

Adding Ramda

A comparison of vanilla and Ramda solutions with map. Benefits of Ramda include currying, point-free, and the data-last pattern. (6 min. read)

We got everyone's first name using `map`.




```
const upperAndReverseFirstNames = (users) => {  
  return users.map(upperAndReverseFirstName);  
};
```

How much would Ramda help?

index.js

users.json

```
import { map } from 'ramda';  
import users from './users.json';  
  
const result = map(upperAndReverseFirstName, users);  
  
console.log({ result });
```



Not much. In fact, it's more complicated because we're importing `map` instead of just using the built-in array method.

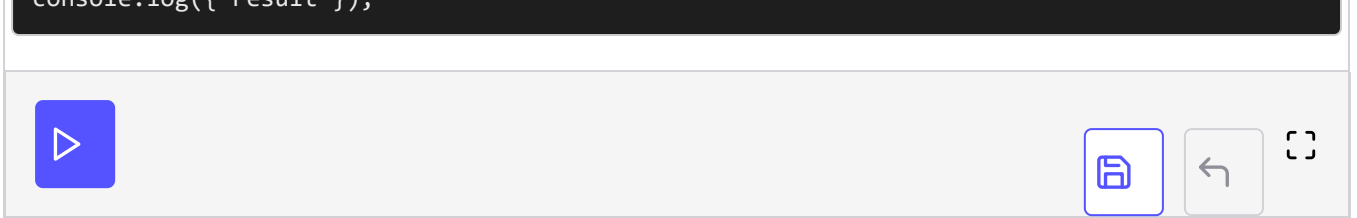
But here's something `Array.map` can't do...

index.js

users.json

```
import { map } from 'ramda';  
import users from './users.json';  
  
const upperAndReverseFirstNames = map(upperAndReverseFirstName);  
  
const result = upperAndReverseFirstNames(users);  
  
console.log({ result });
```





Everything's Curried

This, my friend, is one of Ramda's unique selling points that I mentioned earlier: **everything's curried**.

A curried function can be supplied some parameters now, and others later. And thanks to HOFs we can store the result of giving `map` one parameter in a variable to be used later.

So doing

```
const upperAndReverseFirstNames = map(upperAndReverseFirstName);
```

“preloads” Ramda's `map` function, returning a new function that expects a list to operate on. That's why the next line

```
const result = upperAndReverseFirstNames(users);
```

was possible because after receiving all required parameters, `map` went and did the work!

Point-Free

Also, notice that the mention of a `user` parameter has been eliminated. Those two functions don't explicitly say the data they're operating on. We've given them pretty good variable names so we know exactly what's happening, but do you see any parameters being specified?

This is a style enabled by HOFs and currying called **point-free** and it can be beautiful.

We've replaced this

```
const upperAndReverseFirstNames = (users) => {  
  return users.map(upperAndReverseFirstName);  
};
```

with this

```
const upperAndReverseFirstNames = map(upperAndReverseFirstName);
```



Two fewer lines and even more declarative than `Array.map`, which I've been praising in this course so far.

Data Last

The last gem to gaze at is the ordering of `map`'s arguments. See how the `users` are supplied *after* the mapping function?

```
// users come first
users.map(upperAndReverseFirstName);

// users come last
map(upperAndReverseFirstName, users)
```



By design Ramda's functions are curried and take the data last, allowing for the best point-free, declarative JavaScript possible.

You can combine functions until you're ready for liftoff. Then when the data finally arrives, watch your pipeline soar like a rocket.

Summary

- Curried functions can take some parameters now, and others later by returning a function.
- Point-free functions don't show their arguments, making them even briefer than their pointed counterparts. This is nice and short

```
const upperAndReverseFirstNames = (users) => users.map(upperAndReverseFirstName);
```



But it's made even shorter by eliminating the `users` argument.

```
const upperAndReverseFirstNames = map(upperAndReverseFirstName);
```



- Ramda's functions take their data last, making point-free as easy as possible.

