Module 3

DOM

JAVA – MODULE 3, DAY 8

ELEVATE YOURSELF

Review

RECAP

JS DOC

Documentation
Function Comments - JSDoc

Comments on functions are typically integrated into the IDE and are used to create documentation of your code for other programmers to use. They follow a standard format called JSDoc.

```
/**
 * Takes two numbers and returns the product of
 * those two numbers.
 *
 * Will return NaN if exactly two numbers are not
 * given.
 *
 * @param {number} multiplicand a number to multiply
 * @param {number} multiplier a number to multiply by
 * @returns {number} product of the two parameters
 */
```

TECH ELEVATOR

Review

FUNCTION DEFINITION

```
1 // Named Function
2 function multiplyByTwo(value) {
3   return value * 2;
4 }
5
6 // Named Function as a variable
7 const namedFunction = function multiplyByTwoInline(value) {
8   return value * 2;
9 }
10
11 // Anonymous Function
12 const anonymousMultiplyByTwo = function(value) {
13   return value * 2;
14 }
15
16 // Arrow Function
17 const multArrowFunction = (value) => {
18   return value * 2;
19 }
20
21 // Arrow Function (single line)
22 const singleLineArrowFunction = (value) => value * 2;
```

Review

FUNCTION PARAMETERS

Function parameters
Parameters are variables that can provide input values to a function. When functions are created, parameter lists indicate what inputs the function can accept.

Optional parameters
Parameters in JavaScript are always optional. So what if a value isn't provided for a parameter when calling a function? if you don't assign a value to a variable, the variable is set to undefined.

Default parameter values
In some cases, there's a reasonable default value that you can use if a parameter value isn't supplied. You can use default parameters to make your functions more robust and useful in varying scenarios.

ARRAY FUNCTIONS

Array functions using anonymous functions

Arrays in JavaScript have many useful functions themselves that use anonymous functions.

forEach()

Performs like a for loop, running a passed in anonymous function for every element of an array.

```
let numbers = [1, 2, 3, 4];

numbers.forEach( (number) => {
    console.log(`This number is ${number}`);
});
```

Review

ARRAY FUNCTIONS

find()

The find() method returns the value of the first element in the provided array that satisfies the provided testing function. If no values satisfy the testing function, undefined is returned.

```
1 const array1 = [2, 12, 16, 181, 59];
2
3 const found = array1.find(element => element > 14);
4
5 console.log(found);
6 // expected output: 16
7
```

Review

ARRAY FUNCTIONS

findIndex()

The findIndex() method returns the index of the first element in the array that satisfies the provided testing function. Otherwise, it returns -1, indicating that no element passed the test.

```
1 const array1 = [2, 9, 15, 49, 44];
2
3 const isLargeNumber = (element) => element > 13;
4
5 console.log(array1.findIndex(isLargeNumber));
6 // expected output: 2
7
```

ARRAY FUNCTIONS

filter()

The filter() method creates a new array with all elements that pass the test implemented by the provided function.

```
1 const words = ['Neo', 'Morpheus', 'Matrix', 'Trinity', 'Apoc', 'Switch'];
2
3 const result = words.filter(word => word.length > 6);
4
5 console.log(result);
6 // expected output: Array ["Morpheus", "Trinity"]
7
```

ARRAY FUNCTIONS

map()

The map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

```
1 const array1 = [1, 4, 9, 16];
2
3 // pass a function to map
4 const map1 = array1.map(x => x * 2);
5
6 console.log(map1);
7 // expected output: Array [2, 8, 18, 32]
8
```

ARRAY FUNCTIONS

reduce()

The reduce() method executes a user-supplied “reducer” callback function on each element of the array, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value.

```
1  const array1 = [1, 2, 3, 4];
2  const reducer = (previousValue, currentValue) => previousValue + currentValue;
3
4  // 1 + 2 + 3 + 4
5  console.log(array1.reduce(reducer));
6  // expected output: 10
7
8  // 5 + 1 + 2 + 3 + 4
9  console.log(array1.reduce(reducer, 5));
10 // expected output: 15
11
```

TECH ELEVATOR

Lecture

AGENDA

- HTML vs DOM
- DOM in Browser
- getElementById / querySelector
- InnerText / InnerHTML
- Create / Insert Element
- Traversing the DOM

TECH ELEVATOR

Lecture

DOCUMENT OBJECT MODEL (DOM)

DOCUMENT OBJECT MODEL (DOM)

The DOM

DOM stands for the Document Object Model. It's the browser's internal representation of the structure of the current web page.

The DOM is an internal data structure that browsers use to represent the structure and content of a web page. When the browser loads an HTML document, it needs to translate that into something that it can use to draw a graphical representation of the page.

Understanding the DOM is important. HTML is static. It's only read once when the page loads, and then it's converted into a DOM. CSS and JavaScript perform their tasks using the DOM, not the static HTML.

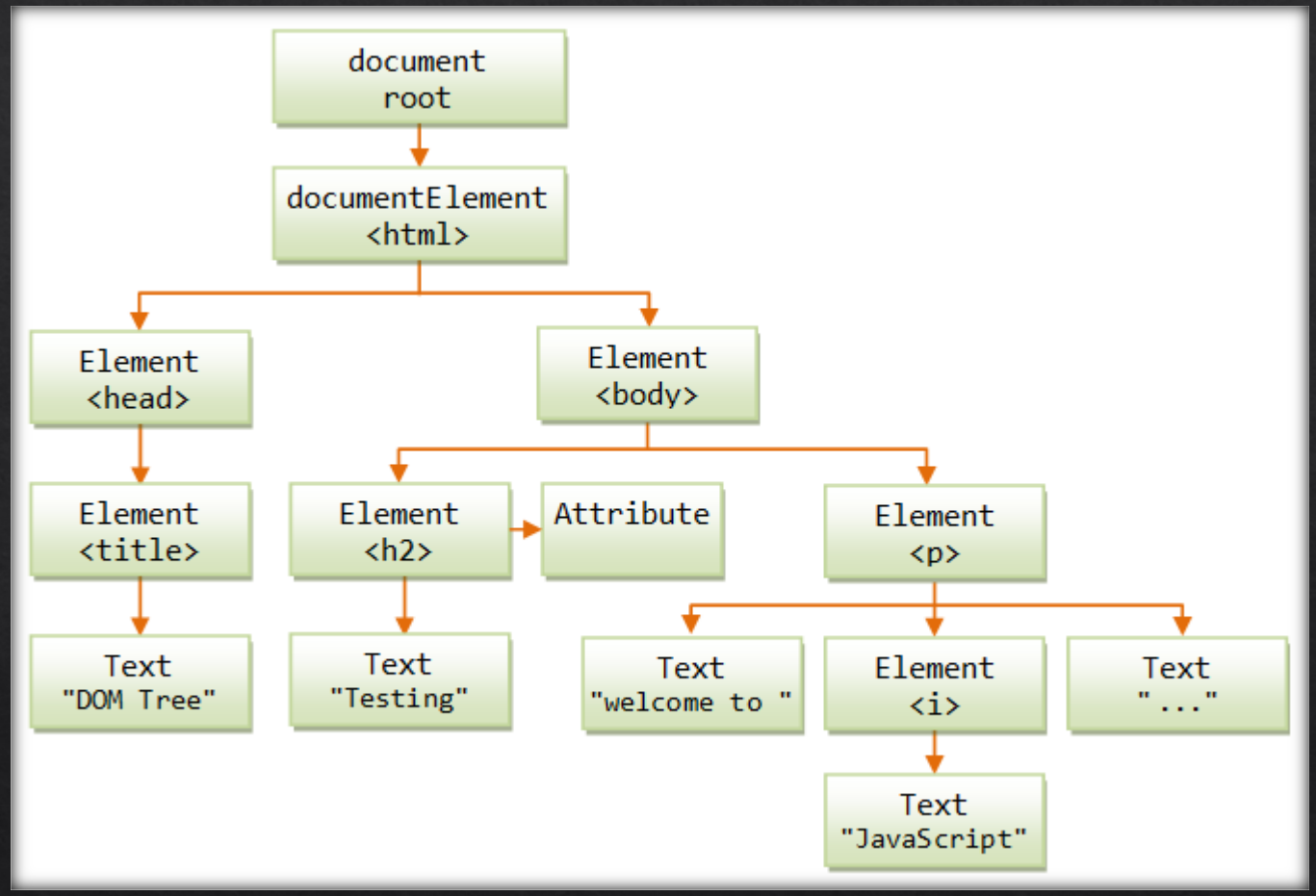
TECH ELEVATOR

HTML

```
<nav style="background-color: #dcdcdc;">
    <div class="container">
        <div class="too-many-divs">
            <div class="navbar-group">
                <ul class="navbar-list">
                    <li class="nav-item"><a href="/Home">Home</a></li>
                    <li class="nav-item"><a href="/Products">Products</a></li>
                    <li class="nav-item"><a href="/Services">Services</a></li>
                </ul>
            </div>
            <div class="navbar-group">
                <ul class="navbar-list"...>
            </div>
        </div>
    </div>
</nav>
```

Lecture

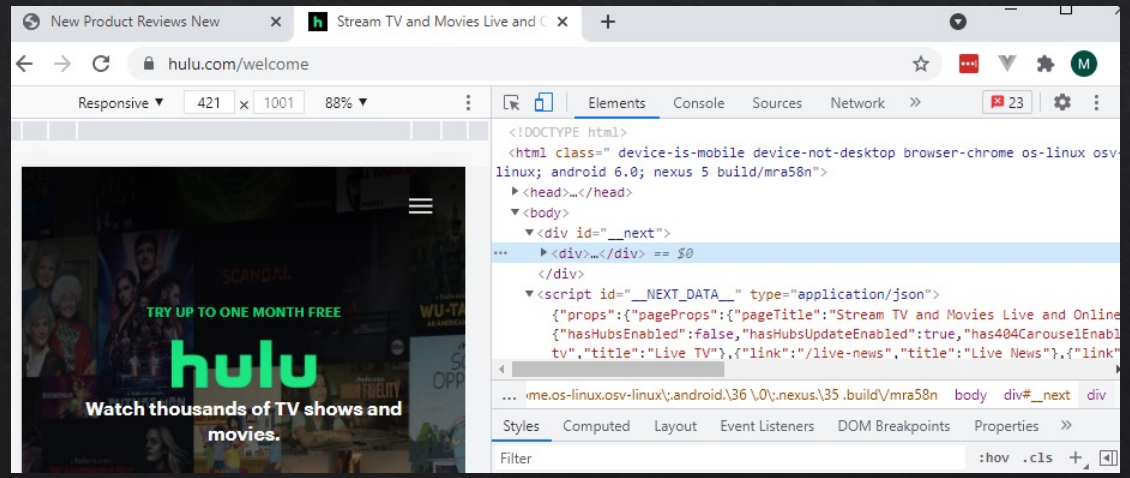
DOM


document
root
documentElement
<html>
Element
<head>
Element
<body>
Element
<title>
Element
<h2>
Attribute
Element
<p>
Text
"DOM Tree"
Text
"Testing"
Text
"welcome to "
Element
<i>
Text
"..."
Text
"JavaScript"

CHECKING THE DOM

To check the DOM, open the developer tools in Firefox or Chrome and go to the Inspector (Firefox) or the Elements (Chrome) tab. This shows an HTML-like view of the DOM as it is at that moment. If you change any JavaScript or CSS in your code, you'll see those changes happen immediately in that view.

This is the best way to understand how your browser interprets the HTML source code and how it changes due to live user interaction.



Lecture

TECH ELEVATOR

Lecture

QUERYING THE DOM

getElementById

Lecture

getElementById()

```
const element = document.getElementById('someElementId');
```

This function gets a single HTML element from the DOM and returns a reference to it.

querySelector

querySelector()

```
const element = document.querySelector('a.active');
```

For selecting single elements that don't have an ID. querySelector() takes a standard CSS selector and returns the first element it finds that matches that selector.

querySelectorAll

querySelectorAll()

```
const paragraphs = document.querySelectorAll('p');
```

If you want to get all of the list items, you can use querySelectorAll() instead. This returns a NodeList of all the elements, which you can use as an array.

TECH ELEVATOR

Lecture

MANIPULATING THE DOM

innerText

innerText

```
1 const element = document.querySelector('#someElementId');
2
3 element.innerText = 'This is the new content of the element';
```

Be cautious of what elements you use innerText on. Because you can use it on any element—even ones that don't normally have their own text such as ul—it'll overwrite child elements if there are any.

innerHTML

Lecture

innerHTML

```
1 const element = document.getElementById('someElementId');
2
3 element.innerHTML = '<p>This could be a security issue!</p>';
```

Danger: Be careful when using innerHTML

Lecture

classList

classList.add(); classList.Remove();

```
1 const el = document.getElementById('someElementId');
2
3 el.classList.add('shiny');
4 el.classList.remove('old');
```

TECH ELEVATOR

Lecture Code

SETTING TEXT, LET'S CODE!

TECH ELEVATOR

ADDING TO THE DOM

Lecture Code

createElement

createElement()

```
1 const myDiv = document.createElement('div');
```

appendChild

Lecture

appendChild()

```
1 const container = document.getElementById('someElementId');
2
3 // Create and populate the element. It does not yet exist in the DOM
4 const el = document.createElement('h1');
5 el.innerText = 'Hello JavaScript';
6
7 // Actually add the element to the DOM as the last child of container
8 container.appendChild(el);
```

TECH ELEVATOR

Lecture Code

ADDING REVIEWS, LET'S CODE!

insertAdjacentElement

insertAdjacentElement()

```
targetElement.insertAdjacentElement(position, element);
```

insertAdjacentElement

insertAdjacentElement()

```
targetElement.insertAdjacentElement(position, element);
```

Parameters

`position`

A `DOMString` representing the position relative to the `targetElement`; must match (case-insensitively) one of the following strings:

- `'beforebegin'`: Before the `targetElement` itself.
- `'afterbegin'`: Just inside the `targetElement`, before its first child.
- `'beforeend'`: Just inside the `targetElement`, after its last child.
- `'afterend'`: After the `targetElement` itself.

`element`

The element to be inserted into the tree.

insertAdjacentElement

insertAdjacentElement()

```
targetElement.insertAdjacentElement(position, element);
```

Parameters

position

A `DOMString` representing the position relative to the `targetElement`; must match (case-insensitively) one of the following strings:

- `'beforebegin'`: Before the `targetElement` itself.
- `'afterbegin'`: Just inside the `targetElement`, before its first child.
- `'beforeend'`: Just inside the `targetElement`, after its last child.
- `'afterend'`: After the `targetElement` itself.

element

The element to be inserted into the tree.

```html
<div>Some Other Element</div>

<!-- beforeBegin -->
<div id="container">
  <!-- afterBegin -->

  <div>Some child</div>

  <!-- beforeEnd -->
</div>
<!-- afterEnd -->

<div>Some Other Element</div>
```

cloneNode()

cloneNode()

The cloneNode() method of the Node interface returns a duplicate of the node on which this method was called. Its parameter controls if the subtree contained in a node is also cloned or not.

```
let p = document.getElementById("para1")
let p_prime = p.cloneNode(true)
```

Warning: cloneNode() may lead to duplicate element IDs in a document!

Lecture

TRAVERSING THE DOM

There may be times when you don't know which selector to choose an element with, or you need to work with a specific child of an element and need to loop through or walk through a list of elements. For this, you can use some element properties to get an element's children or parent.

```
let todoList = document.getElementById('todo-list');
```

You can get all of its immediate children elements through the children property:

```
let todoItems = todoList.children;
```

children returns an HTMLCollection object, which you can turn into a real array with access to map, forEach, and all the other array functions with this:

```
let todoItemsArray = Array.from(todoList.children);
```

Lecture

TRAVERSING THE DOM

You can also get children by calling childNodes:

```
let todoNodes = todoList.childNodes;
```

This returns a NodeList object that contains all the nodes inside that element. You can also pass this to Array.from() to get a normal JavaScript array.

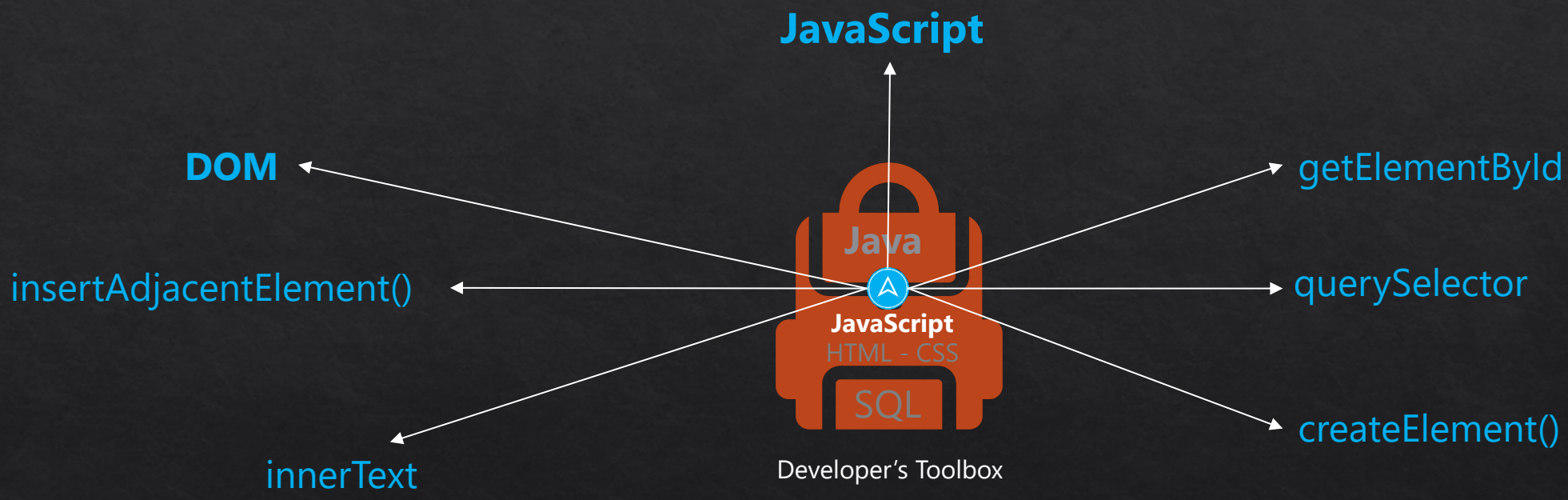
children returns elements that are children of this element. That means that it only contains other HTML elements and not the text that might be in the element.

childNodes returns nodes that are children of this element. That includes text (including whitespace) and comments that are in the DOM.

TECH ELEVATOR

Module 3
DOM

New Tools! Let's Code!



Exercise HW