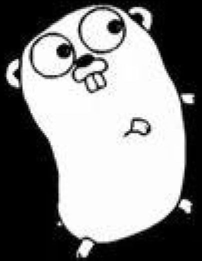
GO



http://golang.org

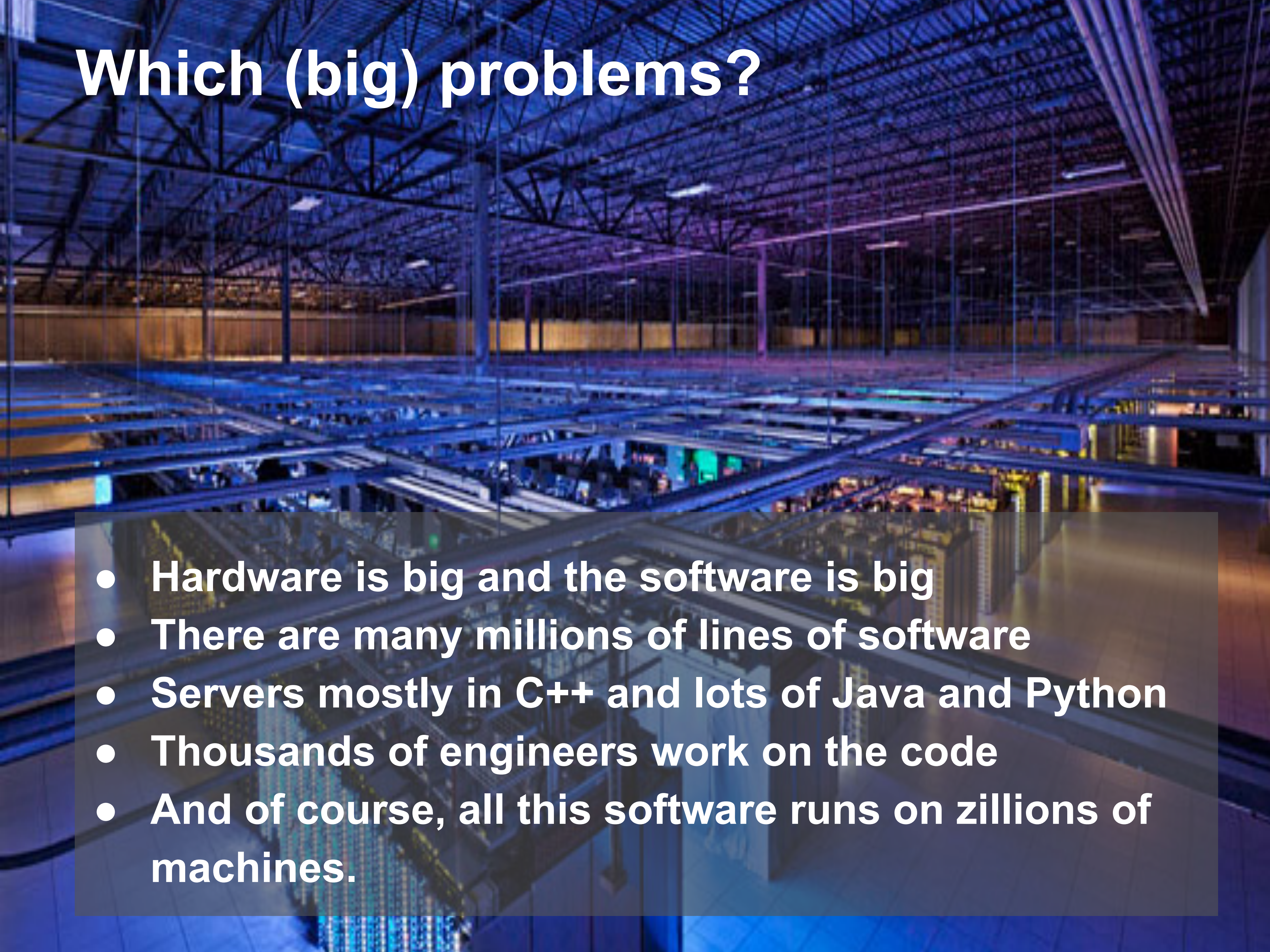
Suelen Goularte Carvalho
suelengc@ime.usp.br

Computação Paralela e Distribuída
Profº Alfredo Goldman
Junho/2015

What is Go?

Go is a programming language designed by Google to help solve ***Google's problems***.

And Google has big problems!



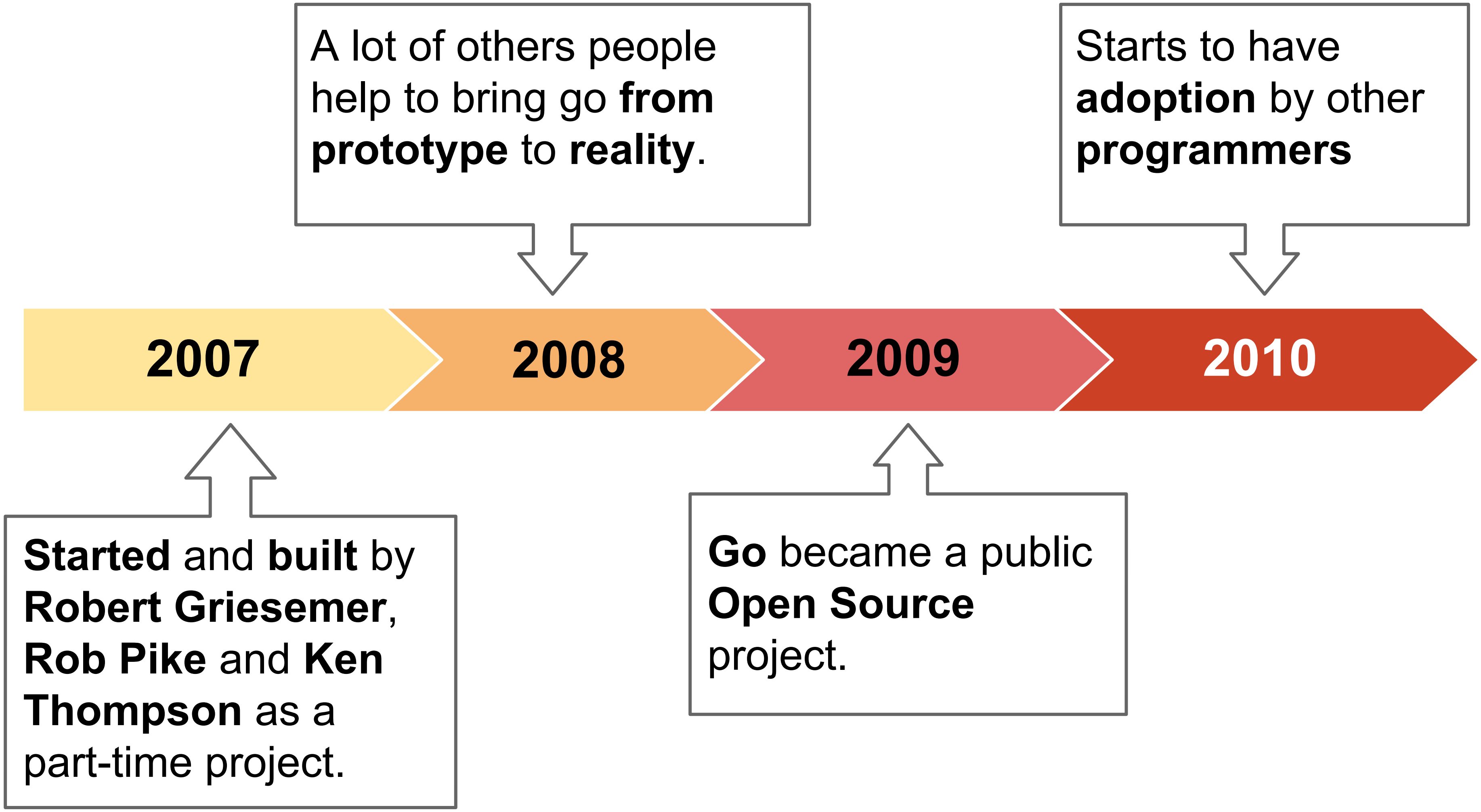
Which (big) problems?

- Hardware is big and the software is big
- There are many millions of lines of software
- Servers mostly in C++ and lots of Java and Python
- Thousands of engineers work on the code
- And of course, all this software runs on zillions of machines.

“ *In short, development at* Google *is <u>big</u>, can be <u>slow</u>, and is often <u>clumsy</u>. But it is* **effective**. ”

https://talks.golang.org/2012/splash.article

History



Started and **built** by **Robert Griesemer**, **Rob Pike** and **Ken Thompson** as a part-time project.

A lot of others people help to bring go **from prototype** to **reality**.

Go became a public **Open Source** project.

Starts to have **adoption** by other **programmers**

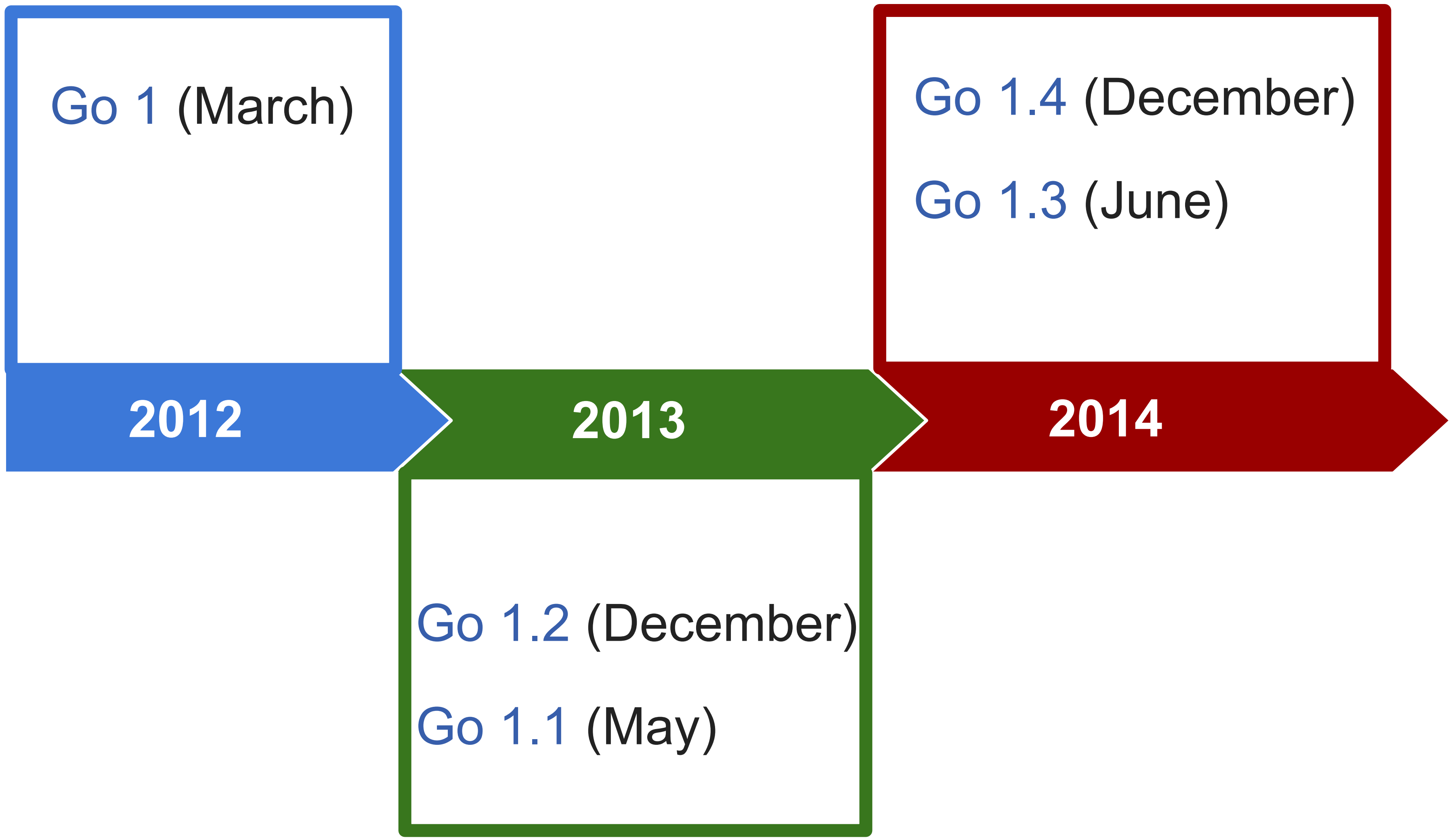
https://golang.org/doc/faq#history

Who were the founders?

- Ken Thompson (B, C, Unix, UTF-8)
- Rob Pike (Unix, UTF-8)
- Robert Griesemer (Hotspot, JVM)

...and a few others engineers at Google

Version history



https://golang.org/project/

Why Go?

- Eliminate slowness
- Eliminate clumsiness
- Improve productive
- Maintain (and improve) scale

*It was designed **by** and **for** people who write, read, debug and maintain large software systems.*

*Go's purpose is **not** to do research programming language design.*

Go's purpose is to make its designers' programming lives better.

What is Go?

Go is a compiled, concurrent, garbage-collected, statically typed language developed at Google.

Go is a **tool** for managing **Go** source code...

Mainly tools:

build	compile packages and dependencies
run	compile and run Go program
clean	remove object files
env	print Go environment information
test	test packages and benchmarks

Others tools:

fix, fmt, get, install, list, tool,
version, vet.

Who are using today?

SOUNDCLOUD

nic.br registro.br

globo.com

CANONICAL

Google

docker

NOKIA
Connecting People

mozilla
FOUNDATION

bitly

heroku

YouTube

vimeo

amazon.com

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

What will you see in Go?

- ❑ Compiled
- ❑ Garbage-collected
- ❑ Has your own runtime
- ❑ Simple syntax
- ❑ Great standard library
- ❑ Cross-platform
- ❑ Object Oriented (without inheritance)
- ❑ Statically and stronger typed
- ❑ Concurrent (*goroutines*)
- ❑ Closures
- ❑ Explicity dependencies
- ❑ Multiple return values
- ❑ Pointers
- ❑ and so on...

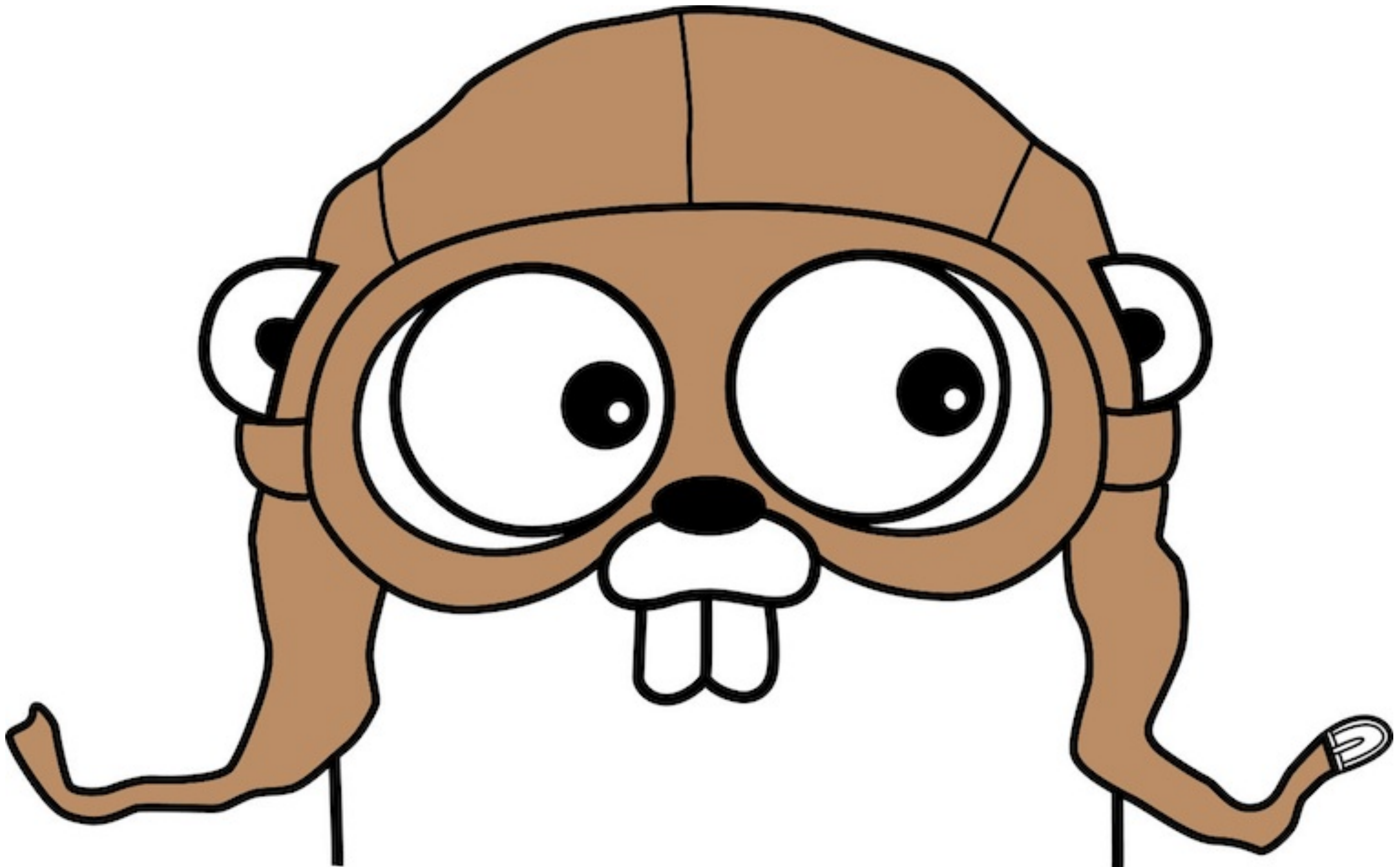
What will you not see in Go?

- ❑ Exception handling
- ❑ Inheritance
- ❑ Generics
- ❑ Assert
- ❑ Method overload

Have not been implemented in favor of efficiency.

GO

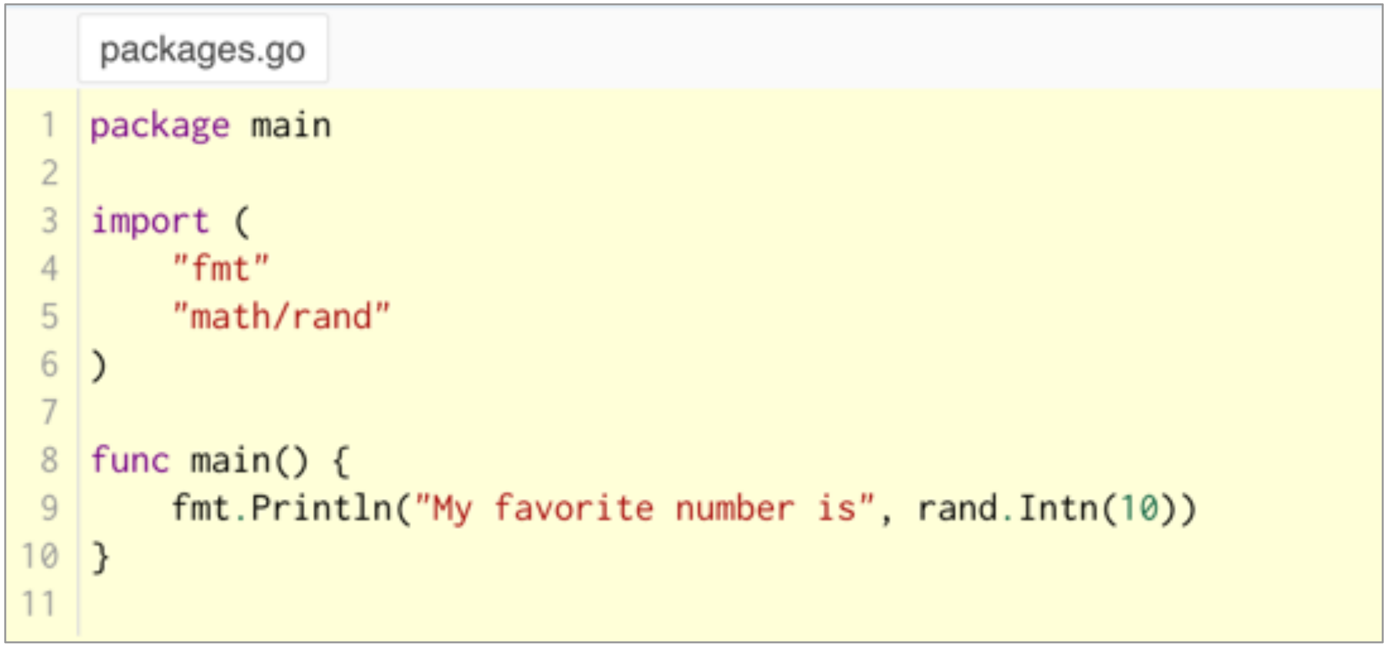
see a bit of `code`!



Packages

- Each Go program are compound per packages
- Programs starts from main package
- This example are using the **`ftm`** and **`math`** packages

```
$ go run packages.go
My favorite number is 1
```



packages.go

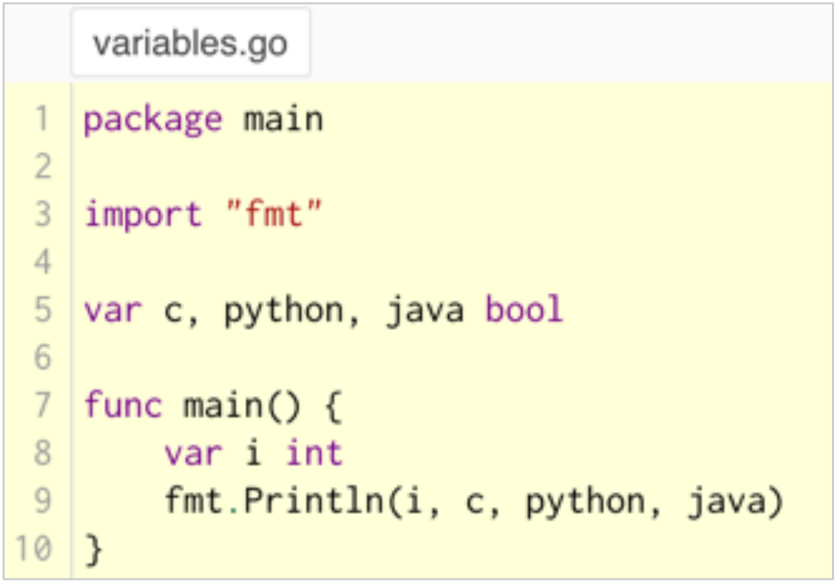
```go
package main

import (
	"fmt"
	"math/rand"
)

func main() {
	fmt.Println("My favorite number is", rand.Intn(10))
}

```

Variables

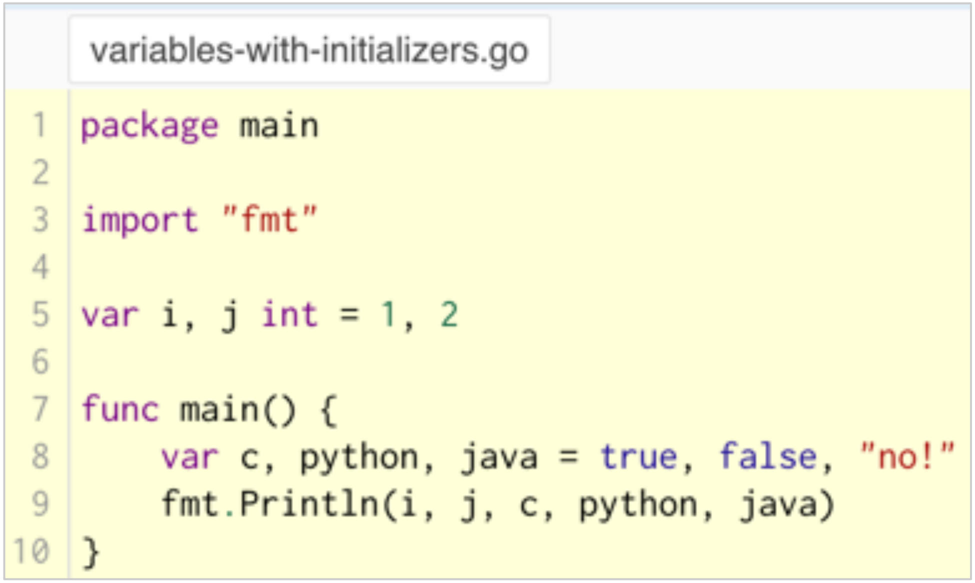
- The **`var`** instruction declares a list of variables
- The type is informed at the end
- The **`var`** instruction could be in a package or in a function
- The **`var`** instruction could includes initializers, 1 per variable. In this case, the type could be ommited because it will be inferred

```
$ go run variables.go
0 false false false
```

```
$ go run variables-with-initiali
1 2 true false no!
```


variables.go

```go
package main

import "fmt"

var c, python, java bool

func main() {
    var i int
    fmt.Println(i, c, python, java)
}
```

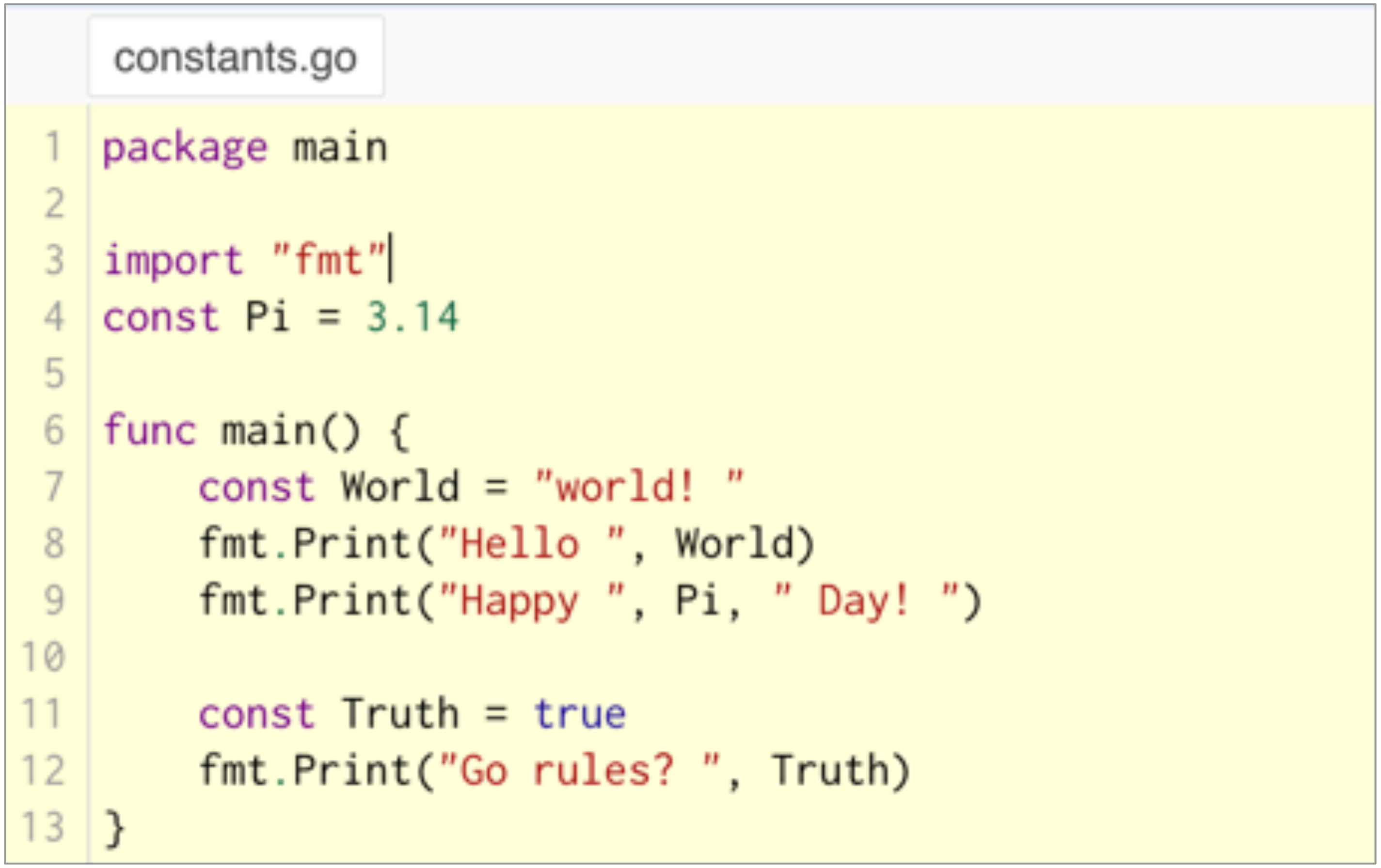

variables-with-initializers.go

```go
package main

import "fmt"

var i, j int = 1, 2

func main() {
    var c, python, java = true, false, "no!"
    fmt.Println(i, j, c, python, java)
}
```

Constants

- Constants are declared like variables but with keyword **`const`**
- Can not use the syntx **`:=`**

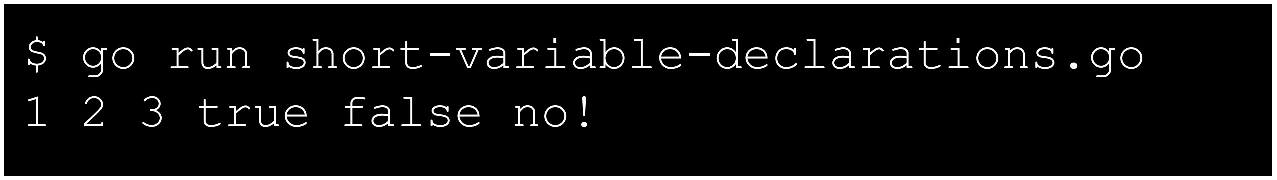
```
$ go run constants.go
Hello world! Happy 3.14 Day! Go rules?
```


constants.go

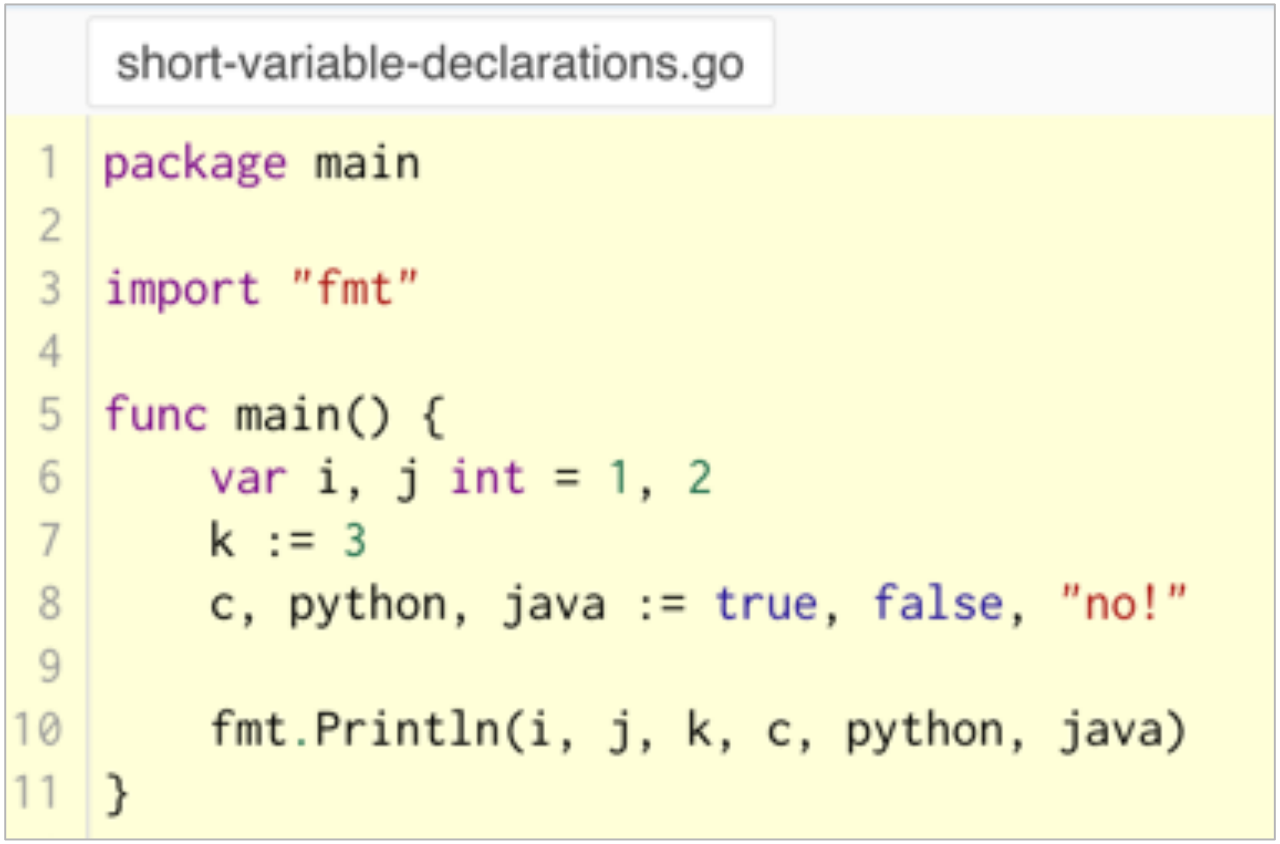
```go
package main

import "fmt"
const Pi = 3.14

func main() {
	const World = "world! "
	fmt.Print("Hello ", World)
	fmt.Print("Happy ", Pi, " Day! ")

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

Short variables declarations

- Inside a function, the short attribution instruction **`:=`** can be used instead of a **`var`** declaration



```
$ go run short-variable-declarations.go
1 2 3 true false no!
```



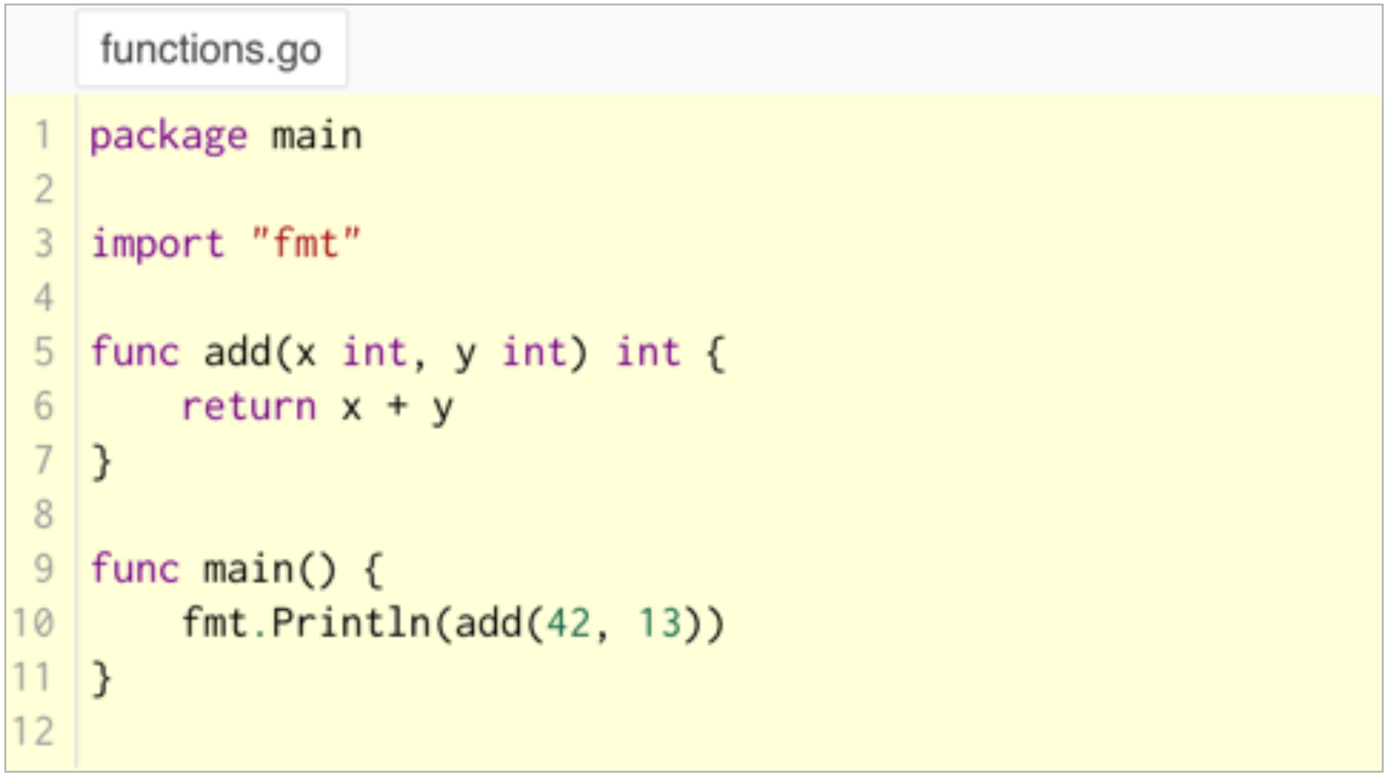
short-variable-declarations.go

```go
package main

import "fmt"

func main() {
	var i, j int = 1, 2
	k := 3
	c, python, java := true, false, "no!"

	fmt.Println(i, j, k, c, python, java)
}
```

Functions

- Functions could have zero or more arguments
- Notice that the type comes after the parameter name, like variables

```
$ go run functions.go
55
```



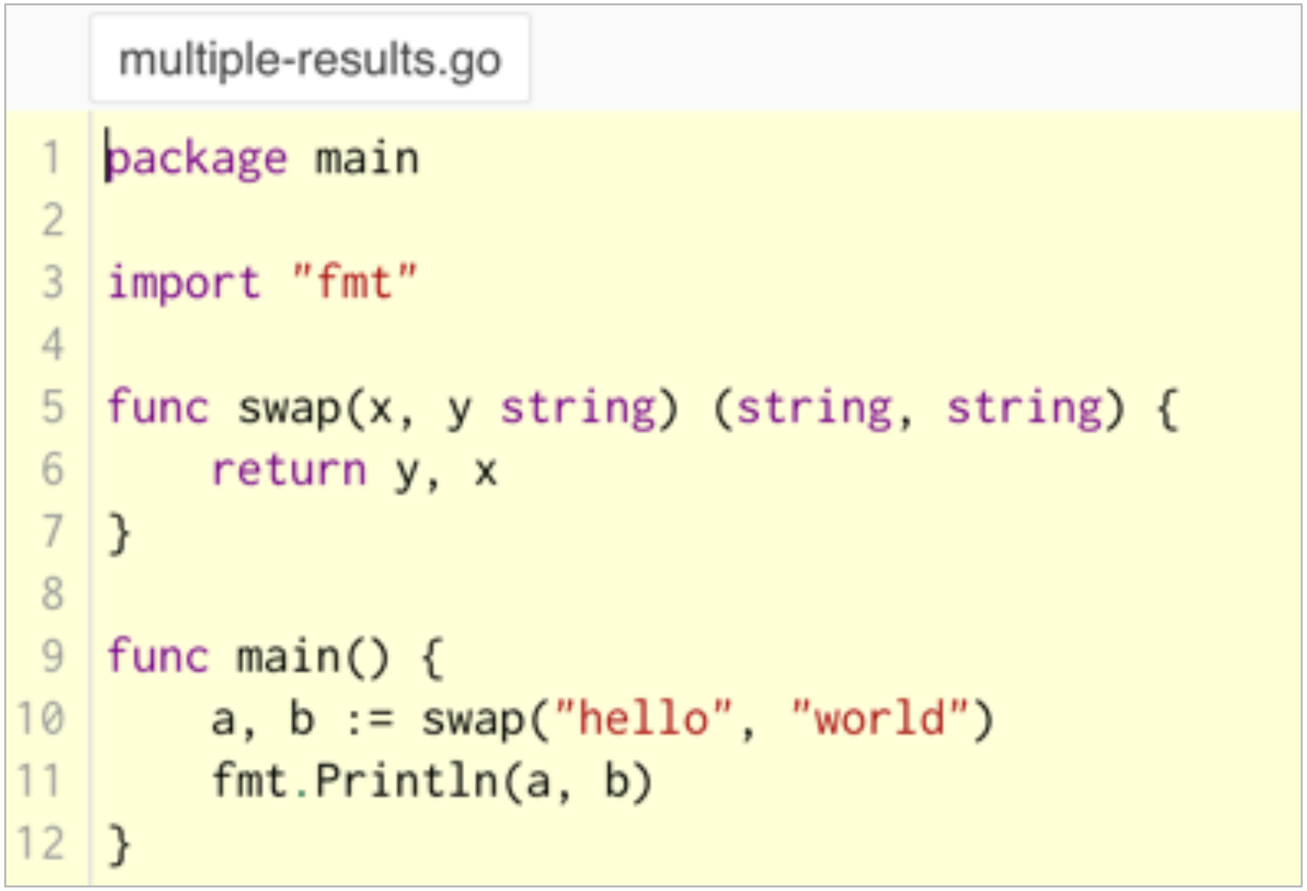
functions.go

```go
package main

import "fmt"

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

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

```

Multiple return values

- A function can have multiple return values

```
$ go run multiple-results.go
world hello
```



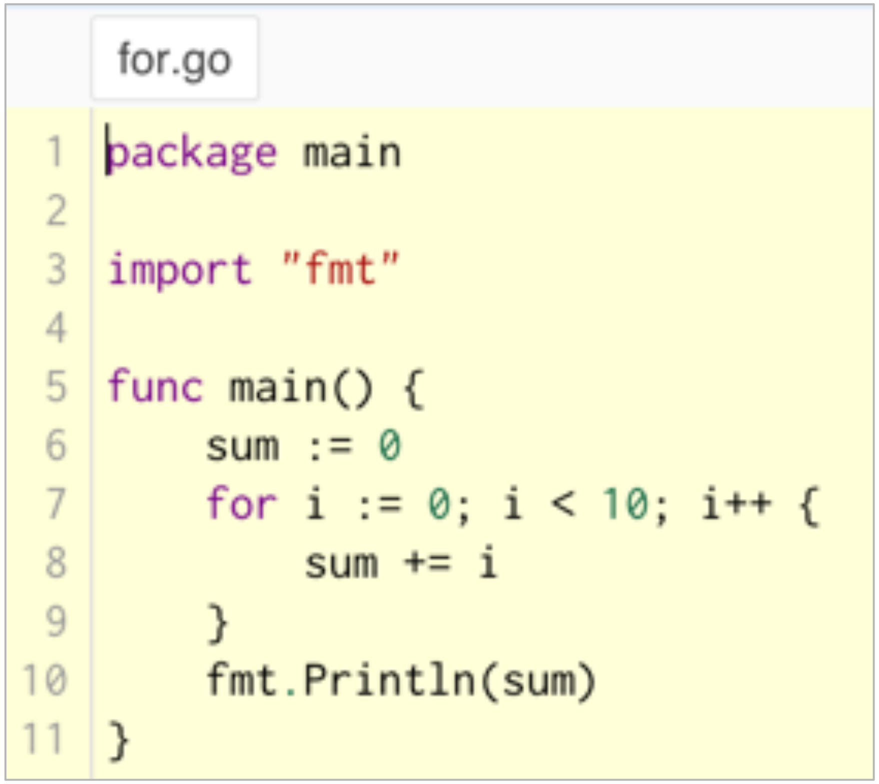
```go
package main

import "fmt"

func swap(x, y string) (string, string) {
	return y, x
}

func main() {
	a, b := swap("hello", "world")
	fmt.Println(a, b)
}
```

Looping For

- Go has just **`for`** as looping structure
- It is very similar with C or Java code, except for **`( )`**
- Start and end declarations can be empty

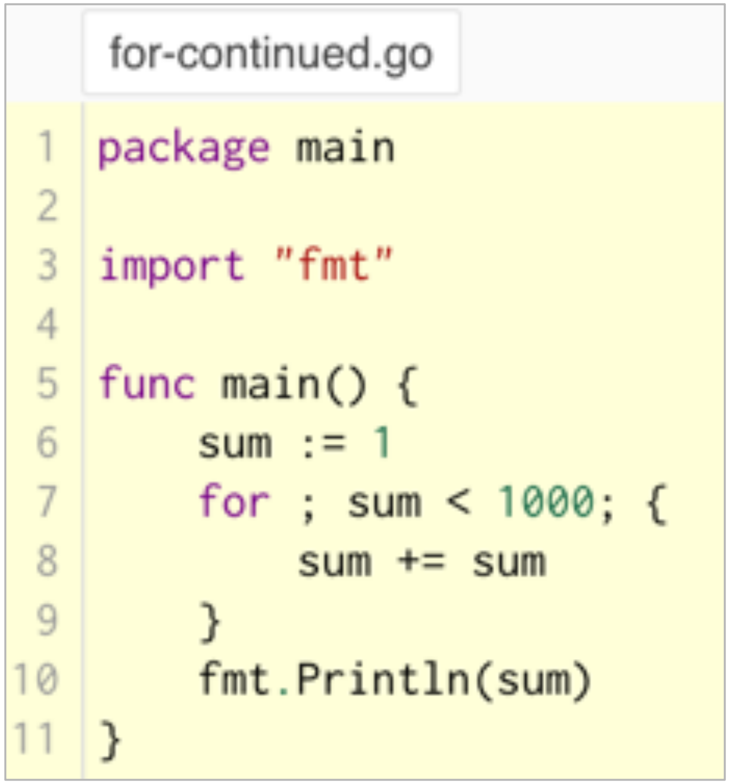
```
$ go run for.go
45
```

```
$ go run for-continu
1024
```



for.go

```go
package main

import "fmt"

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



for-continued.go

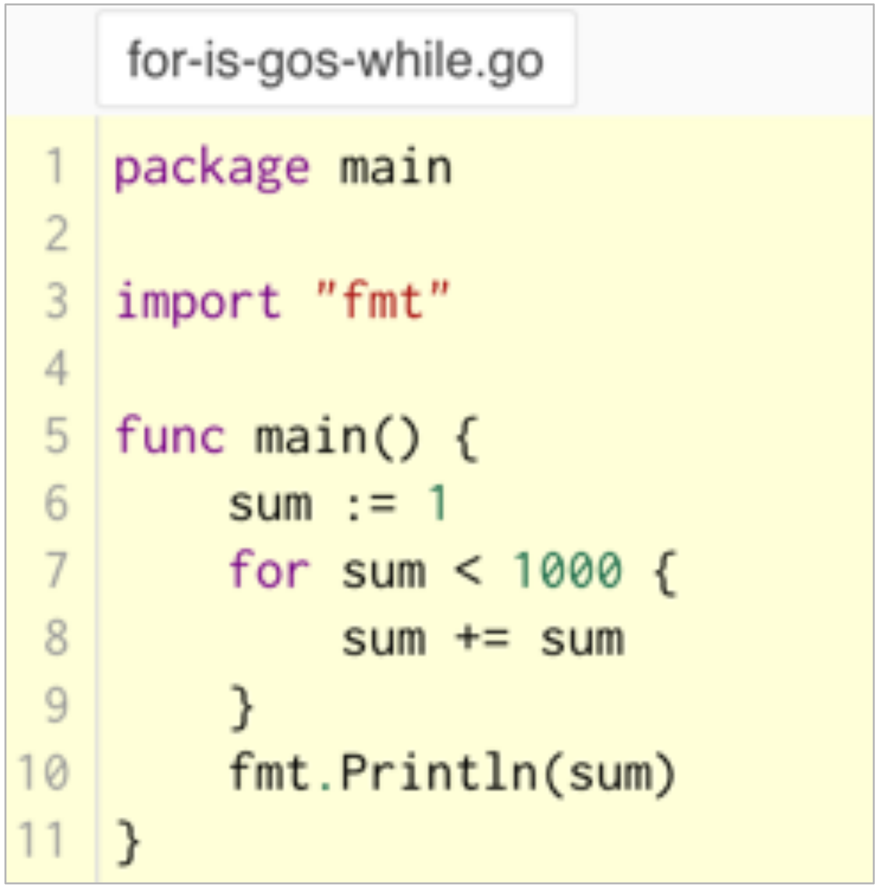
```go
package main

import "fmt"

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

Looping "while" and forever

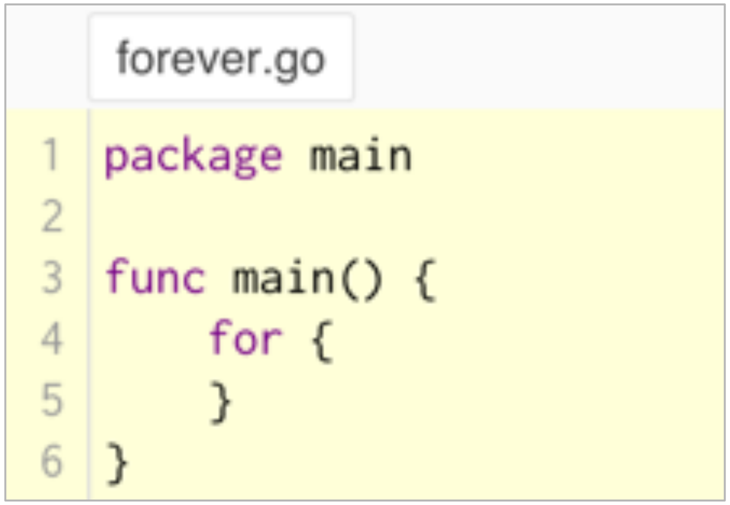
- Semicolon can be removed and you will have **`while`**
- **`for`** can run forever

```
$ go run for-is-go-while.go
1024
```

```
$ go run forever.go
process took too long
```



```go
package main

import "fmt"

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

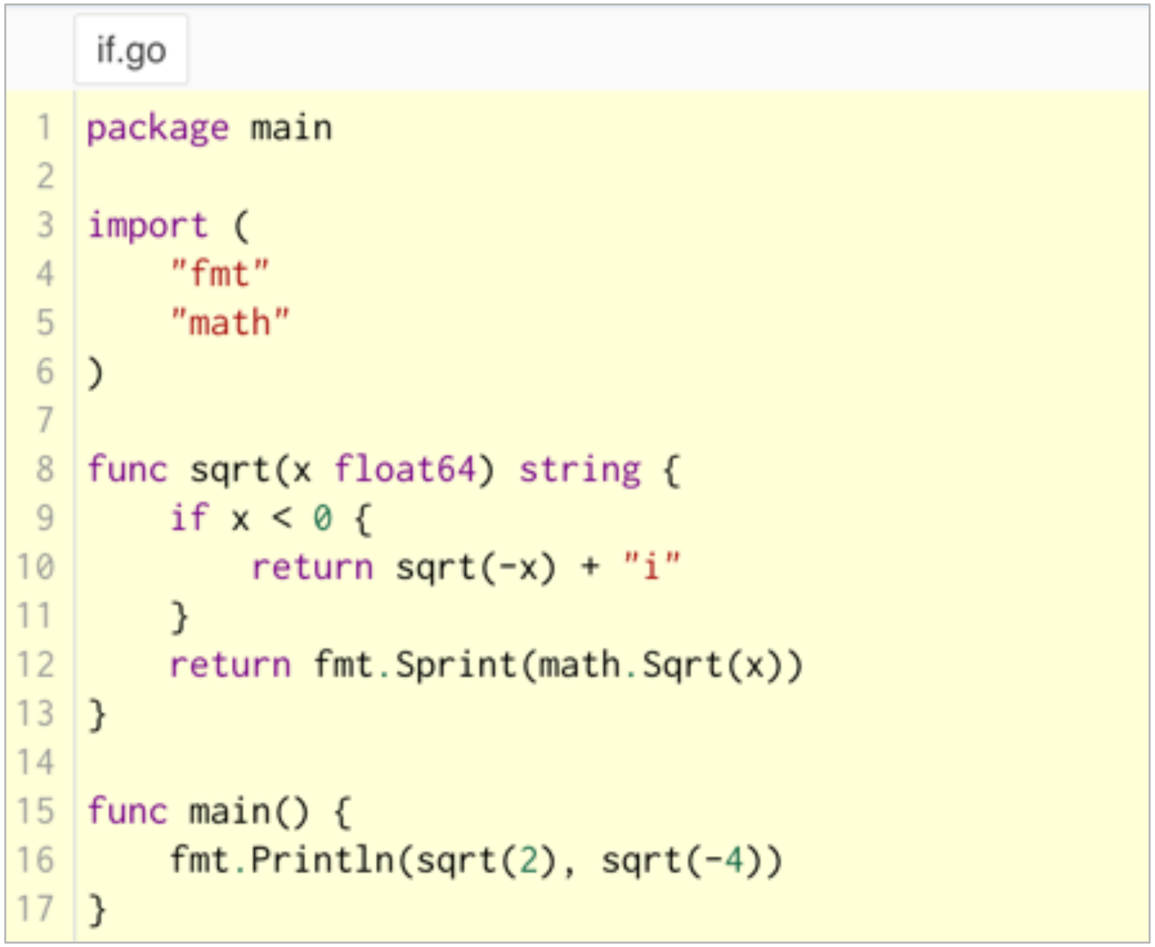


```go
package main

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

if Condition

- It is very similar with C or Java code, except for **( )**

```
$ go run if.go
1.4142135623730951 2i
```

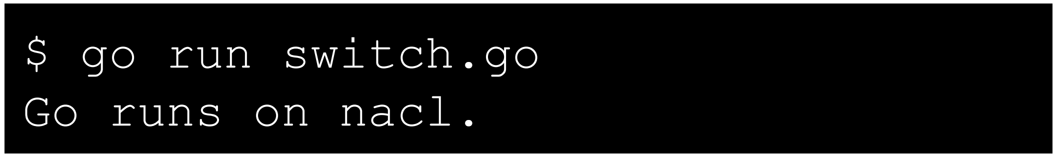


if.go

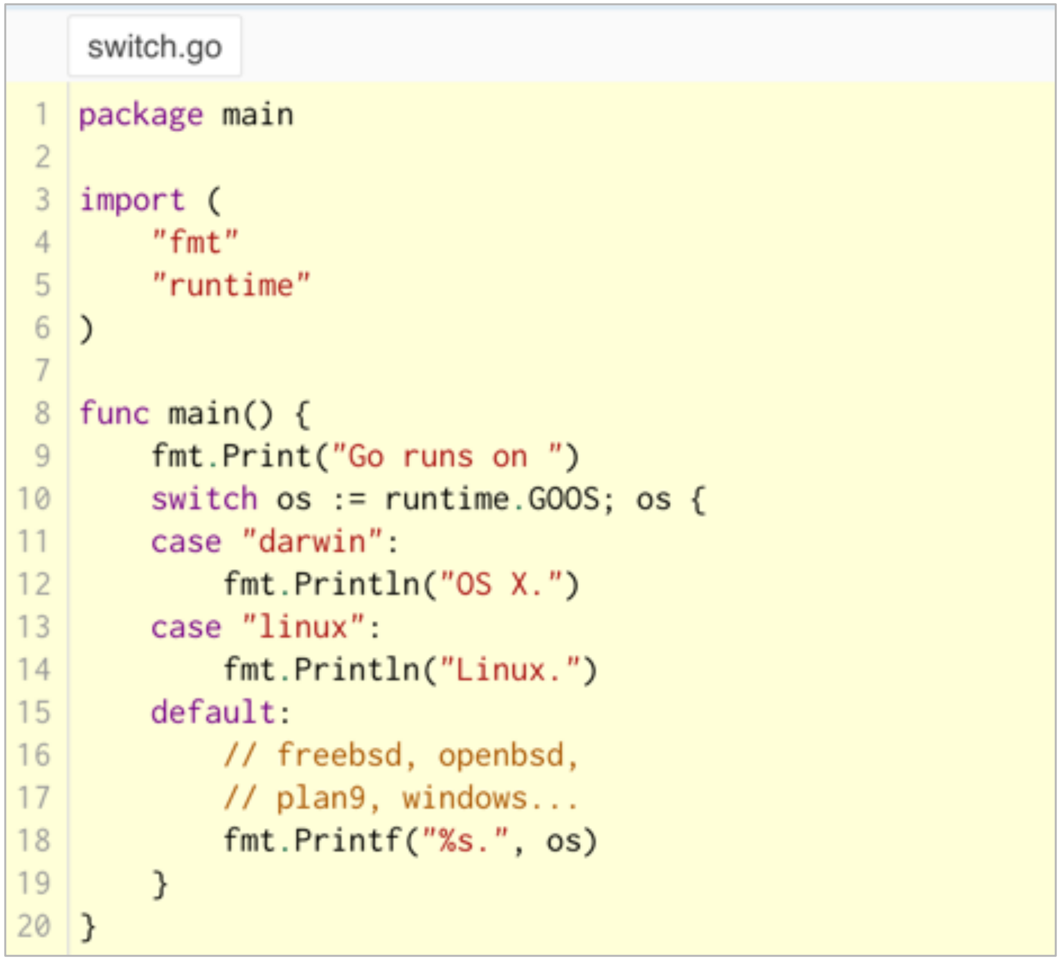
```go
package main

import (
	"fmt"
	"math"
)

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

func main() {
	fmt.Println(sqrt(2), sqrt(-4))
}
```

Switch Condition

- It is very similar with C or Java code, except for ( )



```
$ go run switch.go
Go runs on nacl.
```



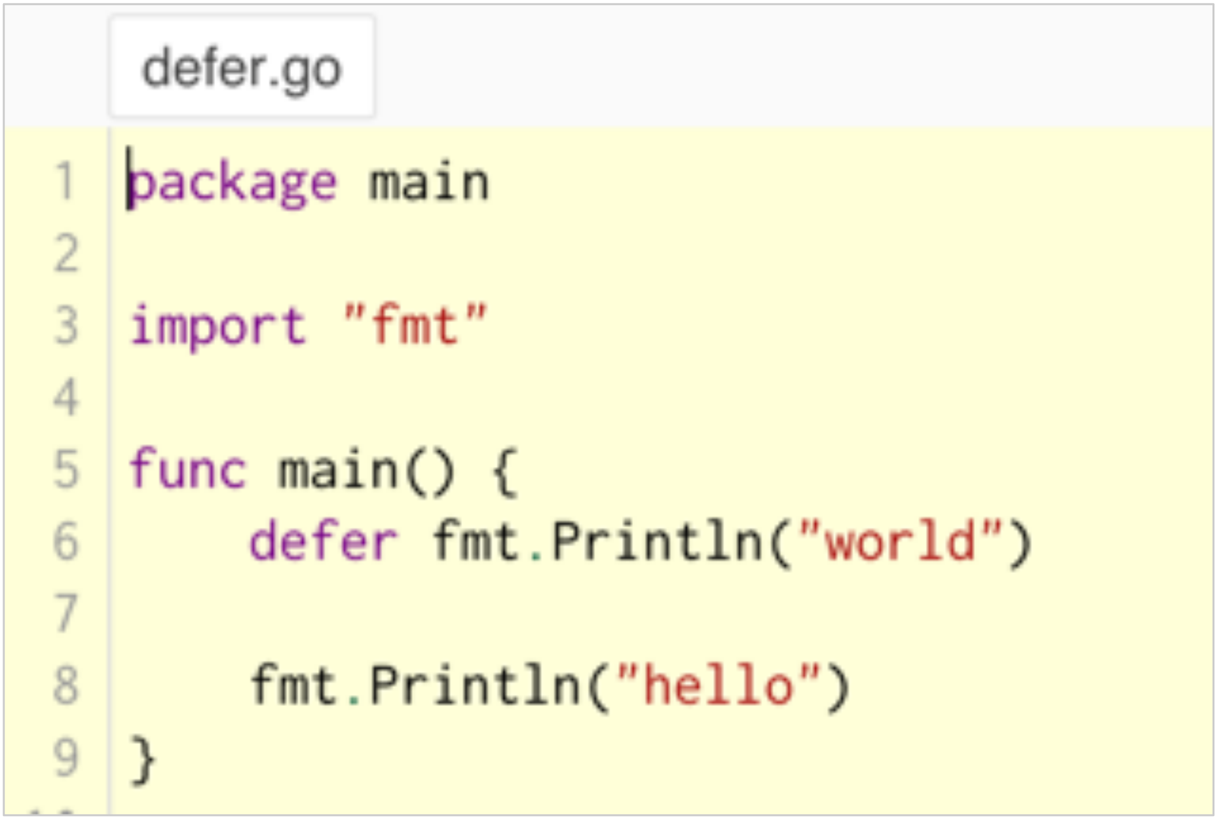
switch.go

```go
package main

import (
	"fmt"
	"runtime"
)

func main() {
	fmt.Print("Go runs on ")
	switch os := runtime.GOOS; os {
	case "darwin":
		fmt.Println("OS X.")
	case "linux":
		fmt.Println("Linux.")
	default:
		// freebsd, openbsd,
		// plan9, windows...
		fmt.Printf("%s.", os)
	}
}
```

Defer

- Postponing the execution of a function until the function returns
- The arguments of the deferred calls are evaluated immediately

```
$ go run defer.go
hello world
```



```go
package main

import "fmt"

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

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

What more?

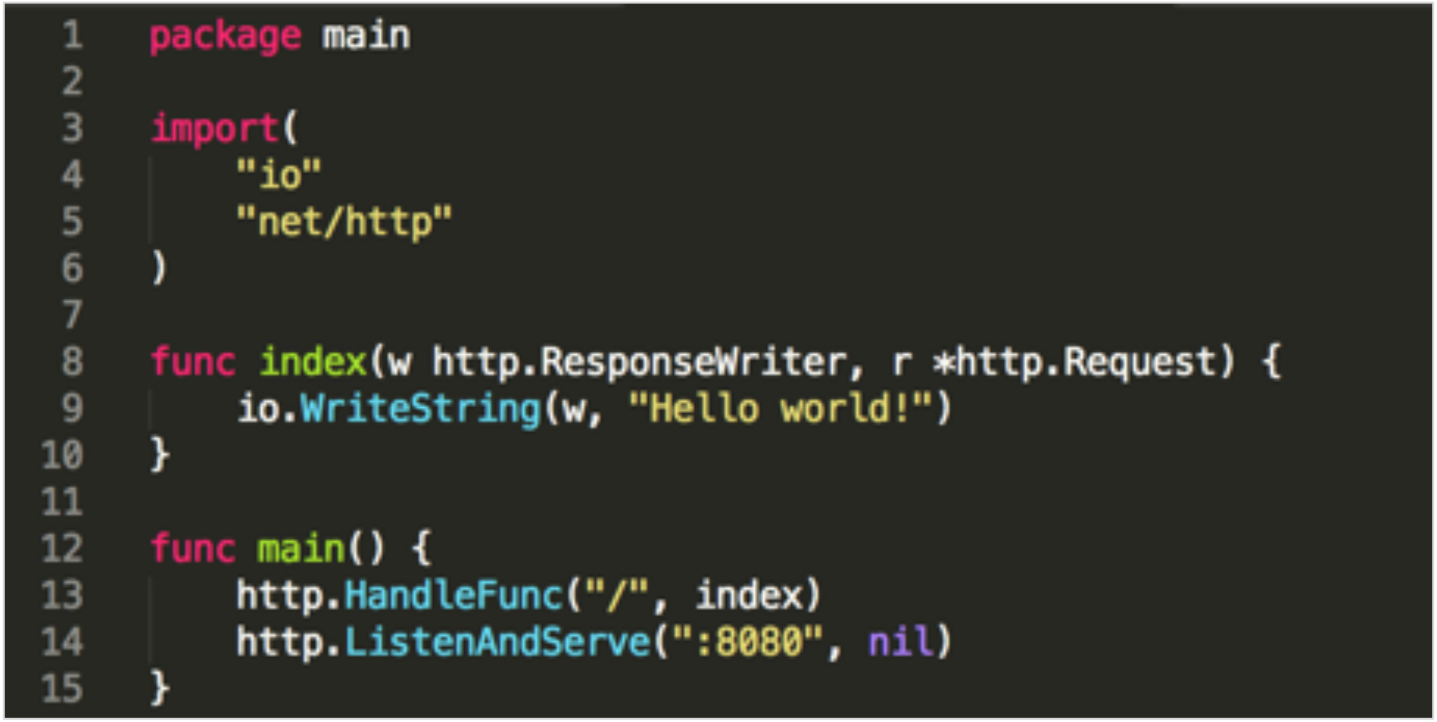
- Pointer
- Struct
- Matrix
- Slice
- Range
- Map
- Value function
- Closures
- Method
- Interface
- Stringer
- Error
- and a lot of more!!!

http://go-tour-br.appspot.com

A web server

- It is just simple to build a web server with 15 lines or less!!
  Could you belive that???

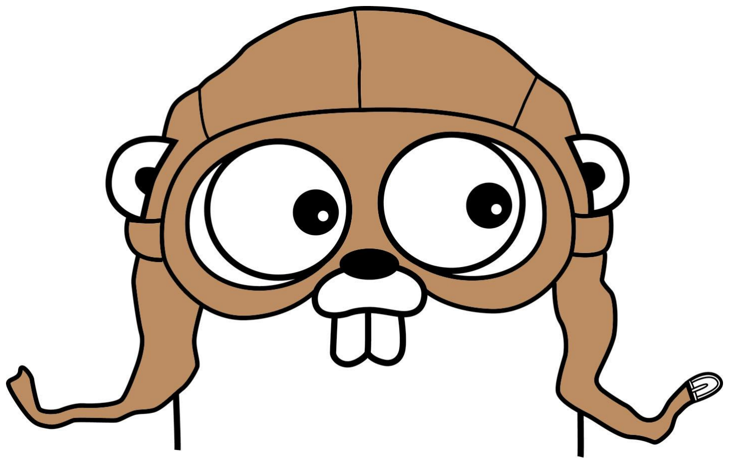
```
$ go run http.go
```



```go
package main

import(
	"io"
	"net/http"
)

func index(w http.ResponseWriter, r *http.Request) {
	io.WriteString(w, "Hello world!")
}

func main() {
	http.HandleFunc("/", index)
	http.ListenAndServe(":8080", nil)
}
```

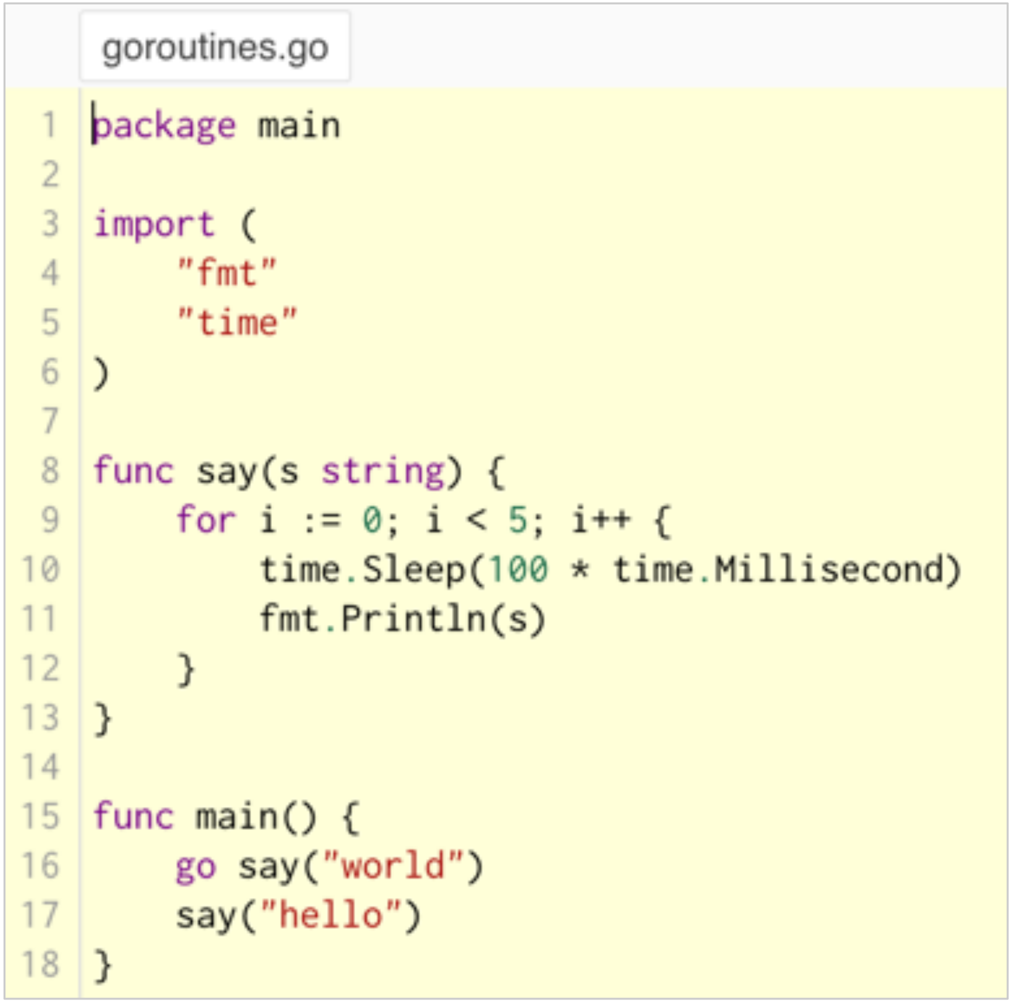
Concurrency (goroutines)

- To execute a goroutine, just **go**!
- To send or receive information between the goroutines, use **channels**
- Use the **GOMAXPROCS** environment variable to define the amount of threads



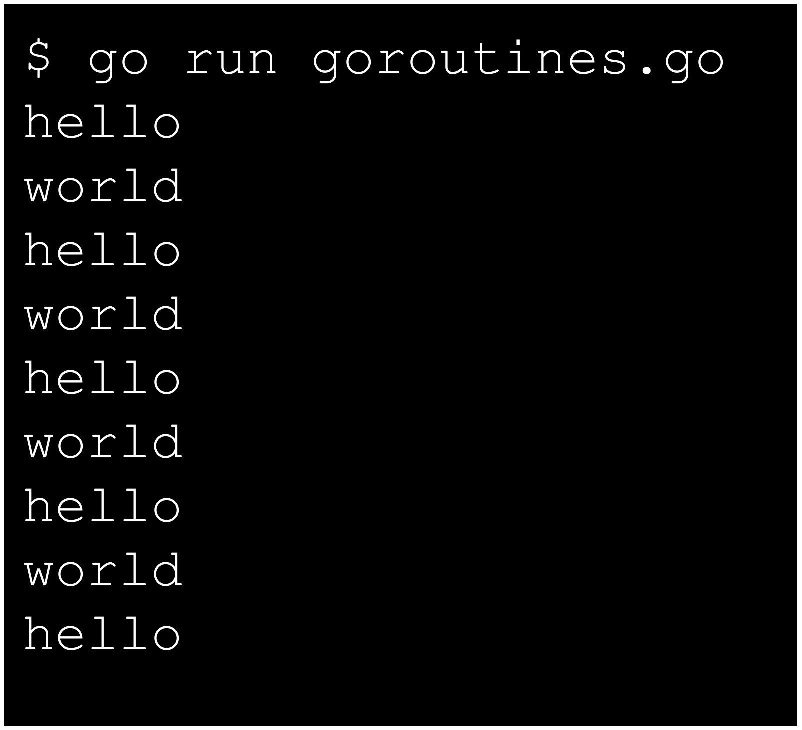
Goroutines

- A goroutine is a lightweight thread managed by Go runtime



goroutines.go

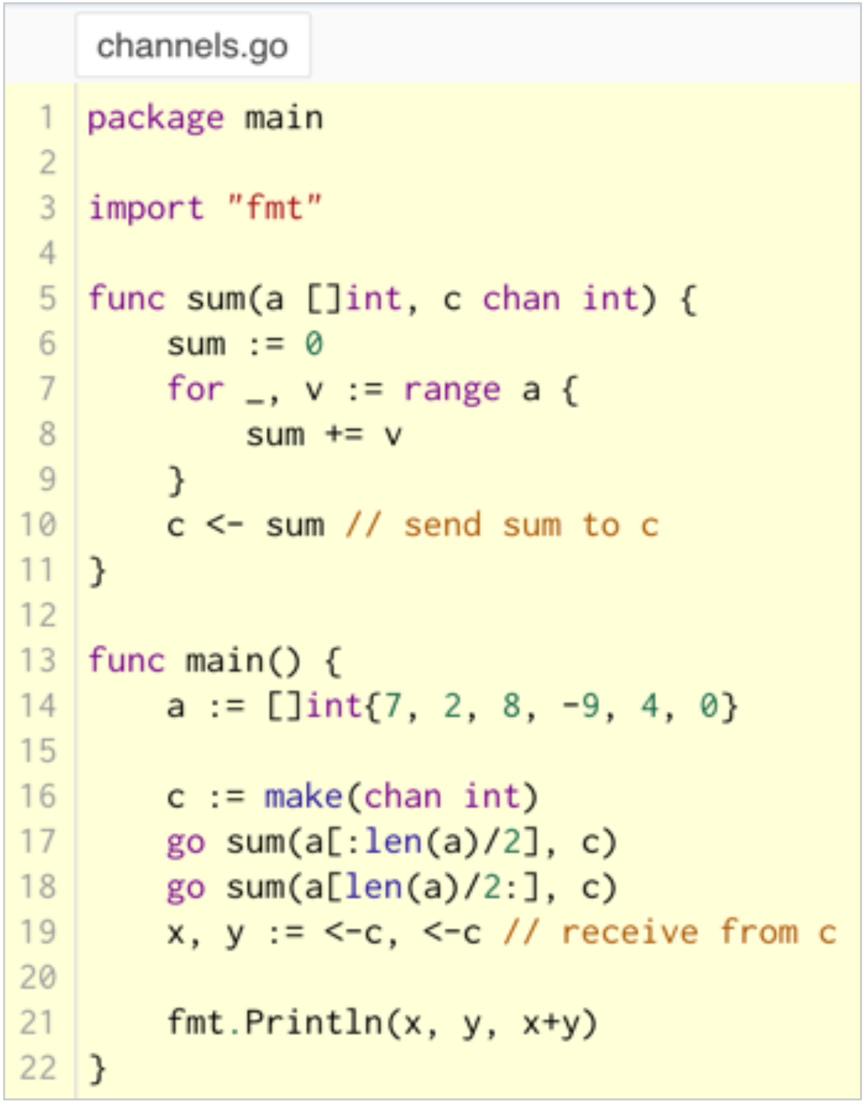
```go
package main

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")
}
```



```
$ go run goroutines.go
hello
world
hello
world
hello
world
hello
world
hello
```

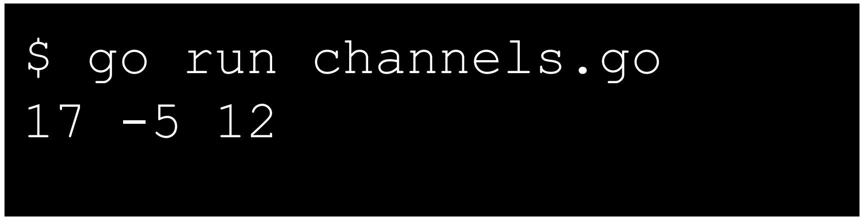
Channels

- Channels are typed's conduit through which you can send and receive values with the channel operator **<-**



channels.go

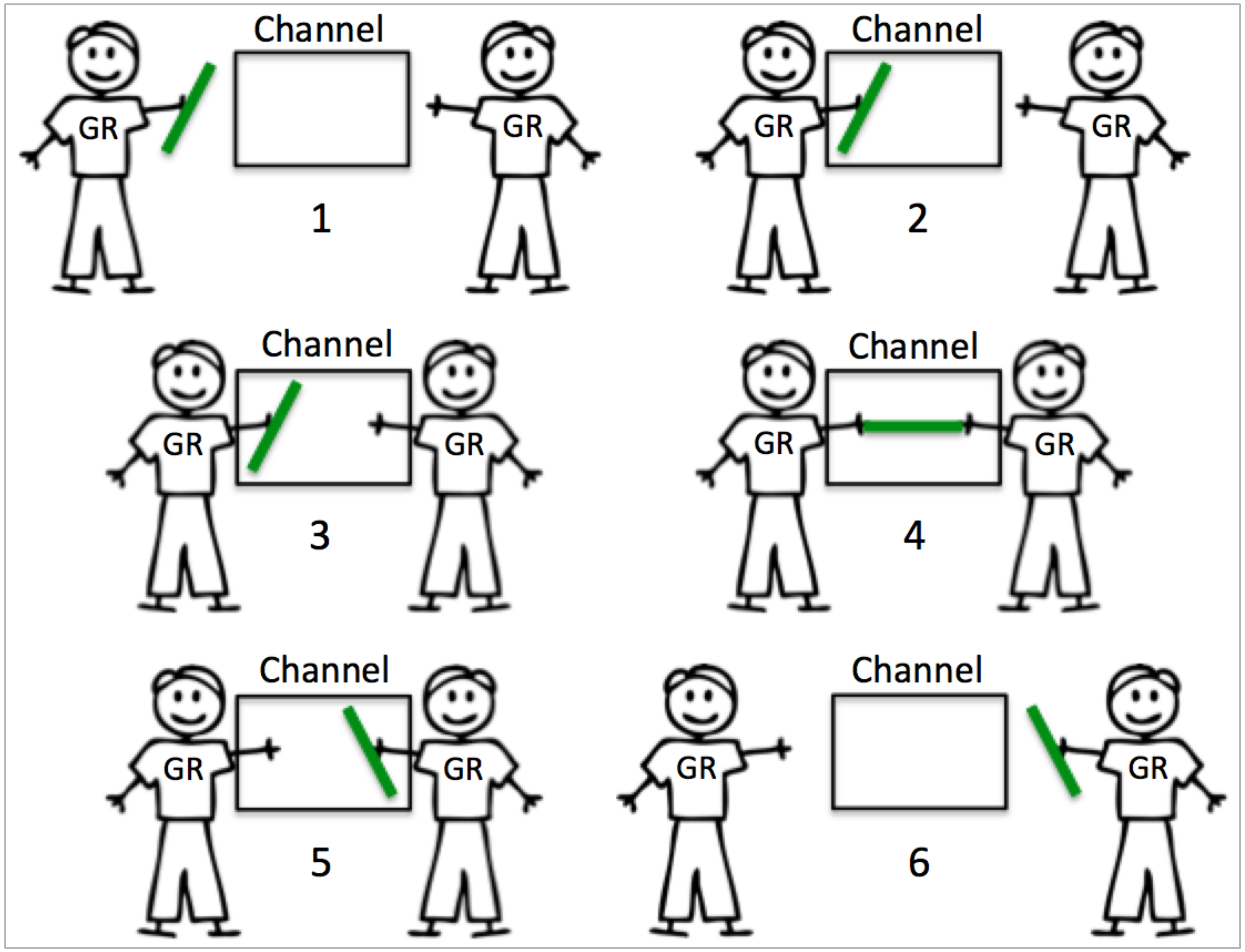
```go
package main

import "fmt"

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

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

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

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



```
$ go run channels.go
17 -5 12
```

Unbuffered Channels

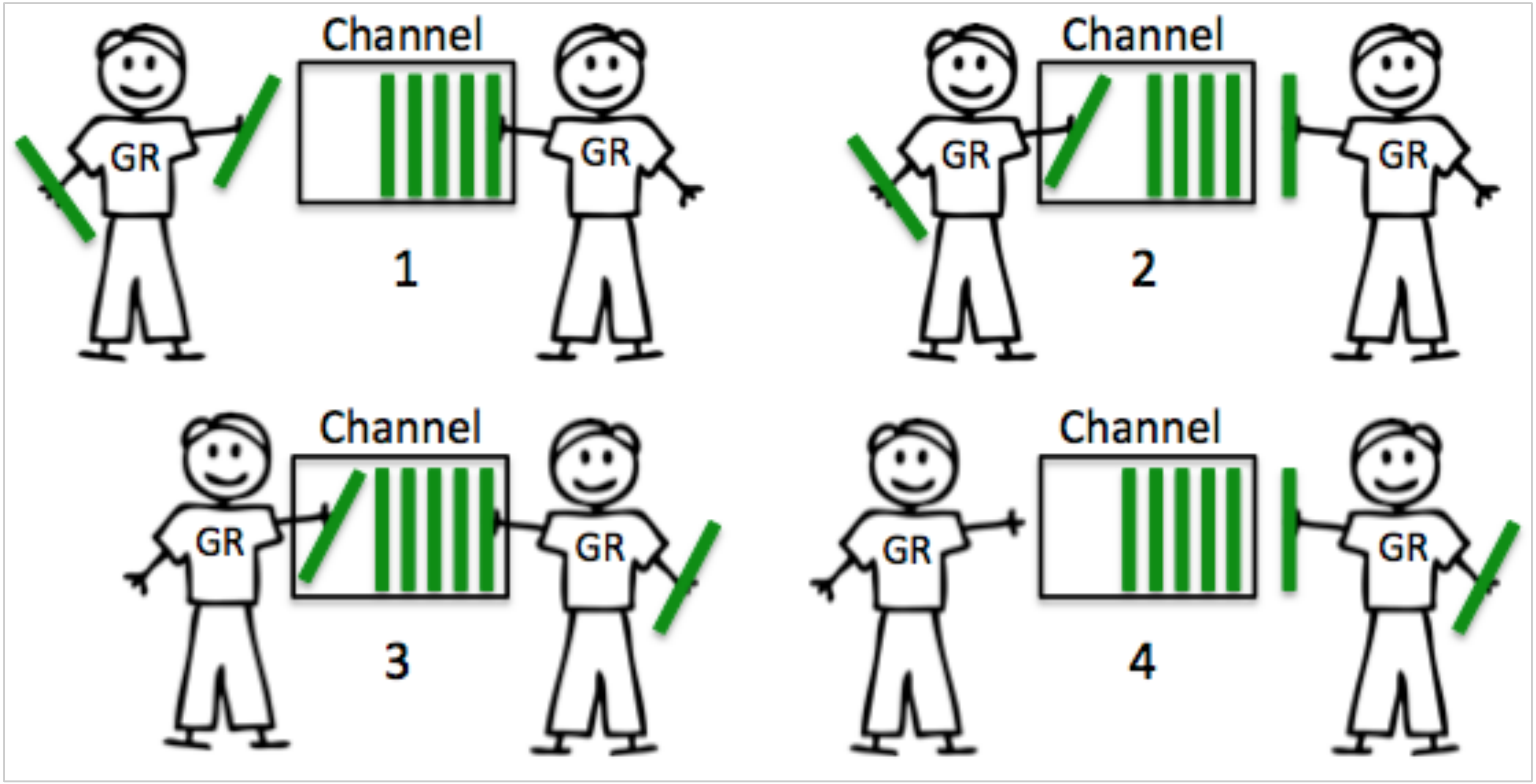
```
c := make (chan int)
```



http://www.goinggo.net/2014/02/the-nature-of-channels-in-go.html

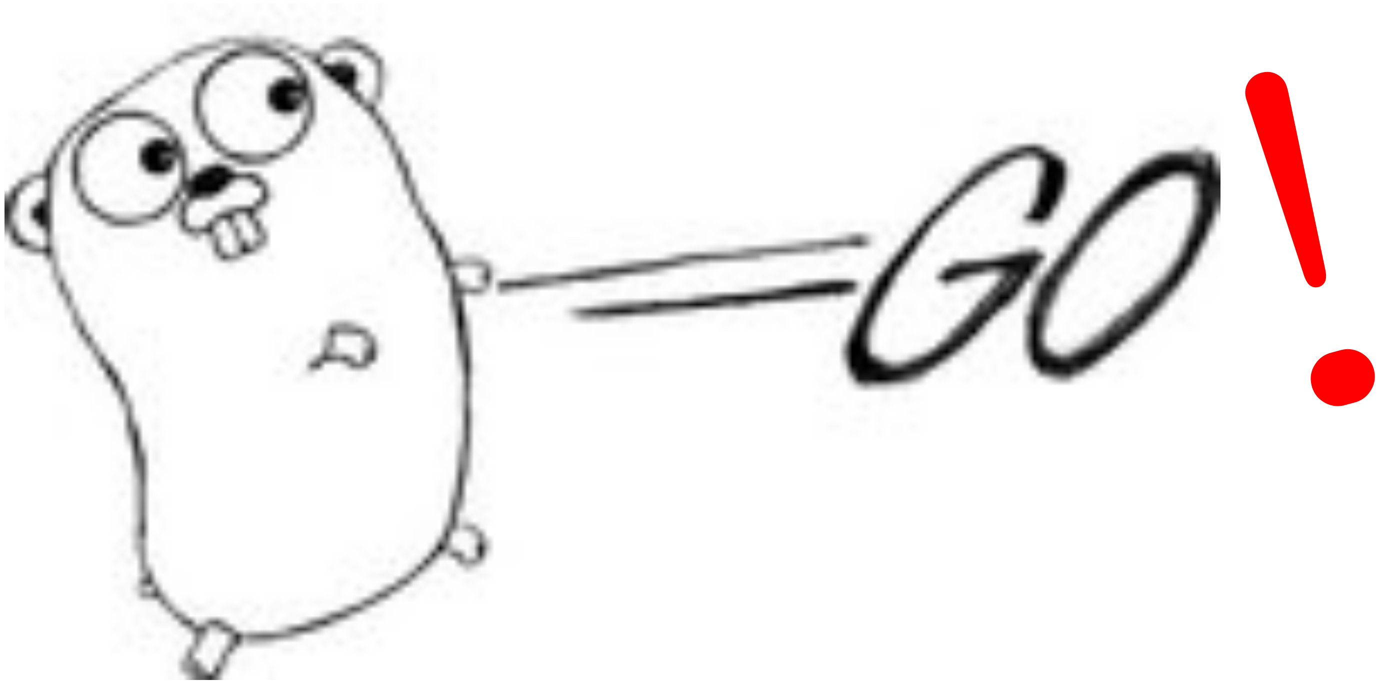
Buffered Channels

```
c := make (chan int, 10)
```



http://www.goinggo.net/2014/02/the-nature-of-channels-in-go.html

Now you are ready to



Bibliografia

- http://golang.org
- http://go-tour-br.appspot.com/
- https://tour.golang.org
- http://www.golangbr.org/
- https://vimeo.com/49718712
- http://gophercon.com
- http://www.infoq.com/br/news/2014/09/go-1-3
- http://www.casadocodigo.com.br/products/livro-google-go
- https://pt.wikipedia.org/wiki/Inferno_(sistema_operacional)
- http://www.grokpodcast.com/series/a-linguagem-go/
- https://pt.wikipedia.org/wiki/Go_(linguagem_de_programação)
- https://gobyexample.com
- http://www.goinggo.net/2014/02/the-nature-of-channels-in-go.html
- http://www.goinggo.net/2013/09/detecting-race-conditions-with-go.html?m=1
- https://en.wikipedia.org/wiki/Green_threads
- http://www.toptal.com/go/go-programming-a-step-by-step-introductory-tutorial

Questions?