Ramda

A practical functional library for JavaScript programmers.

Build passing (https://github.com/ramda/ramda/actions?query=workflow%3ABuild) Test Coverage (https://codeclimate.com/github/ramda/ramda/test_coverage) npm package 0.28.0 (https://www.npmjs.org/package/ramda) available on deno.land/x (https://deno.land/x/ramda@v0.27.2) PUBLISHED ON NEST.LAND (https://nest.land/package/ramda) gitter join chat (https://gitter.im/ramda/ramda?utm_source=badge&utm_medium=badge&utm_campaign=pr-badge&utm_content=badge)

Why Ramda?

There are already several excellent libraries with a functional flavor. Typically, they are meant to be general-purpose toolkits, suitable for working in multiple paradigms. Ramda has a more focused goal. We wanted a library designed specifically for a functional programming style, one that makes it easy to create functional pipelines, one that never mutates user data.

What's Different?

The primary distinguishing features of Ramda are:

- Ramda emphasizes a purer functional style. Immutability and side-effect free functions are at the heart of its design philosophy. This can help you get the job done with simple, elegant code.
- Ramda functions are automatically curried. This allows you to easily build up new functions from old ones simply by not supplying the final parameters.
- The parameters to Ramda functions are arranged to make it convenient for currying. The data to be operated on is generally supplied last.

The last two points together make it very easy to build functions as sequences of simpler functions, each of which transforms the data and passes it along to the next. Ramda is designed to support this style of coding.

Introductions

- Introducing Ramda (http://buzzdecafe.github.io/code/2014/05/16/introducing-ramda) by Buzz de Cafe
- Why Ramda? (http://fr.umio.us/why-ramda/) by Scott Sauyet
- Favoring Curry (http://fr.umio.us/favoring-curry/) by Scott Sauyet
- Why Curry Helps (https://hughfdjackson.com/javascript/why-curry-helps/) by Hugh Jackson
- Hey Underscore, You're Doing It Wrong! (https://www.youtube.com/watch?v=m3svKOdZijA&app=desktop) by Brian Lonsdorf
- Thinking in Ramda (https://randycoulman.com/blog/categories/thinking-in-ramda) by Randy Coulman

Philosophy

Using Ramda should feel much like just using JavaScript. It is practical, functional JavaScript. We're not introducing lambda expressions in strings, we're not borrowing consed lists, we're not porting over all of the Clojure functions.

Our basic data structures are plain JavaScript objects, and our usual collections are JavaScript arrays. We also keep other native features of JavaScript, such as functions as objects with properties.

Functional programming is in good part about immutable objects and side-effect free functions. While Ramda does not *enforce* this, it enables such style to be as frictionless as possible.

We aim for an implementation both clean and elegant, but the API is king. We sacrifice a great deal of implementation elegance for even a slightly cleaner API.

Last but not least, Ramda strives for performance. A reliable and quick implementation wins over any notions of functional purity.

OPEN CHAT (HTTPS://GITTER.IM/RAMDA/RAMDA)

Installation

To use with node:

```
$ npm install ramda
```

Then in the console:

```
const R = require('ramda');
```

To use directly in Deno (https://deno.land):

```
import * as R from "https://deno.land/x/ramda@v0.27.2/mod.ts";
```

or using Nest.land:

```
import * as R from "https://x.nest.land/ramda@0.27.2/mod.ts";
```

To use directly in the browser:

```
<script src="path/to/yourCopyOf/ramda.js"></script>
```

or the minified version:

```
<script src="path/to/yourCopyOf/ramda.min.js"></script>
```

or from a CDN, either cdnjs:

```
<script src="//cdnjs.cloudflare.com/ajax/libs/ramda/0.25.0/ramda.min.js"></script>
```

or one of the below links from jsDelivr (http://jsdelivr.com):

```
<script src="//cdn.jsdelivr.net/npm/ramda@0.25.0/dist/ramda.min.js"></script>
<script src="//cdn.jsdelivr.net/npm/ramda@0.25/dist/ramda.min.js"></script>
<script src="//cdn.jsdelivr.net/npm/ramda@latest/dist/ramda.min.js"></script>
```

(note that using `latest` is taking a significant risk that ramda API changes could break your code.)

These script tags add the variable `R` on the browser's global scope.

Or you can inject ramda into virtually any unsuspecting website using the bookmarklet (https://github.com/ramda/ramda/blob/master/BOOKMARKLET.md).

Note for versions > 0.25 Ramda versions > 0.25 don't have a default export. So instead of `import R from 'ramda';` , one has to use `import * as R from 'ramda';` Or better yet, import only the required functions via `import { functionName } from 'ramda';`

Note for ES6 module and browsers In order to access to the ES6 module in browsers, one has to provide the content of the **es** directory (see below for the build instructions) and use `import * as R from './node_modules/ramda/es/index.js';`

Build

`npm run build` creates `es` , `src` directories and updates both **dist/ramda.js** and **dist/ramda.min.js**

Partial Builds

It is possible to build Ramda with a subset of the functionality to reduce its file size. Ramda's build system supports this with command line flags. For example if you're using `R.compose` , `R.reduce` , and `R.filter` you can create a partial build with:

```
npm run --silent partial-build compose reduce filter > dist/ramda.custom.js
```

This requires having Node/io.js installed and ramda's dependencies installed (just use `npm install` before running partial build).

Install specific functions

OPEN CHAT (HTTPS://GITTER.IM/RAMDA/RAMDA)

Install individual functions (https://bitsrc.io/ramda/ramda) with bit, npm and yarn without installing the whole library.

Documentation

Please review the API documentation (https://ramdajs.com/docs/).

Also available is our Cookbook (https://github.com/ramda/ramda/wiki/Cookbook) of functions built from Ramda that you may find useful.

The Name

Ok, so we like sheep. That's all. It's a short name, not already taken. It could as easily have been `eweda` , but then we would be forced to say *eweda lamb!*, and no one wants that. For non-English speakers, lambs are baby sheep, ewes are female sheep, and rams are male sheep. So perhaps ramda is a grown-up lambda... but probably not.

Running The Test Suite

Console:

To run the test suite from the console, you need to have `mocha` installed:

```
npm install -g mocha
```

Then from the root of the project, you can just call

```
mocha
```

Alternately, if you've installed the dependencies, via:

```
npm install
```

then you can run the tests (and get detailed output) by running:

```
npm test
```

Browser:

You can use testem (https://github.com/airportyh/testem) to test across different browsers (or even headlessly), with livereloading of tests. Install testem (`npm install -g testem`) and run `testem` . Open the link provided in your browser and you will see the results in your terminal.

If you have *PhantomJS* installed, you can run `testem -l phantomjs` to run the tests completely headlessly.

Usage

For `v0.25` and up, import the whole library or pick ES modules directly from the library:

```
import * as R from 'ramda'

const {identity} = R
R.map(identity, [1, 2, 3])
```

Destructuring imports from ramda *does not necessarily prevent importing the entire library*. You can manually cherry-pick methods like the following, which would only grab the parts necessary for `identity` to work:

```
import identity from 'ramda/src/identity'

identity()
```

Manually cherry picking methods is cumbersome, however. Most bundlers like Webpack and Rollup offer tree-shaking as a way to drop unused Ramda code and reduce bundle size, but their performance varies, discussed here (https://github.com/scabbiaza/ramda-webpack-tree-shaking-examples). Here is a summary of the optimal setup based on what technology you are using:

OPEN CHAT (HTTPS://GITTER.IM/RAMDA/RAMDA)

1. Webpack + Babel - use `babel-plugin-ramda` (https://github.com/megawac/babel-plugin-ramda) to automatically cherry pick methods. Discussion here (https://www.andrewsouthpaw.com/ramda-webpack-and-tree-shaking/), example here (https://github.com/AndrewSouthpaw/ramda-webpack-tree-shaking-examples/blob/master/07-webpack-babel-plugin-ramda/package.json)
2. Webpack only - use `UglifyJS` plugin for treeshaking along with the `ModuleConcatenationPlugin` . Discussion here (https://github.com/ramda/ramda/issues/2355), with an example setup here (https://github.com/scabbiaza/ramda-webpack-tree-shaking-examples/blob/master/06-webpack-scope-hoisted/webpack.config.js)
3. Rollup - does a fine job properly treeshaking, no special work needed; example here (https://github.com/scabbiaza/ramda-webpack-tree-shaking-examples/blob/master/07-rollup-ramda-tree-shaking/rollup.config.js)

Typings

- TypeScript (https://www.npmjs.com/package/@types/ramda)
- Flow (https://github.com/flowtype/flow-typed/tree/master/definitions/npm/ramda_v0.x.x)

Translations

- Chinese(中文) (http://ramda.cn/)
- Ukrainian(Українська) (https://github.com/ivanzusko/ramda)
- Portuguese(BR) (https://github.com/renansj/ramda)
- Russian(Русский) (https://github.com/Guck111/ramda)
- Spanish(ES) (https://github.com/wirecobweb/ramda)

Funding

If you wish to donate to Ramda please see our Open Collective (https://opencollective.com/ramda) page. Thank you!

Acknowledgements

Thanks to J. C. Phillipps (http://www.jcphillipps.com) for the Ramda logo. Ramda logo artwork © 2014 J. C. Phillipps. Licensed Creative Commons CC BY-NC-SA 3.0 (http://creativecommons.org/licenses/by-nc-sa/3.0/).

OPEN CHAT (HTTPS://GITTER.IM/RAMDA/RAMDA)