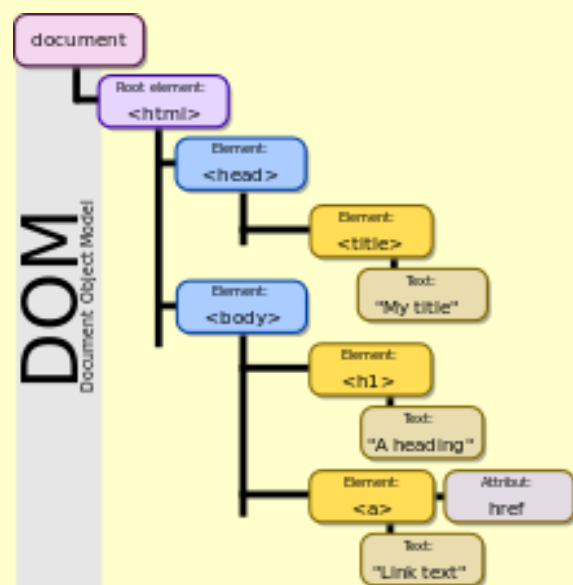


DOM [Document Object Model]



The DOM is essentially a programming interface that browsers use to render source HTML 'sections' as objects that contain all parts of a web page. The DOM can be manipulated dynamically by JavaScript via the process of Event Listening/Handling. A simple event could be a button click. JS can be programmed to listen/handle the click then modifying the DOM, e.g. Hiding a div and displaying a new div. [REFERENCE 1](#) & [REFERENCE 2](#) + [TUTORIAL VIDEO](#)

Functional Programming [Asynchronous]

```

let myVar = function doStuff(stuff) {
  console.log(stuff);
}; //Function expression (function assigned to variable)

function doStuff(age) {
  const INTRO = `Your age is: `;
  function showAge() {
    return INTRO + age;
  } //Closure
  return showAge();
}
doStuff(41); //Calls function with closure. Your age is 41

button.addEventListener('click', () => {
  console.log(`Button clicked.`);
}); //Callback is the anonymous arrow function
  
```

Terms:

- ~ **Pure function:** Doesn't depend on or modify variables out of its scope.
- ~ **Function expression:** Function assigned to variable.
- ~ **Closure:** Inner function with access to outer function data as reference (not value) including parameters.
- ~ **Callback:** Function expression passed as parameter to other function or anonymous arrow function in parameter list.
- ~ **IIFE:** Immediately Invoked Function Expression. Now simply [block scope](#): `{alert(`Hello!`)}; //Please see this caveat though.`

Maps & Sets

```

let myMap = new Map(); //Instantiate a new Map()
myMap.set(73301, 'Austin'); //Key=zip, value=city
console.log(myMap.size); //Print Map() size
console.log(myMap.get(73301)); //Prints Austin
myMap.clear(); //Erase Map()
myMap.delete(73301); //Delete value from key
for (let value of myMap.values()) {
  console.log(value);
} //Print all values in Map(). keys() for keys

let mySet = new Set();
mySet.add('Thaumaturgy'); //Adds new item to set
console.log(mySet.size) //Print Set() size
mySet.clear() //Clears the Set()
mySet.delete('Thaumaturgy'); //Delete entry
Iterate as Map() above
  
```

Maps are similar to arrays except they access data via keys vs. index values. Use **WeakMap()** to store by reference vs. **Map()** which stores by value.

Sets are simply arrays that can't contain duplicate data, but methods are similar to **Map()**.

Random

```
myRand = Math.floor((Math.random() * 20) + 1);
```

Code above generates a random number from 1-20. For 0-20 do this:

```
myRand = Math.floor((Math.random() * 21);
```

Ajax [Client-Side] (Becoming Deprecated to [Fetch API](#))

```

performAjax(requestNum, sendToNode, callback) {
  let bustCache = '?' + new Date().getTime();
  const XHR = new XMLHttpRequest(); //THIS is Ajax!!
  XHR.open('POST', document.url + bustCache, true);
  XHR.setRequestHeader('X-Requested-with', requestNum);
  XHR.send(sendToNode);
  XHR.onload = () => {
    if (XHR.readyState == 4 && XHR.status == 200 && callback) {
      return callback(XHR.responseText);
    } else {
      return `ERROR`;
    }
  };
}
  
```

Simple client method to pass data to Node.js server & handle server response. Use **JSON.stringify()** to send & **JSON.parse()** to receive.

Respond to Client Ajax [Server-Side]

```

if (request.method === 'POST' && request.headers['x-requested-with'] === 'XMLHttpRequest0') {
  const FORMIDABLE = require('formidable');
  let formData = {};
  new FORMIDABLE.IncomingForm().parse(request).on('field', (field, name) => {
    formData[field] = name;
  }).on('error', (err) => {
    next(err);
  }).on('end', () => {
    DATA_HANDLER.addData(formData); //points to external class that writes data to DB
    formData = JSON.stringify(formData);
    response.writeHead(200, {'content-type': 'application/json'});
    response.end(formData);
  });
}
  
```

Simple Node.js routine to receive data from DOM & return results using **JSON.stringify()**.

Asynchronous File I/O [Server-Side]

```
const IO = require('fs'); //Library for file I/O
handleUserData(data, callback) {
  data = JSON.parse(data);
  const FILE_PATH = 'data/users.csv';
  IO.readFile(FILE_PATH, 'utf8', (err, file) => {
    let user = {};
    const COLUMNS = 4;
    let tempArray, finalData = [];
    tempArray = file.split(/\r?\n/); //Remove newlines
    for (let i = 0; i < tempArray.length; i++) {
      finalData[i] = tempArray[i].split(/,/).slice(0, COLUMNS);
    }
    for (let i = 0; i < finalData.length; i++) {
      if (data === finalData[i][0]) {
        user = JSON.stringify({
          'email': finalData[i][0],
          'position': finalData[i][1],
          'lastName': finalData[i][2],
          'firstName': finalData[i][3]
        });
        break;
      } else {
        user = 'false';
      }
    }
    callback(user);
  });
}
```

Local & Session Storage [Client-Side] (Pseudo-browser DB functionality)

```
sessionStorage.setItem('day', document.getElementById('day').value);
sessionStorage.getItem('day'); //Returns value stored at 'day' key
localStorage.setItem('day', document.getElementById('day').value);
localStorage.getItem('day'); //Returns value stored at 'day' key
removeItem('key'); //Remove 1 item
clear(); //Erase all storage
```

localStorage is non-volatile, **sessionStorage** is volatile.
Stores data in Map (key/value pair) format.

Module Support [Server-Side ([for now](#))]

```
module.exports = ClassName; //use @ bottom of class file to export to
instantiator/consumer of this class
```

```
const CLASS_NAME = require('./ClassName'); //Use @ top of consumer to
import external class file
```

Modules is simply the mechanism to keep you class files separate and pull them together with **module.exports** so foreign classes can instantiate objects of each other.

DOM Event Listening/Handling [Client-Side]

```
document.getElementById('continue').addEventListener('click', () => {
  this.performAjax('XMLHttpRequest0',
    JSON.stringify(document.getElementById('getEmail').value), (response) => {
      if (response === 'false') {
        alert('You must provide your proper email address to continue.');
      } else {
        this.user = JSON.parse(response);
        document.getElementById('login').style.display = 'none';
        document.getElementById('log').style.display = 'block';
        document.getElementById('name').innerHTML = `${this.user.firstName}
          ${this.user.lastName}`;
      }
    }
  ));
```

Simple method that demonstrates **addEventListener()** technique for listening for DOM events & **anonymous arrow function callback** for handling event. Events list [HERE](#).

Important DOM stuff:

[http request](#) ~ client -> server. Use GET to receive from server, use POST to transmit to server.

[Node.js http response](#) ~ server -> client. Use **writeHead()**, **write()**, & **end()** to return data to client.

document.getElementById('wiggles') //Affect one element with tag attribute id value 'wiggles'
document.getElementsByTagName() //Affect all elements of tag type. e.g. spans, divs
document.getElementsByName('woot') //Affect all elements with the same tag attribute name woot
document.getElementById('wiggles').value //Sets or gets element value. Mostly used with forms

Miscellaneous

Regular Expressions: Technique for querying data for patterns, i.e searching for matches in a string. e.g.: `if(/^cat/).test(myVar)) {}` //true if word starts with 'cat'

VERY useful for validating. Must-have skill for any programming career. [Regex tester](#)

Date/Time: `let myDate = new Date();` //Creates new date object

`myDate.getFullYear();` //Returns 4-digit year

`myDate.getMonth();` //Returns 0-11 month

`myDate.getHours();` //Returns 0-23 hour

Hiding/Showing Elements:

```
document.getElementById('woot').style.visibility = 'hidden'; //Invisible element
document.getElementById('woot').style.visibility = 'visible'; //Shows element
document.getElementById('woot').style.display = 'none'; //Essentially removes element
document.getElementById('woot').style.display = 'block'; //Shows element
```

Proper way to start client-side JavaScript:

```
window.addEventListener('load', () => {
  new main(); //Instantiate new object of your main.js class
});
```

Find elements in the DOM:

Use **querySelector()** & **querySelectorAll()** to find elements by CSS selector (id or class).
 e.g.: `let element = document.querySelector(".row");` //returns first div with class=row