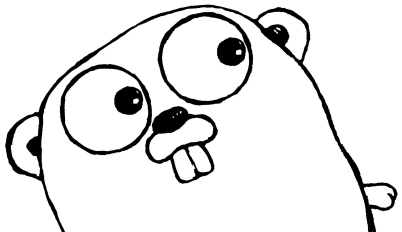
Google

The Go Programming Language

An Introduction

Wednesday April 6th, 2016

Brandon Sprague



Overview

Introduction

Basics

Data Structures

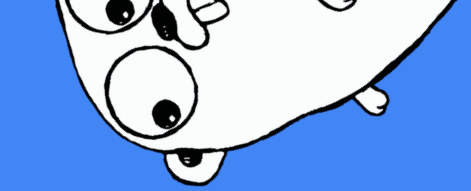
Concurrency

Interfaces

Writing Good Go Code

The Fun Stuff

Resources



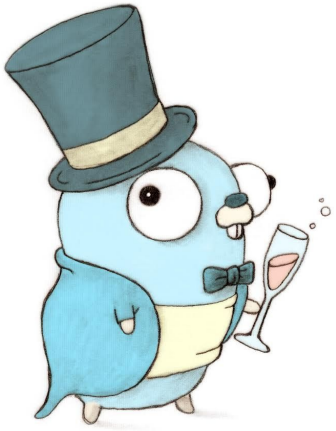
Introduction

Background

History

High-Level Overview

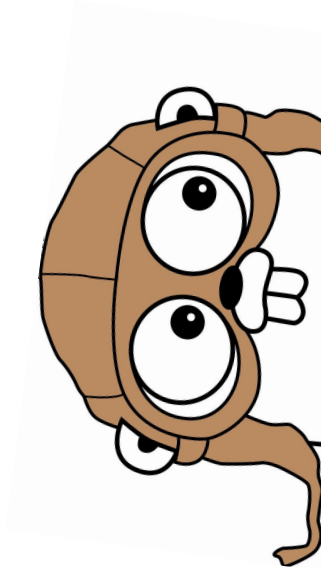
Background



Literally nothing

Stuff I've Built in Go

- ECE Project
  - [https://github.com/bcspragu/ReachabilityAnalyzer]
- Radiotation
  - [https://github.com/bcspragu/Radiotation]
- Gobots
  - [https://github.com/bcspragu/Gobots]
- Blog

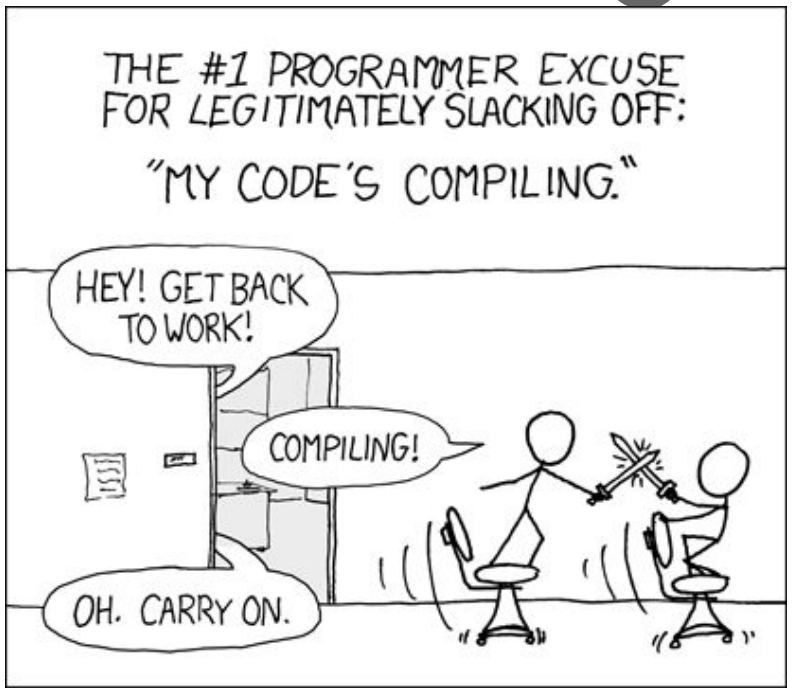


```
Foo foo = Foo.newBuilder().setFoosity(Foo.SOME_FOO).build();
```

```
foo:Foo *myFoo = new foo::Foo(foo::FOO_INIT)
```

History

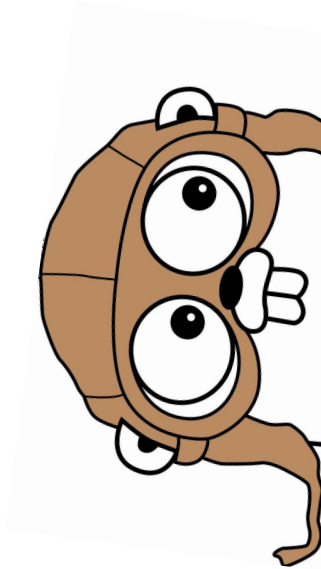
What does Google need?



So... what did they come up with?

Go’s Main Traits

- Compiles quickly
- Concurrency built-in
- Garbage-collected
- Statically-typed
- Runs on pretty much everything
- “Object-oriented”
  - But no classes
  - Or inheritance



Basics

Hello World

Declaring Variables

Control Structures

Functions

Packages

Hello World

```
package main

import "fmt"

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

Variables

```
package main

import "fmt"

func main() {
   var i, j int = 1, 2
   k := 3
   c, python, java := true, false, "no!"
  // Prints 1 2 3 true false no!
   fmt.Println(i, j, k, c, python, java)
}
```

Types

```
import (
    "fmt"
    "math/cmplx"
)

var (
    ToBe   bool       = false
    MaxInt uint64     = 1<<64 - 1
    z      complex128 = cmplx.Sqrt(-5 + 12i)
)

func main() {
    const f = "%T(%v)\n"
    fmt.Printf(f, ToBe, ToBe) // bool(false)
    fmt.Printf(f, MaxInt, MaxInt) // uint64(18446744073709551615)
    fmt.Printf(f, z, z) // complex128((2+3i))
}
```

Functions can be variables/types too

```
func adder() func(int) int {
    sum := 0
    return func(x int) int {
        sum += x
        return sum
    }
}
```

Zero values

```
import "fmt"

func main() {
   var i int
   var f float64
   var b bool
   var s string
   // Prints 0 0 false ""
   fmt.Printf("%v %v %v %q\n", i, f, b, s)
}
```

Type Conversions

```
import (
   "math"
)

func main() {
   var x, y int = 3, 4
   var f float64 = math.Sqrt(float64(x*x + y*y))
   var z uint = uint(f)
}
```

Constants

```
import "fmt"

const Pi = 3.14

func main() {
    const World = "世界"
    fmt.Println("Hello", World)
    fmt.Println("Happy", Pi, "Day")

    const Truth = true
    fmt.Println("Go rules?", Truth)
}
```

For Loops

```
import "fmt"

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

“While” Loops

```
import "fmt"

func main() {
   sum := 1
   for sum < 1000 {
       sum += sum
   }
   fmt.Println(sum)
}
```

Infinite Loops...what

```
package main

func main() {
   for {
   }
}
```

If Statements

```
import (
    "fmt"
    "math"
)

func sqrt(x float64) string {
    if x < 0 {
        return sqrt(-x) + "i"
    }
    return fmt.Sprint(math.Sqrt(x))
}
```

Fancy If Statements

```
import (
   "math"
)

func pow(x, n, lim float64) float64 {
   if v := math.Pow(x, n); v < lim {
       return v
   }
   return lim
}
```

Switch

```
import (
    "runtime"
)

func main() {
    switch os := runtime.GOOS; os {
    case "darwin":
         // Do Mac Stuff
    case "linux":
         // Do Stallman stuff
    default:
         // Do freebsd, openbsd,
         // plan9, windows... stuff
    }
}
```

More Switch

```
import (
    "fmt"
    "time"
)

func main() {
    t := time.Now()
    switch {
    case t.Hour() < 12:
          fmt.Println("Good morning!")
    case t.Hour() < 17:
          fmt.Println("Good afternoon.")
    default:
          fmt.Println("Good evening.")
    }
}
```

Defer

```
import "fmt"

func main() {
   defer fmt.Println("world")

   fmt.Println("hello")
}
```

Packages

```
import (
	"bytes"
	"encoding/gob"
	"errors"
	"fmt"
	"log"
	"sort"
	"strings"
	"time"

	"zombiezen.com/go/capnproto2"

	"github.com/bcspragu/Gobots/botapi"
	"github.com/boltdb/bolt"
)
```

Data Structures

Structs

Arrays

Slices

Maps

Structs

```
import "fmt"

type Vertex struct {
    X int
    Y int
}

func main() {
    v := Vertex{1, 2}
    v.X = 4
    fmt.Println(v.X)
}
```

Structs ‘n Pointers

```
import "fmt"

type Vertex struct {
    X int
    Y int
}

func main() {
    v := Vertex{Y: 2}
    p := &v
    p.X = 1e9
}
```

Methods

```
type Vertex struct {
    X, Y float64
}

func (v Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}

func (v *Vertex) Scale(f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}
```

More Methods

```
type MyFloat float64

func (f MyFloat) Abs() float64 {
    if f < 0 {
        return float64(-f)
    }
    return float64(f)
}
```

Arrays

```
import "fmt"

func main() {
   var a [2]string
   a[0] = "Hello"
   a[1] = "World"
   fmt.Println(a[0], a[1])
   fmt.Println(a)
}
```

Slices

```
import "fmt"

func main() {
   primes := []int{2, 3, 5, 7, 11, 13}

   for i := 0; i < len(primes); i++ {
       fmt.Printf("primes[%d] == %d\n", i, primes[i])
   }
}
```

Slicing Slices

```
import "fmt"

func main() {
   s := []int{2, 3, 5, 7, 11, 13}

   fmt.Println("s[1:4] ==", s[1:4]) // s[1:4] == [3 5 7]

   fmt.Println("s[:3] ==", s[:3]) // s[:3] == [2 3 5]

   fmt.Println("s[4:] ==", s[4:])  // s[4:] == [11 13]
}
```

Making Slices

```
import "fmt"

func main() {
    // Also used for maps and channels
    a := make([]int, 5)
}
```

Nil Slices and Adding Elements

```
import "fmt"

func main() {
   var s []int
   if s == nil {
       // Do stuff
   }
   s = append(s, 1)
}
```

For … range

```
var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}

func main() {
    for i, v := range pow {
        // Access i and v here
    }
    for _, value := range pow {
        // If you only need the value
    }
    for i := range pow {
        // If you only need the index
    }
}
```

Maps

```
type Vertex struct {
    Lat, Long float64
}

var m map[string]Vertex

func main() {
    m = make(map[string]Vertex)
    m["Bell Labs"] = Vertex{
        40.68433, -74.39967,
    }
}
```

More Maps

```
type Vertex struct {
    Lat, Long float64
}

var m map[string]Vertex

func main() {
    m = make(map[string]Vertex)
    m["Bell Labs"] = Vertex{
        40.68433, -74.39967,
    }
    if e, ok := m["Menlo Park"]; ok {
        // Do something with e
    }
    delete(m, "Bell Labs")
}
```

Interfaces

An example

The Empty Interface: interface{}

Usage in Standard Library

Interfaces

```
import “math"

type Abser interface {
	Abs() float64
}

func main() {
	var a Abser
	f := MyFloat(-math.Sqrt2)
	v := Vertex{3, 4}

	a = f  // a MyFloat implements
Abser
	a = &v // a *Vertex implements
Abser
}
```

```
type MyFloat float64

func (f MyFloat) Abs() float64 {
	if f < 0 {
		return float64(-f)
	}
	return float64(f)
}

type Vertex struct {
	X, Y float64
}

func (v *Vertex) Abs() float64 {
	return math.Sqrt(v.X*v.X +
v.Y*v.Y)
}
```

The Empty interface{}

```
import "fmt"

func do(i interface{}) {
    switch v := i.(type) {
    case int:
        fmt.Printf("Twice %v is %v\n", v, v*2)
    case string:
        fmt.Printf("%q is %v bytes long\n", v, len(v))
    default:
        fmt.Printf("I don't know about type %T!\n", v)
    }
}
```

interface{} in the Standard Library

```
// In the encoding/json package
func Marshal(v interface{}) ([]byte, error)
func Unmarshal(data []byte, v interface{}) error

// In the reflect package
func DeepEqual(x, y interface{}) bool
func ValueOf(i interface{}) Value
```

Interfaces in the Standard Library

```
// In the fmt package
type Stringer interface {
    String() string
}

// In the io package
type Reader interface {
        Read(p []byte) (n int, err error)
}
type Writer interface {
        Write(p []byte) (n int, err error)
}
```

Concurrency

Goroutines

Channels

Select Statement

Goroutines

```
import (
	"fmt"
	"time"
)

func say(s string) {
	for i := 0; i < 5; i++ {
		time.Sleep(100 * time.Millisecond)
		fmt.Println(s)
	}
}

func main() {
	go say("world")
	say("hello")
}
```

Channels

```
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum // send sum to c
}
```

```
func main() {
    s := []int{7, 2, 8, -9, 4, 0}

    c := make(chan int)
    go sum(s[:len(s)/2], c)
    go sum(s[len(s)/2:], c)
    x, y := <-c, <-c // receive from c

    fmt.Println(x, y, x+y)
}
```

The Select Statement

```
func fibonacci(c, quit chan int) {
    x, y := 0, 1
    for {
        select {
        case c <- x:
            x, y = y, x+y
        case <-quit:
            fmt.Println("quit")
            return
        }
    }
}
```

Writing Good Go Code

Idiomatic Go

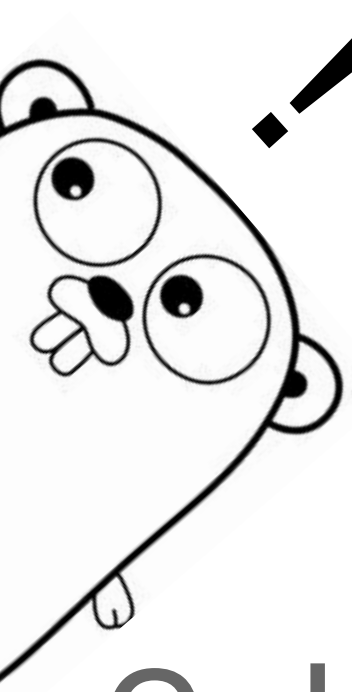
Testing

Using the tooling

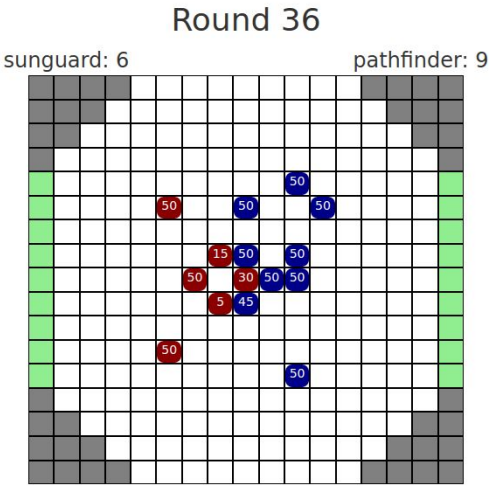
Google

The Fun Stuff

Built to play



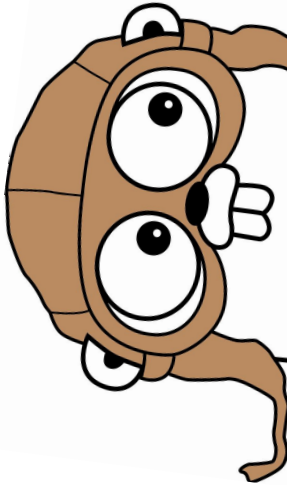
Gobots!



Useful Resources and Tools

From the Go Authors

- Installing Go
  - https://golang.org/doc/install
- Effective Go - Goes through each feature and when to use them
  - https://golang.org/doc/effective_go.html
- Standard Library Documentation (also available via `go doc`)
  - https://golang.org/pkg/
- Go Frequently Asked Questions
  - https://golang.org/doc/faq
- Official Go Blog
  - https://blog.golang.org/



From...other places

- Documentation for building a Gobot
  - https://godoc.org/github.com/bcspragu/Gobots/game
  - Rules for RobotGame, which GobotGame is based on: https://robotgame.net/rules
- Info on using io.Reader
  - https://www.datadoghq.com/blog/crossing-streams-love-letter-gos-io-reader/
- All sorts of Go examples
  - https://gobyexample.com

What’s next?

Questions?