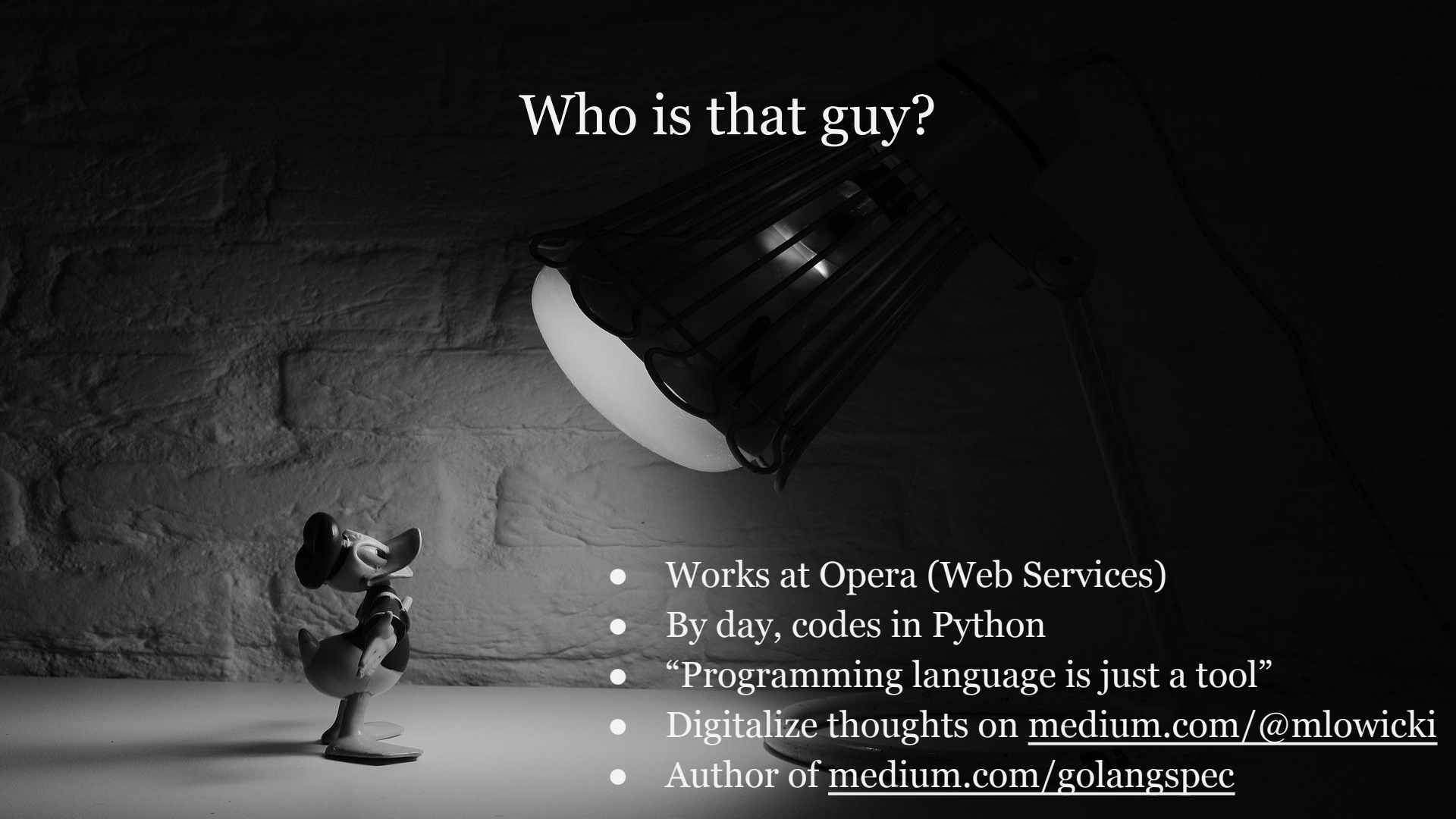
Starter pack for programming in Golang

Michał Łowicki

Who is that guy?



- Works at Opera (Web Services)
- By day, codes in Python
- “Programming language is just a tool”
- Digitalize thoughts on medium.com/@mlowicki
- Author of medium.com/golangspec

Why Go?

Works really well for creating network services because of built-in concurrency and standard library with support for many protocols.

Basic facts

- Open sourced in 2009
- Current version is 1.7.4
- Compiled (single binary with compiled-in runtime)
- Garbage collector included
- Statically typed
- Concurrent
- Built-in Unicode support

New face of the old friend

```
package main

import "fmt"

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

https://play.golang.org/p/uHYbRyPaZ_

Anatomy of .go file

```
package foo // package name (required)

// zero or more import declarations

import (
    "fmt"
    "strings"
)
import "crypto/md5"

// zero or more top-level declarations

func Foo(input string) {
    input = strings.ToTitle(input)
    fmt.Printf("md5(%s) == %x\n", input, md5.Sum([]byte(input)))
}
```

https://medium.com/golangspec/anatomy-of-go-source-file-3550d1f3608d

Development environment

- go tool operates on workspaces
- workspace usually contains many repositories (like Git)
- each repo contains packages (one or many)

How to Write Go Code

```
> pwd
/Users/mlowicki/projects
> mkdir mygo && cd $_
> export GOPATH=/Users/mlowicki/projects/mygo
> mkdir -p $GOPATH/src/github.com/mlowicki/hello
> vim $GOPATH/src/github.com/mlowicki/hello/hello.go
...
> cat $GOPATH/src/github.com/mlowicki/hello/hello.go
package main

import "fmt"

func main() {
      fmt.Println("hello world!")
}
> go install github.com/mlowicki/hello
> tree
.
├── bin
│   └── hello
└── src
    └── github.com
        └── mlowicki
            └── hello
                └── hello.go

5 directories, 2 files
> $GOPATH/bin/hello
hello world!
```

play.golang.org

Executable’s entry point

```
package main

import "github.com/mlowicki/foo"

func main() {
    foo.Foo("bar")
}
```

Automatic semicolon insertion

Go's lexer frees programmer from typing semicolons

```
>  go tool compile -x src/github.com/mlowicki/hello/hello.go
lex: PACKAGE
...
lex: ident main
lex: TOKEN '('
lex: TOKEN ')'
lex: TOKEN '{'
lex: ident fmt
lex: TOKEN '.'
lex: ident Println
lex: TOKEN '('
lex: string literal
lex: TOKEN ')'
lex: implicit semi
lex: TOKEN '}'
lex: implicit semi
```

Automatic semicolon insertion in Go

```
package main

import "fmt"

func main() {
    if true
    {
       fmt.Println("whatever")
    }
}

> go install github.com/mlowicki/hello
src/github.com/mlowicki/hello/hello.go:6: true evaluated but not used
src/github.com/mlowicki/hello/hello.go:7: missing condition in if statement

> go tool compile -x $GOPATH/src/github.com/mlowicki/hello/hello.go
…
lex: IF
lex: ident true
lex: implicit semi
…
```

```
package main

import "fmt"

func main() {
    if true {
        fmt.Println("whatever")
    }
}
```

```
> go install github.com/mlowicki/hello/
> $GOPATH/bin/hello
whatever
```

lexical vs dynamic scoping

Dynamic scoping

```
> cat scope.sh
#!/bin/bash

x=1

function g () {
    echo $x
}

function f () {
    local x=2
    g
}

f
> ./scope.sh
2
```

Lexical scoping

```
package main

import "fmt"

func main() {
    x := 1

    g := func() {
        fmt.Println(x)
    }

    f := func() {
        x := 2
        _ = x // to avoid "declared and not used" error
        g()
    }

    f()
}
```

https://play.golang.org/p/skMoy1bOnA

Function

```
func IsEven(number int64) bool {
    return number % 2 == 0
}

func Filter(numbers []int64, filter func(int64) bool) []int64 {
    res := make([]int64, 0, len(numbers))
    for _, number := range numbers {
        if filter(number) {
            res = append(res, number)
        }
    }
    return res
}

…
Filter([]int64{1, 2, 3, 4, 5}, IsEven)  // [2 4]
```

https://play.golang.org/p/n0E55Xcobb

Closure

```
func DeltaX(delta int64) func(int64) int64 {
    return func(base int64) int64 {
        return base + delta
    }
}

func main() {
    nums := [...]int64{1,2,3,4,5}
    delta5 := DeltaX(5)
    for idx, num := range nums {
        nums[idx] = delta5(num)
    }
    fmt.Println(nums) // [6 7 8 9 10]
}
```

https://play.golang.org/p/OoThKvy9ro
https://en.wikipedia.org/wiki/Closure_(computer_programming)

Control statements (some)

```
for i := 0; i < 3; i++ {
    fmt.Println(i)
}
var i int
for i < 3 {
    fmt.Println(i)
    i += 1
}
i = 0
for {
    if i > 2 {
        break
    }
    fmt.Println(i)
    i += 1
}
```

https://play.golang.org/p/QIzrbdvS_9

if with simple statement

```
package main

import "fmt"

func f() int {
    return 11
}

func main() {
    if res := f(); res > 10 {
        fmt.Println(res)
    }
}
```

https://play.golang.org/p/wMiN6g6znN

https://medium.com/golangspec/simple-statement-notion-in-go-b8afddfc7916

Map

A map is an **unordered** group of elements of one type, indexed by a set of unique keys of another type.

```
counters := make(map[string]int64)
counters["foo"] = 1
counters["bar"] += 2
fmt.Println(counters) // map[foo:1 bar:2]
delete(counters, "bar")
fmt.Println(counters) // map[foo:1]
fmt.Println(counters["bar"]) // 0
if _, ok := counters["bar"]; !ok {
    fmt.Println("'bar' not found")
}
counters["bar"] = 2
for key, value := range counters {
    fmt.Printf("%s: %v\n", key, value) // order is randomized!
}
```

https://play.golang.org/p/xqgws9G-gH

Array

- Sequence of elements of a single type
- Length is part of array's type

```
languages := [4]string{"Go", "Python", "JavaScript", "C"}
languages[3] = "C++"
fmt.Println(languages) // [Go Python JavaScript C++]
fmt.Println(len(languages)) // 4
maze := [3][3]int{{0, 0, 0}, {0, 1, 0}, {1, 1, 2}}  // same as [3]([3]int)
fmt.Println(maze) // [[0 0 0] [0 1 0] [1 1 2]]
for idx, value := range maze {
    fmt.Printf("%d: %v\n", idx, value)
}
// 0: [0 0 0]
// 1: [0 1 0]
// 2: [1 1 2]
```

https://play.golang.org/p/9ebr6F8wLe

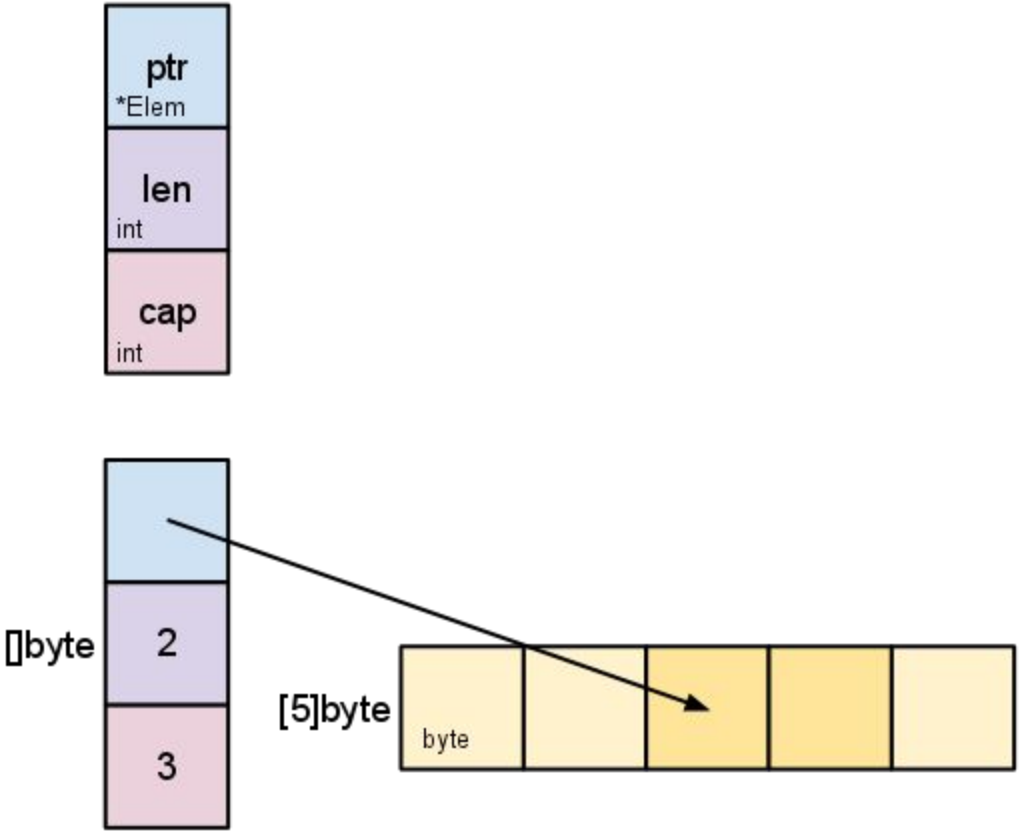
Slice

```
numbers := []int64{5, 4, 3, 2, 1}
fmt.Println(len(numbers))  // 5
fmt.Println(cap(numbers))  // 5
numbers[4] = 0
fmt.Println(numbers) // [5 4 3 2 0]

numbers = numbers[1:4]
fmt.Println(len(numbers))  // 3
fmt.Println(cap(numbers))  // 4
fmt.Println(numbers)       // [4 3 2]

numbers = numbers[:4]
fmt.Println(numbers) // [4 3 2 0]
```

Go Slices: usage and internals



[Go Slices: usage and internals]

Struct

Sequence of named elements (fields). Each field has name and type.

…

```
type T struct {
    name  string
    age   int64
    attrs map[string]string
}

func main() {
    t := T{name: "Michał", age: 31, attrs: map[string]string{"type":
"geek"}}
    t.age -= 1
    t.attrs["city"] = "Wrocław"
    fmt.Println(t)  // {Michał 30 map[type:geek city:Wrocław]}
}
```

https://play.golang.org/p/9-yJtiSyLy

Method

```
package main

import "fmt"

type Person struct {
    name string
    age  int64
}

func (p *Person) IsAdult() bool {
    return p.age >= 18
}

func main() {
    me := Person{"Michał", 31}
    fmt.Println(me.IsAdult())  // true
}
```

https://play.golang.org/p/Wl8oJE_1uZ

Embedding fields

```
…

type Employee struct {
    department string
    Person
}

func main() {
    person := Person{"Michał", 31}
    employee := Employee{department: "Web Services", Person: person}
    fmt.Println(employee.IsAdult())  // true
}
```

https://play.golang.org/p/gR3GEf1L35

Concurrent programming language

Language shipped with features supporting concurrent programming like statement to run things concurrently or mechanism to easily communicate between such concurrent things.

concurrency vs. parallelism

Concurrency Is Not Parallelism (by Rob Pike)

Program implemented in Go during its execution consists of one or more *goroutines*

https://medium.com/golangspec/panicking-like-a-gopher-367a9ce04bb8

Goroutine

Independent, concurrent activity managed by Golang runtime's scheduler and multiplexed on OS threads.

```
package main

import "net/http"

func Download() {
    http.Get("https://httpbin.org/delay/5")
}

func main() {
    Download()
    Download()
}

> time ./bin/noconcurrency

real    0m12.176s
user    0m0.102s
sys     0m0.025s
```

```
package main

import "net/http"

func Download() {
    http.Get("https://httpbin.org/delay/5")
}

func main() {
    ch := make(chan struct{})
    go func() {
        Download()
        ch <- struct{}{}
    }()
    Download()
    <-ch
}

> time ./bin/concurrency

real    0m5.937s
user    0m0.109s
sys     0m0.019s
```

Channel

```
package main

import "time"

func Producer(output chan<- int) {
    for i := 0; i < 10; i++ {
        time.Sleep(100 * time.Millisecond)
        output <- i
    }
    close(output)
}

func Consumer(input <-chan int, done chan struct{}) {
    for {
        _, ok := <-input
        if !ok {
            break
        }
        time.Sleep(300 * time.Millisecond) // consuming...
    }
    done <- struct{}{}
}
```

```
func main() {
    ch := make(chan int)
    done := make(chan struct{})
    go Producer(ch)
    go Consumer(ch, done)
    <-done
}
```

```
> time ./bin/noconcurrency

real    0m3.129s
user    0m0.002s
sys     0m0.006s
```

```
func main() {
    ch := make(chan int)
    done := make(chan struct{})
    go Producer(ch)
    go Consumer(ch, done)
    go Consumer(ch, done)
    go Consumer(ch, done)
    <-done
}
```

```
> time ./bin/concurrency

real    0m1.126s
user    0m0.001s
sys     0m0.006s
```

```
func main() {
    ch := make(chan int)
    done := make(chan struct{})
    go Producer(ch)
    go Consumer(ch, done)
    go Consumer(ch, done)
    go Consumer(ch, done)
    <-done
    fmt.Println(runtime.NumGoroutine())  // 3
}
```

https://medium.com/golangspec/goroutine-leak-400063aef468

The main goroutine

```
package main

import (
    "fmt"
    "time"
)

func main() {
    go func() {
        time.Sleep(1 * time.Second)
        fmt.Println("I'm done!")
    }()
}
```

https://play.golang.org/p/elNw8YydIs

Fix #1

```
package main

import (
    "fmt"
    "time"
)

func main() {
    ch := make(chan struct{})
    go func() {
        time.Sleep(1 * time.Second)
        fmt.Println("I'm done!")
        ch <- struct{}{}
    }()
    <-ch
}
```

https://play.golang.org/p/aOW-5o_pC-

Fix #2

```
package main

import (
    "fmt"
    "sync"
    "time"
)

func main() {
    var wg sync.WaitGroup
    wg.Add(1)
    go func() {
        defer wg.Done()
        time.Sleep(1 * time.Second)
        fmt.Println("I'm done!")
    }()
    wg.Wait()
}
```

https://play.golang.org/p/dnqYvqe9gZ

Share memory by communication

Don’t communicate by sharing memory

Interface

```
type I interface {
    transform(int) int
}

type T struct {}

func (t T) transform(input int) int {
    return input + 1
}

func f(i I) {
    fmt.Println(i.transform(1))
}

func main() {
    f(T{}) // 2
}
```

https://play.golang.org/p/78lOuIVVF8

```
if youAreLookingForInternship || youAreLookingForJob {
    go email("mlowicki@opera.com")
}

if youAreLookingForInterestingReading {
   go browse("https://medium.com/golangspec")
}

if youHaveQuestions {
    go justAskNow()
}
```


FROM NOW ON,
YOU'RE ON YOUR OWN!


It’s just a beginning…

Ideas for future talk(s)

- blocks (scoping)
- synchronization
- concurrent patterns
- buffered channels
- select statement
- how to write network service (talking HTTP or rough TCP)
- error handling
- testing
- go get
- more on slices (f.ex. append or copy built-ins)
- ...