PUZZLE ITC

changing IT for the better

Techworkshop 2017: Go

Khôi Tran

Goals (by the end of the day)

Know what Go is (and isn't)

How to set up a go environment

Able to write and compile a go program

Know where to go from there

Online resources (presentation, links):

https://github.com/KeeTraxx/pitc-go-workshop

Block 1 (10:00 - 11:30) : Introduction + Go tooling installation

Block 1 (11:30 - 12:00) : Go Hands-On Introduction

Lunch

Block 2 (14:00 - 15:30) : Go Hands-On

Coffee break

Block 3 (15:50 - 16:40) : Wrap-Up and Presentation

Agenda

1

Go introduction

Brief history of Go

2007: Created at Google by Robert Griesemer, Rob Pike, and Ken Thompson

2009: Announcement, Open Source (BSD-style license)

2012: Go 1.0

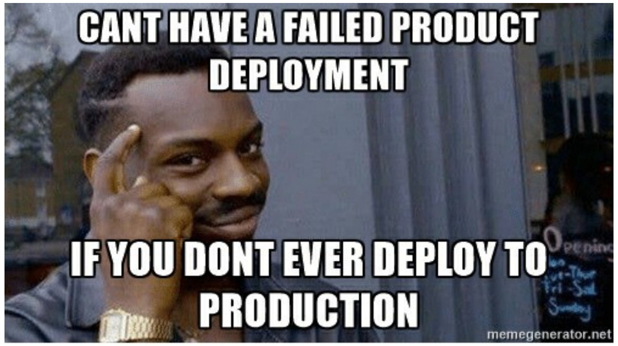
2015: Go 1.5, written in Go, no more C

2016: Go 1.6, vendoring, built in race detector

Present: Current stable release: Go 1.9

Who uses Go in production?

- Google: Kubernetes, Youtube, dl.google.com, ...
- GitHub: Releases API upload endpoint
- Facebook: several internal libraries
- Twitter: REST Endpoint
- StackExchange: Monitoring and alerting system (bosun.org)
- SoundCloud: several internal libaries, build & deployment system
- Netflix: Rend - Memcached-Compatible Server & Proxy
- Dropbox: Caching, Error Handling, SqlBuilder, memcache, ...
- Docker: Almost all of it


CANT HAVE A FAILED PRODUCT DEPLOYMENT
IF YOU DONT EVER DEPLOY TO PRODUCTION
memegenerator.net

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

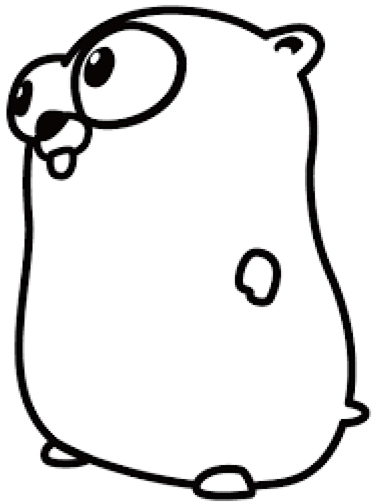
Go in Switzerland?

- Google
- Centralway
- Swisscom
- Tamedia
- Mobiliar
- PostFinance
- SmallPdf
- Vshn

Go language features

- Syntax derived from: C, Pascal/Oberon (Declaration, Packages)
- Compiled
  (mind CPU architecture and libc vs musl!)
- Statically typed language
- Garbage collection

Go build-in tools



- Testing go test

```
go test
```

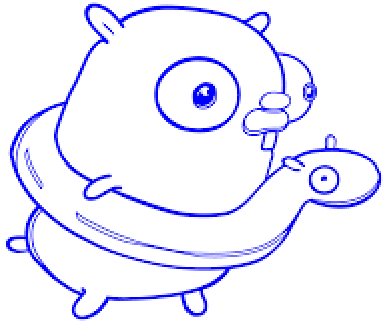
- Benchmarking

```
go test -bench=.
```

- Crosscompiling

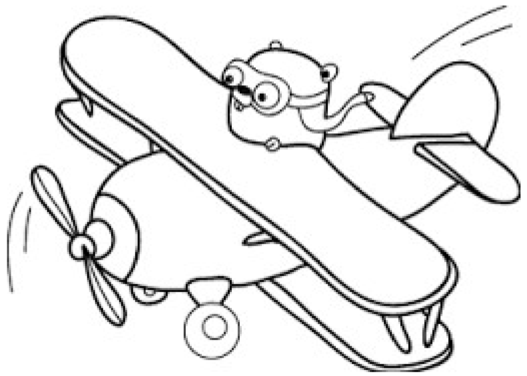
```
GOOS="linux" GOARCH="arm" \
  go build -v .
```

Go official tools



gofmt	format source code
godoc	show, generate and serve documentation
goimports	gofmt + imports cleanup
gorename	type-safe renaming
lint	go linting
cover	generate test coverage reports
race detector	find race conditions

Go 3rd party tools

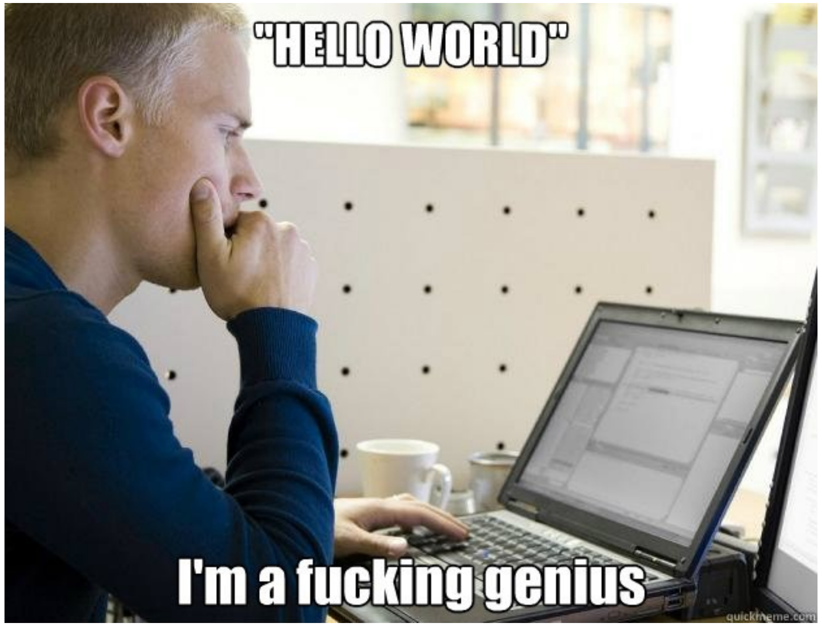


gometalinter	run several checkers / linters in parallel
gocode	autocompletion daemon
godef	track symbol definitions
deadcode	unused code checker
gocyclo	computes cyclomatic complexity
varcheck	find unused global variables and constants
structcheck	find unused struct fields
errheck	finds unceсked error return values
dupl	finds potentially duplicated code
ineffassign	detect unused assignments
unconvert	detect redundant type conversions
goconst	detect repeated strings that could be replaced with constants

Recommended IDE: vscode (as of 2017)

- Completion Lists (using gocode)
- Signature Help (using gogetdoc or godef+godoc)
- Quick Info (using gogetdoc or godef+godoc)
- Goto Definition (using gogetdoc or godef+godoc)
- Find References (using guru)
- File outline (using go-outline)
- Workspace symbol search (using go-symbols)
- Rename (using gorename)
- Build-on-save (using go build and go test)
- Lint-on-save (using golint or gometalinter)
- Format (using goreturns or goimports or gofmt)
- Generate unit tests skeleton (using gotests)
- Add Imports (using gopkgs)
- Add/Remove Tags on struct fields (using gomodifytags)
- Semantic/Syntactic error reporting as you type (using gotype-live)
- Run Tests under the cursor, in current file, in current package, in the whole workspace (using go test)
- Generate method stubs for interfaces (using impl)
- Debugging (using delve)

Go hello world



Go syntax - odd stuff

- **no semicolons**
- **raw string literals with backticks**

```
quote := `A "quaterpounder" with cheese`
quote := "A \"quaterpounder\" with cheese"
```

- **no ? operator**

~~`nextmovie := likesScifi ? "Mr. Robot" : "House of Cards"`~~

- **multiline statements must have operator before newline**

```
fmt.Printf("%d %d",
 "one",
 "two",
)
```

- **no while or do loops, just for**
- **variable declaration, name before type**

```
var countryNames []string
```

Go syntax - odd stuff (2)

- **switch statements don't need `break` but have `fallthrough`**

```
switch status {
  case "asleep":
    wakeUp()
  case "tired":
    drinkCoffee()
    fallthrough
  case "awake":
    doWork()
  case "lazy":
    motivate()
}
```

- **`null` is `nil` and only pointers can be `nil`**

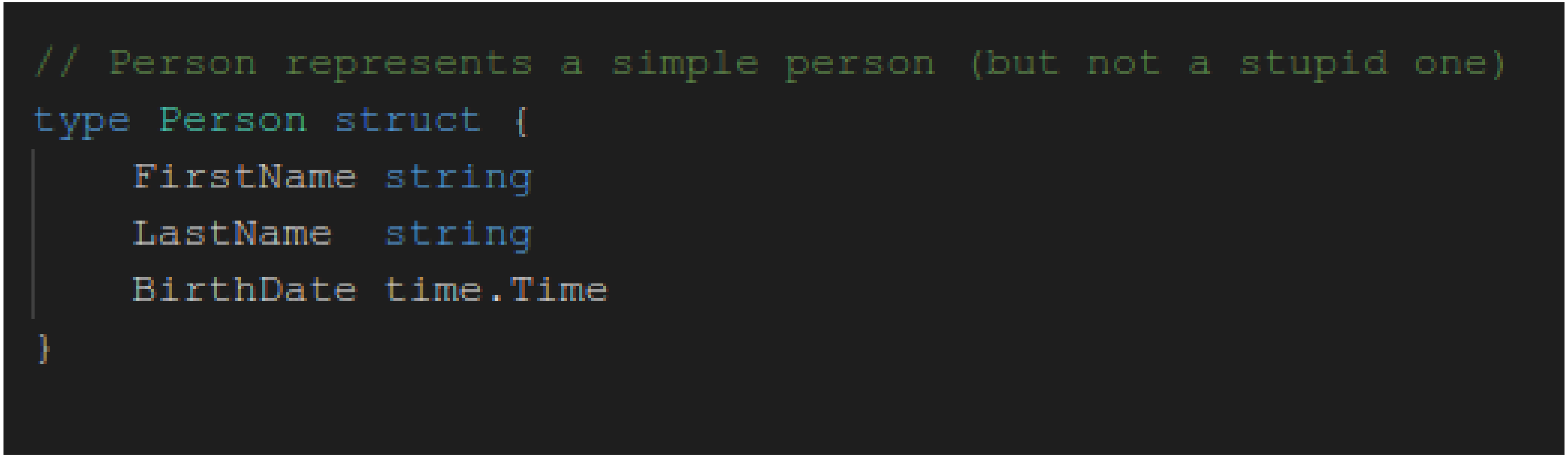
  `(or pointer-like structures like pointers, interfaces, maps, slices, channels and function types)`

```
var s string // s == ""

var sptr *string // sptr == nil

var i int // i == 0

var i *int // i == nil
```

Go language properties

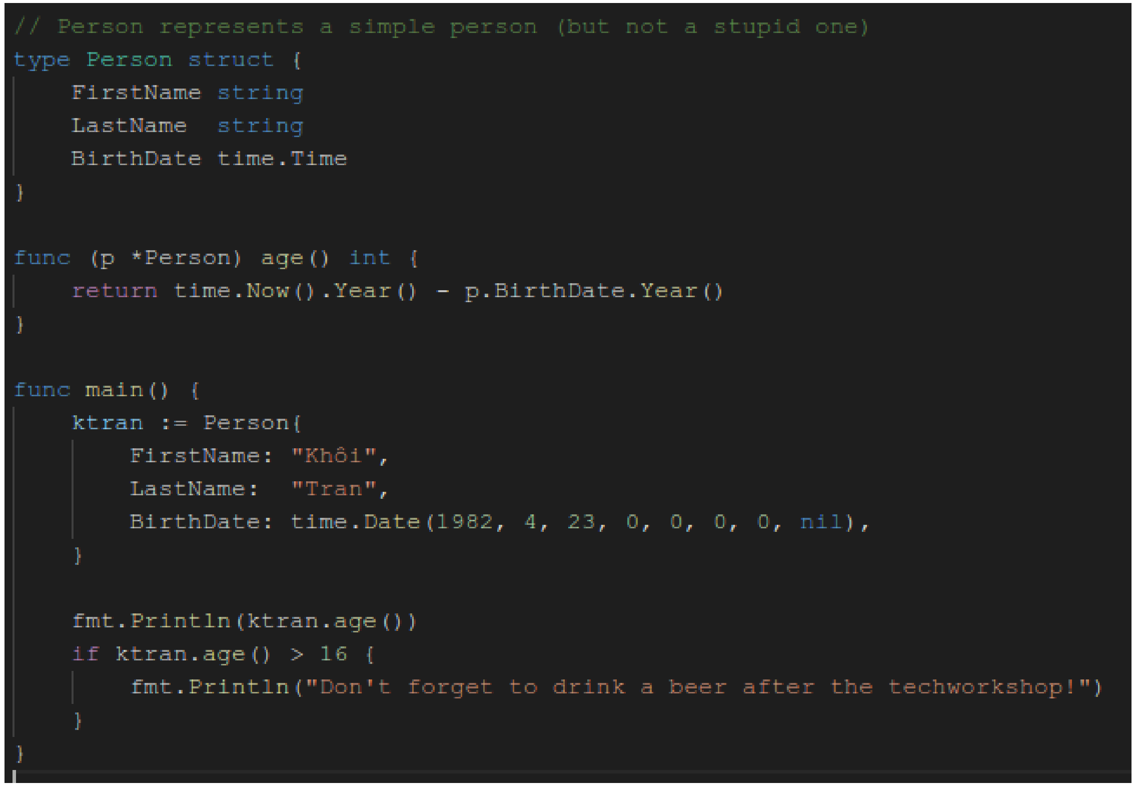
- **Struct and struct methods**
- **Field tags**
- **Error handling**
- **Multiple return values**
- **Slices (aka Arrays) and Maps**
- **Pointers** (but no shenanigans like C)
- Packages / Imports
- Concurrency with goroutines and channels

Structs



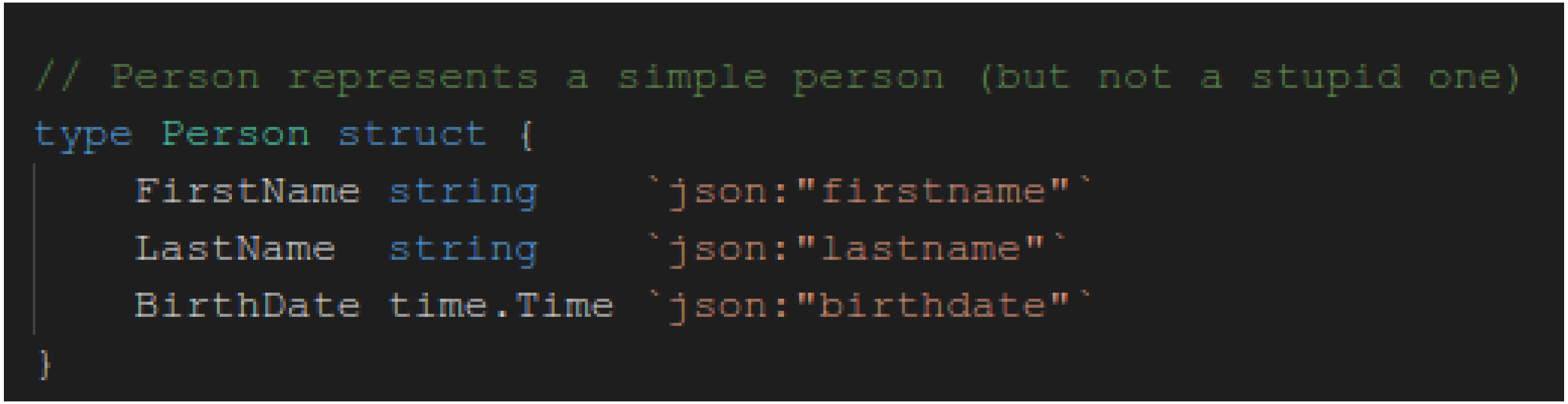
```
// Person represents a simple person (but not a stupid one)
type Person struct {
    FirstName string
    LastName  string
    BirthDate time.Time
}
```

Struct Methods



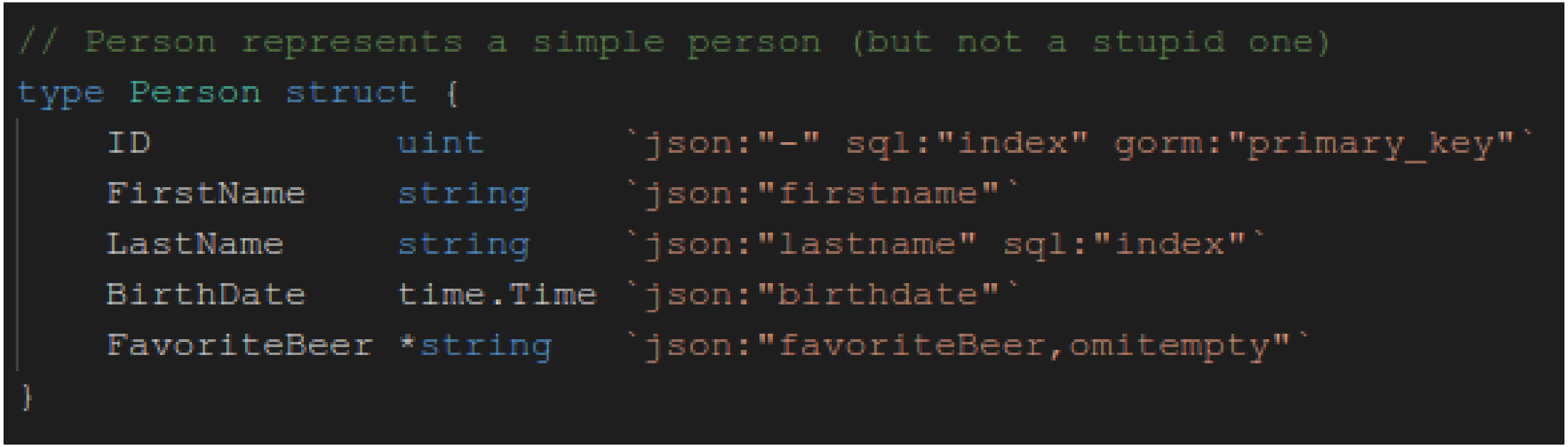
```go
// Person represents a simple person (but not a stupid one)
type Person struct {
    FirstName string
    LastName  string
    BirthDate time.Time
}

func (p *Person) age() int {
    return time.Now().Year() - p.BirthDate.Year()
}

func main() {
    ktran := Person{
        FirstName: "Khôi",
        LastName:  "Tran",
        BirthDate: time.Date(1982, 4, 23, 0, 0, 0, 0, nil),
    }

    fmt.Println(ktran.age())
    if ktran.age() > 16 {
        fmt.Println("Don't forget to drink a beer after the techworkshop!")
    }
}
```

Field tags



```go
// Person represents a simple person (but not a stupid one)
type Person struct {
    FirstName string    `json:"firstname"`
    LastName  string    `json:"lastname"`
    BirthDate time.Time `json:"birthdate"`
}
```

Field tags (adv.)



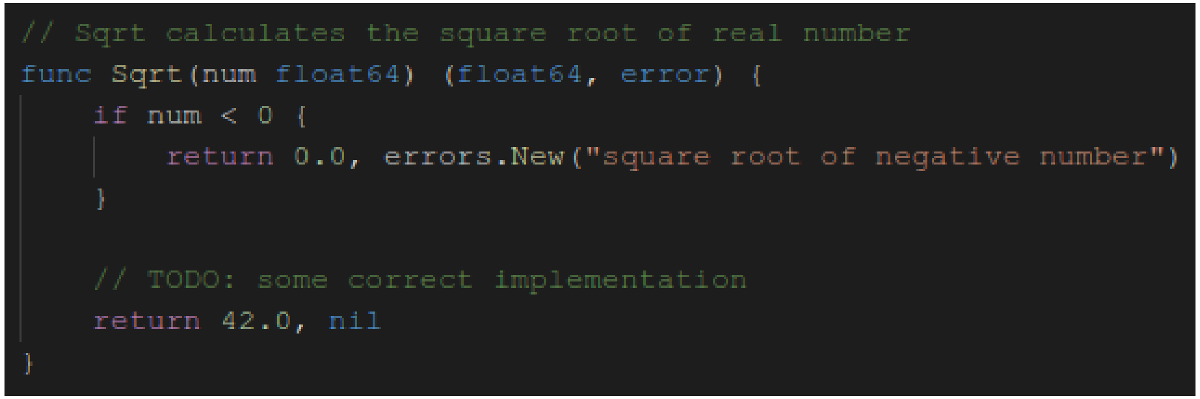
```go
// Person represents a simple person (but not a stupid one)
type Person struct {
	ID            uint      `json:"-" sql:"index" gorm:"primary_key"`
	FirstName     string    `json:"firstname"`
	LastName      string    `json:"lastname" sql:"index"`
	BirthDate     time.Time `json:"birthdate"`
	FavoriteBeer  *string   `json:"favoriteBeer,omitempty"`
}
```

Error handling

- Go has a build-in type `error`

- Functions that can fail should return an nil error on success

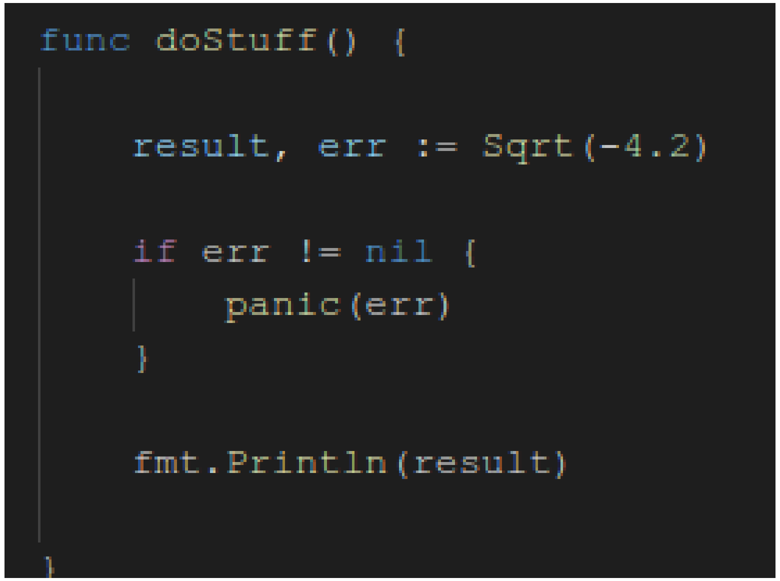
- Tip: always handle errors immediately somehow



```go
// Sqrt calculates the square root of real number
func Sqrt(num float64) (float64, error) {
    if num < 0 {
        return 0.0, errors.New("square root of negative number")
    }

    // TODO: some correct implementation
    return 42.0, nil
}
```

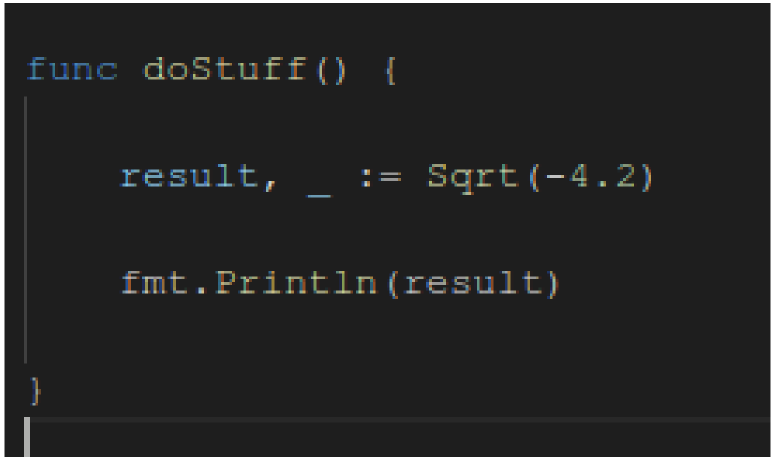
Multiple return values

- Most often used for error handling



```
func doStuff() {

    result, err := Sqrt(-4.2)

    if err != nil {
        panic(err)
    }

    fmt.Println(result)

}
```

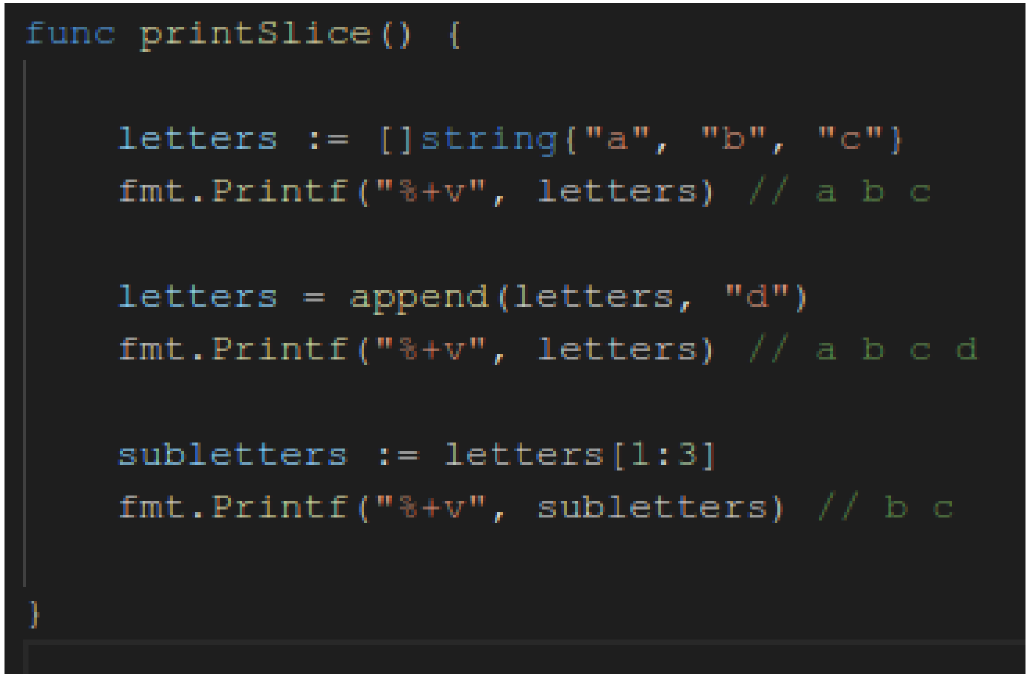
- Can be ignored using the blank identifier



```
func doStuff() {

    result, _ := Sqrt(-4.2)

    fmt.Println(result)

}
```

Slices aka the Go arrays

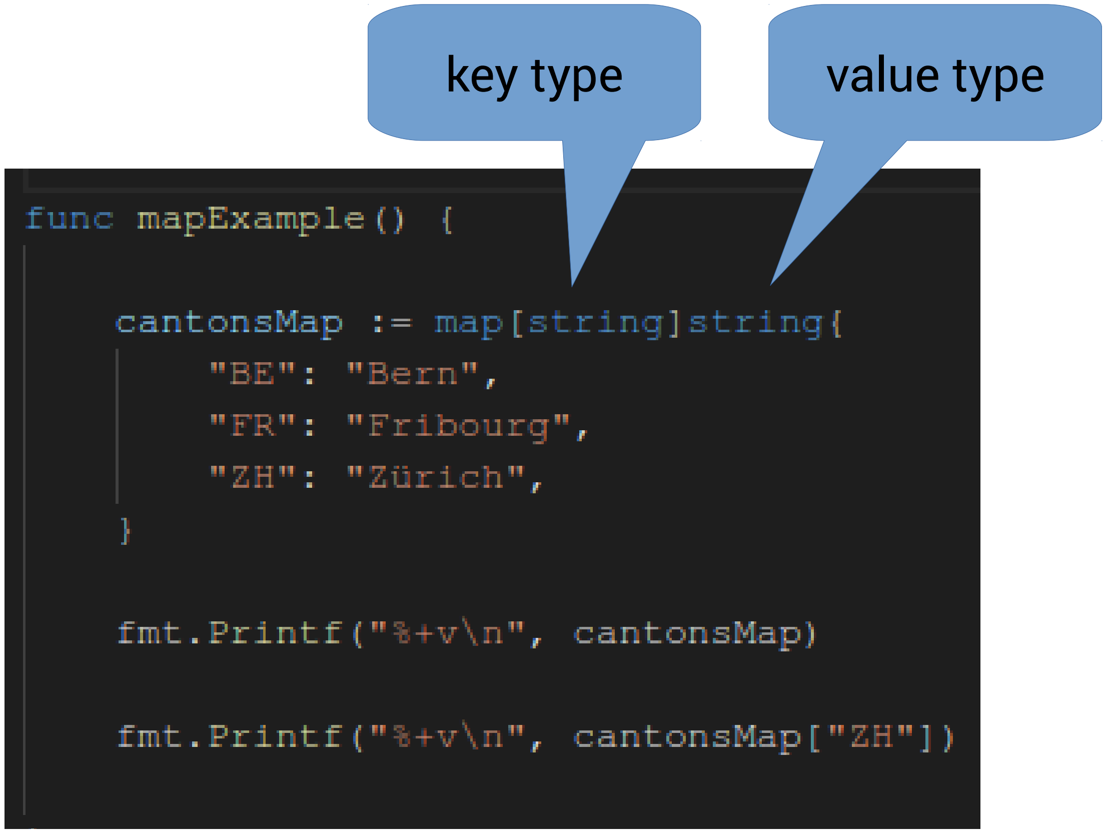
- Slices are built on top of arrays
- Slices have no fixed length
- Slices have length and capacity



```go
func printSlice() {

    letters := []string{"a", "b", "c"}
    fmt.Printf("%+v", letters) // a b c

    letters = append(letters, "d")
    fmt.Printf("%+v", letters) // a b c d

    subletters := letters[1:3]
    fmt.Printf("%+v", subletters) // b c

}
```

Maps

- Example:



```
func mapExample() {

    cantonsMap := map[string]string{
        "BE": "Bern",
        "FR": "Fribourg",
        "ZH": "Zürich",
    }

    fmt.Printf("%+v\n", cantonsMap)

    fmt.Printf("%+v\n", cantonsMap["ZH"])
```

Pointers

Similar to C pointers:

- *Foo to denote pointer type
- * Operator for deferencing
- & Operator for referencing
- No pointer arithmetics

Hints:

Variable must be modified ==> Pass pointer

Variable is a large struct ==> Pass pointer
(avoids expensive copying)

Variable is a map or slice ==> Pass value
(they are reference types already)

Setup Go Environment: gvm

Written in Go, runs on Linux, OSX, Windows

How to install:

1. https://github.com/andrewkroh/gvm#installation
2. Add `eval "$(gvm 1.9)"` to your `~/.bash_rc`
3. `mkdir ~/go`

https://github.com/andrewkroh/gv
m

Install vscode

1. https://code.visualstudio.com/docs/setup/linux
2. Install Go extension
3. CTRL-SHIFT-P ==> Go: Install/Update Tools

THE TWO STATES OF EVERY PROGRAMMER
I AM A GOD.
I HAVE NO IDEA WHAT I'M DOING.

Go Hands-On

Exercise 1: Coffee or Beer?

- GitHub Skeleton:

  https://github.com/KeeTraxx/pitc-go-coffee-or-beer.git

  1) Implement your code in coffeeorbeer.go
  2) Run "go test" to test against the test suite

  Focus: Basic go syntax, go testing

Exercise 2: Simple CLI for a REST Service

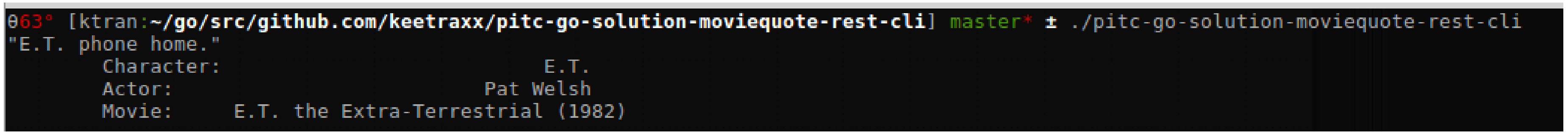
- GitHub Skeleton:

  git clone https://github.com/KeeTraxx/pitc-go-exercise-moviequote-rest-cli

- GET Requests to:
  http://pitc-moviequote.ose3-lab.puzzle.ch/v1/moviequotes/random

Focus: Writing Go Structs with field tags, using net/http for HTTP requests

Result



```
063° [ktran:~/go/src/github.com/keetraxx/pitc-go-solution-moviequote-rest-cli] master* ± ./pitc-go-solution-moviequote-rest-cli
"E.T. phone home."
        Character:                          E.T.
        Actor:                         Pat Welsh
        Movie:     E.T. the Extra-Terrestrial (1982)
```

Exercise 3: Simple REST API

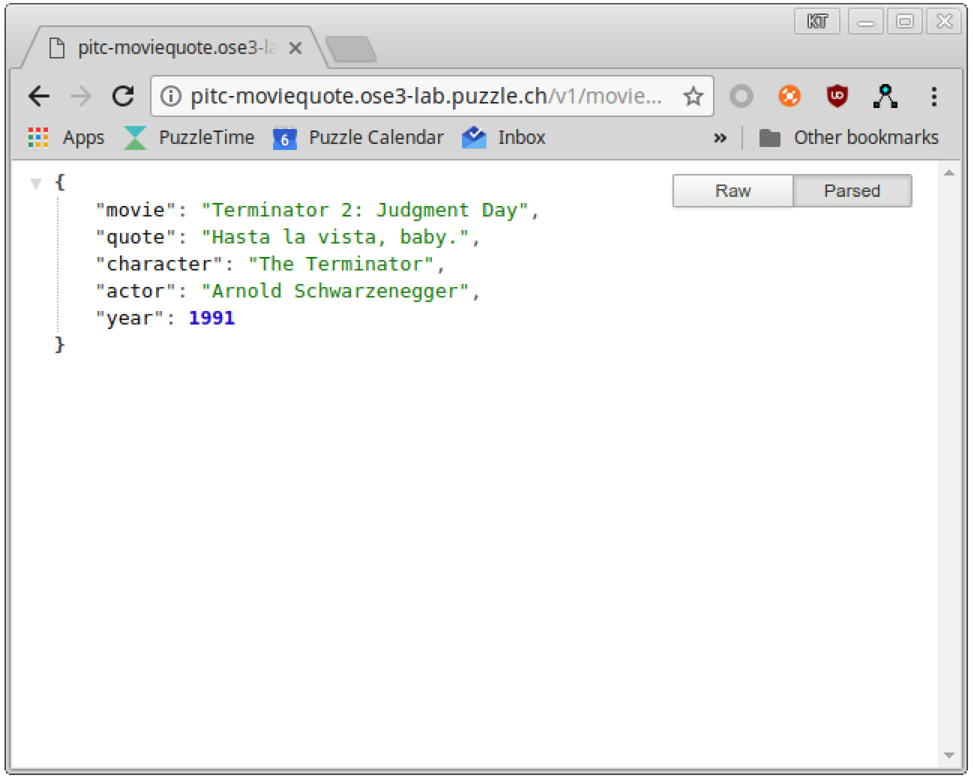
- GitHub Skeleton:

  git clone https://github.com/KeeTraxx/pitc-go-exercise-moviequote-rest-api

- Look at TODOs in the code
- Look in README.md how to deploy on OpenShift

Focus:
use core library net/http for http requests
json (de-)serialization

Result



```
{
    "movie": "Terminator 2: Judgment Day",
    "quote": "Hasta la vista, baby.",
    "character": "The Terminator",
    "actor": "Arnold Schwarzenegger",
    "year": 1991
}
```

Exercise 4: REST CLI for SBB Connections

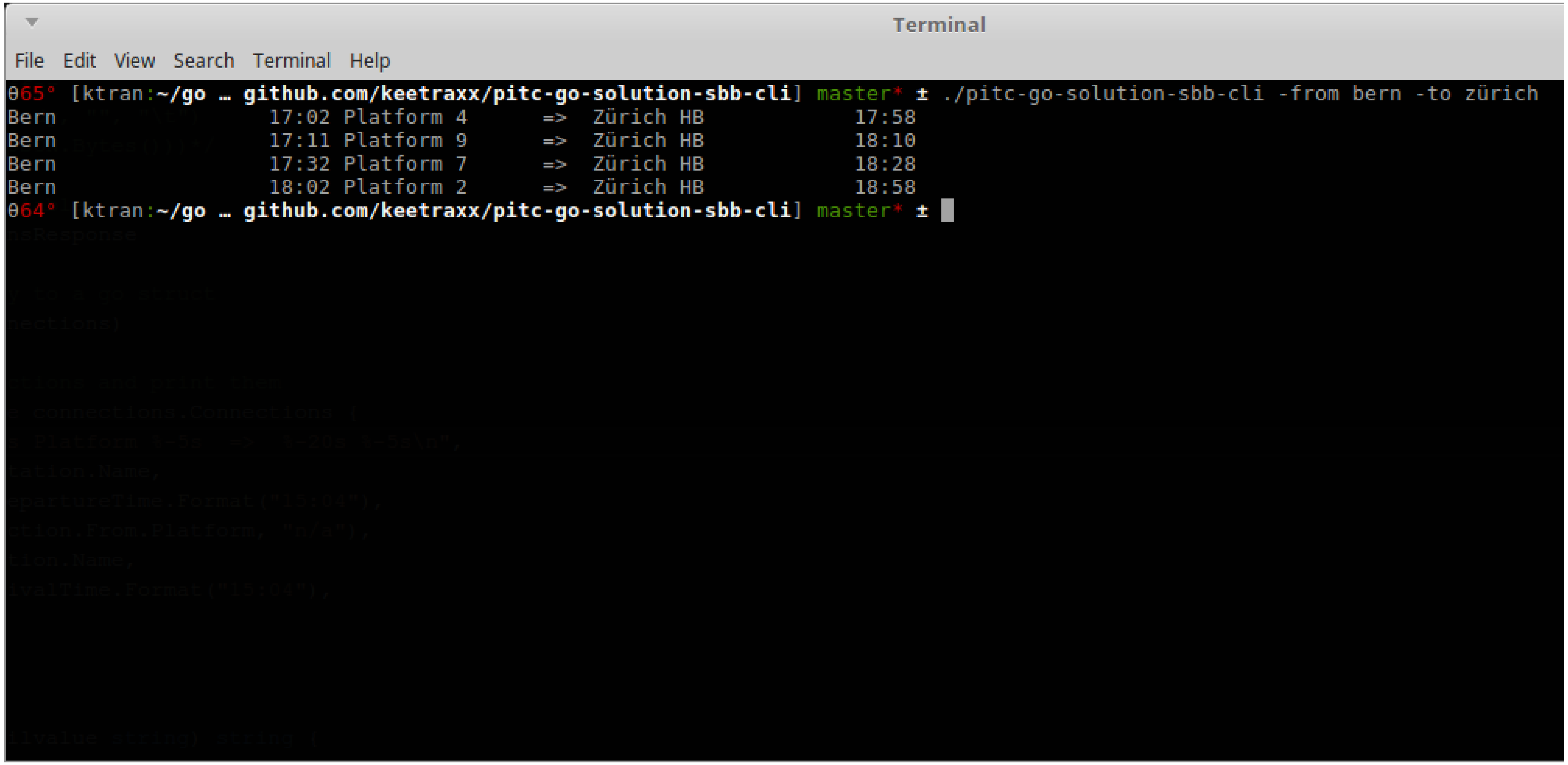
- GitHub Skeleton:

  git clone https://github.com/KeeTraxx/go-cli-skeleton.git

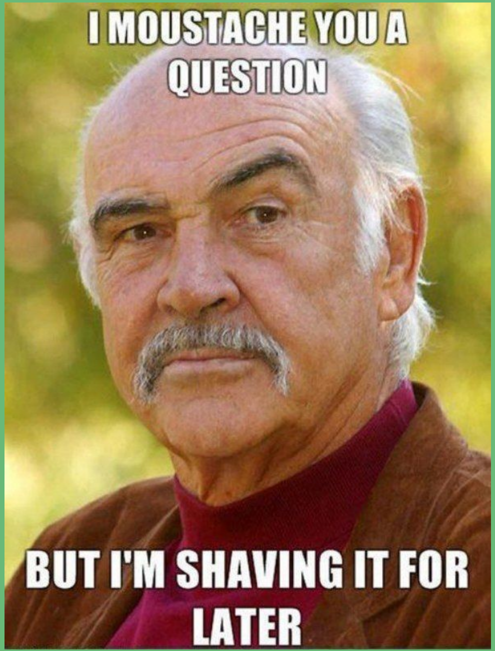
- GET Requests to:
  http://transport.opendata.ch/v1/connections?from=bern&to=thun

Focus: net/http library to make http requests

Result



```
θ65° [ktran:~/go … github.com/keetraxx/pitc-go-solution-sbb-cli] master* ± ./pitc-go-solution-sbb-cli -from bern -to zürich
Bern                 17:02 Platform 4      =>  Zürich HB             17:58
Bern                 17:11 Platform 9      =>  Zürich HB             18:10
Bern                 17:32 Platform 7      =>  Zürich HB             18:28
Bern                 18:02 Platform 2      =>  Zürich HB             18:58
θ64° [ktran:~/go … github.com/keetraxx/pitc-go-solution-sbb-cli] master* ±
```



Wrap-Up

Thank you!

...for not having slept during the presentation