CS/COE 1520

THE DOM AND
EVENT-DRIVEN PROGRAMMING

JAVASCRIPT AND THE CONSOLE.LOG

JAVASCRIPT AND CONSOLE.LOG

- In the previous lectures we have used JavaScript to log text into the console window
- Users typically don't see the console window
- Instead of using the console, we need to be able to change the page rendered by the web browser…

THE DOCUMENT OBJECT MODEL (DOM)

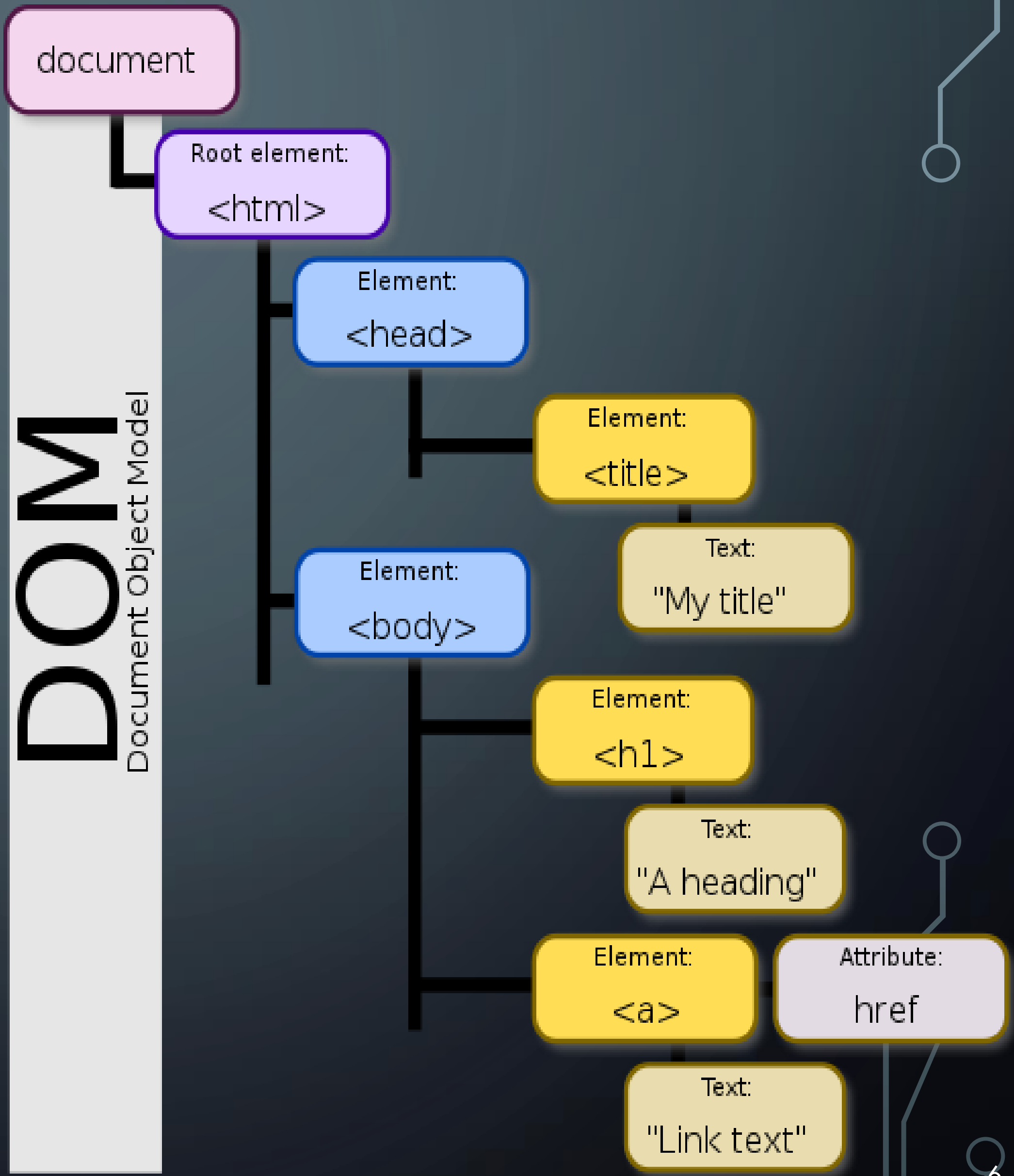
DOCUMENT OBJECT MODEL (DOM)

- HTML is very carefully structured
- Built up without planning because of an immediate need 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:

```
<!-- My document -->
<html>
<head>
  <title>My title</title>
</head>
<body>
  <h1>A heading</h1>
  <a href="www.example.com">
    Link text
  </a>
</body>
</html>
```



DOCUMENT OBJECT MODEL (DOM)

- `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

See examples:

Type document in the console…

children_first_level.html and children_all_levels.html

ELEMENTS VERSUS NODES

- `document.children` **provides a list of the Elements**
  - An element is one specific type of node
  - there are many other types of nodes: text nodes, comment nodes, document nodes,…
- `Node.childNodes` **provides a list of the children of a given node**
  - A node is the generic name for any type of object in the DOM hierarchy
  - A node could be one of the built-in DOM elements such as document or document.body
  - it could be an HTML tag specified in the HTML such as <input> or <p> or it could be a `text` node that is created by the system to hold a block of text inside another element
  - A node is any DOM object.

See Examples:

children_all_levels.html and nodes_all_levels.html

DOCUMENT OBJECT MODEL (DOM)

- `document` ➔ Object representing the document as a whole
- `document.write()` adds to the HTML being rendered
- A `document.write()` called after the page loads will overwrite the current document
- This allows us display output to the *user* via JS!
- Features:
  - 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

DOCUMENT OBJECT MODEL (DOM)

- `document.createElement(tagname)` and `document.appendChild(element)`:
    - can be used to add 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

See example creating_element.html

MODIFYING THE CONTENT OF A DOM NODE

FIRST STEP: FINDING THE NODE/ELEMENT

FINDING ELEMENTS IN THE DOCUMENT

- Either traverse the entire structure or use an ID
- CSS has an easy way to select elements from the document
  - CSS selectors!
- JQuery was a very popular JS library that provided a way to use CSS selectors to select elements from the document
  - Also did away alot of DOM and cross-browser support code
  - But, including JQuery has a cost
    - The jQuery function $() is expensive. Calling it repeatedly is extremely inefficient

FINDING ELEMENTS IN THE DOCUMENT

- While almost necessary a few years ago, can be avoided now for more lightweight/standardized options
  - `document.querySelector(`*`selector`*`)`
  - `document.querySelectorAll(selector)`
- Other options:
  - `document.getElementById(selector)`
  - `document.getElementsByClassName(selector)`

SECOND STEP:
CHANGING THE NODE CONTENT

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

MODIFYING THE CONTENT OF A DOM NODE

- `textContent` property of DOM Nodes can be use
- It is **mutable**, so assigning it a new value will updated the content of a Node in the DOM tree
  - Potentially replacing descendant nodes!
- Similar properties:
  - `innerText`
    - Returns only visible elements
  - `innerHTML`
    - Assigning a value to `innerHTML` will cause it to be rendered as a part of the HTML document

See Example innerHTLM_versus_InnerText.js

LISTENING TO EVENTS

WHEN SHOULD DOM MODIFICATIONS OCCUR?

- Response to a mouse click
- 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 will *listen* for events to occur, and then run specified functions when they do
    - `EventTarget.addEventListener()` can be use to assign a function to execute when an event occurs

See Example js10_more_dom.html

EVENTS AREN'T JUST GENERATED BY USERS

- We can avoid needing a `<script>` tag in the `<body>` of our HTML by listening for the event that indicates the document in loaded, and running a JS function to handle that event
  - `window.addEventListener("load", func, false);`
    - Fired when the whole page has loaded, including all dependent resources such as `window.addEventListener("DOMContentLoaded", func, false);`
    - Stylesheets and images
    - Fired when the initial HTML document has been completely loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading

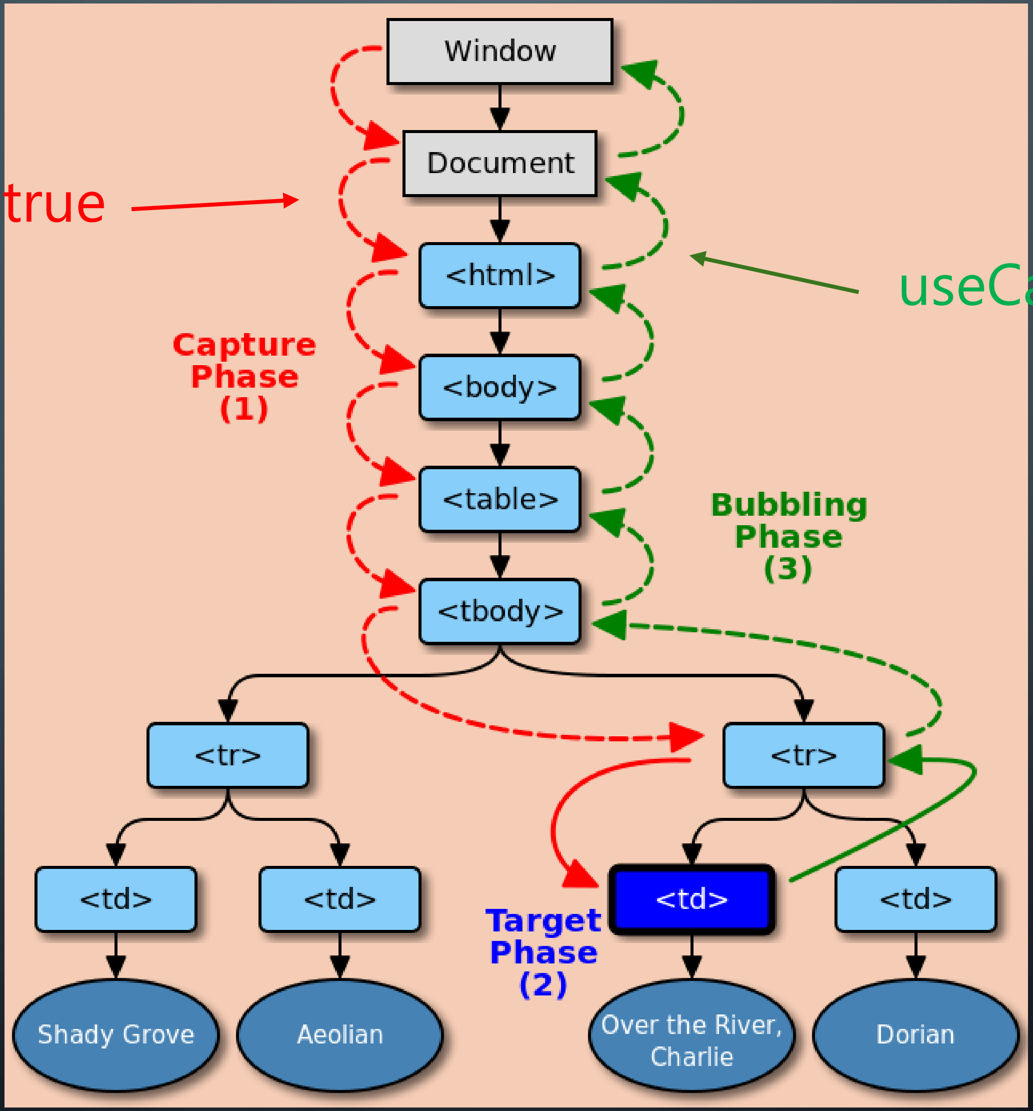
WHAT IS THE THIRD PARAMETER IN `ADDEVENTLISTENER()`?

- The `useCapture` parameter
  - Optional boolean, defaults to `false`
  - `true: bubbling`
  - `false: capturing`
- Event bubbling and capturing are two ways of propagating events that occur in an element that is nested within another element, when both elements have registered a handle for that event.
- Consider table entries (`td` elements).
  - They're contained within table rows
    - Which are contained within table bodies
      - Which are contained within tables
        - Which are contained within the body of the document
    - Use the structure of the DOM to determine!

USECAPTURE TRUE AND FALSE

useCapture = true

useCapture = false



BUBBLING AND CAPTURING EXAMPLES

For bubbling and capturing example, see:

bubbling_capturing_example.html

js12_capture_dom.html

THIS

- We've seen `this` before
  - When a function is called as a constructor (i.e., after `new`), `this` refers to the object being constructed
  - E.g.:

```
function TV(brand, size, injacks, outjacks) {
     this.brand = brand;
     this.size = size;
     this.jacks = new Object();
     this.jacks.input = injacks;
     this.jacks.output = outjacks;
}
```

SIMILAR USE IN OBJECT METHODS

```
function show_properties() {
        document.write("Here is your TV: <br />");
        document.write("Brand: ", this.brand,"<br />");
        document.write("Size: ", this.size, "<br />");
        document.write("Input Jacks: ");
        document.write(this.jacks.input, "<br />");
        document.write("Output Jacks: ");
        document.write(this.jacks.output, "<br />");
}
my_tv.display = show_properties;
```

THIS IN AN EVENT HANDLER

- When a function is used as an event handler, its `this` is set to the element the event fired from

```
function makeRed() {
   this.style.backgroundColor = "#FF0000";
}
let d = document.getElementById("theDiv");
d.addEventListener("click", makeRed, true);
```