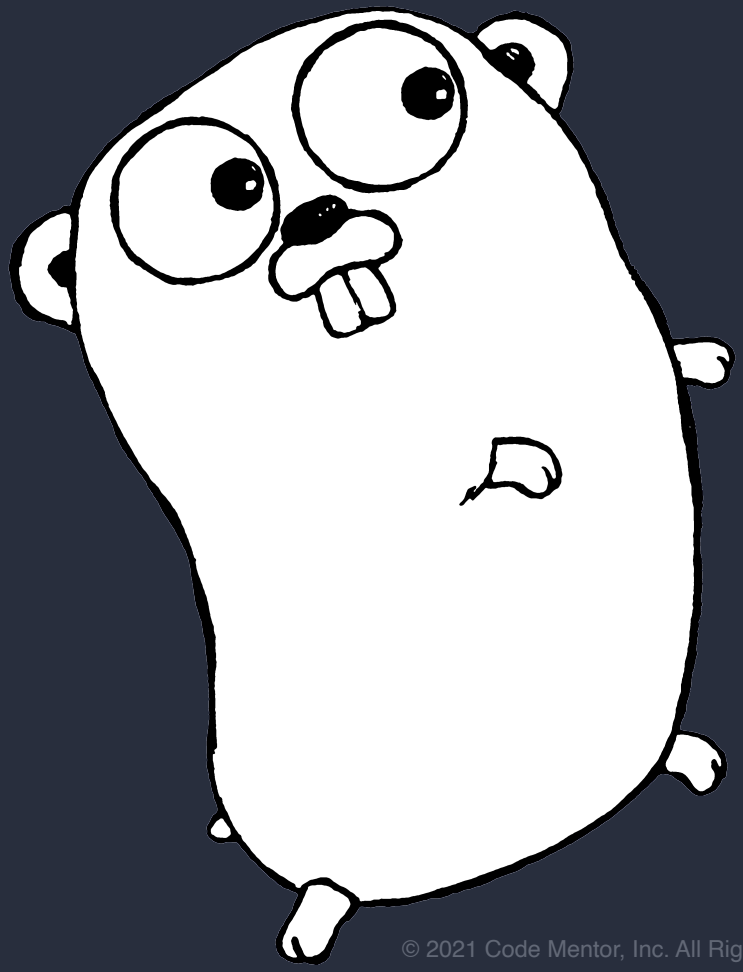
Go(lang) for Java Developers Workshop



Ken Sipe

Distribute Application Engineer

Apache Mesos Contributor, Kubernetes Commiter
Apache Committer Myriad, Open DCOS
Developer: Embedded, C++, Java, Groovy, Grails, C#, GoLang

@KenSipe

Agenda

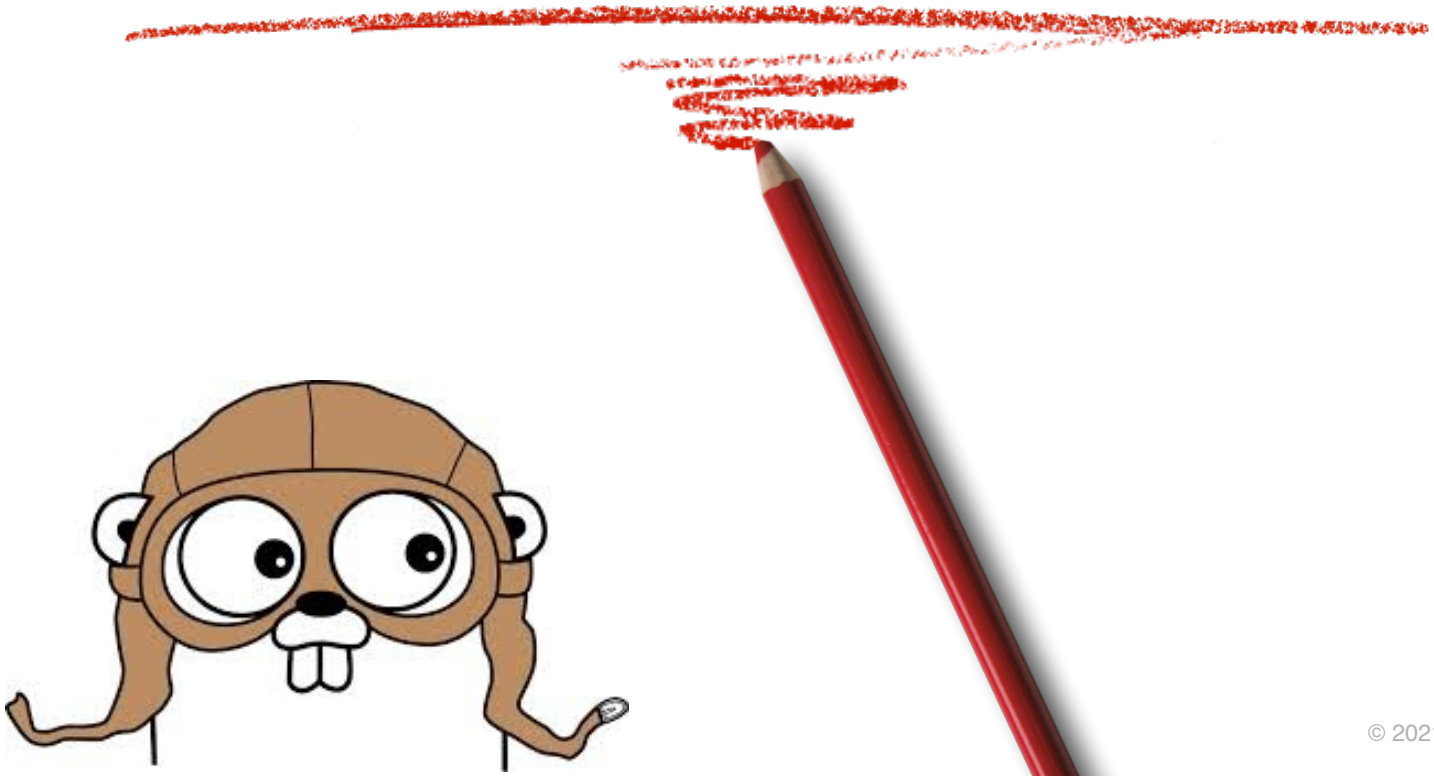
- Overview
- Getting Started
- Types
- Functions
- Branches / Loops
- Interfaces
- Concurrency

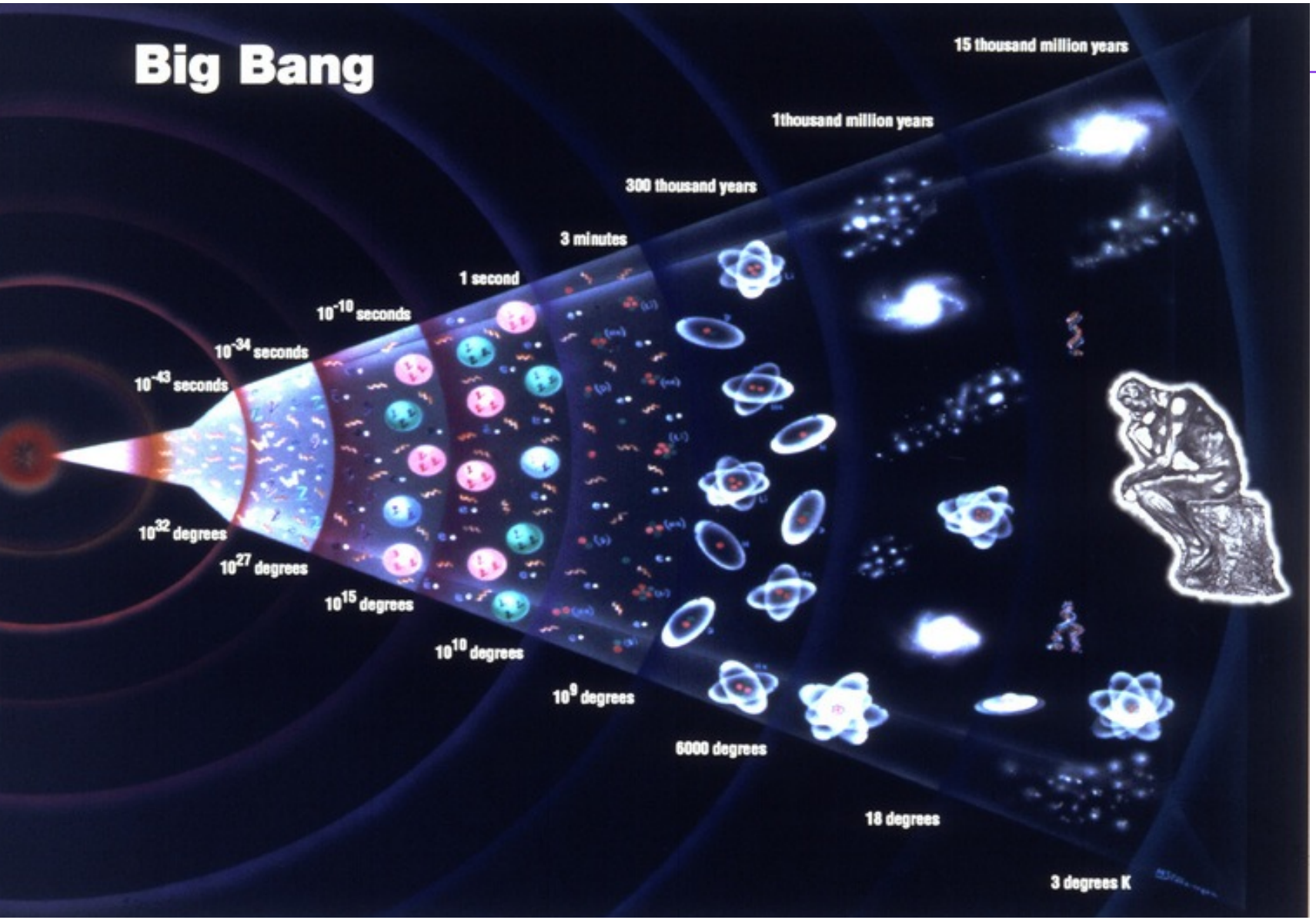
Workshop

- Git
- Go(lang): 1.13+
- Account (free): https://openweathermap.org/api

- Labs: https://github.com/kensipe/go-labs
- Solutions: https://github.com/codementor/wman

Go Overview





Go Releases

- Nov, 2009
  - Go Announced
- May, 2010
  - Go used internally at Google
- March, 2012
  - Go 1.0 Released
- Aug 24, 2018
  - Go 1.11
- Sept 9, 2019
  - Go 1.13
- Mar, 2022
  - Go 1.18.0 - Generics
- Feb, 2023
  - Go 1.20.0
- Aug, 2023 - built-in lang features
  - Go 1.21.0

Go Value Prop

- Easy to use language
  - increase productivity
  - reduce maintenance
  - keeping it simple
- Executional Efficient
- Type-safe
- latency-free GC
- Fast Compiles

Language Simplicity

- No Inheritance
- No Assertions
- No Method Overloading
- Implementing an Interface
- Inference assignment
- No Classes :)

- NOT a derived from the c-family

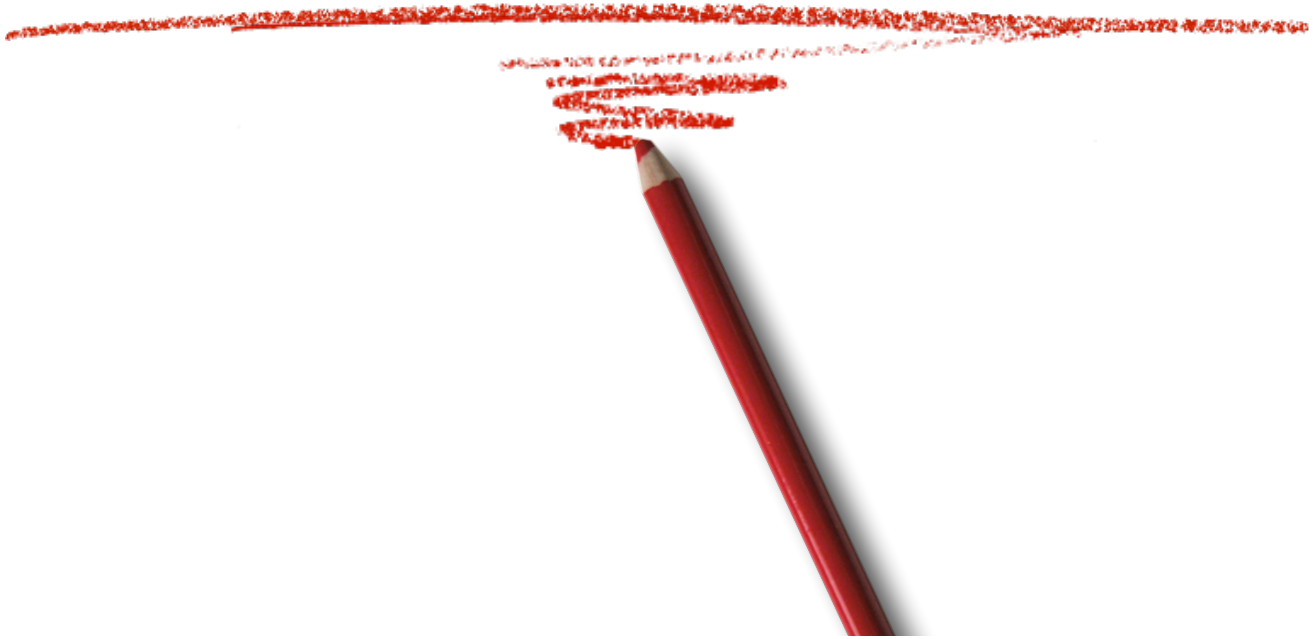
Golang is opinionated

- format is defined
  - go fmt <src>
- one way
- brackets {
- }
  - the way God intended them :)
- no semi-colons

Important Value Add

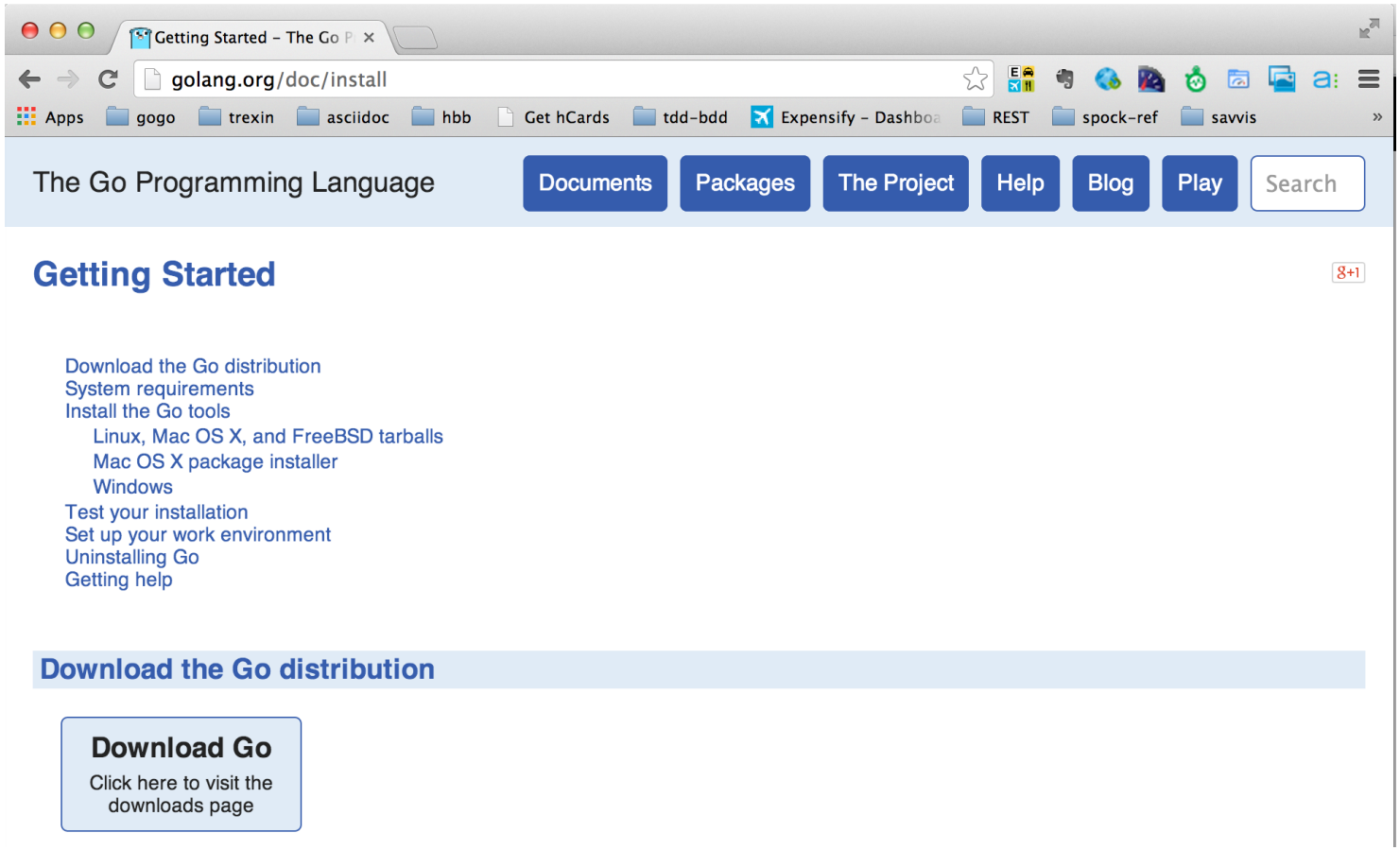
- Compiled
- Garbage Collected
- Concurrent

Getting Started



Multi-platform Go

Operating system	Architectures
FreeBSD 7 or later	amd64, 386, arm
Linux 2.6.23 or later with glibc	amd64, 386, arm
Mac OS X 10.6 or later	amd64, 386
Windows 2000 or later	amd64, 386



Lets Code

Java vs Go

- Must have a Class
- Main is part of a Class
- Source under <proj>/src
- Must compile than run with proper classpath
- By Convention class is in file of class name (# of files == # of classes)

- No classes
- Main is just main
- Full project is in the src tree GOPATH
- Functions in files… less of a convention

Tour of Go

http://tour.golang.org/#1

```
package main
import "fmt"

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

IDE

VSCode
https://code.visualstudio.com/
https://marketplace.visualstudio.com/items?itemName=ms-vscode.Go

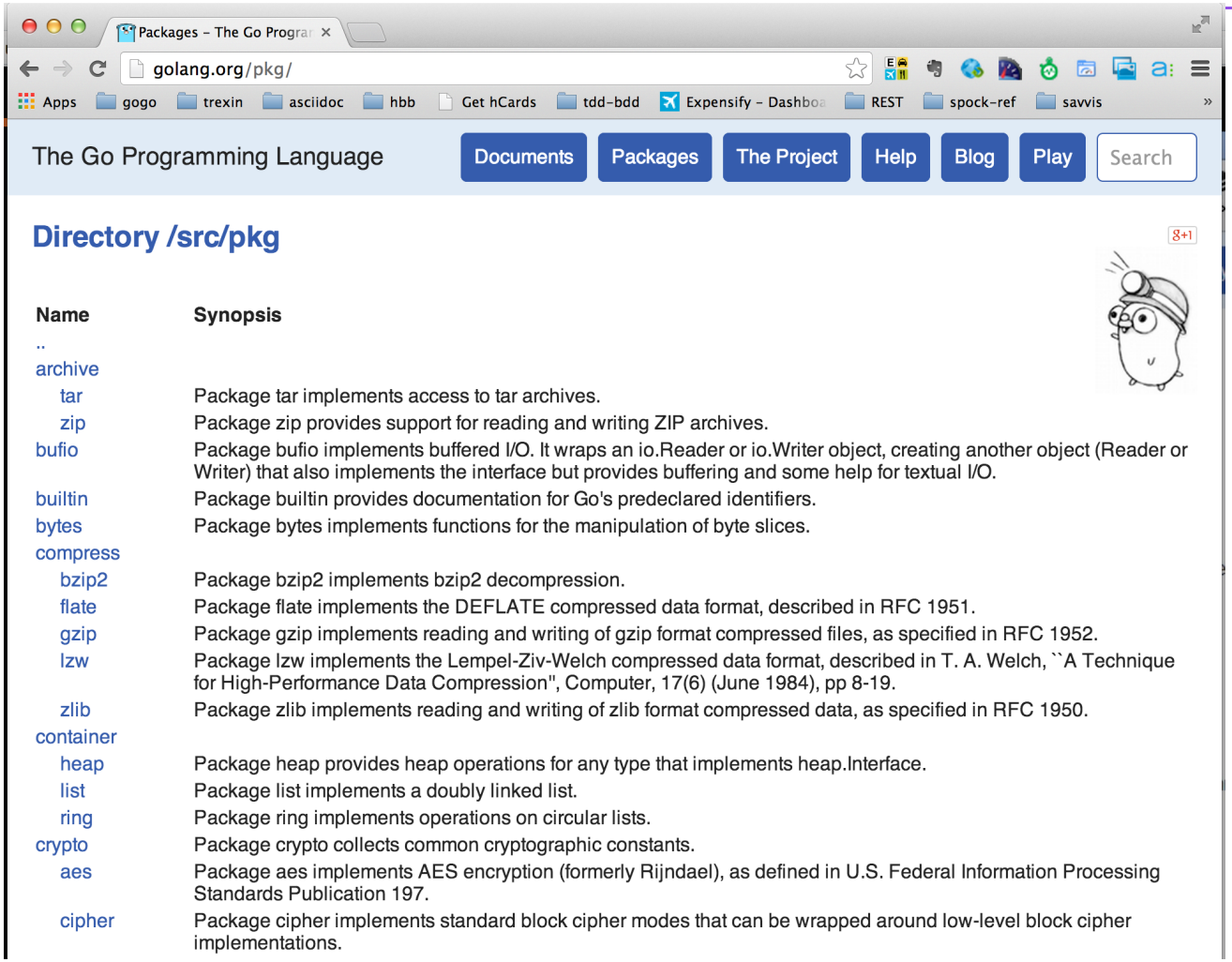
GoLand
https://www.jetbrains.com/go/

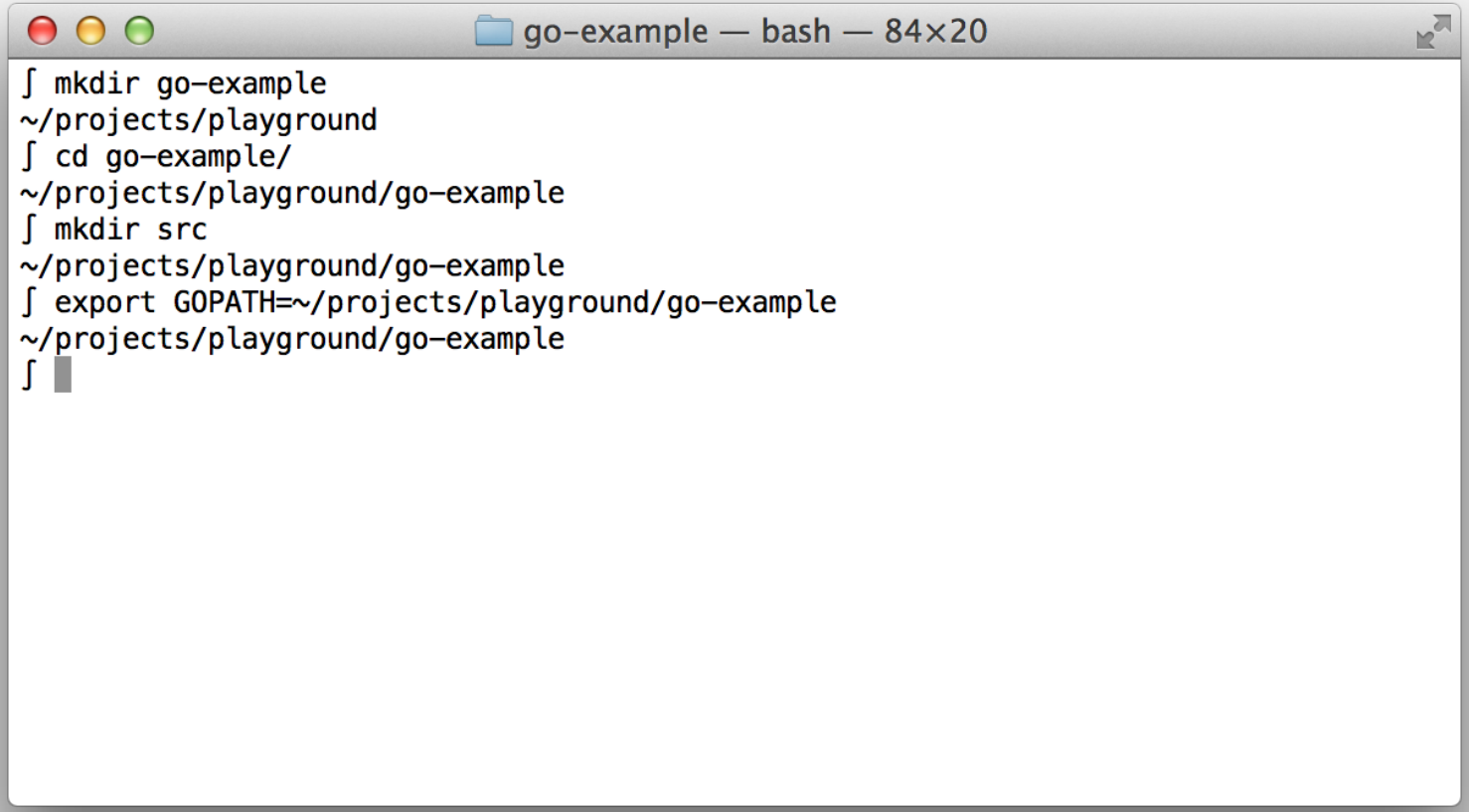
Packages

```
import (
    "fmt"
    "net/http"
)
```

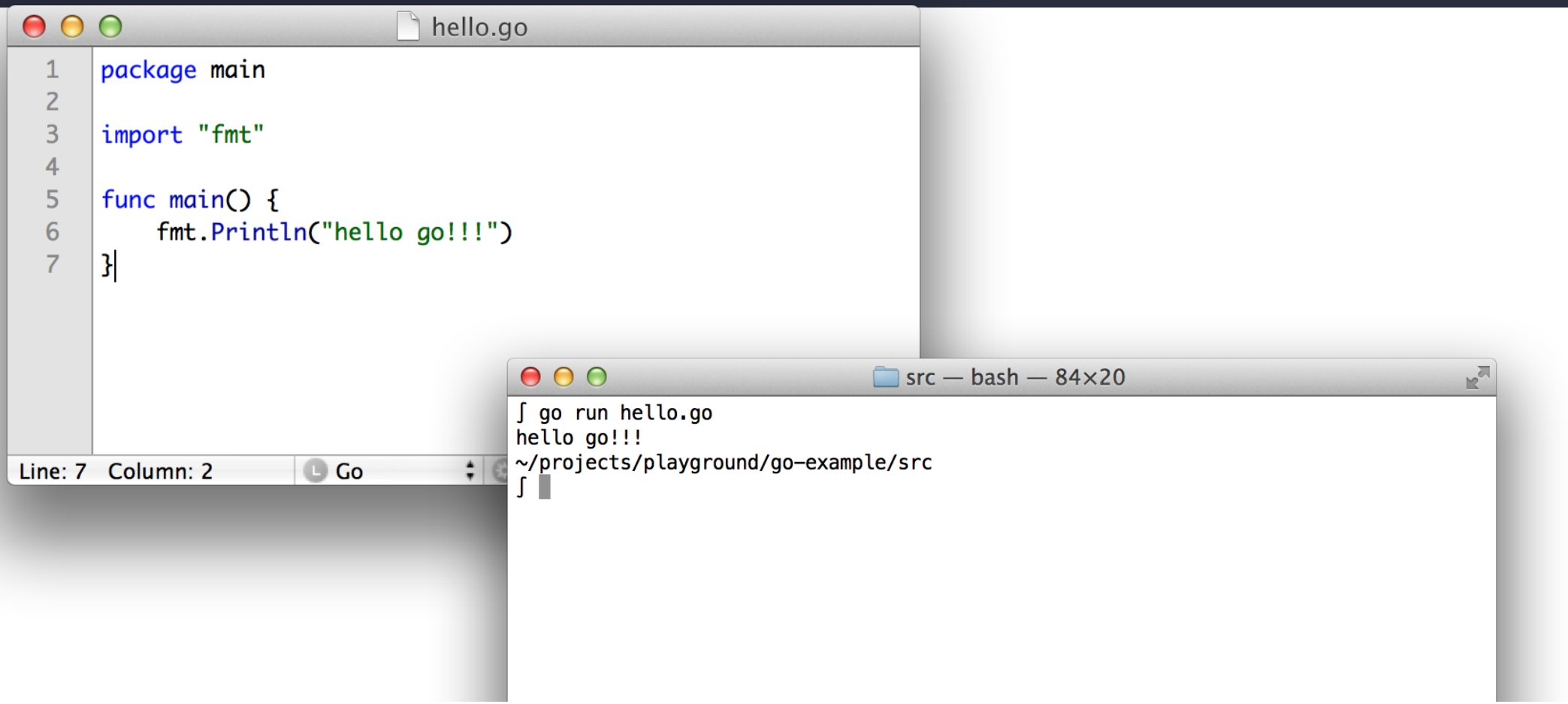
```
import  "fmt"
import  "net/http"
```

- Way to modularize Code
- Similar to namespaces





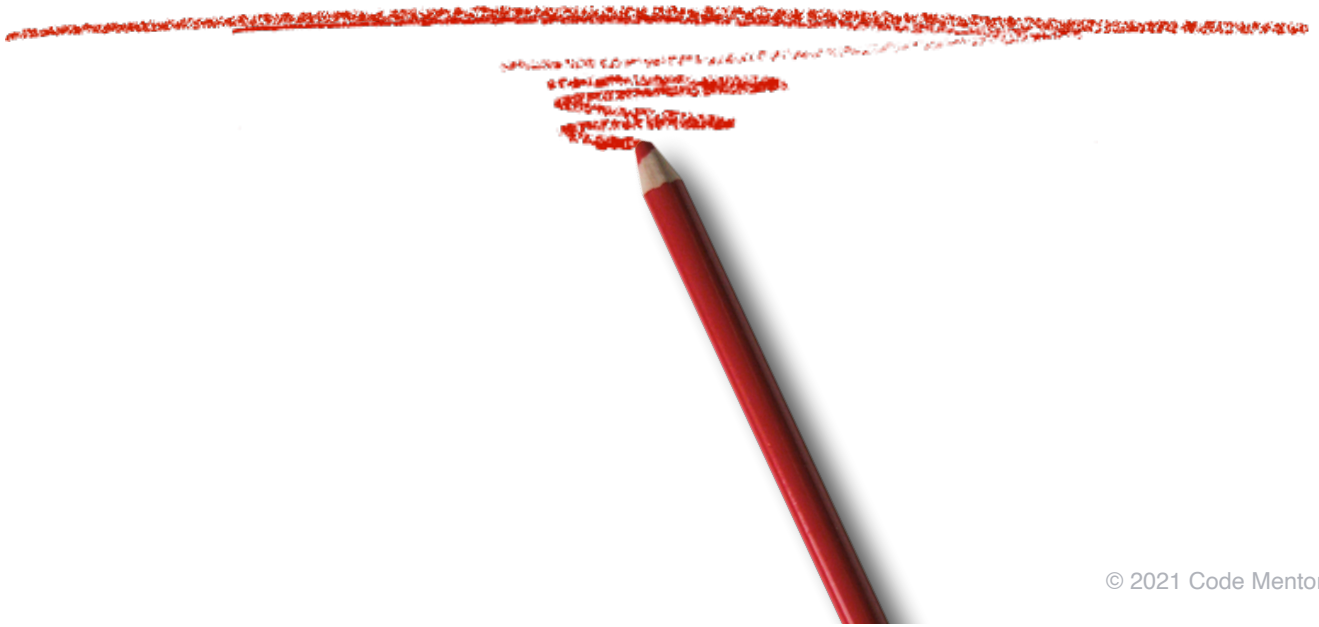
Hello World


hello.go
1 package main
2
3 import "fmt"
4
5 func main() {
6 fmt.Println("hello go!!!")
7 }
Line: 7 Column: 2
Go
src — bash — 84×20
∫ go run hello.go
hello go!!!
~/projects/playground/go-example/src
∫

Lab 1: Go Project Setup and Layout

https://github.com/kensipe/go-labs

Go Types and Collections



Types

- Basic Types
  - bool
  - string
  - int, int8, int16, int32, int64
  - uint, uint8, uint16, uint32, uint64, uintptr
  - byte (uint8)
  - rune (int32)
  - float32, float64
  - complex64, complex128

Types

- Other types
  - Array
  - Slice
  - Struct
  - Pointer
  - Function
  - Interface
  - Map
  - Channel

Define a variable

```
var message string

message = "hello go world"

fmt.Println(message)
```

```
var message string
var a, b, c int

message = "hello go world"

fmt.Println(message, a, b, c)
```

```go
var message string = "hello go world"
var a, b, c int = 1, 2, 3

fmt.Println(message, a, b, c)
```

type inference

```
var message = "hello go world"
var a, b, c  = 1, 2, 3

fmt.Println(message, a, b, c)
```

```
var message = "hello go world"
var a, b, c  = 1, true, 3

fmt.Println(message, a, b, c)
```

=

```
message := "hello go world"
a, b, c  := 1, true, 3

fmt.Println(message, a, b, c)
```

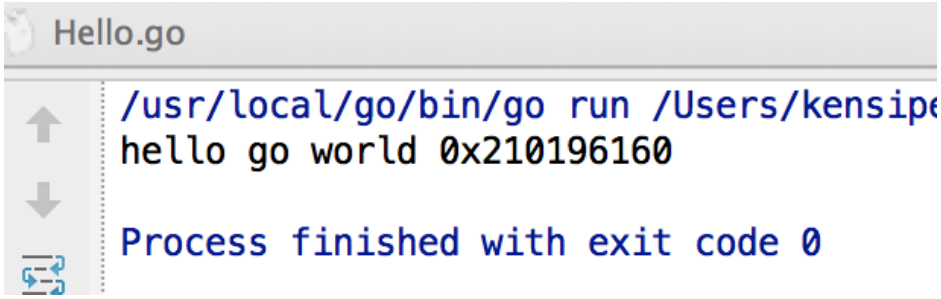
Passing Variables

- Values are passed by value
- need to pass by reference?
  - pass a pointer

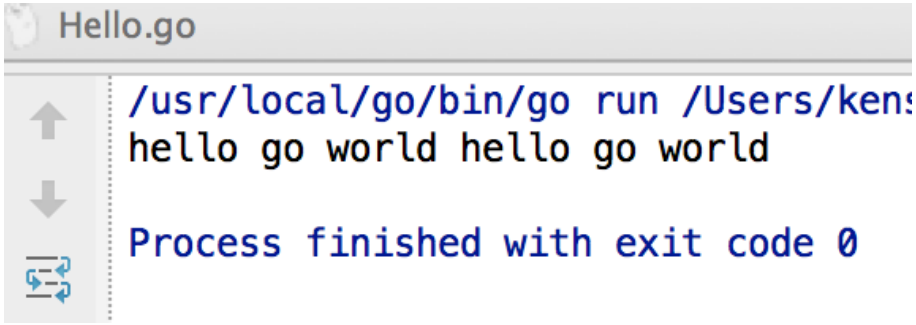
Passing by Reference

```go
func main() {

    message := "hello go world"
    a := 1

    foo(&a)
    fmt.Println(message, a)

}

func foo(count *int) {
    fmt.Println("count: ", *count)
    *count++
}
```

```go
message := "hello go world"

// pointers
var greeting *string = &message

fmt.Println(message, greeting)
```



```go
message := "hello go world"

// pointers
var greeting *string = &message

fmt.Println(message, *greeting)
```



- No Classes
- However…
- User Defined Types

User Defined Types

```
type Salutation string

func main() {

    var message Salutation = "hello go world"
    fmt.Println(message)

}
```

```go
type Salutation struct {
    name string
    greeting string
}

func main() {

    var message = Salutation{ "ken", "hello"}

    fmt.Println(message.greeting, message.name)

}
```

Struct Init Options

```
var message = Salutation{ "ken", "hello"}
```

```
var message = Salutation{name: "ken", greeting: "hello"}
```

```
var message = Salutation{}
message.name = "ken"
message.greeting = "hello"
```

Const

```
const (
    PI       = 3.14
    Language = "Go"
)

const (
    First  = iota
    Second = iota
    Third  = iota
)
```

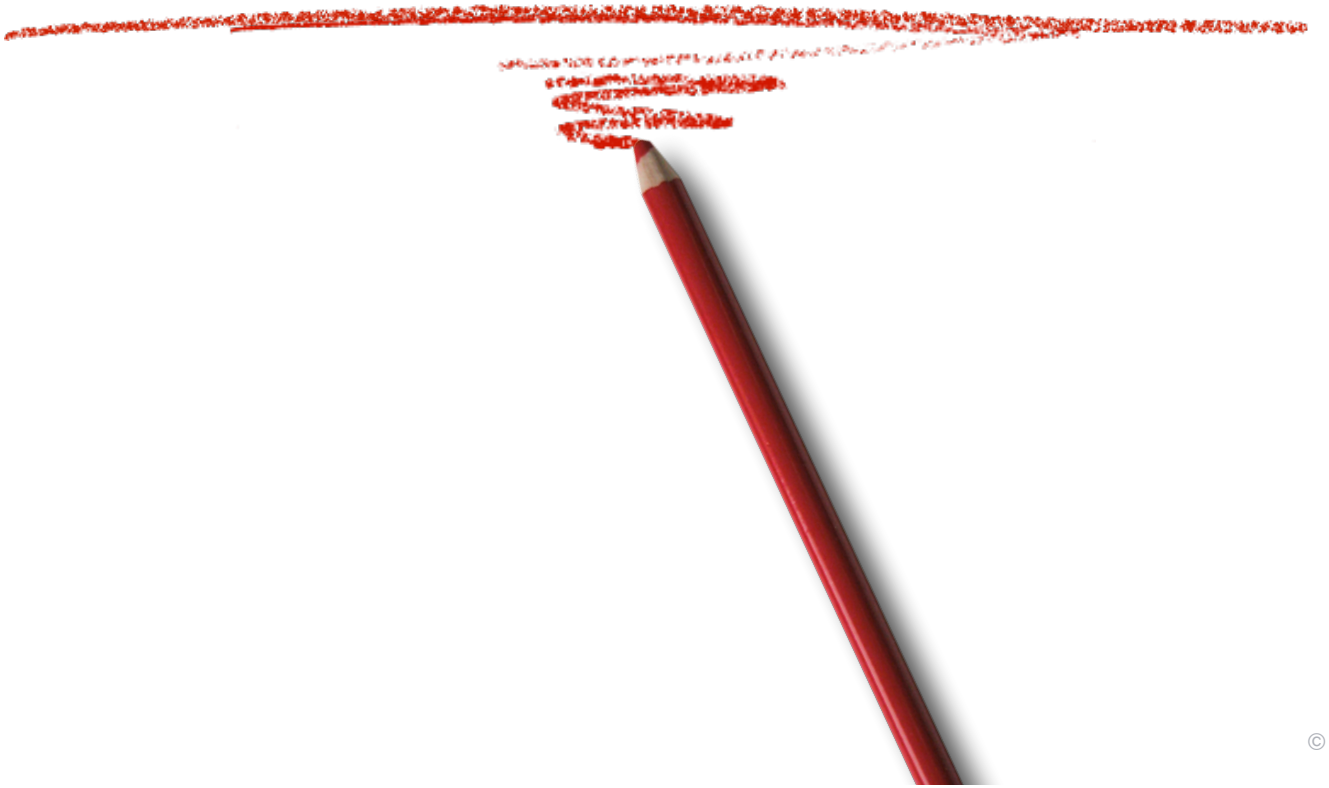
Maps

```
func GetPrefix(name string) (prefix string) {

    prefixMap := map[string]string {
        "Bob" : "Mr. ",
        "Joe" :  "Dr. ",
        "Amy" : "Dr. ",
        "Mary" : "Mrs. ",
    }

    return prefixMap[name]
}
```

Lab 2: Functions, Data Structures and Tests

https://github.com/kensipe/go-labs

Go Module



Go Modules

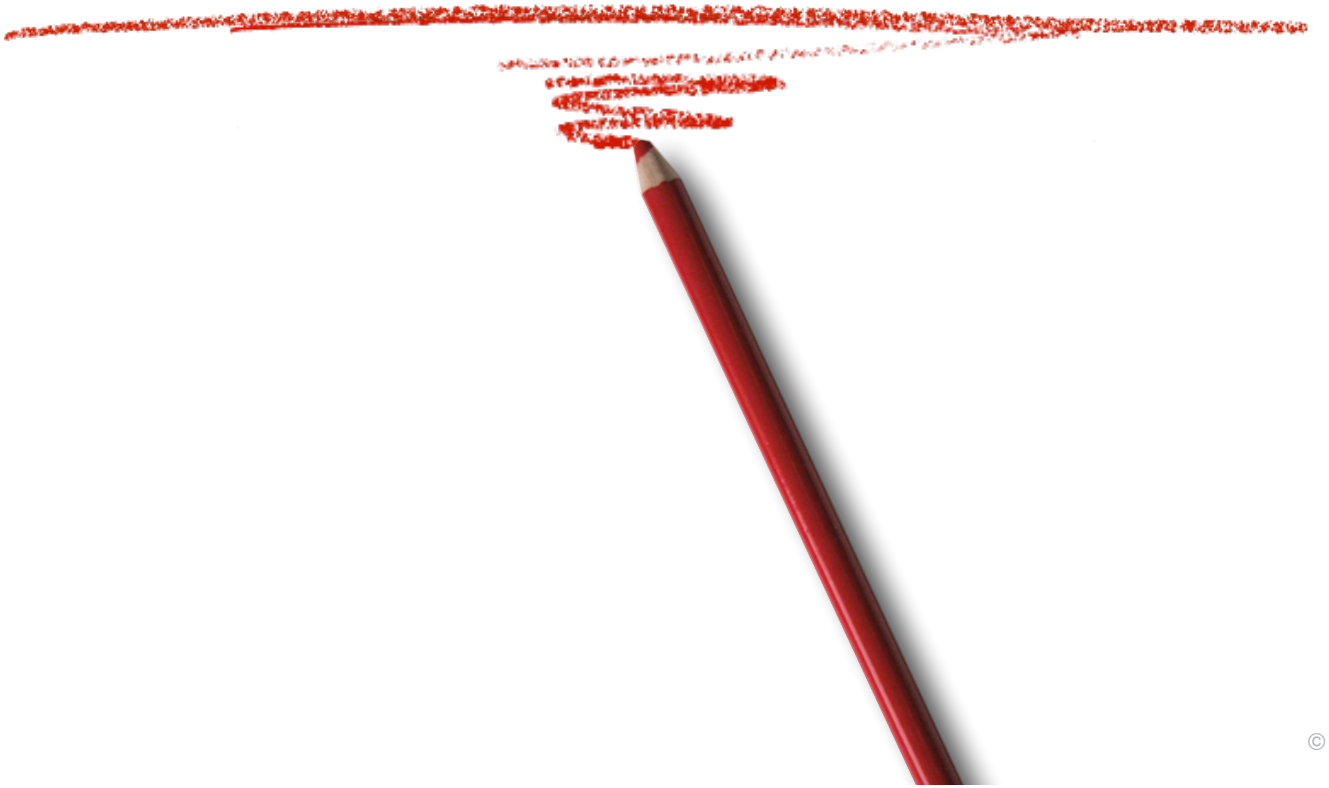
- Starting a project: Go mod init
- Getting dependencies: go get -u github.com/spf13/cobra/cobra

- Go mod verify
- Go mod tidy

Lab 3: Go Module Dependencies

https://github.com/kensipe/go-labs

Go Func



Go Functions

- Multiple return values
- Use it like a type (pass functions)
- function literals (Closure)
  - a function inside a function

```go
type Salutation struct {
    name     string
    greeting string
}

func Greet(message Salutation) {
    fmt.Println(message.greeting, message.name)
}

func main() {

    var message = Salutation{}
    message.name = "ken"
    message.greeting = "hello"

    Greet(message)

}
```

```go
type Salutation struct {
    name     string
    greeting string
}

func Greet(message Salutation) {
    fmt.Println(CreateMessage(message.greeting, message.name))
}

func CreateMessage(greeting, name string) string {
    return greeting + " " + name
}

func main() {

    var message = Salutation{}
    message.name = "ken"
    message.greeting = "hello"

    Greet(message)

}
```

```go
func Greet(message Salutation) {
    greeting, altgreeting := CreateMessage(message.greeting, message.name)

    fmt.Println(greeting)
    fmt.Println(altgreeting)
}

func CreateMessage(greeting, name string) (string, string) {
    return greeting + " " + name, "hey! " + name
}
```

```go
func Greet(message Salutation) {
    _, altgreeting := CreateMessage(message.greeting, message.name)

    fmt.Println(altgreeting)
}

func CreateMessage(greeting, name string) (string, string) {
    return greeting + " " + name, "hey! " + name
}
```

```
func CreateMessage(greeting, name string) (message string, altgreeting string) {
    message = greeting+" "+name
    altgreeting = "hey! "+name
    return
}
```

Variadic Functions

```
func Greet(message Salutation) {
    _, altgreeting := CreateMessage(message.greeting, message.name, "Vic")

    fmt.Println(altgreeting)
}

func CreateMessage(greeting string, name ...string) (message string, altgreeting string)
    message = greeting+" "+name[0]
    altgreeting = "hey! "+name[1]
    return
}
```

Methods

```
func (message Salutation) Greet(do Printer) {
    greeting, altgreeting := CreateMessage(message.greeting, message.name, "Vic")

    do(greeting)
    do(altgreeting)
}
```

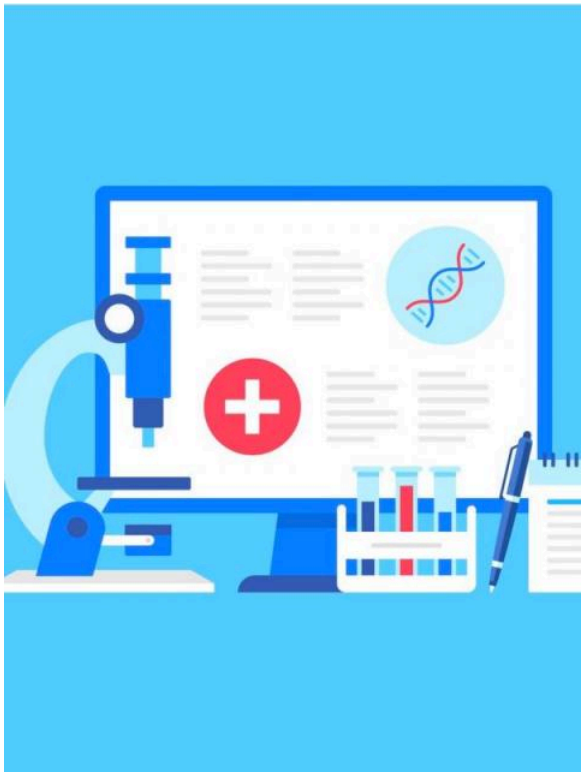
```
func main() {

    var message = Salutation{}
    message.name = "ken"
    message.greeting = "hello"

    message.Greet(CreatePrintFunction("!!!"))

}
```

```go
func (message Salutation) Rename(newName string) {
    message.name = newName
}
```

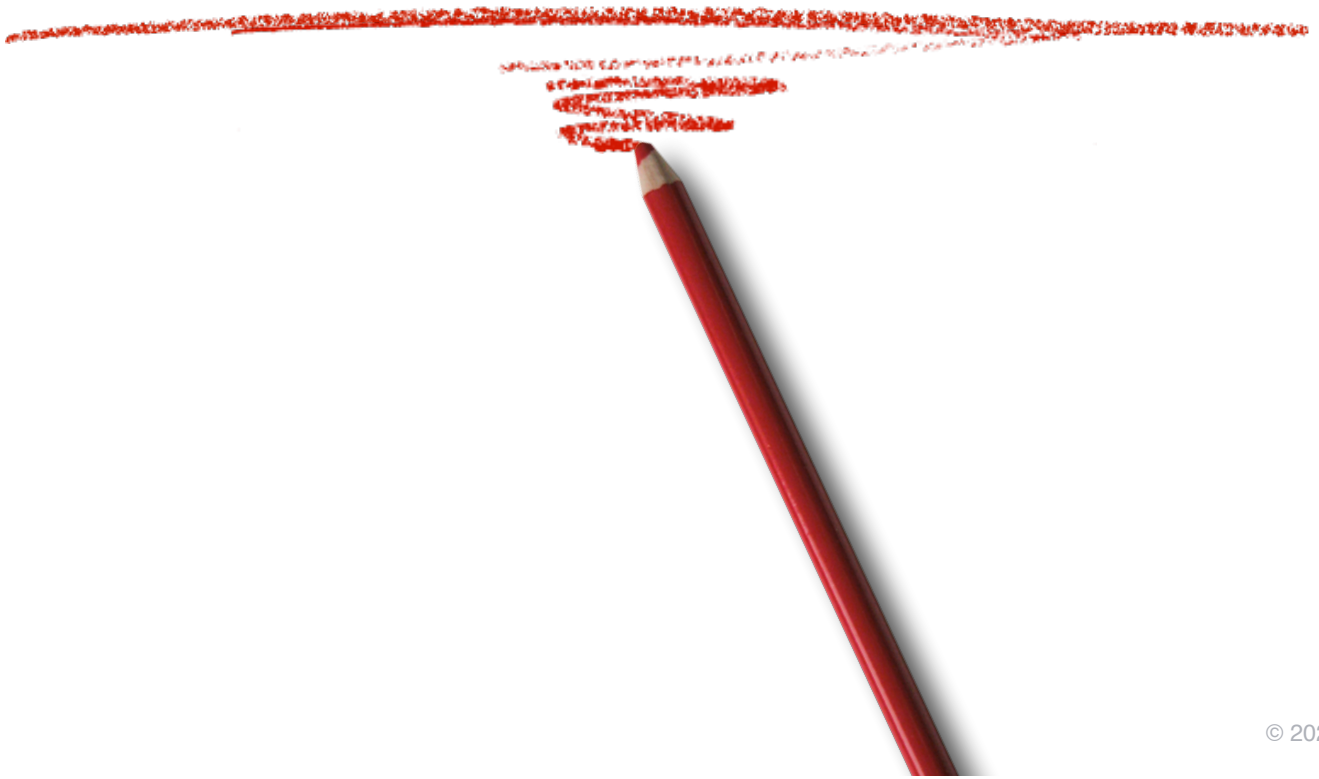
```go
func (message *Salutation) Rename(newName string) {
    message.name = newName
}
```

Lab 4: Data Conversion, Linting And Build Tags

https://github.com/kensipe/go-labs



Interfaces



Interfaces in Go

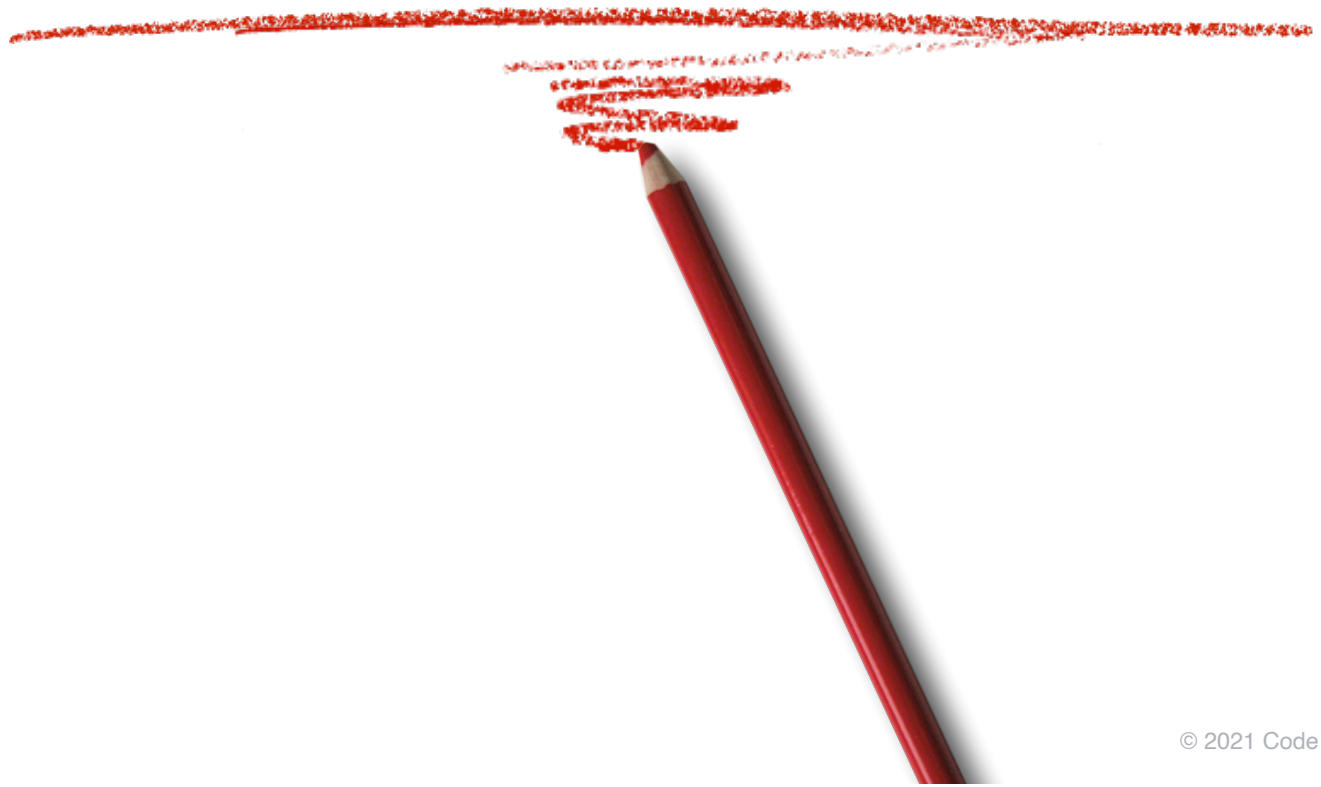
- If a type has the methods of an interface…
- then it implements that interface

```go
type Renamable interface {
    Rename(newName string)
}

func RenameToJoe(r Renamable) {
    r.Rename("Joe")
}

func (message *Salutation) Rename(newName string) {
    message.name = newName
}
```

Universal Interface

- Everything implements
  - interface{}

Useful Pkgs



Useful Packages

package	purpose	examples
fmt	formatting i/o	Printf, Scanf
os	OS interface	Open, Read, Write
strconv	numbers <-> strings	Atoi, Atof, Itoa
log	event logging	Logger, Printf
bytes	byte arrays	Compare, Buffer

Useful Packages

package	purpose	examples
bufio	i/o buffers	
strings	string utils	
path/filepath	files and directories	

fmt

- fmt.Printf("%d %d %g\n", 1, 2, 3.5)
- fmt.Print(1, " ", 2, " ", 3.5, "\n")
- fmt.Println(1, 2, 3.5)

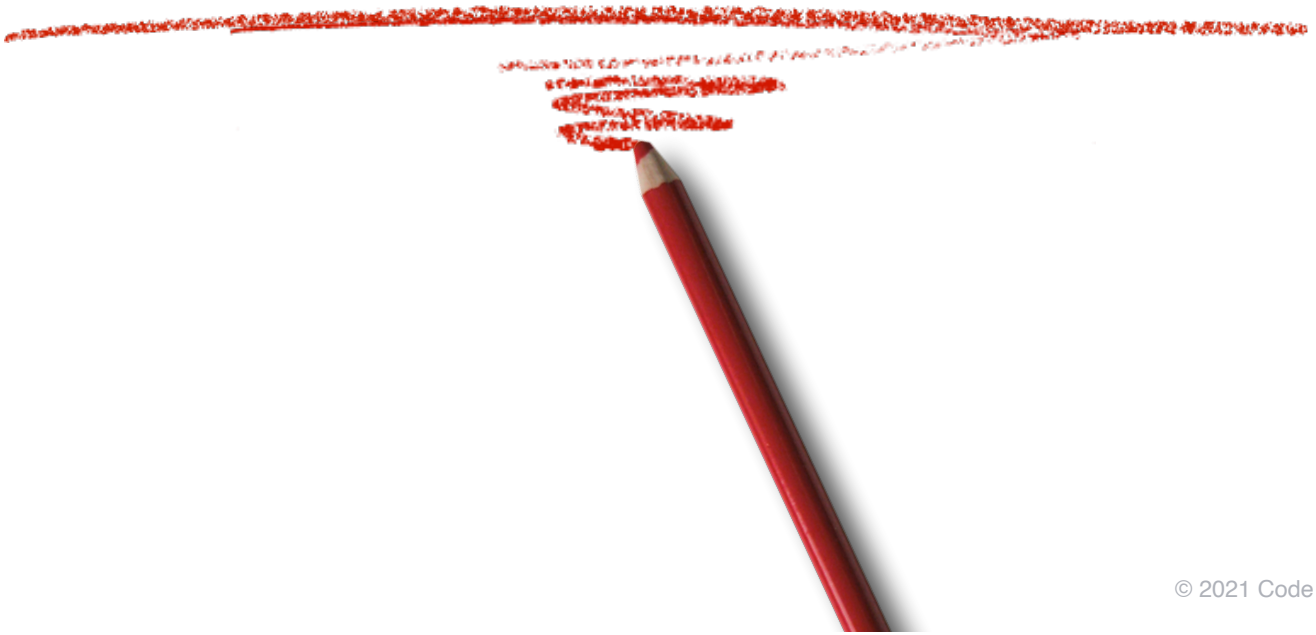
Getting Help

- Commands:
  - go doc fmt
  - go doc fmt.PrintF

Lab 5: JSON Decoding and Files

https://github.com/kensipe/go-labs

Control Structures



if Statement

```
func (message Salutation) Greet(do Printer, isFormal bool) {
    greeting, altgreeting := CreateMessage(message.Greeting, message.Name, "Vic")

    if isFormal {
        do(greeting)
    } else {
        do(altgreeting)
    }
}
```

Scoping with If

```
func (message Salutation) Greet(do Printer, isFormal bool) {
    greeting, altgreeting := CreateMessage(message.Greeting, message.Name, "Vic")

    if title := "Mr. "; isFormal {
        do(title + greeting)
    } else {
        do(altgreeting)
    }
}
```

Switch

```
func (message Salutation) Greet(do Printer, isFormal bool) {
    greeting, altgreeting := CreateMessage(message.Greeting, message.Name, "Vic")

    if title := GetPrefix(message.Name); isFormal {
        do(title + greeting)
    } else {
        do(altgreeting)
    }
}

func GetPrefix(name string) (prefix string) {

    switch name {
        case "Bob" : prefix = "Mr. "
        case "Joe", "Amy" : prefix = "Dr. "
        case "Mary" : prefix = "Mrs. "
        default : prefix = "Dude "
    }

    return
}
```

Switch on Type

```
func TypeSwitch(x interface{}) {
    switch x.(type) {
    case int:
        fmt.Println("int")
    case string:
        fmt.Println("int")
    case Salutation:
        fmt.Println("Salutation")
    default:
        fmt.Println("unknown")
    }
}
```

Looping

For Loop

```
func (message Salutation) Greet(do Printer, isFormal bool, times int) {

    greeting, altgreeting := CreateMessage(message.Greeting, message.Name, "Vic")


    for i := 0; i < times; i ++ {

        if title := GetPrefix(message.Name); isFormal {
            do(title + greeting)
        } else {
            do(altgreeting)
        }
    }
}
```

Ranges

- Type of Ranges
  - Array
  - Slice
  - String
  - map
  - Channel

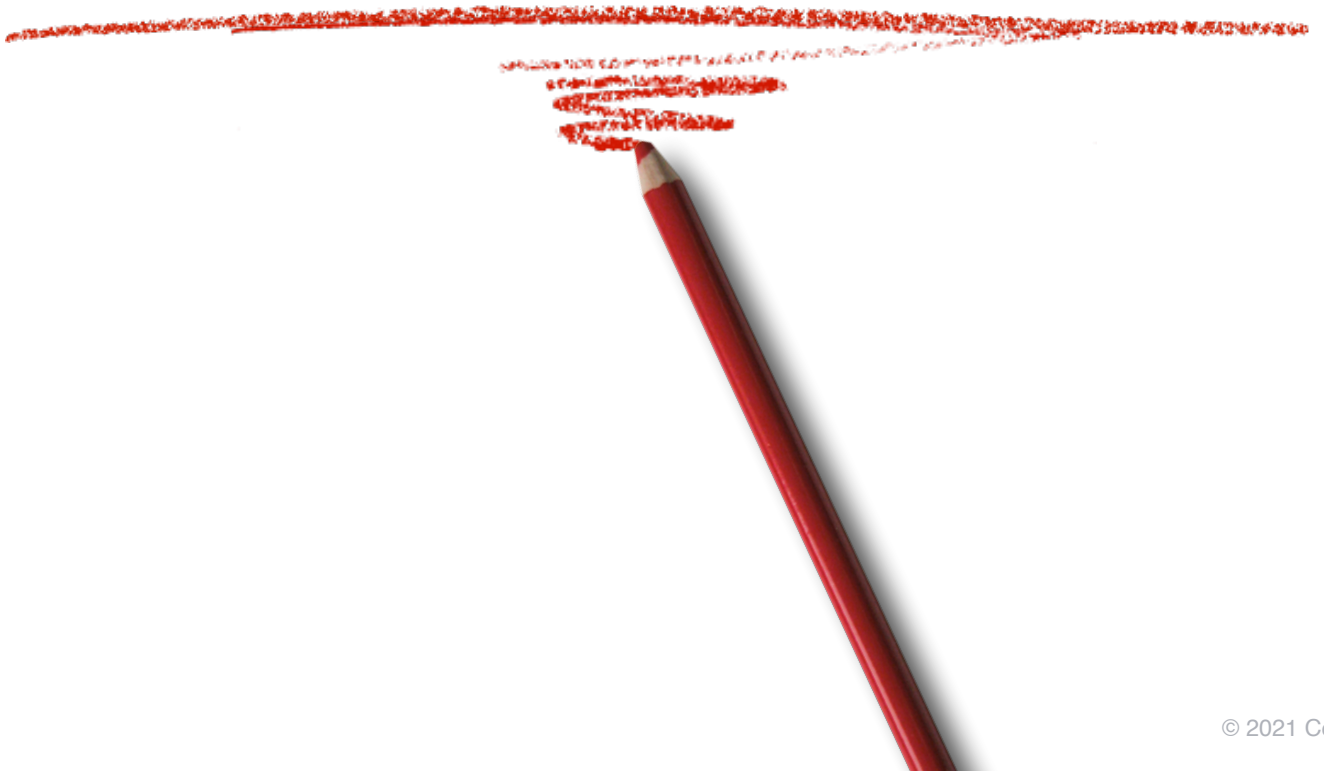
For Range

```
func Greet(message []Salutation, do Printer, isFormal bool, times int) {

    for _, s := range message {

        greeting, altgreeting := CreateMessage(s.Greeting, s.Name, "Vic")
        if title := GetPrefix(s.Name); isFormal {
            do(title + greeting)
        } else {
            do(altgreeting)
        }
    }
}
```

Lab 6: HTTP Requests to Go Data Structures

https://github.com/kensipe/go-labs

Concurrency



- Go Routines
- Channels

Channels

- ch := make(chan int)
  - create new channel
- ch <- 5
  - write to channel
- i := <- ch
  - read from channel

Lab 7: Go Routines and Channels

https://github.com/kensipe/go-labs

Thank You!

@kensipe

kensipe@gmail.com

https://k8s.slack.com/#/