Class 02: User interaction

This week we will focus on getting used to working with vanilla JavaScript and develop our understanding of DOM Scripting so that we can build thoughtful user interactions into our designs. Getting a good core understanding of vanilla (plain) JavaScript is the key objective.

The three-layer web (reminder)
Keeping JavaScript out of markup
Designing user interactions
    Creating HTML with JavaScript (DOM Scripting + Progressive Enhancement)
    Loops and decision-making
    Adding event listeners/handlers to elements
    Creating functions
    Running functions when events occur
    Using local storage

References (JavaScript)

JavaScript and jQuery by Jon Duckett
JavaScript for Web Designers by Mat Marquis (probably the best read to begin)
Learning Web Design (5th Ed) by Jennifer Robbins
Eloquent JavaScript (3rd Ed) by Marijn Haverbeke

JavaScript

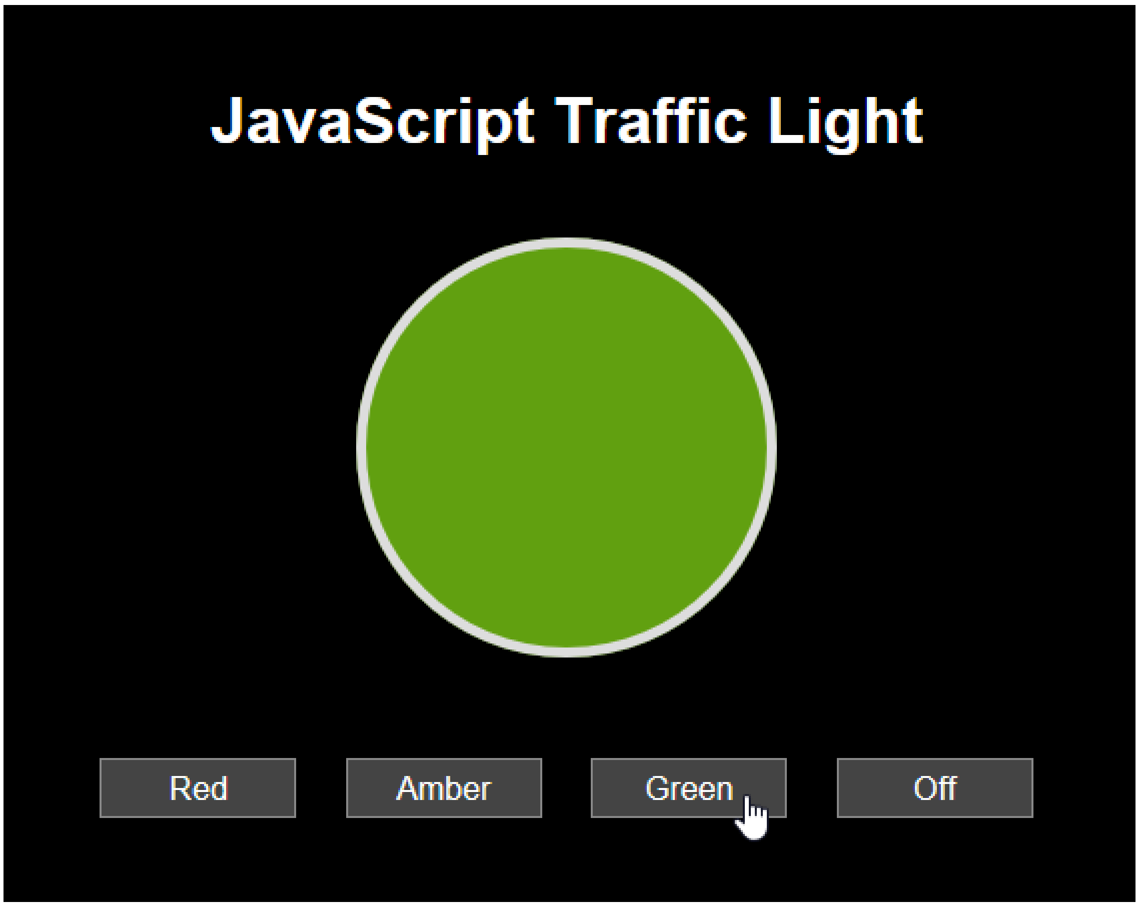
https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
https://www.codecademy.com/learn/introduction-to-javascript
http://eloquentjavascript.net/
https://developer.mozilla.org/en-US/docs/Web/JavaScript

Homework (JavaScript)

Read: The material below.
    Catch up on any reading you didn't manage from Class 1.
    Work through the Introduction to JavaScript Part 3 and Part 4 video tutorials on Moodle.

JavaScript: the DOM and event handlers

This example project demonstrates several important principles in web design and in the use of modern JavaScript. The resulting webpage is very simple and consists of a `<h1>` header, an empty `<div>`, styled to look like a circular light, and an unordered list `<ul>` with four list items `<li>` styled to look like buttons (in principle, buttons should be marked up as `<button>`, but we're using `<li>` here to keep things simple). A script is used to change the background colour of the `<div>` when the list items are clicked. See this script in action.

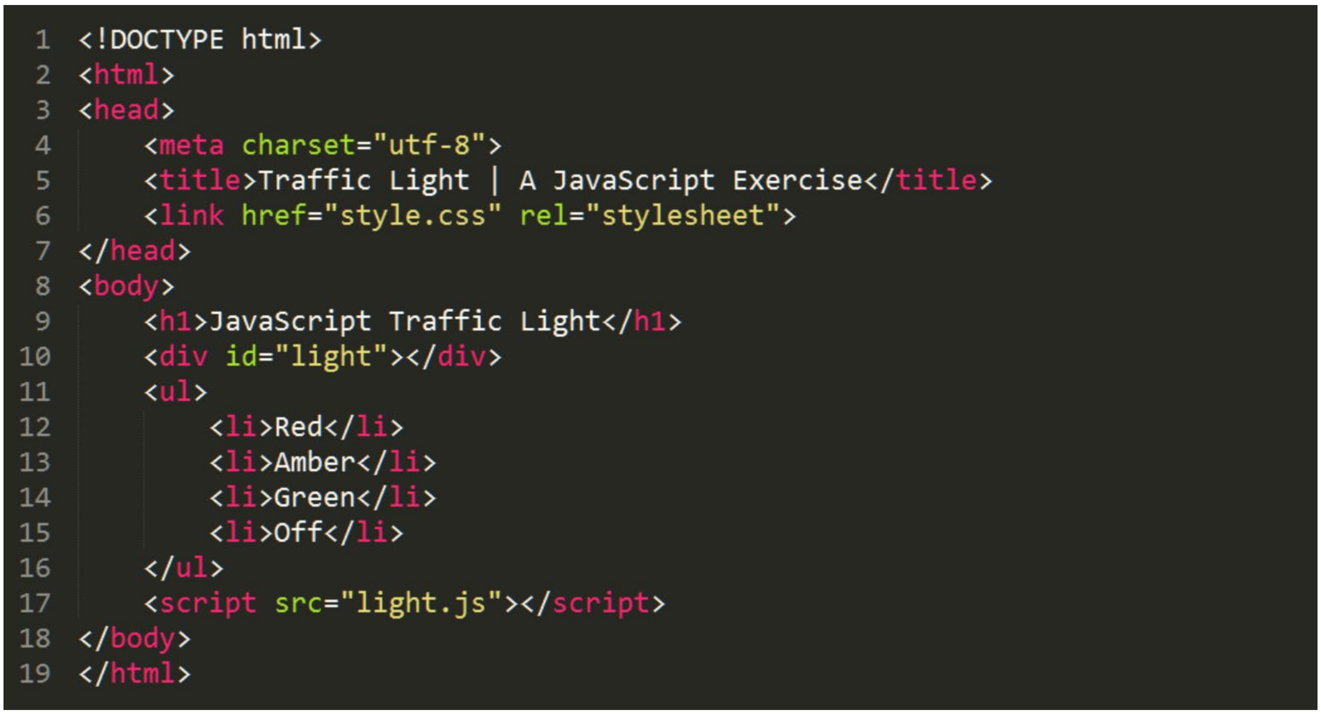


The Web Standards Model

Probably the most important principle demonstrated by this project is the "Web Standards Model". Three client-side technologies, HTML, CSS and JavaScript are all used to create the finished webpage but they are kept separate and in their own files. We begin with the HTML, which provides structure to the document and meaning to the content. This is progressively enhanced with CSS to style the presentation of the document. Finally, the document is progressively enhanced once again by adding JavaScript, which controls behaviours on the webpage.

In order to demonstrate how the page works, I've kept things as simple as possible and placed each of the three components (HTML, CSS and JavaScript) in a file of their own...

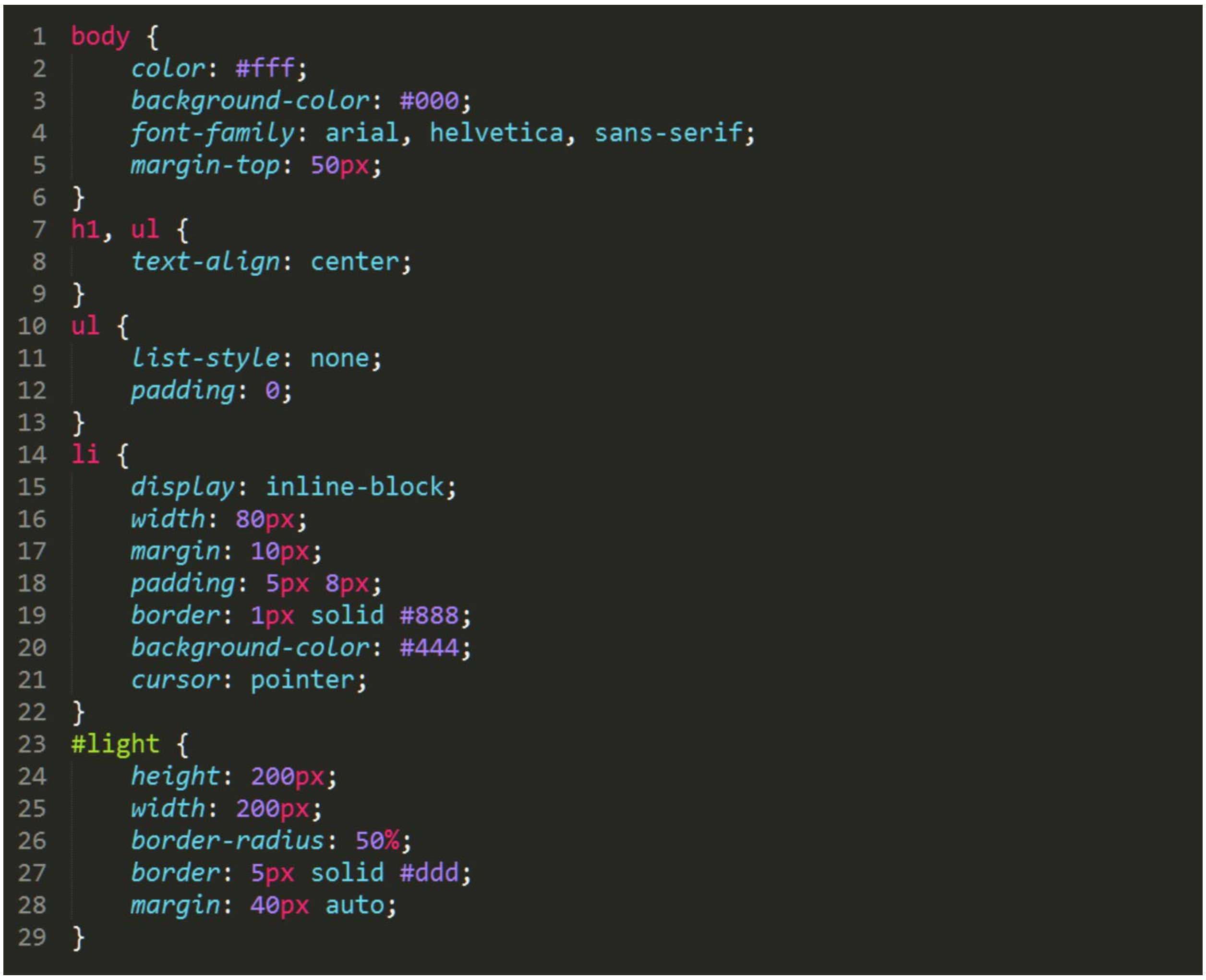
The HTML file



```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Traffic Light | A JavaScript Exercise</title>
    <link href="style.css" rel="stylesheet">
</head>
<body>
    <h1>JavaScript Traffic Light</h1>
    <div id="light"></div>
    <ul>
        <li>Red</li>
        <li>Amber</li>
        <li>Green</li>
        <li>Off</li>
    </ul>
    <script src="light.js"></script>
</body>
</html>
```

As you can see above, the HTML file is very simple. Note that there is no CSS or JavaScript in this file, it's pure HTML. However, there is a link to CSS in the `<head>` section of the file and a `<script>` tag placed immediately before the closing body tag `</body>` that points to a JavaScript file. Note also that the `<div>` has an `id="light"` attribute. This will be used to target that element from CSS when we style it and from JavaScript when we want to change it. There's nothing special about this file, it's just ordinary, well-formed HTML5.

The CSS file

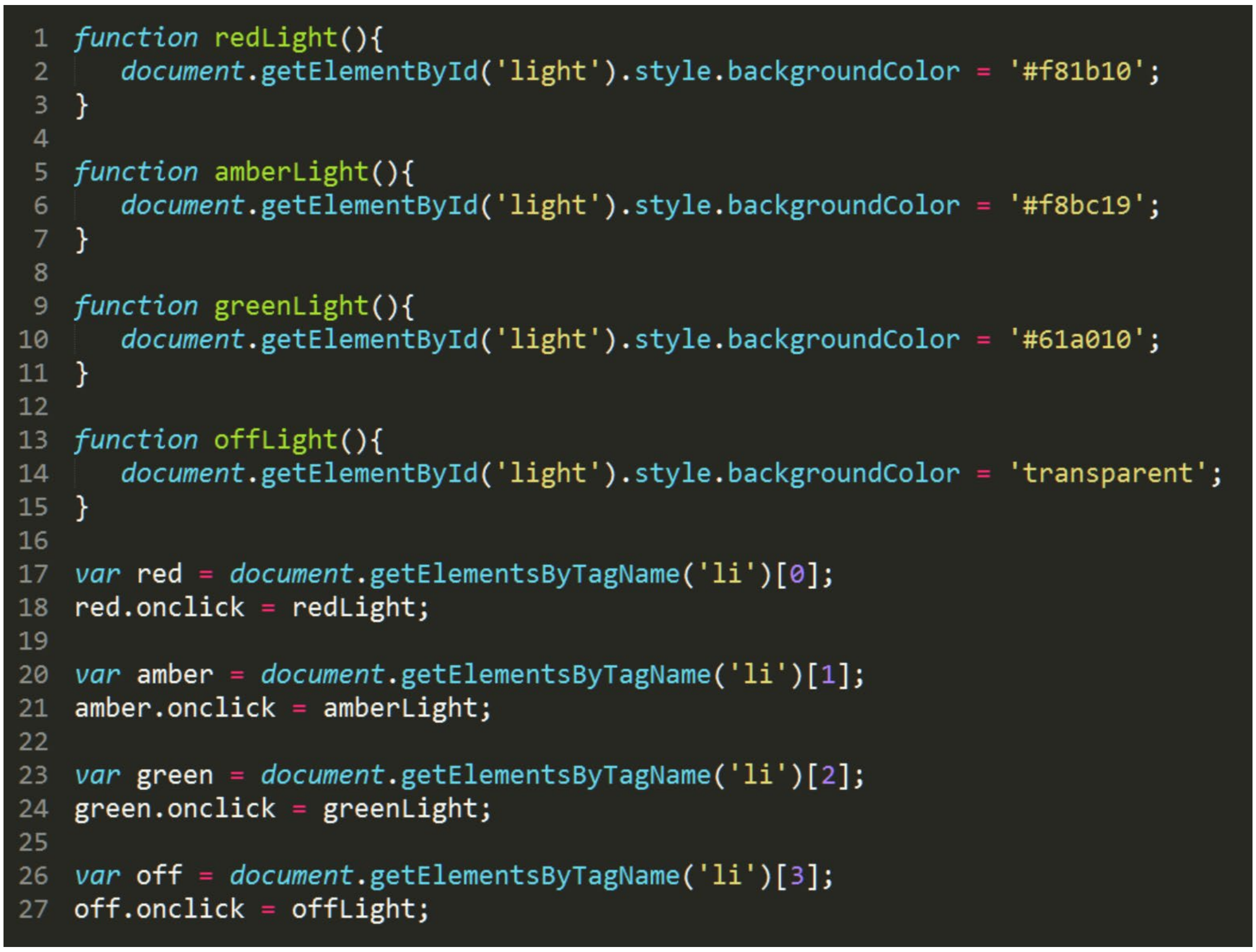


```css
body {
    color: #fff;
    background-color: #000;
    font-family: arial, helvetica, sans-serif;
    margin-top: 50px;
}
h1, ul {
    text-align: center;
}
ul {
    list-style: none;
    padding: 0;
}
li {
    display: inline-block;
    width: 80px;
    margin: 10px;
    padding: 5px 8px;
    border: 1px solid #888;
    background-color: #444;
    cursor: pointer;
}
#light {
    height: 200px;
    width: 200px;
    border-radius: 50%;
    border: 5px solid #ddd;
    margin: 40px auto;
}
```

As this is a very simple webpage, I haven't bothered with a CSS reset but for most projects, you should probably have one. The five style rules shown above are all the CSS needed to style the page.

Since the `<div>` is empty, it's given an explicit height and width of the same value to create a square. The `border-radius: 50%;` declaration is then used to transform the square into a circle. Since no background colour is initially set, a border is added so it can be seen. The list items are styled to look and behave like buttons. The `cursor: pointer;` declaration is used to give some user feedback on hover.

The JavaScript file (version 1)



```
 1 function redLight(){
 2     document.getElementById('light').style.backgroundColor = '#f81b10';
 3 }
 4 
 5 function amberLight(){
 6     document.getElementById('light').style.backgroundColor = '#f8bc19';
 7 }
 8 
 9 function greenLight(){
10     document.getElementById('light').style.backgroundColor = '#61a010';
11 }
12 
13 function offLight(){
14     document.getElementById('light').style.backgroundColor = 'transparent';
15 }
16 
17 var red = document.getElementsByTagName('li')[0];
18 red.onclick = redLight;
19 
20 var amber = document.getElementsByTagName('li')[1];
21 amber.onclick = amberLight;
22 
23 var green = document.getElementsByTagName('li')[2];
24 green.onclick = greenLight;
25 
26 var off = document.getElementsByTagName('li')[3];
27 off.onclick = offLight;
```

The JavaScript above does all the work of listening out for which button has been clicked and then changing the colour of the light accordingly. The script is composed of four functions and four event handlers. Each of the event handlers listens out for a click on one of the buttons. When a button is clicked, the event handler triggers the appropriate function to run, changing the background colour of the `<div>`.

For example, on line 17 a variable `red` is used to store a reference to the first button. The statement uses the `getElementsByTagName` DOM query to select all list items in the document and then refines the selection by using the `[0]` node list index (node lists and arrays always begin with zero, so this is the first item in the node list). Then, on line 18 the `onclick` event is bound to the element reference in the `red` variable and the function `redLight` is assigned to that event on that element.

If the user clicks the red button, the function starting on line 1 is run. Line 1 simply starts the function and gives it a name. Line 2 uses the `getElementById` DOM query to select the element with the ID `light`. It then selects the background colour of that element and sets it to the value shown. Line 3 simply closes the function.

Each of the event handlers and associated functions work in exactly the same way.

As usual with JavaScript, there are several ways to achieve the same outcome. Adding an event handler/listener can be done in one of two ways. In the example above, we've used this method:

```
red.onclick = redLight;
```

That works perfectly well, but we could also use a more modern method:

```
red.addEventListener(‘click’, redLight);
```

The key benefit of the second option is that it allows for several events to be added to an element. The first option allows only the one stated.

Version 1 of this script is intentionally verbose in order to make it easy to understand. Notice that there is quite a lot of repetition and although this makes the script easy to read and understand, it's not optimal. In general, scripts should be DRY (Don't Repeat Yourself).

The JavaScript file (version 2)

```
1  function light(colour){
2      document.getElementById('light').style.backgroundColor = colour;
3  }
4
5  var button = document.getElementsByTagName('li');
6  button[0].onclick = function(){light('#f81b10');};
7  button[1].onclick = function(){light('#f8bc19');};
8  button[2].onclick = function(){light('#61a010');};
9  button[3].onclick = function(){light('transparent');};
```

This version of the script does exactly the same job but is much more compact and with less repetition.

The key difference here is that we are using only one function to do the same job as the four functions in the previous script. We do this by using a function *parameter*. You can see this on line 1 represented by the word `colour`. This is used to pass an *argument* to the statement on line 2 in pretty much the same way as a variable can pass a value. You can see that on lines 6 to 9, each of the calls to the `light` function passes a different argument, shown in parenthesis.

The second notable change is that we are using a variable called `button` to hold the node list of list items. This means we only need to run the query once and not four times. This is obviously more efficient, making the script faster and giving the browser less work to do.

Both scripts (version 1 and version 2) are functionally identical. The page behaves the same way, irrespective of which is used but the second is more satisfactory because it achieves the same outcome in a more efficient and compact way.

When you first start working with JavaScript, don't worry too much about efficiency. The primary objective is to get your script working. But once it does work, consider whether it might be possible to make it more efficient.

See Learn DOM Scripting for more information.

David Watson, January 2024