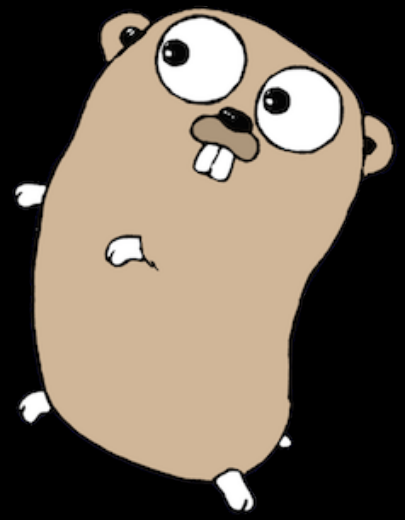
Golang from Scala developer's perspective

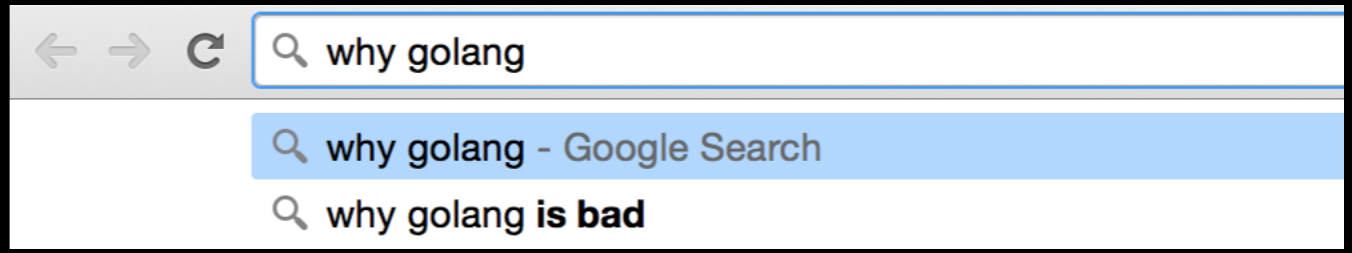


About me

Svetlana Bozhko
Backend Scala Developer @ Adform

The goal

Is Go really so ~~bad~~ imperative?



Content

Short intro into Go

Tooling

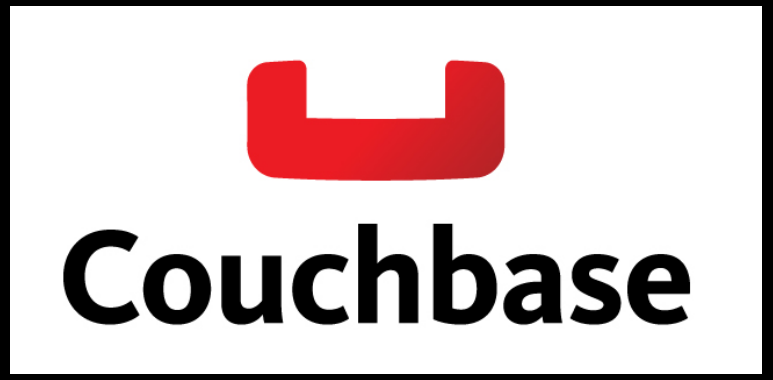
FP vs. Go

IMHO

Usage

https://github.com/golang/go/wiki/GoUsers

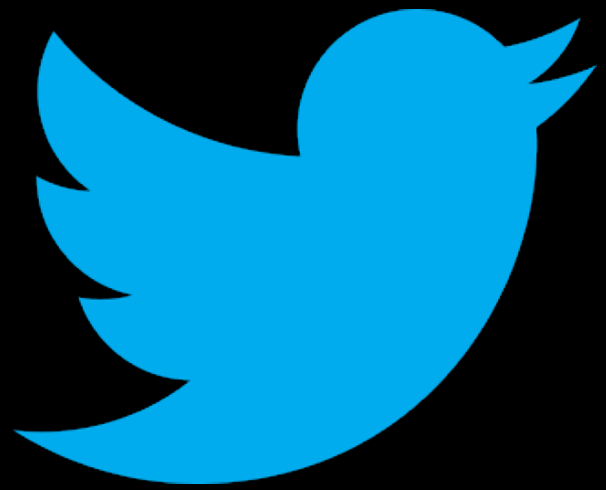

Google


Couchbase


UBER


Dropbox


Booking.com




heroku


AEROSPIKE


IBM

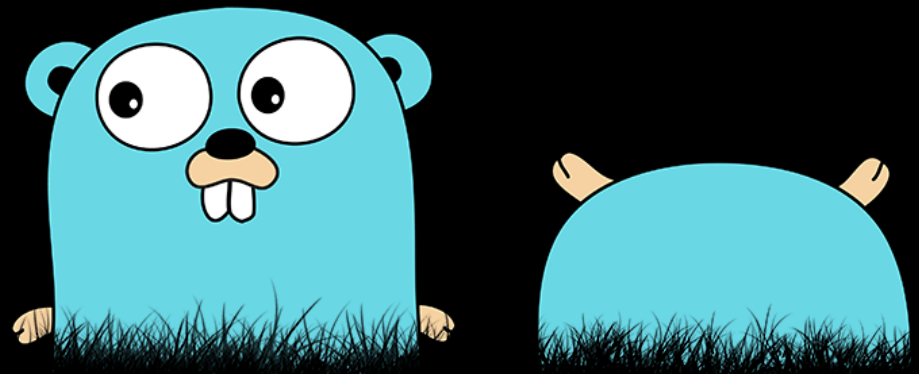
How to start?

http://tour.golang.org/

http://golangshow.com/

Go

Good parts



Go is actually a pretty simple language!

Even too simple for 2015

Paul Phillips
@extempore2

Читать

Rust and Scala drown you in complexity. Go drowns you in simplicity.

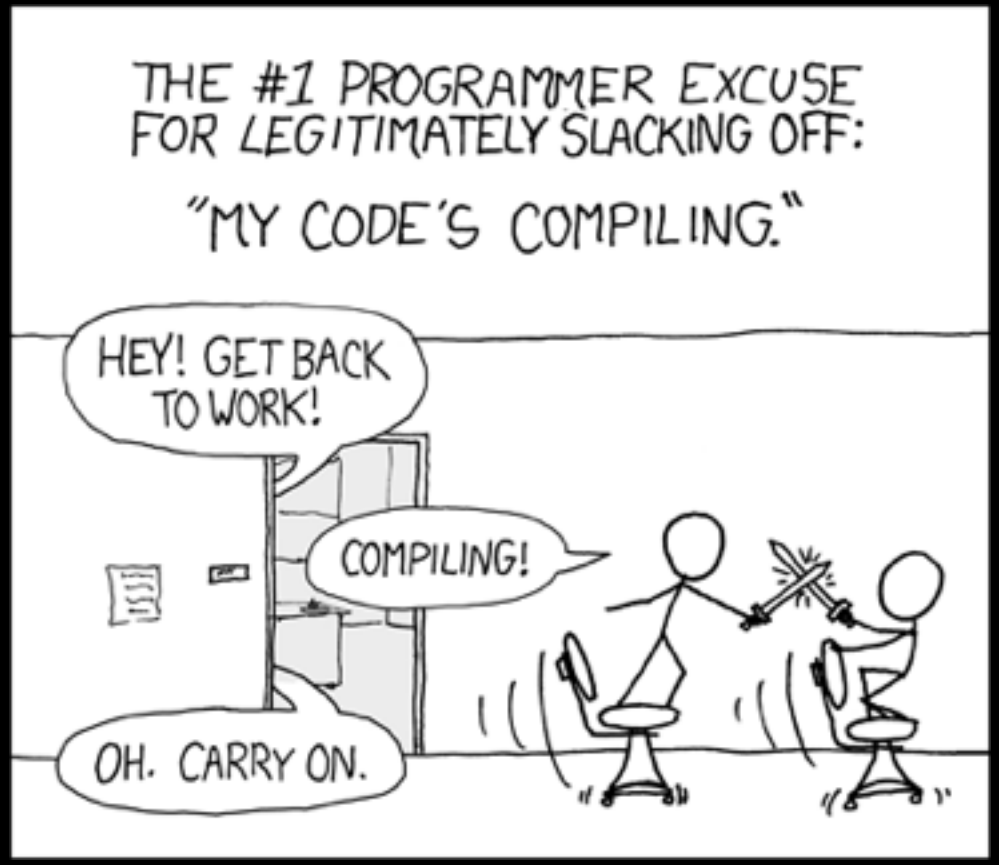
Paul Phillips
@extempore2

Читать

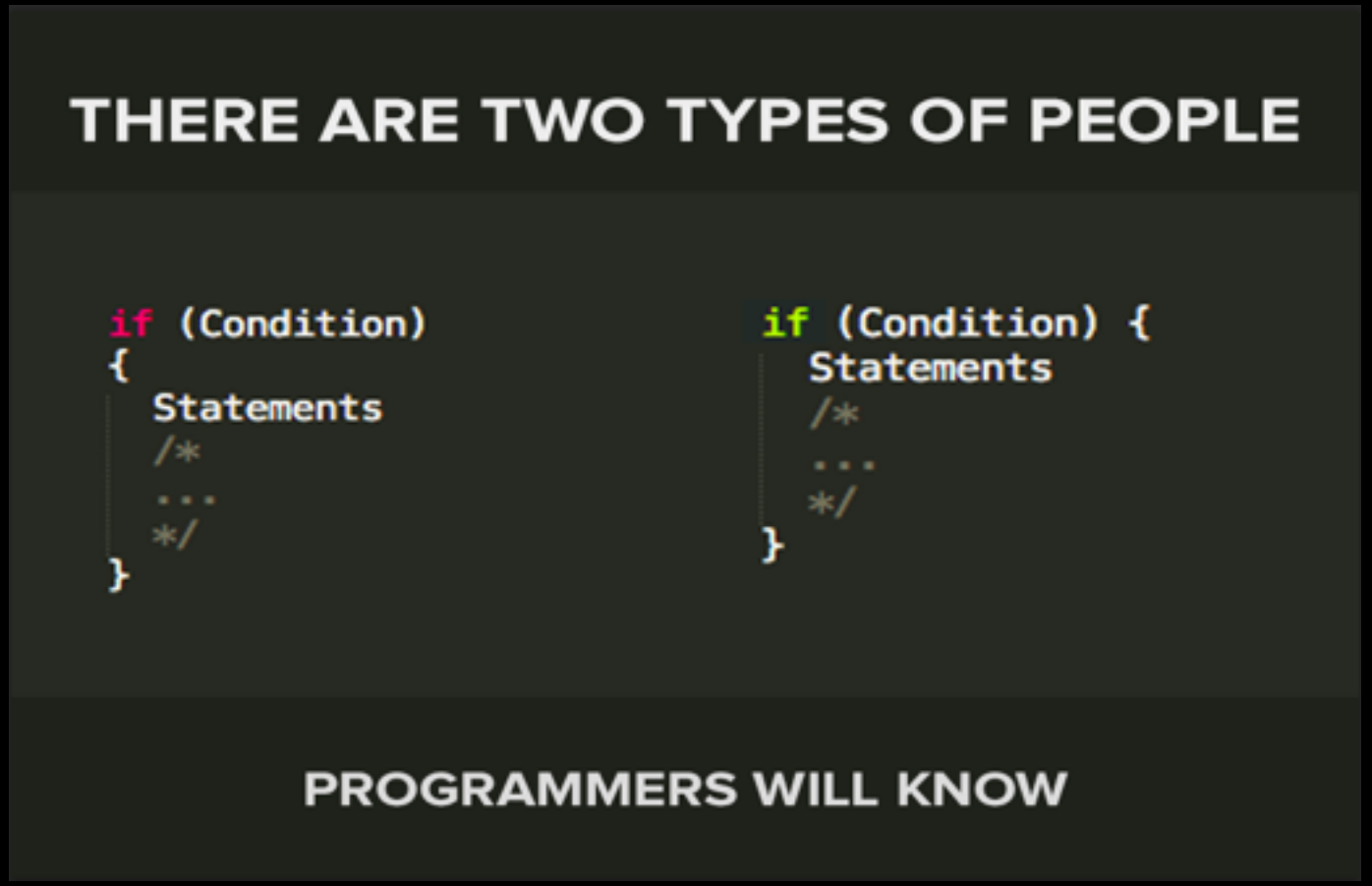
You often see languages which are fighting the last war. Go is fighting the War of 1812.

Advanced GC

“Build time matters more than anything else” - Rob Pike



gofmt


THERE ARE TWO TYPES OF PEOPLE
if (Condition)
{
Statements
/*
...
*/
}
if (Condition) {
Statements
/*
...
*/
}
PROGRAMMERS WILL KNOW

Quite good tools

build-in (version, get, build, …)

editors (vim, intellij, sublime, …)

Concurrency

- Channels

send data between threads easily, asynchronously or synchronously

- GoRoutines

spin up another thread, just like that!

No Language Is Perfect...

Go Mentality

- Smallest possible feature set
- It’s okay to copy some code
- Productivity instead of hyper-elegant code

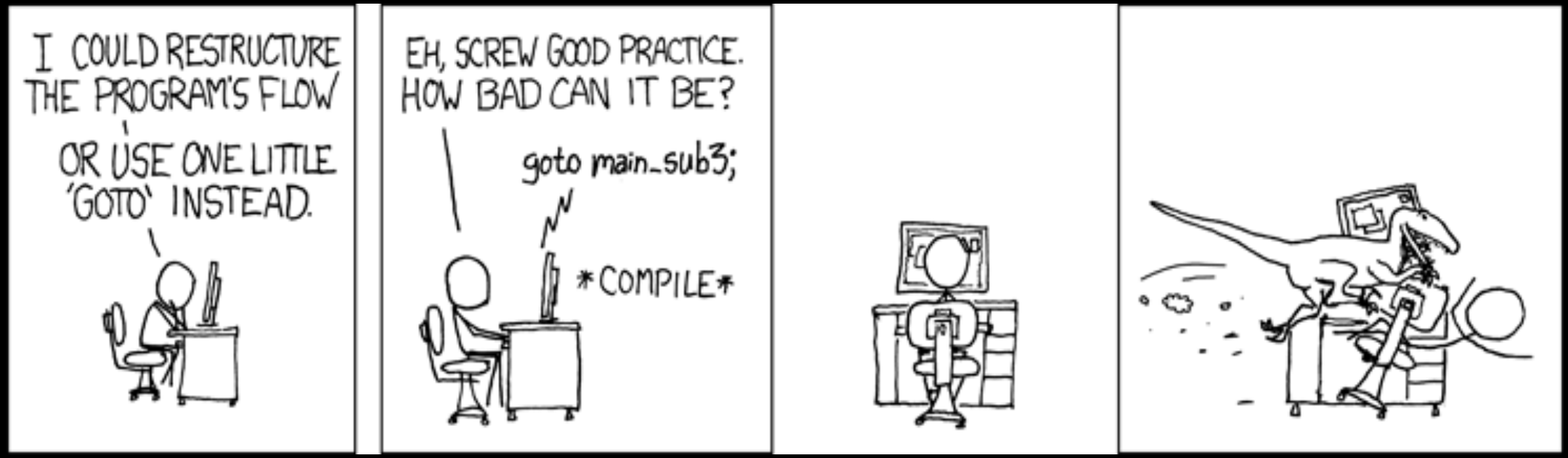
Let’s get rid of

- Implicit numeric conversion (int != int32)
- ‘Implements’ keyword
- ‘Classes’
- Constructor/destructor
- Function overloading
- Pointer arithmetic
- Exceptions and try/catch
- Generics
- …

Strict compiler

which interrupts you sometimes

Wait, Go has ‘goto’?



Wait, how to debug?

GDB, but it’s not very good way

Dependency management

vendor (since 1.5)

FP vs. Go

Wikipedia:

“Functional programming is a programming paradigm — a style of building the structure and elements of computer programs — that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.”

What it means?

No mutable data (no side effect)

No mutable state (no implicit, hidden state)

Same result returned by functions called with the same inputs

Functions are pure functions in the mathematical sense

Advantages?

Cleaner code

Referential transparency

Referential transparency

Memoization

Cache results

Idempotence

Modularization

We have no state

Ease of debugging

Functions are isolated and very easy to debug

Referential transparency

Parallelization

Functions calls are independent

We can parallelize in different process/CPUs/computers/...

res = func1(a, b) + func2(a, c)

Referential transparency

With no shared data, concurrency gets a lot simpler:

No semaphores

No monitors

No locks

No race-conditions

No dead-locks

Nonetheless

Golang has built-in “sync” package with mutexes

Don’t Update, Create!

String

```
name := "Functional"
name = name + " Vilnius"


const firstName = "Functional"
const lastName = "Vilnius"
const name = firstName + " " + lastName
```

Arrays

```
languages := [4]string{"Java", "Scala"}
languages[2] = "Clojure"
languages[3] = "Groovy"
languages // ["Java", "Scala", "Clojure", "Groovy"]



languages := []string{"Java", "Scala"}
allLanguages := append(languages, "Kotlin", "Clojure", "Groovy")
fmt.Println(allLanguages)
```

Maps

```
ages := map[string]int{"Bob": 32}
ages["Sally"] = 33
```

```
func mergeMaps(mapA, mapB map[string]int) map[string]int {
    allAges := make(map[string]int, len(mapA)+len(mapB))
    for k, v := range mapA {
        allAges[k] = v
    }
    for k, v := range mapB {
        allAges[k] = v
    }
    return allAges
}

ages1 := map[string]int{"Bob": 32}
ages2 := map[string]int{"Sally": 33}
allAges := mergeMaps(ages1, ages2)
```

Higher order functions

```
func printingFunction(f func(string) string) {
    result := f("Vilnius")
    fmt.Println(result)
}

f := func(s string) string {
    return "Hello " + s
}

printingFunction(f)
```

FP libraries

https://github.com/yanatan16/itertools

https://github.com/tobyhede/go-underscore

Higher order functions

Map

```
s := []string{"Functional", "Vilnius"}

fn := func(s interface{}) interface{} {
  return s.(string) + "!"
}

m := un.Map(ToI(s), fn)
fmt.Println(m) //["Functional!", "Vilnius!"]
```

Filter

```
predicate := func(i interface{}) bool {
    return i.(uint64) > 7
}
Filter(predicate, Uint64(5, 6, 7, 8, 9, 10))
//[8, 9, 10]
```

Reduce

```
accum := func(oneArg interface{}, anotherArg interface{}) interface{} {
    return len(oneArg.(string)) + len(anotherArg.(string))
}
Reduce(New("test", "project"), accum, string).(uint64)
// result 11
```

Closures

```
add6 := addValue(6)
add9 := addValue(9)
add6(10) // result 16
add9(10) // result 19


func addValue(a int) func() int {
    return func(b int) int {  // anonymous function
        return a + b }
}
```

Currying and Partial Functions

```
func plusFunction(x, y int) int {
    return x + y
}

func partialPlusFunction(x int) func(int) int {
    return func(y int) int {
        return plusFunction(x, y)
    }
}

func main() {
    plusFive := partialPlusFunction(5)
    fmt.Println(plusFive(10)) //prints 15
}
```

Eager vs Lazy Evaluation

Golang channels and goroutines enable the creation of generators that could be a way to have lazy evaluation

Recursion

> Do you plan to add tail call optimization to Go eventually?

It is already there in 6g/8g for certain cases, and in gccgo somewhat more generally.

We do not currently plan to change the language to require that compilers implement tail call optimization in all cases. If you must have a tail call, you use a loop or a goto statement.

Ian

FP & OOP

It is possible do FP in OOP?

Yes it is!

OOP is orthogonal to FP. At least in theory…

Exercise!

What is the sum of the first 10 natural number whose square value is divisible by 2?

Exercise!

Imperative

```
func main() {
    n, numElements, s := 1, 0, 0
    for numElements < 10 {
        if n * n % 2 == 0 {
            s += n
            numElements++
        }
        n++ }
    fmt.Println(s)
}
```

Functional

```
sum := func(memo interface{}, el interface{}) interface{} {
    return memo.(float64) + el.(float64)
}
pred := func(i interface{}) bool {
    return (i.(uint64) * i.(uint64)) % 2 == 0
}
values := make([]int, 100)
for num := 1; num <= 100; num++ {
    values = append(values, num)
}
Reduce(Filter(pred, values), sum, uint64).(uint64)
```

Learn at least one functional language

It will open your mind to a new paradigm becoming you a better programmer

Conclusion

Technologies are awesome!

The problem is ~~people~~ how we use them.

Thank you!

Questions?

Svetlana Bozhko aka @SBozhko

http://devzen.ru/

svt.bozhko@gmail.com

Links

https://golang-ru.slack.com/messages

https://www.youtube.com/watch?list=PLPHSBhlVtTyfwIKn7r_a5xkzzMu-iey-w&v=cGXorQkw3JE