Comparators

Comparators turn any comparing function into a sorting function, without fretting browser inconsistencies. (5 min. read)

Ascend and Descend

You won't need these as often as the main sorting functions, but they can be useful in some scenarios.

The functions we just saw, `ascend`/`descend`, are comparators. They take a function to apply against each value.

Here's a sort by height.

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

const people = [{
  height: 23
}, {
  height: 230
}, {
  height: 2.3
}];

const getHeight = (x) => x.height;
const byHeight = ascend(getHeight);

const result = sort(byHeight, people);

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

Ramda's `prop` function can replace the `getHeight` function.

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

const people = [{
  height: 23
}, {
  height: 230
}, {
  height: 2.3

}];

const byHeight = ascend(prop('height'));

const result = sort(byHeight, people);

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

Descend sorts in the opposite direction, greatest first.

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

const people = [{
  height: 23
}, {
  height: 230
}, {
  height: 2.3
}];

const byHeight = descend(prop('height'));

const result = sort(byHeight, people);

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

And `path` lets you easily compare things, no matter how nested their properties are.

```
import { ascend, path, sort } from 'ramda';

const people = [{
  name: 'I am second',
  metadata: {
    attributes: {
      height: {
        value: 23
      }
    }
  }
}, {
  name: 'I am last',
  metadata: {
    attributes: {
      height: {
        value: 230
      }
    }
  }
}, {
  name: 'I am first',
  metadata: {
    attributes: {
      height: {
        value: 2.3
      }
    }
  }
}];

const getHeight = path(['metadata', 'attributes', 'height', 'value']);
const byHeight = ascend(getHeight);

const result = sort(byHeight, people);

console.log(result);
```

Lower-Level Comparator

Ramda also has a `comparator` function. It creates comparator functions out of regular functions that compare two elements.

This function, for example...

```
const byHeight = (a, b) => a.height > b.height;
```

...works with `sort` in modern browsers.

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

const people = [{ height: 20 }, { height: 10 }];
const byHeight = (a, b) => a.height > b.height;

const result = sort(byHeight, people);

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

But since `byHeight` returns a boolean, older browsers like Internet Explorer won't sort `people` correctly. Older browsers require your comparator to return a number!

See this StackOverflow answer for more details.

A proper comparator looks like this

```
const byHeight = (a, b) => {
  if (a.height > b.height) {
    return 1;
  }

  if (a.height < b.height) {
    return -1;
  }

  return 0;
};
```

Allowing this sort to work everywhere.

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

const people = [{ height: 20 }, { height: 10 }];

const byHeight = (a, b) => {
  if (a.height > b.height) {
    return 1;
  }

  if (a.height < b.height) {
    return -1;
  }

  return 0;
};

const result = sort(byHeight, people);

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

A bit verbose. Fortunately, Ramda turns any comparison function into a proper comparator. All we need is the `comparator` function.

Take your original `byHeight` function and flip `>` to `<`...

```
const byHeight = (a, b) => a.height < b.height;
```

And wrap it in `comparator`. I'd prefer renaming it so the comparator function can be named `byHeight`.

```
// rename this
const compareHeights = (a, b) => a.height < b.height;

// giving this a proper name
const byHeight = comparator(compareHeights)
```

Now use it.

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

const people = [{ height: 20 }, { height: 10 }];

const compareHeights = (a, b) => a.height < b.height;
const byHeight = comparator(compareHeights);

const result = sort(byHeight, people);

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