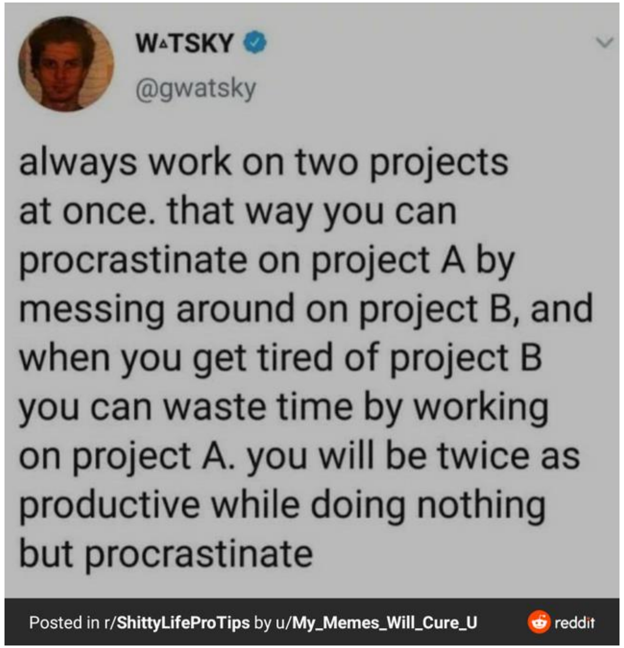

W▵TSKY
@gwatsky

always work on two projects at once. that way you can procrastinate on project A by messing around on project B, and when you get tired of project B you can waste time by working on project A. you will be twice as productive while doing nothing but procrastinate

Posted in r/ShittyLifeProTips by u/My_Memes_Will_Cure_U reddit

Module 3-7

DOM

Objectives

- Difference between the DOM and HTML
- Select elements from the DOM
- Describe the DOM structure
- innerText on HTML elements
- Create new DOM elements
- Traverse the DOM
- Investigate the living DOM in the browser

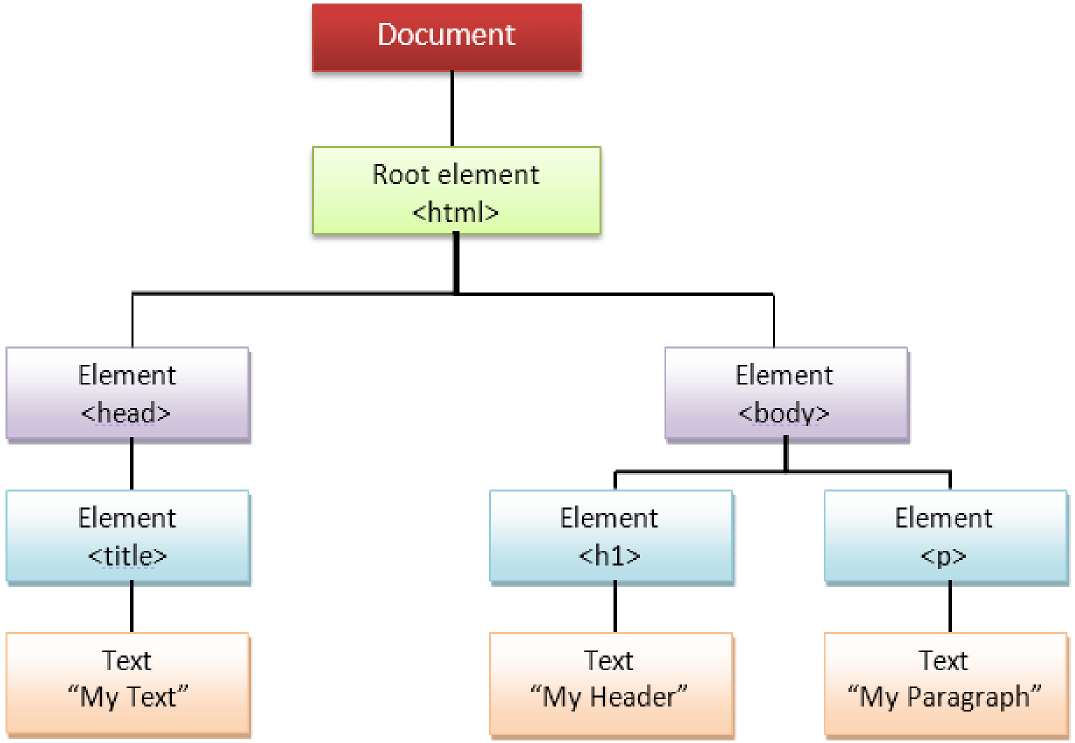
Document Object Model

- The Document 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.

DOM vs. HTML

- The DOM is a model of a document with an associated API for manipulating it.
- HTML is markup language that lets you represent a certain kind of DOM in text.
- DOM is tree model to represent HTML.
- DOM doesn’t always match the HTML source code

DOM


Document
Root element <html>
Element <head>
Element <body>
Element <title>
Element <h1>
Element <p>
Text "My Text"
Text "My Header"
Text "My Paragraph"

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:

HTML

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

JS

```
let introParagraph = document.getElementById('intro');

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 introParagrah.
- 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.
  - innerHTML that takes input from a user sets your page up for XSS
  - Rule of thumb - if you want to change text, use innerText like we have done here.

Cross-Site Scripting attack (XSS)

- Hackers execute malicious JavaScript within a victim’s browser
  - Code is run within user’s browser
  - Code sits on top of legitimate website, tricking browsers into executing malware
- Persistent XSS
- Reflected XSS
- Self XSS
- Blind XSS
- DOM-based XSS

https://sucuri.net/guides/what-is-cross-site-scripting/

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:

HTML

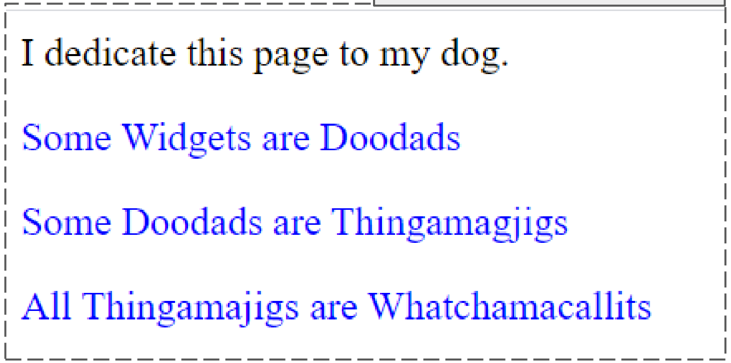
```html
<p id='intro'>I dedicate this page to my dog Horace</p>
<p class = 'content'>Some Widgets are Doodads</p>
<p class = 'content'>Some Doodads are Thingamagjigs</p>
<p class = 'content'>All Thingamajigs are Whatchamacallits</p>
```

JS

```js
let paragraphs = document.querySelectorAll('.content');
console.log(paragraphs.length);

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

browser:



I dedicate this page to my dog.

Some Widgets are Doodads

Some Doodads are Thingamagjigs

All Thingamajigs are Whatchamacallits

querySelectorAll

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

HTML

```html
<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>
```

JS

```javascript
let paragraphs = document.querySelectorAll('p');
console.log(paragraphs.length);

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

browser:

I dedicate this page to my dog.

Some Widgets are Doodads

Some Doodads are Thingamagjigs

All Thingamajigs 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!

value and checked properties

value gets the value from a text field. checked returns status of radio or checkbox elements:

HTML

```html
Name: <input type="text" id="myText" value="Mickey"><br><br>
<form>
  What color do you prefer?<br>
  <input type="radio" name="colors" id="red">
  <label for="red">Red</label><br>
  <input type="radio" name="colors" id="blue">
  <label for="blue">Blue</label>
</form>
```

JS

```javascript
document.getElementById("myText").value = "Johnny Bravo";
document.getElementById("red").checked = true;
```

Using value, set the text field to “Johnny Bravo”

Using checked, set the red box to true (or checked).

Creating DOM Elements

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

HTML

```html
<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>
```

JS

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

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

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

We identify the parent.

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:

HTML

```
<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>
```

JS

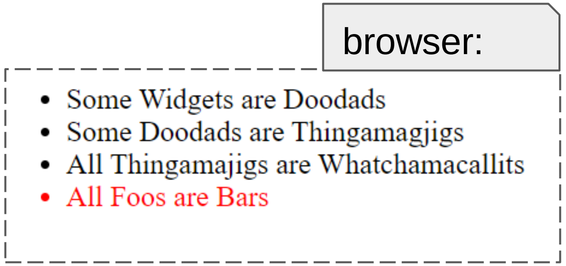
```
let extraListItem = document.createElement('li');
extraListItem.innerText = 'All Foos are Bars';
extraListItem.setAttribute('class', 'importantStuff');

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

CSS

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

browser:



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

Inserting elements into the DOM

- insertAdjacentElement
  - beforeBegin
  - afterBegin
  - beforeEnd
  - afterEnd

Selecting children with children and childNodes

- children
  - Returns an HTML collection, which you can turn into array
  - Returns elements that are children
    - Only contains HTML elements
    - Not text that might be in element
- childNodes
  - Returns a NodeList object that contains all nodes inside element (can also turn into array)
  - Returns nodes that are childnre of element
    - Includes text and comments that are in DOM

parentNode and adjacent elements

- parentNode
    - Returns parent of element
- Adjacent elements
    - nextElementSibling
    - previousElementSibling

Let’s Try This Out!