DOM Manipulation-2

Adding/Removing Event listeners

Adding and removing event listeners in the DOM allows you to respond to various user interactions, such as clicks, keypresses, mouse movements, etc. Here's how you can add and remove event listeners using JavaScript:

Adding an Event Listener:

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Event Listener Example</title>
</head>
<body>
  <button id="myButton">Click Me</button>

  <script>
    function handleClick() {
      console.log('Button clicked!');
    }

    // Adding an event listener to the button
    let button = document.getElementById('myButton');
    button.addEventListener('click', handleClick);
  </script>
</body>
</html>
```

Explanation:

- The HTML contains a button element with the ID 'myButton'.
- JavaScript code adds an event listener to the button using `addEventListener`.

- The function `handleClick` is the event handler that will execute when the button is clicked.

Removing an Event Listener:

To remove an event listener, you use the `removeEventListener` method, passing in the same function reference used when adding the listener.

Example:

```
// Removing the event listener from the button
button.removeEventListener('click', handleClick);
```

Explanation:

- This line of code removes the event listener from the button that was previously added.
- The `handleClick` function will no longer be triggered when the button is clicked.

Remember:

- When removing event listeners, it's important to use the same function reference and the same event type that was used while adding the listener.
- Anonymous functions cannot be removed using `removeEventListener` because they don't have a reference.

Adding and removing event listeners dynamically allows you to control the behavior of elements based on different conditions in your application, enabling better interactivity and responsiveness in your web pages.

Accessing parent/child/sibling elements:

In the Document Object Model (DOM), you can access parent, child, and sibling elements using various properties and methods provided by JavaScript. Here are some ways to navigate the DOM tree:

Accessing Parent Elements:

parentNode:

- `parentNode` property returns the parent node of the specified element.

Example:

```
let childElement = document.getElementById('childElementId');
let parentElement = childElement.parentNode;
```

Accessing Child Elements:

childNodes:

- `childNodes` property returns a collection of child nodes as a NodeList.

Example:

```
let parentElement = document.getElementById('parentElementId');
let children = parentElement.childNodes;
```

children:

- `children` property returns an HTMLCollection of child elements.

Example:

```
let parentElement = document.getElementById('parentElementId');
let childElements = parentElement.children;
```

firstChild and lastChild:

- `firstChild` returns the first child node.
- `lastChild` returns the last child node.

Example:

```
let parentElement = document.getElementById('parentElementId');
let firstChild = parentElement.firstChild;
let lastChild = parentElement.lastChild;
```

Accessing Sibling Elements:

previousSibling and nextSibling:

- `previousSibling` returns the previous sibling node.
- `nextSibling` returns the next sibling node.

Example:

```
let element = document.getElementById('elementId');
let previousSibling = element.previousSibling;
let nextSibling = element.nextSibling;
```

previousElementSibling and nextElementSibling:

- `previousElementSibling` returns the previous sibling element.
- `nextElementSibling` returns the next sibling element.

Example:

```
let element = document.getElementById('elementId');
let previousElementSibling = element.previousElementSibling;
let nextElementSibling = element.nextElementSibling;
```

These methods and properties allow you to navigate and access different parts of the DOM tree, enabling you to manipulate or retrieve information about parent, child, and sibling elements as needed within your JavaScript code.

Accessing Element Properties

Accessing element properties in the Document Object Model (DOM) can be done using JavaScript. Elements in the DOM have various properties that provide information about their attributes, styles, content, and more. Here's how you can access some common properties:

Common Element Properties:

Accessing Basic Properties:

- `id`: Accessing the ID of an element.

- `className` or `classList`: Accessing the CSS classes applied to an element.
- `tagName`: Accessing the tag name of an element.

Example:

```
let element = document.getElementById('myElementId');

// Accessing element properties
let elementId = element.id;
let classes = element.className; // or element.classList
let tagName = element.tagName;
```

Accessing Attributes:

- `getAttribute()`: Accessing specific attributes of an element.

Example:

```
let element = document.getElementById('myElementId');
let attributeValue = element.getAttribute('data-custom');
```

Style Properties:

- Accessing and modifying inline styles of an element.

Example:

```
let element = document.getElementById('myElementId');
element.style.color = 'blue';
element.style.fontSize = '16px';
```

Other Properties:

- Elements may have specific properties depending on their type (e.g., `value` for input elements, `textContent` for text content).

Example:

```
let inputElement = document.getElementById('myInput');
let inputValue = inputElement.value;
```

```
let paragraphElement = document.getElementById('myParagraph');
let textContent = paragraphElement.textContent;
```

Accessing Custom Data Attributes:

You can also access custom data attributes using the `dataset` property.

Example:

```
<div id="myDiv" data-info="some data"></div>

let divElement = document.getElementById('myDiv');
let customData = divElement.dataset.info; // Accessing data-info attribute value
```

These properties allow you to retrieve information, modify styles, or interact with elements in the DOM dynamically based on your application's requirements. They provide access to various aspects of an element's structure and content.

Modifying Styles

Certainly! Modifying styles in the Document Object Model (DOM) using JavaScript involves accessing the `style` property of an element, which allows you to change its CSS properties dynamically.

Modifying Inline Styles:

Accessing and Modifying Individual CSS Properties:

You can directly access and modify specific CSS properties of an element using the `style` property.

Example:

```
let element = document.getElementById('myElementId');

// Modifying individual CSS properties
element.style.color = 'blue';
```

```
element.style.fontSize = '16px';
element.style.backgroundColor = '#f0f0f0';
```

Setting CSS with Multiple Properties:

You can set multiple styles at once using the `cssText` property, but it overwrites existing styles completely.

Example:

```
let element = document.getElementById('myElementId');

// Setting multiple styles at once
element.style.cssText = 'color: blue; font-size: 16px; background-color: #f0f0f0;';
```

Adding/Removing CSS Classes:

Adding a CSS Class:

Adding a class to an element involves using the `classList` property's `add()` method.

Example:

```
let element = document.getElementById('myElementId');

// Adding a CSS class
element.classList.add('newClass');
```

Removing a CSS Class:

Removing a class from an element is done using the `remove()` method.

Example:

```
let element = document.getElementById('myElementId');

// Removing a CSS class
```

```
element.classList.remove('oldClass');
```

Toggle CSS Classes:

The `toggle()` method adds a class if it's not present or removes it if it is.

Example:

```
let element = document.getElementById('myElementId');

// Toggling a CSS class
element.classList.toggle('active');
```

These methods enable you to dynamically change an element's styles by directly modifying specific properties or by adding/removing CSS classes, providing flexibility in styling elements based on user interactions or application logic.

Manipulating Forms

Manipulating forms in the Document Object Model (DOM) involves interacting with form elements, retrieving user input, validating data, and handling form submissions using JavaScript. Here are some common operations:

Accessing Form Elements:

Accessing Form Element by ID:

```
let form = document.getElementById('myFormId');
```

Accessing Form Elements by Name:

```
let inputField = document.forms['myFormName']['inputFieldName'];
```

Getting and Setting Input Values:

Getting Input Value:

let inputValue = inputField.value;

Setting Input Value:

inputField.value = 'New value';

Form Submission:

Handling Form Submission:

```
form.addEventListener('submit', function(event) {
   event.preventDefault(); // Prevents default form submission
   // Access form elements, validate data, and perform actions
});
```

Submitting a Form Programmatically:
form.submit(); // Submits the form

Validating Form Data:

Validating Input:

- Access form elements, validate input using conditions, and display validation messages.

Using HTML5 Validation Attributes:

- HTML5 introduced attributes like `required`, `pattern`, etc., for basic form validation.

Resetting a Form:

Resetting Form Values:

form.reset(); // Resets form fields to their initial/default values

Working with Form Events:

Input Event:

```
inputField.addEventListener('input', function(event) {
   // Perform actions as the user inputs data
});
```

Change Event:

```
inputField.addEventListener('change', function(event) {
   // Perform actions when the input field value changes and loses focus
});
```

Accessing Form Controls:

Radio Buttons or Checkboxes:

```
let radioButtons = document.getElementsByName('radioButtonName');
let checkboxes = document.querySelectorAll('input[type="checkbox"]');
```

Select Dropdowns:

```
let selectDropdown = document.getElementById('selectId');
let selectedOption =
selectDropdown.options[selectDropdown.selectedIndex].value;
```

By utilizing these methods and events, you can manipulate form elements, retrieve user input, validate data, and handle form submissions, enabling dynamic and interactive forms within your web applications.