5 занятие. DOM API

Теоретическая часть

- Глобальный объект window
- Дерево элементов
- Поиск, изменение объектов в DOM
  - Поиск DOM-элементов
  - Удаление элемента
  - Работа с содержимым элемента
  - Изменение атрибутов элементов
  - Свойство style
  - Работа с классами classList
  - Создание элемента
  - Клонирование элемента
- Свойство children, дочерние элементы
- Шаблоны

Практика по DOM

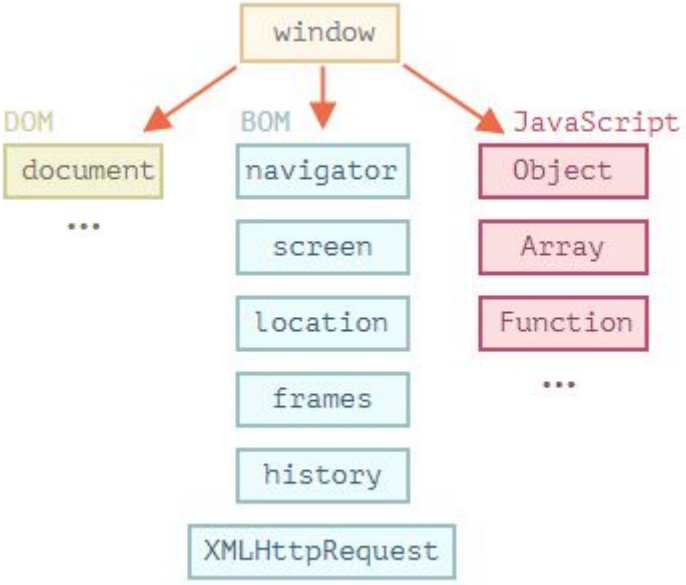
Статьи:

Руководство по DOM - DOM | MDN
Console - Интерфейсы веб API | MDN
Браузерное окружение, спецификации
Руководство по DOM
Манипуляции с DOM на чистом JS
Выразительный JavaScript: Document Object Model (объектная модель документа)
Об объектной модели документа | MDN

Живые и неживые коллекции в JavaScript — Блог HTML Academy

Глобальный объект window

JavaScript в браузере:



Корневой объект **window** выступает в 2 ролях:

1. глобальный объект для JavaScript-кода
2. представляет собой окно браузера и располагает методами для управления им.

DOM — работа с элементами страницы (в том числе SVG-графикой)
BOM — работа с браузером

Document Object Model, сокращённо DOM – *объектная модель документа*, которая представляет все содержимое страницы в виде объектов, которые можно менять.

Объект **document** – основная «входная точка». С его помощью мы можем что-то создавать или менять на странице.

Объектная модель браузера (**Browser Object Model**, BOM) – это дополнительные объекты, предоставляемые браузером, чтобы работать со всем, кроме документа.

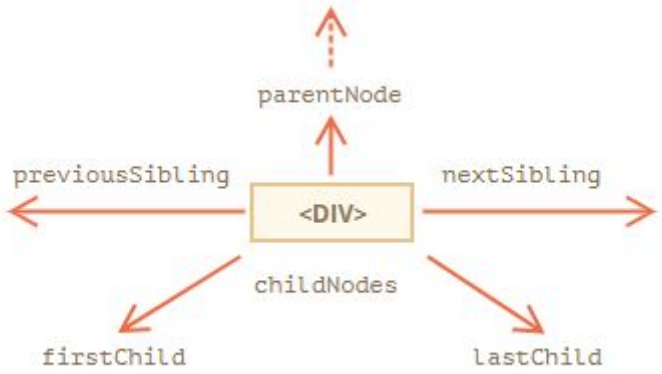
DOM Living Standard на https://dom.spec.whatwg.org

Дерево элементов

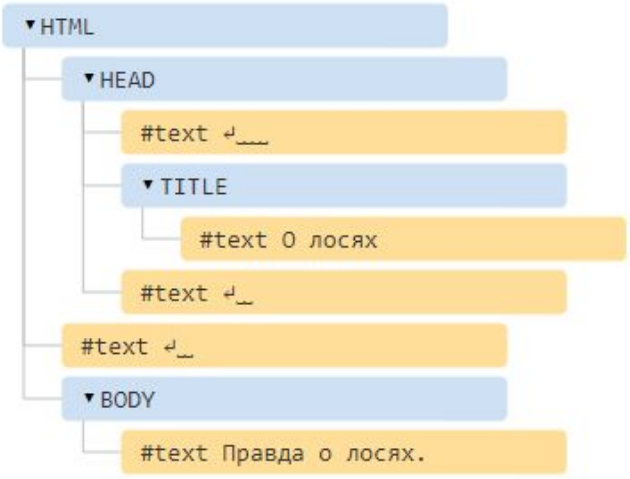
JavaScript особым образом воспринимает разметку: элементы здесь не строки, которые мы пишем в HTML-файлах, а объекты.

Дерево

– **узел(node)** — любой элемент дерева:
  – **родитель (parent)** — элемент, из которого растет узел
  – **дети (child)** — элементы, которые растут из узла
– **корень (root)** — элемент из которого растет дерево, элемент без родителей
– **лист** — элемент дерева, который не имеет детей



Дерево может иметь только один корень. У каждого элемента должно быть не более одного родителя.



Древовидная структура JS объекта:

```
var document = {
        html: {
                head: {
                        title: 'О лосях'
                },
                body: {
                        h1: 'Вся правда о лосях',
                        a: {
                                href: 'index.html',
                                text: 'Link text'
                        }
                }
        }
};
```

методы DOM API - это группа методов, которые позволяют взаимодействовать с какой-то частью программы или интерфейса.

DOM API - это все методы, которые позволяют что-то делать с DOM-элементами.

Поиск, изменение объектов в DOM

Поиск DOM-элементов

- **поиск по ID, тегу или названию класса**

  element = document.**getElementById**(id);
  - находит первый элемент с id

  elements = document.**getElementsByClassName**(names);
  - находит множество (коллекцию) элементов с классом

  elements = document.**getElementsByName**(name);
  - находит коллекцию по указанному name

  elements = document.**getElementsByTagName**(name);
  - находит коллекцию по указанному тегу

- **поиск по CSS-селектору**

  element = document.**querySelector**(selectors);
  - возвращает первый элемент документа, который соответствует указанному селектору или группе селекторов.

  elementList = document.**querySelectorAll**(selectors);
  - возвращает список, содержащий все найденные элементы документа, которые соответствуют указанному селектору.

DOM-коллекция, полученная через querySelectorAll похожа на массив, но им не является. Поэтому ещё одно название таких коллекций — псевдомассив. Перебрать коллекцию можно классическим циклом for.

Удаление элемента

Удалять элементы со страницы можно разными способами, один из самых простых — вызов метода remove на элементе, который нужно удалить.

```
element.remove();
```

Метод из примера выше удалит element из DOM

Работа с содержимым элемента

innerHTML

Свойство интерфейса innerHTML устанавливает или получает HTML разметку дочерних элементов.

```
element.innerHTML = htmlString;
```

Установка значения innerHTML удаляет всё содержимое элемента и заменяет его на узлы, которые были разобраны как HTML, указанными в строке htmlString.

```
document.body.innerHTML = "";    // Заменяет содержимое body на пустую строку
```

---

textContent

Свойство textContent хранит в себе текстовое содержимое элемента.

```
let paragraph = document.querySelector('p');
console.log(paragraph.textContent);
```

Свойство textContent предназначено только для текста, если записать туда HTML-теги, браузер их не поймёт.

Новое значение textContent переписывает всё содержимое элемента

Дополнительно:
.insertAdjacentHTML
.insertAdjacentText
.insertAdjacentElement

Изменение атрибутов элементов

Значением атрибута можно управлять с помощью одноимённого свойства DOM-элемента:

```
var picture = document.createElement('img');
picture.src = 'images/picture.jpg';
```

Таким же образом добавим изображению альтернативный текст, то есть описание фотографии.

```
picture.alt = 'Непотопляемая селфи-палка';
```

Названия атрибутов тегов и свойств DOM-элементов часто (но не всегда) совпадают. Например, чтобы обратиться к классу, нужно воспользоваться свойством className, а чтобы прочитать атрибут for тега label - свойством htmlFor

element.className = 'newclass';

Но удобнее работать с классами с помощью объекта **classList**, так как у одного элемента может быть более 2х классов.

.getAttribute - возвращает значение указанного атрибута элемента
.setAttribute - добавляет новый атрибут или изменяет значение существующего

Свойство style

```
элемент.style.свойствоCSS = "значение";
```

добавление свойств CSS:

```
let topBtn=document.querySelector(".btn-top");
topBtn.style.display="none";
```

Стили, заданные с помощью свойства style, работают так же, как если бы их указали в разметке в атрибуте style самого элемента. Они имеют больший приоритет, чем CSS-правила из файла со стилями.

Используйте style только в тех случаях, когда с помощью классов задачу решать неудобно.

Работа с классами classList

Среди свойств DOM-элементов — объект **classList**. Он содержит методы для управления классами DOM-элемента, в том числе и метод add(). С его помощью мы можем указать, какой класс хотим добавить элементу.

элемент.**classList.remove**('класс'); - удалить класс
элемент.**classList.add**('класс'); - добавить класс
элемент.**classList.toggle**('класс'); - переключатель класса

Обратите внимание, что точку перед названием класса ставить не нужно.

Чтобы проверить, есть ли у элемента класс, используем метод **classList.contains**

document.querySelector('.page').classList.remove('light-theme'); - убрать класс у элемента

Задача: есть список из 5 элементов и массив с информацией о товаре вида:

```
var assortmentData = [
  {
    inStock: true,
    isHit: false
  },
  {
    inStock: false,
    isHit: false
  },
  {
    inStock: true,
    isHit: true
  },
  {
    inStock: true,
    isHit: false
  },
  {
    inStock: false,
    isHit: false
  }
];
```

Нужно повесить соответствующие классы на товары массива:
inStock: Товар в наличии — .good--available. Недоступный товар — .good--unavailable.
isHit: Хит продаж — .good--hit.

Создание элемента

document.createElement

```
var div = document.createElement('div');
```

Создаёт новый элемент с заданным тегом.

.append

```
элемент-родитель.append(добавляемый-элемент);
```

Метод .append добавляет набор объектов в конец (после последнего потомка) родителя.

```
var div=document.createElement("div");
var p=document.createElement("p");
p.innerHTML="qweqe";
div.append(p);
document.body.append(div);
```

Метод append не копирует элементы, а **перемещает**. Если указать в скобках элемент, который уже есть в разметке, этот элемент исчезнет со своего прежнего места и появится там, куда его добавил метод append. Получить таким образом несколько элементов не выйдет.

Задача: добавить циклом 10 абзацев в блок div.ten

Клонирование элемента

.cloneNode

Клонирует (копирует) имеющийся на странице элемент

```
element.cloneNode(false); // Вернёт склонированный элемент без вложенностей
```

У этого метода есть аргумент — булево значение:

- **false:** будет скопирован сам элемент со своими классами и атрибутами, но **без дочерних элементов**.

- **true:** клонируется сам элемент **вместе со всеми вложенностями** - атрибуты, классы и текстовое содержимое всех вложенностей. Такое клонирование называется **глубоким**.

Лучше всегда передавать булево значение, чтобы избежать непредсказуемого поведения в программах.

Задача: Создать на странице 3 блока. Клонировать и добавить на страницу 4й

Свойство children, дочерние элементы

querySelectorAll находит необходимые элементы один раз, фиксирует их в переменной и изменения в DOM на неё никак не влияют. Такая коллекция называется **статичной**.

Живые коллекции элементов называются **динамическими**, тк они реагируют на изменения в DOM. Если один из элементов коллекции будет удалён из DOM, то он пропадёт и из коллекции.

Первый способов получить живую коллекцию - **children**.

```
parentElement.children;
```

```
var elList = parentElement.children;
```

В результате, elList - живая коллекция, состоящая из дочерних элементов узла parentElement, и если у узла детей нет, она будет пустой. Определить это можно, обратившись к свойству length, которое содержит в себе количество элементов в коллекции.

---

parent.firstElementChild

- возвращает первый дочерний элемент объекта (Element) или null если дочерних элементов нет.

```
var childNode = elementNode.firstElementChild;
```

---

parent.lastElementChild

- возвращает последний дочерний элемент объекта или null если нет дочерних элементов.

```
var element = node.lastElementChild;
```

Задача: Что выведет данный код?

```
<ul id="foo">
        <li>First (1)</li>
        <li>Second (2)</li>
        <li>Third (3)</li>
</ul>
<script>
```

```
        var foo = document.getElementById('foo');
         console.log(foo.lastElementChild.textContent);
</script>
```

---

parent.removeChild

- удаляет выбранный дочерний элемент из DOM. Элемент может продолжить существовать, если его предварительно записать в переменную или массив

---

parent.appendChild
перемещает / добавляет потомка к элементу родителю последним

---

parent. insertBefore(el1, el2)

добавляет элемент 1 перед элементом 2. Если вместо el2 написать null, то элемент станет последним

`parent.insertBefore(element, null)`

---

parent.replaceChild(el1, el2)

заменяет элемент. Возвращает значение - элемент, который был заменен в результате операции, чтобы производить с ним какие-то действия дальше.

Задача: пользователь вводит строку вроде “red, green, blue”. Нужно найти цвета в этой строке и создать на странице блоки этих цветов.

Задача: На странице есть какое-то количество элементов. Напишите функцию, удаляющую все эти элементы

Шаблоны

<template> - HTML | MDN
Элемент "template"

Было бы удобно, если бы вся необходимая разметка для будущих элементов уже где-то хранилась. Оставалось только подправить содержимое под каждый элемент. И это можно сделать с помощью тега **template**.

Он хранит в себе шаблон для будущих элементов. Тег template находится там же, где и вся разметка сайта, только его содержимое не отображается на странице.

Обычно каждому тегу template дают уникальное название и записывают в атрибут id.

```
var template = document.querySelector('#text-template');  // Нашли template в документе
var content = template.content;  // Получили содержимое, фрагмент
var text = content.querySelector('.text');  // Нашли нужный шаблон
```

Эту запись можно сократить. Например, записать в отдельную переменную контент, а в другую искомый шаблон.

```
var textTemplate = document.querySelector('#text-template').content;
var text = textTemplate.querySelector('.text');
```

Такая запись удобней, тк сам template обычно не используют. Работа ведётся с его контентом и шаблоном, который в этом контенте хранится.

В шаблоне можно менять текст, классы, а затем добавлять элементы на страницу.

Клонирование и вставка элементов

Нельзя просто так взять один элемент и добавить его много раз на страницу. Вставка сработает только один раз.

Шаблон для карточки хранится в теге template.

```
<body>
  …
  <template id="element-template">
    <div class="el">
      <span></span>
    </div>
  </template>
</body>
```

Задача: Вставить содержимое шаблона на страницу.

Решение:
Для вставки элементов на страницу будем использовать метод **appendChild**.
Он добавляет указанные элементы в конец родительского блока.

Необходимо клонировать внутреннюю часть шаблона и вставить эту копию на страницу сколько угодно раз.

Для это нужно использовать метод **cloneNode**:

```
// Контейнер для карточек
var pool = document.querySelector('.pool');

// Получаем шаблон карточки
var template = document.querySelector('#element-template').content;
var element = template.querySelector('div');

// Клонируем и наполняем элемент

 var clonedElement = element.cloneNode(true);
  clonedElement.children[0].textContent = 3;
  pool.appendChild(clonedElement);
```

Также можно использовать цикл for для многократного добавления элементов

```
for (var i = 1; i <= 10; i++) {
  var clonedElement = element.cloneNode(true);
  clonedElement.children[0].textContent = i;
  pool.appendChild(clonedElement);
}
```

Задача: сверстать карточку товара, создать ее шаблон. Добавить 7 таких карточек с помощью JS

Практика по DOM

1. Задача:
Есть интернет-магазин с готовой вёрсткой. Нужно показывать в интерфейсе актуальную информацию о товарах: спецпредложения и наличие на складе.

Данные приходят в виде массива объектов **catalogData**. Каждый объект соответствует одному товару и содержит свойства isAvailable (в наличии товар или нет) и isSpecial (является ли товар спецпредложением или нет).

Для каждого состояния товара есть соответствующий класс:

product--available для товара в наличии;
product--unavailable соответствует товару, которого в наличии нет;
product--special для спецпредложения.

Массив данных о каждом товаре:

```
let catalogData = [
  {
    isAvailable: true,
    isSpecial: false
  },
  {
    isAvailable: false,
    isSpecial: false
  },
  {
    isAvailable: true,
    isSpecial: true
  },
  {
    isAvailable: true,
    isSpecial: false
  },
  {
    isAvailable: false,
    isSpecial: false
  }
];
```

---

2. Задача:

На сайте магазина мороженого надо отображать актуальное состояние товаров: «в наличии», «нет в наличии», «хит».

Данные по продуктам хранятся в массиве с объектами **assortmentData**, каждый объект соответствует одному товару и содержит свойства:

- **inStock**. Если значение true — мороженое в наличии,
  если false — товара в наличии нет.
- **isHit**. Если значение true — мороженое самое популярное среди покупателей.

Каждому состоянию товара соответствует специальный класс:

- Товар в наличии — **good--available**.
- Недоступный товар — **good--unavailable**.
- Хит продаж — **good--hit**.

Оформи код в виде функции **updateCards**, которая принимает на вход массив с данными. Вызови её, передав assortmentData.

```
var assortmentData = [
 {
   inStock: true,
   isHit: false
 },
 {
   inStock: false,
   isHit: false
 },
 {
   inStock: true,
   isHit: true
 },
 {
   inStock: true,
   isHit: false
 },
 {
   inStock: false,
   isHit: false
 }];
```

3. Задача: нужно добавить карточки товаров на страницу магазина. В этот раз будет приходить полная информация. Данные будут содержать название продукта, его цену, изображение, доступность для заказа и прочую информацию.

Верстальщик показал, как должны быть свёрстаны карточки товаров. Нужно ориентироваться на эти карточки, создавая свою, а затем удалить и добавить все карточки самостоятельно.

Данные о всех товарах хранятся в массиве **cardsData**. Информация о товаре представляет собой объект, каждое свойство которого описывает характеристику товара:

imgUrl — адрес изображения;
text — название товара;
price — цена;
isAvailable - наличие товара;
isSpecial - является ли товар спецпредложением;
specialPrice - цена товара по спецпредложению

```
cardsData = [
  {
    isAvailable: true,
    imgUrl: 'https://www.flaticon.com/svg/static/icons/svg/3612/3612065.svg',
    text: 'Костюм зомби',
    price: 2000,
    isSpecial: false
  },
  {
    isAvailable: false,
    imgUrl: 'https://www.flaticon.com/svg/static/icons/svg/2534/2534536.svg',
    text: 'Костюм огра',
    price: 1500,
    isSpecial: false
  },
  {
    isAvailable: true,
    imgUrl: 'https://www.flaticon.com/svg/static/icons/svg/3612/3612089.svg',
    text: 'Костюм монстра',
    price: 2500,
    isSpecial: false
  },
 {
    isAvailable: true,
    imgUrl: 'https://www.flaticon.com/svg/static/icons/svg/3612/3612274.svg',
    text: 'Костюм смерти',
    price: 2000,
    isSpecial: false
  },

  {
```

```
    isAvailable: true,
    imgUrl: 'https://www.flaticon.com/svg/static/icons/svg/3612/3612347.svg',
    text: 'Костюм вампира',
    price: 4000,
    isSpecial: true,
    specialPrice: 1000
  }
];
```

Алгоритм:
Напишите функцию, которая будет создавать элемент по заданному тегу, классу и содержимому. Добавляйте содержимое безопасным способом с помощью textContent. Добавьте в новую карточку при помощи функции заголовок, картинку и стоимость.

Оптимизируйте код, создав новую функцию, которая будет создавать всю карточку товара по данным объекта.

Оптимизируйте код, добавляя карточки товаров из массива объектов.

---

4. Задача

Помнишь магазин мороженого? Нужно создать карточки товаров, основываясь на данных, полученных с сервера.

Данные — массив объектов **cardsData**, один элемент соответствует одному товару. У каждого объекта есть следующие свойства:

- **inStock** — доступность товара. Если значение true — товар доступен (для такого продукта верстальщик подготовил класс good--available), если false — продукта нет в наличии (товар с классом good--unavailable).
- **imgUrl** — ссылка на изображение товара.
- **text** — название продукта.
- **price** — цена.
- **isHit** — является ли товар хитом продаж. Если значение true — продукт «хитовый». Для такого товара подготовлен класс good--hit.
- **specialOffer** — специальное предложение, которое есть только у хита продаж. Должно находиться в абзаце с классом good-special-offer и быть самым последним дочерним элементом в карточке.

Вот пример вёрстки одной карточки в каталоге:

```
<ul class="goods">
  <li class="good">
```

```
    <h2 class="good__description">Сливочно-кофейное с кусочками шоколада</h2>
    <img class="good__mage" src="gllacy/choco.jpg" alt="Сливочно-кофейное с кусочками
шоколада">
    <p class="good__price">110□/кг</p>
  </li>
  …
</ul>
```

Обрати внимание, что текст в атрибуте alt у изображения должен быть таким же, как и название товара.

Создай функцию **renderCards**, которая будет принимать на вход массив данных, вызови её, передав cardsData, и отрисуй на странице карточки мороженого.
