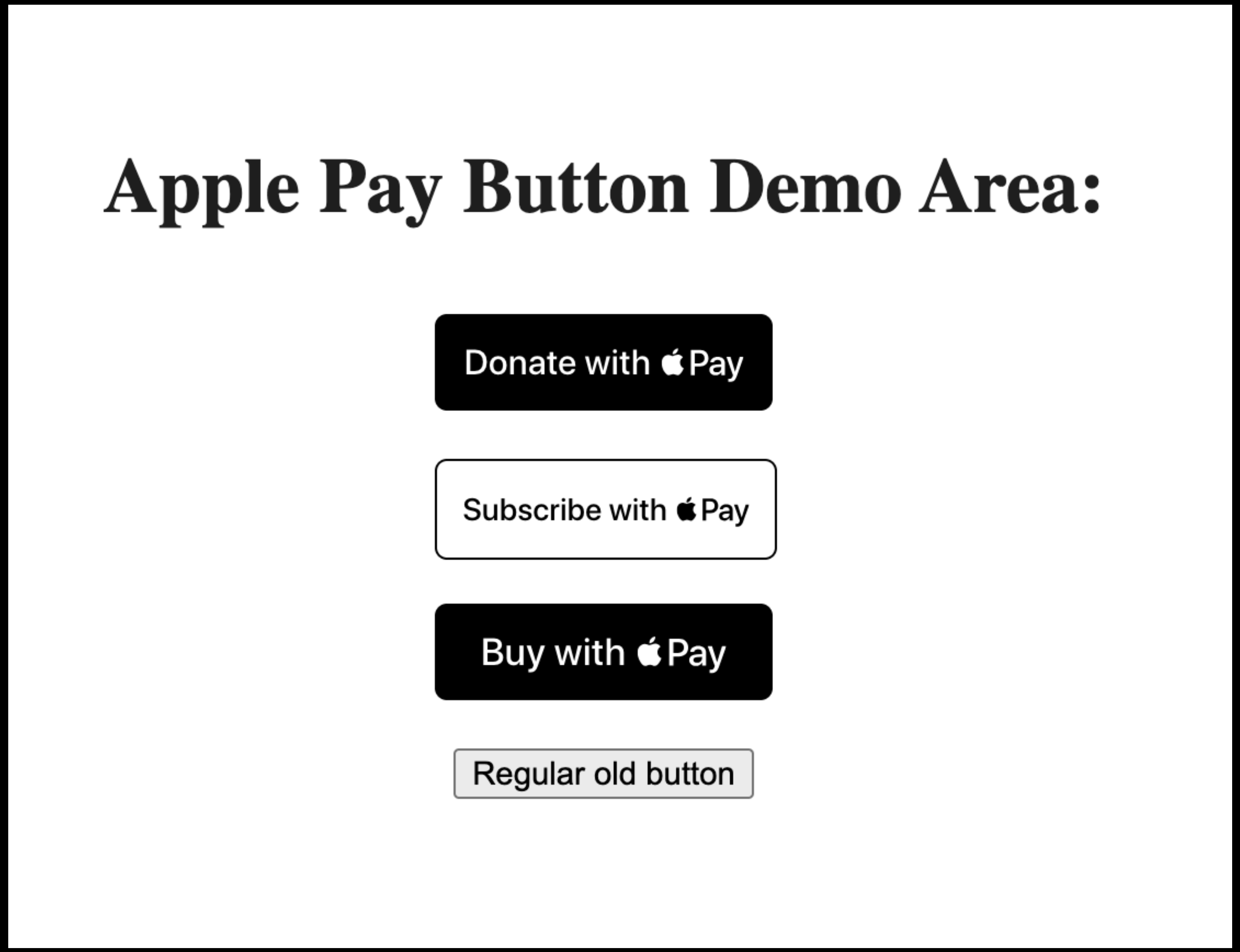
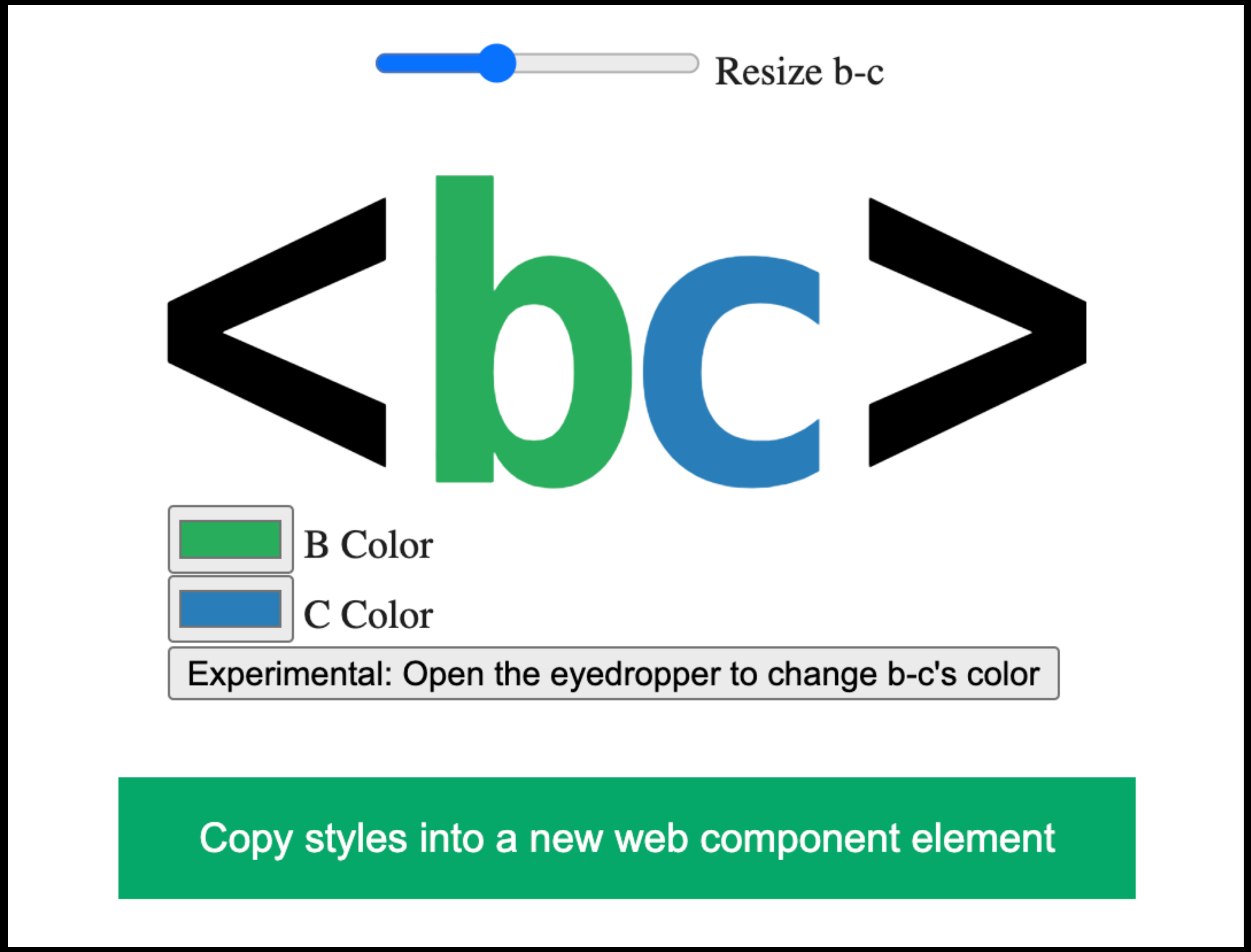
Web Components

A look at the Apple Pay Button and our own BC logo as a web component

Morgan Murrah - 09/05/2024




Resize b-c
<bc>
B Color
C Color
Experimental: Open the eyedropper to change b-c's color
Copy styles into a new web component element

Introduction

Morgan Murrah

- Working in technology and a lot on the web for about 8 years, Lived in Bellingham since 2018
- Atlanta GA (1989-1998, 2016-2017) & Asheville NC (2017) before that.
- Graduated law school and lived in New Zealand before that (1998-2016)
- Changed career in 2016 into technology

Introduction cont. — Career highlights after changing careers:

- 4 years at a startup as a software developer on a team of 3-4 for a small company, LAMP stack
- 10 months AMP HTML Development for a large corporate client
- 1.5 years and counting as a Digital Content Manager working for some big clients
- Credited volunteer at the Worldwide Web Consortium (W3C) 6+ times on a specification, the Web Sustainability Guidelines
- Really appreciate Bellingham Codes (has helped me find my last two jobs)

Thank you's/Credits for inspiration

There are a few to name drop….

- Scott Jehl, Engineer at Squarespace and co-creator of WebPageTest, also creator of the course(s) Web Components Demystified and Lightning-fast Web Performance
- Alex Russell, Partner Product Manager on Microsoft Edge
- Tantek Çelik, web standards lead at Mozilla
- Katrina Grace, independent developer who made Facet the library for making Web Components with mostly HTML that got me into this topic.

All contributed a little to this talk and a lot to web components

Components

Components

Components

Components

Components

Components

Components

Components

What does that even mean?

There are a bunch of things called components

The Wikipedia article looks amateur?

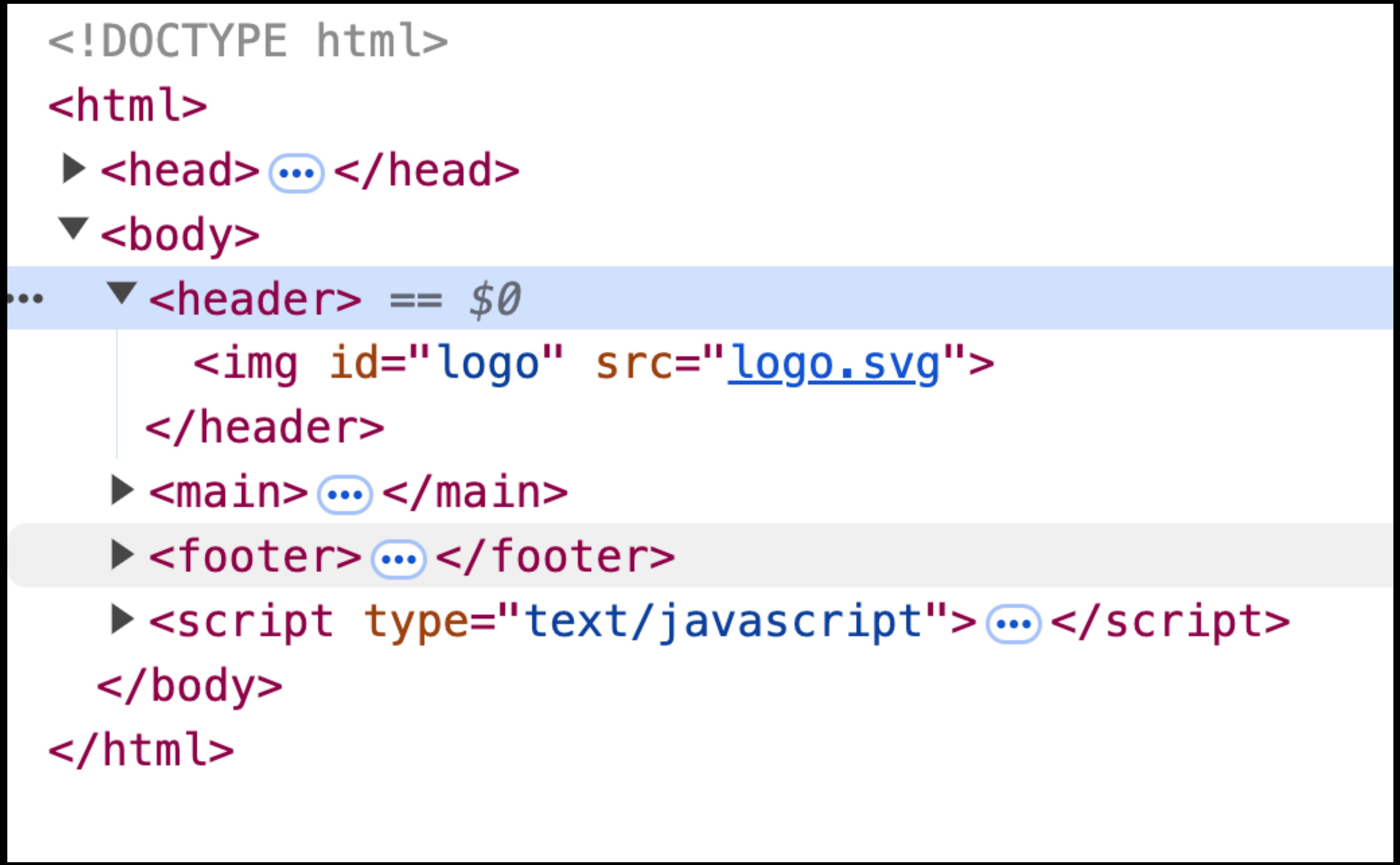
By Web Components, we mean this.
A specific set of features that work together that now enjoy strong browser support.

Browser support	CHROME	OPERA	SAFARI	FIREFOX	EDGE
HTML TEMPLATES	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE
CUSTOM ELEMENTS	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE
SHADOW DOM	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE
ES MODULES	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE	✓ STABLE

Useful background — Stuff that comes with the browser

Going to try and explain importance of things as we go even if we don't fully cover these…

- HTML - Hyper Text Markup Language
- The building block elements of the page embedded in the browser.

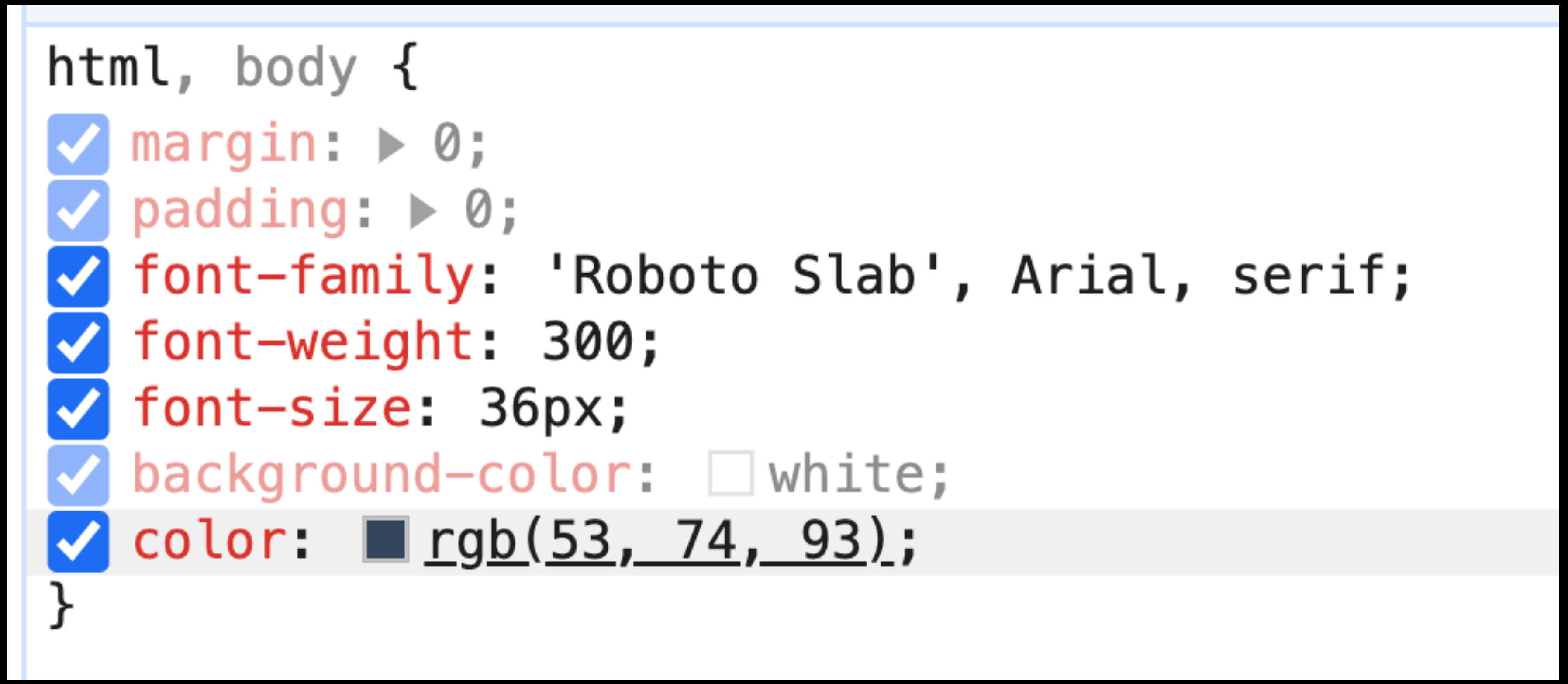


```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body>
…  ▼<header> == $0
      <img id="logo" src="logo.svg">
    </header>
   ▶<main>…</main>
   ▶<footer>…</footer>
   ▶<script type="text/javascript">…</script>
  </body>
</html>
```

CSS

Cascading Style Sheets

- Stylesheets. Controls presentation and behavior of elements. Background colors and fonts and way more.
- Features like variables, functions, pretty amazing stuff can be done with just CSS.
- Is a very highly optimized platform for some features



```
html, body {
✓ margin: ▶ 0;
✓ padding: ▶ 0;
✓ font-family: 'Roboto Slab', Arial, serif;
✓ font-weight: 300;
✓ font-size: 36px;
✓ background-color: white;
✓ color: rgb(53, 74, 93);
}
```

JavaScript - huge topic to try and summarize

- Very important programming language embedded in the browser…
- Not required for web pages to work but basically to be expected on most all interactive web pages. Millions of dollars and whole careers made on JavaScript.
- A lot of web pages wont work without JS enabled.
- Check out the HTTP Almanac — great statistical info on growth of JavaScript and commentary

More useful background - DOM

- Document Object Model (DOM) — connects web pages to scripts or programming languages by representing the structure of a document—such as the HTML representing a web page—in memory.
    - Essentially makes the code into the page “alive” by creating a tree structure that allows for programs to access it, traverse it, manipulate it with JavaScript usually.
    - This exists on every web page opened in a browser

Preface

Focusing on the Why, with a little What and How

- Web Components are in production use out there in some interesting places.
  - Adobe Photoshop for the Web - Extensive use of web components in UI, menus and toolbars and more.
  - SpaceX - Chromium Base UI used by astronauts in space using web components
  - MSFT Edge - Incrementally replacing React contained in Surface UI with Web Components
  - GitHub - uses a bunch of them
  - Apple Pay Button, discussed more later, a durable Web Component intended to work in a wide variety of environments.

What Web Components are not…

- Not a library (although there are many libraries). Some argument whether libraries are necessary to make them usable and useful especially pre-consistent browser support.
- Not a set of particular components (although many sets of components out there).
- Not a startup company or vendor product (although some vendors have heavily adopted web components — See Web Almanac)
- Some contrast made with React components which work differently generally but React also works with Web Component’s to some degree (more later)

What are web components?

Web Components = 3 main things… not just one thing

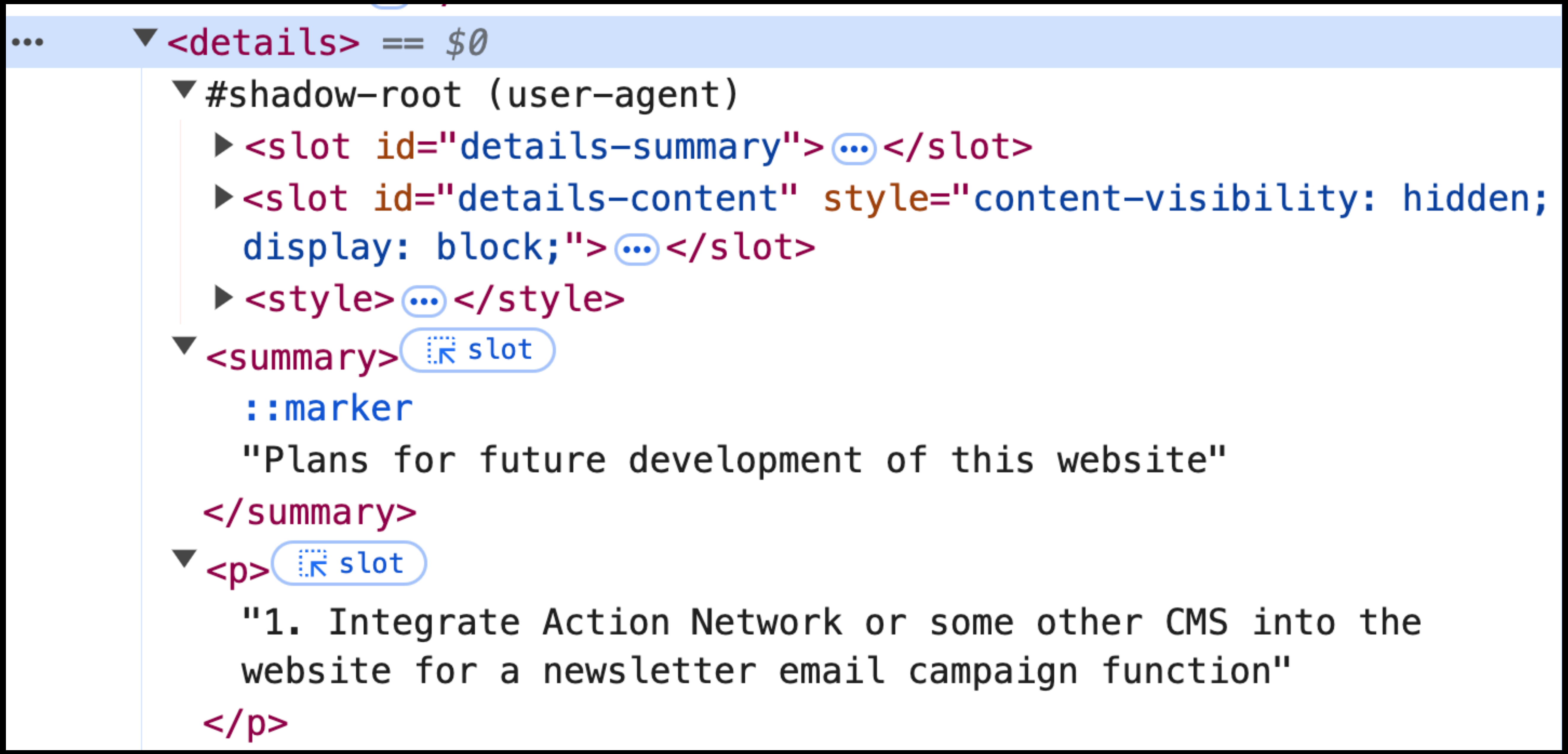
1. **Custom Elements**
    - <my-element></my-element>. Defined in JavaScript by extending the HTMLElement Class. Within some constraints name it what you want!
2. **HTML Templates**
    - <template></template> —Snippets of HTML hidden by default for reuse. (Won’t discuss much, not always necessary, but v. Useful)
3. **Shadow Document Object Model**
    - Lots to potentially talk about— Shadow DOM allows for encapsulation.

Built-in vs. Custom Elements

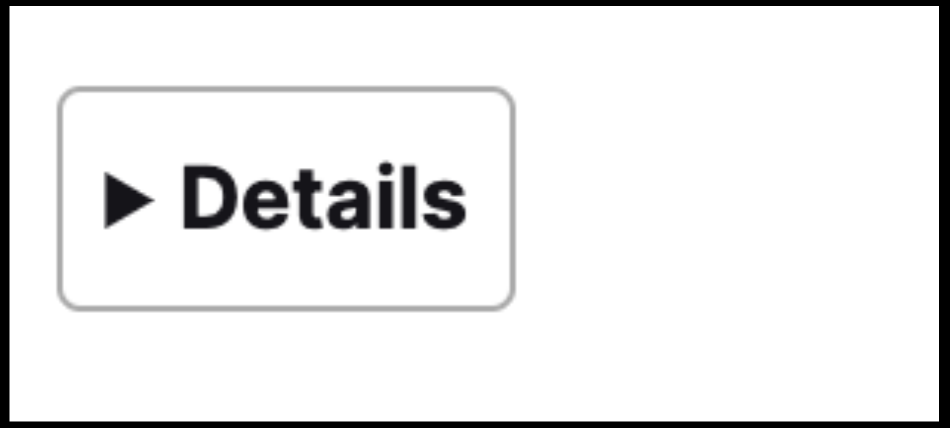
- Built-ins are the elements we ‘know and love’ built into the browser—many of you will recognize some elements.
- Built-ins usually come packaged with ‘user agent styles’ but they can be restyled almost completely. Built-ins have some semantic meaning that can be important for things like accessibility out of the box.
    - <p></p> tags are intended for paragraphs.
    - <button></button> is intended for a button.
- There are around 150ish built-ins.

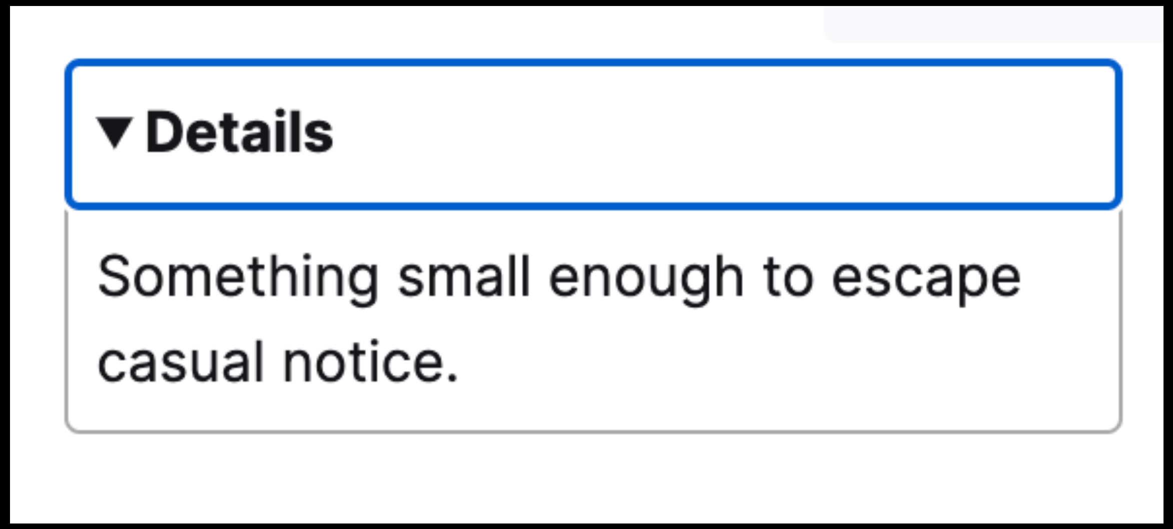
Built-in Elements w/ Shadow Document Object Model

- Fun Note/preview: Some Built-in elements already in browsers have features similar to web components and use the Shadow Document Object Model.
- For example, the <details><summary></summary></details> element combo has shadow root “out of the box” by default i.e (user-agent) shadow-root.



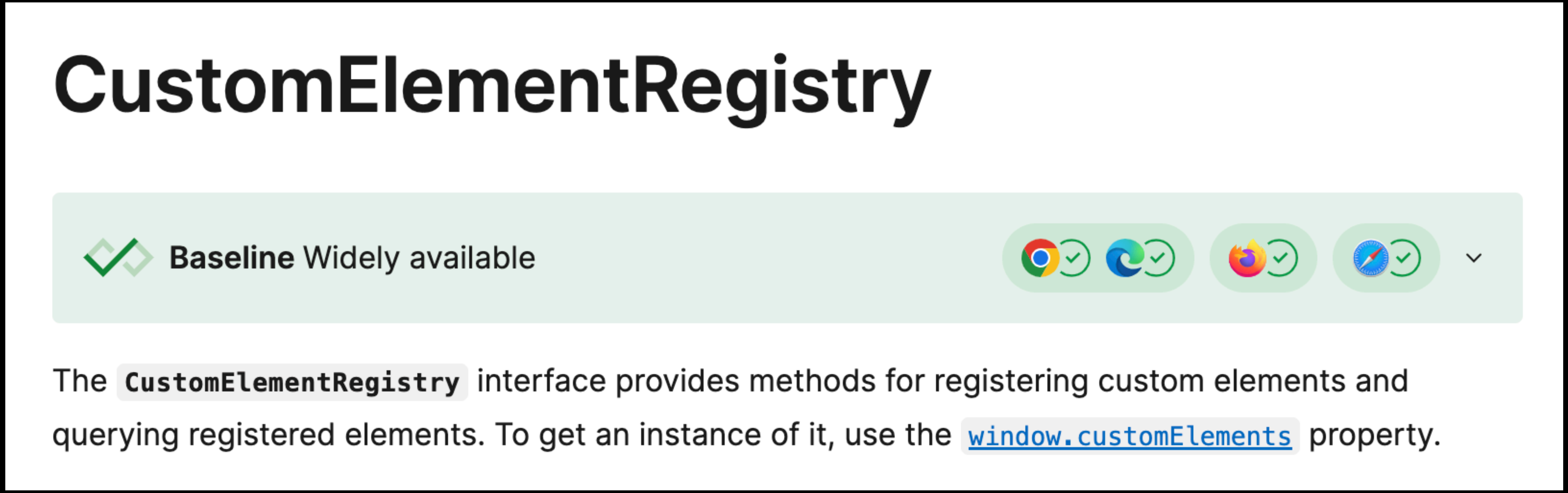
```
••• ▼<details> == $0
      ▼#shadow-root (user-agent)
        ▶<slot id="details-summary">…</slot>
        ▶<slot id="details-content" style="content-visibility: hidden;
         display: block;">…</slot>
        ▶<style>…</style>
      ▼<summary> slot
         ::marker
         "Plans for future development of this website"
       </summary>
      ▼<p> slot
         "1. Integrate Action Network or some other CMS into the
         website for a newsletter email campaign function"
       </p>
```





Built-in and Custom Elements

Both fully fledged HTMLElements!

- Custom elements are what we define and register with JavaScript into the browser, in a process defined by the HTML Spec. As we will see, they are full featured HTMLElements.
- 

CustomElementRegistry

Baseline Widely available

The `CustomElementRegistry` interface provides methods for registering custom elements and querying registered elements. To get an instance of it, use the `window.customElements` property.

- Naming convention, a letter or a word and a dash, i.e `a-` or `b-c` or `my-element`.
  https://html.spec.whatwg.org/#valid-custom-element-name

A **valid custom element name** is a sequence of characters *name* that meets all of the following requirements:

- *name* must match the `PotentialCustomElementName` production:

```
PotentialCustomElementName ::=
    [a-z] (PCENChar)* '-' (PCENChar)*

PCENChar ::=
    "-" | "." | [0-9] | "_" | [a-z] | #xB7 | [#xC0-#xD6] | [#xD8-#xF6] | [#xF8-#x37D] | [#x37F-#x1FFF] | [#x200C-#x200D] |
    [#x203F-#x2040] | [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-
    #xEFFFF]
```

  This uses the EBNF notation from the *XML* specification. [XML]

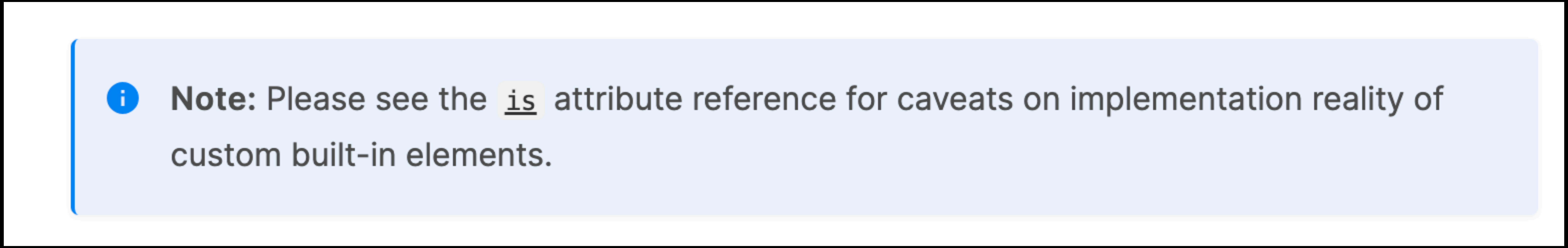
- *name* must not be any of the following:
  - `annotation-xml`
  - `color-profile`
  - `font-face`
  - `font-face-src`
  - `font-face-uri`
  - `font-face-format`
  - `font-face-name`
  - `missing-glyph`

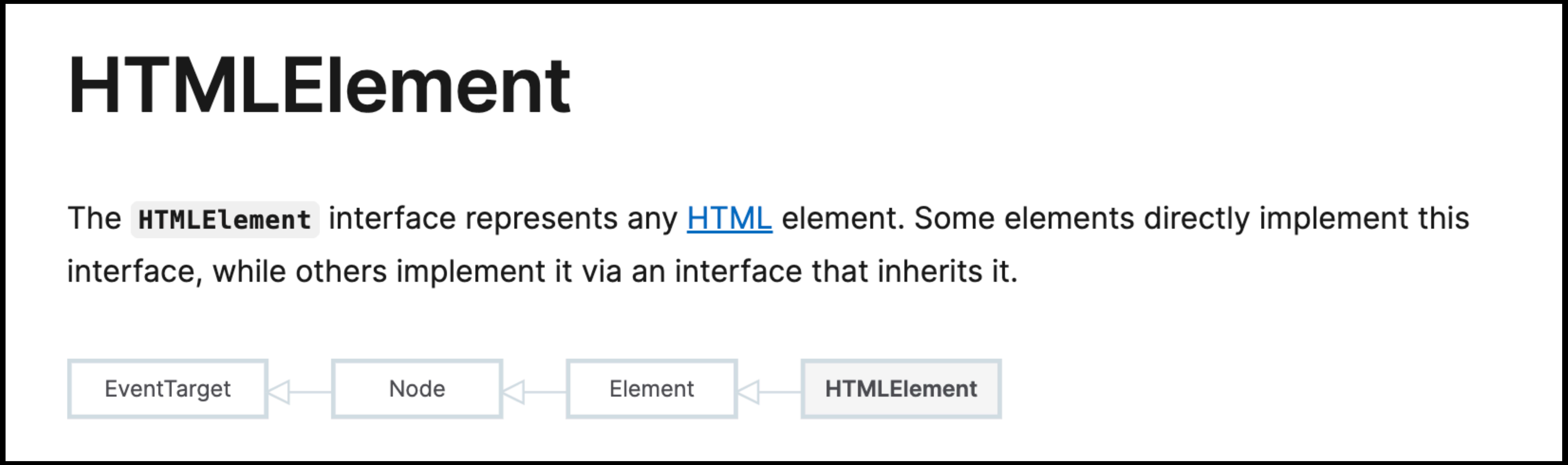
Custom Elements: Features and constraints

- Follow HTML Rules to be treated like a full HTMLelement !
  - Example. <ul> element should not contain non <li> children elements, if making a list item web component needs to be inserted within a `li` tag to comply
  - Wrong:
    - <ul><list-item-component></list-item-component></ul>
  - Right:
    - <ul><li><list-item-component></list-item-component></li></ul>
- Cant be void elements i.e Cannot do <my-custom-element /> only

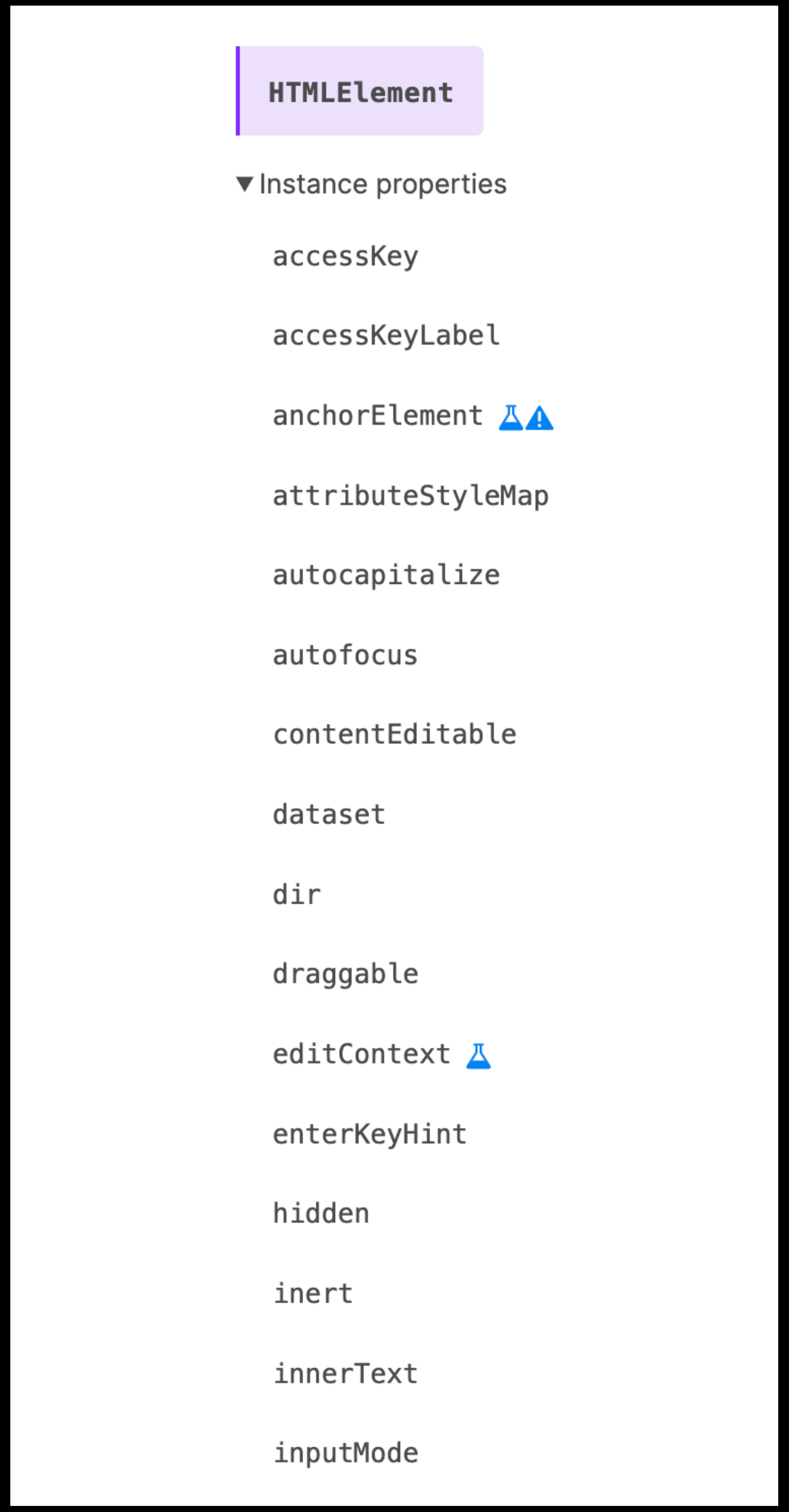
Important tip re “Custom Built-ins”

- Avoid “Custom Built-ins” if you want Safari support
- Example Custom Built in `<p is=“example-defined-element”></p>`
- MDN not completely clear on this point weirdly enough so important to stress, its arguably a dead end.


Note: Please see the `is` attribute reference for caveats on implementation reality of custom built-in elements.


HTMLElement

The `HTMLElement` interface represents any HTML element. Some elements directly implement this interface, while others implement it via an interface that inherits it.


HTMLElement

▼Instance properties

accessKey
accessKeyLabel
anchorElement
attributeStyleMap
autocapitalize
autofocus
contentEditable
dataset
dir
draggable
editContext
enterKeyHint
hidden
inert
innerText
inputMode

```
class extends HTMLElement {
```

These elements you are about to see are full HTML Elements

Prepare yourself.

<b-c></b-c>
https://bc-web-component.netlify.app/

&&

<apple-pay-button></apple-pay-button>

DEMO TIME

Features of Apple Pay Button Web Component

- Features of Apple Pay - Demo https://applepaydemo.apple.com/
- Display Apple Pay Button using JavaScript https://developer.apple.com/documentation/apple_pay_on_the_web/displaying_apple_pay_buttons_using_javascript
- See Apple’s Human Interface Guidelines: https://developer.apple.com/design/human-interface-guidelines/apple-pay#Using-Apple-Pay-buttons

Observations

- Simplicity of installation, ubiquity of use, practical need and want for some people. Ctrl+C, Ctrl+V, Get closer to being paid.
- Small footprint. Not very big at all.
- Apple exerts control over behavior and markup. Vitally important to brand integrity.
- Guard Rails (‘visible but protected’). Limited ability to change style by developer and a great number of pre-defined attribute options from the SDK. Shadow DOM used to set important styles hidden from rest of page, complex SVG markup — too much to copy and paste and trust…
- Guard Barriers: Buttons hidden and in disabled states if not enabled correctly with SDK.
- Script tag doing work for localization, security tokens and licensing. Doing some heavy lifting!

Questions /
Interactive code session

RE what Frameworks work better or worse with web components
https://custom-elements-everywhere.com/

Thank you!

Will share slides and links on Slack eventually