



# GO

## EL LENGUAJE DE PROGRAMACION DE GOOGLE

*Cristian Zambrano*

*David Casas*

*Veronica Espinel*

*Jhon carrascal*





# HISTORIA



*Golang, también conocido como Go, fue creado en Google en 2007 por Robert Griesemer, Rob Pike y Ken Thompson. Fue lanzado públicamente en 2009. El objetivo era crear un lenguaje eficiente y fácil de usar para manejar grandes sistemas y escalabilidad.*

*Motivación detrás del lenguaje: Golang fue diseñado para abordar problemas comunes en el desarrollo de software, como tiempos de compilación largos, gestión complicada de dependencias y concurrencia.*

# VERSION ACTUAL

Versión 1.0 lanzada, marcando la estabilidad del lenguaje.

2009

Primer versión publica

2012

Última versión estable, Golang 1.22, con mejoras en el rendimiento y nuevas características.

2024

## Principales características y mejoras:

- Mejoras en el rendimiento del compilador.
- Nuevas herramientas para el análisis de código.
- Mejoras en el manejo de módulos y dependencias.

# UBICACIÓN EN EL RANKING Y UTILIDAD DEL LENGUAJE

Golang se encuentra en el top 20 de los lenguajes más populares según el índice TIOBE y es uno de los lenguajes más utilizados en GitHub.

Golang es un lenguaje de programación que sirve para muchas cosas, por ejemplo: apps, compilación de plataforma cruzada, arquitectura IT, bases de datos, inteligencia artificial, páginas web, debugs, configuración de servidores, sistemas operativos, soluciones en la nube y más.

De acuerdo con los expertos, es como tener lo mejor de Python y de C en un solo lugar.

Programming Language
 Python
 C
 C++
 Java
 C#
 JavaScript
 SQL
 Go
 Visual Basic
 PHP

# METODOS Y PROPIEDADES

Go no tiene clases en el sentido tradicional de la programación orientada a objetos. En su lugar, utiliza structs y métodos para agrupar datos y comportamientos.

Los structs son tipos de datos compuestos que pueden tener métodos asociados.

# SINTAXIS BÁSICA DE GO



Todos los archivos de GO  
deben pertenecer a un  
paquete

Importa los paquetes que  
se necesitan para que el  
programa funcione

Si la primera letra  
comienza en:

Minúscula la variable  
será privada

Mayúscula la  
variable será pública

Debes indicar el  
tipo de dato

La función main es lo  
primero que se ejecuta

```
package main
import "fmt"
var noExportada string
var Exportada string
func main() {
    saludo := "Hola mundo"
    fmt.Println(saludo)
}
```

Con el operador := se puede declarar  
variables y asignar un valor en lugar  
de usar var y el tipo de dato

# ESTRUCTURAS (STRUCTS)

struct en Go es un tipo de datos compuesto que agrupa campos para representar un objeto.

En este ejemplo, Person es una estructura con dos campos: Name y Age. Se crea una instancia de Person llamada p y se accede a sus propiedades utilizando la notación de punto (p.Name y p.Age).

```
package main

import "fmt"

type Person struct {
    Name string
    Age  int
}

func main() {
    p := Person{Name: "Alice", Age: 30}
    fmt.Println(p.Name) // Acceso a la propiedad Name
    fmt.Println(p.Age) // Acceso a la propiedad Age
}
```

# METODOS

```
package main

import "fmt"

type Person struct {
    Name string
    Age  int
}

// Método asociado al tipo Person
func (p Person) Greet() string {
    return "Hello, my name is " + p.Name
}

func main() {
    p := Person{Name: "Bob", Age: 25}
    fmt.Println(p.Greet()) // Llamada al método Greet
}
```

Un método en Go es una función con un receptor. Los métodos se asocian a tipos definidos (estructuras), permitiendo que las estructuras tengan comportamientos.

---

El método Greet está asociado a la estructura Person. El receptor p Person indica que Greet puede ser llamado en cualquier instancia de Person. Cuando se llama a p.Greet(), devuelve un saludo personalizado.

# ENCAPSULAMIENTO

El encapsulamiento en Go está definido a nivel de Package y no posee modificadores de visibilidad, si un atributo es declarado con letra inicial mayúsculas, significa que es exportado fuera del paquete, en cambio con letra inicial minúscula sería el equivalente al private en C# o Java.

Aquí, la estructura person y sus campos name y age son privados porque sus nombres comienzan con minúscula. Sin embargo, el método Greet es público y puede ser accedido desde fuera del paquete porque su nombre comienza con mayúscula.

```
package main

import "fmt"

type person struct {
    name string
    age  int
}

// Método público
func (p person) Greet() string {
    return "Hello, my name is " + p.name
}

func main() {
    p := person{name: "Charlie", age: 35}
    fmt.Println(p.Greet())
}
```



# PROPIEDADES

## ◆ Getters y Setters

Aunque Go no tiene getters y setters explícitos como otros lenguajes, se pueden crear métodos para manejar el acceso y la modificación de los campos de una estructura.

---

En este ejemplo, GetName y SetName son métodos que actúan como getter y setter para el campo privado name de la estructura Person. Esto permite controlar el acceso y la modificación de name de manera controlada.

```
package main

import "fmt"

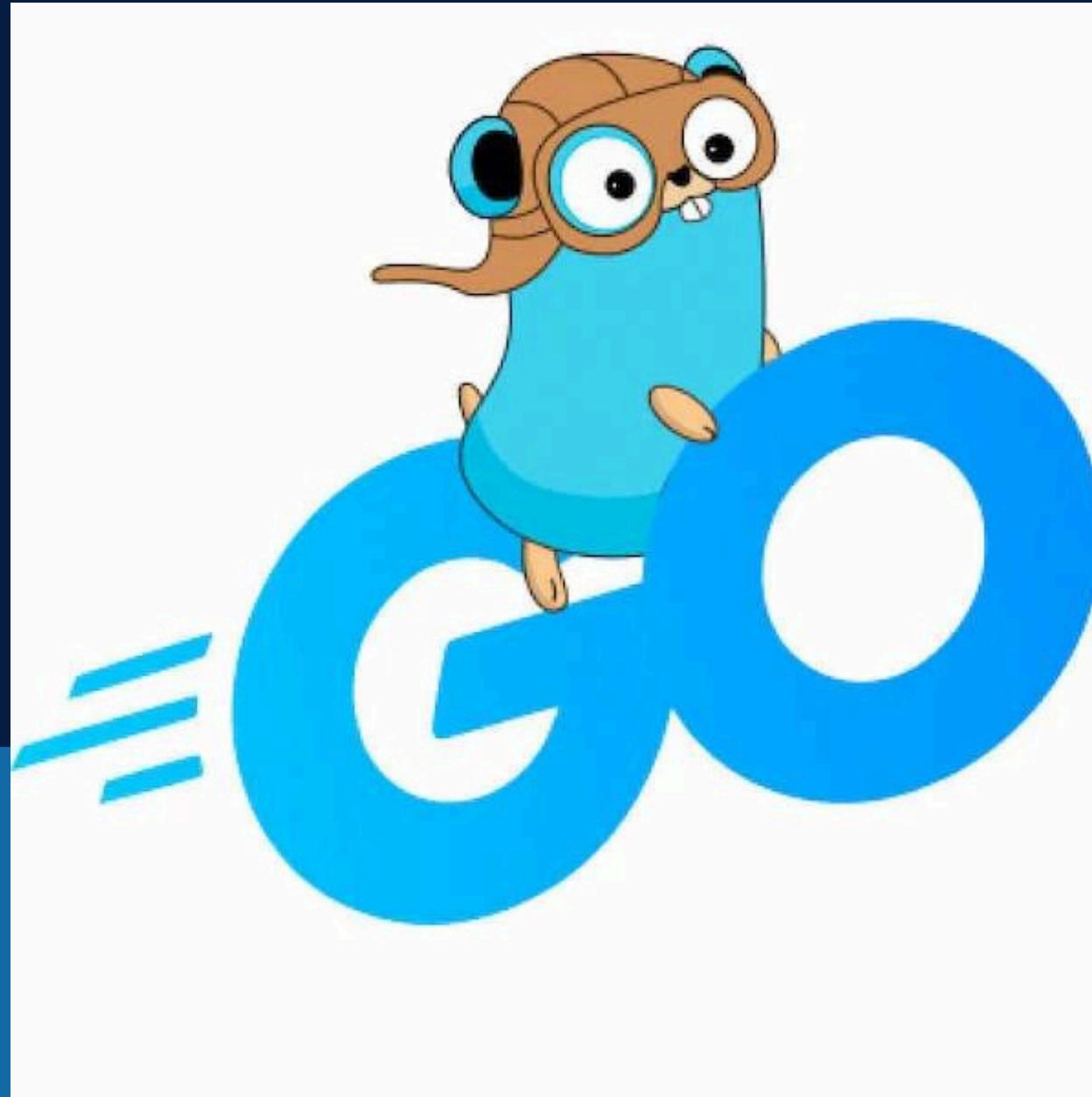
type Person struct {
    name string
    age  int
}

// Getter para el campo name
func (p *Person) GetName() string {
    return p.name
}

// Setter para el campo name
func (p *Person) SetName(name string) {
    p.name = name
}

func main() {
    p := Person{name: "David", age: 40}
    fmt.Println(p.GetName()) // Llamada al getter
    p.SetName("Daniel")
    fmt.Println(p.GetName()) // Verificación del setter
}
```

# ASOCIACIÓN, AGREGACIÓN Y COMPOSICIÓN



Son conceptos importantes en la programación orientada a objetos, y aunque Go no tiene clases tradicionales, estos conceptos se pueden implementar usando estructuras (structs)

## Asociación

La asociación es una relación simple entre dos estructuras que pueden existir independientemente. En Go, esto se puede representar mediante campos de tipo struct en otro struct.

```
type Address struct {  
    City     string  
    ZipCode string  
}  
  
type Person struct {  
    Name      string  
    Address   Address  
}
```

Aquí, Person tiene una relación de asociación con Address. La estructura Address puede existir independientemente de Person.

## Agregación

La agregación es una forma más débil de composición donde las estructuras pueden existir independientemente, pero una estructura contiene a otra.

```
type Department struct {  
    Name      string  
    Employees []Person  
}
```

En este caso, Department tiene una relación de agregación con Person.

# Composición

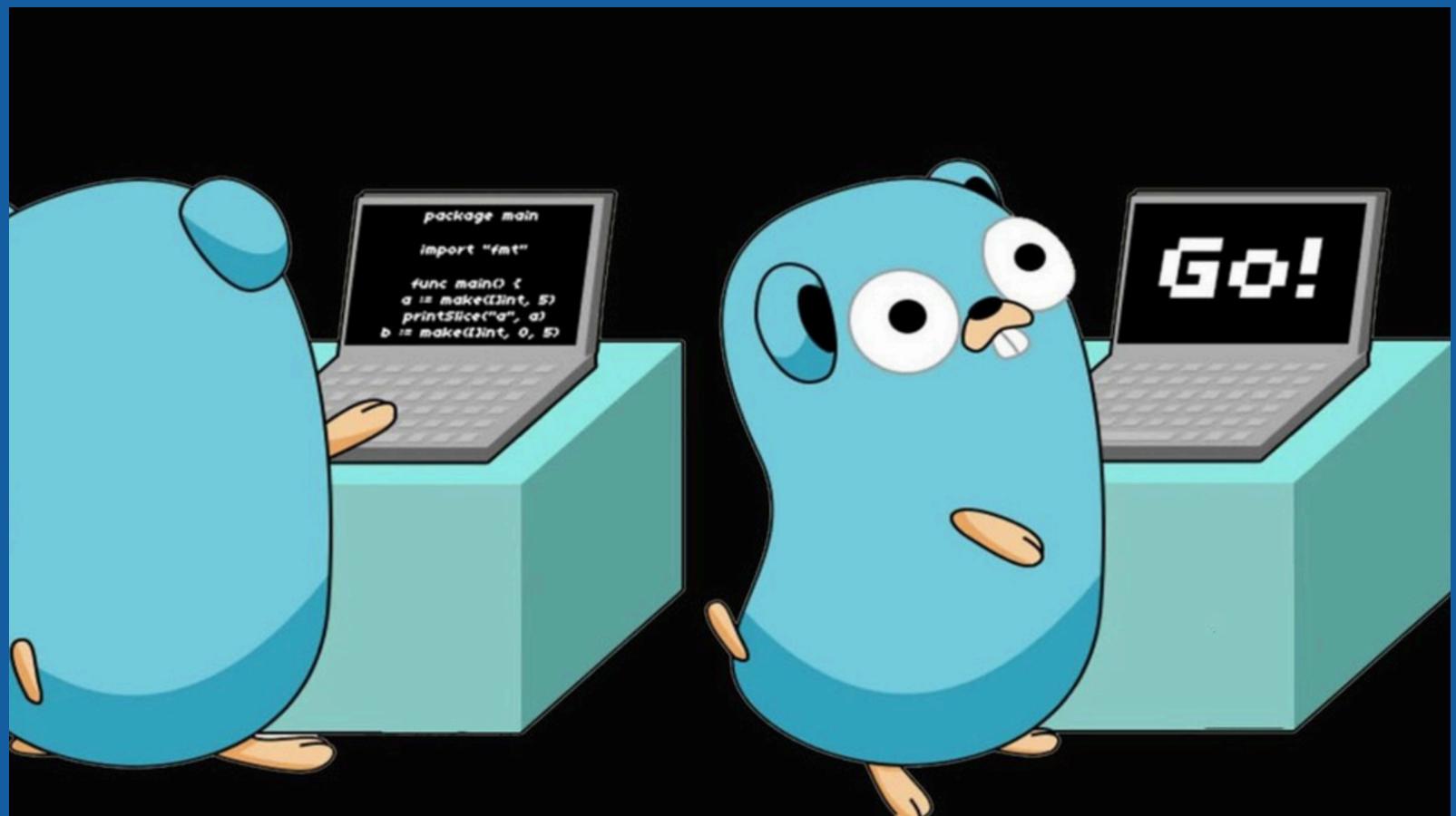
La composición es una relación más fuerte donde una estructura no puede existir sin la otra.

Aquí, Car tiene una relación de composición con Engine. El Engine es una parte integral del Car y no puede existir sin él. Si destruyes el Car, el Engine también dejará de existir

```
type Engine struct {  
    Horsepower int  
}  
  
type Car struct {  
    Brand string  
    Engine Engine  
}
```

# HERENCIA

La herencia, como probablemente sabemos por otros lenguajes de programación, es aquel mecanismo por el cual un objeto se basa en otro, utilizando sus mismos atributos y comportamiento . Go no dispone de herencia, pero permite la composición





# POLIMORFISMO

el polimorfismo se logra mediante interfaces. Las interfaces permiten definir un conjunto de métodos que una estructura debe implementar.

---

La interfaz Animal define un método Speak(). Las estructuras Dog y Cat implementan esta interfaz. La función MakeAnimalSpeak toma un Animal como argumento y llama al método Speak. Esto muestra cómo Go utiliza interfaces para lograr el polimorfismo.

```
// Definición de la interfaz
type Animal interface {
    Speak() string
}

// Implementación de la interfaz en struct Dog
type Dog struct {
    Name string
}

func (d Dog) Speak() string {
    return "Woof!"
}

// Implementación de la interfaz en struct Cat
type Cat struct {
    Name string
}

func (c Cat) Speak() string {
    return "Meow!"
}

// Función que toma una interfaz como argumento
func MakeAnimalSpeak(a Animal) {
    fmt.Println(a.Speak())
}

func main() {
    dog := Dog{Name: "Fido"}
    cat := Cat{Name: "Whiskers"}

    MakeAnimalSpeak(dog)
    MakeAnimalSpeak(cat)
}
```

# GUI: DESKTOP, CLI (COMANDOS/CONSOLA), WEB Y MÓVIL.

## Desktop

Las aplicaciones de escritorio (GUI) en Go no son tan comunes como en otros lenguajes que se centran más en la interfaz de usuario, como Java o C#. Sin embargo, existen bibliotecas como fyne y andlabs/ui que permiten a los desarrolladores crear aplicaciones de escritorio con interfaces gráficas.

- Fyne: Es una biblioteca para construir aplicaciones de escritorio modernas y fáciles de usar. Es conocida por su simplicidad y capacidad para crear aplicaciones multiplataforma (Windows, macOS, Linux).
- andlabs/ui: Es otra biblioteca para construir interfaces gráficas, aunque es menos popular y menos activa que fyne.

# GUI: DESKTOP, CLI (COMANDOS/CONSOLA), WEB Y MÓVIL.

## CLI (Comandos/Consola)

Go es altamente eficiente para construir herramientas de línea de comandos (CLI) debido a su simplicidad y eficiencia. Las bibliotecas populares para este propósito incluyen cobra y urfave/cli.

- Cobra: Es una biblioteca poderosa para crear aplicaciones CLI. Se utiliza en proyectos como Kubernetes. Proporciona un marco estructurado para definir comandos y sus parámetros, lo que facilita la creación de herramientas de línea de comandos robustas y escalables.
- urfave/cli: Es otra opción popular para construir aplicaciones CLI en Go. Es conocida por su simplicidad y facilidad de uso.

# GUI: DESKTOP, CLI (COMANDOS/CONSOLA), WEB Y MÓVIL.

## Web

Go es ampliamente utilizado en el desarrollo web, especialmente en la construcción de servidores web y API. Los frameworks más destacados incluyen Gin, Echo y Fiber.

- Gin: Es un framework web ligero y de alto rendimiento, ideal para crear APIs RESTful. Es conocido por su velocidad y simplicidad.
- Echo: Otro framework web que se enfoca en la eficiencia y simplicidad, proporcionando características robustas para la construcción de aplicaciones web y APIs.
- Fiber: Está inspirado en Express.js (un framework de Node.js) y está diseñado para ser fácil de usar y muy rápido.

# GUI: DESKTOP, CLI (COMANDOS/CONSOLA), WEB Y MÓVIL.

## Móvil

Go también puede ser utilizado para el desarrollo de aplicaciones móviles, aunque no es su uso más común. La herramienta gomobile permite compilar aplicaciones Go para Android e iOS.

- **gomobile:** Es una herramienta y paquete que facilita la creación de aplicaciones móviles con Go. Permite a los desarrolladores escribir código en Go y compilarlo para las plataformas móviles principales.

# GRACIAS POR SU ATENCIÓN

