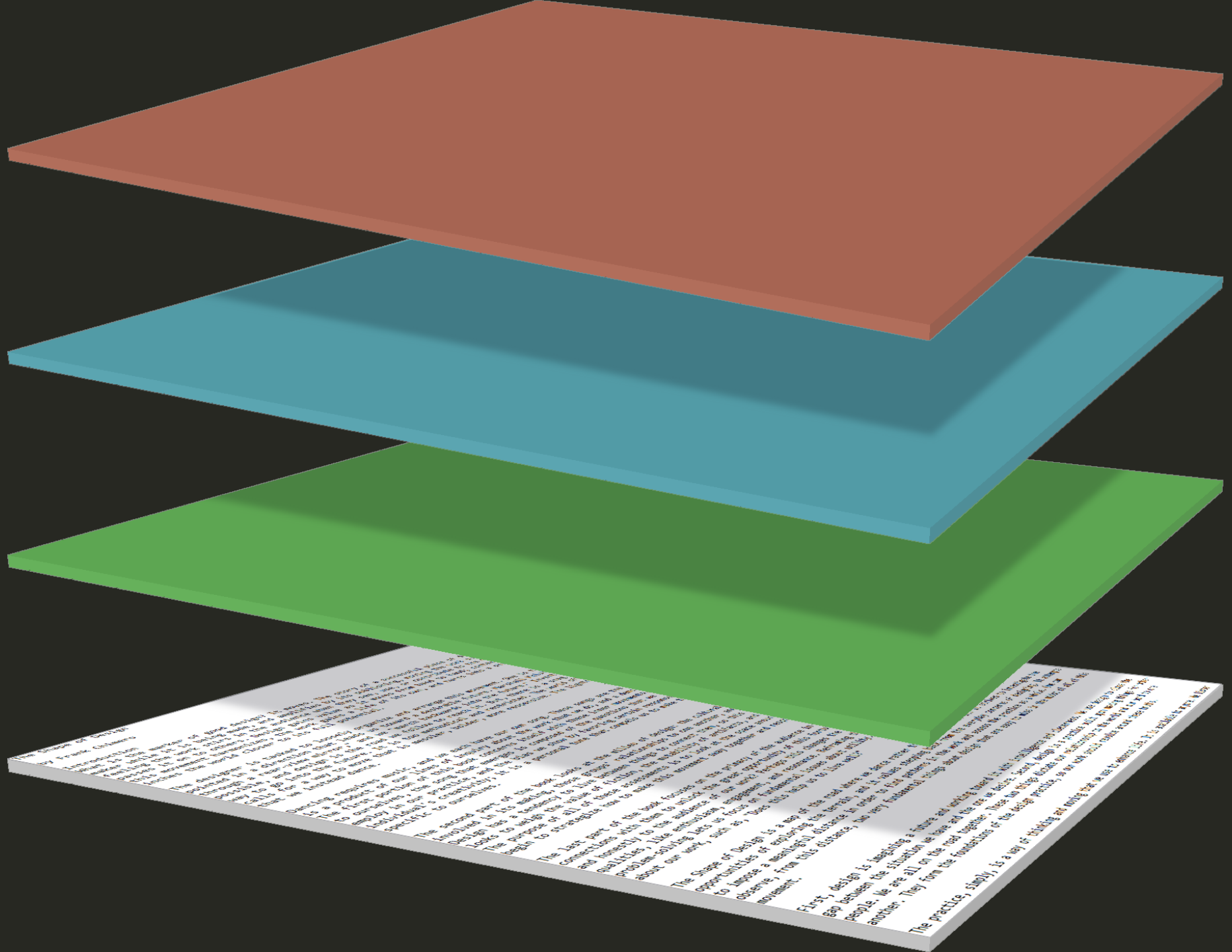
The behaviour layer
JavaScript: Part 2

Content Management



Behaviour

JavaScript

Presentation

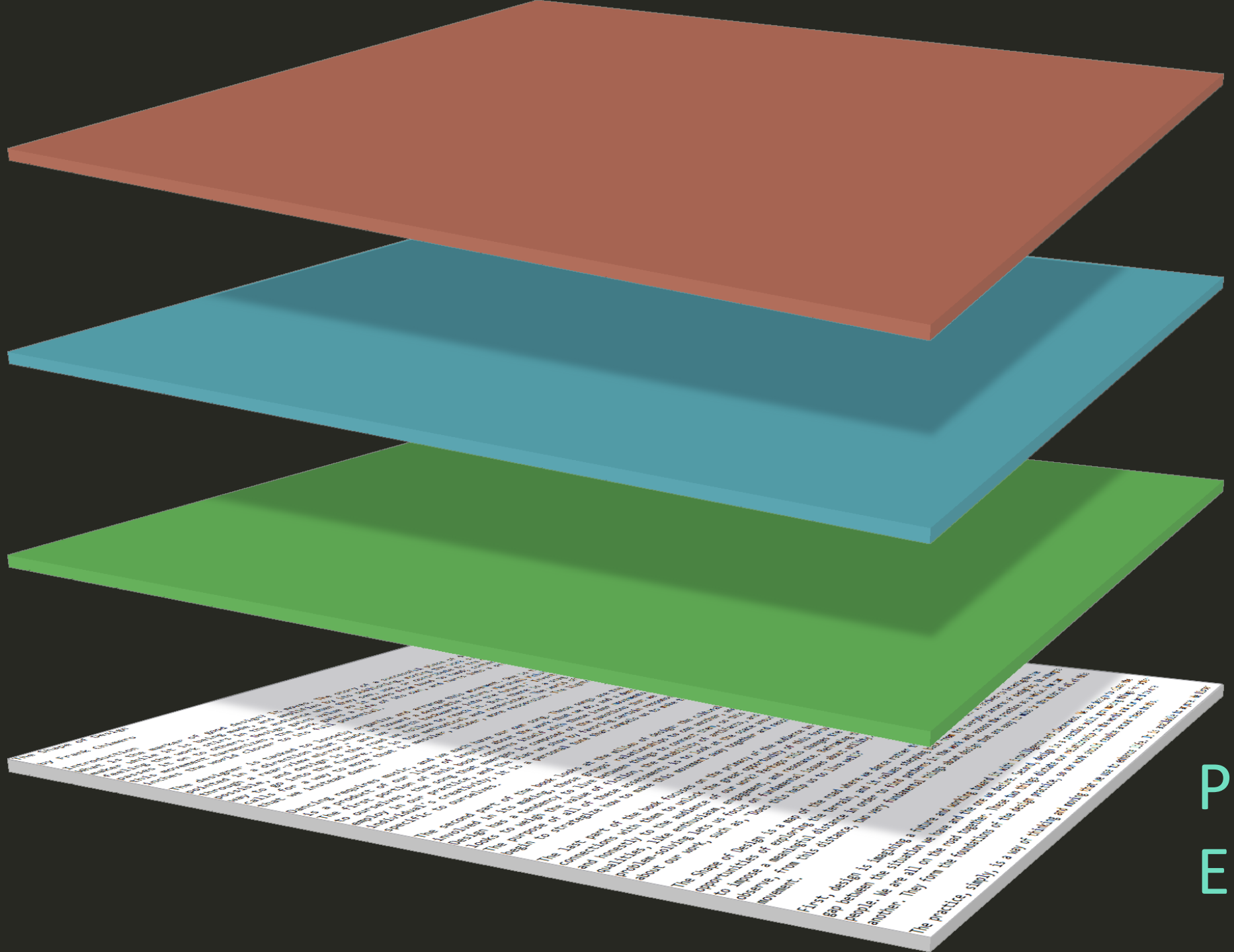
CSS

Structure

HTML

Content

Text, images etc.



Richness of experience

Progressive Enhancement

All three layers of the web standards model combined…

structure | presentation | behaviour | webpage in browser

The browser loads the HTML file and then any linked resources such as the CSS file, the JavaScript file and any media files (e.g. images). A single webpage is built from many separate files.

<noscript></noscript>

```
<noscript><p>Hello, JavaScript seems to be
unavailable. I'm afraid these two buttons
won't work without it.</p></noscript>
```

When JavaScript fails, we can help our users understand that our application won't work without it.

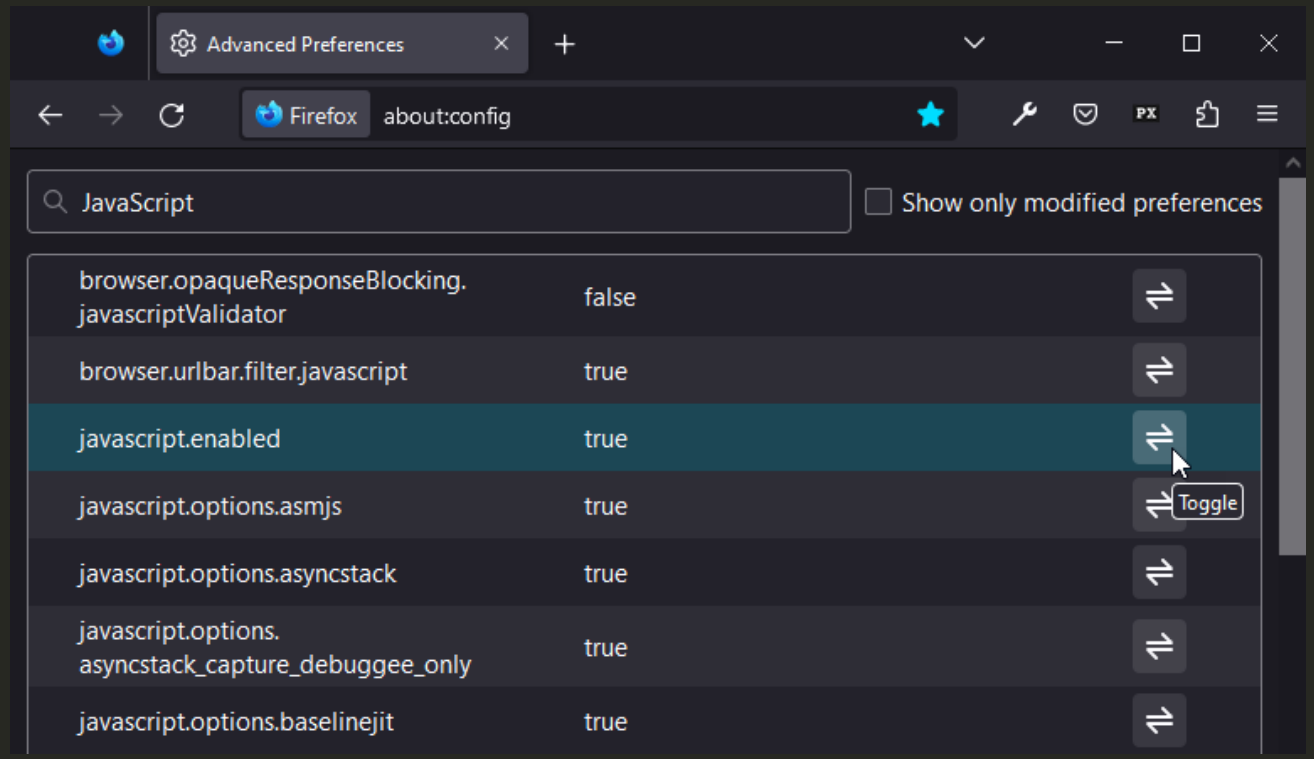
When JavaScript is unavailable, all content (including HTML) between the opening and closing *noscript* tags will be displayed. The content can be styles with CSS just like any other content.

Note: The *noscript* approach works well in many contexts but sometimes it would be better not to show the non-working buttons at all. In such a situation, we can create the buttons with JavaScript. That way, if JavaScript is not available, the user does not see any buttons. This is the approach taken for the scripts in this presentation.

Testing JavaScript fallbacks

about:config

Type this in the Firefox address bar and then click the button that says, “Accept the Risk and Continue”. Use the search function to search for “JavaScript”. In the resulting list, you’ll see a preference called “javascript.enabled”. By default, this is set to “true”. Click the double arrow on the right to toggle the value to “false”. Refresh your browser to see how the page looks without JavaScript. Click the arrow again to set the value back to “true”.



Controlling the flow of a script

All the scripts we've written so far will run automatically when the document is loaded in the browser. The trick to building successful user interactions is in being able to tell the browser when a certain part of a script should be run. To do this, we create a bit of script called a *function*.

```
function functionName(){
   do something when the function is called
}
```

Building user interactions

```
<script>
1. Create the markup (links, buttons etc.);
2. Add event listeners (check if a link is clicked);
3. Build the function to run (when the link is clicked);
</script>
```

Building user interactions is, essentially, a three-step process. The first step is to create the markup we need for the user controls. The second step is to add an event listener on the element used as a control. The third is to add a JavaScript function, that will run when the control is triggered.

Creating HTML elements with JavaScript

```
let ul = document.createElement('ul');
let parent = document.querySelector('main');
parent.appendChild(ul);
```

Adding an element to a document is a three-step process, represented by the three statements above:

1. Create the element you want and assign to a variable.
2. Select the parent element.
3. Add the new element to its parent.

createElement

1. This method allows us to create any HTML element. However, it isn't immediately added to the document (DOM tree), instead it is held in a variable, ready to be inserted where we want.

Select parent

2. You may use any convenient method to select the parent element. Here we're using querySelector but you could use getElementById, for example. The parent element is assigned to a variable.

appendChild

3. This method will add the specified new element as the *last* child of the selected parent element. If you need to add a new element as the *first* child, you may use the prepend method.

Adding HTML elements with JavaScript

```
let li = document.createElement('li');
ul.appendChild(li);
```

Adding an element to another element is also a simple, three-step process:

1. Create the element you want and assign to a variable.
2. Select the parent element if it isn't already assigned to a variable.
3. Add the new element to its parent.

createElement

1. This method allows us to create any HTML element. However, it isn't immediately added to the document (DOM tree), instead it is held in a variable, ready to be inserted where we want.

Select parent

2. You may use any convenient method to select the parent element. Here we're using querySelector but you could use getElementById, for example. The parent element is assigned to a variable.

appendChild

3. This method will add the specified new element as the *last* child of the selected parent element. If you need to add a new element as the *first* child, you may use the prepend method.

Enhancing HTML elements with JavaScript

```
let element = document.querySelector('li');
element.className = 'name';
element.textContent = 'Some text';
```

Any element in the current document can be selected and manipulated:

1. We can add a class.
2. We can add/replace text between opening and closing tags.

className

1. This property allows us to add a class to the specified element. In the example above, we're adding the class "name" to the first list item:

```
<li class="name"></li>
```

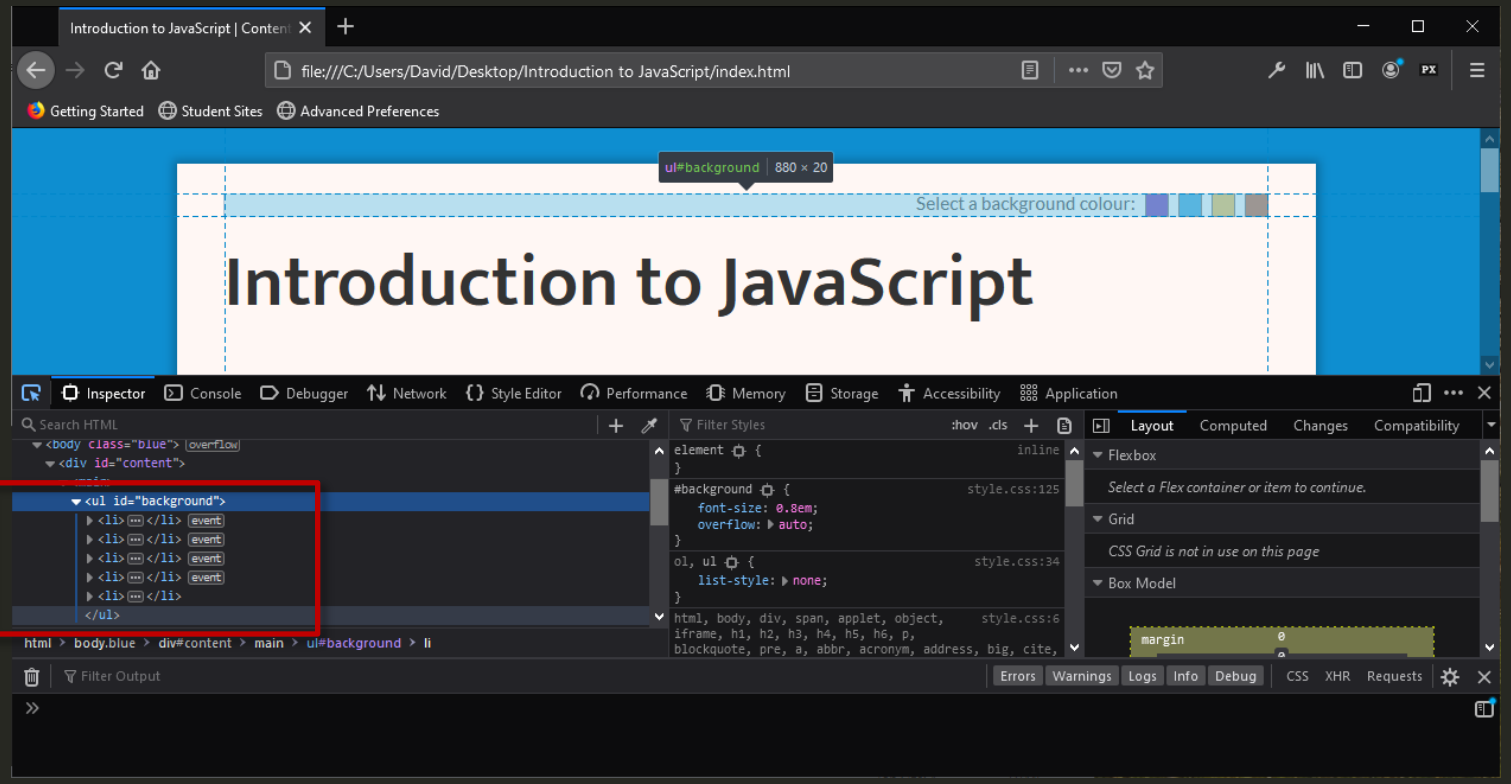
textContent

2. In the third statement, we are using the textContent property to add some text to the same element, and the result is:

```
<li class="name">Some text</li>
```

Bear in mind that this generated code is not visible in the document source, it can only be viewed in the inspector…

Viewing generated markup



The inspector panel in Firefox Developer can be used to check any markup generated using JavaScript – you won't see it by viewing source.

Adding an event listener

```
let element = document.querySelector('li');
element.addEventListener('click', function);
```

Adding an event listener is a simple two-step process:

1. Select the element to which you want to add an event listener (in this case the second list item) and assign it to a variable.
2. Use the addEventListener method and specify the type of event (in this case a click) and the name of the function to run when the event takes place.

addEventListener

This method is used to add an event listener (sometimes referred to as an event handler) to an element. In the wild, you may see alternative ways of doing this. For example, the following older version achieves a similar outcome:

element.onclick = function

Building a function

```
function addClass(){
      element.className = 'highlight';
}
```

function

A function is a self-contained script that is given a name. The function will run when its name is called. In the example above, class="highlight" is added to the element stored in the element variable when the function is run.

className

This method is used to add a class to an element. It works in the same way as the id property, used for adding an ID.

This is a very simple function that consists of a single statement, but functions can contain many statements and perform more complex tasks.

A mostly complete script

1

```
function addClass( ){
li.className = 'highlight';
} //function ends
```

2

```
let ul = document.createElement('ul');
let parent = document.querySelector('main');
parent.appendChild(ul);
let li = document.createElement('li');
ul.appendChild(li);
li.addEventListener('click', addClass);
```

The first part of this script (1) will only run when the function addClass is called. The rest of the script (2) will run as soon as the document is loaded in the browser.

The script creates an unordered list and assigns it to a variable called "ul". It then selects the main element, assigning it to a variable called "parent". Next, it adds the unordered list at the end of main using the appendChild method. It then creates a list item, assigning it to a variable called "li" and adds it to the unordered list. Finally, the event listener is added to the list item. Now, when the list item is clicked, the addClass function will run.

A summary of the three steps

```
<script>
1. Create the markup (links, buttons etc.);
2. Add event listeners (check if a link is clicked);
3. Build the function to run (when the link is clicked);
</script>
```

1 We use JavaScript to create elements that will be used for user interactions. The reason we do this is so that those elements do not appear to the user if JavaScript is not available. There's no point having a button on your page if it does nothing.

2 Once the elements have been created and added to the document, we add an event listener that specifies what sort of event to listen for and what function to run when that event occurs. The event could be a click or a hover.

3 Finally, we build a function that does something (usually to some other element) when the event occurs. The function can be placed anywhere in the script. It is read by the browser when the page loads, but it doesn't run until the associated event occurs.

Worked example

This example combines all the techniques we have explored over the two JavaScript sessions. Have a look at the HTML, the CSS and the JavaScript files to make sure you have a good understanding of DOM Scripting.

As always, we learn best by doing, so try modifying this example or build one of your own.

https://www.websitearchitecture.co.uk/resources/examples/user-interaction/

Remembering user choices with localStorage

```
localStorage.setItem('colour-choice', colour);
```

```
const colour = localStorage.getItem('colour-choice');
```

localStorage.setItem

The setItem method is used with the localStorage object to save data to the user's browser. It does this by specifying a name/value pair. In this case, we're calling our data "colour-choice" and assigning it the value contained in the colour variable.

localStorage.getItem

This method is used to get the value of an item that has been saved in localStorage. In this example, we are getting the value of the item called colour-choice and assigning it to a variable called colour.

Local storage is a better way to store user data than cookies because the data never leaves the browser and there are therefore no privacy issues.

Worked example

When you feel ready to move to the next level, work through the CSS Theme Switcher tutorial, which includes the use of localStorage.

If you can learn everything described in that tutorial, you are well on your way to developing a good understanding of JavaScript.

https://explainers.dev/css-theme-switcher/

JavaScript is fun!

JavaScript might seem daunting if you've never done any scripting before, but like all languages, it has a relatively simple and logical syntax. Once you learn this and get to know some of the common methods used in DOM Scripting, you'll find that making your web pages come to life can be quite rewarding and maybe even fun.

https://developer.mozilla.org/en-US/docs/Web/JavaScript

```
function endSlideshow(){
    el = document.getElementById("slideshow");
    el.style.display = "none";
    return true;
}
```