JS Functional Programming Notes

ONLINE TOOLS

ramda REPL

https://ramdajs.com/repl/?v=0.22.1

learn ramda interactive

https://davesnx.github.io/learn-ramda/

javascript array explorer

https://sdras.github.io/array-explorer/

LIBRARIES

ramda

https://github.com/ramda/ramda

ramda-fantasy

https://github.com/ramda/ramda-fantasy

sanctuary

https://github.com/sanctuary-js

Either: https://github.com/sanctuary-js/sanctuary-either

crocks

Algebraic Data Types

https://crocks.dev/

https://github.com/evilsoft/crocks

DrBoolean github (Brian Lonsdorf)

https://github.com/DrBoolean

immutable

https://immutable-js.github.io/immutable-js/

immutable-ext

fantasyland extensions for immutable

https://github.com/DrBoolean/immutable-ext

lodash FP

https://github.com/lodash/lodash/wiki/FP-Guide

objectmodel

http://objectmodel.js.org/

monads

https://github.com/sniptt-official/monads

golang-underscore

https://github.com/ahl5esoft/golang-underscore

itertools for golang

https://github.com/yanatan16/itertools

TESTING

JSCheck

https://www.crockford.com/jscheck.html

fast-check

https://github.com/dubzzz/fast-check

https://github.com/dubzzz/fast-check/blob/master/documentation/HandsOnPropertyBasedJs.md

https://codesandbox.io/s/github/dubzzz/fast-check/tree/master/example?previewwindow=tests

JSVerify

https://jsverify.github.io/

REFERENCES / SPECS

Functional Programming Jargon

https://github.com/hemanth/functional-programming-jargon

Fantasy Land

https://github.com/fantasyland/fantasy-land

Fantasy Land specification guide

https://sanderv1992.github.io/fp/

Static Land

https://github.com/fantasyland/static-land

BOOKS

Mostly Adequate Guide to Functional Programming () (Brian Lonsdorf) (free eBook) ⭐

https://mostly-adequate.gitbooks.io/mostly-adequate-guide/content/

JavaScript Allongé, the "Six" Edition (Reg “raganwald” Braithwaite) (free eBook)

https://leanpub.com/javascriptallongesix/read

Eloquent Javascript

https://eloquentjavascript.net/

Learn You A Haskell

http://learnyouahaskell.com/chapters

Discover Functional JavaScript (Cristian Salcescu)

https://read.amazon.com/kp/embed?asin=B07PBQJYYG&preview=newtab&linkCode=kpe&ref_=cm_sw_r_kb_dp_cm5KCbE5BDJGE&reshareId=ZN6DYFMMW0106FVH5N9F&reshareChannel=system

https://www.amazon.com/dp/B07PBQJYYG

https://read.amazon.com/kp/embed?asin=B07PBQJYYG&preview=newtab&linkCode=kpe&ref_=cm_sw_r_kb_dp_cm5KCbE5BDJGE&reshareId=ZN6DYFMMW0106FVH5N9F&reshareChannel=system

Fundamentals of Functional Programming (Preethi Kasireddy)

https://gumroad.com/l/CkFTI

Functional Programming, Simplified

https://alvinalexander.com/scala/functional-programming-simplified-book/

Functional Programming Patterns: A Cookbook

https://www.freecodecamp.org/news/functional-programming-patterns-cookbook-3a0dfe2d7e0a/

CODE SAMPLES

Discover Functional JavaScript (Cristian Salcescu)

https://github.com/cristi-salcescu/discover-functional-javascript

ARTICLES

freecodecamp.org

https://www.freecodecamp.org/news/tag/functional-programming/

Functional Programming Patterns: A Cookbook (Karthik Iyengar)

https://www.freecodecamp.org/news/functional-programming-patterns-cookbook-3a0dfe2d7e0a/

Eric Elliott's "Composing Software" Series

https://gist.github.com/Geoff-Ford/51024380f4426d2bdca633d9217f9bcc

Handling null and undefined in JavaScript (Eric Elliott)

https://medium.com/javascript-scene/handling-null-and-undefined-in-javascript-1500c65d51ae

Functional Programming In JavaScript — With Practical Examples (Part 1) (rajaraodv)

https://medium.com/free-code-camp/functional-programming-in-js-with-practical-examples-part-1-87c2b0dbc276#.8dao66caq

Functional Programming In JavaScript — With Practical Examples (Part 2) (rajaraodv)

https://www.freecodecamp.org/news/functional-programming-in-js-with-practical-examples-part-2-429d2e8ccc9e/

A Beginner's Guide to Ramda (Part 1) — Currying and composition (Richard Tan)

https://itnext.io/a-beginners-guide-to-ramda-part-1-7e4a34972e97

Programming with Uncertain Values (Keith Alexander)

https://kwijibo.github.io/uncertain-values/

Closures, Curried Functions, and Cool Abstractions in JavaScript (TK)

https://www.freecodecamp.org/news/playing-around-with-closures-currying-and-cool-abstractions/

Free Sample Lesson: Pure Functions (Eric Elliott)

https://ericelliottjs.com/premium-content/lesson-pure-functions

WHAT YOU SHOULD KNOW ABOUT FUNCTIONAL PROGRAMMING (Tammy Xu)

https://builtin.com/software-engineering-perspectives/functional-programming

Functional Programming (various articles) (Jeremy Daly)

https://www.jeremydaly.com/functional-programming/

Functional JavaScript: Learn by refactoring (Jordan Eldredge)

https://jordaneldredge.com/blog/functional-javascript-learn-by-refactoring/

Functional Refactoring in JavaScript (Arnaud Lemaire)

https://medium.com/software-craftsman/functional-refactoring-in-javascript-c0fe718f4efb

A Quick Functional Refactoring Tip (George Rodier)

https://dev.to/grodier/a-quick-functional-refactoring-tip-4ifb

Refactoring Functions in JavaScript (Zeyad Etman)

https://dev.to/zeyadetman/refactoring-functions-kje

How point-free composition will make you a better functional programmer (Cristian Salcescu)

https://www.freecodecamp.org/news/how-point-free-composition-will-make-you-a-better-functional-programmer-33dcb910303a/

Monoidal Contravariant Functors are actually useful! (Brian Lonsdorf)

https://medium.com/@drboolean/monoidal-contravariant-functors-are-actually-useful-1032211045c4#.polugsx2a

LISTS

Awesome FP JS (Curated list)

https://project-awesome.org/stoeffel/awesome-fp-js

Awesome Functional Programming

https://awesomeopensource.com/project/xgrommx/awesome-functional-programming

VIDEOS

Immutable Data (Lee Byron, Facebook - React.js conf 2015)

https://www.youtube.com/watch?v=I7IdS-PbEgI&feature=youtu.be

Functional Programming Patterns for the Non-mathematician (Brian Lonsdorf) 2014

very fast tour of fp patterns

https://www.youtube.com/watch?v=AvgwKjTPMmM

Brian Lonsdorf on vimeo.com

https://vimeo.com/user7981506

A Million Ways to Fold in JS (Brian Lonsdorf) 2016

quick overview of recursion in FP

https://www.youtube.com/watch?v=JZSoPZUoR58

Oh Composable World! (Brian Lonsdorf - React Rally) 2016

a bit more in depth

https://www.youtube.com/watch?v=SfWR3dKnFIo

Hardcore Functional Programming in JavaScript, v2 (course) (Brian Lonsdorf)

https://frontendmasters.com/courses/hardcore-js-v2/

JSJ 388: Functional Programming with Brian Lonsdorf (podcast)

https://devchat.tv/js-jabber/jsj-388-functional-programming-with-brian-lonsdorf/

Functional JS - Working with ADTs - 01 (TheEvilSoft - Crocks)

https://www.youtube.com/watch?v=fIb1lOVh6cY&list=PLjvgv-FpMo7XRVFZjZsWXJ5nVmRJ5a5Hv&index=1

Category Theory (Bartosz Milewski)

https://www.youtube.com/playlist?list=PLbgaMIhjbmEnaH_LTkxLI7FMa2HsnawM_

The easiest way to make your code more functional (Eric Normand)

https://www.youtube.com/watch?v=3mbTa2wgmTU

Refactoring JavaScript code into tiny, pure, composable functions (Fun Fun Function)

https://www.youtube.com/watch?v=cUrEedgvJSk

Refactor Global Variables Out of Functions (Useful Programmer)

https://www.youtube.com/watch?v=6yFubyh1bvM

Conflict-Free Replicated Data Types (CRDT) for Distributed JavaScript Apps (TL;DR)

functional programming techniques for Google Docs-like shard apps

https://www.youtube.com/watch?v=M8-WFTjZoA0&t=61s

EXAMPLES

Turn a function that takes a value into one that takes a list (functor)

One use is to allow ordinary functions to take arguments wrapped in Maybe / Either

```
// mapToString :: (a -> b) -> ([a] -> [b])

const mapToString = R.map(R.toString);
```

Working with Result type

Quick print

```
console.log(res.inspect());
```

Run a "normal" function on it (one that doesn't take or return a Result)

```
res.map(func)
```

Run a "safe" function on it (one that doesn't take Result but does return a Result)

```
join(res.map(func))
```

Use a "normal" uncurried function that takes two arguments

```
liftA2(curry(Object.assign), res1, res2)
```

Note that `map()` can be thought of as `liftA1()`

Use the Result's value or handle error

```
res.either(errHandlingFunc, successHandlingFunc)
```

fold = join(map()) (like map but unwrap)

fold == either

unfold = while loop

chain =flatmap

lazy promise .fork() == Result.either()

NOTES ON VIDEOS FROM TheEvilSoft YOUTUBE CHANNEL (creator of Crocks https://crocks.dev/)

Ramda is still useful when using Crocks

Ramda contains many utility / helper functions designed for composition while Crocks focuses on implementing ADTs.

Applicative Functor

Needs two methods

- of()

    - takes an item and lifts it (wraps it) into the functional type *without delving into the item*. List.of(array) returns a 1-element list, where the element contains the entire array
- ap()
    - applies contained function(s) to another ADT (*of the same type?*), e.g. List.of(R.mergeRight).ap(List(arrayOfObjects1)).ap(List(arrayOfObjects2)) creates cross-product array1 x array2

Crocks ADTs

List

Wraps a list of values, similar to a javascript array.

- List(array) - creates a list of same length as array
- List.of(array) - creates list with 1 element, containing entire array
- List(function) and List.of(function) both seem to create a 1-element list that works the same

Pair

Wraps a pair of values

- supports .bimap(f1, f2), which will return a Pair(Pair[0].map(f1), Pair[1].map(f2))
- map(f) implementation: will apply function to second element, returning Pair(Pair[0], Pair[1].map(f))
- chain(f): imposes some requirements on f (https://crocks.dev/docs/crocks/Pair.html#chain)

State

- Monad, wrapping stateful calculations

Interface methods on (some) ADTs

chain(f)

Apply function f to inside of ADT, return an ADT containing the result. The function f must return an ADT(?) of the same type(?)

map(f)

- takes a function, applies internally to data type, returns same data type
- example 1: Lists, arrays - apply the function to each element, constructs a new List or array
- example 2: Result - if it wraps an Err, will ignore function and return Result(Err()). If it wraps an Ok(value), will return Result(Ok(f(value))

reduce(f, acc, data)

- reduce enumerable data to a single value, starting with acc(umulator) and calling f(acc, data[n]) repeatedly

Crocks helpers

assign(object1, object2)

Like Javascript Object.assign(), except instead of modifying one of the objects, will create a new object with shallow copies of the k/v pairs from original objects. Also filters out keys with a value of *undefined*.

liftA2(functionWith2Args, arg1, arg2)

Takes a standard function, wraps it in a functional wrapper that supports ap(), then applies it to wrapped arg1 then wrapped arg2, then unwraps and returns the result.

Functional equivalent for 'return x'

.map(constant(x))

Chaining different ADTs together

Use transformation functions e.g.

result_1.chain(maybeToResult(function_returning_maybe))

NOTES ON JavaScript Allongé, the "Six" Edition

https://leanpub.com/javascriptallongesix/read

Lesser known JS features

Comma

Takes two arguments, evaluates them both, and itself evaluates to the value of the right-hand argument: `(1 + 1, 2 + 2) => 4`

Can be used to trigger side effects with expression in the left-hand argument.

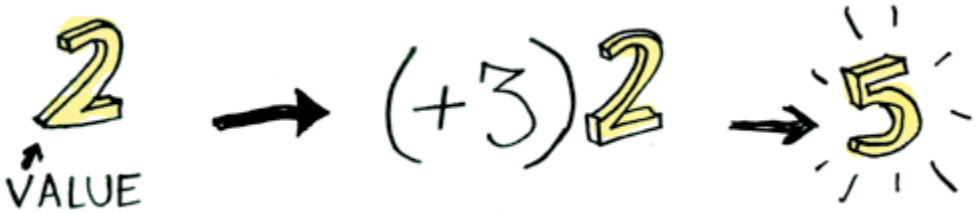
void / undefined

`void X => undefined`, always. By convention, use void 0. More bullet-proof than using literal *undefined* because *undefined* can be reassigned to a different value but *void* cannot.

Diagrams

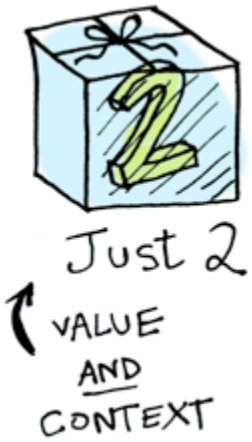
(from https://www.adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html)

'Regular' value

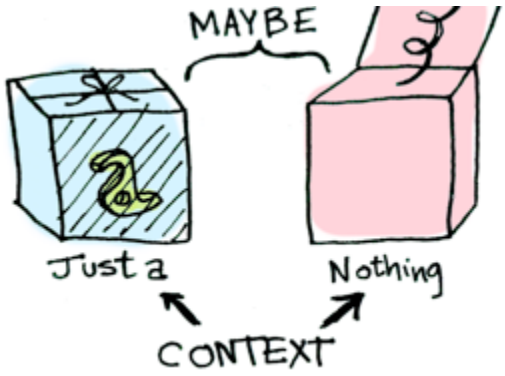


Functor (mappable)

Values are wrapped in a context

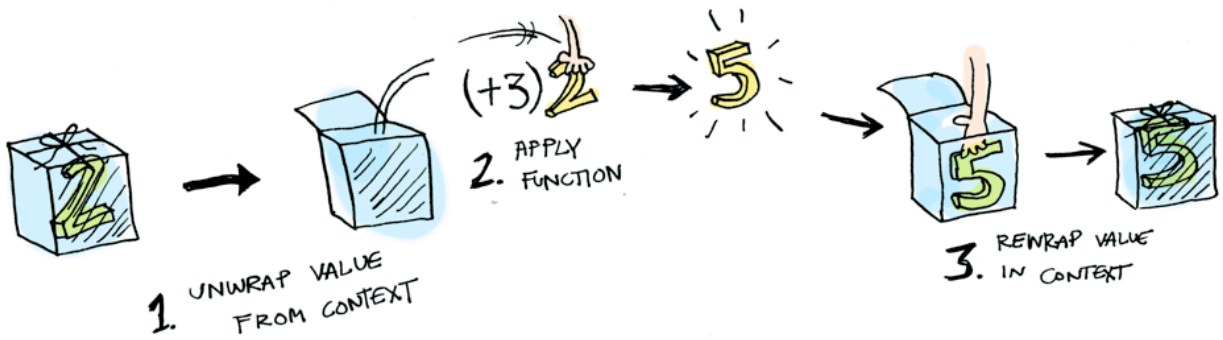


The "Maybe" functor

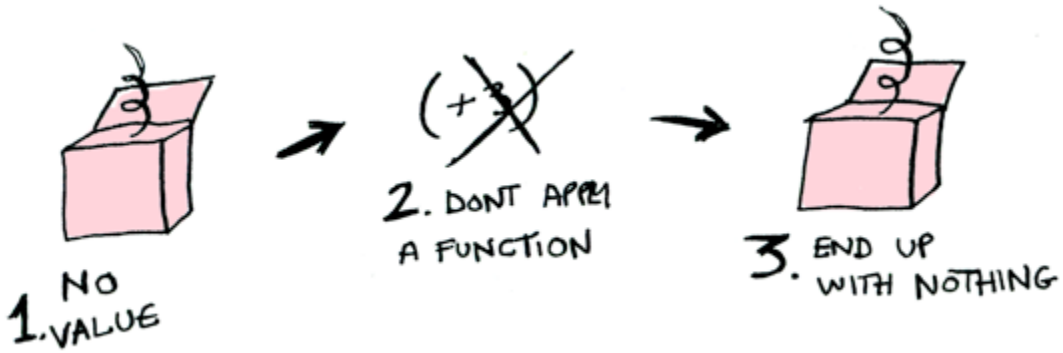


Mapping a function to a functor

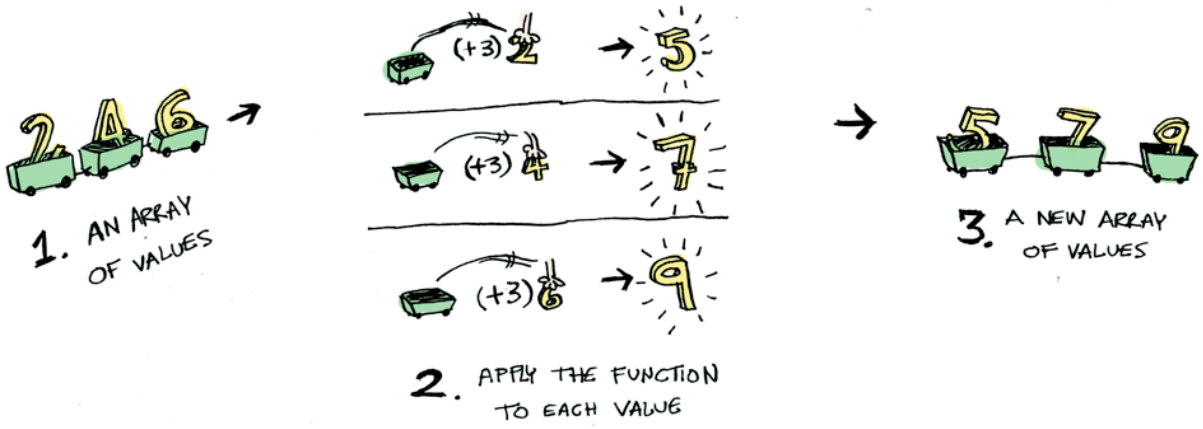
Return value is wrapped in same type of functor



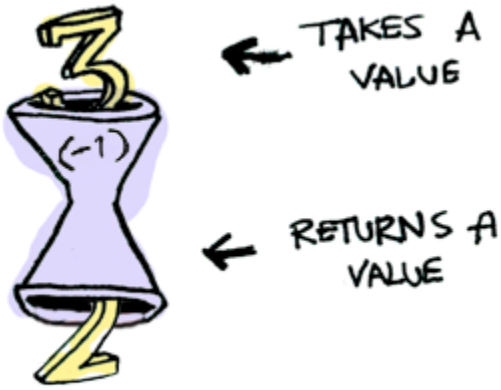
Mapping a function to a "Nothing" functor



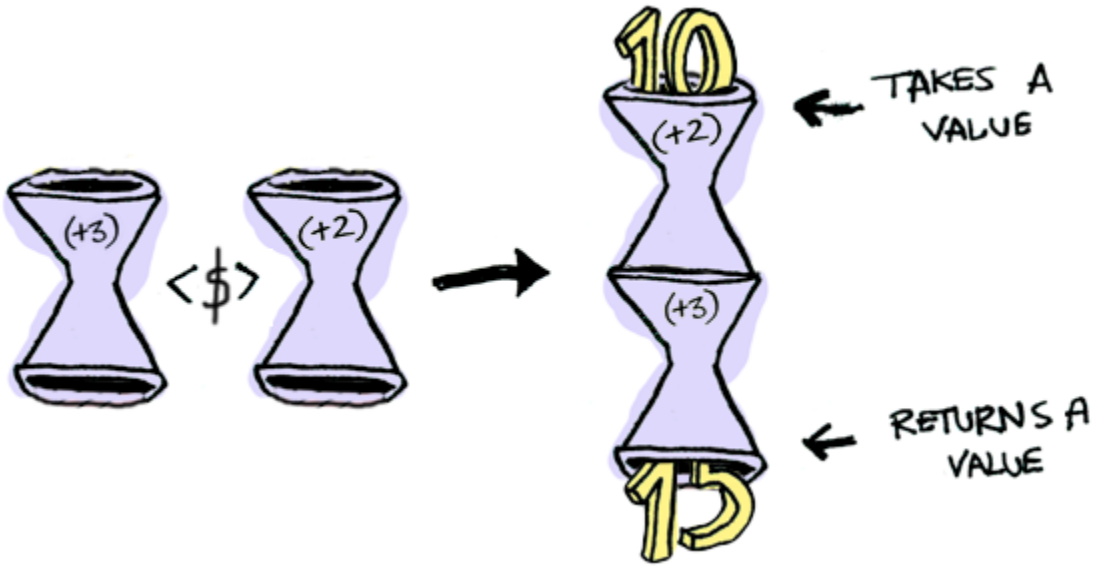
Mapping a function to an "Array" functor



'Regular' Function



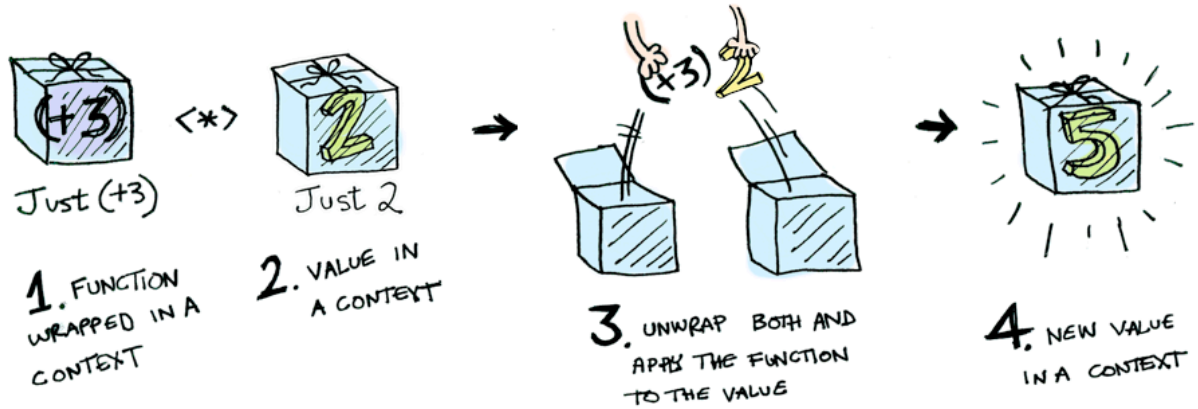
Compose functions



Applicative

Functions are wrapped too, not just values

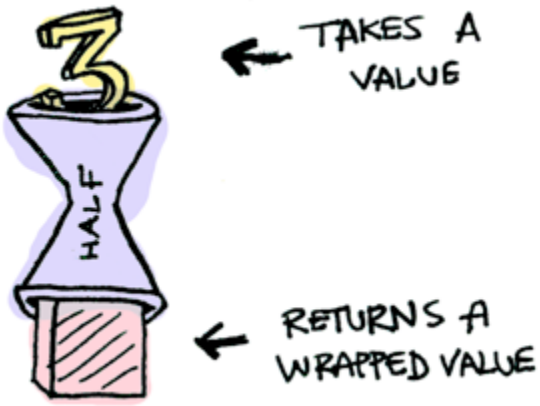
Apply a wrapped function to a wrapped value



Function that takes a 'regular' value and returns a wrapped value

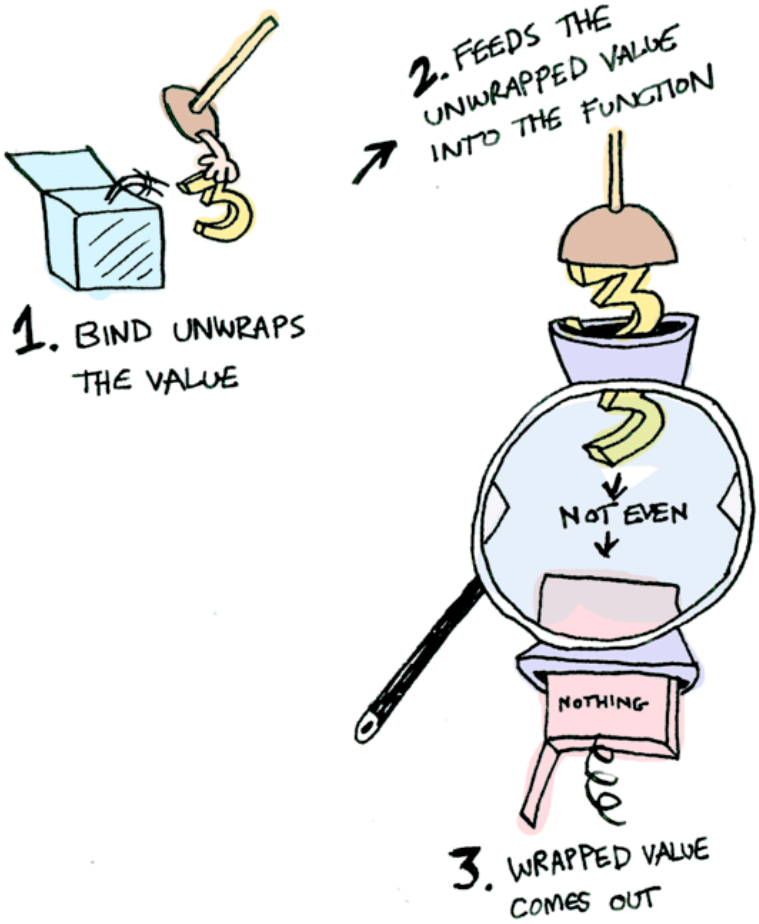
Function 'Half'

const half = x => x % 2 === 0 ? Just(x/2) : Nothing



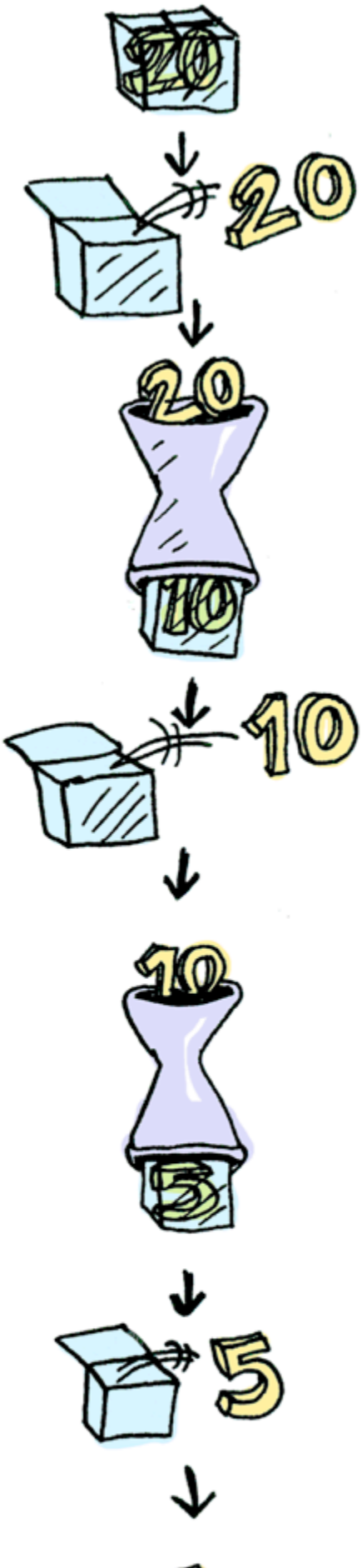
Monad takes a wrapped value and adapts to feeds to such a function)a function that takes regular values and returns wrapped ones)

'Maybe' monad, using function 'hal()'

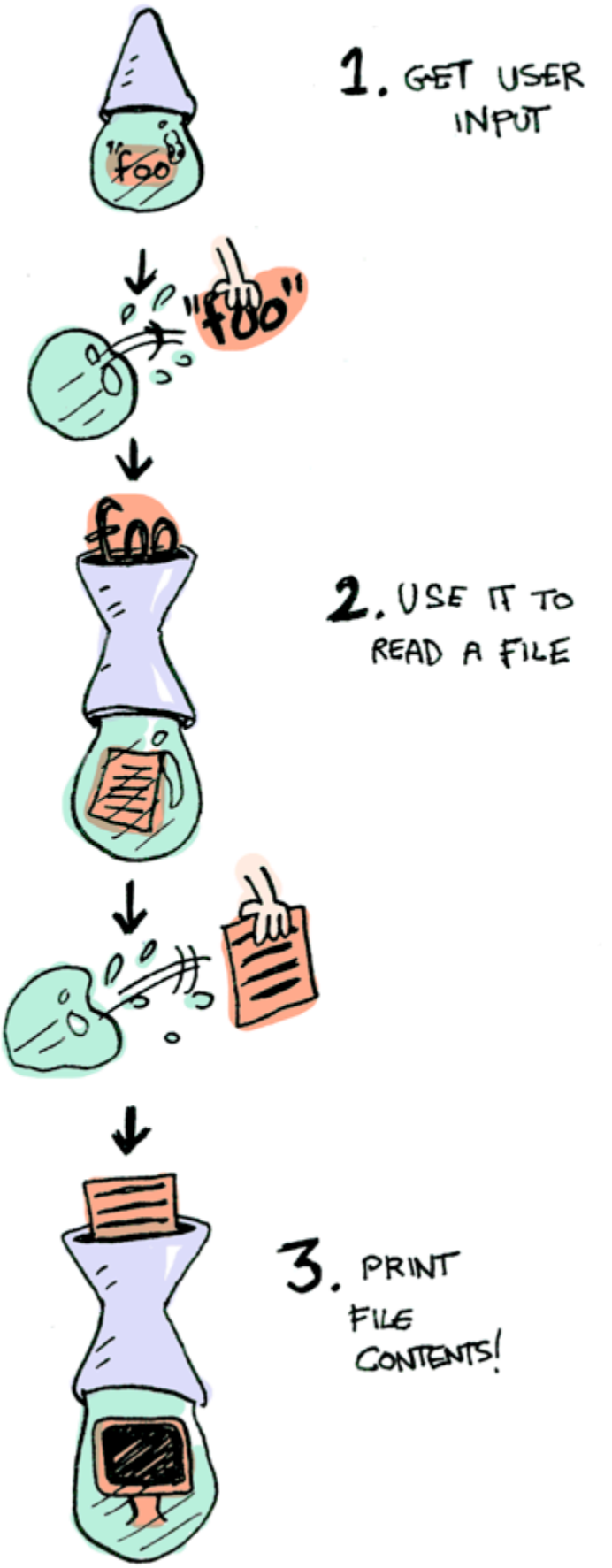


Monads can be chained

"Maybe" monad, fed into 'half()' repeatedly


20
20
20
10
10
10
5
5

Another example: (the IO monad)


"foo"
"foo"
foo
1. GET USER INPUT
2. USE IT TO READ A FILE
3. PRINT FILE CONTENTS!

Summary

Something can be more than one of the following (e.g. 'Maybe' is a functor, an applicative, AND a monad.

- **functors**: you apply a function to a wrapped value using *map()*
- **applicatives**: you apply a wrapped function to a wrapped value using *ap()*
- **monads**: you apply a function that returns a wrapped value, to a wrapped value using *chain()*