



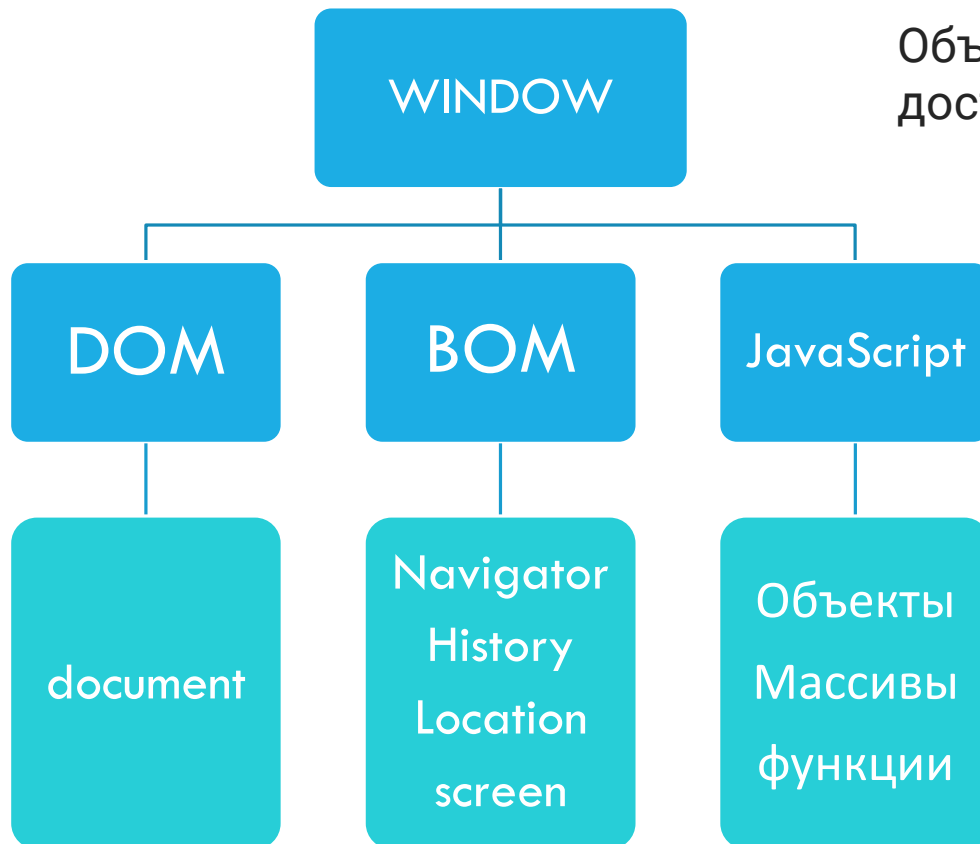
# ОСНОВЫ WEB-ТЕХНОЛОГИЙ

Лекция 6

# СТРУКТУРА ЗАНЯТИЯ

- Браузерное окружение
- Навигация по DOM
- Свойства DOM-узлов
- Поиск по дереву документ
- Внесение изменений в DOM-дерево
- Управление стилями элементов
- Браузерные события

# БРАУЗЕРНОЕ ОКРУЖЕНИЕ



Объекты браузерного окружения,  
доступные языку JavaScript

# БРАУЗЕРНОЕ ОКРУЖЕНИЕ

Помимо возможностей самого языка, мы получаем доступ к ряду объектов:

**Объект window** (корневой элемент, представляющий собой окно браузера и включающий методы управления им);

**Объект DOM** (Document object model, объектная модель документа) – совокупность всех сущностей веб-страницы в виде дерева;

**Объект BOM** (Browser object model, объектная модель браузера) – дополнительные инструменты для непосредственной работы с просмотрщиком.

# DOM SWEET DOM

DOM = Document Object Model

В соответствии с объектной моделью, каждый HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом.

# ОБЪЕКТ WINDOW

Позволяет обращаться к переменным и функциям в любом месте программы. Ключевое слово **window** можно опускать.

```
> window.prompt('Готовы к JS?')
```

```
< 'да'
```

---

```
> prompt('Готовы к JS?')
```

```
< 'не уверен'
```

---

Перечень всех свойств и методов объекта **window** огромен (увидеть можно, если ввести в консоли **window**)

# МОДЕЛЬ DOM

Все теги, комментарии и текст web-ресурса становятся объектами, с которыми можно работать.

```
> document.body.style.fontSize = "30px"  
< '30px'
```

---

# ОБЪЕКТНАЯ МОДЕЛЬ БРАУЗЕРА - ВОМ

Для доступа к инструментам самого браузера используется **ВОМ**. Из этого окружения можно получить данные о просмотрщике, операционной системе, текущем адресе ресурса, истории посещения страниц и пр.

```
> navigator.platform
```

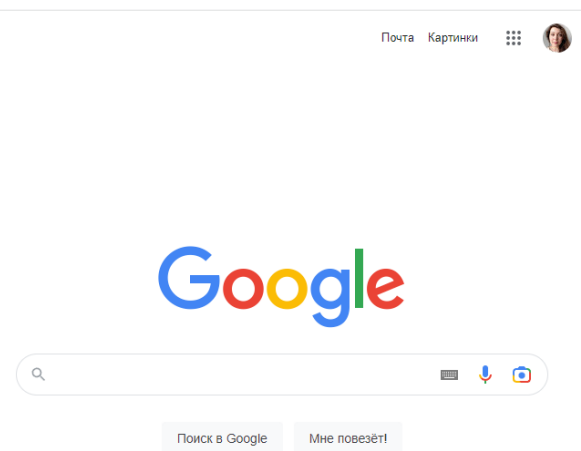
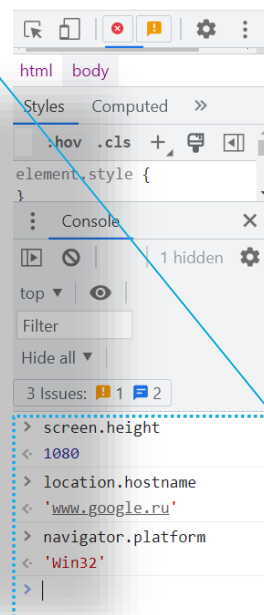
```
< 'Win32'
```

```
> location.hostname
```

```
< 'www.google.ru'
```

```
> screen.height
```

```
< 1080
```



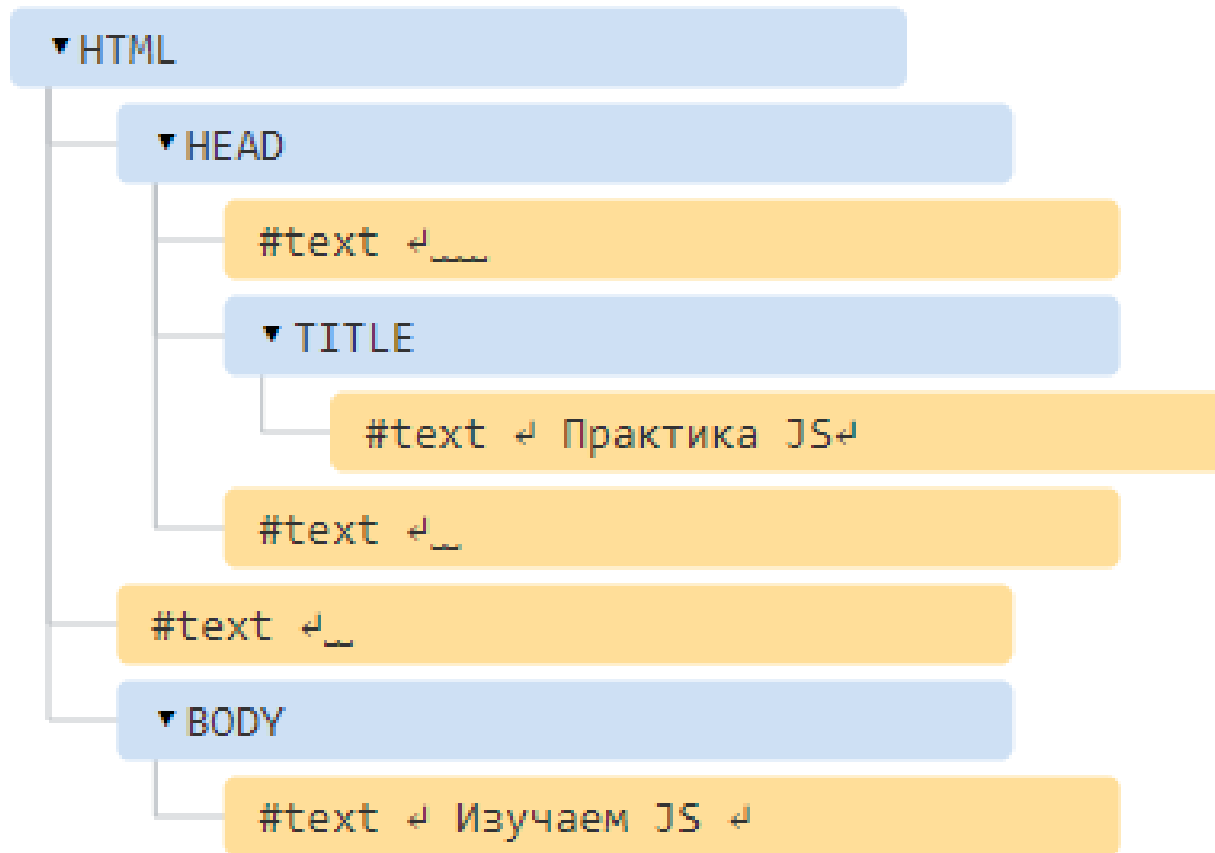


# ПОДРОБНЕЕ О DOM

Допустим, у нас есть страница следующего вида:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Практика JS</title>
5  </head>
6  <body>
7    Изучаем JS
8  </body>
9  </html>
```

# DOM ДЛЯ ЭТОГО ДОКУМЕНТА



ВАЖНО!

Все, что есть в HTML,  
даже комментарии,  
является частью DOM.

# НАВИГАЦИЯ ПО DOM-ДЕРЕВУ

## 1. Корневые элементы:

- **document.documentElement** (самый верхний узел, соответствует тегу **<html>**, включает в себя всё содержимое документа);
- **document.head** (заголовочная часть web-страницы);
- **document.body** (тело документа, содержимое тега **<body>**).

# НАВИГАЦИЯ ПО DOM-ДЕРЕВУ

**2. Узел-родитель (parentNode** –  
непосредственный потомок  
конкретного объекта)

# НАВИГАЦИЯ ПО DOM-ДЕРЕВУ

## 3. Дети, потомки:

- **childNodes** (коллекция потомков определенного объекта);
- **firstChild** (первый ребенок);
- **lastChild** (последний дочерний элемент).

# НАВИГАЦИЯ ПО DOM-ДЕРЕВУ

## 4. Соседи (один уровень иерархии):

- **nextSibling** (следующий сосед того же родителя);
- **previousSibling** (предыдущий элемент внутри родителя, находящийся на том же уровне, что и изначальный).

DOM

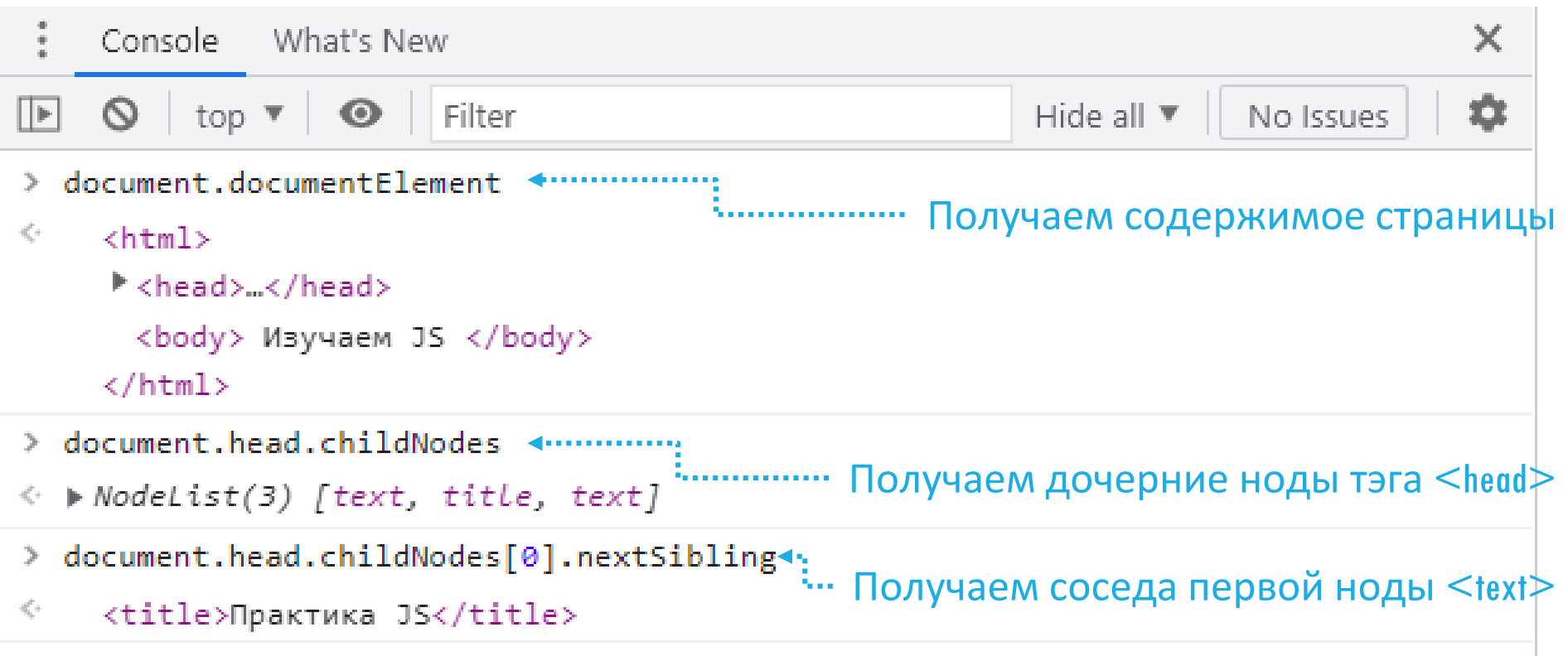
**Interviewer : How does DOM work?**

**ME:**

**FAMILY**



# ИЗУЧАЕМ DOM



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays three JavaScript commands and their corresponding DOM tree outputs. Blue dotted arrows and text annotations explain each step:

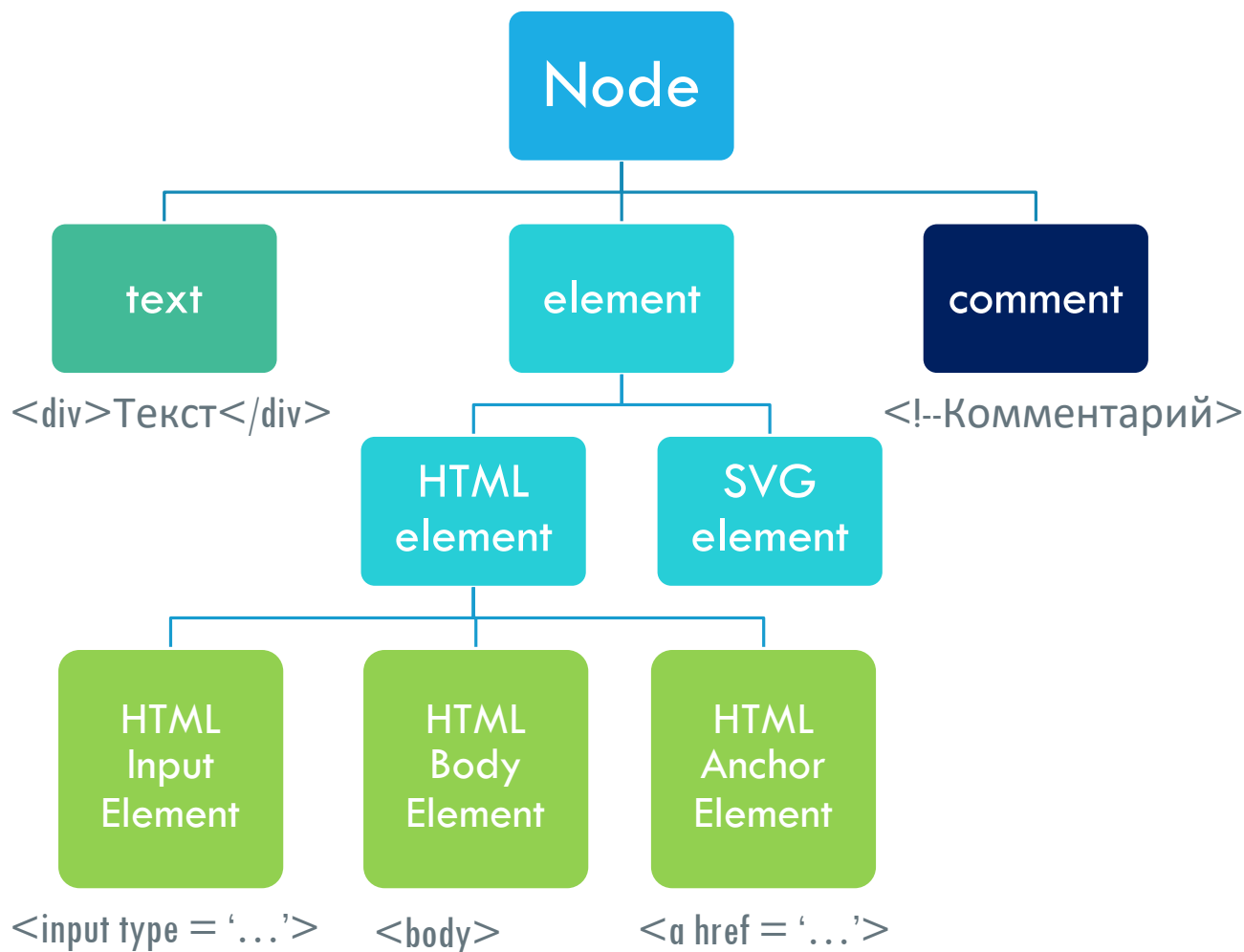
- Step 1:** The command `document.documentElement` is entered. The output is the full HTML document structure: `<html>`, `<head>...</head>`, and `<body> Изучаем JS </body>`. A blue dotted arrow points from the text 'Получаем содержимое страницы' to the `document.documentElement` command.
- Step 2:** The command `document.head.childNodes` is entered. The output is a `NodeList(3)` containing `[text, title, text]`. A blue dotted arrow points from the text 'Получаем дочерние ноды тэга <head>' to the `document.head.childNodes` command.
- Step 3:** The command `document.head.childNodes[0].nextSibling` is entered. The output is the `<title>Практика JS</title>` element. A blue dotted arrow points from the text 'Получаем соседа первой ноды <text>' to the `document.head.childNodes[0].nextSibling` command.

# КЛАССЫ DOM-УЗЛОВ

**DOM-узлы** принадлежат к определенным **классам**, имеющим иерархию:

- Класс **EventTarget** — класс-основа, позволяющий объектам поддерживать события;
- Класс **Node** — ключевой класс для узлов, позволяющий им быть обработанными при помощи свойств и методов: **childNodes**, **nextSibling** и др.;
- Класс **Element** — отвечает за навигацию на уровне элементов и снабжает их методами поиска. Является базой для **SVGElement**, **XMLElement** и **HTMLElement**;
- **HTMLElement** — позволяет наследоваться остальным тегам (**HTMLInputElement**, **HTMLAnchorElement**, **HTMLBodyElement**) и наделяет их свойствами (**click()**, **focus()** и др.).

# ИЕРАРХИЯ КЛАССОВ DOM-УЗЛОВ



# ИЕРАРХИЯ КЛАССОВ DOM-УЗЛОВ

Чтобы понять принадлежность объекта к классу имеется специальная инструкция: **instanceof**. Узнать принадлежность можно и через **console.dir()**

На основании узла **DOM** возможно выяснить имя тега элемента: **nodeName** и **tagName**. Второе свойство имеется только у объектов класса **Element**, а первое доступно всем.

# СВОЙСТВА УЗЛОВ

Свойство	Описание
innerHTML	Доступ к внутреннему контенту элемента (представлен в виде строки)
outerHTML	Показывает содержимое элемента вместе с ним самим
nodeValue	Содержимое текстового узла или комментария
data	Аналогично nodeValue за некоторыми нюансами (направлено исключительно на текст)
textContent	Внутренний текст объекта за вычетом любых тегов
hidden	Отвечает за видимость тега на странице.

# ПРИМЕР

```
> document.body.innerHTML  
< '\n  <h1 hidden="">Изучаем JS</h1>\n\n\n'  
-----  
> document.body.outerHTML  
< '<body>\n  <h1 hidden="">Изучаем JS</h1>\n\n\n</body>'  
-----  
> document.body.firstChild.hidden = true  
< true  
-----  
> document.body.firstChild.data = 'Новый текст'  
< 'Новый текст'
```

Содержимое тела документа

Содержимое тега <body> вместе с ним самим

Меняем видимость тега <h1>

Меняем содержимое тега <body>

# ДРУГИЕ СВОЙСТВА

Свойство	Описание
value	Получение значения у элементов input, textarea, select
href	Ссылка
Id	Получение идентификатора, в случае его наличия
className	Название всех классов объекта

# ВАЖНО

Свойства **DOM-объектов** не следует путать с атрибутами HTML-элементов. Важно запомнить, что свойства способны принимать любые значения, а атрибутами могут быть только строки.



# СВОИ СВОЙСТВА

> `document.body.myProperty = 'Созданное свойство'`

< `'Созданное свойство'`

---

> `document.body.myProperty`

< `'Созданное свойство'`

---

# ВЗАИМОДЕЙСТВИЕ С ПОЛЬЗОВАТЕЛЬСКИМИ АТТРИБУТАМИ

Метод	Описание
hasAttribute()	Проверка наличия атрибута у элемента
getAttribute()	Получение значения атрибута
setAttribute()	Создание атрибута с некоторым значением
removeAttribute()	Удаление атрибута.

# ПРИМЕР

```
> document.body.hasAttribute('layout')
```

```
< false
```

---

```
> document.body.setAttribute('layout', 'flexible')
```

```
< undefined
```

---

```
> document.body.getAttribute('layout')
```

```
< 'flexible'
```

---

```
> document.body.removeAttribute('layout')
```

```
< undefined
```

---

```
> document.body.hasAttribute('layout')
```

```
< false
```

---

# ПОИСК ПО ДЕРЕВУ ДОКУМЕНТА

Функция	Описание
<code>getElementById()</code>	Ищет объект по идентификатору
<code>getElementsByName()</code>	Позволяет выделить элементы с заданным атрибутом «name»
<code>getElementsByTagName()</code>	Обнаружение элементов по названию тега
<code>getElementsByClassName()</code>	Поиск на основании класса
<code>querySelector()</code>	Находит первый элемент на основании CSS-селектора
<code>querySelectorAll()</code>	Ищет все объекты по заданному селектору

# ПОИСК ПО ДЕРЕВУ ДОКУМЕНТА

> document.getElementById('first\_header')

< <h1 id="first\_header">Поиск в дереве документа</h1>

> document.getElementsByName('last')

< ► NodeList [div.main\_div]

Выберем последнюю статью на основании свойства «name»

> document.getElementsByTagName('div')

< ► HTMLCollection(3) [div.main\_div, div.main\_div, div.main\_div, Last: div.main\_div]

Найдем div на странице по тегу

Выведем список статей с классом «main\_div»

> document.getElementsByClassName('main\_div')

< ► HTMLCollection(3) [div.main\_div, div.main\_div, div.main\_div, Last: div.main\_div]

> document.querySelector('.main\_div')

< <div class="main\_div">Первый блок</div>

Получим доступ к первой статье

# ВНЕСЕНИЕ ИЗМЕНЕНИЙ В DOM-ДЕРЕВО

При помощи JavaScript мы можем не только находить элементы в web-документе или менять их свойства, но и **вносить изменения** в структуру DOM-дерева:

- Создавать новые теги
- Удалять имеющиеся
- Заменять их на другие

# ВНЕСЕНИЕ ИЗМЕНЕНИЙ В DOM-ДЕРЕВО

Функция	Описание
<code>createElement()</code>	Создание определенного тега
<code>createTextNode()</code>	Объявление текстового узла с некоторым содержимым
<code>remove()</code>	Удаление элемента
<code>cloneNode()</code>	Полная копия желаемого объекта
<code>append()</code>	Добавление узла в конец

# ВНЕСЕНИЕ ИЗМЕНЕНИЙ В DOM-ДЕРЕВО

Функция	Описание
<code>prepend()</code>	Вставка объекта в начало узла
<code>before()</code>	Добавление элемента перед узлом
<code>after()</code>	Вставка объекта после узла
<code>replaceWith()</code>	Замена содержимого



# УПРАВЛЕНИЕ СТИЛЯМИ ЭЛЕМЕНТОВ

**Взаимодействие с классами** для изменения HTML-элементов может осуществляться как при помощи свойства **style**, так и непосредственно обращением к классам напрямую.

# ИЗМЕНЕНИЕ КЛАССОВ

Объект **classList** позволяет менять, удалять или добавлять классы к элементам. Здесь имеется 4 метода:

Метод **add()** – добавить новый класс;

Метод **remove()** – удалить класс у тега;

Метод **toggle()** – добавить класс при отсутствии, а иначе - удалить;

Метод **contains()** – проверяет наличие класса.

# ИЗМЕНЕНИЕ АТТРИБУТОВ ТЕГОВ

Изменять и определять конкретные атрибуты тегов достаточно просто: нужно лишь к ним обратиться при помощи соответствующих методов. Важно запомнить, что в JavaScript не используются дефисы, поэтому составные свойства пишутся в **верблюжьей нотации**

font-size (CSS) .....→ fontSize (JS)

# СБРОС СТИЛЕЙ

Для сброса стилей  
применяется  
свойство **cssText**.

# ПОЛУЧЕНИЕ СТИЛЕЙ ЭЛЕМЕНТОВ

Метод **getComputedStyle()** позволяет решить эту проблему.

# БРАУЗЕРНЫЕ СОБЫТИЯ

Событие	Описание
click	Самое часто используемое, срабатывает при нажатии левой кнопки мыши
contextmenu	Реагирует на правую кнопку мыши
mousemove	Связано с движением мышки
submit	Срабатывает при отправке формы
keydown, keyup	Реагирует на нажатие и отпускание кнопки на клавиатуре
transitionend	Реагирует на завершение анимации
offline	Срабатывает при отсутствии доступа к сети

# ОБРАБОТКА СОБЫТИЯ ЧЕРЕЗ СВОЙСТВО ТЭГА

Например,

```
<body onclick="console.log('Нажали!')"></body>
```

# ОБРАБОТКА СОБЫТИЯ ЧЕРЕЗ ВЫБОР ЭЛЕМЕНТА В JS-ФАЙЛЕ

```
function clicked() {  
  console.log('Нажали!');  
}
```

```
document.body.onclick = clicked;
```



# ОБРАБОТКА СОБЫТИЯ ЧЕРЕЗ СПЕЦИАЛЬНЫЕ МЕТОДЫ

Сюда относятся следующие  
методы:

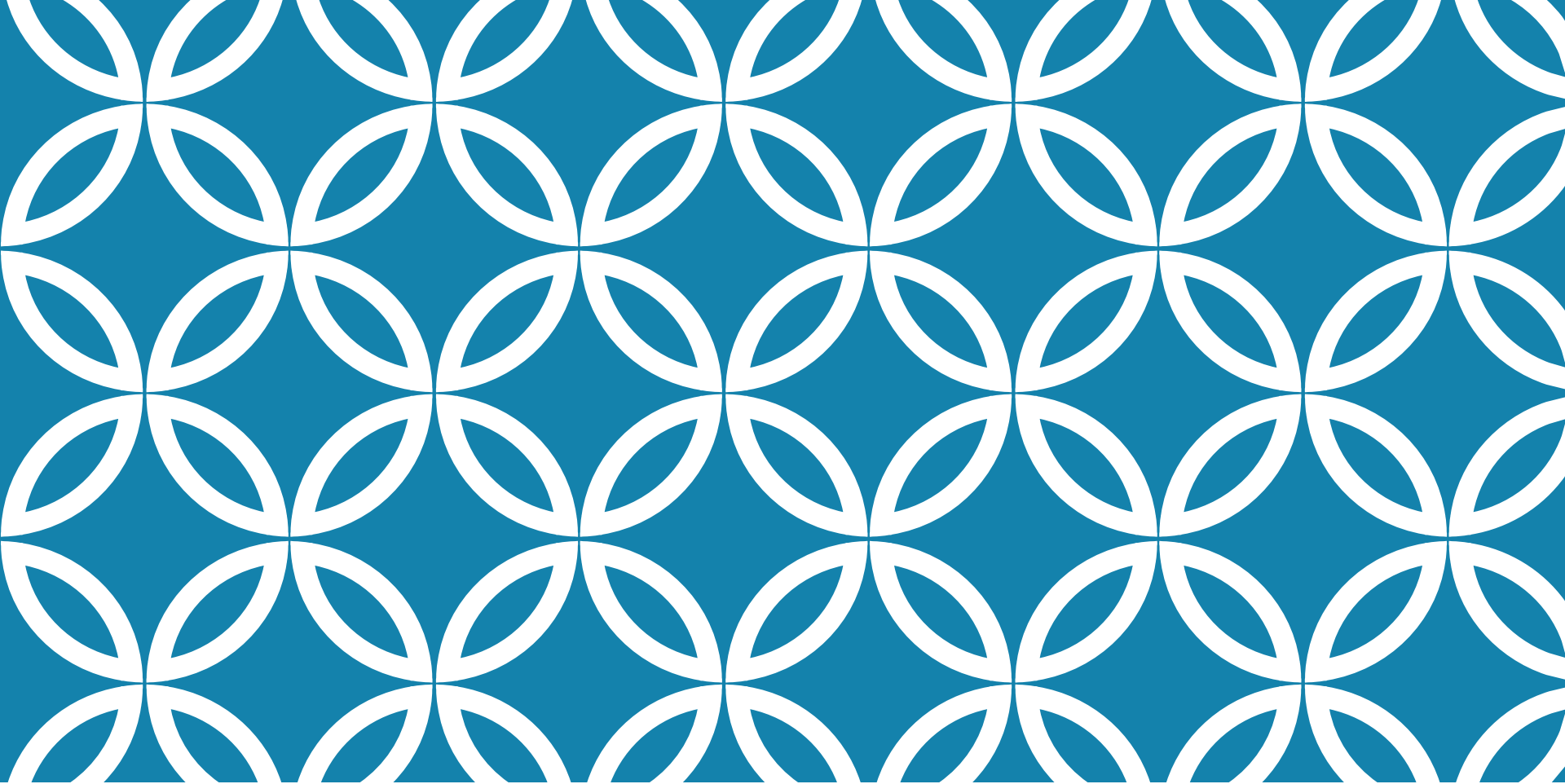
- **addEventListener()**
- **removeEventListener()**

# ПАРАМЕТР EVENT ФУНКЦИИ-ОБРАБОТЧИКА

Зачастую требуется не просто обработать некое событие, но и понять детали того, что именно произошло. В этом случае в функции-обработчики передается параметр **event**, имеющий массу полезных свойств.

# ДЕЛЕГИРОВАНИЕ СОБЫТИЙ

Оно нужно для того, чтобы уменьшить количество кода. Предположим, у родительского элемента в наличии большое количество потомков. Каждому из них мы бы хотели присвоить одно и то же событие. Удобнее всего задать его предку, а в зависимости от положения мыши, например, передавать его только дочернему объекту.



**СПАСИБО ЗА  
ВНИМАНИЕ!**

конец лекции 5