Lesson:

Methods of DOM - Part 1

PW SKILLS

Topics Covered

1. getElementById()
2. getElementsByClassName()
3. getElementsByTagName()
4. querySelector()
5. querySelectorAll()
6. Internal HTML object collections

In this section, we will explore various methods of the DOM (Document Object Model) that allow JavaScript to access and retrieve specific elements from the web page. These methods provide different ways to select elements based on their attributes, tags, or CSS selectors. Let's dive into each method

getElementById()

The **getElementById()** method is a commonly used method in the Document Object Model (DOM) of web pages. It is a JavaScript method that allows you to retrieve a reference to an HTML element on a web page using its unique ID. This method returns a reference to the first element with the specified ID that is found in the document.

Here's a detailed explanation of how to use the **getElementById()** method in JavaScript:

Identify the ID of the HTML element you want to retrieve.
Before using the **getElementById()** method, you need to know the ID of the HTML element you want to retrieve. The ID should be unique within the HTML document.

Use the **getElementById()** method to retrieve the HTML element.
The **getElementById()** method is called on the document object, which represents the entire HTML document. The method takes one argument, which is a string representing the ID of the element you want to retrieve. For example, the following code retrieves the HTML element with an ID of "myElement":

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

The **getElementById()** method returns a reference to the HTML element with the specified ID. This reference can be used to access the properties and methods of the element.

getElementsByClassName()

The **getElementsByClassName()** method is a built-in method in JavaScript that is used to retrieve a list of all the elements that have a specific class name. This method is used to select one or more elements on a web page based on the value of their class attribute. The method returns an HTMLCollection, which is an array-like object containing all the elements that match the specified class name.

Here's a detailed explanation of how to use the **getElementsByClassName()** method in JavaScript:

Identify the class name of the HTML elements you want to retrieve.

Before using the **getElementsByClassName()** method, you need to know the class name of the HTML elements you want to retrieve. The class name should be unique enough to select only the elements you want to retrieve.

Use the **getElementsByClassName()** method to retrieve the HTML elements.
The **getElementsByClassName()** method is called on the document object, which represents the entire HTML document. The method takes one argument, which is a string representing the class name of the elements you want to retrieve. For example, the following code retrieves all the HTML elements with a class name of "myClass":

```
var elements = document.getElementsByClassName("myClass");
```

The **getElementsByClassName()** method returns an HTMLCollection, which is an array-like object that contains all the HTML elements that match the specified class name. This collection can be accessed using a numeric index or using a loop.
In summary, the **getElementsByClassName()** method is a JavaScript method used to retrieve a list of HTML elements by their class name. This method is commonly used to manipulate the content and attributes of HTML elements on a web page.

getElementsByTagName()

The **getElementsByTagName()** method is a built-in method in JavaScript that is used to retrieve a list of all the elements with a specific tag name. This method is used to select one or more elements on a web page based on the value of their HTML tag. The method returns an HTMLCollection, which is an array-like object containing all the elements that match the specified tag name.

Here's a detailed explanation of how to use the **getElementsByTagName()** method in JavaScript:

Identify the tag name of the HTML elements you want to retrieve.
Before using the **getElementsByTagName()** method, you need to know the tag name of the HTML elements you want to retrieve. The tag name should be unique enough to select only the elements you want to retrieve.

Use the **getElementsByTagName()** method to retrieve the HTML elements.

The **getElementsByTagName()** method is called on the document object, which represents the entire HTML document. The method takes one argument, which is a string representing the tag name of the elements you want to retrieve. For example, the following code retrieves all the HTML elements with a tag name of "p":

```
var elements = document.getElementsByTagName("p");
```

The **getElementsByTagName()** method returns an HTMLCollection, which is an array-like object that contains all the HTML elements that match the specified tag name. This collection can be accessed using a numeric index or using a loop.
In summary, the **getElementsByTagName()** method is a JavaScript method used to retrieve a list of HTML elements by their tag name. This method is commonly used to manipulate the content and attributes of HTML elements on a web page.

querySelector()

The **querySelector()** method is a built-in method in JavaScript that allows you to select the first element that matches a specified CSS selector. It is commonly used to retrieve and manipulate HTML elements on a web page.

The method is called on the document object, which represents the entire HTML document, and takes one argument, which is a string representing the CSS selector of the element you want to retrieve. The CSS selector can be any valid CSS selector, including class names, IDs, element types, attribute selectors, and more.

Here are some examples of CSS selectors that can be used with the **querySelector()** method:

- **Retrieving an element by ID:**

  #myId: Selects the element with the ID of **"myId"**

```
var element = document.querySelector("#myId");
```

- **Retrieving an element by class name:**

  .myClass: Selects all elements with the class name of **"myClass"**

```
var element = document.querySelector(".myClass");
```

- **Retrieving an element by tag name:**

  p: Selects all paragraph elements

```
var element = document.querySelector("p");
```

This code retrieves the first paragraph element on the page.

- **Retrieving an element using a complex selector:**

```
var element = document.querySelector("ul li:nth-child(2)");
```

This code retrieves the second li element within a ul element on the page.

- **Retrieving an element with an attribute selector:**

```
var element = document.querySelector("a[href='#']");
```

This code retrieves the first anchor element on the page with an href attribute equal to #.

- **Retrieving a child element within a parent element:**

```
var element = document.querySelector("#myParent .myChild");
```

This code retrieves the first child element with a class name of "myChild" within an element with an ID of "myParent".

These are just a few examples of how the querySelector() method can be used to retrieve elements on a web page. The method is very versatile and can be used with a wide range of CSS selectors to target specific elements.

querySelectorAll()

querySelectorAll() is a method available in the Document Object Model (DOM) API that allows you to retrieve a list of all elements in the document that match a specified CSS selector. This method returns a NodeList object, which is similar to an array, that contains all of the matching elements in the order in which they appear in the document.

The **querySelectorAll()** method takes a single parameter, which is a string representing the CSS selector to use when selecting elements. The CSS selector syntax is used to specify the criteria for selecting elements based on their attributes, classes, or other properties. The syntax for using **querySelectorAll()** is as follows:

```
document.querySelectorAll(selector)
```

The selector parameter is a string that represents the CSS selector to be used when selecting elements. It can be any valid CSS selector, including class selectors, ID selectors, attribute selectors, and more. Here are some examples of valid CSS selectors:

```
/* Select all elements with a class of "my-class" */
.my-class

/* Select the element with an ID of "my-id" */
#my-id

/* Select all elements with a "data-type" attribute */
[data-type]

/* Select all "a" elements with a "href" attribute that
contains "example.com" */
a[href*="example.com"]
```

querySelectorAll() returns a NodeList object that contains all of the elements that match the specified selector. The NodeList is similar to an array, but it is a static list that does not update automatically as the document changes. You can access individual elements in the NodeList using the item() method or by using array-style bracket notation. For example, to get the first element in the NodeList, you can use:

```
const firstElement = document.querySelectorAll(selector)[0];
```

You can also loop through the NodeList using a for loop or using array methods such as forEach(), map(), or filter(). For example, to loop through all of the elements that match a selector and log their text content to the console, you could use the following code:

```
const elements = document.querySelectorAll(selector);

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

Alternatively, you could use the forEach() method to achieve the same result:

```
document.querySelectorAll(selector).forEach((element) => {
  console.log(element.textContent);
});
```

In summary, querySelectorAll() is a powerful method in the DOM API that allows you to select and manipulate elements in the document based on their attributes, classes, or other properties.

Internal HTML object collections

In web development, efficient access and manipulation of elements are key to creating dynamic and interactive web pages. The Document Object Model (DOM) offers a powerful feature known as "internal HTML object collections." These collections are built-in properties of the document object, providing developers with a streamlined way to interact with specific types of elements on a web page.

Internal HTML object collections are like specialized containers that hold related elements. They allow you to access, iterate through, and manipulate groups of elements without the need to individually select each one. This not only simplifies your code but also enhances the performance of your web applications.

Let's explore some of the commonly used internal HTML object collections:

1. **document.forms Collection:**

   The **forms** collection provides access to all HTML form elements on the web page

```
const allForms = document.forms;  // Returns a collection of
all forms on the page
const myForm = document.forms.myFormName;  // Access a
specific form by its name attribute
const firstForm = document.forms[0];  // Access the first
form in the collection
```

2. **document.images Collection:**

   The **images** collection provides access to all HTML <img> elements on the web page.

```
const allImages = document.images;  // Returns a collection
of all images on the page
const firstImage = document.images[0];  // Access the first
image in the collection
```

3. **document.links Collection:**

   The **links** collection provides access to all HTML anchor <a> elements with the 'href' attribute on the web page.

```
const allLinks = document.links;  // Returns a collection of
all anchor (link) elements on the page
const firstLink = document.links[0];  // Access the first
link in the collection
```

4. **document.getElementsByTagName() Method:**

   Although not a direct collection, the **getElementsByTagName()** method returns a collection-like HTMLCollection of elements with the specified tag name.

```
const paragraphs = document.getElementsByTagName('p');  //
Returns a collection of all <p> elements on the page
const headings = document.getElementsByTagName('h1');  //
Returns a collection of all <h1> elements on the page
```

5. **document.getElementByClassName() Method:**

   Similar to the previous method, **getElementsByClassName()** returns a collection-like HTMLCollection of elements with the specified class name.

```
const highlightedElements =
document.getElementsByClassName('highlight');  // Returns a
collection of all elements with the 'highlight' class
```

These internal HTML object collections make it easier to work with specific groups of elements on a web page. You can iterate through the collections, access individual elements, and manipulate their properties and attributes. They are particularly useful when you want to apply changes or behavior to a set of related elements without having to manually select each one.

It's important to note that the collections returned by these properties and methods are "live" collections, meaning they are updated automatically when the DOM changes. If elements are added or removed from the page, the collections will reflect those changes. However, since these collections are live, be cautious when modifying them in loops, as the loop behavior can change dynamically as elements are added or removed.