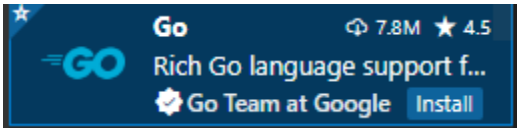
Golang – GO

Lenguaje compilado open source, desarrollado por **Google**

Para chequear si se instaló correctamente, en cmd:

go version

Genera también variables de entorno para un uso más cómodo

Instalar en **Visual Studio Code** la extensión 

➔ Se ejecuta:

Con el comando **go run archivo.go**

En caso de necesitar ejecutar varios archivos

go run archivo1.go archivo2.go

go build archivo.go → para compilar y generar el ejecutable, luego podremos hacer archivo.exe

Si tenemos problemas para ejecutar y depurar desde el visual code, usar este comando

```
go env -w GO111MODULE=auto
```

Lenguaje de tipado estático → los tipos de datos deben estar definidos y mantenerse a lo largo de la ejecución del programa

Tipos de datos numéricos

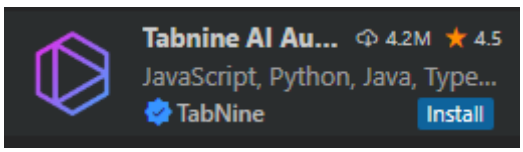
Enteros → **int**

Enteros sin signo (no permite negativos) → **uint**

Flotantes → **float32 / float64**

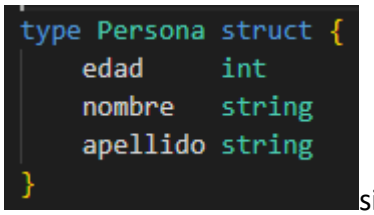
No puedo sumar un entero con un flotante

Extensión de Visual Studio Code que autocompleta el código

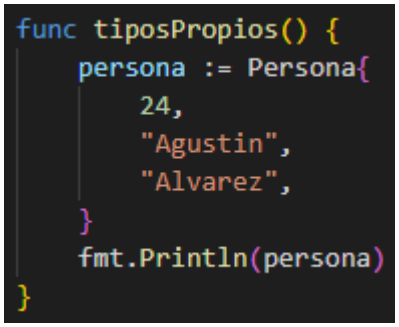


La , me permite concatenar strings con números, el signo + permite solo concatenar entre strings. Además, la coma (,) me agrega un espacio entre los objetos concatenados.

Creación de tipos propios dentro de GO



similar a la definición de una clase en otros lenguajes



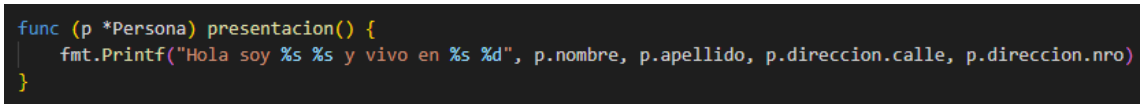
proceso similar a la instanciación de una clase (generación de un objeto de tipo persona)

Podemos incluir mas tipos propios dentro del creado

Dentro de un Printf podemos incluir variables en las comillas de la siguiente forma:

%s → strings %d → int %T → tipo de dato %v → variable

Definición de un método para la clase Persona



Código completo

```go
type Persona struct {
    edad      int
    nombre    string
    apellido  string
    direccion Direccion
}

type Direccion struct {
    calle string
    nro   int
}

func (p *Persona) presentacion() {
    fmt.Printf("Hola soy %s %s y vivo en %s %d", p.nombre, p.apellido, p.direccion.calle, p.direccion.nro)
}

func tiposPropios() {

    dir := Direccion{
        "Belgrano",
        120,
    }
    persona := Persona{
        24,
        "Agustin",
        "Alvarez",
        dir,
    }
    //fmt.Println(persona)
    persona.presentacion()
}
```

Definición de variables

```go
func main() {
    var animal string = "Gato"
    var cadena1, cadena2 string = "hola1", "hola2"

    fmt.Println(animal, cadena1, cadena2)

    var (
        cadena3 string = "hola"
        num     int    = 3
        flag    bool   = false
    )
    fmt.Println(num, cadena3, flag)
}
```

```go
perro := "Miru"
fmt.Printf("%v, %T\n", perro, perro)
```

de esta forma el mismo compilador asume cual es el tipo que posee la variable

Valores por defecto que toman las variables

Numéricos → toman 0 como valor por predeterminado

Booleanos → false

Strings → cadena vacía

Alcance o scope de las variables

Global: todo el código

Local: dentro de una función (cuidado con las llaves)

Constantes

Valores que se van a mantener a lo largo de la ejecución del programa

También tienen un scope, como las variables

Punteros

Ubicación en memoria de una variable

Con le ampersand podemos acceder a ella → &var

Paso por valor → copia del valor

Paso por referencia → ubicación de memoria

Funciones

Función main ( ) es el punto de partida de ejecución del programa, no puede llevar parámetros ni devolver valores.

```
//otra forma de retornar un valor
func retornaAlgo2(precio float32) (final float32) {
    final = precio * 1.75
    return
}
```

```
//puede recibir cualquier cantidad de parametros
func parametrosVariables(nums ...int) {
    fmt.Println(nums, "")
}
```

Funciones lambda → funciones anónimas

```
//funcion lambda
func cuadrado(lado float32) (area func() float32) {
    area = func() float32 {
        return lado * lado
    }
    return
}
```

```go
area := cuadrado(5)
fmt.Println("El area es:", area())
```

Operadores

Comparación → ==, !=, >, <, etc.

Aritméticos → con el modulo Math tendremos mayor cantidad de operadores

Lógicos → &&, ||, !

```go
func mayorEdad(a int) {
    if a > 0 && a < 18 {
        fmt.Println("Eres niño")
    } else if a > 18 && a < 65 {
        fmt.Println("Eres adulto")
    } else if a > 65 {
        fmt.Println("Eres anciano")
    } else {
        fmt.Println("Edad invalida!")
    }
}
```

```go
func edad(a int) {
    switch {
    case a > 0 && a < 18:
        fmt.Println("Eres menor de edad!")
    case a >= 18 && a <= 65:
        fmt.Println("Eres mayor de edad!")
    case a > 65:
        fmt.Println("Eres un anciano!")
    default:
        fmt.Println("Edad no valida!")
    }
}
```

```go
func menu(opcion string) {
    switch opcion {
    case "1":
        fmt.Println("Opcion nro 1!")
    case "2":
        fmt.Println("Opcion nro 2!")
    case "3":
        fmt.Println("Opcion nro 3!")
    default:
        fmt.Println("Opcion no valida!")
    }
}
```

```go
func bucle() {
    var iteraciones int
    fmt.Println("Ingrese el nro de iteraciones:")
    fmt.Scanln(&iteraciones)

    for i := 1; i <= iteraciones; i++ {
        fmt.Println("Iteracion nro ", i)
    }
}
```

```go
func bucle2() {
    var array [10]int

    for i := 0; i < len(array); i++ {
        array[i] = i + 1
    }
    fmt.Println(array)
}
```

Slice

No necesito saber el tamaño al momento de definirlo, le defino un tamaño pero puedo modificarlo fácilmente. No puedo eliminar un elemento del slice, son inmutables. Para eliminar un elemento debo reconstruir el slice

```go
var animales = make([]string, 3)
animales[0] = "gato"
animales[1] = "perro"
animales[2] = "caballo"
fmt.Println(animales)
```

```go
animales = append(animales, "conejo")
```

```go
//borrar elemento
animales = append([]string{}, animales[0], animales[1], animales[3])
```

```
animales2 := animales[0:2]
```
arranca desde el índice 0 y toma dos elementos

```
animales3 := animales[2:]
```
arranca desde el índice 0 hasta el final

```
animales = append(animales2, animales3...)
```

```
animales = append(animales[0:2], animales[3:]...)
fmt.Println(animales)
```

```
//no es una copia, apunta a la misma direccion de memoria
personas2 := personas
fmt.Println(personas2)
personas[0] = "Maria"

fmt.Println(personas2, personas)
```
entonces si modifico un slice también se modifica el otro

```
//esto si realiza una copia, tambien existe una funcion copy()
personas2 := append([]string{}, personas...)
fmt.Println(personas2)
```

Mapas

Accedemos a los elementos a través de sus keys

Tiene una estructura key:value

```
func mapa() {
    var meses = make(map[string]int)
    meses["Enero"] = 1
    meses["Febrero"] = 2
    meses["Marzo"] = 3
    meses["Abril"] = 4
    meses["Mayo"] = 5
    meses["Junio"] = 6

    fmt.Println(meses)
}
```
en este caso es una key de string y un value int. Se puede usar cualquier combinación

```
for key, value := range meses {
    fmt.Printf("El mes %v es el numero %v \n", key, value)
}
```
recorrer un mapa

Podemos agregar fácilmente elementos

```
//eliminar un elemento del mapa
delete(meses, "Marzo")
```

```
//ordenar un mapa
claves := []string{}
for clave := range meses {
    claves = append(claves, clave)
}

sort.Strings(claves)

for _, clave := range claves {
    fmt.Println(clave, meses[clave])
}
```

Estructuras

Campos públicos → accesible desde cualquier parte de la aplicación – mayúsculas. Permite ser exportado

Campos privados → solo se puede acceder desde el propio paquete – minúsculas. No permite ser exportado

Se establece con el primer caracter del nombre de la función, atributo, estructura o lo que fuere.

```
type Persona struct {
    nombre   string
    apellido string
    edad     int
}
```

la estructura es Publica, pero los atributos son privados

Si usamos punteros (&, *) para asignar el valor, si modifico uno se modifica el otro.

```
a2 := &a1
a1.Nombre = "Perro"
fmt.Printf("%v\n", a1)
fmt.Printf("%v\n", *a2)
```

Interfaces → plantilla de métodos para implementarlos en otra clase

```go
type Conexion interface {
    Conectar() string
    Desconectar() string
}

type ConexionSQL struct {
    Puerto int
}

func (con *ConexionSQL) Conectar() string {
    return "Conexion SQL exitosa"
}

func (con *ConexionSQL) Desconectar() string {
    return "Conexion SQL cerrada"
}
```

```go
func DatosConexion(c Conexion) {
    fmt.Println(c.Conectar())
    fmt.Println(c.Desconectar())
}
```

```go
conSQL := ConexionSQL{Puerto: 3306}
conPracle := ConexionOracle{Puerto: 1500}

DatosConexion(&conSQL)
DatosConexion(&conPracle)
```

Practica con strings

Rune → código para indicar algo dentro de un string

```
/*
\n  Nueva línea
\r  Retorno de carro
\t  Tabulador horizontal
\v  Tabulador vertical
\   Diagonal invertida
'   Coma sencilla
\\" Comillas dobles
*/
```

Texto multilínea

```
func textoMulti() {
    texto := `Me llamo Agustin
    tengo 23 años
    Juego futbol`
    fmt.Println(texto)
}
```

Concatenar strings → +

```
func concatenarStrings() {
    prueba := "Me llamo Agustin \n"
    prueba += "tengo 30 años"
    fmt.Println(prueba)

}
```

```
PS D:\CURSOS\Development\Golang\practica_strings> go run .\practica_strings.go
Me llamo Agustin
tengo 30 años
```

Librerías como strings, strconv, fmt

Conversión de tipos

```
func convertir() {
    prueba := "Soy el numero " + strconv.Itoa(23)
    fmt.Println(prueba)
}
```

convierte de int a string

FormatInt → convierte a binario, Hexa, base10, etc.

fmt.Sprint() → convierte a string

```
func metodos() {
    prueba := "Hola soy Agustin"

    if strings.Contains(prueba, "soy") {
        fmt.Println("Contiene la palabra soy")
    } else {
        fmt.Println("No contiene la palabra soy")
    }
}
```

es sensitive case

compare(“cadena1”, “cadena2”) → compara las cadenas y da 0, 1 o -1 según resultado

```go
pruebaMay := strings.ToUpper(prueba)
fmt.Println(pruebaMay)

pruebaMin := strings.ToLower(prueba)
fmt.Println(pruebaMin)
```

```go
func busqueda() {
    prueba := "Hola soy Agustin"
    pos := strings.Index(prueba, "Agus")
    fmt.Println(pos)
}
```
devuelve -1 si no encuentra

```go
func ejercicio3() {
    cadena := "la mosca aparecio"
    ult := strings.LastIndex(cadena, "a")
    fmt.Println(ult)
}
```

```go
func borrarEspacios() {
    prueba := " Hola soy Agustin "
    fmt.Println(strings.TrimSpace(prueba))
}
```

```go
array := strings.Split(prueba, " ")
fmt.Println(array[1])
```

Errores en Go → un error viene como último retorno de la función que lo generó

Manejo de errores

```go
import (
    "fmt"
    "io/ioutil"
)

// vamos a leer un fichero y lanzar un error si no lo encuentra
func main() {
    texto, err := ioutil.ReadFile("prueba.txt")
    //si el error es distinto de null
    if err != nil {
        fmt.Println("Error al leer el archivo!")
        return
    }
    //pasar a string para que muestre el texto como esta en el documento
    fmt.Println(string(texto))
}
```

Tipo error

```go
if err != nil {
    fmt.Println(err)
    err2 := errors.New("Archivo no encontrado jeje")
    fmt.Println(err2)
    return
}
```

creamos un nuevo objeto de tipo error que nos sea útil. Necesitamos importar la librería “errors”

```go
func sumaConError(num1 int, num2 int) (int, error) {
    if num1 > 100 || num2 > 100 {
        err := fmt.Errorf("No ingresar numeros mayores a 100")
        //retorno 0 como resultado y el error
        return 0, err
    }
    //retorno el resultado y null como error
    return num1 + num2, nil
}
```

Panic

Función en GO que nos permite detener por completo el flujo de ejecución del programa

```go
func finEjecucion() {
    fmt.Println("Hasta aqui llega la ejecucion")
    panic("Paro la ejecucion")
}
```

Concurrencia

Ejecutar en orden parcial, de a pedazos, en desorden

Funciones y bloqueos

```go
import (
    "fmt"
    "time"
)

func main() {
    res := procesoLargo()
    fmt.Println(res)
}

func procesoLargo() string {
    //va a tardar 10 segundos en continuar la ejecucion
    time.Sleep(time.Second * 10)
    return "pasaron 10 segundos"
}
```

Go-rutinas

Ejecutar de forma concurrente varios procesos

```go
func main() {
    //go rutina que retorna algun valor
    go func() {
        res := procesoLargo()
        fmt.Println(res)
    }()
    //go rutina que no devuelve ningun valor
    go otroProcesoLargo()
    time.Sleep(time.Second * 10)
    fmt.Println("Fin del programa!")
}
```

Canales / Channels

Permiten a los datos moverse dentro y fuera de las rutinas y facilitar la comunicación entre ellas. En GO se comparte la memoria para esto

```go
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum // enviar sum al canal
}

func main() {
    s := []int{7, 2, 8, -9, 4, 0}

    c := make(chan int)
    go sum(s[:len(s)/2], c)
    go sum(s[len(s)/2:], c)
    x, y := <-c, <-c // recepcion del canal

    fmt.Println(x, y, x+y)
}
```

Buffer

Mantiene el canal abierto hasta que el receptor está disponible

```go
func EjemploBuffer() {
    //en el segundo parametro ponemos el tamaño que tendra el buffer
    canal := make(chan string, 1)
    //no podria almacenar otro valor en el buffer, xq el tamaño asignado fue de 1
    canal <- "Agustin"
    fmt.Println(<-canal)
    //en este punto ya esta liberado el buffer y ahi si podriamos
    //asignarle otro valor
    canal <- "Alvarez"
    fmt.Println(<-canal)

}
```

Iterar un canal

Recorrer los valores que va tomando el canal. Para que no de un mensaje de warning debemos cerrar el canal.

```go
func EjemploBuffer2() {
    canal := make(chan string, 3)
    canal <- "Agustin"
    canal <- "Ezequiel"
    canal <- "Alvarez"
    close(canal)
    //iterar los valores que va tomando el canal
    for c := range canal {
        fmt.Println(c)
    }
}
```

Podemos definir canales como de solo lectura, solo escritura o lectura escritura

```go
func CanalesReadWrite(c chan string) {
    fmt.Println(<-c)
    c <- "Agus"
    close(c)
    for i := range c {
        fmt.Println(i)
    }
}

func CanalesOnlyRead(c <-chan string) {
    fmt.Println(<-c)
    for i := range c {
        fmt.Println(i)
    }
}
```

```go
func CanalesOnlyWrite(c chan<- string) {
    c <- "Agus"
}
```

Canales múltiples y timeouts

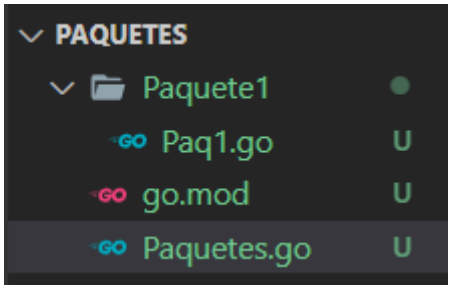
Permite por ejemplo que podamos capturar al primer proceso que termina de ejecutarse. Además, el timeout permite ponerle un limite de tiempo a una rutina

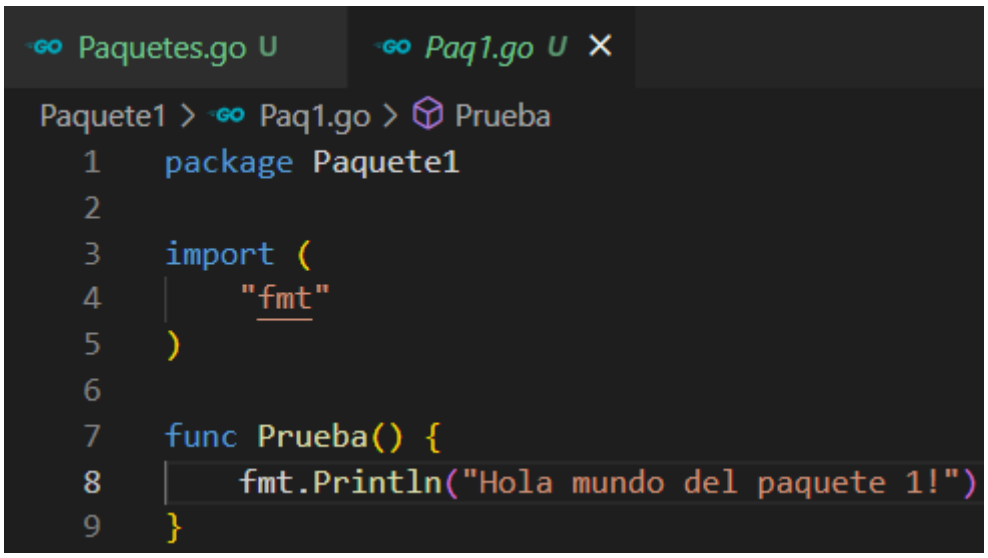
```go
//creamos los 3 canales
canal1 := make(chan string, 1)
canal2 := make(chan string, 1)
canal3 := make(chan string, 1)
//llamamos a las go-rutinas para que se ejecuten
//al mismo tiempo las 3 funciones
go Random1(canal1)
go Random2(canal2)
go Random3(canal3)
//sentencia select para que tome al proceso
//que termino primero
select {
case ganador := <-canal1:
    fmt.Println(ganador)
case ganador := <-canal2:
    fmt.Println(ganador)
case ganador := <-canal3:
    fmt.Println(ganador)
case <-time.After(time.Second * 50):
    fmt.Println("Erorr timeout...")
}
```

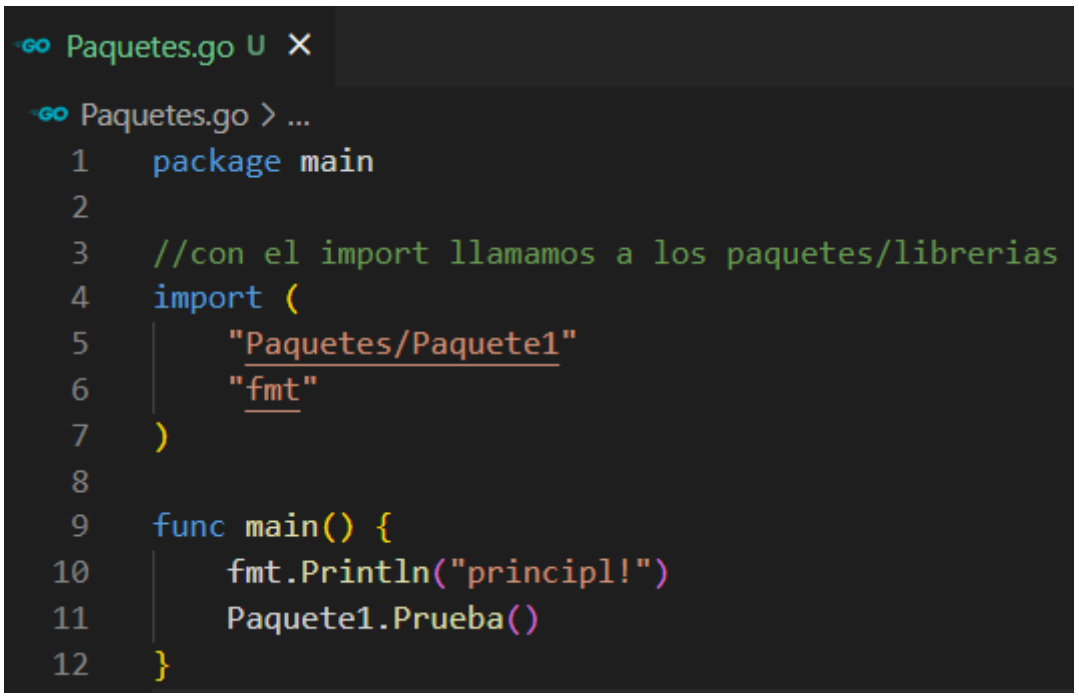
Creación de paquetes

En consola:

go mod init “Paquetes” → esto nos va a permitir importar los paquetes y además crea el archivo mod.go, el cual tiene las referencias necesarias







Public y Private

Mayúsculas → accesible desde todos los paquetes

Minúsculas → accesible desde el paquete actual únicamente

*haciendo referencia a la primera letra del elemento

Paquetes.go U PublicPrivate.go U

PublicPrivate > PublicPrivate.go > ...

```go
package PublicPrivate

import "fmt"

func privada() {
    //esta funcion no puede ser accedida desde
    //otro paquete
}

func Publica() {
    //esta funcion si puede ser accedida
    //desde otro paquete
    fmt.Println("Funcion publica del paquete PublicPrivate!")
}

```

```go
type Persona2 struct {
    nombre   string
    apellido string
}
```

```go
func CrearPersona2() Persona2 {
    return Persona2{}
}
```

```go
// necesitamos ubicar el puntero del objeto persona2
func (p *Persona2) SetNombre(nombre string) {
    p.nombre = nombre
}

func (p *Persona2) SetApellido(apellido string) {
    p.apellido = apellido
}

func (p *Persona2) GetNombre() string {
    return p.nombre
}
func (p *Persona2) GetApellido() string {
    return p.apellido
}
```

```go
persona2 := PublicPrivate.CrearPersona2()
persona2.SetNombre("Agustin")
persona2.SetApellido("Alvarez")
fmt.Println(persona2.GetNombre())
fmt.Println(persona2.GetApellido())
```
→ llamada desde main

Logs

Package log

Permite registrar lo que pasa en el sistema

logs.go > main

```go
package main

import (
    "fmt"
    "log"
)

func main() {
    fmt.Println("Hola mundo fmt!")
    log.Println("Hola mundo log!")
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS D:\CURSOS\Development\Golang\logs> go run .\logs.go
Hola mundo fmt!
2022/11/02 09:17:49 Hola mundo log!
```

```go
func main() {
    //muestra el mensaje añadiendo fecha y hora
    log.Println("Hola mundo log!")
    //muestra el mensaje y detiene ejecucion
    log.Fatal("Error fatal!")

    err := errors.New("Error de prueba!")
    log.Println(err)
}
```

Usar archivos como logs

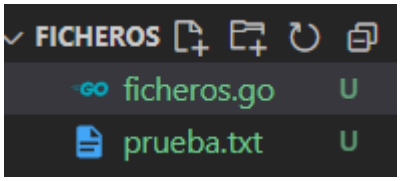
```go
//escribir log en un archivo, en caso de no existir lo crea
f, err := os.OpenFile("Prueba.log", os.O_APPEND|os.O_CREATE|os.O_RDWR, 0666)
if err != nil {
    log.Fatal(err)
}

//defer se utiliza para ejecutar algo una vez que main acabo
defer f.Close()

log.SetOutput(f)

for i := 0; i <= 50; i++ {
    log.Printf("Pueba escritura en log %v", i)
}
```

Ficheros / Archivos

➔ Paquete **ioutil**

Leer un fichero

```go
func leer() {
    ficheroArray, err := ioutil.ReadFile("prueba.txt")
    if err != nil {
        log.Fatal(err)
    }
    contenido := string(ficheroArray)
    fmt.Println(contenido)
}
```



Escribir un fichero

```go
func escribir() {
    arrayBytes := []byte("Me llamo Agustin!")
    err := ioutil.WriteFile("prueba2.txt", arrayBytes, 0664)
    if err != nil {
        log.Fatal(err)
    }
}
```

Si el archivo no existe, lo crea

Listar el contenido de un directorio

```go
func listarDir() {
    listaFicheros, err := ioutil.ReadDir(".")
    if err != nil {
        log.Fatal(err)
    }
    for _, f := range listaFicheros {
        fmt.Println("Nombre archivo:", f.Name())
        fmt.Println("Tamaño:", f.Size(), "bytes")
        fmt.Println("Es directorio?", f.IsDir())
    }

}
```

Copiar un fichero

Paquete **os**

```go
func copiar() {
    fichero, err := os.Open("prueba.txt")
    if err != nil {
        log.Fatal(err)
    }
    defer fichero.Close()

    copiaFichero, err := os.OpenFile("copia.txt", os.O_RDWR|os.O_CREATE, 0666)
    if err != nil {
        log.Fatal(err)
    }
    defer copiaFichero.Close()

    resultado, err := io.Copy(copiaFichero, fichero)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println(resultado)
}
```

Borrar ficheros

```go
func borrar() {
    err := os.Remove("copia.txt")
    if err != nil {
        log.Fatal(err)
    }
}
```

Clientes HTTP

Net/http → libreria

GET

```go
func main() {
    clienteHttp := &http.Client{}
    url := "https://httpbin.org/get"
    peticion, err := http.NewRequest("GET", url, nil)
    if err != nil {
        log.Fatalf("Error %v", err)
    }

    peticion.Header.Add("X-Prueba", "Ejemplo")
    respuesta, err := clienteHttp.Do(peticion)
    if err != nil {
        log.Fatalf("Error %v", err)
    }
    defer respuesta.Body.Close()

    respuestaBytes, err := ioutil.ReadAll(respuesta.Body)
    if err != nil {
        log.Fatalf("Error %v", err)
    }

    respuestaString := string(respuestaBytes)
    log.Printf("Response code: %d", respuesta.StatusCode)
    log.Printf("Response body: '%s'", respuestaString)
}
```

```
PS D:\CURSOS\Development\Golang\clientes_http> go run .\peticiones.go
2022/11/08 10:37:20 Response code: 200
2022/11/08 10:37:20 Response body: '{
  "args": {},
  "headers": {
    "Accept-Encoding": "gzip",
    "Host": "httpbin.org",
    "User-Agent": "Go-http-client/2.0",
    "X-Amzn-Trace-Id": "Root=1-636a5b91-09561e3d43bd35993e5b5741",
    "X-Prueba": "Ejemplo"
  },
  "origin": "190.210.1.73",
  "url": "https://httpbin.org/get"
}
```

POST (uso de librería encoding/json)

```go
func post() {
    clienteHttp := &http.Client{}
    url := "https://httpbin.org/post"

    type Animal struct {
        Nombre  string
        Especie string
    }
    ani := Animal{
        Nombre:  "Miru",
        Especie: "Perro",
    }

    aniJson, err := json.Marshal(ani)
    if err != nil {
        log.Fatalf("Error con el JSON %v", err)
    }

    peticion, err := http.NewRequest("POST", url, bytes.NewBuffer(aniJson))
    if err != nil {
        log.Fatalf("Error en peticion %v", err)
    }
```

```go
    peticion, err := http.NewRequest("POST", url, bytes.NewBuffer(aniJson))
    if err != nil {
        log.Fatalf("Error en peticion %v", err)
    }

    peticion.Header.Add("Content-Type", "application/json")
    respuesta, err := clienteHttp.Do(peticion)
    if err != nil {
        log.Fatalf("Error en peticion %v", err)
    }
    defer respuesta.Body.Close()
    body, err := ioutil.ReadAll(respuesta.Body)
    if err != nil {
        log.Fatalf("Error %v", err)
    }

    res := string(body)
    log.Printf("Response code: %d", respuesta.StatusCode)
    log.Printf("Headers: '%q'", respuesta.Header)
    contentType := respuesta.Header.Get("Content-Type")
    log.Printf("Content-type: '%s'", contentType)
    log.Printf("Response body: '%s'", res)
}
```

```
2022/11/08 10:51:51 Response code: 200
2022/11/08 10:51:51 Headers: 'map["Access-Control-Allow-Credentials":["true"] "Access-Control-Allow-Origin":["*"] "C
ontent-Length":["495"] "Content-Type":["application/json"] "Date":["Tue, 08 Nov 2022 13:51:52 GMT"] "Server":["gunic
orn/19.9.0"]]'
2022/11/08 10:51:51 Content-type: 'application/json'
2022/11/08 10:51:51 Response body: '{
  "args": {},
  "data": "{\"Nombre\":\"Miru\",\"Especie\":\"Perro\"}",
  "files": {},
  "form": {},
  "headers": {
    "Accept-Encoding": "gzip",
    "Content-Length": "35",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "Go-http-client/2.0",
    "X-Amzn-Trace-Id": "Root=1-636a5ef8-722820560dea8b2c3dbeee5f"
  },
  "json": {
    "Especie": "Perro",
    "Nombre": "Miru"
  },
  "origin": "190.210.1.73",
  "url": "https://httpbin.org/post"
}
```

PUT

```go
func put() {
    clienteHttp := &http.Client{}
    url := "https://httpbin.org/put"

    type Animal struct {
        Nombre  string
        Especie string
    }
    ani := Animal{
        Nombre:  "Miru",
        Especie: "Perro",
    }

    aniJson, err := json.Marshal(ani)
    if err != nil {
        log.Fatalf("Error con el JSON %v", err)
    }

    peticion, err := http.NewRequest("PUT", url, bytes.NewBuffer(aniJson))
    if err != nil {
        log.Fatalf("Error en peticion %v", err)
    }
```

```go
    peticion.Header.Add("Content-Type", "application/json")
    respuesta, err := clienteHttp.Do(peticion)
    if err != nil {
        log.Fatalf("Error en peticion %v", err)
    }
    defer respuesta.Body.Close()
    body, err := ioutil.ReadAll(respuesta.Body)
    if err != nil {
        log.Fatalf("Error %v", err)
    }

    res := string(body)
    log.Printf("Response code: %d", respuesta.StatusCode)
    log.Printf("Headers: '%q'", respuesta.Header)
    contentType := respuesta.Header.Get("Content-Type")
    log.Printf("Content-type: '%s'", contentType)
    log.Printf("Response body: '%s'", res)
}
```

```
PS D:\CURSOS\Development\Golang\clientes_http> go run .\peticiones.go
2022/11/15 12:08:25 Response code: 200
2022/11/15 12:08:25 Headers: 'map["Access-Control-Allow-Credentials":["true"] "Access-Control-Allow-Origin":["*"]
 "Content-Length":["494"] "Content-Type":["application/json"] "Date":["Tue, 15 Nov 2022 15:08:34 GMT"] "Server":[
"gunicorn/19.9.0"]]'
2022/11/15 12:08:25 Content-type: 'application/json'
2022/11/15 12:08:25 Response body: '{
  "args": {},
  "data": "{\"Nombre\":\"Miru\",\"Especie\":\"Perro\"}",
  "files": {},
  "form": {},
  "headers": {
    "Accept-Encoding": "gzip",
    "Content-Length": "35",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "Go-http-client/2.0",
    "X-Amzn-Trace-Id": "Root=1-6373ab72-7b5429a304fd4790646cac51"
  },
  "json": {
    "Especie": "Perro",
    "Nombre": "Miru"
  },
  "origin": "190.210.1.73",
  "url": "https://httpbin.org/put"
}
```

DELETE

```go
func delete() {
    clienteHttp := &http.Client{}
    url := "https://httpbin.org/delete"

    type Animal struct {
        Nombre  string
        Especie string
    }
    ani := Animal{
        Nombre:  "Miru",
        Especie: "Perro",
    }

    aniJson, err := json.Marshal(ani)
    if err != nil {
        log.Fatalf("Error con el JSON %v", err)
    }
    peticion, err := http.NewRequest("DELETE", url, bytes.NewBuffer(aniJson))
    if err != nil {
        log.Fatalf("Error en peticion %v", err)
    }
    peticion.Header.Add("Content-Type", "application/json")
    respuesta, err := clienteHttp.Do(peticion)
    if err != nil {
        log.Fatalf("Error en peticion %v", err)
    }
    defer respuesta.Body.Close()
```

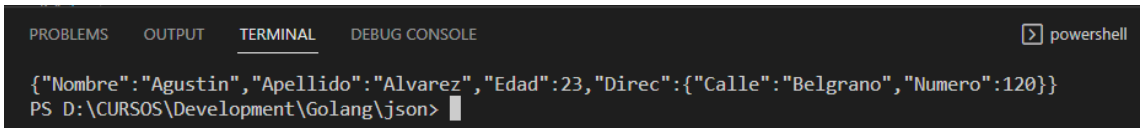
```go
    body, err := ioutil.ReadAll(respuesta.Body)
    if err != nil {
        log.Fatalf("Error %v", err)
    }

    res := string(body)
    log.Printf("Response code: %d", respuesta.StatusCode)
    log.Printf("Headers: '%q'", respuesta.Header)
    contentType := respuesta.Header.Get("Content-Type")
    log.Printf("Content-type: '%s'", contentType)
    log.Printf("Response body: '%s'", res)
}
```

```
PS D:\CURSOS\Development\Golang\clientes_http> go run .\peticiones.go
2022/11/15 12:16:44 Response code: 200
2022/11/15 12:16:44 Headers: 'map["Access-Control-Allow-Credentials":["true"] "Access-Control-Allow-Origin":["*"] "C
ontent-Length":["497"] "Content-Type":["application/json"] "Date":["Tue, 15 Nov 2022 15:16:53 GMT"] "Server":["gunic
orn/19.9.0"]]'
2022/11/15 12:16:44 Content-type: 'application/json'
2022/11/15 12:16:44 Response body: '{
  "args": {},
  "data": "{\"Nombre\":\"Miru\",\"Especie\":\"Perro\"}",
  "files": {},
  "form": {},
  "headers": {
    "Accept-Encoding": "gzip",
    "Content-Length": "35",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "Go-http-client/2.0",
    "X-Amzn-Trace-Id": "Root=1-6373ad65-73e2821855a45d016e63198d"
  },
  "json": {
    "Especie": "Perro",
    "Nombre": "Miru"
  },
  "origin": "190.210.1.73",
  "url": "https://httpbin.org/delete"
}
```

JSON

Uso de JSON en golang

```go
import (
    "encoding/json"
    "fmt"
    "log"
)

func main() {
    dir := Direccion{
        Calle:  "Belgrano",
        Numero: 120,
    }

    per := Persona{
        Nombre:   "Agustin",
        Apellido: "Alvarez",
        Edad:     23,
        Direc:    dir,
    }

    dataJson, err := json.Marshal(per)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(string(dataJson))
}
```



Data fields

```go
type Mascota struct {
    Nombre  string   `json:"nombre"`
    Edad    int      `json:"edad,omitempty"`
    Especie string   `json:"especie,omitempty"`
    Vacunas []string `json:"vacunas"`
}

func DatosMascota() {
    m := Mascota{
        Nombre:  "Miranda",
        Edad:    4,
        Especie: "Bulldog Frances",
        Vacunas: []string{"Rabia", "Parbovirus"},
    }
    dataJson, err := json.Marshal(m)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(string(dataJson))
}
```

Decodificar JSON

```go
func DecodeJson() {
    datosJson := `{"Nombre":"Agustin","Apellido":"Alvarez","Edad":23,"Direc":{"Calle":"Belgrano",
    per := Persona{}
    err := json.Unmarshal([]byte(datosJson), &per)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Printf("%+v\n", per)
}
```

Mapeo de estructuras

JSON	Go
Null	nil
Boolean	bool
Number	float64
String	string
Array	[]interface{}
Object	map[string]interface{}