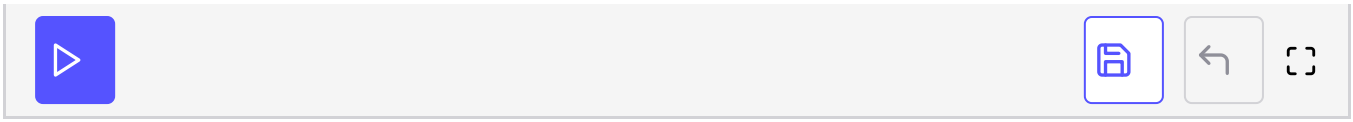
when/unless

Using when/unless in functional pipelines (4 min. read)

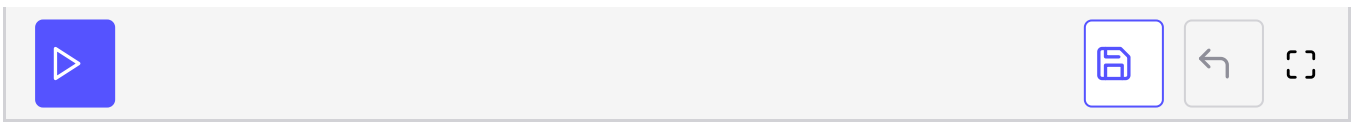
Sometimes you only need the `if` statement, and the `else` simply returns the value unchanged.

```
const isEven = (num) => num % 2 === 0;
const doubleIfEven = (num) => {
  if (isEven(num)) {
    return num * 2;
  }

  return num;
};

console.log(
  doubleIfEven(100),
  doubleIfEven(101)
);
```



Again, ternaries can work well here.

```
const isEven = (num) => num % 2 === 0;
const doubleIfEven = (num) => isEven(num) ? num * 2 : num;

console.log(
  doubleIfEven(100),
  doubleIfEven(101)
);
```

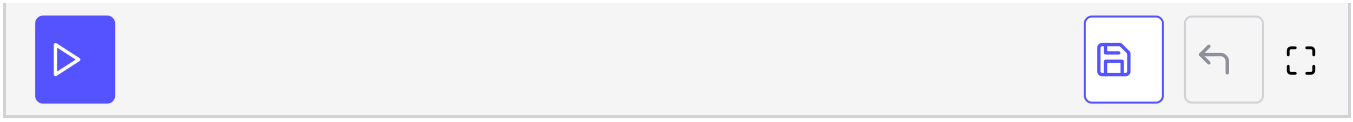


But we can also use the `when` function. It takes three arguments

1. Predicate (function that returns `true` or `false`)
2. Function to run if predicate returns true

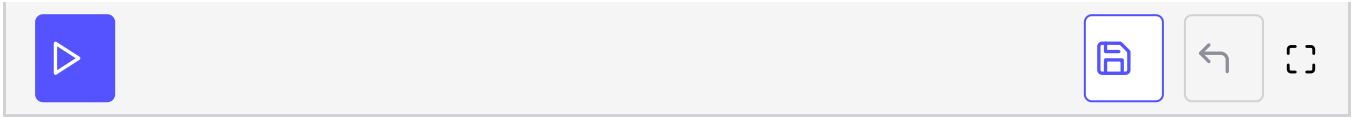
3. The value to use

```
import { when } from 'ramda';

const isEven = (num) => num % 2 === 0;

const doubleIfEven = when(
  isEven,
  (num) => num * 2
);

console.log(
  doubleIfEven(100),
  doubleIfEven(101)
);
```



We can make it point-free.

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

const isEven = (num) => num % 2 === 0;

const doubleIfEven = when(
  isEven,
  multiply(2)
);

console.log(
  doubleIfEven(100),
  doubleIfEven(101)
);
```

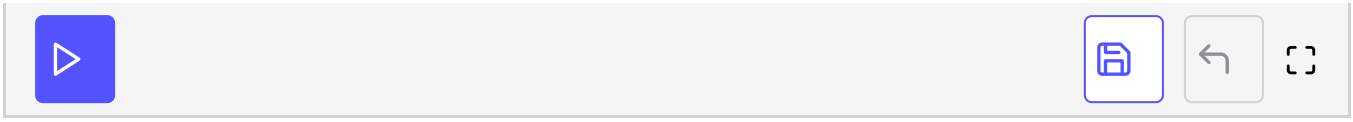


Conveniently enough, Ramda lets you express the opposite logic using `unless`.

This runs your function if the predicate returns `false`.

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

const isEven = (num) => num % 2 === 0;

const doubleIfOdd = unless(
  isEven,
  multiply(2)
);

console.log(
```

```
  doubleIfOdd(100),
  doubleIfOdd(101)
);
```



Now this function only doubles *odd* numbers. If we wanted `doubleIfEven`, our predicate must flip as well.

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

const isOdd = (num) => num % 2 !== 0;

const doubleIfEven = unless(
  isOdd,
  multiply(2)
);

console.log(
  doubleIfEven(100),
  doubleIfEven(101)
);
```