Tel Aviv, Feb 11 2019

Go Secrets & Gotchas

Yarel Maman

HERE mobility

@YarelMam

Hello, 世界

Welcome to a tour of the Go programming language.

The tour is divided into a list of modules that you can access by clicking on A Tour of Go on the top left of the page.

You can also view the table of contents at any time by clicking on the menu on the top right of the page.

Throughout the tour you will find a series of slides and exercises for you to complete.

You can navigate through them using

"previous" or `PageUp` to go to the previous page,

"next" or `PageDown` to go to the next page.

The tour is interactive. Click the Run button now (or press `Shift` + `Enter`) to compile and run the program on a remote server. The result is displayed below the code.

These example programs demonstrate different aspects of Go. The programs in the tour are meant to be starting points for your own experimentation.

Edit the program and run it again.

When you click on Format (shortcut: `Ctrl` + `Enter`), the text in the editor is formatted using the gofmt tool. You can switch syntax highlighting on and off by clicking on the syntax button.

When you're ready to move on, click the right arrow below or type the `PageDown` key.

hello.go

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello, 世界")
7 }
8
```

Effective Go

Go is Simple

Secrets & Gotchas

Language

“Everything in Go is passed by value”

Go FAQ

Passing a slice

```
func appendToSlice(someSlice []int) {
   someSlice = append(someSlice, 2)
}


func main() {
   someSlice := make([]int, 0)
   fmt.Printf("Slice before %v\n", someSlice)
   appendToSlice(someSlice)
   fmt.Printf("Slice after %v\n", someSlice)

}
```

Passing a slice

```
func appendToSlice(someSlice []int) {
   someSlice = append(someSlice, 2)
}


func main() {
   someSlice := make([]int, 0)
   fmt.Printf("Slice before %v\n", someSlice)
   appendToSlice(someSlice)
   fmt.Printf("Slice after %v\n", someSlice)

}
```

Output:

```
Slice before []
Slice after []
```

Passing a map

```
func setInMap(someMap map[int]int) {
    someMap[0] = 0
}

func main() {
    someMap := make(map[int]int)
    fmt.Printf("Map before %v\n", someMap)
    setInMap(someMap)
    fmt.Printf("Map after %v\n", someMap)
}
```

Passing a map

```
func setInMap(someMap map[int]int) {
    someMap[0] = 0
}

func main() {
    someMap := make(map[int]int)
    fmt.Printf("Map before %v\n", someMap)
    setInMap(someMap)
    fmt.Printf("Map after %v\n", someMap)
}
```

Output:

```
Map before map[]
Map after map[0:0]
```

Slices

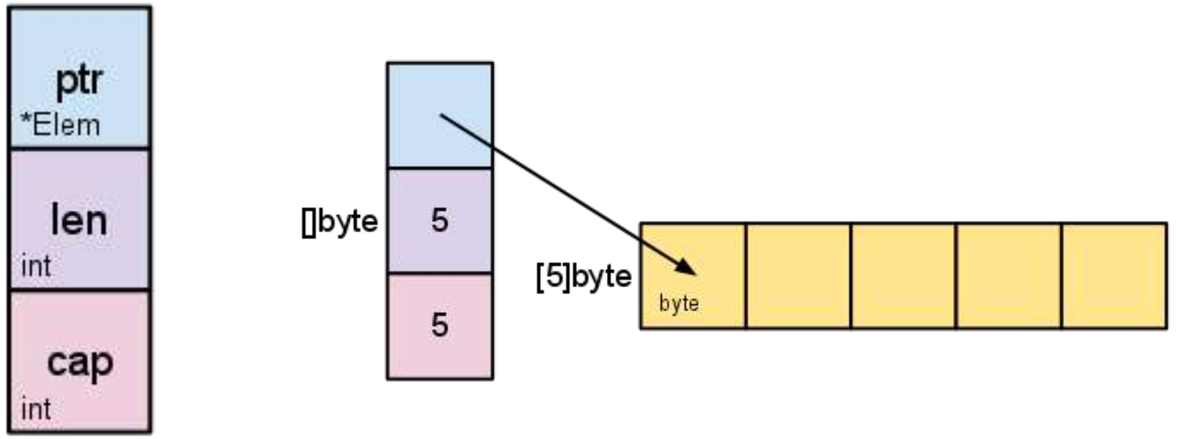
Credit: Louise Lyshøj

Slices in a nutshell

- A slice is a struct

(runtime/slice.go)

```
type slice struct {
    array unsafe.Pointer
    len   int
    cap   int
}
```



Taken from https://blog.golang.org/go-slices-usage-and-internals

Maps in a nutshell

- A map is a pointer to a struct (hmap)

(runtime/map.go)

```
func makemap(t *maptype, hint int, h *hmap) *hmap {
    ..
}
```

Going back to our example

- appendToSlice - gets a copy of a struct

```
func appendToSlice(someSlice []int) {
    someSlice = append(someSlice, 2)
}
```

Going back to our example

- appendToSlice - gets a copy of a struct

```
func appendToSlice(someSlice []int) {
    someSlice = append(someSlice, 2)
}
```

- setInMap - gets a copy of a pointer (to an hmap struct)

```
func setInMap(someMap map[int]int) {
    someMap[0] = 0
}
```

```
type item struct {
    id int
}
```

```
type item struct {
   id int
}
```

```
m := make(map[int]item, 1)
…
m[0].id = 1
```

```
type item struct {
    id int
}
```

```
m := make(map[int]item, 1)
…
m[0].id = 1
```

Compile error: cannot assign to struct field m[0].id in map

```
type item struct {
    id int
}
```

- Map values are not addressable

```
m := make(map[int]item, 1)
…
m[0].id = 1
```

Compile error: cannot assign to struct field m[0].id in map

```
type item struct {
    id int
}
```

- Map values are not addressable

```
m := make(map[int]item, 1)
…
m[0].id = 1
```

Compile error: cannot assign to struct field m[0].id in map

- Slice values are addressable

```
s := make([]item, 1)
…
s[0].id = 1
```

```
type item struct {
    id int
}
```

- Map values are not addressable

```
m := make(map[int]item, 1)
…
m[0].id = 1
```

Compile error: cannot assign to struct field m[0].id in map

- Slice values are addressable

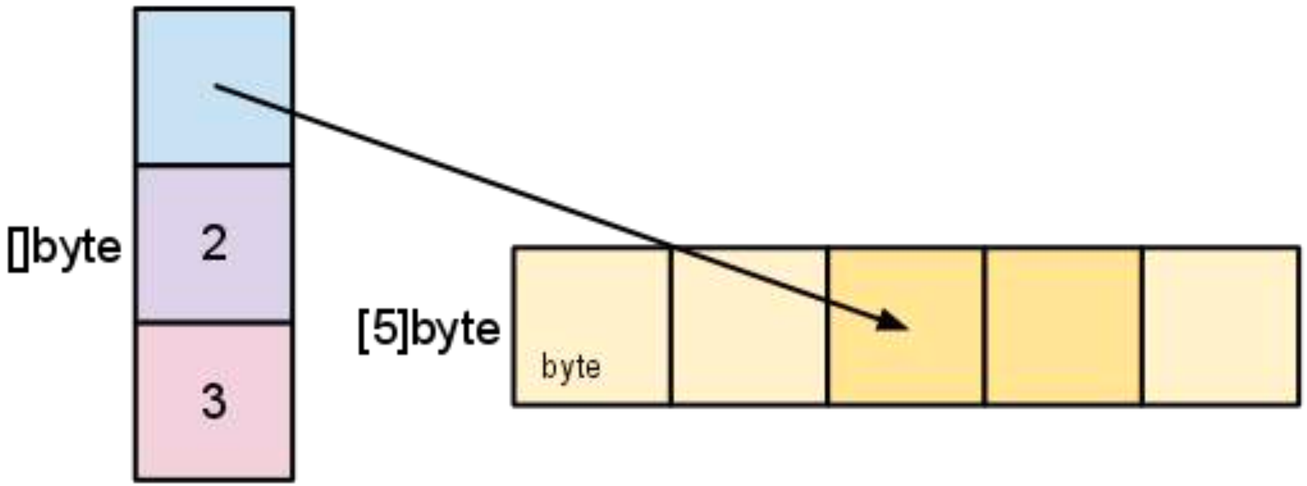
```
s := make([]item, 1)
…
s[0].id = 1
```

Open **proposal: spec: cannot assign to a field of a map element directly: m["foo"].f = x** #3117

griesemer opened this Issue on Feb 23, 2012 · 30 comments

Slicing a slice

"Slicing does not copy the slice's data. It creates a new slice that points to the original array" (Slice internals)




```
s := make([]byte, 5)
s := s[2:4]
```

“memory leaks” when slicing

Example taken from https://blog.golang.org/go-slices-usage-and-internals

“memory leaks” when slicing

```
var digitRegexp = regexp.MustCompile("[0-9]+")

func FindDigits(filename string) []byte {
     b, _ := ioutil.ReadFile(filename)
     return digitRegexp.Find(b)
}
```

Example taken from https://blog.golang.org/go-slices-usage-and-internals

“memory leaks” when slicing

Example file contents:
00aaaaaaaaaaaa …… (File size – 3.7MB)

```
var digitRegexp = regexp.MustCompile("[0-9]+")

func FindDigits(filename string) []byte {
    b, _ := ioutil.ReadFile(filename)
    return digitRegexp.Find(b)
}
```

Example taken from https://blog.golang.org/go-slices-usage-and-internals

“memory leaks” when slicing

Example file contents:
00aaaaaaaaaaaa …… (File size – 3.7MB)

```
var digitRegexp = regexp.MustCompile("[0-9]+")

func FindDigits(filename string) []byte {
    b, _ := ioutil.ReadFile(filename)
    return digitRegexp.Find(b)
}
```

Reads an entire file contents to a byte slice

Slice header of b:

Data = 0xc000090120

Len = 3756660

Cap = 3757172

Example taken from https://blog.golang.org/go-slices-usage-and-internals

“memory leaks” when slicing

Example file contents:
00aaaaaaaaaaaa …… (File size – 3.7MB)

```
var digitRegexp = regexp.MustCompile("[0-9]+")

func FindDigits(filename string) []byte {
    b, _ := ioutil.ReadFile(filename)
    return digitRegexp.Find(b)
}
```

Reads an entire file contents to a byte slice

Slice header of b:

Data = 0xc000090120

Len = 3756660

Cap = 3757172

Returns the first group of the regex match by **slicing b**.

Returned slice header:

Data = 0xc000090120

Len = 2

Cap = 3757172

Example taken from https://blog.golang.org/go-slices-usage-and-internals

“memory leaks” when slicing

Example file contents:
00aaaaaaaaaaaa …… (File size – 3.7MB)

```
var digitRegexp = regexp.MustCompile("[0-9]+")

func FindDigits(filename string) []byte {
    b, _ := ioutil.ReadFile(filename)
    return digitRegexp.Find(b)
}
```

Reads an entire file contents to a byte slice

Slice header of b:

Data = 0xc000090120
Len = 3756660
Cap = 3757172

Returns the first group of the regex match by **slicing b**.

Returned slice header:

Data = 0xc000090120
Len = 2
Cap = 3757172

Possible solution:

Example taken from https://blog.golang.org/go-slices-usage-and-internals

“memory leaks” when slicing

Example file contents:
00aaaaaaaaaaaa …… (File size – 3.7MB)

```
var digitRegexp = regexp.MustCompile("[0-9]+")

func FindDigits(filename string) []byte {
    b, _ := ioutil.ReadFile(filename)
    return digitRegexp.Find(b)
}
```

Reads an entire file contents to a byte slice

Slice header of b:

Data = 0xc000090120
Len = 3756660
Cap = 3757172

Returns the first group of the regex match by **slicing b**.

Returned slice header:

Data = 0xc000090120
Len = 2
Cap = 3757172

Possible solution:

```
func CopyDigits(filename string) []byte {
    b, _ := ioutil.ReadFile(filename)
    b = digitRegexp.Find(b)
    c := make([]byte, len(b))
    copy(c, b)
    return c
}
```

Example taken from https://blog.golang.org/go-slices-usage-and-internals

“memory leaks” when slicing

Example file contents:
00aaaaaaaaaaaa …… (File size – 3.7MB)

```
var digitRegexp = regexp.MustCompile("[0-9]+")

func FindDigits(filename string) []byte {
    b, _ := ioutil.ReadFile(filename)
    return digitRegexp.Find(b)
}
```

Reads an entire file contents to a byte slice

Slice header of b:

Data = 0xc000090120
Len = 3756660
Cap = 3757172

Returns the first group of the regex match by **slicing b**.

Returned slice header:

Data = 0xc000090120
Len = 2
Cap = 3757172

Possible solution:

```
func CopyDigits(filename string) []byte {
    b, _ := ioutil.ReadFile(filename)
    b = digitRegexp.Find(b)
    c := make([]byte, len(b))
    copy(c, b)
    return c
}
```

Making a copy of the sliced buffer, and returning it.

Returned Slice header:

Data = 0xc0004420b0
Len = 2
Cap = 2

Example taken from https://blog.golang.org/go-slices-usage-and-internals

Interfaces and nil values

```
var counter *int
fmt.Println(counter == nil)

var metric interface{}
fmt.Println(metric == nil)

metric = counter
fmt.Println(metric == nil)
```

Interfaces and nil values

```
var counter *int
fmt.Println(counter == nil)

var metric interface{}
fmt.Println(metric == nil)

metric = counter
fmt.Println(metric == nil)
```

Output:

Interfaces and nil values

```
var counter *int
fmt.Println(counter == nil)

var metric interface{}
fmt.Println(metric == nil)

metric = counter
fmt.Println(metric == nil)
```

Output:

```
true
```

Interfaces and nil values

```
var counter *int
fmt.Println(counter == nil)

var metric interface{}
fmt.Println(metric == nil)

metric = counter
fmt.Println(metric == nil)
```

Output:

```
true
true
```

Interfaces and nil values

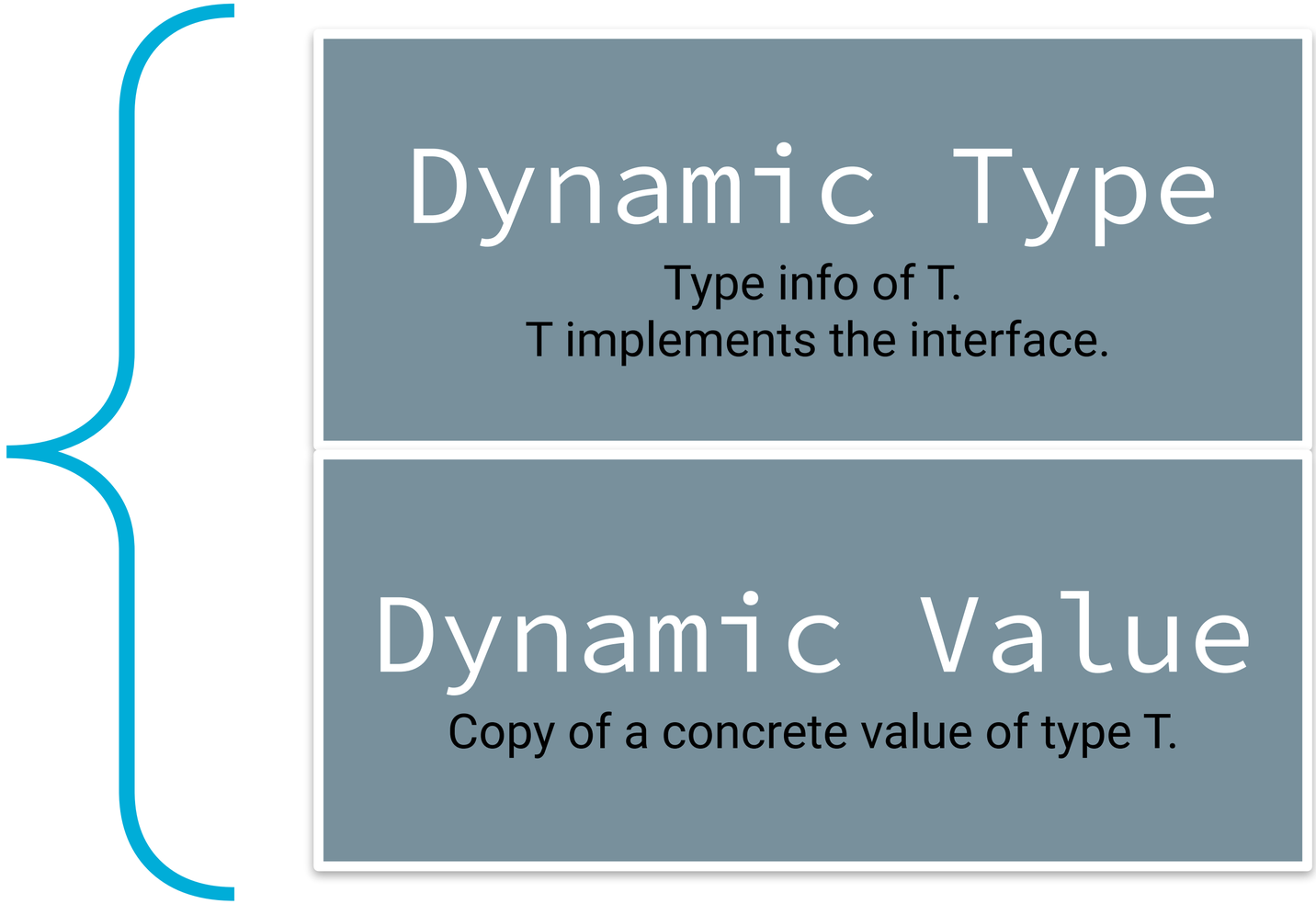
```
var counter *int
fmt.Println(counter == nil)

var metric interface{}
fmt.Println(metric == nil)

metric = counter
fmt.Println(metric == nil)
```

Output:

```
true
true
false
```

Go interface values in a nutshell

Interface value



Interfaces and nil values

```
var counter *int

var metric interface{}

fmt.Printf("metric before: V=%v T=%T\n",
  metric, metric)

metric = counter

fmt.Printf("metric after: V=%v T=%T\n",
  metric, metric)
```

Interfaces and nil values

```
var counter *int

var metric interface{}

fmt.Printf("metric before: V=%v T=%T\n",
  metric, metric)

metric = counter

fmt.Printf("metric after: V=%v T=%T\n",
  metric, metric)
```

Output:

```
metric before: V=<nil> T=<nil>
metric after: V=<nil> T=*int
```

“An interface value is nil only if the V and T are both unset.”

Go FAQ

Interfaces and nil values

```
func returnsError() error {
   var p *MyError = nil
   if bad() {
      p = ErrBad
   }

   return p // Will always return a non-nil error.
}
```

Example taken from https://golang.org/doc/faq#nil_error

Interfaces and nil values

- A possible solution

```
func returnsError() error {
  var err error
  if bad() {
    err = ErrBad
  }

  return err
}
```

Interfaces and nil values

- A possible solution

```
func returnsError() error {
  var err error
  if bad() {
    err = ErrBad
  }

  return err
}
```

For more, watch “Understanding nil”

Using goroutines on loop iterator variables

```
values := []int{1, 2, 3, 4}

for _, val := range values {
  go func() {
    fmt.Printf("val %v, addr %v\n",
      val, &val)
  }()
}
```

Using goroutines on loop iterator variables

```
values := []int{1, 2, 3, 4}

for _, val := range values {
  go func() {
    fmt.Printf("val %v, addr %v\n",
      val, &val)
  }()
}
```

Output:

```
val 4, addr 0xc000014078
val 4, addr 0xc000014078
val 4, addr 0xc000014078
val 4, addr 0xc000014078
```

Using goroutines on loop iterator variables

```
values := []int{1, 2, 3, 4}

for _, val := range values {
  go func(val int) {
    fmt.Printf("val %v, addr %v\n",
      val, &val)
  }(val)
}
```

Using goroutines on loop iterator variables

```
values := []int{1, 2, 3, 4}

for _, val := range values {
  go func(val int) {
    fmt.Printf("val %v, addr %v\n",
      val, &val)
  }(val)
}
```

Output:

```
val 1, addr 0xc000014088
val 3, addr 0xc00009e000
val 2, addr 0xc000086000
val 4, addr 0xc000014078
```

Odd solution

```
values := []int{1, 2, 3, 4}

for _, val := range values {
 val := val

 go func() {
  fmt.Println(val)
 }()
}
```

Credit: Ewa Gillen

Odd solution

```
values := []int{1, 2, 3, 4}

for _, val := range values {
 val := val

 go func() {
  fmt.Println(val)
 }()
}
```

Credit: Ewa Gillen

Open **proposal: spec: redefine range loop variables in each iteration** #20733

halleknast opened this Issue on Jun 19, 2017 · 24 comments

Using goroutines on loop iterator variables

```
$ go vet -rangeloops goroutine.go

./goroutine.go:14: loop variable val captured
by func literal
```

Defer in a loop

```
for fileName := range fileNameC {
    f, err := os.Open(fileName)
    if err != nil { /* ERROR HANDLING */ }
    defer f.Close()
    …
}
```

Standard libs

Copying sync primitives

Copying sync primitives

- Go sync primitives must not be copied

sync/mutex.go

```
// A Mutex is a mutual exclusion lock.
// The zero value for a Mutex is an unlocked mutex.
// A Mutex must not be copied after first use.
type Mutex struct {
    state int32
    sema  uint32
}
```

Copying sync primitives

- Go sync primitives must not be copied

sync/mutex.go

```
// A Mutex is a mutual exclusion lock.
// The zero value for a Mutex is an unlocked mutex.
// A Mutex must not be copied after first use.
type Mutex struct {
   state int32
   sema  uint32
}
```

- Types containing sync primitives values must not be copied

```
// This type must not be copied
type Value struct {
   sync.RWMutex
   …
}
```

Copying sync primitives

```
type pond struct {
    sync.Mutex
    ducks map[string]duck
}

func (p pond) addDuck(duck duck) {
    p.Lock()
    p.ducks[duck.name] = duck
    p.Unlock()
}
```

Copying sync primitives

```
type pond struct {
    sync.Mutex
    ducks map[string]duck
}

func (p pond) addDuck(duck duck) {
    p.Lock()
    p.ducks[duck.name] = duck
    p.Unlock()
}
```

Value receiver

Copying sync primitives

```
type pond struct {
    sync.Mutex
    ducks map[string]duck
}

func (p pond) addDuck(duck duck) {
    p.Lock()
    p.ducks[duck.name] = duck
    p.Unlock()
}
```

Value receiver

See

“CodeReviewComments”

For guidelines

Use “go vet” to easily detect it

```
$ go vet -copylocks mutex.go

mutex.go:24: addDuck passes lock by value:
sync_copies.pond
```

Using maps concurrently

- Go's builtin maps are not safe for concurrent write, or concurrent read and write

```
go func() {
   fmt.Println(m[0]) // DATA RACE!
}()

m[0] = 0 // DATA RACE!
```

Using maps concurrently

- Go's builtin maps are not safe for concurrent write, or concurrent read and write

```
go func() {
   fmt.Println(m[0]) // DATA RACE!
}()

m[0] = 0 // DATA RACE!
```

- concurrent read (only) is safe

```
m[0] = 0
…
go fmt.Println(m[0])
go fmt.Println(m[0])
```

The runtime may panic!

```
fatal error: concurrent map writes

goroutine 1194 [running]:
runtime.throw(0x1fe22bf, 0x15)
 /usr/local/go/src/runtime/panic.go:596
+0x95 fp=0xc420ca8dc8 sp=0xc420ca8da8
runtime.mapassign(0x1e03b40, 0xc420bbb890,
0xc420ca8ea0, 0x1b)
 /usr/local/go/src/runtime/hashmap.go:499
+0x667 fp=0xc420ca8e68 sp=0xc420ca8dc8
```

Sync package

```
type Cond
    func NewCond(l Locker) *Cond
    func (c *Cond) Broadcast()
    func (c *Cond) Signal()
    func (c *Cond) Wait()
type Locker
type Map
    func (m *Map) Delete(key interface{})
    func (m *Map) Load(key interface{}) (value interface{}, ok bool)
    func (m *Map) LoadOrStore(key, value interface{}) (actual interface{}, loaded bool)
    func (m *Map) Range(f func(key, value interface{}) bool)
    func (m *Map) Store(key, value interface{})
type Mutex
    func (m *Mutex) Lock()
    func (m *Mutex) Unlock()
type Once
    func (o *Once) Do(f func())
type Pool
    func (p *Pool) Get() interface{}
    func (p *Pool) Put(x interface{})
type RWMutex
    func (rw *RWMutex) Lock()
    func (rw *RWMutex) RLock()
    func (rw *RWMutex) RLocker() Locker
    func (rw *RWMutex) RUnlock()
    func (rw *RWMutex) Unlock()
type WaitGroup
    func (wg *WaitGroup) Add(delta int)
    func (wg *WaitGroup) Done()
    func (wg *WaitGroup) Wait()
```

sync.Map

type **Map**

Map is like a Go map[interface{}]interface{} but is safe for concurrent use by multiple goroutines without additional locking or coordination. Loads, stores, and deletes run in amortized constant time.

Go Data Race Detector

$ go run -race example.go

```
go func() {
   fmt.Println(m[0])
}()

m[0] = 0
```

```
==================
WARNING: DATA RACE
Read at 0x00c0000ac048 by goroutine 6:
  main.main.func1()
     gosecrets/maps/example.go:9 +0x64

Previous write at 0x00c0000ac048 by main goroutine:
  main.main()
     gosecrets/maps/example.go:12 +0x91

Goroutine 6 (running) created at:
  main.main()
     gosecrets/maps/example.go:8 +0x59
==================
```

Closing words

Credit: Ben White

READ THE SOURCE

- Package Docs
- The go spec
- The go source code

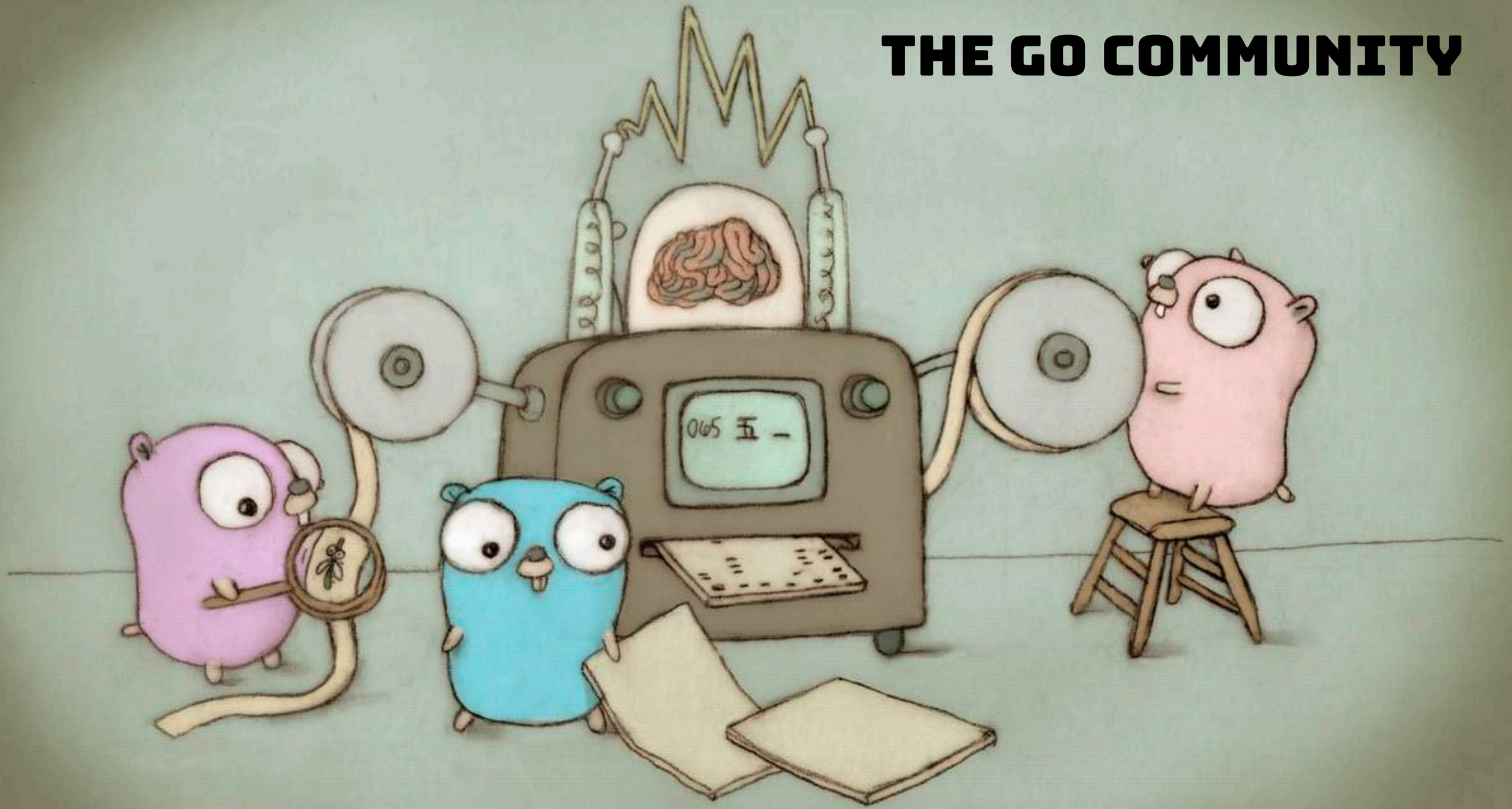
EXPERIMENT

Credit: Arif Riyanto

DON'T KEEP IT A SECRET

Credit: Ben White

THE GO COMMUNITY



Credit: Renee French

Community

- Keeping up to date with the language development
- Reading Go Github issues
- Reading the change log of Go releases
- Subscribing to newsletters such as: Golang Weekly
- Golang-nuts (Google groups)
- r/golang (Reddit)

FIN.
Thanks!

Twitter: @YarelMam
Email: yarel.maman@gmail.com

Slides by: https://blog.golang.org/go-brand