

# Aprende a programar

## El lenguaje de programación de Google (golang)

[Video promo](#)

[Guía del curso](#)

[¿Por Qué Go?](#)

[Cómo tener éxito](#)

[Recursos del Curso](#)

[Documentación](#)

[Acelerando el aprendizaje](#)

[Variables, valores y tipo](#)

[Playground](#)

[Hola mundo](#)

[Operador de declaración corta](#)

[La palabra clave var](#)

[Explorando tipos de datos](#)

[Valor Cero](#)

[El paquete fmt](#)

[Creando tu propio tipo](#)

[Conversión, no casting](#)

[Ejercicios - Ninja Nivel 1](#)

[Contribuye con tu código](#)

[Ejercicio práctico #1](#)

[Ejercicio práctico #2](#)

[Ejercicio práctico #3](#)

[Ejercicio práctico #4](#)

[Ejercicio práctico #5](#)

[Ejercicio práctico #6](#)

[Fundamentos de Programación](#)

[Tipo Booleano](#)

[Cómo funciona la computadora](#)

[Tipos Numéricos](#)

[Tipo String](#)

[Sistemas numéricos](#)

[Constantes](#)

[lota](#)

[Bit shifting](#)

[Ejercicios prácticos - Ninja Nivel 2](#)

[Ejercicio práctico #1](#)

[Ejercicio práctico #2](#)

[Ejercicio práctico #3](#)

[Ejercicio práctico #4](#)

[Ejercicio práctico #5](#)

[Ejercicio práctico #6](#)

[Ejercicio práctico #7](#)

## [Control de Flujo](#)

[Entendiendo el control de flujo](#)

[Ciclo - init, condition, post](#)

[Ciclo - ciclos anidados](#)

[Ciclo - Declaración for](#)

[Ciclo - break & continue](#)

[Ciclo - Imprimiendo ascii](#)

[Condicional - Declaración if](#)

[Condicional - if, else if, else](#)

[Ciclo, condicional, módulo](#)

[Condicional - Declaración switch](#)

[Condicional - documentación de switch](#)

[Operadores lógicos condicionales](#)

## [Ejercicios - Ninja Nivel 3](#)

[Ejercicio práctico #1](#)

[Ejercicio práctico #2](#)

[Ejercicio Práctico #3](#)

[Ejercicio Práctico #4](#)

[Ejercicio Práctico #5](#)

[Ejercicio Práctico #6](#)

[Ejercicio Práctico #7](#)

[Ejercicio Práctico #8](#)

[Ejercicio Práctico #9](#)

[Ejercicio Práctico #10](#)

## [Agrupando Datos](#)

[Arreglos \(arrays\)](#)

[Slice - literal compuesto](#)

[Slice - for range](#)

[Slice - Dividiendo un slice](#)

[Slice - Añadiendo a un slice](#)

[Slice - Eliminando elementos de un slice](#)

[Slice - make](#)

[Slice - slices multidimensionales](#)

[Slice - El arreglo subyacente](#)

[Mapa \(map\) - introducción](#)

[Map - Agregar elemento y range](#)

[Map - Borrar](#)

#### [Ejercicios - Ninja Nivel 4](#)

[Ejercicio Práctico #1](#)

[Ejercicio Práctico #2](#)

[Ejercicio Práctico #3](#)

[Ejercicio Práctico #4](#)

[Ejercicio Práctico #5](#)

[Ejercicio Práctico #6](#)

[Ejercicio Práctico #7](#)

[Ejercicio Práctico #8](#)

[Ejercicio Práctico #9](#)

[Ejercicio Práctico #10](#)

#### [Structs](#)

[Struct](#)

[Structs embebidos](#)

[Leyendo la documentación](#)

[Structs Anónimos](#)

[Aclaración](#)

#### [Ejercicios - Ninja Nivel 5](#)

[Ejercicio práctico #1](#)

[Ejercicio práctico #2](#)

[Ejercicio práctico #3](#)

[Ejercicio práctico #4](#)

#### [Funciones](#)

[Sintaxis](#)

[Parámetros variados](#)

[Desplegando un slice](#)

[Defer](#)

[Métodos](#)

[Interfaces y polimorfismo](#)

[Funciones Anónimas](#)

[Expresión función](#)

[Retornando una func](#)

[Callback](#)

[Closure](#)

[Recursividad](#)

## Ejercicios - Ninja Nivel 6

[Ejercicio práctico #1](#)

[Ejercicio práctico #2](#)

[Ejercicio práctico #3](#)

[Ejercicio práctico #4](#)

[Ejercicio práctico #5](#)

[Ejercicio práctico #6](#)

[Ejercicio práctico #7](#)

[Ejercicio práctico #8](#)

[Ejercicio práctico #9](#)

[Ejercicio práctico #10](#)

[Ejercicio práctico #11](#)

## Pointers

[Que son punteros \(pointers\)?](#)

[Cuando usar pointers](#)

[Method Sets](#)

## Ejercicios - Ninja Nivel 7

[Ejercicio Práctico #1](#)

[Ejercicio Práctico #2](#)

## Aplicación

[Documentación JSON](#)

[JSON marshal](#)

[JSON unmarshal](#)

[Interfaz Writer](#)

[Sort](#)

[Sort Personalizado](#)

[bcrypt](#)

## Ejercicios - Ninja Nivel 8

[Ejercicio Práctico #1](#)

[Ejercicio Práctico #2](#)

[Ejercicio Práctico #3](#)

[Ejercicio Práctico #4](#)

[Ejercicio Práctico #5](#)

## Concurrencia

[Concurrencia vs paralelismo](#)

[WaitGroup](#)

[Revisión de Method sets](#)

[Documentación](#)

[Condición Race](#)

[Mutex](#)

[Atomic](#)

## [Tu entorno de desarrollo](#)

[La terminal](#)

[Bash en Windows](#)

[Instalando Go](#)

[El Go workspace](#)

[Variables de entorno](#)

[IDE's y Editores de Texto](#)

[Comandos Go](#)

[Repos Git](#)

[Explorando Gitlab](#)

[Compilación cruzada](#)

[Paquetes](#)

## [Ejercicios - Ninja Nivel 9](#)

[Ejercicio práctico #1](#)

[Ejercicio práctico #2](#)

[Ejercicio práctico #3](#)

[Ejercicio práctico #4](#)

[Ejercicio práctico #5](#)

[Ejercicio práctico #6](#)

[Ejercicio práctico #7](#)

## [Channels \(Canales\)](#)

[Entendiendo canales \(channels\)](#)

[Canales direccionales](#)

[Usando canales](#)

[Range](#)

[Select](#)

[Idioma Coma ok](#)

[Fan in](#)

[Fan out](#)

[Context](#)

## [Ejercicios - Ninja Nivel 10](#)

[Ejercicio Práctico #1](#)

[Ejercicio Práctico #2](#)

[Ejercicio Práctico #3](#)

[Ejercicio Práctico #4](#)

[Ejercicio Práctico #5](#)

[Ejercicio Práctico #6](#)

## Ejercicio Práctico #7

### Manejo de Errores

#### Entendiendo

¿Porqué Go no tiene excepciones?

#### Chequeando errors

#### Printing & logging

#### Recover

#### Errores con info

### Ejercicios - Ninja Nivel 11

#### Ejercicio Práctico #1

#### Ejercicio Práctico #2

#### Ejercicio Práctico #3

#### Ejercicio Práctico #4

#### Ejercicio Práctico #5

### Escribiendo Documentación

#### Introducción

#### go doc

#### godoc

#### godoc.org

#### Escribiendo Documentación

### Ejercicios - Ninja Nivel 12

#### Ejercicio Práctico #1

#### Ejercicio Práctico #2

#### Ejercicio Práctico #3

### Testing & Benchmarking

#### Introducción

#### Tabla de tests

#### Example tests

#### Golint

#### Benchmark

#### Coverage (Cobertura)

#### Ejemplos de Benchmark

#### Revisión

### Ejercicios - Ninja Nivel 13

#### Ejercicio Práctico #1

#### Ejercicio Práctico #2

#### Ejercicio Práctico #3

### DESPEDIDA

# Video promo

## 1. bienvenido

Bienvenido a ...

## 2. credenciales

Mi nombre es Eduar Tua y ...

## 3. beneficios

Estoy contento de poder enseñarte ...

## 4. curriculum

El currículo que estaremos estudiando es ...

## 5. audiencia

Para quienes son nuevos en programación ... Para quienes poseen algo de experiencia ...

## 6. call to action

video: 00x

# Guía del curso

## ¿Por Qué Go?

- [Go es uno de los lenguajes mejor pagados en América](#)
- Credenciales de Go
  - Google
  - **Rob Pike**
    - Unix, UTF-8
  - **Robert Griesemer**
    - Estudio bajo la tutela del creador de Pascal
  - **Ken Thompson**
    - Único responsable del diseño e implementación del sistema operativo original Unix.
    - Inventó el lenguaje de programación B, El predecesor directo del lenguaje C. Si, leíste bien - el inventó el lenguaje de programación que vino antes de C.
- [Porqué Go](#)
  - **Compilación eficiente**
    - Go crea programas compilados
      - Hay un garbage collector (GC)
      - No hay máquina virtual
  - **Ejecución eficiente**
  - **Fácil de programar**
  - El propósito de Go
- Para que es bueno Go

- Lo que hace Google / Servicios web en escala
- networking
  - http, tcp, udp
- Concurrencia / paralelismo
- Sistemas
- Automatización, herramientas de línea de comandos
- [Principios Guía de diseño](#)
  - Expresivo, exhaustivo, sofisticado.
  - Limpio, claro, fácil de leer.
- [Compañías usando Go](#)
  - Google, YouTube, Google Confidential, Docker, Kubernetes, InfluxDB, Twitter, Apple

Archivos: [Porqué Go - La presentación](#)

video: 001

## Cómo tener éxito

Entendiendo lo que ha convertido a algunos en personas exitosas, puede ayudarnos a convertirnos en personas exitosas también. Éstos son principios que me han ayudado a ser más exitoso. Aprendiendo de otros y de experiencia propia. **Los comparto para que te ayuden a ser también muy exitoso en el curso y en el día a día de nuestra vida cotidiana:**

- Enfocarse
  - Bill Gates y Warren Buffett
- Ejecutar toda tarea tiempo
- Firmeza y determinación
- Bill Gates, "Ponte en frente de lo que viene y deja que te golpee."
- 7 [Hábitos de personas efectivas](#)
- profesores
  - Gota a gota se llena el recipiente.
  - persistentemente, pacientemente, encontrarás el éxito
  - Cómo lograr ver el futuro
  - Técnica Pomodoro
  - Video de **Richard St. John: 8 secrets of success**
  -

video: 002

## Recursos del Curso

Este documento de contenido del curso forma parte del mismo. **Por favor lea todas las descripciones de los videos en esta guía.** Esto forma parte del proceso de aprendizaje.

Cuando lees la descripciones:

- Los conceptos que estás aprendiendo se fortalecen



- Te aprendes el material más rápido

Además, algunas veces proveo información adicional en las descripciones de los videos. Algunas veces grabo una clase, luego recuerdo que hay algún recurso u otro tipo de información, el cuál deberías saber. Algunos de esos recursos son bastante valiosos.

#### Ejemplos de Código

- downloading zip
- searching for code

Sitio Web de Caleb Doxsey y su libro

<http://www.doxsey.net/>

<http://amzn.to/1OnFtPY>

go by example

<https://gobyexample.com/>

Libro de Bill Kennedy

<http://amzn.to/1kGGsPv>

Libro de Donovan and Kernighan

<http://amzn.to/1RIM5HP>

#### **Eduar Tua en YouTube**

<https://www.youtube.com/user/eduartua>

#### **En Twitter**

<https://twitter.com/eduartua>

#### **Archivos:**

- **Outline del curso:** <https://goo.gl/RxVxBJ>
- **Recursos:** <https://goo.gl/3zbi7y>

video: 003

## Documentación

- **Documentación oficial**
  - [language spec](#)
  - [effective go](#)
- **golang.org vs godoc.org**
  - [golang.org](#)
    - Librería estándar
    - src

- godoc.org
  - Librería estándar Y paquetes de terceros

video: 004

## Acelerando el aprendizaje

Puedes **incrementar la velocidad de los videos que estás reproduciendo**. No todos lo sabemos. Es algo que deberías incluir y practicar cuando comienzas tus cursos. Ver video con velocidad aumentada ayuda a muchos estudiantes. También si en el Outline del curso la tabla de contenido no está activada, puedes activarla yendo a **vista y luego show document outline**.


video: 005

# Variables, valores y tipo

## Playground

- share
- import
- format
- run
- editors / IDEs
  - Configuramos luego
  - VS Code
  - JetBrains gogland
  - Atom
  - Sublime
- <https://forum.golangbridge.org/>
- **“Go idiomatico”**
  - idioms son patrones de lenguaje
  - Cuando alguien escribe “Go idiomático” está escribiendo código en Go de la forma es que es escrito por la comunidad de Go.

# id·i·om

/ˈɪdēəm/ 

*noun*

plural noun: **idioms**

1. a group of words established by usage as having a meaning not deducible from those of the individual words (e.g., *rain cats and dogs*, *see the light*).  
*synonyms:* **language**, mode of expression, turn of phrase, **style**, **speech**, **locution**, **diction**, **usage**, **phraseology**, **phrasing**, **phrase**, **vocabulary**, **terminology**, **parlance**, **jargon**, **argot**, **cant**, **patter**, **tongue**, **vernacular**, *informal lingo*  
"these musicians all work in the gospel idiom"
2. a characteristic mode of expression in music or art.  
"they were both working in a neo-impressionist idiom"



Translations, word origin, and more definitions

video: 006

## Hola mundo

- Estructura básica de un programa en Go:
  - package main
  - **func main**
    - **Punto de entrada o inicio a tu programa**
    - Cuando tu código sale de la función main, tu programa ha terminado
- Cantidad variable de parámetros
  - el "...<algún tipo>" es como especificamos un **variadic parameter**
  - El tipo "interface{}" es una interfaz vacía
    - Todos los tipos son de tipo "**interface{}"**
  - Entonces el parámetro "...interface{}" significa "ingresa la cantidad de argumentos que quieras"
- Deshaciéndose de los returns
  - uso **del caracter "\_"** para deshacerte de los returns
- No puedes tener variables sin usar en tu código
  - Esto es contaminación de código
  - El compilador no lo permite
- Usamos esta notación en Go
  - **paquete.Identificador**
    - ejemplo: fmt.Println()
      - Leeremos: "del paquete fmt Estoy usando la función Println"
  - Un identificador es el nombre de una variable, constante, función
- paquetes

- Código que está ya escrito el cual puedes usar
- imports

código: <https://play.golang.org/p/8PsiA-cHV0F>

video: 007

## Operador de declaración corta

- Terminología
  - **Palabras claves**
    - Son palabras reservadas para uso interno de Go
      - Algunas veces se les llama “palabras reservadas”
      - No puedes usar una palabra clave para algo distinto a lo que está destinada.
  - **Operador**
    - En “2 + 4” el “+” es el OPERADOR
    - Un operador es un caracter que representa una acción, como por ejemplo “+” es un OPERADOR aritmético que representa adición.
    -
  - **operando**
    - En “2 + 2” los “2” son OPERANDOS
  - **Declaración**
    - En programación una declaración es el elemento más pequeño de un programa que expresa una acción que va a ser llevada a cabo. Es una instrucción que la cual le da el comando a la computadora para ejecutar una acción específica. Un programa es formado por una secuencia de una o más declaraciones.
  - **Expresión**
    - En programación una expresión es una combinación de uno o más valores explícitos, constantes, variables, operadores y funciones que el lenguaje de programación interpreta y computa para producir otro valor. For ejemplo, 2+3 es una expresión el cual evalúa a 5.
- Mascota de golang
  - <https://github.com/golang/go/tree/master/doc/gopher>
  - Renne French

código: <https://play.golang.org/p/5JEX1q2H9qb>

video: 008

## La palabra clave var

- **Parentesis**

( )

- **Llaves**

{ }

- Dónde puede ser usada var
  - Cualquier lugar dentro del paquete
- scope
  - Dónde una variable existe y es accesible
  - Buena práctica: mantén el scope lo más “reducido” posible

código: [https://play.golang.org/p/c\\_1zxUfGmQN](https://play.golang.org/p/c_1zxUfGmQN)

video: 009

## Explorando tipos de datos

- DECLARAR una VARIABLE de un cierto TIPO, sólo puede contener VALORES de ese TIPO
- “Go suffers no fools.”
  - like “dead men tell no tales”
- var z int = 21
  - package scope
- **Tipos de datos primitivos**
  - En ciencias de computación, un tipo de datos primitivo es uno de los siguientes:
    - **Un tipo de datos básico** es un tipo de datos que proporcionado por un lenguaje de programación como un bloque básico de construcción de código. La mayoría de los lenguajes permiten recursivamente construir tipos de datos compuestos más complejos, utilizando como inicio tipos de datos básicos.
    - **Un tipo de datos interno** es un tipo de datos al cual el lenguaje proporciona soporte interno.
  - En la mayoría de los lenguajes de programación, todos los tipos de datos básicos son de incorporación interna del lenguaje. Además, muchos lenguajes también proporcionan un conjunto de tipos de datos compuestos. Hay varias opiniones de si un tipo de dato incorporado internamente en el lenguaje que no es básico debería ser considerado “primitivo”.
  - [https://en.wikipedia.org/wiki/Primitive\\_data\\_type](https://en.wikipedia.org/wiki/Primitive_data_type)
- **Tipos de datos compuestos**
  - En ciencias de computación, un **tipo de datos compuesto** es cualquier tipo de dato el cual puede ser construido en un programa usando los tipos de datos internos del lenguaje de programación u otros tipos de datos compuestos. Algunas veces es llamado **structure** o **aggregate data type**, aunque el último término puede referirse a arreglos, listas, entre otros. **El acto de construir un tipo de datos compuesto es conocido como *composición***

code:

- <https://play.golang.org/p/1UVCK5QddGU>
- <https://play.golang.org/p/LrI7oRoVzz>

video: 010

## Valor Cero

- Entendiendo el **valor cero**
  - false para booleanos
  - 0 para integers
  - 0.0 para floats
  - "" para strings
  - nil para
    - pointers
    - funciones
    - interfaces
    - slices
    - channels
    - maps
- Usar el operador de declaración corta de variables mientras más sea posible
- Usar var para
  - Valor cero
  - package scope

code: [https://play.golang.org/p/v3ez\\_kTwjn](https://play.golang.org/p/v3ez_kTwjn)

video: 011

## El paquete fmt

- Configuración básica del código
  - usando var
    - using zero value
  - Usando el operador de declaración corta
  - [https://play.golang.org/p/jynLS\\_0PaI9](https://play.golang.org/p/jynLS_0PaI9)
  - <https://play.golang.org/p/j2Nx1wdgC6>
- %v
  - <https://play.golang.org/p/51WOG55ril>
- Caracteres de escape como \n o \t
  - [https://golang.org/ref/spec#Rune\\_literals](https://golang.org/ref/spec#Rune_literals)
- **Imprimiendo con format**
  - <https://play.golang.org/p/eTdCR8DzniO>
  - Diferencia entre las funciones que encontramos en el paquete "fmt"
    - grupo #1 - Impresión general a la salida estándar
      - [func Print\(a ...interface{}\) \(n int, err error\)](#)
      - [func Printf\(format string, a ...interface{}\) \(n int, err error\)](#)
      - [func Println\(a ...interface{}\) \(n int, err error\)](#)
    - grupo #2 - imprimiendo a un string el cual puede ser asignado a una variable
      - [func Sprint\(a ...interface{}\) string](#)

- [func Sprintf\(format string, a ...interface{}\) string](#)
- [func Sprintln\(a ...interface{}\) string](#)
- group #3 - Imprimiendo a un archivo o a una a la respuesta de una petición hecha a un servidor web
  - [func Fprint\(w io.Writer, a ...interface{}\) \(n int, err error\)](#)
  - [func Fprintf\(w io.Writer, format string, a ...interface{}\) \(n int, err error\)](#)
  - [func Fprintln\(w io.Writer, a ...interface{}\) \(n int, err error\)](#)

video: 012

## Creando tu propio tipo

- **Podemos crear nuestro propio tipo de dato en Go**
  - <https://play.golang.org/p/K2MC5M2Vk5E>

video: 013

## Conversión, no casting

**Go tiene un lenguaje para hablar de su lenguaje.** Algunos términos han sido descartados porque traen asociados conceptos diferentes. Una nueva mirada al futuro de la programación. Nuevas palabras para hablar de algunos conceptos. No hablamos de objetos, en Go hablamos de crear TIPOS y VALORES de ciertos TIPOS. No hablamos de casting, hablamos de CONVERSIÓN y ASERCIÓN.

- [https://play.golang.org/p/11bTFIgQS\\_v](https://play.golang.org/p/11bTFIgQS_v)

video: 014

# Ejercicios - Ninja Nivel 1

## Contribuye con tu código

A medida que vayas avanzando en los ejercicios prácticos, si llegas a escribir código el cuál te gustaría compartir con el resto de la clase, envíame un tweet con el link ( <https://twitter.com/eduartua> ) y lo agregare al outline del curso.

video: 015

## Ejercicio práctico #1

- Usando el operador de declaración corta, **ASIGNA** los siguientes **VALORES** a **VARIABLES** con los IDENTIFICADORES "x", "y" y "z"
  - 42
  - "James Bond"
  - true
- Luego imprime los valores almacenados en esas variables usando
  - Una sola declaración de la función println

- b. Múltiples declaraciones de println

código: Aquí la solución: <https://play.golang.org/p/yYXnWXIQNa>

video: 016

## Ejercicio práctico #2

1. Usa var para **DECLARAR** tres **VARIABLES**. Las variables deben tener scope a nivel de paquete. No asignar **VALORES** a las variables. Usa los siguientes **IDENTIFICADORES** para las variables y asegúrate de que las variables son de los siguientes TIPOS (lo quiere decir que pueden almacenar VALORES de ese **TIPO**).
  - a. identificador "x" tipo int
  - b. identificador "y" tipo string
  - c. identificador "z" tipo bool
2. En main
  - a. Imprime los valores de cada identificador
  - b. El compilador asigna valores a las variables. ¿Cómo son llamados esos valores?

código: <https://play.golang.org/p/jzHwSlles9>

video: 017

## Ejercicio práctico #3

Usando el código del ejercicio anterior,

1. En scope a nivel de paquete, asigna los siguientes valores a las tres variables
  - a. a x asigne 42
  - b. a y asigne "James Bond"
  - c. a z asigne true
2. en main
  - a. Usa fmt.Sprintf para imprimir todos los VALORES en un solo string. ASIGNA el valor retornado de TIPO string usando el operador de declaración corta a la VARIABLE con el IDENTIFICADOR "s"
  - b. Imprime el valor almacenado por la variable "s"

código: [https://play.golang.org/p/QFctSQB\\_h3](https://play.golang.org/p/QFctSQB_h3)

video: 018

## Ejercicio práctico #4

- Información - documentación interesante y nueva terminología "**tipo subyacente, raíz o tipo implícito**"
  - <https://golang.org/ref/spec#Types>

Para este ejercicio

1. Crea tu propio tipo. Haz que el tipo subyacente, raíz o implícito sea un int.
2. Crea una VARIABLE de tu nuevo TIPO con el IDENTIFICADOR "x" usando la palabra clave "VAR"
3. En func main



- a. Imprime el valor de la variable “x”
- b. Imprime el tipo de la variable “x”
- c. Asigna 42 a la VARIABLE “x” usando el OPERADOR “=”
- d. Imprime el valor de la variable “x”

código: <https://play.golang.org/p/HzqoNjYYF2a>

video: 019

## Ejercicio práctico #5

A partir del código del ejercicio anterior

1. A nivel de paquete usando la palabra clave “var”, crear una VARIABLE con el IDENTIFICADOR “y”. La variable debería ser del mismo TIPO SUBYACENTE que tu TIPO “x” creado anteriormente
  - a. ejemplo:

```
type hotdog int

var x hotdog
var y int
```

2. en func main
  - a. Esto lo debería tener listo
    - i. Imprimir el valor de la variable “x”
    - ii. Imprimir el tipo de la variable “x”
    - iii. Asigna un VALOR a la VARIABLE “x” usando el OPERADOR “=”
    - iv. Imprime el valor de la variable “x”
  - b. Ahora haces esto
    - i. Ahora usa CONVERSIÓN para convertir el TIPO del VALOR almacenado en “x” al TIPO IMPLÍCITO
      1. Usa el operador “=” para ASIGNAR ese valor a “y”
      2. Imprime el valor almacenado en “y”
      3. Imprime el tipo de “y”

código: aquí la solución: <https://play.golang.org/p/uxlqNOSemOF>

video: 020

## Ejercicio práctico #6

[Realiza este quiz.](#)

video: 021

# Fundamentos de Programación

## Tipo Booleano

1. Un **booleano** es un TIPO binario el cual puede tener dos posibles valores “verdadero” y “falso”
2. Cuando usas un **operador de comparación de igualdad**, este es una expresión el cual evalúa a un valor booleano
  - a. **==**
  - b. **<=**
  - c. **>=**
  - d. **!=**
  - e. **<**
  - f. **>**

code

1. <https://play.golang.org/p/WmPsOkVzwS>
2. [https://play.golang.org/p/inS\\_7F0HdC](https://play.golang.org/p/inS_7F0HdC)

video: 022

## Cómo funciona la computadora

- computers run on electricity
- electricity has two discrete states: ON and OFF
- we can create **coding schemes** for “on” or “off” states

files: [presentación](#)

video: 023

## Tipos Numéricos

- Enteros
  - Números sin decimales
    - También conocidos como enteros
    -
  - int & uint
    - “Tamaños de implementación-específica”
  - Todos los tipos numéricos son diferentes excepto
    - **byte** el cual es un alias para **uint8**
    - **Rune** el cual es un alias para **int32**
  - Los tipos son únicos
    - “Esto es un lenguaje de programación estático”
    - “Las conversiones son requeridas cuando mezclamos diferentes tipos de datos numéricos en una expresión o asignación. Por ejemplo, int32 e int no son del mismo tipo aún cuando pueden tener el mismo tamaño en una arquitectura particular.” ([fuente](#))

- Regla de oro: solo usa **int**
- floating point
  - Números con decimales
    - También conocidos como números reales
  - Regla de oro: solo usa **float64**
- Overflows
- Lectura recomendada - [Libro de Caleb Doxsey](#)

código:

- <https://play.golang.org/p/OdWUH8uva6>
- <https://play.golang.org/p/0JpmCYezs1>
- Este no corre: <https://play.golang.org/p/O7nFEn8nXz>
- int8
  - <https://play.golang.org/p/lcOtgm6YKA>
  - No corre: <https://play.golang.org/p/YbwTa1YT4i>
  - <https://play.golang.org/p/exwG0ijjRf>
  - No corre: <https://play.golang.org/p/sy16rgifWF>

runtime package

- GOOS
- GOARCH
- <https://play.golang.org/p/1vp5DImIMM>

video: 024

## Tipo String

- [Blog post de golang sobre strings](#)
- “Un valor string es una (posiblemente vacía) secuencia de bytes. Los strings son inmutables: una vez creados, es imposible cambiar su contenido.”

code:

- Buenos ejemplos - lecture prep:
  - <https://play.golang.org/p/LUbFEJEope>
  - <https://play.golang.org/p/JjWLMcAsCU>
- live coding:
  - <https://play.golang.org/p/iBen9LOcpBG>

video: 025

## Sistemas numéricos

Como humanos, tenemos diferentes sistemas para expresar las cantidades de algo. Usando el sistema numérico decimal, podemos decir que tenemos 7 naranjas; 42 manzanas o 20 dólares. Otros sistemas numéricos usados en computación incluyen el **sistema numérico binario** y el **sistema numérico hexadecimal**.

video: 026

## Constantes

- Un simple valor que no cambia.
- Sólo existe en el momento de compilación..
- Hay constantes con **TIPO** y **SIN TIPO**
  - `const hola = "Hola, mundo"`
  - `const typedHello string = "Hello, World"`
- Constante SIN TIPO
  - Un valor constante que no tiene un tipo fijo
    - “constante de algún **tipo**”
    - No es forzada a obedecer las reglas estrictas que previenen combinar diferentemente valores con un tipo.
  - Una constante sin tipo puede ser implícitamente convertida por el compilador.
- Es este concepto de constante *sin tipo* lo que hace posible que usemos constantes en Go con libertad.
  - Muy útil, por ejemplo
    - Cuál es el tipo de 42?
      - `int?`
      - `uint?`
      - `float64?`
    - Si no tuviéramos constante SIN TIPO (constantes de algún tipo), tendríamos que hacer conversión en cada valor literal que usamos.
      - Y eso no sería muy agradable
- Código usado en el video
  - [https://play.golang.org/p/GuCvb\\_xlvAo](https://play.golang.org/p/GuCvb_xlvAo)
- Preparación
  - <https://play.golang.org/p/IVURPQe-N4>

video: 027

## Iota

Dentro de una declaración, el identificador pre-declarado *iota* representa una **sucesión de constantes enteras sin tipo**. Es reiniciado a 0 cada vez que la palabra constante aparece en el código. Puede ser usada para construir un conjunto de constantes relacionadas:

- código usado en el video
  - [https://play.golang.org/p/\\_cSkz\\_b28t](https://play.golang.org/p/_cSkz_b28t)
- lecture prep
  - <https://play.golang.org/p/YOabnTj5OI>
  - <https://play.golang.org/p/c5SmcFzzBM>

video: 028

## Bit shifting

Bit shifting es cuando **desplazan dígitos binarios a la izquierda o a la derecha**. Podemos usar bit shifting junto con iota, para construir constantes creativas.

**Buen artículo**

- <https://goo.gl/gnbjMQ>

**Código en video**

- <https://play.golang.org/p/YnMevatXIP>
- [https://play.golang.org/p/q\\_IGHQ2am4](https://play.golang.org/p/q_IGHQ2am4)
- <https://play.golang.org/p/lwNVIOcrLG>

**Código para preparación:**

- <https://play.golang.org/p/3oxB39hYJ>
- <https://play.golang.org/p/7MOnbhx4R4>
- Este es instructivo, muestra todos los tipos de shifting:  
<https://play.golang.org/p/DK6Ub7Sotx>

**video: 029**

## Ejercicios prácticos - Ninja Nivel 2

### Ejercicio práctico #1

Escribe un programa que imprima un número en **decimal, binario, y hexadecimal**

solución: <https://play.golang.org/p/bAQxcEuK8O>

**video: 030**

### Ejercicio práctico #2

Usando los siguientes operadores, escribe expresiones y asigna sus valores a variables:

- a. ==
- b. <=
- c. >=
- d. !=
- e. <
- f. >

Imprime los valores de las variables.

Solución: <https://play.golang.org/p/76R-poSzaY>

**video: 031**

### Ejercicio práctico #3

Crea **constantes CON TIPO y SIN TIPO**. Imprime el valor de las mismas.

solución: <https://play.golang.org/p/NutvJXWUx2>

**video: 032**

## Ejercicio práctico #4

Escribe un programa que

- Asignar un int a una variable
- Imprímelo en decimal, binario, y hex
- **Has shifts de bits** de ese int una posición a la izquierda y asigna eso a una variable
- Imprime esa variable en decimal, binario, y hex

solución: <https://play.golang.org/p/Ms964T8SbH>

video: 033

## Ejercicio práctico #5

Crea una variable de tipo string usando un **string literal no interpretado (raw string literal)**.

Imprímelo.

solución: <https://play.golang.org/p/dLy36A-V-w>

video: 034

## Ejercicio práctico #6

Usando **iota**, crea 4 constantes para los **PRÓXIMOS** 4 años. Imprime los valores de las constantes.

solución: <https://play.golang.org/p/MDLF3v5EGT>

video: 035

## Ejercicio práctico #7

[Haz este quiz](#)

video: 036

# Control de Flujo

## Entendiendo el control de flujo

La computadora lee los programas de cierta forma, así como nosotros leemos libros de cierta forma. Cuando, como humanos, leemos libros, en culturas occidentales, lo hacemos de la parte frontal hacia la trasera del libro, de izquierda a derecha y de arriba hacia abajo. Cuando las computadoras leen el código de un programa, lo hacen de arriba hacia abajo. Esto es conocido como lectura en SECUENCIA y a su vez conocido como control de flujo secuencial. Hay otras dos declaraciones el cuál pueden afectar cómo la computadora lee el código. Una computadora puede encontrarse con un CICLO o LOOP. Si se encuentra con uno de esos entrará en un bucle e iterará sobre él de una manera específica. Por eso es también conocido como control de flujo ITERATIVO. Finalmente, hay también control de flujo CONDICIONAL. Cuando la computadora se encuentra con una CONDICIÓN, como una "declaración if" o una "declaración

de switch" la computadora tomará acciones diferentes dependiendo de alguna condición. Entonces las tres formas en la que la computadora lee el código son: **SECUENCIAL, LOOP, CONDICIONAL**

- secuencia
- loop / iterativo
  - for loop
    - init, cond, post
  - bool (con while)
    - infinite
  - Con do-while
    - break
  - continuo
  - anidado
- condicionales
  - Declaraciones switch / case / default
    - Sin caída predeterminada
    - Creando caída (creating fall-through)
    - Casos múltiples
    - Casos pueden ser expresiones
      - Evaluar a true, ellos corren
    - tipo
  - if
    - El operador de negación
      - !
    - Declaración de inicialización
    - if / else
    - if / else if / else
    - if / else if / else if / ... / else

video: 037

## Ciclo - init, condition, post

- Ciclo for
  - **inicialización, condition, post**

code:

- init,condition, post <https://play.golang.org/p/Wp5cT2laMx>
- for: <https://play.golang.org/p/A0GUGbqi9>

video: 038

## Ciclo - ciclos anidados

- Ciclo for
  - **Ciclos anidados**
- código:
  - <https://play.golang.org/p/fZlnUiWE-YT>

- <https://play.golang.org/p/z0nR4dtQqYb>

video: 039

## Ciclo - Declaración for

Hay tres formas con las que puedes construir ciclos for en Go - todas usan la **palabra clave "for"**:

- for init; condición; post { }
- for condición { }
- for { }

Respecto a la documentación de la declaración "for"

- Especificaciones del lenguaje
- effective go

código

- <https://play.golang.org/p/7tsaMiSNLAK>
- <https://play.golang.org/p/NxFmachGpae>

video: 040

## Ciclo - break & continue

Ciclo For

- break
- continue

Encontrando el resto, también conocido como **modulus**

- %

código:

- resto: [https://play.golang.org/p/\\_BNQa7c8d8](https://play.golang.org/p/_BNQa7c8d8)
- break & continue: <https://play.golang.org/p/uqh2SDENAE>

video: 041

## Ciclo - Imprimiendo ascii

Podemos usar **impresión con formato** para imprimir:

- Valor decimal con **%d**
- Valor hexadecimal con **%#x**
- Código unipoint **%#U**
- Una tabulación con **\t**
- Una línea nueva con **\n**

código:

- <https://play.golang.org/p/SnzlisWesT>

video: 042

## Condicional - Declaración if

**Declaraciones If**



- booleano
  - true
  - false
- El operador de negación
  - !
- **Declaración de initialization**

código:

- <https://play.golang.org/p/vz-qKTA7a2>
- Declaración de inicialización: <https://play.golang.org/p/CxpQu6fLzN>

video: 043

## Condicional - if, else if, else

- if / else
  - <https://play.golang.org/p/GrPgcCVvFDv>
- if / else if / else
  - <https://play.golang.org/p/UjQZylQa8jM>
- **if / else if / else if / ... / else**
  - <https://play.golang.org/p/h5W2qA3tJdR>

video: 044

## Ciclo, condicional, módulo

Iteramos para mejorar nuestras vidas. Esto se cumple para cualquier cosa que estemos haciendo. Escribimos código con errores antes de que escribamos código sin errores. Un profesor una vez llamó a esto “perfección de la imperfección.” “Tú eres perfecto de la manera que eres, y siempre habrá oportunidad de mejora.” El punto es- el código que escribimos para imprimir número pares puede se hecho de una mejor manera. Vamos a escribir código que es más legible. Esto servirá como una revisión de los conceptos que estamos aprendiendo: **loops, condicionales, sentencia if y el operador módulo.**

video: 045

## Condicional - Declaración switch

### **Declaración Switch**

- Declaraciones switch / case / default
  - Sin caer en la declaración fall-through
  - Creando fall-through
  - multiples casos
  - casos pueden ser expresiones
    - Evalúan a verdadero, corren.

código:

- switch: <https://play.golang.org/p/-k8WY52ejRw>
- Sin default - con fallthrough: <https://play.golang.org/p/8q69SJsreQh>
- Con default - fallthrough: <https://play.golang.org/p/oViV02BIXZb>

- Default <https://play.golang.org/p/yFuoOAC3ZNY>
- Switch con valor - cuál coincide?: <https://play.golang.org/p/lk32aRDK0Ri>
- switch on multiple matches for a case: [https://play.golang.org/p/cidJ5iMrj\\_R](https://play.golang.org/p/cidJ5iMrj_R)

video: 046

## Condicional - documentación de switch

**Sentirse cómodo mientras leemos la documentación es importante.** Dedicar un poco de tiempo para que veamos la documentación de Go es importante, te será útil ver cómo se lee e interpreta la misma y te hará sentir más cómodo con ella.

video: 047

## Operadores lógicos condicionales

A qué evalúan las siguientes expresiones :

- `fmt.Println(2 == 2 && 3 == 3)`
- `fmt.Println(true && false)`
- `fmt.Println(5 == 5 || 6 == 6)`
- `fmt.Println(true || false)`
- `fmt.Println(!true)`

código

- <https://play.golang.org/p/ukFrIC66uv>
- <https://play.golang.org/p/WQ8PjCCC42>

([source](#))

video: 048

## Ejercicios - Ninja Nivel 3

### Ejercicio práctico #1

Imprime todos los números del 1 al 10,000

solución: <https://play.golang.org/p/voDiuiDGGw>

video: 049

### Ejercicio práctico #2

Imprime el rune code point de las letras del alfabeto en mayúsculas tres veces. La salida debe ser como esto:

65

U+0041 'A'

U+0041 'A'

U+0041 'A'

66

U+0042 'B'

U+0042 'B'

U+0042 'B'

... hasta el resto de letras en el alfabeto.

solución: <https://play.golang.org/p/1OjnCX1D5H>

**video: 050**

## Ejercicio Práctico #3

Crea un ciclo usando la siguiente sintaxis

- for condición { }

Haz que imprima los años que has vivido.

solución: <https://play.golang.org/p/tnyqBPJ-i5>

**video: 051**

## Ejercicio Práctico #4

Crea un ciclo for usando esta sintaxis

- for { }

Haz que imprima los años que has vivido.

solución: <https://play.golang.org/p/9VpnB-l1Pz>

**video: 052**

## Ejercicio Práctico #5

Imprime el resto o módulo, el cual es resultado de dividir entre 4 cada número en el rango de 10 y 100.

solución: [https://play.golang.org/p/Omf4\\_1Hutaw](https://play.golang.org/p/Omf4_1Hutaw)

**video: 053**

## Ejercicio Práctico #6

Crea un programa que muestre el “if statement” en acción.

solución: <https://play.golang.org/p/F94TBCLEi1L>

**video: 054**

## Ejercicio Práctico #7

Usando el ejercicio anterior, crea un programa que use “else if” y “else”.

solución: <https://play.golang.org/p/n8xZqaoXJe1>

**video: 055**

## Ejercicio Práctico #8

Crea un programa que use una declaración switch sin expresión especificada.

solución: <https://play.golang.org/p/dh4EdilaROJ>

**video: 056**

## Ejercicio Práctico #9

Crea un programa que use una declaración switch con la expresión de switch especificada como una variable de TIPO string y el IDENTIFICADOR “deporteFav”.

solution: <https://play.golang.org/p/3kvkltfuZbs>

video: 057

## Ejercicio Práctico #10

Escribe el resultado de las siguientes declaraciones:

- `fmt.Println(true && true)`
- `fmt.Println(true && false)`
- `fmt.Println(true || true)`
- `fmt.Println(true || false)`
- `fmt.Println(!true)`

solucion:

video: 058

## Agrupando Datos

- **Arreglo (Array)**
  - Es una secuencia enumerada de elementos de un mismo tipo.
  - No cambia en tamaño
  - **Usado por funcionalidades internas de Go; generalmente no es recomendado que lo uses en tu código.**
  - [https://golang.org/ref/spec#Array\\_types](https://golang.org/ref/spec#Array_types)
- **Slice**
  - Están contruidos sobre el tipo arreglo.
  - **Los valores que están contenidos en un slice son del mismo tipo.**
  - Este si puede cambiar en tamaño.
  - Tiene una longitud y una capacidad.
  - [https://golang.org/ref/spec#Slice\\_types](https://golang.org/ref/spec#Slice_types)
- **map**
  - **Almacenamiento key / value**
  - Es un grupo de elementos de un tipo, llamado element type, **sin orden** de agrupación, indexados por un conjunto único de keys de otro tipo, llamado key type.
  - [https://golang.org/ref/spec#Map\\_types](https://golang.org/ref/spec#Map_types)
- **struct**
  - Una estructura de datos
  - Un tipo compuesto
  - Nos permite **poner juntos valores de diferentes tipos.**
  - [https://golang.org/ref/spec#Struct\\_types](https://golang.org/ref/spec#Struct_types)

## Arreglos (arrays)

Los arreglos son mayormente usados como un bloque constructor en el lenguaje de programación Go. En algunas circunstancias, podríamos usar un array, pero en la mayoría de los casos la recomendación es **usar slices en vez de arreglos**.

código

- <https://play.golang.org/p/f-7aufl2DO>

video: 059

## Slice - literal compuesto

UN SLICE almacena VALORES del mismo TIPO. Si quisiéramos almacenar todos nuestros números favoritos usamos un SLICE de ints. Si quisiera almacenar todas mis comidas favoritas usaría un SLICE de strings. Usaremos un **LITERAL COMPUESTO para crear un slice**. Un literal compuesto es cuando colocamos el TIPO seguido de LLAVES y luego colocamos los valores en el área dentro de las llaves.

código

- <https://play.golang.org/p/5hDQvSeNMKi>

video: 060

## Slice - for range

Podemos **iterar sobre los valores en un slice con la cláusula range**. También le podemos agregar elementos a un slice mediante el uso del índice.

código

- <https://play.golang.org/p/DUmOJht9D7a>

video: 061

## Slice - Dividiendo un slice

Podemos dividir un Slice, lo que quiere decir que podemos **cortar y desechar partes de un slice**. Hacemos esto con el operador dos puntos ( : ).

código

- [https://play.golang.org/p/uu7zeInpg\\_n](https://play.golang.org/p/uu7zeInpg_n)

video: 062

## Slice - Añadiendo a un slice

**Para añadir valores a un slice, debemos usar la función especial integrada append**. Esta función retorna un slice del mismo tipo.

código

- <https://play.golang.org/p/GB7DdkiBdBp>

video: 063

## Slice - Eliminado elementos de un slice

Podemos **eliminar elementos de un slice usando append y slicing (dividiendo)**. Este es un maravilloso y elegante ejemplo de porqué Go es súper cool y cómo provee facilidad de programación.

código:

<https://play.golang.org/p/f4OgTtgwckW>

video: 064

## Slice - make

Los slices son contruidos sobre los arreglos. Un slice es dinámico, así este crecerá en tamaño. Sin embargo, el arreglo subyacente, no crece en tamaño. Cuando creamos un slice, **podemos usar la función predefinida interna make para especificar que tan grande debería ser nuestro slice y también qué tan grande debería ser el arreglo subyacente**.

Esto puede mejorar un poco el desempeño del programa.

- `make([]T, length, capacity)`
- `make([]int, 50, 100)`

código

- <https://play.golang.org/p/07hH1b-hvD>

video: 065

## Slice - slices multidimensionales

Un slice multidimensional es como una hoja de cálculo. Puede tener **un slice de slices de algún tipo**. Suena confuso? En este video lo aclaramos.

code

- <https://play.golang.org/p/4QD0-9xPOaP>

video: 066

## Slice - El arreglo subyacente

Subyacente a cada slice habrá un arreglo (array). Un slice es realmente una estructura de datos el cual tiene tres partes:

1. Un puntero a un arreglo
2. Longitud (len)
3. Capacidad (cap)

En este video, vamos a explorar la relación entre el slice y el arreglo subyacente.

código

- Un nuevo arreglo es asignado
  - <https://play.golang.org/p/pDapelh8uQp>
- El mismo arreglo es usado para DOS slices
  - <https://play.golang.org/p/bHaOVUjFwWx>
  - También puedes “desechar” la variable y lo mismo ocurre
    - <https://play.golang.org/p/NL8p7Z8R3E>
- Viendo a LEN & CAP de un slice

- <https://play.golang.org/p/GvEnKI3x-A>

video: 067

## Mapa (map) - introducción

Un mapa es un tipo de dato de almacenamiento llave-valor. Esto quiere decir que almacenas algún valor y accedes al mismo con una llave. Por ejemplo, podría almacenar el valor “numeroTel” y acceder a él con la llave “nombreAmigo”. De esta manera puedo ingresar el nombre de mi amigo y el mapa me retornará su número telefónico. La sintaxis para el mapa es map[llave]valor. La llave puede ser de cualquier tipo que permita comparación (por ejemplo, podría usar un string o un int, ya que puedo comparar si dos strings son iguales o si dos enteros son iguales. Es importante denotar que los mapas son desordenados. Si imprimes todas las llaves y valores de un mapa se hará de forma aleatoria. El idioma **comma ok** es también cubierto en este video. **Un mapa es la estructura de datos perfecta cuando necesitas buscar datos de manera rápida.**

código:

- <https://play.golang.org/p/eyYVn605Nd7>

video: 068

## Map - Agregar elemento y range

Aquí mostramos como **agregar un elemento a un mapa**. También muestro **cómo usar el ciclo for range para imprimir** las llaves y valores de un mapa.

código:

- <https://play.golang.org/p/RTuBRiW087>

video: 069

## Map - Borrar

Borras un elemento de un mapa usando **delete(<nombre del mapa>, “llave”)**. No arroja ningún error si usas una llave que no existe. Para confirmar que borraste un par llave-valor, verifica que el par existe con el idioma **comma ok**.

código:

- <https://play.golang.org/p/HsAd0VGAD7X>

video: 070

# Ejercicios - Ninja Nivel 4

## Ejercicio Práctico #1

- Usando un COMPOSITE LITERAL:
  - Crea un ARREGLO el cual tenga 5 VALORES de TIPO int

- Asigna VALORES a cada posición dada por los índices.
- Itera con Range sobre el arreglo e imprime los valores.
- Usando el paquete fmt
  - Imprime el TIPO del arreglo

**solución:** <https://play.golang.org/p/tD0SzV3hdf>

**video:** 071

## Ejercicio Práctico #2

- Usando un COMPOSITE LITERAL:
  - Crea un SLICE de TIPO int
  - Asigna 10 VALORES
- Haz Range sobre el slice e imprime los valores.
- Usando format para imprimir
  - Imprime el TIPO del slice

**solución:** <https://play.golang.org/p/sAQeFB7DIs>

**video:** 072

## Ejercicio Práctico #3

Usando el código del ejercicio anterior, usa SLICING para crear los siguientes nuevos slices el cual serán impresos:

- [42 43 44 45 46]
- [47 48 49 50 51]
- [44 45 46 47 48]
- [43 44 45 46 47]

**solution:** <https://play.golang.org/p/SGfiULXzAB>

**video:** 073

## Ejercicio Práctico #4

Sigue los siguientes pasos:

- Comienza con este slice
  - `x := []int{42, 43, 44, 45, 46, 47, 48, 49, 50, 51}`
- Agrégale el siguiente valor
  - 52
- Imprime el slice
- En UNA DECLARACIÓN agrega al slice los siguientes valores
  - 53
  - 54
  - 55
- Imprime el slice
- Agregale al Slice los siguientes valores
  - `y := []int{56, 57, 58, 59, 60}`
- print out the slice



**solution:** <https://play.golang.org/p/QUYhtSBaDS>

**video:** 074

## Ejercicio Práctico #5

Para BORRAR de un slice, usamos APPEND en conjunto con SLICING(dividir). Para este ejercicio sigue los siguientes pasos:

- Comienza con un slice
  - `x := []int{42, 43, 44, 45, 46, 47, 48, 49, 50, 51}`
- Usa APPEND & SLICING para obtener los siguientes valores el cual se los debes asignar a una variable "y" y luego imprimir:
  - `[42, 43, 44, 48, 49, 50, 51]`

**solución:** <https://play.golang.org/p/u8zpHLfgSE>

**video:** 075

## Ejercicio Práctico #6

Crea un slice para almacenar los nombres de todos los estados en los Estados Unidos de América. ¿Cuál es la longitud del slice? ¿Cuál es la capacidad del Slice? Imprime todos los valores con su índice de posición, sin utilizar la cláusula range. Aquí la lista de los estados:

```
` Alabama`, ` Alaska`, ` Arizona`, ` Arkansas`, ` California`, ` Colorado`, ` Connecticut`, ` Delaware`, ` Florida`, ` Georgia`, ` Hawaii`, ` Idaho`, ` Illinois`, ` Indiana`, ` Iowa`, ` Kansas`, ` Kentucky`, ` Louisiana`, ` Maine`, ` Maryland`, ` Massachusetts`, ` Michigan`, ` Minnesota`, ` Mississippi`, ` Missouri`, ` Montana`, ` Nebraska`, ` Nevada`, ` New Hampshire`, ` New Jersey`, ` New Mexico`, ` New York`, ` North Carolina`, ` North Dakota`, ` Ohio`, ` Oklahoma`, ` Oregon`, ` Pennsylvania`, ` Rhode Island`, ` South Carolina`, ` South Dakota`, ` Tennessee`, ` Texas`, ` Utah`, ` Vermont`, ` Virginia`, ` Washington`, ` West Virginia`, ` Wisconsin`, ` Wyoming`,
```

**solución:** <https://play.golang.org/p/tRKQDQuQCE>

**video:** 076

## Ejercicio Práctico #7

Crear un slice de slice de string ([][]string). Almacena los siguientes valores en un slice multi-dimensional:

```
"Batman", "Jefe", "Vestido de negro"  
"Robin", "Ayudante", "Vestido de colores"
```

Haz range sobre los registros, luego sobre los datos de cada registro.

**solución:** <https://play.golang.org/p/1R2qljsrire>

**video:** 077

## Ejercicio Práctico #8

Crea un mapa con una llave TIPO string el cual representa el “nombre\_apellido” de una persona y un valor de TIPO []string el cual almacena sus cosas favoritas. Almacena tres registros en tu mapa. Imprime todos sus valores con su índice de posición en el slice.

```
`eduar_tua`, `computadoras`, `montaña`, `playa`  
`carlos_ramon`, `leer`, `comprar`, `música`  
`juan_bimba`, `helado`, `pintar`, `bailar`
```

**solución:** <https://play.golang.org/p/W0UwGn3VPLJ>

**video:** 078

## Ejercicio Práctico #9

Usando el código del ejemplo anterior, agrega un registro a tu mapa, ahora imprime el mapa usando “range”

**solución:** <https://play.golang.org/p/LJE4ZPsPsoB>

**video:** 079

## Ejercicio Práctico #10

Usando el código del ejemplo anterior, elimina un registro a tu mapa, ahora imprime el mapa usando “range”

**solución:** <https://play.golang.org/p/lAr9tdq0Esx>

**video:** 080

# Structs

## Struct

Un Struct es un tipo de dato compuesto. (tipos de datos compuestos, también, tipos de datos agregados o tipos de datos complejos). Los **structs nos permiten componer valores con otros valores de diferentes tipos.**

**código:** <https://play.golang.org/p/vBkiFGZpyDA>

**video:**081

## Structs embebidos

Podemos **tomar un struct y embeberlo en otro struct.** Cuando haces esto el tipo interno es promovido al tipo externo.

**código:** <https://play.golang.org/p/0WOiEeLv-si>

**video:** 082

## Leyendo la documentación

Es bueno familiarizarse con el lenguaje usado para hablar del lenguaje Go. Las "especificaciones del lenguaje" pueden resultar difíciles de leer. **Me parece bueno que invirtamos algo de tiempo leyendo las especificaciones juntos, de esa manera puedes ir ganando un poco de habilidad y ser capaz de leer la documentación por tu propia cuenta.**

código:

- <https://play.golang.org/p/KkV2YNuQit>

video: 083

## Structs Anónimos

También podemos crear structs anónimos. **Un struct anónimo es un struct el cual no es asociado con un identificador en específico.**

código:

- <https://play.golang.org/p/A9hS9FINAAAn>
- <https://play.golang.org/p/iFaBb4mGCug>

video: 084

## Aclaración

- Es todo acerca de tipo
  - [Es go un lenguaje orientado a objetos?](#) Go tiene aspectos de OOP. Pero todo es acerca de **TIPOS**. Creamos **TIPOS** en Go; **TIPOS** definidos por el usuario. Entonces podemos tener **VALORES** de ese tipo. Podemos asignar **VALORES** de un **TIPO** definido por el usuario a **VARIABLES**.
- Go es Orientado a Objetos
  1. **Encapsulación**
    - a. Estado ("campos")
    - b. Comportamiento ("métodos")
    - c. Exportado & No exportado; visible & no visible
  2. **Reusabilidad**
    - a. herencia ("tipos embebidos")
  3. **Polimorfismo**
    - a. interfaces
  4. **Overriding**
    - a. "promoción"
- OOP Tradicional
  1. Clases
    - a. Estructura de dato describiendo un tipo de objeto
    - b. Puedes crear "instancias"/ "objetos" de la clase / prototipo
    - c. Las clases almacenan ambos:
      - i. estado / datos / campos
      - ii. comportamiento / métodos

- d. público / privado
- 2. Herencia
- En Go:
  1. No creas clases, creas un **TIPO**
  2. No creas instancias, creas un **VALOR** de un **TIPO**
- Tipos definidos por el usuario
  - Podemos declarar un nuevo tipo
  - foo
    - El tipo subyacente de foo es int
    - Conversión a int
      - int(miEdad)
      - Convirtiendo tipo foo a tipo int
  - ES UNA MALA PRÁCTICA HACER ALIAS DE TIPOS
    - Excepción:
      - Si necesitas asignarle métodos a un tipo
      - Ver el paquete de tiempo para un ejemplo [godoc.org/time](https://godoc.org/time)
        - type Duration int64
        - Duration tiene métodos asignados
- Tipos Nombrados vs Tipos Anónimos
  - Tipos anónimos son indeterminados. Aún no han sido declarados como un tipo. El compilador tiene flexibilidad con tipos anónimos. Puedes asignar un tipo anónimo a una variable declarada de cierto tipo. Si la asignación puede ocurrir, el compilador hará el trabajo de determinar el tipo; el compilador hará una conversión implícita. No puedes asignar un tipo nombrado a un tipo de diferente nombre.
- Alineamiento arquitectónico y más
  - Convención: organiza tus campos lógicamente. Legibilidad y claridad ganan en rendimiento como punto crítico. Go será de buen rendimiento. Ve primero por legibilidad. Sin embargo, si estás en una situación donde necesitas darle prioridad al rendimiento: agrega los campos del más grande al de menor tamaño, por ejemplo: int 64, int32, float32, bool

código:

- <https://play.golang.org/p/NMzj9PDil2k>

video: 085

## Ejercicios - Ninja Nivel 5

### Ejercicio práctico #1

Crea tu propio tipo "persona" el cual tendrá un tipo subyacente tipo "struct" de manera que pueda almacenar la siguiente data:

- Nombre

- Apellido
- Sabores de helado favoritos

Crea dos VALORES de TIPO persona. Imprime los valores, usa range sobre los elementos en el slice el cual almacena los valores de helados favoritos.

**solución:**

- [https://play.golang.org/p/3\\_LXuPsT3Q3](https://play.golang.org/p/3_LXuPsT3Q3)

**video: 086**

## Ejercicio práctico #2

Usa el código del ejemplo anterior y almacena los valores de tipo persona en un mapa con la llave apellido. Accede a cada valor en el mapa. Imprime los valores. Imprime también los valores haciendo range sobre el slice.

**solution:** <https://play.golang.org/p/h5Uf4CwZt2S>

**video: 087**

## Ejercicio práctico #3

- Crea un nuevo tipo: vehículo.
  - El tipo subyacente es un struct.
  - Los campos:
    - puertas
    - color
- Crea dos nuevos tipos: camión & sedán.
  - El tipo subyacente de cada uno de esos tipo es un struct.
  - Embebe el tipo “vehículo” en ambos camión y sedán.
  - Dale al camión el campo “cuatroRuedas” el cual será un booleano.
  - Dale al sedán el campo “lujoso” el cual será un booleano.
- Con los structs vehículo, camión y sedán:
  - Usando un composite literal, crea un valor de tipo camión y asígnale valor a los campos.
  - Usando un composite literal, crea un valor de tipo sedan y asígnale valor a los campos.
- Imprime cada uno de los valores.
- Imprime un solo valor de cada uno de eso valores.

**solución:** <https://play.golang.org/p/GchZPj5QDdk>

**video: 088**

## Ejercicio práctico #4

Crea y usa un struct anónimo.

**solución:** <https://play.golang.org/p/mYvXb0nuc2S>

**video: 089**

# Funciones

## Sintaxis

**func (receptor) identificador(parámetros) (returns) { código }**

Conoce la diferencia entre parámetros y argumentos

- Definimos las funciones con **parámetros** (si lleva alguno)
- Llamamos las funciones y les pasamos **argumentos** (si lleva alguno)

Todo en Go es PASADO POR VALOR

Propósito de las funciones

- Abstraer código
- Reutilización del código

código:

- **Func básica**
  - <https://play.golang.org/p/ILdfeuxINU7>
- **Toma un argumento**
  - <https://play.golang.org/p/AVqqlc7D0TC>
- **return**
  - [https://play.golang.org/p/Ow\\_iVLh\\_osE](https://play.golang.org/p/Ow_iVLh_osE)
- **múltiples returns**
  - <https://play.golang.org/p/KExFn6S-5Lu>

video: 090

## Parámetros variados

Puedes crear **una func el cual toma un número ilimitado de argumentos**. Cuando haces esto, se conoce como “parámetro variado” o en inglés como “variadic parameter.” Cuando usas el operador que forma parte de los elementos de léxico “...T” para indicar un parámetro variado (donde “T” representa algún tipo).

código:

- <https://play.golang.org/p/HUMscA9uinH>

video: 091

## Desplegando un slice

Cuando tienes un slice de algún tipo, puedes **pasar cada uno de los valores individuales en un slice usando el operador “...”**

código:

- **Desplegando un de int**
  - <https://play.golang.org/p/cKeegZDy-PT>
- **Pasando ningún o más valores**
  - <https://play.golang.org/p/4wyHRJm4wX1>
- **El parámetro variado tiene que ser el parámetro final**
  - <https://play.golang.org/p/euQ8PDQ8RN>

video: 092

## Defer

- Una declaración "defer" (diferir) invoca a una función cuya **ejecución es diferida para el momento en el que la función donde está contenida retorna**, en cualquiera de los siguientes casos: o la función ejecuta una declaración de retorno, o llega al final de su cuerpo o porque la gorutina (goroutine) correspondiente está en pánico (panicking).

código: <https://play.golang.org/p/AYY3AN4LQ6>

video: 093

## Métodos

**Un método no es más que una FUNC adjuntada a un TIPO.** Cuando adjuntas una función a un tipo es un método de ese tipo. Adjuntas una función a un tipo con el RECEPTOR.

code: <https://play.golang.org/p/ku16LHWfoEZ>

video: 094

## Interfaces y polimorfismo

**En Go, los valores pueden ser de más de un tipo.** Una interfaz permite a un valor ser de más de un tipo. Creamos una interfaz usando la sintaxis : "palabra clave identificador tipo" entonces para una interfaz sería: "type humano interface" Luego definimos cuáles método(s) debe tener un tipo para implementar esa interfaz. Si un TIPO tiene los métodos requeridos, el cual podrían ser ninguno, (la interfaz vacía denotada por interface{}), entonces ese TIPO *implícitamente implementa* la interfaz y es **también** de ese tipo de interfaz. En Go, los valores pueden ser más de un tipo.

Código: <https://play.golang.org/p/x9LYdnRel7R>

video: 095.1

código: <https://play.golang.org/p/rZH2Efbpot>

Video: 095.2

## Funciones Anónimas

**Funciones anónimas auto-ejecutables**

código: <https://play.golang.org/p/fpDrtaP8yRB>

video: 096

## Expresión función

**Asignando una func a una variable**

código: <https://play.golang.org/p/IR0EuIUTw6h>

video: 097

## Retornando una func

**Puedes retornar una función desde una función.** Así es como se hace.

código:

- Retornando un string
  - <https://play.golang.org/p/hD9AdHBtFIM>
- Retornando una func
  - paso 1: <https://play.golang.org/p/lfTbQRy5NDS>
  - paso 2 - limpieza: <https://play.golang.org/p/l2ehL0R7aWy>
  - paso 3 - limpieza: <https://play.golang.org/p/FSjvOfY0wW>
  - paso 4 - limpieza: <https://play.golang.org/p/7wbv9KNlhK>
  - paso 5 - limpieza: <https://play.golang.org/p/vW0IGelAox>

video: 098

## Callback

- **Pasando una func como un argumento**
- La programación funcional no es algo que se recomienda hacer con Go, sin embargo, es bueno estar al tanto de los callbacks.
- idiomatic go: escribe código que sea claro, simple y legible.

código:

- preparación: <https://play.golang.org/p/NgFVEpZgJQW>
- Explicación:
  - Operadores matemáticos: [https://play.golang.org/p/sTDJ3l\\_rlj](https://play.golang.org/p/sTDJ3l_rlj)
  - Sólo la función suma: <https://play.golang.org/p/TEZChnAYlq>
  - Número pares: <https://play.golang.org/p/RKHjy9BI6j>
  - Número impares: [https://play.golang.org/p/Nf3\\_KrpidO](https://play.golang.org/p/Nf3_KrpidO)

video: 099

## Closure

- En un scope que encierra otros scopes
  - Las variables declaradas en el scope externo son accesibles en los scopes internos.
- **Los closures nos ayudan a limitar el scope de las variables**

código:

- scope de x: <https://play.golang.org/p/YWuniJtu2R>
- scope de x reducido a func main: <https://play.golang.org/p/4hqrzybcFc>
- Bloque de código en bloque de código con y: [https://play.golang.org/p/6Hyqe\\_aU-R](https://play.golang.org/p/6Hyqe_aU-R)
- Encerrando una variable en un bloque de código:  
<https://play.golang.org/p/fHez3lg8wc>

video: 100



## Recursividad

- **Una func que se llama a ella misma**
- Ejemplo factorial

### Código:

- factorial con recursividad: <https://play.golang.org/p/fd9nXrqEGi>
- factorial con ciclo: <https://play.golang.org/p/-GOTqkEUcY>

video: 101

## Ejercicios - Ninja Nivel 6

### Ejercicio práctico #1

- Revisión
  - funciones
  - Propósito de las funciones
    - Código abstracto
    - Reusabilidad de código
      - DRY - Don't Repeat Yourself
  - func, receptor, identificador, parámetros, retornos
  - parámetros vs argumentos
  - Variadic func
    - Múltiples parámetros "variadic"
    - Múltiples argumentos "variadic"
  - retornos
    - múltiples retornos
    - Retornos nombrados - yuck!
  - Expresiones func
    - Asignando una función a una variable
  - callbacks
    - Pasando una func a otra func como un argumento
  - closure
    - Un scope encerrando otro scope
    - Variables declaradas en el scope externo son accesibles en el scope interno
    - Los closure nos ayudan a limitar el scope de las variables
  - Recursividad
    - Una función que se llama a sí misma
    - factorial
- Ejercicio práctico
  - crea una func con el identificador foo que retorne un int
  - crea una func con el identificador bar que retorne un int y un string
  - Llama ambas funciones
  - Imprime sus resultados

código: <https://play.golang.org/p/8V8HiU2xAdg>

video: 102

## Ejercicio práctico #2

- Crea una función con el identificador foo que
  - Tome un parámetro variable de tipo int
  - Pásale un valor de tipo []int a la función (haz el despliegue del []int)
  - Retorna la suma de todos los valores de tipo int pasados como argumento.
- Crea una func con el identificador bar que
  - Tome un parámetro de tipo []int
  - Retorne la suma de todos los valores de tipo int pasados.

código: <https://play.golang.org/p/B0yRxtBQPD>

video: 103

## Ejercicio práctico #3

Usa la palabra clave “defer” para mostrar que una función diferida corre después que la función que la contiene finaliza o retorna.

code: <https://play.golang.org/p/XluEuUD0Nw>

video: 104

## Ejercicio práctico #4

- Crea un struct con
  - El identificador “persona”
  - Los campos:
    - Nombre
    - Apellido
    - Edad
- Adjunta un método al tipo persona con
  - El identificador “presentar”
  - El método debe hacer que el tipo persona diga su nombre y edad
- Crea un valor de tipo persona
- Llama al método usando el valor tipo persona

código: <https://play.golang.org/p/NnXyWdqbbw>

video: 105

## Ejercicio práctico #5

- Crea un tipo **CUADRADO**
- Crea un tipo **CÍRCULO**
- Adjunta un método a cada uno que calcule y retorne el **ÁREA**
  - Área de un círculo=  $\pi r^2$
  - Área de un cuadrado =  $L * H$

- Crea un tipo **FORMA** que defina una interfaz como cualquier cosa que tenga el método **ÁREA**
- Crea una func **INFO** el cuál toma un tipo forma e imprime el área de la misma.
- Crea un valor de tipo cuadrado
- Crea un valor de tipo círculo
- Usa la func info para imprimir el área de un cuadrado
- Usa la func info para imprimir el área de un círculo

código: <https://play.golang.org/p/e2uPK-HmKnB>

video: 106

## Ejercicio práctico #6

- Crea y usa una func anónima

código: [https://play.golang.org/p/S\\_IMj3gQk0e](https://play.golang.org/p/S_IMj3gQk0e)

video: 107

## Ejercicio práctico #7

- Asigna una función a una variable, luego llama esa función

código: <https://play.golang.org/p/ZmzsoU3tkoS>

video: 108

## Ejercicio práctico #8

- Crea una func el cual retorna una func
- Asigna la func retornada a una variable
- Llama la func retornada

código: <https://play.golang.org/p/c2HwqVE1Rd>

video: 109

## Ejercicio práctico #9

Un “callback” es cuando le pasamos una función a otra función como argumento. Para este ejercicio,

- Pasa una func a otra función como argumento

código: <https://play.golang.org/p/0yGYPKh1y7>

video: 110

## Ejercicio práctico #10

Closure es cuando “encerramos” el scope de una variable en un bloque de código. Para este ejercicio, crea una func el cual “encierra” el scope de una variable:

código: <https://play.golang.org/p/a56uWtoFSL>

video: 111

## Ejercicio práctico #11

La mejor forma de aprender es enseñar. Para este ejercicio,

- Escoge uno de los ejercicios realizados en esta sección, o usa el ejemplo de recursividad hecho con factorial
- descarga, instala, y corre el programa
  - <https://obsproject.com/>
- Graba un video donde APAREZCAS enseñando el tema
- Sube el video a youtube
- Comparte el video en twitter y etiquetame en él ( <https://twitter.com/eduartua> ) de manera que pueda verlo!

video: 112

## Pointers

### Que son punteros (pointers)?

Todos los valores son almacenados en memoria. Cada ubicación en memoria tiene una dirección. **Un pointer es una dirección de memoria.**

```
&
*int
*
```

código

- Código del video: <https://play.golang.org/p/Ysv5Adn3V1>
- paso 1 - toma una dirección & : <https://play.golang.org/p/gWjNjPQl1Ga>
- paso 2 - dereference \* : <https://play.golang.org/p/V42XtkXCfC3>
- paso 3 - dereference \* : <https://play.golang.org/p/xKU-Vw5enbo>

video: 113

### Cuando usar pointers

Los pointers te permiten compartir un valor almacenado en alguna ubicación de la memoria.

Usa pointers cuando

1. No quieres pasar una cantidad grande de datos
2. Quieres cambiar los datos en esa ubicación

**Todo en Go es pasado por valor.** Olvida cualquier concepto o frase puedas traer de otros lenguajes. Pasar por referencia, pasar por copia - olvida esas frases. "Pasar por valor." Ésa es la única frase que deberías saber y recordar. Es la única frase que deberías usar. Pasar por valor. Todo en Go es pasado por valor. En Go, lo que ves es lo que obtienes - what you see is what you get (wysiwyg). Mira lo que está ocurriendo. Éso es lo que obtienes..

código:

- paso 1 sin pointer: <https://play.golang.org/p/lxsWkhTaYv>
- paso 2 pointer: <https://play.golang.org/p/Xul19kjFmb>

video: 114

## Method Sets

El Method Sets determinan cuáles métodos adjuntar a un TIPO. Es exactamente lo que el nombre dice: ¿Cuál es la colección de métodos de un tipo dado? Es su colección de métodos.

**IMPORTANTE: “el method set de un tipo determina las INTERFACES que el tipo implementa.....”**

~ golang spec

El “importante” de arriba no fué mencionado en este video pero será discutido en la sección de “concurrency” en un video llamado “revisión de method sets”.

- Un RECEPTOR NO-POINTER
  - Funciona con valores que son POINTERS o NO-POINTERS.
- Un RECEPTOR POINTER
  - Sólo funciona con valores que son POINTERS.

Receptores      Valores

-----  
(t T)      T y \*T  
(t \*T)      \*T

código:

- RECEPTOR NO-POINTER Y VALOR NO-POINTER
  - <https://play.golang.org/p/6kWpBxX-7M0>
- RECEPTOR NO-POINTER Y VALOR POINTER
  - <https://play.golang.org/p/VkgUCslxkXo>
- RECEPTOR POINTER Y VALOR POINTER
  - <https://play.golang.org/p/5wTCiEhPBne>
- RECEPTOR POINTER Y VALOR NO-POINTER
  - <https://play.golang.org/p/IM7mOR7-QEZ> ( este código no corre )
  - Este código no corre - nota la diferencia - el method set determina las INTERFACES que el tipo implementa
    - <https://play.golang.org/p/KK8qjsAWBZ>

video: 115

## Ejercicios - Ninja Nivel 7

### Ejercicio Práctico #1

- Crea un valor y asígnalo a una variable.

- Imprime la dirección de ese valor.

code: <https://play.golang.org/p/57kW8xd0qT>

video: 116

## Ejercicio Práctico #2

- Crea un struct persona
- Crea una función llamada “cambiame” con un \*persona como parámetro
  - Cambia el valor almacenado en la dirección de memoria del \*persona
    - **Importante**
      - Para desreferenciar un struct, usa (\*valor).campo
        - p1.nombre
        - (\*p1).nombre
      - “Como una excepción, si el tipo de x es un tipo puntero con nombre y (\*x).c es una expresión válida de selección denotando un campo (pero no un método), x.c es una forma abreviada de (\*x).c”.
        - <https://golang.org/ref/spec#Selectors>
- Crea un valor de tipo persona
  - Imprime el valor
- En func main
  - llama “cambiame”
- En func main
  - Imprime el valor

código: [https://play.golang.org/p/C\\_ZxTKCek33](https://play.golang.org/p/C_ZxTKCek33)

video: 117

# Aplicación

## Documentación JSON

Entendemos pointers; entendemos métodos. Ahora tenemos el conocimiento suficiente para comenzar a usar la biblioteca estándar. Este video te dará **orientará en cómo hago para abordar, leer y trabajar con la biblioteca estándar.**

video: 118

## JSON marshal

Aquí vamos a ver un ejemplo de cómo puedes hacer **marshal de datos en Go a JSON.**

También, este video muestra como en el caso de un identificador - minúsculas o mayúsculas, determina si los datos pueden o no ser exportados.

código:

- <https://play.golang.org/p/D-VjrZiHCSB>

video: 119

## JSON unmarshal

**Podemos tomar un JSON y regresarlo a nuestro programa en Go haciendo unmarshalling a ese JSON.** Aquí les dejo excelentes recursos para entender y trabajar JSON.

Buenos sitios web:

- <http://rawgit.com/>
- <https://mholt.github.io/json-to-go/>
- <https://github.com/goestoeleven>

código:

- [Entendiendo JSON página rawgit HTML](#)
- <https://play.golang.org/p/YubQ5h2zvFI>

video: 120

## Interfaz Writer

Entendiendo **la interfaz writer del paquete io**. También, una última nota acerca de trabajar con JSON: codifica y decodifica.

código: <https://play.golang.org/p/3Txh-dKQBf>

video: 121

## Sort

**El paquete sort nos permite ordenar slices.**

código:

- Código inicial:
  - <https://play.golang.org/p/iglGnMv6AN>
- Ordenado
  - <https://play.golang.org/p/8UkvEdzQOk>

video: 122

## Sort Personalizado

Aquí vemos cómo **ordenamos los campos de un struct**.

código:

- Código inicial:
  - [https://play.golang.org/p/7ppe\\_VewcOm](https://play.golang.org/p/7ppe_VewcOm)
- Ordenados
  - Por edad: <https://play.golang.org/p/wByliMnvbj1>
  - Por nombre: <https://play.golang.org/p/6qKrlQ0Bpqr>

video: 123

## bcrypt

Muy seguido, actualmente escuchamos acerca de bases de datos que han sido hackeadas y los datos de los usuarios comprometidos. No hay excusas para esto. Nosotros, como programadores, tenemos las herramientas para proteger la información de los usuarios. Bcrypt es una de las herramientas que puedes usar para proteger los datos de los usuarios. **Usando bcrypt, obtendremos una mayor comprensión de cómo leer e implementar código de la biblioteca estándar.**

código:

- <https://goo.gl/ZVKnRx>

video: 124

## Ejercicios - Ninja Nivel 8

### Ejercicio Práctico #1

Comenzando [con este código](#), marshal el slice []usuario a JSON. Hay una pequeña curva en este ejercicio - recuerda preguntarte qué necesitas para EXPORTAR un valor de un paquete.

**solución:** [https://play.golang.org/p/TXB\\_7dz0YQL](https://play.golang.org/p/TXB_7dz0YQL)

video: 125

### Ejercicio Práctico #2

Comenzando [con este código](#), unmarshal el JSON a una estructura de datos de Go. Pista:

<https://mholt.github.io/json-to-go/>

código:

- Preparación de código (para tu info, no es necesario para este ejercicio):
  - <https://play.golang.org/p/41uUpeVyZbJ>
- solution:
  - <https://play.golang.org/p/rhhkGOZ7eRE>

video: 126

### Ejercicio Práctico #3

Comenzando [con este código](#), codifica a JSON el []usuario enviando el resultado a Stdout.

Pista: Necesitarás usar `json.NewEncoder(os.Stdout).encode(v interface{})`

**solución:** <https://play.golang.org/p/Mqvso2cX-te>

- Definición de Stdout:
  - [Standard Output Definition](#)

video: 127



## Ejercicio Práctico #4

Comenzando [con este código](#), ordena el []int y []string.

**solución:** <https://play.golang.org/p/0wSkyEkUU0E>

**video:** 128

## Ejercicio Práctico #5

Comenzando [con este código](#), ordena el []usuario por

- Edad
- Apellido

También ordena el []string "Dichos" para cada usuario

- Imprime todo de una manera agradable

**solución:** <https://play.golang.org/p/KSp6RtCYczv>

**video:** 129

# Concurrencia

## Concurrencia vs paralelismo

Aunque cuando la gente escucha la palabra *concurrencia* frecuentemente piensan en *paralelismo*, un concepto relacionado pero que realmente son diferentes. En programación, concurrencia es la *composición* de independientemente ejecutar procesos, mientras que paralelismo es la *ejecución* simultánea de (posiblemente relacionados) procesos computacionales. La concurrencia se trata de lidiar con muchas cosas a la vez. El paralelismo se trata de hacer muchas cosas a la vez.

**video:** 130

## WaitGroup

**Un WaitGroup espera que una colección de gorutinas terminen su trabajo.** La gorutina de main llama Add para configurar el número de gorutinas por las que tiene que esperar. Luego cada una de las gorutinas corre y llama a Done cuando terminan. Al mismo tiempo, Wait puede ser usado para bloquear hasta que todas las gorutinas hayan finalizado. Escribir código concurrente es súper fácil: todo lo que tenemos que hacer es **poner "go" en frente de una llamada a una función o método.**

- runtime.NumCPU()
- runtime.NumGoroutine()
- sync.WaitGroup
  - func (wg \*WaitGroup) Add(delta int)
  - func (wg \*WaitGroup) Done()
  - func (wg \*WaitGroup) Wait()

**Imagen de race conditions:**

- <https://goo.gl/JSC7M2>

código:

- Código inicial:
  - <https://play.golang.org/p/bnI0akWF9f>
- Finalizado:
  - <https://play.golang.org/p/VDioqpZ65h>

video: 131

## Revisión de Method sets

**“El method set de un tipo determina las INTERFACES que ese tipo implementa.....”** ~ [golang spec](#)

Receivers	Values
-----	
(t T)	T and *T
(t *T)	*T

- Si tienes un \*T, puedes llamar a los métodos que tienen un receptor tipo \*T, así como a los métodos que tienen un receptor tipo T.
- Si tienes un T y es direccionable ([addressable](#)), puedes llamar a métodos que tienen un receptor de tipo \*T así como a métodos que tienen un receptor de tipo T, porque la llamada de método t.Meth() será equivalente a (& t). Meth() ([Llamadas](#)).
- Si tiene un T y no es direccionable, solo puede llamar a métodos que tienen un tipo de receptor de T, no \* T.
- Si tiene una interfaz I, y algunos o todos los métodos en el conjunto de métodos de I son proporcionados por métodos con un receptor de \*T (y el resto es proporcionado por métodos con un receptor de T), entonces \*T satisface la interfaz I, pero T no. Esto se debe a que el conjunto de métodos de \*T incluye T, pero no al revés (vuelve al primer punto nuevamente).

En resumen, puedes mezclar y emparejar métodos con receptores de valores de tipo T y métodos con receptores de punteros, y usarlos con variables que contengan valores y punteros, sin preocuparse por cuál es cuál. Ambos funcionarán, y la sintaxis es la misma. Sin embargo, si se necesitan métodos con receptores de tipo puntero para satisfacer una interfaz, sólo un puntero será asignable a la interfaz; un valor no será válido.

code: <https://play.golang.org/p/IM7mOR7-QEZ>

video: 132

## Documentación

Son llamadas goroutines porque los términos existentes —threads, coroutines, procesos, entre otros— transmiten connotaciones inexactas. Una goroutine tiene un modelo simple: es una función ejecutándose concurrentemente con otras goroutines en el mismo espacio de memoria (address space).

código:

- <https://play.golang.org/p/IBKFKCwrue>

video: 133

## Condición Race

Aquí tenemos una [imagen](#) de la race condition que vamos a recrear. **Race conditions no son buen código. Una race condition te dará resultados impredecibles. Veremos como arreglar esta race condition en el próximo video.**

código: <https://play.golang.org/p/a-tdD-7ITld>

video: 134

## Mutex

Un "mutex" es un bloqueo de exclusión mutua. **Los mutexes nos permiten bloquear nuestro código para que solo una gorutina pueda acceder a ese bloque de código bloqueado a la vez.**

código: [https://github.com/GoesToEleven/golang-web-dev/tree/master/000\\_temp/52-race-condition](https://github.com/GoesToEleven/golang-web-dev/tree/master/000_temp/52-race-condition)

video: 135

## Atomic

Podemos **usar el paquete atomic para también prevenir una race condición** en nuestro código incremental.

código: [https://github.com/GoesToEleven/golang-web-dev/tree/master/000\\_temp/52-race-condition](https://github.com/GoesToEleven/golang-web-dev/tree/master/000_temp/52-race-condition)

video: 136

# Tu entorno de desarrollo

## La terminal

- Terminología
  - GUI = graphical user interface
  - Línea de comandos
    - CLI = command line interface
  - terminal = entorno para entrada/salida de texto
    - console = terminal física
  - **shell / bash**
    - unix / linux / mac
  - **command prompt** / windows command / cmd / dos prompt
    - windows

- Comandos shell / bash
  - pwd
  - which -> muestra si está instalado el software (binario)
  - ls
  - ls -la
    - Permisos
      - owner(propietario), group(grupo), world(resto del mundo)
      - R (read - lectura), w (write - escritura), x (execution - ejecución)
      - 4, 2, 1

d = directorio

rw-rw-rw = owner, group, world

owner(dueño) group(grupo) bytes última modificación ocultos & nombre

```
total 72
drwxr-xr-x+ 20 spockmcleod staff 680 Jul 31 15:44 .
drwxr-xr-x 5 root admin 170 Dec 30 2016 ..
-r----- 1 spockmcleod staff 7 Dec 30 2016 .CFUserTextEncoding
-rw-r--r--@ 1 spockmcleod staff 14340 Aug 1 06:52 .DS_Store
drwx----- 19 spockmcleod staff 646 Aug 1 07:07 .Trash
drwxr-xr-x 14 spockmcleod staff 476 Jul 26 13:56 .atom
-rw----- 1 spockmcleod staff 10321 Aug 1 07:16 .bash_history
drwx----- 41 spockmcleod staff 1394 Aug 1 07:16 .bash_sessions
drwx----- 3 spockmcleod staff 102 Aug 15 2016 .cups
-rw----- 1 spockmcleod staff 1024 Dec 30 2016 .rnd
drwx----- 4 spockmcleod staff 136 Feb 15 14:48 Applications
drwx-----+ 4 spockmcleod staff 136 Jul 26 13:16 Desktop
drwx-----+ 9 spockmcleod staff 306 Jul 31 16:43 Documents
drwx-----+ 4 spockmcleod staff 136 Jul 31 10:12 Downloads
```

- touch
  - touch temp.txt
- Limpiar pantalla
  - Ctrl + l ó command + k
  - clear
- chmod
  - chmod options permisos de los archivos
  - chmod 777 temp.txt
- cd
- cd ../
- env
- rm <file or folder name>
  - rm -rf <file or folder name>
- .bash\_profile & .bashrc
  - **.bash\_profile** es ejecutado para login shells, mientras que **.bashrc** es ejecutado para no-login shells interactivos. Cuando haces login (tecleas username y password) via consola, ya sea directamente en la máquina o manera remota vía ssh: .bash\_profile es ejecutado para configurar tu

shell antes de cualquier comando inicial. Si iniciamos otra terminal ya logueados entonces se configura con .bashrc.

- nano <file name>
- cat <file name>
- grep
  - cat temp2.txt | grep enter
  - ls | grep -i documents
- [all commands](#)

video: 137

## Bash en Windows

- linux en Windows
  - Características de desarrollador
    - Linux subsystem for Windows (beta)
  - bash
  - [Artículos con pasos para instalar subsistema linux](#)
- <https://git-scm.com/>

video: 138

## Instalando Go

- descargar go
- Comandos go
  - go version
  - go env
  - go help
- checksum

video: 139

## El Go workspace

- Una carpeta - cualquier nombre, cualquier ubicación
  - bin
  - pkg
  - src
    - github.com
      - <github.com username>
        - Carpeta con código de proyecto / repo
        - Carpeta con código de proyecto / repo
        - Carpeta con código de proyecto / repo
        - Carpeta con código de proyecto / repo
        - ...
        - Carpeta con código de proyecto / repo
- namespacing

- **go get**
  - Gestión de paquetes
- GOPATH
  - Apunta a tu workspace
- GOROOT
  - Apunta a tu instalación de binarios de Go

video: 140

## Variables de entorno

- Las variables de entorno son un conjunto de valores dinámicos que pueden afectar la manera en que se comportan los procesos que están corriendo en una computadora. Forman parte del entorno en el que los procesos corren.

```
# Go programming
export GOPATH="/Users/toddmcleod/go"
export PATH="$PATH:/Users/toddmcleod/go/bin"
```

```
Todds-MacBook-Pro:~ toddmcleod$ go env
GOARCH="amd64"
GOBIN=""
GOEXE=""
GOHOSTARCH="amd64"
GOHOSTOS="darwin"
GOOS="darwin"
GOPATH="/Users/toddmcleod/go"
GORACE=""
GOROOT="/usr/local/go"
GOTOOLDIR="/usr/local/go/pkg/tool/darwin_amd64"
GCCGO="gccgo"
CC="clang"
GOGCCFLAGS="-fPIC -m64 -pthread -fno-caret-diagnostics -Qunused-arguments -fmessage-length=0 -gno-record-gcc-switches -fno-common"
CXX="clang++"
CGO_ENABLED="1"
PKG_CONFIG="pkg-config"
CGO_CFLAGS="-g -O2"
CGO_CPPFLAGS=""
CGO_CXXFLAGS="-g -O2"
CGO_FFLAGS="-g -O2"
CGO_LDFLAGS="-g -O2"
Todds-MacBook-Pro:~ toddmcleod$ echo $PATH
/Users/toddmcleod/mongodb/bin:/Users/toddmcleod/google-cloud-sdk/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/go/bin:/Users/toddmcleod/go/bin
Todds-MacBook-Pro:~ toddmcleod$
```

video: 141

## IDE's y Editores de Texto

- VS Code
- Goland
- Atom.io
- Sublime
- Vim

- Emacs

## Gogland

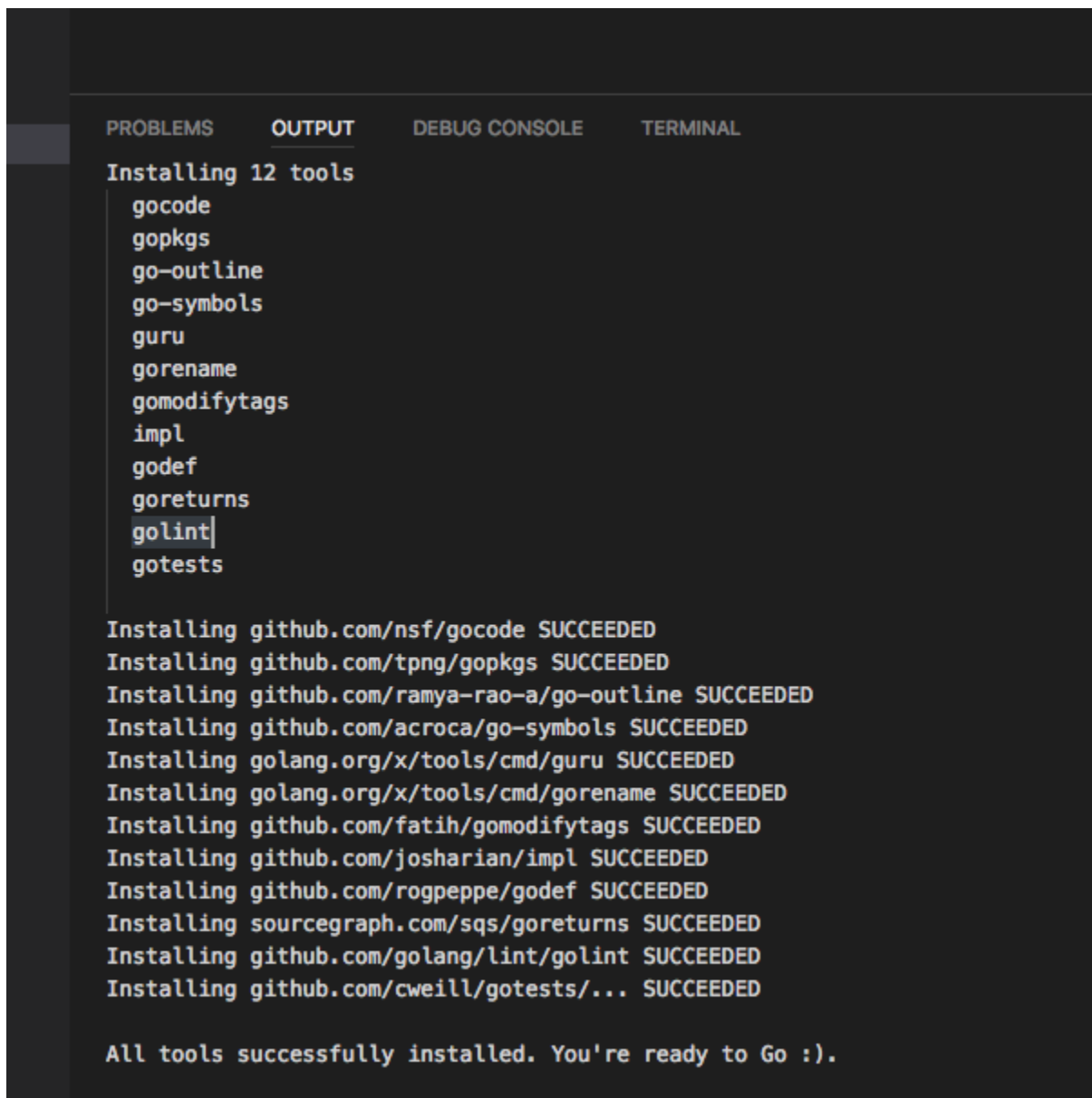
<https://www.jetbrains.com/go/>

- **go get**
  - `go get -d github.com/eduardtua/go-programming`

## VS CODE

<https://code.visualstudio.com/>

`go get -v github.com/nsf/gocode`



The screenshot shows a VS Code terminal window with the following content:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Installing 12 tools
gocode
gopkgs
go-outline
go-symbols
guru
gorename
gomodifytags
impl
godef
goreturns
golint
gotests

Installing github.com/nsf/gocode SUCCEEDED
Installing github.com/tpng/gopkgs SUCCEEDED
Installing github.com/ramya-rao-a/go-outline SUCCEEDED
Installing github.com/acroca/go-symbols SUCCEEDED
Installing golang.org/x/tools/cmd/guru SUCCEEDED
Installing golang.org/x/tools/cmd/gorename SUCCEEDED
Installing github.com/fatih/gomodifytags SUCCEEDED
Installing github.com/josharian/impl SUCCEEDED
Installing github.com/rogpeppe/godef SUCCEEDED
Installing sourcegraph.com/sqs/goreturns SUCCEEDED
Installing github.com/golang/lint/golint SUCCEEDED
Installing github.com/cweill/gotests/... SUCCEEDED

All tools successfully installed. You're ready to Go :).
```

video: 142

## Comandos Go

- **go version**
- **go env**
- **go help**
- **go fmt**
  - **./...**
- **go run**
  - Necesita el nombre del archivo, ejemplo: go run main.go
  - go run <nombre de archivo>
  - go run \*.go
- **go build**
  - Para un solo paquete main :
    - Construye el archivo
    - Si hay algún error, lo reporta
    - Si no hay errores, coloca un ejecutable en el directorio
  - Para múltiples paquetes o un archivo no main:
    - Crea el archivo
    - Si hay algún error, lo reporta
    - Luego se desecha el archivo binario
- **go install**
  - Para un solo ejecutable:
    - Compila el programa (lo crea)
    - Nombra el ejecutable como el directorio que contiene el código.
    - Coloca el ejecutable en **workspace / bin**
      - \$GOPATH / bin
  - Para paquetes:
    - Compila el paquete (lo crea)
    - Coloca el ejecutable en **workspace / pkg**
      - \$GOPATH / pkg
    - Crea un archivo tipo fichero(archive file) con extensión .a
- **flags**
  - **-race**

video: 143

## Repos Git

- Creando un repositorio
  - Crea un repo en gitlab.com
  - Crea un directorio con el mismo nombre en tu computadora
    - Colócalo en tu workspace de go
      - gitlab.com
        - Tu nombre de usuario
          - Nombre del repo
  - Corre “git init” en este directorio



- Este comando crea un repositorio git en este directorio
- Agrega un archivo a esta carpeta
  - “.gitignore” es una buena opción para agregar
- Comienza a rastrear este archivo
  - git add --all
- commit los cambios
  - git commit -m “Algún mensaje en el commit”
- Sigue las instrucciones, cuando creas tu repo, para conectar tu repo EN TU COMPUTADORA con GITLAB - reemplazando el nombre por el de tu repo:
  - git remote add origin git@gitlab.com:eduar/nombrederepo.git
  - git push -u origin master
- Commandos git
  - git status
  - git add --all
  - git commit -m “some message”
  - git push

**video: 144**

## Explorando Gitlab

- Clonando un repo
  - git clone <repo>
- ssh
  - Linux / mac
    - ssh-keygen -t rsa
      - id\_rsa
        - Tu llave privada es guardada como id\_rsa file en el directorio .ssh y se usa para verificar la llave pública que usas
      - id\_rsa.pub
        - Esta es tu llave pública. Puedes compartirla con el mundo.
- git remote
  - Ver repo remoto

**video: 145**

## Compilación cruzada

- GOOS & GOARCH
  - <http://godoc.org/runtime#pkg-constants>
- GOOS=linux GOARCH=amd64 go build test.go

**video: 146**

## Paquetes

- Una carpeta, varios archivos

- Declaración de paquete en cada archivo.
- Scope de paquete
  - Las cosas de un archivo pueden ser accesibles desde otro archivo
- Los imports tienen scope de archivo
- Exportado / No exportado
  - También, visible / no visible
  - No decimos (generalmente hablando): público / privado
  - Usar mayúsculas
    - Escribir con mayúsculas: exportado, visible fuera del paquete
    - Minúsculas: no exportado, no visible fuera del paquete.

video: 147

## Ejercicios - Ninja Nivel 9

### Ejercicio práctico #1

- Además de la gorutina principal, lanza dos gorutinas adicionales
  - Cada gorutina debe imprimir algo en pantalla
- Usa waitgroups para asegurarte que cada gorutina finalice antes de que el programa termine

código: <https://gitlab.com/eduar/go-programming>

video: 148

### Ejercicio práctico #2

Este ejercicio te ayudará a reforzar el concepto de method sets:

- Crea un tipo struct persona
- Adjunta el método hablar usando un receptor de tipo puntero
  - \*persona
- Crea un tipo interfaz humano
  - Para implícitamente implementar esa interfaz, un tipo humano debe tener el método hablar
- Crea la función "diAlgo"
  - Haz que tome un humano como parámetro
  - Haz que llame al método hablar
- Muestra lo siguiente en tu código
  - PUEDES pasar un valor de tipo \*persona a diAlgo
  - NO puedes pasar un valor de tipo persona a diAlgo
- Aquí hay una pista si necesitas ayuda
  - [https://play.golang.org/p/JQd6LsU\\_L-K](https://play.golang.org/p/JQd6LsU_L-K)

Receptores      Valores

-----  
(t T)      T y \*T  
(t \*T)     \*T

código: <https://gitlab.com/eduar/go-programming>

video: 149

## Ejercicio práctico #3

- Usando gorutinas, crea un programa incremento
  - Haz que una variable guarde el valor del incremento
  - Lanza varias gorutinas
    - cada gorutina deberá
      - Leer el valor del incremento
        - Almacenarlo en una nueva variable
      - Ceder el procesador con runtime.Gosched()
      - Incrementa la nueva variable
      - Escribe el valor en la nueva variable de vuelta a la variable incremento
- Usa waitgroups para esperar que todas las gorutinas finalicen
- Lo anterior generará una race condition.
- Comprueba que es una race condition usando el flag -race
- Si necesitas ayuda, aquí tienes una pista: <https://play.golang.org/p/a-tdD-7ITld>

código: <https://gitlab.com/eduar/go-programming>

video: 150

## Ejercicio práctico #4

Arregla la race condition que creaste en el ejercicio anterior usando un mutex

- Tiene sentido eliminar runtime.Gosched()

código: <https://gitlab.com/eduar/go-programming>

video: 151

## Ejercicio práctico #5

Arreglar la race condition que creaste en el ejercicio #4 usando el paquete atomic

código: <https://gitlab.com/eduar/go-programming>

video: 152

## Ejercicio práctico #6

Crea un programa que imprima tu OS y ARCH. Usa los siguientes comandos para correrlo

- go run
- go build
- go install

código: <https://gitlab.com/eduar/go-programming>

video: 153

## Ejercicio práctico #7

Descarga [OBS](#). Haz una grabación enseñando algún tema del lenguaje de programación Go.

Algunos de los temas que puedes enseñar:

- [¿Porqué Go?](#)
- Instalando go
- GOROOT & GOPATH
  - Variables de entorno
- Hola mundo
- Comandos de go
  - help
- variables
  - Operador de declaración corta
- Constantes
- Ciclos
  - init, cond, post
  - break
  - continue
- Funciones
  - func (receptor) identificador(params) (retornos) { código }
- Métodos
- Interfaces
- Method sets
- Tipo
  - Conversión
- Concurrencia vs paralelismo
- Gorutinas
- waitgroups
- mutex
- atomic

Después de grabar, sube el video a youtube. Luego tweet el enlace de tu video y [etiquetame en el tweet](#) de manera que pueda verlo.

código: <https://gitlab.com/eduar/go-programming>

video: 154

## Channels (Canales)

### Entendiendo canales (channels)

Introducción a Channels

- Creando un canal  
c := make(chan int)
- Colocando un valor en un canal  
c <- 42
- Tomando un valor de un canal  
<-c
- buffered channels  
c := make(chan int, 4)
- Los channels bloquean
  - Son como corredores en una carrera de relevo
    - Están sincronizados
    - Tienen que pasar/recibir el valor al mismo tiempo como los corredores en la carrera de relevo tienen que pasar / recibir testigo uno al otro al mismo tiempo.
      - Un corredor no puede pasar el bastón en un momento dado
      - y luego, un tiempo después, haga que el otro corredor reciba el testigo
      - El testigo es pasado / recibido por los corredores al mismo tiempo
  - El valor es pasado / recibido sincronizadamente; al mismo tiempo.
- Los canales nos permiten pasar valores entre gorutinas.

código:

- **No funciona**
  - <https://play.golang.org/p/XPgsj2xS0F>
  - **IMPORTANTE: LOS CANALES BLOQUEAN**
    - Los canales permiten
      - coordinación / sincronización / orquestación
      - buffering (buffered channels)
- **Enviar y recibir**
  - <https://play.golang.org/p/SHr3lpX4so>
- **buffer**
  - <https://play.golang.org/p/hsttb2qEJi>
    - “La capacidad, en número de elementos, establece el tamaño del buffer en el canal. Si la capacidad es cero o no se especifica, el canal es sin búfer (unbuffered) y la comunicación es exitosa solamente cuando ambos, el transmisor y el receptor están listos.” [Golang Spec](#)
- **buffer no funciona**
  - <https://play.golang.org/p/epLsvcivJS>
- **buffer**
  - [https://play.golang.org/p/\\_6bSI5fc17](https://play.golang.org/p/_6bSI5fc17)

código: <https://gitlab.com/eduar/go-programming>

video: 155

## Canales direccionales

Tipo de canal. Lee de izquierda a derecha.

código:

- **Viendo el tipo**
  - Código del video anterior
    - <https://play.golang.org/p/a98otBr4eX>
  - Enviando y recibiendo (bidireccional)
    - <https://play.golang.org/p/TD7DXPWTrFw>
    - **“Enviar y recibir” significa “enviar y recibir”**
      - <https://play.golang.org/p/SHr3lpX4so>
        - Ya visto en el código arriba
    - **Enviar significa enviar**
      - **error:** “operación inválida: <-cs (recibir desde un canal send-only chan<-int)”
        - <https://play.golang.org/p/oB-p3KMiH6>
    - **Recibir significa recibir**
      - **error:** “operación inválida: cr <- 42 (enviar a un canal receive-only tipo <-chan int)”
        - <https://play.golang.org/p/DBRuelmEq>
    - “Un canal puede estar limitado solo para enviar o solo para recibir [general a específico] por la conversión o asignación.” [Golang Spec](#)
      - No asigna
        - específico a general
          - <https://play.golang.org/p/mjUuk9IKGgq>
        - específico a específico
          - <https://play.golang.org/p/8JkOnEi7-a>
      - Asigna
        - general a específico
          - <https://play.golang.org/p/-dBJf3TJtoL>
      - Conversión
        - general a específico funciona
          - <https://play.golang.org/p/tlPJDaqDNt0>
        - Específico a general no funciona
          - <https://play.golang.org/p/4sOKuQRHq7>

código: <https://gitlab.com/eduar/go-programming>

video: 156

## Usando canales

- Crear un canal general
- En funciones puedes especificar
  - Canal receive-only

- Puedes recibir valores del canal
- Un parámetro canal receptor
- En la función, puedes solamente extraer valores del canal
- No puedes cerrar un canal receptor
- Canal send-only
  - Puedes enviar valores al canal
  - No puedes recibir/extraer/leer desde el canal
  - Solamente puedes enviar valores al canal

código:

- <https://play.golang.org/p/Ry1MuWipmX8>
- <https://gitlab.com/eduar/go-programming>

video: 157

## Range

Range deja de leer desde un canal cuando el canal es cerrado.

código:

- Cerrando un canal
  - [https://play.golang.org/p/b29\\_j-QgSa](https://play.golang.org/p/b29_j-QgSa)
- range sobre un canal
  - <https://play.golang.org/p/VAObmtWcPlq>

código: <https://gitlab.com/eduar/go-programming>

video: 158

## Select

La declaración select extrae el valor de cualquier canal que tenga un valor listo para ser extraído.

código:

- Trabajando con el código anterior
  - <https://play.golang.org/p/E3Dzy95wUQy>
- Sin cerrar los canales par impar
  - <https://play.golang.org/p/hkcf-bpOS1Y>

código: <https://gitlab.com/eduar/go-programming>

video: 159

## Idioma Coma ok

El idioma coma ok con select.

código:

- Cerrando el canal salir y el idioma coma ok
  - <https://play.golang.org/p/2fqKUziopJz>
  - con bool
    - <https://play.golang.org/p/mcQksC7pHiQ>

- Con int
  - [https://play.golang.org/p/sK1f\\_oGfeSu](https://play.golang.org/p/sK1f_oGfeSu)
- Sólo el idioma coma ok
  - <https://play.golang.org/p/6LPzCtZeT3>
  - <https://play.golang.org/p/dToDc0zJhZ>
- Limpieza del código arriba - idioma coma ok
  - paso 1 - código del idioma coma ok reducido
    - <https://play.golang.org/p/-sTOZRwFaa9>
  - step 2 - eliminar los underscores - desecho de variable
    - <https://play.golang.org/p/1dwld-iwauQ>
  - step 3 - cambiar salir de bool a int
    - <https://play.golang.org/p/0a62VkopvVf>
- Select para recibir
  - <https://play.golang.org/p/phXtlWpbrUe>
- Select para enviar
  - <https://play.golang.org/p/SKMulIHTKyN>
- Interesante
  - No corre
    - [https://play.golang.org/p/sBJV\\_HAFIA\\_O](https://play.golang.org/p/sBJV_HAFIA_O)
  - Corre
    - <https://play.golang.org/p/MvVhDtZs7cs>
  - Corre de otra forma - Este me gusta más
    - El canal salir fue eliminado
    - Select fue eliminado
      - Haz range sobre el canal usado
    - <https://play.golang.org/p/b9ylmbB7IGt>

video: 160

## Fan in

Tomando valores de varios canales y colocándolos en un canal.

código:

- Código de Eduar
  - [https://play.golang.org/p/q\\_ebfcGii8q](https://play.golang.org/p/q_ebfcGii8q)
- Rob Pike's code
  - <https://play.golang.org/p/buy30qw5MM>

video: 161

## Fan out

Consiste en tomar algún trabajo a realizar y convertirlo en varias porciones de trabajo con varias gorutinas.

código:

- fan out in
  - <https://play.golang.org/p/mK0CIR2CZqi>



- Regulando rendimiento
  - [https://play.golang.org/p/dZuRIDWqD\\_E](https://play.golang.org/p/dZuRIDWqD_E)

video: 162

## Context

En servidores de Go, cada request entrante es manejado por su propia gorutina. Los request handlers usualmente inicializan gorutinas adicionales para acceder a backends tales como bases de datos o servicios RPC. El conjunto de gorutinas que trabajan en un request típicamente necesitan acceso a valores que son específicos del request, tales como identidad del usuario final, tokens de autorización y un tiempo límite del request. Cuando un request es cancelado o termina su tiempo, todas las gorutinas trabajando para ese request deberían finalizar rápidamente de manera que el sistema pueda reclamar cualquier recurso que estén usando. En Google, desarrollaron un paquete context que hace fácil pasar valores que pertenecen al scope del request, señales de cancelación y deadlines a través de APIs hacia las gorutinas que están involucradas en manejar un request. El paquete está públicamente disponible como context. Este artículo describe cómo usar el paquete, provee un ejemplo completo y funcional.

### Lecturas recomendadas:

- <https://blog.golang.org/context>
- <https://medium.com/@matryer/context-has-arrived-per-request-state-in-go-1-7-4d095be83bd8>
- <https://peter.bourgon.org/blog/2016/07/11/context.html>

### código:

- Explorando context
  - background
    - <https://play.golang.org/p/S10al3TXzxl>
  - WithCancel
    - Desechando CancelFunc
      - <https://play.golang.org/p/XOknf0aSpx>
    - Usando CancelFunc
      - <https://play.golang.org/p/4ESWI73der5>
    - Example
      - <https://play.golang.org/p/nzZgwgHurBv>
- `func WithCancel(parent Context) (ctx Context, cancel CancelFunc)`
  - <https://play.golang.org/p/WabMOiQnw3a>
- Cancelando gorutinas con deadline
- `func WithDeadline(parent Context, deadline time.Time) (Context, CancelFunc)`
  - <https://play.golang.org/p/Q6mVdQqYTt>
- Con timeout
- `func WithTimeout(parent Context, timeout time.Duration) (Context, CancelFunc)`
  - <https://play.golang.org/p/XwepAIHYf4C>
- Con valor
- `func WithValue(parent Context, key, val interface{}) Context`

- <https://play.golang.org/p/kJAwlpIBnJM>

video: 163

## Ejercicios - Ninja Nivel 10

### Ejercicio Práctico #1

- Haz que [este código](#) funcione:
  - Usando una función literal, también conocida como, función anónima autoejecutable
    - Solución: <https://play.golang.org/p/SHr3lpX4so>
  - Un canal con búfer
    - Solución: <https://play.golang.org/p/Y0Hx6lZc3U>

video: 164

### Ejercicio Práctico #2

- Haz que este código funcione:
  - <https://play.golang.org/p/oB-p3KMiH6>
    - Solución: <https://play.golang.org/p/isnJ8hMMKq>
  - <https://play.golang.org/p/DBRuelmEq>
    - Solución: <https://play.golang.org/p/mqw750EPp4>

video: 165

### Ejercicio Práctico #3

- Comenzando con [este código](#), extrae los valores del canal usando un ciclo for range
- solución: <https://play.golang.org/p/U2iGzRTtbxg>

video: 166

### Ejercicio Práctico #4

- Comenzando con [este código](#), extrae los valores del canal usando una declaración select
- solución: <https://play.golang.org/p/ZnxiCoJLe1l>

video: 167

### Ejercicio Práctico #5

- Demuestra el uso del idioma coma ok con [este código](#).
- solución: <https://play.golang.org/p/qh2ywLB5OG>

video: 168

## Ejercicio Práctico #6

- Escribe un programa que:
  - Ponga 100 números en un canal
  - Extraiga los números del canal y los imprima

**solución:** <https://play.golang.org/p/OZqLJbMeAC6>

**video:** 169

## Ejercicio Práctico #7

- Escribe un programa que:
  - Lance 10 gorutinas
    - Cada gorutina agrega 10 números a un canal
  - Extrae los números del canal e imprímelos

**soluciones:**

- <https://play.golang.org/p/bkAy9ECzVpd>
- <https://play.golang.org/p/83Vh5ymfhS2>

**video:** 170

# Manejo de Errores

## Entendiendo

- <https://golang.org/doc/faq#exceptions>
  - Go no apoya el idioma try / catch / finally

### ¿Porqué Go no tiene excepciones?

Creemos que acoplar excepciones a una estructura de control, como en la expresión try-catch-finally, da como resultado un código complicado. También tiende a incentivar a los programadores a etiquetar demasiados errores comunes, como por ejemplo no abrir un archivo, como excepcionales.

Go toma un enfoque diferente. Para el manejo simple de errores, los retornos multi-valor de las funciones en Go facilitan el reporte de un error sin sobrecargar el valor de retorno. [Un tipo de error canónico, junto con otras características de Go](#), hace que el manejo de errores sea agradable, pero bastante diferente del de otros lenguajes.

Go también tiene un algunas de funciones integradas para señalar y recuperarse de condiciones realmente excepcionales. El mecanismo de recuperación se ejecuta solamente como parte del estado de una función que es generado después de un error, el cual es suficiente para manejar una catástrofe pero no requiere estructuras de control adicionales y, cuando se usa bien, puede dar como resultado un código limpio de manejo de errores.

- [https://en.wikipedia.org/wiki/Exception\\_handling#Criticism](https://en.wikipedia.org/wiki/Exception_handling#Criticism)
  - [Notice Hoare's work also influenced goroutines and channels](#)
- <https://blog.golang.org/error-handling-and-go>

video: 171

## Chequeando errors

Normalmente, escribimos el código con errores antes de escribirlo sin errores. **Siempre chequea si hay errores. Siempre, siempre, siempre.\***

(\*casi siempre)

código:

- <https://gitlab.com/eduar/go-programming>

video: 172

## Printing & logging

Tienes varias opciones a escoger cuando necesitas imprimir o hacer logging de un error:

- `fmt.Println()`
- `log.Println()`
- `log.Fatalln()`
  - `os.Exit()`
- `log.Panicln()`
  - Funciones diferidas corren
  - Se puede usar "recover"
- `panic()`

código:

- <https://gitlab.com/eduar/go-programming>

video: 173

## Recover

<https://blog.golang.org/defer-panic-and-recover>

código:

- <https://play.golang.org/p/HI4uG55ait>
- <https://play.golang.org/p/Zocncqtwak>

video: 174

## Errores con info

Podemos agregar información a nuestros errores con

- `errors.New()`
  - `fmt.Errorf()`
- `builtin.error`

"Los valores Error en Go no son especiales, son como cualquier otro valor, y por ende tienes el lenguaje entero a tu disposición." - Rob Pike

código:

- <https://gitlab.com/eduar/go-programming>

video: 175

## Ejercicios - Ninja Nivel 11

### Ejercicio Práctico #1

Comienza con [este código](#). En vez de usar el identificador blank (underscore), asegúrate de que el código esté chequeando y manejando el error.

**solución:**

- <https://play.golang.org/p/tn8oiuL1Yn>

video: 176

### Ejercicio Práctico #2

Comienza con [este código](#). Crea un mensaje de error personalizado usando "fmt.Errorf".

**solución:**

- <https://play.golang.org/p/p3xCMEpg5Hc>
- <https://play.golang.org/p/repWMzfOOiu>
- <https://play.golang.org/p/9RFq8BXHWm5>

video: 177

### Ejercicio Práctico #3

Crea un struct "errorPer" el cual implemente la interfaz builtin.error. Crea una función "foo" que tenga un valor de tipo error como parámetro. Crea un valor de tipo "errorPer" y pásalo a "foo".

Si necesitas un pista, [aquí hay una](#).

**solución:**

- <https://play.golang.org/p/AUpl7MCqPGI>
- **assertion**
  - <https://play.golang.org/p/l3IKqOu8mJo>
- **conversion**
  - <https://play.golang.org/p/i046JdkXH8x>

video: 178

### Ejercicio Práctico #4

Comenzando con [este código](#), usa el struct raiz.Error como valor de tipo error. Si quieres usa estos valores para tu

- lat "50.2289 N"
- long "99.4656 W"

**solución:**

- <https://play.golang.org/p/Ezz-YX3tCcv>

video: 179

## Ejercicio Práctico #5

En la siguiente sección vamos aprender lo esencial a como hacer pruebas. Para este ejercicio, toma un poco de tiempo para ver cuánto puedes descubrir acerca de testing leyendo la [documentación de testing](#) y también [el artículo de Caleb Doxsey acerca de testing](#). Trata de entender y hacer funcionar un ejemplo básico sobre pruebas.

video: 180

## Escribiendo Documentación

### Introducción

Antes de escribir documentación, vamos a leer documentación. Hay varias cosas que debemos saber sobre la documentación:

- `godoc.org`
  - Documentación de la biblioteca estándar y paquetes de terceros.
- `golang.org`
  - Documentación de la biblioteca estándar.
- `go doc`
  - Comando para leer documentación en la línea de comandos.
- `godoc`
  - Comando para leer documentación en la línea de comandos.
  - También puede correr un servidor local mostrando la documentación

video: 180-a

### go doc

`go doc` imprime la documentación para un paquete, constante, función, tipo, var o método

- `go doc` acepta cero, uno, o dos **argumentos**.
  - **Cero o ninguno**
    - Imprime la documentación del paquete ubicado en el actual directorio
      - **go doc**
  - **un**
    - argumento con representación de sintaxis-de-Go del elemento a ser documentado
      - fyi: `<sim>` también conocido como “identificador”
        - **go doc <paquete>**
        - **go doc <sim>[.<método>]**
        - **go doc [<paquete>.<sim>[.<método>]]**
        - **go doc [<paquete>.] [<sim>.<método>]**
      - La documentación que es impresa es la del primer elemento en esta lista que tiene éxito. Si hay un símbolo pero no paquete, es

escogido el paquete en el actual directorio. Sin embargo, si el argumento comienza con la letra mayúscula se asume que es un símbolo en el directorio actual.

- **dos**

- El primer argumento debe ser una ruta completa hacia un paquete

- **go doc <paq> <sim>[.<método>]**

- ejemplos

Examples:

```
go doc
    Show documentation for current package.
go doc Foo
    Show documentation for Foo in the current package.
    (Foo starts with a capital letter so it cannot match
    a package path.)
go doc encoding/json
    Show documentation for the encoding/json package.
go doc json
    Shorthand for encoding/json.
go doc json.Number (or go doc json.number)
    Show documentation and method summary for json.Number.
go doc json.Number.Int64 (or go doc json.number.int64)
    Show documentation for json.Number's Int64 method.
go doc cmd/doc
    Show package docs for the doc command.
go doc -cmd cmd/doc
    Show package docs and exported symbols within the doc command.
go doc template.new
    Show documentation for html/template's New function.
    (html/template is lexically before text/template)
go doc text/template.new # One argument
    Show documentation for text/template's New function.
go doc text/template new # Two arguments
    Show documentation for text/template's New function.
```

At least in the current tree, these invocations all print the documentation for json.Decoder's Decode method:

```
go doc json.Decoder.Decode
go doc json.decoder.decode
go doc json.decode
cd go/src/encoding/json; go doc decode
```

video: 180-b

## godoc

[Godoc](#) extrae y genera documentación para programas en Go. Tiene dos modos

- **Sin el flag -http**

- Este es el modo línea de comandos; imprime la documentación en texto a la salida estándar y finaliza.
- **-src** flag
  - godoc imprime en forma de código fuente la interfaz exportada de un paquete en Go, o la implementación de un identificador específico exportado.

```
godoc fmt                # documentation for package fmt
godoc fmt Printf         # documentation for fmt.Printf
godoc cmd/go            # force documentation for the go command
godoc -src fmt          # fmt package interface in Go source form
godoc -src fmt Printf    # implementation of fmt.Printf
```

- **con -http flag**

- Corre como un servidor web y presenta la documentación como una página web
- **godoc -http=:8080**
  - <http://localhost:8080/>

video: 180-c

## godoc.org

- Coloca la url de tu código en godoc.org
  - Tu documentación aparecerá en godoc.org
  - “refresh” en la parte baja de la página si en algún momento se desactualiza

video: 180-d

## Escribiendo Documentación

La documentación es una gran parte de hacer que el software sea accesible y mantenible. Por supuesto, debe estar bien escrito y ser precisa, pero también debe ser fácil de escribir y mantener. Idealmente, debe estar acoplado al código en sí para que la documentación evolucione junto con el código. Cuanto más fácil sea para los programadores producir una buena documentación, mejor para todos.

- <https://blog.golang.org/godoc-documenting-go-code>
  - **godoc** parses Go source code - incluyendo comentarios, y produce documentación como HTML o texto sin formato. El resultado final es una documentación estrechamente unida al código que documenta. Por ejemplo, a través de la interfaz web de godoc puedes **navegar desde la documentación de una función hasta su implementación con un click.**
  - los comentarios son solo buenos comentarios, del tipo que te gustaría leer, incluso si **godoc** no existiera.
  - **A documentar**
    - Un tipo, variable, constante, función y paquete,
    - **escribe un comentario** directamente antes de su declaración, sin una línea en blanco intermedia.



- **Comienza con el nombre del elemento**
- Para paquetes
  - La primera oración aparece en la lista de paquetes (servidor local corrido con godoc)
  - Si la documentación es enorme, coloca un archivo dedicado que se debe llamar **doc.go**
    - Por ejemplo: [package fmt](#)
  - Lo mejor del enfoque mínimo de Godoc es lo fácil que es usarlo. Como resultado, una gran cantidad de código Go, que incluye toda la biblioteca estándar, ya sigue estas convenciones.
- ejemplo
  - [errors package](#)

código: <https://gitlab.com/eduar/go-programming>

video: 180-e

## Ejercicios - Ninja Nivel 12

### Ejercicio Práctico #1

Crea un paquete perro. El paquete perro debería tener una función exportada “Edad” el cuál toma años humano y los retorna como años de perro (1 año humano = 7 años de perro).

Documenta tu código con comentarios. Usa este código en func main.

- Corre el programa y asegúrate de que funciona.
- Corre un servidor local con godoc y revisa la documentación.

solución: <https://gitlab.com/eduar/go-programming>

video: 186

### Ejercicio Práctico #2

Has push del código hacia Github. Has que tu documentación aparezca en godoc.org y toma un screenshot. Borra el código de github. Actualiza godoc.org de manera que ya no tenga tu código. Tweeteame (<https://twitter.com/eduartua>) tu screenshot.

solución: <https://gitlab.com/eduar/go-programming>

video: 187

### Ejercicio Práctico #3

Usa godoc en la línea de comandos para ver la documentación de:

- fmt
- fmt Print
- strings
- strconv

solución: <https://gitlab.com/eduar/go-programming>

video: 188

## Testing & Benchmarking

### Introducción

Los tests deben

- Estar en un archivo que termina con “**\_test.go**”
- Estar en **el mismo paquete** el cual está siendo probado
- Estar en una func con la firma “**func TestXxx(\*testing.T)**”

Correr un test

- **go test**

Cómo tratar con un test que ha fallado

- usa **t.Error** para señalar una falla

Idioma recomendado

- **Expected (esperaba)**
- **Got (obtuvo)**

código: <https://gitlab.com/eduar/go-programming>

video: 187

### Tabla de tests

Podemos escribir **una serie de tests para correr**. Esto nos permite probar una variedad de situaciones.

código: <https://gitlab.com/eduar/go-programming>

video: 188

### Example tests

Examples se muestran en la documentación.

- **godoc -http :8080**
- <https://blog.golang.org/examples>
- **go test ./...**

código: <https://gitlab.com/eduar/go-programming>

video: 189

### Golint

- **gofmt**
  - Formatea código go code
- **go vet**

- Informa sobre constructs sospechosos
  - **golint**
    - Informa sobre estilo de código pobre
- <https://github.com/golang/lint>

video: 190

## Benchmark

Parte del paquete testing nos permite medir la velocidad de nuestro código. Esto también podría ser llamado “midiendo el rendimiento” de tu código, o “benchmarking” tu código - averiguar qué tan rápido se ejecuta el código.

**código:** <https://gitlab.com/eduar/go-programming>

video: 191

## Coverage (Cobertura)

Coverage (o cobertura) en programación es que tanto de nuestro código es cubierto por tests (pruebas). Podemos usar el flag “-cover” para correr un análisis de cobertura en nuestro código. Podemos usar el flag y el nombre del archivo requerido “-coverprofile <algún nombre de archivo>” para escribir nuestro análisis de cobertura a un archivo.

**código:**

- **go test -cover**
  - **go test -coverprofile c.out**
    - Mostrándolo en el navegador:
      - **go tool cover -html=c.out**
    - Aprender más
      - **go tool cover -h**
  - <https://gitlab.com/eduar/go-programming>

video: 192

## Ejemplos de Benchmark

Acá veremos algunos ejemplos que muestran el benchmarking en acción. Esto incluye comparar concatenación manual con strings.Join

**código:**

- <https://gitlab.com/eduar/go-programming>

video: 193

## Revisión

Diferentes comandos útiles cuando hacemos benchmarks, ejemplos y tests.

- **godoc -http=:8080**
- **go test**
- **go test -bench .**
  - *No olvides el “.” en la línea arriba*
- **go test -cover**

- go test -coverprofile c.out
- go tool cover -html=c.out

código:

- <https://gitlab.com/eduar/go-programming>

video: 194

## Ejercicios - Ninja Nivel 13

### Ejercicio Práctico #1

Comienza con [este código](#). Obtenga el código listo para hacer BET en él (agregue benchmarks, examples, tests). Ejecute lo siguiente en este orden:

- tests
- benchmarks
- coverage
- coverage mostrado en el navegador
- examples mostrados en documentación en el navegador web

**solución:**

- <https://gitlab.com/eduar/go-programming>

video: 195

### Ejercicio Práctico #2

Comienza con [este código](#). Obtén el código listo para BET (agrega benchmarks, examples, tests) **sin embargo** no escribas un ejemplo para la función que retorna un mapa; solamente escribe una prueba para ella como un reto extra. Agrégale documentación al código. Sigue el siguiente orden:

- tests
- benchmarks
- coverage
- coverage shown in web browser
- examples shown in documentation in a web browser

**solución:**

- <https://gitlab.com/eduar/go-programming>

video: 196

### Ejercicio Práctico #3

Comienza con [este código](#). Obtén el código listo para BET (agrega benchmarks, examples, tests). Escribe una tabla de pruebas. Agrega documentación al código. Corre lo siguiente en ese orden:

- tests

- benchmarks
- coverage
- coverage mostrado en el navegador
- examples mostrados en la documentación en el navegador

Útil saber:

- <https://play.golang.org/p/4GUqs1HMpp>
- <https://play.golang.org/p/AMp0Hq3rIrm>

solución:

- <https://gitlab.com/eduar/go-programming>

video: 197

## DESPEDIDA

Haz hecho un excelente trabajo - el trabajo más difícil. Esto no es algo que solamente te va hacer mejor persona a ti sino también a quienes te acompañan. Mientras mejoras como persona también mejoras al mundo. Las habilidades que estás obteniendo son unas de las más demandadas hoy en día: saber programar y saber cómo usar el lenguaje de programación Go.

Próximos pasos

- [Sígueme en Twitter](#)
- [Mi curso de desarrollo web "Go Web" próximamente](#)
- <https://goo.gl/uNb5QJ>
  - <http://amzn.to/1RIM5HP>
  - <http://amzn.to/1kGGsPv>
- **Sitios web interesantes:**
  - <https://godecl.org/>
  - <http://exercism.io/languages/go/about>
  - <http://gocode.io/>

video: 198