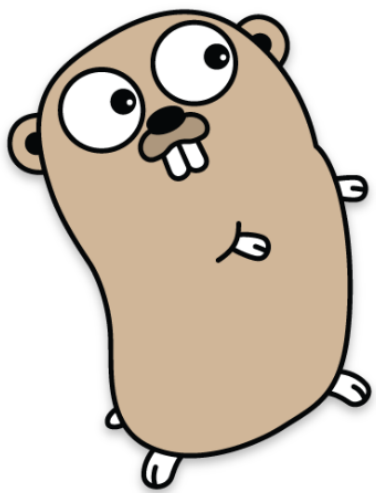
Paradigmas de Lenguajes y Programación

Golang

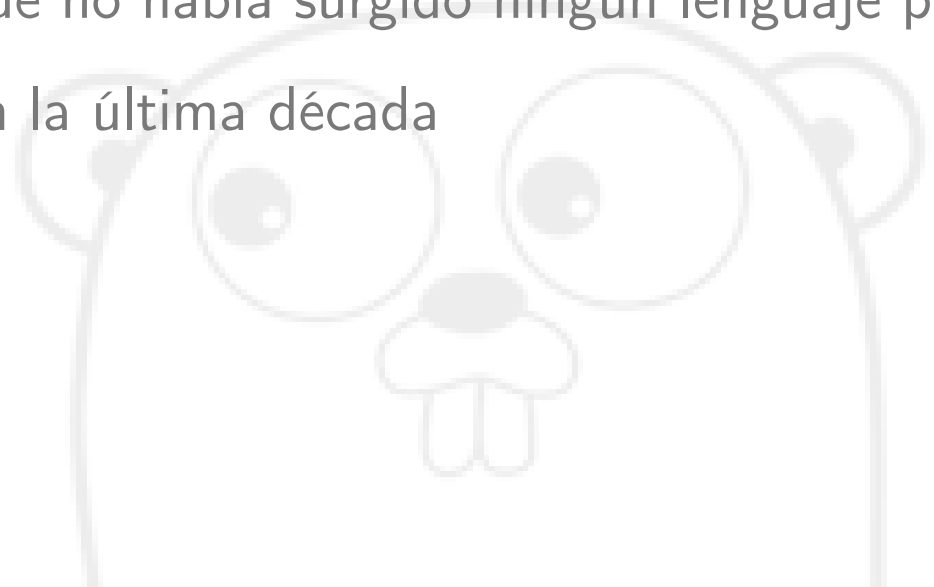
Luciano Serruya Aloisi

5 de julio de 2018



Introducción

- Robert Griesemer, Rob Pike, y Ken Thompson comenzaron el proyecto en el 2007
- Para fines del 2009 fue oficialmente presentado
- La principal motivación fue que no había surgido ningún lenguaje para sistemas grandes (*grandes*) en la última década



Características

- Fuertemente tipado (muy fuerte)
- Tipado estático
- *Memory-safe* (gracias a su recolector de basura)
- Diseñado para ser seguro y performante

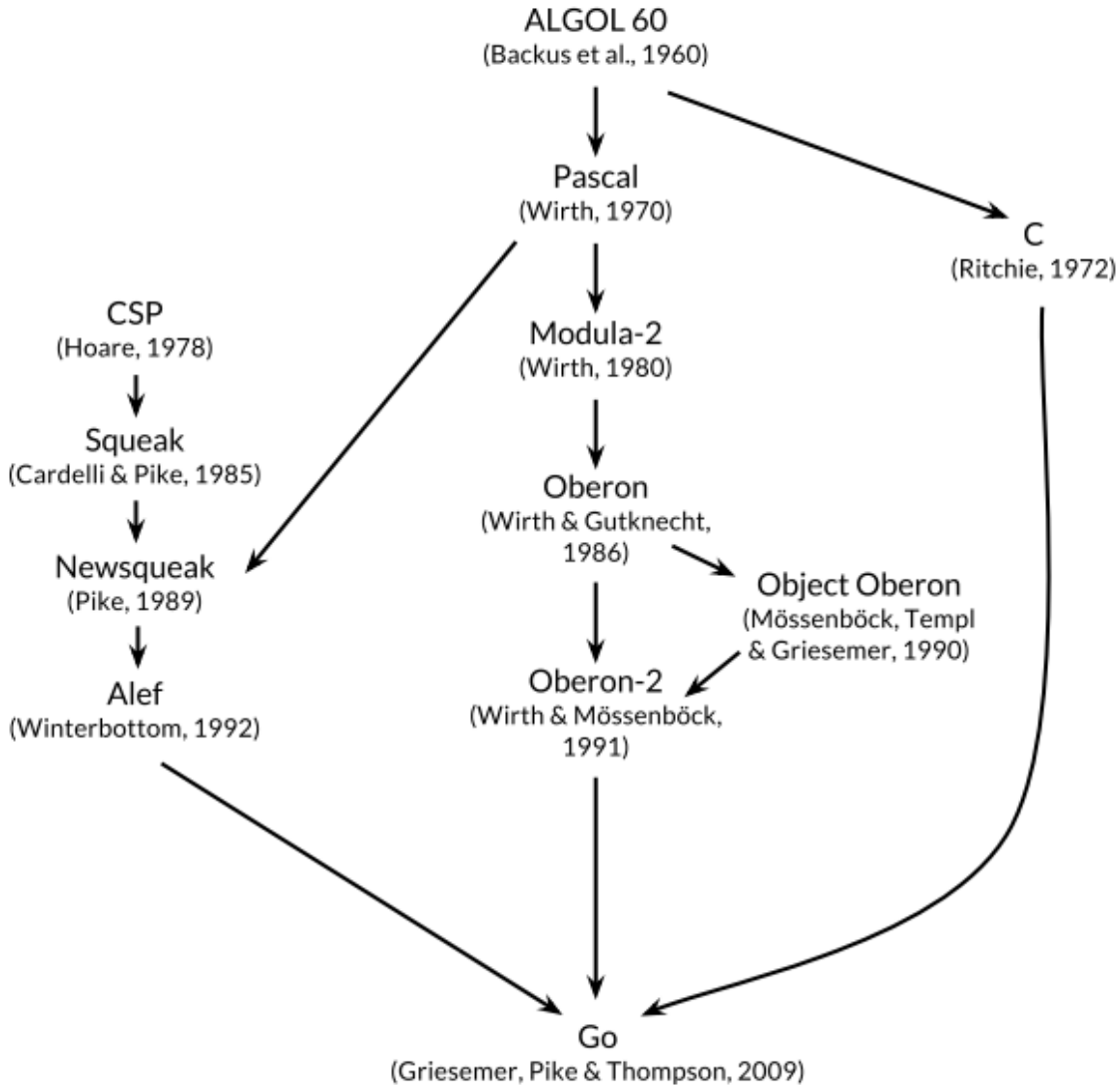


Figura: Ancestros de Go [2]

Demo 1

```go
package main

import "fmt"

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

"Hola mundo!" en Go

Estructuras de control

- `if..else`
- `for`
- `switch`

Tipos de datos simples

- Numéricos
  - Enteros
  - Flotantes
- Cadenas
- Booleanos

Tipos de datos complejos

- Arreglos (tamaño fijo)
- *Slices* (arreglos con tamaños variantes)
- Mapas
- Punteros
- **Registros**

Demo 2

```go
func imprimeMapa(mapa map[string]int) {
    for k, v := range mapa {
        fmt.Printf("%s: %d\n", k, v)
    }
}
```

Iterando sobre un mapa

Paradigmas

- Principalmente **procedural**
- También permite emular objetos con los **registros**, **definición de métodos**, **registros embebidos**, e **interfases**
  - Se pueden definir métodos para *casi cualquier tipo de dato* [1], no solamente para aquellos definidos por el usuario

Demo 3

```
1 type Persona struct{
2     nombre string
3     edad int
4 }
5 func (pers *Persona) Saludar() string {
6     return fmt.Sprtinf("Hola, mi nombre
          es %s", pers.nombre)
7 }
```

Definición de un registro y un método

Demo 4

```
1  type Saludador interface {
2      Saludar() string
3  }
4
5  func Saluden(saludadores ...Saludador) {
6      for _, obj := range saludadores {
7          fmt.Printf("%s\n", obj.Saludar()
               )
8      }
9  }
10
11 unaPersona := Persona{"unNombre", 32}
12 Saluden(&unaPersona)
13
14 // "Hola! mi nombre es unNombre"
```

Demo 5

```
 1  type Alumno struct {
 2      Persona
 3      notas []int
 4  }
 5
 6  unAlumno := Alumno{
 7      Persona{"unNombre", 21},
 8      int[]{7,8,9},
 9  }
10
11  fmt.Printf("%s\n", unAlumno.Saludar())
```

Estructuras complejas mediante composición, no herencia

Demo 6

```go
file, err := os.Open("test.txt")
if err != nil {
    // manejar el error
    return
}
```

Manejo de situaciones inespereadas a través de devolución de valores de error

Defer, Panic, Recover

- Go no maneja el concepto de *excepciones*
- Dispone de las funciones `panic` y `recover`
  - `panic`: pone el hilo de ejecución en *estado de pánico*
  - `recover`: recupera el control del hilo de ejecución
- Una llamada a `recover` durante una ejecución normal no tiene ningún efecto
- `recover` debe estar dentro de una **función diferida**

Demo 7

- Las *funciones diferidas* se invocan antes de que termine la ejecución de la función actual
- Sirven para cerrar archivos, liberar recursos, y manejar errores

```go
func main() {
    defer func() {
        fmt.Printf("No' vamoooo'\n")
    }
    fmt.Printf("Algo re importante...\n"
        )
}
// "Algo re importante..."
// "No' vamoooo'"
```

Función diferida

Concurrencia

- Go ejecuta porciones de código de forma concurrente a través de las `goroutines`
- No necesariamente se corresponde una `goroutine` con un hilo del SO
- Para comunicarse y sincronizarse varias `goroutines` se utilizan **canales**
- Se definen con un tipo de dato, que será el tipo de dato que viajará sobre el canal

Demo 8

```
1 func hello() {
2     fmt.Println("Hello world goroutine")
3 }
4 func main(){
5     go hello()
6     fmt.Println("main function")
7 }
```

Ejecución concurrente con goroutines [3]

Canales

- Las operaciones sobre un canal son **bloqueantes**
  - Si una `goroutine` intenta leer (`<- c`) sobre un canal vacío, se bloqueará hasta que otra `goroutine` escriba en ese canal
  - Si una `goroutine` intenta escribir (`c <-`) en un canal que esté lleno (en el caso de los canales con buffer), se bloqueará hasta que otra `goroutine` lea del canal

¿Qué pasa con los otros paradigmas?

Para implementar otros paradigmas será necesario

¿Qué pasa con los otros paradigmas?

Para implementar otros paradigmas será necesario

¿Qué pasa con los otros paradigmas?

Para implementar otros paradigmas será necesario

¿Preguntas?

¡Gracias!

Referencias


Ivo Balbaert. «The Way to Go: A Thorough Introduction to the Go Programming Language». En: iUniverse, 2012. Cap. 10. Structs and Methods.


Alan A. A. Donovan; Brian W. Kernighan. «The Go Programming Language». En: Addison-Wesley, 2016. Cap. Preface.


Naveen Ramanathan. *Part 21: Goroutines*. Jul. de 2017. URL: `https://golangbot.com/goroutines/`.