READING W04

I ALREDY TESTED ALL THE CODE BELOW!!!

1 - How to create an Object :

const superman = {

name: 'Superman',

'real name': 'Clark Kent',

height: 75,

weight: 235,

hero: true,

villain: false,

allies: ['Batman','Supergirl','Superboy'],

fly() {

return 'Up, up and away!';

}

};

I can add many elements that I want to!!!

2 – Simple example how to create a web page:

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Justice League</title>

</head>

<body>

<header>

<h1 id='title'>Justice League</h1>

</header>

<ul id='roster'>

<li class='hero'>Superman</li>

<li class='vigilante hero' id='bats'>Batman</li>

<li class='hero'>Wonder Woman</li>

</ul>

</body>

</html>

It’s not a big deal!!!!!

### 3 - Getting and element By ID

The getElementById() method does exactly what it says on the tin. It returns a reference to the element with a unique id attribute that is given as an argument. For example, we can get a reference to the <h1> heading element with the id of 'title' in the 'heroes.html' page by writing this in the console:

const h1 = document.getElementById('title');

*Every id attribute should be unique to just one element*(Make sure you follow this rule – it's not enforced by the HTML parser, but odd things can happen in your code if you have more than one element with the same ID). This method will return a reference to the unique element with the ID provided as an argument. For this reason, it’s a very quick way of finding elements in a document. It’s also supported in all the major browsers, and is probably the most commonly used method of accessing elements on a web page.

If no element exists with the ID provided, null is returned. --- VERY IMPORTANT!!!!!

### 4 - The Date Object

Date objects contain information about dates and times. Each object represents a single moment in time.

#### **Constructor Function**

A constructor function is used to create a new date object using the new operator:

const today = new Date();

The variable today now points to a Date object. To see what the date is, we use the toString() method that all objects have:

today.toString();

<< 'Tue Feb 14 2017 16:35:18 GMT+0000 (GMT)'

------ VERY COOL THING!

I ALREDY TESTED THIS CODE:

const game = {

start(quiz){

this.questions = [...quiz];

this.score = 0;

// main game loop

for(const question of this.questions){

this.question = question;

this.ask();

}

// end of main game loop

this.gameOver();

},

ask(){

const question = `What is ${this.question.name}'s real name?`;

const response = prompt(question);

this.check(response);

},

check(response){

const answer = this.question.realName;

if(response === answer){

alert('Correct!');

this.score++;

} else {

alert(`Wrong! The correct answer was ${answer}`);

}

},

gameOver(){

alert(`Game Over, you scored ${this.score} point${this.score !== 1 ? 's' : ''}`);

}

}

### 5 - GET Elements By Their Class Name ---- COOL THING

 getElementsByClassName() will return a live node list of all elements that have the class name that is supplied as an argument. For example, we can return a collection of all elements with the class of 'hero' using the following code:

const heroes = document.getElementsByClassName('hero');

Note that, in this case, it is exactly the same collection that was returned when we found all of the list items previously.

There are three elements on the page that have the class name of hero , which we can test by querying the length property:

heroes.length;

<< 3

Note that if there are no elements with the given class, an HTML collection is still returned, but it will have a length of 0:

document.getElementsByClassName('villain').length;

<< 0

 document.getElementsByClassName is supported in all the major modern browsers, but was only supported in Internet Explorer 9 and later.

6 – NODE:

### Navigating the DOM Tree

Node objects have a number of properties and methods for navigating around the document tree. Once you have a reference to an element, you can walk along the document tree to find other nodes.

The childNodes property is a list of all the nodes that are children of the node concerned. The following example will return all the child nodes of the element with an id attribute of roster :

const heroes = document.getElementById('roster');

heroes.childNodes

<< NodeList [#text "

", <li class="hero">, #text "

", <li id="bats">, #text "

", <li class="hero">, #text "

", <li class="hero">, #text "

IMPORTANT: Note that the childNodes property returns*all*the nodes that are children of an element. This will include any text nodes, and since whitespace is treated as a text node, there will often be empty text nodes in this collection.

## 7 - **Setting An Element’s Attributes**

The setAttribute can change the value of an element’s attributes. It takes two arguments: the attribute that you wish to change, and the new value of that attribute.

For example, if we wanted to change the class of the element in the wonderWoman variable to 'villain', we could do so using this code:

wonderWoman.setAttribute('class', 'villain');

<< undefined

wonderWoman.getAttribute('class');

<< "villain"

IMPORTANT: If an element does not have an attribute, the setAttribute method can be used to add it to the element. For example, we can add an id of 'amazon' to the wonderWoman element:

wonderWoman.setAttribute('id','amazon');

wonderWoman.getAttribute('id');

<< 'amazon'

#### 8 - **Creating An Element**

The document object has a createElement() method that takes a tag name as a parameter and returns that element. For example, we could create a new list item as a DOM fragment in memory by writing the following in the console:

const flash = document.createElement('li');

THIS ELEMENT IS EMPTY AT THIS TIME!!!!!!!!!!

**--- IF I WANT TO ADD AN ELEMENTE I NEED TO CREAT A NODE!!!**

**Creating a Text Node**

A text node can be created using the document.createTextNode() method. It takes a parameter, which is a string containing the text that goes in the node. Let's create the text to go in our new element:

const flashText = document.createTextNode('Flash');

Now we have an element node and a text node, but they are not linked together ― we need to append the text node to the paragraph node.

#### **A Function To Create Elements**

When we created our new list item element, all we specified was the type of tag and the text inside it. These will form the parameters of our function. The function will then perform the two steps we used to create the new element, and then return that element:

function createElement (tag,text) {

const el = document.createElement(tag);

el.textContent = text;

return el

}

#### **Adding Elements to the Page**

We have already seen the appendChild() method. This can be called on a node to add a new child node. The new node will always be added at the end of any existing child nodes. The following example will add the flash element we created above to the end of the <ul> element, as shown below:

heroes.appendChild(flash);

### Replacing Elements on a Page

### --- I didn’t know that I could replace an element

The replaceChild() method can be used to replace one node with another. It’s called on the parent node and has two parameters: the new node and the node that is to be replaced. For example, if we wanted to change the content of the <h1> tag that makes the title of the page, we could replace the text node with a new one, like so:

const h1 = document.getElementById('title');

const oldText = h1.firstChild;

const newText = document.createTextNode('Justice League of America');

h1.replaceChild(newText,oldText);

### 9 - DOM Manipulation Example.

A good example of how libraries can help save time is in DOM manipulation. The DOM API provides all the tools required to manipulate the DOM, but some can be verbose and take several lines of code to attain even the most basic of tasks.

For example, if we wanted to add a class to a paragraph element referenced by the variable para , then append another paragraph on the end, we could do it using the following:

para.classList.add('important');

const newPara = document.createElement('p');

newPara.textContent = 'Another Paragraph';

para.appendChild(newPara);

Yet by using the jQuery library, we can achieve the same result using a single line of code:

$(para).addClass('important').append('<p>Another Paragraph</p>');

--- I need to study more about it !!!!!!

HOW import code -file!!!

For example, a very simple Pi module would have the following code saved in a file called 'pi.js':

export const PI = 3.1415926;

This would then be imported into your main JavaScript file, main.js using the following code:

import { PI } from './pi.js';

This would then allow you to use the variable PI inside the main.js file.

Functions can also be exported from a module. For example, we could create a library for our stats functions that we used earlier:

function square(x) {

return x \* x;

}

function sum(array, callback) {

if(callback) {

array = array.map(callback);

}

return array.reduce((a,b) => a + b );

}

function variance(array) {

return sum(array,square)/array.length - square(mean(array))

}

function mean(array) {

return sum(array) / array.length;

}

export {

variance,

mean

}