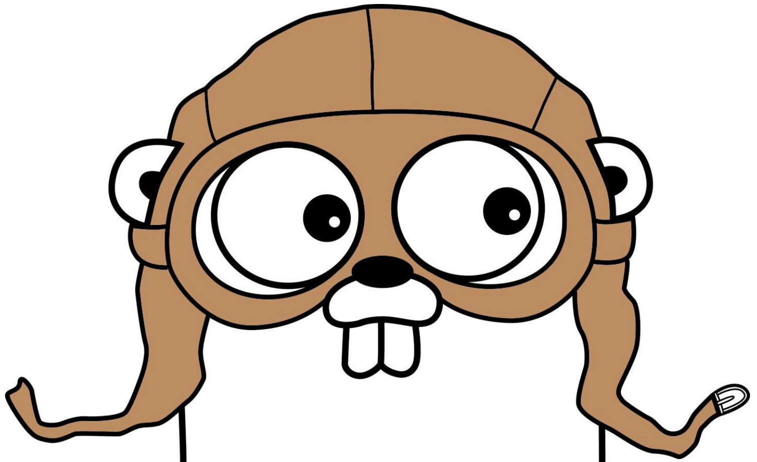
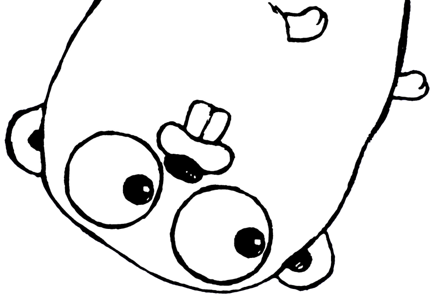
coding in GO

GDG Sri Lanka

Monthly meetup November 2015 **NSBM**

Raveen Perera





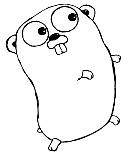
GO

Fast, compiled language, directly to machine code and spearheaded by Google

History

Created by Robert Griesemer, Rob Pike, Ken Thompson

Developed in **2007** and first stable open source release **2009 (BSD)**



What’s so special about **GO ?**

Compilation

Very **fast compilation** (seconds)

No VM needed

GOs **Assembler**

Tools

go **fmt** go **vet**

go **test** go **doc**

Concurrency

Asynchronous processes called **GOroutines**

Channels used to pass data between routines

Standard Library

net/http

flag

encoding/json encoding/xml

Simplicity in Syntax

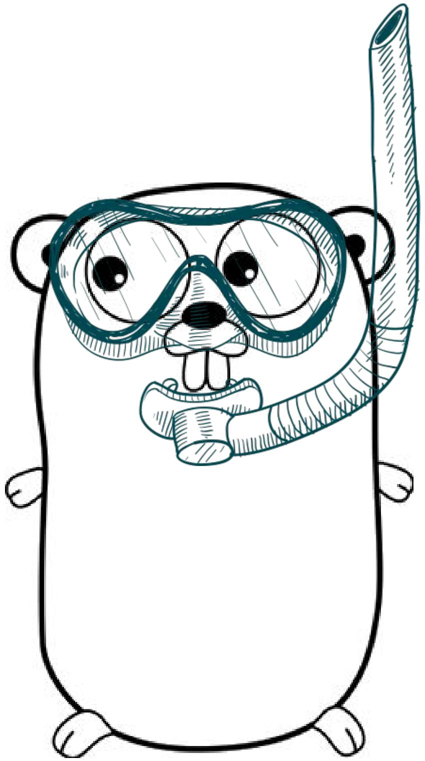
GO stands between **C** and **Python**

Highly **influenced** by many other popular **programming languages**

Deployment

Can be compiled to a **single binary** file

You can build and compile in your **host or server**



Let’s dive in

Download and install GO

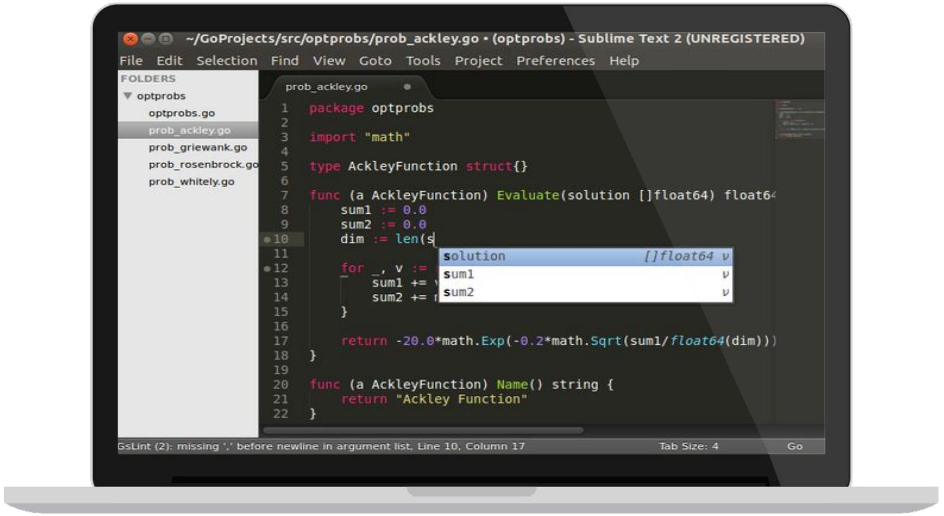
https://golang.org/dl/

Download and install **Sublime Text 3**

http://www.sublimetext.com/3

and install **GOSublime** plugin

https://github.com/DisposaBoy/GoSublime

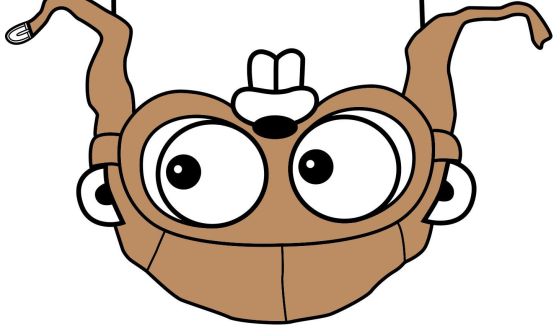


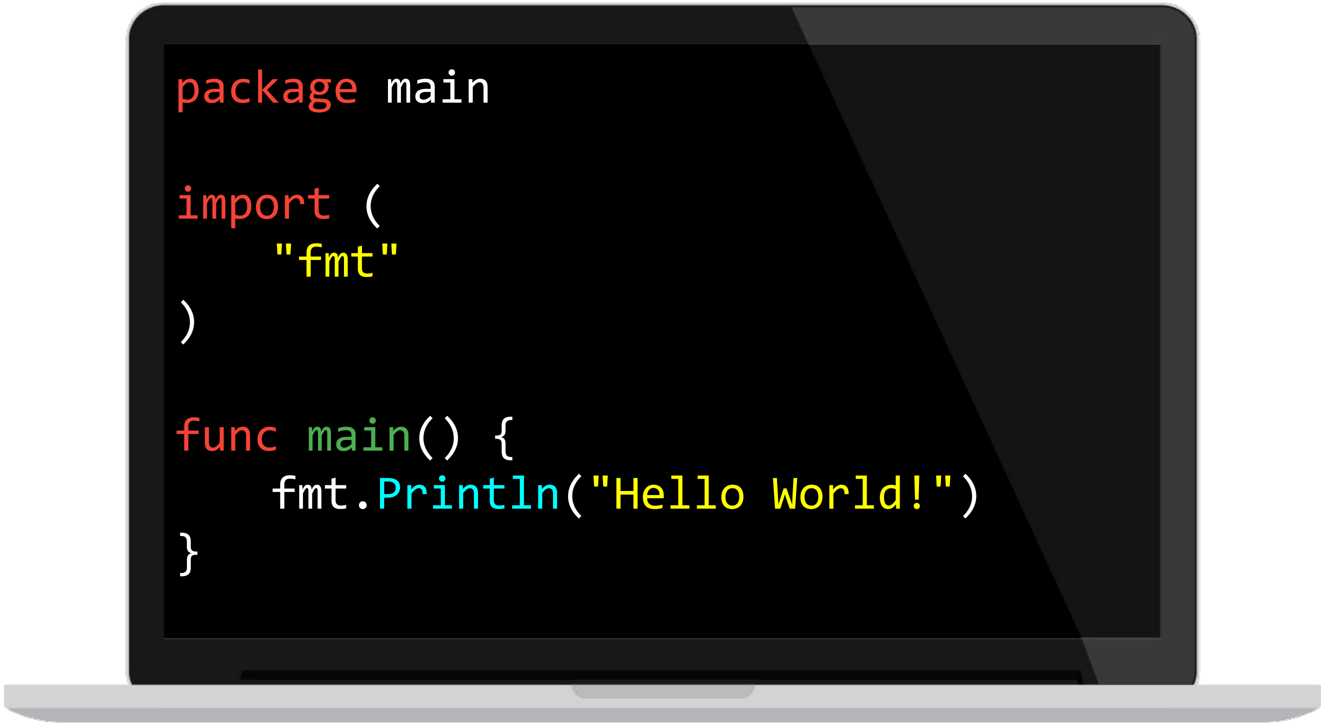
~/GoProjects/src/optprobs/prob_ackley.go • (optprobs) - Sublime Text 2 (UNREGISTERED)

```
package optprobs

import "math"

type AckleyFunction struct{}

func (a AckleyFunction) Evaluate(solution []float64) float6
    sum1 := 0.0
    sum2 := 0.0
    dim := len(s

    for _, v :=
        sum1 +=
        sum2 +=
    }

    return -20.0*math.Exp(-0.2*math.Sqrt(sum1/float64(dim)))
}

func (a AckleyFunction) Name() string {
    return "Ackley Function"
}
```

```
solution    []float64 v
sum1                  v
sum2                  v
```

GsLint (2): missing ',' before newline in argument list, Line 10, Column 17





```go
package main

import (
    "fmt"
)

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

```
$ go run helloworld.go
```

The Basics

https://golang.org/ref/spec

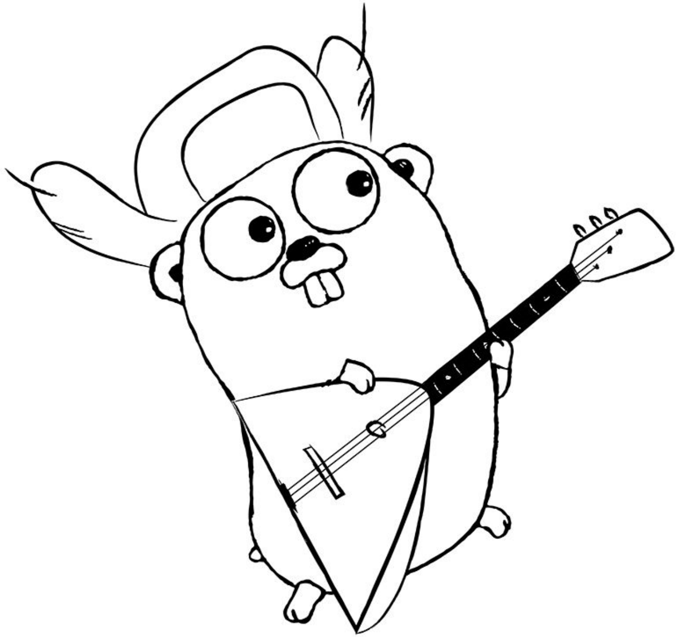
Variables

```
var age int = 40

name := “John Doe”

const pi float64 = 3.14

strings “ ” or ` `
```

```
bool true false

+   -    *   /   %

&& || ! == != >= <=
```

Loops

```
for i := 0; i < count; i++ {

  }

for i, value := range array {

  }

for i <= 10 {

    i++

  }
```



Conditions

```
if i > 10 {

    } else if i > 5 {

    } else {

}
```

```
switch grade {

    case 75: fmt.Println("A")

    default: fmt.Println("nothing")

  }
```

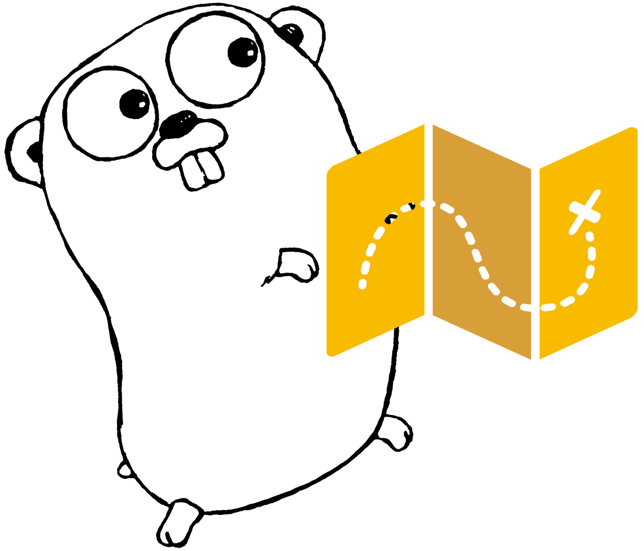
Arrays

```
var myArray[5] int

myArray := [5]int {1,2,3,4,5}

mySlice := []int {1,2,3,4,5} //Slice has no size declaration

mySlice2 := mySlice[3:5]

slice := make([]int, 5, 10)

slice = append(slice,0,1)
```

Maps

Just like dictionaries in python

```
grades := make(map[string] int)

grades[“John”] = 80

delete(grades,”John”)
```



Functions

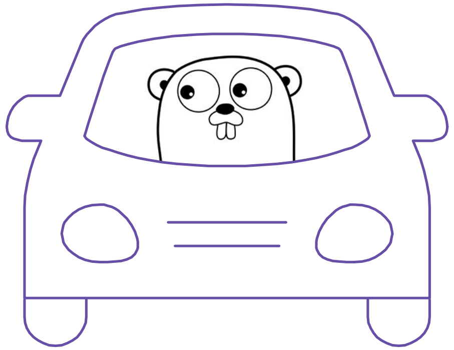
```
func myFunc(number int) int {

return number + 5

}

func myFunc(number int) (int,int) {

return number + 5, number +6

}
```

Executes after the enclosing function

```
defer myFunc()
```

Undefined number of variables

```
func uParams(args ...int) int {

}
```



Functions - defer() and panic()

```
func divide(num1 int, num2 int) int {
  defer func()  {
    fmt.Println(recover())
  }()
  answer := num1/ num2
  return answer
}
```

```
func divide() {
  defer func()  {
    fmt.Println(recover())
  }()
  panic(“sending to recover”)
}
```

Closure

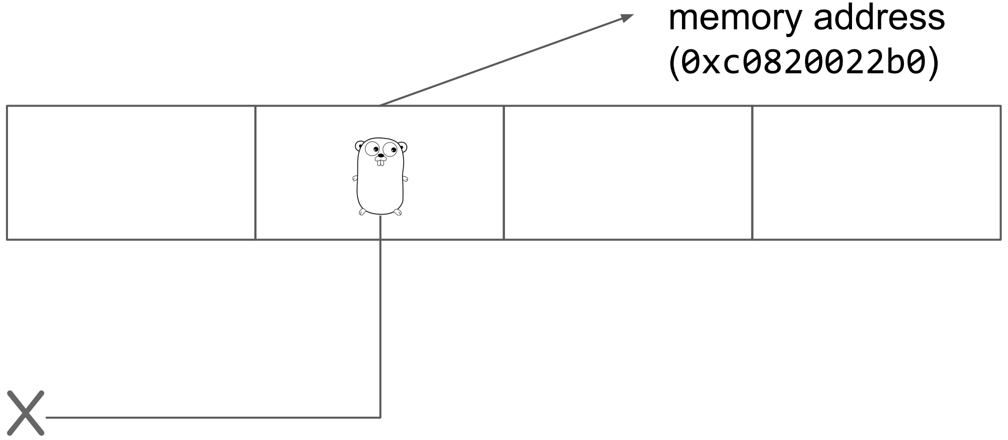
Declaring a function inside another

```
func main() {
    myfunc := func() int {}
    myfunc()
}
```

Pointers

```
x := 8

changeX(&x)


func changeX(x *int){

    *x = 10

}
```

```
myPointer := new(int)
```



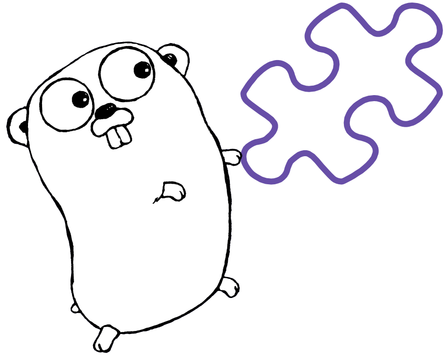
Structures

Go is not object oriented

```
type Circle struct {

     var radius float64

     var name string

}
```

```
func (circle Circle) area() float64 {

     return circle.radius*circle.radius*3.14

}
```

```
myCircle = Circle{name:”circle1” , radius:5}
```



Handling Concurrency

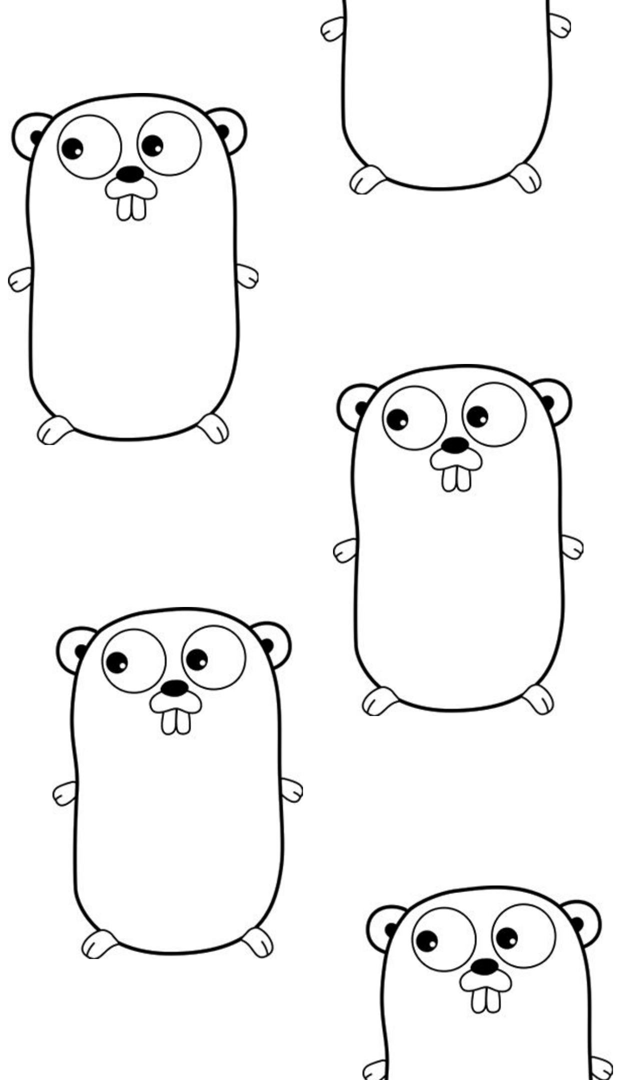
GORoutines

Not expensive as threads
Multiple Goroutines without cost

Channels

GORoutines reads and writes values from an to channels to communicate

https://golang.org/pkg/sync/atomic/



GO http

Route
Handles the requests and determines which function should handle that request

Handler
The function that executes when a request is made

Server
The networking code which handles the requests and routes (Serve mux) multiplexer, http request router

Simple http server

Simple respond writer

```go
package main

import "net/http"

func main() {
  http.HandleFunc("/", homeHandler)
  http.ListenAndServe(":8100", nil)
}

func homeHandler(w http.ResponseWriter, r *http.Request) {
  w.Write([]byte("ආයුබෝවන්))
}
```

Gorilla Toolkit

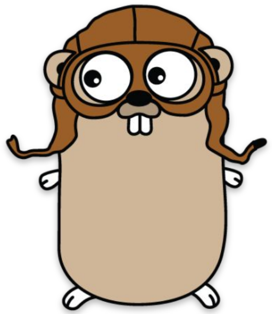
Gorilla Mux

```
$ go get github.com/gorilla/mux
```

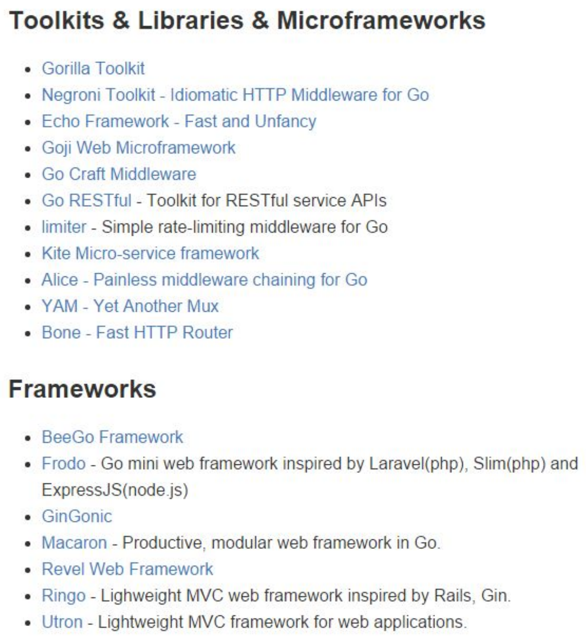
simple buffer writer

Gorilla sessions

```
$ go get github.com/gorilla/sessions
```



GO Frameworks Toolkits and Micro Frameworks



Toolkits & Libraries & Microframeworks

- Gorilla Toolkit
- Negroni Toolkit - Idiomatic HTTP Middleware for Go
- Echo Framework - Fast and Unfancy
- Goji Web Microframework
- Go Craft Middleware
- Go RESTful - Toolkit for RESTful service APIs
- limiter - Simple rate-limiting middleware for Go
- Kite Micro-service framework
- Alice - Painless middleware chaining for Go
- YAM - Yet Another Mux
- Bone - Fast HTTP Router

Frameworks

- BeeGo Framework
- Frodo - Go mini web framework inspired by Laravel(php), Slim(php) and ExpressJS(node.js)
- GinGonic
- Macaron - Productive, modular web framework in Go.
- Revel Web Framework
- Ringo - Lighweight MVC web framework inspired by Rails, Gin.
- Utron - Lightweight MVC framework for web applications.

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

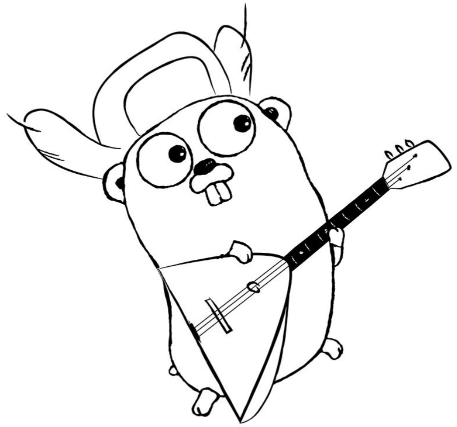
Who uses Go

Dropbox

YouTube

mongoDB

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



Thank You

GDG Sri Lanka

Monthly meetup November 2015 **NSBM**

Raveen Perera