LINKÖPING UNIVERSITY

729G87 Interaction Programming

Lecture 5 – Web Components

Philipp Hock, PhD
philipp.hock@liu.se

Web Components

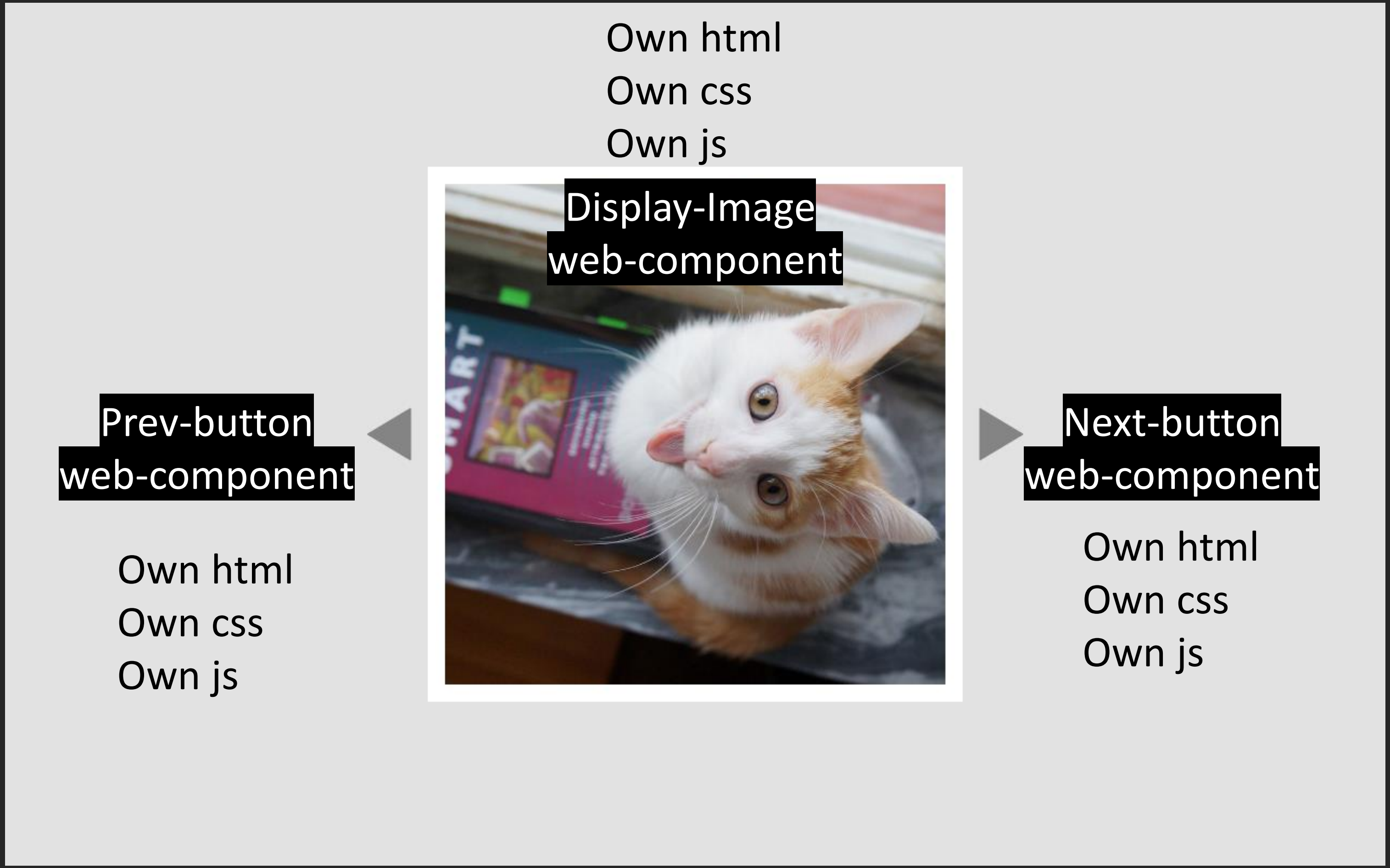
- Similar to modern UI Libraries
  - React
  - Svelte
  - Vue
  - Angular
- Use of the Custom Elements API
- Encapsulate HTML, CSS and JavaScript in a custom elements that can be used in your HTML code

Component-based arachitecture

1. **Reusability**. They are designed to plug into a variety of applications without the need for modification or special accommodations.
2. **Extensibility**. A component can be combined with others to create new behaviors.
3. **Replaceability**. Components with similar functionality can be swapped.
4. **Encapsulation**. Components are self-contained and expose functionality through interfaces while hiding the details of internal processes.
5. **Independence**. Components have minimal dependencies on other components and can operate in different environments and contexts.

Component-based arachitecture

- Modular Design: Systems are divided into reusable, self-contained components.
- Reusability: Components can be used in different projects or parts of a project.
- Interactions: Components communicate through well-defined interfaces.
- Isolation: Components encapsulate their own logic, reducing dependencies.
- Scalability: Independent development allows easy system expansion.
- Maintenance: Updates to one component have minimal impact on others.


Own html
Own css
Own js
Display-Image web-component
Prev-button web-component
Own html
Own css
Own js
Next-button web-component
Own html
Own css
Own js

Building blocks

- Custom elements:
  - API for defining new elements that can be used in HTML
- Shadow DOM: A separate DOM with its own styles (CSS) that we can attach to a custom element - CSS from "normal" page does not affect the shadow DOM!
- HTML templates: Special element type
  - not rendered in the browser
  - can be cloned and used as a template
- Using templates and slots
  - reusable HTML structure using <template> and <slot> elements

Custom HTML Element

- A Web Component is a new HTML-Tag
  - Actually an (not really) infinite number of new Tags
- Object oriented approach
  - Inherits from HTMLElement, HTMLParagraphElement,…
- Uses lifecycle methods
  - connectedCallback: called when the element is appended to a document
  - disconnectedCallback: called when the element is removed from the document
  - attributeChangedCallback: called when specified attributes change
  - …

Recap Classes

```
const square = new Rectangle(10, 10);
console.log(`width: ${square.width}`); // 10
console.log(`area: ${square.area}`); // 100
console.log(`initial color: ${square.color}`);
square.color = "magenta";
console.log(`color magenta?: ${square.color}`); l
square.color = "red";
console.log(`color red?: ${square.color}`);
```

```
class Rectangle {

  constructor(height, width) {
    this.height = height;
    this.width = width;
    this._color = null;
  }
  // getter for _color property
  get color() {
    return this._color;
  }
  // setter for color property
  set color(value) {
      this._color = value;
  }

  // getter for area property
  get area() {
    // NOTE: use this.methodname() to call a method
    // defined in the class
    return this.calcArea();
  }

  // method for calculating the area
  calcArea() {
    return this.height * this.width;
  }
}
```

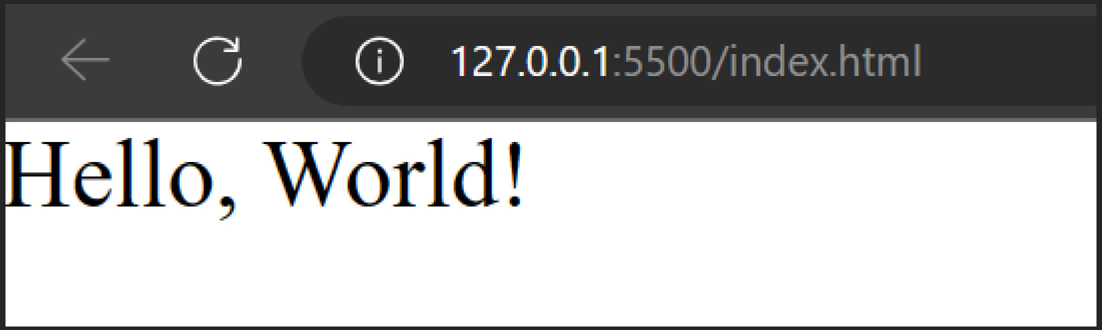
```
class BillyShelf {
  constructor(width, height) {
    this.width = width;
    this.height = height;
    this.elements = [];
  }

  addElement(element) {
    this.elements.push(element);
  }

  displayInfo() {
    console.log(`Billy Shelf - Width: ${this.width}, Height: ${this.height}`);
    console.log("Elements:", this.elements.join(", "));
  }
}
```

```javascript
// Creating instances of BillyShelf
const billy1 = new BillyShelf(80, 200); // Width: 80, Height: 200
billy1.addElement("Books");
billy1.addElement("Decor");

const billy2 = new BillyShelf(60, 180); // Width: 60, Height: 180
billy2.addElement("Candles");
billy2.addElement("Plants");

// Displaying information about the Billy shelves
billy1.displayInfo();
console.log("\n");
billy2.displayInfo();
```

```
// Subclass inheriting from BillyShelf
class Bookshelf extends BillyShelf {
  constructor(width, height, numShelves) {
    // Call the constructor of the superclass using super()
    super(width, height);
    this.numShelves = numShelves;
  }

  displayBookshelfInfo() {
    console.log(`Number of Shelves: ${this.numShelves}`);
  }
}

// Creating instances
const smallBookshelf = new Bookshelf(60, 150, 3);
const largeBookshelf = new Bookshelf(80, 200, 5);

// Adding elements to bookshelves
smallBookshelf.addElement("Books");
largeBookshelf.addElement("Novels");
smallBookshelf.displayInfo();
smallBookshelf.displayBookshelfInfo();
```

Simple Web Component

```
<script defer>
    class MyElement extends HTMLElement {
        constructor() {
            super();
            this.attachShadow({ mode: 'open' });
            // Create a text node with "Hello, World!"
            const textNode = document.createTextNode('Hello, World!');
            // Append the text node to the shadow DOM
            this.shadowRoot.appendChild(textNode);
        }
        connectedCallback() {
            // this is where you add event listeners to elements in the shadow DOM
        }
    }
    customElements.define("my-element", MyElement);

</script>

<body>
    <my-element></my-element>
</body>
```



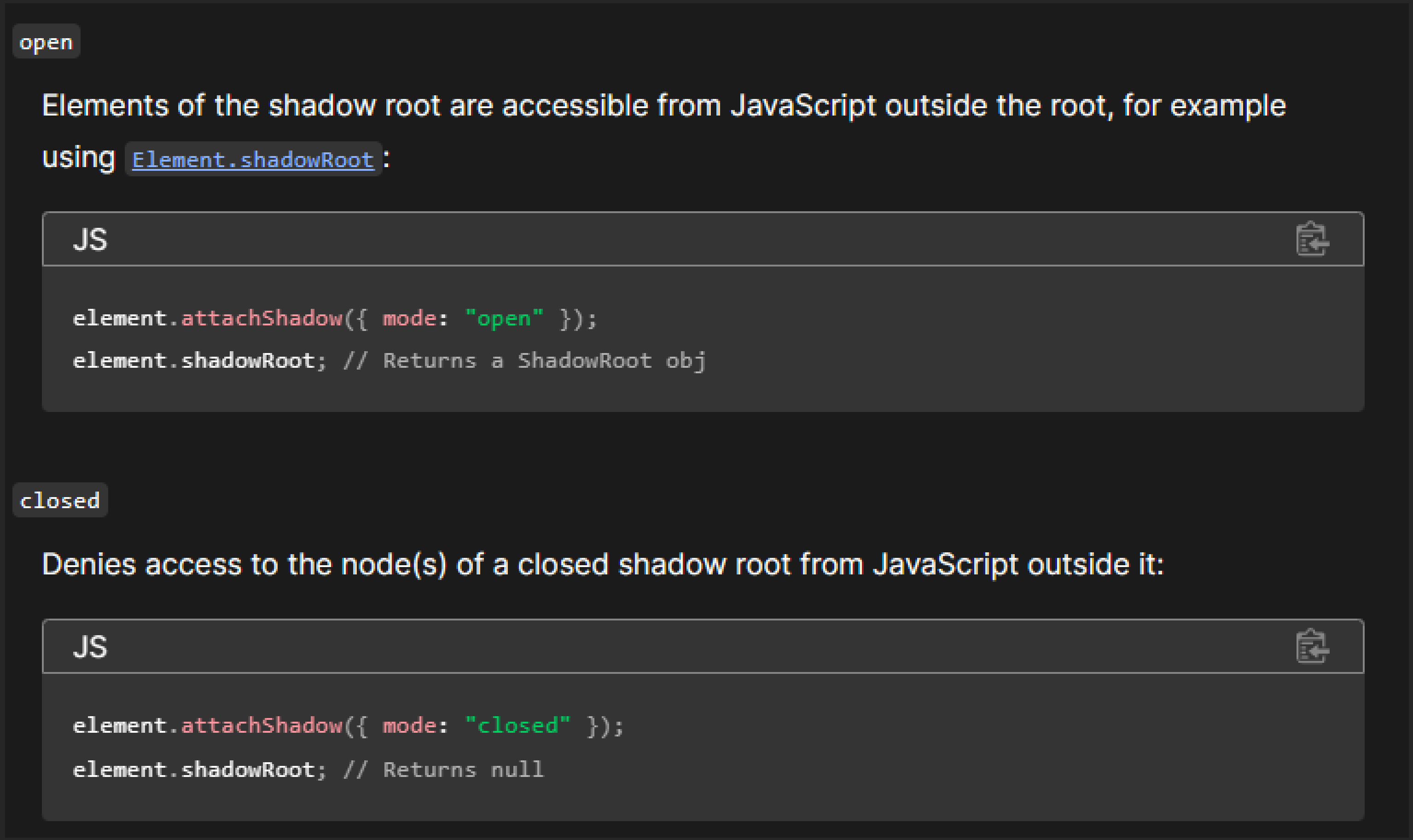
127.0.0.1:5500/index.html

Hello, World!

https://codepen.io/Philipp-Hock/pen/GRPRMrW

attachShadow

https://developer.mozilla.org/en-US/docs/Web/API/Element/attachShadow



`open`

Elements of the shadow root are accessible from JavaScript outside the root, for example using `Element.shadowRoot`:

JS

```js
element.attachShadow({ mode: "open" });
element.shadowRoot; // Returns a ShadowRoot obj
```

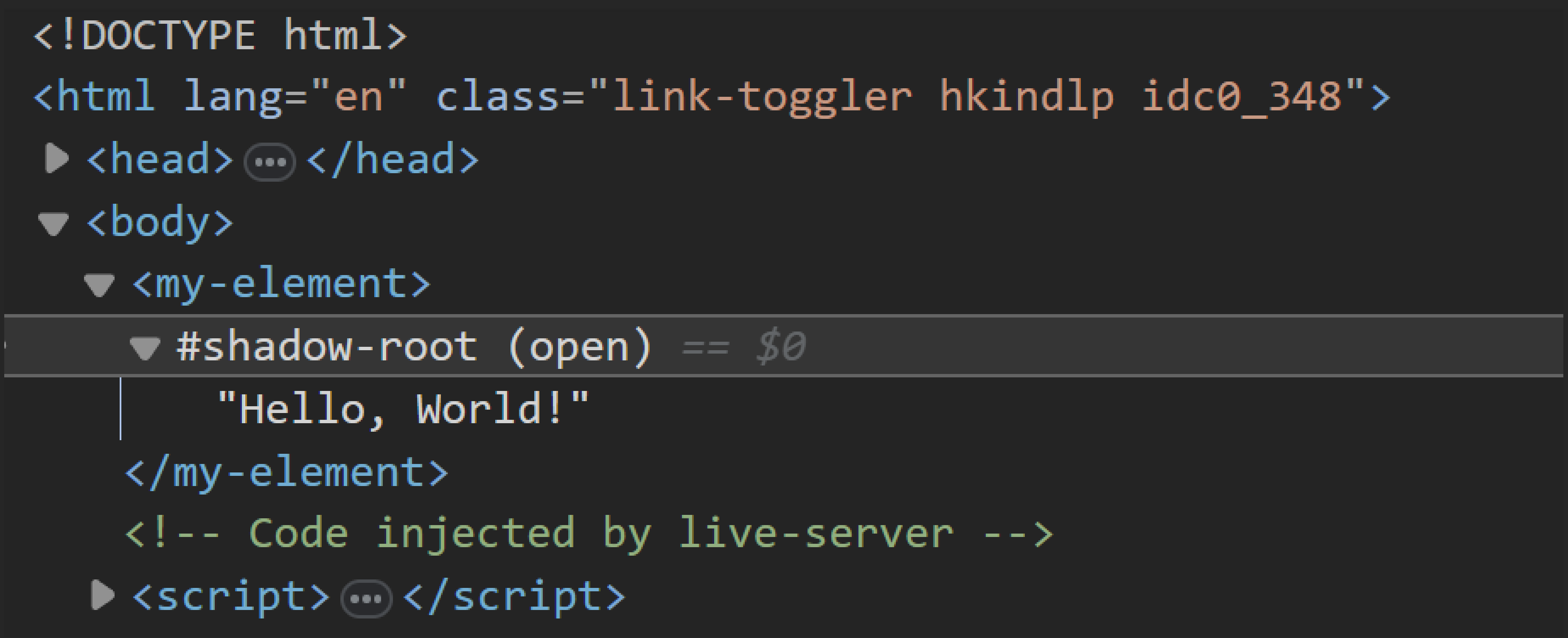
`closed`

Denies access to the node(s) of a closed shadow root from JavaScript outside it:

JS

```js
element.attachShadow({ mode: "closed" });
element.shadowRoot; // Returns null
```

Simple Web Component



```
<!DOCTYPE html>
<html lang="en" class="link-toggler hkindlp idc0_348">
 ▶ <head> ••• </head>
 ▼ <body>
   ▼ <my-element>
     ▼ #shadow-root (open) == $0
         "Hello, World!"
     </my-element>
     <!-- Code injected by live-server -->
   ▶ <script> ••• </script>
```

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        p {
            background-color: red;
        }
    </style>
</head>


<script type="module">
   ...
</script>


<body>
    <my-element></my-element>
    <p>not in component</p>
</body>

</html>
```

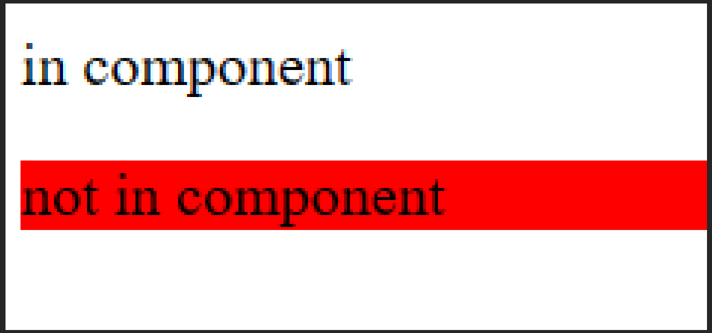
```
<script type="module">
    class MyElement extends HTMLElement {
        constructor() {
            super();
            this.attachShadow({ mode: 'open' });
            const content = `
                <p>in component</p>
            `;
            this.shadowRoot.innerHTML = content;
        }
    }
    customElements.define("my-element", MyElement);
</script>
```

```
<body>
    <my-element></my-element>
    <p>not in component</p>
</body>
```

```
<script type="module">
    class MyElement extends HTMLElement {
        constructor() {
            super();
            this.attachShadow({ mode: 'open' });
            const content = /*html*/`
                <p>in component</p>
            `;
            this.shadowRoot.innerHTML = content;
        }
    }
    customElements.define("my-element", MyElement);
</script>
```

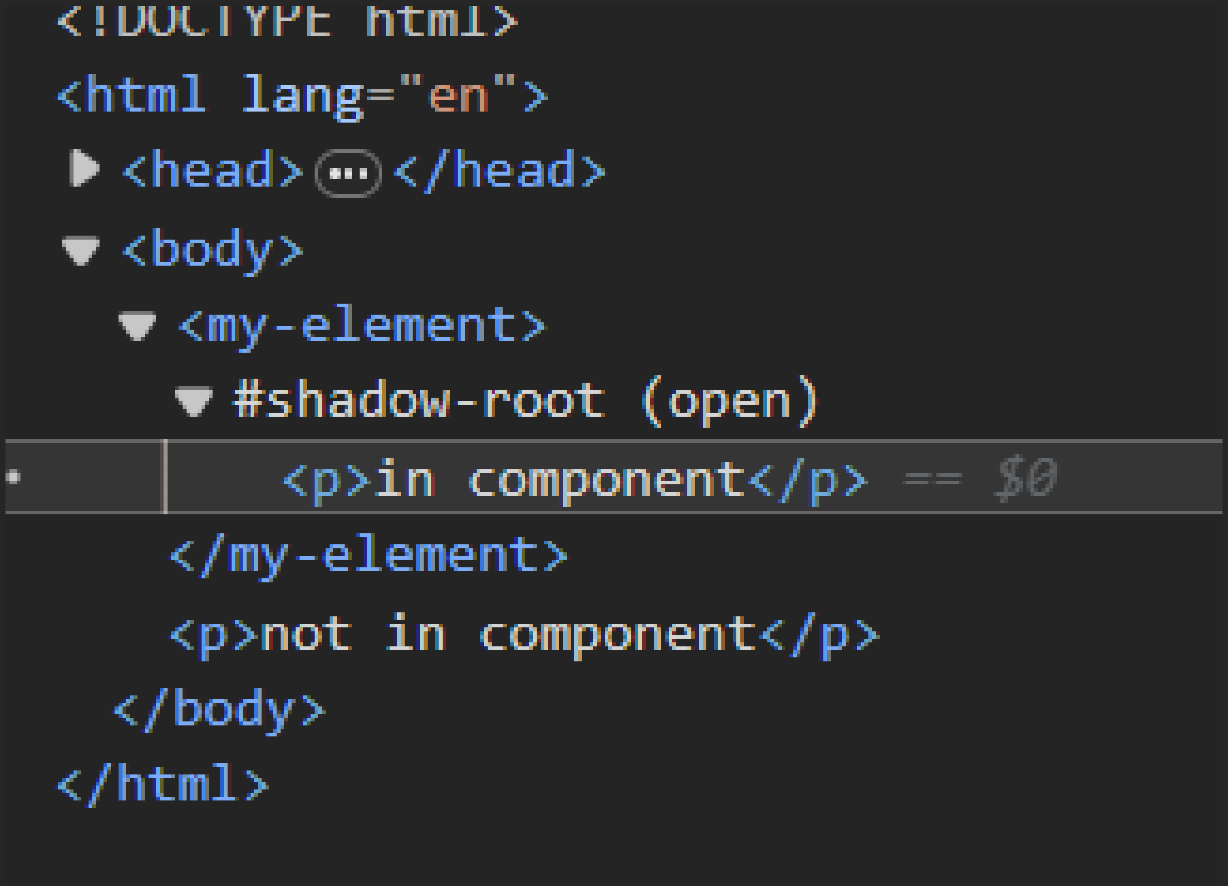
```
<body>
    <my-element></my-element>
    <p>not in component</p>
</body>
```

https://marketplace.visualstudio.com/items?itemName=Tobermory.es6-string-html



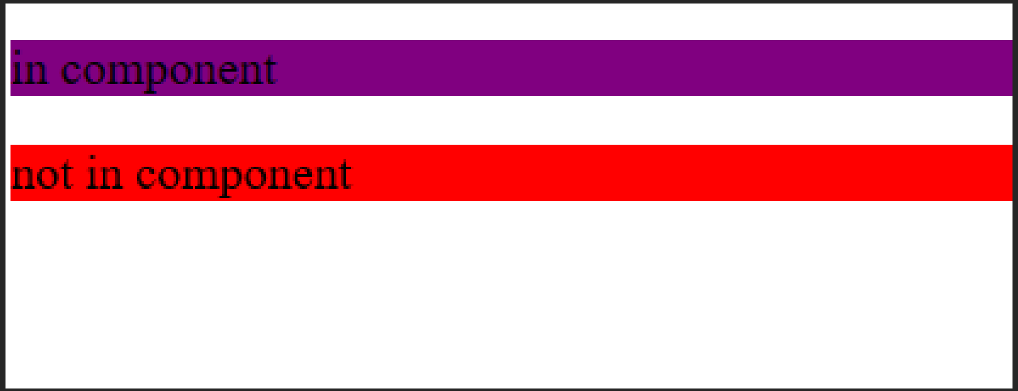
in component

not in component



```
<!DOCTYPE html>
<html lang="en">
 ▶ <head> ⋯ </head>
 ▼ <body>
   ▼ <my-element>
     ▼ #shadow-root (open)
         <p>in component</p> == $0
     </my-element>
     <p>not in component</p>
   </body>
</html>
```

```
<script type="module">
    class MyElement extends HTMLElement {
        constructor() {
            super();
            this.attachShadow({ mode: 'open' });
            const content = /*html*/`
                    <style>
                        p {
                            background-color: purple;
                        }
                    </style>
                    <p>in component</p>
                    `;
            this.shadowRoot.innerHTML = content;
        }
    }
    customElements.define("my-element", MyElement);
</script>

 <body>
     <my-element></my-element>
     <p>not in component</p>
 </body>
```

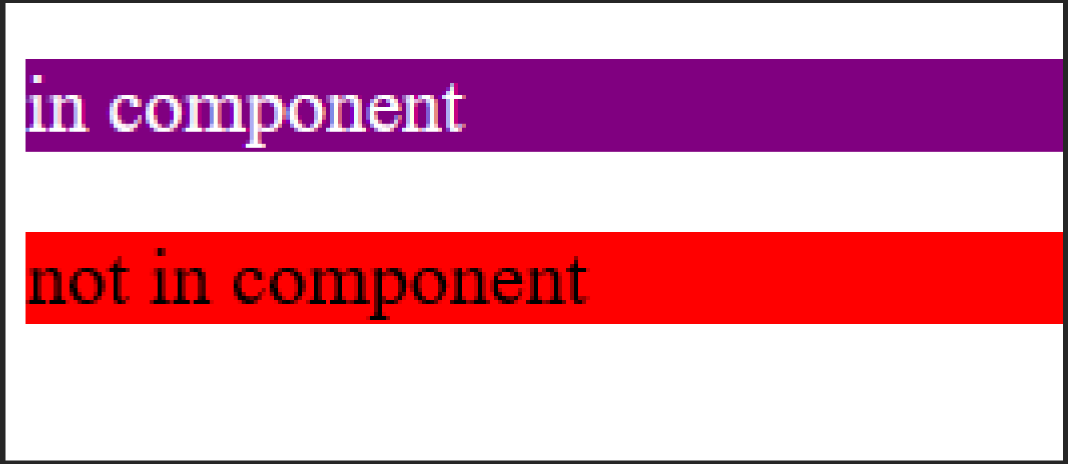


in component

not in component

Shadow DOM in action

```
connectedCallback() {
            const ps = this.shadowRoot.querySelectorAll("p");
            ps.forEach(e => {
                e.style = "color: white"
            })
}
```


in component

not in component

https://codepen.io/Philipp-Hock/pen/mdadBmW

Web Component without shadow DOM

Don’t do this!

```
class MyElement extends HTMLElement {
      constructor() {
        super();
        this.innerHTML = /*html*/`
          <style>
            p {
              color: red;
            }
          </style>
          <p>This paragraph is in the custom element.</p>
        `
      }
    }

customElements.define('my-element', MyElement);
```

Web Component without shadow DOM

Don't do this!

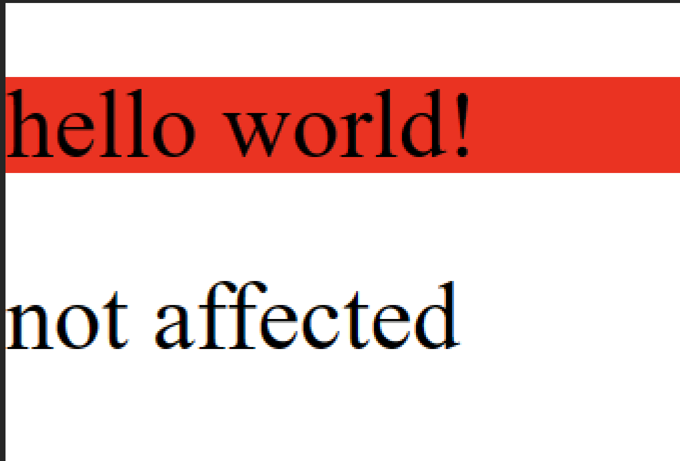
```
class MyElement extends HTMLElement {
        constructor() {
            super();
            this.attachShadow({ mode: 'open' });
            const paragraph = document.createElement('p');
            paragraph.textContent = "Not affected";
            $\boxed{document.body.appendChild(paragraph);}$
        }


}
    customElements.define("my-element", MyElement);
```

<template>

```
<body>
    <template id="myTemplate">
        <style>
          p {
            background-color: red;
          }
        </style>
        <p>hello world!</p>
      </template>

    <my-element></my-element>
    <p>not affected</p>
</body>
```

<template>

```
class MyElement extends HTMLElement {
    constructor() {
        super();
        this.attachShadow({ mode: 'open' });
        const template = document.querySelector('#myTemplate');
        const templateContent = template.content.cloneNode(true);
        this.shadowRoot.appendChild(templateContent);

    }
}
customElements.define("my-element", MyElement);
```



```
hello world!

not affected
```

placing templates directly in index.html can disperse code and hinder component self-containment

Cats!

https://codepen.io/Philipp-Hock/pen/xxmxXLx

Cats!

```
<template id="catTemplate">
        <link rel="stylesheet" href="css/catStyle.css">
        <div class="card">
        <div class="image">
          <img>
        </div>
        <div class="description">
          <p></p>
        </div>
      </div>
</template>

<cat-elem data-image="https://cataas.com/cat/cute?width=200&height=200"
          data-name="Chilli">
</cat-elem>

<cat-elem data-image="https://cataas.com/cat/ugly?width=200&height=200"
          data-name="Mex">
</cat-elem>
```

Cats!

```
class Cat extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({mode: 'open'});
    const template = document.querySelector("#catTemplate")
    this.shadowRoot.appendChild(template.content.cloneNode(true));
  }

  connectedCallback() {

    let cardImage = this.shadowRoot.querySelector(".image img");
    cardImage.setAttribute('src', this.dataset.image);
    let description = this.shadowRoot.querySelector(".description p");
    description.textContent = this.dataset.name;

  }

}

window.customElements.define('cat-elem', Cat);
```

Styling the web-component element itself

Inside web-component style

```
:host{
  display: inline-block;
}
```

Cats with slots?

Cats with slots?

```html
<cat-elem>
    <img slot="image" src="https://cataas.com/cat/cute?width=200&height=200">
    <span slot="name">Mex</span>
</cat-elem>

<cat-elem>
    <img slot="image" src="https://cataas.com/cat/ugly?width=200&height=200">
    <span slot="name">Chilly</span>
</cat-elem>
```

Cats with slots?

```html
<template id="catTemplate">
    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/catStyle.css">

    <div class="card">
        <div class="image">
          <slot name="image">IMAGE GOES HERE</slot>
        </div>
        <div class="description">
          <p><slot name="name">NAME GOES HERE</slot></p>
        </div>
      </div>
</template>
```

https://codepen.io/Philipp-Hock/pen/WNLNZZy

Cats with slots?

```
class Cat extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({mode: 'open'});
    const template = document.querySelector("#catTemplate")
    this.shadowRoot.appendChild(template.content.cloneNode(true));
  }


}

window.customElements.define('cat-elem', Cat);
```

Styling slots

```
<div class="card">
  <div class="image">
    <slot name="image">IMAGE GOES HERE</slot>
  </div>
  <div class="description">
    <p><slot name="name">NAME GOES HERE</slot></p>
  </div>
</div>
```

```
::slotted(img) {
  height: 100%;
  display: block;

}
```

Refer to slot parent CSS using:
slot[name=‘image‘]

Styling slots: workaround

- Don’t style slots!

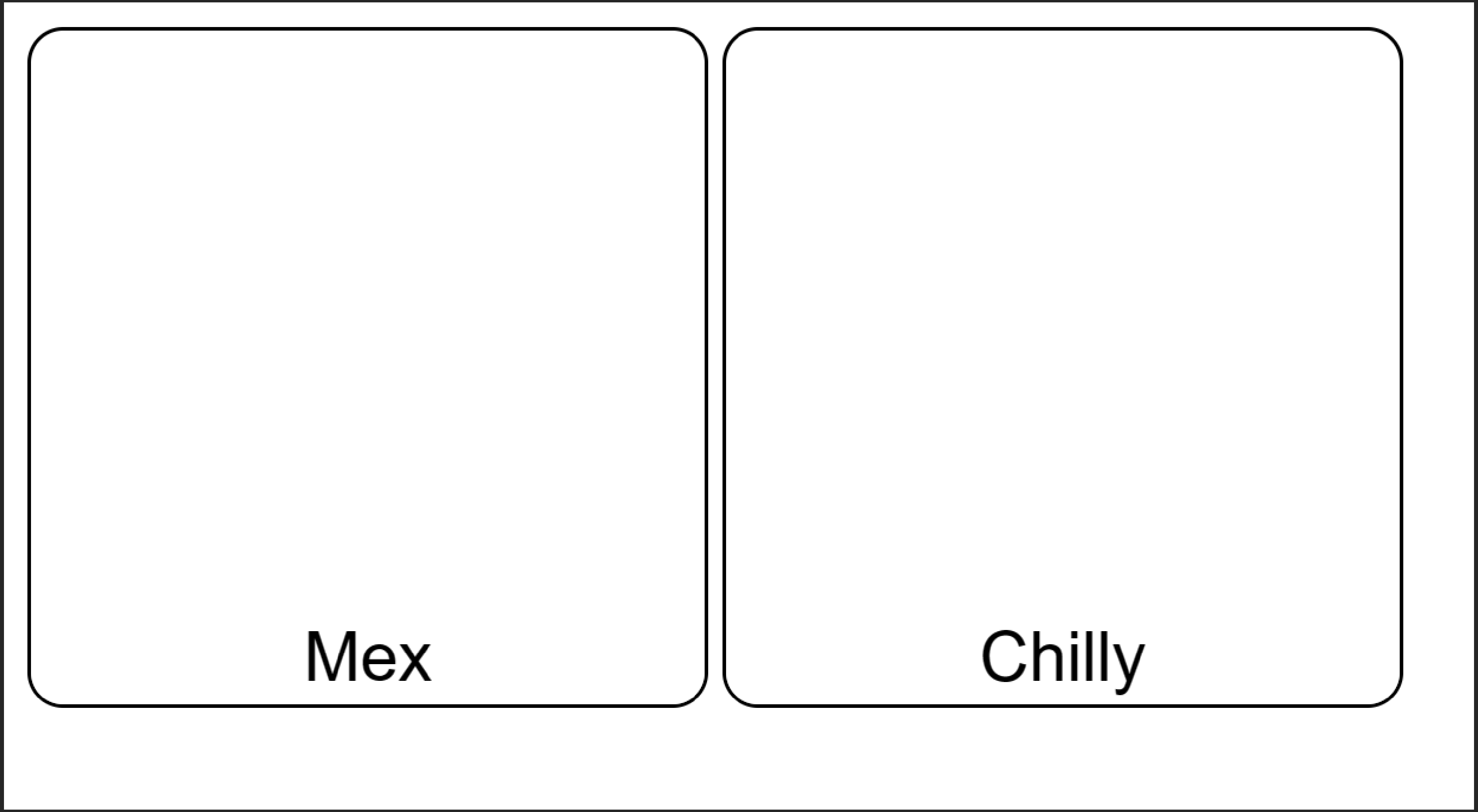
```
<template>
<p class="aclass">
  <slot name="name-of-slot">TEXT MISSING</slot>
<p>
</template>
```

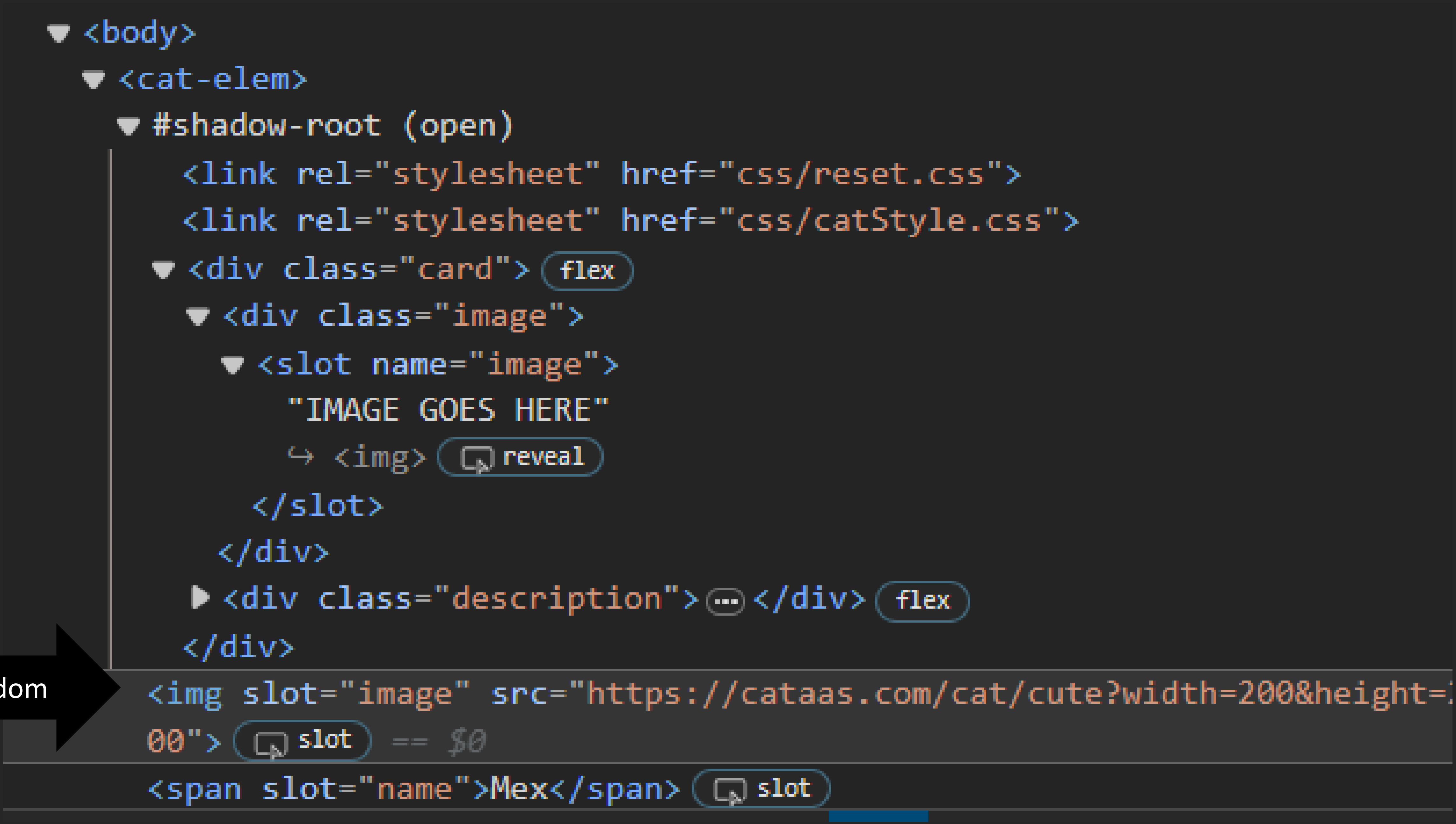
```
p { text-align: center; }
```

Problem with slots

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
<!--    <link rel="stylesheet" href="css/reset.css"> -->
    <link rel="stylesheet" href="css/layout.css">
    <link rel="stylesheet" href="css/style.css">

    <style>
        img{
            display: none;
        }
    </style>
</head>
```

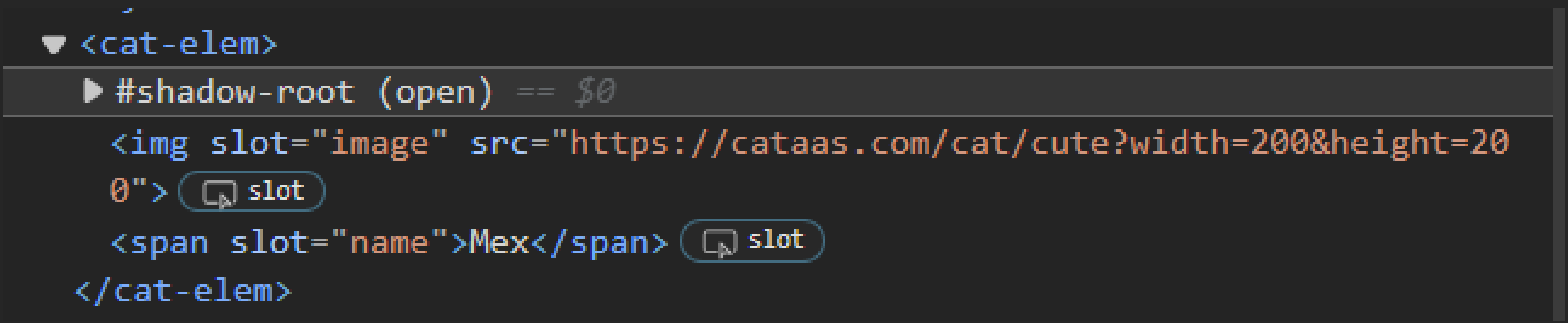
Problem with slots





```
▼ <body>
  ▼ <cat-elem>
    ▼ #shadow-root (open)
        <link rel="stylesheet" href="css/reset.css">
        <link rel="stylesheet" href="css/catStyle.css">
      ▼ <div class="card"> flex
        ▼ <div class="image">
          ▼ <slot name="image">
              "IMAGE GOES HERE"
              ↪ <img> reveal
            </slot>
          </div>
        ▶ <div class="description">…</div> flex
        </div>
      <img slot="image" src="https://cataas.com/cat/cute?width=200&height=
      00"> slot == $0
      <span slot="name">Mex</span> slot
```

adow dom



```
▼ <cat-elem>
    ▶ #shadow-root (open) == $0
      <img slot="image" src="https://cataas.com/cat/cute?width=200&height=20
      0"> slot
      <span slot="name">Mex</span> slot
  </cat-elem>
```

Styling slots

- Slotted elements still exist in the main DOM so they are also affected by CSS in the main DOM.

- Best practice: Use slots if you are styling the elements using CSS from the main DOM
  - Breaks idea behind component

Solution: Moving elements in shadow dom

```
<cat-elem>
    <img src="https://cataas.com/cat/cute?width=200&height=200">
    <p>Mex</p>
</cat-elem>

<cat-elem>
    <img src="https://cataas.com/cat/ugly?width=200&height=200">
    <p>Chilly</p>
</cat-elem>
```

https://codepen.io/Philipp-Hock/pen/poqoWpQ

Solution: Moving elements in shadow dom

```
<template id="catTemplate">
    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/catStyle.css">

    <div class="card">
        <div class="image">
        </div>
        <div class="description">
        </div>
      </div>
</template>
```

Solution: Moving elements in shadow dom

```
connectedCallback() {

  // this will refer to the custom element
  this.shadowRoot.querySelector(".image").appendChild(
    this.querySelector("img"));

  this.shadowRoot.querySelector(".description").appendChild(
    this.querySelector("p"));
}
```

Custom Events

```
connectedCallback() {
    this.meow();
}

meow(){
  const event = new Event("meow");
  this.dispatchEvent(event);
  setTimeout(()=>{this.meow()},1000);
  //more on https://developer.mozilla.org/en-US/docs/Web/API/Event/Event
}
```

https://codepen.io/Philipp-Hock/pen/qBLBPob

Custom Events

```
<script defer>

    const cat = document.querySelectorAll("body > cat-elem")[1];
    cat.addEventListener("meow", ()=>{
        console.log("cat has meowd");
    });

</script>
```

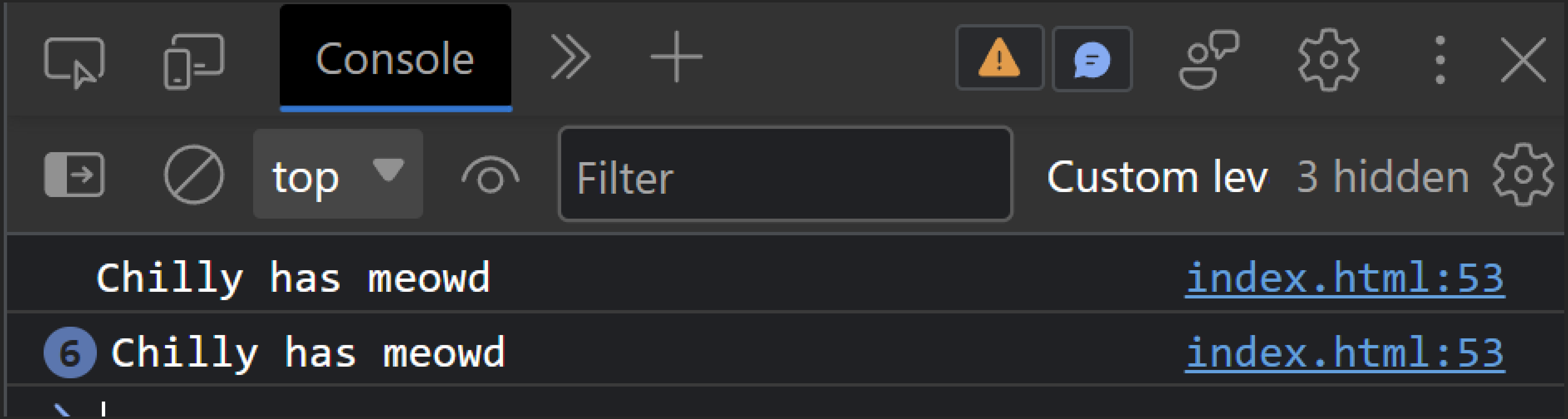
Custom Events

```
connectedCallback() {
    this.name = this.shadowRoot.querySelector(".description p").textContent;
    this.meow();
}

meow(){
  const event = new CustomEvent("meow",{detail: this.name });
  this.dispatchEvent(event);
  setTimeout(()=>{this.meow()},1000);
  //more on https://developer.mozilla.org/en-US/docs/Web/API/Event/Event
}
```

https://codepen.io/Philipp-Hock/pen/wvRvrmm

Custom Events

```
<script defer>

    const cat = document.querySelectorAll("body > cat-elem")[1];
    cat.addEventListener("meow", (e)=>{
        console.log(`${e.detail} has meowd`);
    });

</script>
```



```
  Chilly has meowd                                index.html:53
6 Chilly has meowd                                index.html:53
```

Web component interaction

- https://gitlab.liu.se/729g87/HT2023/webcomponents