Hello Golang

A modern programming language

Martin Czygan

Gopher

June 8, 2017 18:00 CEST

jenadevs meetup at Friedrich-Schiller-Universität Jena

About me

- Gopher since 2013
- Programmer at Leipzig University Library
- Co-Author of Getting Started with Python Data Analysis
- Consultant on data processing themes
- Trainer at Python Academy

About me

A few open source projects: esbulk, solrbulk, microblob, gluish, metha, marctools.

Presentations at LPUG about pandas, luigi, neural nets.

Workshop on Go interfaces at Golab, an European Go conference in Italy.

My language log

BASIC, Pascal, Perl, *Bash*, Ruby, *Java*, *C*, C++, *PHP*, *JavaScript*, *Python*, *Go*.

Outline

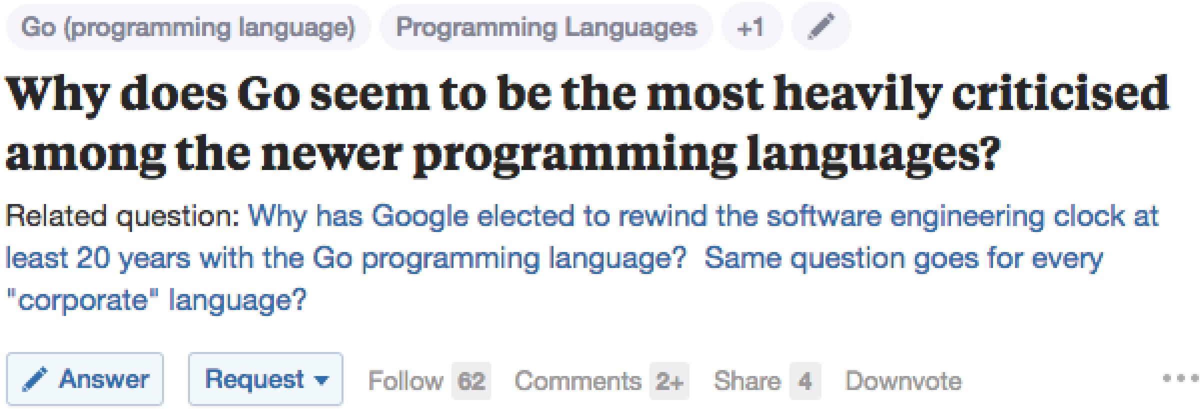
First: slides

- Go: its users and critics, language constructs
- Go and OO, Go and Concurrency
- The Go development workflow

Then: hands-on, if you want:

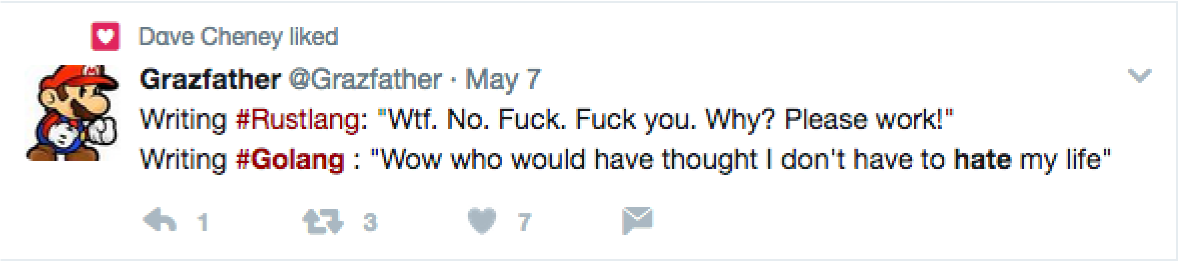
- Get Go installed
- Write a simple (web service | concurrent program) in Go
- Write a Docker storage plugin

Question: Is Go a modern language?



Why does Go seem to be the most heavily criticised among the newer programming languages?

Related question: Why has Google elected to rewind the software engineering clock at least 20 years with the Go programming language? Same question goes for every "corporate" language?

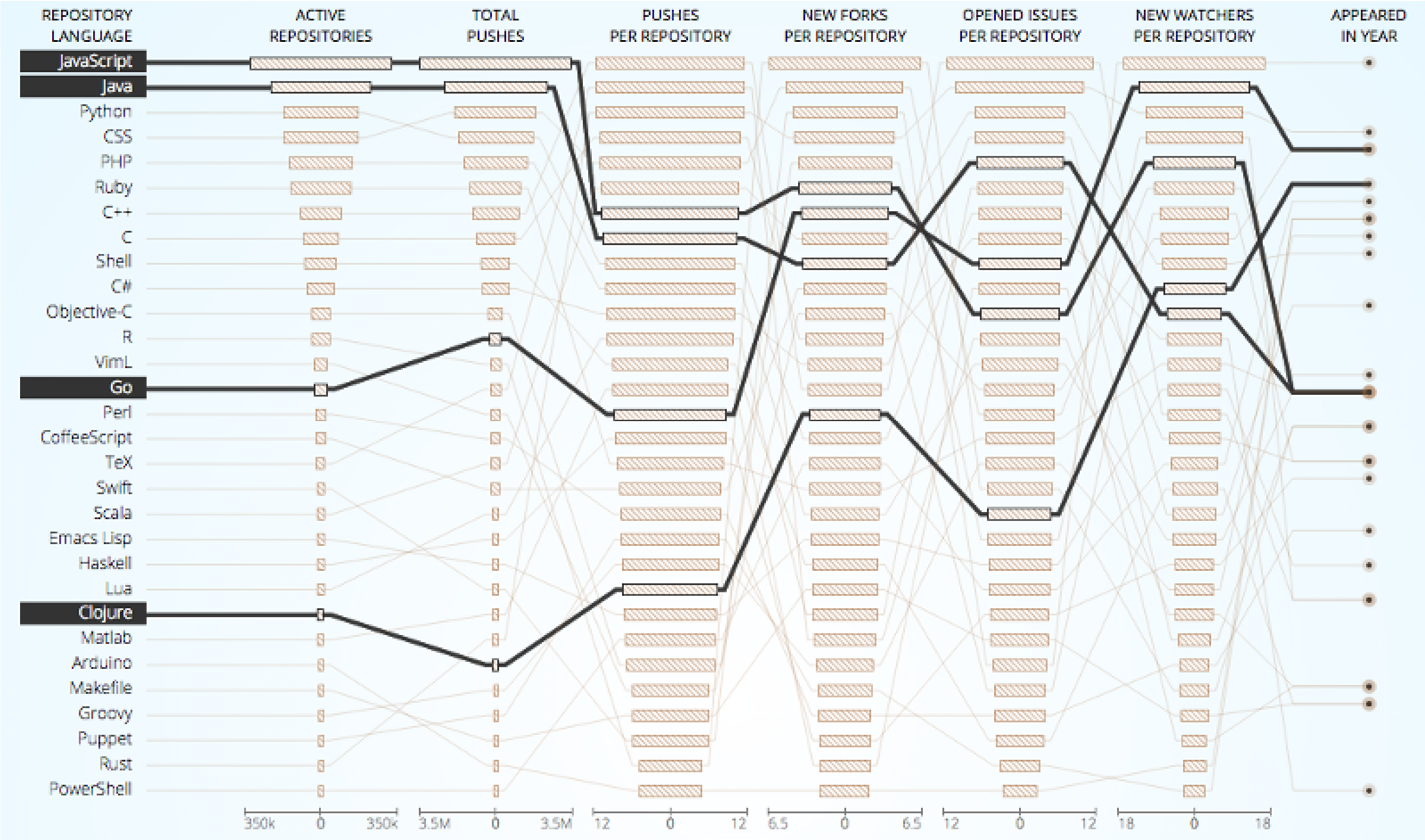

Dave Cheney liked

Grazfather @Grazfather · May 7

Writing #Rustlang: "Wtf. No. Fuck. Fuck you. Why? Please work!"
Writing **#Golang** : "Wow who would have thought I don't have to **hate** my life"

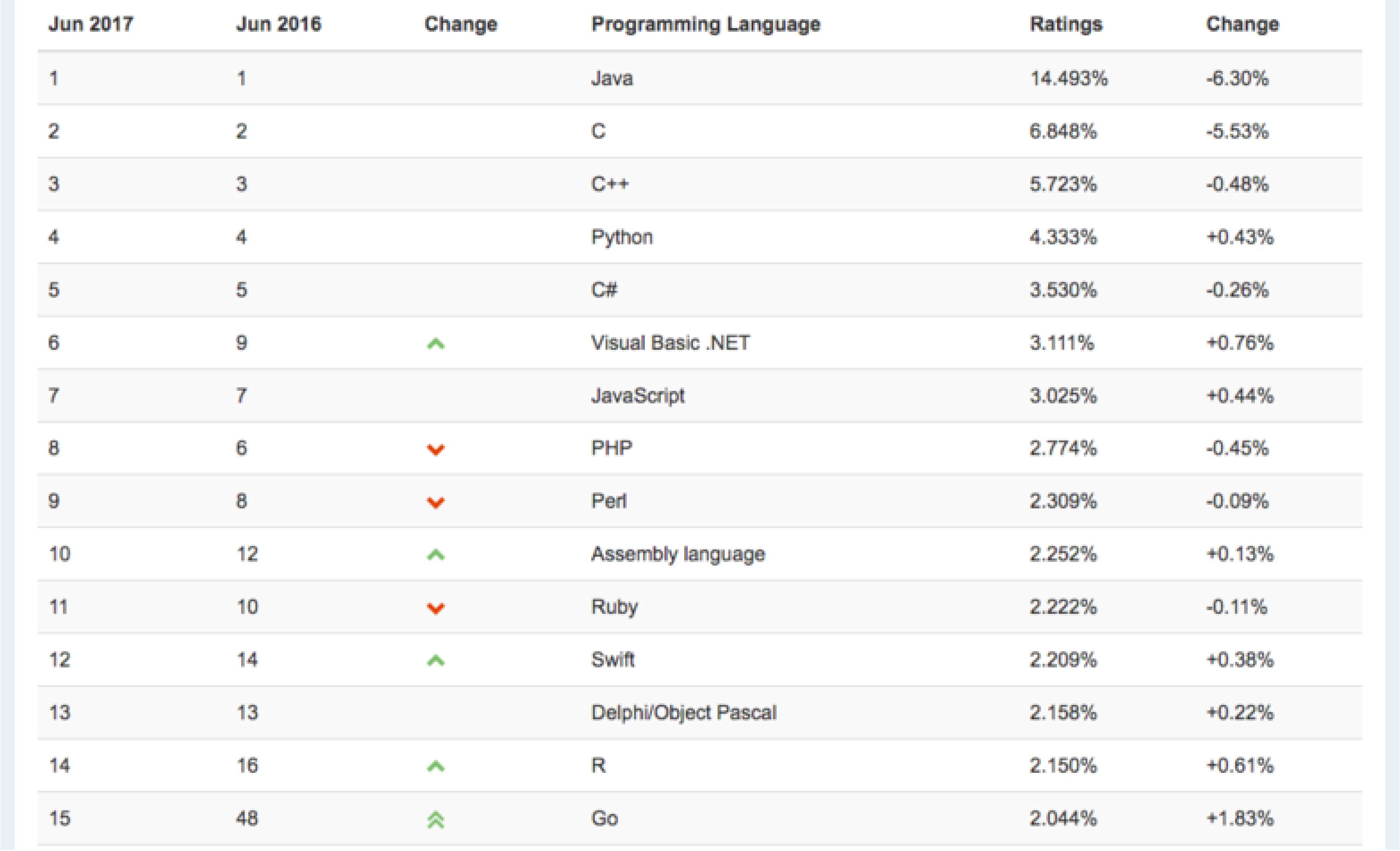
GitHub Activity (2016)

From GoLang or the future of the dev:



TIOBE

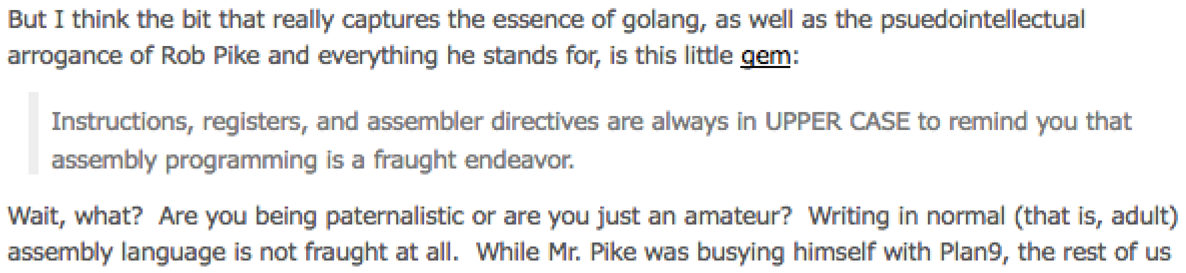
From June 2017:



Jun 2017	Jun 2016	Change	Programming Language	Ratings	Change
1	1		Java	14.493%	-6.30%
2	2		C	6.848%	-5.53%
3	3		C++	5.723%	-0.48%
4	4		Python	4.333%	+0.43%
5	5		C#	3.530%	-0.26%
6	9	⌃	Visual Basic .NET	3.111%	+0.76%
7	7		JavaScript	3.025%	+0.44%
8	6	⌄	PHP	2.774%	-0.45%
9	8	⌄	Perl	2.309%	-0.09%
10	12	⌃	Assembly language	2.252%	+0.13%
11	10	⌄	Ruby	2.222%	-0.11%
12	14	⌃	Swift	2.209%	+0.38%
13	13		Delphi/Object Pascal	2.158%	+0.22%
14	16	⌃	R	2.150%	+0.61%
15	48	⇈	Go	2.044%	+1.83%

Golang is trash

Golang is trash (2014):

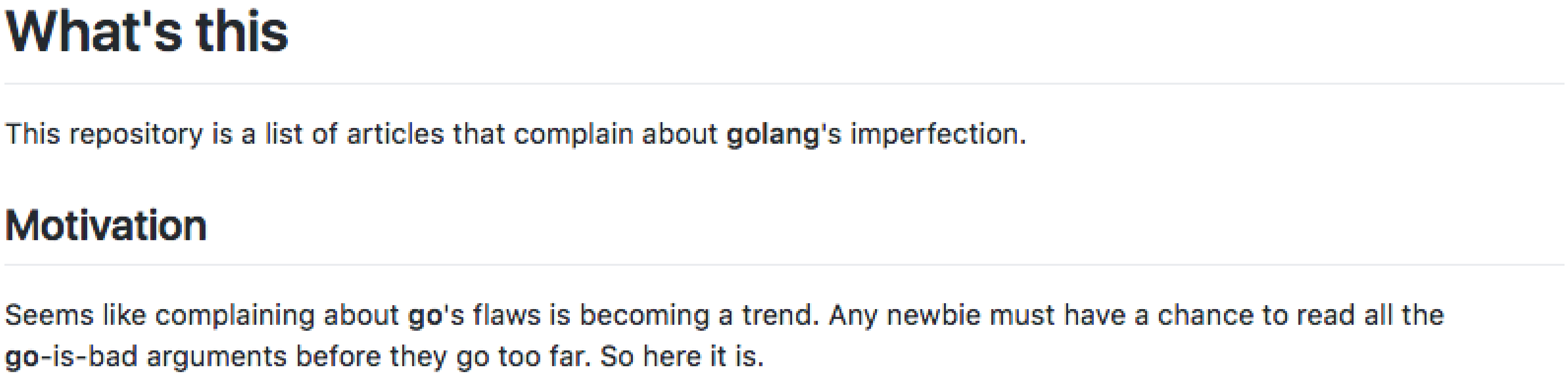


But I think the bit that really captures the essence of golang, as well as the psuedointellectual arrogance of Rob Pike and everything he stands for, is this little gem:

> Instructions, registers, and assembler directives are always in UPPER CASE to remind you that assembly programming is a fraught endeavor.

Wait, what? Are you being paternalistic or are you just an amateur? Writing in normal (that is, adult) assembly language is not fraught at all. While Mr. Pike was busying himself with Plan9, the rest of us

github.com/ksimka/go-is-not-good (1233 stars):

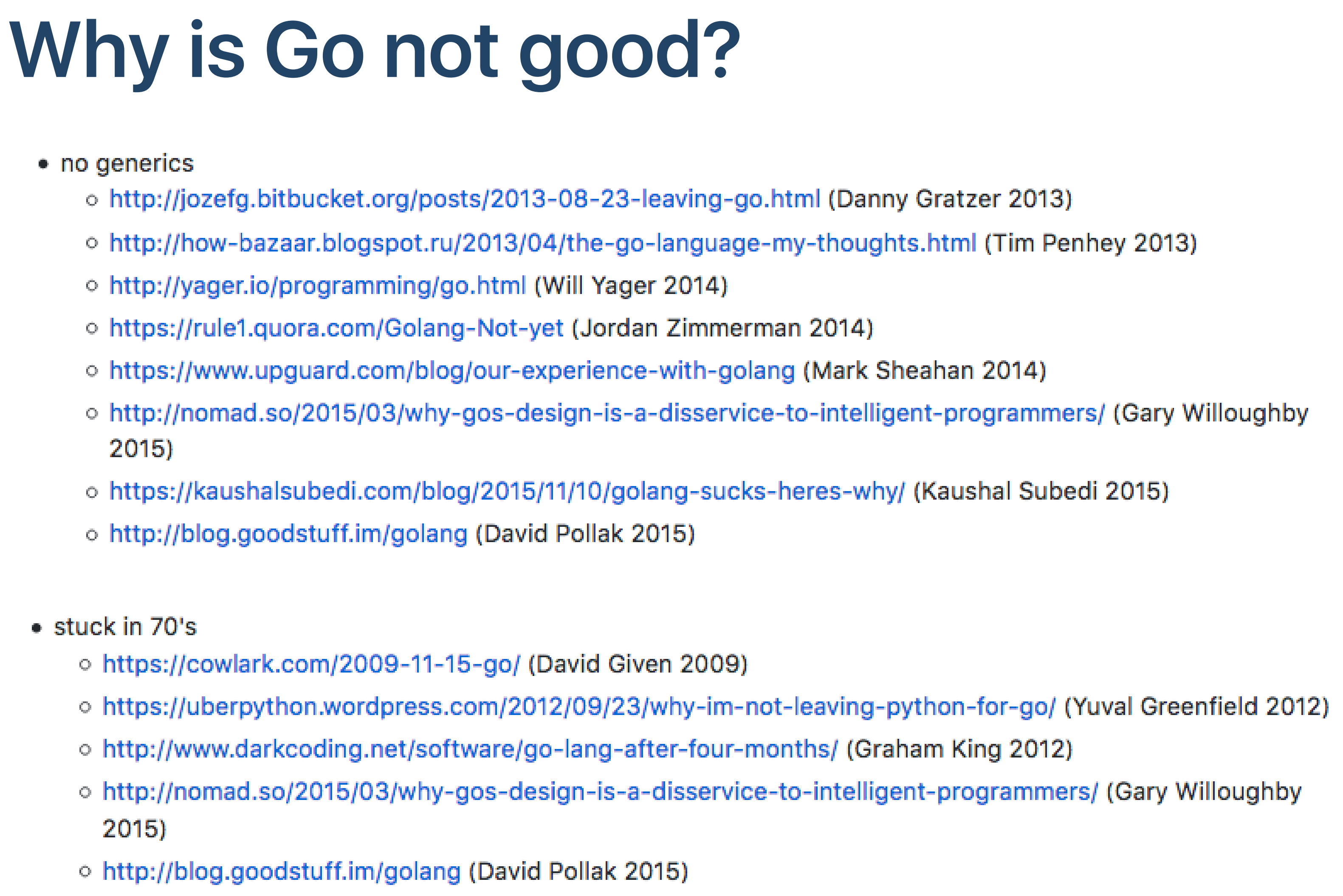


What's this

This repository is a list of articles that complain about **golang**'s imperfection.

Motivation

Seems like complaining about **go**'s flaws is becoming a trend. Any newbie must have a chance to read all the **go**-is-bad arguments before they go too far. So here it is.

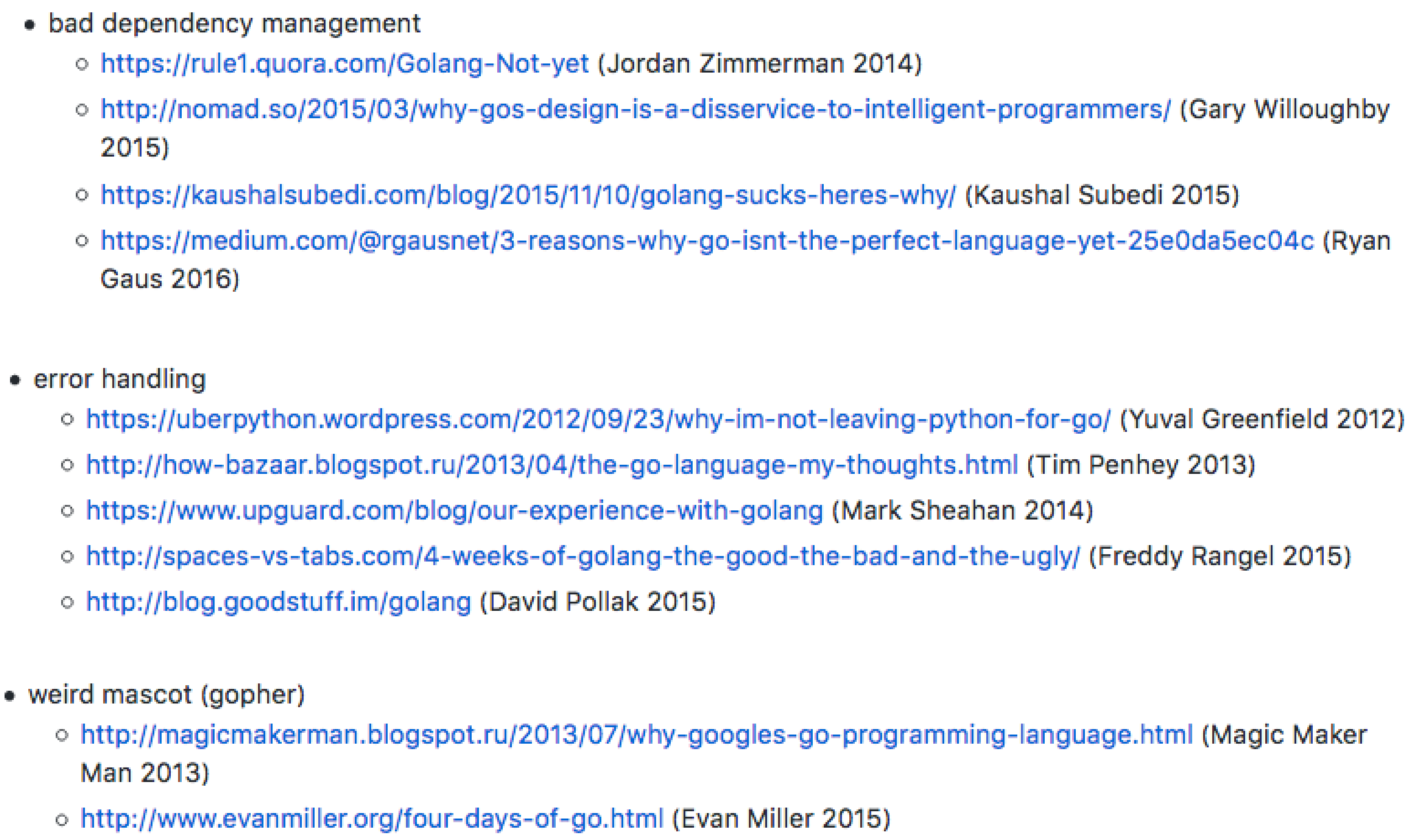


Why is Go not good?

- no generics
  - http://jozefg.bitbucket.org/posts/2013-08-23-leaving-go.html (Danny Gratzer 2013)
  - http://how-bazaar.blogspot.ru/2013/04/the-go-language-my-thoughts.html (Tim Penhey 2013)
  - http://yager.io/programming/go.html (Will Yager 2014)
  - https://rule1.quora.com/Golang-Not-yet (Jordan Zimmerman 2014)
  - https://www.upguard.com/blog/our-experience-with-golang (Mark Sheahan 2014)
  - http://nomad.so/2015/03/why-gos-design-is-a-disservice-to-intelligent-programmers/ (Gary Willoughby 2015)
  - https://kaushalsubedi.com/blog/2015/11/10/golang-sucks-heres-why/ (Kaushal Subedi 2015)
  - http://blog.goodstuff.im/golang (David Pollak 2015)

- stuck in 70's
  - https://cowlark.com/2009-11-15-go/ (David Given 2009)
  - https://uberpython.wordpress.com/2012/09/23/why-im-not-leaving-python-for-go/ (Yuval Greenfield 2012)
  - http://www.darkcoding.net/software/go-lang-after-four-months/ (Graham King 2012)
  - http://nomad.so/2015/03/why-gos-design-is-a-disservice-to-intelligent-programmers/ (Gary Willoughby 2015)
  - http://blog.goodstuff.im/golang (David Pollak 2015)

Why is Go not good?



- bad dependency management
  - https://rule1.quora.com/Golang-Not-yet (Jordan Zimmerman 2014)
  - http://nomad.so/2015/03/why-gos-design-is-a-disservice-to-intelligent-programmers/ (Gary Willoughby 2015)
  - https://kaushalsubedi.com/blog/2015/11/10/golang-sucks-heres-why/ (Kaushal Subedi 2015)
  - https://medium.com/@rgausnet/3-reasons-why-go-isnt-the-perfect-language-yet-25e0da5ec04c (Ryan Gaus 2016)

- error handling
  - https://uberpython.wordpress.com/2012/09/23/why-im-not-leaving-python-for-go/ (Yuval Greenfield 2012)
  - http://how-bazaar.blogspot.ru/2013/04/the-go-language-my-thoughts.html (Tim Penhey 2013)
  - https://www.upguard.com/blog/our-experience-with-golang (Mark Sheahan 2014)
  - http://spaces-vs-tabs.com/4-weeks-of-golang-the-good-the-bad-and-the-ugly/ (Freddy Rangel 2015)
  - http://blog.goodstuff.im/golang (David Pollak 2015)

- weird mascot (gopher)
  - http://magicmakerman.blogspot.ru/2013/07/why-googles-go-programming-language.html (Magic Maker Man 2013)
  - http://www.evanmiller.org/four-days-of-go.html (Evan Miller 2015)

Why is Go not good?

The list goes on and on:

- designed for stupid people
- no OOP
- no exceptions
- no versioning model
- too opinionated
- too simple

So, why do I use it?

- I was curious about Ken Thompsons' experiment.
- I like production code and low operational overhead (e.g. install, maybe config, run).
- With Go, I mostly think about the problem, not about the language.

A small language

- 25 keywords

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

Hello World

```
package main

import "fmt"

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

Play.

Declaring variables

```go
package main

import (
    "fmt"
    "time"
)

var timeout time.Duration
var N = 4

func main() {
    n, k := N, 2.0
    fmt.Printf("n=%d, k=%0.3f, timeout=%s", n, k, timeout)
}
```

Play.

Every type has a zero value

```go
package main

import "fmt"

func main() {
    var a float64
    var b int16
    var c string
    fmt.Printf("a=%v, b=%v, c=%v, len(c)=%v",
        a, b, c, len(c))
}
```

Play.

Various numberic types

```
type                                 size in bytes

byte, uint8, int8                     1
uint16, int16                         2
uint32, int32, float32                4
uint64, int64, float64, complex64     8
complex128                           16
```

- https://golang.org/pkg/builtin/

Basic types

```
bool

string

int  int8  int16  int32  int64
uint uint8 uint16 uint32 uint64 uintptr

byte // alias for uint8

rune // alias for int32
     // represents a Unicode code point

float32 float64

complex64 complex128
```

Only one loop construct

```go
package main

import "fmt"

const Prefix = ">> "

func main() {
    for i := 0; i < 5; i++ {
        log.Printf("%s %0d", Prefix, i)
    }
}
```

Play. Break, continue as you expect.

Slices

```go
package main

import "fmt"

func main() {

    cities := []string{"Jena", "Weimar", "Erfurt"}

    for i, city := range cities {
        fmt.Println(i, city)
    }
}
```

Functions

```go
package main

import "fmt"

func Hello(name string) (string, error) {
        if len(name) < 2 {
                return "", fmt.Errorf("name too short")
        }
        return fmt.Sprintf("Hello %s", name), nil
}

func main() {
    greeting, err := Hello("a")
    fmt.Println(greeting, err)
}
```

Play.

If needs no parentheses

```go
package main

import "log"

func main() {
    a, b := 4, 3
    if a < b {
        log.Println("a smaller b")
    } else {
        log.Println("a not smaller b")
    }
}
```

Play.

Keywords to go (13)

```
        default              interface    select
case    defer         go     map          struct
chan                  goto                switch
        fallthrough                       type
```

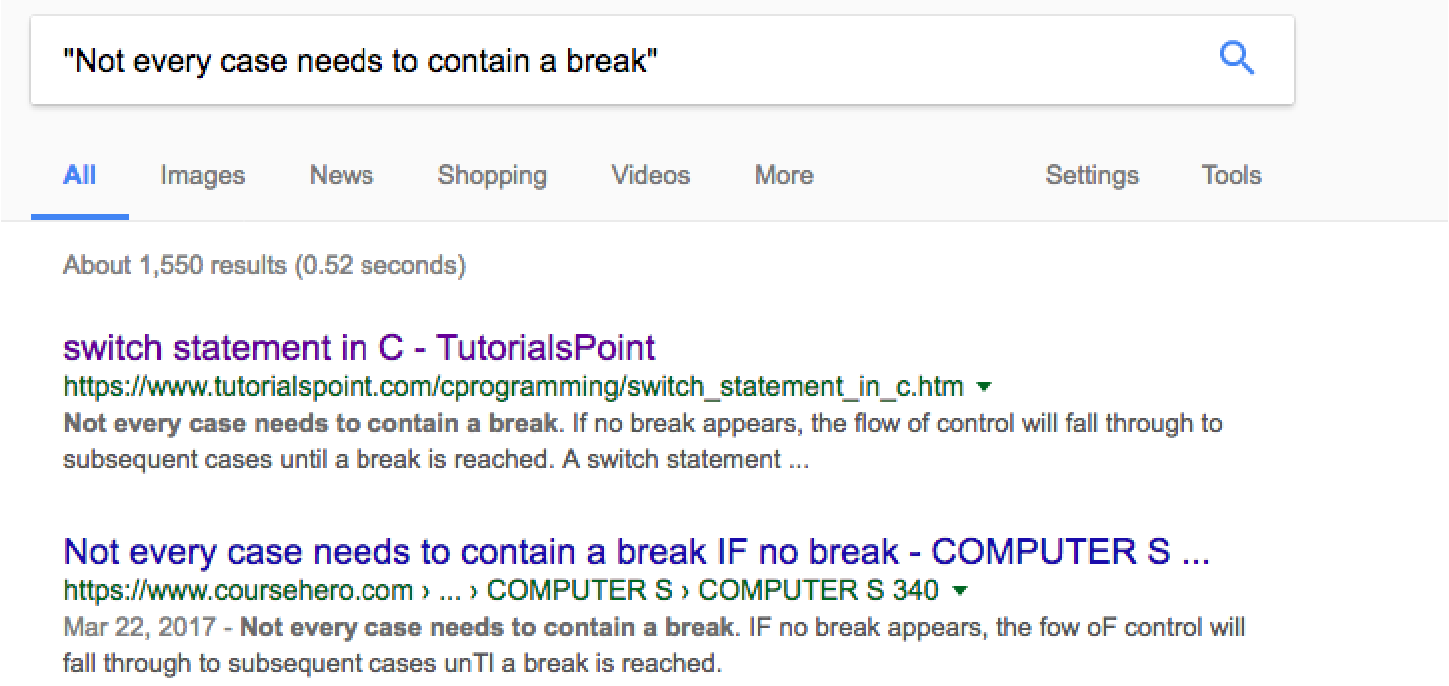
Braching with switch, case and default

```go
package main

import "fmt"

func main() {
    s := "A"
    switch s {
    case "A":
        fmt.Println("a")
    case "B":
        fmt.Println("b")
    default:
        fmt.Println("?")
    }
}
```

Play.

Fallthrough

- A design mistake correction from the C language



"Not every case needs to contain a break"

About 1,550 results (0.52 seconds)

switch statement in C - TutorialsPoint
https://www.tutorialspoint.com/cprogramming/switch_statement_in_c.htm
Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached. A switch statement ...

Not every case needs to contain a break IF no break - COMPUTER S ...
https://www.coursehero.com › ... › COMPUTER S › COMPUTER S 340
Mar 22, 2017 - **Not every case needs to contain a break**. IF no break appears, the fow oF control will fall through to subsequent cases unTl a break is reached.

Fallthrough

- Example, ascii85

```
var v uint32
switch len(src) {
default:
    v |= uint32(src[3])
    fallthrough
case 3:
    v |= uint32(src[2]) << 8
    fallthrough
case 2:
    v |= uint32(src[1]) << 16
    fallthrough
case 1:
    v |= uint32(src[0]) << 24
}
```

Play.

Keywords to go (9)

```
                             interface    select
        defer         go     map          struct
chan                  goto
                                          type
```

Defer

- Defer is wonderful.

```
package main

func f() error {
    defer fmt.Println("exiting f")
    if rand.Float64() > 0.5 {
        fmt.Println("f failed")
    }
    return nil
}

func main() {
    f()
}
```

Play.

Defer

- Use cases: closing file, connections, response bodies, profiling
- make code much more readable, but has performance implications

Keywords to go (8)

```
                             interface    select
                      go     map          struct
chan                  goto
                                          type
```

Keywords to go (7)

```
                             interface    select
                      go     map          struct
chan

                                          type
```

Hashmaps

```
package main

import "fmt"

func main() {
        m := map[string]string{
                "Meetup":   "jenadevs",
                "Location": "FSU Jena",
        }
        fmt.Println(m)
}
```

Play.

Keywords to go (6)

```
                             interface    select
                      go                  struct
chan
                                          type
```

- Concurrency: go, chan, select
- OO: type, struct, interface

OO in Go

- no classes
- composition over inheritance
- small interfaces
- no explicit declarations

Custom types

- before we see compound types, let's look at something simpler

Custom types

```go
package main

import "fmt"

type Celsius float64

func main() {
        var temp Celsius
        fmt.Printf("below %v degree", temp)
}
```

Play.

Functions on custom types

```go
package main

import "fmt"

type Celsius float64

func (c Celsius) String() string {
    return fmt.Sprintf("%0.1f°", c)
}

func main() {
    var temp Celsius
    fmt.Printf("below %s degree", temp)
}
```

Compound types

```go
package main

import "fmt"

type Meetup struct {
        Name     string
        Location string
}

func main() {
    meetup := Meetup{
        Name:     "jenadevs",
        Location: "FSU Jena",
    }
    fmt.Printf("%+v", meetup)
}
```

Play.

Compound types (play)

```
package main

import "fmt"

type Address struct {
        City   string
        Street string
}

type Meetup struct {
        Name     string
        Location Address
}

func main() {
        meetup := Meetup{"jenadevs", Address{
                Street: "Fürstengraben 1",
                City:   "Jena"}}
        fmt.Printf("%+v", meetup)
}
```

Defining Methods on Types

```
type Client struct {
    scheme string
    host   string
    proto  string
    ...
}

...

func (cli *Client) ContainerList(...) (..., error) {
...
}
```

- moby/client/client.go
- moby/client/container_list.go

Types

- basic types (int, float, complex64, string, rune, byte, bool)
- slices (variable sized array)
- maps (hashmaps)
- struct types (compound types)

A few more types

A few more builtin types:

- array types (fixed size)
- pointer types (Pointers reference a location in memory where a value is stored rather than the value itself)
- function types (functions are first class objects)
- interface types
- channel types

Arrays

- rarely used

```go
package main

import "fmt"

func main() {
    var v [3]int64
    fmt.Println(v)
}
```

Play.

Pointers

```
package main

import "fmt"

func main() {
        var x = 42
        fmt.Printf("%v", &x)
}
```

Play.

Pointers

```
package main

import "fmt"

func main() {
        x := new(int32)
        fmt.Printf("%T", x)
}
```

Play.

Pointers

You will see (use) pointer receivers on struct methods:

```
func (cli *Client) ContainerList ...
```

- required, if a method mutates the compound type
- even, if it is just a single method, for consistency, all methods should use a pointer receiver

Function types

- lots of fun
- closures

```
package main

import "fmt"

func main() {
        f := func(s string) string {
                return fmt.Sprintf("<%s>", s)
        }
        fmt.Println(f("functional"))
}
```

Play.

Function types

```
package main

type Converter func(string) string

func Convert(value string, f Converter) string {
        return f(value)
}

func main() {
        // ...
}
```

Play.

Interface types

- set of methods
- satisfied implicitly

Interface types

```
package main

type Starter interface {
    Start() error
}

type Container struct {
        ID string
}

func (c Container) Start() error {
    // ...
}
...
```

Play.

Interface types

> The bigger the interface, the weaker the abstraction.

Interface types

- Go has small interfaces
- Example: package io

```
type Reader interface {
    Read([]byte) (n int, err error)
}

type Writer interface {
    Write([]byte) (n int, err error)
}

type ReadWriter interface {
    Reader
    Writer
}
```

Interface types

Can small interfaces be useful?

- Explore IO workshop

IO

> ... satisfied implictly. But that's actually not the most important thing about Go's interfaces. The really most important thing is the culture around them that's captured by this proverb, which is that the smaller the interface is the more useful it is.

> io.Reader, io.Writer and the empty interface are the three most important interfaces in the entire ecosystem, and they have an average of 2/3 of a method.

Empty interface

```go
package main

import "fmt"

func main() {
    var x interface{}
    x = 5
    fmt.Printf("%v, %T\n", x, x)
    x = "Hello"
    fmt.Printf("%v, %T\n", x, x)
}
```

Type assertion

```go
package main

import "fmt"

func IsString(v interface{}) bool {
    _, ok := v.(string)
    return ok
}

func main() {
    fmt.Println(IsString(23))
    fmt.Println(IsString("23"))
}
```

Play.

Polymorphism

- via interfaces
- no explicit declaration

```
package main

import "fmt"

type Number struct{ x int }

func (n Number) String() string { return fmt.Sprintf("<Number %

func main() {
    five := Number{5}
    fmt.Println(five)
}
```

Play.

Interface advantages

- no dependence between interface and implementation
- easy testing
- avoids overdesign, rigid hierarchy of inheritance-based OO

> The source of all generality in the Go language.

- https://talks.golang.org/2014/research.slide#20

(Requires some boilerplate, e.g. sort.Interface)

TODO

- go tool
- go build, install, test, vet
- testing, benchmarks
- concurrency
- resources (ref/spec, docs, godoc)
- dependency management
- cool projects in Go (fogleman, k8s, docker, termui)

Concurrency

- based on Communicating Sequential Processes (CSP), 1978
- avoids explicit locks

> Do not communicate by sharing memory; instead, share memory by communicating.

Concurrency

Three elements:

- goroutines
- channels
- select statement

Concurrency: goroutines

- the go keyword start a function in a separate lightweigth thread

Concurrency: goroutines

```go
package main

import (
        "fmt"
        "time"
)

func f() {
        time.Sleep(1 * time.Second)
        fmt.Println("f")
}

func main() {
        go f()
        fmt.Println("main")
        time.Sleep(2 * time.Second)
        fmt.Println("main")
}
```

Play.

Concurrency: goroutines

- easy to start (many)

```
package main

import (
        "fmt"
)

func main() {
        N := 1000
        for i := 0; i < N; i++ {
                go func() {
                        x := 0
                        x++
                }()
        }
        fmt.Println("done")
}
```

Play.

Concurrency: channels

- How to communicate between goroutines: enter channels.
- Channels: typed conduits for synchronisation and communication

Concurrency: channels

```go
package main

import "fmt"

func main() {
        ch := make(chan string)
        go func() {
                ch <- "Hello"
        }()
        fmt.Println(<-ch)
}
```

Concurrency: channels

```
package main

// ...

func a(ch chan string) {
        for msg := range ch {
                fmt.Println(msg)
        }
}

func main() {
        ch := make(chan string)
        go a(ch)
        ch <- "Hello"
        ch <- "World"
        close(ch)
        time.Sleep(1 * time.Second)
}
```

Play.

> In Hoare's CSP language, processes communicate by sending or receiving values from named unbuffered channels. Since the channels are unbuffered, the send operation blocks until the value has been transferred to a receiver, thus providing a mechanism for synchronization.

Channels

```
package main

import "fmt"

func main() {
    c := make(chan string)
    go func() {
        c <- "Hello"
        c <- "World"
    }()
    fmt.Println(<-c, <-c)
}
```

Select statement

- select statement is similar to a switch but works with channels

> The select statement lets a goroutine wait on multiple communication
> operations. A select blocks until one of its cases can run, then it executes
> that case.

Select statement

```
func main() {
	ch := make(chan int)
	go func() {
		select {
		case <-time.After(1 * time.Second):
			log.Fatal("timeout")
		case v := <-ch:
			log.Println(v)
			return
		}
	}()
	time.Sleep(1100 * time.Millisecond)
	ch <- 42
	time.Sleep(1 * time.Second)
}
```

Play.

Assorted themes

- standard library tour
- tools

Workshop

- a simple concurrent program that fetches URLs
- a web service, using net/http

Cool Projects

- NES simulator
- https://github.com/gizak/termui
- https://github.com/peco/peco
- https://github.com/coreos/etcd
- https://github.com/schachmat/wego
- https://github.com/chrislusf/seaweedfs
- https://github.com/minio/minio
- http://nsq.io/

Web frameworks

- gorilla
- echo
- ...

Installation

- https://golang.org/doc/install

Examples

- concurrent program
- web service
- chat server
- docker storage plugin
- docker api example