INTRODUCCIÓN A LA CLASE

Go Bases

Objetivos de esta clase

Los objetivos de esta clase son:

- Comprender, generar y utilizar punteros en Go.
- Comprender y utilizar Go Routines
- Comprender y generar canales en Go.

¡Vamos por ello!

PUNTEROS

Go Bases

// ¿Qué es un Puntero?

“Un puntero es un tipo de datos cuyo valor es una dirección de memoria que se refiere a (o "apunta a") otro valor almacenado en otra parte de la memoria.”

Punteros

// Aplicar punteros en GO

IT BOARDING

BOOTCAMP

Punteros en GO

Una de las ventajas de Go es que nos permite trabajar con punteros.

¿Cómo lo hacemos?

El tipo `*T` es un puntero al valor de `T`. Su valor inicial es nil.

```
{}  var p *int
```

En la variable `p` tendremos un puntero de un valor de tipo de dato int.

Obtener el puntero de una variable

Para obtener la dirección de memoria de una variable (puntero) lo que debemos hacer es utilizar el operador `&` de la siguiente manera:

```
{}
    i := 42
    p = &i
```

En la variable `i` tendremos el valor y en la variable `p` el puntero

Obtener el valor de un puntero

Para obtener el valor de un puntero lo que debemos hacer es utilizar el operador `*` de la siguiente manera:

```
{}
var p *int
i := 42
p = &i
fmt.Println(*p) // lee i a través del puntero p
```

Asignar un valor a través del puntero

Para asignar el valor a través de un puntero lo que debemos hacer es utilizar el operador `*` de la siguiente manera:

```
var p *int
i := 42
p = &i
*p = 21         // asigna a i a través del puntero p
fmt.Println(i)  // podemos ver reflejado el cambio de i
```

Punteros: ejemplo completo

{}

```go
package main

import "fmt"

func main() {
   var p *int

   i := 42
   p = &i

   fmt.Println(p)  // dirección de memoria
   fmt.Println(*p) // lee i a través del puntero p
   *p = 21         // asigna a i a través del puntero p
   fmt.Println(i)  // podemos ver reflejado el cambio de i
}
```

Punteros ejemplo 2

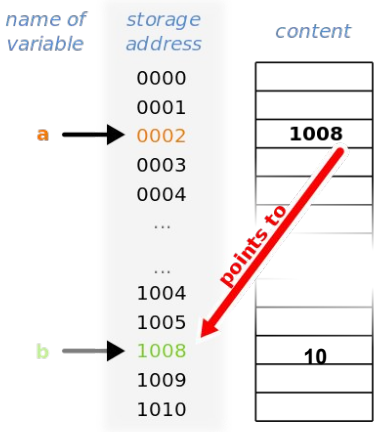
```
{}
```

```go
package main

import "fmt"

func main() {
   var a *int
   b := 10
   a = &b
   fmt.Println(&b)  // dirección de memoria de b (1008)
   fmt.Println(a)  // guarda la dirección de memoria de b
(1008)

   fmt.Println(b)  // valor de b (10)
   fmt.Println(*a) // valor a donde apunta a (10)
}
```



Conclusión...

Un puntero es un dirección de memoria que hace referencia a otro valor.

Punteros por parámetro

//

IT BOARDING

BOOTCAMP

// ¿Cómo podemos pasar un puntero en una función?

Podemos pasar punteros como parámetros en funciones y trabajar con la misma dirección de memoria, dentro y fuera de la función.

Declaración de variable

Vamos a declarar una variable de tipo entero para luego trabajarla dentro de nuestra función.

```
{}
valor := 3
fmt.Println(valor)
```

Declaración de variable

Declararemos una función que reciba como parámetro un puntero de integer.

```
{}
func Incrementar(v *int){
}
```

Si quisiéramos incrementar el valor del puntero ¿Cómo lo haríamos utilizando un operador de incremento (variable++) ?

Respuesta Incorrecta

Si la respuesta es la siguiente, **es incorrecto**

```
{}
func Incrementar(v *int){
    v++
}
```

v hace referencia a la dirección de memoria no al valor!

Incrementar valor de un puntero

Para incrementarlo primero debemos hacer referencia al valor mediante `*` y luego realizar el operador de incremento.

```
{}
func Incrementar(v *int){
    (*v)++
}
```

Referencia de una variable

A nuestra función debemos pasarle la **referencia** de la variable instanciada, apuntando a dicha variable con la dirección de memoria.

```
valor := 3
Incrementar(&valor)
fmt.Println(valor) // valor 4
```

// Para concluir

Hemos visto cómo comprender y utilizar punteros en Go.

¡A seguir aprendiendo!

IT BOARDING

BOOTCAMP

CANALES

Go Bases

// ¿Qué es un canal?

Un canal es una tubería (pipe) que conecta Go Routines, pudiendo enviar valores a través de ella.
Pero antes de aprender Canales...

Go Routines

// Func

IT BOARDING

BOOTCAMP

// ¿Qué son las Go Routines?

Las Go Routines nos permiten trabajar en concurrentemente, cuando ejecutamos una Go Routine lo que hacemos es lanzar una función para que se ejecute en simultáneo con el resto del programa.

Función a procesar

En este caso tenemos una función que se encargará de hacer un determinado proceso, para ello agregaremos una pausa de un segundo en él, para simular una funcionalidad que accede a una Base de Datos o envía información a una API.

```
{}
func proceso(i int) {
    fmt.Println(i, "-Inicia")
    time.Sleep(1000 * time.Millisecond)
    fmt.Println(i, "-Término")
}
```

Procesar en hilo principal

Ejecutaremos nuestra función 10 veces, veremos que para que inicialice el 2do procesamiento el 1ero deberia haber terminado.

```
{}
func main() {
    for i := 0; i < 10; i++{
        proceso(i)
    }
}
```

Pausar fin ejecucion

Agregaremos una pausa de 5 segundos antes de terminar el programa, ya veremos porque...

```
{}
func main() {
    for i := 0; i < 10; i++{
        proceso(i)
    }
    time.Sleep(5000 * time.Millisecond)
    fmt.Println("Termino el programa")
}
```

// ¿Cómo podríamos ejecutar la función en simultáneo?

Para ello utilizaremos las Go Routines, generarán un hilo por cada núcleo del equipo. Por ende, si nuestro equipo tiene 4 nucleos, ejecutara la funcion en 4 hilos en paralelo.

Go Routine

Para indicar que queremos ejecutar la función en una Go Routine lo hacemos con la palabra reservada **go**.
Vemos cómo se ejecuta de forma simultánea y no de forma lineal:

```
{}
func main() {
    for i := 0; i < 10; i++{
        go proceso(i)
    }
    time.Sleep(5000 * time.Millisecond)
    fmt.Println("Termino el programa")
}
```

Go Routine

Si quitamos la pausa de 5 segundos que agregamos al final se terminara el programa antes que terminen las Go Routines.
¿Como podriamos hacer que el programa espere hasta que terminen todas las rutinas?

```
{}
func main() {
    for i := 0; i < 10; i++{
        go proceso(i)
    }
}
```

Una alternativa para ello es usar Canales.

Canales

// Func

IT BOARDING

BOOTCAMP

Un canal nos va a permitir enviar valores a las Go Routines y esperar hasta recibir dicho valor.

Sintaxis

Para definir un canal de tipo entero lo hacemos de la siguiente manera:
utilizando la palabra reservada **chan**

```
{}  c := make(chan int)
```

Recibir canal como parámetro

Debemos recibir como parámetro el canal en nuestra función, para ello lo definimos como **chan int** al ser un canal de enteros

```
{}
func proceso(i int, c chan int) {
    fmt.Println(i, "-Inicia")
    time.Sleep(1000 * time.Millisecond)
    fmt.Println(i, "-Término")
}
```

Enviar valor a canal

Una vez que terminamos de procesar debemos enviarle el valor al canal, en este caso le enviaremos el valor de i

{}

```
func proceso(i int, c chan int) {
    fmt.Println(i, "-Inicia")
    time.Sleep(1000 * time.Millisecond)
    fmt.Println(i, "-Termino")
    c <- i
}
```

Ejecutar ejemplo

Vamos a correr el programa ejecutando solo una función.
Vemos que sigue terminando el programa antes que termine la rutina.
¿Qué pasó? ¿Qué nos está faltando?

{}

```
func main() {
    c := make(chan int)
    go proceso(1, c)
    fmt.Println("Termino el programa")
}
```

Recibir valor

Nos está faltando recibir el valor del canal, para que el programa quede esperando hasta recibir ese valor.
Para indicarle al programa que estamos esperando para recibir el valor del canal de la variable **c** lo hacemos con **<-c**

```
<-c // recibimos el valor del canal
variable := <-c // recibimos y lo asignamos a una variable
fmt.Println("Termino el programa en ", <-c) // recibimos y lo imprimimos
```

Ejecutar ejemplo

De esta manera, el programa terminará una vez que se le asigne el valor en el canal

{}

```
func main() {
    c := make(chan int)
    go proceso(1, c)
    fmt.Println("Termino el programa")
    <-c
}
```

Ejecutar ejemplo con for

```
{}
func main() {
    c := make(chan int)

    for i := 0; i < 10; i++{
        go proceso(i, c)
    }

    for i := 0; i < 10; i++{
        fmt.Println("Termino el programa en ", <-c)
    }
}
```

// Para concluir

Hemos visto cómo comprender y generar Canales en Go

¡A seguir aprendiendo!

IT BOARDING

BOOTCAMP

Gracias.

IT BOARDING

BOOTCAMP