Recipes for reducing cognitive load

Yet another talk about idiomatic Go

Federico Paolinelli - Red Hat

Why this talk?

MetalLB

MetalLB is a load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols (github.com/metallb/metallb)

- 5k stars on github
- ~ 600 PRs since I started maintaining the project
- ~ 40k LOC

About me

Telco Network Team @ Red Hat

Contributed to:

- Athens
- KubeVirt
- SR-IOV Network Operator
- OPA Gatekeeper
- OVN-Kubernetes
- CNI Plugins
- MetalLB

hachyderm.io/@fedepaol

@fedepaol

fedepaol@gmail.com

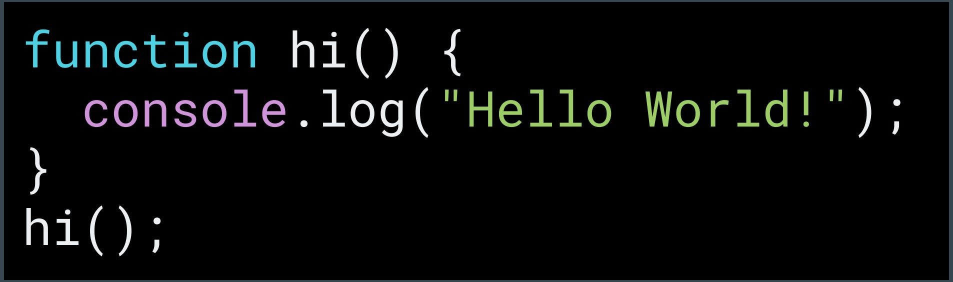
Cognitive Load

Cognitive Load

“Cognitive load refers to the amount of effort that is exerted or required while reasoning and thinking. Any mental process, from memory to perception to language, creates a cognitive load because it requires energy and effort. When cognitive load is high, thought processes are potentially interfered with. To the UX designer, a common goal when designing interfaces would be to keep users’ cognitive load to a minimum.”.

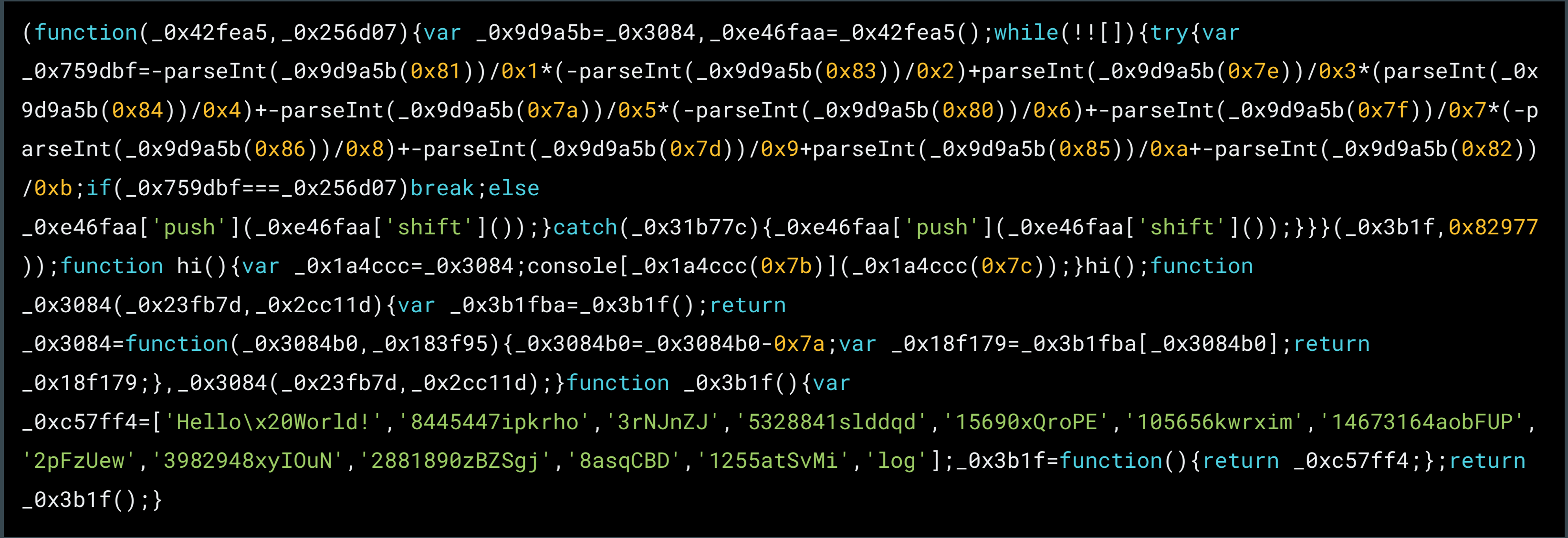
Wikipedia

Cognitive Load



```
function hi() {
  console.log("Hello World!");
}
hi();
```

Cognitive Load



```
(function(_0x42fea5,_0x256d07){var _0x9d9a5b=_0x3084,_0xe46faa=_0x42fea5();while(!![]){try{var
_0x759dbf=-parseInt(_0x9d9a5b(0x81))/0x1*(-parseInt(_0x9d9a5b(0x83))/0x2)+parseInt(_0x9d9a5b(0x7e))/0x3*(parseInt(_0x
9d9a5b(0x84))/0x4)+-parseInt(_0x9d9a5b(0x7a))/0x5*(-parseInt(_0x9d9a5b(0x80))/0x6)+-parseInt(_0x9d9a5b(0x7f))/0x7*(-p
arseInt(_0x9d9a5b(0x86))/0x8)+-parseInt(_0x9d9a5b(0x7d))/0x9+parseInt(_0x9d9a5b(0x85))/0xa+-parseInt(_0x9d9a5b(0x82))
/0xb;if(_0x759dbf===_0x256d07)break;else
_0xe46faa['push'](_0xe46faa['shift']());}catch(_0x31b77c){_0xe46faa['push'](_0xe46faa['shift']());}}}(_0x3b1f,0x82977
));function hi(){var _0x1a4ccc=_0x3084;console[_0x1a4ccc(0x7b)](_0x1a4ccc(0x7c));}hi();function
_0x3084(_0x23fb7d,_0x2cc11d){var _0x3b1fba=_0x3b1f();return
_0x3084=function(_0x3084b0,_0x183f95){_0x3084b0=_0x3084b0-0x7a;var _0x18f179=_0x3b1fba[_0x3084b0];return
_0x18f179;},_0x3084(_0x23fb7d,_0x2cc11d);}function _0x3b1f(){var
_0xc57ff4=['Hello\x20World!','8445447ipkrho','3rNJnZJ','5328841slddqd','15690xQroPE','105656kwrxim','14673164aobFUP',
'2pFzUew','3982948xyIOuN','2881890zBZSgj','8asqCBD','1255atSvMi','log'];_0x3b1f=function(){return _0xc57ff4;};return
_0x3b1f();}
```

https://flic.kr/p/2n5YeoS

Let’s see the recipes

Disclaimer!

https://flic.kr/p/2kzvoG

The two sides of readability

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

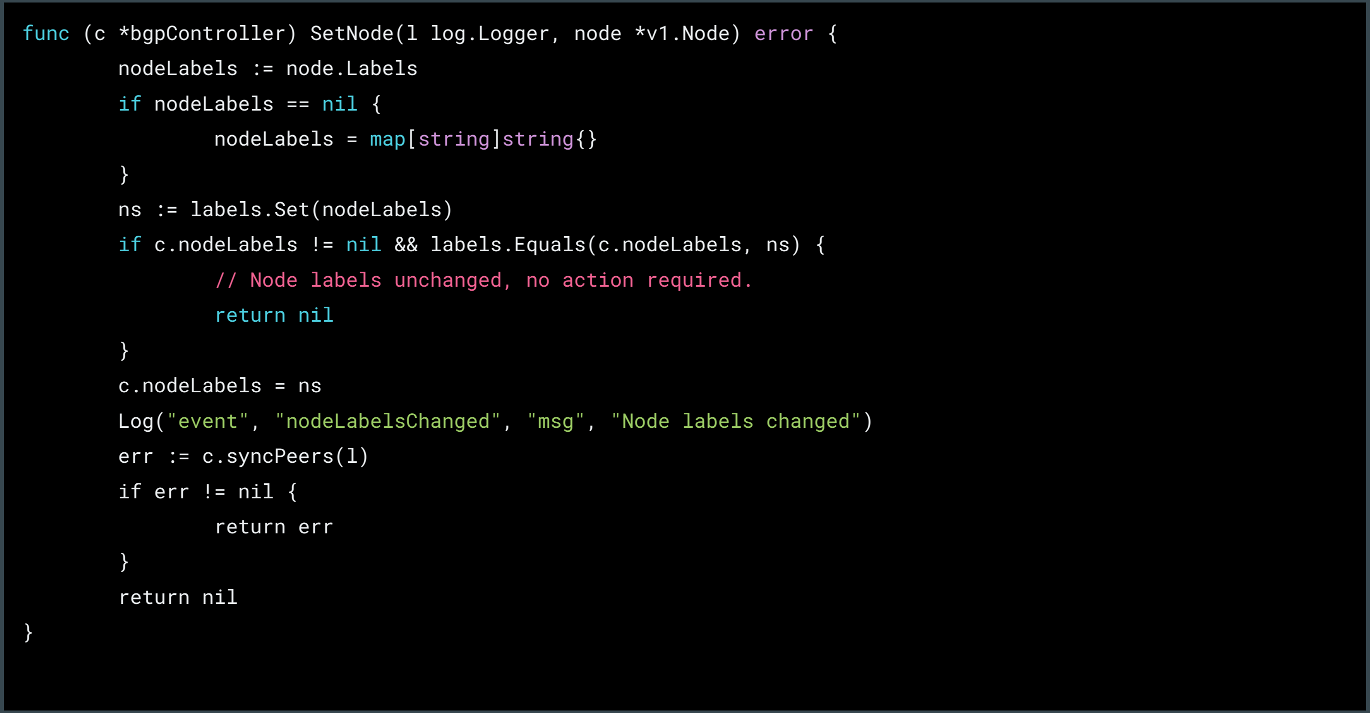
```go
func yyyy(a, b int) int {
    return xxxx(a, b) + 1
}
```

```go
func yyyy(a, b int) int {
    return sum(a, b) + 1
}
```

https://flic.kr/p/XxeMM2

Line of Sight

Line of Sight

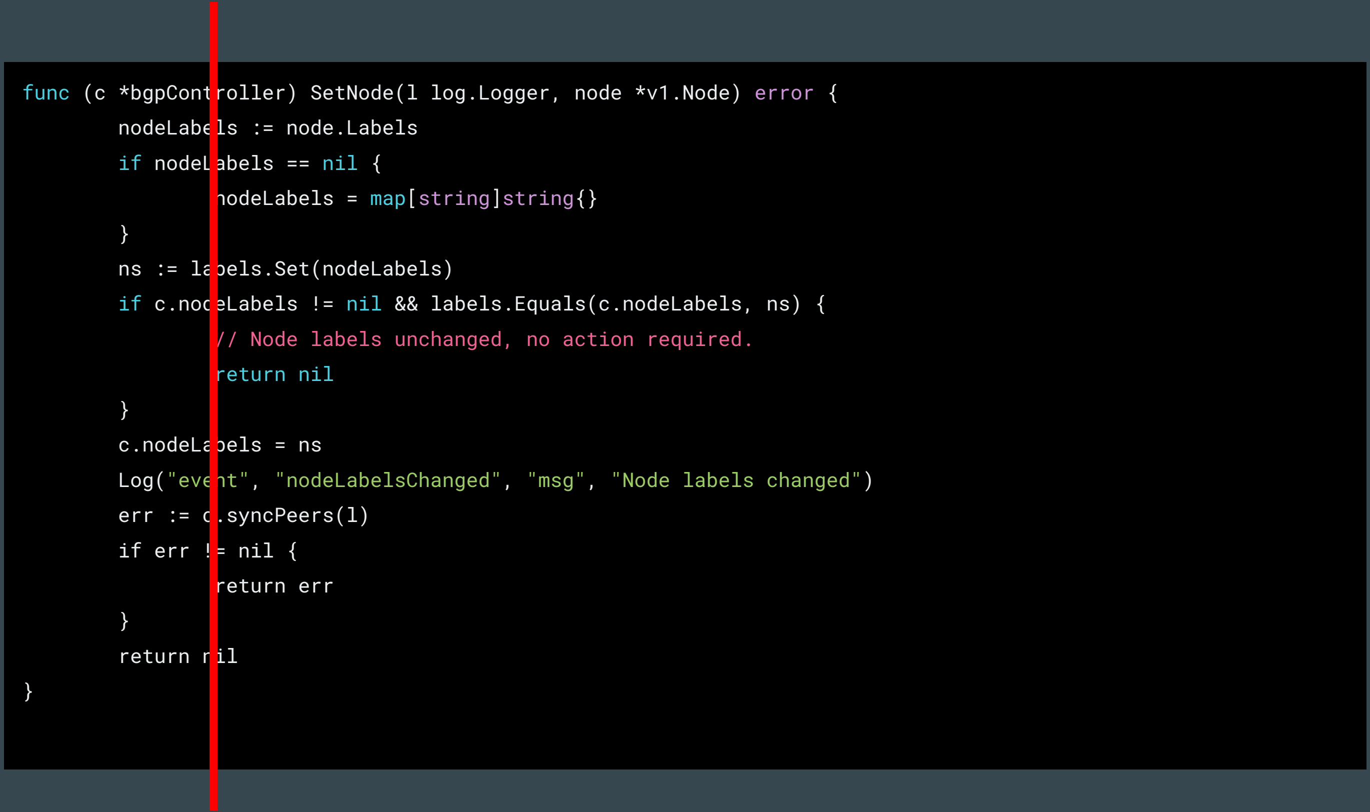


```go
func (c *bgpController) SetNode(l log.Logger, node *v1.Node) error {
        nodeLabels := node.Labels
        if nodeLabels == nil {
                nodeLabels = map[string]string{}
        }
        ns := labels.Set(nodeLabels)
        if c.nodeLabels != nil && labels.Equals(c.nodeLabels, ns) {
                // Node labels unchanged, no action required.
                return nil
        }
        c.nodeLabels = ns
        Log("event", "nodeLabelsChanged", "msg", "Node labels changed")
        err := c.syncPeers(l)
        if err != nil {
                return err
        }
        return nil
}
```

Line of Sight

```
func (c *bgpController) SetNode(l log.Logger, node *v1.Node) error {
        nodeLabels := node.Labels
        if nodeLabels == nil {
                nodeLabels = map[string]string{}
        }
        ns := labels.Set(nodeLabels)
        if c.nodeLabels != nil && labels.Equals(c.nodeLabels, ns) {
                // Node labels unchanged, no action required.
                return nil
        }
        c.nodeLabels = ns
        Log("event", "nodeLabelsChanged", "msg", "Node labels changed")
        err := c.syncPeers(l)
        if err != nil {
                return err
        }
        return nil
}
```

Line of Sight



```go
func (c *bgpController) SetNode(l log.Logger, node *v1.Node) error {
        nodeLabels := node.Labels
        if nodeLabels == nil {
                nodeLabels = map[string]string{}
        }
        ns := labels.Set(nodeLabels)
        if c.nodeLabels != nil && labels.Equals(c.nodeLabels, ns) {
                // Node labels unchanged, no action required.
                return nil
        }
        c.nodeLabels = ns
        Log("event", "nodeLabelsChanged", "msg", "Node labels changed")
        err := c.syncPeers(l)
        if err != nil {
                return err
        }
        return nil
}
```

Line of sight

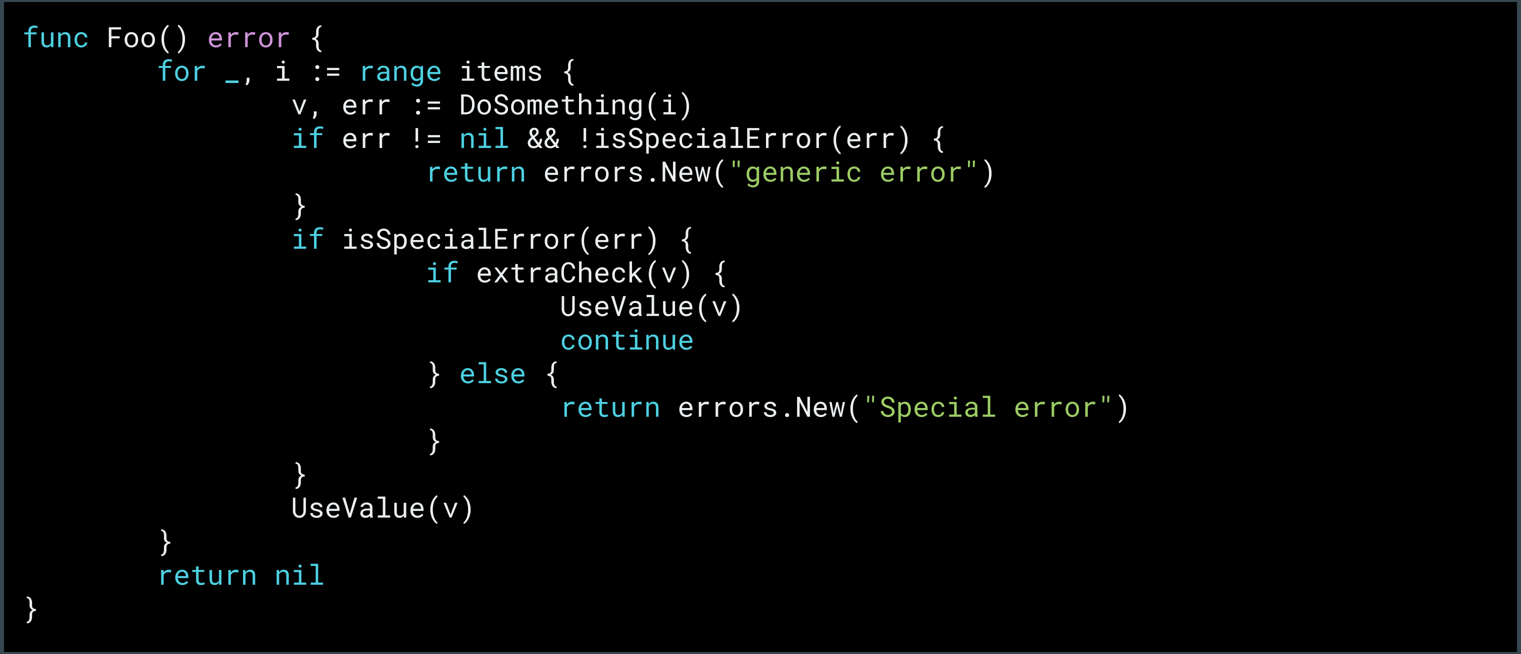
- Try to eliminate elses
- Return early
- Avoid extra nesting
- Wrap in functions

An Example:

```go
func Foo() error {
        for _, i := range items {
                v, err := DoSomething(i)
                if err != nil {
                        if strings.Contains(err.Error(), "special case") {
                                if extraCheck(v) {
                                        UseValue(v)
                                        continue
                                } else {
                                        return errors.New("Special error")
                                }
                        } else {
                                return errors.New("generic error")
                        }
                }
                UseValue(v)
        }
        return nil
}
```

Align to the left

Flip errors and return early



```go
func Foo() error {
        for _, i := range items {
                v, err := DoSomething(i)
                if err != nil && !isSpecialError(err) {
                        return errors.New("generic error")
                }
                if isSpecialError(err) {
                        if extraCheck(v) {
                                UseValue(v)
                                continue
                        } else {
                                return errors.New("Special error")
                        }
                }
                UseValue(v)
        }
        return nil
}
```

Align to the left

Prioritize return vs elses (avoid elses in general)

```go
func Foo() error {
        for _, i := range items {
                v, err := DoSomething(i)
                if err != nil && !isSpecialError(err) {
                        return errors.New("generic error")
                }
                if isSpecialError(err) {
                        if extraCheck(v) {
                                UseValue(v)
                                continue
                        }
                        return errors.New("Special error")
                }
                UseValue(v)
        }
        return nil
}
```

Align to the left

Consider wrapping in a function to leverage more returns

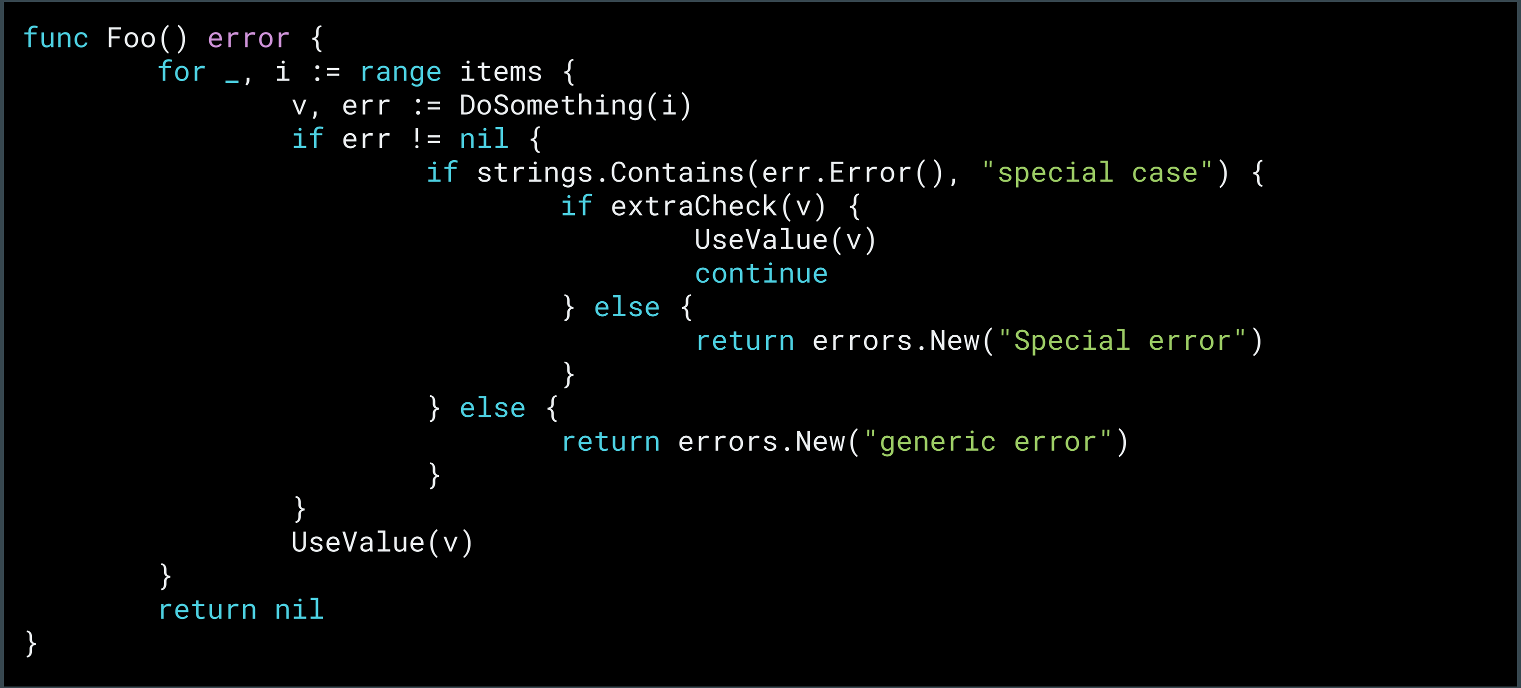
```go
func HandleItem(i Item) error {
        v, err := DoSomething(i)
        if err != nil && !isSpecialError(err) {
                return errors.New("generic error")
        }
        if isSpecialError(err) {
                if !extraCheck(v) {
                        return errors.New("Special error")
                }
        }
        UseValue(v)
        return nil
}
```

Align to the left

And leverage more returns

```go
func HandleItem(i Item) error {
        v, err := DoSomething(i)
        if err != nil && !isSpecialError(err) {
                return errors.New("generic error")
        }
        if isSpecialError(err) && !extraCheck(v) {
                return errors.New("Special error")
        }
        UseValue(v)
        return nil
}
```

Align to the left



```
func Foo() error {
        for _, i := range items {
                v, err := DoSomething(i)
                if err != nil {
                        if strings.Contains(err.Error(), "special case") {
                                if extraCheck(v) {
                                        UseValue(v)
                                        continue
                                } else {
                                        return errors.New("Special error")
                                }
                        } else {
                                return errors.New("generic error")
                        }
                }
                UseValue(v)
        }
        return nil
}
```

Line of sight

Tips for a good line of sight:

- *Align the happy path to the left; you should quickly be able to scan down one column to see the expected execution flow*
- *Don't hide happy path logic inside a nest of indented braces*
- *Exit early from your function*
- *Avoid else returns; consider flipping the if statement*
- *Put the happy return statement as the very last line*
- *Extract functions and methods to keep bodies small and readable*
- *If you need big indented bodies, consider giving them their own function*

(from medium.com/@matryer/line-of-sight-in-code-186dd7cdea88)

https://flic.kr/p/8QPvk2

Package Names

Package Names

“There are only two hard things in Computer Science: cache invalidation and naming things.”

Phil Karlton

Package Names

Writing a good Go package starts with its name. Think of your package's name as an elevator pitch, you have to describe what it does using just one word.

(from dave.cheney.net/2019/01/08/avoid-package-names-like-base-util-or-common)

Package Names

Writing a good Go package starts with its name. Think of your package’s name as an elevator pitch, you have to describe what it does using just one word.

(from dave.cheney.net/2019/01/08/avoid-package-names-like-base-util-or-common)

A package name and its contents' names are coupled, since client code uses them together

(from go.dev/blog/package-names)

The package is part of the name

```go
package util


func CopyNode() *Node {
        ...
}
```

Caller

```go
n := util.CopyNode()
```

The package is part of the name

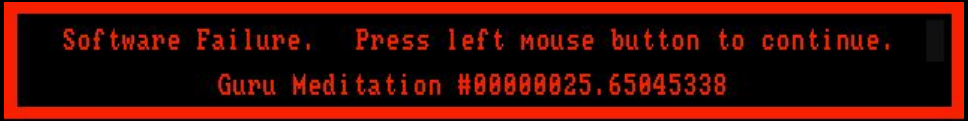
```go
package node


func Copy() *Node {
        ...
}
```

Caller

```go
n := node.Copy()
```

Util / common package name should be avoided

Avoid meaningless package names. Packages named util, common, or misc provide clients with no sense of what the package contains.
(from go.dev/blog/package-names)



```
Software Failure.   Press left mouse button to continue.
        Guru Meditation #00000025.65045338
```

Errors are types

The most frequent way

```
if strings.Contains(err.Error(), "special case") {

}
```

Asserting errors from Go 1.13

```
var ErrNotFound = errors.New("not found")

if errors.Is(err, ErrNotFound) {
    // something wasn't found
}
```

Asserting errors from Go 1.13

```
type NotFoundError struct {
    Name string
}

func (e *NotFoundError) Error() string { return
e.Name + ": not found" }


var e *QueryError
if errors.As(err, &e) {

}
```

In the simplest case, the errors.Is function behaves like a comparison to a sentinel error, and the errors.As function behaves like a type assertion. When operating on wrapped errors, however, these functions consider all the errors in a chain.
(from go.dev/blog/go1.13-errors)

Wrapping Errors

```
func Foo() error {
        err := FuncThatReturnsErrNotFound()
        if err != nil {
                return errors.Wrap(err, "Foo failed”)
        }
}


err := Foo()

if errors.Is(err, ErrNotFound) {

    ...

}
```

Wrapping Errors

```
func Foo() error {
        err := FuncThatReturnsErrNotFound()
        if err != nil {
                return fmt.Errorf("Foo failed for %w”, err)
        }
}


err := Foo()
if errors.Is(err, ErrNotFound) {
    ...
}
```

https://flic.kr/p/qannbe

Pure Functions

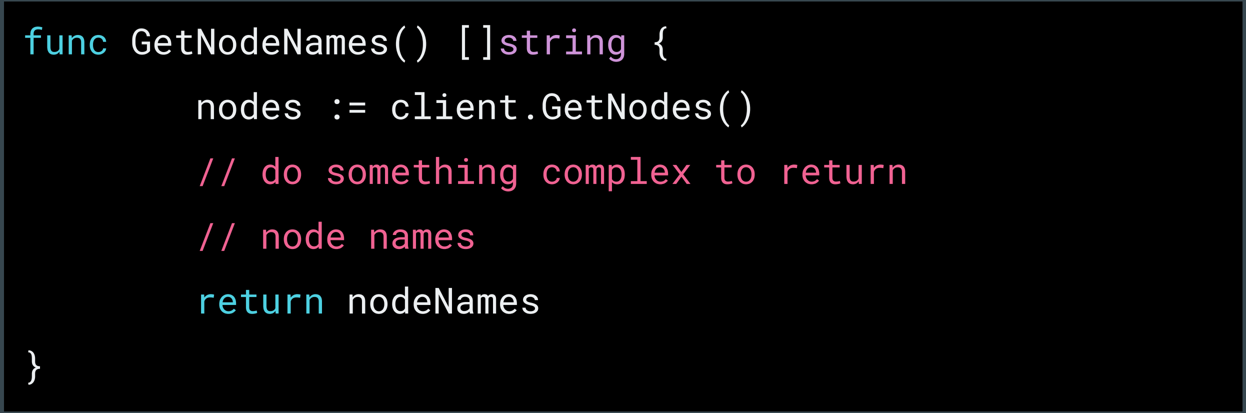
Pure Functions

In computer programming, a pure function is a function that has the following properties:[1][2]

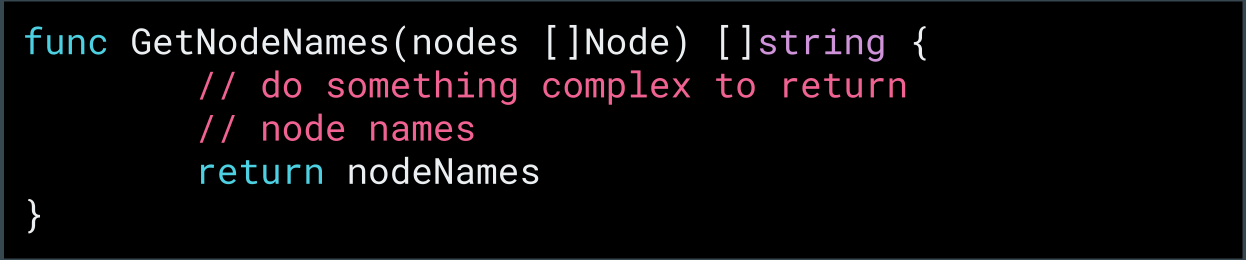
- *the function return values are identical for identical arguments (no variation with local static variables, non-local variables, mutable reference arguments or input streams), and*
- *the function application has no side effects (no mutation of local static variables, non-local variables, mutable reference arguments or input/output streams).*

(from en.wikipedia.org/wiki/Pure_function)

Independent of the state



```go
func GetNodeNames() []string {
        nodes := client.GetNodes()
        // do something complex to return
        // node names
        return nodeNames
}
```



```go
func GetNodeNames(nodes []Node) []string {
        // do something complex to return
        // node names
        return nodeNames
}
```

No side effects

```
func CheckNode(n *Node) error {
    // check other stuff...
    if n.Name == "" {
        n.Name = "unknown"
    }
}

err := CheckNode(&n)
```

No side effects

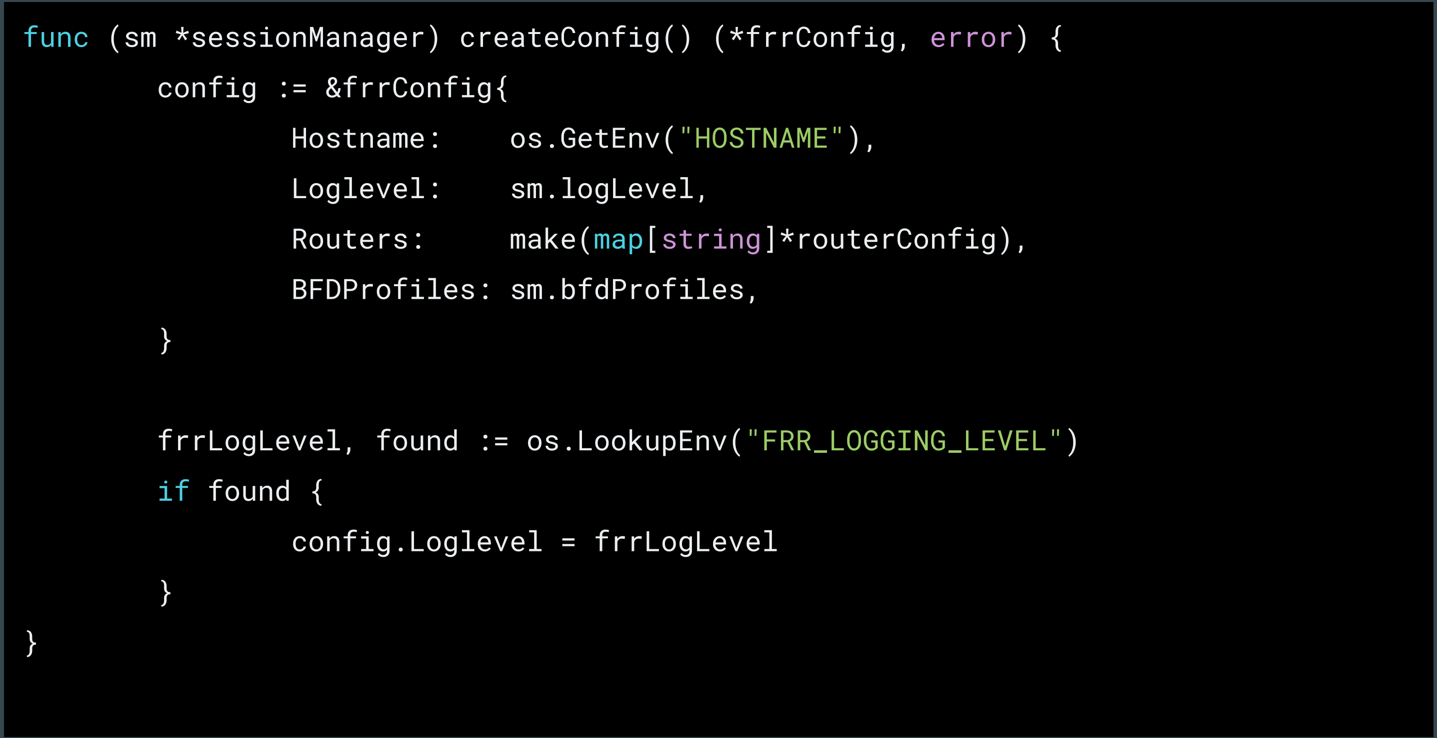
```
func CheckNodeAndSetName(n *Node) error {
    // check other stuff...
    if n.Name == "" {
        n.Name = "unknown"
    }
}

err := CheckNodeAndSetName(&n)
```

No side effects

```
err := CheckNode(n)
if n.Name == “” {
    n.Name = "unknown"
}
```

A note about environment variables

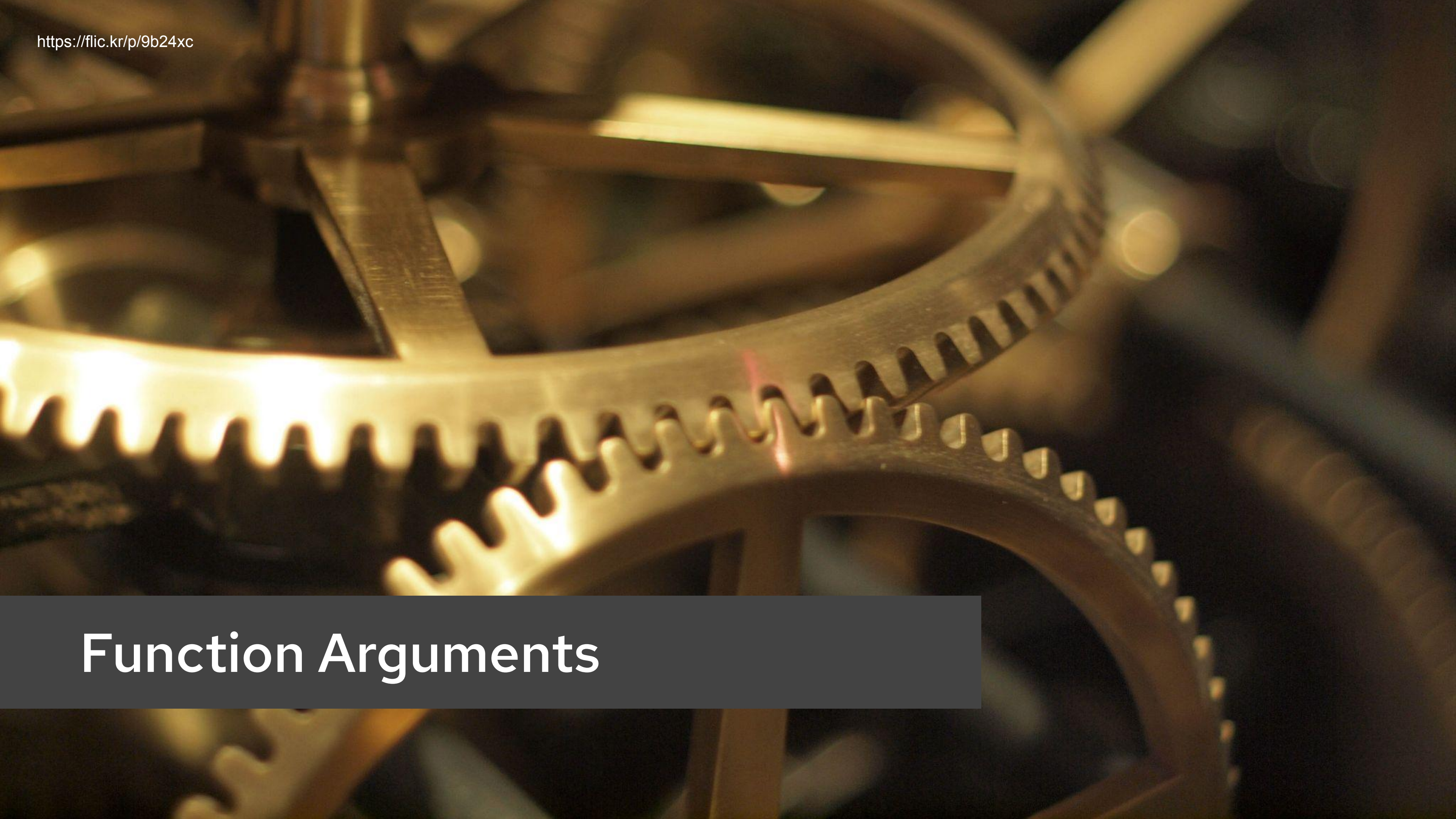
Reading environment variables



```
func (sm *sessionManager) createConfig() (*frrConfig, error) {
        config := &frrConfig{
                Hostname:    os.GetEnv("HOSTNAME"),
                Loglevel:    sm.logLevel,
                Routers:     make(map[string]*routerConfig),
                BFDProfiles: sm.bfdProfiles,
        }

        frrLogLevel, found := os.LookupEnv("FRR_LOGGING_LEVEL")
        if found {
                config.Loglevel = frrLogLevel
        }
}
```

Reading environment variables

- It’s hard to track all the parameters accepted by an executable
- It’s hard to understand what influences the behavior of a function from the calling site



https://flic.kr/p/9b24xc

Function Arguments

The mysterious booleans

The mysterious booleans

```go
func (r *Controller) Setup(name string) error {}
```

The mysterious booleans

```go
func (r *Controller) Setup(name string,
                           enableWebhook bool) error {}
```

The mysterious booleans

```
func (r *Controller) Setup(name string,
                           enableWebhook,
                           enableDeployment bool) error {}
```

The mysterious booleans

```go
func (r *Controller) Setup(name string,
                           enableWebhook,
                           enableDeployment,
                           resetState bool) error {}
```

The mysterious booleans

```
c.Setup("first", true, false, true)


c.Setup("second", false, true, true)
```

The mysterious booleans

```
const (
    WebhookDisabled = false
    WebhookEnabled  = true
)

c.Setup("foo", WebhookEnabled, DeploymentDisabled, ResetState)
```

Function Overloading

(or the lack of)

Function Overloading

```
func CreateService(name string) Service {}
```

Function Overloading

```go
func CreateService(name string) Service {}


func CreateServiceWithBackend(name string, backend Backend) Service {}
```

Function Overloading

```go
func CreateService(name string) Service {}


func CreateServiceWithBackend(name string, backend Backend) Service {}


func CreateServiceWithIP(name string, ip net.IP) Service {}
```

Function Overloading

```
func CreateService(name string) Service {}


func CreateServiceWithBackend(name string, backend Backend) Service {}


func CreateServiceWithIP(name string, ip net.IP) Service {}


func CreateServiceIPBackend(name string, backend Backend, ip net.IP) Service {}
```

Functional Options to the rescue!

```
func CreateService(name string, options ...func(*Service)) Service {
        res := Service{} // something more meaningful
        for _, o := range options {
                o(&res)
        }
}
```

Functional Options to the rescue!

```
func CreateService(name string, options ...func(*Service)) Service {
        res := Service{} // something more meaningful
        for _, o := range options {
                o(&res)
        }
}


func main() {
     CreateService("foo", func(s *Service){
        s.Backend = b
        s.IP = ip
     })
}
```

Functional Options to the rescue!

```go
func WithBackend(b Backend) func(*Service) {
        return func(s *Service) {
                s.Backend = b
        }
}

func main() {
     CreateService("foo", WithBackend(b), WithIP(ip))
}
```

https://dave.cheney.net/2014/10/17/functional-options-for-friendly-apis

Methods that can be functions

Methods that can be functions

```
func (c *Controller) SumTwoNumbers(a, b int) int {
        return a + b
}

x := c.SumTwoNumbers(2, 3)
```

Methods that can be functions

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

x := SumTwoNumbers(2, 3)
```

Pointers

ALPE TRE POTENZE
LAGO NERO
100-00
VAL DI LUCE
CA' COPPI
DOGANA
505

Pointers

```
err := DoSomethingWithNode(n)


err := DoSomethingElseWithNode(&n)
```

Pointers - Exception!

```
n := sync.Mutex{}
err := DoSomethingMutex(n)

err := DoSomethingElseMutex(&n)
```

Pointers - Exception!

```
~~err := DoSomethingMutex(n)~~


err := DoSomethingElseMutex(&n)
```

*In general, do not copy a value of type T if its methods are associated with the pointer type, *T.*

https://github.com/golang/go/wiki/CodeReviewComments#copying

How about performance?

How about performance?

Optimize for readability, not performance

and use Go tooling to measure performance bottlenecks

https://flic.kr/p/b4HUXX

The code should read like a newspaper

“Think of a well-written newspaper article. You read it vertically. At the top, you expect a headline that will tell you what the story is about and allows you to decided whether it is something you want to read. The first paragraph gives you a synopsis of the whole story, hiding all the details while giving you the broad-brush concepts. As you continue downward, the details increase until you have all the dates, names, quotes, claims, and other minutiae. We would like a source file to be like a newspaper article.”

Robert C. Martin - Clean Code

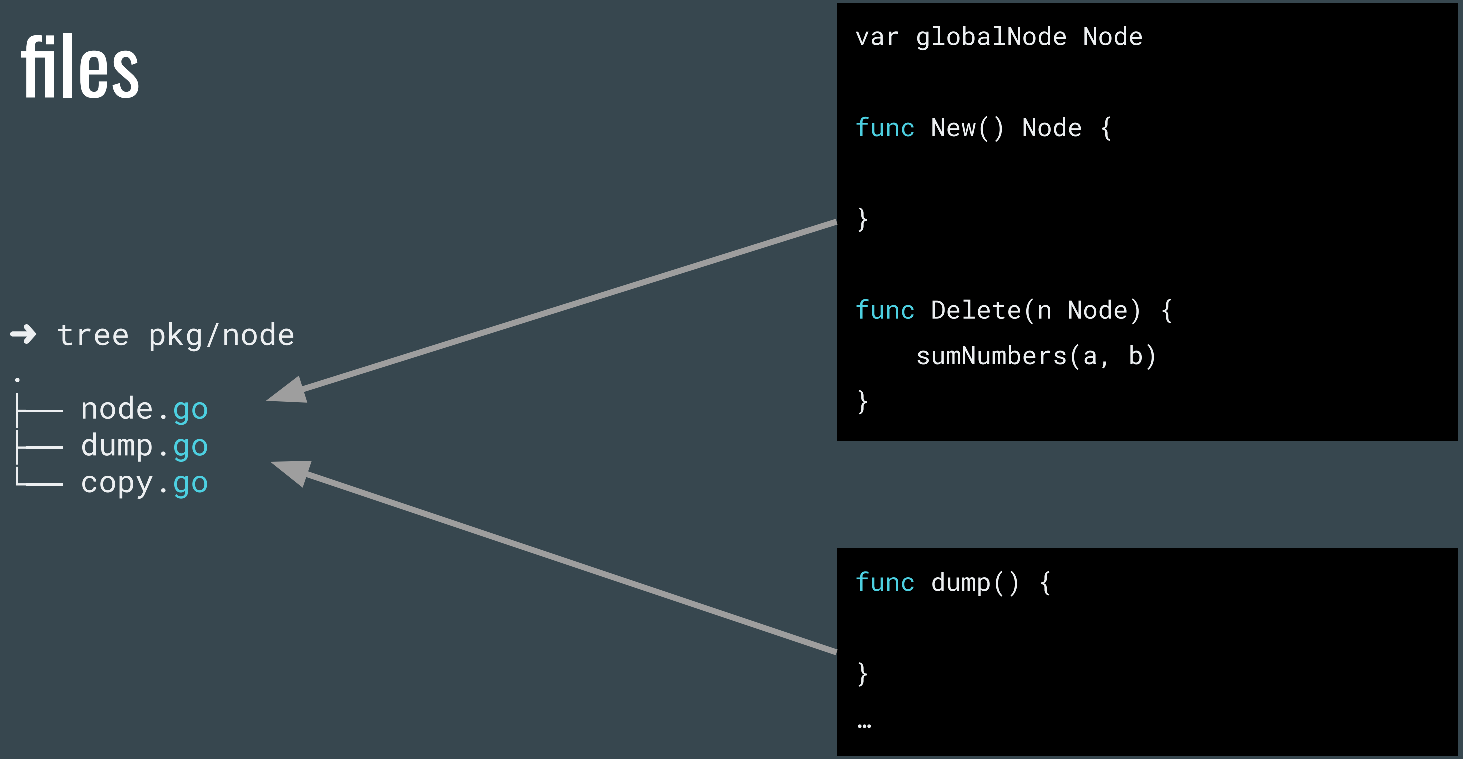
Reading like a newspaper

- Move the package public fields on top of the file
- Move the util functions on the bottom of the file
- Consider splitting the package into multiple files
- Name the main entry point of the package after the package
- Put main() to the top of the file

Order Matters

```
var globalNode Node

func New() Node {

}

func Delete(n Node) {
    sumNumbers(a, b)
}

func sumNumbers(a, b int) int {

}

func dump() {

}
```

Split to files



```
➜ tree pkg/node
.
├── node.go
├── dump.go
└── copy.go
```

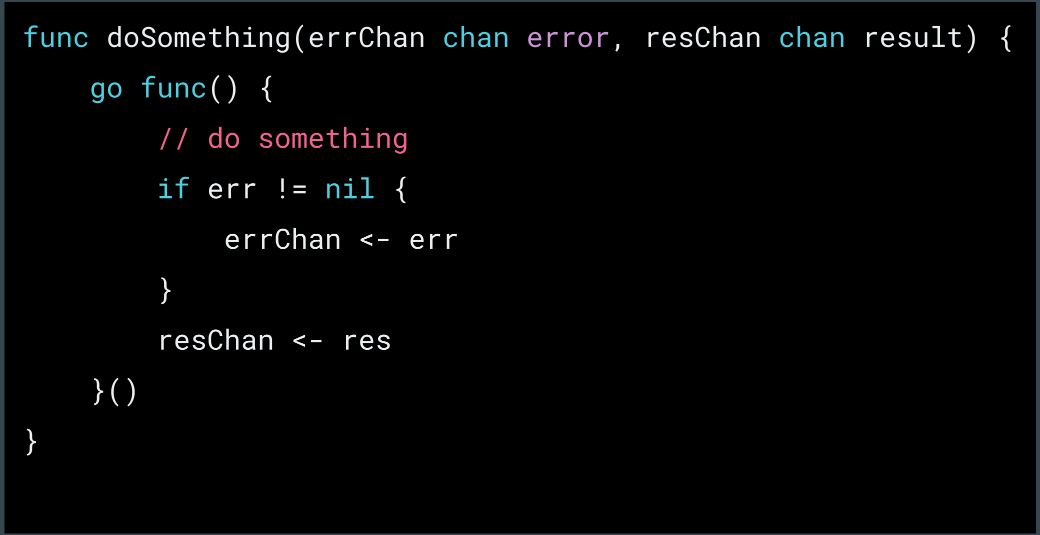
```
var globalNode Node

func New() Node {

}

func Delete(n Node) {
    sumNumbers(a, b)
}
```

```
func dump() {

}
…
```

Asynchronous functions

https://flic.kr/p/pBLFiw

Asynchronous functions



```go
func doSomething(errChan chan error, resChan chan result) {
    go func() {
        // do something
        if err != nil {
            errChan <- err
        }
        resChan <- res
    }()
}
```

Asynchronous functions

```
func doSomething() (result, error) {
    // do something
    return res, nil
}

go func() {
    res, err := doSomething()
    if err != nil {
        errChan <- err
    }
    resChan <- res
}()
```

Asynchronous functions

Synchronous functions keep goroutines localized within a call, making it easier to reason about their lifetimes and avoid leaks and data races. They're also easier to test: the caller can pass an input and check the output without the need for polling or synchronization.

https://github.com/golang/go/wiki/CodeReviewComments#synchronous-functions

https://flic.kr/p/oqPNCn

Functions that lie

Functions that lie

```
ClearNode(n)
```

Functions that lie

```
ClearNode(n)

func ClearNode(n Node) {
    if n.Name == "donotclean" {
        return
    }
    // clean
}
```

Functions that lie

```
if n.Name == "donotclean" {
    ClearNode(n)
}

func ClearNode(n Node) {
    // clean
}
```

Wrapping up

The Pareto principle states that for many outcomes, roughly 80% of consequences come from 20% of causes (the "vital few")

https://en.wikipedia.org/wiki/Pareto_principle

Simplicity is complicated
but the clarity is worth the fight

(Rob Pike)

Thanks!

Any questions?

@fedepaol

hachyderm.io/@fedepaol

fedepaol@gmail.com

fpaoline@redhat.com

Slides at: speakerdeck.com/fedepaol

https://flic.kr/p/4XpCsF

Interfaces

Unnecessary Interfaces

```go
type parser struct {
    ... // snip
}

func (p *parser) Parse(s string) (*Config, error) {
    ... // snip
}

type Parser interface {
    Parse(s string) (*Config, error)
}
```

Unnecessary Interfaces

```go
type parser struct {
    ... // snip
}

func (p *parser) Parse(s string) (*Config, error) {
    ... // snip
}

type IParser interface {
    Parse(s string) (*Config, error)
}
```

Unnecessary Interfaces

```
type Parser struct {
    ... // snip
}

func (p *Parser) Parse(s string) (*Config, error) {
    ... // snip
}
```