Validatem

An ergonomic and modular validation system for Javascript code - argument validation, arbitrary value validation, everything.

What is Validatem?

> This article is derived from the documentation at https://www.npmjs.com/package/@validatem/core.

The last validation library you'll ever need.

- Does **every kind of validation**, and does it well: it doesn't matter whether you're validating function arguments, form data, JSON request bodies, configuration files, or whatever else. As long as it's structured data of some sort, Validatem can deal with it.
- Supports the notion of **virtual properties** in validation errors, which means that even if your data *isn't* already structured data (eg. an encoded string of some sort), you can bring your own parser, and have it integrate cleanly.
- **Easy to read**; both the code that *uses* Validatem, and the validation error messages that it produces! Your validation code doubles as in-code format documentation, and users get clear feedback about what's wrong.
- Fully **composable**: it's trivial to use third-party validators, or to write your own (reusable!) validators, whether fully custom or made up of a few other validators chained together.
- Supports **value transformation**, which means that you can even encode things like "this value defaults to X" or "when this value is a number, it will be wrapped like so" in your validation code; this can save you a bunch of boilerplate, and makes your validation code *even more complete* as format documentation.
- Validatem has a **small and modular core**, and combined with its composability, this means you won't pull any more code into your project than is strictly necessary to make your validators work! This is also an important part of making Validatem suitable for use in libraries, eg. for argument validation.
- Many **off-the-shelf validators** are already available! You can find the full list here.
- Extensively **documented**, with clear documentation on what is considered valid, and what is not. Likewise, the plumbing libraries that you can use to write your own validators and combinators, are also well-documented.

While Validatem is suitable for any sort of validation, this unique combination of features and design choices makes it *especially* useful for validating arguments in the public API of libraries, unlike other validation libraries!

For example, you might write something like the following (from the `icssify` library):

```
module.exports = function (browserify, options) {
  validateArguments(arguments, {
    browserify: required,
```

```
  options: allowExtraProperties({
    mode: oneOf([ "local", "global" ]),
    before: arrayOf([ required, isPostcssPlugin ]),
    after: arrayOf([ required, isPostcssPlugin ]),
    extensions: arrayOf([ required, isString ])
  })
});

// Implementation code goes here ...
};
```

And calling it like so:

```
icssify(undefined, {
mode: "nonExistentMode",
before: [ NaN ],
unspecifiedButAllowedOption: true
})
```

... would then produce an error like this:

```
ValidationError: One or more validation errors occurred:
 - At browserify: Required value is missing
 - At options -> mode: Must be one of: 'local', 'global'
 - At options -> before -> 0: Must be a PostCSS plugin
```

Why are there so many packages?

> This article is derived from the documentation at https://www.npmjs.com/package/@validatem/core.

Dependencies often introduce a lot of unnecessary complexity into a project. To avoid that problem, I've designed Validatem to consist of a lot of small, separately-usable pieces - even much of the core plumbing has been split up that way, specifically the bits that may be used by validator and combinator functions.

This may sound counterintuitive; doesn't more dependencies mean *more* complexity? But in practice, "a dependency" in and of itself doesn't have a complexity cost at all; it's the code that is *in* the dependency where the complexity lies. The bigger a dependency is, the more complexity there is in that dependency, and the bigger the chance that some part of that complexity isn't even being used in your project!

By packaging things as granularly as possible, you end up only importing code into your project *that you are actually using*. Any bit of logic that's never used, is somewhere in a package that is never even installed. As an example: using 10 modules with 1 function each, will add less complexity to your project than using 1 module with 100 functions.

This has a lot of benefits, for both you and me:

- **Easier to audit/review:** When only the code you're actually using is added to your project, there will be less code for you to review. And because each piece is designed to be loosely coupled and extensively documented, you can review each (tiny) piece in isolation; without having to trawl through mountains of source code to figure out how it's being called and what assumptions are being made there.
- **Easier to version and maintain:** Most of the modules for Validatem will be completely done and feature-complete the moment they are written, never needing any updates at all. When occasionally a module *does* need an update, it will almost certainly not be one that breaks the API, because the API for each module is so simple that there isn't much *to* break.
- **Easier to upgrade:** Because of almost nothing ever breaking the API, it also means that you'll rarely need to manually upgrade anything, if ever! The vast majority of module updates can be automatically applied, even many years into the future, *even* if a new (breaking) version of `validatem/@core` is ever released down the line.

- **Easier to fork:** If for any reason you want to fork any part of Validatem, you can do so easily - *without* also having to maintain a big pile of validators, combinators, internals, and so on. You only need to fork and maintain the bit where you want to deviate from the official releases.

Of course, there being so many packages means it can be more difficult to find the specific package you want. That is why the Validatem website has an extensive, categorized list of all the validators, combinators and utilities available for Validatem. Both officially-maintained ones, and third-party modules!