JS in the Browser && DOM && Debugging

Recap & Intro

- Over the past few weeks we've gone over the fundamentals of JavaScript
- As we move into the using JS in the browser, you will notice that it's a natural step.
- All the hard work you've put in will result in
  - The ability to quickly build real-time applications
  - A better understanding of JS libraries and the code of others
  - a lot of FUN!
- Tonight's class is going to focus 3 things
  - how to load/interact with JS in your browser
  - your browser's developer tools
  - how to debug your code

Loading a Script

Load a script into an HTML page by adding a script tag

In the head tag

```html
<html>
  <head>
    <meta charset="utf-8">
    <title>A Page with JS</title>
    <script src="javascript/your_file.js" charset="utf-8"></script>
  </head>
  <body>
  </body>
</html>
```

In the body

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>A Page with JS</title>
  </head>
  <body>
    <h1> Hello World</h1>
    <script type="text/javascript" src="javascript/your_file.js" charset="utf-8"></sc
  </body>
</html>
```

Blocking and Performance considerations

When loading a script in the head, you'll want to consider the concept of blocking.

When a page loads:

- Resources in the page are blocked from downloading if they are below the script.
- Elements are blocked from rendering if they are below the script.

This can seriously affect a user's experience, and slow down your app's performance. So, be mindful of where you are referencing your JS.

####Basic example script running in a page

index.html

```
<body>
  <h1 id="greeting"> Hi! </h1>
  <script type="text/javascript" src="javascript/main.js"></script>
</body>
```

javascript/main.js

```
function greetOnLoad () {
    var name = prompt("Hi! What's your name?");
    var myelement = document.getElementById("greeting");
    greeting.innerHTML= "Hello " + name + ", it's nice to meet you!";
}
greetOnLoad();
```

Inline vs External

Above you have seen how to load a script *externally*. However, you can also embed your js right in your html page. This is called *inline* scripting.

Inline Examples

```
  <body>
    <p id="example">Lorem ipsum dolor sit amet <br> Yes, this is a random paragraph 
    <button type="button" onclick="modifyColor()">Click Me</button>
    <script>
    function modifyColor() {
        document.getElementById("example").style.color = "aqua";
        document.getElementById("example").style.backgroundColor = "gray";
    }
    </script>
  </body>
```

```
<body>
  <button type="button" onclick="alert(Math.floor(Math.random() * (7 - 1)) + 1)">Cli
  <script type="text/javascript" src="javascript/main.js"></script>
</body>
```

While inline scripting may seem convenient, it is not recommended.

Comparison

Inline scripts

- Difficult to maintiain
- Larger HTML file size
- Does not cache

External scripts

- Maintainable
- Once an external script is downloaded, the browser stores it in the cache so if another page reference it no additional download is required.
- Can be used to load client code on demand and reduce overall download time and size.

Chrome Dev Tools Part1 / JS console

Chrome includes developer tools that let you:

- inspect the state of JS on your page
- send commands
- debug

In addition to that

- variables (from your `.js` files) are visible and editable

to access your console you can:

- right click
- choose: inspect element
- click on the console tab

Alternatively, you can use the keyboard shortcut `command` + `option` + `j`

By now, you are well familiar with console.log, However, your console has a few other handy actions

- console.warn
- console.error

Let's see what both look like in action:

Global Objects

The global object is created as soon as the Javascript interpreter starts. For Javascript running inside a browser, this means that as soon as a new page is loaded by the browser, the global object will be created.

Meaning: When scripts are loaded into an html document, the browser inserts some globals

In browsers, the `window` object doubles as the `global` object

Essentially, when your working with JS in the browser, `this` initially refers the `window`

some examples:

```
window.alert();
window.prompt();
window.confirm();
```

is the same as:

```
this.alert();
this.prompt();
this.confirm();
```

Other kinds of data types that loaded into global include:

- functions such as: `parseFloat()`
- properties such as `NaN`
- objects such as `Math`

In summary, the `Window` object is basically a representation of an individual browser window.

Document Object Model

The most important global object is `window.document`. It represents the HTML document loaded in the window.

- DOM is a tree of nodes

- DOM provides methods for finding and editing all nodes from JS

The HTML DOM Document Object

The HTML DOM Element Object

The DOM and the nodes have an API that lets you find nodes and edit them directly. This isn't terribly useful by itself, but it lays foundational concepts.

running `window.document` in your console will give you access to the entire Document Object

Find a node

```
//Get an element ById
var myElement = document.getElementById('myEl');
//Get a collection of elements by tag name
var paragraphs = document.getElementsByTagName('p');
//Get a collection of elements by class name
var activeEls = document.getElementsByClassName('active');
//Get elements by a CSS Selector
var matches = document.querySelectorAll('#header p.active');
```

We'll dive deeper into the DOM in our next class.

Location

The `window.location` object can be used to get the current page address (URL) and to redirect the browser to a new page.

```
window.location.href
// returns the href (URL) of the current page
window.location.hash
// sets or returns the anchor part of a URL, including the hash sign (#)
```

```
window.location.hostname
//returns the domain name of the web host
```

To see this in action, try this is the console (from any page)

```
window.location.href = "http://www.google.com"
```

History

The `history` interface allows to manipulate the browser session history, that is the pages visited in the tab or frame that the *current* page is loaded in.

- `window.history.back();` Goes to the previous page in session history, (the same action as when the user clicks the browser's Back button)
- `window.history.forward();` Goes to the next page in session history, (the same action as when the user clicks the browser's Forward button)

- `window.history.go();`

    - takes an argument
    - `window.history.go(1);` moves forward one page
    - `window.history.go(-1);` moves back one page

- HTML5 pushstate The `history.pushState()` method adds a state to the browsers history.

```
var stateObj = { foo: "bar" };
history.pushState(stateObj, "page 2", "bar.html");
```

Running the above code will cause the URL bar to display `http://your-site-name.com/bar.html`

`pushState()` takes three parameters:

- a state object
- a title (which is currently ignored)
- an optional URL

Dimensions and Position

- `window.innerWidth` returns the inner width of a window's content area.
- `window.innerHeight` returns the inner height of a window's content area.

note: these properties are read-only.

Another nifty feature is `window.scrollY` and `window.scrollX`

X and Y represent the X and Y axis of the window

- Y is the number of pixels that the document is currently scrolled from the top
- X is the number of pixels that the document is currently scrolled from the left

```
// scroll to the bottom of a page and then paste this into the console
if (window.scrollY) {
  window.scroll(0, 0);  // will reset the scroll position to the top left of the docu
}
```

```
var y = window.scrollY;
window.scroll(0, 0);
window.scroll(0, y);
```

localStorage & sessionStorage

Storage objects are simple key-value pairs (similar to objects) they stay intact through page loads.

- The keys can be strings or integers
- the values are always strings.
- You can access these values like an object
- or with the `getItem()` and `setItem()` methods.

The `sessionStorage` property allows you to access a session `Storage` object.

- Data stored in `sessionStorage` gets cleared when the page session ends.
- A page session lasts for as long as the browser is open and survives over page reloads
- Opening a page in a new tab or window will cause a new session to be initiated

`localStorage` while property similar to `sessionStorage` has:

- no expiration time
- persists even when the browser is closed and re-opened

The `setItem()` method will add (or update) a key/value pair to the Storage object

```
localStorage.setItem('bgcolor', 'aqua');
```

While the `getItem()` method will return that key's value.

```
localStorage.getItem('bgcolor'); // "aqua"
```

Mozilla has a great demonstration at `http://mdn.github.io/web-storage-demo/`

head over there and play around with it, while calling `localStorage` to see the key/value pairs

This is a great time to introduce the idea of serialization...

You may have noticed that these key/value pairs resemble JSON. However, Attempting to store a JSON object in local storage will result in an error.

Local storage requires you to store strings of information. *To store a JSON object in local storage you will need to convert it into a JSON-formatted string*, using the `JSON.stringify()` function.

```
var user = {
  name : "Shane",
  job : "instructor"
};
var storeUser = JSON.stringify(user);
localStorage.setItem('userInfo', storeUser);
localStorage.getItem('userInfo');
// "{"name":"Shane","job":"instructor"}"
var parsedData = localStorage.getItem('userInfo');
console.log(parsedData);
// {"name":"Shane","job":"instructor"}
```

Exercise 1

- Loading your Rock Paper Scissors script into an HTML page
- Once loaded:
  - Access the variables/arrays
  - Call functions from console
- Make the program interactive
- Ask the user for input
- Place the winning results on the page

##Chrome developer tools Part 2

Elements tab

- Edit HTML
- edit styles
- find by magnifying glass

Sources / Debugging

- Setting breakpoints / conditional
- Step over, into, out
- Watch variables

Network tab

- timeline / performance
- Headers / response

Resources

- LocalStorage
- Cookies,
- AppCache

Exercise

Debug the following code:

```
var littleOne = [];
var howTheyMarch = ["one by one", "two by two", "three by three", "four by four", "fi
function theAntsGoMarching(){
    for (var i = 0; i < howTheyMarch.length; i++){
        howManyByHowMany;
        console.log(littleOne[i]);
        console.log("And they all go marching down to the ground to get out of the ra
    }
}
theAntsGoMarching();
littleOne[0] = "The little one stops to suck her thumb";
littleOne[1] = "The little one stops to tie his shoe";
littleOne[2] = "The little one stops to climb a tree";
littleOne[3] = "The little one stops to shut the door";
littleOne[4] = "The little one stops to take a dive";
littleOne[5] = "The little one stops to pick up sticks";
littleOne[6] = "The little one stops to pray to heaven";
littleOne[7] = "The little one stops to roll a skate";
littleOne[8] = "The little one stops to check the time";
littleOne[9] = "The little one stops to shut The End";
function howManyByHowMany(number){
    var numbers = howTheyMarch;
    var hurrah = " hurrah, hurrah \n";
    var march = "The ants go marching " + numbers;
  debugger
```

```
    console.log(march + hurrah + march + hurrah + march);
}
```

- Create a new project in your TTS directory titled antsGoMarching
  - add an index.html file
  - add a javascript file
  - link the two files together
- In your JS file paste the above code
- there are several errors in this code sample
  - use your debugger to track those errors down
- We've already given you a `debugger` statement to start with.
- Your finished output should be something like this:

Exercise Answer:

```
var littleOne = [];
var howTheyMarch = ["one by one", "two by two", "three by three", "four by four", "f
littleOne[0] = "The little one stops to suck her thumb";
littleOne[1] = "The little one stops to tie his shoe";
littleOne[2] = "The little one stops to climb a tree";
littleOne[3] = "The little one stops to shut the door";
littleOne[4] = "The little one stops to take a dive";
littleOne[5] = "The little one stops to pick up sticks";
littleOne[6] = "The little one stops to pray to heaven";
littleOne[7] = "The little one stops to roll a skate";
littleOne[8] = "The little one stops to check the time";
littleOne[9] = "The little one stops to shut The End";
function theAntsGoMarching(){
    for (var i = 0; i < howTheyMarch.length; i++){
        howManyByHowMany(i);
        console.log(littleOne[i]);
        console.log("And they all go marching down to the ground to get out of the ra
    }
}
function howManyByHowMany(number){
    var numbers = howTheyMarch[number];
    var hurrah = " hurrah, hurrah \n";
    var march = "The ants go marching " + numbers;
    console.log(march + hurrah + march + hurrah + march);
}
theAntsGoMarching();
```

Exercise

- Same as before, work through this code using the Developer Debugging tools

```
var woolOwners = [
  {
      "master": 1
  },
  {
      "dame": 1
  },
  {
    "little boy": 1,
    "location": "down the lane"
  }
];
var bags = haveYouAnyWool;
var haveYouAnyWool = function() {
    for (var i = 0; i < woolOwners.length; i++) {
    var totalBags = totalBags + i;
    }
    return (i);
};
function baabaaBlackSheep() {
    console.log("BaaBaa BlackSheep have you any wool?");
    if (bags > 0) {
        console.log("yes sir, yes sir " + totalBags + " bags full");
  }
}
function oneForMy() {
    for (var i = 0; i < 2; i++) {
        people = Object.keys(woolOwners);
        var person = people.toString();
        console.log("one for my " + person);
    }
}
baabaaBlackSheep();
oneForMy();
var boy = Object.keys(woolOwners[2]);
var littleBoy = boy[2];
var whereHeLives = littleBoy.location;
console.log("one for the " + littleBoy + " that lives " + whereHeLives);
```

- your output should be:

<!--

- Some scripts with errors that you have to debug
  - Error in a script
  - Network error
  - Setting breakpoints
  - step over, step into

    - Set and get stuff from local storage -->

Homework

###Due 8/11/16

- Complete the BaaBaaBlackSheep debugging challenge (individually)
    - Driver - push the code to repo with the naming convention: `js_debugging_YOUR_TEAMS_INTIALS_HERE`
    - Navigator - clone the repoe and create a new branch to work from

Due 8/16/16

- Read this excellent blog post about developer tools.
- Complete the Discover DevTools course at CodeSchool

Lesson 8 - DOM - Traversal, Manipulation and Events

Recap & Intro

- Last week we introduced working with JavaScript in the browser
    - window object
    - DOM
- Today, we're going to dive in to working with the DOM.

DOM

To recap, the Document Object Model (DOM) is a JavaScript representation of the HTML document loaded into the browser. The DOM API lets you:

- Find elements (nodes) in the document
- Edit, add, or remove nodes
- Attach event handlers that respond to user input.

Find Nodes

There are a few different ways to get a node or array of nodes from the document.

getElementById()

Gets a single node based on id attribute

HTML

```
<input id="username"></input>
```

JS

```
//Get a single node
var el = document.getElementById('username');
```

getElementsByTagName() & getElementsByClassName()

Gets an array of nodes based on a tag name or className

HTML

```
<input type="text" class="error"></input>
<input type="password"></input>
<button>submit</button>
```

JS

```
//Get all inputs
var inputs = document.getElementsByTagName('input');
var inError = document.getElementsByClassName('error');
inputs.length; //2
inError.length; //1
```

querySelector & querySelectorAll

In practice, these are the only DOM selectors you will ever need. They take a CSS selector as an argument, which means you can easily duplicate the functionality from the other DOM selection functions.

HTML

```
<input type="text" class="error"></input>
<input type="password" class="password"></input>
<button>submit</button>
```

JS

```
//Get all inputs
var firstButton = document.querySelector('button');
var inError = document.querySelectorAll('input.error');
firstButton //single button node
inError //Node list of inputs with class 'error'
```

NodeList vs Array

It seems like querySelectorAll should return an Array of elements. In fact, it returns a nodeList, which offers a similar, but not identical API to Array.

```
var links = document.querySelectorAll('a');
//Works!
var linkCount = links.length;
var firstLink = links[0];
//Doesn't work!
links.forEach(function(link){
    //do something with link
});
```

Array methods like `forEach`, `map`, `reduce`, and so on, don't work. Luckily, its easy enough to convert a nodeList into an Array;

```
var links = document.querySelectorAll('a');
var arrayOfLinks = Array.from(links);
```

Traversing the DOM

You can use the children, parent, nextElmentSibling, and previousElementSibling attributes to find nodes relative to a node you have. This is called "traversing the DOM".

Children

Use the children property to gets a nodeList of all the nodes contained in the node.

HTML

```
<ul>
  <li>Item 1</li>
    <li>Item 2</li>
</ul>
```

JS

```
var listItems = document.querySelector('ul').children;
listItems.length; //2
```

Siblings and Parents

Use parent, nextElementSibling, and previousElementSibling to find nodes up the tree and across it.

HTML

```
<header>
    <ul>
        <li class="first">Item 1</li>
        <li class="selected">Item 2</li>
        <li class="last">Item 3</li>
    </ul>
</header>
<section>
    Hello!
</section>
```

JS

```
var selectedItem = document.querySelector('li.selected')
var first = selectedItem.previousElementSibling;
var last = selectedItem.nextElementSibling;
var list = selectedItem.parentElement;
var header = selectedItem.parentElement.parentElement;
var section = selectedItem.parentElement.parentElement.nextElementSibling;
```

Exercise 1: Selecting Nodes

```
<html>
    <body>
        <header>
            <ul>
                <li class="first">Item 1</li>
                <li class="selected">Item 2</li>
                <li class="last">Item 3</li>
            </ul>
        </header>
        <div class="col">
            <section>
                <h2>Section 1</h2>
            </section>
            <section class="current">
                <h2 class="highlight">Section 2</h2>
            </section>
            <section>
                <h1>Section 2</h1>
            </section>
        </div>
    </body>
</html>
```

1. Get the header element
2. Get all the section elements
3. Get the section element with class="current"
4. Get the section that comes after the current section
5. Get the h2 node from the section before the 'current' section
6. Get the div that contains the section that has an h2 with a class of 'highlight'
7. Get all the sections that contain an H2 (using a single statement);

Exercise 1 Answer

```
//Laziness is your friend...
var q = document.querySelector.bind(document)
var qa = document.querySelectorAll.bind(document);
q('header');
qa('section');
q('section.current');
q('section.current').nextElementSibling;
q('section.current').previousElementSibling.children[0];
q('h2.highlight').parentElement.parentElement;
 var foo = Array.from(qa('section h2'))
    .map(function(headline){
        return headline.parentElement;
    })
```

Editing a node

A Node object has some useful properties and methods to let you access its contents and edit its appearance and content.

innerHTML

The sledgehammer approach. Get or set the html text inside a node. This is really simple and sufficient in most cases.

HTML

```
<div>
  <a href="#">Click me</a>
</div>
```

JS

```
//Get all inputs
var div = document.querySelector('div');
var a = document.querySelector('a');
a.innerHTML; //"click me"
div.innerHTML; //'<a href="#">Click me</a>'
a.innerHTML = "Updated link text";
```

Attributes

Get and set attributes like object properties

```
<a href="http://google.com" name="googleLink">Click me</a>
```

```
var a = document.querySelector('a');
//Get an attribute
a.href; //"http://google.com"
//Set an attribute
a.name = 'new link name';
//Add a new attribute
a.target = "_blank";
```

Removing nodes

Use `remove` to remove a node from a document.

HTML

```
<header>
    <ul>
        <li class="first">Item 1</li>
        <li class="selected">Item 2</li>
        <li class="last">Item 3</li>
    </ul>
</header>
<section>
    Hello!
</section>
```

JS

```
//Remove the first list item
document.querySelector('.first').remove();
```

Adding nodes

Create a node using `document.createElement('tagname')` and `node.appendChild(el)`

HTML

```
<header>
    <ul>
        <li>Item 1</li>
    </ul>
</header>
```

JS

```
var newLI = document.createElement('li');
newLI.innerHTML = "Item 2";
var list = document.querySelector('ul');
list.appendChild(newLI); //Insert after item 1
```

Exercise 2: Editing the DOM

```
<html>
    <body>
        <h2>Shopping List</h2>
        <ul id="list">
            <li>Ramen</li>
            <li>Coffee</li>
            <li>Poptarts</li>
            <li>Twinkies</li>
        </ul>
    </body>
</html>
```

1. Update the 'Coffee' item to say 'Fair Trade Coffee'
2. Remove 'Twinkies' from the list
3. Add an item 'Cheese Whiz'
4. Clear the list and programmatically add items from the array `['protein powder', 'muscle milk', 'power bars']`
5. Add the class 'important' to the muscle milk item.

Exercise 2: Answer

```
var list = document.querySelector('#list');
//1
list.children[1].innerHTML = "Fair Trade Coffee";
//2
list.children[3].remove();
//3
var cheese = document.createElement('li');
cheese.innerHTML = 'Cheese Whiz';
list.appendChild(cheese);
//4
list.innerHTML = '';
['protein powder', 'muscle milk', 'power bars'].forEach(function(itemText){
    var li = document.createElement('li');
    li.innerHTML = itemText;
    list.appendChild(li);
})
//5
list.children[1].className = "important"
```

- Update text of an item
- Remove an item
- Add an item
- Add a series of items based on data
- Update attributes based on data

DOM Events

As previously discussed, async programming is important in JS. DOM events allow us to make use of asnychronous functions

Elements emit events based on user input. You can run code in response to them. Events include:

- Mouse events - click, mouseover, mouseout
- Keyboard events - keydown, keyup, etc
- Form events - submit, blur, focus, change,

- window events - load, hashchange, etc.
- touch events - touchstart, touchend, etc.

Check out w3 DOM Events for a more complete description of DOM Events

```
var el = document.getElementById('myEl');
el.addEventListener('click', function(event){
    alert('clicked!');
})
//Combine with DOM editing
el.addEventListener('mouseover', function(event){
    el.innerHTML('over');
})
```

Event object

You may be wondering what that event parameter is... *An event object is passed to the event handler that describes what happened*. The event object is different depending on the type of event.

Events include:

- target - element where event occurred
- Mouse: clientX, clientY
- Keyboard: keyCode, shiftKey

Event Bubbling

When an event is triggered on an element, it then gets fired on that element's parents, all the way to the top.

- event.target is the element where the event originally occurred
- event.currentTarget is the element running the event handler (this!).

```
<div class="outer">
    <div class="inner">click me</div>
</div>
```

```
document.querySelector('.outer').addEventListener('click', function(e){
    // e represents the event (element that has been clicked)
    console.log(e.target, e.currentTarget);
    //inner, outer
})
```

Exercise & Homework

Let's create a simple todo application

- Show an unordered list of todo's
- Show an input to enter a new todo
- Show a button to add a todo. When the button is clicked:
    - The text from the input box is used to add a list item to the bottom of the list
    - The text from the input box is cleared out.
- When the user clicks on a list item, it is removed

- Extra Credit: - When a list item is clicked, cross it out, then remove it after 1 second.

- Complete the CodeSchool jQuery course

Reading

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

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

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