

# Module 4-4

DOM

# Domain Object Model

- The Domain Object Model (DOM for short) is a tree representation of all the HTML elements on a given web page.
- Most browsers have a “Developer Tools” interface that allows for quick inspection of a DOM element and how it relates to other elements on the page.
- The focus of today’s lecture is how to use JavaScript to interact with the DOM.

# Chrome Developer Tools Demo

# DOM Elements: ID's and Classes

Let's review id and classes for HTML elements. Consider the following HTML code:

```
<p id='intro'>I dedicate this page to my dog Horace</p>

<p class = 'content'>Some Widgets are Doodads</p>
<p class = 'content'>Some Doodads are Thingamagjigs</p>
<p class = 'content'>All Thingamajigs are Whatchamacallits</p>
```

- The first paragraph is marked with an id - ideally we use an id to uniquely identify one element.
- All other paragraphs are marked with a class - ideally we can apply a class to several elements that we feel share some commonality.


# DOM Elements: Properties

The id and class names are properties of a DOM Object. We have already dealt with a lot of these properties while learning CSS: height, width, color, etc.

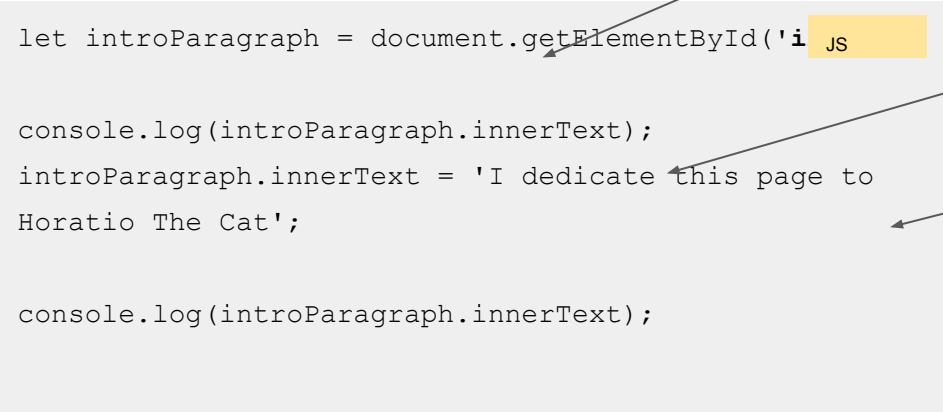
# getElementById

We can use `getElementById` to identify and assign a DOM element to a JavaScript variable. We can then interrogate or change its properties. Consider this example:

```
<body>  
<p id='intro'>I dedicate this page to my dog.</p>  
<script src="thisScript.js"></script>  
</body>
```



```
let introParagraph = document.getElementById('i  
console.log(introParagraph.innerText);  
introParagraph.innerText = 'I dedicate this page to  
Horatio The Cat';  
  
console.log(introParagraph.innerText);
```



- Note that we start off by targeting the intro paragraph, since we know it has an id of intro we can use the `getElementById` method.
- We assigned this DOM object to a variable called `introParagraph`.
- We changed the `innerText` property to contain a different sentence.

# getElementById

- The end result of this example is that the HTML page will have “I dedicate this page to Horatio The Cat”, thus changing the original text.
- There is a similar property called innerHTML, that should be avoided as it allows for injection of unwanted JavaScript content beyond the text.
  - Rule of thumb - if you want to change text, use innerText like we have done here.

# querySelectorAll

- getElementById is useful for identifying one DOM element but sometimes we need to identify several elements in one blow.
- In order to do this, we can leverage querySelectorAll which will return all matching elements and place them in an array.



# querySelectorAll

Let's look at this example again:

```
<p id='intro'>I dedicate this page to my dog Horace</p>  
<p class = 'content'>Some Widgets are Doodads</p>  
<p class = 'content'>Some Doodads are Thingamajigs</p>  
<p class = 'content'>All Thingamajigs are Whatchamacallits</p>
```

HTML

```
let paragraphs = document.querySelectorAll('.content');  
console.log(paragraphs.length);  
  
for (i = 0; i < paragraphs.length; i++) {  
  let paragraph = paragraphs[i];  
  paragraph.style.color = 'blue';  
}
```

JS

browser:

I dedicate this page to my dog.

Some Widgets are Doodads

Some Doodads are Thingamajigs

All Thingamajigs are Whatchamacallits

# querySelectorAll

Here's another example note what we've passed to the querySelectorAll method:

```
<p id='intro'>I dedicate this page to my dog Horace</p>  
<p class = 'content'>Some Widgets are Doodads</p>  
<p class = 'content'>Some Doodads are Thingamagjigs</p>  
<p class = 'content'>All Thingamagjigs are Whatchamacallits</p>
```

HTML

```
let paragraphs = document.querySelectorAll('p');  
console.log(paragraphs.length);  
  
for (i = 0; i < paragraphs.length; i++) {  
  let paragraph = paragraphs[i];  
  paragraph.style.color = 'blue';  
}
```

JS

browser:

I dedicate this page to my dog.  
Some Widgets are Doodads  
Some Doodads are Thingamagjigs  
All Thingamagjigs are Whatchamacallits

# querySelector

Finally, we have `querySelector()` which returns the first element found that matches a given criteria.

```
<p id='intro'>I dedicate this page to my dog Horace</p>  
<p class = 'content'>Some Widgets are Doodads</p>  
<p class = 'content'>Some Doodads are Thingamagjigs</p>  
<p class = 'content'>All Thingamajigs are Whatchamacallits</p>
```

```
let paragraph = document.querySelector('p');  
console.log(paragraphs.innerText);
```

“I dedicate this page to my  
dog Horace”

Let's Try This Out!

# Creating DOM Elements

We can create brand new DOM elements from scratch. Consider the following code:

```
<ul id='theList'>  
  <li>Some Widgets are Doodads</li>  
  <li>Some Doodads are Thingamajigs</li>  
  <li>All Thingamajigs are Whatchamacallits</li>  
</ul>  
<script src="thisScript.js"></script>
```

HTML

A brand new element (a list item) is being created.

We identify the parent.

```
let extraListItem = document.createElement('li');  
extraListItem.innerText = 'All Foos are Bars';
```

JS

```
let parentList = document.getElementById('theList');  
parentList.appendChild(extraListItem);
```

Append the brand new element to the parent.

# Assigning a class to an element

We can create brand new DOM elements from scratch. Consider the following code:

```
<ul id='theList'>
  <li>Some Widgets are Doodads</li>
  <li>Some Doodads are Thingamagjigs</li>
  <li>All Thingamajigs are Whatchamacallits</li>
</ul>
<script src="thisScript.js"></script>
```

HTML

```
.importantStuff {
  color:red;
}
```

CSS

```
let extraListItem = document.createElement('li');
extraListItem.innerText = 'All Foos are Bars';
extraListItem.setAttribute('class', 'importantStuff');
```

```
let parentList = document.getElementById('theList');
parentList.appendChild(extraListItem);
```

browser:

- Some Widgets are Doodads
- Some Doodads are Thingamagjigs
- All Thingamajigs are Whatchamacallits
- All Foos are Bars

Let's Try This Out!