Sort Functions

Overview of Ramda's sorting functions. (4 min. read)

Sort Functions

The native `sort` is destructive, meaning it mutates the array.

```
const nums = [3, 2, 4, 1];
const result = nums.sort((a, b) => a - b)

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

Ramda provides a non-destructive `sort` function.

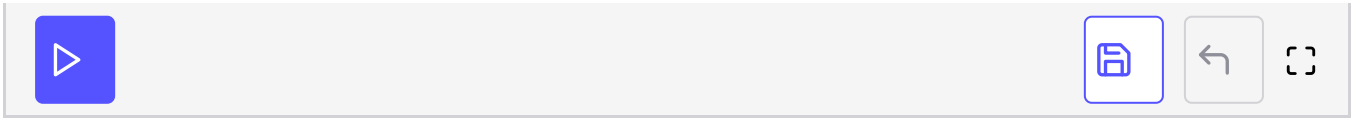
```
import { sort } from 'ramda';

const nums = [3, 2, 4, 1];
const result = sort((a, b) => a - b, nums);

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

`sortBy` applies a function to your values before comparing them. Sorting by absolute value, for example, is trivial now.

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

const result = sortBy(Math.abs, [-100, 1, -50, 0]);

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

If you'd like to sort by multiple criteria, `sortWith` is your friend. This example sorts people by their age, then name.

```
import { ascend, prop, sortWith } from 'ramda';

const people = [{
  name: 'Bobo',
  age: 25
}, {
  name: 'Cam',
  age: 25
}, {
  name: 'Al',
  age: 29
}];

const result = sortWith([
  ascend(prop('age')),
  ascend(prop('name'))
], people);

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



Ramda's `ascend` functions wraps around `prop('age')`, telling it "Sort by age, ascending (smallest first)".

```
ascend(prop('age'))
```

Same with `prop('name')`: "Sort by name, ascending (A, B, C, D...)."

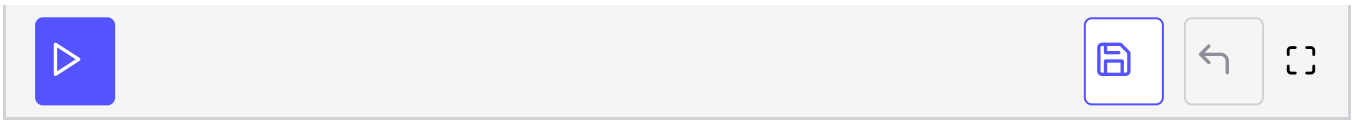
```
ascend(prop('name'))
```

So if they're the same age, `sortWith` will then sort by name.

For the opposite effect, use `descend`.

```
import { descend, prop, sortWith } from 'ramda';

const people = [{
  name: 'Bobo',
  age: 25
}, {
  name: 'Cam',
  age: 25
}, {
  name: 'Al',
  age: 29
```

```
}];

const result = sortWith([

  descend(prop('age')),
  descend(prop('name'))
], people);

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



Now `people`'s sorted in opposite order.