The background is a dark blue-green color with various abstract geometric shapes and patterns. There are several white and yellow curved lines, some straight lines, and some rectangular shapes. A large yellow rectangle is on the left side. A large white rectangle is at the top center. A large yellow rectangle is on the right side. A large white rectangle is at the bottom center. A large yellow rectangle is on the left side. A large white rectangle is at the top center. A large yellow rectangle is on the right side. A large white rectangle is at the bottom center.

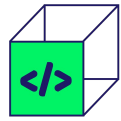
# Curso de **Go Intermedio: Programación Orientada a Objetos y Concurrencia**

**Néstor Escoto**



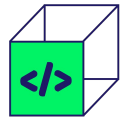


# Introducción al curso



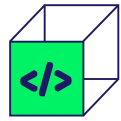
# Golang

- Lenguaje compilado.
- Potente librería estándar.
- Manejo de concurrencia a través de GoRoutines y Channels.
- Diseñado por Google, Ken Thompson (UNIX) parte del equipo de diseño.



# Golang

- Muy popular en aplicaciones de CLI y Backend.
- Docker, Kubernetes y Terraform están escritos en Golang.
- Muchos proyectos de la CNCF están escritos en Go.
- Muy utilizado para escribir malware.



# Golang

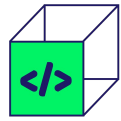
- Según StackOverflow, es el tercer lenguaje mejor pagado a nivel global.
- Según StackOverflow, es la quinta tecnología más amada por los desarrolladores y la tercera más deseada para trabajar.





# **Cosas que debes de saber**

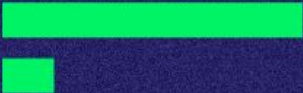




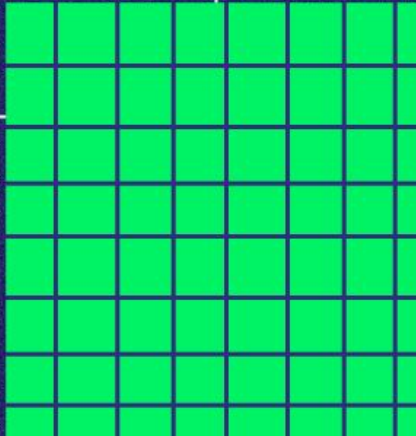

# Conocimientos requeridos

- Declaración de variables
- Condicionales
- Sintaxis básica
- Declaracion de GoRoutines y Channels
- Slices y maps
- Apuntadores





**Qué cosas  
terminaré  
sabiendo**

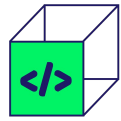






# Conocimientos adquiridos

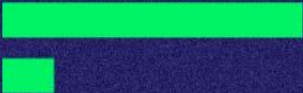
- ¿Es Go orientado a objetos?
- Cómo aplicar los conceptos de POO en Golang.
- Crear Test Unitarios en Go para aplicar TDD.
- Calcular el Code Coverage de mi proyecto.




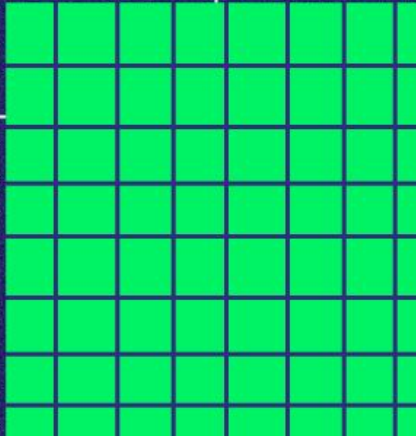
# Conocimientos adquiridos

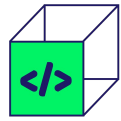
- Análisis del Profiling en tus programas.
- Cómo multiplexar mensajes de canales.
- Técnicas para la creación de Worker Pools concurrentes.
- Crear Test Unitarios en Go para aplicar TDD.
- Crear un Job Queue concurrente.





# **Repaso general: variables, condicionales, slices y map**





# Repaso general


- Repasar los conceptos importantes del curso anterior que serán cruciales para este curso.
- Definir structs y darles funcionalidad utilizando receiver functions.
- Apuntadores y cómo estos nos ayudan a utilizar/modificar las variables correctas y no generar copias.





# Repaso general

- Go utiliza las GoRoutines como mecanismo para aplicar concurrencia.
- Channels como medios de comunicación entre diferentes rutinas.



# **Repaso general: GoRoutines y apuntadores**





**¿Es Go orientado  
a objetos?**



# ¿Es Go orientado a objetos?

- POO se ha convertido en uno de los paradigmas de programación predominante en la industria.
- POO puede llegar a ser muy riguroso, pero a cambio permite una alta reutilización de código y la aplicación de un sinnúmero de patrones de diseño.





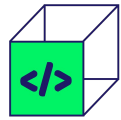
# ¿Es Go orientado a objetos?

- Go puede alcanzar la aplicación de los conceptos de POO, pero de una forma diferente a lenguajes como Java o Python.



# Structs vs. classes





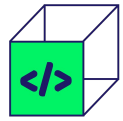
# Structs vs. clases

- El objetivo de una clase es definir una serie de propiedades y métodos que un objeto puede usar para alcanzar diferentes objetivos.
- Go utiliza Structs para generar “nuevos tipos” de datos que se pueden utilizar para cumplir tareas en específico.



# Métodos y funciones





# Métodos y funciones

Algunos lenguajes de programación implementan la filosofía que TODO debe de ser un objeto, sin embargo, no siempre es algo aplicable, por ejemplo, en una librería con utilidades que no pertenecen a un dominio específico.



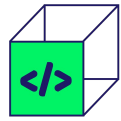
# Métodos y funciones

Go permite que este tipo de “utilidades” no tengan que pertenecer obligatoriamente a un struct.





# Constructores



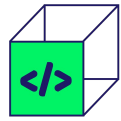
# Constructores

- Los constructores permiten la instanciación de una clase a un objeto, asimismo permite definir propiedades predefinidas.
- En Go podemos utilizar funciones que puedan crear structs con propiedades que nosotros pasamos como parámetros.





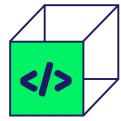
# Herencia



# Herencia

Go no permite el concepto de herencia tal y como lo conocemos de lenguajes como TypeScript; en otros lenguajes de programación se utiliza la herencia como una forma de alcanzar reutilización de código y polimorfismo.





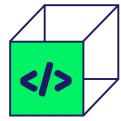
# Herencia

Para alcanzar este mismo objetivo Go aplica un principio llamado Composition Over Inheritance que utilizando un campo anónimo en un struct puede “heredar” todas las propiedades y métodos.



# Interfaces





# Interfaces

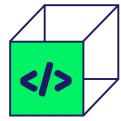
- Diferentes lenguajes de programación utilizan sintaxis explícita para decir que una clase implementa una interfaz.
- Go lo hace de manera implícita lo que permite la reutilización de código y el polimorfismo.



# Abstract factory







# Abstract factory

Patrón de diseño de tipo creacional que permite la producción de objetos de la misma familia o tipo sin especificar su clase concreta, permitiendo esa determinación en tiempo de ejecución.



# **Implementación final de Abstract factory**





# Implementación final de Abstract Factory

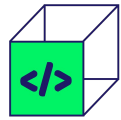
Go te permite implementar muchos de los patrones de diseño que están basados en polimorfismo mediante la utilización de interfaces.



# Funciones anónimas








# Funciones anónimas

- Múltiples lenguajes de programación permiten la creación de funciones que no tienen un nombre, JavaScript es quizá uno en dónde es más común es ver este patrón.
- Go permite la creación de estas funciones anónimas, aunque deben ser usadas con cuidado para evitar romper el principio de DRY.



# **Funciones variádicas y retornos con nombre**





# Funciones variádicas y retornos con nombre

- Las funciones variádicas nos permiten utilizar como slices los argumentos de funciones de los cuales no sabemos su longitud exacta.
- Los retornos con nombre nos permitirán definir variables antes de definir el cuerpo de la función, por lo cual solo utilizaremos return para devolverlos.



# Go Modules

## ¿Cómo los puedo utilizar?





# Go Modules

## ¿Cómo los puedo utilizar?

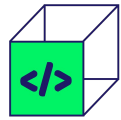
En Node podemos usar npm para generar un árbol de dependencias de nuestro proyectos, en Golang existen los modules con los cuales podremos definir en qué paquetes tenemos dependencias para poder instalarlas, asimismo utiliza el comando `go get -u package.name` para instalarlos.



# Go Modules

## **Creando nuestro módulo**





# Go Modules

## **Creando nuestro módulo**

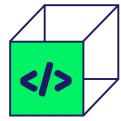
Para generar un módulo nuevo, utilizaremos `go mod init my.module`, a partir de aquí seremos capaces de generar un módulo que otras personas puedan utilizar.



# Testing







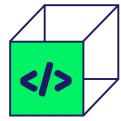
# Testing

- Por lo general, muchos lenguajes de programación utilizan dependencias externas para la creación y ejecución de tests.
- Golang, haciendo realce de su magnífica librería estándar, nos permite crear los tests que necesitamos para tener un código más robusto.



# Code coverage





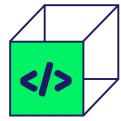
# Code coverage

El code coverage es una métrica que nos permitirá identificar las partes de nuestro código que no han sido probadas y que potencialmente tienen el riesgo de contener un bug.



# Profiling





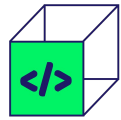
# Profiling

El profiling nos ayudará a encontrar en nuestro código aquellas partes críticas que influyen en una ejecución lenta o con mucho consumo de memoria. Con estas técnicas sabremos en qué enfocarnos a la hora de mejorar nuestro código.



# Testing usando mocks





# Testing usando mocks

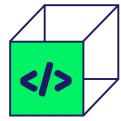
Cuando escribimos un test unitario que tiene dependencias en diferentes servicios nos vemos en la necesidad de crear “mocks” que emulen comportamientos de esos servicios con el objetivo que nuestro test se encargue de probar la funcionalidad que nos interesa y no la de las dependencias.



# Implementando Mocks







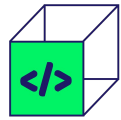
# Implementando Mocks

Es muy importante guardar los valores originales de las funciones que estamos utilizando para realizar nuestros tests en caso de que en test posteriores decidamos evaluar otras partes del código que también dependen de esas funciones



# **Unbuffered channels y buffered channels**





# Unbuffered channels y buffered channels

Go utiliza dos tipos de canales para las comunicaciones, los unbuffered channels son aquellos que tienen una cola de longitud 0, es decir, se utilizan para comunicación síncrona.



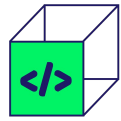
# Unbuffered channels y buffered channels

También existen Buffered Channels que permiten definir una longitud de buffer, es decir, se pueden almacenar ciertos valores antes de empezar a leer.





# Waitgroup



# Waitgroup

Cuando se trabaja con GoRoutines no siempre se planea que estas envíen datos a través de canales entre unas y otras, en estos casos se puede utilizar un Wait Group para alcanzar ese bloqueo de rutinas que es necesario.





# Buffered channels como semáforos



# Buffered channels como semáforos

Estos channels pueden ser utilizados para que actúen como semáforos y permitan una cantidad limitada de ejecuciones para perseguir objetivos como evitar la ejecución de una cantidad indeterminada de rutinas.





# **Definiendo channels de lectura y escritura**



# Definiendo channels de lectura y escritura

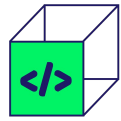
Cuando se trabaja con channels existe la gran probabilidad de crear un deadlock si no somos cuidadosos con su utilización, una forma de mitigar parte de este riesgo es definiendo canales de lectura o escritura, pero no ambos.





# Worker pools





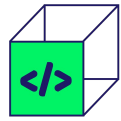
# Worker pools

Los worker pools permiten la creación de múltiples trabajadores que llevarán a cabo determinadas tareas, en este caso Go puede explotar las GoRoutines para alcanzar worker pools concurrentes.





# Multiplexación



# Multiplexación

Cuando una rutina se está comunicando con varios channels es muy útil utilizar la palabra reservada `select` para poder interactuar de una manera más ordenada con todos los mensajes que están siendo recibidos.





# **Definiendo Workers, Jobs y Dispatchers**






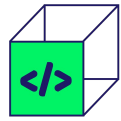
# Definiendo Workers, Jobs y Dispatchers

Golang permite utilizar conceptos como los de *worker pools* y, en combinación con *buffered channels*, la creación de *job queues* que utilizando concurrencia nos permitirán tener un alto rendimiento a la hora de la creación de muchas tareas.





# **Creando web server para procesar jobs**



# Creando web server para procesar jobs

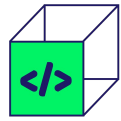
Con la librería estándar de Go y utilizando el paquete net, somos capaces de crear un servidor que será el que atienda las peticiones y asignará los nuevos workers para que lleven a cabo los trabajos que se está buscando conseguir.





# Resumen del curso



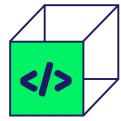


# Resumen del curso

## **Conceptos que ahora manejas**

- Puedes responder una de las grandes incógnitas de Go, aplicar la mayoría de los conceptos de POO y generar código con altos estándares.
- Ya puedes generar tus propios módulos y compartirlos con los demás manteniendo un registro de las dependencias.





# Resumen del curso

## **Conceptos que ahora manejas**

Eres capaz de crear tests unitarios para tu código, analizar las partes críticas que podrías mejorar y tener una métrica de qué código necesitas testear para comprobar la existencia de bugs.



# Resumen del curso

## **Conceptos que ahora manejas**

Tienes conocimiento de los diferentes tipos de Channels y lo que involucra cada uno, errores comunes que podrías enfrentar como ser Deadlocks y utilizarlos para crear semáforos a la hora de ejecutar GoRoutines.





# Resumen del curso

## **Conceptos que ahora manejas**

- Has aprendido nuevas técnicas para la concurrencia; como usar multiplexación, worker pools y waitgroups.
- Has creado una job queue concurrente que es capaz de ejecutar múltiples tareas utilizando GoRoutines.