GoLang Introduction

Marcin Kaciuba Mateusz Greń

dreamlab

Agenda

1. Basics knowledge
2. Pros & cons
3. Basic overview
4. Concurrency
5. Tooling
6. Real world project walkthrough
7. Quiz
8. Pizza
9. DIY simple HTTP server & crawler

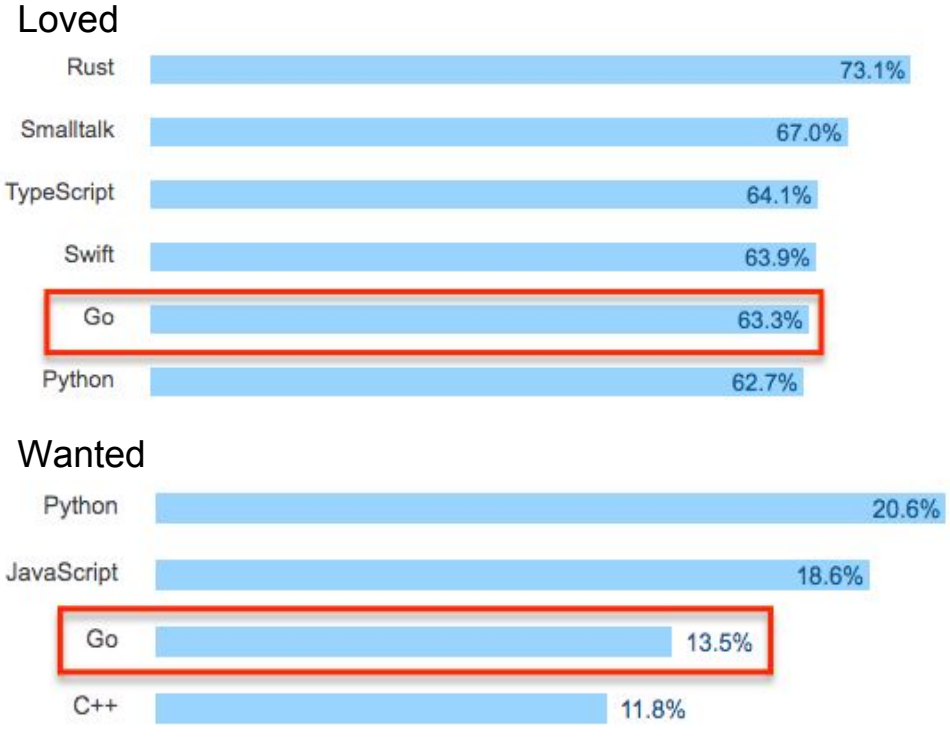
1. Basic knownloadge

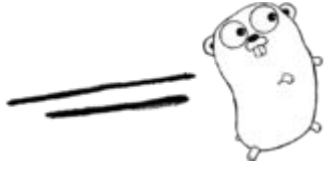
First appeared November 10, 2009; 8 years ago

Last stable release February 16, 2018;

Projects using Go

- Docker
- Prometheus
- Kubernetes
- Consul
- InfluxDB
- ...





2. Pros & Cons

- Easy to learn
- Compiled - fast
- Easy concurrency
- Big and growing community
- Statically typed
- Native binaries
- Garbage collector
- Big standard library
- Very strict compiler

- No generics
- No stable dependency management yet
- JSON encoding/decoding
- Garbage collector
- Use only standard library!
- Very strict compiler

3. Basic overview

Variables

```
// declaration
var myVar string

// assignment
myVar = "Jak leci?"

// declaration & assignment
myOtherVar := "A spoko"

// declaration & assignment of slice
myTab := []string{"jak", "spoko", "to", "spoko"}

// declaration of multiple variables
var nextTab, n = []string{"hej"}, 19

// constants
const n = 100000000
const d = 2e10 / n

fmt.Println(int64(d))
fmt.Println(math.Sin(d))
```

3. Basic overview

Basic types

```
// Strings and chars
stringVar := "String"
charVar := 'a'

// Integers
var a int // also int8 int16 int32 int64
var b uint // also uint8 uint16 uint32 uint64
simpleInt := 5

// floats
var c float32 // also float64
simpleFloat := 5.5

// special
var d byte // alias for uint8
var e rune // alias for int32
var f complex128 // also complex64 part of math/cmplx
```

Is it possible to declare multiple types of variables in single declaration in Go?

```
var a, b, c = 3, 4, "foo"
```

a. Yes
b. No

What is the result of the following code?

```
package main

import "fmt"

func main() {
    var a int8 = 3
    var b int16 = 4

    sum := a + b

    fmt.Println(sum)
}
```

a. 7
b. 4
c. 3
d. Invalid operation

3. Basic overview

Advanced types - arrays

```
// Declare an array
var a[5] int

// Set
a[4] = 100

// Get
b := a[4]

// Array length
length := len(a)

// Declare and initialize
aa := [5]int{1, 2, 3, 4, 5}

// Declare a 2D array
var twoD [2][3]int
```

2. Basic overview

Advanced types - slices

```
// Make a slice
s := make([]string, 3)

// Set
s[0] = "a"

// Get
a := s[0]

// Length
length := len(s)

// Append
s = append(s, "b")

// Copy
s2 := make([]string, len(s))
copy(s2, s)

// Slice
l := s[2:3]
// also :3 - up to 3 without 3; 2: - from 2 including 2
```

3. Basic overview

Advanced types - maps

```go
// Make a map
m := make(map[string]int)

// Set
m["k1"] = 7

// Get
v := m["k1"]

// Delete
delete(m, "k1")

// Distinguish between empty values and non-existent ones
if value, exists := m["k1"]; exists {
    fmt.Println("Value exists")
    fmt.Println(value)
}
```

3. Basic overview

Flow control - if

```
// Simple version
if a == 2 {
    fmt.Println("variable is equal 2")
} else {
    fmt.Println("variable is NOT equal 2")
}

// Cascading
if a == 0 {
    fmt.Println("variable is equal 0")
} else if a == 3 {
    fmt.Println("variable is equal 3")
} else {
    fmt.Println("variable is equal to neither 0 nor 3")
}

// Declaration in statement
if c := b.GetValue(); c < 5 {
    fmt.Println("b's value is less than 5")
}
// c no longer in scope
```

3. Basic overview

Flow control - for

```
// Simple version
i := 1
for i <= 3 {
    i = i + 1
}

// Classic for-loop
for j := 1; j <= 9; j++ {
    if j == 2 {
        continue
    } else if j == 5 {
        break
    }
    fmt.Println(j)
}

// For without condition loops until break
for {
    fmt.Println("loop")
    break
}
```

3. Basic overview

Flow control - switch

```
// Simple version
i := 2
switch i {
case 1:
    fmt.Println("one")
case 2:
    fmt.Println("two")
default:
    fmt.Println("something different")
}

// You can combine multiple cases
switch i {
case 1, 2:
    fmt.Println("one or two")
default:
    fmt.Println("something different")
}
```

Does Go have a ternary operator?

a. Yes
b. No

3. Basic overview

Functions

```
// Single return function
func times(a int, b int) int {
    return a * b
}

// Named returns
func threeSum(a, b, c int) (d int) {
   d = a + b
   d = d + c
   return
}

// Multiple returns
func ThreePrimes() (int, int, int) {
   return 2, 3, 5
}

first := times(2, 3)
second := threeSum(1, 2, 3)
thirdOne, _, thirdThree := ThreePrimes()
```

3. Basic overview

Pointers

```
// takes an int value
func zeroval(ival int) {
	ival = 0
}

// takes a pointer to int value
func zeroptr(iptr *int) {
	*iptr = 0
}

func main() {
	i := 1
	fmt.Println("initial:", i) // 1

	zeroval(i)
	fmt.Println("zeroval:", i) // 1

	// &i returns the memory address of i
	zeroptr(&i)
	fmt.Println("zeroptr:", i) // 0
}
```

3. Basic overview

Structures

```
// Define exported struct
type User  struct {
    ID int // public field
    name string // “private” field. accessible only in package
}

// Embed User into Admin struct
type Admin struct {
    User
    Privileges []string
}

// Function that operates on object of type User
func (u User) Login(user,  password string) (error, bool) {
}

// Create object of type User
ziomek := User{1, "Ziomek"}

// Change object field
ziomek.ID = 2
ziomek.Login("admin", "admin")
```

3. Basic overview

Interfaces

```
// define method for login
type User interface {
    Login(user, password string) (error, bool)
}

type Admin struct {
    Privileges []string
}

// function that operates on object of type Admin
func (a Admin) Login(user, passwd string) (error, bool) {
}

type SuperAdmin struct {
}

// function that operates on object of type User
func (a SuperAdmin) Login(user, passwd string) (error, bool) {
}

// to function login we can pass Admin and SuperAdmin
func LoginUser(u User) {
}
```

Arrays are value types. When passed to a function, are they passed as a copy or reference?

a. Copy
b. Reference

Which of the following variables are accessible from another external package?

```
package main

var (
    aName string
    BigBro string
    爱 string
)

func main() {
}
```

a. aName, BigBro
b. BigBro
c. aName, BigBro, 爱
d. BigBro, 爱

4. Concurrency

Goroutines

```
// this task takes 3 seconds to complete
func longTask(txt string) {
    for i := 0; i < 3; i++ {
        fmt.Println(txt, ":", i)
        time.Sleep(1 * time.Second)
    }
}

func main() {
    fmt.Println("hello from main thread")
    longTask("hello from longTask")
    fmt.Println("longTask has ended")

    // we start a new goroutine
    go longTask("hello from longTask")

    // this gets printed right away
    fmt.Println("longTask might still be on")

    // wait for user input
    fmt.Scanln()
}
```

4. Concurrency

Channels

```
// create channel
messages := make(chan string)

// write to channel, this is blocking operation so to work
we should do it in other thread
go func () {
    messages <- "elo"
}()

// receive data from channel
txt := <- messages
fmt.Println(txt)
```

4. Concurrency

Buffered channels
Channel directions

```
// takes a read-only and a write-only channel
func pass (in <-chan string, out chan<- string) {
   txt := <- in
   out <- txt
}

func main() {
    // create buffered channel
    in := make(chan string, 1)
    out := make(chan string, 1)

   // channel is buffered, no need for concurrent reader
   in <- "hello"
   pass(in, out)

   // message passed from channel to channel
   result := <- out
   fmt.Println(result)

   // channels can be closed
   close(in)
}
```

4. Concurrency

Select

```
func main() {
    // create unbuffered channels
    ch1 := make(chan string)
    ch2 := make(chan string)

    // no messages so default is executed
    select {
    case msg := <- ch1:
        fmt.Println("msg received on ch1", msg)
    case msg := <- ch2:
        fmt.Println("msg received on ch2”, msg)
    default:
        fmt.Println("no messages")
    }
    // no listener attached so default is executed
    select {
    case ch1 <- "hello":
        fmt.Println("message sent")
    default:
        fmt.Println("failed to send")
    }
}
```

What's the default buffer size of the channel in Go?

a. 0
b. 1
c. No default size

5. Tooling

go tool vet

```
package main

import "fmt"

func main() {
    str := "hello world!"
    // wrong format
    fmt.Printf("%d\n", str)

    // wrong type
    fmt.Printf("%s\n", &str)
}
```

```
$ go tool vet tool-vet.go
tool-vet.go:8: arg str for printf verb %d of wrong type:
string
tool-vet.go:11: arg &str for printf verb %s of wrong
type: *string
```

5. Tooling

go fmt

```
package main

import "fmt"

type mystruct struct {
    val int
    veryLongValue int// comment

}

func main() {
    fmt.Println("hello")
}

$ go fmt format.go
```

5. Tooling

go tool pprof

[More info]

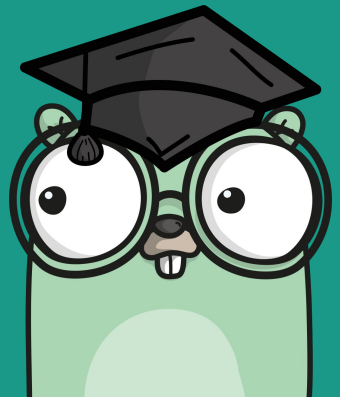
```
0      1.21s (flat, cum) 61.73% of Total
.          .      5:)
.          .      6:
.          .      7:func calculate(a, b int) (d int) {
.          .      8:    d = a + b
.          .      9:    for i := 0; i < d; i++ {
.      1.21s     10:          tab := make([]string, d+1)
.          .     11:          tab[0] = "DL"
.          .     12:    }
.          .     13:    return
.          .     14:}
.          .     15:

610235  610235    (flat, cum)   100% of Total
.          .      5:)
.          .      6:
.          .      7:func calculate(a, b int) (d int) {
.          .      8:    d = a + b
.          .      9:    for i := 0; i < d; i++ {
610235  610235   10:          tab := make([]string, d+1)
.          .     11:          tab[0] = "aaa"
.          .     12:    }
.          .     13:    return
.          .     14:}
.          .     15:
```

6. Project walkthrough

https://github.com/aldor007/mort

Quiz



Assuming that both getx and gety are defined, does this code compile or not?

```
package main
import "fmt"

func main () {
	x, err := getx()
	y, err = gety()

	if err != nil {
	}

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

a. Yes, it does
b. No, it doesn't

Assuming that both getx and gety are defined, does this code compile or not?

```
package main
import "fmt"

func main () {
	x, err := getx()
	x, err := getx()
	y, err := gety()

	if err != nil {
	}

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

a. Yes, it does
b. No, it doesn't

Which of the following is not a bool type in Go?

a. true
b. false
c. 0
d. All of the above

Which of the following is true about for loop statement in Go?

a. If condition is available, then for loop executes as long as condition is true.
b. If range is available, then for loop executes for each item in the range.
c. Both of the above.
d. None of the above.

Open Question:
This code deadlocks.
Why? How to fix it.

```
package main

import "fmt"
import "time"

func wait (c chan<- string) {
	time.Sleep(time.Duration(5));
c<-"done"
}

func main () {
	c := make(chan string)
	wait(c)
	<-c
	fmt.Printf("finished")
}
```

Pizza

6. Simple HTTP server

https://github.com/DreamLab/dl-academy-go/tree/master/golang-talk-examples/crawler-step-by-step