Learning Superpowers

Leveraging Obsidian for technical learning

Belén Albeza @ladybenko

Who am I?

Belén Albeza
@ladybenko

belenalbeza.com (coding) / benkoalbeza.com (writing)

Concepts

 Commonplace book

 Digital Garden

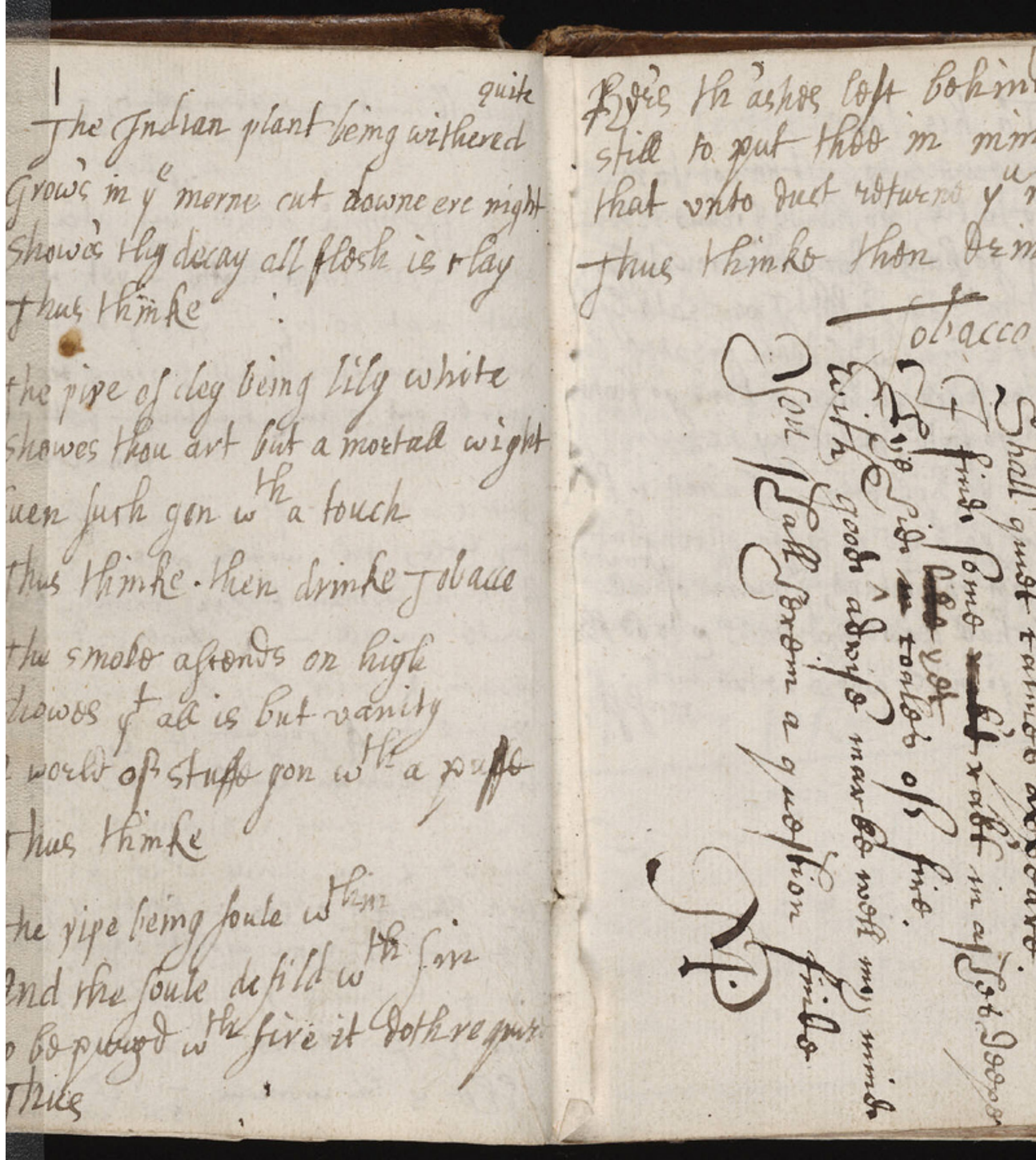
 Zettlekasten

 Tools for Thought

Commonplace book

A lifelong companion

- A notebook to compile knowledge and experiences
- Personal
- People would write down quotes, ideas, recipes, letters, tables with info, etc.



Digital Garden

The original website

- “Old-school” website
- Non-linear / chronologic structure
- Pages are densely linked to aid navigation and enhance discovery

brainfuck
chip8
turquoise
uxn
orca

varvara

drifblim
beetbug

wiki.xxiivv.com

The Sound Of Plumpkins

Varvara is a clean-slate computing stack based on the Uxn CPU.

This personal computer system, built on top of the {Uxn} virtual machine, was designed to run audio and visual applications. To see a list of compatible software, see {roms}, and the

Zettlekasten

An information system

- An analog personal information system
- Based on pen&paper and loads of index cards
- Used by Niklas Luhmann (a prolific scientist)
- Notes had a unique ID that would allow to have “hyperlinks”

Tools for Thought

A new generation of apps

- Beyond notes apps
- They allow us to capture of thoughts in writing, drawings, diagrams, audio...
  - And also make connections between them
- Ex: Notion, Obsidian, Roam, Logseq, Tana, Muse, etc.

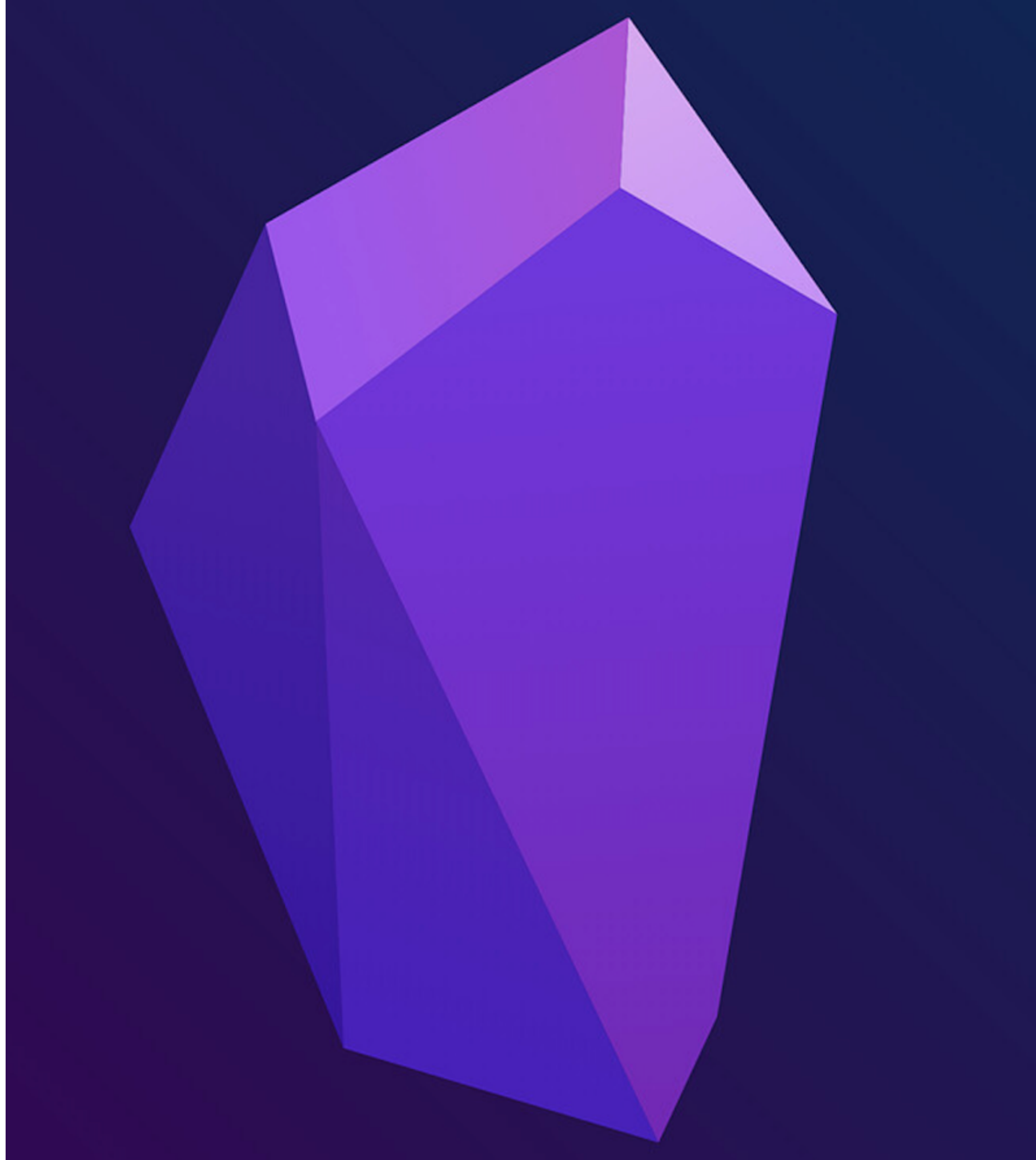
Can we have it all?

We can build our own commonplace book
in the form of a digital garden
and have a zettlekasten-like system to help with organization

Obsidian

Tool for Thought app

- **Markdown**-based notes/wiki app
- Free for personal use
- **Data** is stored locally
  - 1 note == 1 file
- **Extensible** via API + JS plugins
- Fast search
- Active community



A note
in Obsidian

- **Markdown** file
- Frontmatter for **metadata** (tags, aliases, our custom data, etc)
- Codeblocks, links to other notes, etc.
- Backlinks and **graph**

```
---
tags: state/bud on/dev/go
alias: ["Concurrency in Go", "Concurrency", "Concurrent", "Goroutines",
"Goroutine"]
---
```

I

Go 105

Go 105 supports Concurrency 5 via goroutines

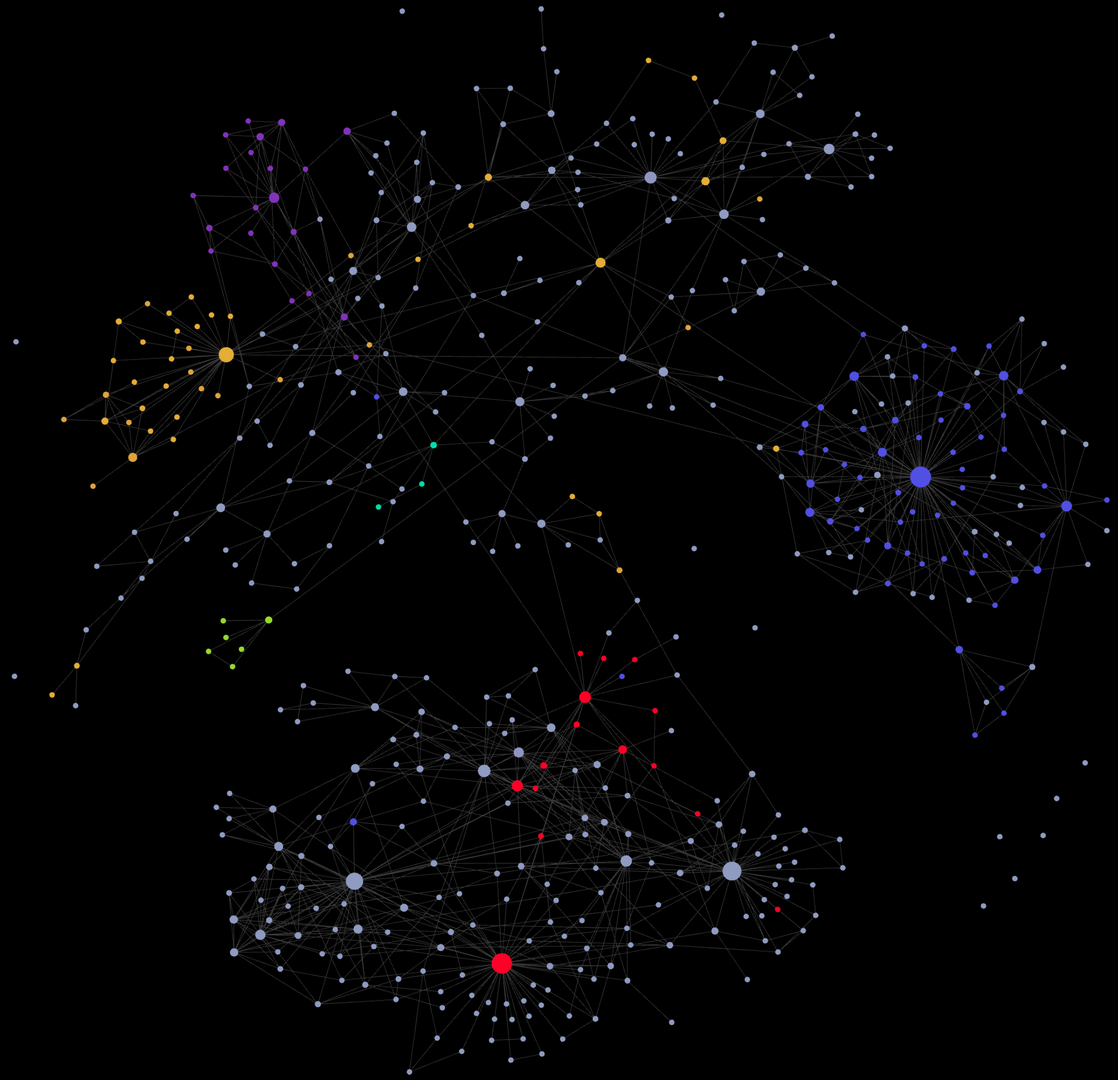
Go 105 supports Concurrency 5 via **goroutines**. A goroutine is a function that is capable of running Concurrently 5 with other functions. They are **lightweight** Threads managed by the Go runtime.

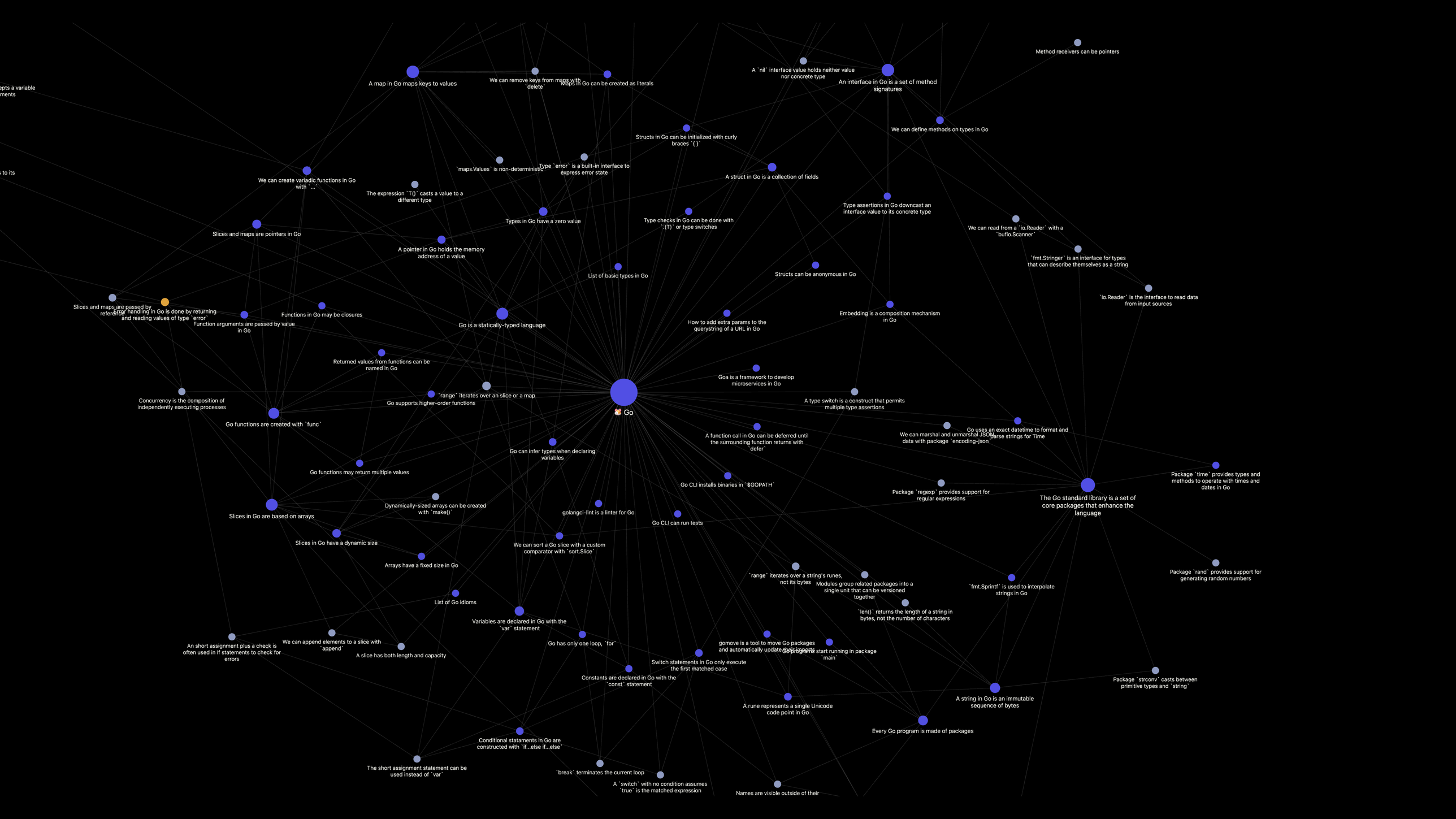
In Go 105, goroutines are created with the keyword `go`:

```go
func say(msg string) {
  fmt.Println(msg)
}

// ...
func main() {
  go say("hello")
  // ...
}
```

Goroutines run in the same address space, so access to shared memory must be synchronized. See Package `sync` implements basic synchronization primitives




Go
A map in Go maps keys to values
We can remove keys from maps with `delete`
Maps in Go can be created as literals
A `nil` interface value holds neither value nor concrete type
An interface in Go is a set of method signatures
Method receivers can be pointers
We can define methods on types in Go
Structs in Go can be initialized with curly braces `{}`
`maps.Values` is non-deterministic
Type `error` is a built-in interface to express error state
A struct in Go is a collection of fields
We can create variadic functions in Go with `...`
The expression `T()` casts a value to a different type
Type assertions in Go downcast an interface value to its concrete type
Types in Go have a zero value
Type checks in Go can be done with `.(T)` or type switches
We can read from a `io.Reader` with a `bufio.Scanner`
Slices and maps are pointers in Go
A pointer in Go holds the memory address of a value
`fmt.Stringer` is an interface for types that can describe themselves as a string
List of basic types in Go
Structs can be anonymous in Go
`io.Reader` is the interface to read data from input sources
Slices and maps are passed by reference
Error handling in Go is done by returning and reading values of type `error`
Functions in Go may be closures
Function arguments are passed by value in Go
Go is a statically-typed language
How to add extra params to the querystring of a URL in Go
Embedding is a composition mechanism in Go
Returned values from functions can be named in Go
Goa is a framework to develop microservices in Go
Concurrency is the composition of independently executing processes
`range` iterates over an slice or a map
Go supports higher-order functions
A type switch is a construct that permits multiple type assertions
Go functions are created with `func`
A function call in Go can be deferred until the surrounding function returns with `defer`
We can marshal and unmarshal JSON data with package `encoding-json`
Go uses an exact datetime to format and parse strings for Time
Go can infer types when declaring variables
Go functions may return multiple values
Package `time` provides types and methods to operate with times and dates in Go
Go CLI installs binaries in `$GOPATH`
Package `regexp` provides support for regular expressions
The Go standard library is a set of core packages that enhance the language
Dynamically-sized arrays can be created with `make()`
golangci-lint is a linter for Go
Slices in Go are based on arrays
Go CLI can run tests
Slices in Go have a dynamic size
We can sort a Go slice with a custom comparator with `sort.Slice`
Arrays have a fixed size in Go
Package `rand` provides support for generating random numbers
`range` iterates over a string's runes, not its bytes
Modules group related packages into a single unit that can be versioned together
`fmt.Sprintf` is used to interpolate strings in Go
`len()` returns the length of a string in bytes, not the number of characters
List of Go idioms
Variables are declared in Go with the `var` statement
An short assignment plus a check is often used in If statements to check for errors
We can append elements to a slice with `append`
Go has only one loop, `for`
gomove is a tool to move Go packages and automatically update dependents
Programs start running in package `main`
A slice has both length and capacity
Switch statements in Go only execute the first matched case
Constants are declared in Go with the `const` statement
Package `strconv` casts between primitive types and `string`
A string in Go is an immutable sequence of bytes
A rune represents a single Unicode code point in Go
Every Go program is made of packages
Conditional statements in Go are constructed with `if...else if...else`
The short assignment statement can be used instead of `var`
`break` terminates the current loop
A `switch` with no condition assumes `true` is the matched expression
Names are visible outside of their

My vault

Started ~2 years ago

- 1000+ notes
- For technical learning & reference
- Collecting ideas
- Also for other topics! Fiction writing, RPG DM’ing, cooking, etc.
- Some personal notes, simple checklists, etc.

How do I work with my vault?

And why?

1 note = 1 idea

Atomic notes

- I try to keep notes small and short.
- Lookup is easier this way
- Connecting to other notes is easier too
- Less friction

Testing in Rust [6]

A test is just a normal function with the `#[test]` attribute

In 🦀 Rust [49], we just need to use the `#[test]` attribute [1] to transform a regular function into a test [6].

```
#[test]
fn example() {
    assert!(true, "Panic message");
    assert_eq!(1 + 1, 2);
}
```

We can use the `assert!` and `assert_eq!` macros for Assertions [1].

The tests for a module are usually included in their own submodule, with the config `test`:

```
#[cfg(test)]
mod tests {
  use super::*;

  #[test]
  fn a_test_here() {
    // ...
  }
}
```

My own words

Make notes, not “take” them

- We learn best when we do an effort of **synthesizing**, **connecting**, etc.
- “Rubber ducking”, “Feynman technique”... all of them involve ***us* explaining** things
- An active effort in **curating** (avoid collecting & forgetting)
- Build your own **opinions**

🐹 Go 105

Error handling 1 in 🐹 Go 105 is done by returning and reading values of Type `error` 4

For Functions 11 that return `error` 4, a `nil` value means success, while a non-nil value indicates that there has been an error:

```
js, err := json.Marshal(data)
if err ≠ nil {
  return err
}
```

In a way, Go's error handling is similar to Error handling in Rust 3, with the exception that Go's Type `error` 4 is not an Either monad 2 and does not have the `?` operator. This makes error handling in Go half-baked and terrible repetitive, with a lot of boilerplate code.

Opinionated

Related:

- An short assignment plus a check is often used in If statements to check for errors 2

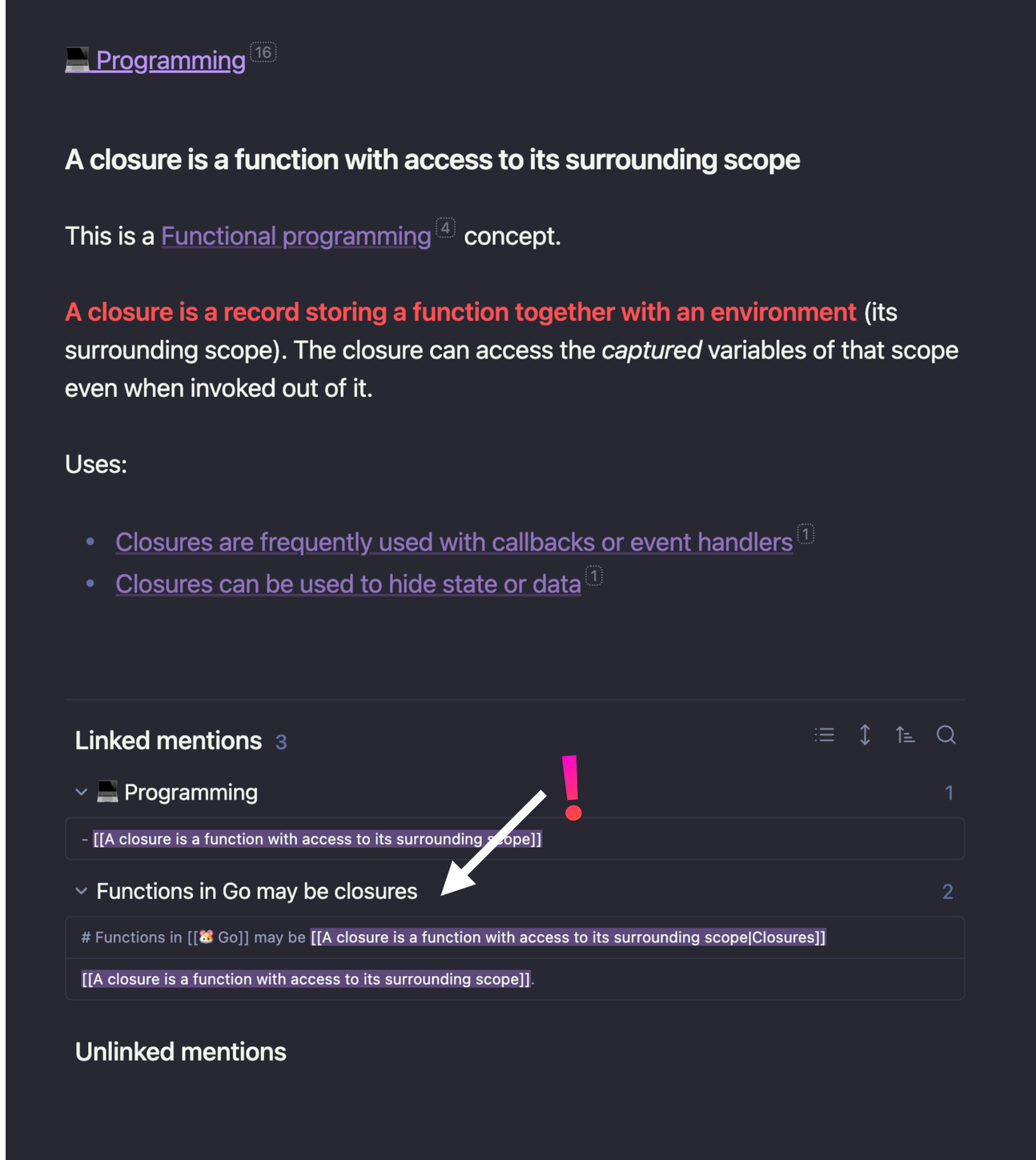
Further reading:

- *Error handling in Go* (article from the official Go blog)

Link, link, link!

Make connections between notes

- A note with connections:
  - Enhances discoverability
- But also...
  - Improves our learning
  - Might spark new ideas



Always WIP

The goal is “evergreen” notes

- I use tags to categorize stages:
- 🌱 **#seedling**: just created, need more connections
- 🌿 **#bud**: more worked on, but still need more connections or development
- 🌲 **#forest**: curated, well-connected evergreen notes

```
---
tags: state/seedling on/gamedev/pico8
---
```

PICO-8 has a hidden 64x64 mode

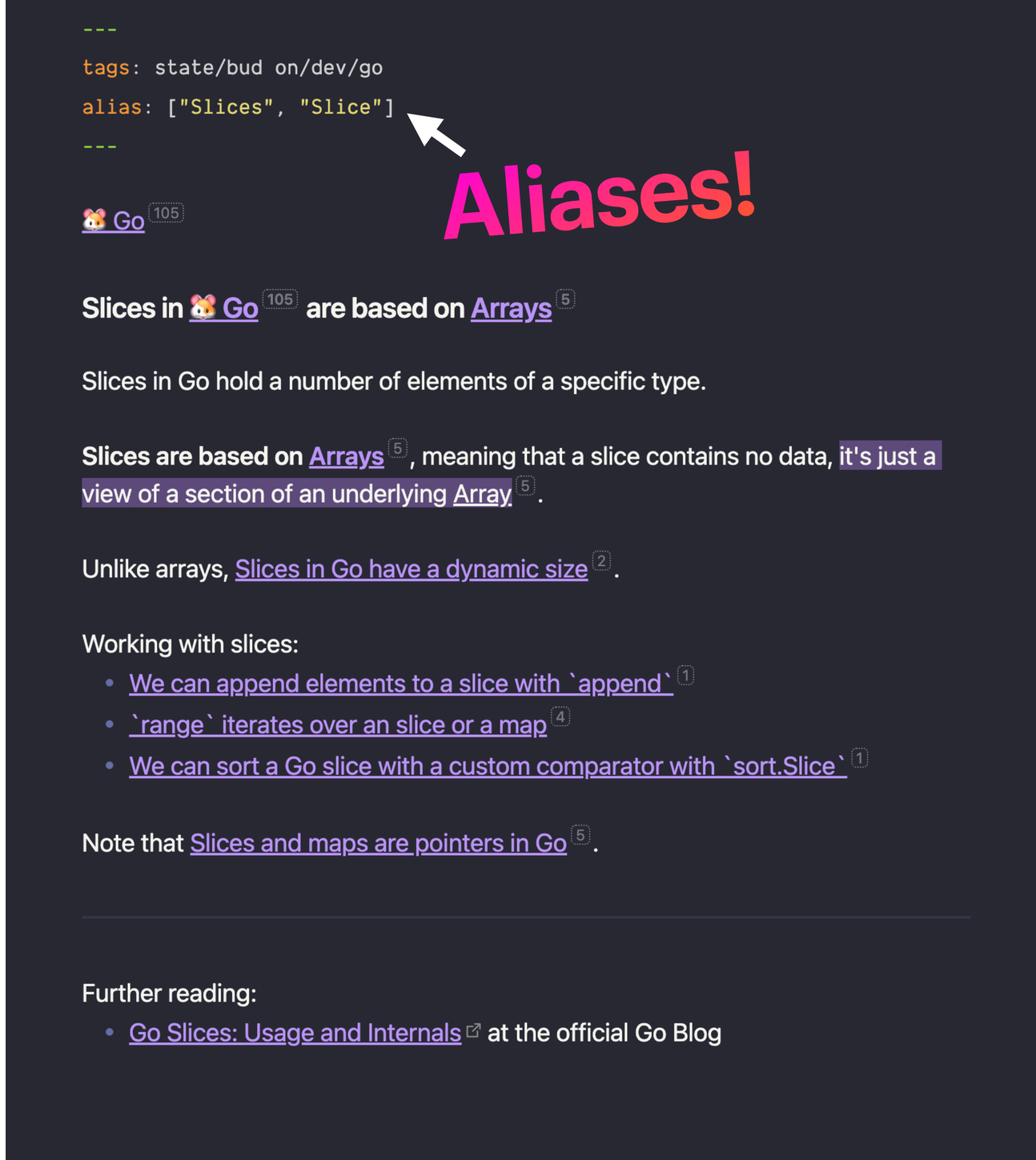
We can enable this mode with:

```lua
poke(0x5f2c,3)
```

Titles as statements

Embrace a flat structure

- I use a mostly **flat note structure** to avoid thinking “where do I file this note”
- Instead of “Go slices” I have notes like: *“Slices in Go are based on arrays”*, *“Slices in Go have a dynamic size”*, *“Slices and maps are pointers in Go”*, etc.
- Avoids filename collisions
- Helps keeping your notes **atomic**



Let’s see some types of notes

It might give you ideas for your own vault

An evergreen note

A note in my own words

- Either my own thoughts or...
- ...a synthesis of multiple sources

Front-end web development [10]

All JavaScript [15] frameworks aim to solve the same problems

All Front-end web development [10] JavaScript frameworks [3] (like React [16], Vue [1], etc.) aim to solve the same problems:

- Re-use common **UI widgets**.
- **Data-binding**. That is, keeping in sync the state of the application with the UI. *Both ways*.
  - For this we need a way to update the DOM that is performant.
- Provide a high-level **architecture**. Some frameworks implement this out of the box, like Angular, but even though most frameworks are agnostic about this, in reality there are plugins (some of them official) that provide this and have become a *de facto* standard, like Vuex or Redux (which implement Flux).

An idea

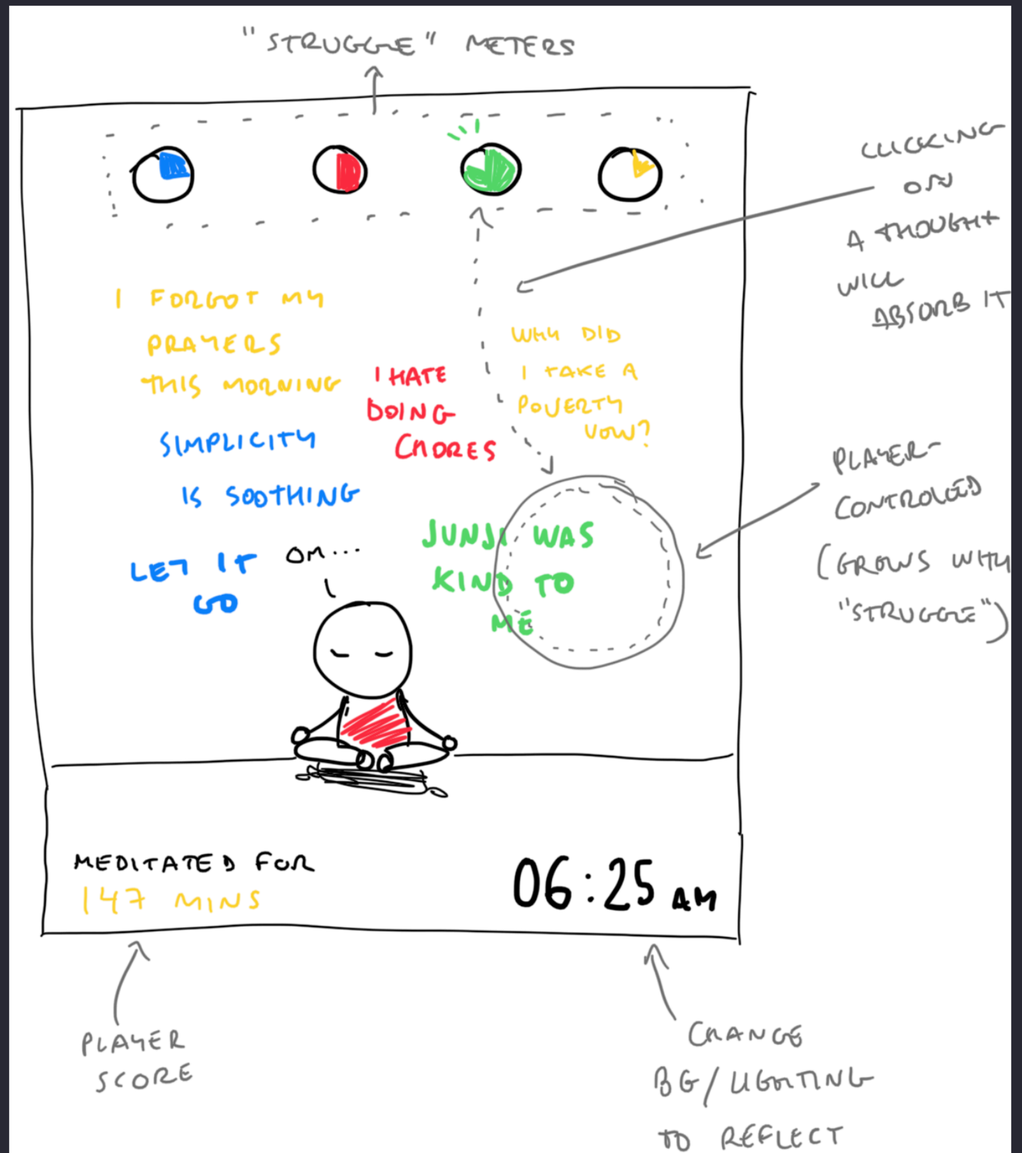
So I don’t forget

- Ideas for side projects, such as games, apps, etc.
- Ideas for characters or pieces of world building for fiction writing / TTRPG, etc.
- It helps giving **peace of mind** and making sure I’m not forgetting these things 😌

List of game ideas [3]

A meditation casual game

Concept:



A literature note

Sometimes I do *take* notes

- Capture quotes, fragments of videos, etc.
- Grab the **highlights** of an ebook I read and compile the most relevant.
- Summarize **3-5 key points** of a book, lesson, etc. as takeaways (so I don’t have to read all the highlights/notes)

There's only one principle in software: "High cohesion, low coupling"

Author:: Flipper
URL:: https://twitter.com/flipper83/status/1591734073667588098

> **Quote**
>
> The more I code, the more sure I am there's only one principle in software: High cohesion, low coupling [5].
>
> It's an universal principle from which other techniques derive to reach that very same goal.
>
> —Flipper [4]

This is a quote about ⚒ Software engineering [11] and the "High cohesion, low coupling" principle [5].

A Map of Content

Like an index, but with superpowers

- *Your* **gateway** to a topic.
- It’s developed over time, as you add notes about a topic and make connections.
- It’s not a mere list of notes: add your own words to make it personal.

⛩ 日本語

This MOC [14] is about the Japanese language. This topic is related to 📕 Language learning [12] and Japan [5].

See Japanese expressions [44] or how to say things.

Some important 文法 (Grammar) [18] concepts are 助詞 (Particles) [50].

Other high-level concepts are:

- Japanese is a economical language [9]
- Pitch accent is important in Japanese [1]
- Japanese is a high-context language [4]
- Onomatopoeias are frequently used in Japanese [3]
- 相槌 are a form of expressing rapport [1]
- 四字熟語 (yojijukugo) are expressions composed by four kanji [2]

Japanese has different speech registers, including Casual speech [19], 敬語 (honorific speech) [4], etc. Also, Spoken Japanese [9] has its particularities.

In terms of my own learning, here is a list of my own Persistent puzzles [3] in Japanese.

I'm also compiling a collection of Japanese Resources [2].

Here's my 日本語の練習 (Japanese Practice) [1].

A learning log

So I can see my progress

- I like to do a summary after doing a code kata (what I practiced, which language I used, etc.)
- You can use this for other topics: language learning, a project development, etc.

Dependency inversion principle[4]: map, parser, etc. are passed down in the constructor rather than required by the `Rover` class.

✨ 2023-01-17: Mars Rover kata walkthrough (TypeScript)[1]

- In 🟦 TypeScript[29]
- Separating the parsing + execution of commands from the rover movement/turning logic (still not a full implementation of the Command pattern[5])
- Applied some SOLID principles[6]:
  - Dependency inversion principle[4], since the rover is passed objects for the map and the VM that executes commands. This is clearly seen too in the integration tests.
  - Liskov substitution principle[4] by depending on Interfaces[4] rather than concrete types.
  - Single-responsibility principle[5] by separating the map, the rover and the VM. Although it could have been improved (as well as applying Interface segregation principle[4]) if I had implemented the Command pattern[5], probably.

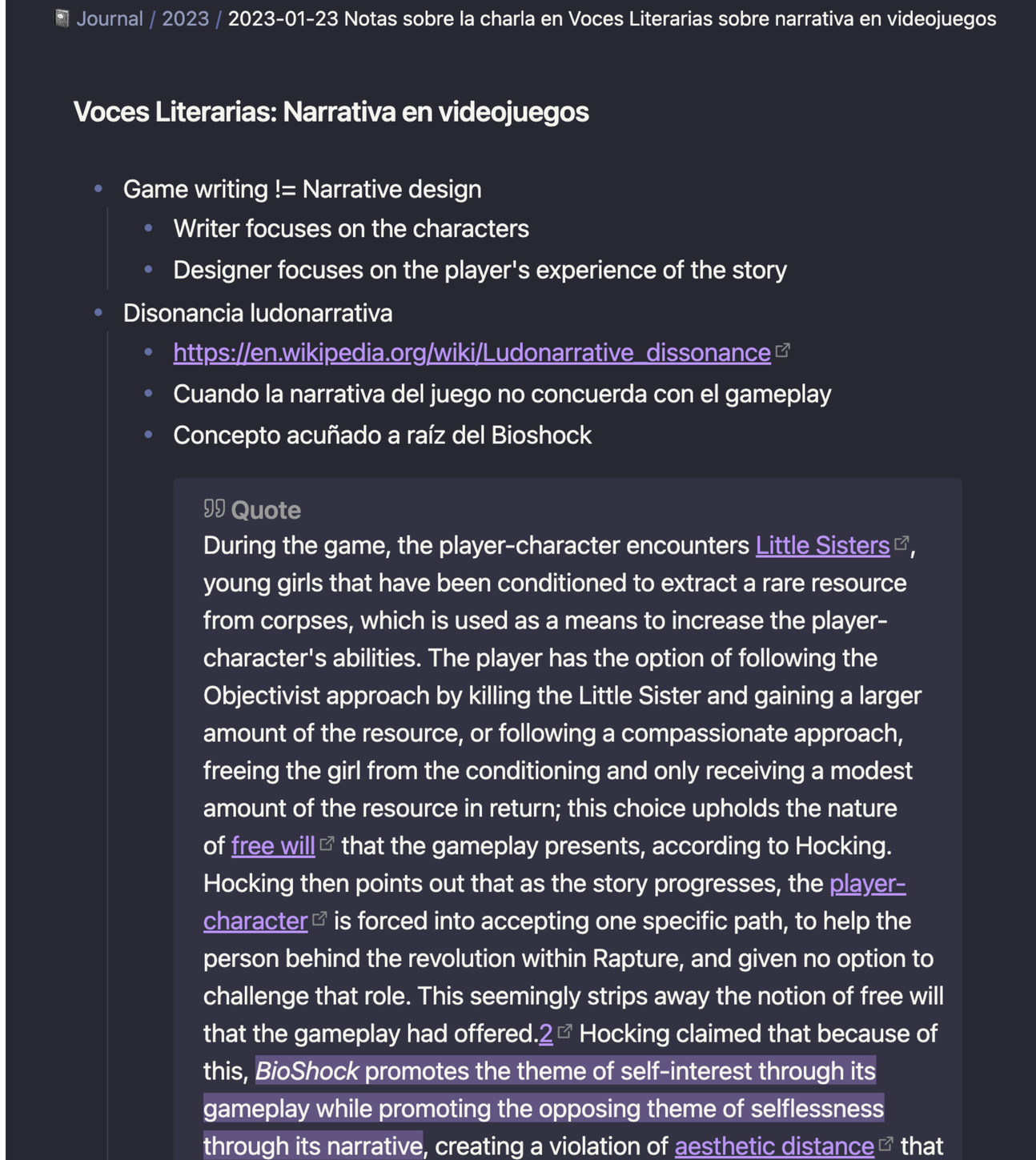
✨ 2023-02-05

- In 🦀 Rust[49] (code)
- With the Command pattern[5] for separation of concerns.
- SOLID principles[6]:
  - Dependency inversion principle[4] since the Rover is passed the map and the CPU objects.
  - Liskov substitution principle[4] since the different parts depend on Traits[1], not concrete types.
  - Interface segregation principle[4], since the traits are small and high-cohesive.

A dated entry

Some thoughts are tied to a day

- Since a end-of-the-year reflection…
- …to “I’m moving to a new city” checklist…
- …or “I just need a place to do a brain dump”



Why?

What’s the point of all of this?

Two modes of working

For your brain when learning

Focus

We’re actively engaged

- Reading
- Writing
- Active recall

Diffuse

We’re “chilling”

- **Making connections**
- Creative thinking

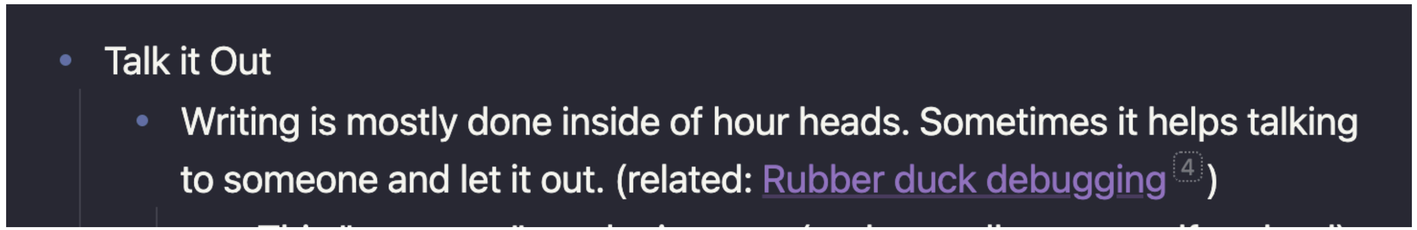
Source: "Learning How to Learn” by Barbara Oakley

Can we generate ideas on demand?

That’s the question we need to ask ourselves

(The answer is YES)

Ideas crossing boundaries



1. Can I bring other debugging techniques to my writing?
2. What can I bring from my writing to my coding?

Download starter vault

https://github.com/belen-albeza/obsidian-starter-vault

Resources
mentioned in the talk

- “Learning How to Learn” course in Coursera
- Obsidian app
- How to create MOCs video by Nick Milo
- Starter vault repository
- “A Brief History & Ethos of the Digital Garden” article
- The Zettlekasten Method website