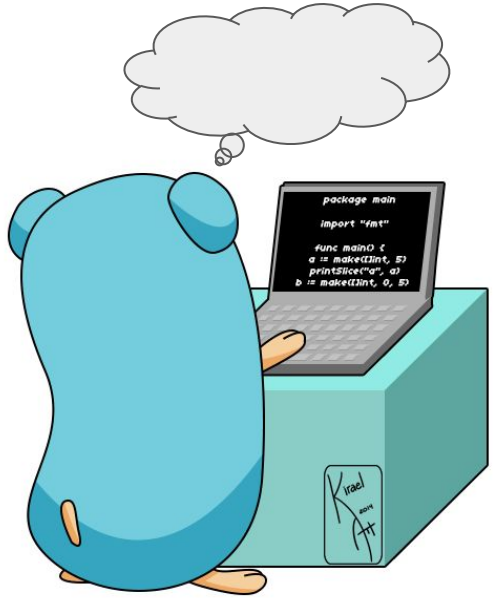
Introduction to the programming language Go (Golang)

Content

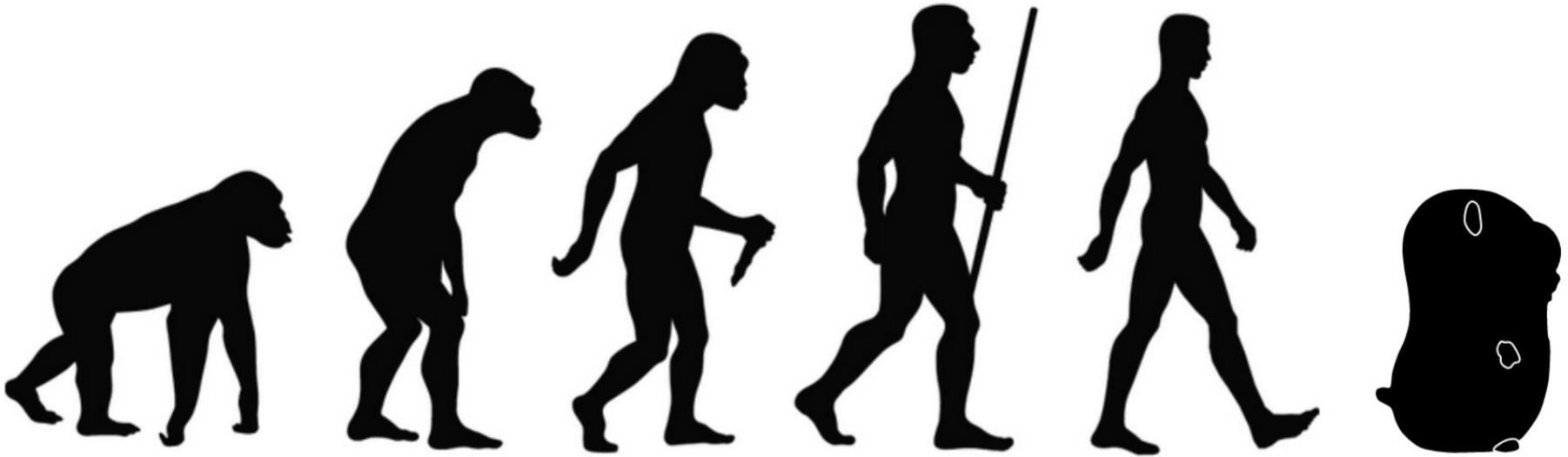
- Introduction
- History
- Applications
- Language and syntax
- Interfaces
- Goroutines and channels
- Error handling
- Packages
- Overview Tooling & IDE’s
- Idiomatic Go
- Formatting and Documentation: go fmt and go doc
- Test, Example
- Modules
- Miscellaneous

The syntax is "C-like":

```
package main

import "fmt"

func main() {
   fmt.Printf("Hello, World\n")
   fmt.Println("嗨")
}
```



History



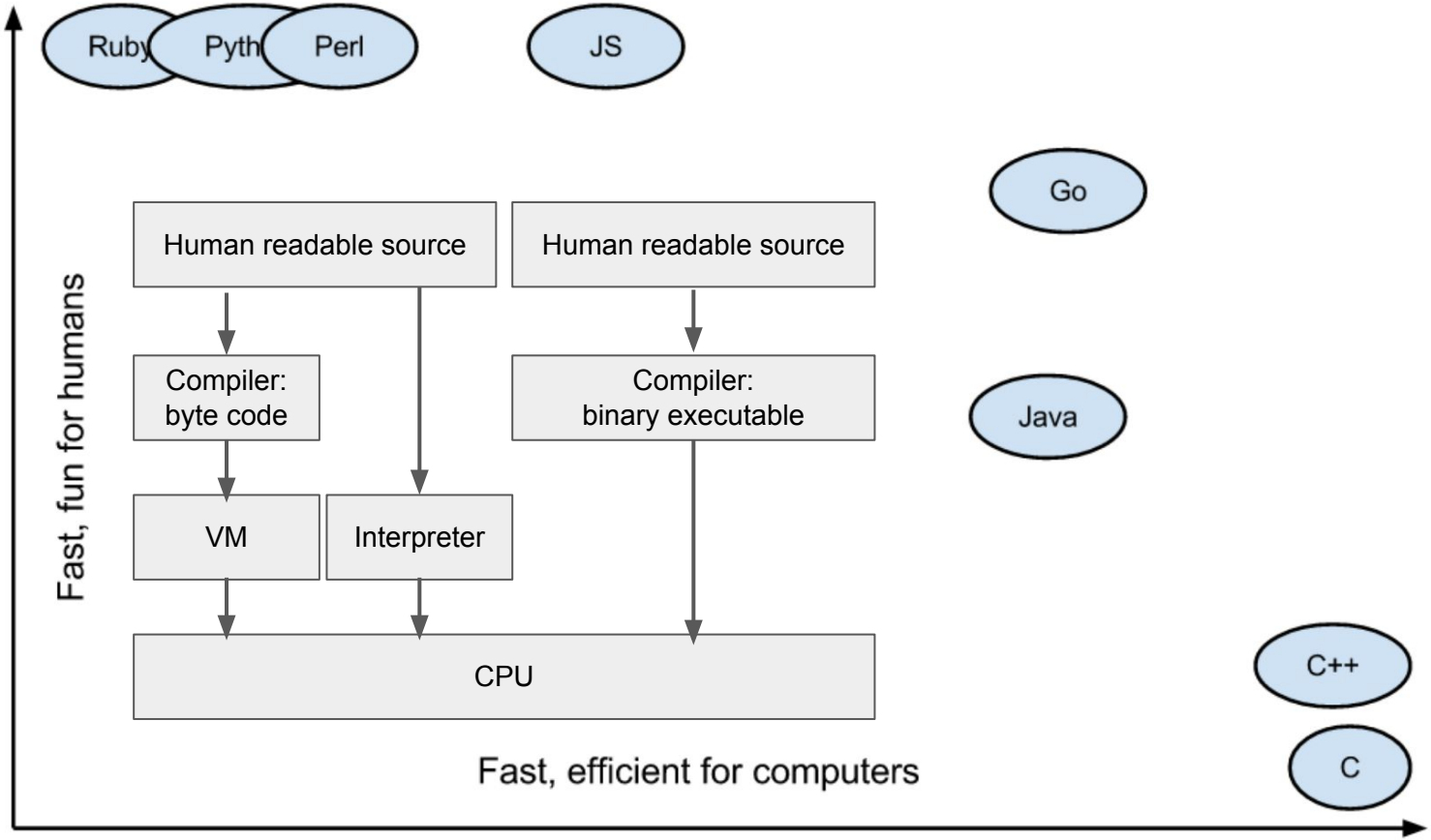
Founding fathers: Rob Pike, Ken Thompson en Robert Griesemer

Golang: How It Started

The language's creators (all from Google) had a clear goal — design a programming language that would be easy to use, but still be able to cover the main challenges while working with the company's intricate systems:

The goals of the Go project were to eliminate the slowness and clumsiness of software development at Google, and thereby to make the process more productive and scalable. The language was designed by and for people who write and read and debug and maintain large software systems.

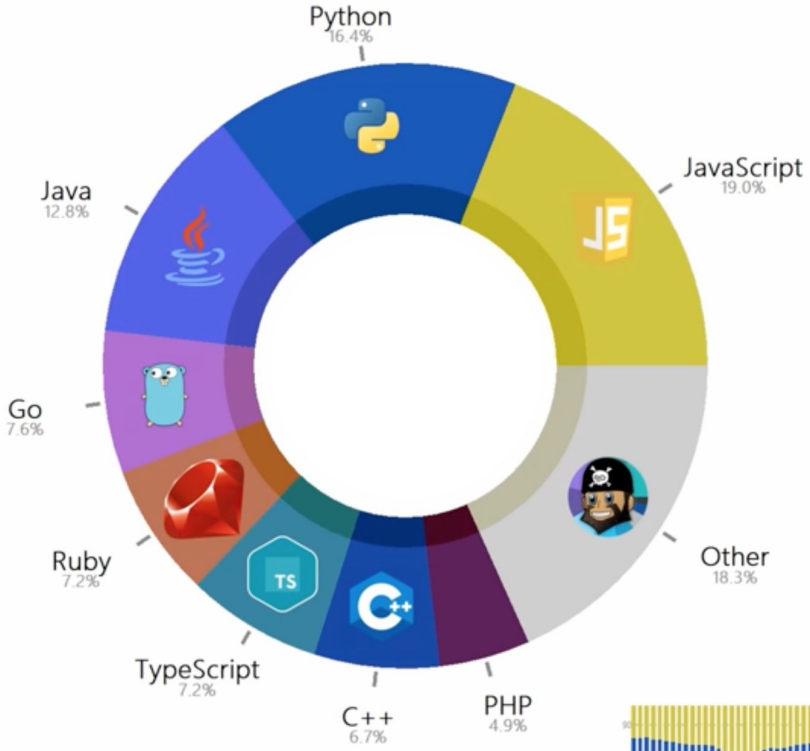
- Rob Pike, Creator of Golang -



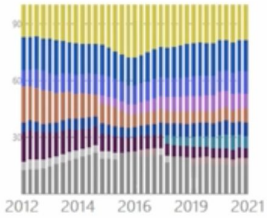
Code readability vs, Efficiency.

MOST POPULAR PROGRAMMING LANGUAGES

According to public GitHub Repositories



Jun 2021



Applications

- Replacement of scripts
- Prototyping
- Network/Backbone
- Microservices
- Complex design/engineering

……. micro kernel, …. etc.

- Docker
- Etcd
- Kubernetes
- Revel
- Gin-gonic
- Influxdb
- Hugo
- Prometheus
- Railgun
- Openshift
- Terraform
- Grafana

Language: variables

- Basic types:

  constants

  int's, float's, string, complex, bool, struct, pointer

  arrays of the above

- Composed types (initialize with **`make`**):

  slice, map (associative array) and channel

- Strongly typed: assigning **`int32`** to **`int64`**: no longer implicit conversion

```
const EOF int = -1
var Euro float32 = 2.2
```

```
type Currency struct {
    Amount float32
    Symbol string
}
```

```
var Dollars [10]float32
```

Control structures

```
if Count > 9 && Count < 12 {
    Ready = true
} else {
    Ready = false
}
```

```
for index, value := range Dollars {
    fmt.Println(index, value)
}
```

```
for i := 0; i < len(Dollars); i++ {
    fmt.Println(i, Dollars[i])
}
```

```
switch Count {
 case 10:
         falltrough
 case 11:
        Ready = true
 default:
        Ready = false

}
```

```
switch {
 case Count > 9 && Count < 12:
        Ready = true
 default:
        Ready = false

}
```

Functions

```
// DivideSafe first checks if the divider is not 0

func DivideSafe(number, divider int) (r int, ok bool) {
    if divider != 0 {
        r = number / divider
        ok = true
    }
    return r,ok
}

func main() {
    result, testok := DivideSafe(5, 0)

    if testok {
        // ....
    }
}
```

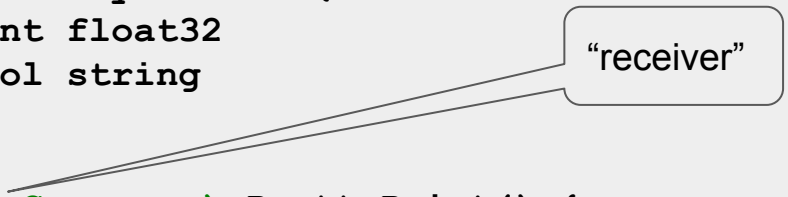
First class functions

```
fp := DivideSafe
r, ok := fp(10, 2)
```

```
anf := func(number, divider int) (r int, ok bool) {
    if divider != 0 {
        r = number / divider
        ok = true
    }
    return r, ok
}

r, ok = anf(10, 2)
```

Methods: linking functions on `struct`'s

```
type Currency struct {
   Amount float32
   Symbol string
}

func (b Currency) PrettyPrint() {
     fmt.Printf("You have: %s%.2f\n", b.Symbol, b.Amount)
}

func main() {

     var MoneyBag Currency = Currency{2.2, "€"}

     Moneybag.PrettyPrint()
}
```



OO in GO: Interfaces

- An interface is the declaration of method-signatures with a common name:

```
type StdoutPrinter interface {
    PrettyPrint()
}
```

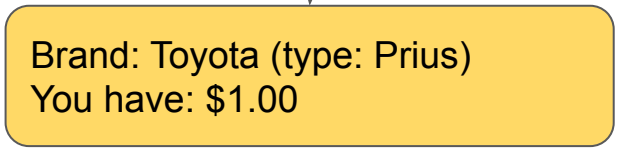
- if all declared methods really exist for a given struct than the interface is said to be *“implemented”*

- Interfaces are the OO part of Go (polymorfism)

Interfaces

```
type Car struct {
   Brand string
   Type string
}

// Car implements the interface StdoutPrinter:
func (c Car) PrettyPrint() {
    fmt.Printf("Brand: %s (type: %s)\n", c.Brand, c.Type)
}
```

```
func PrintAnyStuffToStdout(stuff StdoutPrinter) {
    stuff.PrettyPrint()
}

car := Car{"Toyota", "Prius"}
money := Currency{1.0, "$"}

PrintAnyStuffToStdout(car)
PrintAnyStuffToStdout(money)
```



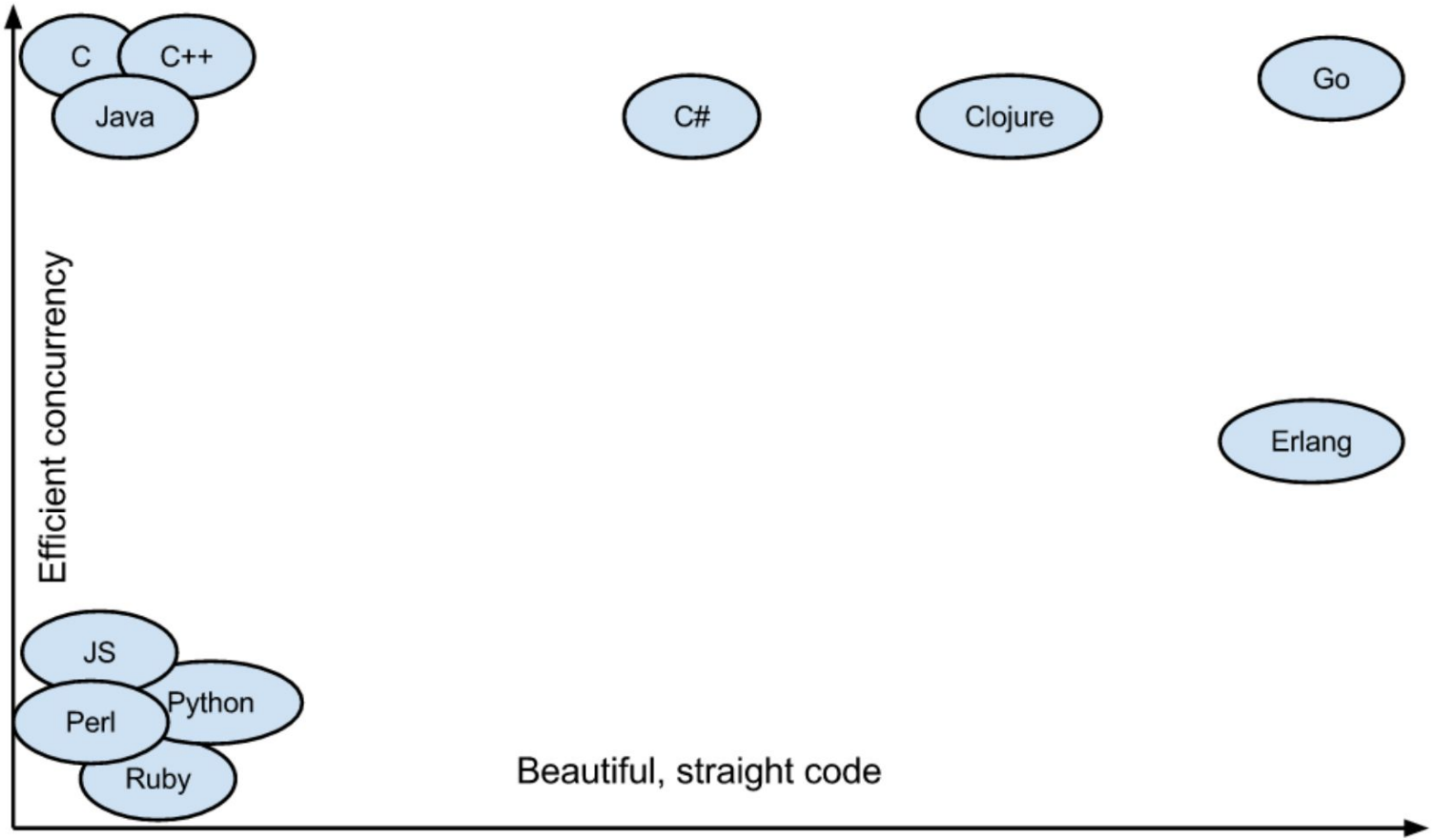
Goroutines: concurrency

A *goroutine* is a lightweight thread managed by the Go runtime

- Each goroutine has a minimal overhead (2kb)
- The schedular Gosched divides goroutines over OS threads
- Gosched puts blocking goroutines on a separate OS thread
- 100.000+ goroutines in parallel is pretty normal
- They communicate and synchronise with channels

Consequence: another way of thinking on multi-threading:

Concurrency is not Parallelism



Go takes good of both the worlds. Easy to write concurrent and efficient to manage concurrency

Starting Goroutines: keyword go

```
func f(from string) {
    for i := 0; i < 3; i++ {
      time.Sleep(50 * time.Millisecond)
      fmt.Println(from, ":", i)
    }
}

func main() {

    go f("goroutine")

    f("direct")

    time.Sleep(2 * time.Second) // prevent that main
                                // finishes to early
}
```

```
goroutine : 0
direct : 0
goroutine : 1
direct : 1
goroutine : 2
direct : 2
```

```
direct : 0
goroutine : 0
direct : 1
goroutine : 1
goroutine : 2
direct : 2
```

Channels

Channels are like "pipes" that connects goroutines.
One goroutine writes a value in a channel and another goroutine reads the value from the channel.

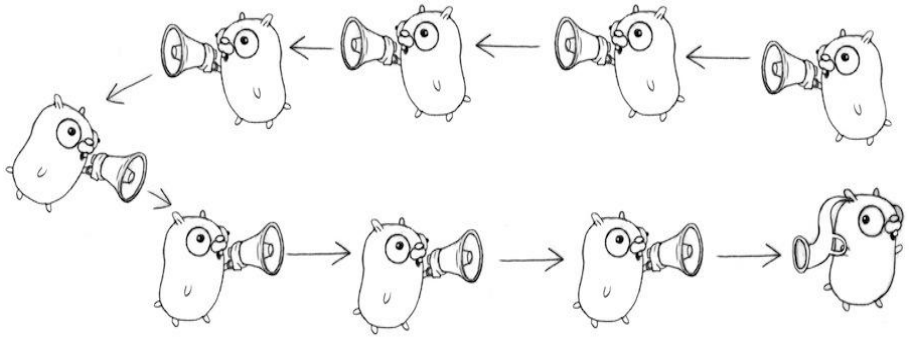
```
func ping(p chan string) {
    p <- "ping"
}

func main() {

    messages := make(chan string)

    go ping(messages)

    msg := <-messages

    fmt.Println(msg)
}
```

Channels

- Synchronisation:
  - a reader of a channel blocks until something is written to the channel
  - a writer to a channel blocks until a reader is present

- Non-blocking channels are possible
- A channel can have capacity for more than one object
- So there is no reason to communicate with global variables:

Don't communicate by sharing memory,
share memory by communicating



defer

- `defer` postpones a function call until the moment of return:

```
func main() {
    f, err := os.Create("/tmp/data.txt")
    if err != nil {
        return
    }
    defer f.Close()
    fmt.Fprintln(f, "data")
}
```

`panic()`

- `panic()` is called if a runtime error occurs:
  - divide by 0
  - index out of bound
- `panic()` can be called in a program as well:

```
panic("Help, no filesystem found")
```

- At `panic()` the function execution stops, but `defer`'s still are executed
  - the caller also `panic()`'s , etc. until `main()` itself `panic()`'s
  - the program terminates with a stack-trace on stderr
  - this proces can be interrupted with a special test: `recover()`

Try not to catch errors: `recover()`

Golang error philosofie in short:

- If you can test for an error situation, then `return` with the error indication
  - divide by 0
  - if opening a file fails
  - errors are variables: it is possible to program with them
- In situations where an unforeseen event can occur, use `recover()`
  - catch panic's in packages (e.g. with network/protocol errors in package net/http)
  - restart crashed goroutine

recover()

```
func p(n int) int {
    defer func() {
      r := recover()
      if r != nil {
        fmt.Println("recovered from ", r)
      }
    }()

    return 10 / n
}

func main() {
    p(0)
    fmt.Println("Hurray! I am still alive")
}
```

```
recovered from  runtime error: integer divide by zero
```

Packages

- Go has a standard set of packages
  - From string manipulations to network services
  - Also lots of packages can be found on the internet

archive tar zip **bufio builtin bytes compress** bzip2 flate gzip lzw zlib **container** heap list ring **context crypto** aes cipher des dsa ecdsa elliptic hmac md5 rand rc4 rsa sha1 sha256 sha512 subtle tls x509 pkix **database** sql driver **debug** dwarf elf gosym macho pe plan9obj **encoding** ascii85 asn1 base32 base64 binary csv gob hex json pem xml **errors expvar flag fmt go** ast build constant doc format importer parser printer scanner token types **hash** adler32 crc32 crc64 fnv **html** template **image** color palette draw gif jpeg png **index** suffixarray **io** ioutil **log** syslog **math** big bits cmplx rand **mime** multipart quotedprintable **net** http cgi cookiejar fcgi httptest httptrace httputil pprof mail rpc jsonrpc smtp textproto url **os** exec signal user **path** filepath **plugin reflect regexp** syntax **runtime** cgo debug msan pprof race trace **sort strconv strings sync** atomic **syscall** js **testing** iotest quick **text** scanner tabwriter template parse **time unicode** utf16 utf8 **unsafe**

Simple static webserver

```
package main

import (
    "net/http"
)

func main() {
    http.ListenAndServe(":8080",
          http.FileServer(http.Dir("/usr/share/doc")))
}
```

HTTP Get

```
func main() {
    res ,err := http.Get("http://www.nu.nl/index.html")
    if err != nil {
        log.Fatal(err)
    }
    robots, err := ioutil.ReadAll(res.Body)
    res.Body.Close()
    if err != nil {
        log.Fatal(err)
    }
    fmt.Printf("%s", robots)
}
```

Tooling

- go env
- go help
- go run
- go build
- go get
- go list
- go mod
- go clean
- gofmt
- go doc
- go test
- go tool
- go vet
- golint
- pprof
- go fix
- go bug
- go delve

- Viewing Environment Information
- Development
  - Running Code
  - Fetching Dependencies
  - Refactoring Code
  - Viewing Go Documentation
- Testing
  - Running Tests
  - Profiling Test Coverage
  - Stress Testing
  - Testing all Dependencies
- Pre-Commit Checks
  - Formatting Code
  - Performing Static Analysis
  - Linting Code
  - Tidying and Verifying your Dependencies

- Build and Deployment
  - Building an Executable
  - Cross-Compilation
  - Using Compiler and Linker Flags
- Diagnosing Problems and Making Optimizations
  - Running and Comparing Benchmarks
  - Profiling and Tracing
  - Checking for Race Conditions
- Managing Dependencies
- Upgrading to a New Go Release
- Reporting Bugs
- Debugger

Performance: Profiling & Tracing

```
func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Split(bufio.ScanWords)
    counts := make(map[string]int)
    for scanner.Scan() {
        word := strings.ToLower(scanner.Text())
        counts[word]++
    }

    var ordered []Count
    for word, count := range counts {
        ordered = append(ordered, Count{word, count})
    }
    sort.Slice(ordered, func(i, j int) bool {
        return ordered[i].Count > ordered[j].Count
    })

    for _, count := range ordered {
        fmt.Println(string(count.Word), count.Count)
    }
}

type Count struct {
    Word  string
    Count int
}
```

Performance comparison: counting words, ignore case:

```
tr 'A-Z' 'a-z' | tr -s ' ' '\n' | sort | uniq -c | sort -nr
```

C	0.96	
Go	1.12	
Rust	1.38	
Java	1.40	
PHP	1.40	
C#	1.50	
C++	1.69	
Perl	1.81	
JavaScript	1.88	
Python	2.21	
Lua	2.50	
Ruby	3.17	
AWK	3.55	
Shell	14.81	(example above)

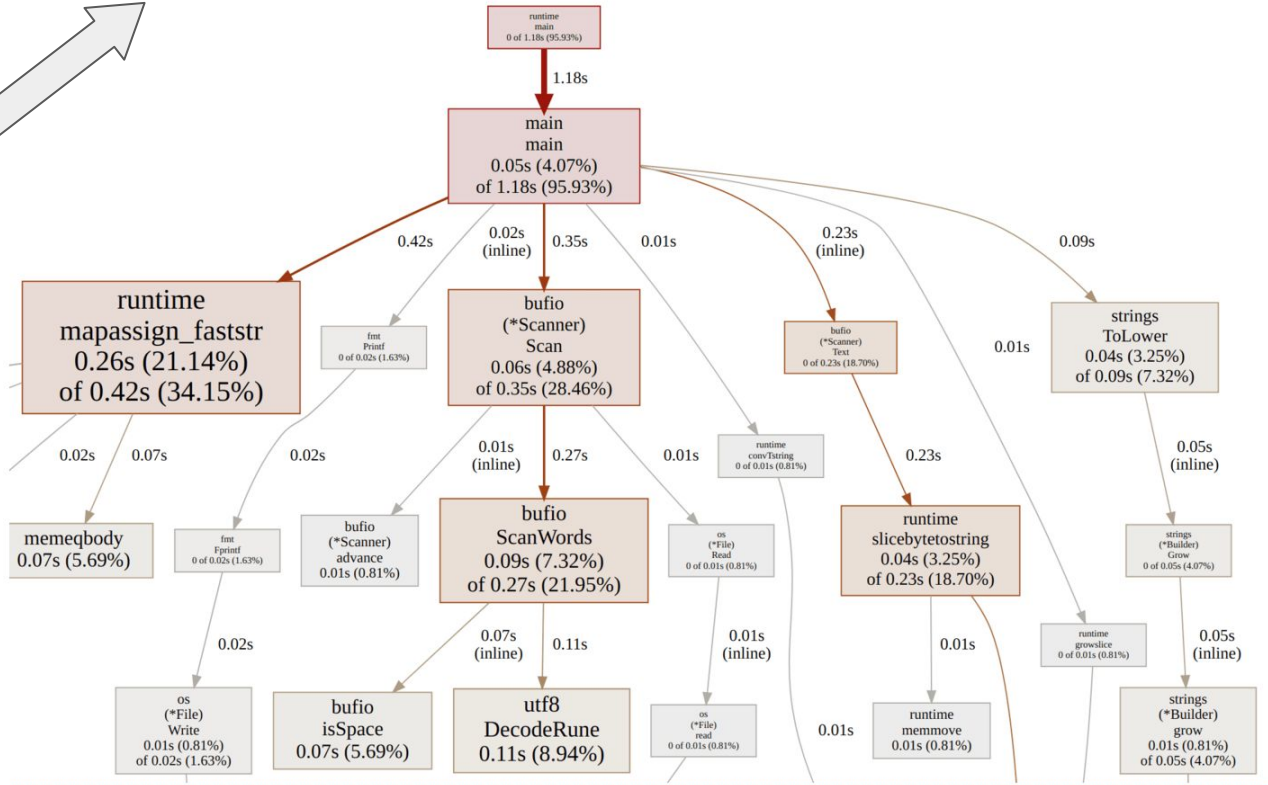
https://benhoyt.com/writings/count-words/

Profiling: PPROF

```
// Add code:

import “runtime/pproff”

f, _ := os.Create("cpuprofile")
pprof.StartCPUProfile(f)
defer pprof.StopCPUProfile()
```

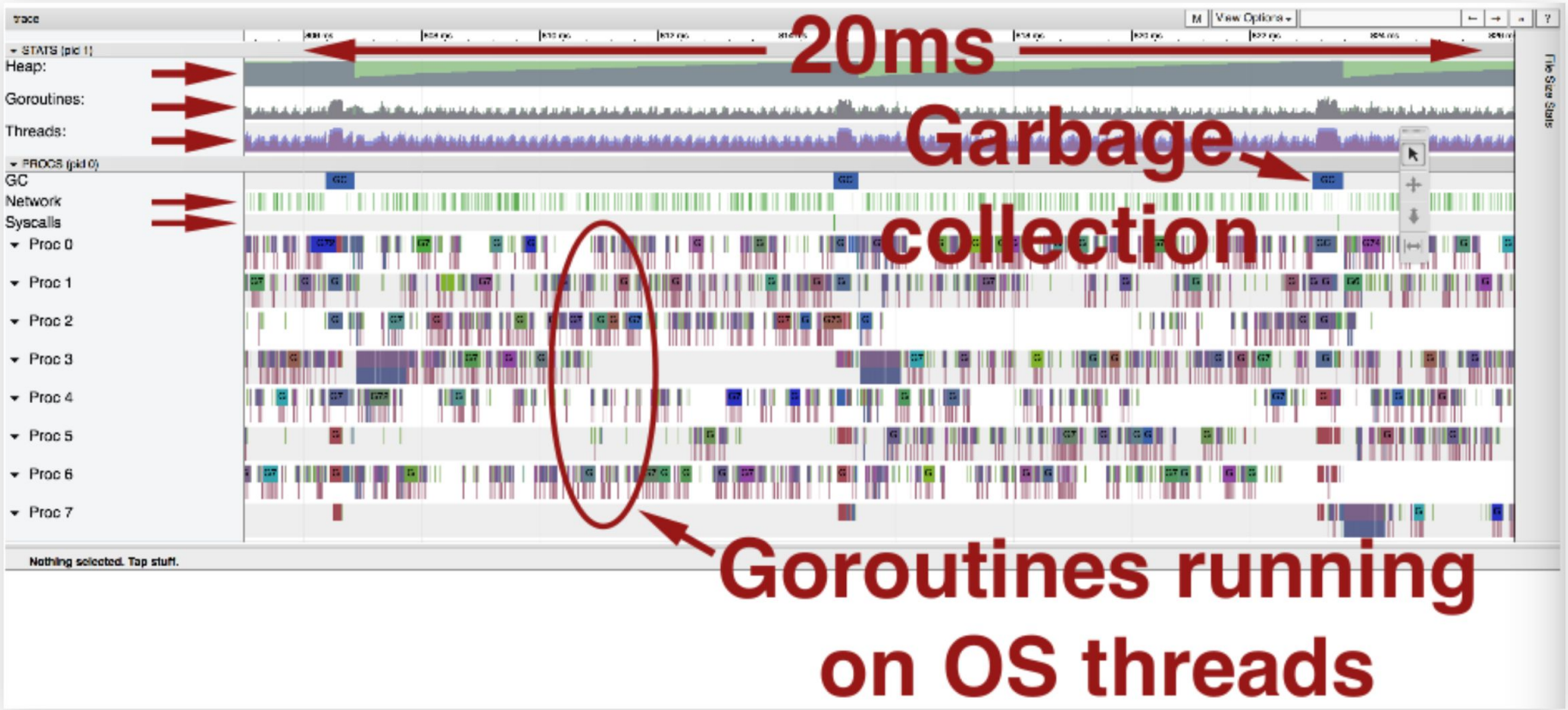
// run program, it creates file: cpuprofile

```
$ go tool pprof -http=:7777 cpuprofile
Serving web UI on http://localhost:7777
```

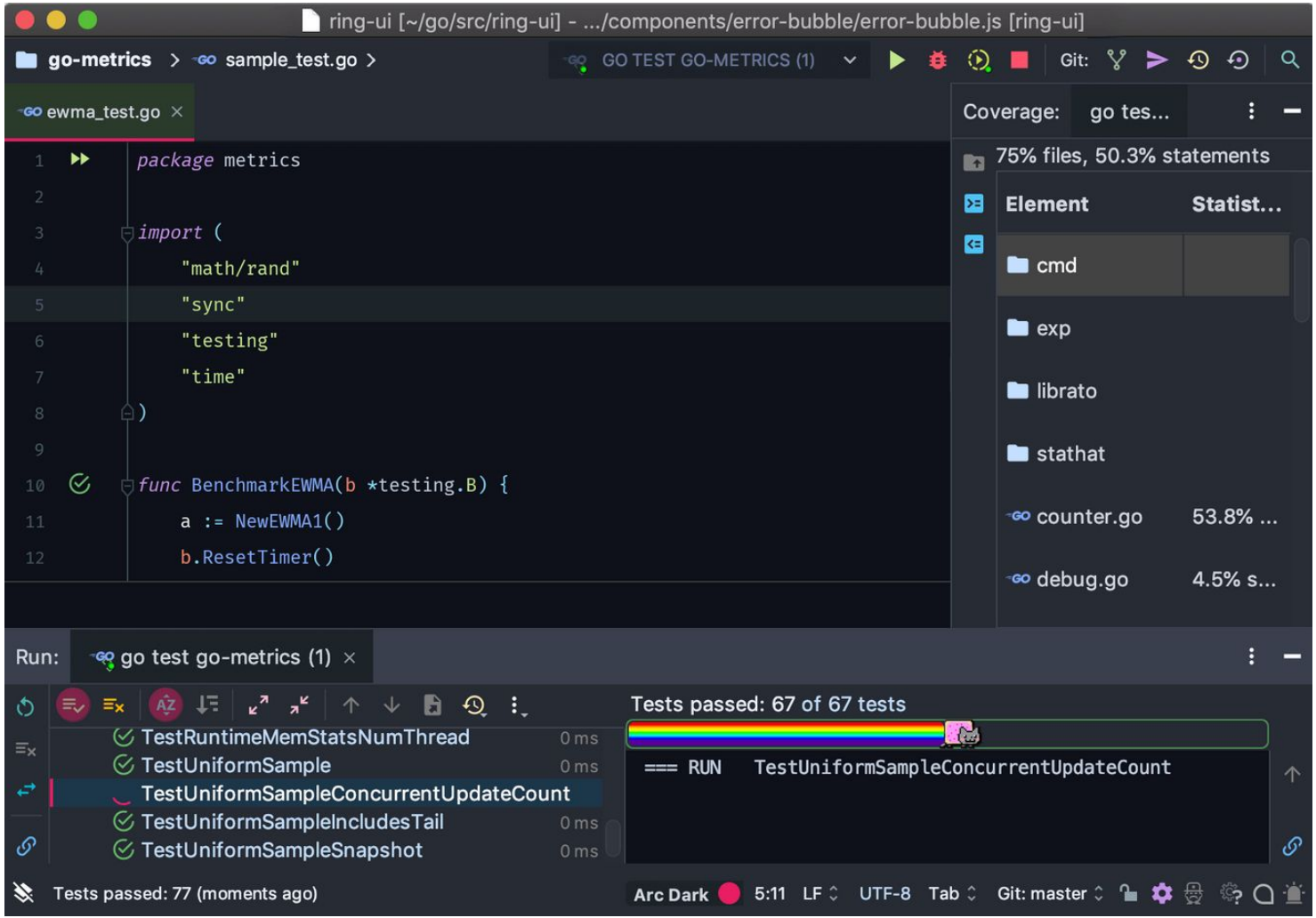


Tracing: trace

```
$ go test -trace=trace.out
$ go tool trace trace.out
```



IDE’s


ring-ui [~/go/src/ring-ui] - .../components/error-bubble/error-bubble.js [ring-ui]
go-metrics > sample_test.go >
GO TEST GO-METRICS (1)
Git:
ewma_test.go
Coverage: go tes...
75% files, 50.3% statements
Element
Statist...
cmd
exp
librato
stathat
counter.go 53.8% ...
debug.go 4.5% s...
package metrics
import (
"math/rand"
"sync"
"testing"
"time"
)
func BenchmarkEWMA(b *testing.B) {
a := NewEWMA1()
b.ResetTimer()
Run:
go test go-metrics (1)
Tests passed: 67 of 67 tests
TestRuntimeMemStatsNumThread 0 ms
TestUniformSample 0 ms
TestUniformSampleConcurrentUpdateCount
TestUniformSampleIncludesTail 0 ms
TestUniformSampleSnapshot 0 ms
=== RUN TestUniformSampleConcurrentUpdateCount
Tests passed: 77 (moments ago)
Arc Dark 5:11 LF UTF-8 Tab Git: master

Editor support and IDE’s

- Atom
- BBedit
- Brackets
- Builder
- Eclipse
- Emacs
- Gedit
- Geany
- Gocode
- godef
- GoLand
- Gotags
- GoWorks
- IntelliJ IDEA Ultimate
- jEdit
- joe
- Lime Text
- LiteIDE
- Notepad++
- Source Insight
- Sublime Text
- Textadept
- TextMate
- TextWrangler
- Vim & Neovim
- Visual Studio
- Visual Studio Code
- GNU Nano
- Zeus

Idiomatic Go

- “Idioms are more like guidelines than rules”
  - https://golang.org/doc/effective_go.html

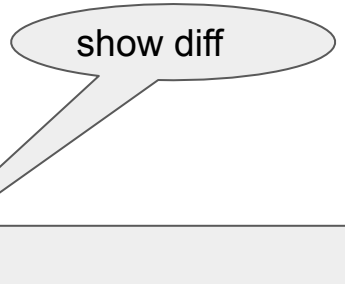
```
err := demo1()
if err != nil {
    return err
}
err = demo2()
if err != nil {
    return err
}
err = demo3()
if err != nil {
    return err
}
return nil
```

CamelCase snake_oil


```
err := demo1()
if err == nil {
    err = demo2()
    if err == nil {
        err = demo3()
        if err == nil {
                return nil
        }
    }
}
return err
```

Gofmt

- With **`gofmt`** (re-)format your Go source code:
  - So the lay out is the same for everybody
  - Rewrite code with the -r flag :




```
var foo int

func bar() {
    foo = 1
    fmt.Println("fool")
}
```

```
$ gofmt -d -w -r 'foo -> Foo' .
-var foo int
+var Foo int

 func bar() {
-   foo = 1
+   Foo = 1
    fmt.Println("fool")
 }
```

go doc

- `go doc` shows the documentation of packages:

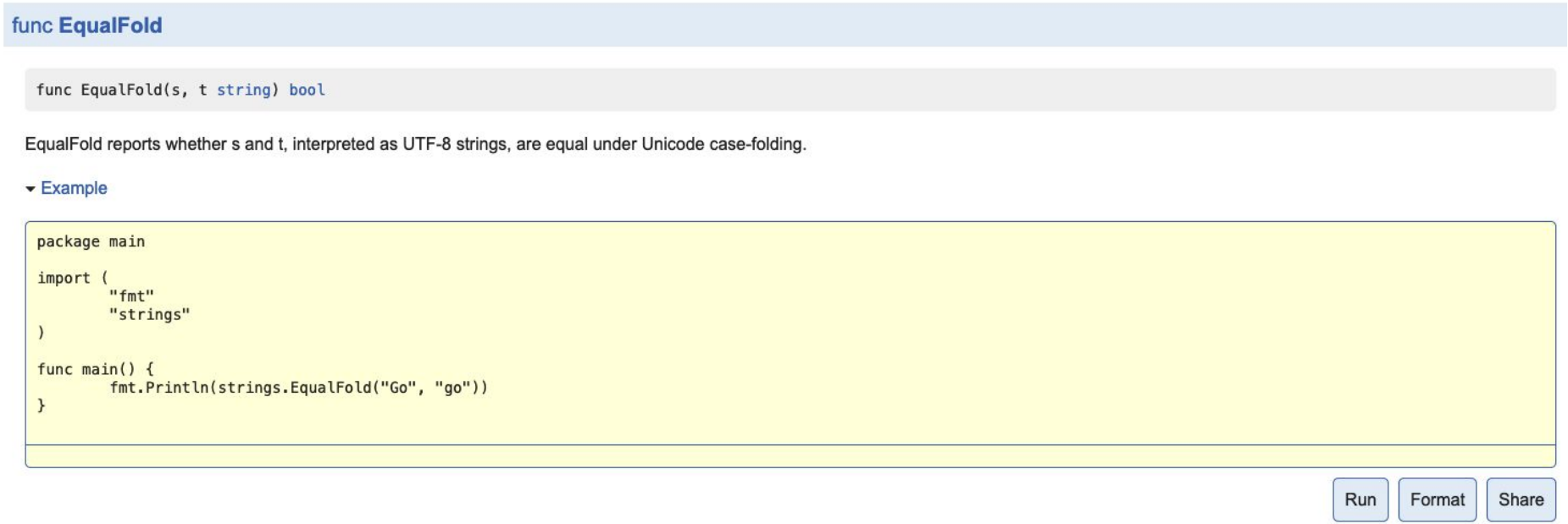
```
$ go doc strings.EqualFold
func EqualFold(s, t string) bool
    EqualFold reports whether s and t, interpreted as
    UTF-8 strings, are equal under Unicode case-folding.
```

Documentation is pulled from the source

```
// EqualFold reports whether s and t, interpreted as UTF-8
// strings, are equal under Unicode case-folding.
func EqualFold(s, t string) bool {
    for s != "" && t != "" {
                    ..................................
```

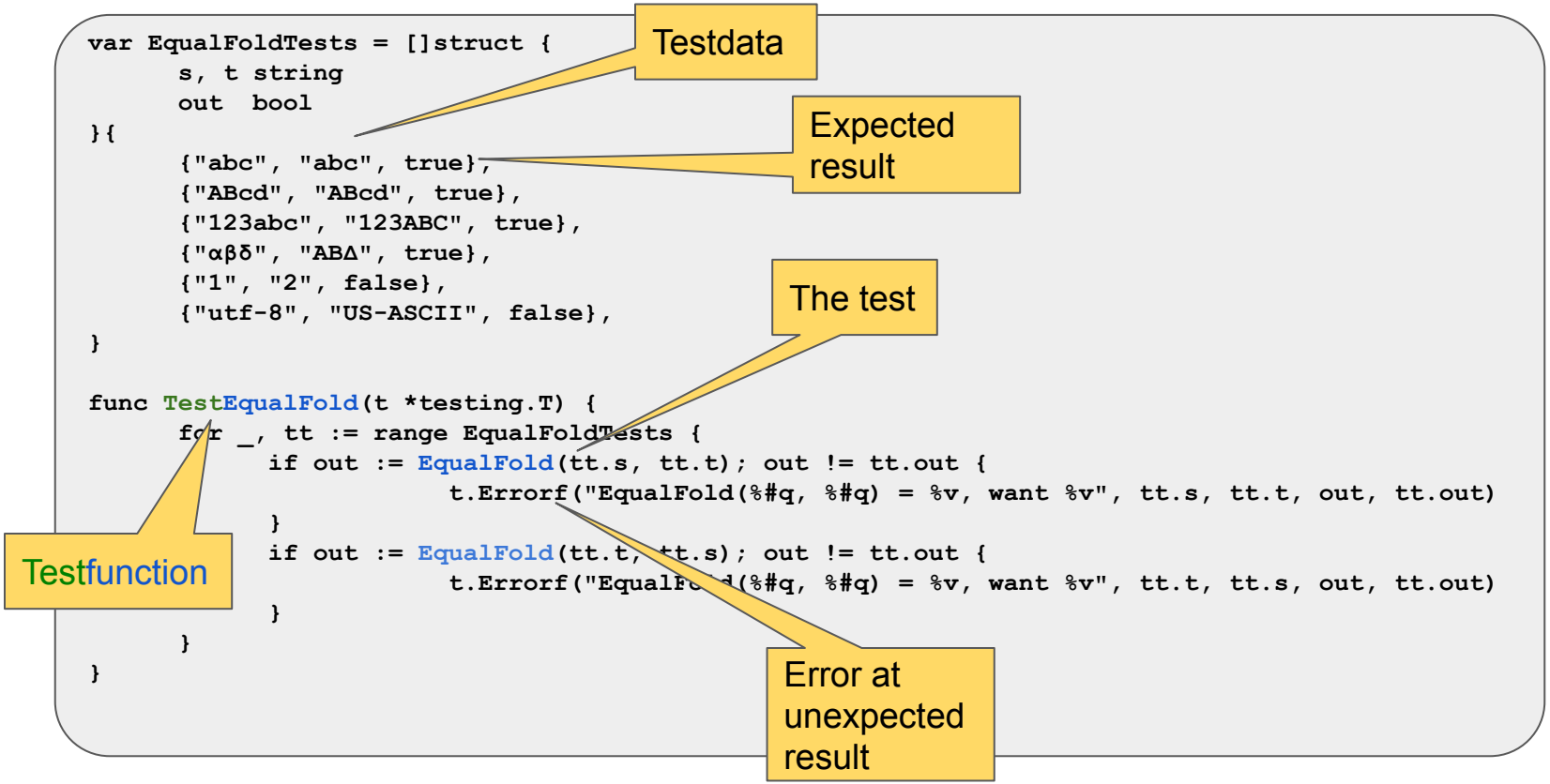
$ godoc -http=:8080

- HTML server
- Example also is pulled from the package source
  - automatically generates Go Playground example



go test

- The `go test` tool enables unit testing, e.g. for **`EqualFold`**:



`go mod init`

- From Go 1.12 “modules” are available
- Designed for version management of packages and manifesto
- Schema: v.1.2.3
  - 1: major change, not compatible
  - .2: extra functions, backward compatible
  - ..3: minor changes: bugfixes
- Mix of versions is possible for different functions from one package, e.g.:
  - `oldfmt.Println()` from v1.2.3
  - `fmt.Print()` from v2.0.0
  - very flexible for migration and refactoring
- Manifesto: Registration SHA of packages + download path + compiler version:
  - The mod file describes all necessary packages + versies
  - Hashes provide for control on authenticity of a package

Embedding static files

```
package main

// https://golang.org/pkg/embed/

import _ "embed"
import "fmt"

//go:embed hello.txt
var s string

func main() {
        fmt.Print(s)
}
```

```
//Complete “File system”


//go:embed image/* template/*
//go:embed html/index.html
var content embed.FS

data, _ :=
content.ReadFile("image/p.png")
```

Generics

```
func PrintINT(s []int) {        // Print integer slice
                                // Sigh.. copy function for
                                // string float etc. etc.
    for _, v := range s { fmt.Print(v) }
}

func Print[T any](s []T) {      // Print any type slice
    for _, v := range s { fmt.Print(v) }
}

func main() {
    Print([]string{"Hello, ", "playground\n"})
    Print([]int{1, 2, 3})
}
```

https://go2goplay.golang.org/p/Kkfsyl0l6Gb

Miscellanious

- Cgo: Embed C (and include and linker directives) in Go
- Cross compiling: generate binaries for other architectures
- Webasm: Compile a Go binary to run in a browser (Chrome, etc).
- Tinygo: Flash Go binaries to micro controllers (Blue Pill etc).