Edier Santiago Moyano

Aprendices

Instructor:
CESAR ARTURO ESQUIVEL CORTES

SENA SERVICIO NACIONAL DE APRENDIZAJE

ADSO 2901879

IBAGUÉ – TOLIMA

Centro De Industria y la construcción

Contenido

¿Qué es Go?

Go, también conocido como **Golang**, es un lenguaje de programación de código abierto creado por **Google** en 2007 y lanzado oficialmente en 2009. Fue desarrollado principalmente por **Robert Griesemer, Rob Pike y Ken Thompson**, tres ingenieros veteranos con experiencia en sistemas como Unix y C.

¿Para qué fue creado Go?

Go fue diseñado para resolver problemas comunes en el desarrollo de software moderno, especialmente en **infraestructura, sistemas distribuidos y aplicaciones web a gran escala**. Los objetivos principales del lenguaje incluyen:

1. Simplicidad y productividad: Go busca combinar el rendimiento y control de lenguajes como **C** con la simplicidad y velocidad de desarrollo de lenguajes modernos como **Python**. Tiene una sintaxis clara y un conjunto de características reducido, lo cual evita la complejidad innecesaria.

2. Concurrencia eficiente:Uno de los aspectos más innovadores de Go es su modelo de **concurrencia** basado en "goroutines", que permite ejecutar múltiples tareas al mismo tiempo de manera muy liviana y eficiente, ideal para aplicaciones de red, servidores web o procesamiento paralelo.

3. Compilación rápida: Go se compila muy rápido, lo cual ayuda a mejorar el ciclo de desarrollo, algo crucial para equipos que trabajan en proyectos grandes y complejos.

4. Herramientas incorporadas:Go incluye herramientas integradas para **formatear código**, **compilar**, **hacer pruebas (testing)**, **documentar** y **administrar dependencias**. Esto fomenta buenas prácticas y facilita la colaboración en equipos.

5. Despliegue sencillo:Go compila directamente a **binarios ejecutables estáticos**, lo que facilita su distribución y despliegue sin necesidad de intérpretes ni entornos de ejecución adicionales.

Como instalar Go/Golang

1. Vaya a https://go.dev/doc/install y descargue el paquete para su sistema operativo.

   Ejecuta el instalador y al finalizar el proceso tendrás **go** disponible.

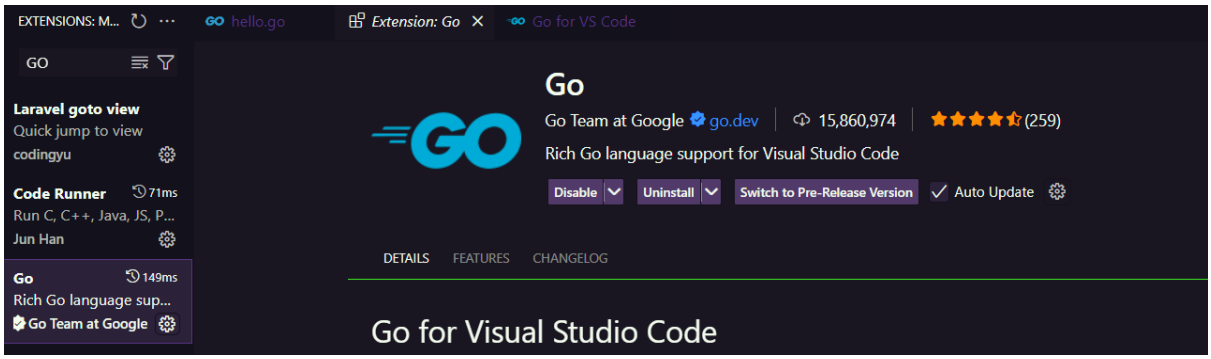
2. Luego abre la terminal y ejecuta go version y deberias de ver algo asi:

```
C:\Users\edier>go version
go version go1.24.3 windows/amd64
```

Cómo Configurar su editor

Recomiendo utilizar **Visual Studio Code** como editor durante este manual estaremos haciendo uso de este.

Lee Go en Visual Studio Code para una configuración rápida y operativa. Como mínimo, instala la extensión de Go .



Esta extensión te hará la vida más fácil, ya que proporciona IntelliSense (resaltado de sintaxis, autocompletado, información al pasar el mouse, resaltado de errores, etc.)

Como hacer nuestro primer hola mundo

Primero debemos de crear un archivo con ala extencion .go **hello.go** dentro de este agregamos este contenido \

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

Cómo se organiza un archivo .go

Los programas Go se organizan por paquetes , cada .go primero se declara de qué paquete es parte

Un paquete puede estar compuesto por varios archivos o por un solo archivo , de igual manera un programa puede contener un solo paquete .
El paquete **Main** es el punto de entrada del programa e identifica un programa ejecutable.

```go
package main
```

Import

```go
import "fmt"
```

Usamos la palabra clave **import** para importar algún paquete.

fmt es un paquete integrado el cual es proporcionado por Go el cual nos proporciona utilidades de entrada/salida.
Go cuenta con una gran biblioteca lista para ser usadas en cualquier cosa.

Puede leer sobre todas las características que **fmt** proporciona este paquete en la documentación oficial .

Declaraciones: funciones, variables, structs, etc.

Aquí escribes el **código funcional**: funciones, estructuras, interfaces, variables, constantes, etc.

```
func saludar(nombre string) string {
    return "Hola, " + nombre
}
```

func main()

Si el paquete es **main**, esta función es el **punto de entrada** del programa.

```
func main() {
    mensaje := saludar("Santiago")
    fmt.Println(mensaje)
}
```

Tipos de datos

Go es un lenguaje tipado .

cómo declarar una variable especificando su tipo

```
var age int
```

o se puede dejar que Go infiera el tipo a partir del valor inicial asignado:

```
var age = 10
```

Los tipos básicos en Go son:

- Enteros (int, int8, int16, int32, rune, int64, uint, uintptr, uint8, uint16, uint64)
- Flotantes (float32, float64), útiles para representar decimales
- Tipos complejos (complex64, complex128), útiles en matemáticas
- Byte (byte), representa un solo carácter ASCII
- Cadenas (string), un conjunto de bytes
- Booleanos (bool), verdaderos o falsos

Estructuras de control de go

Las estructuras de control permiten ejecutar instrucciones de manera condicional o repetitiva. Go incluye `if`, `switch` y `for`.

Condicionales (`if`, `else`, `switch`)

- `if` y `else` funcionan igual que en otros lenguajes como C o Java, pero no requieren paréntesis en la condición.
- `switch` es más simple que en otros lenguajes, ya que no requiere `break` y evalúa automáticamente.

```go
if edad >= 18 {
    fmt.Println("Eres mayor de edad")
} else {
    fmt.Println("Eres menor de edad")
}

switch dia {
case "lunes":
    fmt.Println("Inicio de semana")
case "viernes":
    fmt.Println("Fin de semana cerca")
default:
    fmt.Println("Día cualquiera")
}
```

Bucles (for**)**

Go solo tiene un tipo de bucle: `for`, que puede comportarse como `while`.

```go
for i := 0; i < 5; i++ {
    fmt.Println(i)
}

// Bucle tipo while
contador := 0
for contador < 5 {
    fmt.Println(contador)
    contador++
}
```

Funciones

En Go, una función es un bloque de código reutilizable que recibe parámetros y puede retornar valores. Su estructura es simple:

```go
func nombreFuncion(parámetros) tipoDeRetorno {
    // código
    return valor
}
```

Ejemplo:

```go
func saludar(nombre string) string {
    return "Hola, " + nombre
}

func sumar(a int, b int) int {
    return a + b
}

// Función con múltiples valores de retorno
func dividir(dividendo, divisor float64) (float64, error) {
    if divisor == 0 {
        return 0, fmt.Errorf("No se puede dividir entre cero")
    }
    return dividendo / divisor, nil
}
```

En este ejemplo se definen tres funciones: una que saluda, otra que suma, y una tercera que divide y maneja el error si el divisor es cero` .

Structs y métodos

Un `struct` es una estructura de datos que agrupa distintos campos. Se parece a una clase en otros lenguajes pero sin herencia.

```go
type Persona struct {
    Nombre string
    Edad   int
}
```

Puedes crear métodos para los structs, incluso modificarlos usando punteros.

```go
func (p Persona) Saludar() {
    fmt.Println("Hola, soy", p.Nombre)
}

func (p *Persona) CumplirAnios() {
    p.Edad++
}

func main() {
    p := Persona{Nombre: "Laura", Edad: 30}
    p.Saludar()
    p.CumplirAnios()
    fmt.Println(p.Edad)
}
```

En en ejemplo se crea una persona con el nombre de Laura, la hace saludar y luego incrementa su edad usando un método.

Concurrencias en Go

Go facilita la programación concurrente mediante `goroutines` y `channels`, lo que permite ejecutar tareas en paralelo de forma segura.

Goroutines

Se inician anteponiendo go a una función.

Ejecuta **`decirHola`** en paralelo, pero necesita `Sleep` para dar tiempo a que termine.

```go
func decirHola() {
    fmt.Println("Hola desde una goroutine")
}

func main() {
    go decirHola()
    fmt.Println("Hola desde main")
```

Channels

Se usan para pasar datos entre goroutines.

```go
func saludar(c chan string) {
    c <- "Hola desde canal"
}

func main() {
    canal := make(chan string)
    go saludar(canal)
    mensaje := <-canal
    fmt.Println(mensaje)
}
```

Manejo de errores con Go

Go no usa **`try/catch`.** En su lugar, las funciones retornan un `error` como segundo valor.

Ejemplo

Intenta dividir 10 por 0 y captura el error retornado por la función.

```go
func dividir(a, b int) (int, error) {
    if b == 0 {
        return 0, fmt.Errorf("No se puede dividir entre cero")
    }
    return a / b, nil
}

func main() {
```

Paquetes y Módulos

Los paquetes (`package`) agrupan funciones relacionadas. Todo archivo empieza con **`package nombre`.** Los módulos (`go mod`) gestionan dependencias.

Crear y usar paquete

```go
// saludo/saludo.go
package saludo

func Saludar(nombre string) string {
    return "Hola, " + nombre
}

// main.go
package main

import (
    "fmt"
    "tu_proyecto/saludo"
)

func main() {
    mensaje := saludo.Saludar("Andrés")
    fmt.Println(mensaje)
}
```

Testing básico en Go

Go incluye soporte nativo para pruebas automatizadas con el paquete `testing`.
Las funciones de prueba empiezan con `Test` y se guardan en archivos `*_test.go`.

```go
// saludo_test.go
package saludo

import "testing"

func TestSaludar(t *testing.T) {
    resultado := Saludar("Juan")
    esperado := "Hola, Juan"

    if resultado != esperado {
        t.Errorf("Esperado %s, pero obtuvo %s", esperado, resul
    }
}
```

Prueba que la función **`Saludar`** retorne el mensaje correcto para el nombre "Juan".

Los test se ejecutan asi:

```
go test
```