Let's get ready to Go!

Marga Manterola

14.11.2015

About me

Born in Argentina, living in Munich since 2012.

Site Reliability Engineer for Google.

Developed code in C, C++, Java, Python and Go.

Spent the last 10+ years working mostly with Python.

Started my first Go project early this year.

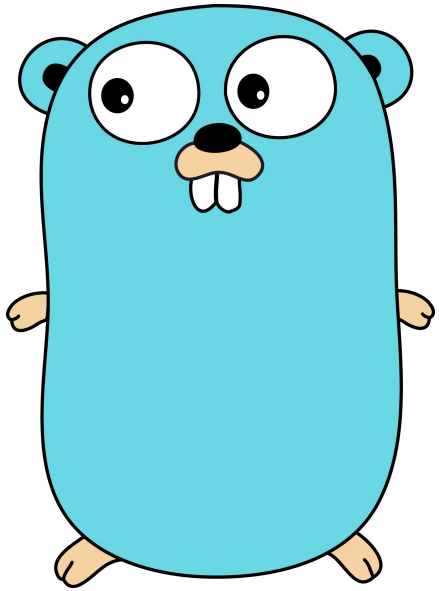
Debian Developer since 2005.

A bit of history

Go was created in 2007 by Robert Griesemer, Rob Pike and Ken Thompson, released to the public in 2009.

The goal was to create a C-like language, that incorporated the good things from modern languages.

Go 1.0 was released on 2012, latest version is Go 1.5



Go's Gopher created by Renee French

> It wasn't enough to just add features to existing programming languages, because sometimes you can get more in the long run by taking things away.

Rob Pike

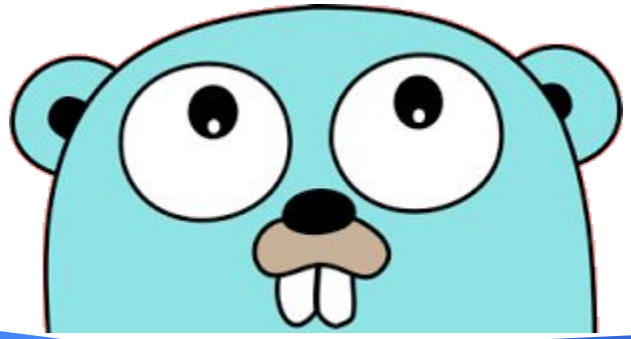
What Go is not

Go is not an interpreted language.

Go is not a replacement for Python.

Go is not the answer to every programming need.

Go is not very well named: for Google searches use *golang*.



> Go is an open source programming language that makes it easy to build simple, reliable, and efficient software

From the Go web site at http://golang.org

Source: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis non erat sem

What Go is

Go is an open source compiled language.

Go is statically typed, quite strict about type checking.

Go is designed for concurrency, has garbage collection, unicode support, and some dynamic typing.

Go is usually a good choice when building networked software.

Go is *this century's C*.

Some Go goodies

The compiler catches tons of errors.

Very powerful and complete libraries included with the language.

Native concurrency support.

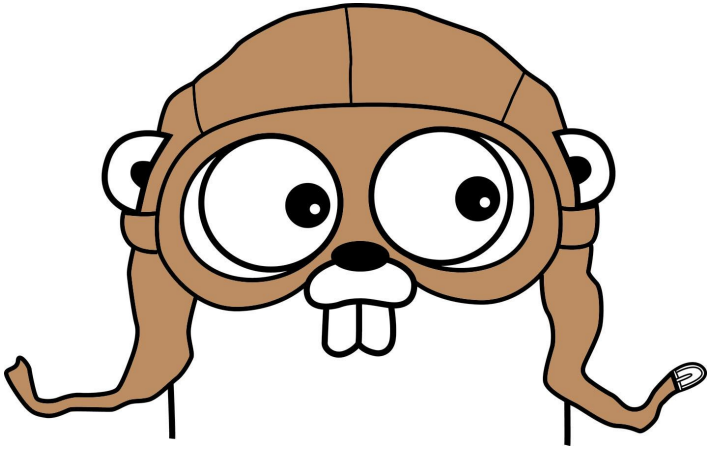
The Go Playground serves as a quick and easy way of trying things out: http://play.golang.org

Go looks a bit like C

```
package main

import "fmt"

func main() {
     sum := 0
     for i := 0; i < 10; i++ {
          sum += i
     }
     fmt.Println(sum)
}
```



Some Go quirks

Types are declared the other way round

- → `c *int` instead of `int* c`
- → `a []byte` instead of `byte[] a`

The case of the first letter of a function determines its visibility

Error handling is ***strange***

Some Go snippets

```
func hasPort(s string) bool {
  return strings.LastIndex(s, ":") > strings.LastIndex(s, "]")
}

func IsRegular(path string) (bool, error) {
    fi, err := FileStat(path)
    if err != nil {
        return false, err
    }
    return fi.Mode().IsRegular(), nil
}
```

Source: https://blitiri.com.ar/p/kxd/

Interfaces

Go doesn't have fully fledged objects.

- It has structs with associated methods.
  Any piece of code can associate methods to any type.

Polymorphism is done through interfaces.

- The interface is declared as a set of functions.
  Every type that satisfies the interface automatically can be used as a variable of that type.

More Go snippets

```
func (kc *KeyConfig) Key() (key []byte, err error) {
   return ioutil.ReadFile(kc.keyPath)
}

func (req *Request) Printf(format string, a ...interface{}) {
    msg := fmt.Sprintf(format, a...)
    msg = fmt.Sprintf("%s %s %s", req.RemoteAddr, req.URL.Path, msg)
    logging.Output(2, msg)
}
```

Source: https://blitiri.com.ar/p/kxd/

Concurrency in Go

Go uses *goroutines* for concurrency

- Any function started with go func is a goroutine.

They can communicate through **channels**

- Channels synchronize goroutines by blocking on sending or receiving.

Goroutines are lightweight and cheap.

Demo!

Questions?

THANK YOU