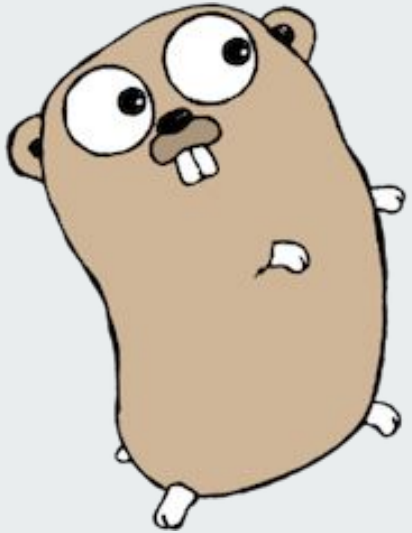
A Day With Go

Who we are?

PickMe

Engineering

Why GO?



Garbage Collected

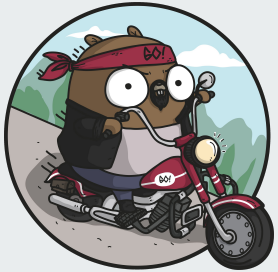
Modern

Interoperability

Lightweight Syntax

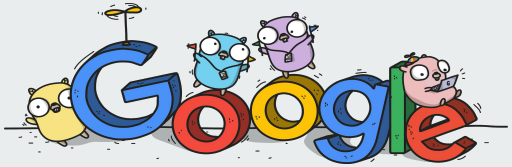
Strictly Typed

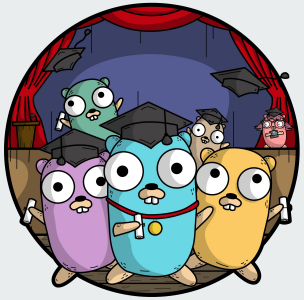
Concurrent



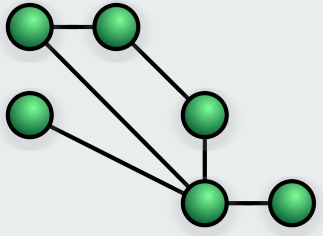
Fast

Cross Platform

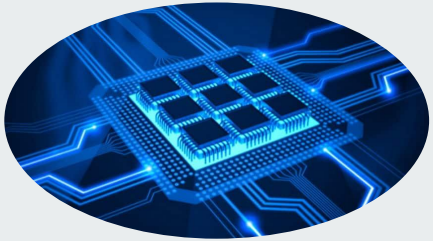




Matured

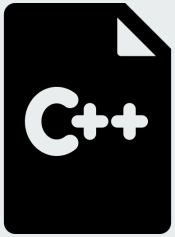


Distributed Systems



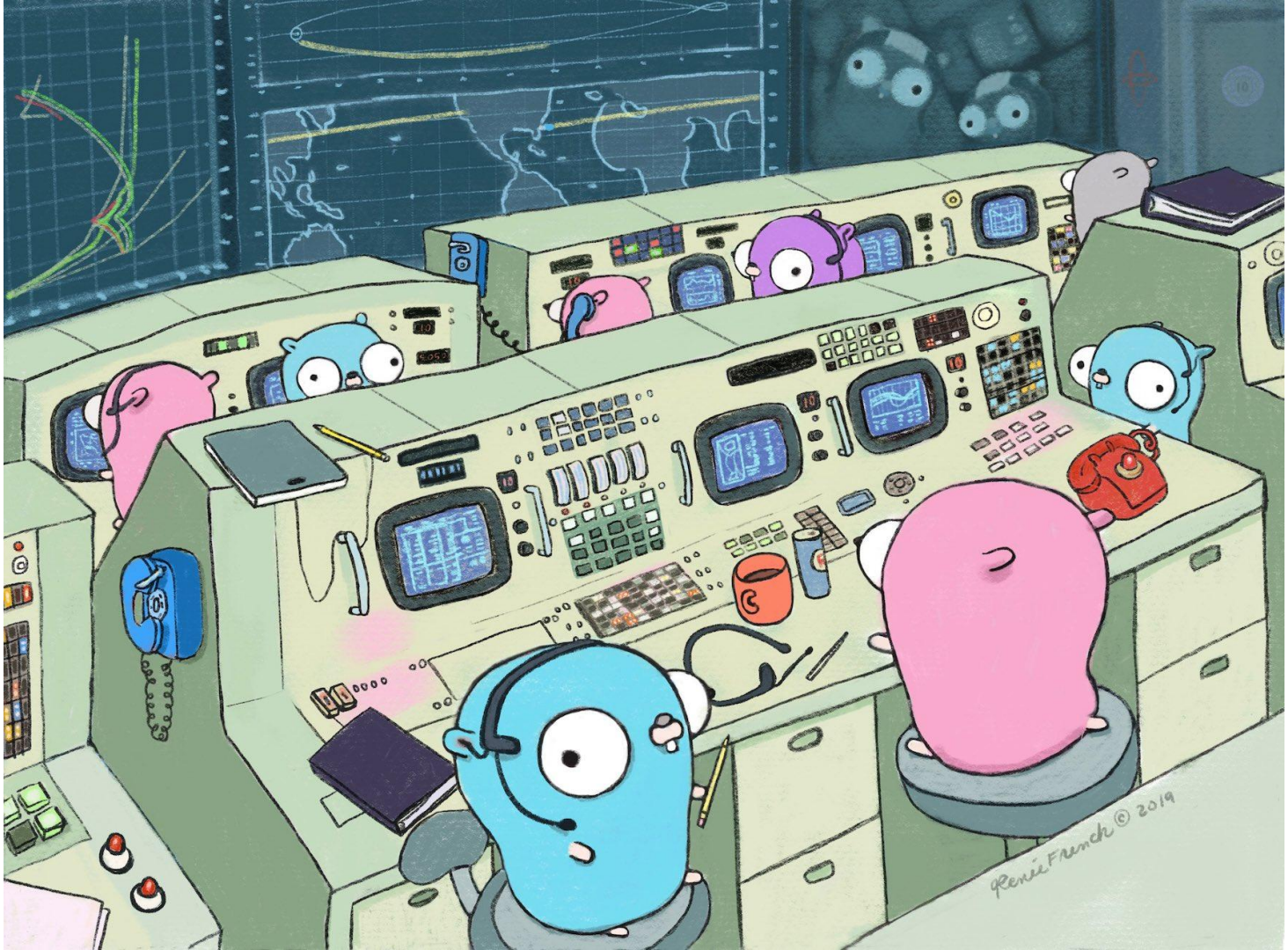
Multicore

Fast Compilation



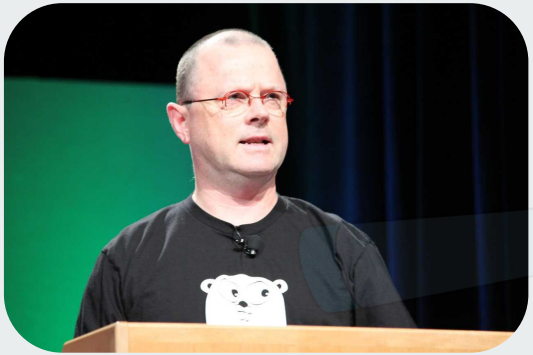
Complexity, Weight

No Static Checking


Renée French © 2019

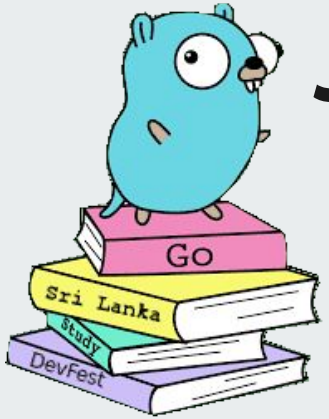
Who?

Robert Griesemer
V8 javascript engine, Java Hotspot VM



Rob Pike
Unix, plan9,UTF-8

Ken Thompson
Unix, plan9, B language, UTF-8



Journey from Zero to Hero

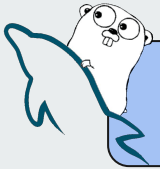
Application standardization

REST APIs

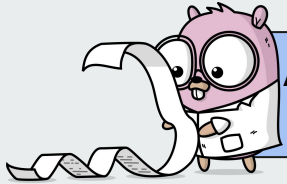
Basics of GO

Testing and Benchmarking

Monitoring with Prometheus and Grafana



Persist data in MySQL

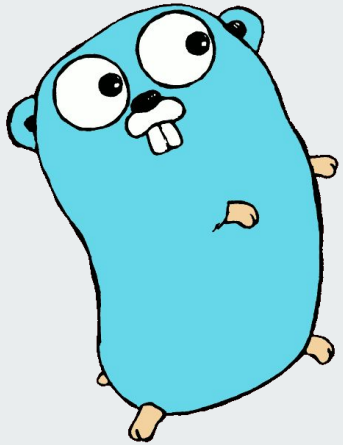


Application Profiling

Playing with GoLang

https://play.golang.org/

Installation



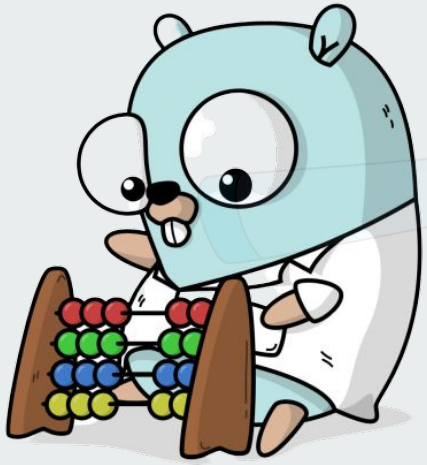
Wget https://dl.google.com/go/go1.12.13.linux-amd64.tar.gz
tar -C /usr/local -xzf go1.12.13.linux-amd64.tar.gz
export PATH=$PATH:/usr/local/go/bin

Starting Development With Go

- IDEs
  - VS Code
  - GoLand
- Hello World Program
- Run
- Compilation

Basic Go Understanding

https://gobyexample.com/



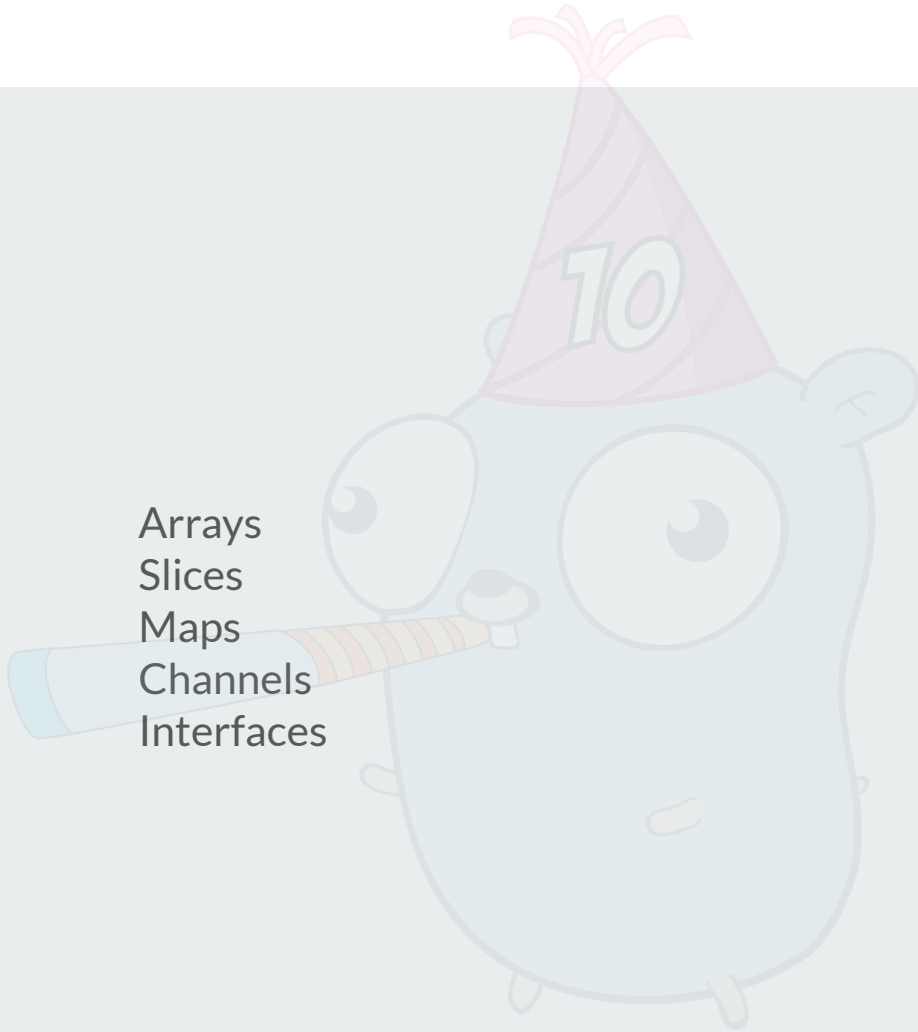
Packages

```go
package main

import "fmt" // Formatted IO

func main() {
  fmt.Println("Hello, World!")
}
```

Types

int, int8, int32, int64
uint8,uint16,uint32,uint64
byte (uint8)
float32, float64
string
rune (int32)
boolean
uintptr
nil

Arrays
Slices
Maps
Channels
Interfaces

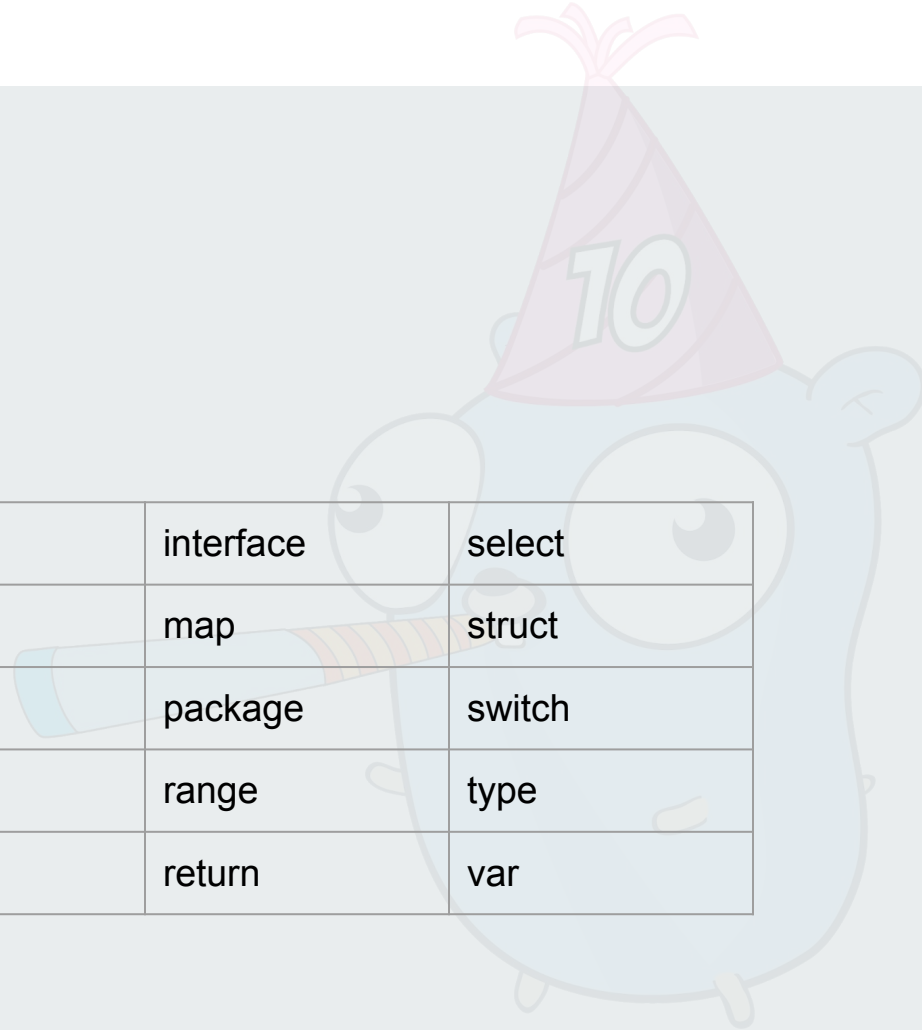


Keywords?

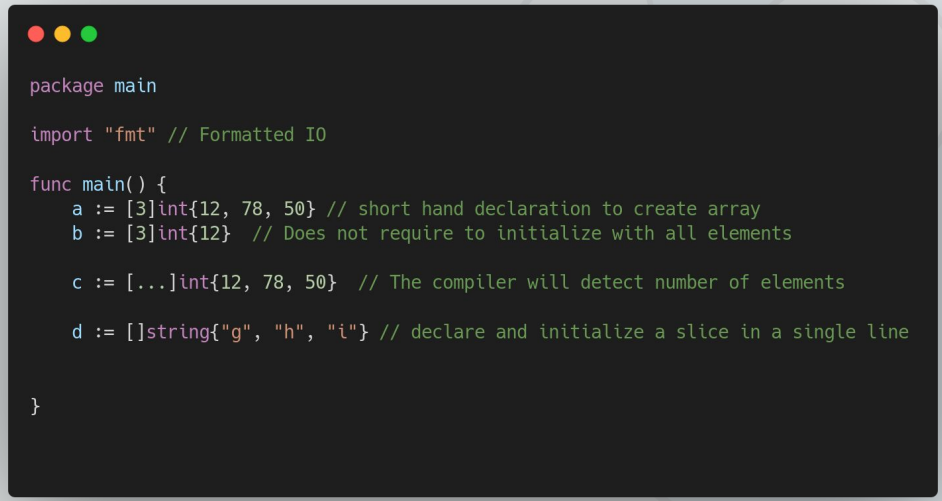
Golang

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

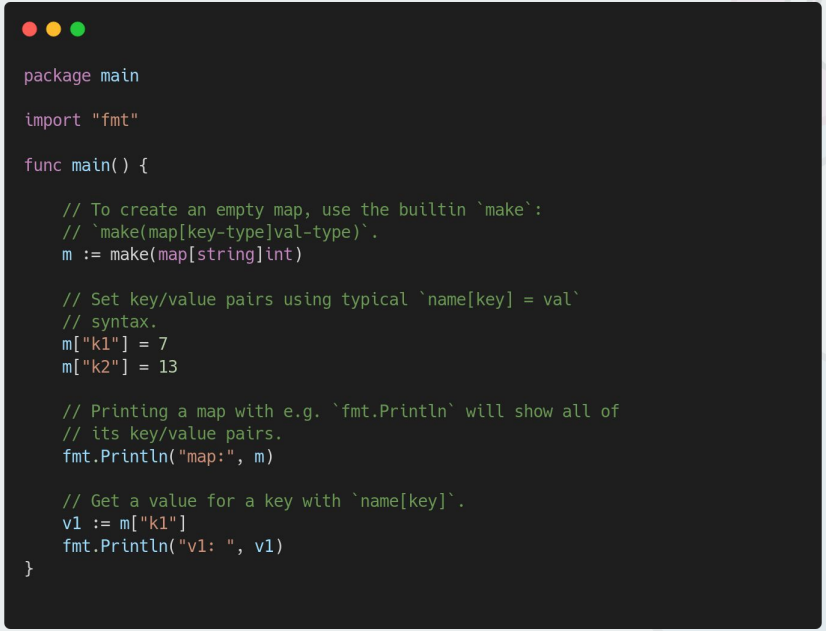


Arrays and Slices



```go
package main

import "fmt" // Formatted IO

func main() {
    a := [3]int{12, 78, 50} // short hand declaration to create array
    b := [3]int{12}  // Does not require to initialize with all elements

    c := [...]int{12, 78, 50}  // The compiler will detect number of elements

    d := []string{"g", "h", "i"} // declare and initialize a slice in a single line


}
```

Maps



```go
package main

import "fmt"

func main() {

    // To create an empty map, use the builtin `make`:
    // `make(map[key-type]val-type)`.
    m := make(map[string]int)

    // Set key/value pairs using typical `name[key] = val`
    // syntax.
    m["k1"] = 7
    m["k2"] = 13

    // Printing a map with e.g. `fmt.Println` will show all of
    // its key/value pairs.
    fmt.Println("map:", m)

    // Get a value for a key with `name[key]`.
    v1 := m["k1"]
    fmt.Println("v1: ", v1)
}
```

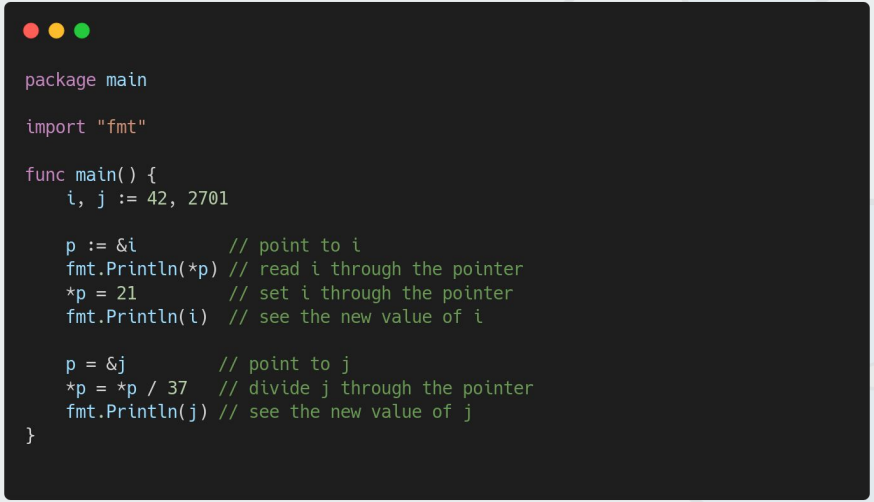
Structures

```
package main

import "fmt"

type Books struct {
   title string
   author string
   subject string
   book_id int
}

func main(){

}
```

Loops

```go
package main

import "fmt"

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

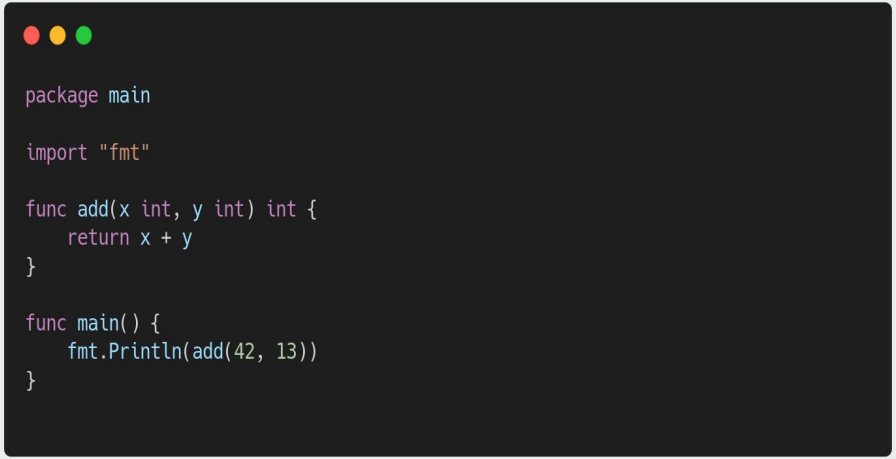
Pointers



```
package main

import "fmt"

func main() {
    i, j := 42, 2701

    p := &i         // point to i
    fmt.Println(*p) // read i through the pointer
    *p = 21         // set i through the pointer
    fmt.Println(i)  // see the new value of i

    p = &j         // point to j
    *p = *p / 37   // divide j through the pointer
    fmt.Println(j) // see the new value of j
}
```

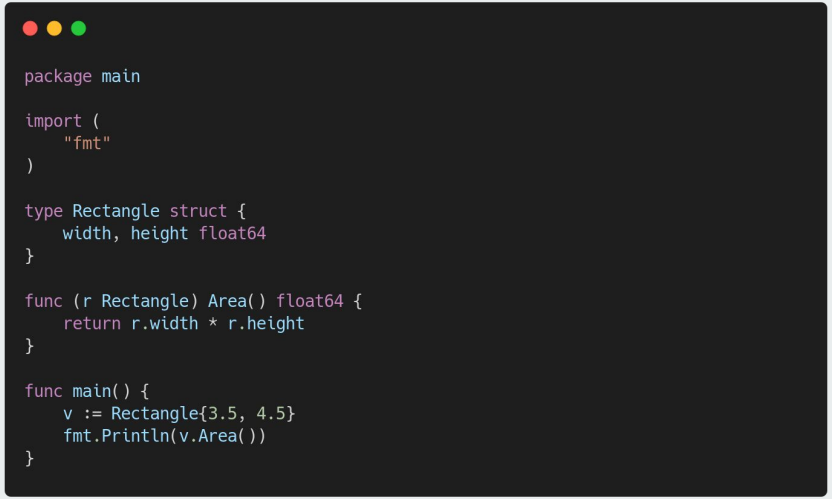
Functions and Methods

Functions



```go
package main

import "fmt"

func add(x int, y int) int {
    return x + y
}

func main() {
    fmt.Println(add(42, 13))
}
```

Methods



```go
package main

import (
    "fmt"
)

type Rectangle struct {
    width, height float64
}

func (r Rectangle) Area() float64 {
    return r.width * r.height
}

func main() {
    v := Rectangle{3.5, 4.5}
    fmt.Println(v.Area())
}
```

Conditions

if / else
Switch case

If/ Else

```go
package main

import "fmt"

func main() {

    // Here's a basic example.
    if 7%2 == 0 {
        fmt.Println("7 is even")
    } else {
        fmt.Println("7 is odd")
    }
}
```

Switch Case

```go
package main

import (
    "fmt"
)

func main() {

    // Here's a basic `switch`.
    i := 2
    fmt.Print("Write ", i, " as ")
    switch i {
    case 1:
        fmt.Println("one")
    case 2:
        fmt.Println("two")
    case 3:
        fmt.Println("three")
    }
}
```

Let’s say “Hello, World”

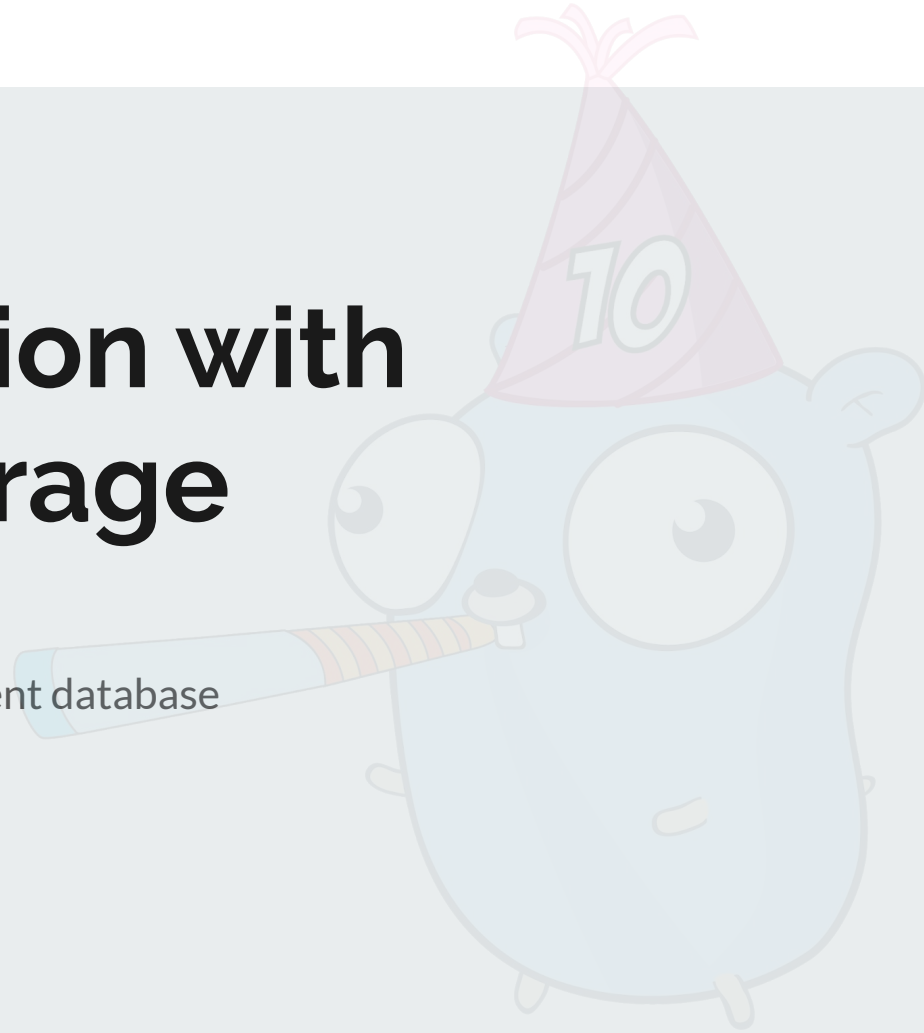
Simple Hello, World app
Hello world with simple HTTP response
HTTP Server with Handler
Server Mux Implementation

Session 2

1.5 Hours

CRUD application with in-memory storage

Will be extend this by integrating a persistent database



REST API

- <endpoint>/person POST
- <endpoint>/person GET
- <endpoint>/person/:id GET
- <endpoint>/person/:id PUT
- <endpoint>/person/:id DELETE

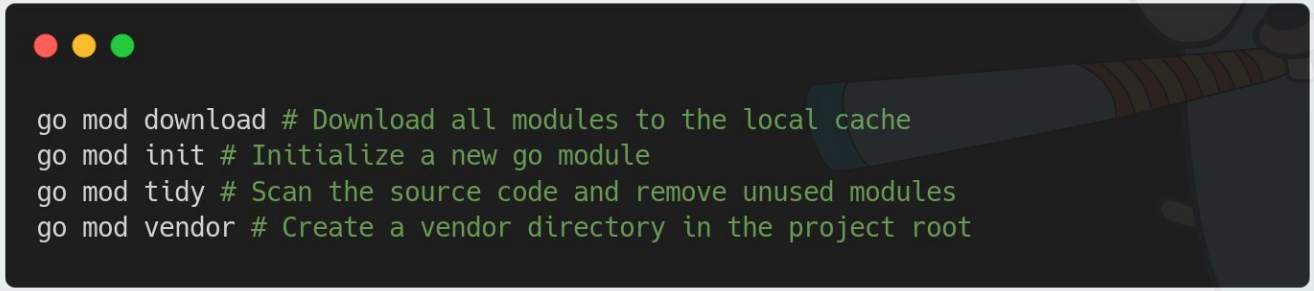
Clone the source

https://github.com/devfest2k19/gowork.git

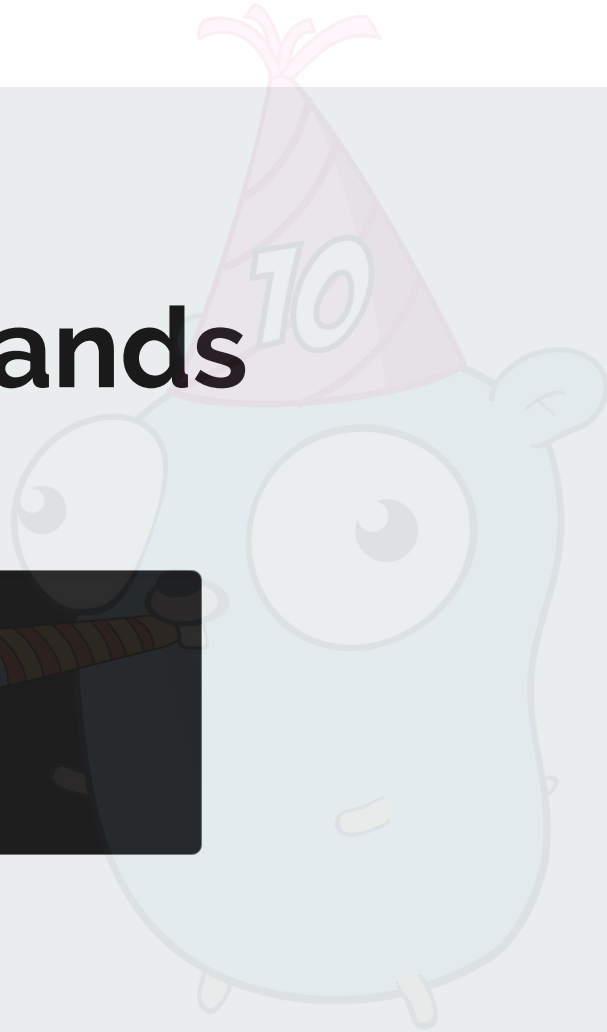
Dependency Management

How to use go mod?

Basic Go mod commands



```
go mod download # Download all modules to the local cache
go mod init # Initialize a new go module
go mod tidy # Scan the source code and remove unused modules
go mod vendor # Create a vendor directory in the project root
```



Persisting Data

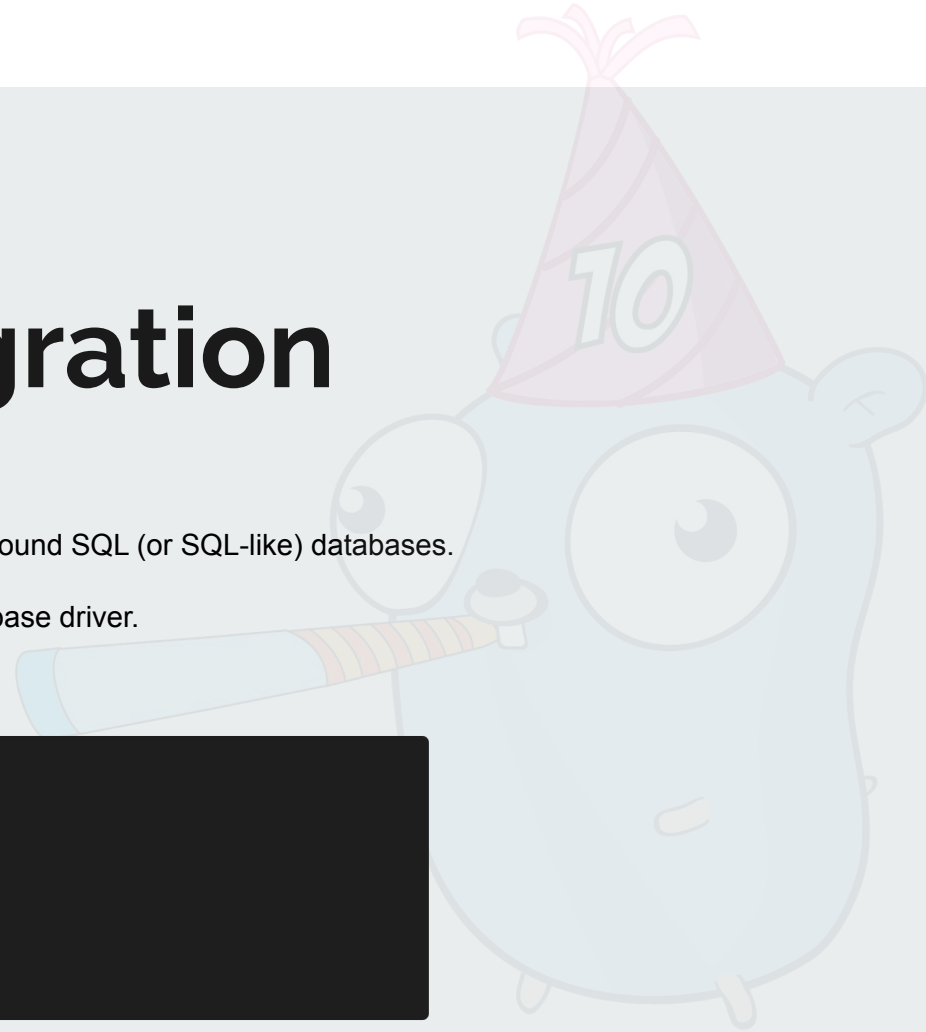
Database Integration

The database/sql package provides a generic interface around SQL (or SQL-like) databases.

The sql package must be used in conjunction with a database driver.

Ex: MySQL, PostgreSQL

```go
package adapters

import (
    "database/sql"

    _ "github.com/go-sql-driver/mysql"
)
```



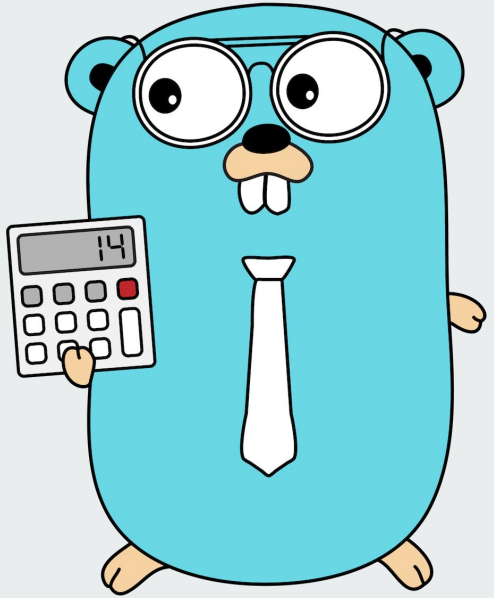
Database Credentials

IP: **35.213.163.65**
Port: **3306**
User : **devfest_go**
Pass: **goturns10**
DB: **phonebook**

Session 3

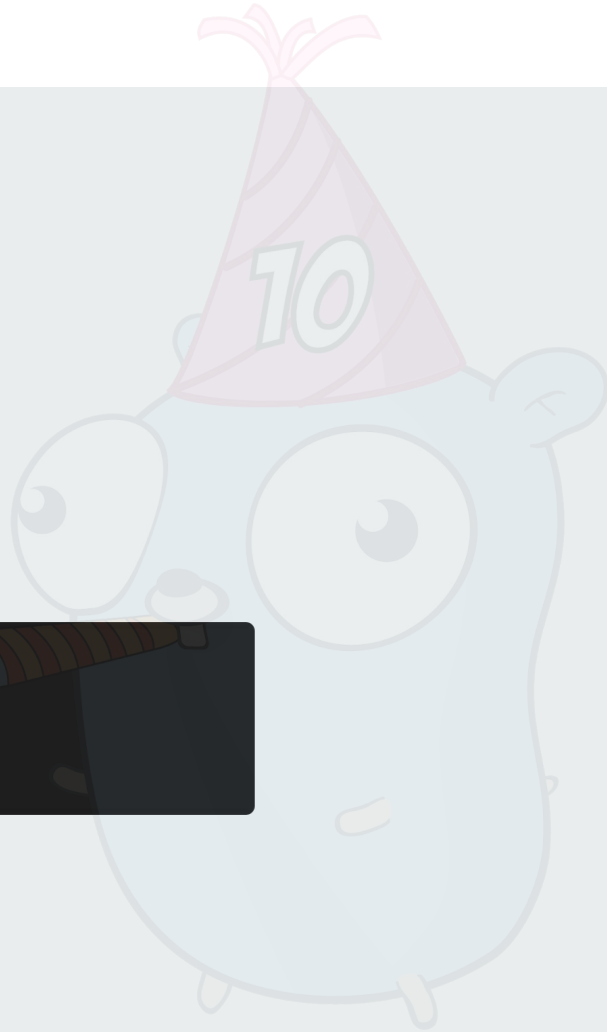
2 Hours

Unit testing in Go

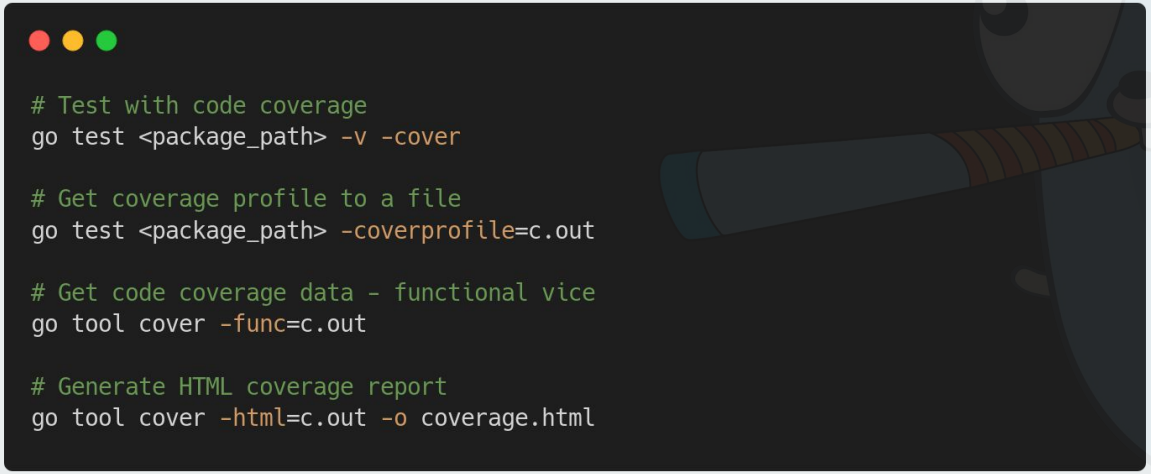


How to run unit tests?

Code Coverage



Code Coverage tools



```
# Test with code coverage
go test <package_path> -v -cover

# Get coverage profile to a file
go test <package_path> -coverprofile=c.out

# Get code coverage data - functional vice
go tool cover -func=c.out

# Generate HTML coverage report
go tool cover -html=c.out -o coverage.html
```

Benchmarking in GO

How to run benchmarks?

Log Standardization

github.com/pickme-go/log

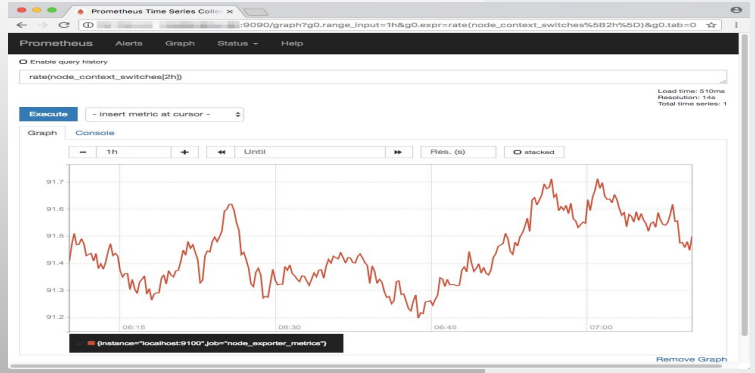
Application Monitoring

- Prometheus/ Grafana
- Instrumenting the application

Install Prometheus

Install prometheus in Ubuntu

```
wget https://github.com/prometheus/prometheus/releases/download/v2.14.0/prometheus-2.14.0.linux-amd64.tar.gz
tar xvfz prometheus-2.14.0.linux-amd64.tar.gz
cd prometheus-2.14.0.linux-amd64
vim prometheus.yml
#####---- insert your job details ----#####
- job_name: 'dev-fest-crud'
   static_configs:
         - targets: ['localhost:8001']

#####-------------------------------#####
sudo ./prometheus
```



Install Grafana

Install grafana in Ubuntu

```
wget https://dl.grafana.com/oss/release/grafana-6.4.4.linux-amd64.tar.gz

tar -zxvf grafana-6.4.4.linux-amd64.tar.gz

bin/grafana-server
```

Application Profiling

- Use of go pprof package

```
import (
  _ "net/http/pprof" // https://golang.org/pkg/net/http/pprof/
)
```

```
    go func() {
        log.Info(http.ListenAndServe(":6060", nil))
    }()
```

Concurrency

Goroutines
Go channels
Sync package

Concurrency & parallelism

Concurrency is not parallelism

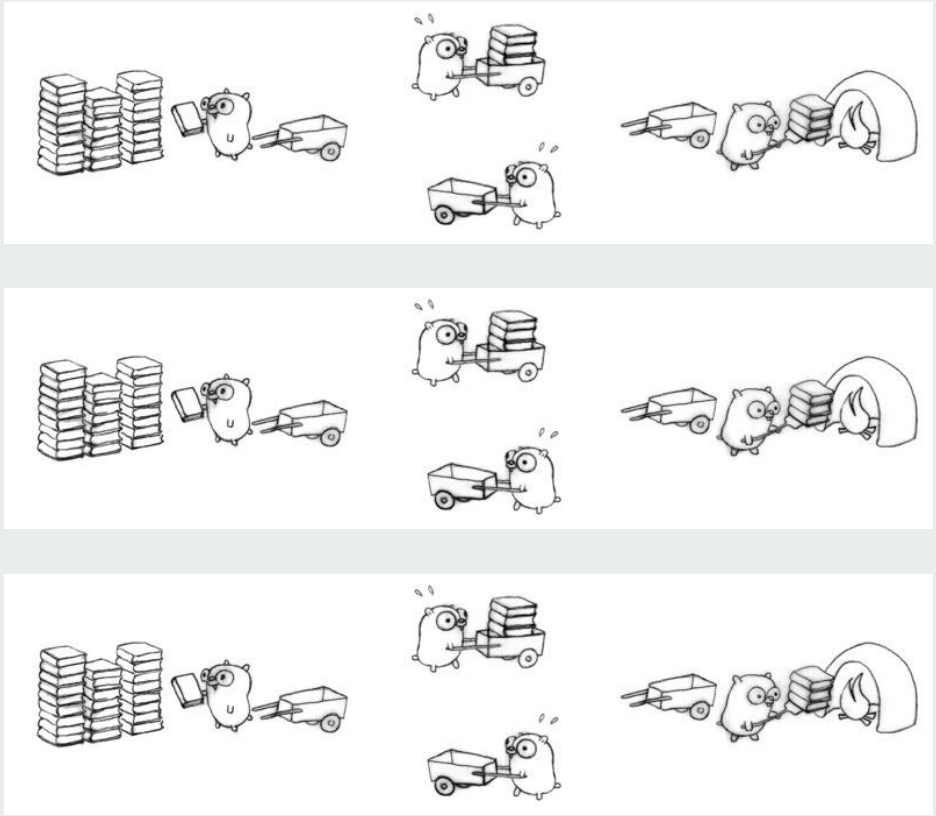
“Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.” — Rob Pike

Concurrency

Tasks start, run and complete in an interleaved fashion

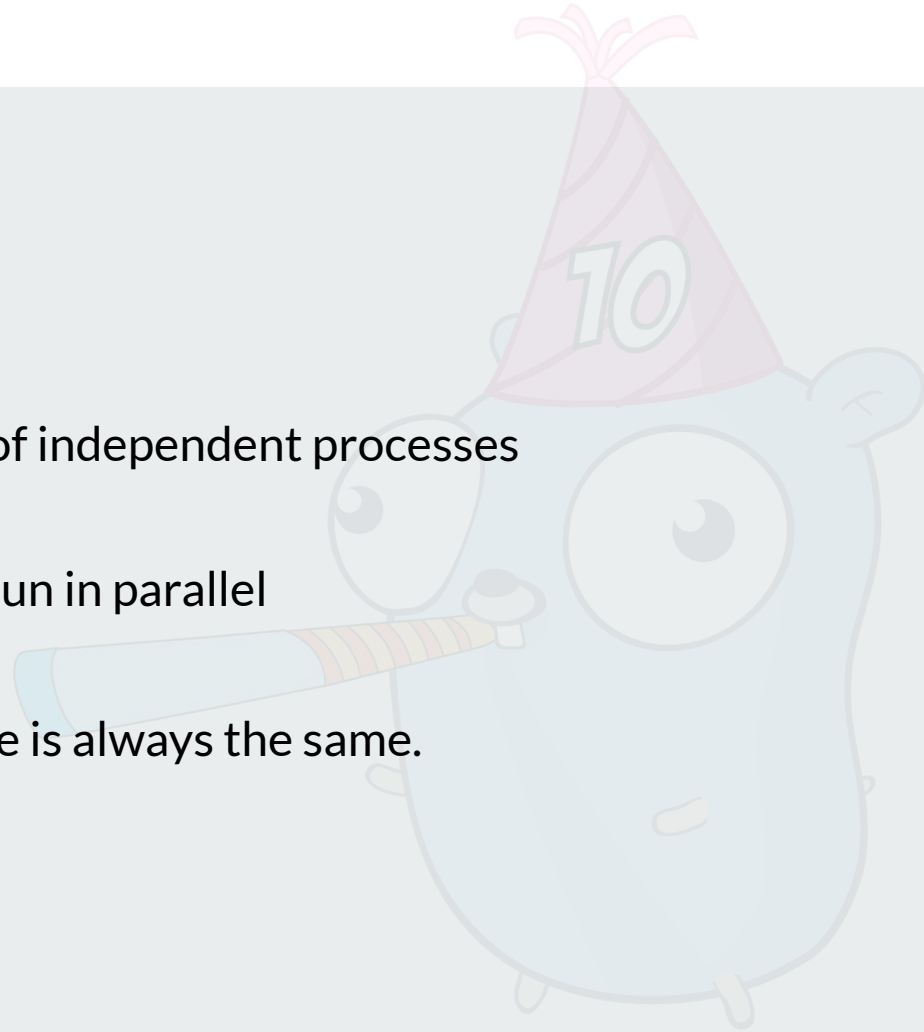
Parallelism

Tasks run simultaneously



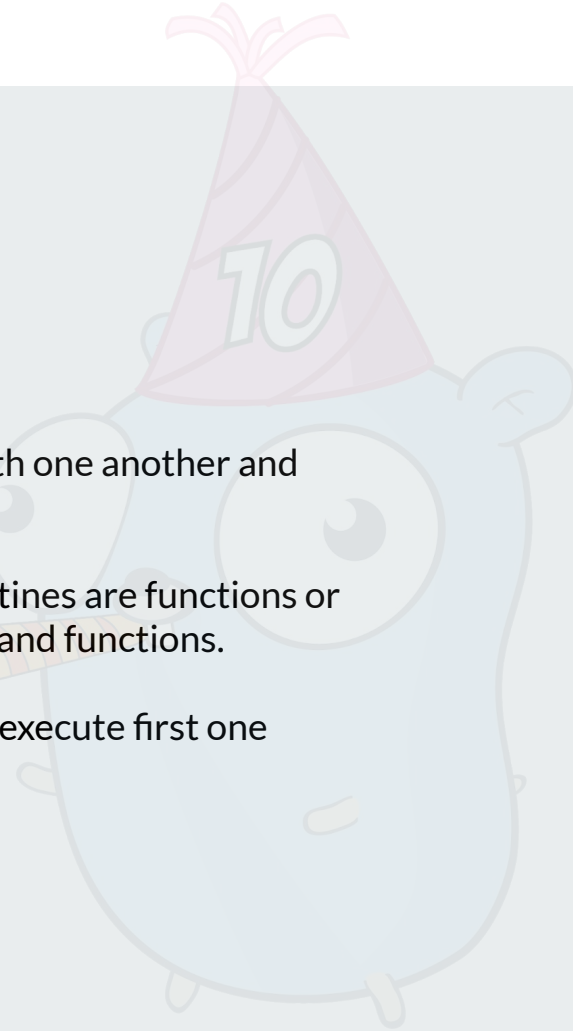
Concurrency applied

- **Design** our program as a collection of independent processes
- **Design** these processes eventually run in parallel
- **Design** our code so that the outcome is always the same.



Go’s Concurrency Tool Set

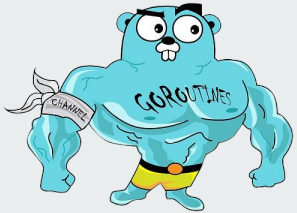
- **Goroutines**
  - Channels provide a way for goroutines to communicate with one another and synchronize their execution.
- **Channels**
  - In Go, concurrency is achieved by using Goroutines. Goroutines are functions or methods which can run concurrently with others methods and functions.
- **Select**
  - Like switch case. Select can listen to multiple channels and execute first one triggered.
- **Sync package**
  - Handle waiting for goroutines and mutex locking



Goroutines vs Threads

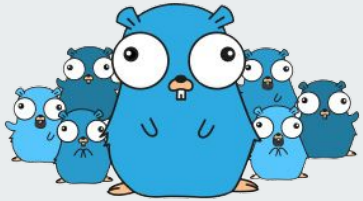
Growable segmented stacks

Faster startup time



Built in primitives for safe communication

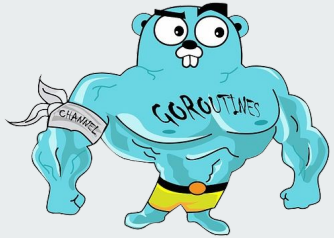
Avoid resorting to mutexes



No 1:1 mapping with os threads

Higher concurrency with lesser complexity

Goroutines




```go
package main

import "fmt"

func printNumbers(n int){
   fmt.Printf("[printNumbers]: %v",n)
}

func main(){
    fmt.Println("[main] start")
    printNumbers(2)
    fmt.Println("[main] end")
}
```

https://play.golang.org/p/wjtm0aGdLos

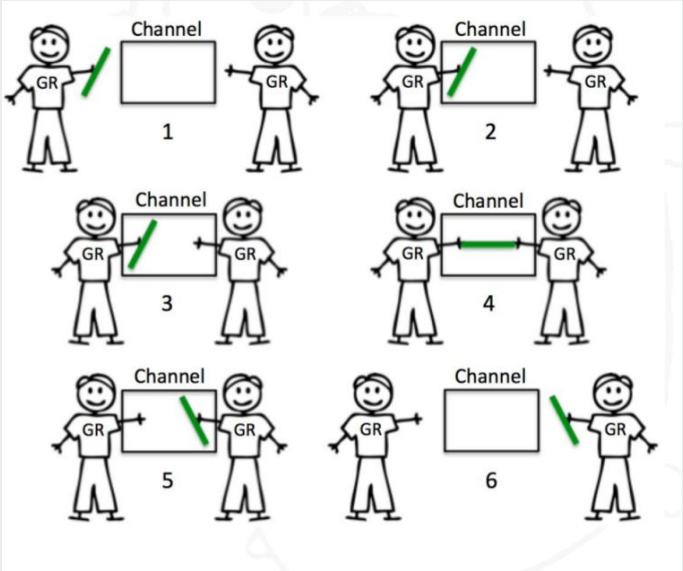
```go
package main

import "fmt"

func printNumbers(n int){
   fmt.Printf("[printNumbers]: %v",n)
}

func main(){
    fmt.Println("[main] start")
    go printNumbers(2)
    fmt.Println("[main] end")
}
```

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

```go
package main

import (
"fmt"
"time"
)

func printNumbers(n int){
   fmt.Printf("[printNumbers]: %v\n",n)
}

func main(){
    fmt.Println("[main] start")
    go printNumbers(2)
    time.Sleep(2)
    fmt.Println("[main] end")
}
```

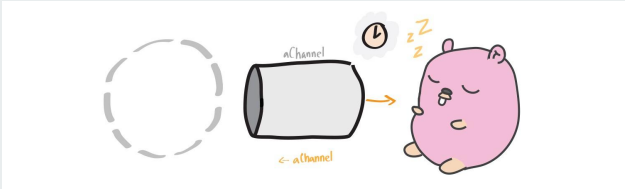
https://play.golang.org/p/A2rjrYajDYP

Go Channels

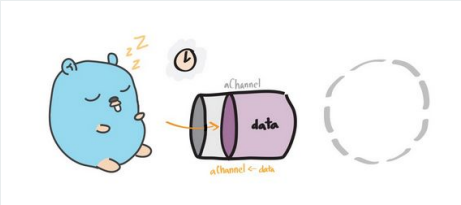


Unbuffered channels

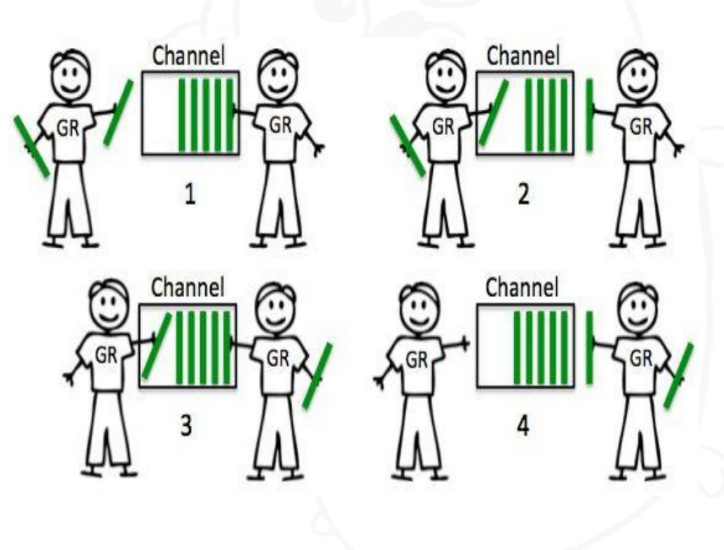
- No sender case



- No receiver case

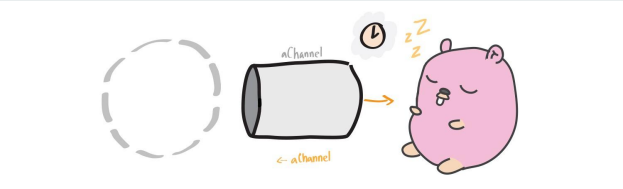


Go Channels

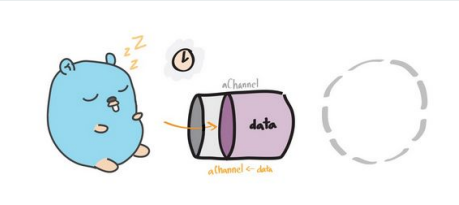


Buffered channels

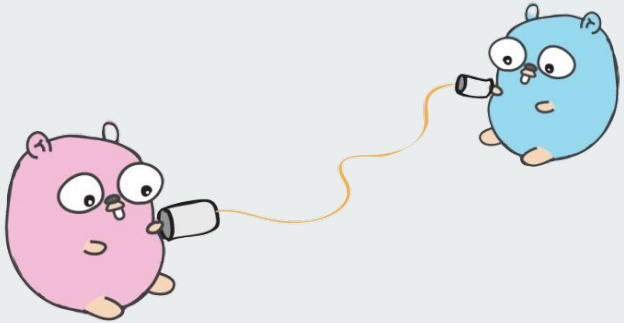
- No sender case



- No receiver case



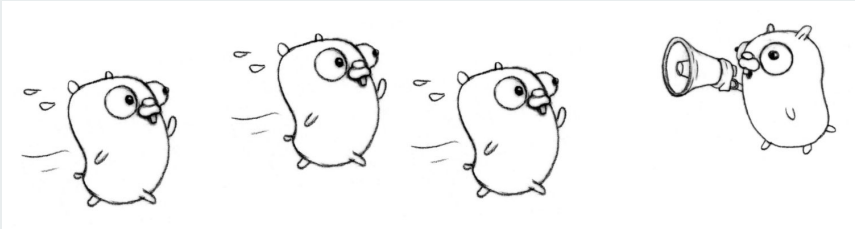
Go Channels



- How they work
  - https://play.golang.org/p/_N1bd-hXQsD
- Reading in a Loop
  - https://play.golang.org/p/xHqSwfpu5RL
- Unbuffered VS Buffered
  - https://play.golang.org/p/xHqSwfpu5RL
  - https://play.golang.org/p/bCRQ4QagrAr
- Closing
  - https://play.golang.org/p/5cfIdSxOxbg
  - https://play.golang.org/p/FgSEQw7gJ80

Don't communicate by sharing memory, share memory by communicating!

Wait groups



https://play.golang.org/p/pluDSf4uYZJ

- A WaitGroup waits for a collection of goroutines to finish.
- The main goroutine calls Add to set the number of goroutines to wait for.
- Each of the goroutines runs and calls Done when finished.
- At the same time, Wait can be used to block until all goroutines have finished.

Mutexes

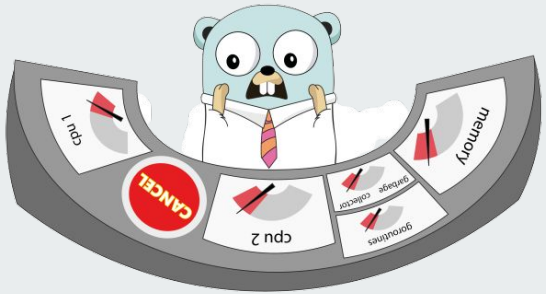
- channels are great for communication among goroutines
- But what if we don't need communication?
- What if we just want to make sure only one goroutine can access a variable at a time to avoid conflicts?

https://play.golang.org/p/7C0RPPG8eTP

Channels orchestrate, mutexes serialize!

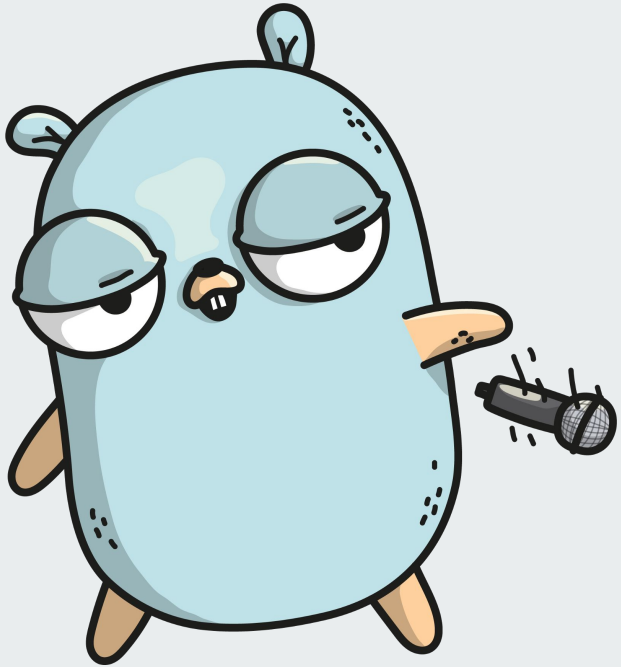
Context

Carries deadlines, cancelation signals, and other request-scoped values across API boundaries and between processes.



- Cancellation
  - https://play.golang.org/p/QJtUFvv-I9M
- Value passing
  - https://play.golang.org/p/lo0U6LaGCS7

Question time




Thank You!