

# GOLANG LANGUAGE

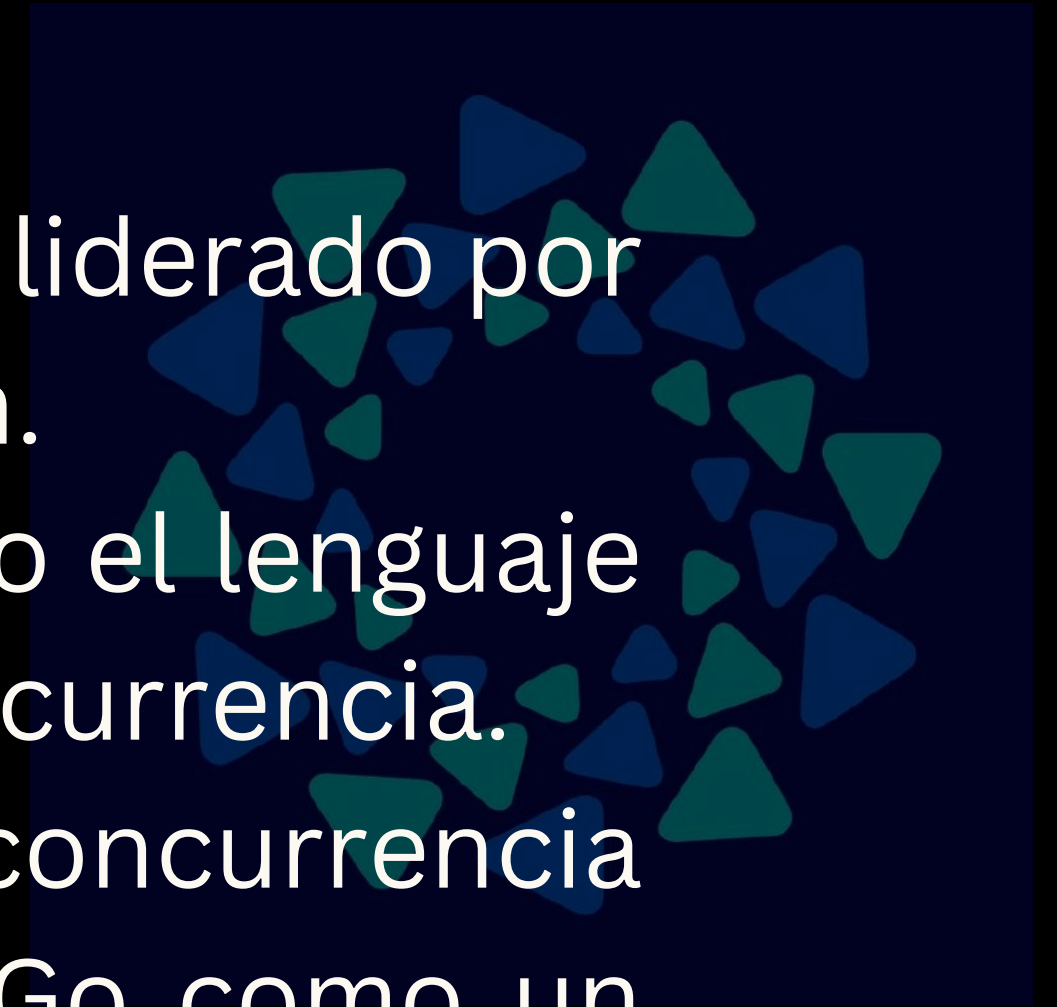
Jeyson Camilo Aguilar Tarazona  
Jimmy D'Franco Velandia Orozco  
Nicolas Oswaldo Florez Duarte  
Sebastián Zarate Rojas

1152459  
1152468  
1152474  
1152456



# UN POCO DE SU HISTORIA...

- **2007:** Creación del proyecto Go por Google, liderado por Robert Griesemer, Rob Pike y Ken Thompson.
- **2009:** Lanzamiento de Go 1.0, estableciendo el lenguaje con enfoque en simplicidad, eficiencia y concurrencia.
- **2012:** Mejora significativa en el manejo de concurrencia con goroutines y canales, consolidando a Go como un lenguaje ideal para aplicaciones concurrentes.
- **2015:** Go 1.5 elimina la dependencia del C, mejorando la portabilidad y la independencia del lenguaje.
- **2022:** Go 1.18 introduce el soporte para generics, ampliando las capacidades y flexibilidad del lenguaje.



# ¿Cómo iniciar un proyecto?

package Main

import

func main







# Tipos, declaración e inicialización de variables

```
1  var entero int
2  var booleano bool
3  var flotante float32
4  var cadenas string
5  var unidad uint
6  var (
7      a int
8      b string
9  )
10 var (c, d string)
11 var (f int; g string)
12
```

```
1  func main() {
2      var entero int = 0
3      var booleano bool = true
4      var flotante float32 = 0.0
5      var cadenas string = "Hola mundo"
6      var unidad uint = 1
7      var (
8          a int      = 0
9          b string = "b"
10     )
11     c := 0
12     d, e, f, g := 1, false, "Adios Mundo", 3.14
13
14     fmt.Println(entero, booleano, flotante, cadenas,
15                 unidad, a, b, c, d, e, f, g)
16 }
```

# Condicionales, ciclos y funciones

Solo está el ciclo for,  
para declarar una  
función que retorna un  
valor: nombre +  
parámetros + tipo de  
dato de retorno

```
1 func sumar(a, b int) {  
2     fmt.Println(a + b)  
3 }  
4 func multiplicar(a int, b int) int{  
5     return (a + b)  
6 }  
7 func main() {  
8     sumar(1, 2)  
9     if a > f {  
10        fmt.Println("Es verdadero")  
11    }else {  
12        fmt.Println("Es Falso")  
13    }  
14    for i := 0; i < 10; i++ {  
15        fmt.Println(i)  
16    }  
17 }
```



# “CLASES” Y “OBJETOS”

## ESTRUCTURAS CON PROPIEDADES

```
1 type persona struct {  
2     nombre string  
3     edad int  
4 }  
5  
6 func main() {  
7     var p persona  
8     fmt.Scanln(&p.nombre)  
9     fmt.Println(p)  
10    juan := persona{nombre: "juan"}  
11    carlos := persona{"carlos", 17}  
12    fmt.Println(juan, carlos)  
13 }
```

## MÉTODOS DE ESTRUCTURAS

```
1 func (p persona) cambiarNombre() {  
2     p.nombre = "larry"  
3     fmt.Println(p)  
4 }  
5 func (p *persona) cambiarEdad() {  
6     p.edad = 6  
7     fmt.Println(p)  
8 }  
9 func main() {  
10    p := persona{nombre: "pepito", edad: 2}  
11    q := persona{nombre: "alberto", edad: 3}  
12    p.cambiarNombre()  
13    r := persona{nombre: "felipe", edad: 4}  
14    fmt.Println(p, q, r)  
15    p.cambiarEdad()  
16 }
```

```
{larry 2}  
{pepito 2} {alberto 3} {felipe 4}  
&{pepito 6}
```

# ENCAPSULAMIENTO

```
● ● ●  
1  var nombre string  
2  
3  type persona struct {  
4      nombre string  
5      edad  int  
6  }  
7  
8  func cambiarNombre() {  
9      nombre = "larry"  
10 }
```

Esto es privado

```
● ● ●  
1  var Nombre string  
2  
3  type Persona struct {  
4      nombre string  
5      edad  int  
6  }  
7  
8  func CambiarNombre() {  
9      Nombre = "larry"  
10 }
```

Esto es público

# Contenedores

- Array
- Slice
- Map
- List
- Struct

```
var numeros [5]int // Array de tamaño fijo
numeros[0] = 10
numeros[1] = 20
numeros[2] = 30
```

```
edades := map[string]int{
    "Juan": 25,
    "Ana": 30,
    "Carlos": 35,
}

edades["María"] = 28 // Agregar una nueva clave-valor
```



# Asociación

relación débil

no necesariamente depende

En Go, esto se modela comúnmente pasando referencias (punteros) de una estructura a otra.

```
type Empleado struct {
    Nombre string
}

type Proyecto struct {
    Nombre    string
    Empleados []*Empleado // Relación asociativa con Empl
}

func main() {
    empleado := &Empleado{Nombre: "Juan"}
    proyecto := Proyecto{
        Nombre:    "Desarrollo Web",
        Empleados: []*Empleado{empleado},
    }
}
```

# Agregación

forma más específica de asociación

una estructura (el "todo") contiene referencias a otras estructuras (las "partes")

En Go, esto se modela comúnmente pasando referencias (punteros) de una estructura a otra.

```
type Departamento struct {  
    Nombre string  
}  
  
type Empresa struct {  
    Nombre      string  
    Departamentos []*Departamento // Relación de agregación  
}  
  
func main() {  
    departamento := &Departamento{Nombre: "IT"}  
    empresa := Empresa{  
        Nombre:      "Tech Corp",  
        Departamentos: []*Departamento{departamento},  
    }  
}
```



# Composición

relación más fuerte

las partes son componentes esenciales de un todo.

embebido de structs

```
type Direccion struct {  
    Calle string  
    Ciudad string  
}  
  
type Persona struct {  
    Nombre string  
    Direccion Direccion // Relación de composición  
}  
  
func main() {  
    persona := Persona{  
        Nombre: "Ana",  
        Direccion: Direccion{  
            Calle: "Calle 123",  
            Ciudad: "Bogotá",  
        },  
    }  
}
```



# Herencia

Campos anónimos

Sobreescritura de métodos

```
1 package main
2
3 import "fmt"
4
5 // Estructura base
6 type Persona struct {
7     Nombre string
8     Edad   int
9 }
10
11 // Método asociado a Persona
12 func (p Persona) Saludar() {
13     fmt.Printf("Hola, soy %s y tengo %d años.\n", p.Nombre, p.Edad)
14 }
15
16 // Estructura que "hereda" de Persona
17 type Empleado struct {
18     Persona      // Composición: Persona está incluida en Empleado
19     Puesto string // Campo adicional
20 }
21
22 func main() {
23     // Crear un empleado
24     empleado := Empleado{
25         Persona: Persona{
26             Nombre: "Ana",
27             Edad:   30,
28         },
29         Puesto: "Desarrolladora",
30     }
31
32     // Acceder directamente a los métodos de Persona desde Empleado
33     empleado.Saludar()
34     fmt.Printf("Trabajo como %s.\n", empleado.Puesto)
35 }
```

# Polimorfismo

- Interfaces
- Relación implícita
- Interfaz vacía

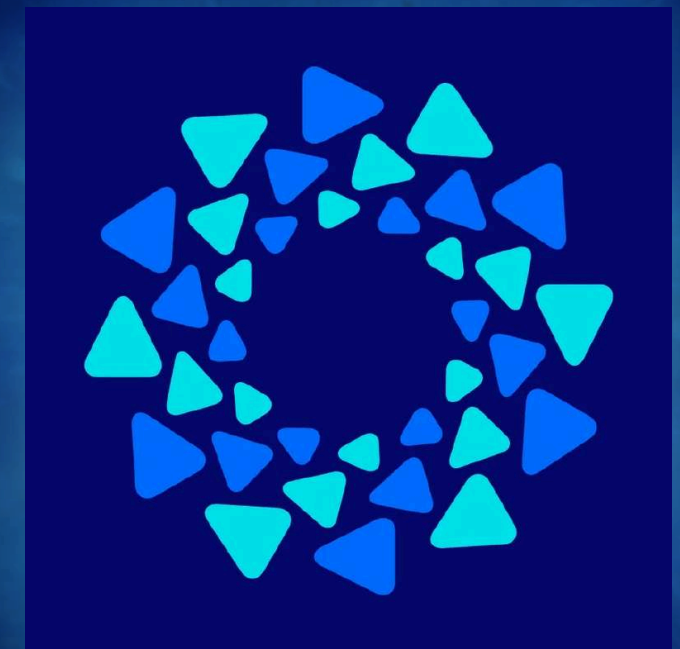
```
1 package main
2
3 import "fmt"
4
5 // Definimos una interfaz
6 type Hablador interface {
7     Hablar() string
8 }
9
10 // Structs que implementan la interfaz
11 type Persona struct {
12     Nombre string
13 }
14
15 func (p Persona) Hablar() string {
16     return "Hola, soy " + p.Nombre
17 }
18
19 type Perro struct {
20     Nombre string
21 }
22
23 func (p Perro) Hablar() string {
24     return "Guau, soy " + p.Nombre
25 }
26
27 func main() {
28     var habladores []Hablador = []Hablador{
29         Persona{Nombre: "Ana"},
30         Perro{Nombre: "Fido"},
31     }
32
33     // Iteramos sobre diferentes tipos usando la interfaz
34     for _, hablador := range habladores {
35         fmt.Println(hablador.Hablar())
36     }
37 }
38
```



# ¿Que es Fyne?

**Fyne es un framework de código abierto para desarrollar aplicaciones gráficas de usuario (GUI) en Go (Golang).**

**Permite crear aplicaciones multiplataforma: Windows, macOS, Linux, iOS y Android.**



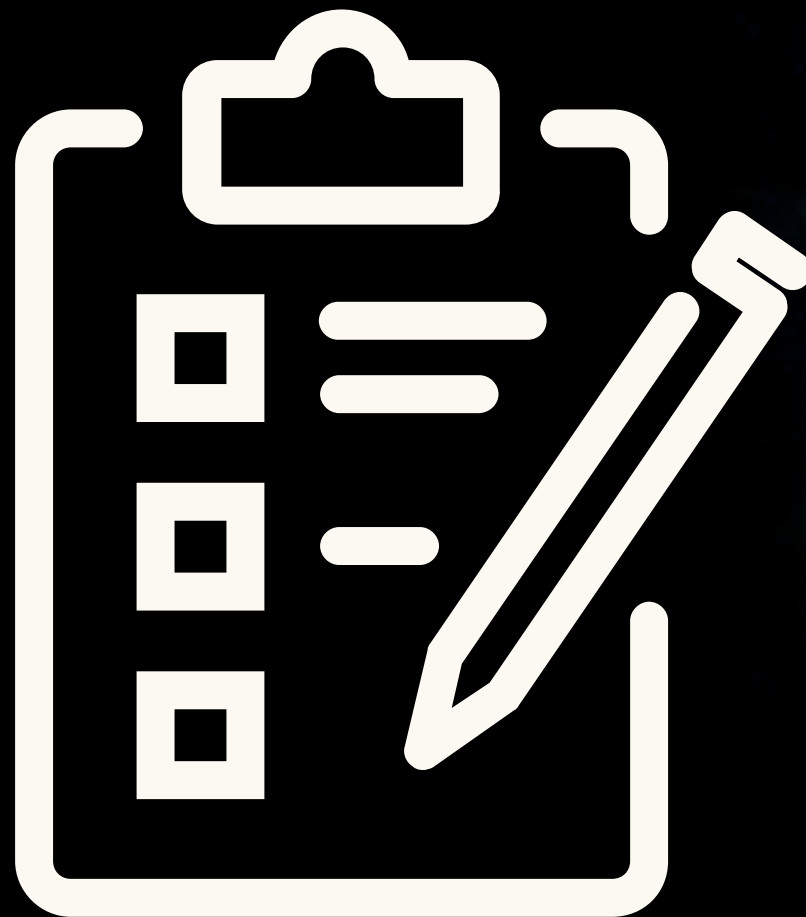


# Características Principales

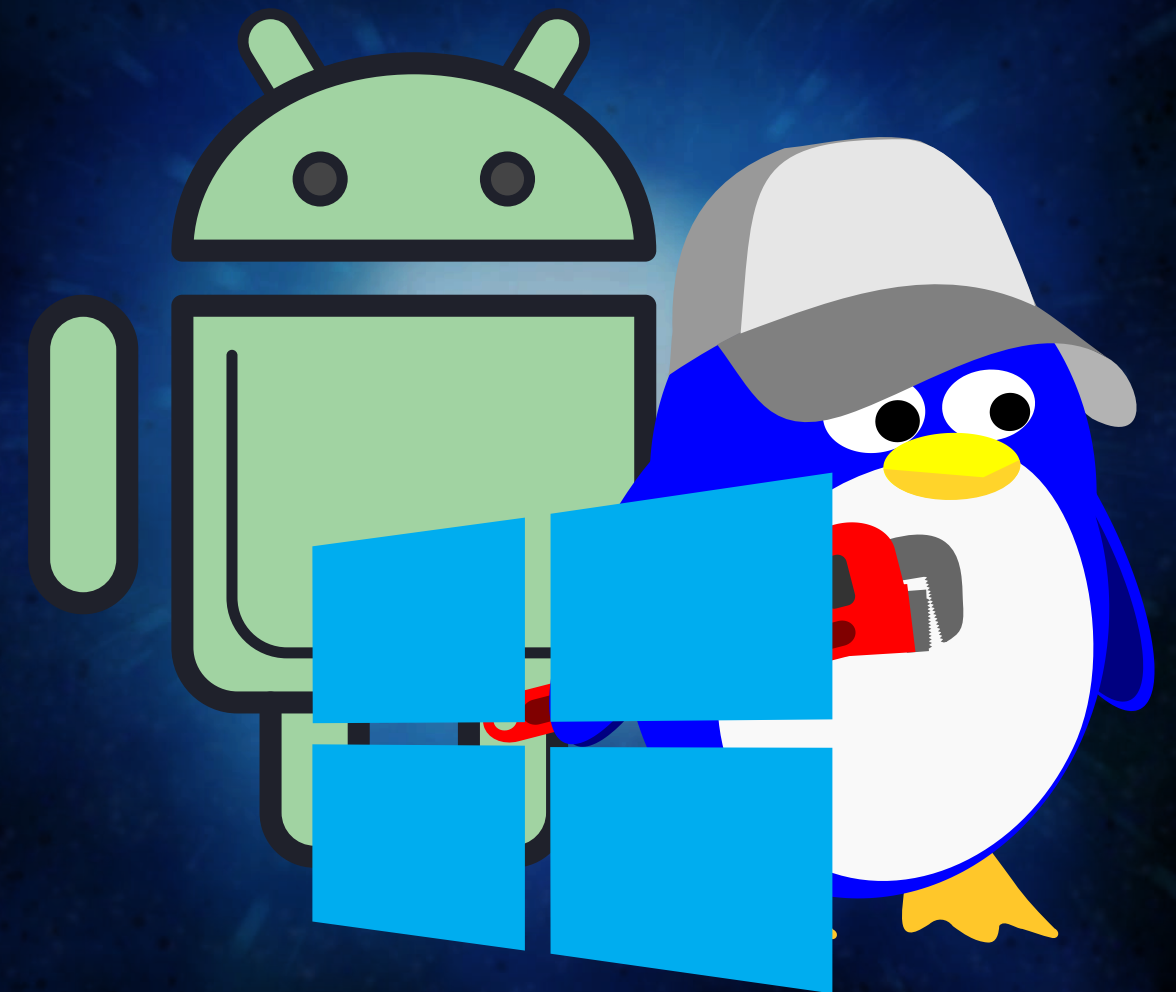
Rendimiento  
eficiente



Widgets



Multiplataforma



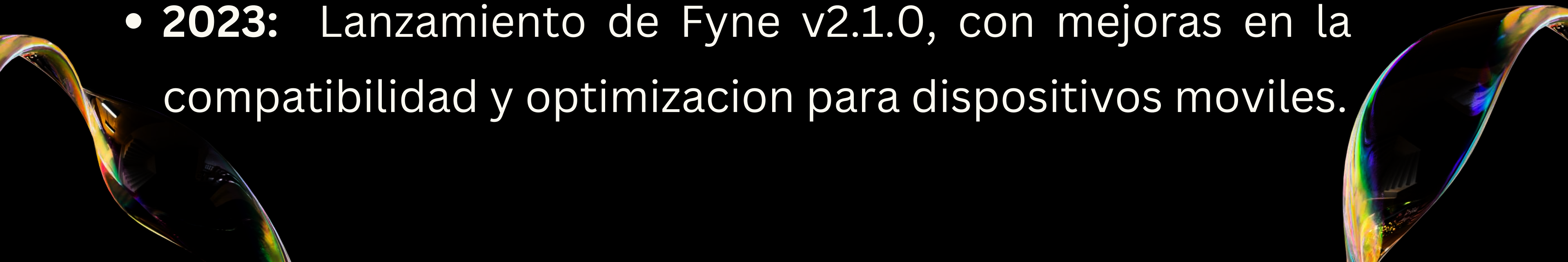
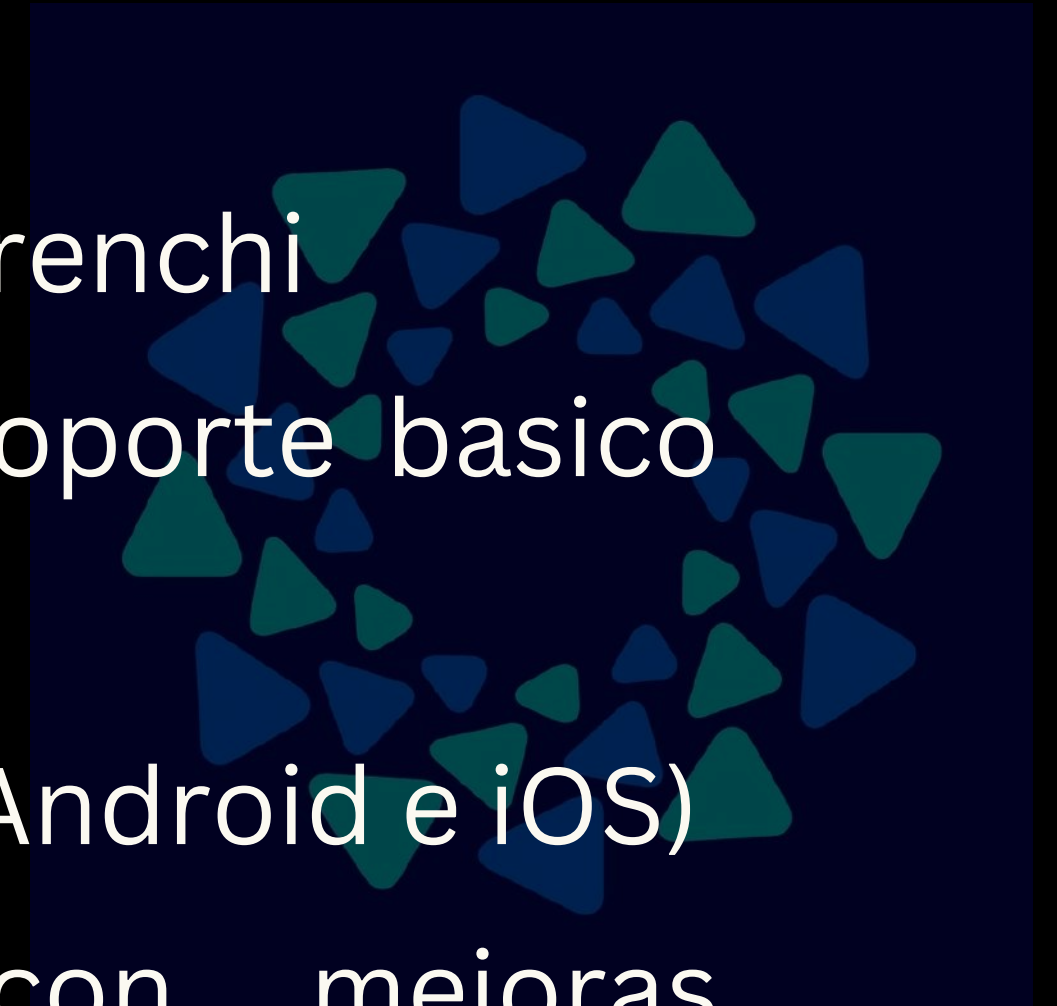
## Ultimas Versiones

**Versión más reciente: v2.1.0 (lanzada en 2023).**

- Mejoras en rendimiento y compatibilidad móvil.
- Nuevos widgets y mejoras en diseño responsivo.
- Actualizaciones de dependencias y soporte de plataformas.

# UN POCO DE SU HISTORIA...

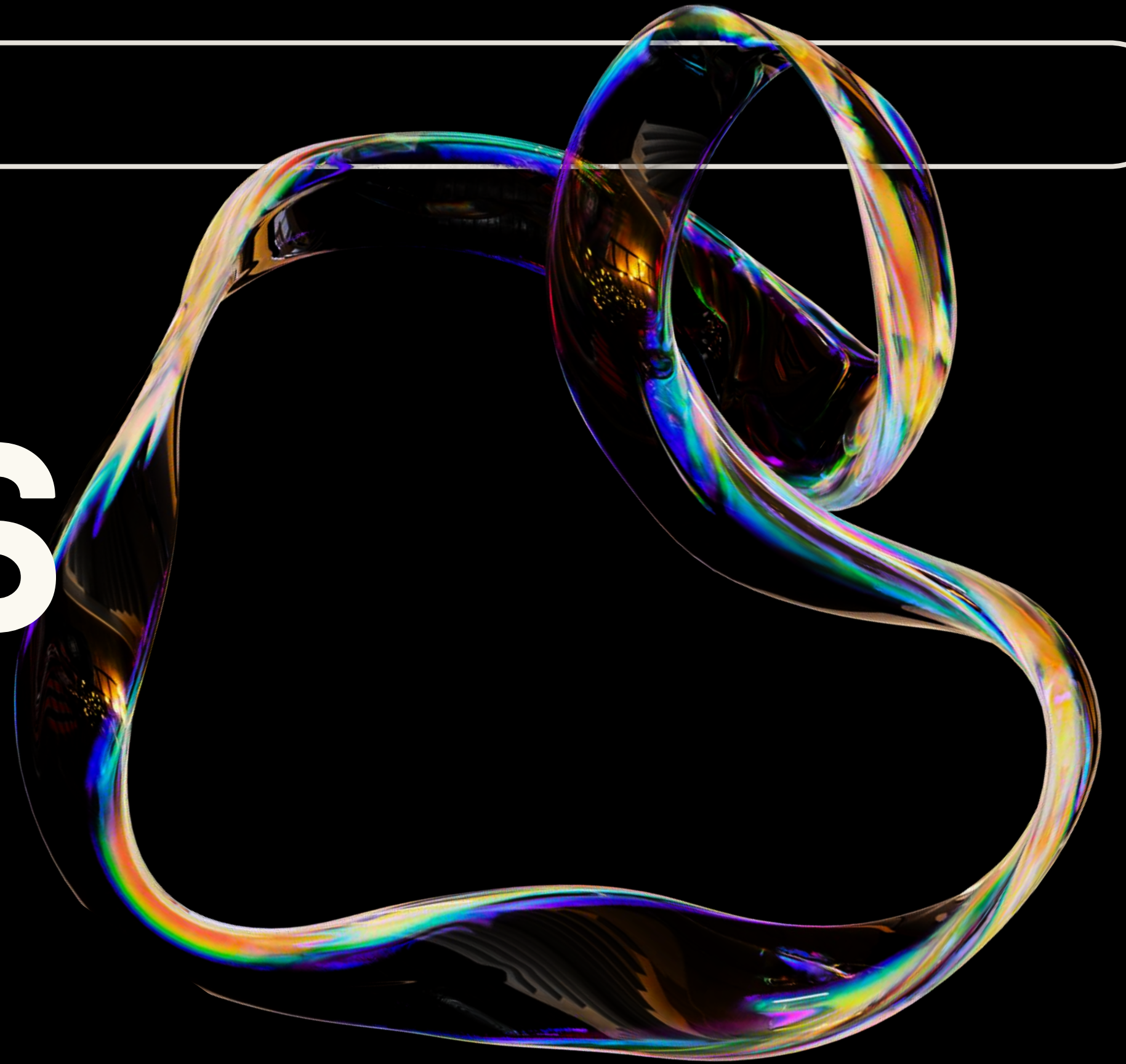
- **2018:** Fundacion del proyecto por Enrico Frenchi
- **2019:** Primeras versiones estables, con soporte basico para GUI de escritorio.
- **2020:** Soporte para plataformas moviles (Android e iOS)
- **2021:** Introduccion de Fyne 2.0, con mejoras significativas en redimimiento y nuevos Widgets
- **2023:** Lanzamiento de Fyne v2.1.0, con mejoras en la compatibilidad y optimizacion para dispositivos moviles.









**¡GRACIAS**  
**POR SU ATENCIÓN!**



 [www.reallygreatsite.com](http://www.reallygreatsite.com)

 [hello@reallygreatsite.com](mailto:hello@reallygreatsite.com)

 123 Anywhere Street., Any  
City