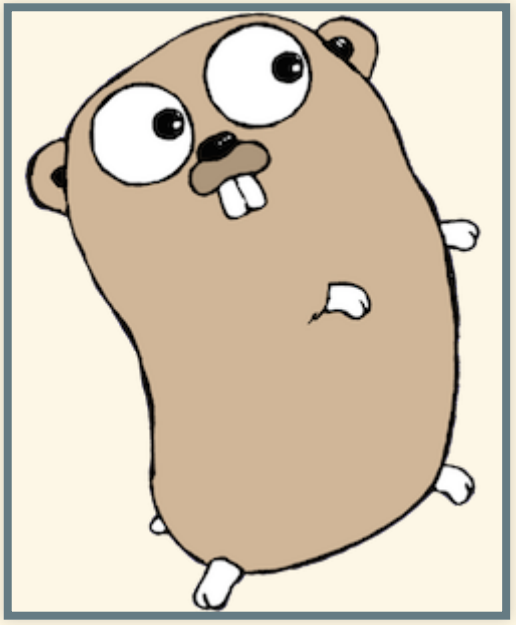
GO

"Go is an open source programming language that makes it easy to build simple, reliable, and efficient software"



expressive, statically typed, compiled

```go
package main

import "fmt"

func main() {
    fmt.Println("hello, world")
}
```

```
$ go build main.go
$ ./main
hello, world
$ # equivalent
$ go run main.go
hello, world
```

most of my code examples are not executable as is

package/import statements are omitted for better readability

BASICS

TYPES

```
bool // example: true
string // example: "string"

int  int8  int16  int32  int64 // example: 0
uint uint8 uint16 uint32 uint64 uintptr // example: 0

byte // alias for uint8

rune // alias for int32
     // represents a Unicode code point
     // example: 'a'

float32 float64 // example:  0.1

complex64 complex128 // example: (0+0i)
```

ZERO VALUES

type	null type
numeric types	`0`
boolean types	`false`
strings	`""`
structs	`nil`

VARIABLES

```
var x int
var y int
var z int

func main() {
    // equivalent to
    var x, y, z int
}
```

```go
var x = 1
var y = 2
var z = 3

func main() {
    // equivalent to
    var x, y, z = 1, 2, 3
}
```

```go
func main() {
    x := 1
    y := 2
    z := 3
}
```

```
const (
    c1 = 'c'
    c2 = "s"
    c3 = true
    c4 = 0
)
```

FUNCTIONS

```
func func1(arg1 int) int {
    return arg1
}
```

```go
func func2(arg1, arg2 int) (int, int, error) {
    return arg1, arg2, nil
}
```

FLOW CONTROL

for

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

```
i := 0
for ; i < 10; {
    fmt.Println(i)
    i += 1
}
```

```
i := 0
for i < 10 {
    fmt.Println(i)
    i += 1
}
```

```
for {
    // ever
}
```

if

```
if i < 10 {

}
```

```
if i < 10 {

} else {

}
```

```
if i := 5; i < 3 {

}
```

switch

```
abc := "abc"
switch abc {
case "abc":
    fmt.Println(":)")
case "cba":
    fmt.Println("uhm")
default:
    fmt.Println(abc)
}
```

```go
func someFunction(s string) string {
    fmt.Printf("I will return %s\n", s)
    return s
}

func main() {
    switch abc := "abc"; abc {
    case someFunction("abc"):
        fmt.Println(":)")
    case someFunction("cba"):
        fmt.Println("uhm")
    default:
        fmt.Println(abc)
    }
}
```

How often will `someFunction` run?

```
i := 0
switch {
case i < 10:
case i < 5:
case i < 1:
}
```

defer

```
func main() {
    defer fmt.Println("Second")
    fmt.Println("First")
}
```

```
First
Second
```

```go
func main() {
    defer fmt.Println(3)
    defer fmt.Println(2)
    defer fmt.Println(1)
    defer fmt.Println(0)
}
```

```
0
1
2
3
```

ADVANCED TYPES

struct

```
type Point struct {
    X, Y int
}

func main() {
    p := Point{X: 1, Y: 2}
    // equivalent
    p = Point{1, 2}
    fmt.Println(p)
    fmt.Println(p.X, p.Y)
}
```

```
{1 2}
1 2
```

Arrays

```
var a [2]int
a[0] = 0
a[1] = 1
// equivalent
a = [2]int{0, 1}
```

Slices

```
a := [5]int{0, 1, 2, 3, 4}
sl := a[1:4]
fmt.Println(sl)
sl[0] = 42
fmt.Println(a)
```

```
[1 2 3]
[0 42 2 3 4]
```

```
sl := []int{42, 2, 3}
```

```
sl := make([]int, 1)
fmt.Println(sl[0])
sl[0] = 1
sl = append(sl, 2)
fmt.Println(sl)
```

```
0
[1 2]
```

iteration over slices

```
sl := []string{"Hello", "World"}
for idx := range sl {
}
```

```
for idx, value := range sl {
}
```

```
for _, value := range sl {
}
```

maps

```
m := make(map[string]int)
m := map[string]int{
    "abc": 0,
}
```

```
// insert/update
m["def"] = 0
// retrieve
m["def"]
// delete
delete(m, "def")
// test key is present
value, isPresent := m["ghi"]
```

Pointer

```
i := 1
p := &i // pointer to i
fmt.Println(*p) // read i through p
*p = 2 // modify i through p
```

```go
func Clear(p *Point) {
    p.X = 0
    p.Y = 0
}
```

METHODS

```go
type Point struct {
    X, Y int
}

func (p Point) Sum() int {
    return p.X + p.Y
}

// fancy alternative for
func Sum(p Point) int {
    return p.X + p.Y
}

func main() {
    p := Point{1, 2}
    p.Sum()
    // fancy alternative for
    Sum(p)
}
```

```go
func (p *Point) IncX() {
    p.X = p.X + 1
}

func (p *Point) IncY() {
    p.Y = p.Y + 1
}

func (p *Point) Clear() {
    p.X = 0
    p.Y = 0
}

func main() {
    p.IncX()
    p.IncY()
}
```

INTERFACES

```
type Clearable interface {
    Clear()
}
```

```go
func clearMe(c Clearable) {
    c.Clear()
}

func main() {
    clearMe(&Point{1, 2})
}
```

```
interface{}
// dynamic typing \o/
fmt.Printf("%s, %d", "Hello World", 1)
```

```go
var s interface{} = 1

sAsString := s.(string) // panic!
sAsInt := s.(int)
```

```
switch s.(type) {
case int:
case string:
case bool:
}
```

ERROR HANDLING

- there are no exceptions
- error handling is (hopefully) done explicitly

```
_, err := strconv.Atoi("77")
if err != nil {
    log.Fatal(err)
}
```

```go
func add(i1, i2 int) (int, error) {
    if r := i1 + i2; r < 100 {
        return r, nil
    } else {
        return 0, errors.New("oO")
    }
}

func main() {
    _, err := add(1, 99)
    if err != nil {
        log.Fatal(err)
    }
}
```

GOROUTINES

```go
func calculate(i int) {
    result := 0
    for ; i < 100; i++ {
        result = result + i + 42
    }
    fmt.Printf("I'm done: %d\n", result)
}

func main() {
    go calculate(5)
    fmt.Println("Done")
}
```

```go
func calculate(i int, results chan int) {
    result := 0
    for ; i < 100; i++ {
        result = result + i + 42
    }
    results <- result
}

func main() {
    results := make(chan int)
    go calculate(5, results)
    go calculate(10, results)
    result1, result2 := <- results, <- results
    fmt.Printf("Result1: %d Result2: %d\n", result1, result2)
}
```

```go
func sendSequential(s string, transport chan rune) {
    for _, val := range s {
        transport <- val
    }
    close(transport)
}

func main() {
    transport := make(chan rune)
    go sendSequential("Hello World", transport)
    for val := range transport {
        fmt.Print(string(val))
    }
    fmt.Print("\n")
}
```

PACKAGES

```go
package main

import "fmt"

func main() {
    fmt.Println("hello, world")
}
```

Directory src/fmt/

Documentation: **fmt**

File	Bytes	Modified
..		
doc.go	14559	2018-11-02 22:11:42 +0000 UTC
example_test.go	551	2018-11-02 22:11:42 +0000 UTC
export_test.go	219	2018-11-02 22:11:42 +0000 UTC
fmt_test.go	56275	2018-11-02 22:11:42 +0000 UTC
format.go	12673	2018-11-02 22:11:42 +0000 UTC
print.go	30006	2018-11-02 22:11:42 +0000 UTC
scan.go	31951	2018-11-02 22:11:42 +0000 UTC
scan_test.go	38408	2018-11-02 22:11:42 +0000 UTC
stringer_test.go	2156	2018-11-02 22:11:42 +0000 UTC

```
package fmt
// Println formats using the default formats for its operands and
// Spaces are always added between operands and a newline is appe
// It returns the number of bytes written and any write error enc
func Println(a ...interface{}) (n int, err error) {
    return Fprintln(os.Stdout, a...)
}
```

```
import "net/http"

resp, err := http.Get("http://example.com/")
```

IMPORTS

```go
package main

import "fmt"

func main() {
    fmt.Println("hello, world")
}
```

```go
package world

import "fmt"

func hello(name string) {
    fmt.Printf("hello, %s\n", name)
}
```

```go
package main

import "world"

func main() {
    world.hello("world")
}
```

Will this code compile?

let's assume the import will just work ™

```
$ go run main.go
./main.go:8:2: cannot refer to unexported name world.hello
./main.go:8:2: undefined: world.hello
```

SCOPES

```
package world

func Hello(name string) {
    fmt.Printf("Hello, %s\n", name)
}

func hello(name string) {
    fmt.Printf("hello, %s\n", name)
}
```

Object scopes are defined by their capitalization

```
var a // unexported
var B // exported
type C struct { // exported

}
func func1() { // unexported

}
```

DEPENDENCY MANAGEMENT

```
$ go get github.com/gorilla/mux
$ tree $GOPATH
/home/jseydel/go
├── bin
├── pkg
└── src
    └── github.com
        └── gorilla
            └── mux
```

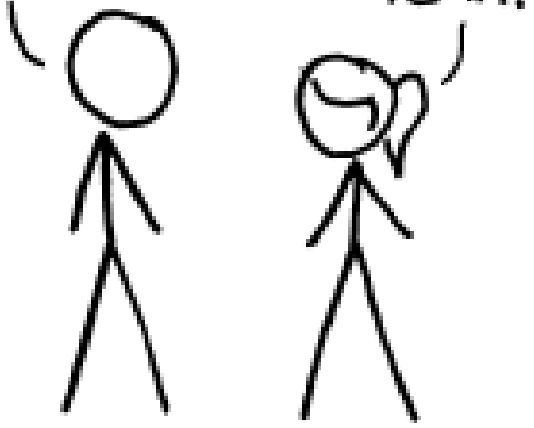
```go
package main

import "github.com/gorilla/mux"

func main() {
    r := mux.NewRouter()
}
```

Tool Name	Url	Reference Count (Feb 2017)	Reference Count (Nov 2017)
Makefile	Makefile	199	181
dep	dep	N/A	94
godep	godep	119	90
govendor	govendor	65	84
glide	glide	64	77
gvt	gvt	25	16
trash	trash	7	13
submodule	submodule	8	6
gpm/johnny-deps	gpm johnny-deps	7	6
glock	glock	5	4
gom	gom	4	2
gopack	gopack	3	2
gopm	gopm	3	1
goop	goop	1	1
gvend	gvend	2	0

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



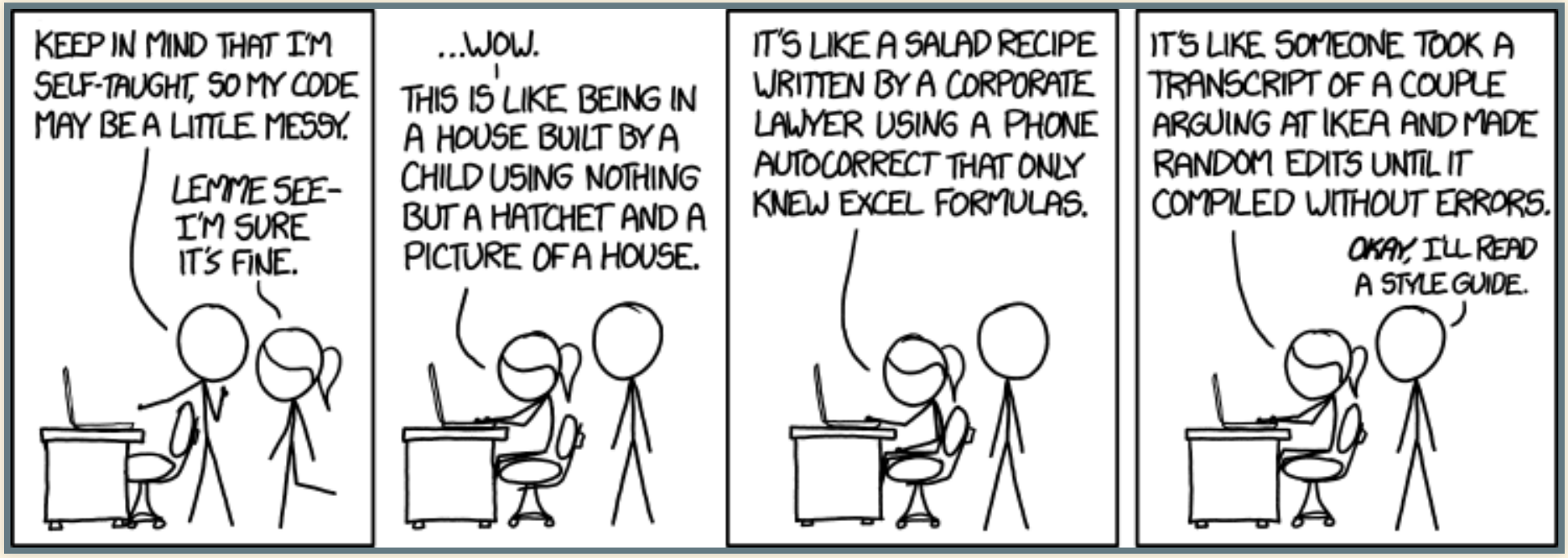
SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

go mod

TOOLING

go fmt


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE- I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

go vet

```
fmt.Printf("%s %s %s", "hello world")
```

```
$ go vet main.go
./main.go:6: Printf format %s reads arg #2, but call has 1 arg
```

go imports

REFERENCES

- A Tour of Go
- Package Documentation
- Effective Go

DEMO & TASKS

https://git.io/fpJ78

THANKS FOR YOUR ATTENTION AND KEEP GOING