How Ramda Uses Them

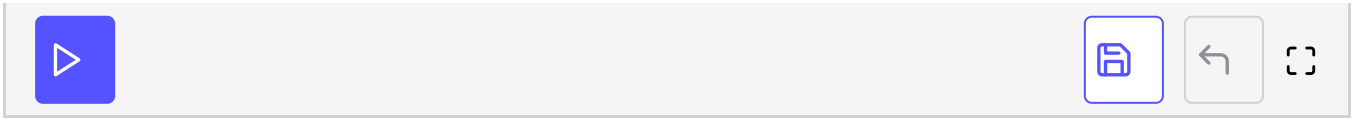
Dive into Ramda's map function and how to override it in your functors. (5 min. read)

Ramda's `map` function is great for arrays.

```
import { map, multiply } from 'ramda';

const functor = [1, 2, 3];
const result = map(multiply(2), functor);

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

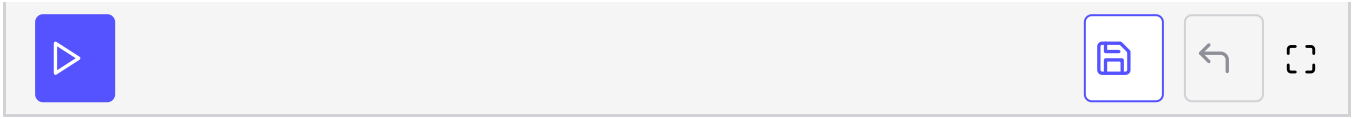


But did you know it also works for objects?

```
import { map, multiply } from 'ramda';

const functor = { x: 1, y: 2, z: 3 };
const result = map(multiply(2), functor);

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

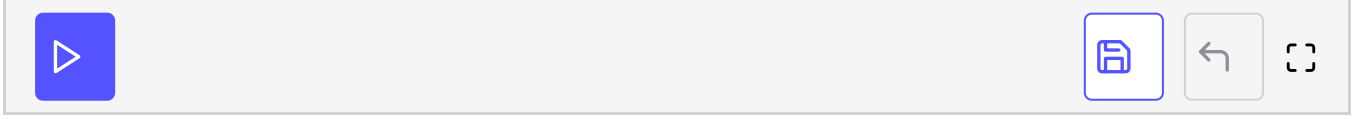


And strings!

```
import { concat, map } from 'ramda';

const functor = 'Hello';
const result = map(concat('yay'), functor);

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



They can all be looped over and transformed, so `map` supports them!

They can all be looped over and transformed, so `map` supports them.

You can even override `map` by defining a special method.

```
import { map, multiply } from 'ramda';

const functor = {
  value: 10,
  'fantasy-land/map': () => 'You have been overridden!'
};

const result = map(multiply(2), functor);

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

Notice `map` ignored `multiply(2)` and went straight to the override function. This is because it looks out for functors with the `fantasy-land/map` method.

Defining this method means “Give me full control of the mapping function”. Whatever it returns will be used.

Fantasy Land...?

Yep. This is a specification for using functors and their friends in JavaScript. Definitely beyond the scope of this course, but useful background info.

Summary

- Ramda's `map` works with arrays, objects, and strings.

- Functors with a `fantasy-land/map` method can override `map` and return whatever.

Who Cares?

The next topic, lenses, employ this strategy to a great extent. Without `map`'s override capabilities, lenses would need a *big* refactoring.