Intro to Go

Golang Cairo #1
8 April 2017

Mohamed Bassem

History

- Announced in 2009
- Go 1.0 was released in March 2012
- Open source
- Created at Google
- Current version is Go 1.8

Who's using Go?

- Google (most notably kubernetes)
- Docker
- Dropbox
- Uber
- Cloudflare
- And many more ... (https://github.com/golang/go/wiki/GoUsers)

What is Go?

Go is :

- Simple
- Fast (benchmarksgame.alioth.debian.org/ (http://benchmarksgame.alioth.debian.org/))
- Compiled
- Statically typed
- Concurrent
- Garbage Collected

What is Go?

Go binaries are statically linked.
Build the binary, copy it to the server and run it.

```
$ go build
$ scp ./binary user@ip ~/binary
$ ssh user@ip ~/binary
```

Go is cross compiled (golang.org/doc/install/source#environment (https://golang.org/doc/install/source#environment)).

You can compile to any target platform from your machine.

```
$ GOOS=windows GOARCH=amd64 go build
```

Hello World

```
package main

import "fmt"

func main() {
    fmt.Println("Hello World!")
}
```

Run

The Go way

Packages

Go code is organised into packages.

Packages map to directories on the file system.

A single directory can contain one and only one package.

Packages are addressed starting from what's called `GOPATH` (except stdlib).

Visibility

Go doesn't have access modifiers (public, private, protected ..).

If a variable/function starts with an uppercase it's exported.

Assume you have:

```
package math

func Add(a int, b int) int {
    return a + b
}
```

Then

```
import "github.com/MohamedBassem/meetup/math"

func main() {
    fmt.Println(math.Add(1,2)) // 3
}
```

Multiple returns

```
func splitAddress(address string) (string, string) {
    parts := strings.Split(address, ":")
    return parts[0], parts[1]
}

func main() {
    host, port := splitAddress("localhost:8080")
    fmt.Println(host)
    fmt.Println(port)
}
```

Run

Error handling

Go doesn't have exceptions.

If a function can fail, its last return value should be of type **error**.

Error handling is done with normal if conditions.

```
ret, err := aFunctionThatCanFail()
if err != nil {
    // Handle the error or return it
}
```

defer

No more :

```
func do() {
    fmt.Println("START")
    if something {
        fmt.Println("END")
        return
    }
    // Do stuff
    fmt.Println("END")
}
```

`defer` defers the execution of a certain call to when the function returns.

```
func do() {
    fmt.Println("START")
    defer fmt.Println("END")
    if something {
        return
    }
    // Do stuff
}
```

Interfaces

You don't need to :

```
class Test implements Stringable
```

If you implement the methods that an interface requires then you implement the interface.

```
type Stringer interface {
    String() string
}
type Test struct {
    msg string
}

func (t Test) String() string {
    return t.msg
}
func main() {
    var x Stringer = Test{msg: "Hello World"}
    fmt.Println(x.String())
}
```

Run

What about `interface{}`?

goroutines

You can think of goroutines as lightweight threads.

Run a function synchronously :

```
longRunningFunction()
```

Run a function asynchronously (in a seperate goroutine) :

```
go longRunningFunction()
```

Goroutines are cheap. You can have thousands of them in a single program.

Channels

Channels are a builtin type that you can send and receive value through.

It's used for communication between goroutines.

```
ch := make(chan string)

go func() {
    ch <- "Hello World!"
}()

val := <-ch
fmt.Printf("Received: %v\n", val)
```

Run

Thing You Won't Find

- Generics (Maybe soon?)
- Overrdiding functions or operators
- Functional style functions (map, reduce, ..)

Toolings

- Go has a standard code style (format) `gofmt`
- `go vet` for common mistakes
- `go doc` for the documentation
- Built in test framework `go test`

Editor integrations
- vim
- emacs
- sublime
- vs code
- ..

Examples

Sleep Sort

```
var wg sync.WaitGroup
x := []int{3, 1, 4, 2, 5}

for _, i := range x {
    go func(a int) {
        wg.Add(1)
        defer wg.Done()
        time.Sleep(time.Duration(a) * time.Second)
        fmt.Println(a)
    }(i)
}
wg.Wait()
```

Run

Producer-Consumer

```
var wg sync.WaitGroup
inChan := make(chan int)

// Consumers
for i := 0; i < 3; i++ {
    go func(id int) {
        wg.Add(1)
        defer wg.Done()
        for job := range inChan {
            time.Sleep(time.Duration(rand.Intn(3)) * time.Second)
            fmt.Printf("Consumer with id %v, done with job: %v\n", id, job)
        }
    }(i)
}

// Producers
for i := 0; i < 20; i++ {
    inChan <- i
}
close(inChan)

wg.Wait()
```

Run

Web server

A web server that reads the 'name' query param and responds with a plain text greeting.

```
func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, req *http.Request) {
        name := req.URL.Query().Get("name")
        fmt.Printf("Got: %v\n", name)
        fmt.Fprintf(w, "Hello %v\n", name)
    })

    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Run

Workshop

Workshop

- Install Go : golang.org/doc/install (https://golang.org/doc/install).
- Start learning Go : tour.golang.org (https://tour.golang.org).
- Go standard package documentation (https://godoc.org/-/go)
- Start solving the problems on Godge.

Thank you

Mohamed Bassem
@MohamedBassem (http://twitter.com/MohamedBassem)