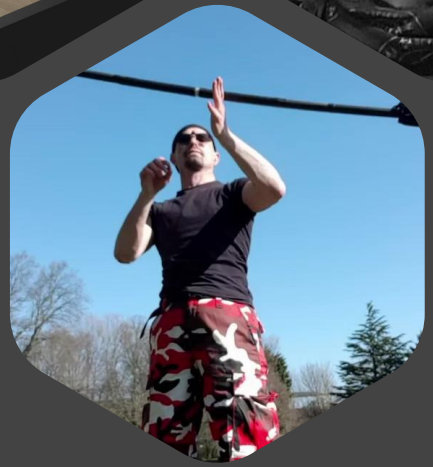
The Seven Deadly(ish) Gopher Sins

Who am I?



John Gregory

Senior Software Engineer at Admiral Money, artist, fire spinner and proud Gopher

/in/necrophonic

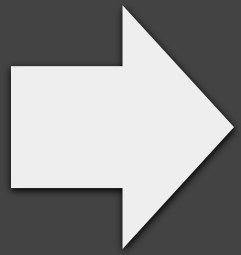
Lust

[lŭst]

“
Fools rush in where
Gophers fear to tread.
”

```
func main() {
  doCoolStuff()
}
```

```
func main() {
  doCoolStuff()
}
```



```
func main() {
  go doCoolStuff()
}
```

```go
func sum(ns []string) (int, error) {
    sum := 0
    for _, s := range ns {
        n, err := strconv.Atoi(s)
        if err ≠ nil {
            return -1, err
        }
        sum += n
    }
    return sum, nil
}
```

Starting with a func to convert and sum a range of numbers.

```go
func sum(ns []string) (int, error) {
    sum := 0
    for _, s := range ns {
        n, err := strconv.Atoi(s)
        if err ≠ nil {
            return -1, err
        }
        sum += n
    }
    return sum, nil
}
```

```go
func sum(ns []string) (int, error) {
    sum := 0
    errChan := make(chan error)
    numberChan := make(chan int)

    for _, s := range ns {
        go func(si string) {
            n, err := strconv.Atoi(si)
            if err ≠ nil {
                errChan ← err
                return
            }
            numberChan ← n
        }(s)
    }

    for i := 1; i ⩽ len(ns); i++ {
        select {
        case err := ←errChan:
            return -1, err
        case n := ←numberChan:
            sum += n
        }
    }
    return sum, nil
}
```

Starting with a func to convert and sum a range of numbers.

Once we add goroutines we can get benefits, but also an increase in complexity.

Avoiding the Go sin of Lust

- *You don't need goroutines to "do Go properly".*
- *Wait to use complex features like goroutines or generics when you genuinely need them.*

Wrath

[răth]

> *Using panics instead of checked errors is like trying to stop a car with a wall instead of using the brakes.*

```go
func handler(w http.ResponseWriter, r *http.Request) {
    mascot := r.URL.Query().Get("mascot")
    reAwesome := regexp.MustCompile(`[Gg]ophers?`)
    if reAwesome.MatchString(mascot) {
        fmt.Fprint(w, "Go is awesome!")
        return
    }
    fmt.Fprintf(w, "Well I like %s too!", mascot)
    return
}
```

Never do this.

If you do need to compile at runtime use the non-Must version.

```go
func handler(w http.ResponseWriter, r *http.Request) {
    mascot := r.URL.Query().Get("mascot")
    reAwesome := regexp.MustCompile(`[Gg]ophers?`)
    if reAwesome.MatchString(mascot) {
        fmt.Fprint(w, "Go is awesome!")
        return
    }
    fmt.Fprintf(w, "Well I like %s too!", mascot)
    return
}
```

Never do this.

If you do need to compile at runtime use the non-Must version.

```go
var reAwesome = regexp.MustCompile(`[Gg]ophers?`)

func handler(w http.ResponseWriter, r *http.Request) {
    mascot := r.URL.Query().Get("mascot")
    if reAwesome.MatchString(mascot) {
        fmt.Fprint(w, "Go is awesome!")
        return
    }
    fmt.Fprintf(w, "Well I like %s too!", mascot)
    return
}
```

Prefer instead to pre-compile the regex at compile time.

Avoiding the Go sin of Wrath

- Always prefer checked errors to panics.
- Only use recover patterns as a last resort, not as standard error handling.
- If you must use Must, ensure they're always executed at start up - not user run time.

Greed

[grēd]

“*There are known knowns, known unknowns and unknown unknowns ...*

... and over engineering.”


Let's add authentication
Maybe we should add a CMS?
What about internationalisation?
Maybe we want to use a database?
These types could change. Should we use generics?
How about command line options?

Avoiding the Go sin of Greed

- Don't over-engineer yourself into a corner.
- Speaking of actual, generics. Only use them sparingly when you have a genuine need.

Sloth

[slōth]

“ *Laziness can be a virtue, but only when you put the effort in.* ”

```
i += 1
```

Given a simple piece of code without surrounding context.

```
i += 1
```

Given a simple piece of code without surrounding context.

```
// Add 1 to i
i += 1
```

The *sinful laziness* is to add a comment that merely says ***what*** it does.

```
i += 1
```

Given a simple piece of code without surrounding context.

```
// Add 1 to i
i += 1
```

The *sinful laziness* is to add a comment that merely says ***what*** it does.

```
// Increment usages to show how many times
// this resource has been accesed. This is
// important as any resource accessed over
// 10 times will be flagged for review.
usages += 1
```

To turn our sin to virtue, instead explain ***why*** the action is performed.

```go
var m map[string]any
err := json.Unmarshal(data, &m)
if err ≠ nil {
  return err
}
```

Handling errors without context can make understanding what went wrong harder.

```go
var m map[string]any
err := json.Unmarshal(data, &m)
if err ≠ nil {
  return err
}
```

Handling errors without context can make understanding what went wrong harder.

```go
var m map[string]any
err := json.Unmarshal(data, &m)
if err ≠ nil {
  return fmt.Errorf(
    "decode customer data: %w",
    err,
  )
}
```

By wrapping with some extra info using `fmt.Errorf` we get better insight into what was happening when it occurred.

Using `%w` also allows us to easily unwrap nested errors.

Avoiding the Go sin of Sloth

- Ensure comments explain why, not what.
- Don't allow errors to bubble without context.
- Don't skimp on effort now to make your understanding easier later.

Gluttony

[glŭt'n-ē]

“

You probably don’t need that snorkel on your city break.

”

Only core library

```go
package main

import (
    "fmt"
    "log"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "Hello Gophers!")
    })
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

```
module mymodule

go 1.20
```

Using a framework

```go
package main

import (
    "github.com/labstack/echo/v4"
    "net/http"
)

func main() {
    e := echo.New()
    e.GET("/", func(c echo.Context) error {
        return c.String(http.StatusOK, "Hello Gophers!")
    })
    e.Logger.Fatal(e.Start(":8080"))
}
```

```
module mymodule

go 1.20

require github.com/labstack/echo/v4 v4.11.1

require (
    github.com/labstack/gommon v0.4.0 // indirect
    github.com/mattn/go-colorable v0.1.13 // indirect
    github.com/mattn/go-isatty v0.0.19 // indirect
    github.com/valyala/bytebufferpool v1.0.0 // indirect
    github.com/valyala/fasttemplate v1.2.2 // indirect
    golang.org/x/crypto v0.11.0 // indirect
    golang.org/x/net v0.13.0 // indirect
    golang.org/x/sys v0.11.0 // indirect
    golang.org/x/text v0.12.0 // indirect
)
```

Avoiding the Go sin of Gluttony

- Start simple, then layer complexity, rather than using a framework or library by default.
- Prefer libraries with good support and recent commits.
- Don't use multiple libraries for the same thing.

Envy

[en-vee]

> *The grass isn't greener in the other codebase.*

```go
func FeedPupper() error {
    if isPupHungry {
        err := feedPup()
        if err == nil {
            if isGoodPup {
                rubBelly()
            }
        } else {
            return err
        }
    } else {
        return errors.New("not hungry!")
    }
    return nil
}
```

A nested layout as you may recognise as similar to something like Python.

```go
func FeedPupper() error {
    if !isPupHungry {
        return errors.New("not hungry!")
    }

    if err := feedPup(); err != nil {
        return err
    }

    if isGoodPup {
        rubBelly()
    }
    return nil
}
```

Re-written to be more as you would expect in Go, with left-edge happy path.

Example inspired by Mat Ryer's "Code: Align the happy path to the left edge" [1]

Avoiding the Go sin of Envy

- *Keep things simple. Don't adopt patterns unless you actually need them.*
- *You don't need interfaces for everything!*
- *Acquaint yourself with Effective Go[2] for clean coding.*

Pride

[prīd]

“*Assuming you always know what’s best for everyone.*”

Git Diff Check

/necrophonic/git-diff-check

Git Diff Check

/necrophonic/git-diff-check

Go Eliza

/necrophonic/eliza-slackbot
/necrophonic/go-eliza

Avoiding the Go sin of Pride

- Don't artificially restrict access to models etc unless you have a good reason.
- Avoid running goroutines in libraries. Prefer to provide functions a user can choose to run as a goroutine instead.

... and lastly the greatest sin

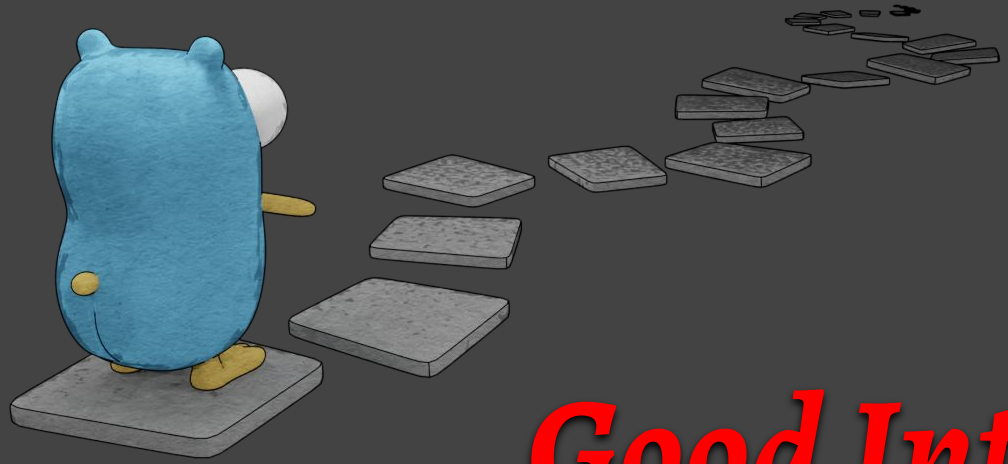
(yes, I know that makes 8)

... and lastly the greatest sin

(yes, I know that makes 8)

Complexity

The road to Complexity is paved...



...with Good Intentions

Thanks for listening!

Any questions?

Find me

/necrophonic

/in/necrophonic

/cafpanda

@shadowgophr

Code formatting: carbon
Gopher renders: blender ©J. Gregory

References

[1] Mat Ryer - Code: Align the happy path to the left edge:

https://medium.com/@matryer/line-of-sight-in-code-186dd7cdea88

[2] Effective Go:

https://go.dev/doc/effective_go