**Введение**

Документ содержит подробное описание используемых инструментов пользовательского интерфейса lessons. Предназначен для разработчиков.

**HTML**

В качестве языка разметки используется HTML 5. Список основных тэгов, используемых в проекте:

* *<!DOCTYPE html>:* Определяет тип документа как HTML 5.
* *<html>:* Обозначает начало и конец HTML-документа.
* *<head>:* Содержит метаданные о странице, такие как заголовок, метаданные и стили.
* *<title>:* Определяет заголовок страницы, который отображается в заголовке окна браузера и в списке закладок.
* *<body>:* Содержит содержимое страницы, которое отображается в окне браузера.
* *<header>:* Определяет область заголовка страницы, которая обычно содержит логотип, навигацию и другие элементы заголовка.
* *<nav>:* Определяет навигационную область, которая обычно содержит ссылки на другие страницы сайта.
* *<section>:* Определяет раздел страницы, который может содержать отдельный блок контента.
* *<article>:* Определяет статью или самостоятельный блок контента, который может быть повторно использован на других страницах.
* *<aside>:* Определяет боковой блок контента, который не является основным содержанием страницы.
* *<footer>:* Определяет нижний колонтитул страницы, который обычно содержит контактную информацию, ссылки на социальные сети и другие элементы.
* *<h1> — <h6>:* Определяют заголовки разных уровней.
* *<p>:* Определяет абзац текста.
* *<ol>:* Определяет упорядоченный список.
* *<ul>:* Определяет неупорядоченный список.
* *<li>:* Определяет элемент списка.
* *<div>:* Определяет блок контента, который можно использовать для группировки элементов.
* *<span>:* Определяет встроенный элемент, который не влияет на структуру документа.

**CSS**

В проекте используются следующие основные CSS-свойства

* **margin**: определяет отступы между дочерними элементами внутри flex-контейнера.
* **padding**: определяет внутренние отступы вокруг дочерних элементов внутри flex-контейнера.
* **background-color**: определяет цвет фона элемента.
* **border**: определяет границы вокруг дочерних элементов внутри flex-контейнера.
* **width**: определяет ширину дочернего элемента внутри flex-контейнера.
* **height**: определяет высоту дочернего элемента внутри flex-контейнера.
* **max-width**: определяет максимальную ширину дочернего элемента внутри flex-контейнера.
* **max-height**: определяет максимальную высоту дочернего элемента внутри flex-контейнера.
* **min-width**: определяет минимальную ширину дочернего элемента внутри flex-контейнера.
* **min-height**: определяет минимальную высоту дочернего элемента внутри flex-контейнера.
* **visibility**: определяет видимость дочернего элемента внутри flex-контейнера
* **position**: определяет положение элемента на странице.
* **border**: определяет границу элемента.
* **font-family**: определяет шрифт текста.
* **font-size**: определяет размер шрифта.
* **font-style**: определяет наклон шрифта.
* **font-weight**: определяет жирность шрифта.
* **text-align**: выравнивает текст по горизонтали.
* **top, right, bottom, left**: используются для точного позиционирования элементов.
* **z-index**: определяет порядок наложения элементов.
* **outline**: это свойство предназначено для короткой записи следующих CSS-свойств:
  + outline-color
  + outline-style
  + outline-width

Свойство outline может быть задано одним, двум или тремя перечисленными ниже значениями. Порядок следования значений не имеет значения. Как и в случае с другими сокращёнными свойствами

* **transition**: позволяет определять переходное состояние между двумя состояниями элемента

Для адаптивного дизайна использовалась модель компоновки Flexbox.  
Flexbox — это модуль CSS, который позволяет удобно управлять расположением, порядком, размерами и отступами между элементами веб-страницы.

* **display**: определяет, как элемент будет отображаться внутри flex-контейнера –
* **flex-direction**: указывает направление потока дочерних элементов внутри flex-контейнера
* **justify-content**: регулирует выравнивание дочерних элементов вдоль главной оси контейнера
* **flex-wrap**: определяет, могут ли дочерние элементы переноситься на следующую строку при нехватке места

Для адаптирования UI под различные устройства используются media запросы. Media-запросы — это инструмент в CSS, который позволяет адаптировать внешний вид веб-страницы в зависимости от размера экрана устройства, на котором она отображается. Они позволяют создавать адаптивный дизайн, который будет корректно отображаться на различных устройствах, таких как настольные компьютеры, ноутбуки, планшеты и смартфоны.

Media-запросы используются для определения условий, при которых должен применяться определённый стиль. Эти условия могут быть связаны с размером экрана, разрешением, ориентацией и другими характеристиками устройства.

Синтаксис media-запроса:

*@media <условие> {*

*<стили>*

*}*

<условие> — это условие, при котором должен применяться стиль, а <стили> — это набор стилей, которые будут применяться при выполнении условия.

Пример media-запроса:

*@media screen and (max-width: 600px) {*

*/\* Стили для экранов с шириной меньше 600 пикселей \*/*

*}*

**CSS Variables**: Для определения общих свойств (в проекте – цветовой палитры) используется функционал CSS переменных. CSS ключевое слово var используется для создания пользовательских CSS переменных. Это позволяет определить значения, которые могут быть многократно использованы в CSS-правилах, что обеспечивает легкость в поддержке и изменении стилей.

**Объявление пользовательской переменной:**

Чтобы создать пользовательскую переменную, используется следующий синтаксис:

*:root {*

*--main-color: #ff0000;*

*}*

Здесь, с помощью селектора :root, выбирается корневой элемент документа (обычно <html>), а --main-color - имя переменной, которая будет содержать значение #ff0000.

**Использование переменной:**

Для использования переменной, объявленной с помощью var(), достаточно поместить ее внутри функции var() и указать имя переменной:

*.my-element {*

*color: var(--main-color);*

*}*

Здесь значение переменной --main-color будет применяться к свойству color для всех элементов с классом .my-element.

**Наследование значений:**

Переменные также могут быть унаследованы. Если переменная не определена для определенного элемента, она будет наследовать значение от своего родителя.

**Предварительные значение:**

Если переменная не определена, все свойства, использующие ее, будут игнорироваться. Однако, можно предоставить значение по умолчанию, указав его после имени переменной:

*.my-element {*

*color: var(--main-color, #000000);*

*}*

В этом случае, если переменная --main-color не определена, будет использовано значение #000000.

Использование пользовательских переменных с var в CSS позволяет легко изменять стили веб-страницы, управлять цветами, шрифтами, отступами и многими другими параметрами без необходимости переписывать код.

**JavaScript и DOM**

DOM (Document Object Model) - это структура, которая представляет собой иерархическое дерево объектов, представляющих все элементы веб-страницы. Каждый узел в DOM - это объект, представляющий часть содержимого страницы, такой как элементы HTML, атрибуты и текстовое содержимое.

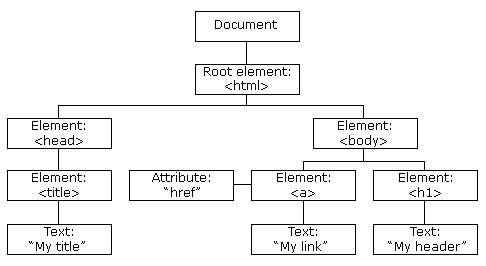
DOM состоит из узлов, которые являются различными типами объектов:

Элементы (element nodes) - представляют собой HTML-теги страницы, такие как <div>, <p>, <ul> и другие.

Атрибуты (attribute nodes) - атрибуты элементов, такие как id, class, src и т.д.

Текстовые узлы (text nodes) - содержат текстовое содержимое элемента.

JavaScript может взаимодействовать с DOM, используя специальные методы и свойства. С помощью JavaScript можно изменять содержимое страницы, добавлять новые элементы, удалять или изменять существующие элементы.



JavaScript может взаимодействовать с DOM следующим образом:

* Доступ к элементам DOM с помощью методов getElementById, getElementsByClassName, getElementsByTagName, querySelector и querySelectorAll.
* Изменение содержимого с помощью свойства innerHTML или innerText. **innerHTML** - свойство элемента, содержащее HTML-код этого элемента и его потомков. Это свойство позволяет программно манипулировать содержимым элемента. **innerText** - свойство элемента, содержащее текстовый контент этого элемента и его потомков без учёта тегов форматирования
* Добавление и удаление элементов с помощью методов createElement, removeElement с последующей их вставкой в DOM древо с помощью appendChild, insertBefore.
* Изменение стилей с помощью свойства style, добавляя классы с помощью методов addClass и removeClass.
* Обработка событий посредством назначения обработчиков событий на элементы с помощью методов addEventListener.
* Создание анимации на основе изменения свойств стилей или позиций элементов.

Взаимодействие происходит посредством взаимодействия со следующими основными глобальными объектами:

* **document**: относится к объекту, который представляет собой текущее окно браузера или веб-страницу. Этот объект предоставляет методы и свойства для взаимодействия с элементами страницы, такими как теги, стили, скрипты. Он позволяет манипулировать содержимым страницы, добавлять или удалять элементы, обрабатывать события мыши и клавиатуры.
* **window**: объект, представляющий собой окно, содержащее DOM документ. Содержит свойства для работы с URL текущей вкладки браузера:
  + **window.location**: cвойство только для чтения. возвращает объект location с информацией о текущем расположении документа
  + **window.history**: является свойством только для чтения, ссылающимся на объект history, который предоставляет интерфейс для манипулирования историей сессии браузера (страницы, посещённые в текущей вкладке или фрейме, который был загружен на текущей странице)

Выборка элементов из DOM дерева для последующей работы с ними происходит с помощью следующих методов:

* **getElementById**: возвращает ссылку на элемент по его идентификатору (ID); идентификатор является строкой, которая может быть использована для идентификации элемента; она может быть определена при помощи атрибута id в HTML или из скрипта. Если совпадений не найдено, возвращает значение null.
* **querySelector**: метод, который возвращает первый элемент (Element) документа, который соответствует указанному селектору или группе селекторов. Если совпадений не найдено, возвращает значение null.
* **getElementByClassName:** возвращает ссылку на первый элемент по его классу (class); . Если совпадений не найдено, возвращает значение null.

Для определения event-ов в ответ на действия пользователя используется специальный метод **addEventListener**, который используется для прослушивания определенных событий на элементе DOM и выполнения определенной функции (callback-а) при возникновении этих событий.

Основными параметрами метода **addEventListener** являются:

**Тип события:** Это строка, указывающая тип события, на которое нужно прослушивать. Наиболее распространенные события:

* click
* focus
* drag
* mousemove
* mouseover
* vouseleave
* dblclick
* contextmenu
* keydown
* keypress
* copy
* paste
* resize
* submit

**Обработчик события:** функция, которая будет вызываться при возникновении указанного события. Обычно это анонимная функция или ссылка на уже существующую функцию. Принимает в качестве аргумента объект события, с помощью которого можно получить сведения об объекте, с которым было связано событие. Эти сведения хранятся в объекте target.

* **target**: свойство интерфейса Event является ссылкой на объект, который был инициатором события.
* **target.closest**: функция, которая возвращает ближайший родительский элемент (или сам элемент), который соответствует заданному CSS-селектору или null, если таковых элементов вообще нет.
* **target.classList** — это readonly свойство, которое содержит текущую коллекцию всех атрибутов class элемента target.
  + **add**:метод, который добавляет заданные токены в список, исключая те, которые уже присутствуют.
  + **remove**:метод, который удаляет указанные токены из списка.
  + **toggle**: метод, который удаляет существующий токен из списка и возвращает false. Если токен не существует, он добавляется, и функция возвращает true.

**Использование фазы захвата:** Необязательный параметр, принимающий логическое значение (true/false) и определяющий, будут ли обработчики вызываться на фазе захвата или на фазе всплытия. По умолчанию установлен в false (используется фаза всплытия).

Пример использования **addEventListener**:

*document.getElementById(‘example-btn’).addEventListener('click', function() {*

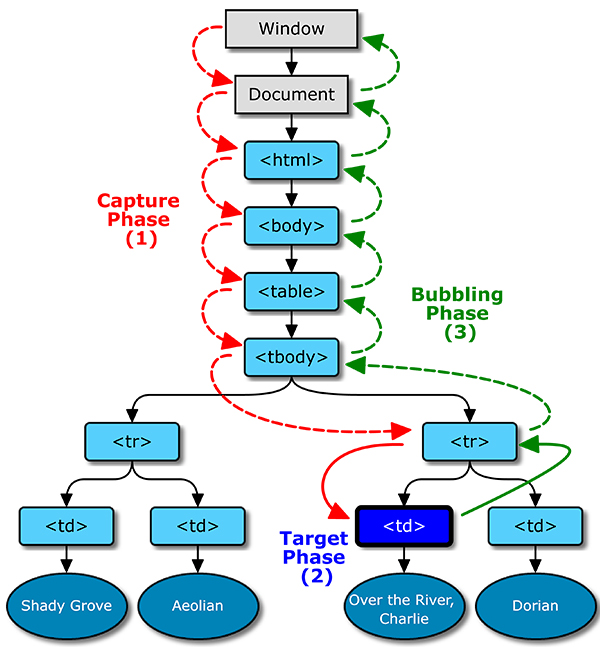
*alert('Кнопка была нажата!');*

*});*

В этом примере при нажатии кнопки с идентификатором "example-btn" будет вызвана анонимная функция, которая показывает всплывающее окно с сообщением "Кнопка была нажата!".

Можно добавлять несколько обработчиков событий на один элемент, таким образом реализуя множественную функциональность при возникновении одного и того же события.

**Bubbling и Capturing.** Обработка событий в JavaScript основана на концепциях всплытия и погружения, которые описывают порядок, в котором события проходят через элементы DOM на веб-странице. Всплытие (bubbling) и погружение (capturing) представляют собой два этапа прохождения события от корневого элемента к целевому элементу и обратно.



**Фаза погружения (capturе phase):** В этой фазе событие начинает свой путь от корневого элемента (document), спускаясь к целевому элементу, на котором произошло событие.

Обработчики событий, добавленные с параметром useCapture: true (или третьим параметром addEventListener), вызываются на этой фазе.

**Целевая фаза (target phase):** Событие достигает целевого элемента, на котором оно было запущено, и вызываются соответствующие обработчики событий.

**Фаза всплытия (Bubbling phase):** После выполнения обработчиков событий на целевом элементе, событие начинает всплывать обратно к корневому элементу.Обработчики событий, добавленные без параметра useCapture или с параметром useCapture: false, вызываются на этой фазе.