregar a un slice

Es común tener que agregar elementos a un slice. Para realizar esta tarea, Go nos provee de la función **append()**. Veamos cómo funciona:

{ }

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

Esta función recibe, como primer parámetro “s”, el slice de tipo “T” (al cual queremos agregarle un valor), y el resto de los parámetros son los valores de tipo “T” que queramos agregar. Esta retorna un slice con todos los elementos anteriores más los nuevos.

{ }

```
var s []int  
s = append(s, 2, 3, 4)
```

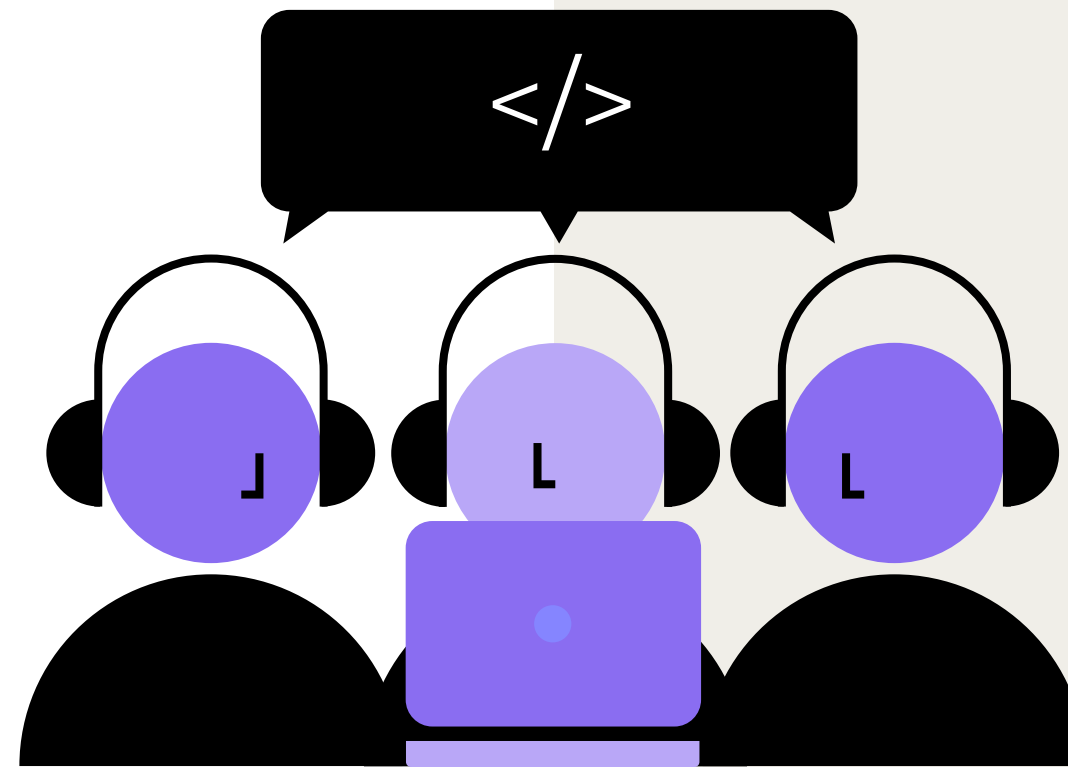
# Arrays

vs.

# Slices

- Los arrays tienen un tamaño definido, el cual debemos definir al momento de instanciarlo.

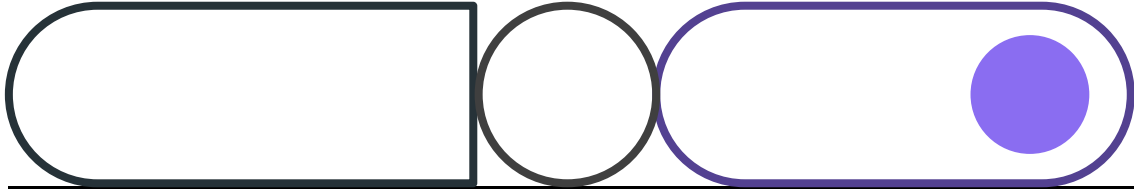
- Mientras que los slices manejan el tamaño de forma dinámica, pudiendo incrementar en tiempo de ejecución.



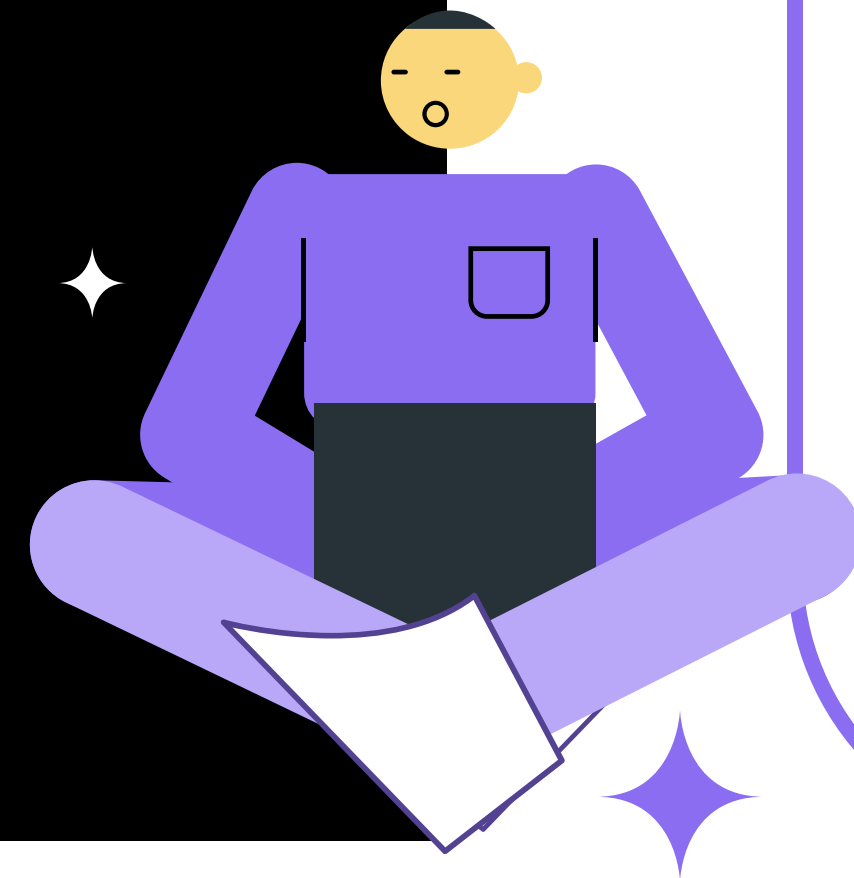
03

# Maps





Los **maps** nos permiten crear variables de tipo **clave-valor**, definiendo un tipo de dato para las claves y uno para los valores.



# Declaración de un map

Podemos instanciar un **map** de dos maneras:

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

La función **make()** toma como argumento el tipo de map y devuelve un map inicializado.



La función **make** nos sirve para inicializarlo, pero no podremos introducir datos en la misma sentencia de inicialización.



# Longitud de un map

Podemos determinar cuántos elementos clave-valor tiene un map con la función **len()**:

{}

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

Esta función devuelve cero para un mapa no inicializado.



# Acceder a elementos

Para acceder a un elemento de un map, llamamos al nombre del mismo, seguido por el nombre de la clave que queremos acceder, entre corchetes.

La fortaleza de un map es su capacidad para recuperar datos rápidamente de un valor según la clave. Una clave funciona como un índice, apuntando al valor asociado con dicha clave.

{ }

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

# Agregar elementos

La adición de un elemento al map se realiza utilizando una nueva clave de índice y asignándole un valor.

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

# Actualizar valores

Podemos actualizar el valor de un elemento específico consultando su nombre de clave:

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

{}

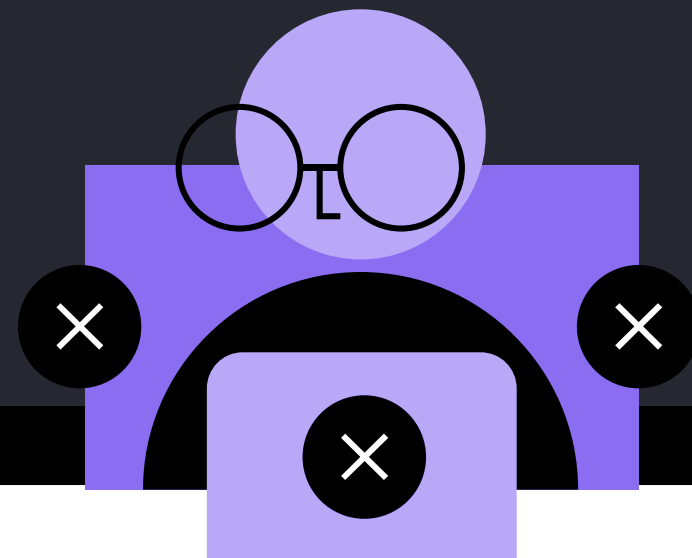


# Eliminar elementos

Go nos proporciona una función para el borrado de elementos de un map:

{}

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



¡Muchas gracias!