Chapter 3.2: JavaScript DOM

Welcome to the next step in your JavaScript journey! This tutorial dives into the Document Object Model (DOM), which is like a map of your web page that lets JavaScript interact with it—think changing text, adding buttons, or responding to clicks. We'll go through each topic step-by-step, combining related ideas (like finding elements and changing them) to make it easy to follow. You'll use a simple HTML file to try out the examples, either in your browser's console (press F12 or Ctrl+Shift+I, then click Console) or in a file like the one below. Let's make your web pages come alive!

Here's the HTML structure we'll use for our examples:

```
<!DOCTYPE html>
<html>
<head>
    <title>DOM Tutorial</title>
    <style> .highlight { color: red; } </style>
</head>
<body>
    <h1 id="title">Hello DOM</h1>
    <p class="text">Paragraph 1</p>
    <p class="text">Paragraph 2</p>
    <ul>
        <li>First item</li>
        <li>Last item</li>
    </ul>
    <a href="https://example.com">Link</a>
    <img src="image.jpg" alt="Image">
    <form id="myForm">
        <input type="text" id="name" value="Enter name">
        <button type="submit">Submit</button>
    </form>
    <script>
        // Your JS code here
    </script>
</body>
</html>
```

Put your JavaScript code in the `<script>` tag or a separate `.js` file. Let's get started and have fun exploring the DOM!

1. Introduction to DOM Elements with JavaScript

The DOM is like a tree where every HTML tag (like `<h1>` or `<p>`) is a branch or leaf (called a node). JavaScript can use this tree to find, change, or interact with parts of your page, making it dynamic and interactive.

- **Root Nodes**: `document` (the whole page), `document.body` (the `<body>` part), `document.head` (the `<head>` part).
- **Element Types**: Tags like `<div>`, `<p>`, or `<button>` become objects you can work with.

Example

```
console.log(document);  // The entire document object
console.log(document.body);  // Body element
```

Explanation: This code is like peeking at the blueprint of your web page! It logs the entire `document` (the whole page's structure) and the `<body>` element to the console. Save the HTML above in a file (e.g., `dom.html`), open it in a browser, press F12 to open the console, and type this code. You'll see the `document` and `<body>` objects. It's like saying, "Show me the whole page and its main content area!" Try it and see what your page's structure looks like.

Tip: Always work with the DOM after the page loads (see DOMContentLoaded in section 7).

2. Accessing DOM Elements

To change or interact with parts of your page, you first need to find them. JavaScript gives you several ways to grab elements, like finding a book in a library by its ID or category.

- `getElementById('id')`: Finds one element by its unique ID.
- `getElementsByClassName('class')`: Gets a list (HTMLCollection) of elements with the same class.
- `getElementsByTagName('tag')`: Gets a list of elements by tag name (e.g., all `<p>` tags).
- `querySelector('selector')`: Finds the first element matching a CSS selector (like `#id` or `.class`).
- `querySelectorAll('selector')`: Gets all matching elements as a NodeList.

Some lists update automatically (live), while others are static.

Examples of Accessing Elements, Including Multi-Components

```
// Single element by ID
let title = document.getElementById('title');
console.log(title);  // <h1 id="title">Hello DOM</h1>

// Multiple by class
let paragraphs = document.getElementsByClassName('text');
console.log(paragraphs[0]);  // First <p>
for (let p of paragraphs) {
    console.log(p.textContent);  // Logs text of each
}

// By tag
let listItems = document.getElementsByTagName('li');
console.log(listItems.length);  // 2

// Query selectors (more flexible)
let firstP = document.querySelector('.text');  // First match
let allP = document.querySelectorAll('.text');  // All matches
console.log(allP);  // NodeList of paragraphs
```

Explanation: This code is your treasure map to find elements on the page! First, we grab the `<h1>` with `id="title"` and log it. Then, we find all `<p>` elements with `class="text"`, log the first one, and use a loop to print their text ("Paragraph 1" and "Paragraph 2"). We also count `<li>` tags (there are 2) and use `querySelector` to get the first `.text` paragraph and `querySelectorAll` to get all of them. Try this in your HTML file's `<script>` tag or console: log `title.textContent` or change the loop to add "Hi!" to each paragraph's text. It's like picking specific items from your page to play with!

Tip: `querySelectorAll` is super powerful for tricky selectors like `'.text:nth-child(2)'` to grab specific elements.

3. Navigating DOM: firstChild, lastChild, childNodes, etc.

Once you've found an element, you can explore its "family" (like its children or parents) to move around the DOM tree.

- `firstChild`: The first child node (might be text or whitespace).
- `lastChild`: The last child node.
- `childNodes`: All child nodes (including text and elements) as a NodeList.
- Others: `parentNode`, `nextSibling`, `previousSibling`.

For just elements (skipping text nodes): `firstElementChild`, `lastElementChild`, `children`.

Examples

```
let ul = document.querySelector('ul');
console.log(ul.firstChild);  // Might be text node (whitespace)
console.log(ul.firstElementChild.textContent);  // 'First item'

console.log(ul.lastElementChild.textContent);  // 'Last item'

let children = ul.children;  // HTMLCollection of <li>s
for (let child of children) {
    console.log(child.textContent);
}

let allNodes = ul.childNodes;  // Includes text nodes between <li>s
console.log(allNodes.length);  // More than 2 if whitespace
```

Explanation: Think of the DOM as a family tree! This code finds the `<ul>` (unordered list) and checks its kids. `firstChild` might log a text node (like whitespace between tags), but `firstElementChild.textContent` gets the text of the first `<li>` ("First item"). We also get the last `<li>`'s text ("Last item"), loop through all `<li>` elements to print their text, and count all nodes (including whitespace). Try this in your HTML file: add a new `<li>` to the `<ul>`, run the code, and see how the count changes. It's like exploring a treehouse to find all the rooms!

4. Accessing and Modifying Element Properties

Elements have properties (like their ID or text) that you can read or change, like editing a label on a box.

- `id`: The element's ID.
- `className`: Its class(es) as a string.
- `innerHTML`: The HTML inside (can include tags).
- `textContent`: Just the plain text (no tags).
- `href`: For links.
- `src`: For images/scripts.
- `value`: For form inputs.

Examples

```
let link = document.querySelector('a');
console.log(link.href);  // 'https://example.com'
link.href = 'https://new.com';  // Change link

let input = document.getElementById('name');
console.log(input.value);  // 'Enter name'
input.value = 'New value';  // Update input

let title = document.getElementById('title');
console.log(title.textContent);  // 'Hello DOM'
title.innerHTML = '<em>Updated</em> Title';  // Adds HTML

console.log(title.className);  // '' (empty)
title.className = 'highlight';  // Adds class
```

Explanation: This code lets you peek inside and tweak elements! We find the `<a>` (link) and log its `href` (URL), then change it to a new URL. We grab the input with `id="name"`, log its value ("Enter name"), and change it. For the `<h1>` with `id="title"`, we log its text ("Hello DOM"), add some HTML to make it italic, and give it the `highlight` class to turn it red (thanks to the CSS in the HTML). Try this in your HTML file: change `link.href` to another URL or set `title.textContent = 'Hi!'`. It's like redecorating parts of your page!

Warning: Be careful with `innerHTML` if using user input—it can cause security issues.

5. Changing CSS with JavaScript via style

You can change how elements look by tweaking their `style` property, like painting a wall.

Examples

```
let title = document.getElementById('title');
title.style.color = 'blue';  // Changes text color
title.style.backgroundColor = 'yellow';
title.style.fontSize = '2em';
```

```
// CamelCase for multi-word properties
title.style.borderRadius = '5px';
```

Explanation: This code makes the `<h1>` with `id="title"` look fancy! We change its text color to blue, background to yellow, font size to 2em (bigger text), and add a rounded border. Notice how `backgroundColor` uses camelCase (no hyphens). Try this in your HTML file: open it in a browser, add this code in the `<script>` tag, and see the `<h1>` change. Experiment by setting `title.style.color = 'green'` or changing `fontSize`. It's like giving your page a makeover!

Tip: For classes, use `classList.add('class')`, `classList.remove('class')`, or `classList.toggle('class')` instead of changing `className` directly.

6. Manipulating the DOM: createElement, appendChild, removeChild, replaceChild, etc.

You can add, remove, or swap elements on your page, like rearranging furniture.

- `document.createElement('tag')`: Makes a new element.
- `appendChild(node)`: Adds an element as a child.
- `removeChild(node)`: Removes a child.
- `replaceChild(new, old)`: Swaps one element for another.
- Others: `insertBefore(new, reference)`.

Examples

```
let newP = document.createElement('p');  // Create <p>
newP.textContent = 'New paragraph';
document.body.appendChild(newP);  // Add to body

let firstP = document.querySelector('.text');
document.body.removeChild(firstP);  // Remove it

let secondP = document.querySelector('.text');
let replacement = document.createElement('span');
replacement.textContent = 'Replaced';
secondP.parentNode.replaceChild(replacement, secondP);  // Replace
```

Explanation: This code lets you build and rearrange your page! First, we create a new `<p>` with text "New paragraph" and add it to the `<body>`. Then, we find the first `<p>` with `class="text"` and remove it. Finally, we replace the second `<p>` with a `<span>` saying "Replaced". Try this in your HTML file: save it, open it, and check how the page changes. Add another `<p>` with different text or replace something else, like an `<li>`. It's like being a web page architect!

7. DOMContentLoaded: Event and When to Use It

The `DOMContentLoaded` event waits until the HTML is fully loaded (before images or other slow stuff). It's like making sure the stage is set before starting the show.

Example

```
document.addEventListener('DOMContentLoaded', function() {
    // Safe to access DOM here
    let title = document.getElementById('title');
    console.log(title);  // Won't be null
});
```

Explanation: This code waits for the HTML to load before running, so we know the `<h1>` with `id="title"` exists. It logs the `<h1>` element to the console. Try this in your HTML file's `<script>` tag: save it, open it, and check the console (F12). Without this, your code might run too early and miss elements. Try removing `DOMContentLoaded` and putting the script in the `<head>`—it might fail! This event is like making sure all your toys are out of the box before playing.

When to Use: Put scripts at the end of `<body>` or use this event to avoid "element not found" errors.

8. Event Handlers: Adding, Accessing, and Conditional Logic

Event handlers let your page respond to actions like clicks. Use `addEventListener` for flexibility (better than inline `onclick`).

- Access the event object: Gives details like `event.target` (the element clicked).
- Conditional: Use `if` to make decisions based on event details.

Examples

```
let button = document.querySelector('button');
button.addEventListener('click', function(event) {
    console.log('Clicked!', event.target);  // The button element

    if (event.ctrlKey) {  // Check if Ctrl was pressed
        console.log('Ctrl + Click detected');
    }
});

// Remove: button.removeEventListener('click', func); (func must be named)
```

Explanation: This code makes the `<button>` in your HTML exciting! We add a `click` event listener that logs "Clicked!" and the button itself when clicked. If you hold the Ctrl key while clicking, it logs "Ctrl + Click detected". Try this in your HTML file: add it to the `<script>` tag, click the button, and try Ctrl+clicking. Check the console (F12) to see the messages. It's like teaching the button to talk back when you poke it!

9. Data Attributes in HTML

You can add custom info to HTML tags using `data-` attributes and access them with `dataset`.

Example

HTML: `<button data-user="Alice">Click</button>`

```
let btn = document.querySelector('button');
console.log(btn.dataset.user);  // 'Alice'
btn.dataset.user = 'Bob';  // Update
```

Explanation: This code is like putting a name tag on a button! In the HTML, the button has a `data-user="Alice"` attribute. We find the button, log its `data-user` value ("Alice"), and change it to "Bob". Add the HTML button to your file, then try this code in the `<script>` tag. Check the console (F12) and use the browser's "Inspect" tool to see the updated `data-user`. Try adding a new `data-` attribute, like `data-color="blue"`, and log it. It's like sticking extra notes on your elements!

10. Mouse Events: click, dblclick, mouseup, mousedown, mouseover, mouseout, mousemove

These events respond to mouse actions, like clicking or hovering.

Examples

```
let div = document.createElement('div');
div.textContent = 'Hover me';
document.body.appendChild(div);

div.addEventListener('click', () => console.log('Click'));
div.addEventListener('dblclick', () => console.log('Double click'));
div.addEventListener('mousedown', () => console.log('Mouse down'));
div.addEventListener('mouseup', () => console.log('Mouse up'));
div.addEventListener('mouseover', () => div.style.background = 'green');
div.addEventListener('mouseout', () => div.style.background = '');
div.addEventListener('mousemove', (e) => console.log(e.clientX,
e.clientY));  // Tracks position
```

Explanation: This code creates a `<div>` that says "Hover me" and adds it to the page. It listens for all sorts of mouse actions: clicking, double-clicking, pressing/releasing the mouse, hovering, leaving, and moving. When you hover, it turns green; when you move away, it clears. Moving the mouse logs its position. Try this in your HTML file: add it to the `<script>` tag, open the page, and interact with the `<div>`. Check the console (F12) for messages and watch the color change. It's like making a playground for your mouse!

11. Keyboard Events: keydown and keyup

These events catch when you press or release keys on your keyboard.

Examples

```
document.addEventListener('keydown', (e) => {
    console.log('Key down:', e.key);  // e.key is the key pressed
```

```
    if (e.key === 'Enter') {
        console.log('Enter pressed!');
    }
});

document.addEventListener('keyup', (e) => console.log('Key up:', e.key));
```

Explanation: This code listens for keyboard action! When you press any key, it logs “Key down” and the key (like “a” or “Enter”). If you press Enter, it adds a special message. When you release a key, it logs “Key up” and the key. Try this in your HTML file’s `<script>` tag: open the page, press some keys (like Enter or “a”), and check the console (F12). Try changing it to log a message for the “Space” key. It’s like making your page listen to your keyboard!

12. Form Events: blur, change, focus, reset, select, submit

These events handle interactions with forms, like typing or submitting.

- `focus`: When you click into an input.
- `blur`: When you click away from an input.
- `change`: When the input’s value changes.
- `select`: When text is selected in an input.
- `submit`: When the form is submitted.
- `reset`: When the form is reset.

Examples

```
let input = document.getElementById('name');
input.addEventListener('focus', () => input.style.border = '2px solid
blue');
input.addEventListener('blur', () => input.style.border = '');
input.addEventListener('change', () => console.log('Value changed to:',
input.value));

let form = document.getElementById('myForm');
form.addEventListener('submit', (e) => {
    e.preventDefault();  // Prevent page reload
    console.log('Form submitted!');
});
form.addEventListener('reset', () => console.log('Form reset'));
```

Explanation: This code makes the form in your HTML interactive! For the input with `id="name"`, it adds a blue border when you click into it (`focus`), removes it when you click away (`blur`), and logs the new text when you change it (`change`). For the form with `id="myForm"`, it logs when you submit it (but stops the page from reloading with `e.preventDefault()`) and when you reset it. Try this in your HTML file: type in the input, click Submit, or Reset, and check the console (F12). Try changing the border color to red. It’s like making your form talk to you!

Note: The query had “blue” instead of “blur”—we’re using “blur” here.

13. Form Validation: Checking Empty, Null, Numbers with isNaN, isFinite, etc.

Validation checks if form inputs are okay before submitting, like making sure a name isn't blank.

- Empty: Check if `value === ''` or `!value.trim()`.
- Null/Undefined: Check if `value == null`.
- Numbers: Use `isNaN(value)` (not a number) or `isFinite(value)` (not infinite).

Examples

```
function validateForm() {
    let name = document.getElementById('name').value;

    if (name === '' || name == null) {
        alert('Name is empty or null');
        return false;
    }

    let age = parseFloat(prompt('Enter age'));
    if (isNaN(age)) {
        alert('Not a number');
    } else if (!isFinite(age)) {
        alert('Infinite value');
    } else {
        console.log('Valid age:', age);
    }

    return true;
}
```

Explanation: This `validateForm` function checks if the form's input is good. It grabs the value of the input with `id="name"` and alerts if it's empty or missing. Then it asks for an age via a popup, checks if it's a valid number (not text or blank) and finite (not infinite), and logs the age if okay. Try this in your HTML file: add it to the `<script>` tag, call `validateForm()` in the console, and try leaving the input blank or entering a weird age like "abc". It's like a gatekeeper making sure your form is filled out right!

14. Submitting Forms via JS Using .submit()

You can submit a form with JavaScript, like clicking the Submit button automatically.

Example

```
let form = document.getElementById('myForm');
if (validateForm()) {
    form.submit();  // Submits the form
}
```

Explanation: This code finds the form with `id="myForm"` and submits it if `validateForm()` says everything's okay. It's like telling the form, "You're good to go—send it!" Try this in your HTML file: add the `validateForm` function from above, then run this code in the `<script>` tag or console. Try it with a valid input and an empty one to see the validation in action. It's a cool way to control your form programmatically!

Tip: Use with `e.preventDefault()` in handlers to check inputs first.

15. Regular Expressions (Regex): Basics with () [] | $? * + etc.

Regex is like a search tool for finding patterns in text, using special symbols.

- `()`: Groups parts together.
- `[]`: Matches any character in a set (e.g., `[a-z]` for lowercase letters).
- `|`: Means "or" (e.g., `cat|dog`).
- `$`: Marks the end of a string.
- `?`: Makes something optional (0 or 1 times).
- `*`: Matches 0 or more times.
- `+`: Matches 1 or more times.
- `.`: Matches any character.

Little Examples

```
let str = 'Hello123 world!';

// Match digits: \d+ (one or more digits)
let digits = /\d+/g;  // g flag for global
console.log(str.match(digits));  // ['123']

// Email pattern: Simple example
let emailRegex = /^[a-z]+@[a-z]+\.[a-z]+$/i;  // ^ start, $ end, i case-
insensitive
console.log(emailRegex.test('user@example.com'));  // true

// Optional: colou?r (u optional)
console.log(/colou?r/.test('color'));  // true
console.log(/colou?r/.test('colour'));  // true

// Groups and OR: (cat|dog)
let animal = /(cat|dog) food/;
console.log(animal.exec('cat food')[1]);  // 'cat'

// Any char one or more: .+
console.log(/.+/.test('anything'));  // true
```

Explanation: Regex is like a secret code for finding text patterns! This code tests a string ("Hello123 world!"). First, it finds all digits (`\d+`) and logs "123". Then it checks if "user@example.com" looks like an email (it does!). It tests if "color" or "colour" matches (both work because `u?` makes the "u" optional). It grabs "cat" from "cat food" using a group, and checks if a string has any characters. Try this in your console: test

`emailRegex.test('test@site.com')` or change the regex to match something else, like `[0-9]+` for numbers. It's like being a text detective!

Tip: Use `.test()` to check if a pattern matches, `.match()` to find matches, or `.replace()` to swap text.

Cheat Sheet

Topic	Description	Key Points/Examples
DOM Intro	Interface to interact with HTML as a tree of objects.	`document.body`, `document.head` – Access page structure.

Accessing Elements	Methods to select HTML elements for manipulation.	`getElementById('id')`
		`getElementsByClassName('class')` → collection
		`getElementsByTagName('tag')`
		`querySelector('.selector')` – First match
		`querySelectorAll()` – All matches as NodeList

Navigation	Traverse DOM to access related elements or nodes.	`element.firstElementChild`, `lastElementChild` `element.children` – Element kids `childNodes` – All nodes incl. text
Properties	Read or modify attributes of HTML elements.	`elem.id`, `className`, `innerHTML` (HTML), `textContent` (text) `href`, `src`, `value`
Change CSS	Modify element styles using inline CSS.	`elem.style.color = 'red'` `elem.style.backgroundColor = 'blue'` (camelCase)

Manipulate DOM	Dynamically create, add, remove, or replace elements.	`document.createElement('div')`
		`parent.appendChild(child)`
		`parent.removeChild(child)`
		`parent.replaceChild(new, old)`

DOMContentLoaded	Event ensuring DOM is fully loaded before access.	`document.addEventListener('DOMContentLoaded', func)` – Run after DOM loads.
Event Handlers	Functions triggered by user or system events.	`elem.addEventListener('click', (e) => { if (e.key === 'A') { ... } })` `e.target` – Element that triggered.
Data Attributes	Store custom data in HTML elements.	HTML: `data-key="value"` JS: `elem.dataset.key`

Topic	Description	Key Points/Examples
Mouse Events	Handle mouse interactions with elements.	`click`, `dblclick`, `mousedown/up`, `mouseover/out`, `mousemove` (e.clientX/Y)
Keyboard Events	Respond to keyboard key presses and releases.	`keydown/up` (e.key)
Form Events	Manage interactions with form elements.	`focus`, `blur`, `change`, `select`, `submit` (e.preventDefault()), `reset`
Form Validation	Check form inputs for validity before submission.	Empty: `value === ''` Null: `value == null` Number: `isNaN(val)`, `isFinite(val)`
Submit Form	Programmatically submit a form after validation.	`form.submit()` – After validation.
Regex Basics	Pattern matching for strings using special syntax.	`/pattern/` – e.g., `\d+` (digits), `[a-z]`, `(group)`, `| OR`, `? optional`, `* 0+`, `+ 1+`, `. any`, `$ end` Methods: `.test(str)`, `.match(str)`