



*** NEW ENGLAND ***
DRUPAL CAMP

SCOPE IN CSS WITH AND WITHOUT JAVASCRIPT

Brian Perry
November 23, 2019

Slides: <http://bit.ly/css-scope-ned>

BRIAN PERRY

- Lead Front End Dev at Bounteous
- Rocking the Chicago 'burbs
- From Cranston (free street cred!)
- Lover of all things components...
...and Nintendo



d.o: brianperry

twitter: bricomedy

github: backlineint

nintendo: wabrian

brianperryinteractive.com



bounteous



CSS has **global scope**.

CSS has **global scope**.

That's great when you want rules to apply globally...

... and not so great when you don't.



A screenshot of a tweet from Thomas Fuchs (@thomasfuchs) on Twitter. The tweet contains a pun about CSS properties and a barstool. The interface shows the user's profile picture, name, handle, and several icons (floppy disk, person, cactus, checkmark). The tweet text is split into two lines. Below the text, it shows 3,112 likes and the time '10:10 AM - Jul 28, 2014'. At the bottom, it says '3,158 people are talking about this' with a right-pointing arrow.

 **Thomas Fuchs**     
@thomasfuchs

Two CSS properties walk into a bar.

A barstool in a completely different bar falls over.

 3,112 10:10 AM - Jul 28, 2014 

 3,158 people are talking about this 

Many **CSS in JavaScript** solutions provide **component scope**.

CSS: global scope

CSS-in-JS: component scope

People disagree about these
approaches...

because... internet.

I think we should **take advantage of both.**

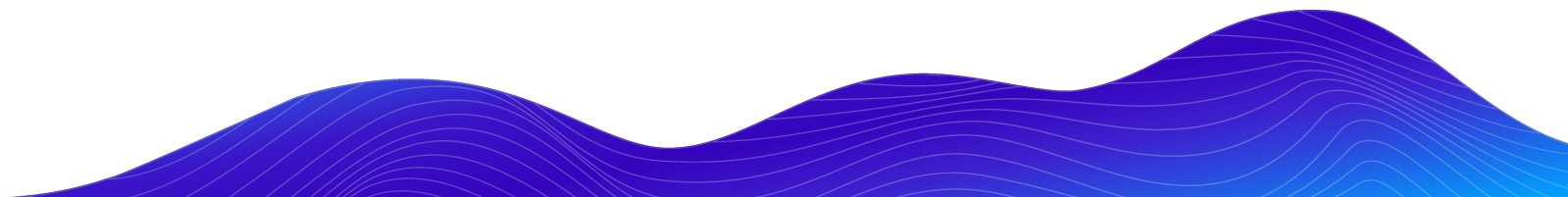


Thank You!

Brian Perry

Lead Front End Developer

Email: brian.perry@bounteous.com





*** NEW ENGLAND ***

DRUPAL CAMP

SCOPE IN CSS WITH AND WITHOUT JAVASCRIPT

Brian Perry

November 23, 2019

Slides: <http://bit.ly/css-scope-ned>

CSS has **global scope.**

```
HTML
1 <h1>The same font everywhere!</h1>
2 <p>Even here!</p>

CSS
1 html {
2   font-family: "Comic Sans MS", "Comic Sans", cursive;
3 }
4
5 /*
6 p, h1 {
7   font-family: "Comic Sans MS", "Comic Sans", cursive;
8 }
9 */
10
```

The same font everywhere!

Even here!

Codepen: <https://codepen.io/brianperry/pen/rNNQpyK>

THE CASCADE (THE C IN CSS)

Stylesheets can have three different origins

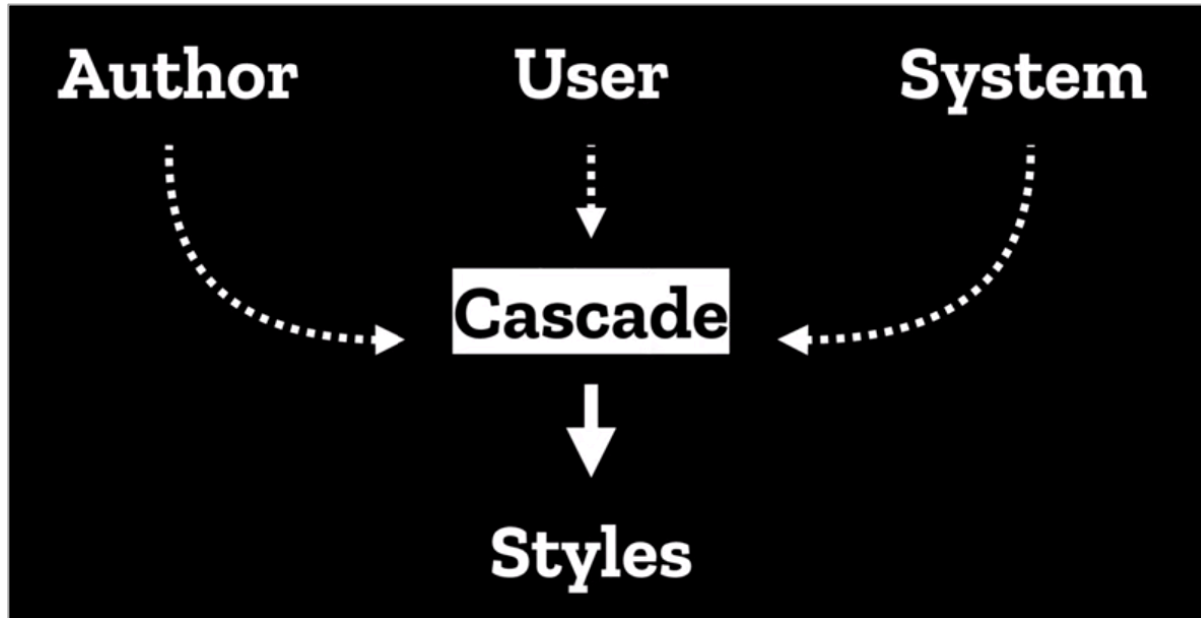
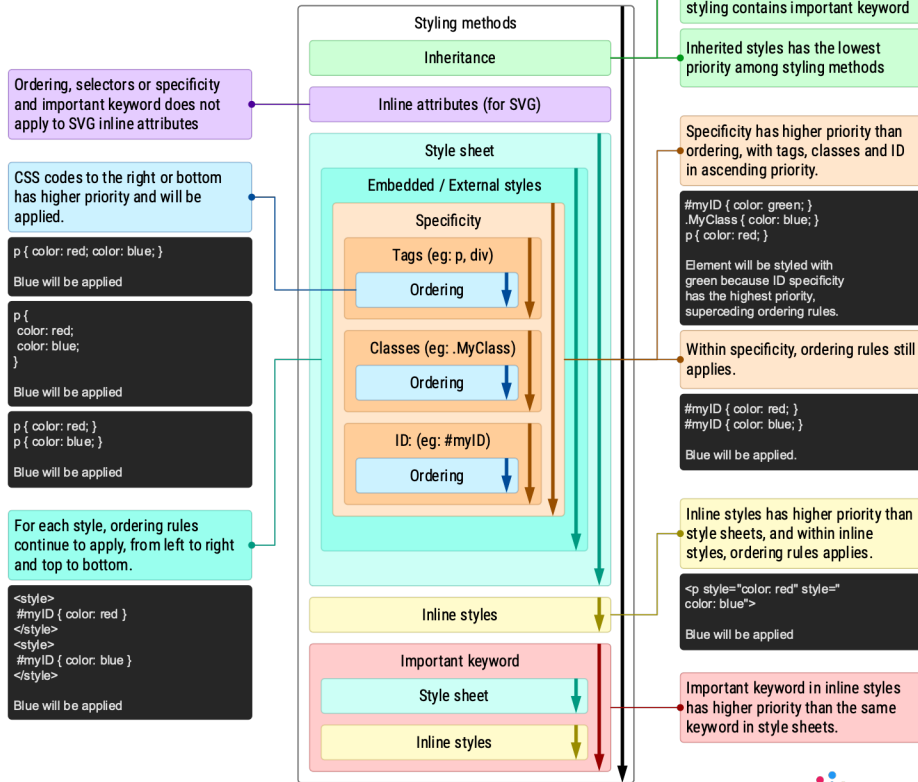


Image: <https://www.miriamsuzanne.com/2019/10/03/css-is-weird/>

The definitive guide to CSS styling order

Includes CSS stylings for SVG



The way all rules are combined is very easy to understand and remember.



Image: <https://css-tricks.com/the-c-in-css-the-cascade/#article-header-id-11>

CASCADING ORDER (ACCORDING TO W3C SPEC)

- Find all declarations that apply to the element and property for the target media type.
- **Sort according to importance and origin** (ascending order)
 - **User agent** declarations
 - **User normal** declarations
 - **Author normal** declarations
 - **Author !important** declarations
 - **User !important** declarations
- Sort rules with same origin and importance by **specificity**
- If all else fails, sort by **order specified**



Source: <https://www.w3.org/TR/CSS2/cascade.html#cascade>

SPECIFICITY



element selector
Specificity: 0,0,1



class selector
attribute selector
pseudo-class
Specificity: 0,1,0 (10)



id selector
Specificity: 1,0,0



style attribute
Specificity: 1,0,0,0

(Note: The universal selector and inherited selectors have a specificity of 0, 0, 0, 0)

Source: https://stuffandnonsense.co.uk/archives/css_specificity_wars.html

Stuff & Nonsense



a

1 x element selector

Sith power: 0,0,1



pa

2 x element selectors

Sith power: 0,0,2



.foo

1 x class selector *

Sith power: 0,1,0



a.foo

1 x element selector
1 x class selector

Sith power: 0,1,1



pa.foo

2 x element selectors
1 x class selector

Sith power: 0,1,2



.foo.bar

2 x class selectors

Sith power: 0,2,0



p.foo a.bar

2 x element selectors
2 x class selectors

Sith power: 0,2,2



#foo

1 x id selector

Sith power: 1,0,0



a#foo

1 x element selector
1 x id selector

Sith power: 1,0,1



.foo a#bar

1 x element selector
1 x class selector
1 x id selector

Sith power: 1,1,1



.foo .foo #foo

2 x class selectors
1 x id selector

Sith power: 1,2,0



style

1 x style attribute

Sith power: 1,0,0,0



* Same specificity
class selector =
attribute attribute =
pseudo-classes



!important

CSS-in-JS can provide **component scope** ...and other advantages.

WHY CSS-IN-JS?

- **Component scope** – styles can't 'leak out'
- **Co-locate styles with components**
 - Positive developer experience
 - An advantage for distributed and composable components
- **Dead code elimination** – bundler knows if css is being used
- **Worry free naming** – don't have to worry about conflicts or enforcing a naming convention
- **Respond to props** rather than juggling class names
- And yes, probably fear / lack of knowledge in some cases

AND WHY NOT?

- You **still need to know how CSS works** (even the hard stuff)
- Comes with a **bundle size / performance cost**
- **Potential barrier of entry** for CSS experts who aren't JS experts
- **Additional layers of abstraction**
 - Some approaches use less css more object-like syntax
 - Potential syntax highlighting issues
- Risk of **inconsistency / one offs**

So which approach is **right**?



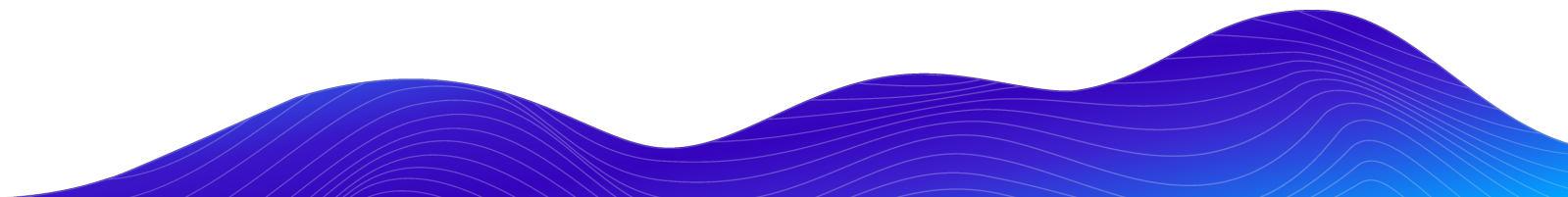
Why don't we have both?

Thank You!

Brian Perry

Lead Front End Developer

Email: brian.perry@bounteous.com





*** NEW ENGLAND ***
DRUPAL CAMP

SCOPE IN CSS WITH AND WITHOUT JAVASCRIPT

Brian Perry
November 23, 2019

Slides: <http://bit.ly/css-scope-ned>

CSS has **global scope.**













HEY, REMEMBER CSS?


Cascade




Specificity

Stuff & Nonsense

 a 1 x element selector Sith power: 0,0,1	 p a 2 x element selectors Sith power: 0,0,2	 .foo 1 x class selector * Sith power: 0,1,0	 a.foo 1 x element selector 1 x class selector Sith power: 0,1,1
 p a.foo 2 x element selectors 1 x class selector Sith power: 0,1,2	 .foo_bar 2 x class selectors Sith power: 0,2,0	 p.foo a.bar 2 x element selectors 2 x class selectors Sith power: 0,2,2	 #foo 1 x id selector Sith power: 1,0,0
 a#foo 1 x element selector 1 x id selector Sith power: 1,0,1	 .foo a#bar 1 x class selector 1 x class selector 1 x id selector Sith power: 1,1,1	 .foo.foo #foo 2 x class selectors 1 x id selector Sith power: 1,2,0	 style 1 x style attribute Sith power: 1,0,0,0

 * Same specificity
class selector =
attribute attribute =
pseudo-classes


!important

Inheritance



Some of this is tricky. **We should get
our act together...**

CSS METHODOLOGIES

- Popular Methodologies
 - Object-Oriented CSS (OOCSS)
 - Block, Element, Modifier (BEM)
 - Scalable and Modular Architecture for CSS (SMACSS)
- File organization
- Naming conventions
- Can create something that 'feels' like component scope.

BEM

The screenshot shows a CodePen editor for a project named "BEM-ojjs" by Brian Perry. The editor is divided into three main sections: HTML, CSS (SCSS), and JS. The HTML section contains the following code:

```
1 <p>Based on Dave House's <a  
  href="https://medium.com/@davehouse_80809/bem-for-everyone-  
  else-89ccc8ad66f2">BEM For Everyone Else</a>.</p>  
2  
3 <div class="container">  
4   <div class="face">  
5   </div>  
6   Block: <br /> .face  
7  
8   <div class="face">  
9     <div class="face__eyes"></div>  
10    <div class="face__mouth"></div>  
11  </div>  
12  Elements: <br /> .face__eyes / .face__mouth  
13  
14  <div class="face">  
15    <div class="face__eyes"></div>  
16    <div class="face__mouth face__mouth--open"></div>  
17  </div>  
18  Modifier: <br />  
19  .face__eyes.face__eyes--open  
20  
21  <div class="face">  
22    <div class="face__eyes face__eyes--open"></div>  
23    <div class="face__mouth face__mouth--open"></div>  
24  </div>
```

The rendered output on the right shows three yellow circular faces. The first is a plain yellow circle. The second has two horizontal red lines for eyes and a horizontal red line for a mouth. The third is partially visible at the bottom of the frame.

Based on Dave House's [BEM For Everyone Else](https://medium.com/@davehouse_80809/bem-for-everyone-else-89ccc8ad66f2).

Block:
.face

Elements:
.face__eyes / .face__mouth

Concept: https://medium.com/@davehouse_80809/bem-for-everyone-else-89ccc8ad66f2 Codepen: <https://codepen.io/brianperry/pen/BaaMYMJ>

BEM IS GREAT!

- Understandable structure that unifies markup and styles
- Specific and component focused classes

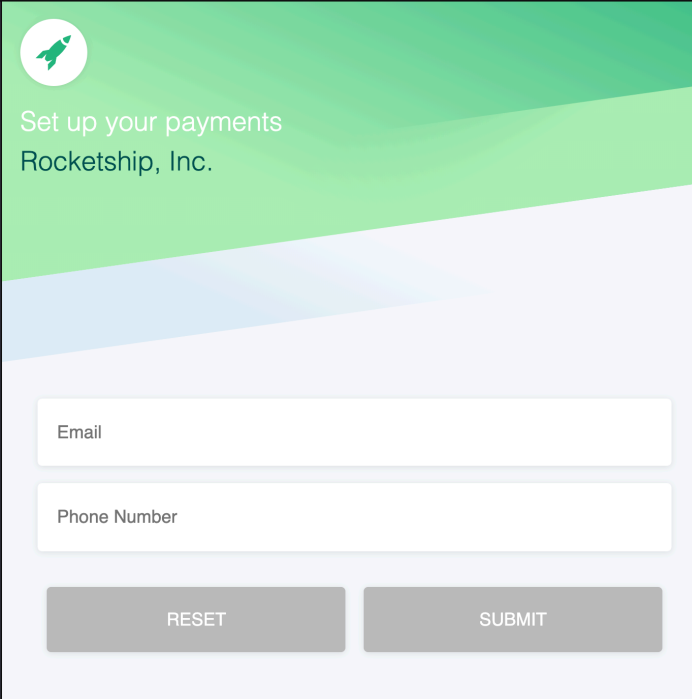
But...

- In practice often like component scope, but not truly scoped to component
- Not enforced at the tooling level
 - (sass-lint can help a bit here)
- Requires discipline across dev team
- Naming is still hard

CSS-in-JS can provide **component scope.**

COMPONENT SCOPE - STYLED COMPONENTS

```
styled-components ▾
index.js  form.js  header.js  +
1 import React, { Component } from 'react';
2 import styled from 'styled-components';
3
4 const Form = styled.form`
5   display: flex;
6   flex-direction: column;
7   align-items: center;
8   justify-content: center;
9   width: 100%;
10  padding: 2rem;
11  box-sizing: border-box;
12  z-index: 2;
13 `;
14
15 const Input = styled.input`
16   display: block;
17   width: 100%;
18   margin-bottom: 1rem;
19   padding: 1.25rem 1rem;
20   box-sizing: border-box;
21   border-radius: 0.25rem;
22   border: 1px solid transparent;
23   box-shadow: 0 1px 6px rgba(0, 0, 0, 0.1);
24   &:focus {
25     border-color: #6772e5;
26     outline: none;
27     box-shadow: 0 1px 6px rgba(103, 114, 229, 0.5);
28   }
29 `;
30
31 const ButtonContainer = styled.div`
32   display: flex;
33   flex-direction: row;
34   width: 100%;
35 `;
36
37 const Button = styled.button`
38   display: block;
39   background-color: #bbb;
40   color: white;
41   border: none;
42   width: 100%;
43   padding: 1.25rem 1rem;
44   box-sizing: border-box;
45   border-radius: 0.25rem;
46   text-transform: uppercase;
```



The image shows a styled-components playground interface. On the left, there is a code editor with a dark background and light-colored text. The code defines three styled components: a form, an input field, and a button container. The form is a flex column with centered items, 100% width, and 2rem padding. The input field is a block-level element with 100% width, 1.25rem padding, and a focus state with a blue border and shadow. The button container is a flex row with 100% width. The button is a block-level element with a gray background, white text, and 1.25rem padding. On the right, the rendered output is shown. It features a green header with a white circular logo containing a green rocketship icon. Below the header, the text 'Set up your payments' and 'Rocketship, Inc.' is displayed. The main content area has a light gray background and contains an email input field, a phone number input field, and two gray buttons labeled 'RESET' and 'SUBMIT'.

Source: <https://www.cssinjsplayground.com/>

VUE SINGLE FILE COMPONENTS

The screenshot displays a development environment for a Vue Single File Component (SFC) project. The code editor shows the following code:

```
1 <template>
2   <div id="app">
3     
4     <HelloWorld msg="CSS in Vue Single File Components">
5     <GlobalHeading heading="Global Heading Component"/>
6     <ScopedHeading heading="Scoped Heading Component"/>
7   </div>
8 </template>
9
10 <script>
11 import HelloWorld from "./components/HelloWorld";
12 import GlobalHeading from "./components/GlobalHeading";
13 import ScopedHeading from "./components/ScopedHeading";
14
15 export default {
16   name: "App",
17   components: {
18     HelloWorld,
19     GlobalHeading,
20     ScopedHeading
21   }
22 };
23 </script>
24
25 <style>
26 #app {
27   font-family: "Avenir", Helvetica, Arial, sans-serif;
28   -webkit-font-smoothing: antialiased;
29   -moz-osx-font-smoothing: grayscale;
30   text-align: center;
```

The browser view shows the rendered output of the SFC. It features a large green and blue Vue logo at the top. Below the logo, the text "CSS in Vue Single File Components" is displayed. Three components are listed below:

- Traditional H3
- Global Heading Component
- Scoped Heading Component** (highlighted with a red border)

Source: <https://codesandbox.io/s/vue-sfc-css-scope-m3ndm>

You probably know what slide is
coming next...



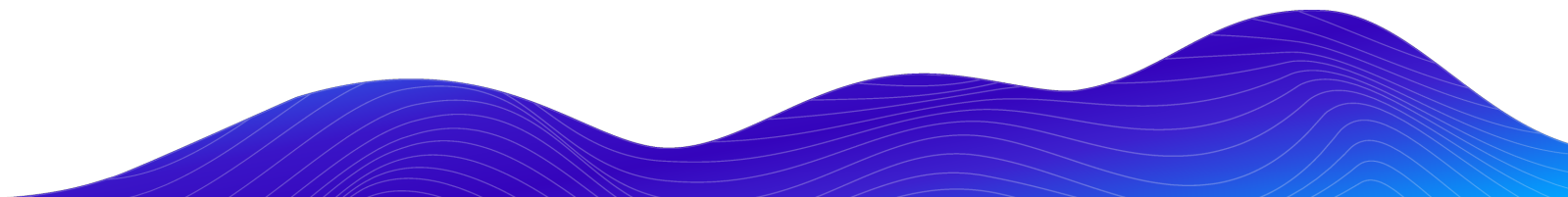
Why don't we have both?

Thank You!

Brian Perry

Lead Front End Developer

Email: brian.perry@bounteous.com





*** NEW ENGLAND ***
DRUPAL CAMP

SCOPE IN CSS WITH AND WITHOUT JAVASCRIPT

Brian Perry
November 23, 2019

Slides: <http://bit.ly/css-scope-ned>

CSS has **global scope.**

HEY, REMEMBER CSS AND BEM?

Cascade



Specificity

Stuff & Nonsense

 a 1 x element selector Sith power: 0,0,1	 p a 2 x element selectors Sith power: 0,0,2	 .foo 1 x class selector * Sith power: 0,1,0	 a.foo 1 x element selector 1 x class selector Sith power: 0,1,1
 p a.foo 2 x element selectors 1 x class selector Sith power: 0,1,2	 .foo.bar 2 x class selectors Sith power: 0,2,0	 p.foo a.bar 2 x element selectors 2 x class selectors Sith power: 0,2,2	#foo selector icon"/> #foo 1 x id selector Sith power: 1,0,0
 a#foo 1 x element selector 1 x id selector Sith power: 1,0,1	 .foo a#bar 1 x element selector 1 x class selector 1 x id selector Sith power: 1,1,1	 .foo.foo #foo 2 x class selectors 1 x id selector Sith power: 1,2,0	 style 1 x style attribute Sith power: 1,0,0,0

*Some specificity
class selector =
attribute selector =
pseudo-classes

Important

Inheritance



BEM



Modifier:
`.face_eyes``.face_eyes--open`
`.face_mouth``.face_mouth--open`

But could 'regular' CSS have
component scope?

But can 'regular' CSS have
component scope?

Not really, but...

ATOMIC (FUNCTIONAL) CSS IS A THING

- Small single purpose classes named based on visual function
- Immutable CSS
 - Classes don't change
 - Visual control shifts from CSS to markup
- Approaches:
 - Static – pre-existing set of utility classes
 - Programmatic – CSS generated based on markup

TAILWIND

A utility-first framework for rapidly building custom designs

```
1 <div class="bg-white rounded-lg">
2   
3   <div>
4     <h2>Erin Lindford</h2>
5     <div>Customer Support</div>
6     <div>erinlindford@example.com</div>
7     <div>(555) 765-4321</div>
8   </div>
9 </div>
```



Erin Lindford
Customer Support
erinlindford@example.com
(555) 765-4321

I'LL BE HONEST...



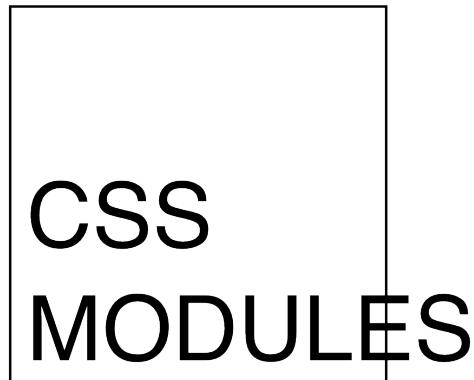
Some people love it, but to me it feels more like not using CSS...

CSS-in-JS can provide **component scope.**

But could CSS-in-JS be **more like**
'regular' CSS?

CSS MODULES

- Not a package
- Handled by a build process (typically Webpack)
- Scopes styles locally by default
- Can play nice with Sass
- Ships with Create React App and Gatsby
- Has a super snarky logo



SCOPED CSS – PROJECT CARD

Featured Projects

JavaScript

Gatsby

A framework based on React that helps developers build blazing fast websites and apps

[More info >](#)

PHP

Tome

A static site generator for Drupal 8

[More info >](#)

View All

See all projects listed on the site or add something new

[More info >](#)

```
1 // ProjectCard.module.scss
2
3
4 .project-card {
5   box-shadow: 0px 0px 2px 1px rgba(25,17,34,0.1);
6   background-color: white;
7   padding: 2rem;
8   display: block;
9   text-decoration: none;
10  border-radius: 10px;
11  width: 100%;
12
13  &:hover {
14    box-shadow: 0px 0px 2px 1px rgba(6, 120, 190, 0.3);
15  }
16
17  .title {
18    font-family: 'Overpass', Sans-Serif;
19    font-size: 30px;
20  }
21
22  .language {
23    color: #31373f;
24    font-size: 14px;
25  }
26
27  .description {
28    color: #31373f;
29  }
30
31  p {
32    font-size: 14px;
33    margin-bottom: 0;
34  }
35 }
```

```
4 // ProjectCard.js
5
6 import styles from "../ProjectCard.module.scss"
7
8 const ProjectCard = (props) => (
9   <Link
10     to={props.link}
11     key={props.link}
12   >
13     <div className={styles.projectCard}>
14       <div className={styles.language}>{props.language}</div>
15       <div className={styles.title}>{props.title}</div>
16       <div className={styles.description}>{props.description}</div>
17       <p>More info ></p>
18     </div>
19   </Link>
20 );
21
22 export default ProjectCard
23
```

SCOPED CLASS NAMES

The screenshot shows a web browser's developer tools interface. The top navigation bar includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, and Audits. A warning icon with the number '1' is visible on the right. The 'Elements' panel on the left displays a tree view of the DOM. A specific element is selected and highlighted in blue: a `<div class="ProjectCard-module--project-card--3xTSe">` element. This element contains several nested `<div>` elements, including one with `class="ProjectCard-module--language--26Z4S">JavaScript</div>` and another with `class="ProjectCard-module--title--2Fmpt">Gatsby</div>`. Below these is a `<p>More info ></p>` element. The 'Styles' panel on the right shows the CSS rules for the selected element. The filter is set to `:hov .cls`. The style sheet is identified as `ProjectCard.module.scss:1`. The CSS rules for `project-card--3xTSe` are:

```
.ProjectCard-module--project-card--3xTSe {  
  box-shadow:  
    0px 0px 2px 1px rgba(25, 17, 34,  
  background-color: white;  
  padding: 2rem;  
  display: block;  
  text-decoration: none;  
  border-radius: 10px;  
  width: 100%;  
}
```



Why don't we have both?

GLOBAL CSS

```
1 |
2 |   // ProjectCard.js
3 |
4 |   import React from "react"
5 |   import { Link } from "gatsby"
6 |
7 |   import "../ProjectCard.scss"
8 |
9 |   const ProjectCard = (props) => (
10 |     <Link
11 |       to={props.link}
12 |       key={props.link}
13 |     >
14 |       <div className="project-card">
15 |         <div className="language">{props.language}</div>
16 |         <div className="title">{props.title}</div>
17 |         <div className="description">{props.description}</div>
18 |         <p>More info ></p>
19 |       </div>
20 |     </Link>
21 |   );
22 |
23 |   export default ProjectCard
24 |
```

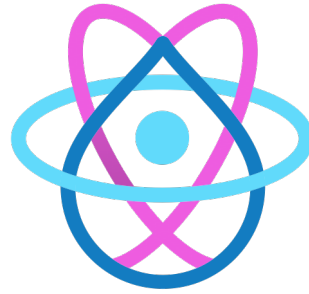
STYLES SHARED BY MULTIPLE TEMPLATING ENGINES

Drupal



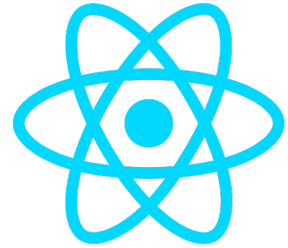
- Twig
- Unique styles and components

Combined



- Global styles that apply to both.
- Few (if any) shared components

React



- JSX
- Unique styles and components

STYLES SHARED BY MULTIPLE TEMPLATING ENGINES

Drupal



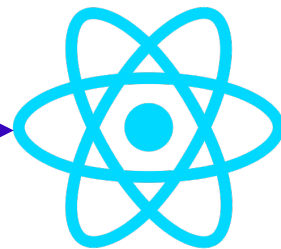
- Global CSS file - combines design system partials and Drupal specific partials

Sass

CSS
MODULES

- Projects consume partials as needed

React



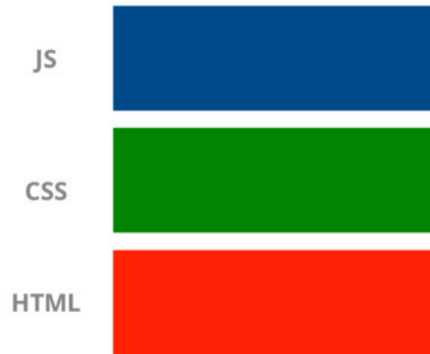
- Imported in JS
- Mix of global and component scope
- Also project specific CSS

We **can** have both!



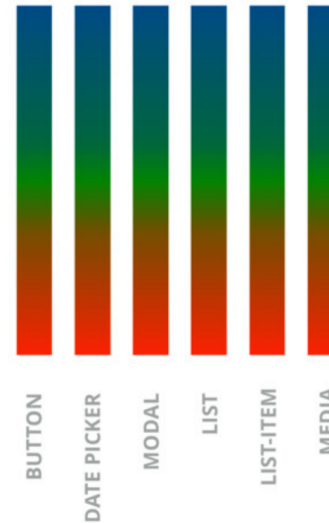
SEPARATION OF CONCERNS IS IN THE EYE OF THE BEHOLDER

Separation of Concerns



Separation of Concerns

(only, from a different point of view)



Source: <https://speakerdeck.com/didoo/let-there-be-peace-on-css?slide=62>

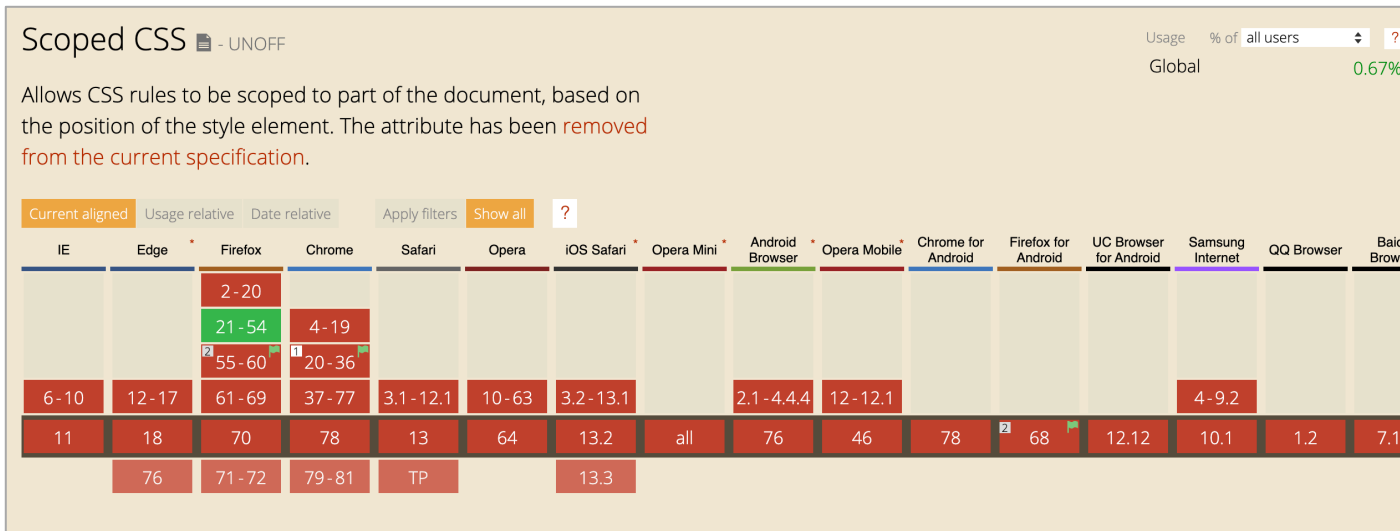
But could we have component scope
with **less JS?**

POPULAR EXTENSIONS BECOME OFFICIAL

- CSS Preprocessors influence CSS
 - CSS Custom Properties
 - CSS Nesting
- Popular Drupal contrib projects make their way into core (I said Drupal, aren't you proud?)
 - JSON:API
 - Panels Ecosystem -> Layout Builder
- Babel lets you develop with cutting edge JS today
 - Transpile today, browser native tomorrow.

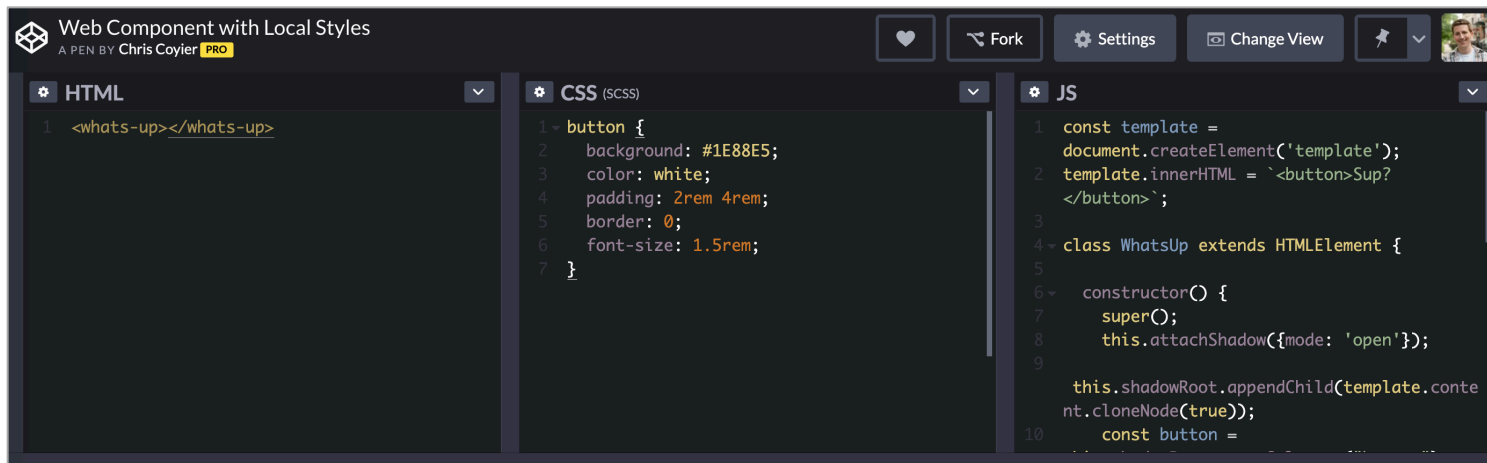
COULD WE SEE THE SAME FOR CSS SCOPE?

- HTML Scoped Attribute... kind of didn't happen.
 - `<style scoped>`



COULD WE SEE THE SAME FOR CSS SCOPE?

- Web Components / Shadow DOM as a solution?
 - Shadow DOM styles scoped locally by default.
 - Approaches also exist to use or selectively override global styles.
 - Still requires writing a decent amount of JS.



The screenshot shows a code editor with three panels: HTML, CSS (SCSS), and JS. The title bar reads "Web Component with Local Styles" by Chris Coyier. The HTML panel contains a single line of code: `<whats-up></whats-up>`. The CSS panel shows a SCSS class `button` with the following styles: `background: #1E88E5;`, `color: white;`, `padding: 2rem 4rem;`, `border: 0;`, and `font-size: 1.5rem;`. The JS panel shows a class `WhatsUp` extending `HTMLElement`, with a constructor that creates a template, attaches a shadow DOM, and appends the template content to the shadow root.

```
HTML
1 <whats-up></whats-up>

CSS (SCSS)
1 button {
2   background: #1E88E5;
3   color: white;
4   padding: 2rem 4rem;
5   border: 0;
6   font-size: 1.5rem;
7 }

JS
1 const template =
2   document.createElement('template');
3   template.innerHTML = `<button>Sup?
4   </button>`;
5
6 class WhatsUp extends HTMLElement {
7
8   constructor() {
9     super();
10    this.attachShadow({mode: 'open'});
11
12    this.shadowRoot.appendChild(template.conte
13    nt.cloneNode(true));
14    const button =
```



Impossible to see, the future is.

A COUNTERPOINT:

PUTTING THE "C" BACK IN CSS: MODERN TECHNIQUES FOR TODAY'S WEBSITE

There's been a lot of discussion about CSS-in-JS and all the benefits it gives. It might have its place in Javascript applications, but what if you want to reuse your CSS across multiple websites? I'll talk about why plain old CSS and preprocessors are still relevant and why you'd still want to use them on your project.

In this talk, I'll go over:

- Why CSS-in-JS isn't the answer for every scenario
- A basic introduction to BEM and how to use it in your design system or website
- Why preprocessors such as Sass still have a place in the front-end development world
- What makes the 'cascade' in Cascading Style Sheets so useful

You will come away from this session knowing why CSS and preprocessors are still relevant today and how you can implement more modern CSS practices to your design system or website.

This session assumes a basic knowledge of CSS and basic understanding of modern front-end development practices.

Audience Level: Beginner

Topic: Front end

Timeslot: 2:30 - 3:15pm

Conf Day: [2019 Sessions Day \(Saturday\)](#)

Room: Gaige Classroom 201

Thank You!

Brian Perry

Lead Front End Developer

Email: brian.perry@bounteous.com

