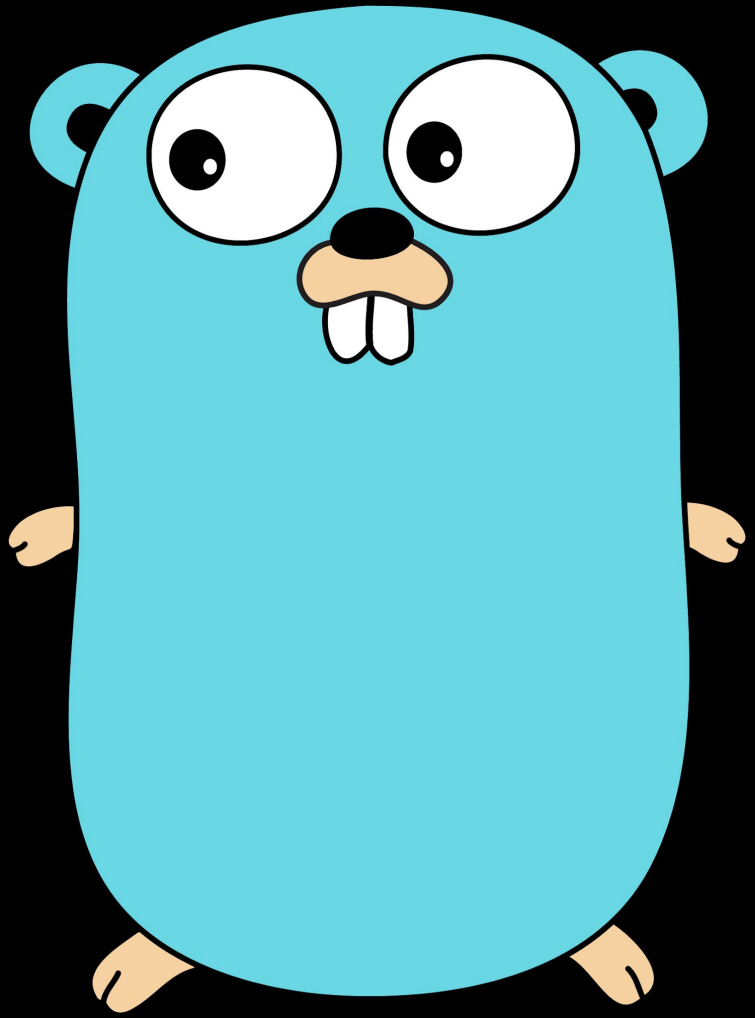
The Go (Golang) Programming Language

—

By Christoph Zauner (https://zauner.NLLK.net/)



Why another language?

- Make onboarding of new developers as easy as possible.
- Programs should be easily deployable and be as resource efficient as possible.

Simple?

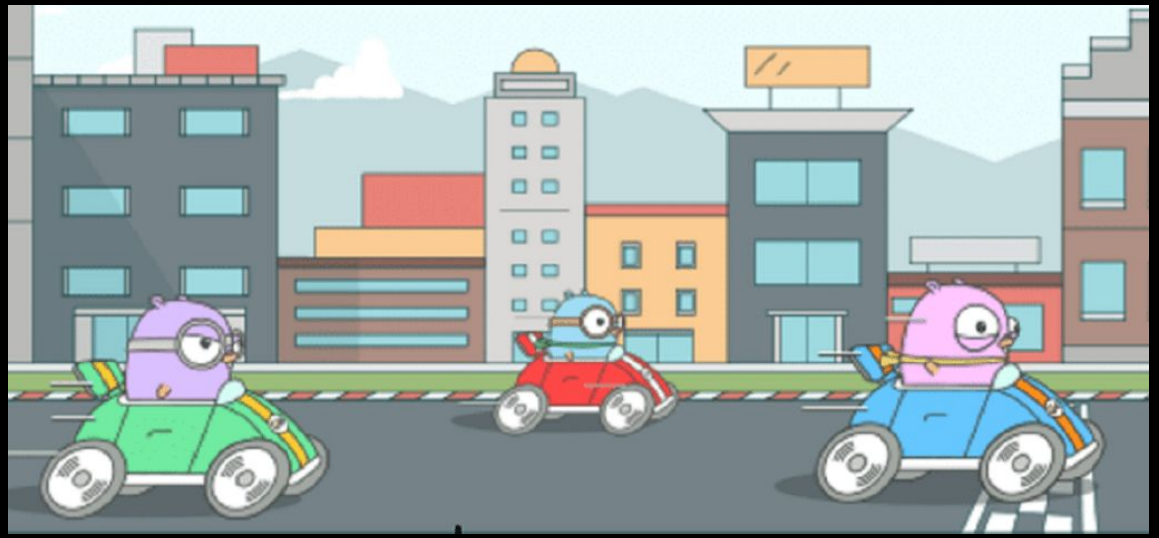
Spec comparison

Simple?

Spec comparison

- Go Language Specification
  - 75 A4 pages
- Java Language Specification
  - 772 A4 pages
- C# Language Specification
  - 516 A4 pages
- Javascript Language Specification
  - 805 A4 pages
- C++ Language Specification
  - 1579 A4 pages
- Python Language Reference
  - 165 A4 pages

Easily deployable and resource efficient?



Native Binary

Go program

```
zaunerc@2217PC12387 $ ldd snooop-semanteer-testtool

linux-vdso.so.1 => (0x00007ffd76fc4000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
/lib64/ld-linux-x86-64.so.2

zaunerc@2217PC12387 $
```

Native Binary

C++ program

```
zaunerc@2217PC12387:~$ ldd /usr/bin/kate

linux-vdso.so.1 => (0x00007ffd0c524000)
libKF5TextEditor.so.5 => /usr/<...>/libKF5TextEditor.so.5
libKF5Parts.so.5 => /usr/lib/x86_64-linux-gnu/libKF5Parts.so.5

<114 lines omitted>

libsqlite3.so.0 => /usr/lib/x86_64-linux-gnu/libsqlite3.so.0
libcrypt.so.1 => /lib/x86_64-linux-gnu/libcrypt.so.1

zaunerc@2217PC12387:~$
```

Cross Compilation

Easy...

```
$ GOOS=windows GOARCH=amd64 go build snooop-semanteer-testtool.go
$ file snooop-semanteer-testtool.exe

PE32+ executable (console) x86-64 (stripped to external PDB), for
MS Windows

$ go build snooop-semanteer-testtool.go
$ file snooop-semanteer-testtool

ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, with debug_info,
not stripped

$ GOOS=js GOARCH=wasm go build snooop-semanteer-testtool.go
$ file snooop-semanteer-testtool

WebAssembly (wasm) binary module version 0x1 (MVP)
```

Instruction Sets

$GOARCH

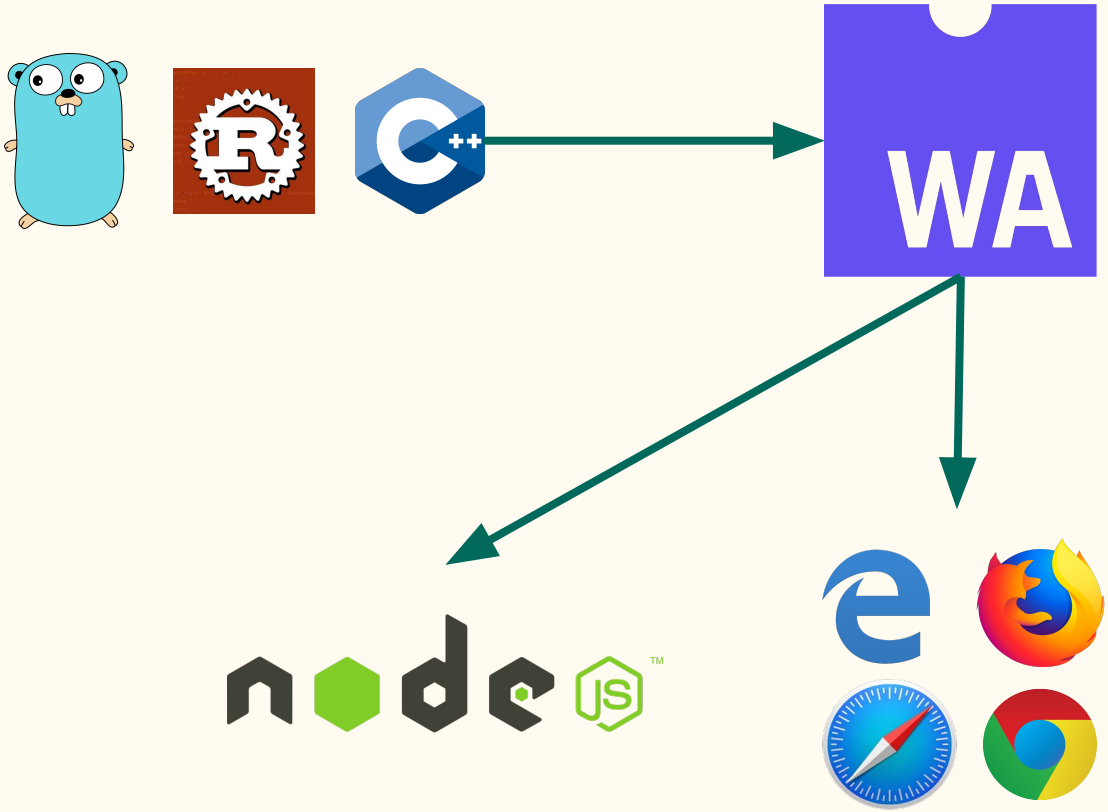
- amd64 (also known as x86-64)
- 386 (x86 or x86-32)
- arm
- arm64 (AArch64)
- ppc64, ppc64le (64-bit PowerPC big- and little-endian)
- mips, mipsle (32-bit MIPS big- and little-endian)
- mips64, mips64le (64-bit MIPS big- and little-endian)
- s390x (IBM System z)
- wasm (WebAssembly)

Operating Systems

$GOOS

- Android
- DragonFly BSD
- FreeBSD
- Javascript (js)
- Linux
- NetBSD
- OpenBSD
- Darwin (macOS and iOS)
- Plan 9
- Solaris
- Windows

WebAssembly Showcase



Web Assembly

Links

- WebAssembly Explorer
  - An online tool which allows you play around with a C/C++ compiler and understand how WebAssembly code is produced, delivered, and ultimately consumed by the browser.

- WebAssembly Fiddle, lets you write, share, and run WebAssembly code snippets in the browser.

- WebAssembly Code Explorer: For an even deeper dive, you can inspect WebAssembly binaries to understand how WebAssembly code is encoded at a binary level.

- WebAssembly Porting Examples and Demos
  - QEMU, KDE programs, ...

The basics

The basics

- Unlike e.g. C# or Java Go does not run in a virtual machine
  - Benefit: You won't ever have to tweak you VMs memory settings (e.g. JVM max heap size).
- It is a compiled language.
  - All depedencies are statically linked by default.
  - Dynamically linking is also supported.
- Cross-compiling is supported without the need install complicated toolchains.
- Go is statically-typed.
- Using reflection is possible (package reflect).

The basics

- No support for generics.
- Strings offer UTF-8 support out of the box.
- “nil” is used as the null value for e.g. pointers. But basic types like e.g. Strings (“”), the boolean type (false) and numeric types (0) are always initalized.
- Multiple return values are supported.

```
func SumAndProduct(x, y int) (int, int) {
  return (x + y), (x * y)
}

sum, product := SumAndProduct(2, 3)
```

The basics

Types

- Boolean
- Numeric
- String
- Array
- Slice
- Struct
- Pointer
- Function
- Interface
- Map
- Channel
- Go Spec about Types

```
// numeric type examples

var smallInteger int8
var bigInteger int64

// slice examples

aSlice := make([]int64, 10)

// struct type examples

type Rectangle struct {
  A int
  B int
}

// pointer to a number

ptr := &bigInteger
```

The basics

Access Modifiers

- Lower Letter means unexported
  - The function, variable, ... is only accessible within their package.
- Capital Letter means exported
  - The function, variable, … is accessible from outside their package as well.

```
PublicCtr int; // exported - accessible outside package
privateCtr int; // unexported - only accessible within package
```

The basics

Pointers

- By default every parameter is passed by value.
- Even objects are passed by value
  - Objects are called structs in Go anyway.
- Go allows you to create pointers to variables or structs explicitly.
  - Think of pointers as references as known in Java or C#.
- Pointer arithmetic like in C is not supported.

The basics

- The compiler checks the formatting of the source code. If there are error you just have to execute gofmt to fix the formatting.
- Variables are block scoped.
- Go does not support function/method overloading.
- File names do not play any role. E.g. when using a function from another package you use the following qualifier (note that there is no file name present in the qualifier):

```
<package name>.<function name>
```

The basics

Dependency management

Go modules

- **“go modules” is the way to go**
  - The experiment was called vgo.
  - Further information can be found here.
- Go supports vendoring
  - Local copies of external dependencies to satisfy imports of those dependencies.
- If you want to publish a library you will also have to share the source code for it. There is no such thing as header files as in C/C++.

Dependency management

Deprecated

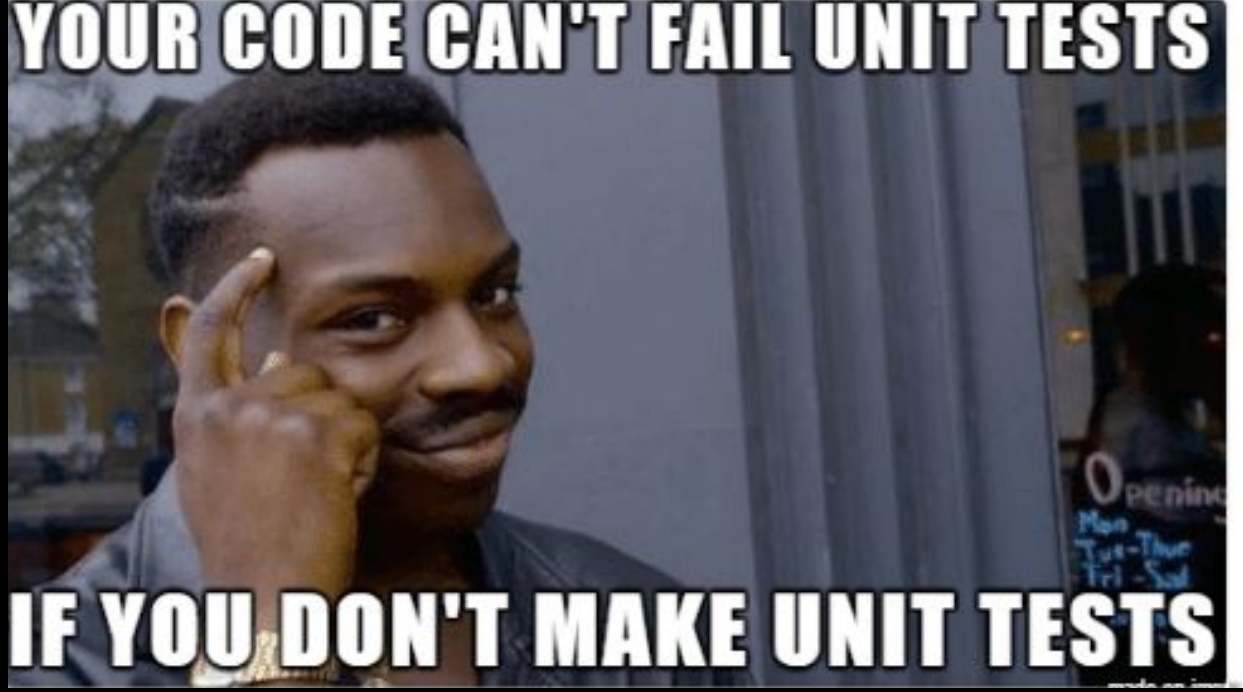
- Deprecated official approaches
  - dep
  - go get without using modules
    - the go command automatically chooses between using modules and using GOPATH depending on where it is run. If a go command runs outside GOPATH/src and there is a go.mod file in the current directory or one of its parent directories, then modules are enabled (otherwise not)
- Deprecated third party dependency management tools
  - glide
  - govendor
  - ...

Go 2

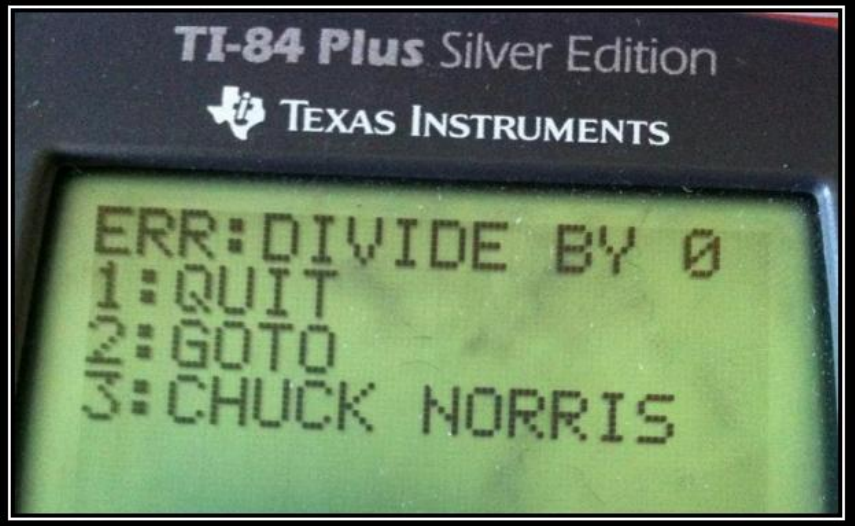
- Modules (depedency management)
  - As of Go 1.11 in beta stage.
- Generics
- Error handling

See https://blog.golang.org/go2draft for more details.

Testing



Error Handling



DIVIDE BY ZERO

YOUR ARGUMENT IS INVALID.

Concurrency

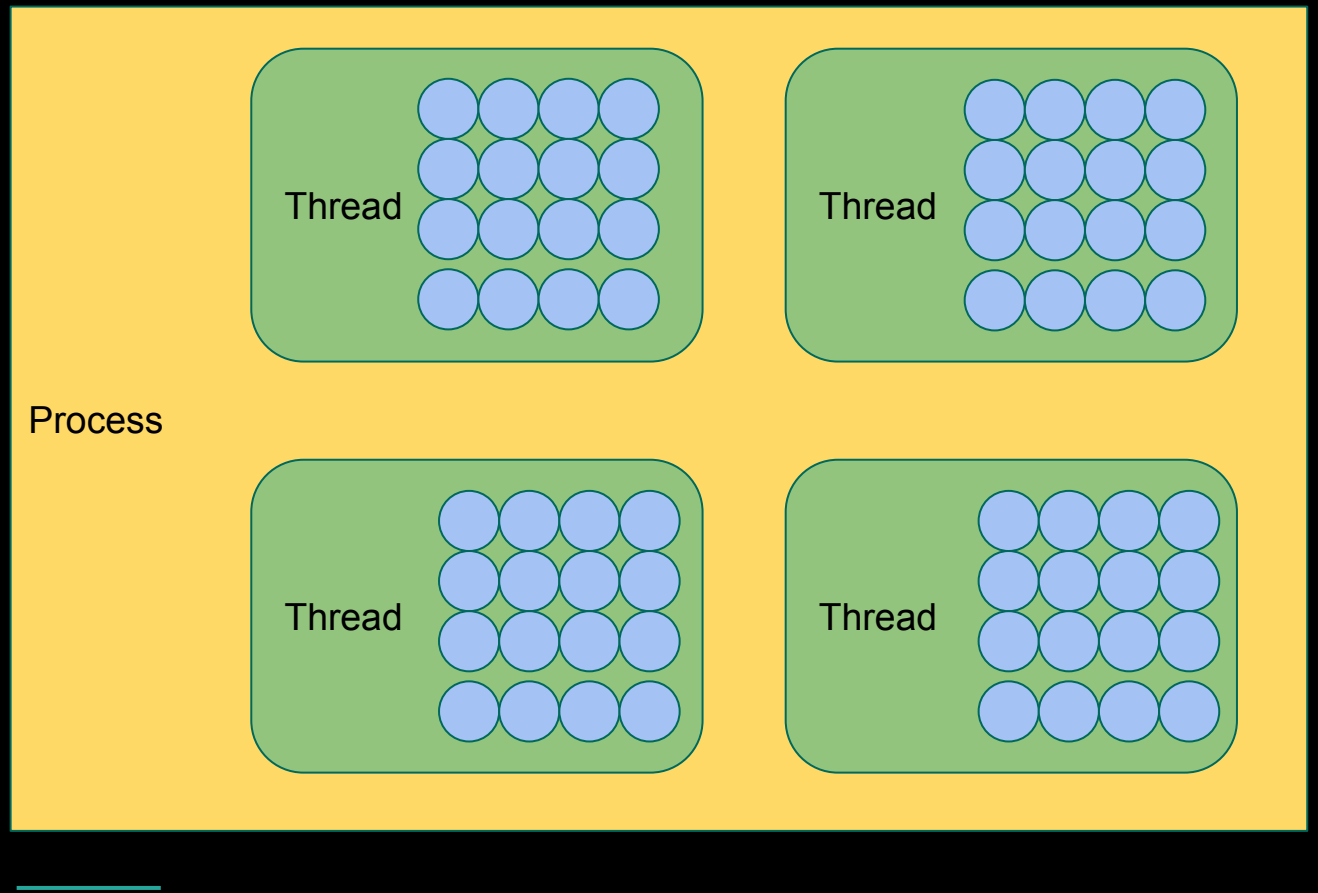
Concurrency

Go Routines

- A Go routine is a lightweight thread.
  - Memory footprint Go Routine ~ 4KiB
  - Memory footprint POSIX thread ~ 1 to 8 MiB
- The generic term is Co Routine.
- The Go scheduler can handle tenth of thousands of concurrent Go routines.

Concurrency

Go Routines



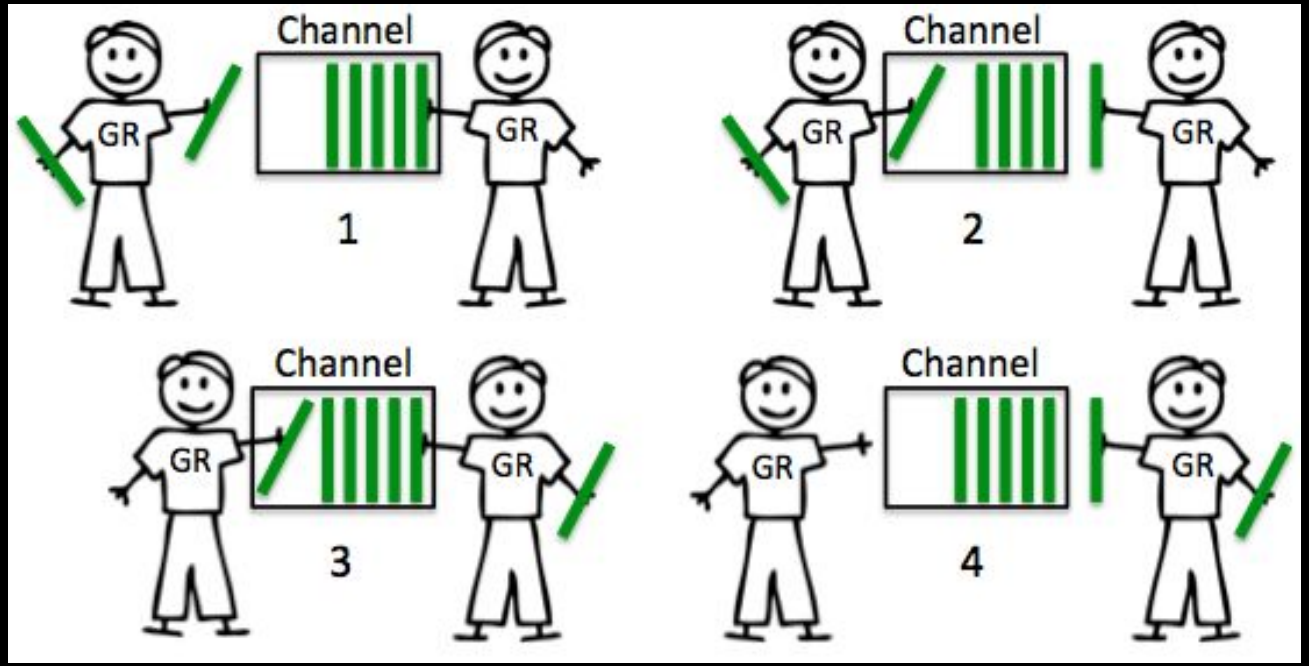
Concurrency

Channels

- Channels are a typed conduit through which you can send and receive values with the channel operator.
- By default, sends and receives block until the other side is ready. This allows goroutines to synchronize without explicit locks or condition variables.
- Do not communicate by sharing memory; instead, share memory by communicating. Only one goroutine has access to the value at any given time. Data races cannot occur, by design.

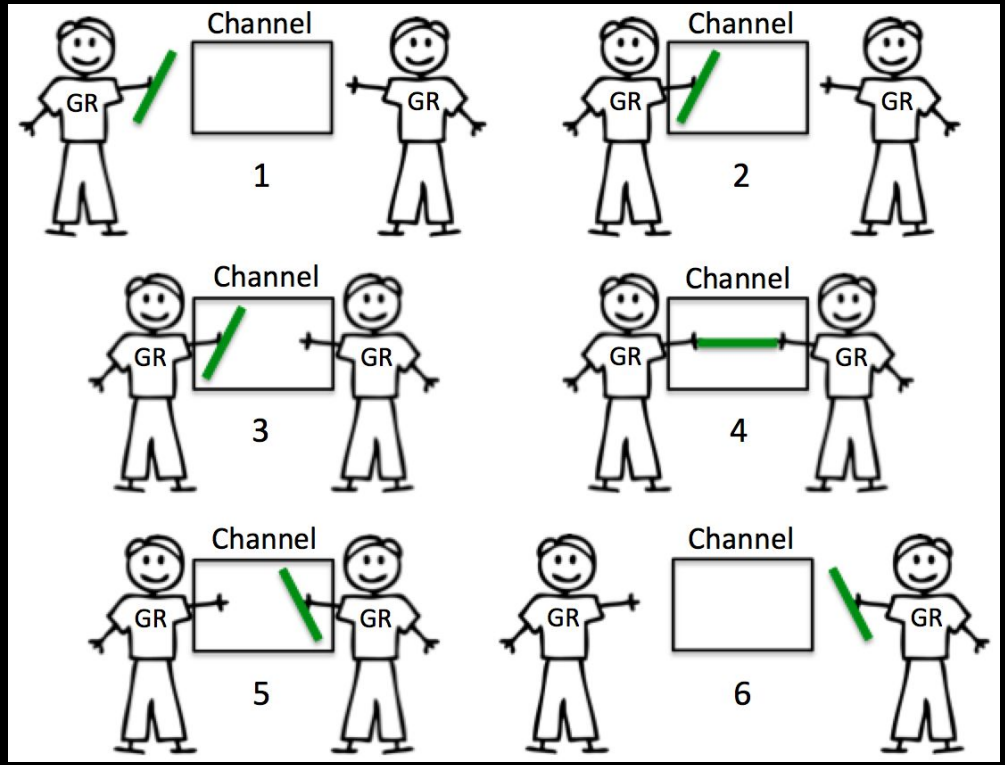
Concurrency

Buffered Channels



Concurrency

Unbuffered Channels



Patching the binary

Pointers

structs

Types

Concrete Types

- Concrete type (int8, int16, int32, int64, struct, slice, array, ...)
  - Explains what the layout of the type in the memory is.
  - You know what they look like in memory.
- Behaviour attached to data through methods

```
type Number int

func (n Number) Positive() bool {
  Return n>0
}
```

Types

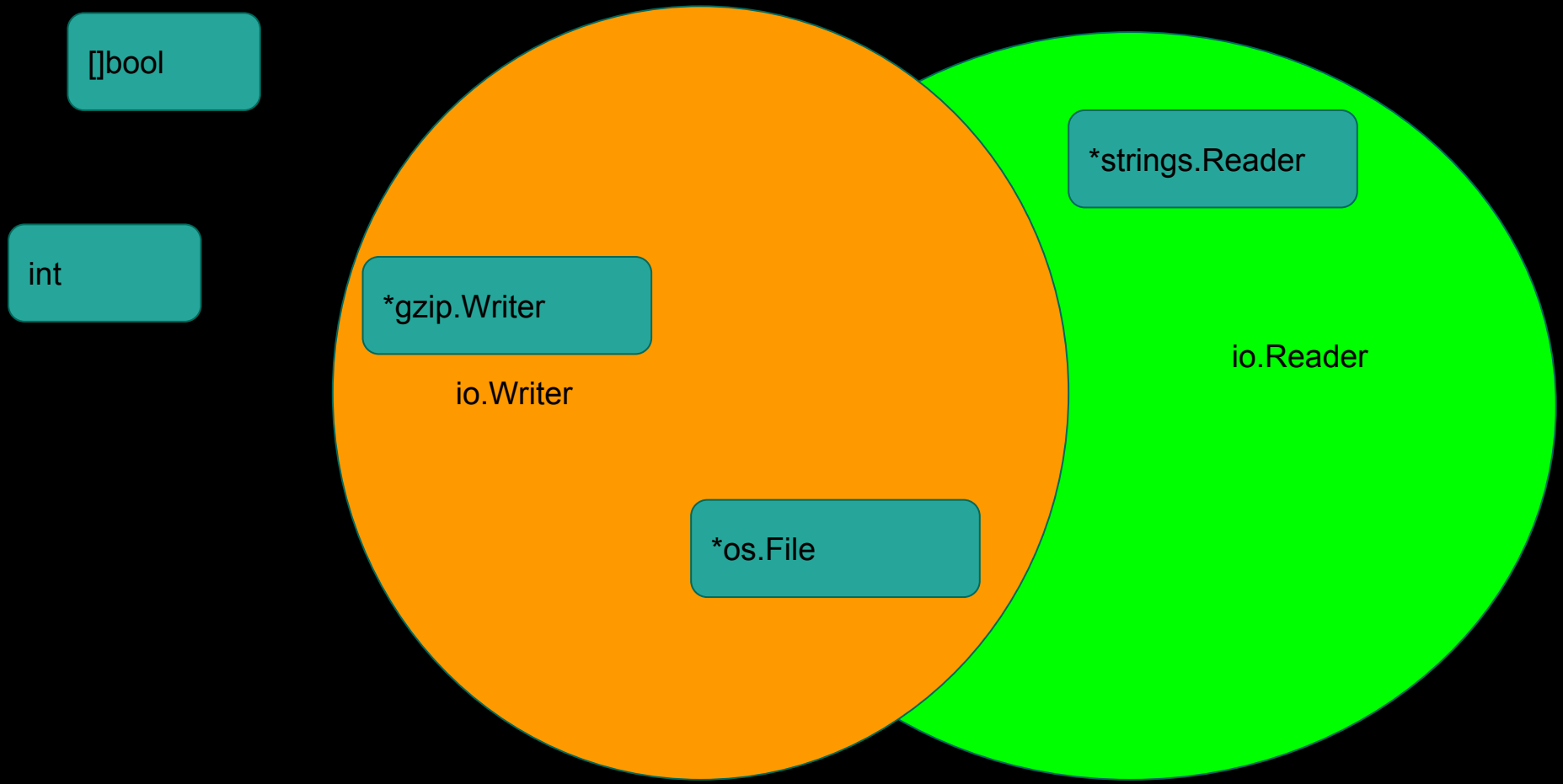
Abstract Types

- They describe behaviour (io.Reader, io.Writer, fmt.Stringer)
  - Behaviour in terms of possible values, possible operations on data of this type and the behaviour of these operations
- They define a set of methods, without specifying the receiver.
- **The only abstract type in Go is the interface.**
- They define a space in the area of concrete types.

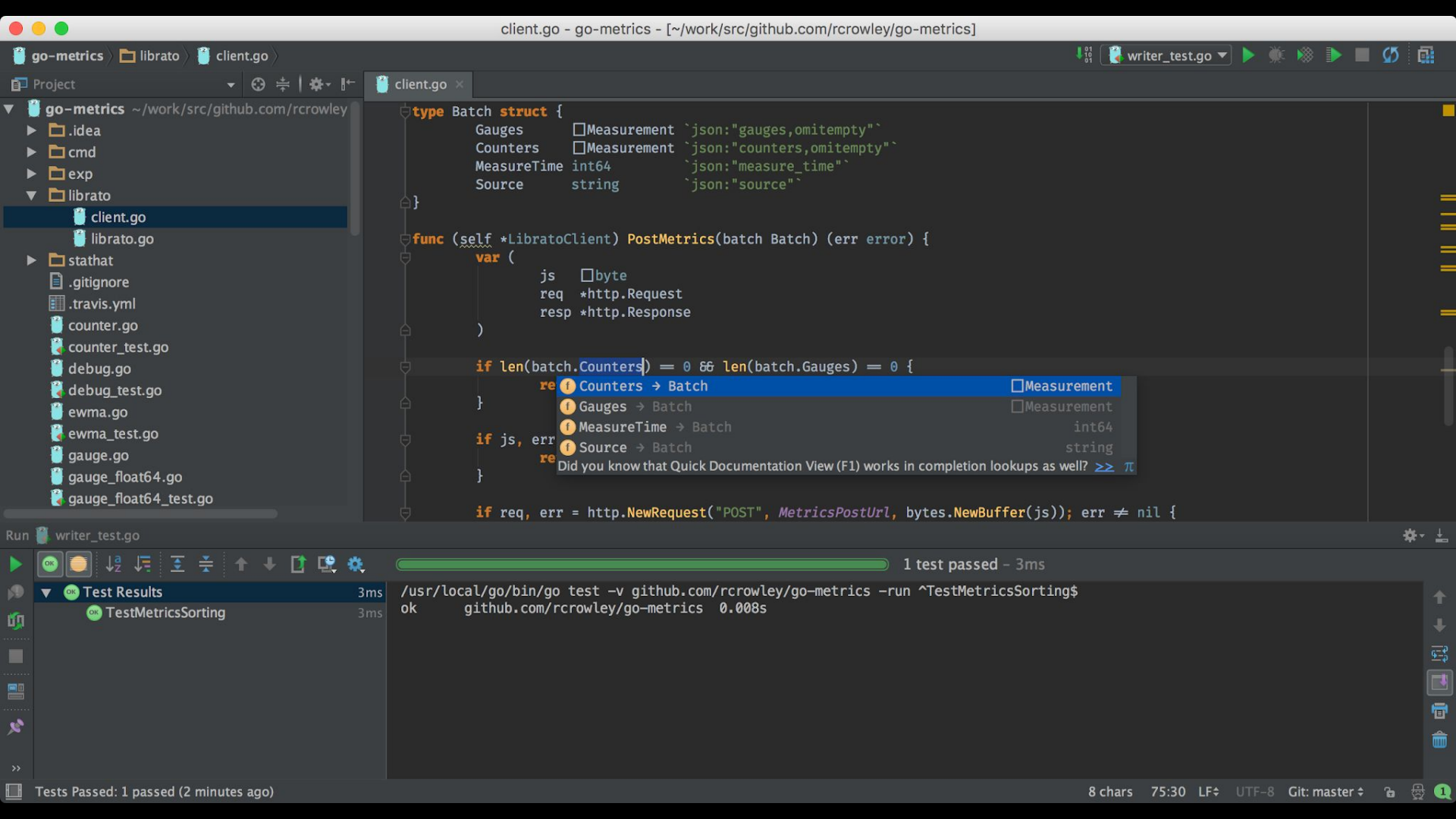
```
Type Positiver interface {
  Positive() bool
}
```

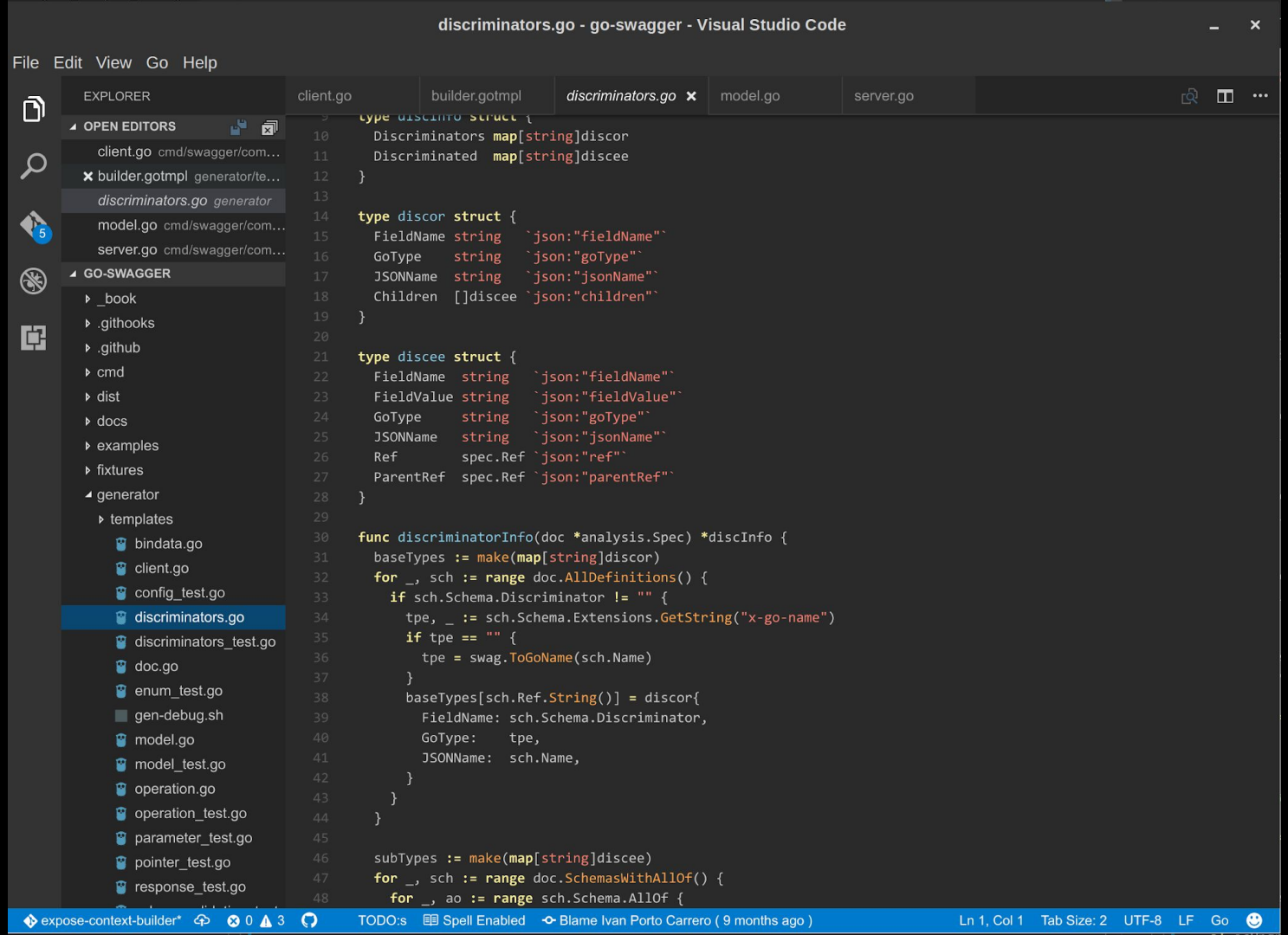
Interfaces

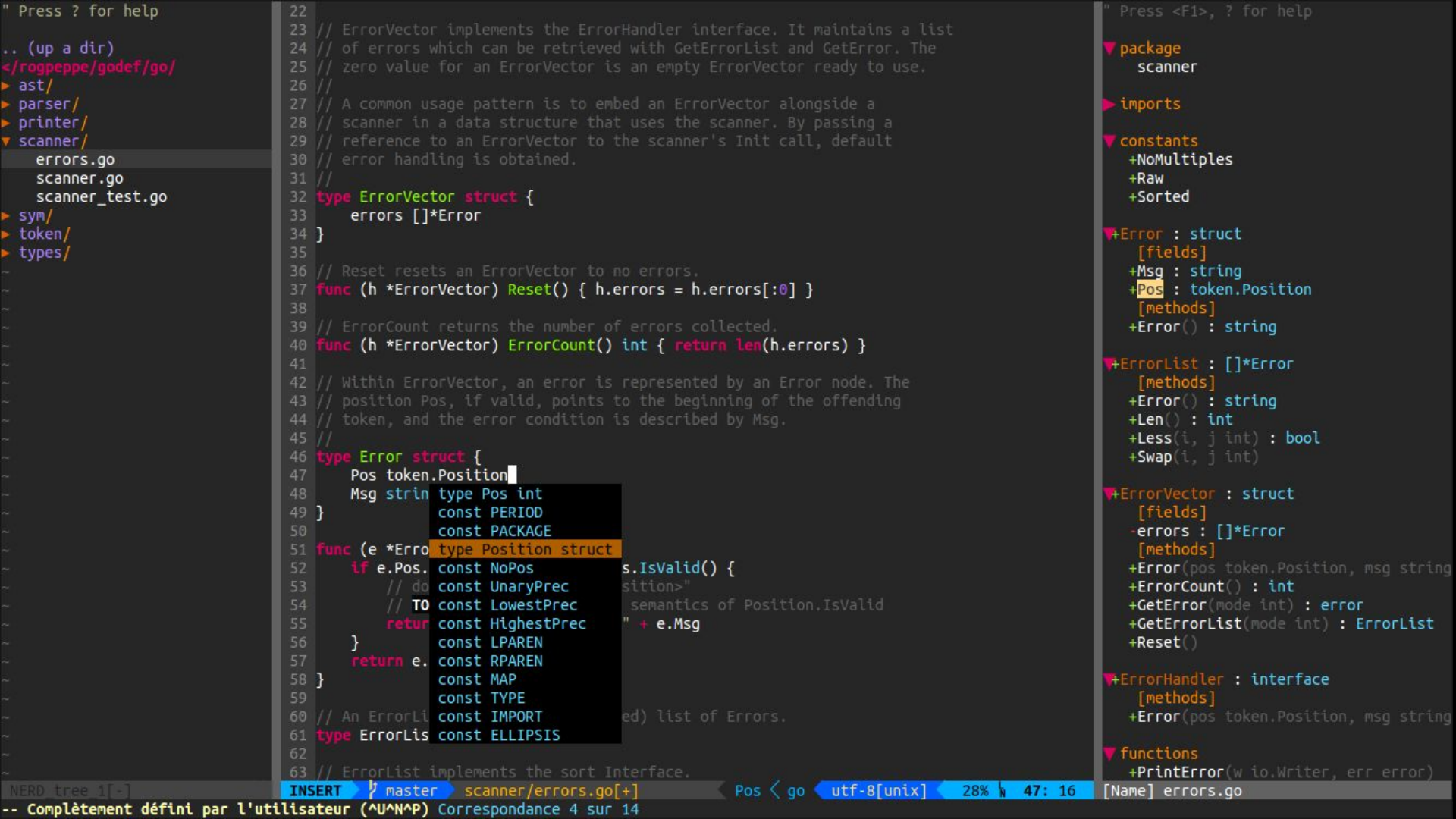
- Go uses implicit satisfaction when it comes to decide which types implement an interface.
  - No matter what type. If they have that method they are the one that I want.
- This is opposed to other languages like e.g. Java which use explicit satisfaction (see e.g. the interface keyword in Java).


[]bool
int
*gzip.Writer
io.Writer
*os.File
*strings.Reader
io.Reader

Editor Support


client.go - go-metrics - [~/work/src/github.com/rcrowley/go-metrics]
go-metrics
librato
client.go
writer_test.go
Project
client.go
go-metrics ~/work/src/github.com/rcrowley
.idea
cmd
exp
librato
client.go
librato.go
stathat
.gitignore
.travis.yml
counter.go
counter_test.go
debug.go
debug_test.go
ewma.go
ewma_test.go
gauge.go
gauge_float64.go
gauge_float64_test.go
type Batch struct {
Gauges Measurement `json:"gauges,omitempty"`
Counters Measurement `json:"counters,omitempty"`
MeasureTime int64 `json:"measure_time"`
Source string `json:"source"`
}
func (self *LibratoClient) PostMetrics(batch Batch) (err error) {
var (
js byte
req *http.Request
resp *http.Response
)
if len(batch.Counters) == 0 && len(batch.Gauges) == 0 {
re
}
if js, err
re
}
if req, err = http.NewRequest("POST", MetricsPostUrl, bytes.NewBuffer(js)); err != nil {
Counters → Batch Measurement
Gauges → Batch Measurement
MeasureTime → Batch int64
Source → Batch string
Did you know that Quick Documentation View (F1) works in completion lookups as well?
Run writer_test.go
1 test passed – 3ms
Test Results 3ms
TestMetricsSorting 3ms
/usr/local/go/bin/go test -v github.com/rcrowley/go-metrics -run ^TestMetricsSorting$
ok github.com/rcrowley/go-metrics 0.008s
Tests Passed: 1 passed (2 minutes ago)
8 chars 75:30 LF UTF-8 Git: master


discriminators.go - go-swagger - Visual Studio Code
File Edit View Go Help
EXPLORER
client.go
builder.gotmpl
discriminators.go
model.go
server.go
OPEN EDITORS
client.go cmd/swagger/com...
builder.gotmpl generator/te...
discriminators.go generator
model.go cmd/swagger/com...
server.go cmd/swagger/com...
GO-SWAGGER
_book
.githooks
.github
cmd
dist
docs
examples
fixtures
generator
templates
bindata.go
client.go
config_test.go
discriminators.go
discriminators_test.go
doc.go
enum_test.go
gen-debug.sh
model.go
model_test.go
operation.go
operation_test.go
parameter_test.go
pointer_test.go
response_test.go
type discInfo struct {
Discriminators map[string]discor
Discriminated map[string]discee
}
type discor struct {
FieldName string `json:"fieldName"`
GoType string `json:"goType"`
JSONName string `json:"jsonName"`
Children []discee `json:"children"`
}
type discee struct {
FieldName string `json:"fieldName"`
FieldValue string `json:"fieldValue"`
GoType string `json:"goType"`
JSONName string `json:"jsonName"`
Ref spec.Ref `json:"ref"`
ParentRef spec.Ref `json:"parentRef"`
}
func discriminatorInfo(doc *analysis.Spec) *discInfo {
baseTypes := make(map[string]discor)
for _, sch := range doc.AllDefinitions() {
if sch.Schema.Discriminator != "" {
tpe, _ := sch.Schema.Extensions.GetString("x-go-name")
if tpe == "" {
tpe = swag.ToGoName(sch.Name)
}
baseTypes[sch.Ref.String()] = discor{
FieldName: sch.Schema.Discriminator,
GoType: tpe,
JSONName: sch.Name,
}
}
}
subTypes := make(map[string]discee)
for _, sch := range doc.SchemasWithAllOf() {
for _, ao := range sch.Schema.AllOf {
expose-context-builder*
0 3
TODO:s
Spell Enabled
Blame Ivan Porto Carrero (9 months ago)
Ln 1, Col 1
Tab Size: 2
UTF-8
LF
Go


" Press ? for help
.. (up a dir)
</rogpeppe/godef/go/
ast/
parser/
printer/
scanner/
errors.go
scanner.go
scanner_test.go
sym/
token/
types/
// ErrorVector implements the ErrorHandler interface. It maintains a list
// of errors which can be retrieved with GetErrorList and GetError. The
// zero value for an ErrorVector is an empty ErrorVector ready to use.
// A common usage pattern is to embed an ErrorVector alongside a
// scanner in a data structure that uses the scanner. By passing a
// reference to an ErrorVector to the scanner's Init call, default
// error handling is obtained.
type ErrorVector struct {
errors []*Error
}
// Reset resets an ErrorVector to no errors.
func (h *ErrorVector) Reset() { h.errors = h.errors[:0] }
// ErrorCount returns the number of errors collected.
func (h *ErrorVector) ErrorCount() int { return len(h.errors) }
// Within ErrorVector, an error is represented by an Error node. The
// position Pos, if valid, points to the beginning of the offending
// token, and the error condition is described by Msg.
type Error struct {
Pos token.Position
Msg strin
}
func (e *Erro
if e.Pos.
// do
// TO
retur
}
return e.
}
// An ErrorLi
type ErrorLis
// ErrorList implements the sort Interface.
type Type int
const PERIOD
const PACKAGE
type Position struct
const NoPos
const UnaryPrec
const LowestPrec
const HighestPrec
const LPAREN
const RPAREN
const MAP
const TYPE
const IMPORT
const ELLIPSIS
s.IsValid() {
sition>"
semantics of Position.IsValid
" + e.Msg
ed) list of Errors.
NERD_tree_1[-]
INSERT
master
scanner/errors.go[+]
Pos
go
utf-8[unix]
28%
47: 16
-- Complètement défini par l'utilisateur (^U^N^P) Correspondance 4 sur 14
" Press <F1>, ? for help
package
scanner
imports
constants
+NoMultiples
+Raw
+Sorted
+Error : struct
[fields]
+Msg : string
+Pos : token.Position
[methods]
+Error() : string
+ErrorList : []*Error
[methods]
+Error() : string
+Len() : int
+Less(i, j int) : bool
+Swap(i, j int)
+ErrorVector : struct
[fields]
-errors : []*Error
[methods]
+Error(pos token.Position, msg string
+ErrorCount() : int
+GetError(mode int) : error
+GetErrorList(mode int) : ErrorList
+Reset()
+ErrorHandler : interface
[methods]
+Error(pos token.Position, msg string
functions
+PrintError(w io.Writer, err error)
[Name] errors.go

DELVE

A Debugger for the Go Programming Language


Microsoft


moz://a


NETFLIX


DIGITAL CITY
SOLUTIONS


Google

The End