GO

BOOTCAMP

Clase en vivo

//Go Bases

IT BOARDING

BOOTCAMP

Objetivos de esta clase

- Implementar estructuras de control.
- Repasar distintos operadores.
- Conocer los arrays, slices y maps.
- Entender sus diferencias y cuándo conviene aplicar cada uno.

Índice

IT BOARDING
BOOTCAMP

1 | Repaso

IT BOARDING

BOOTCAMP

IF else

IF else

En el siguiente ejemplo podemos ver el uso de las sintaxis if, else y else if..

{}

```go
package main

import "fmt"

func main() {
   var salary float64 = 4000
   if salary <= 3000 {
       fmt.Println("This person must pay taxes")
   } else if salary <= 4000 {
       fmt.Printf("They must pay $%4.2f of their salary\n", (salary/100)*10)
   } else {
       fmt.Printf("They must pay $%4.2f of their salary\n", (salary/100)*15)
   }
}
```

Switch

Switch sin condición

Podemos utilizar un `switch` sin condición, agregando la condición en el case.
Ej `case condition:`

```go
package main

import "fmt"

func main() {
   var age uint8 = 18
   switch {
   case age >= 150:
       fmt.Println("Are you immortal?")
   case edad >= 18:
       fmt.Println("You are an adult!")
   default:
       fmt.Println("You’re not an adult yet")
   }
}
```

Switch con múltiples casos

Los case pueden tener múltiples valores separadas por comas como podemos ver en el siguiente ejemplo:

{}

```go
package main

import "fmt"

func main() {
   day := "sunday"

   switch day {
   case "monday", "tuesday", "wednesday", "thursday", "friday":
       fmt.Printf("%s is a workday\n", day)
   default:
       fmt.Printf("%s is a weekend day\n", day)
   }
}
```

Switch con fallthrough

Veamos una implementación usando fallthrough

{}

```
package main

import "fmt"

func main() {

    for i := 1; i <= 100; i++ {
        switch {
        case i%15 == 0:
            fmt.Print("Fizz")
            fallthrough
        case i%5 == 0:
            fmt.Print("Buzz")
        case i%3 == 0:
            fmt.Print("Fizz")
        default:
            fmt.Printf("%d", i)
        }
        fmt.Print(" ")
    }
}
```

Bucle For

For range

a palabra reservada **range** itera por los elementos de un array/slice, string o map, retornando siempre **2** variables. Tenemos:

- **Array or slice:** primero el ***índice***, segundo el ***elemento*** de la lista.
- **String:** primero el ***índice***, segunda es la representación del carácter en *rune int.*
- **Map:** primero la ***key***, segundo el ***value*** del par *clave-valor.*

Veamos un ejemplo con un slice:

```
{}
fruits := []string{"apple", "banana", "pear"}
for i, fruit := range fruit {
   fmt.Println(i, fruit)
}
```

Bucle infinito

```
sum := 0
for {
    sum++
}
// never arrives since the loop is infinite
```

Bucle “*while*”

```
    sum := 1
    for sum < 10 {
        sum += sum
    }
    fmt.Println(sum)
```

Saltar a la siguiente iteración

Puede ser útil pasar a la siguiente iteración de un bucle antes que termine de correr todo el código. Esto se hace con la palabra reservada **continue**.

Ejemplo: solo imprime los *números impares*, cuando el módulo entre 2 es 0 es un número par, entra en la condición y saltea la iteración.

```
{}
for i := 0; i < 10; i++{
    if i % 2 == 0 {
        continue
    }
    fmt.Println(i, "is odd")
}
```

Romper un bucle

Romper un bucle antes de que termine puede ser útil, especialmente en un bucle infinito. La palabra reservada **break** nos permite terminar con la ejecución del bucle.

```
{}
sum := 0
for {
    sum++
    if sum >= 1000 {
        break
    }
}
fmt.Println(sum)
```

Operadores

Operadores

{}

```go
package main
import "fmt"
func main() {
    x, y, z := 10, 20, 30
    //Arithmetic Operators
    fmt.Println("x+y=", x+y) //x+y=30
    fmt.Println("x*y=", x*y) //x*y=200
    y--
    fmt.Println("y--=", y) //y--=19
    //Assignment Operators
    x = y
    fmt.Println("x=y", x) //x=y 19
    x += y
    fmt.Println("x+=y", x) //x+=y 38
    x = 100
    x /= y
    fmt.Println("x/=y", x) //x/=y 5 ...
```

```go
...//Comparison operators
    fmt.Println(x == y) //false
    fmt.Println(x != y) //true
    fmt.Println(x > y)  //false
    fmt.Println(x >= y) //false
    //Logical operators
    fmt.Println(x < y && x > z)      //false
    fmt.Println(x < y || x > z)      //true
    fmt.Println(!(x == y && x > z)) //true
    //Address operators
    var pint *int
    pint = &x
    fmt.Println(pint)  //0xc0000a0a0
    fmt.Println(*pint) //5
}
```

2 | Arrays

IT BOARDING

BOOTCAMP

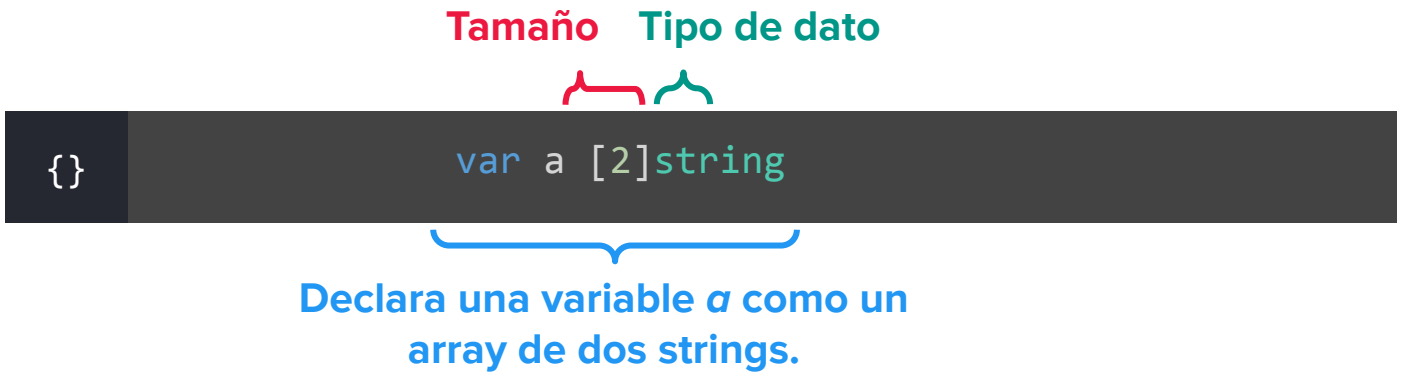
// ¿Qué son los Arrays?

Es una colección de items del mismo tipo de datos que estan almacenados de forma contigua en la memoria

IT BOARDING
BOOTCAMP

Array: declaración

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



Array: asignar valores

Para asignar un valor a un array hay que especificarle el índice seguido por el valor.


```
{}
    a[0] = "Hello"
    a[1] = "World"
```

Array: 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)
```

A codear...

Array: ejemplo completo

```
{}
```

```go
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)
}
```

3 Slices

IT BOARDING

BOOTCAMP

Al igual que los arrays, los Slices nos permiten almacenar un conjunto de datos homogéneo, es decir, todos ellos del mismo tipo.

Slice

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.

`var s []T` esto es un slice con elementos de tipo `T`

Para obtener un valor de un slice lo hacemos de la siguiente manera:

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

Crear un Slices con make()

También los slice se pueden crear con la function make()

La función make 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
```

Slice: obtener rango

También se puede hacer con los arrays

Otra forma de obtener los valores de un slice es en base a un rango que esté formado por dos índices, uno de inicio y otro de fin (separados con dos puntos).

{}

```
package main
import "fmt"

func main() {
   primes := []int{2, 3, 5, 7, 11, 13}
   fmt.Println(primes[1:4]) // If we do not put a value after the : it takes until the end
of the elements of the slice and vice versa.
}
```

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

Slice longitud y capacidad

Un Slice tiene tanto una longitud como también una capacidad.

- 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(s) y cap(s).

Agregar a un Slice

Es común tener que agregar elementos a un slice, Go nos provee una función para esto. ¿Cómo funciona la función append?

```
{}  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 resto de los parámetros son los valores de tipo `(T)` que queramos agregar. La misma retorna un slice con todos los elementos anteriores más los nuevos.

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

A codear...

4 | Maps

IT BOARDING

BOOTCAMP

// ¿Qué son los 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

Para instanciar un map lo podemos hacer 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

Para determinar cuántos elementos (clave-valor) tiene un map, lo podemos hacer con una función que nos proporciona Go para esto len().

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

La función len () 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.

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

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

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

Puede 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)
```

Recorrer elementos de un Map

El for nos permite recorrer los elementos de nuestro map.

```
{}
var students = map[string]int{"Benjamin": 20, "Nahuel": 26}
for key, element := range students {
    fmt.Println("Key:", key, "=>", "Element:", element)
}
```

A codear...

Actividad

Conclusiones

En esta clase repasamos las estructuras de control, cómo puede ser el bucle For y también vimos cómo usar los operadores.

Aprendimos sobre estructuras de datos: Arrays, Slices y Maps. Conocimos sus diferencias y en qué caso usar cada una.

Gracias.

IT BOARDING

BOOTCAMP