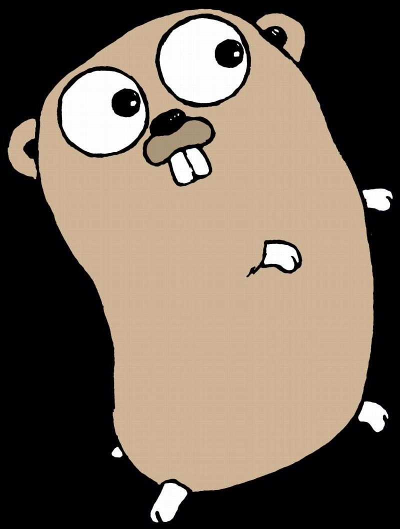
GO

An Introduction

Matt William Chase



GO GOPHER

GopherCon Renee French

Agenda

- Code Example 1: Binary Search
- Go Paradigm
- Go Translation
- Go Typing
- Go Scope
- Parameter Passing
- Arrays, Slices, Structs
- Code Example 2: QuickSort
- Go Domains
- Code Example 3: Web Applications

Binary Search

```go
/*
 * Executed at Runtime, Calls the Binary Search
*/
func main() {
    searchSpace := []int {1,2,3,4,5,6,7,8,9,10}
    fmt.Println(binarySearch(searchSpace, 2)) // 1
}
/*
 * Performs a Binary Search
*/
func binarySearch(space []int, target int) (int){

    // If target in space, min will hold index
    min := 0
    max := len(space) - 1
```

Go is Multi-Paradigm

- Imperative
- Functional *
- Object Oriented **

Functional

- Higher-Order Functions
- First Class Functions
- Closures
- Multiple Return Values
- Recursion

No: Immutable Variables, Pure Functions

Object Oriented

- Encapsulation
- Interfaces -> Polymorphism
- Type Member Functions
- Namespacing

No: Classes, Objects, Inheritance

Go is Compiled

- Fast Compile Times Were a Design Goal
- Uses a Compile / Linking Model
- Language Grammar is Regular

```
go run hello.go
```

Go is Statically & Strongly Typed

- Once defined, variables can not change type.

```
var x int = 10
```

- Type Inferences

```
x := 10
```

- No implicit conversions.
- Duck Typing with Interfaces / Structs

If it looks like a duck and quacks like a duck, it’s a duck.

Duck Typing

```go
package main
import "fmt"
type Singer interface{
   Singing(string)
}
type Person struct{
   name string
   age int
}
type Keyboard struct{
```

Go is Statically Scoped

Uses block scoping with brackets

Scoping

```go
func main(){
    x := 2
    if true{
        x := 3
        fmt.Println(x)  // 3
    }
    fmt.Println(x)      // 2
    printX()
}

```

Go Passes by Value & Pointer

Pass By Value / Pointer

```go
func main(){
    x := 1
    byValue(x)
    fmt.Println(x)  // 1
    byPointer(&x)
    fmt.Println(x)  // 2
}
func byValue(x int){
    x += 1
}
func byPointer(x *int){
```

Go Structural Variables

- Array, Slices, and Struct Types
- Arrays are numbered sequence of elements of a single type.
- Slices are a contiguous segment of an underlying array.
- Structs are sequence of named elements, called fields, each of which has a name and a type.

Structs

- Structs allow for "modeling" an object.
- Provide a method for type functions.

Structs

```go
func main(){
    myLine := Line{x1: 0, x2: 5, y1: 0, y2: 5}
    fmt.Println("Slope is ", myLine.slope())
    // Slope is 1
}
type Line struct {
    x1 float64
    x2 float64
    y1 float64
    y2 float64
```

Go Domains: Server Side

- Network applications need concurrency
- Go has native concurrency support (goroutines,channels)
- Examples: APIs, Web Servers, Web Apps
- Real World: Google App Engine, Docker, Revel

Go Domains: Command Line Interface (CLI)

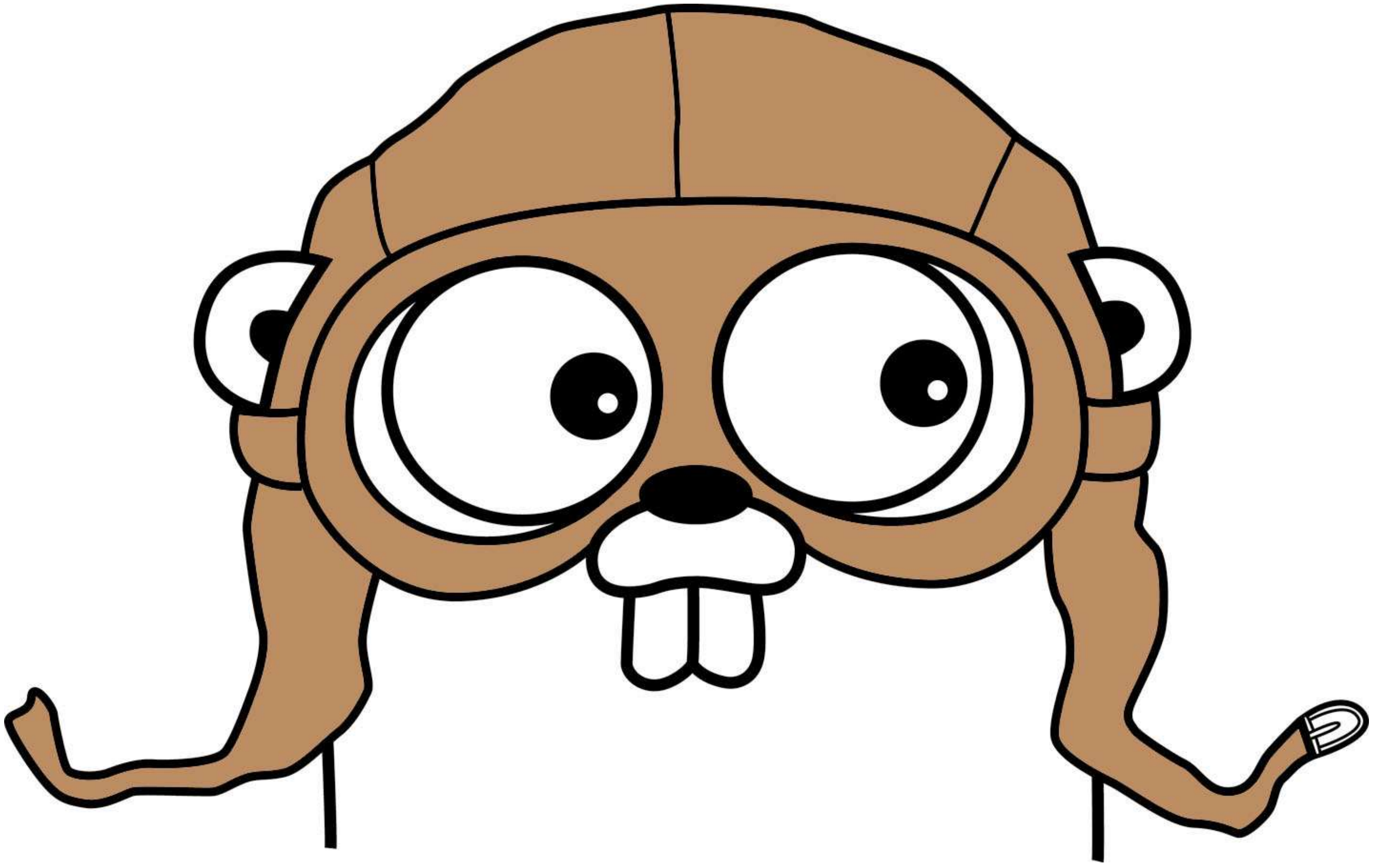
- Executables are stand-alone (No external dependencies)
- Consistent Behavior

Go Domains: Interesting Projects

Bibliography

- Sedar Yegulalp - What's the Go language really good for?
-

QuantScape/golang-research



Formidable React