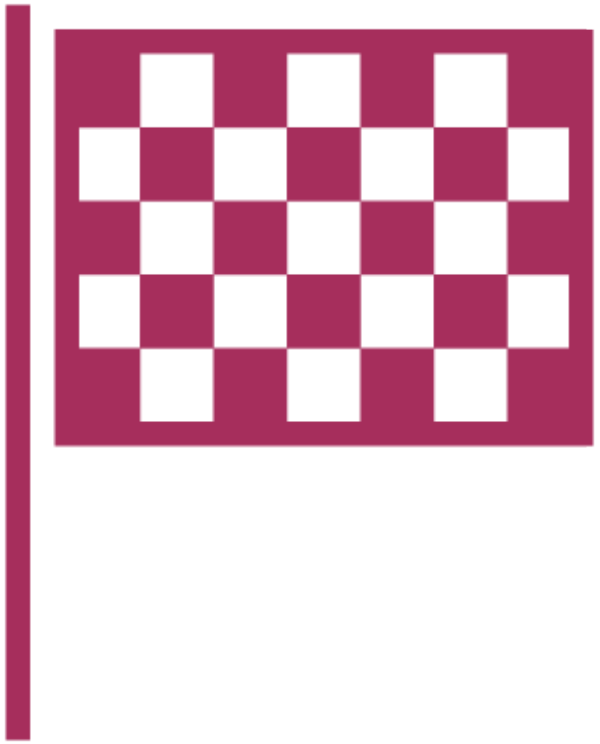
Functional JavaScript Libraries Playbook

A QUICK FUNCTIONAL JAVASCRIPT 101

David Mann

@MannD | Labs.HeirloomSoftware.com

The Libraries



- Immutable.js
- Ramda
- Folktale
- Fkit
- Sanctuary
- Monet

But What About *(insert name of library here)* ?

Current & Active

Fantasy Land Spec

Underscore & LoDash?

Course Overview

Functional Programming 101

Choosing a Library

Immutable.js

Ramda

Folktale

FKit

Sanctuary

Monet

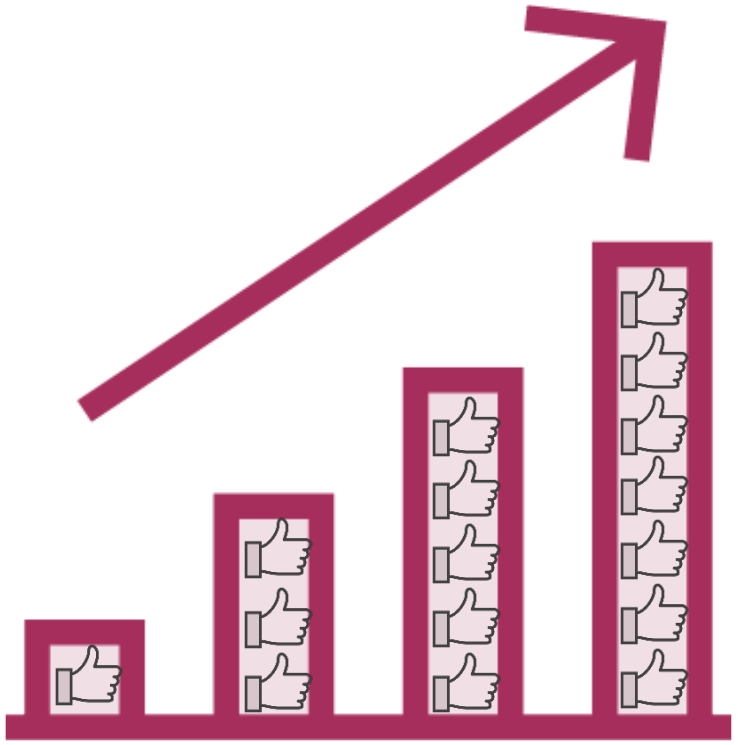
Using Functional JavaScript in Popular Frameworks/Libraries

Topics

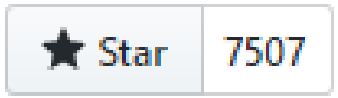
- Getting Started
- Pure Functions
- Composition
- Higher-order Functions
- Currying
- Immutable
- Closure
- A Visit to Fantasy-Land (The Spec)
- Type Notations

Functional JavaScript 101

The Important Parts



Beware the Hype Monster!



The *Wrong* Reason to pick a technology!

Why Functional Programming?

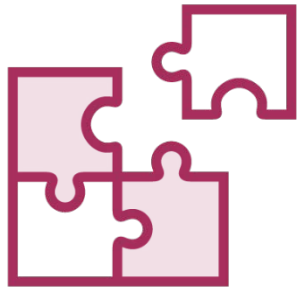
Predictable

Modular

Safe

Key Terms and Concepts

Pure Functions

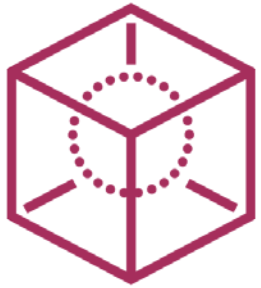


Composition

Higher-order Function

Currying

Immutable Data



Closure

Pure Functions

~~Side Effects~~

Same Output

No External Dependencies

```
const func1 = (a, b) => a + b;
```

Pure?

- No Side Effects
- Same Output for Same Input
- No External Dependencies

```
const c = 5;
const func2 = (a, b) => a + b + c;
```

Pure?

- ✓ No Side Effects
- ✗ Same Output for Same Input
- ✗ No External Dependencies

```
const func3 = (a, b) => a + b + func1(a, b);
```

Pure?

- ✓ No Side Effects
- ✗ Same Output for Same Input
- ✗ No External Dependencies

```
const c = 5;
const func4 = (a, b) => {
    c = a + b;
};
```

Pure?

- ✗ No Side Effects
- ✓ Same Output for Same Input
- ✓ No External Dependencies

```
const func5 = (a, b) => {
    console.log(a + b);
};
```

Pure?

- ✗ No Side Effects
- ✓ Same Output for Same Input
- ✗ No External Dependencies

```
const func6 = (a, b) => {
    func1(a, b);
    return a + b;
};
```

Pure?

- ✗ No Side Effects
- ✓ Same Output for Same Input
- ✗ No External Dependencies

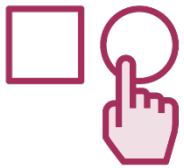
Pure?

No Side Effects

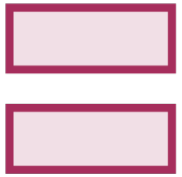
Same Output for Same Input

No External Dependencies

Pure Functions



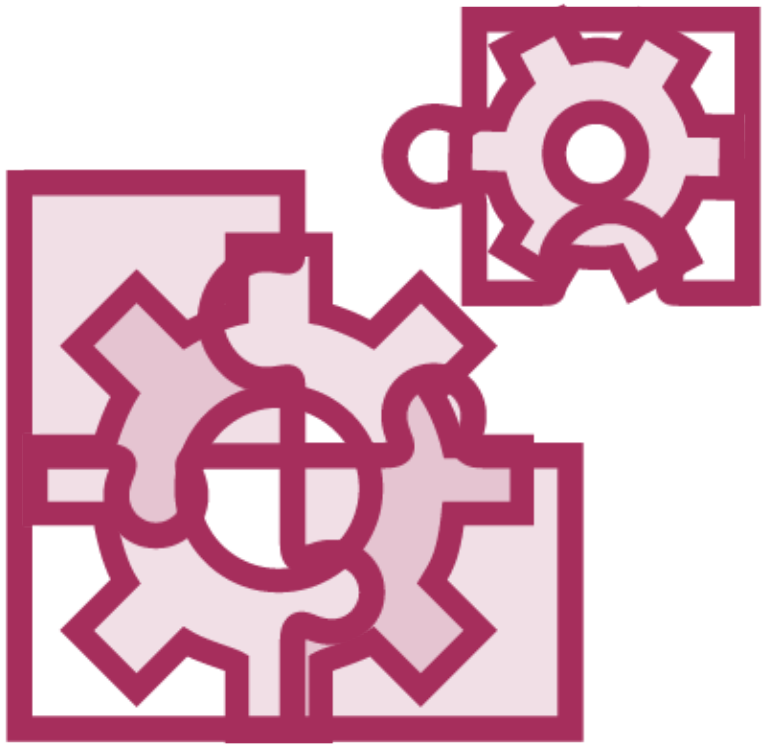
Separation of calculation and mutation



Always returns *something* (not *undefined or null*)

Highly testable

Composition



Functions as Building Blocks

Process / Machine

Composition

1

Functions do one thing

Functions named for that thing

Functions chained

Higher-order Functions

Functions as parameters

Returns a function

Higher-order Functions

```
function takesAFunction(fn, z){
    return fn('dave' + z);
}

function returnsAFunction() {
    return function(param1) {
        console.log('hello ' + param1)
    }
}


takesAFunction(returnsAFunction(), '!');

// hello dave!
```

Higher-Order Functions

```
function equivalentFirstOrderFunction(z){
    console.log('hello dave' + z);
}
```

Curry

func([1..n]) → func[1..n](1)

```
function add(a, b){
    return a + b;
}

function add5(b){
    return add(5, b);
}

add5(1); // 6
```

Partial Application

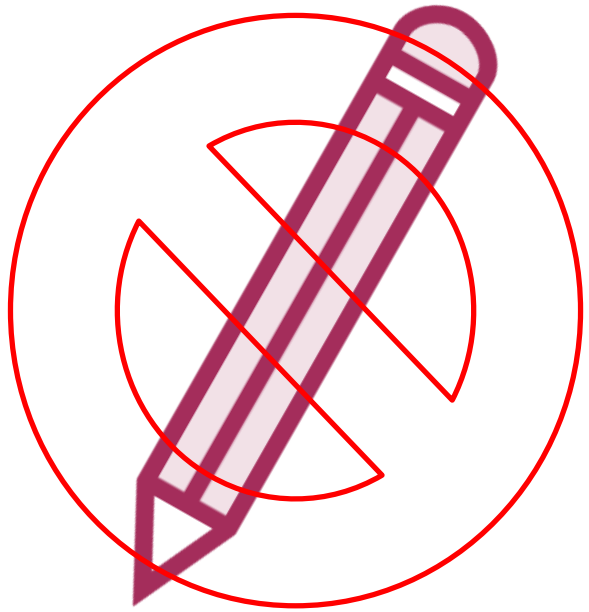
All currying is a partial application

Not all partial application is currying

```
function sum(a, b, c){
    return a + b + c;
}

function sum5(b, c){
    return add(5, b, c);
}
```

Immutable



~~Data Changes~~

Performance

Objects/Arrays

Immutable Objects

```
let obj = {
  Prop1 = 'hello',
  Prop2 = 'world'
};
```

No

```
obj.Prop1 = 'new value';
```

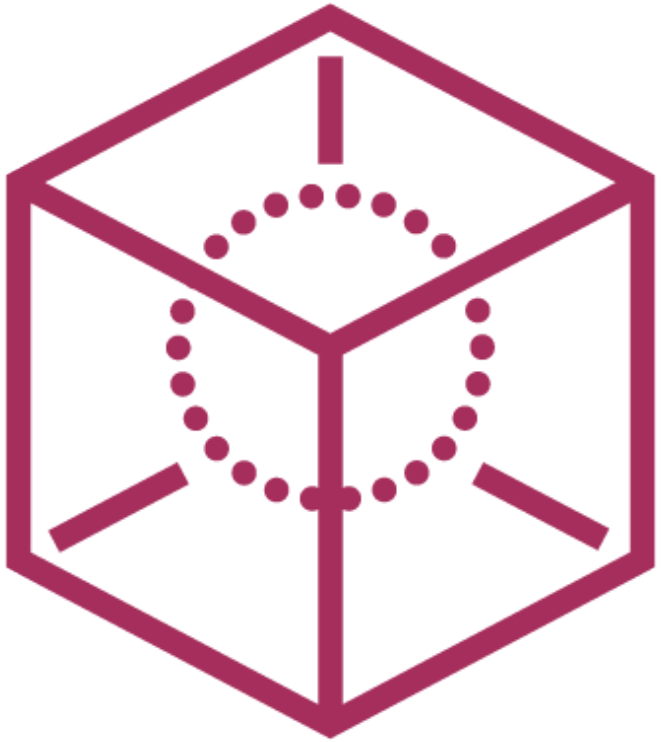
Yes

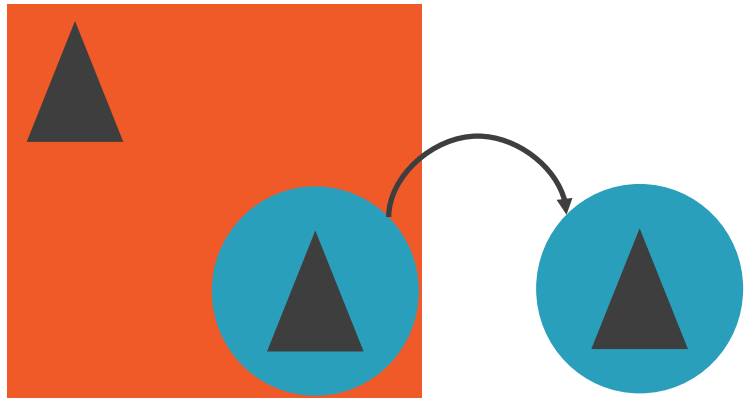
(Must return new object)

```
obj = obj.setProp('Prop1', 'new value');
```

```
setProp(propName: string, val: any): MyObject {
    // using lodash deepClone
    const rv = _.deepClone(this);
    rv[propName] = val;
    Object.freeze(rv);
    return rv;
}
```

Closure





```
function orangeSquare() {
    var triangle = 'hello from inside’;
    var blueCircle = function() {
        console.log(triangle);
    };
    return blueCircle;
}
var outsideVar = orangeSquare();

outsideVar(); // hello from inside
```

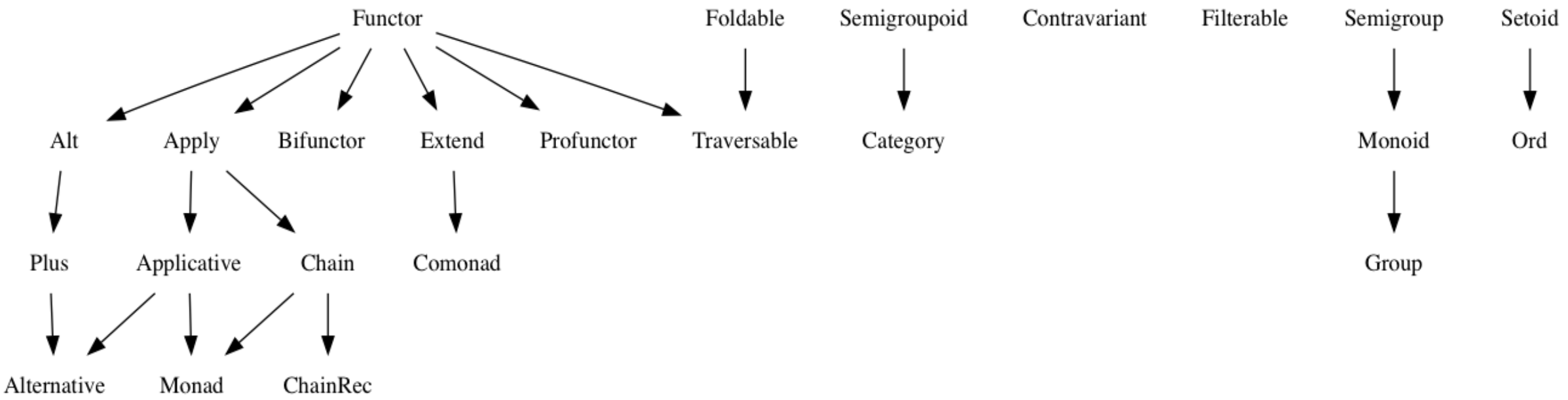
Fantasy Land?

So?

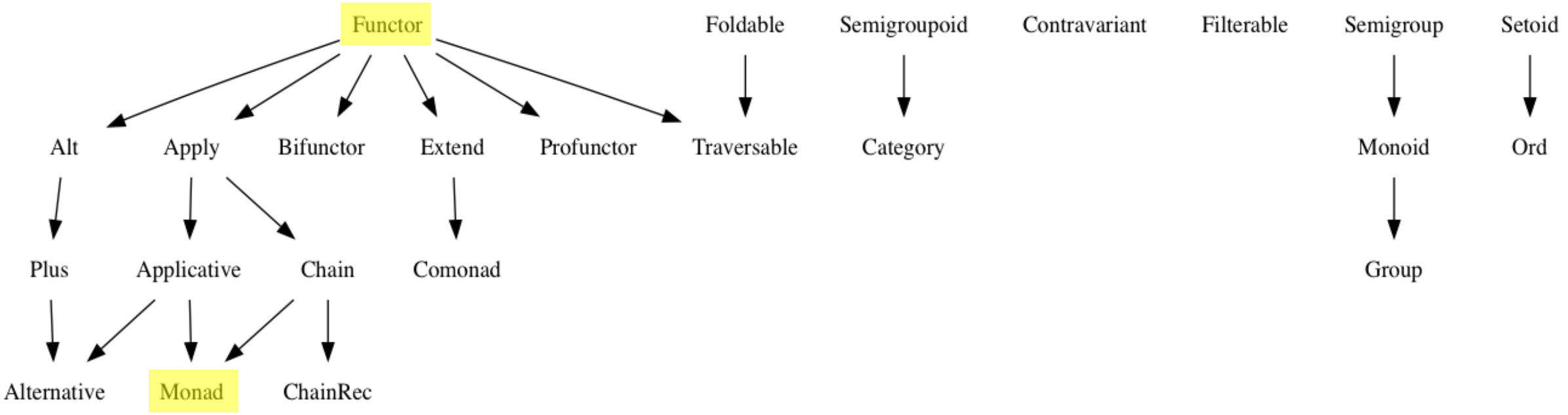
“Specification for interoperability of common algebraic structures in JavaScript”

- **Fantasy Land Specification**

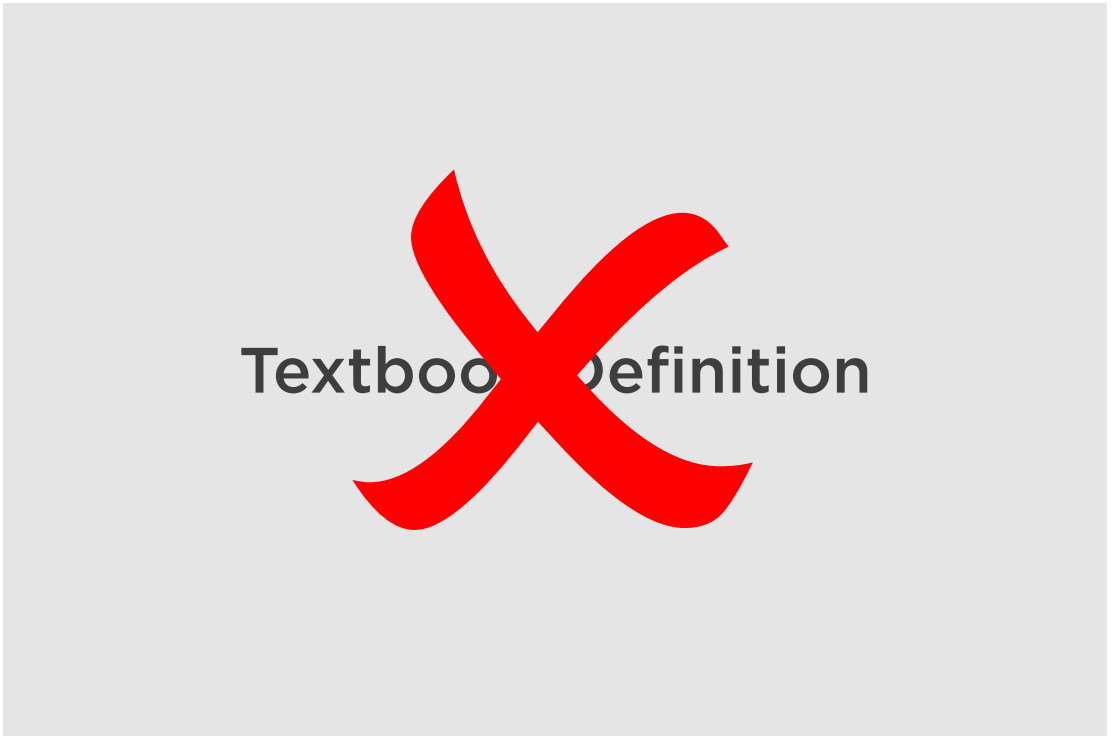
Algebraic Structures?



Algebraic Structures?



Goal



Simplified Definition

Typical Usage

Functor

Can be *mapped*

```
myFunctor.map(fn)
```

Monad

Type of *functor*

Container

In

Out

You’re probably already using Monads

#WhatChuTalkinAboutWillis?

Promises

Observables

Monad

Maybe

Either

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

Algebras

Setoid

1. `a.equals(a) === true` (reflexivity)
2. `a.equals(b) === b.equals(a)` (symmetry)
3. If `a.equals(b)` and `b.equals(c)`, then `a.equals(c)` (transitivity)

`equals` method

```
equals :: Setoid a => a ~> a -> Boolean
```

A value which has a Setoid must provide an `equals` method. The `equals` method takes one argument:

```
a.equals(b)
```

1. `b` must be a value of the same Setoid
    i. If `b` is not the same Setoid, behaviour of `equals` is unspecified (returning `false` is recommended).
2. `equals` must return a boolean (`true` or `false`).

Key Takeaways

~~Hype~~

Key Terms/Concepts

- Pure Functions
- Composition
- Higher-order Functions
- Currying
- Immutable Data
- Closure

Fantasy-Land

Next Up

Immutable.js