Why They're Nice

Our lives revolve around abstraction—the practice of making complex things easier to increase productivity. HOFs can abstract us away from step-by-step procedures, instead opting for readable, reusable functions everywhere. (6 min. read)

I'd say every sizable JavaScript application you've written uses higher-order functions (HOFs).

We've seen `map`, `filter`, and `reduce`, of course.

```javascript
const nums = [1, 2, 3, 4, 5, 6, 7, 8];

const doubled = nums.map((x) => x * 2);
const evensOnly = nums.filter((x) => x % 2 === 0);
const total = nums.reduce((x, y) => x + y, 0);

console.log({
  doubled,
  evensOnly,
  total
});
```

But have we paused to think about what these HOFs are actually doing?

Not in terms of implementation, obviously, there's a loop and some internal state. I'm referring to these HOFs' collective purpose.

Abstracting Over Actions

We're taught that the simplest way to abstract a piece of logic is to put it in a function. JavaScript makes this fairly straightforward.

```javascript
const doThing = () => // do something
```

`doThing` probably "does a thing" with a string, number, or date and that's all we need to know. That logic's now safely tucked away.

This is abstraction **at the level of values**, which everyone does.

But we learned that JavaScript treats functions as values too! Using them as *data* allows for HOFs that can "do a thing" with functions instead of strings or numbers.

So we can abstract even higher, **at the level of actions**. Why's this great?

Humans Really Like Abstraction

Abstraction's a fundamental CS concept that makes us all more productive. Did you write your React/Redux app in machine code? Did you manually move each bit to get a working program?

You, like the rest of us, write JavaScript on an abstraction *that runs on an abstraction that runs on an abstraction that runs on an abstraction that runs on an abstraction*...I'm not sure where this ends.

Let's even use a scenario from everyday life: how do you ask for a cup of water? Do you list every step involved in getting it from the office pantry?

- Exit cubical
- Walk 10 steps
- Turn left
- Enter pantry
- ...

Or do you simply let your coworker figure out how to do it? The process of finding the pantry has been abstracted because your coworker (hopefully) knows where it is.

Whether its talking or doing, good abstractions let us focus on getting things done without sweating implementation details.

The question is how do we apply that level of abstraction to writing apps? How do we bridge the gap between the way we communicate about programs and the way we actually write them?
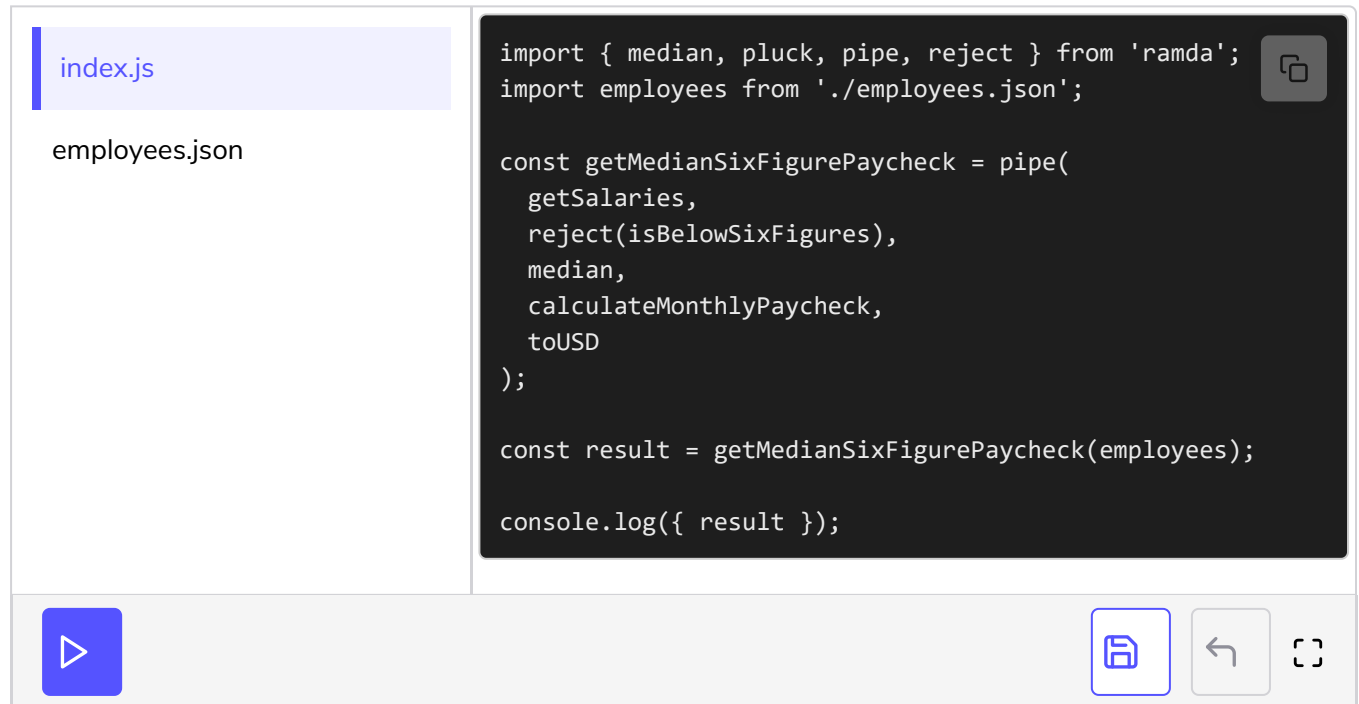
Let There Be HOFs

Higher-order functions are the mighty medium for coding this way. It's the

glue that bonds us and our code, letting us both communicate in terms of actions.

Without them, functional programming wouldn't be nearly as practical. But thanks to them everything can be declarative, abstracted at the value *and* action level, and remain coherent.

Resulting in snippets like this

index.js

employees.json

```js
import { median, pluck, pipe, reject } from 'ramda';
import employees from './employees.json';

const getMedianSixFigurePaycheck = pipe(
  getSalaries,
  reject(isBelowSixFigures),
  median,
  calculateMonthlyPaycheck,
  toUSD
);

const result = getMedianSixFigurePaycheck(employees);

console.log({ result });
```

It looks weird now, but we'll see it a lot more very soon. `pipe` lets you run functions in a sequence. Reading the steps, we get

1. Get all salaries from a list
2. Exclude salaries below six figures (< $100,000)
3. Find the median salary
4. Calculate how much they make each month
5. Format in dollars (USD)

This is the 1-to-1 relationship we want between our communication and code. FP can help us get there, with some practice.

Next, we'll cover the sugar to HOFs' spice: function composition. It makes everything nice!

Summary

- Functions let us communicate code through actions, allowing for more

meaningful and comprehensible discussions.

- *Higher-order* functions supercharge this by allowing us to abstract at an even higher level.

- Ramda takes full advantage of HOFs, letting you create functional pipelines with minimal syntax.