Advanced JavaScript for Web Sites and Web Applications

JavaScript Recap

`console.log()`

- The `console` object is a *host object*
  - not part of the JavaScript specifications
- We can use it's `log` method to inspect expressions and variables.
- The information displayed will vary, depending on what we pass to it

Viewing the log

- Remember, you can view the `console` via the browser's built-in developer tools
  - Firefox: `Ctrl + Shift + i`
  - Chrome: `Ctrl + Shift + i`
  - Internet Explorer: `F12`
- Or, you can use third party plugins such as Firebug.

Using console.log()

```
var a = 5,
    b = 10,
    c = {
        name: "Gerard",
        height: 188,
        heightInInches: function (){
            return this.height/2.54;
        }
    };
console.log(a);
console.log(a > b);
console.log(c);
console.log(c.heightInInches);
```

`getElementById()`

- Get a reference to an element by its *id* attribute

```
element = document.getElementById(id);
```

- *id* is a string representing the ID of the element we want to select
- It will return a reference to the element
- If no matching ID is found in the DOM, it will return `null`
- More info

Using `getElementById()`

```
<h1 id="header">Hello</h1>
<p>Blah blah blah...</p>
<p>Blah blah blah...</p>
```

```
// Get reference to element with ID of "header"
var target = document.getElementById("header");
// Do things with the element
console.log(target.textContent);
```

querySelectorAll()

- Returns an array-like list of all the elements that match the *CSS selector* passed to it.

```
document.querySelectorAll(selector)
```

- If there are no results, it returns an empty list
- *Selector* must be supported by the browser
  - e.g. CSS3 electors will not work with IE8
- More info

Using `querySelectorAll()`

```
// Get all paragraphs in page
document.querySelectorAll("p");

// Get all paragraphs with class "special"
document.querySelectorAll("p.special");

// Get all paragraphs in div with class "content"
document.querySelectorAll("div.content p");
```

`insertAdjacentHTML()`

- Add HTML to the DOM with `insertAdjacentHTML()`

```
element.insertAdjacentHTML(position, elToAdd);
```

- *position* is relative to *element* and can be:
  - “beforebegin”, “afterbegin”, “beforeend”, “afterend”
- *elToAdd* is a string which will be interpreted as HTML and added to the DOM.
- More info

Using `insertAdjacentHTML()`

```
// Reference to body element
var target = document.body;

// New HTML content to add to DOM
var toAdd = "<p><strong>Hi</strong></p>";

// Insert new content before closing body tag
target.insertAdjacentHTML("beforeend", toAdd);
```

addEventListener()

- Run code in response to browser events:

```
el.addEventListener(eventName, functionName);
```

- *el* is the element to attach the event handler to.
- *eventName* is the type of event to listen for (click, mouseup, mousehover, keyup, swipe etc.)
- *functionName* is the name of a function to run (or an anonymous function).
- More info

`addEventListener()` with named function

```
// The function to run
function sayHello(event) {
    console.log("Hello world!");
}

// Element to listen for clicks on
var trigger = document.getElementById("myButton");

// Adding the listener
trigger.addEventListener("click", sayHello);
```

`addEventListener()` with anonymous function

```
// Element to listen for clicks on
var trigger = document.getElementById("myButton");

// Adding the listener
trigger.addEventListener(
    "click",
    function (event) {
        console.log("Hello world!");
    }
);
```

textContent

- Read or set the text content of an element with `textContent`

```
var element = document.body;

// get text content and store in "orig"
var orig = element.textContent;

// set new text content
element.textContent = "New content!";
```

- More info

Arrays

- Remember…
- Arrays are collections of things:

```
var myArray = ["a", "b", "c", "d"];
```

- Array indexes are sequential numbers:
  - The 1st element's index is 0
  - The last element's index is the length of the array - 1

Arrays - accessing values

- To access something stored in an array, we use it's *index*:

```
var myArray = ["a", "b", "c", "d"];

var firstElement = myArray[0]; // "a"
var thirdElement = myArray[2]; // "c"
```

- To get the number of elements in an array:

```
var totalElements = myArray.length; // 4
```

Iterating an array

- We can take advantage of these sequential indexes and the array's *length* property to create a `for` loop that iterates the indexes of an array
- More info

Iterating an array

```
var myArray = ["a", "b", "c", "d"];
var totalEls = myArray.length;

// Value of "i" increases on each iteration
for (var i = 0; i < totalEls; i++) {

    // Use "i" to get elements by their index
    console.log (myArray[i]);

}
```

`classList`

- Manipulate the HTML *classes* assigned to an element with `classList`

```
var classes = element.classList
```

- `classList` returns an array-like list of the classes assigned to an element
- It also has several useful methods that allow us to query and manipulate the element's classes
  - `add()`, `remove()`, `toggle()`, `contains()`
- More info

Adding a class with `classList`

```
var el = document.getElementById("header");

// add a class to element
el.classList.add("my-class");

// "my-class" will be in list
console.log(el.classList);
```

Removing a class with `classList`

```
var el = document.getElementById("header");

// remove a class from element
el.classList.remove("my-class");

// "my-class" will not be in list
console.log(el.classList);
```

More fun with `classList`

```
var el = document.getElementById("header");

// toggle a class on element
// If element has my-class, it gets removed
// If element doesn't have my-class, it gets added
el.classList.toggle("my-class");

// Check if element has class (returns true/false)
var hasClass = el.classList.contains("my-class");
```

`classList` is clever!

- If you add a class that the element already has, it won't add it again.
- If you try to remove a class that an element doesn't have, it will do nothing (no error)