Índice

[**¿Qué es, por qué y quienes utilizan Go?** 1](#_Toc81754504)

[**¿Qué es?** 1](#_Toc81754505)

[**¿Porqué Go/Goland?** 1](#_Toc81754506)

[**¿Dónde se usa?** 1](#_Toc81754507)

[**Tipos de datos primitivos** 2](#_Toc81754508)

[**Número enteros** 2](#_Toc81754509)

[**Números enteros positivos** 2](#_Toc81754510)

[**Números decimales** 3](#_Toc81754511)

[**Textos y boléanos** 3](#_Toc81754512)

[**Número complejos** 3](#_Toc81754513)

[**Paquete fmt: algo más que imprimir en consola** 3](#_Toc81754514)

[**Operadores lógicos y de comparación** 5](#_Toc81754515)

[**Operadores de comparación** 5](#_Toc81754516)

[**Operadores lógicos** 5](#_Toc81754517)

[**Operador AND** 5](#_Toc81754518)

[**Operador OR** 6](#_Toc81754519)

[**Operador NOT** 6](#_Toc81754520)

[**Cheat Sheet Go** 6](#_Toc81754521)

[**Hola mundo** 7](#_Toc81754522)

[**¿Hacer una impresión en consola rápida?** 7](#_Toc81754523)

[**Importar una librería sin usarla** 8](#_Toc81754524)

[**Agregar un alias a un import (no suele usarse, pero es bueno saberlo)** 9](#_Toc81754525)

[**Diferentes formas de declarar variables** 9](#_Toc81754526)

[**Zero values de primitivos** 9](#_Toc81754527)

[**Incremental y decremental** 10](#_Toc81754528)

[**Imprimir tipo de variables (hay otras formas, pero esta es la más fácil)** 10](#_Toc81754529)

[**Función para tomar los errores (ahorra mucho código)** 10](#_Toc81754530)

[**Arrays vs Slices** 11](#_Toc81754531)

[**Slice de interfaces (Úsalo con sabiduría)** 11](#_Toc81754532)

[**Asegurarnos si un key existe en el map** 12](#_Toc81754533)

[**Punteros** 12](#_Toc81754534)

[**Comandos de Go modules** 13](#_Toc81754535)

[**Librerías para desarrollo Web con Go** 14](#_Toc81754536)

[**FrontEnd – Hugo** 14](#_Toc81754537)

[**BackEnd** 15](#_Toc81754538)

[**Echo** 15](#_Toc81754539)

[**Gin-Gonic** 16](#_Toc81754540)

[**Beego** 16](#_Toc81754541)

[**Revel** 17](#_Toc81754542)

[**Gorilla** 17](#_Toc81754543)

[**Buffalo** 18](#_Toc81754544)

# **¿Qué es, por qué y quienes utilizan Go?**

## **¿Qué es?**

* Lenguaje compilado y estáticamente tipado (se debe indicar el **tipo** de variable o constante para que guarde algún valor en él.)
* Maneja procesos pesados, es potente, pero amigable.
* Se utiliza Go/Goland para nombrarlo.
* Los programadores de este lenguaje se hacen llamar **gophers**.

## **¿Porqué Go/Goland?**

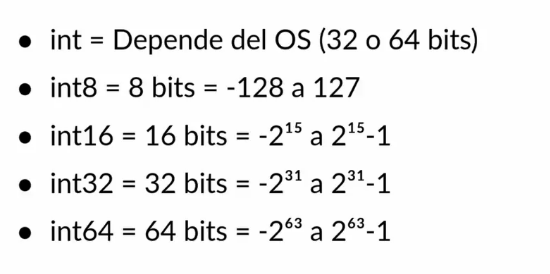
* Gran velocidad de compilación.
* Tiene alto rendimiento para tareas pesadas.
* Maneja soporte nativo por concurrencia.
* Un Gopher puede ganar $74k al año.
* Facilita ajustar sintaxis de forma nativa.
* Comunidad receptiva, contribuye y apoya.

## **¿Dónde se usa?**

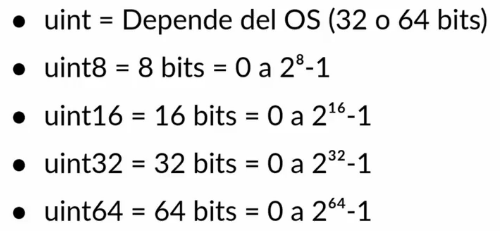
* Mercado Libre.
* Twich.
* Twitter.
* Uber.
* Docker y Kubernetes.

# **Tipos de datos primitivos**

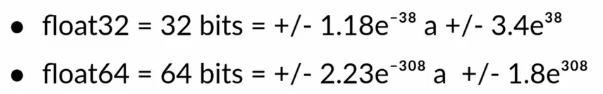
## **Número enteros**



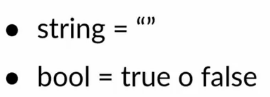
## **Números enteros positivos**



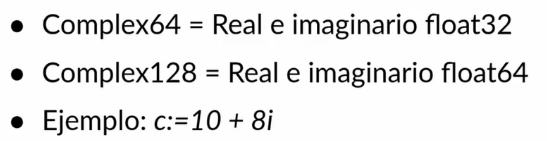
## **Números decimales**



## **Textos y boléanos**



## **Número complejos**



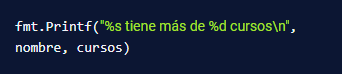
# **Paquete fmt: algo más que imprimir en consola**

El paquete fmt es el que se encarga de administrar los inputs y outputs de la terminal.

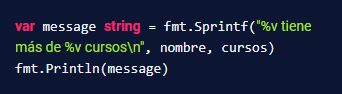
* **Println**: Es un print normal con un salto de línea al final. Ejemplo:



* **Printf**: Es un print al cual le puedes especificar el tipo de objeto que le vas a dar. Ejemplo:

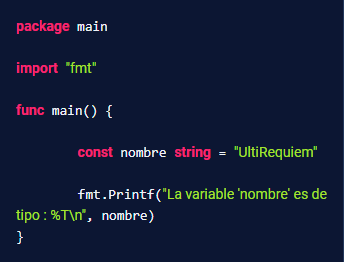


* **Sprintf**: No imprime nada en consola, simplemente lo guarda como un String. Ejemplo de uso:



**Imprimir el tipo de dato**

Con este paquete podemos imprimir en consola el tipo de dato de **variables** o **constantes**. Ejemplo de uso:



Documentación oficial de fmt [aquí](https://pkg.go.dev/fmt).

# **Operadores lógicos y de comparación**

Son operadores que nos permiten hacer una comparación de condiciones y en caso de cumplirse como sino ejecutarán un código determinado. Si se cumple es VERDADERO/TRUE y si no se cumple son FALSO/FALSE.

Empecemos con los operadores de comparación:

## **Operadores de comparación**

Son aquellos que retornan TRUE o FALSE en caso de cumplirse o no una expresión. Son los siguientes:

* **valor1 == valor2**: Retorna **TRUE** si valor1 y valor2 son exactamente iguales.
* **valor1 != valor2**: Retorna **TRUE** si valor1 es diferente de valor2.
* **valor1 < valor2**: Retorna **TRUE** si valor1 es menor que valor2
* **valor1 > valor2**: Retorna **TRUE** si valor1 es mayor que valor2
* **valor1 >= valor2**: Retorna **TRUE** si valor1 es igual o mayor que valor2
* **valor1 <= valor2**: Retorna **TRUE** si valor1 es menor o igual que valor2.

## **Operadores lógicos**

Son aquellos que retorna TRUE o FALSE si cumplen o no una condición utilizando [puertas lógicas](https://youtu.be/Pfyuv5ZnNNw).

## **Operador AND**

Este operador indica que todas las condiciones declaradas deben cumplirse para poderse marcar como TRUE. En Go, se utiliza este símbolo **&&**.

**Ejemplo1**: 1>0 && 2 >0 Esto retornará TRUE porque tanto la primera como la segunda condición son verdaderas.

**Ejemplo2**: 2<0 && 1 > 0 Esto retornará FALSE porque una de las condiciones no es verdadera.

## **Operador OR**

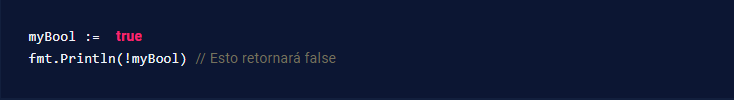
Este operador indica que al menos una de las condiciones debe cumplirse para marcarse como TRUE. En Go, se representa con el símbolo **||**.

**Ejemplo**: 2<0 || 1 > 0 Esto retornará TRUE porque la segunda condición se cumple, a pesar que la primera no.

## **Operador NOT**

Este operador retornará el opuesto al booleano que está dentro de la variable.

Ejemplo



Una vez ya estudiado la teoría, en la siguiente clase vamos a ver cómo utilizarlo con más detalles en Go.

# **Cheat Sheet Go**

En esta lectura te llevarás algunos tópicos que siempre vale la pena tener a la mano al momento de programar en Go, además te dejo algunos tips extras.

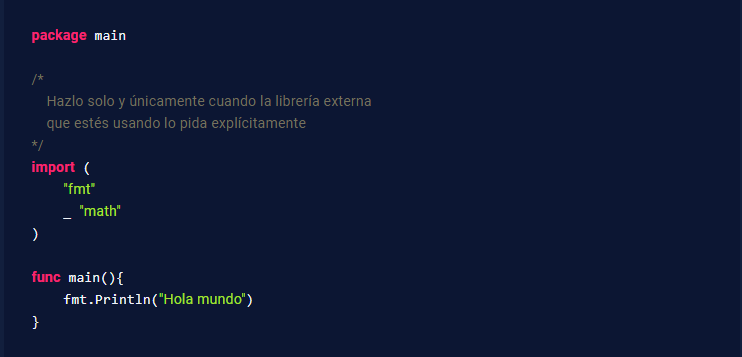
## **Hola mundo**



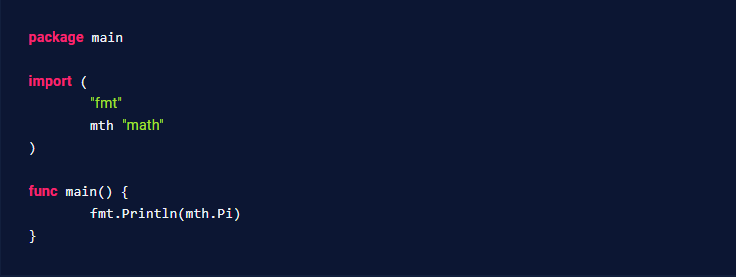
## **¿Hacer una impresión en consola rápida?**



## **Importar una librería sin usarla**



## **Agregar un alias a un import (no suele usarse, pero es bueno saberlo)**



## **Diferentes formas de declarar variables**



## **Zero values de primitivos**



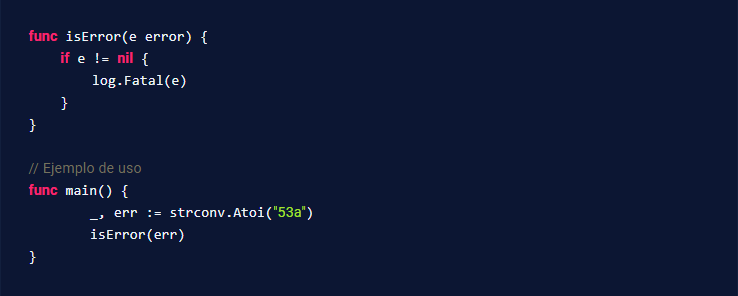
## **Incremental y decremental**



## **Imprimir tipo de variables (hay otras formas, pero esta es la más fácil)**



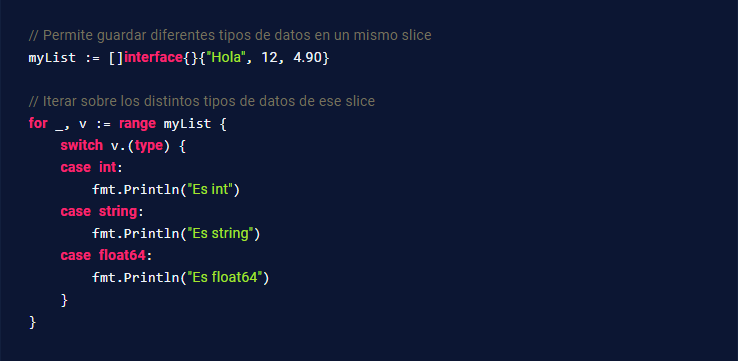
## **Función para tomar los errores (ahorra mucho código)**



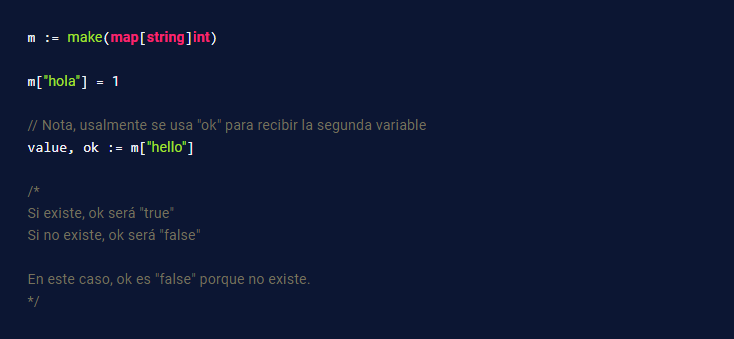
## **Arrays vs Slices**



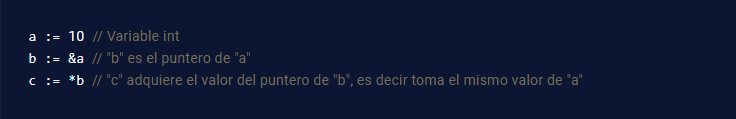
## **Slice de interfaces (Úsalo con sabiduría)**



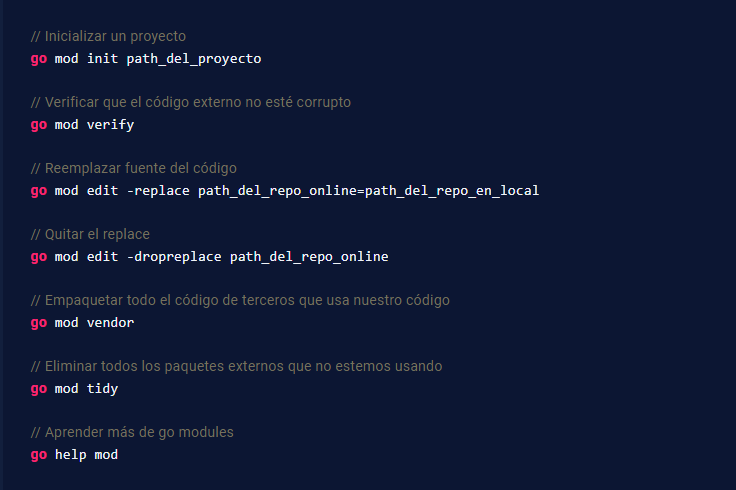
## **Asegurarnos si un key existe en el map**



## **Punteros**



## **Comandos de Go modules**



# **Librerías para desarrollo Web con Go**

Actualmente puedes usar **Go** para crear tu aplicación web sin necesidad de implementar librerías externas como sucede en muchos otros lenguajes. Aunque claro, esto tiene sus pros y sus contras. Pero al final del día todo dependerá del flujo de trabajo de tu equipo.

Sin embargo, te quiero compartir las **librerías** y **frameworks** de desarrollo web que se encuentran entre las más populares.

## **FrontEnd – Hugo**



[Hugo](https://gohugo.io/) es el framework más popular para usar Go en el FrontEnd, claro no es que se utilice en FrontEnd específicamente, sino que tiene su propio template para generar archivos estáticos (sin Backend, solo HTML, CSS y JavaScript).

Pero la **mayor ventaja** de **Hugo**, es que tienen [**Hugo Themes**](https://themes.gohugo.io/)una tienda de **templates** donde muchos miembros de la comunidad suben sus temas para disponerlos al resto de la comunidad para su uso, solo debes prestar atención a la licencia. Pero encontrarás temas con todo tipo de features, desde uso para portafolio, compatibilidad con Google Anlytics y mucho más.

Y si lo combinas con **Vercel** (herramienta de Deployment automático) podrás desplegar tu aplicación de forma gratuita.

## **BackEnd**

Para el caso de BackEnd tenemos muchas opciones, las que te recomiendo probar son las siguientes:

### **Echo**



[Echo](https://echo.labstack.com/) es el **Framework** es uno de los más utilizados. Esto porque reúne varias características:

* Es minimalista, evitando tener código que no vamos a usar.
* Fácil de escalar y un excelente rendimiento al momento de compilar.
* Tiene su sistema de enrutamiento, pudiendo incluso implementar middlewares (proceso de autenticación)
* Tiene su propia interfaz para implementar websockets (conexiones en tiempo real)
* Puedes usarlo tanto para crear APIs como páginas web completas.
* Es uno de los pocos framewords que tiene su propia implementación de JWT (otro tipo de autenticación muy usado en APIs)

### **Gin-Gonic**



[**Gin-Gonic**](https://gin-gonic.com/) es un framework al que fácilmente le puedes agarrar cariño y esto es porque la forma como interactúas para crear Apps es muy intuitiva. De momento no es de los Frameworks que más destaque, pero tampoco se queda atrás. Ideal para tus primeros proyectos de BackEnd con Go.

### **Beego**



[Beego](https://beego.me/) por bastante tiempo llegó a ser uno de los Frameworks más usados, no es que haya dejado de serlo, pero trajo a la comunidad 3 grandes features:

* Su propio ORM (Object Relational Mapping), básicamente son un conjunto de métodos para interactuar con la base de datos desde consulta a creación de tablas.
* Un CLI (Command Line Interface) llamado **bee** con el cual puedes generar proyecto ya sea para hacer una API como página web completa, empaquetar el proyecto listo para llevarlo a producción y hacer migraciones de bases de datos.
* Un admin, ¡sí un admin! Con el cual puedes ver estadísticas de las rutas de tu app, verificar el estado de la base de datos y ejecutar tareas de forma manual
* Hoy en día Beego tiene muchos más features como por ejemplo crear el **Dockerfile** para que tu app se ejecute en Docker. Si vas a crear múltiples apps seguidamente, te recomiendo este Framework.

### **Revel**



[Revel](https://revel.github.io/) es uno de los pesos pesados en desarrollo web para Go incluso fue conocido como el django de Go (django es el framework web más popular para Python) pero dejó de recibir actualizaciones por un tiempo significativo y también perdió un poco esa posición.

Hoy en día, Revel no solo ha llegado con nuevo look, sino que también le han invertido a tener una de las documentaciones más completas incluso con apps de ejemplo.

### **Gorilla**



[Gorilla](https://www.gorillatoolkit.org/) **no es un Framework**, ellos se definen como con herramientas para web. No tiene un **ORM**, tampoco un **CLI**, tampoco **render** de HTML, tampoco un **admin**. Entonces ¿por qué está en esta lista?

**Pues en gorilla se enfocaron en una cosa hasta hacerla muy bien**, estos son los **WebSockets** (para mantener conexiones en tiempo real por ejemplo un chat, algo en lo que Go destaca y por mucho). Con el paso del tiempo han agregado más features, como por ejemplo manejos de cookies, enrutador, etc.

Puedes desarrollar una web completa usando las librerías nativas de Go y con gorilla tienes esos extras que te ahorran implementación. Han hecho tan bien el código que incluso framworks web lo usan en su código sobre todo la parte de websockets.

### **Buffalo**



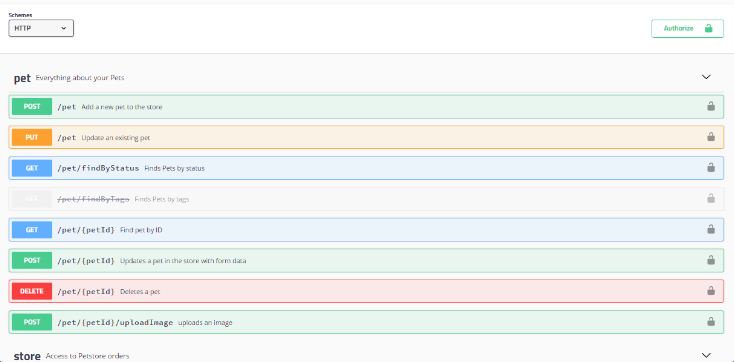
[Buffalo](https://gobuffalo.io/en/) **es otro de los pesos pesados** en frameworks para web con Go. Hoy en día, es uno de los más completos tiene **CLI**, **ORM**, manejo de **cookies**, **middlewares**, **templates** para reder de HTML, **empaquetador** con Docker e incluso puedes **hacer despliegue a Heroku**, **Digital Ocean** y **Azure** con el mismo CLI de Buffalo. Junto a otros features adicionales.

Sin embargo, el que sea más completo no significa que sea fácil de implementarlo. Es fácilmente recomendable para aquellas aplicaciones que necesite ser robustas desde el principio, mi sugerencia acá es que no seas el único(ca) desarrollador(ra) en el equipo ya que el mantenimiento es tan pesado como lo es la librería.

**Extra**

Como programdor(ra) en más de una ocasión debes crear documentación de la API que haz desarrollado, pues no suele ser de las tareas más divertidas al desarrollar. Pero si te dijera que puedes crear documentación en la medida que desarrollas ¿me creerías?

Digamos que sí me crees, ¿me seguirías creyendo si te dijera que generaría toda una interfaz parecida a esta?



Esto se pone más interesante, ahora imagina que esa documentación autogenerada además maneja autenticación e incluso es interactiva. Puedes verlo por ti mismo(ma) en este [swagger de ejemplo](https://petstore.swagger.io/).

Allí es donde entra [swaggo](https://github.com/swaggo/swag)



[swaggo](https://github.com/swaggo/swag) es la forma como podemos implementar **swagger** en nuestro código en Go. De los frameworks que te he compartido acá, es compatible con **Gin-Gonic**, **echo** y **búffalo**. **Beego**, por su parte, está incluido en su **CLI**.

Como punto adicional, dependiendo de la librería los comentarios para hacer la documentación en swagger pueden generarse de forma automática o bien requiere una cierta intervención manual.