1. Что такое DOM/BOM  
DOM (Document Object Model) и BOM (Browser Object Model) — это две ключевые концепции, связанные с работой браузера и взаимодействием JavaScript с веб-страницами. Вот их краткое объяснение:

1. DOM (Document Object Model)

DOM — это представление структуры HTML или XML-документа в виде объектной модели. Оно позволяет JavaScript взаимодействовать с элементами веб-страницы, изменять их содержимое, стиль и структуру.

Основные характеристики DOM:

Иерархическая структура: Вся веб-страница представлена как дерево объектов (узлов). Например: <html>, <body>, <div>.

Манипуляции элементами: Вы можете добавлять, удалять или изменять элементы страницы с помощью методов JavaScript.

События: DOM позволяет обрабатывать события, такие как клики, ввод текста или прокрутка.

// Изменение содержимого элемента

document.getElementById("title").textContent = "Привет, мир!";

2. BOM (Browser Object Model)

BOM — это объектная модель, которая предоставляет интерфейсы для взаимодействия с браузером и его функциональностью (не связанной с HTML-документом).

Основные возможности BOM:

Навигация: Доступ к текущему URL, переходы на другие страницы.

console.log(window.location.href); // Вывод текущего URL

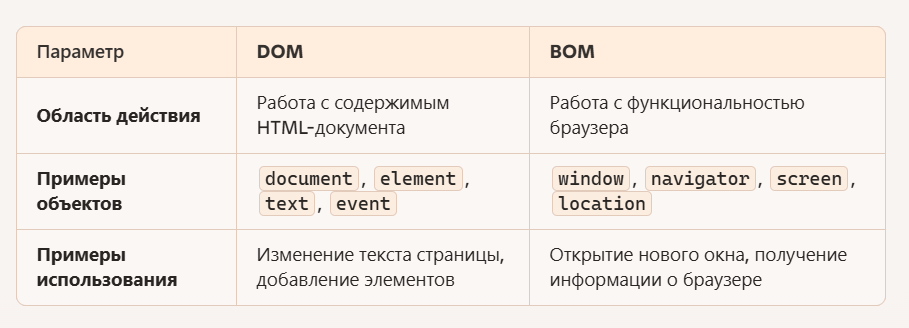
Диалоги: Вызов alert(), confirm() и prompt().

alert("Это BOM в действии!");

Манипуляции с окнами: Открытие и закрытие окон, управление их размерами.

window.open("https://example.com");

Работа с navigator: Доступ к информации о браузере и операционной системе.

console.log(navigator.userAgent); // Информация о браузере  
  
  
DOM предназначен для взаимодействия с содержимым страницы.

BOM предоставляет доступ к функциональности браузера.

Что такое DOM и его структура (узлы, элементы, атрибуты, текстовые узлы).  
DOM (Document Object Model) — это объектная модель документа, которая представляет структуру HTML или XML-документа в виде дерева узлов. Это позволяет программам, например JavaScript, динамически изменять содержание, структуру и стиль веб-страницы. Каждый компонент документа становится отдельным объектом, с которым можно взаимодействовать через DOM API.

Структура DOM

DOM представлен как дерево узлов, где каждый элемент, атрибут или текстовый фрагмент является узлом дерева. Разберём основные понятия:

1. Узлы (Nodes):

Это базовые элементы DOM. Каждый элемент, атрибут, комментарий или текст в документе является узлом. Все узлы имеют типы, определяющие их роль:

Элемент (Element Node): Представляет теги HTML, например <div>, <p>.

Текстовый узел (Text Node): Представляет текст внутри элементов.

Атрибут (Attribute Node): Представляет атрибуты тега, такие как id, class.

<div id="container">Привет, мир!</div>

DOM-дерево:

1. Элемент <div> (Element Node)

└─ Текст "Привет, мир!" (Text Node)

2. Элементы (Element Nodes):

Элементы — это HTML-теги, которые являются основой DOM-структуры. Они содержат:

Имя тега (div, span, h1 и т.д.).

Список атрибутов.

Вложенные узлы (дети).

const element = document.getElementById("container");

console.log(element.tagName); // "DIV"

3. Атрибуты (Attribute Nodes):

Атрибуты содержат дополнительную информацию об элементах. Они являются частью элемента, но не отображаются в виде узлов дерева.

<div id="box" class="highlight"></div>

Доступ к атрибутам:

const element = document.getElementById("box");

console.log(element.id); // "box"

console.log(element.getAttribute("class")); // "highlight"

4. Текстовые узлы (Text Nodes):

Текст внутри элементов представлен текстовыми узлами. Даже пробелы между элементами могут быть текстовыми узлами.

<p>Текстовый узел</p>

Доступ к тексту:

const paragraph = document.querySelector("p");

console.log(paragraph.textContent); // "Текстовый узел"

Иерархия узлов в DOM

Каждый узел может иметь родителя, дочерние элементы и соседей:

Родитель (parentNode): Узел, который содержит текущий узел.

Дочерние элементы (childNodes): Вложенные узлы текущего узла.

Соседи (sibling): Узлы, которые находятся на одном уровне.

<ul>

<li>Элемент 1</li>

<li>Элемент 2</li>

</ul>

Для li:

Родитель: ul.

Соседи: другие элементы li.

Итог: DOM — это мощный инструмент для работы с HTML-документами. Он позволяет легко манипулировать элементами, добавлять новые узлы, изменять атрибуты и управлять текстом.

Различия между document и window.  
document и window — это два ключевых объекта в браузерной среде, которые используются для работы с веб-страницей. Несмотря на их тесную связь, они имеют разные функции и роли. Давайте разберём их основные различия:

1. window

Определение: window представляет собой глобальный объект браузера. Это "окно", в котором загружается веб-страница.

Функционал: Содержит методы и свойства, связанные с самой средой браузера, такие как управление окнами, таймерами, диалоговыми окнами и навигацией.

Примеры свойств и методов

window.alert() — отображает всплывающее окно.

window.location — предоставляет доступ к текущему URL и позволяет его изменять.

window.open() — открывает новое окно или вкладку.

Особенности:

Является глобальным контекстом выполнения: вызовы alert(), location.href и других методов можно писать без упоминания window, так как это подразумевается автоматически.

console.log(window.innerWidth); // Ширина окна браузера

2. document

Определение: document — это объект, который представляет загруженный HTML-документ. Он является частью window и предоставляет доступ ко всему содержимому веб-страницы.

Функционал: Позволяет взаимодействовать с DOM (структурой документа), изменять HTML, управлять событиями и стилями.

Примеры свойств и методов

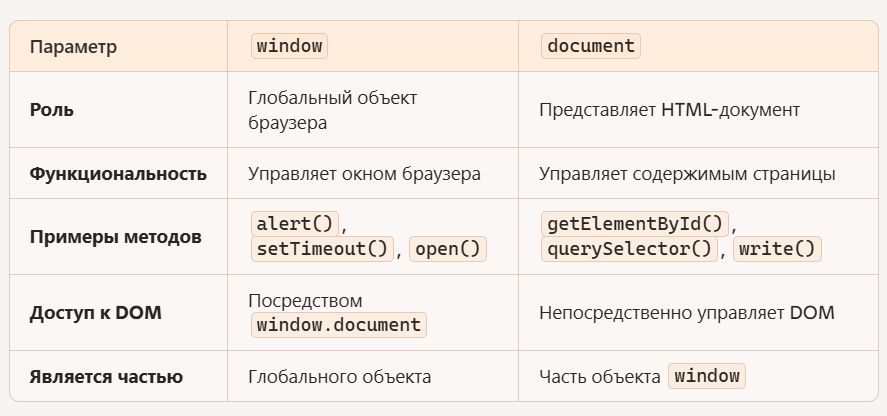
document.getElementById() — находит элемент по его ID.

document.querySelector() — возвращает первый элемент, соответствующий CSS-селектору.

document.title — задаёт или возвращает заголовок страницы

Особенности:

Позволяет динамически изменять содержимое страницы: добавлять, удалять, изменять элементы.

console.log(document.title); // Заголовок текущей страницы  
  
Взаимосвязь:

document является свойством объекта window. Когда вы пишете document.getElementById(), вы фактически вызываете window.document.getElementById().

Взаимодействие с document: document.body, document.head, document.title.  
Взаимодействие с объектом document предоставляет доступ к различным частям HTML-документа. Давайте рассмотрим три ключевых свойства: document.body, document.head и document.title.

1. document.body

Описание: Свойство document.body содержит объект, представляющий <body> текущего HTML-документа.

Использование: Позволяет работать с содержимым страницы: добавлять, удалять или изменять элементы.

console.log(document.body); // Получает содержимое тега <body>

document.body.style.backgroundColor = "lightblue"; // Изменяет цвет фона страницы

document.body.innerHTML += "<p>Новый параграф</p>"; // Добавляет новый элемент на страницу

2. document.head

Описание: Свойство document.head содержит объект, представляющий <head> текущего HTML-документа.

Использование: Позволяет получать или изменять метаданные страницы, такие как <title>, <meta>, <link>.

console.log(document.head); // Получает содержимое тега <head>

const metaTag = document.createElement("meta");

metaTag.name = "description";

metaTag.content = "Пример взаимодействия с document.head";

document.head.appendChild(metaTag); // Добавляет новый метатег в <head>

3. document.title

Описание: Свойство document.title позволяет получать или изменять содержимое тега <title> в документе.

Использование: Читает или задаёт заголовок страницы (видимый в вкладке браузера).

console.log(document.title); // Выводит текущий заголовок страницы

document.title = "Новый заголовок"; // Изменяет заголовок страницы  


Различия между innerHTML, textContent, outerHTML.  
innerHTML, textContent и outerHTML — это свойства DOM, которые используются для работы с содержимым элементов. У каждого из них свои особенности и случаи применения. Давайте разберём их различия:

1. innerHTML

Описание: Возвращает или устанавливает HTML-код, содержащийся внутри элемента. Включает разметку HTML, если она присутствует.

Когда использовать: Когда нужно работать с HTML-разметкой (например, добавление тегов, вложенных элементов).

const div = document.createElement("div");

div.innerHTML = "<p>Это <strong>текст</strong> с HTML.</p>";

console.log(div.innerHTML);

// Output: "<p>Это <strong>текст</strong> с HTML.</p>"

2. textContent

Описание: Возвращает или устанавливает текстовое содержимое элемента, не учитывая HTML-разметку. Отображает только текст, даже если внутри элемента есть вложенные теги.

Когда использовать: Когда нужно получить или изменить только текст, игнорируя любые теги HTML.

const div = document.createElement("div");

div.innerHTML = "<p>Это <strong>текст</strong>.</p>";

console.log(div.textContent);

// Output: "Это текст."

3. outerHTML

Описание: Возвращает или устанавливает HTML-код элемента вместе с самим элементом (включая текущий тег).

Когда использовать: Когда нужно заменить или получить элемент целиком, включая его тег.

const div = document.createElement("div");

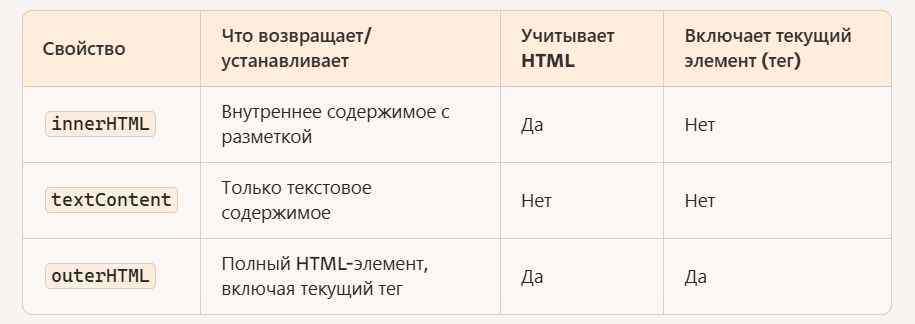
div.innerHTML = "<p>Пример текста.</p>";

console.log(div.outerHTML);

// Output: "<div><p>Пример текста.</p></div>"

div.outerHTML = "<section>Новый элемент</section>";

console.log(div);

// div не изменится в DOM, так как outerHTML удаляет текущий узел из памяти!  
  
innerHTML: Работа с разметкой внутри элемента. Используйте, если нужны вложенные теги.

textContent: Работа только с текстом, игнорируя HTML.

outerHTML: Работа с полным элементом, включая его тег.

Объект window (глобальный объект).  
window — это глобальный объект в браузере, представляющий окно или вкладку, в которой загружена веб-страница. Он содержит все свойства и методы, необходимые для взаимодействия с браузером, такими как навигация, работа с таймерами, диалогами и многое другое.

Ключевые особенности window:

Глобальный контекст:

Все переменные и функции, объявленные в глобальной области видимости, становятся свойствами объекта window.

var a = 10;

console.log(window.a); // 10

Доступность по умолчанию:

Необязательно явно указывать window, так как он уже является глобальным. Например:

alert("Привет!"); // То же самое, что window.alert("Привет!")

Объединяет важные объекты:

document: Для работы с DOM (структурой HTML-документа).

navigator: Для получения информации о браузере и устройстве.

location: Для работы с текущим URL.

history: Для навигации по истории посещений.

Основные свойства и методы window:

1. Навигация и работа с URL:

window.location: Доступ к текущему URL и возможность его изменения.

console.log(window.location.href); // Выводит текущий URL

window.location.href = "https://example.com"; // Перенаправление

window.history: Позволяет перемещаться по истории браузера.

window.history.back(); // Вернуться назад

window.history.forward(); // Вперёд

2. Работа с окнами:

window.open(url): Открывает новое окно или вкладку.

window.open("https://example.com");

window.close(): Закрывает текущее окно.

3. Таймеры и задержки:

setTimeout и setInterval: Для выполнения функций через определённое время.

setTimeout(() => console.log("Через 3 секунды"), 3000); // Однократная задержка

4. Диалоговые окна:

alert(): Показывает информационное окно.

confirm(): Показывает окно с подтверждением (OK/Cancel).

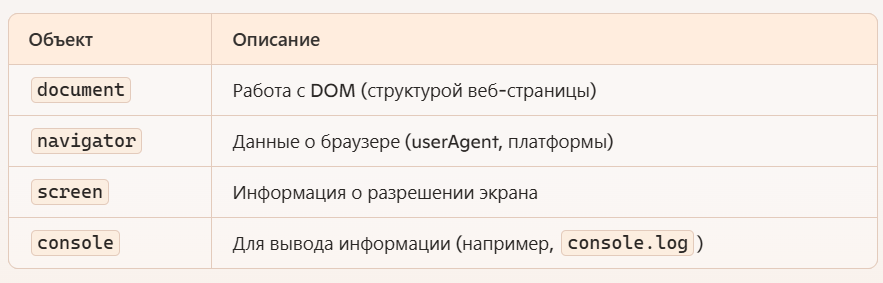
prompt(): Показывает окно для ввода текста.

const name = prompt("Как вас зовут?");

alert(`Привет, ${name}!`);

5. Размеры окна:

window.innerWidth и window.innerHeight: Возвращают размеры видимой области окна.

console.log(window.innerWidth, window.innerHeight); // Ширина и высота окна  


Работа с navigator (информация о браузере).  
navigator — это объект, предоставляющий информацию о браузере, устройстве и системе пользователя. Он входит в состав объекта window и используется для получения сведений, таких как данные об агенте пользователя (userAgent), платформе, языковых настройках и других параметрах.

Основные свойства navigator:

navigator.userAgent:

Возвращает строку, содержащую информацию о браузере и операционной системе.

console.log(navigator.userAgent);

// Пример вывода: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"

navigator.platform:

Указывает платформу устройства, на которой запущен браузер (например, Win32, Linux x86\_64, MacIntel).

console.log(navigator.platform);

// Пример: "Win32"

navigator.language:

Возвращает язык, используемый браузером (например, en-US, ru-RU).

console.log(navigator.language);

// Пример: "ru-RU"

navigator.languages:

Возвращает массив языков, поддерживаемых браузером, в порядке предпочтений.

console.log(navigator.languages);

// Пример: ["ru-RU", "en-US"]

navigator.onLine:

Возвращает true, если устройство подключено к сети, и false, если нет.

console.log(navigator.onLine);

// Пример: true

navigator.cookieEnabled:

Показывает, включены ли файлы cookie в браузере.

console.log(navigator.cookieEnabled);

// Пример: true

navigator.geolocation:

Предоставляет доступ к текущей геолокации пользователя (если разрешено). Используется для получения широты и долготы:

navigator.geolocation.getCurrentPosition(position => {

console.log(position.coords.latitude, position.coords.longitude);

});

Пример практического использования navigator:

Проверка подключения к интернету:

if (navigator.onLine) {

console.log("Вы подключены к интернету.");

} else {

console.log("Вы не подключены к интернету.");

}

Проверка текущего языка:

console.log(`Текущий язык браузера: ${navigator.language}`);

Получение информации о платформе и браузере:

console.log(`Платформа: ${navigator.platform}`);

console.log(`Агент пользователя: ${navigator.userAgent}`);

Итог:Объект navigator предоставляет доступ ко многим полезным данным о браузере и устройстве. Однако стоит помнить, что некоторые свойства (например, geolocation) требуют разрешения от пользователя.

location (перенаправление, получение URL).  
Объект location, который является свойством глобального объекта window, предоставляет интерфейс для работы с URL текущей страницы. Он позволяет получить текущий URL, а также перенаправлять пользователя на другой адрес.

Основные свойства и методы location:

1. Получение текущего URL

location.href: Возвращает полный URL текущей страницы.

console.log(location.href);

// Пример: "https://example.com/page?param=1"

location.protocol: Возвращает протокол (http:, https:, и т.д.).

console.log(location.protocol);

// Пример: "https:"

location.host: Возвращает хост (доменное имя или IP-адрес + порт, если указан).

console.log(location.host);

// Пример: "example.com"

location.pathname: Возвращает путь (часть URL после домена).

console.log(location.pathname);

// Пример: "/page"

location.search: Возвращает строку запроса (параметры после ?).

console.log(location.search);

// Пример: "?param=1"

location.hash: Возвращает фрагмент (часть URL после #).

console.log(location.hash);

// Пример: "#section"

2. Перенаправление на другой URL

location.href = "URL": Перенаправляет пользователя на указанный URL.

location.href = "https://example.com";

location.assign("URL"): Аналогично href, но оставляет текущую страницу в истории браузера.

location.assign("https://example.com");

location.replace("URL"): Перенаправляет на URL без сохранения текущей страницы в истории браузера.

location.replace("https://example.com");

location.reload(): Перезагружает текущую страницу.

Без параметров: перезагружает страницу из кеша.

С параметром true: перезагружает страницу с сервера.

location.reload(); // Перезагрузка из кеша

location.reload(true); // Полная перезагрузка

// Получение текущего URL

console.log(`Вы находитесь на: ${location.href}`);

// Перенаправление

if (location.protocol === "http:") {

location.href = "https://example.com"; // Перенаправление на https

}

// Перезагрузка страницы

document.querySelector("button").addEventListener("click", () => {

location.reload(); // Перезагрузка по кнопке

});

Итог:Объект location — это универсальный инструмент для работы с URL текущей страницы, управления навигацией и перенаправления.

history (управление историей переходов).

Объект history предоставляет методы и свойства для управления историей посещений веб-страниц в текущем окне/вкладке браузера. Он позволяет перемещаться по истории назад, вперёд или менять её без перезагрузки страницы. Это полезно для реализации кнопок "Назад", "Вперёд" и динамической работы с адресами.

Основные свойства и методы history:

1. Свойства:

history.length: Возвращает количество записей в истории текущего окна/вкладки.

console.log(history.length);

// Пример: "5" (в истории 5 страниц)

2. Методы:

a) Перемещение по истории:

history.back(): Перемещает пользователя на предыдущую страницу в истории. Аналогично нажатию кнопки "Назад" в браузере.

history.back();

history.forward(): Перемещает пользователя на следующую страницу в истории, если она существует. Аналогично кнопке "Вперёд" в браузере.

history.forward();

history.go(n): Перемещает на определённое количество шагов в истории:

n > 0: Перемещение вперёд.

n < 0: Перемещение назад.

history.go(-1); // Перейти на одну страницу назад

history.go(2); // Перейти на две страницы вперёд

b) Работа с историей (HTML5):

history.pushState(state, title, url): Добавляет новую запись в историю без перезагрузки страницы.

state: Данные, ассоциированные с записью.

title: Заголовок (не используется большинством браузеров).

url: Новый URL.

history.pushState({ page: 1 }, "Title", "/page1");

console.log(location.href);

// Output: URL изменится на "/page1"

history.replaceState(state, title, url): Заменяет текущую запись в истории, не добавляя новую.

history.replaceState({ page: 2 }, "Title", "/page2");

console.log(location.href);

// Output: URL изменится на "/page2" (без добавления новой записи)

Пример использования:

Реализация кнопок "Назад" и "Вперёд":

javascript

document.getElementById("back").addEventListener("click", () => {

history.back();

});

document.getElementById("forward").addEventListener("click", () => {

history.forward();

});

Управление URL без перезагрузки:

javascript

// Добавляем новую запись в историю

history.pushState({ page: 1 }, "Page 1", "/page1");

// Слушаем изменения состояния

window.onpopstate = (event) => {

console.log("State changed:", event.state);

};

Итог:Объект history позволяет управлять историей посещений и URL-адресами без перезагрузки страницы, что делает его полезным для одностраничных приложений (SPA) и динамических веб-сайтов.

2. Получение элементов со страницы  
1. getElementById

Получает элемент по его уникальному идентификатору (id).

const element = document.getElementById("myId");

console.log(element); // Возвращает элемент с id="myId"

2. getElementsByClassName

Возвращает коллекцию элементов, у которых указан определённый класс (class).

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

console.log(elements[0]); // Первый элемент с class="myClass"

3. getElementsByTagName

Возвращает коллекцию элементов с указанным тегом (tag).

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

console.log(elements.length); // Количество элементов <div>

4. querySelector

Возвращает первый элемент, соответствующий заданному CSS-селектору.

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

console.log(element); // Первый элемент с классом "myClass"

5. querySelectorAll

Возвращает все элементы, соответствующие заданному CSS-селектору (NodeList).

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

elements.forEach(el => console.log(el)); // Все элементы с классом "myClass"  


Методы поиска элементов:

document.getElementById(id)  
document.getElementById(id)

Описание: Возвращает элемент с указанным значением атрибута id. Если элемента с таким id нет, возвращает null.

Особенности:Работает быстро, так как id уникален.

Возвращает только один элемент.

const element = document.getElementById("header");

console.log(element); // Элемент с id="header"

document.getElementsByClassName(className)  
document.getElementsByClassName(className)

Описание: Возвращает коллекцию (HTMLCollection) всех элементов, у которых есть указанный класс.

Особенности:Возвращает "живую" коллекцию, которая автоматически обновляется при изменении DOM.

Необходимо учитывать, что возвращается массивоподобный объект, а не массив.

const elements = document.getElementsByClassName("menu-item");

console.log(elements[0]); // Первый элемент с class="menu-item"

document.getElementsByTagName(tagName)  
document.getElementsByTagName(tagName)

Описание: Возвращает коллекцию (HTMLCollection) всех элементов с указанным именем тега.

Особенности:Возвращает "живую" коллекцию.

Работает со всеми тегами, включая пользовательские.

const divs = document.getElementsByTagName("div");

console.log(divs.length); // Количество элементов <div>

document.querySelector(selector)  
document.querySelector(selector)

Описание: Возвращает первый элемент, соответствующий заданному CSS-селектору.

Особенности:Универсальный метод для поиска с использованием сложных селекторов.

Медленнее, чем getElementById, из-за использования CSS-селекторов.

const element = document.querySelector(".menu-item");

console.log(element); // Первый элемент с class="menu-item"

document.querySelectorAll(selector)

document.querySelectorAll(selector)

Описание: Возвращает все элементы, соответствующие заданному CSS-селектору, в виде статической коллекции (NodeList).

Особенности:Статическая коллекция (не обновляется при изменениях DOM).

Можно использовать метод forEach() для перебора.

const elements = document.querySelectorAll(".menu-item");

elements.forEach(el => console.log(el)); // Вывод всех элементов с class

3. Создание и добавление элементов на страницу

Создание элементов с document.createElement(tagName).  
Метод document.createElement(tagName) используется для создания нового элемента в DOM. Этот элемент создаётся, но не добавляется автоматически на страницу — его необходимо вручную вставить в DOM с помощью методов вроде appendChild или insertBefore.

Как использовать document.createElement

Создание элемента: Вы передаёте имя тега, который хотите создать (например, "div", "p", "span" и т. д.).

const newElement = document.createElement("div");

console.log(newElement); // <div></div>

Добавление содержимого: Можно добавить текст с помощью textContent, innerHTML или других методов.

newElement.textContent = "Это новый элемент";

Установка атрибутов: Для установки атрибутов можно использовать метод setAttribute или напрямую свойства объекта.

newElement.setAttribute("class", "my-class");

newElement.id = "myId"; Вставка элемента в DOM:

Новый элемент добавляется в DOM с помощью методов, таких как appendChild, prepend, или insertBefore.

document.body.appendChild(newElement); // Добавление в конец <body>

// Создаём новый div

const div = document.createElement("div");

// Добавляем текст в новый элемент

div.textContent = "Привет, мир!";

// Устанавливаем атрибуты

div.className = "greeting";

div.id = "greeting-box";

// Вставляем элемент в DOM (например, в <body>)

document.body.appendChild(div);

<div id="greeting-box" class="greeting">Привет, мир!</div>

Другие методы для работы с элементами:

createTextNode(content): Создаёт текстовый узел, который можно добавить в элемент.

const text = document.createTextNode("Только текст");

div.appendChild(text);

append: Добавляет содержимое (элементы или текст) в конец.

document.body.append("Новый текст", div);

prepend: Добавляет элемент или текст в начало.

document.body.prepend(div);

Добавление элементов:

appendChild()

append()

prepend()

insertBefore()

replaceChild()

Удаление элементов:

removeChild()

remove()  
Добавление элементов

1. appendChild()

Описание: Добавляет элемент в конец дочерних элементов родителя.

Пример:

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

const child = document.createElement("div");

child.textContent = "Новый элемент";

parent.appendChild(child);

2. append()

Описание: Добавляет текст или элемент в конец дочерних элементов. Поддерживает добавление нескольких элементов или строк одновременно.

Пример:

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

parent.append("Текст ", document.createElement("span"));

3. prepend()

Описание: Добавляет текст или элемент в начало дочерних элементов.

Пример:

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

parent.prepend("Начало ", document.createElement("span"));

4. insertBefore()

Описание: Вставляет элемент перед указанным дочерним элементом.

Пример:

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

const newChild = document.createElement("div");

const referenceChild = parent.firstChild; // Ссылка на первый дочерний элемент

parent.insertBefore(newChild, referenceChild);

5. replaceChild()

Описание: Заменяет один дочерний элемент на другой.

Пример:

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

const newChild = document.createElement("div");

const oldChild = parent.firstChild; // Ссылка на заменяемый элемент

parent.replaceChild(newChild, oldChild);

Удаление элементов

1. removeChild()

Описание: Удаляет указанный дочерний элемент.

Пример:

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

const child = parent.firstChild; // Ссылка на дочерний элемент

parent.removeChild(child);

2. remove()

Описание: Удаляет элемент непосредственно из его родителя.

Пример:

const element = document.getElementById("child");

element.remove();  
  
  
  
  


Работа с innerHTML vs createElement.

Сравнение innerHTML и createElement

И innerHTML, и createElement позволяют добавлять или изменять содержимое элементов на странице, но у каждого метода свои преимущества, недостатки и подходящие сценарии использования.

1. innerHTML

Описание: Свойство innerHTML позволяет получить или задать HTML-содержимое элемента в виде строки.

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

container.innerHTML = "<p>Это новый элемент</p>"; // Задаёт HTML-код

Плюсы:

Простота: Легко и быстро добавляет HTML-структуры.

Гибкость: Позволяет добавлять сложную разметку (включая вложенные элементы) одной строкой.

Минусы:

Безопасность: Уязвим к XSS-атакам, если вставлять данные, не очищая их.

Уничтожение событий: Если перезаписывать innerHTML, события инициализированные ранее на элементах будут удалены.

Невозможность манипуляции узлами: Работает только с текстовыми строками, не с элементами DOM.

2. createElement

Описание: Метод createElement создаёт новый DOM-элемент, который можно настроить и добавить в дерево вручную.

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

const newElement = document.createElement("p");

newElement.textContent = "Это новый элемент";

container.appendChild(newElement); // Добавляет новый элемент в контейнер

Плюсы:

Гибкость работы с элементами: Позволяет создавать элементы поэтапно и добавлять к ним свойства, события, атрибуты.

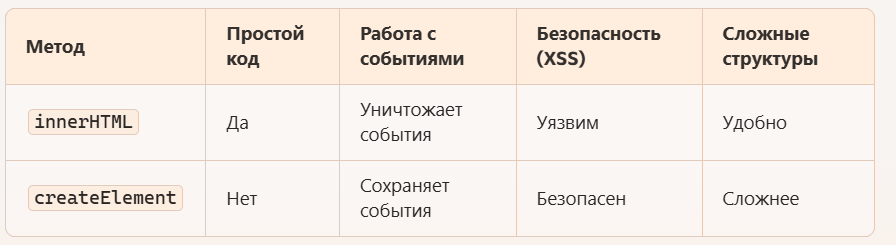
Безопасность: Не подвержен уязвимостям XSS, так как добавляет только "чистые" узлы.

Поддержка событий: Добавленные элементы не стирают ранее установленные обработчики событий.

Минусы:

Кодовая сложность: Требует больше строк кода по сравнению с innerHTML.

Сложность создания сложных структур: Для создания сложных вложенных элементов нужно создавать их по отдельности.

  
  
Когда использовать?

innerHTML:

Если нужно быстро добавить или заменить большое количество HTML-кода.

Если не требуются события для элементов.

createElement:

Если нужно программно создать элементы с настройками (атрибутами, обработчиками событий и т.д.).

Если требуется безопасная вставка данных.  
  
4. Работа с CSS

Изучить:

Изменение стилей через style  
Изменение стилей элементов через свойство style в JavaScript — это удобный способ динамически управлять внешним видом страницы. Свойство style позволяет напрямую задавать CSS-стили элемента, минуя необходимость работы с внешними или встроенными CSS.

Как изменить стиль через style?

Обращение к элементу: Используйте методы getElementById, querySelector или другие для получения элемента.

Изменение стиля:Используйте свойство style для указания CSS-свойств.

Свойства пишутся в стиле CamelCase (например, backgroundColor вместо background-color).

1. Изменение одного стиля

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

element.style.color = "blue"; // Устанавливаем синий цвет текста

element.style.fontSize = "20px"; // Устанавливаем размер шрифта

2. Изменение нескольких стилей

Можно устанавливать несколько свойств поочерёдно:

element.style.backgroundColor = "lightgray";

element.style.border = "1px solid black";

element.style.padding = "10px";

3. Очистка стиля

Чтобы удалить стиль, можно присвоить пустую строку:

element.style.color = ""; // Удаляет установленное значение цвета

Пример: Динамическое изменение стиля при событии

const button = document.getElementById("changeStyleButton");

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

button.addEventListener("click", () => {

box.style.backgroundColor = "yellow";

box.style.width = "150px";

box.style.height = "150px";

box.style.borderRadius = "10px";

});

HTML:

<button id="changeStyleButton">Изменить стиль</button>

<div id="box" style="width: 100px; height: 100px; background-color: red;"></div>

Важно учитывать:

Приоритеты стилей:Стили, указанные через style, имеют более высокий приоритет, чем стили из внешнего CSS.

Динамическая работа: Используйте classList.add или classList.remove для динамического управления классами, если требуется изменить сразу много стилей.

box.classList.add("new-style"); // Добавляет класс с CSS-правилами

Когда использовать style:

Когда нужно изменить одно или несколько свойств элемента.

Когда изменения стилей зависят от действий пользователя или событий (например, клик, прокрутка).

Если изменения не требуют сложных комбинаций стилей, лучше использовать style. Если требуется массовое стилизование — используйте классы.

Добавление и удаление классов:  
Для управления классами элементов в JavaScript используются методы объекта classList. Они позволяют добавлять, удалять, переключать и проверять наличие классов в элементе. Это удобный и современный способ работы с классами, который заменяет более устаревшие подходы вроде изменения свойства className.

Методы classList

1. Добавление классов: add()

Добавляет один или несколько классов к элементу. Если класс уже существует, он не будет добавлен повторно.

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

element.classList.add("new-class", "highlight");

console.log(element.className); // Вывод: "new-class highlight"

2. Удаление классов: remove()

Удаляет один или несколько классов из элемента. Если указанный класс отсутствует, ничего не происходит.

element.classList.remove("highlight", "old-class");

console.log(element.className); // Например, "new-class"

3. Переключение класса: toggle()

Добавляет класс, если его нет, или удаляет его, если он есть. Также можно передать второй аргумент (true или false) для явного включения или выключения класса.

element.classList.toggle("active"); // Переключает класс

element.classList.toggle("active", true); // Гарантированно добавляет класс

element.classList.toggle("active", false); // Гарантированно удаляет класс

4. Проверка наличия класса: contains()

Проверяет, имеет ли элемент указанный класс. Возвращает true или false.

if (element.classList.contains("new-class")) {

console.log("Класс существует!");

}

5. Просмотр и управление всеми классами:

Можно работать со всеми классами элемента через свойство className (строка всех классов).

console.log(element.className); // Выводит список классов как строку

Пример кода: Добавление и удаление классов

const button = document.getElementById("toggleButton");

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

button.addEventListener("click", () => {

box.classList.toggle("highlight"); // Переключаем класс "highlight"

});

HTML:

<button id="toggleButton">Переключить стиль</button>

<div id="box" class="box">Это элемент</div>

CSS:

css

.box {

width: 100px;

height: 100px;

background-color: lightgray;

}

.highlight {

background-color: yellow;

}

Советы: Используйте classList для удобства и читабельности кода.

Использование нескольких методов add, remove, toggle вместо манипуляций строкой через className снижает вероятность ошибок.

classList.add('new-class')

classList.remove('old-class')

classList.toggle('hidden')

classList.contains('active')