# The Go Programming Language

# Hello, world!

```
package main

import "fmt"

func main() {
   fmt.Printf("Hello, 世界\n");
}
```

# Wer hat's verbrochen?

- Robert Griesemer
- Ken Thompson
- Rob Pike
- ende 2007

# Wozu noch eine Sprache?

Eine Alternative zu C
ohne die „Schmerzen“ von C.

# Schmerzen in C?

- Pointer-Arithmetik
- Speicher-Management
- Concurency
- Resource closing
- ...

# Pointer-Arithmetik

- Es gibt Pointer wie in C
- Aber es gibt keine Arithmetik

# Speicher-Management

- reine Alokierung mit `new()`
  - Initialisierung mit Null-Wert (`0`, `0.0`, `false`, `""`)
- Erzeugung & Initialisierung mit `make()`
  - slices, maps, and channels
- Grabage Colelctor

# Concurency

- Channels, nativer Typ
  - synchron
  - Asynchron buffered

```
var myCahn = make(chan string)
MyChan <- „hello“
fmt.println(<-myChan) // hello
```

- Go-Routinen

```
go doSmomeThing() // non-blocking
```

# Deklaration

# Resource closing

```
func doSomething() {
  f, err := openFile(“/foo/bar“)
  defer closeFile(f)
  // number crunching

  return data, err
}
```

# ...

- Kann Unicode nativ
- Strings nicht 0-Byte-terminiert, Länge
- Deklarationen:
  - `int *a, b;` --> **`var`** `a, b *int`
  - `func doSome(a, b int) (x, y int)`
- Multi-Return & Multiasignemnt
  - `var a,b := doSomething()`
  - `return 1, 2`

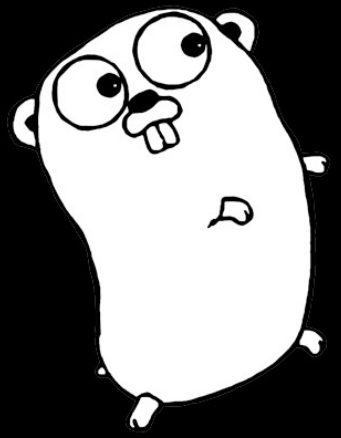
# ...

- iota

```
const (
   BLA int = iota,
   BLUB,
   FOO
)
```

- Scope -> Package
  - lc -> „private“, uc -> „public“
- OOP -> Structs mit funtionen

# GO



http://golang.org