Transducers in JS

-Issues w/ imperative code
-How functional programing can help
-Leveraging function composition
-Code samples

Applications naturally grow in complexity

- The complexity is compounded as requirements begin to stack and more developers touch the code base.

- Dealing with these requirements with imperative code often makes our solutions less extensible and less convenient to maintain and share through out the code base.

- More code means more surface area for bugs to arise

The aim of function programing

“Functional programming's goal is to dominate a system's complexity through the use of formal models, careful attention is given to code's proprieties.”

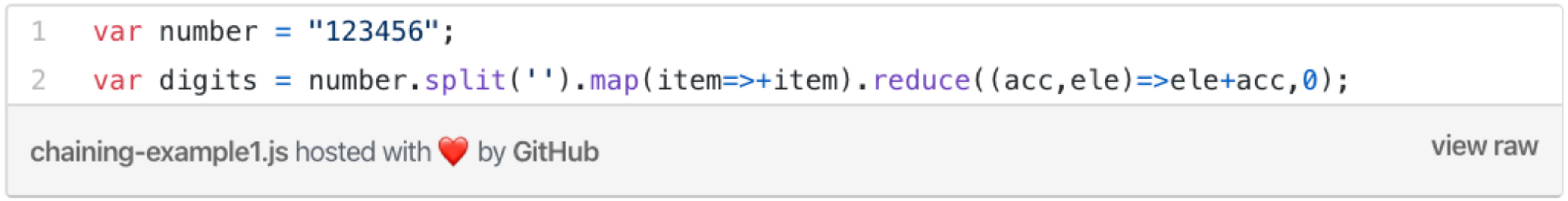
“FP will help teach people the mathematics behind program construction: how to write composable code, how to reason about effects, how to write consistent, general, less ad-hoc APIs” - Giulio Canti

Composition is King

- composition allows us to minimize the footprint of our code. Making it more legible and less difficult for other engineers (or our future selves) to understand

- we build smaller functions with single responsibilities and combine ("compose") in different orders which build transformative pipelines our data can flow through

Method Chaining

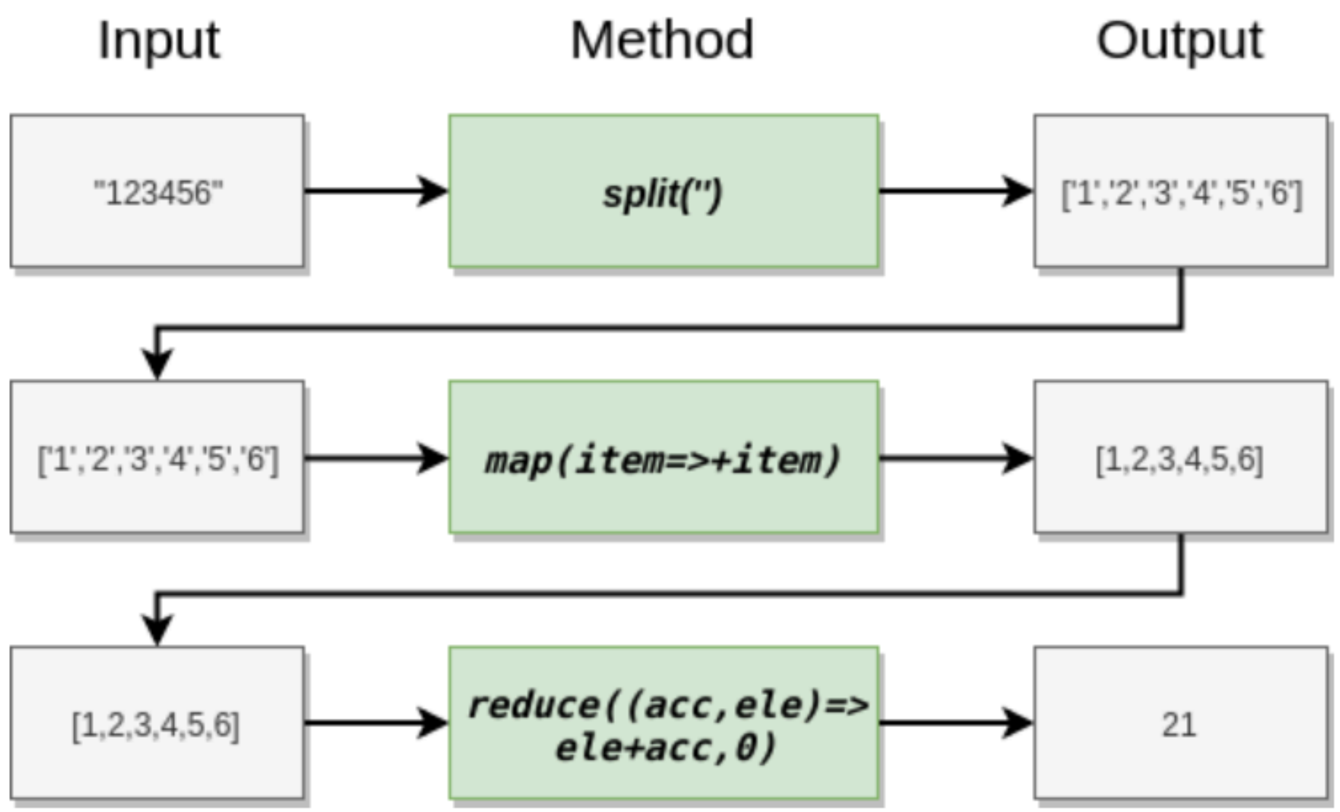


```js
var number = "123456";
var digits = number.split('').map(item=>+item).reduce((acc,ele)=>ele+acc,0);
```

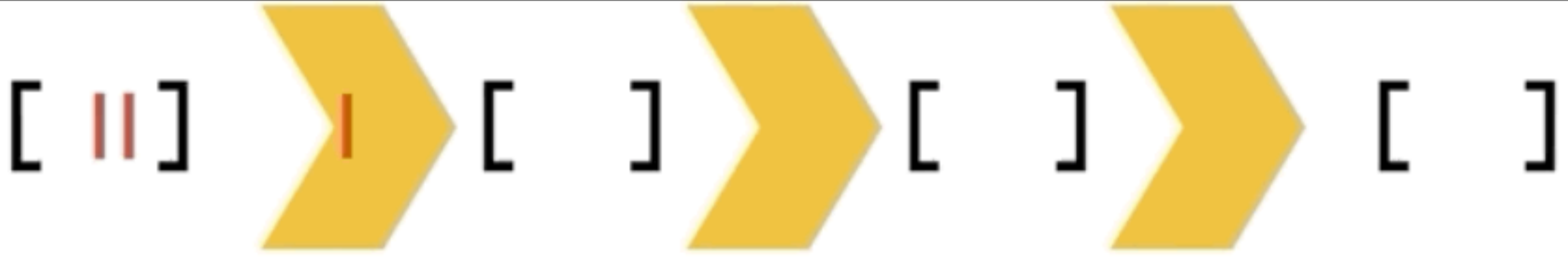
chaining-example1.js hosted with ❤️ by GitHub   view raw

The above example shows adding all the digits in a numeric string.

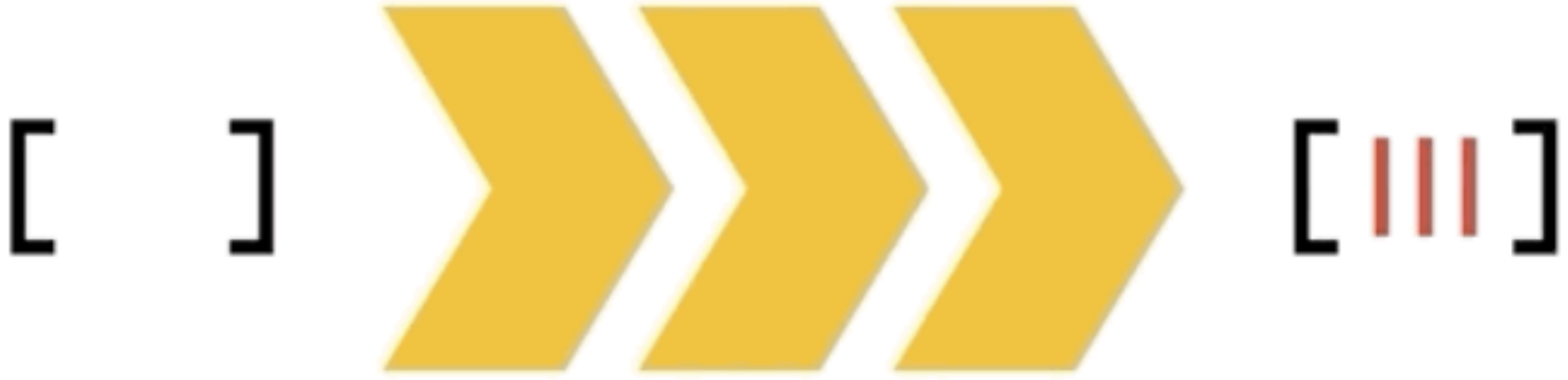
The execution of each of the methods will look like this:



Transduction at a high-level



chained transformations create intermediate arrays



transduced transformations process items one by one into output array

The foundation

```javascript
// simple reducer example
const reducer = (accumulator, currentValue) => {
  accumulator.push(currentValue);
  // if (predicate(currentValue)) {
  //  accumulator.push(currentValue);
  // }
  return accumulator;
};
collection.reduce(reducer, []);
```

```javascript
// compose higher order functions from right to left.
const compose = (...fns) => x => fns.reduceRight((v, f) => f(v), x);
```

Victor Boutté

Twitter - @monsieurBoutte
Github - github.com/monsieurBoutte

This repository:

github.com/monsieurBoutte/transduction-demo