Advanced JavaScript

Section 1: DOM Manipulation, Objects and Functions

1.1 JavaScript Execution Environment

JavaScript is a interpreted language which means that it is run on the fly line-by-line instead of needing to be compiled (compare this to Java which is a compiled language i.e. before any code can be run it must be run through a compiler).

All code we write as developers has to be understood by a computer (i.e. in JavaScript this is the browser). This is the job of the JavaScript engine and every browser has a JavaScript engine of some sort. The job of the engine is to interpret the code the developer writes and tell the browser how to execute the code.

Example of JavaScript engines are: V8 (Chrome), Spidermonkey (FireFox), JavaScriptCore (Safari). As we can see different browsers have different engines and this means that sometimes browsers will behave differently.

When the engine starts to read the JavaScript code the following things occur:

1. Before any code is executed the global execution context is created. This is the global environment (i.e. the window in a web browser) - this is sort of the top level
2. Any time a function is executed, a new execution context gets created and gets added to the call stack

JavaScript is a single threaded language i.e. it can do one thing at a time (as opposed to a multi-threaded language which can do multiple things at a time). The call stack is a data structure that contains information on the order of function calls. The last function that gets called is the first function to get out of the call stack (i.e. LIFO).

1.2 DOM Manipulation

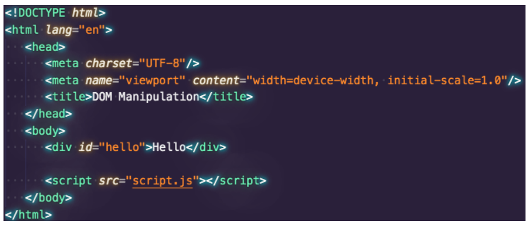
The DOM stands for Document Object Model and is an object-oriented representation of a webpage which can be modified with a scripting language such as JavaScript (definition from Mozilla Developer Network (MDN)).

HTML and JavaScript are two different languages and need a way to interact with each other. The DOM is neither HTML or JavaScript but rather an interface for JavaScript, HTML and CSS to interact with each other. We can think of the DOM as an API (interface) for interaction between the different languages.

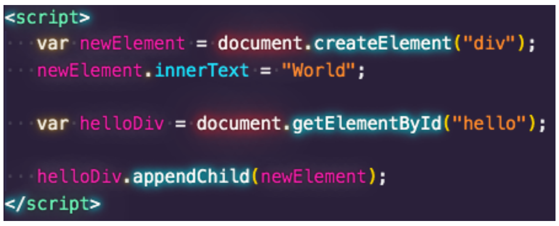
In the example below we are using JavaScript to manipulate the DOM to interact with the HTML webpage. The document is an object with many different properties and methods we can use to manipulate the DOM. This does not mean that we are directly

manipulating/changing the HTML i.e. whatever we do with JavaScript might interact with the DOM but it does not directly interact with the HTML or CSS.
The createElement is a method we can call on the document object to create a new HTML element on the DOM such as a <div> element. The innerText property allows us to add text to the element. This creates a new element but nothing is done with this element to add to the DOM/HTML.

The getElementByID method allows us to grab an element in the HTML and make it into an object representation of that element which is stored in a variable. We can use both elements which are stored in variables to manipulate the DOM. The .appendChild method allows us to append an element to another element.



```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>DOM Manipulation</title>
  </head>
  <body>
    <div id="hello">Hello</div>

    <script src="script.js"></script>
  </body>
</html>
```



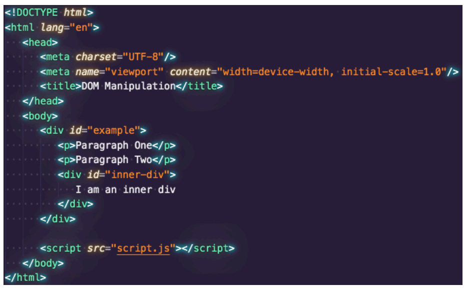
```html
<script>
  var newElement = document.createElement("div");
  newElement.innerText = "World";

  var helloDiv = document.getElementById("hello");

  helloDiv.appendChild(newElement);
</script>
```

Important Note: We could have used the .innerText property to change the div with the id of hello with a different text. This does not change the HTML document; however, this does manipulate the DOM which is the HTML page representation and what gets rendered in the browser window.

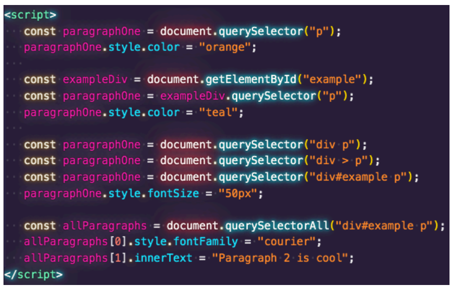
With JavaScript we can manipulate a webpage via the DOM and its core this is what JavaScript code does with a browser and how we can make webpages interactive. We can use the document object model (DOM) to create elements, edit elements and remove elements from the webpage.

Whenever we manipulate the DOM we need for it to be ready first i.e. we need the HTML and CSS to be loaded into the browser and ready for us to manipulate. If we run our JavaScript to manipulate the DOM before it is loaded it might cause errors or the page not to function correctly. The JavaScript code is typically loaded into the HTML via <script> tags at the bottom of the body because the browser reads the HTML document from top to bottom which will load all the content of the webpage before loading the JavaScript which manipulates the DOM i.e. the JavaScript will only run after the DOM is completed loaded.

There are many ways to grab an element from a HTML document using the document object model for example we can use the getElementById, getElementByClass, querySelector and querySelectorAll methods. There are many ways to achieve the same results which is true in every programming language. Below is an example of JavaScript code selecting elements on a HTML document using the DOM object.

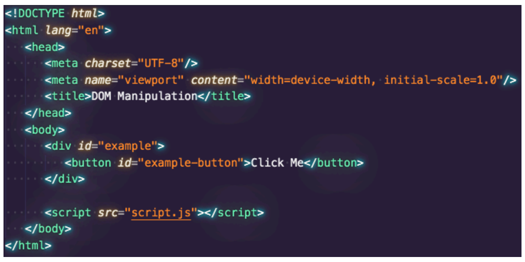


```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>DOM Manipulation</title>
  </head>
  <body>
    <div id="example">
      <p>Paragraph One</p>
      <p>Paragraph Two</p>
      <div id="inner-div">
        I am an inner div
      </div>
    </div>

    <script src="script.js"></script>
  </body>
</html>
```

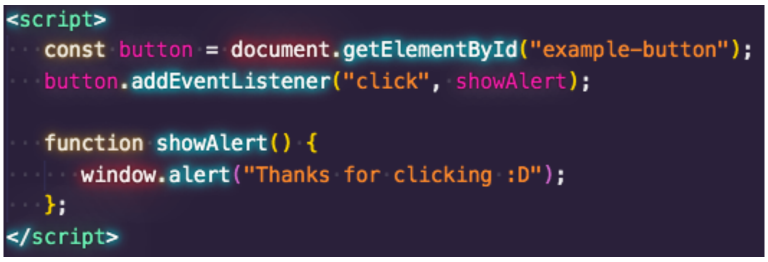


```html
<script>
  const paragraphOne = document.querySelector("p");
  paragraphOne.style.color = "orange";

  const exampleDiv = document.getElementById("example");
  const paragraphOne = exampleDiv.querySelector("p");
  paragraphOne.style.color = "teal";

  const paragraphOne = document.querySelector("div p");
  const paragraphOne = document.querySelector("div > p");
  const paragraphOne = document.querySelector("div#example p");
  paragraphOne.style.fontSize = "50px";

  const allParagraphs = document.querySelectorAll("div#example p");
  allParagraphs[0].style.fontFamily = "courier";
  allParagraphs[1].innerText = "Paragraph 2 is cool";
</script>
```

The addEventListener is a special method which tells JavaScript to listen to an event on an element and when that event occurs to trigger the function. The method has a pointer reference to the function which is passed in as the second argument/parameter to the addEventListener method. In the below example, when the <button> element with an id of example-button is clicked it will add the showAlert function to the call stack and when it resolves it will alert the user with an alert message.



```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>DOM Manipulation</title>
  </head>
  <body>
    <div id="example">
      <button id="example-button">Click Me</button>
    </div>

    <script src="script.js"></script>
  </body>
</html>
```



```html
<script>
  const button = document.getElementById("example-button");
  button.addEventListener("click", showAlert);

  function showAlert() {
    window.alert("Thanks for clicking :D");
  };
</script>
```

This is a few example of how we can use JavaScript with the DOM to manipulate the HTML and CSS of webpages.