W03 - DOM - Document Object Model

- allows access to elements, changing order, content, and attributes of elements.

- ☐ Intro to DOM
- ☐ Getting Elements
  - getElementById • getElementByClassName • getElementsByTagName
  - querySelector • querySelectorAll
- ☐ Navigating DOM
- ☐ Getting and Setting an element's attributes
- ☐ Updating the DOM by creating dynamic markup
- ☐ Changing the CSS of an object

HTML as a network of connected nodes
- Everything on a web page is a node
- DOM is language agnostic

GETTING ELEMENTS

assign body element to variable body:

```
const body = document.body;
```

All nodes have a numerical code.

Code	Type
1.	element
2.	attribute
3.	text
4.	comment
5.	body

Legacy DOM Shortcut Methods

Document. body ← return body element

Document. images ← returns a node list of all images in doc

Document. links ← returns a node list of `<a>` and `<area>` elements that have href attribute

Document. anchors ← returns a node list of `<a>` elements with a name att.

Document. forms ← returns a node list of forms in document.

- node lists are array-like objects - can access using bracket notation

```
for (let i=0, max = document.images.length; i< max; i++){
    -- do something using document.images[i] --}
```

Turn a node list into an array:

```
Const imageArray = Array.from (document.images);
const imageArray = [... document.images];
```

W03 - DOM - Document Object Model

Getting An Element By Its ID

```
const h1 = document.getElementById('title');
```

id attributes should be unique

Getting elements by tag

```
const listItems = document.getElementsByTagName('li');
```

↑ This is a node list
If there aren't any items in document with the tag, an empty node list is returned.

Get elements by Class

```
const heroes = document.getElementsByClassName('hero');
```

Query Selectors

document.querySelector() ← use CSS notation to find first element in document that matches a CSS Selector.

document.querySelectorAll() ← returns a node list off all elements in document matching a css selector.

```
const wonderWoman = document.querySelector('li:last-child'); //return only last item
const ul = document.querySelector('ul#roster'); //reference ul element with id of roster
```

Navigating the DOM Tree

based on the reference of a single element, you can walk around the whole tree

.childNodes ← list of all nodes that are children of the node
.firstChild, .lastChild ← return first and last child nodes.
.parentNode ← returns parent of node
.nextSibling .previousSibling ← step to next or previous

Find Value of a Node

```
<li class='hero'> Wonder Woman </li>
const textNode = wonderWoman.firstChild;
```

W03 - DOM - Document Object Model

Getting and Setting Attributes

.getAttribute() ← returns value of the attribute, null if element does not have that attribute

wonderWoman.setAttribute('class', 'villain'); // changes class attribute to villain

wonderWoman.setAttribute('id', 'amazon'); // can also set an attribute or add an attribute it does not already have

Classes of an Element

className property ← use to set directly

wonderWoman.className = 'hero' ← sets class='hero'

Changing className property of an element will overwrite all other classes an element may have.

classList ← list all classes an element may have

wonderWoman.classList.add('warrior'); // adds warrior to class

wonderWoman.classList.remove('warrior'); // removes warrior from class

wonderWoman.classList.toggle('hero'); // adds if doesn't have it or removes if it does.

↑ returns false on removal and true when it adds the class.

wonderWoman.classList.contains('hero'); // returns true if has class false if it doesn't.

Creating Dynamic Mark-up

.createElement() ← takes a tag name as a parameter

.appendChild() ← use to append a child to a parent ← new nodes are always added to the end of any existing child nodes

.createTextNode() ← use to add text

or

.textContent = 'Flash' ← adds and sets text node content

Adding Elements to the page

.appendChild() .insertBefore()

↑ sets to last child in list of nodes

W03 - DOM - Document Object Model

Replacing Elements on a Page

.replaceChild( ) ← takes two arguments
(newText, oldText)

```
const h1 = document.getElementById('title');
const oldText = h1.firstChild;
const newText = document.createTextNode('Justice League');
h1.replaceChild(newText, oldText);
```

innerHTML

heroes.innerHTML ← returns all child elements of heroes.
returns all raw html if there are lots of elements

can be used to write html inside an element.
Useful for replacing/placing large amount of HTML

Live Collections

node lists returned by document.getElementsByClassName() and document.getElementsByTagName() methods are live collections, update without being called again

UPDATING CSS - element node style property can be updated dynamically, to modify presentation of web page.

••• Any css property with dash is written camelCase •••

background-color → backgroundColor

— or use bracket notation —

↳ `superman.style['background-color'] = 'blue';`

disappearing Act

`Superman.style.display = 'none';`

Checking Style Properties - getComputedStyle() only work for inline and JS-set styles

- CSSStyleDeclaration objects
Will cause problems in change

W03 - DOM - Document Object Model

Better to change/add class to an object to change its formatting rather than change its style.

```
superman.classList.add('highlighted');
```

```
.highlighted {
    border: red 2px solid;
}
```