SPENCER KORMOS

Bloomberg
https://www.linkedin.com/in/spencerkormos

SLog in Anger

Production Patterns for Testable and Concise Structured Logging

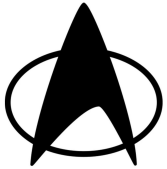


About Me - Professionally

- Software developer since Java 1.3
- Go since v1.7 (~2016)
- Bloomberg since 2022
- Enterprises and startups

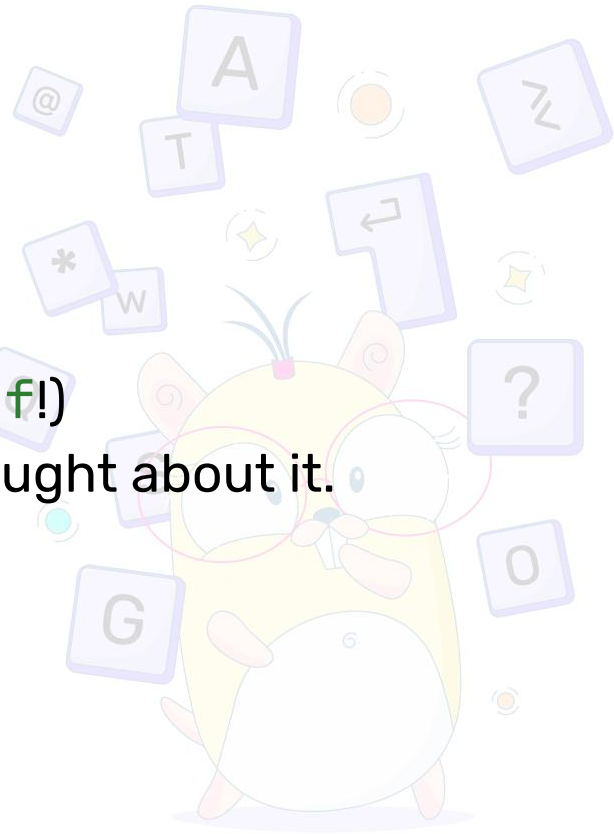


And then something personal...

- Native New Yorker 
- Live Theatre (musicals or plays) 
- Formula One racing enthusiast (44!)
- LLAP! (iykyk)  



About You



I'm assuming you:

- have some familiarity with Go.
- used some logging (even `fmt.Printf`!)
- are currently using `slog`, or have thought about it.

Why this talk?

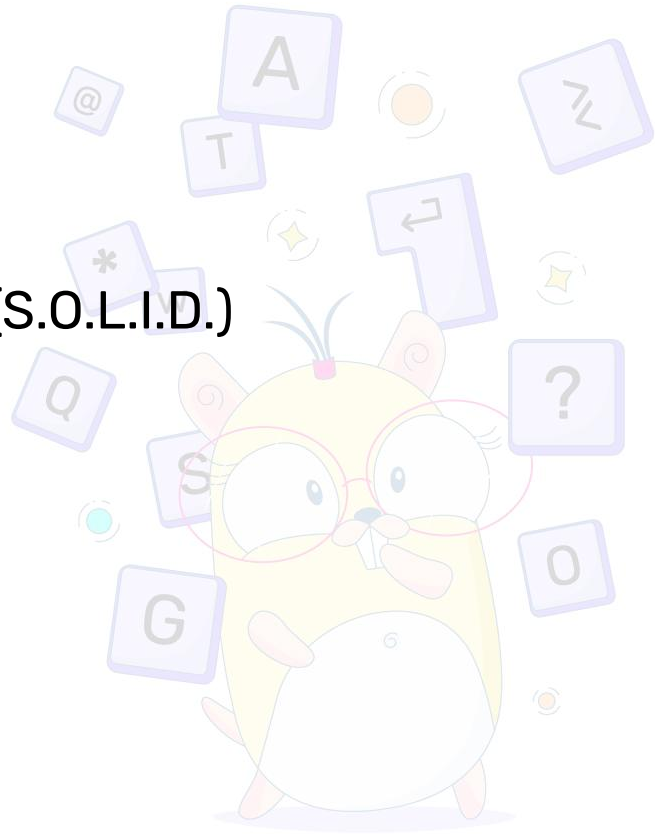
Production-Grade

"What can I do to reduce my burden if there's a Severity-1 outage at 3am, and the CTO is on the support call?"



Why this talk... Maintainability...

- Standards
- Predictable patterns and idioms (S.O.L.I.D.)
- Testable code
- Surge in A.I. Development

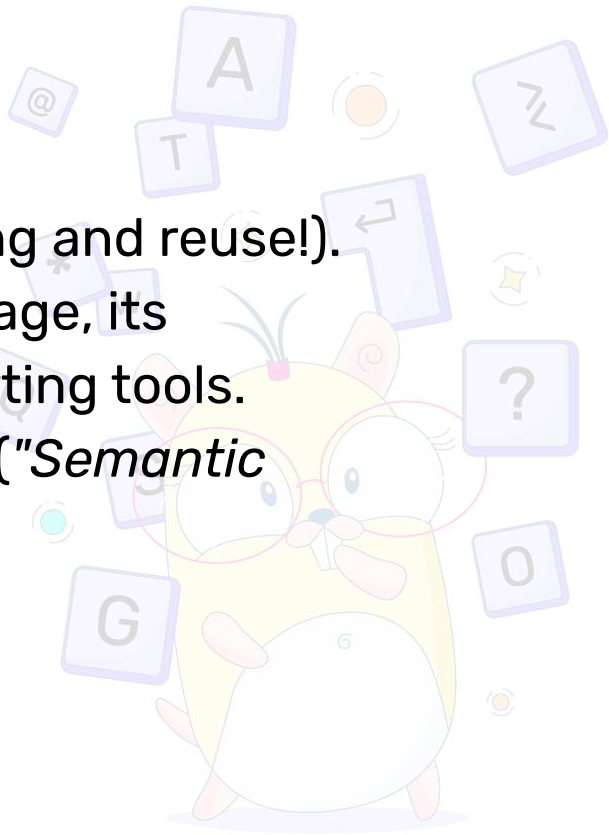


Why this talk... Observability...

- Metrics
- Tracing
- ***Logging***

What I'm hoping you'll get out of this...

- Logging as a first-class citizen (testing and reuse!).
- A better awareness of the `slog` package, its capabilities and patterns, and supporting tools.
- Adopting a style and sticking with it. (*"Semantic Conventions"*, linting)



What is SLog?



What is Structured Logging?

Structured Logging generates log messages in a well-defined format (such as JSON or XML):

- Machine-readable.
- Easier to filter, analyze, and alert on.
- Require minimal or no pre-processing

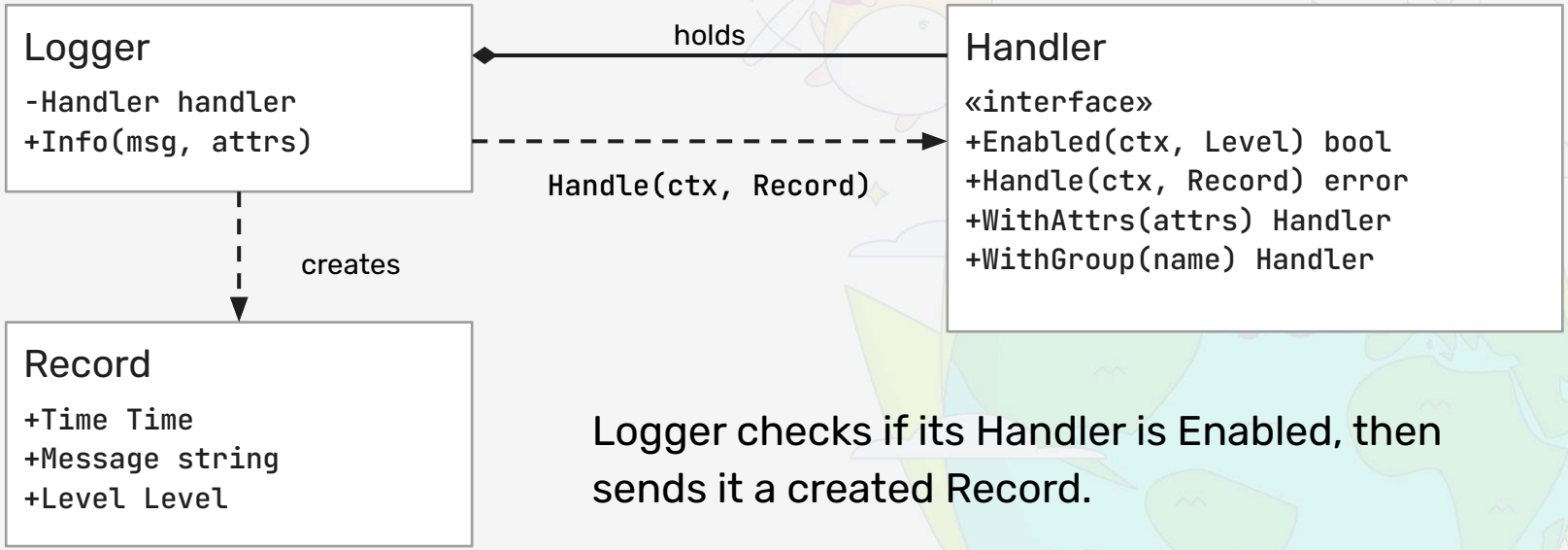
`log/slog` (since Go 1.21 [1]) addresses these requirements.

[1] https://go.dev/doc/go1.21#slog

Overview

`Logger`	Primary entry into the API.
`Record`	Represents a log event. Created by the Logger.
`Handler`	Interface for acting on log events: intercept, adapt, write.
`Value`	Typed wrapper around Go native types; complex through `slog.LogValuer`.
`Attr`	Convenience wrapper for key and `slog.Value` types.
`With(... any)`	Create a child logger with provided attributes on every log message.
`WithGroup(name)`	Create a child logger with all attributes grouped under a common key.

Logger, Handler, and Record



Logger checks if its Handler is Enabled, then sends it a created Record.

Handler

Handles and acts on log records produced by a Logger, including:

- Sending to an `io.Writer` (StdErr, file, etc.).
- Writing to another service or database.
- Decorator / Interceptor for appending and augmenting data.
- Adapter to another logging library.
- Filtering, by returning `true` or `false` from `Enabled()`.

Built-ins: `Text`, `JSON`, `Discard`, and `Multi`.

The `slog.Handler` interface is `context.Context` aware, use `logger.<LEVEL>Context(ctx, message, ...attrs)`.

Logger and Handler example usage

```
jsonLogger := slog.New(slog.NewJSONHandler(os.Stdout, nil))

jsonLogger.Error("user logged in",
    "success", false,               // any
    slog.Int("attempts", 3),        // slog.Attr
    slog.String("name", "alice"))
```

```
{"time":"...","level":"...","msg":"user logged in","success":false,"attempts":3,"name":"alice"}
```

Value

```
type Value struct {
    _   [0]func() // disallow ==; use Value.Equal
    num uint64    // int64, uint64, float64, bool, duration, string length
    any any       // Kind tag — or the value itself when KindAny
}
```

- Common types are stored without boxing them into an `any`.
- Compare with `Value.Equal`, not `==` (*more on this in "Testing"*).

Attr

An `Attr` is a key plus a `Value`:

```
type Attr struct {
    Key   string
    Value Value // non-comparable, so Attr is too — use Attr.Equal
}
```

- Examples: `slog.String`, `slog.Group`, etc.
- Loose key-value pairs are converted to `Attr`s at call time.
- Compare with `Attr.Equal`, not `==`.

With() and WithGroup()

```
logger := baseLogger.With(slog.String("component", "orders"))
logger.Info("service started")                                  // [1]

reqLogger := logger.WithGroup("request").With(
    slog.String("method", r.Method),
    slog.String("path", r.URL.Path),
)
reqLogger.Info("order created", slog.Int("items", 3))           // [2]
```

```
{...,"msg":"service started","component":"orders"}              // [1]
{...,"msg":"order created","component":"orders",
  "request":{"method":"POST","path":"/orders","items":3}}       // [2]
```

Patterns and Anti-Patterns



Anti-Pattern

Using "Default" Logger

```
slog.SetDefault(slog.New(slog.NewJSONHandler(os.Stdout, nil)))
slog.Info("You are seeing me log via Default!")
```

- `SetDefault` is global and mutable.
- Intended to bridge legacy code.
- Using top-level `slog` functions is an implicit API.

Pattern

Flag "Default" Logger

```
func main() {
    handler := slog.NewJSONHandler(os.Stdout, nil)
    logger := slog.New(handler)
    slog.SetDefault(logger.With(slog.String("component", "DEFAULT")))
    // ...
}
```

- Initialize it deliberately in `main()`, tagged with `component=DEFAULT`.
- Prevents accidental calls to the legacy `log` package.
- Still doesn't prevent overrides. Can still help refactoring efforts / prevent misuse.
- Good callout for linting and AI skills.

Pattern

Prefer `Attr` over Loose Key-Value Pairs

```
logger.Error("login failed", logkit.Error(err))                                  // [1]
logger.Error("login failed", "anError", logkit.Error(err))                       // [2]

logger.Info("user login", slog.String("user", "alice"), slog.Int("attempts", 3)) // [3]

logger.Info("user login", 42, "userID")                                          // [4]
```

```
{...,"msg":"login failed","error":"Login Failed due to: ..."}                    // [1]
{...,"msg":"login failed","anError":{"Key":"error","Value":{}}}                  // [2]

{...,"msg":"user login","user":"alice","attempts":3}                             // [3]

{...,"msg":"user login","!BADKEY":42,"!BADKEY":"userID"} // [4] caught by go vet and go test
```

Pattern

Custom `slog.Attr()` Helpers

```
const AttrKeyCause = "cause"
func CauseAttr(c any) slog.Attr {
    if err, ok := c.(error); ok {
        return slog.String(AttrKeyCause, err.Error())
    }
    return slog.Any(AttrKeyCause, c)
}

defer func() {
    if r := recover(); r ≠ nil {
        logger.Error("panic recovered", CauseAttr(r))
    }
}()
```

Pattern

Custom `slog.Attr()` Helpers (cont'd)

```
func PartlyRedactedStringAttr(key, value string) slog.Attr {
    if len(value) ⩽ 3 {
        return slog.String(key, "***")
    }
    return slog.String(key, value[:3]+"***")
}
```

Alternatively, `HandlerOptions.ReplaceAttr` can be used across a whole Handler (e.g. `JSONHandler`).

Pattern

Custom `slog.Attr()` Helpers (cont'd)

Instead of:

```
logger.WithGroup("request").With("method", r.Method, "path", r.URL.Path)
```

```
func RequestAttr(r *http.Request) slog.Attr {
    return slog.Group("request",
        slog.String("method", r.Method),
        slog.String("path", r.URL.Path),
    )
}

logger.Info("handling request", RequestAttr(req))
```

```
{"time":"...","level":"...","msg":"handling request","request":{"method":"GET","path":"/api/users"}}
```

Pattern

Implement `slog.LogValuer`

```
type RequestLogValue struct {
    *http.Request
}

func (r RequestLogValue) LogValue() slog.Value {
    return slog.GroupValue(slog.String("method", r.Method), slog.String("path", r.URL.Path))
}

logger := slog.New(slog.NewJSONHandler(os.Stdout, nil))
req, _ := http.NewRequest("GET", "http://example.com/api/users", nil)
logger.Info("handling request", slog.Any("request", RequestLogValue{req}))
```

```
{"time":"...","level":"INFO","msg":"handling request","request":{"method":"GET","path":"/api/users"}}
```

Pattern

Generating `LogValuer`

A struct tag plus a `stringer`-style generator keeps `LogValue()` in sync with the struct:

```go
//go:generate slogvaluer -type=Order
type Order struct {
    ID     string `slog:"id"`
    Total  int    `slog:"total"`
    APIKey string `slog:"-"`             // skipped
    Email  string `slog:"email,redact"` // redacted
}
```

```go
// Code generated by slogvaluer
// order_slogvaluer.go
func (o Order) LogValue() slog.Value {
    return slog.GroupValue(
        slog.String("id", o.ID),
        slog.Int("total", o.Total),
        slog.String("email", redact(o.Email)),
    )
}
```

```go
order, _ := processor.NextOrder()
logger.Info("order received", slog.Any("order", order))
```

Anti-Pattern

Passing Logger through Context

```
func SpanLoggingMiddleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        logger := MustLoggerFromContext(r.Context())

        attrs := SpanAttrs(trace.SpanContextFromContext(r.Context()))
        spanLogger := slog.New(logger.Handler().WithAttrs(attrs))

        next.ServeHTTP(w, r.WithContext(SetLoggerOnContext(r.Context(), spanLogger)))
    })
}

// ... some wiring here

func MyHttpHandlerFunc(w http.ResponseWriter, r *http.Request) {
    logger := MustLoggerFromContext(r.Context())
    logger.Info("MyHttpHandler called")
}
```

Anti-Pattern

Passing Logger through Context

What happens when `MyHttpHandlerFunc` is called?

- (a) `"MyHttpHandler called"` is written with the span and trace IDs.
- (b) `"MyHttpHandler called"` is written without the span and trace IDs.
- (c) It panics, because there's no logger on the context.
- (d) You can’t tell from this code.

Anti-Pattern

Passing Logger through Context

```
func SpanLoggingMiddleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        logger := MustLoggerFromContext(r.Context())

        attrs := SpanAttrs(trace.SpanContextFromContext(r.Context()))
        spanLogger := slog.New(logger.Handler().WithAttrs(attrs))

        next.ServeHTTP(w, r.WithContext(SetLoggerOnContext(r.Context(), spanLogger)))
    })
}

// ... some wiring here

func MyHttpHandlerFunc(w http.ResponseWriter, r *http.Request) {
    logger := MustLoggerFromContext(r.Context())
    logger.Info("MyHttpHandler called")
}
```

Pattern

Context-Aware Logging

1

```
type OTelHandler struct { ... }

func (h *OTelHandler) Handle(ctx context.Context, rec slog.Record) error { ... }
```

2

```
h := &OTelHandler{slog.NewJSONHandler(os.Stdout, nil)}
srv := NewOrderServer(h)    // NewOrderServer does slog.New(h)
```

3

```
func (s *OrderServer) handleOrder(w http.ResponseWriter, r *http.Request) {
    s.logger.InfoContext(r.Context(), "some message")
}
```

```
{"msg":"some message","spanID":"00f067aa0ba","traceID":"4bf92f3577b34da6a3ce92"}
```

Context-Aware Logging (cont'd)

```go
// idAttrs builds Attrs keyed by AttrKeyOtelSpanID and AttrKeyOtelTraceID.
func SpanAttrs(sc trace.SpanContext) []slog.Attr {
    if !sc.IsValid() {
        return idAttrs("NOT SET", "NOT SET")
    }
    return idAttrs(sc.SpanID().String(), sc.TraceID().String())
}

type OTelHandler struct { inner slog.Handler } // implements slog.Handler

func (h *OTelHandler) Handle(ctx context.Context, rec slog.Record) error {
    rec = rec.Clone() // for safety
    rec.AddAttrs(SpanAttrs(trace.SpanContextFromContext(ctx))...)

    return h.inner.Handle(ctx, rec)
}
```

Pattern

Use the `...Context` Methods

If a `ctx` is in scope, pass it — even if no handler reads it today.

```
logger := slog.New(&OTelHandler{slog.NewJSONHandler(os.Stdout, nil)})

logger.InfoContext(ctx, "some message")

logger.Info("some message")
```

```
{"msg":"some message","spanID":"00f067aa0ba9","traceID":"4bf92f3577b34da6a3ce929d"}

{"msg":"some message","spanID":"NOT SET","traceID":"NOT SET"}
```

Other Items to Note on Custom Handlers

- A canceled context shouldn't stop logging.
- Never panic. Handle bad input and keep going.
- Call `Value.Resolve()` for every `Attr`, so `LogValuers` are honored.
- Return an error from `Handle` only when a write fails, including a nested `Handler`'s error, and never swallow it.

Guide: golang.org/s/slog-handler-guide

Pattern

Use *Logger or Handler as a Direct Dependency

Declare the dependency directly through a constructor — instead of threading it through `context.Context`.

```
func NewCache(h slog.Handler) *Cache {
    return &Cache{logger: slog.New(h)}    // slog.New already panics on a nil Handler
}
```

- Clearer usage: the requirement is visible in the signature, not hidden inside runtime context wiring.
- Easier validation: a constructor can reject a bad dependency immediately.
- Pick one, `Handler` or `*Logger`, and be consistent.

Pattern

`With()` for Local Values

```
type httpHandler struct { logger *slog.Logger }

func (h *httpHandler) Get(w http.ResponseWriter, r *http.Request) {
    reqLogger := h.logger.With(RequestAttr(r))
    reqLogger.InfoContext(r.Context(), "request started")
    // ...
    reqLogger.InfoContext(r.Context(), "request completed")
}
```

```
{"msg":"request started","request":{"method":"GET","path":"/api/users"},"spanID":"...","traceID":"..."}
{"msg":"request completed","request":{"method":"GET","path":"/api/users"},"spanID":"...","traceID":"..."}
```

- Values on `context.Context`, like trace IDs: a context-aware `Handler`.
- Values already in local scope, like the request or a component name: `With()`.

Testing



DiscardHandler

```
func TestCacheAdd(t *testing.T) {
    cache := NewCache(slog.DiscardHandler)    // logging is out of scope here
    // ... test the cache, not the logs
}

func BenchmarkOTelHandler(b *testing.B) {
    h := &OTelHandler{inner: slog.DiscardHandler}
    rec := slog.NewRecord(time.Now(), slog.LevelInfo, "msg", 0)
    for b.Loop() {
        _ = h.Handle(context.Background(), rec)   // call Handle directly
    }
}
```

`Attr` and `Value.Equal(Value)`

```
func TestRequestLogValue(t *testing.T) {
    req := httptest.NewRequest("GET", "/api/users", nil)

    got := RequestLogValue{req}.LogValue()
    want := slog.GroupValue(
        slog.String("method", "GET"),
        slog.String("path", "/api/users"),
    )

    if !got.Equal(want) { t.Errorf("got %v, want %v", got, want) }
}
```

```
if !CauseAttr(errors.New("boom")).Equal(slog.String("cause", "boom")) { t.Errorf(...) }
```

Caveat: `Equal` panics when a `slog.Any` value holds a slice, map, or func.

slogtest.Run()

```
func TestMyHandler(t *testing.T) {
    var buf bytes.Buffer
    slogtest.Run(t,
        func(t *testing.T) slog.Handler {   // a fresh handler for each test case
            buf.Reset()
            return slog.NewJSONHandler(&buf, nil)
        },
        func(t *testing.T) map[string]any { // what that handler wrote
            var m map[string]any
            _ = json.Unmarshal(buf.Bytes(), &m) // a bad parse fails the case
            return m
        },
    )
}
```

- StdLib encodes 17 test-cases; you provide the handler and a way to read its output.
- Each case runs as its own subtest, so a failure is reported by name.

Record.Attrs()

```
// assertHasAttr reports whether rec has an Attr equal to want.
func assertHasAttr(t *testing.T, rec slog.Record, want slog.Attr) bool {
    t.Helper()

    found := false
    rec.Attrs(func(a slog.Attr) bool {
        found = a.Equal(want)
        return !found // break iteration
    })

    if !found { t.Errorf("missing attr %v", want) }
    return found
}

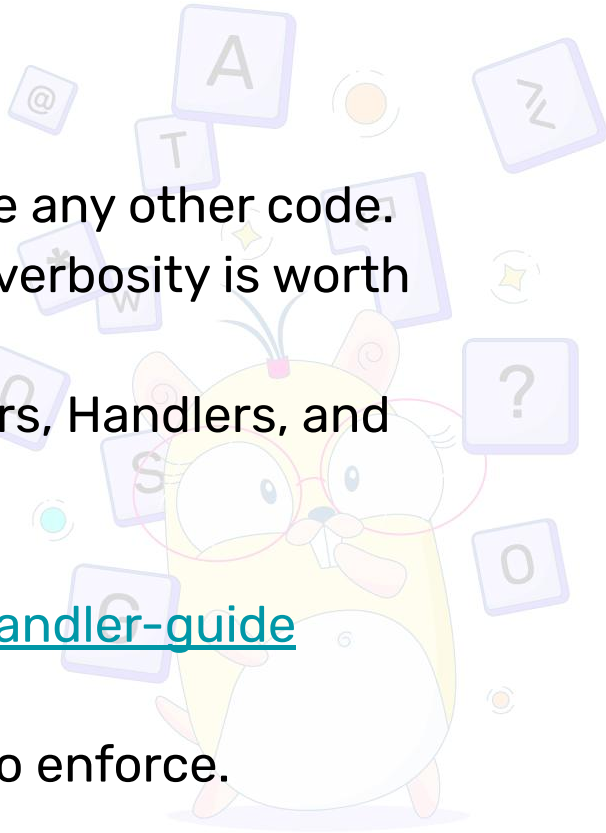
// spy (a captureHandler) stores the last Record it receives.
_ = h.Handle(ctx, slog.Record{})
assertHasAttr(t, spy.rec, slog.String(AttrKeyOtelSpanID, sc.SpanID().String()))
```

Hardening



Hardening

- Invest in reuse and testing: treat logging like any other code.
- Prefer `Attr` over loose key-value pairs: the verbosity is worth the safety.
- A shared `slog` kit: one place for `Attr` helpers, Handlers, and semantic conventions.
- Read the docs! https://pkg.go.dev/log/slog
- Writing a Handler? See golang.org/s/slog-handler-guide
- Explore a linter to enforce the rules strictly.
- An A.I. skill that mirrors the style you want to enforce.



An A.I. Skill for `slog`

- `Logger` or `Handler` as a direct dependency.
- `Attr`s and `...Context` methods.
- Handler rules and testing.
- New Handlers must be tested by `slogtest.Run()`.
- Check `Record.Clone()` is called.
- General best practices.

The Skill in Practice

```
// Avoid: Loose key-pair args to logger methods
logger.InfoContext(ctx, "request handled", "method", r.Method, "path", r.Path,
    "status", code, "elapsed", elapsed)

// Preferred
logger.InfoContext(ctx, "request handled", slog.String("method", r.Method),
    slog.String("path", r.Path), slog.Int("status", code),
    slog.Duration("elapsed", elapsed))
```

```
// Avoid: using io.Discard to remove logger noise
cache := NewCache(slog.NewJSONHandler(io.Discard, nil))

// Preferred: slog.DiscardHandler as a direct Handler replacement
cache := NewCache(slog.DiscardHandler)
```

Thank You!

Questions?



SPEAKER NAME

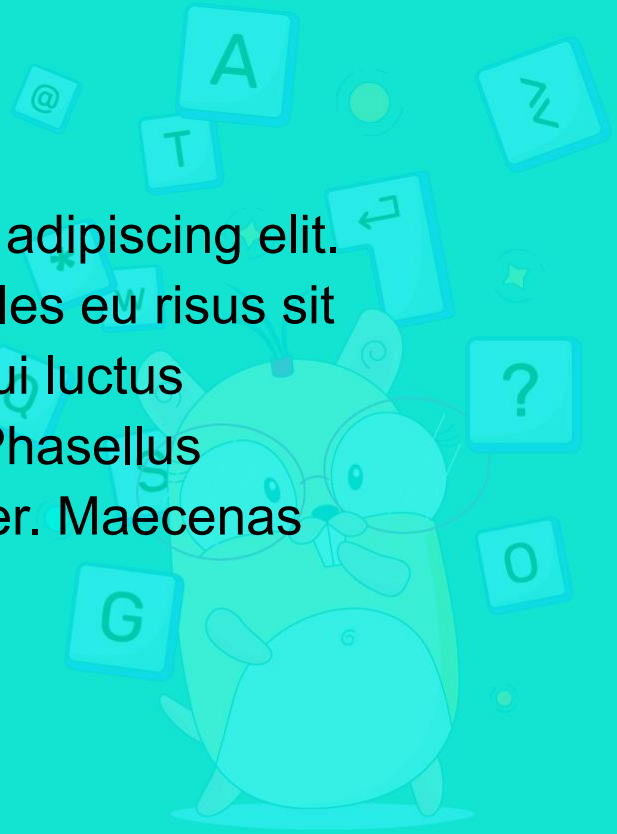
Professional Role

TALK NAME

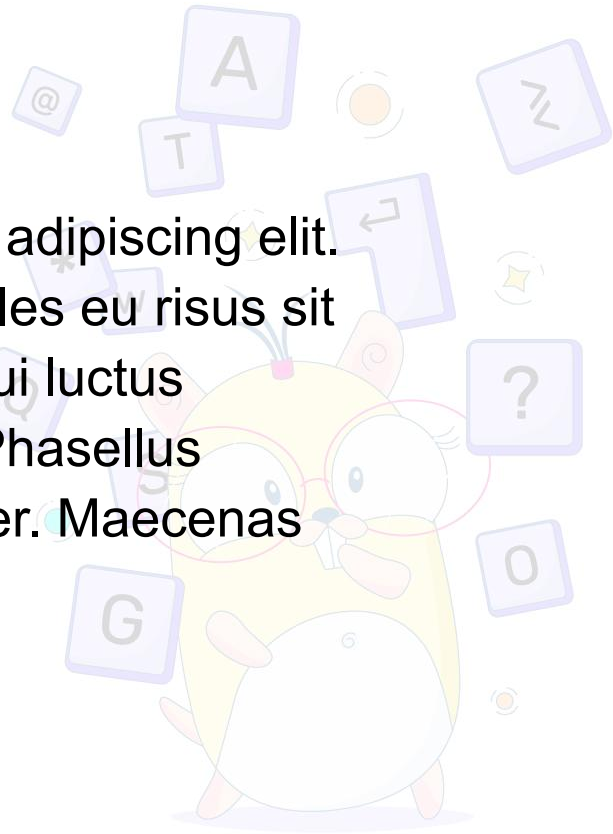


INTRODUCTION

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur vitae turpis mi. Phasellus sodales eu risus sit amet dapibus. Donec semper purus in dui luctus dapibus. Nullam sed ullamcorper urna. Phasellus ullamcorper sodales sem nec ullamcorper. Maecenas blandit urna ac diam tempus viverra.



INTRODUCTION



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur vitae turpis mi. Phasellus sodales eu risus sit amet dapibus. Donec semper purus in dui luctus dapibus. Nullam sed ullamcorper urna. Phasellus ullamcorper sodales sem nec ullamcorper. Maecenas blandit urna ac diam tempus viverra.

SECTION TITLE

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Nam est ligula, faucibus nec urna sed, tristique volutpat massa.
- Phasellus congue leo nunc, eu vehicula enim interdum vitae.

SECTION TITLE

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Nam est ligula, faucibus nec urna sed, tristique volutpat massa.
- Phasellus congue leo nunc, eu vehicula enim interdum vitae.

Subsection Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam est ligula, faucibus nec urna sed, tristique volutpat massa. Phasellus congue leo nunc, eu vehicula enim interdum vitae. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Nam est ligula, faucibus nec urna sed, tristique volutpat massa. Phasellus congue leo nunc, eu vehicula enim interdum vitae.

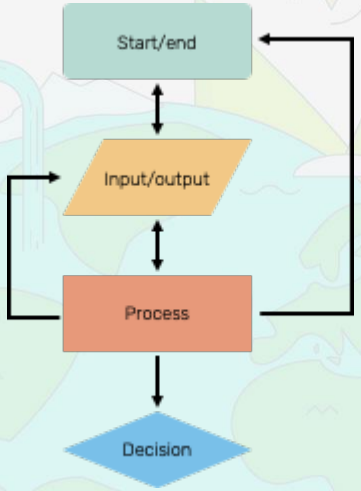
Subsection Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam est ligula, faucibus nec urna sed, tristique volutpat massa. Phasellus congue leo nunc, eu vehicula enim interdum vitae. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Nam est ligula, faucibus nec urna sed, tristique volutpat massa. Phasellus congue leo nunc, eu vehicula enim interdum vitae.

Subsection Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam est ligula, faucibus nec urna sed, tristique volutpat massa. Phasellus congue leo nunc, eu vehicula enim interdum vitae. Lorem ipsum dolor sit amet, consectetur adipiscing elit.



Section Title

Subsection Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Note: For use with the projector it is best to use a light theme in order to improve the accessibility and readability of the code.
You can paste the code from VScode that includes the formatting and optionally adapt the background with the "Fill color" button.

```go
func hello() {
    fmt.Println("Hello world! 🌍")
}
```

Subsection Title

```
func hello() {
    fmt.Println("Hello world! 🌍")
}
```

Subsection Title

```
func hello() {
    fmt.Println("Hello world! 🌍")
}
```

Greetings and conclusion



SPEAKER NAME

mail@speaker.com

