External CSS and the Shadow DOM

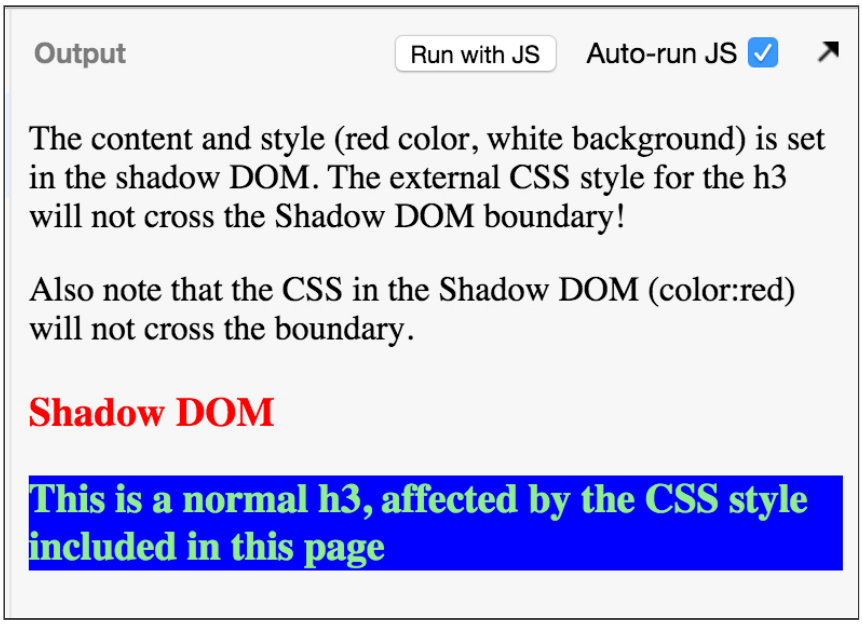
In a previous lesson we saw that: "*CSS styles defined inside Shadow DOM are scoped to the ShadowRoot. This means styles are encapsulated by default: they will not affect the elements outside...*"

However, it is possible to style the content of elements inside a shadow DOM, from an external CSS stylesheet. The CSS Scoping Module defines many options for styling content in a shadow tree!

The content of this page uses examples adapted from this excellent article from HTML5rocks: "Shadow DOM 201, CSS and Styling".

EXAMPLE 1: SEE HOW CSS STYLE ENCAPSULATION WORKS

Example at JSBin:



HTML code:

```
<head>
 ...
```

```
     <style>
       h3 {
         color:lightgreen;
         background-color:blue;
       }
     </style>
    </head>
10. <body>
     ...
     <div>
     <h3>Note, this is a shadow host</h3>
     </div>
     <h3>This is a normal h3, affected by the CSS style included
    in this page</h3>
    ...
```

JavaScript code:

```
    var root =document.querySelector('div').createShadowRoot();

    // We replace the content of the host with this shadow DOM
    content
    root.innerHTML = '<style>h3{ color: red; }</style>' +
     '<h3>Shadow DOM</h3>';
```

Explanations:

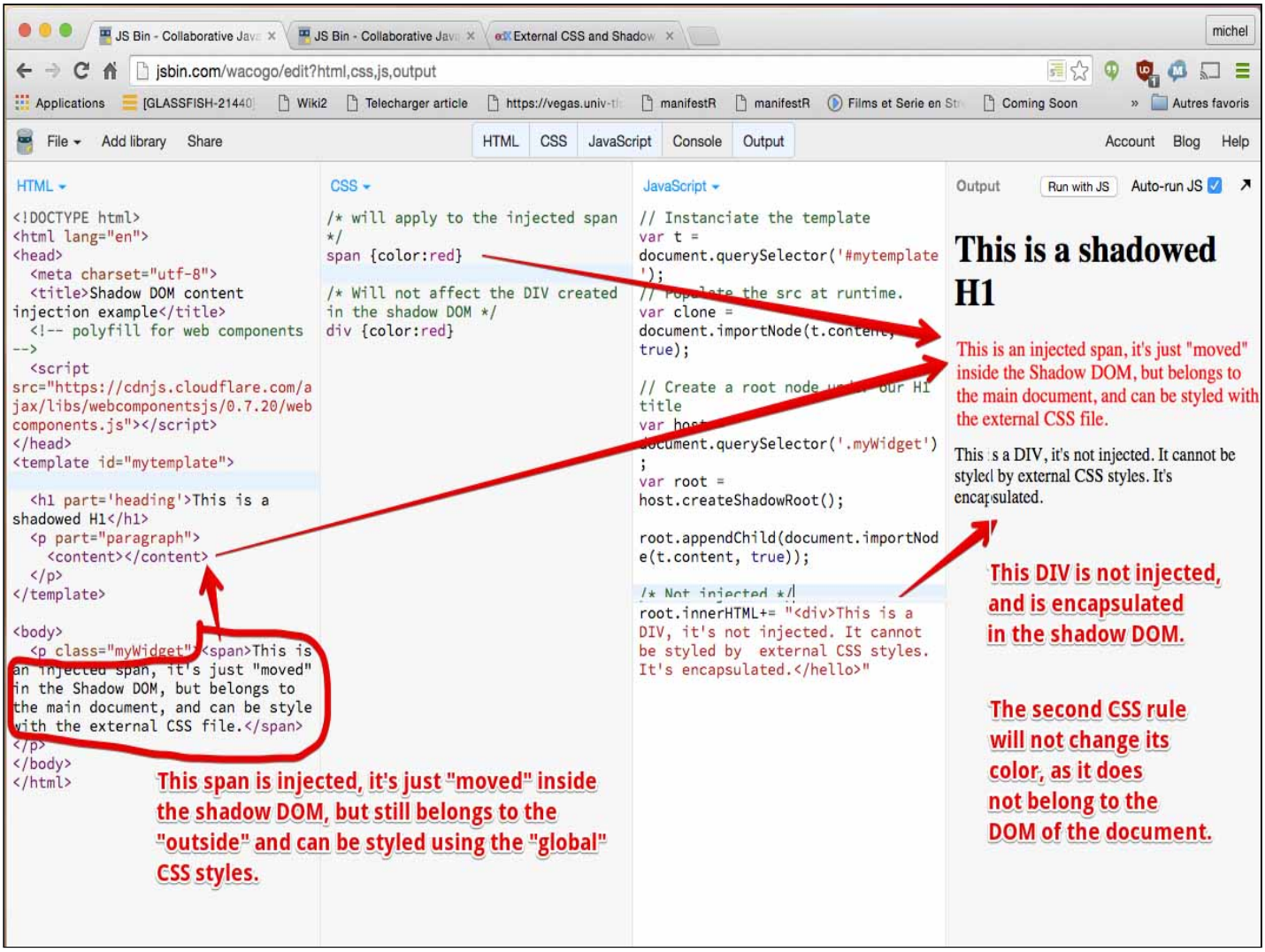
- We injected a style `color:red` into the CSS of the Shadow DOM (*line 4* of the JavaScript code). Only the H3 in the Shadow DOM became red, even with a "global" rule that says that all H3s should be light green with a blue background (*lines 4-8* of the HTML code). Again, styles are scoped by default.
- Other style rules defined on the HTML page that target H3s don't affect the elements in the Shadow DOM. This is because**selectors don't cross the shadow boundary**.

We have style encapsulation from the outside world. Thanks Shadow

DOM!

EXAMPLE 2: INJECTED CONTENT BELONGS TO THE STANDARD DOM AND CAN BE STYLED BY EXTERNAL CSS STYLES

Example at JSBin:



HTML source code:

```
...
<template id="mytemplate">
   <h1 part='heading'>This is a shadowed H1</h1>
   <p part="paragraph">
```

```
      <content></content>
   </p>
</template>

9. <body>
 <p class="myWidget"><span>This is an injected span, it's
just "moved" inside the Shadow DOM, but belongs to the main
document, and can be styled with the external CSS file.
</span>
</p>
</body>
...
```

CSS source code:

```
/* will apply to the injected span */
span {
   color:red;
}

/* Will not affect the DIV created in the shadow DOM */
div {
   color:red;
}
```

JavaScript code:

```
// Instantiate the template
var t =document.querySelector('#mytemplate');
// Populate the src at runtime.
var clone = document.importNode(t.content,true);

// Create a root node under our host
var host =document.querySelector('.myWidget');
var root = host.createShadowRoot();
```

```
    // Add cloned template code
11. root.appendChild(clone);

    /* Add a div, this one is really in the DOM, it's not
    injected */
    root.innerHTML+= "<div>This is a DIV, it's not injected. It
    cannot be styled by external CSS styles. It's encapsulated.
    </hello>"
```

Explanations:

- In the HTML, the content between `<p class="myWidget">` and`</p>` is injected in the template (HTML code, *line 10,* and in the template *line 5*), then the template is cloned and added to the body of the document (JavaScript code, *lines 3, 4 and 11*). This content still belongs to the main HTML page, where it has been defined. Global styles apply on this content: the `span { color:red; }` will change the color of the injected `<span>`.
- In The JavaScript code, at *line 14*, we add new elements in the shadow DOM. These elements are created directly in the shadow DOM and are encapsulated. External CSS styles will not apply! The`div { color:red; }` will not change its color!

THE `:HOST` SELECTOR TO STYLE THE HOST ELEMENT FROM THE SHADOW ROOT

Use the `:host` selector to style the root element, however external styles have an higher priority

It is possible to use some CSS inside the shadow DOM for styling the shadow host, using the `:host` selector. This selector is only usable in a CSS rule inside the shadow DOM, you cannot use it in a regular CSS stylesheet. As it affects an element in the document (not in the shadow DOM), *its priority is lower than CSS rules from the document*.

Example at JBin:

This example uses the :host selector in the CSS of the shadow DOM. Notice that this selector has a lower priority than the external CSS styles (here, the background-color:red is overriden by the global style (that set it to blue). However the content is in uppercase because of the text-transform:uppercase; in the CSS from the shadow root.

THIS CONTENT IS INJECTED

This is a normal h3, affected by the CSS style included in this page

uppercase comes from a CSS rule in the shadow root

HTML code:

```
...
<style>
h3 {
  color:lightgreen;
  background-color:blue;
 }
 </style>
</head>
<body>
...
 <div>
   <h3>This content is injected</h3>
 </div>
...
```

JavaScript code:

```
var root =document.querySelector('div').createShadowRoot();

// We replace the content of the host by this shadow DOM content
root.innerHTML = root.innerHTML = '<style>' +
 ':host { text-transform: uppercase; background-color:red;}' +
 '</style>' +
 '<content></content>';
```

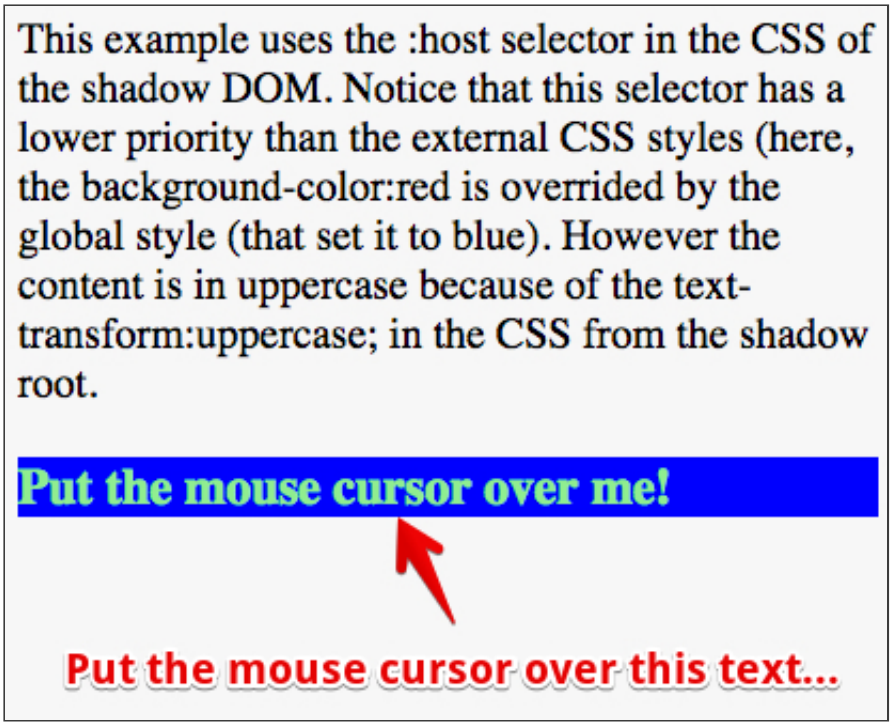
Explanations:

- The code at *line 5* contains a CSS rule that selects the shadow host content (the H3 content at *line 12* of the HTML code). This content is injected at *line 7* of the JS code.
- This rule says "make it uppercase", and indeed, as there is no conflict with another CSS rule, the text is rendered in uppercase.
- This rule also says "make the background color red!". This time, the external, global, stylesheet has a rule that is in conflict with this one (at *line 5* of the HTML code). The external CSS has higher priority, so the text background color will be blue.

One common use of the `:host` selector: reacting to mouse events

You can use the `:host(:hover)`, `:host(:active)`, `:host(:focus)` etc. selectors. Notice the use of parenthesis that are not necessary with regular CSS selectors.

Try this example at JSBin: move the mouse over the shadow host



We just replaced *line 5* in the JavaScript code of the previous example with this one:

```
root.innerHTML = root.innerHTML = '<style>' +
 ':host(:hover) { text-transform: uppercase; }' +
 '</style>' +
 '<content></content>';
```

THE :HOST-CONTEXT SELECTOR FOR STYLING HOSTS THAT ARE CHILDREN OF PARTICULAR ELEMENTS

The `:host-context(<selector>)` pseudo class matches the host element if it or any of its ancestors matches the `<selector>`.

A common use of `:host-context()` is for theming an element based on its surrounds. For example, many people do theming by applying a class to `<html>` or `<body>`:

```
<body class="different">
  <x-foo></x-foo>
</body>
```

You can use `:host-context(.different)` to style the host `<x-foo>` only when it's a descendant of an element with the class `.different`:

```
:host-context(.different) {
  color: red;
}
```

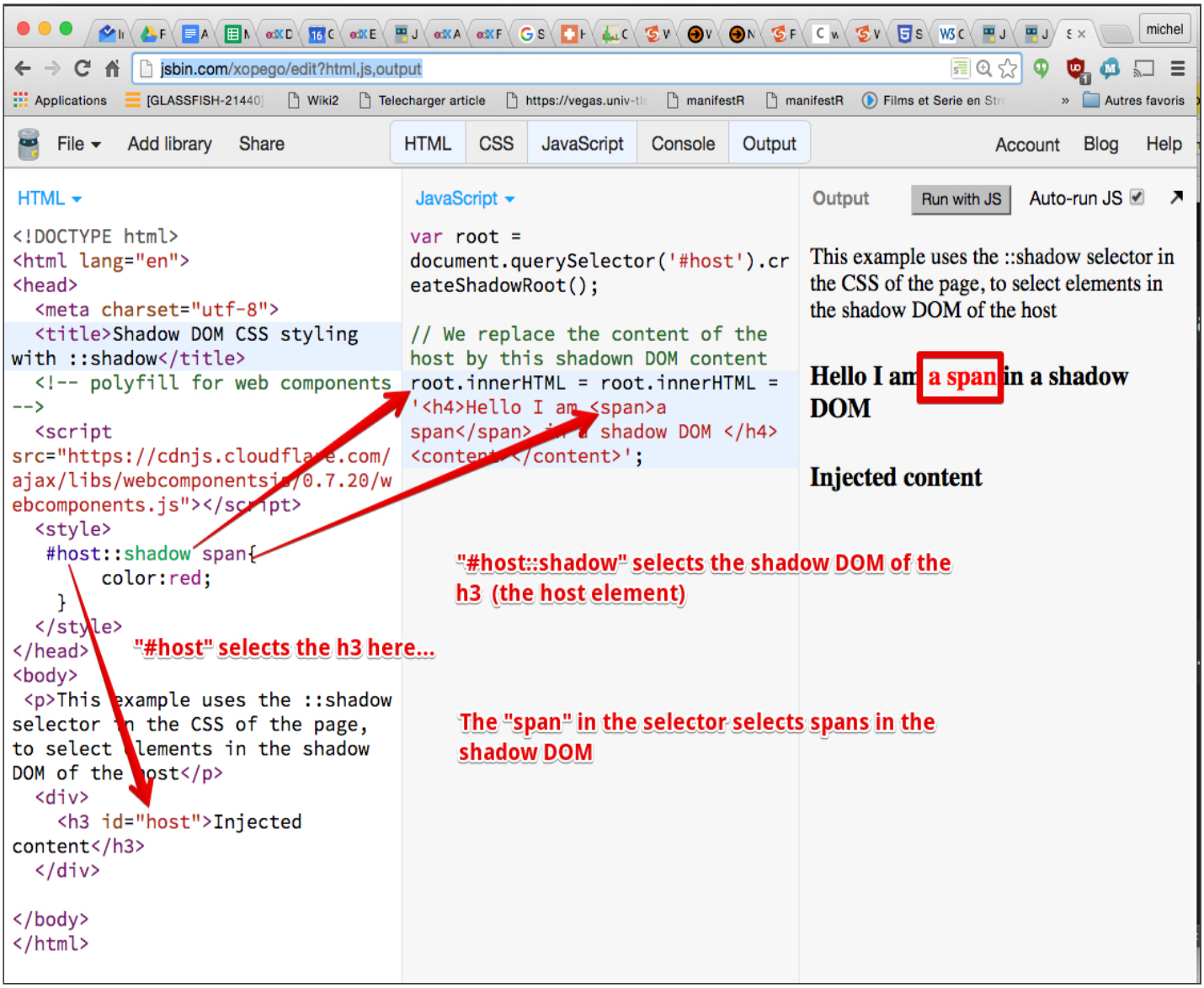
This enables you to encapsulate style rules in an element's Shadow DOM that uniquely style it, if its context matches certain constraints (here: have the CSS class "different").

THE `::SHADOW` PSEUDO ELEMENT: STYLING SHADOW DOM INTERNALS FROM THE OUTSIDE

The `::shadow` pseudo-element allows you to pierce through the Shadow DOM's boundary to style elements within shadow trees!

If an element has at least one Shadow DOM, the `::shadow` pseudo-element matches the shadow root itself. It allows you to write selectors that style elements in this Shadow DOM.

Example at JSBin:



HTML code:

```
<style>
  #host::shadow span{
    color:red;
  }
</style>
</head>
<body>
<div>
    <h3 id="host">Injected content</h3>
</div>
```

JavaScript code:

```
var root =document.querySelector('#host').createShadowRoot();

// We replace the content of the host with this shadow DOM content
root.innerHTML = root.innerHTML = '<h4>Hello I am <span>a span</span> in a shadow DOM </h4><content></content>';
```

Explanations:

- The selector in the HTML code first selected the element with`id="host"` -> the H3 in the page,
- Then `::shadow` selected its shadow DOM,
- Then `span` selected all spans in the shadow DOM of the element.

[ADVANCED] GOING FURTHER WITH CSS AND WEB COMPONENTS?

Do you want to go further? There are advanced topics such as using the `/deep` combinator or styling injected content from inside the Shadow DOM. We recommend reading this article from HTML5Rocks, to learn about these advanced features that are useful for developers of Web Component libraries, as opposed to those who wish to develop a single, independent, component.