Lesson 7

Object review, browser Document Object Model

Lesson Objectives

- Review objects, constructors and methods
- Explain the difference between an HTML page and its associated Document Object Model.
- Access and manipulate elements on a web page using the DOM API
- Use event listeners to trigger events, using the DOM API

Exit ticket questions

Q: How would I reference a js file from another js file, say when making a Hubot?

Exit ticket questions

Q: How would I reference a js file from another js file, say when making a Hubot?

A: This is something you can do in Node pretty easily but you actually can't do it in the browser — browser JS was designed to be completely secure, and not give the JS code access to the files on the user's computer.

We won't be getting into reading and writing files using Node in class, but if anyone has any questions about it outside of class, we'd be happy to go over it.

Exit ticket questions

Q: Is the code and syntax used for the Hubot project exclusive to this type of project and Hubot, or can we use this code elsewhere? is this code part of a different library that we downloaded and installed?

Exit ticket questions

Q: Is the code and syntax used for the Hubot project exclusive to this type of project and Hubot, or can we use this code elsewhere? is this code part of a different library that we downloaded and installed?

A: All the code we used for the Hubot project was either basic Node stuff, or specific to Hubot. There were not really any third party libraries, except ones that Hubot used under the hood. Also, a lot of the ways that we had to do things to interface properly with the Hubot are common patterns in other libraries and frameworks, like creating event listeners and passing them callback functions, and the callback takes a response object. We will see a lot of these patterns soon.

A couple of points from the homeworks:

- Proper indentation is very important for readable code. Most of you are doing it.
  - http://www.w3schools.com/js/js_conventions.asp
    - Sublime Text can indent for you automatically (sort of)
- Best not to use falsiness for checking if something is zero or the empty string:
  - Use `if (x === 0)` instead of `if (!x)`

Object review: Monkeys!

- Create a new file called “monkeys.js”
- Work with a partner to create a monkey object, which has the following properties:

  name
  species
  foodsEaten

- And the following methods:
  - **eatSomething**: takes a thing to eat (as a string) and adds it to foodsEaten.
  - **introduce**: introduces itself (using console.log), including its name, species, and what it's eaten.
- Create 3 monkeys total. Use a constructor function, with arguments passed in, to create your monkeys. Give the monkeys access to the methods above using the constructor’s “prototype” syntax.
- Exercise your monkeys by retrieving their properties and using their methods. Practice using both syntaxes for retrieving properties (dot notation and square bracket notation)

Intro to Chrome console

What is the “DOM”?

- DOM stands for Document Object Model

- It’s a representation in JavaScript of the structure and content of an HTML tree.

What do we mean, ‘HTML tree’?

Element node

Text node

```
<html>
  <head>
    <title>Palace of Fruits</title>
  </head>
  <body>
    <img src="http://bit.ly/1V62kTZ">
    <h1>Some <i>nice</i> fruits</h2>
  </body>
</html>
```

A short video

- Watch a short video (7 min) about how the DOM fits into the rendering cycle for a web page: https://www.youtube.com/watch?v=n1cKlKM3jYI

How we manipulate the DOM

- The DOM is a large data structure that is exposed to JavaScript as an object that we can access, called “document”

- We manipulate the DOM by accessing “document” and all its child objects and arrays

- The whole purpose of the DOM is to give us a hook to manipulate the contents of the web page using JavaScript.

- What does this mean? We can change text and colors, add and remove elements, make them clickable, etc.

JS in the browser

- There are 3 ways to run JS in the browser:

1. Inline
2. In script tags
3. In separate files

Running JS inline

- You can run JS code inline, as strings in html element attributes:
  - **<body onload="window.alert('welcome to my app!’);">**
- Not recommended. Code will get impossible to organize and understand, really fast

Running JS in script tags

- Also not generally recommended
- Perfectly reasonable for playing around, or early stages of project
- Sometimes used for 3rd party code

```
<html>
  <head>
    <script>
        alert('Welcome to my app!');
    </script>
  </head>
  <body>
  </body>
</html>
```

Running JS from separate files

- index.html:

```
<html>
  <head>
    <script>
       alert('Welcome to my app!');
    </script>
  </head>
  <body>
    <script src="app.js"></script>
  </body>
</html>
```

- app.js:

```
alert('Welcome to my app!');
```

window.onload

- The JS can start running and trying to access DOM elements before they've been created.

- To avoid that, wrap all code that tries to immediately access DOM elements in **window.onload:**

```
window.onload = function() {
  alert('page is fully loaded');
  // ... do DOM-related stuff
}
```

Codealong

- Make a lesson7 directory

- Create 2 new files in that directory: **index.html** and **app.js**

Creating elements

- Use **document.createElement** to create an element:
  - document.createElement(“h1");
- Use **document.createTextNode** to create some text:
  - document.createTextNode("Hello dynamic world!”);
- Notice “h1” doesn’t have **<** and **>** around it

Adding elements to the page

- But our new nodes are just floating in memory when you create them. They're not on the page.

- Use **.appendChild** (works on elements) to attach the text node to the new element, and the new element to the page:

  - Note: **document.body** is a special property which gives you access to the html **body** element of the page.

```
var main_heading = document.createElement("h1");
var heading_text = document.createTextNode("Hello dynamic world!");
main_heading.appendChild(heading_text);
document.body.appendChild(main_heading);
```

Steps in adding text to a page:

1. Create the new H1 element using the **document.createElement** method.

2. Create the text using the **createTextNode** method.

3. Attach the text to the newly created **H1** element.

4. Attach the newly created **H1** element to the **body** element.

Retrieving Elements

- Many ways to *retrieve* elements, meaning get a reference to them so we can do stuff with them

document.getElementById

- Retrieves an element based on its unique HTML id

```
<ul>
  <li id="happy">I'm so happy I'm about to be retrieved!</li>
  <li id="sad">I'm feeling left out.</li>
  <li id="indifferent">I don't care.</li>
</ul>
```

```
var happyLi = document.getElementById('happy');
console.log(happyLi);
// => shows the first li element
```

document.querySelector

- Retrieves the first element it finds that matches a CSS selector string

```
<ul>
  <li id="happy" class="in-touch-with-feelings">
    I'm so happy I'm about to be retrieved!</li>
  <li id="sad" class="in-touch-with-feelings">
    I've got a bad feeling about this one again.</li>
  <li id="indifferent">Nope, still don't care.</li>
</ul>
```

```
var li = document.querySelector('.in-touch-with-feelings');
console.log(li);
// => shows the first li element
```

document.querySelectorAll

- Retrieves the a list of all elements it finds that matche a CSS selector string

```
<ul>
  <li id="happy" class="in-touch-with-feelings">
    I'm so happy I'm about to be retrieved!</li>
  <li id="sad" class="in-touch-with-feelings">
    Could it be? Will I be retrieved? O frabjous day!</li>
  <li id="indifferent">I. Do. Not. Care.</li>
</ul>
```

```
var lis = document.querySelectorAll('.in-touch-with-feelings');
console.log(lis);
// => shows the first TWO li elements
```

codealong

- Use the same files you used for the last one.

Intro to manipulating the DOM: Documentation! (MDN)

- Let's take a look at the Mozilla Developer Network documentation about the DOM.

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

    - Specifically: createElement, getElementById, appendChild

    - (also might want to look at createTextNode, querySelector, querySelectorAll)

Homework

- Instructions in files at:

- http://bit.ly/jsdev2-lesson7-hw4