DOM

1. The Document Object Model (DOM) is a programming interface for HTML and XML documents, representing them as a structured tree-like model.
2. The DOM allows developers to access, manipulate, and navigate the elements and content of a web page dynamically.

Representation of the DOM

1. The DOM represents an HTML or XML document as a hierarchical structure known as the DOM tree.
2. The DOM tree consists of various nodes, where each node represents an element, attribute, or text within the document.
3. The relationship between nodes is defined by parent-child relationships, with each node having zero or more child nodes.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <h1 class="first">Document Object Model</h1>

    <!-- This is paragraph -->

    <p id="para">Let's learn about DOM here...</p>
    <div>
        <h3>Inner Headings</h3>
    </div>

</body>
</html>
```

Accessing nodes in DOM

parentNode, childNodes, firstChild, lastChild, nextSibling, previousSibling
It includes all the enters, spaces, comments and html elements.

```
document.body.childNodes
NodeList(9) [text, h1.first, text, comment, text, p#para, text, div, text]
```

parentElement, children, firstElementChild, lastElementChild, nextElementSibling, previousElementSibling
It includes only html elements.

```
document.body.children
HTMLCollection(3) [h1.first, p#para, div, para: p#para]
```

DOM Selectors

`document.getElementById(id)`
Returns the element with the specified ID attribute.

`document.getElementsByClassName(className)`
Returns a collection of elements with the specified class name.

`document.getElementsByTagName(tagName)`
Returns a collection of elements with the specified tag name.

`document.querySelector(selector)`
Returns the first element that matches the specified CSS selector

`document.querySelectorAll(selector)`
Returns a collection of elements that match the specified CSS selector

innerHTML

innerHTML is a property in JavaScript that allows you to access or modify the HTML content (including text and elements) within an element. It provides a convenient way to manipulate the content of an HTML element directly through JavaScript

Getting HTML Content

```
const myElement = document.getElementById("myElement");
const htmlContent = myElement.innerHTML;
console.log(htmlContent);
```

Setting HTML Content

```
const myElement = document.getElementById("myElement");
myElement.innerHTML = "<p>This is a new paragraph.</p>";
```

innerText

```
document.getElementsByTagName("div")[0].innerText
'Inner Headings'
```

textContent

```
document.getElementsByTagName("div")[0].textContent
'\n        Inner Headings\n    '
```

className

The className property in JavaScript is used to access or modify the class attribute of an HTML element. The class attribute is used to apply one or more CSS classes to an element, which can then be used to apply styling or define behavior through CSS rules or JavaScript interactions.

```
const myDiv = document.getElementById("myDiv");

// Getting and displaying the current class name(s)
const classNames = myDiv.className;
console.log(classNames);

// Changing the class name(s)
myDiv.className = "box blue";

// Adding and removing classes
myDiv.className += " highlighted";
myDiv.className = myDiv.className.replace("blue", "green");
```

classList

The classList property in JavaScript is used to access and manipulate the classes of an HTML element. It provides a more convenient and powerful way to work with the classes applied to an element compared to directly using the className property.

The classList property exposes a set of methods that allow you to add, remove, toggle, or check for the presence of classes on an element.

`add(className1, className2, ...)`

Adds one or more classes to the element.

```
const myElement = document.getElementById("myElement");
myElement.classList.add("new-class");
```

`remove(className1, className2, ...)`

Removes one or more classes from the element.

```
const myElement = document.getElementById("myElement");
myElement.classList.remove("old-class");
```

toggle(className, force)
Toggles a class on or off based on the second argument (force). If force is true, the class is added; if force is false, the class is removed. If force is not provided, the class is toggled.

```
const myElement = document.getElementById("myElement");

// Toggle a class on/off
myElement.classList.toggle("active");

// Add a class
myElement.classList.toggle("highlight", true);

// Remove a class
myElement.classList.toggle("inactive", false);
```

contains(className)
Checks if the element has a specific class and returns true or false.

```
const myElement = document.getElementById("myElement");
const hasClass = myElement.classList.contains("some-class");
```

createElement
The createElement method is a fundamental part of the JavaScript Document Object Model (DOM) API. It allows you to create a new HTML element dynamically in your web page and returns a reference to that newly created element. This is particularly useful when you want to add elements to the DOM or generate elements based on dynamic data.

```
// Create a new <div> element
const newDiv = document.createElement("div");

// Set attributes, styles, and content for the new <div>
newDiv.id = "dynamicDiv";
newDiv.className = "box";
newDiv.textContent = "This is a dynamically created div.";

// Append the new <div> to an existing element in the DOM
const container = document.getElementById("container");
container.appendChild(newDiv);
```

Different Ways to add elements in DOM

1. Append (Introduced in ECMAScript 2016)
   append is a more versatile method that allows you to append multiple elements or text to another element. It accepts a variable number of arguments, which can be elements, text strings, or a combination of both.

   ```
   const div = document.getElementsByTagName("div")[0];

   const childDiv = document.createElement("div");
   childDiv.innerText = 'I am a child div.';

   const parentContent = 'I belong to parent.';

   const childPara = document.createElement("p");
   childPara.innerText = 'I am a child para.';

   div.append(childDiv, parentContent, childPara);
   ```

   Output:

   ```
   <div>
         <!-- This is a div -->
         <h3>Inner Headings</h3>
         <div>I am a child div.</div>
         I belong to parent.
         <p>I am a child para.</p>
   </div>
   ```

2. appendChild()
   appendChild is a method used to append a single element as a child to another element. It takes an element node as its parameter and adds it as the last child of the specified parent element.

3. insertAdjacentHTML()
   The insertAdjacentHTML method is a powerful and versatile method available in JavaScript for inserting HTML content into an element at a specified position relative to the element's existing content. It provides a convenient way to dynamically add or modify content within the DOM (Document Object Model).

   The method accepts two arguments: a position value and an HTML string to be inserted.

   ```
   element.insertAdjacentHTML(position, htmlString);
   ```

The position argument specifies where the new HTML content should be inserted in relation to the element's existing content. It can take one of the following values:

'beforebegin': Inserts the HTML content immediately before the element itself.

'afterbegin': Inserts the HTML content as the first child of the element.

'beforeend': Inserts the HTML content as the last child of the element.

'afterend': Inserts the HTML content immediately after the element itself.

```
const parentElement = document.getElementById("parent");

// Insert content before the parent element
parentElement.insertAdjacentHTML('beforebegin', '<p>This is before
the parent element.</p>');

// Insert content as the first child of the parent element
parentElement.insertAdjacentHTML('afterbegin', '<p>This is the first
child.</p>');

// Insert content as the last child of the parent element
parentElement.insertAdjacentHTML('beforeend', '<p>This is the last
child.</p>');

// Insert content after the parent element
parentElement.insertAdjacentHTML('afterend', '<p>This is after the
parent element.</p>');
```

4. insertAdjacentElement()
   It is similar to insertAdjacentHTML, but instead of inserting HTML content as a string, it allows you to insert an actual DOM element.

5. insertBefore()
   The insertBefore method is a fundamental method in the JavaScript Document Object Model (DOM) that allows you to insert a new element before an existing element within the same parent element. This method is often used to dynamically reorder or insert elements in the DOM structure.

```
parentNode.insertBefore(newNode, referenceNode);
```

parentNode is the parent element where you want to insert the new element.

newNode is the element you want to insert.

referenceNode is the existing element that will serve as the reference point. The new element will be inserted before this reference element.

```
// Create a new <li> element
const newListItem = document.createElement("li");
newListItem.textContent = "New Item";

// Get the parent <ul> element
const parentList = document.getElementById("myList");

// Get a reference element (an existing <li> element)
const referenceElement = parentList.querySelector("li:nth-child(3)");

// Insert the new <li> element before the reference element
parentList.insertBefore(newListItem, referenceElement);
```

Different Ways of removing elements from DOM

1. remove()
   The remove method is a relatively new addition to the JavaScript DOM API that allows you to remove an element from the DOM (Document Object Model) structure. It provides a straightforward way to remove an element and its content from the document.

   It removes the element from the DOM along with all of its child elements and content. The remove method is more convenient to use and is often favored for its simplicity.

```
<div id="myDiv">
      <p>This is some content.</p>
</div>

const myDiv = document.getElementById("myDiv");

// Removes the entire <div> element and its contents from the DOM
myDiv.remove();
```

2. removeChild()

   The removeChild method is a longstanding method that has been part of the DOM API for a long time. It is called on the parent element from which you want to remove a specific child element. It requires passing the child element as an argument. It provides more control over which child element to remove, which can be useful in situations where you need to perform additional operations on the removed element before completely detaching it from the DOM.

   ```
   const parentElement = document.getElementById("parent");
   const childElement = document.getElementById("child");
   parentElement.removeChild(childElement);
   ```

Styling in DOM

1. Inline Styles

   You can set inline styles directly on an element using the style property. Inline styles are defined as a JavaScript object where the keys are the CSS properties, and the values are the corresponding values.

   ```
   const element = document.getElementById("myElement");
   element.style.backgroundColor = "blue";
   element.style.fontSize = "16px";
   ```

2. CSS Text

   You can set the style attribute of an element to a string containing CSS rules. This can be useful for adding complex or dynamic styles.

   ```
   const element = document.getElementById("myElement");
   element.style.cssText = "background-color: yellow; font-size: 20px;";
   ```

Attributes & DOM

1. getAttribute()
   The getAttribute method is a built-in JavaScript DOM (Document Object Model) method that allows you to retrieve the value of a specific attribute of an HTML element. It is used to access the value of attributes that are defined in the HTML markup.

   ```
   const attributeValue = element.getAttribute(attributeName);
   ```

   element is the reference to the HTML element you want to retrieve the attribute value from. attributeName is the name of the attribute you want to retrieve.

   ```
   <img id="myImage" src="image.jpg" alt="A beautiful image">

   const imageElement = document.getElementById("myImage");

   // Returns "image.jpg"
   const srcValue = imageElement.getAttribute("src");

   // Returns "A beautiful image"
   const altValue = imageElement.getAttribute("alt");
   ```

   Keep in mind that getAttribute returns a string representing the attribute value, even for boolean attributes like disabled or checked. If an attribute is not present on the element, the method will return null.

   Note that starting with HTML5, many attributes have corresponding properties that can be accessed directly on the element object, such as element.src instead of using getAttribute("src"). These properties often provide a more convenient way to access attribute values, especially for commonly used attributes.

2. setAttribute()
   setAttribute is a method in JavaScript used to modify or set the value of an attribute for a specified HTML element. It is particularly useful when you want to dynamically change or add attributes to elements on the page.

   ```
   element.setAttribute(attributeName, attributeValue);
   ```

   element: The reference to the HTML element on which you want to set the attribute.

   attributeName: The name of the attribute you want to modify or add.

   attributeValue: The value you want to assign to the specified attribute.

Modifying Existing Attributes

You can use setAttribute to modify the value of an existing attribute. For example, to change the src attribute of an image element.

```
<img id="myImage" src="image.jpg" alt="My Image">
```

```
const imageElement = document.getElementById('myImage');
imageElement.setAttribute('src', 'new-image.jpg');
```

Adding New Attributes

If you want to add a new attribute to an element, setAttribute can be used for that as well. For instance, you can add a data attribute to a div element.

```
<div id="myDiv">Hello</div>
```

```
const divElement = document.getElementById('myDiv');
divElement.setAttribute('data-info', 'Some additional info');
```

This will add a data-info attribute with the value "Some additional info" to the div element.

Special Considerations

When using setAttribute, be mindful of the attribute names and their case sensitivity. Attribute names are generally case-insensitive in HTML, but it's recommended to use lowercase for consistency.

If an attribute with the given name already exists, setAttribute will update its value. If it doesn't exist, the method will create a new attribute.

For certain attributes like class, style, and onclick, using setAttribute might not work as expected. It's better to use specific property assignments in those cases.

Removal of Attributes

To remove an attribute from an element, you can set its value to null or use the removeAttribute method.

```
const myElement = document.getElementById('myElement');

// Remove an attribute using setAttribute
myElement.setAttribute('data-info', null);
```

setAttribute provides a flexible way to manipulate attributes in the DOM, making it an essential tool for dynamic web development and interactive user interfaces. However, it's worth noting that when dealing with simple attributes like src, href, and others that have corresponding properties, directly setting the properties (e.g., element.src = 'new-image.jpg') is generally more straightforward and efficient.

3. hasAttribute()

   The hasAttribute method is a built-in JavaScript DOM (Document Object Model) method that allows you to check whether an HTML element has a specific attribute or not. It returns a boolean value (true or false) based on whether the attribute exists on the element or not.

```
const hasAttribute = element.hasAttribute(attributeName);

<input type="text" id="myInput" placeholder="Enter your name">

const inputElement = document.getElementById("myInput");

const hasPlaceholder = inputElement.hasAttribute("placeholder");
// Returns true

const hasValue = inputElement.hasAttribute("value");
// Returns false
```

4. removeAttribute()

   The removeAttribute method is a built-in JavaScript DOM (Document Object Model) method that allows you to remove a specific attribute from an HTML element. It is used to dynamically modify an element's attributes by removing the specified attribute.

```
element.removeAttribute(attributeName);

<a id="myLink" href="https://www.example.com" target="_blank">Visit
Example</a>

const linkElement = document.getElementById("myLink");
linkElement.removeAttribute("target");
// Removes the 'target' attribute
```

   Keep in mind a few points when using removeAttribute.

   1. The method removes the attribute entirely from the element, so any associated behavior or styling will be affected.
   2. If the attribute does not exist on the element, calling removeAttribute will have no effect.
   3. Be cautious when removing attributes that might be essential for the functionality or accessibility of the element.

Attributes

In the DOM (Document Object Model), elements don't have a direct "attributes array" like a traditional array. Instead, the attributes of an element are exposed through the attributes property, which is a collection of Attribute objects representing the attributes of the element. This collection is not an actual JavaScript array but can be treated similarly in some cases.

```
const element = document.getElementById("myElement");

// Accessing attributes by index (not recommended)
const firstAttribute = element.attributes[0];

// Iterating through the attributes
for (let i = 0; i < element.attributes.length; i++) {
  const attribute = element.attributes[i];
  console.log(attribute.name, attribute.value);
}

// Using a forEach-like loop with Array.from
Array.from(element.attributes).forEach(attribute => {
  console.log(attribute.name, attribute.value);
});
```

In the above examples, element.attributes gives you a collection of Attribute objects associated with the element. Each Attribute object has properties like name and value, representing the name and value of the attribute, respectively.

It's important to note that the attributes collection is not a true JavaScript array, so you cannot use array methods directly on it. However, you can convert it into an array using techniques like Array.from, as shown above, and then use array methods.

Keep in mind that working directly with the attributes collection is not as common as using other methods for attribute manipulation, such as getAttribute, setAttribute, and accessing attributes directly as properties (e.g., element.id, element.className). These methods offer more convenience and often better readability.

dataset Property

The dataset property is a feature in HTML and JavaScript that allows you to store custom data attributes on HTML elements. It provides a way to associate arbitrary data with HTML elements without affecting the rendering or behavior of the page. These custom data attributes can then be accessed and manipulated using JavaScript.

HTML
Define a custom data attribute using the data- prefix. For example:

```html
<div id="myElement" data-user-id="123" data-username="john_doe">...</div>
```

JavaScript
You can access and manipulate these custom data attributes using the dataset property in JavaScript:

```javascript
var element = document.getElementById("myElement");

// Accessing data attributes
var userId = element.dataset.userId; // "123"
var username = element.dataset.username; // "john_doe"

// Modifying data attributes
element.dataset.userId = "456";
element.dataset.username = "jane_smith";
```

The dataset property provides a convenient way to store and retrieve data associated with HTML elements. It's often used in conjunction with JavaScript to make web applications more dynamic and interactive. Keep in mind that data attributes are typically used for storing non-visible data and are not intended for styling or layout purposes.