**Classnames**

A simple JavaScript utility for conditionally joining classNames together.

Install with [npm](https://www.npmjs.com/), [Bower](https://bower.io/), or [Yarn](https://yarnpkg.com/):

# via npm

npm install classnames

# via Bower

bower install classnames

# or Yarn (note that it will automatically save the package to your `dependencies` in `package.json`)

yarn add classnames

Use with [Node.js](https://nodejs.org/en/), [Browserify](https://browserify.org/), or [webpack](https://webpack.github.io/):

var classNames = require('classnames');

classNames('foo', 'bar'); // => 'foo bar'

Alternatively, you can simply include index.js on your page with a standalone <script> tag and it will export a global classNames method, or define the module if you are using RequireJS.

**Project philosophy**

We take the stability and performance of this package seriously, because it is run millions of times a day in browsers all around the world. Updates are thoroughly reviewed for performance impacts before being released, and we have a comprehensive test suite.

Classnames follows the [SemVer](https://semver.org/) standard for versioning.

There is also a [Changelog](https://github.com/JedWatson/classnames/blob/master/HISTORY.md).

**Usage**

The classNames function takes any number of arguments which can be a string or object. The argument 'foo' is short for { foo: true }. If the value associated with a given key is falsy, that key won't be included in the output.

classNames('foo', 'bar'); // => 'foo bar'

classNames('foo', { bar: true }); // => 'foo bar'

classNames({ 'foo-bar': true }); // => 'foo-bar'

classNames({ 'foo-bar': false }); // => ''

classNames({ foo: true }, { bar: true }); // => 'foo bar'

classNames({ foo: true, bar: true }); // => 'foo bar'

// lots of arguments of various types

classNames('foo', { bar: true, duck: false }, 'baz', { quux: true }); // => 'foo bar baz quux'

// other falsy values are just ignored

classNames(null, false, 'bar', undefined, 0, 1, { baz: null }, ''); // => 'bar 1'

Arrays will be recursively flattened as per the rules above:

var arr = ['b', { c: true, d: false }];

classNames('a', arr); // => 'a b c'

**Dynamic class names with ES2015**

If you're in an environment that supports [computed keys](https://www.ecma-international.org/ecma-262/6.0/#sec-object-initializer) (available in ES2015 and Babel) you can use dynamic class names:

let buttonType = 'primary';

classNames({ [`btn-${buttonType}`]: true });

**Usage with React.js**

This package is the official replacement for classSet, which was originally shipped in the React.js Addons bundle.

One of its primary use cases is to make dynamic and conditional className props simpler to work with (especially more so than conditional string manipulation). So where you may have the following code to generate a className prop for a <button> in React:

class Button extends React.Component {

// ...

render () {

var btnClass = 'btn';

if (this.state.isPressed) btnClass += ' btn-pressed';

else if (this.state.isHovered) btnClass += ' btn-over';

return <button className={btnClass}>{this.props.label}</button>;

}

}

You can express the conditional classes more simply as an object:

var classNames = require('classnames');

class Button extends React.Component {

// ...

render () {

var btnClass = classNames({

btn: true,

'btn-pressed': this.state.isPressed,

'btn-over': !this.state.isPressed && this.state.isHovered

});

return <button className={btnClass}>{this.props.label}</button>;

}

}

Because you can mix together object, array and string arguments, supporting optional className props is also simpler as only truthy arguments get included in the result:

var btnClass = classNames('btn', this.props.className, {

'btn-pressed': this.state.isPressed,

'btn-over': !this.state.isPressed && this.state.isHovered

});

**Alternate dedupe version**

There is an alternate version of classNames available which correctly dedupes classes and ensures that falsy classes specified in later arguments are excluded from the result set.

This version is slower (about 5x) so it is offered as an opt-in.

To use the dedupe version with Node.js, Browserify, or webpack:

var classNames = require('classnames/dedupe');

classNames('foo', 'foo', 'bar'); // => 'foo bar'

classNames('foo', { foo: false, bar: true }); // => 'bar'

For standalone (global / AMD) use, include dedupe.js in a <script> tag on your page.

**Alternate bind version (for**[**css-modules**](https://github.com/css-modules/css-modules)**)**

If you are using [css-modules](https://github.com/css-modules/css-modules), or a similar approach to abstract class "names" and the real className values that are actually output to the DOM, you may want to use the bind variant.

*Note that in ES2015 environments, it may be better to use the "dynamic class names" approach documented above.*

var classNames = require('classnames/bind');

var styles = {

foo: 'abc',

bar: 'def',

baz: 'xyz'

};

var cx = classNames.bind(styles);

var className = cx('foo', ['bar'], { baz: true }); // => "abc def xyz"

Real-world example:

/\* components/submit-button.js \*/

import { Component } from 'react';

import classNames from 'classnames/bind';

import styles from './submit-button.css';

let cx = classNames.bind(styles);

export default class SubmitButton extends Component {

render () {

let text = this.props.store.submissionInProgress ? 'Processing...' : 'Submit';

let className = cx({

base: true,

inProgress: this.props.store.submissionInProgress,

error: this.props.store.errorOccurred,

disabled: this.props.form.valid,

});

return <button className={className}>{text}</button>;

}

};

**Polyfills needed to support older browsers**

**classNames >=2.0.0**

Array.isArray: see [MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/isArray) for details about unsupported older browsers (e.g. <= IE8) and a simple polyfill.

**License**