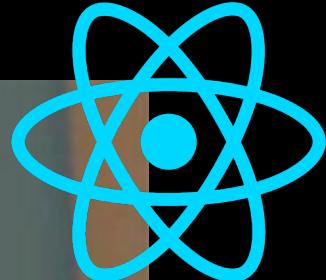


LitElement and Lit-HTML

To infinity and beyond!



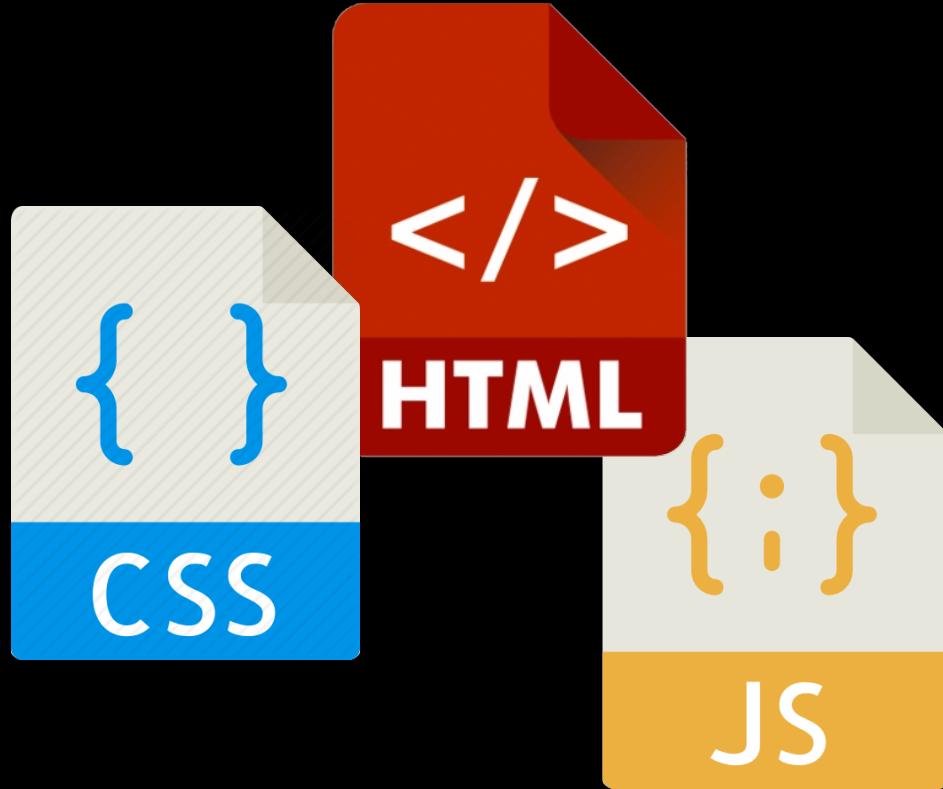
OPINIONS...



OPINIONS EVERYWHERE.

makeameme

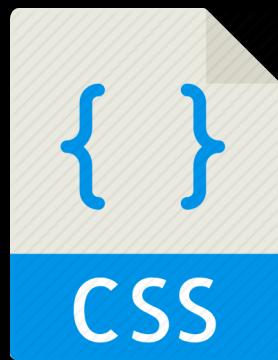
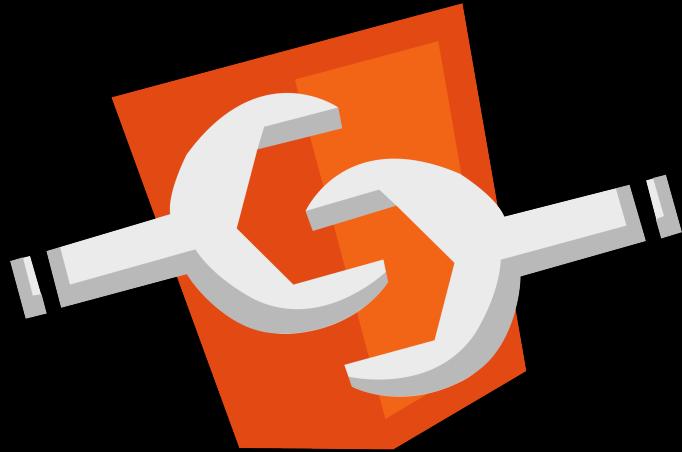




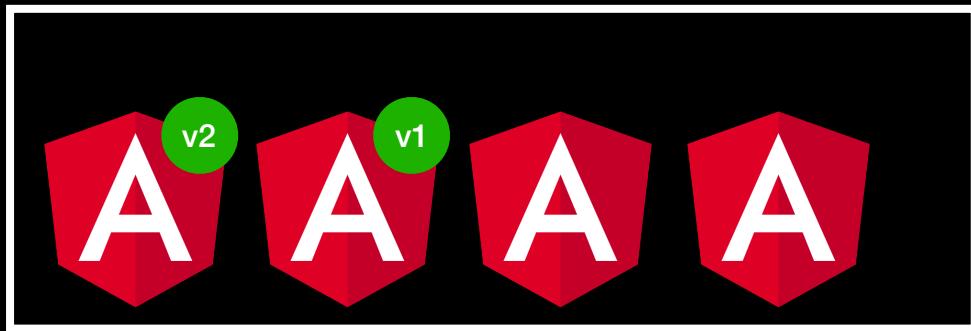
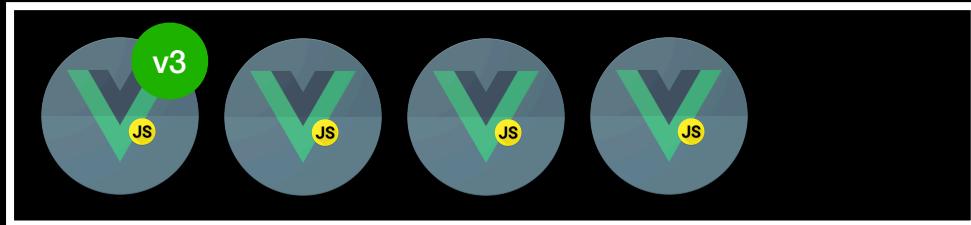
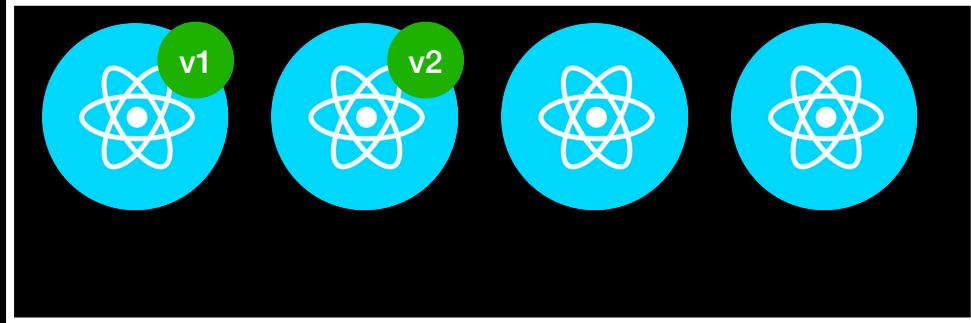
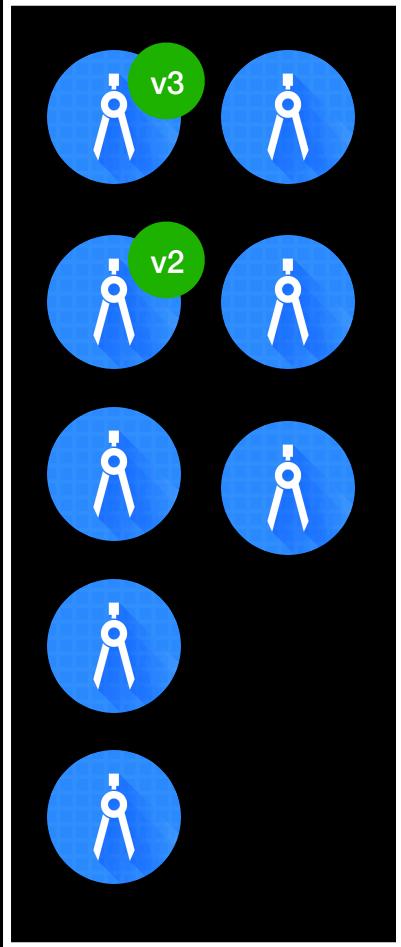


FASHION TRENDS

Some Are Just Stupid.

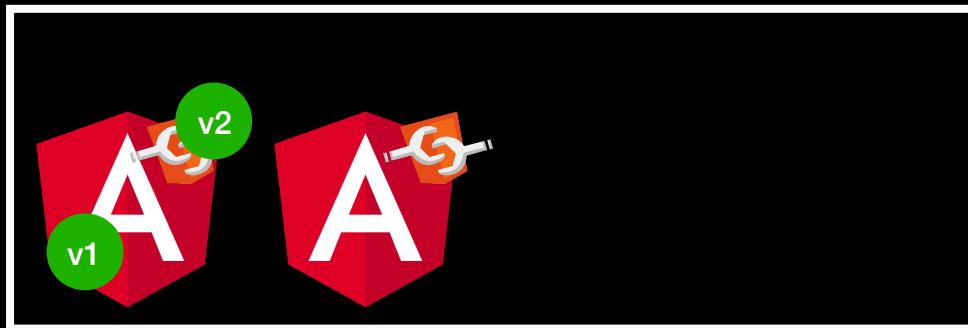
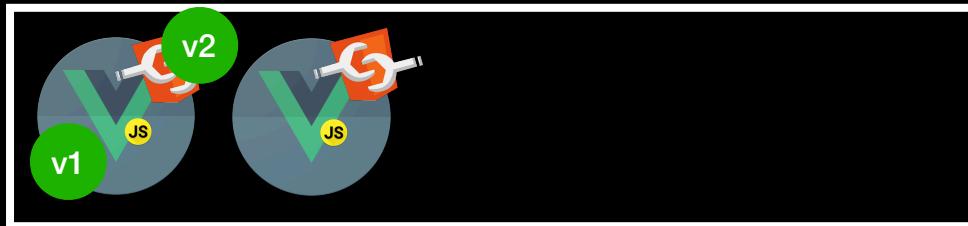
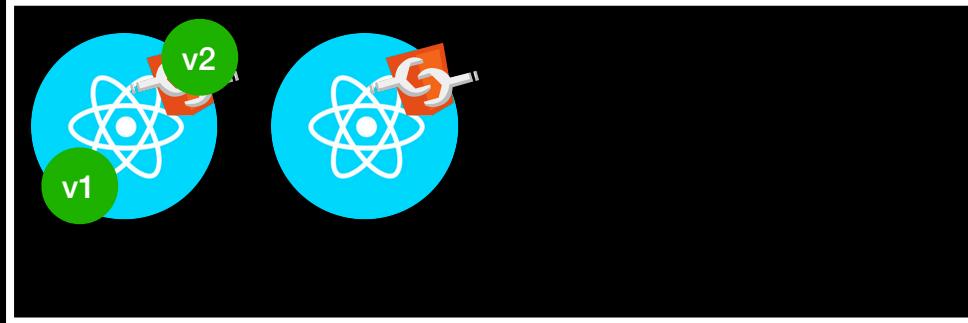
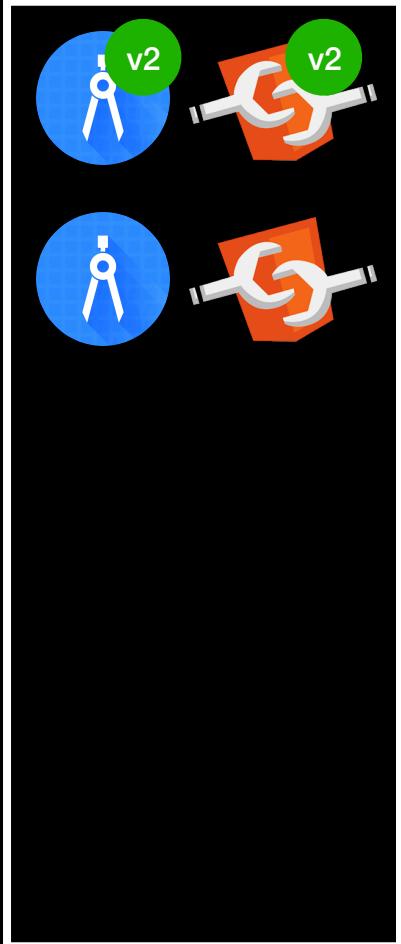


Design System



Customers

Design System



Customers

First there was Polymer ...

- 2015-05-29: Polymer 1.0 released
- 2015-09-24: Build components with ES6 classes
- 2016-03-10: Community Highlights - GE Predix UI
- 2016-05-19: Polymer at Google I/O 2016
- 2017-01-09: [webcomponents.org](#) launched
- 2017-05-15: Polymer 2.0 released
- 2018-05-09: Polymer 3.0 released
- 2018-12-13: lit-html 1.0 released
- 2019-01-11: litElement released

Today ... major software companies have released tools based on Web Component technologies. Apple, Microsoft, and Google of course have made major investments into this tech!

2019-02-05	Lightning-fast templates & Web Components: lit-html & LitElement	 Justin Fagnani
2019-01-11	LitElement release candidate	 Justin Fagnani
2018-12-13	lit-html 1.0 release candidate	 Gray Norton
2018-10-22	Latest releases from the Polymer Project An update on the latest releases from the Polymer Project.	 Polymer Team
2018-10-02	Web Components v0 deprecations With new Web Components specs shipping widely, older web components standards are being deprecated and removed from Chrome.	 Polymer Team
2018-05-25	Polymer Elements 3.0 FAQ FAQ about updating your application to use the Polymer Elements 3.0	 Polymer Team
2018-05-09	Polymer @ Google I/O 2018 Polymer 3.0 released, next-generation products unveiled.	 Polymer Team
2018-05-02	Roadmap update, part 2: FAQ Frequently-asked-questions on our roadmap update.	 Gray Norton
2018-05-02	Roadmap update, part 1: 3.0 and beyond Updates on the Polymer 3.0 release and what comes	

The screenshot shows a Firefox browser window with a dark theme. The title bar says "The Firefox UI is now built with". The address bar shows the URL <https://briangrinstead.com/blog/firefox-webcomponents/>. The page content is as follows:

Brian Grinstead

[about](#) [github](#) [twitter](#) [rss](#)

The Firefox UI is now built with Web Components

A couple of weeks ago, we landed a [commit](#) that took years of effort at Mozilla. It removed “XBL”, which means we’ve completed the process of migrating the Firefox UI to Web Components. It wasn’t easy – but I’ll get to that later.

The Firefox UI can be thought of as a very large single page app. It was built with DOM and JS from the start, which was a bold technology choice for a native application 20+ years ago. Because of this, Mozilla implemented some features necessary to build complex web applications way before the Web platform supported them.

Over time, these features have [evolved](#) into specifications like CSS flexbox and Web Components. When this happens, it’s easy to allow the existing codebase to continue to use the original version, and require the platform supports both features. This makes sense – rewriting legacy code that already works is difficult and not always

Some Projects

- [Spectrum Colorpicker](#)
- [Colorstash](#)
- [FileReader.js](#)
- [videoconverter.js](#)
- [jQuery UI Anglepicker](#)
- [Instant Sprite](#)
- [Gradient Generator](#)
- [TinyColor Color Parsing](#)
- [iOS UIColor Picker](#)
- [bindWithDelay Plugin](#)
- [DevTools Snippets](#)
- [A* Graph Search](#)
- [C# Multipart Form Post](#)

beta.music.apple.com/us/browse

Apple Music

Sign In

Browse

For You

Browse

Radio

Open in iTunes ↗

UPDATED PLAYLIST

Rap Life

Apple Music Hip-Hop

NEW ALBUM

Rare

Selena Gomez

RAP LIFE

You Gotta Hear

NMD New Music Daily

See All

R&B NOW

Get millions of songs. All ad-free.

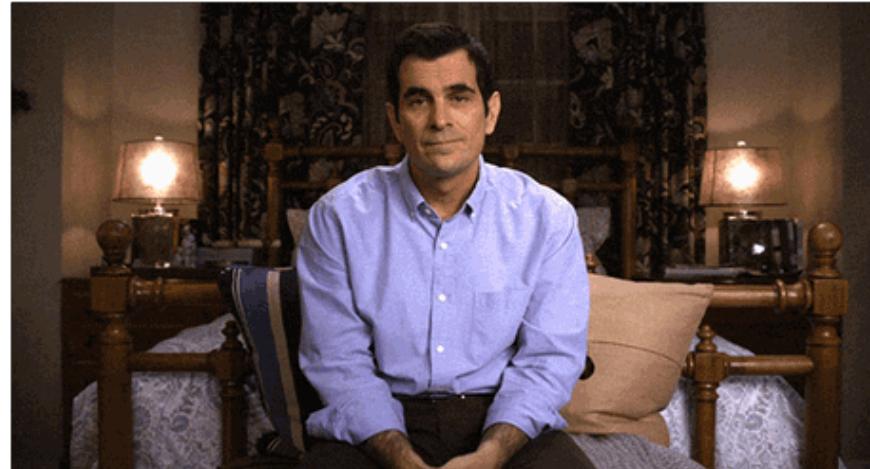
Plus your entire music library on all your devices.

Try It Now

The screenshot shows the Apple Music beta website interface on a Mac OS X desktop. The top navigation bar includes the URL 'beta.music.apple.com/us/browse', the Apple logo, and a 'Sign In' button. On the left, there's a sidebar with 'For You', 'Browse' (which is selected and highlighted in grey), and 'Radio' options, along with a link to 'Open in iTunes'. The main content area is titled 'Browse' and features several promotional sections: 'UPDATED PLAYLIST' for 'Rap Life' from 'Apple Music Hip-Hop', 'NEW ALBUM' for 'Rare' by 'Selena Gomez', a large image of Selena Gomez, and a 'You Gotta Hear' section with 'NMD New Music Daily' and 'R&B NOW' playlists. At the bottom, a purple banner promotes 'millions of songs. All ad-free.' and 'your entire music library on all your devices.', with a 'Try It Now' button.

What about IE?

IE11 is supported with a polyfill supported by
web components.org ... so, meh



The future is now!

1. Chrome
2. Safari
3. Firefox
4. Microsoft Edge



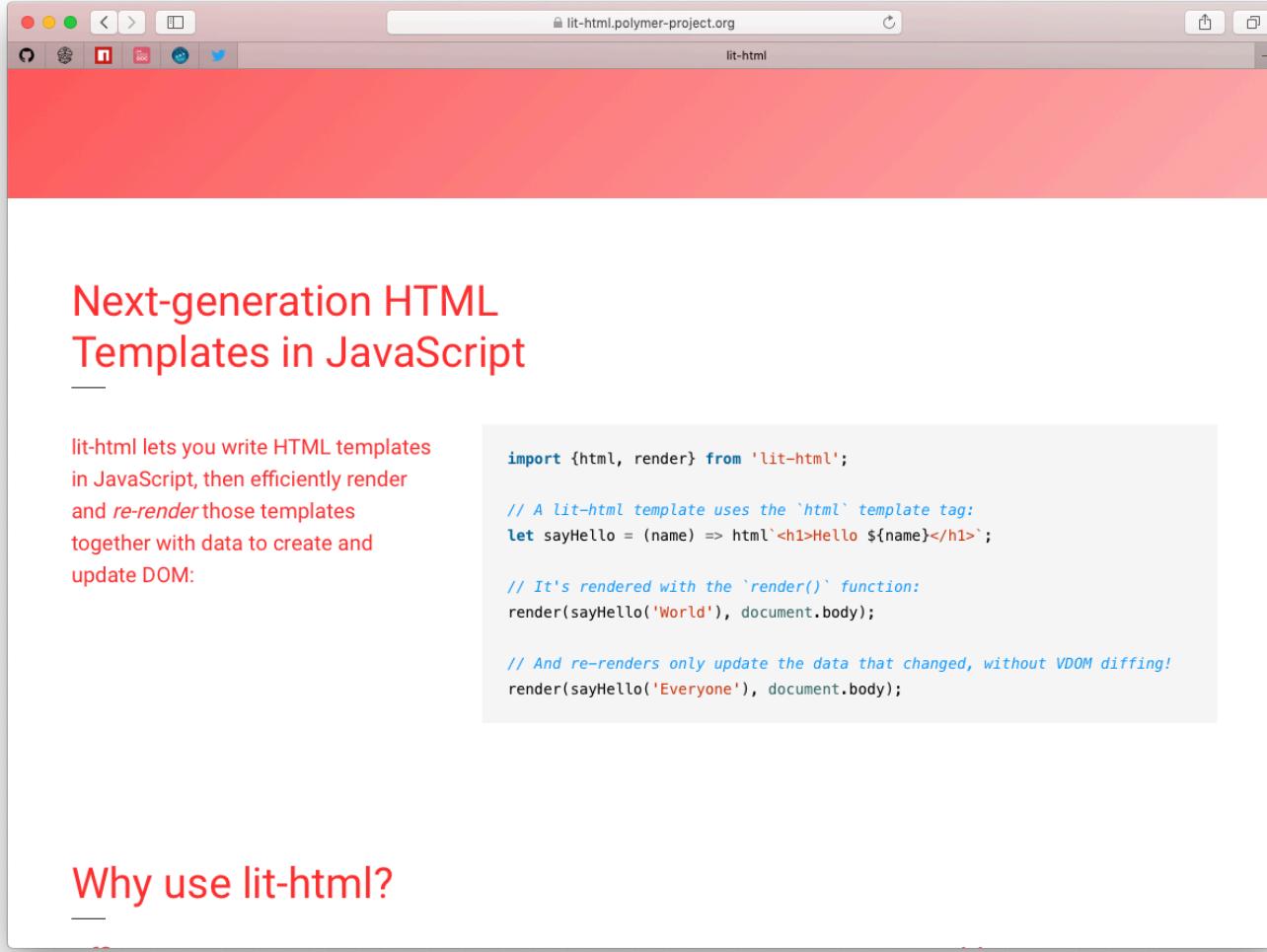
Official release 1.15.20

let's build our 1st component

To build a web component you need a few things ...

1. A computer
2. Web browser
3. Code editor

That's it.[®]



A screenshot of a web browser window titled "lit-html". The page content is displayed on a white background with a red header bar.

Next-generation HTML Templates in JavaScript

lit-html lets you write HTML templates in JavaScript, then efficiently render and *re-render* those templates together with data to create and update DOM:

```
import {html, render} from 'lit-html';

// A lit-html template uses the `html` template tag:
let sayHello = (name) => html`<h1>Hello ${name}</h1>`;

// It's rendered with the `render()` function:
render(sayHello('World'), document.body);

// And re-renders only update the data that changed, without VDOM diffing!
render(sayHello('Everyone'), document.body);
```

Why use lit-html?

The screenshot shows a web browser window with the URL lit-html.polymer-project.org/guide/writing-templates. The page title is "Writing templates – lit-html". The main content area has a red header bar with the text "lit-html". On the left, there is a sidebar with a list of links: Introduction, Getting Started, Writing templates (which is the current page), Styling templates, Rendering templates, Creating directives, Template syntax reference, Concepts, Tools, and Community. The main content area features a large red header "Writing templates". Below it is a bulleted list of features:

- Render static HTML
- Render dynamic text content
- Using expressions
- Bind to attributes
- Bind to properties
- Add event listeners
- Nest and compose templates
 - Conditional templates
 - Repeating templates
 - Rendering nothing
- Caching template results: the cache directive

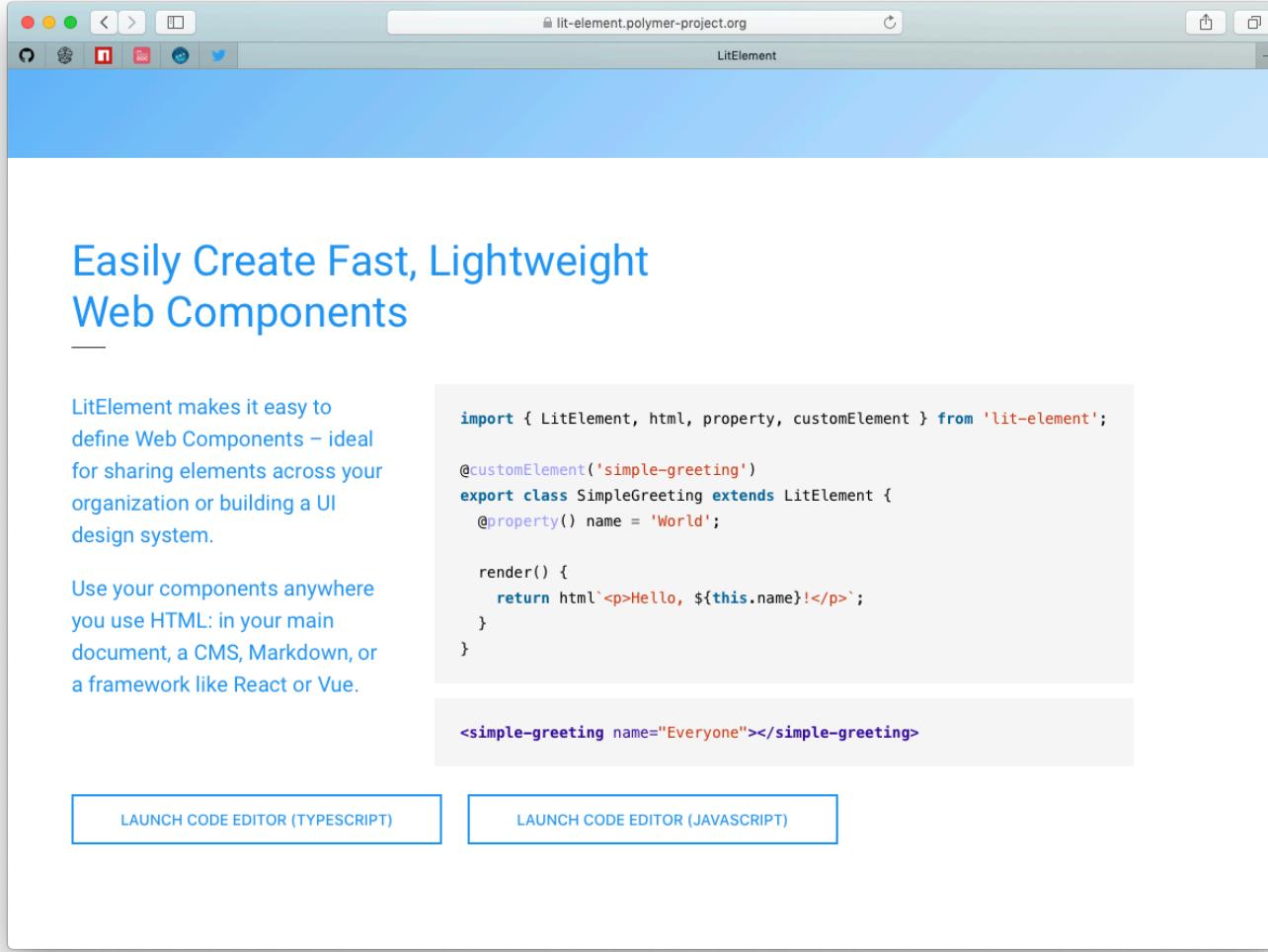
Text below the list states: "lit-html is a templating library that provides fast, efficient rendering and updating of HTML. It lets you express web UI as a function of data." A note below that says: "This section introduces the main features and concepts in lit-html."

Render static HTML

The simplest thing to do in lit-html is to render some static HTML.

```
import {html, render} from 'lit-html';

// Declare a template
```

A screenshot of a web browser window displaying the LitElement homepage. The URL in the address bar is "lit-element.polymer-project.org". The page title is "LitElement". The main heading is "Easily Create Fast, Lightweight Web Components". Below the heading, there is a section about LitElement's ease of use for defining Web Components and sharing them across organizations. Another section discusses using components anywhere HTML is used. To the right, there is a code block for a TypeScript-based component definition and an example of its usage in HTML. At the bottom, there are two buttons: "LAUNCH CODE EDITOR (TYPESCRIPT)" and "LAUNCH CODE EDITOR (JAVASCRIPT)".

Easily Create Fast, Lightweight Web Components

LitElement makes it easy to define Web Components – ideal for sharing elements across your organization or building a UI design system.

Use your components anywhere you use HTML: in your main document, a CMS, Markdown, or a framework like React or Vue.

```
import { LitElement, html, property, customElement } from 'lit-element';

@customElement('simple-greeting')
export class SimpleGreeting extends LitElement {
  @property() name = 'World';

  render() {
    return html`<p>Hello, ${this.name}!</p>`;
  }
}

<simple-greeting name="Everyone"></simple-greeting>
```

[LAUNCH CODE EDITOR \(TYPESCRIPT\)](#)

[LAUNCH CODE EDITOR \(JAVASCRIPT\)](#)

The screenshot shows a web browser window displaying the LitElement documentation at lit-element.polymer-project.org/guide/templates. The page title is "Templates – LitElement". The left sidebar contains navigation links for "Introduction", "Getting Started", "Writing components" (with "Templates" selected), "Styles", "Properties", "Events", "Lifecycle", "Tools" (with "Publish a component" and "Use a component" listed), and "Resources" (with "Community" and "lit-html Guides" listed). The main content area has a large blue header "Templates". Below it is a bulleted list of topics: "Define and render a template" (with sub-points "Design a performant template", "Use properties, loops, and conditionals in a template", "Bind properties to templated elements", and "Render children with the slot element"); "Compose a template from other templates"; "Specify the render root"; "Template syntax cheat sheet"; "Using other lit-html features" (with sub-point "Import and use a lit-html directive"); and "Resources". A note below the list says "Add a template to your component to define internal DOM to implement your component." Another note explains that "To encapsulate the templated DOM LitElement uses shadow DOM. Shadow DOM provides three benefits:" followed by a bulleted list of three benefits: "DOM scoping", "Style scoping", and "Composition". At the bottom, a note states "Where native shadow DOM isn't available, LitElement uses the [Shady CSS polyfill](#)".

Templates

- Define and render a template
 - Design a performant template
 - Use properties, loops, and conditionals in a template
 - Bind properties to templated elements
 - Render children with the slot element
- Compose a template from other templates
- Specify the render root
- Template syntax cheat sheet
- Using other lit-html features
 - Import and use a lit-html directive
- Resources

Add a template to your component to define internal DOM to implement your component.

To encapsulate the templated DOM LitElement uses [shadow DOM](#). Shadow DOM provides three benefits:

- DOM scoping. DOM APIs like `document.querySelector` won't find elements in the component's shadow DOM, so it's harder for global scripts to accidentally break your component.
- Style scoping. You can write encapsulated styles for your shadow DOM that don't affect the rest of the DOM tree.
- Composition. The component's shadow DOM (managed by the component) is separate from the component's children. You can choose how children are rendered in your templated DOM. Component users can add and remove children using standard DOM APIs without accidentally breaking anything in your shadow DOM.

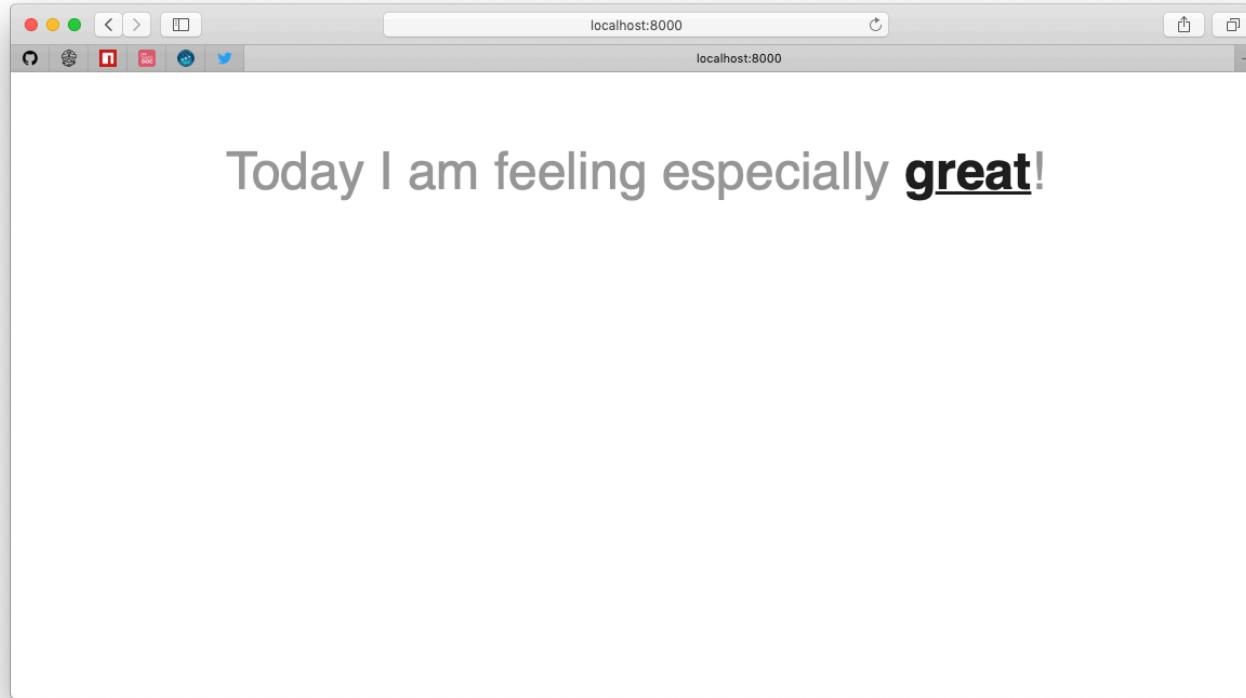
Where native shadow DOM isn't available, LitElement uses the [Shady CSS polyfill](#).

let's build our 1st component

```
● ● ●  
1 import {LitElement, html, css} from 'https://unpkg.com/lit-element/lit-element.js?module';  
2  
3 class MyMood extends LitElement {  
4  
5   static get properties() {  
6     return {  
7       mood: { type: String }  
8     }  
9   }  
10  
11  static get styles() {  
12    return css`  
13      :host {  
14        display: block;  
15        font-family: sans-serif;  
16        font-size: 3rem;  
17        color: #999;  
18        text-align: center;  
19        margin: 4rem 0;  
20      }  
21      span {  
22        text-decoration: underline;  
23        font-weight: bold;  
24        color: #222;  
25      }  
26    `;  
27  }  
28  
29  render() {  
30    return html`  
31      Today I am feeling especially <span>${this.mood}</span>!  
32    `;  
33  }  
34  
35}  
36  
37 customElements.define('my-mood', MyMood);  
38
```

let's use our 1st component

```
● ● ●  
1 <html>  
2 <head>  
3   <!-- Polyfills only needed for Firefox and Edge. -->  
4   <script  
5     src="https://unpkg.com/@webcomponents/webcomponentsjs@latest/webcomponent  
6       s-loader.js"></script>  
7 </head>  
8 <body>  
9   <script type="module" src=".//my-mood.js"></script>  
10 </body>  
11 </html>  
12
```



let's review our 1st component



```
● ● ●

1 import {LitElement, html, css} from 'https://unpkg.com/lit-element/lit-element.js?module';
2
3 class MyMood extends LitElement {
4
5   static get properties() {
6     return {
7       mood: { type: String }
8     }
9   }
10
11  static get styles() {
12    return css`
13      :host {
14        display: block;
15        font-family: sans-serif;
16        font-size: 3rem;
17        color: #999;
18        text-align: center;
19        margin: 4rem 0;
20      }
21      span {
22        text-decoration: underline;
23        font-weight: bold;
24        color: #222;
25      }
26    `;
27  }
28
29  render() {
30    return html`
31      Today I am feeling especially <span>${this.mood}</span>!
32    `;
33  }
34
35 }
36
37 customElements.define('my-mood', MyMood);
38
```

let's
our 1
com

A screenshot of a Mac OS X desktop environment. In the top-left corner, there is a window titled "let's our 1 com". To its right, a web browser window is open at the URL "localhost:8000". The browser shows a single-line text message: "Today I am feeling especially great!". Below this message, a dark rectangular callout box contains the HTML code: "<my-mood mood='great'></my-mood>". At the bottom of the slide, there is a snippet of code with line numbers 34 through 38.

```
1 import {LitElement, html, css} from 'https://unpkg.com/lit-element/lit-element.js?module';
2
3 class MyMood extends LitElement {
4
5   static get properties() {
6     return {
7       mood: {
8         type: String,
9         reflect: true
10      }
11    };
12  }
13
14  render() {
15    return html`

Today I am feeling especially ${this.mood}!

`;
16  }
17}
18
19customElements.define('my-mood', MyMood);
20
21
```

let's build our 3rd component

```
1 import {LitElement, html, css} from 'https://unpkg.com/lit-element/lit-element.js?module';
2
3 class MyAlert extends LitElement {
4
5   static get styles() {
6     return css`  

7       :host {  

8         display: block;  

9         font-family: sans-serif;  

10        text-align: center;  

11        margin: 40px 0;  

12      }  

13      button {  

14        background-color: blue;  

15        padding: 0 1em;  

16        line-height: 2;  

17        border-radius: 5px;  

18        font-size: 3rem;  

19        color: #fff;  

20      }  

21      button:hover {  

22        cursor: hand;  

23      }
24    `;
25  }
26
27  alert = (e) => {
28    console.log(e.target);
29  }
30
31  render() {
32    return html`  

33      <button @click="${this.alert}">  

34        <slot></slot>
35      </button>
36    `;
37  }
38
39 }
40
41 customElements.define('my-alert', MyAlert);
42
```

let's
our
com

A screenshot of a web browser window showing a blue alert box with the text "Hello World!". Below the browser is a dark overlay with the three macOS window control buttons (red, yellow, green) and the text "<my-alert>Hello World!</my-alert>". To the right of the browser is the developer tools sidebar, which includes tabs for Elements, Console, Debugger, and Logs. The Logs tab is selected, showing a log entry: "alert — my-alert.js:28". The sidebar also displays a file icon with "22" files and a gear icon.

localhost:8000

Hello World!

localhost:8000

Elements Console Debugger > + ⚙

All Evaluations Errors Warnings Logs

Console cleared at 5:43:09 PM

E	<slot></slot>	f	alert — my-alert.js:28
E	<slot></slot>	f	alert — my-alert.js:28
	<slot></slot>	f	alert — my-alert.js:28
	<slot></slot>	f	alert — my-alert.js:28

->
script>
>

let's
our
com

The screenshot shows a browser window with two tabs open, both displaying "localhost:8000". The left tab shows a large blue button with the text "Hello World!". The right tab is a developer tools window with the "Elements" and "Styles" tabs selected.

Elements Tab:

```
<html>
  <head>...</head>
  <body>
    <script type="module"
      src="./my-
        alert.js"></script>
    <my-alert>
      <button>Hello
        World</button>
    </my-alert>
  </body>
</html>
```

Styles Tab:

```
button {
  background-color: blue;
  padding: 0 1em;
  line-height: 2;
  border-radius: 5px;
  font-size: 3rem;
  color: #fff;
}
```

A sidebar on the right lists "User Agent Stylesheet" rules for buttons, including properties like align-items, text-align, cursor, color, padding, border-width, and border-style.

let's
our
com

The screenshot shows a browser window with two tabs, both displaying "localhost:8000". The left tab shows a large blue button with the text "Hello World!". Below it is a smaller button labeled "Hello World". The right tab is a developer tools panel with the "Elements" tab selected. The DOM tree on the left shows the following structure:

```
<html>
  <head>...
  <body>
    <script type="module"
      src="./my-
        alert.js"></script>
    <my-alert>
      <slot></slot>
    </my-alert>
    <button>Hello
      World</button> = $0
  </body>
</html>
```

The "Styles" tab on the right displays the CSS rules applied to the "button" element. The "User Agent Stylesheet" section includes the following rules:

```
input:matches([type="button"], [type="submit"], [type="reset"]),
input[type="file"]::-webkit-file-upload-button, button {
  align-items: flex-start;
  text-align: center;
  cursor: default;
  color: buttontext;
  padding-top: 2px;
  padding-right: 6px;
  padding-bottom: 3px;
  padding-left: 6px;
  border-top-width: 2px;
  border-right-width: 2px;
  border-bottom-width: 2px;
  border-left-width: 2px;
  border-top-style: outset;
  border-right-style: outset;
  border-bottom-style: outset;
  border-left-style: outset;
  border-top-color: buttonface;
  border-right-color: buttonface;
  border-bottom-color: buttonface;
  border-left-color: buttonface;
  border-image-source: initial;
  border-image-slice: initial;
  border-image-width: initial;
}
```

At the bottom of the developer tools, there is a "Filter" input field and a "Classes" button.

In closing ...

1. HTML Web Components are ready for prime-time
2. WCs are being used in enterprise web apps by major companies
3. No frameworks or special tooling required
4. 2019 was a coming of age ... 2020 is going to BLOW UP!



Thanks!