**Document Object Model**

**DOM (Document Object Model)** - это программное представление структуры веб-страницы. DOM представляет собой иерархическое дерево, в котором каждый элемент веб-страницы (такой как заголовок, параграф, изображение, ссылка и так далее) представлен в виде объекта, который можно манипулировать с помощью JavaScript.

**Подключения скрипта для работы с DOM**

1. Подключение <script> в конце <body>

Пример:   
<!DOCTYPE html>

<html>

<head>

<title>page - title</title>

</head>

<body>

<p id="demo">Hello.</p>

<script src="myscript.js" ></script>

</body>

</html>

1. Подключение с использованием атрибута **defer.** Атрибут defer для тега <script> полезен при работе с DOM (Document Object Model) в JavaScript, особенно в случаях, когда вам нужно выполнить скрипт после загрузки и разбора всего HTML-документа. Он позволяет отложить выполнение скрипта до завершения парсинга HTML, что полезно, если ваш скрипт взаимодействует с элементами на странице, которые должны быть доступны после загрузки документа.

Пример:

<!DOCTYPE html>

<html>

<head>

<title>Пример с defer</title>

<script src="myscript.js" defer></script>

</head>

<body>

<p id="demo">Это веб-страница с отложенным скриптом.</p>

</body>

</html>

1. Через обработку события

**Объект document** представляет веб-страницу в **DOM (Document Object Model)** и предоставляет интерфейс для взаимодействия с содержанием и структурой этой страницы. document - это один из главных объектов в браузерном JavaScript и позволяет вам доступиться к элементам, управлять содержанием, изменять стили, обрабатывать события и многое другое.

**document.body** - это свойство объекта document в браузерном JavaScript, которое предоставляет доступ к элементу <body> текущей веб-страницы. Этот элемент <body> является одним из основных элементов HTML-документа и содержит основное содержание, которое видно пользователю при отображении страницы в браузере.

Свойство **document.body** позволяет вам манипулировать содержанием и стилями <body> элемента, а также добавлять или удалять элементы и обрабатывать события, связанные с этим элементом.

**document.body.childNodes** - это свойство, которое предоставляет коллекцию всех непосредственных дочерних узлов (например, элементов, текстовых узлов, комментариев) элемента <body> текущей веб-страницы. Этот объект представляет собой коллекцию узлов, которые непосредственно находятся внутри элемента <body>.

**document.body.style** - это свойство, которое предоставляет доступ к стилям (CSS стилям) элемента <body> текущей веб-страницы. С помощью этого свойства вы можете динамически изменять стили элемента <body> с использованием JavaScript.

Примеры использования document.body.style:

Изменение фона элемента <body>

document.body.style.backgroundColor = "lightblue";

**Важно!!! –** названия свойств CSS стилей пишется через **CamelCase**

**Пример margin-right на CamelCase – marginRight**

**Поиск элементов в HTML-документе**

Для поиска элементов в HTML-документе в JavaScript существует несколько методов и свойств.

**getElementById(id):** Этот метод позволяет получить доступ к элементу по его уникальному идентификатору (id). Он возвращает один элемент, соответствующий заданному id

Пример:  
 const element = document.getElementById("myElement");

**getElementsByClassName(className):** Этот метод возвращает коллекцию элементов, которые имеют указанный класс. Может вернуть несколько элементов.

Пример:  
 const elements = document.getElementsByClassName("myClass");

**getElementsByTagName(tagName)**: Этот метод возвращает коллекцию элементов с указанным тегом (например, "div" или "p"). Может вернуть несколько элементов.

Пример:  
 const elements = document.getElementsByTagName("div");

**querySelector(selector):** Этот метод возвращает первый элемент, который соответствует заданному CSS-селектору. Он позволяет более гибко выбирать элементы.

Пример:  
 const element = document.querySelector(".myClass");

**querySelectorAll(selector):** Этот метод возвращает все элементы, которые соответствуют заданному CSS-селектору. Он также позволяет выбирать элементы на основе более сложных правил.

Пример:  
 const elements = document.querySelectorAll(".myClass");

**nextElementSibling** в JavaScript представляет собой ссылку на следующий элемент-брата (соседний элемент на том же уровне иерархии DOM) относительно текущего элемента. Этот элемент-брат должен быть элементом (т.е., HTML-элементом), и свойство **nextElementSibling** игнорирует текстовые узлы, комментарии и другие типы узлов, которые могут находиться между элементами на одном уровне.

Пример:

**<html - блок>**

<ul>

<li>Первый элемент списка</li>

<li>Второй элемент списка</li>

<li id="current">Третий элемент списка</li>

<li>Четвертый элемент списка</li>

<li>Пятый элемент списка</li>

</ul>

**<js- блок>**

const currentElement = document.getElementById("current");

const nextElement = currentElement.nextElementSibling;

if (nextElement) {

// Если соседний элемент существует

console.log("Следующий элемент в списке: ", nextElement.textContent);

} else {

console.log("Нет следующего элемента в списке.");

}

**previousElementSibling** в JavaScript представляет собой ссылку на предыдущий элемент-брата (соседний элемент на том же уровне иерархии DOM) относительно текущего элемента. Этот элемент-брат также должен быть HTML-элементом, и свойство **previousElementSibling** игнорирует текстовые узлы, комментарии и другие типы узлов, которые могут находиться между элементами на одном уровне.

**<html - блок>**

<ul>

<li>Первый элемент списка</li>

<li>Второй элемент списка</li>

<li id="current">Третий элемент списка</li>

<li>Четвертый элемент списка</li>

<li>Пятый элемент списка</li>

</ul>

**<js- блок>**

const previousElement = document.getElementById("current");

const nextElement = currentElement.previousElementSibling;

if (nextElement) {

// Если соседний элемент существует

console.log("Предыдущий элемент в списке: ", previous.textContent);

} else {

console.log("Нет предыдущего элемента в списке.");

}

**Популярные методы и свойства для работы с элементами HTML**

JavaScript предоставляет множество методов и свойств для работы с элементами HTML. Наиболее распространенные методы и свойства для работы с элементами HTML

**Свойства элементов:**

1. **innerHTML**: Получение и установка HTML-содержимого элемента.
2. **textContent**: Получение и установка текстового содержимого элемента.
3. **id**: Получение и установка идентификатора (ID) элемента.
4. **style**: Доступ к стилям элемента для управления его внешним видом.
5. **value**: Получение и установка значения элемента формы, таких как <input> и <textarea>.
6. **attributes:** Это свойство объекта элемента, предоставляющее доступ к коллекции атрибутов элемента.
7. **parentElement:** в JavaScript предоставляет доступ к родительскому элементу текущего элемента в DOM. Это позволяет вам перемещаться вверх по иерархии DOM относительно текущего элемента и взаимодействовать с его родительским элементом

**Методы для работы с элементами:**

1. **createElement(tagName**): Создание нового HTML-элемента.
2. **appendChild(node):** Добавление дочернего узла к элементу.
3. **removeChild(node):** Удаление дочернего узла.
4. **classList.add(‘**class1**’):** используется для добавления одного или нескольких CSS-классов к элементу HTML
5. **classList.toggle(‘**className**’):** это метод объекта classList, который используется для переключения (включения или выключения) класса у элемента. Если класс уже существует у элемента, он будет удален; если класс отсутствует, он будет добавлен
6. **classList.** **remove(‘**className**’):** это метод объекта classList, который используется для удаления класса (или нескольких классов) у элемента. Этот метод позволяет динамически управлять классами элемента, что может быть полезным, например, при изменении стилей или поведения элемента на веб-странице.
7. **getAttribute(**name**):** Этот метод используется для получения значения указанного атрибута элемента. Вы передаете имя атрибута в качестве аргумента.
8. **setAttribute(**name, value**):** Этот метод используется для установки значения указанного атрибута элемента. Вы передаете имя атрибута и новое значение в качестве аргументов.
9. **removeAttribute(name):** Этот метод используется для удаления указанного атрибута у элемента. Вы передаете имя атрибута в качестве аргумента.
10. **hasAttributes():** в JavaScript используется для проверки наличия атрибутов у элемента. Этот метод возвращает true, если элемент имеет хотя бы один атрибут, и false, если у элемента нет атрибутов.
11. **insertBefore(newNode, referenceNode):** Вставка элемента перед другим элементом.
12. **cloneNode(deep):** Клонирование элемента (глубокое или поверхностное). Он принимает один аргумент, deep, который определяет, должна ли быть создана глубокая копия (включая все дочерние элементы) или поверхностная копия (без дочерних элементов). **true** - Создать глубокую копию, **false** - Создать поверхностную копию
13. **closest(selector) -** Метод closest используется для поиска ближайшего родительского элемента, который соответствует заданному селектору (CSS-селектору) или элементу. Этот метод начинает поиск с текущего элемента и двигается вверх по иерархии DOM, пока не найдет элемент, который соответствует заданному селектору, или до тех пор, пока не достигнет корневого элемента (обычно <html>).

**element.closest(selector)**

где:

element - это HTML-элемент, с которого начинается поиск.

selector - это CSS-селектор или имя элемента, который вы ищете.