CS/COE 1520
www.cs.pitt.edu/~nlf4/cs1520/
The DOM and event-driven programming
document.write() adds to the HTML being rendered

- Very handy
  - The JS console log is a bit more out of the way to get to
  - Plus this allows us display output to the user via JS!
    - It is a bit unwieldy, though…
      - Newlines added to the document, not the rendered page
      - Need to write HTML to the document
  - How would you apply it to a detailed web page?
    - I.e., not just a blank document
If only we could access specific HTML elements and alter their properties…

- This is exactly the goal of the Document Object Model (DOM)
  - Built up in an ad-hoc manner over the 1990s by Netscape and Microsoft (independently) to help JS interact with the HTML document being rendered
    - Known now as "Legacy DOM", or DOM Level 0
  - First standard (DOM Level 1) published in 1998
    - Followed by DOM Level 2 in 2000, DOM Level 3 in 2004
    - Latest DOM Level 4 recommendation was published in Nov 2015
The DOM

- Consider the following HTML:

```html
<!-- My document -->
<html>
<head>
  <title>My Document</title>
</head>
<body>
  <h1>Header</h1>
  <p>
    Paragraph
  </p>
</body>
</html>
```

What does the DOM do to help us edit this document as its being rendered?
DOM representation
document

- Object representing the document as a whole
- `document.children` provides a list of the Elements that are a direct child of the document
- `document.body` will reference the `<body>` element of an HTML document
- `document.createElement(tagname)` can be used to add create new Elements with a specified `tagname`
  - To be rendered, the newly created Element must be appended to the document as a child of some Node
    - An HTMLElement is an Element
      - An Element is a Node
- `document.getElementById(id)` allows us to quickly locate Elements with a given value for the `id` attribute
DOM Nodes

- **Node.childNodes** will provide a list of the children of a given node
  - Nodes and Elements, unlike document.children
  - A NodeList, not an array!
    - Though it can still be indexed
- **Node.appendChild(node)** adds a new Node into the document
- **Node.removeChild(child)** removes child from the document
- **Node.replaceChild(new_node, old_child)** replaces old_child with new_node in the document
Would have been nice to be able to see the base page and then trigger the alerts somehow…

  ○ Maybe a click

  ○ Or even hovering the mouse over a portion of the page

This is the basic idea of event-driven programming

  ○ The flow of the program is determined by user actions

    ■ Our applications with *listen* for events to occur, and then run specified functions when they do
Either traverse the entire structure or use an ID

CSS has an easy way to select elements from the document
  - CSS selectors!

JQuery is a very popular JS library that provided a way to use CSS selectors to select elements from the document
  - Also abstracted away a lot of DOM code and cross-browser support
  - How is it imported?

```html
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.2/jquery.min.js"></script>
```

</script>
● Including JQuery has a cost
● While almost necessary a few years ago, can be avoided now for more lightweight options

○ `document.querySelector(selector)`
○ `document.querySelectorAll(selector)`
• E.g., `elt.addEventListener('click', clickTheBox, false);`
  ○ useCapture parameter

• Consider table entries (td elements).
  ○ They're contained within table rows
    ■ Which are contained within tables
      ● Which are contained within the body of the document

• What happens when you want to handle click events on the body of the document, a table within that body, and an element within that table?
  ○ What order should the events fire in?
    ■ Use the structure of the DOM to determine!

... What was the third parameter in `addEventListener()`?
useCapture = true

useCapture = false