DOM Elements

Document Object Model

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the structure of a document as a tree of nodes, with each node representing an element, attribute, or text content. The DOM provides a way for a programmer to access and manipulate the content, structure, and style of a document.

The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript. When a web page is loaded, the browser creates a Document Object Model of the page. This represents the structure of the HTML or XML document, and can be manipulated using JavaScript.
Here's an example of how the DOM can be used to access and modify the contents of an HTML element:

Accessing DOM objects

There are several ways to access DOM (Document Object Model) objects using JavaScript.

Some of the most common methods include:

Accessing Elements by id

getElementById(id): Returns the element with the specified id.

```
<div id="myDiv">This is a div</div>
<script>
let div = document.getElementById("myDiv");
console.log(div);
</script>
```

Accessing Elements by tag name

getElementsByTagName(tagname): Returns a collection of elements with the specified tag name.

```
<ul>
  <li>Item 1</li>
```

```
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
<script>
let listItems = document.getElementsByTagName("li");
console.log(listItems);
</script>
```

Accessing Elements by class name

getElementsByClassName(classname): Returns a collection of elements with the specified class name

```
<!DOCTYPE html>
<html>
<head>
  <title>getElementsByClassName Example</title>
</head>
<body>
  <div class="container">
    <h1>Welcome</h1>
    <p class="highlight">This is a paragraph with a class.</p>
    <p>This is another paragraph without the class.</p>
    <ul>
      <li class="highlight">List item 1</li>
      <li>List item 2</li>
      <li class="highlight">List item 3</li>
    </ul>
  </div>
  <script src="script.js">
// Retrieving elements with the class name 'highlight'
let elements = document.getElementsByClassName('highlight');

// Looping through the elements and modifying their styles
for (let i = 0; i < elements.length; i++) {
  elements[i].style.color = 'red';
  elements[i].style.fontWeight = 'bold';
}
</script>
</body>
</html>
```

Queryselector

- **querySelector(selector):** Returns the first element that matches the specified CSS selector.

```
<div class="myClass">This is a div</div>
<script>
let div = document.querySelector(".myClass");
console.log(div);
</script>
```

- **querySelectorAll(selector):** Returns a collection of all elements that match the specified selector.

```
<div class="myClass">This is a div</div>
<div class="myClass">This is another div</div>
<script>
let divs = document.querySelectorAll(".myClass");

for (let i = 0; i < divs.length; i++) {
    console.log(divs[i].textContent);
}
</script>
```

innerHTML

In JavaScript, the innerHTML property is used to get or set the HTML content of an element. It can be used on any element that can contain HTML content, such as a div, p, span, li, etc.

Here's an example of how to use innerHTML to get the content of an element:

```
<div id="myDiv">
  <p>This is a paragraph.</p>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
  </ul>
</div>
```

```
<script>
let div = document.getElementById("myDiv");
let divContent = div.innerHTML;
console.log(divContent);
// Output: "<p>This is a paragraph.</p><ul><li>Item
1</li><li>Item 2</li></ul>"
</script>
```

innerText

In JavaScript, the innerText property is used to get or set the text content of an element, including all of its child elements. It can be used on any element that can contain text, such as a div, p, span, li, etc.

Here's an example of how to use innerText to get the text content of an element:

```
<div id="myDiv">
  <p>This is a paragraph.</p>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
  </ul>
</div>

<script>
let div = document.getElementById("myDiv");
let divContent = div.innerText;
console.log(divContent);
// Output: "This is a paragraph. Item 1 Item 2"
</script>
```

Manipulating CSS

In JavaScript, you can manipulate the CSS of an element by accessing its style property and modifying its individual CSS properties.
Here's an example of how to change the background color of an element with the id "myDiv" to red:

```
<div id="myDiv">This is a div</div>
<script>
let div = document.getElementById("myDiv");
div.style.backgroundColor = "red";
</script>
```

Changing Attributes

Changing attributes of elements in the JavaScript Document Object Model (DOM) can be done using various methods and properties available. Here's how you can change attributes:

1. **setAttribute() and getAttribute():**

- **setAttribute()** allows setting an attribute for an element.
- **getAttribute()** retrieves the value of a specified attribute on an element.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Change Attribute Example</title>
</head>
<body>
  <img id="myImage" src="image.jpg" alt="Old Image">
  <button onclick="changeImage()">Change Image</button>

  <script>
    function changeImage() {
      let image = document.getElementById('myImage');
      image.setAttribute('src', 'newimage.jpg');
      image.setAttribute('alt', 'New Image');
    }
  </script>
</body>
</html>
```

Explanation:

- The HTML contains an image element with an ID ('myImage') and a button.
- When the button is clicked, the `changeImage()` function is triggered.
- Inside the function, `getElementById()` retrieves the image element.
- `setAttribute()` updates the `src` and `alt` attributes of the image.

2. Direct Property Assignment:

- Some attributes have corresponding properties that can be directly manipulated.
- For example, **`element.src`**, **`element.href`**, **`element.value`**, etc.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Change Attribute Example</title>
</head>
<body>
  <a id="myLink" href="https://www.oldlink.com">Old Link</a>
  <button onclick="changeLink()">Change Link</button>

  <script>
    function changeLink() {
      let link = document.getElementById('myLink');
      link.href = 'https://www.newlink.com';
      link.textContent = 'New Link';
    }
  </script>
</body>
</html>
```

Explanation:

- This example changes the `href` attribute and the text content of an anchor (`<a>`) element.
- The `changeLink()` function modifies the `href` property of the anchor element to a new URL.
- Additionally, it updates the `textContent` property to change the visible link text.

Adding/Removing CSS Classes

Adding or removing CSS classes to elements in the DOM can be accomplished using the `classList` property, which provides methods to manipulate classes on an element.

Adding a CSS Class:

To add a CSS class to an element, you can use the **`classList.add()`** method.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Add CSS Class Example</title>
  <style>
    .highlight {
      color: blue;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <p id="myParagraph">This is a paragraph.</p>
  <button onclick="addClass()">Add Class</button>

  <script>
    function addClass() {
      let paragraph = document.getElementById('myParagraph');
      paragraph.classList.add('highlight');
    }
  </script>
</body>
</html>
```

Explanation:

- The HTML contains a paragraph element and a button.
- Clicking the button triggers the `addClass()` function.
- Inside the function, `getElementById()` retrieves the paragraph element.
- `classList.add()` adds the 'highlight' class to the paragraph element, applying the styles defined in the CSS.

Removing a CSS Class:

To remove a CSS class from an element, use the **`classList.remove()`** method.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Remove CSS Class Example</title>
  <style>
    .highlight {
      color: blue;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <p id="myParagraph" class="highlight">This is a highlighted paragraph.</p>
  <button onclick="removeClass()">Remove Class</button>

  <script>
    function removeClass() {
      let paragraph = document.getElementById('myParagraph');
      paragraph.classList.remove('highlight');
    }
  </script>
</body>
</html>
```

Explanation:

- In this example, the paragraph element initially has the 'highlight' class applied.
- Clicking the button triggers the `removeClass()` function.
- `classList.remove()` removes the 'highlight' class from the paragraph element, removing the associated styles.

Using `classList`, you can dynamically add or remove CSS classes from elements in response to user interactions or other events, providing dynamic styling and behavior to your web pages or applications.

Creating new elements

Creating new elements in the Document Object Model (DOM) can be done using the `document.createElement()` method. This method allows you to dynamically create HTML elements using JavaScript and then append them to the existing document.

Creating and Appending a New Element:

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Create Element Example</title>
  <style>
    .highlight {
      color: blue;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <div id="container">
    <p>This is the existing content.</p>
    <button onclick="createNewElement()">Create Element</button>
  </div>

  <script>
    function createNewElement() {
      // Create a new paragraph element
      let newParagraph = document.createElement('p');

      // Set text content for the new paragraph
      newParagraph.textContent = 'This is a dynamically created paragraph.';

      // Add a class to the new paragraph
      newParagraph.classList.add('highlight');

      // Append the new paragraph to the existing container
      let container = document.getElementById('container');
      container.appendChild(newParagraph);
    }
```

```
  </script>
</body>
</html>
```

Explanation:

- The HTML includes a `div` with existing content and a button.
- Clicking the button triggers the `createNewElement()` function.
- Inside the function, `createElement('p')` creates a new paragraph element.
- `textContent` sets the text inside the new paragraph.
- `classList.add()` adds the 'highlight' class to the new paragraph.
- `appendChild()` appends the newly created paragraph to the existing container.

Inserting the New Element at a Specific Position:

You can also insert the newly created element at a specific position within an element using `insertBefore()`.

Example:

```
// Inserting the new element before an existing element
let existingElement = document.getElementById('existingElement');
let container = existingElement.parentNode;
container.insertBefore(newParagraph, existingElement);
```

In this case, `existingElement` represents an element already present in the DOM. The `insertBefore()` method inserts the `newParagraph` before the `existingElement` within its parent container.

Creating new elements dynamically allows you to generate content on the fly and manipulate the DOM structure to enhance interactivity and flexibility in web applications.