JS: DOM

AIT
Hi-tech team

НАШИ ПРАВИЛА

Включенная камера

Вопросы по поднятой руке

Не перебиваем друг друга

Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору

Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

Повторим

Как вызвать функцию?

Что такое параметры функции? Для чего они нужны?

Каким образом мы можем вернуть значение из функции?

Что такое поднятие(hoisting)?

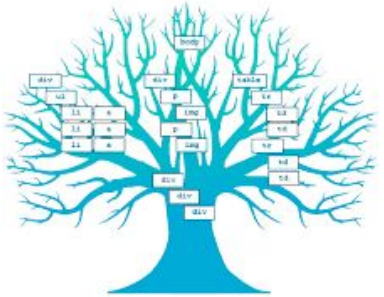
ЦЕЛЬ

Разобраться с понятием DOM. Узнать самые простые методы работы с DOM.

ПЛАН ЗАНЯТИЯ

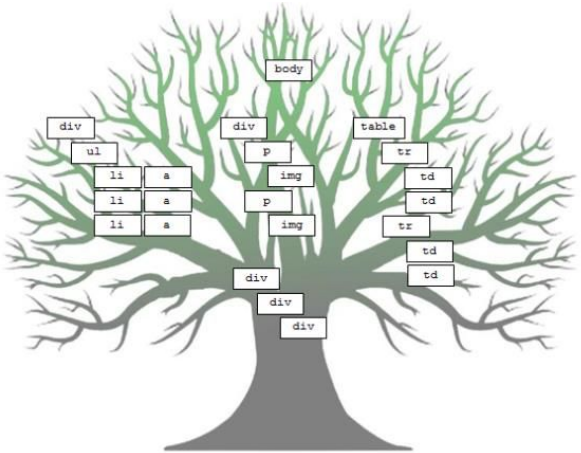
- DOM как объектная модель
- DOM как программный интерфейс
- Типы узлов
- Методы работы с DOM
- Получение данных из формы

DOM (Document Object Model)

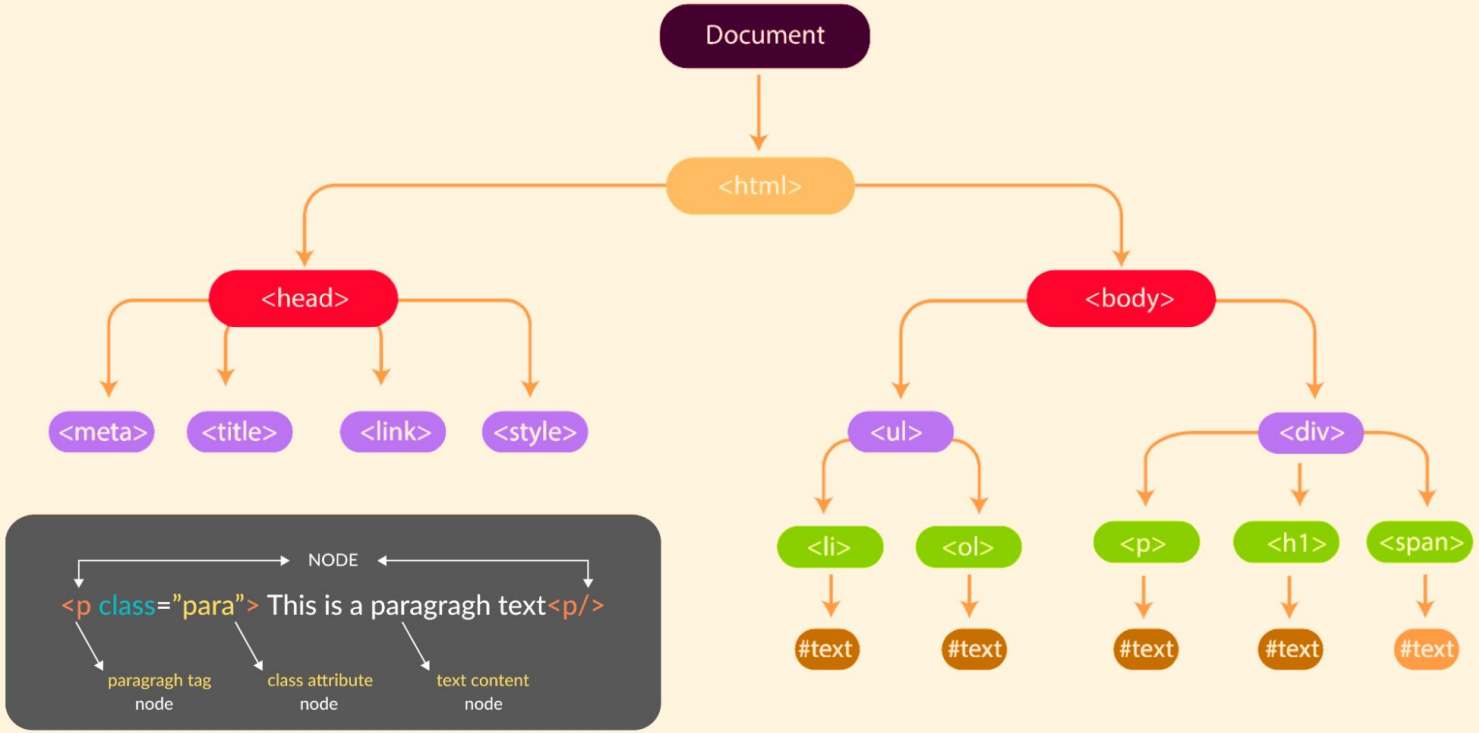


DOM (Document Object Model) – это объектная модель документа, которую браузер создает в памяти компьютера на основании HTML-кода.

Иными словами, это представление HTML-документа в виде дерева тегов. Такое дерево нужно для правильного отображения сайта и внесения изменений на страницах с помощью JavaScript



The DOM Structure/ DOM TREE



Все что имеется в разметке отображается в DOM дереве и выступает узлом этого дерева

Document node

Это вся страница браузера. Все остальное вложено в этот узел – как дети.

Element nodes

Все элементы, такие как заголовки (<h1> to <h6>) или параграфы (<p>) представлены отдельными узлами в дереве. Мы можем получить даже доступ к их атрибутам и текстовому содержанию

Все что имеется в разметке отображается в DOM дереве и выступает узлом этого дерева

Attribute nodes

Если тег элемента содержит атрибуты – эти атрибуты будут представлены в качестве отдельных узлов атрибутов. По сути, это уже не дочерние элементы по отношению к данному элементу, а свойства данного элемента.

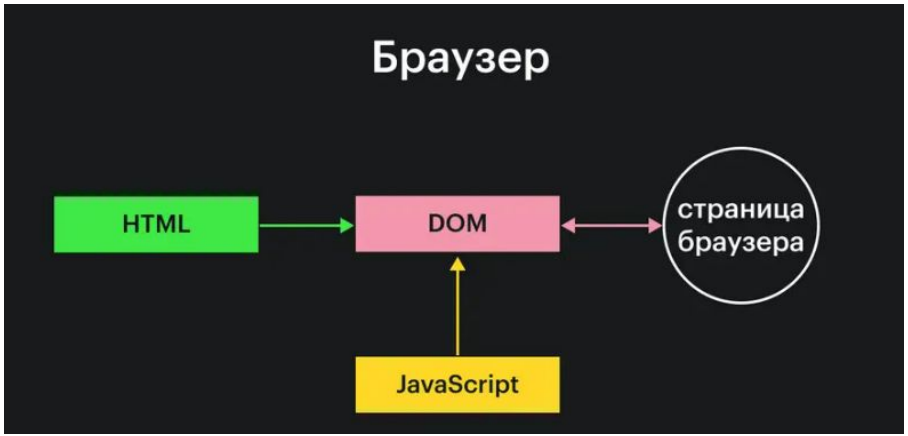
Text nodes

Текст тоже создает отдельный текстовый узел

DOM позволяет управлять HTML-разметкой из JavaScript-кода.

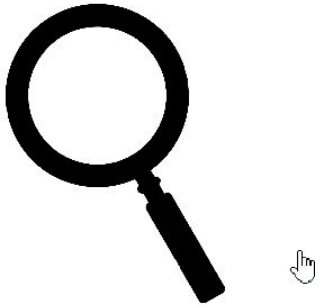
Управление обычно состоит из:

- добавления элементов
- удаления элементов
- изменения стилей и содержимого элементов



Прежде чем управлять элементом его нужно выбрать!

Методы поиска элементов



Атрибуты defer, async

В современных сайтах скрипты обычно «тяжелее», чем HTML: они весят больше, дольше обрабатываются.

Когда браузер загружает HTML и доходит до тега <script>...</script>, он не может продолжать строить DOM. Он должен сначала выполнить скрипт.

Это ведёт к двум важным проблемам:

1. **Скрипты не видят DOM-элементы ниже себя, поэтому к ним нельзя добавить обработчики и т.д.**
2. **Если вверху страницы объёмный скрипт, он «блокирует» страницу. Пользователи не видят содержимое страницы, пока он не загрузится и не запустится:**

Атрибуты defer, async

1. Атрибут defer сообщает браузеру, что он должен продолжать обрабатывать страницу и загружать скрипт в фоновом режиме, а затем запустить этот скрипт, когда DOM дерево будет полностью построено.
2. Атрибут async означает, что скрипт абсолютно независим - страница не ждёт асинхронных скриптов, содержимое обрабатывается и отображается, а остальные скрипты не ждут async, и скрипты с async не ждут другие скрипты

```
<script defer async src="https://javascript.js"></script>
```

Метод 1
Поиск элемента по ID

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

Примечание: возвращает элемент с заданным id. элемент должен иметь атрибут id

Метод 1
Поиск элемента по ID

Задача: получить элемент с id=”elem”

```
let elem = document.getElementById('elem');
```

Метод 2
Поиск элемента по тегу

```
document.getElementsByTagName(tag)
```

Примечание: метод ищет элементы с данным тегом и возвращает их коллекцию.

Метод 2
Поиск элемента по тегу

Задача: получить все элементы div в документе

```
let divs = document.getElementsByTagName('div');
```

Метод 3
Поиск элементов по названию класса

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

Примечание: метод возвращает элементы, которые имеют данный класс

Метод 3
Поиск элементов по названию класса

Задача: получить все элементы с классом article

```
let articles = document.getElementsByClassName('article');
```

Метод 4

Поиск элементов по значению атрибута name

```
document.getElementsByName(name)
```

Примечание: возвращает элементы с заданным атрибутом name

Метод 4

Поиск элементов по значению атрибута name

Задача: получить все элементы со значением атрибута name=”up”

```
let articles = document.getElementsByName(“up”);
```

Метод 5

Поиск по CSS-селектору

(самый универсальный метод поиска)

```
document.querySelectorAll(css selector)
document.querySelector(css selector)
```

Метод 5

Поиск по CSS-селектору

Задача: получает все элементы <li>, которые являются дочерними для

```
let elements = document.querySelectorAll('ul > li');
```

Добавление элементов

1 шаг - Создать элемент

Метод: `document.createElement("tag")`

Создание: `let el = document.createElement("div")`

2 шаг - Заполнить элемент

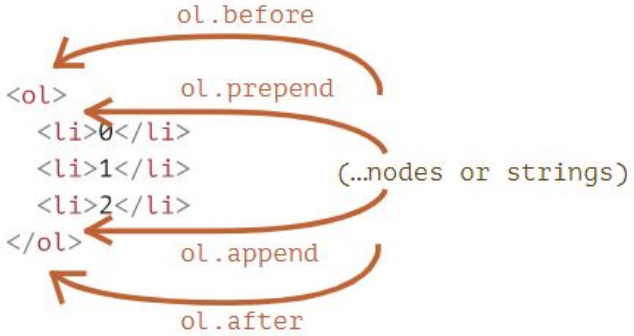
Метод: `node.textContent = “text”`

Заполнение: el.textContent = “Text”

3 шаг - Добавляем элемент в document

Методы:

- **node.append(el) – добавляет узлы или строки в конец node,**
- **node.prepend(el) – вставляет узлы или строки в начало node,**
- **node.before(el) – вставляет узлы или строки до node,**
- **node.after(el) – вставляет узлы или строки после node,**



Изменение элементов

- **textContent**

Позволяет задавать или получать текстовое содержимое элемента и его потомков.

```
let text = element.textContent

element.textContent = “Just text”
```

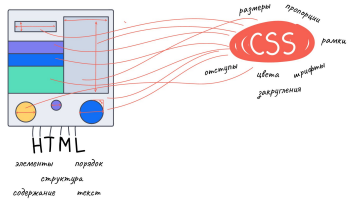
- **innerHTML**

Свойство innerHTML позволяет считать содержимое элемента в виде HTML-строки или установить новый HTML.

```html
<form>
  <label>Логин</label>
  <input type="text" id="login" />
  <div class="error">Введите логин</div>
</form>
```

```
1 const form = document.querySelector('form')
2
3 console.log(form.innerHTML)
4 // '<label>Логин</label><input type="text" id="login" /><div class="error">Вве
5
6 // Меняем содержимое новым html
7 form.innerHTML = '<div class="success">Вход выполнен</div>'
```

- **изменение стилей**



HTML DOM позволяет JavaScript изменять стиль HTML элементов.

Синтаксис:

```
document.getElementById(id).style.свойство = новый стиль
```

Применение:

```
document.getElementById("p2").style.color = "blue";
```

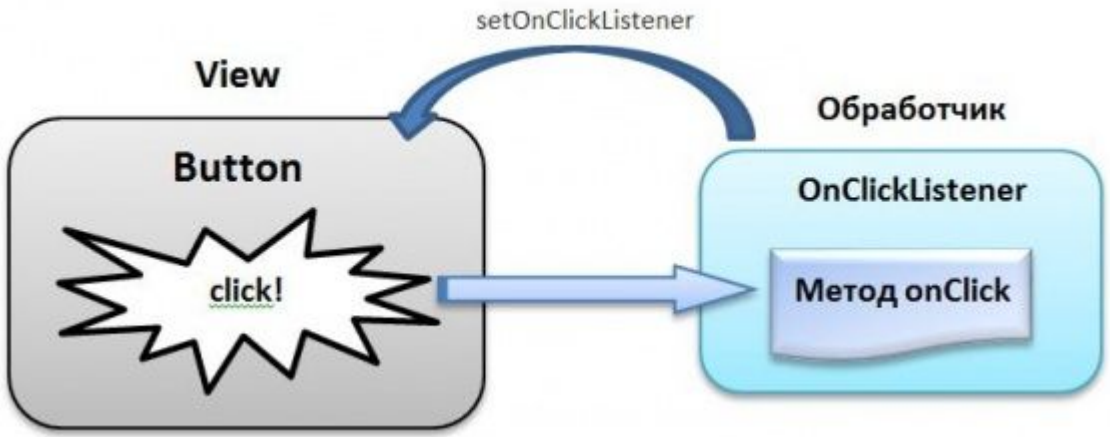
- **удаление элементов**

Для удаления узла есть метод node.remove()

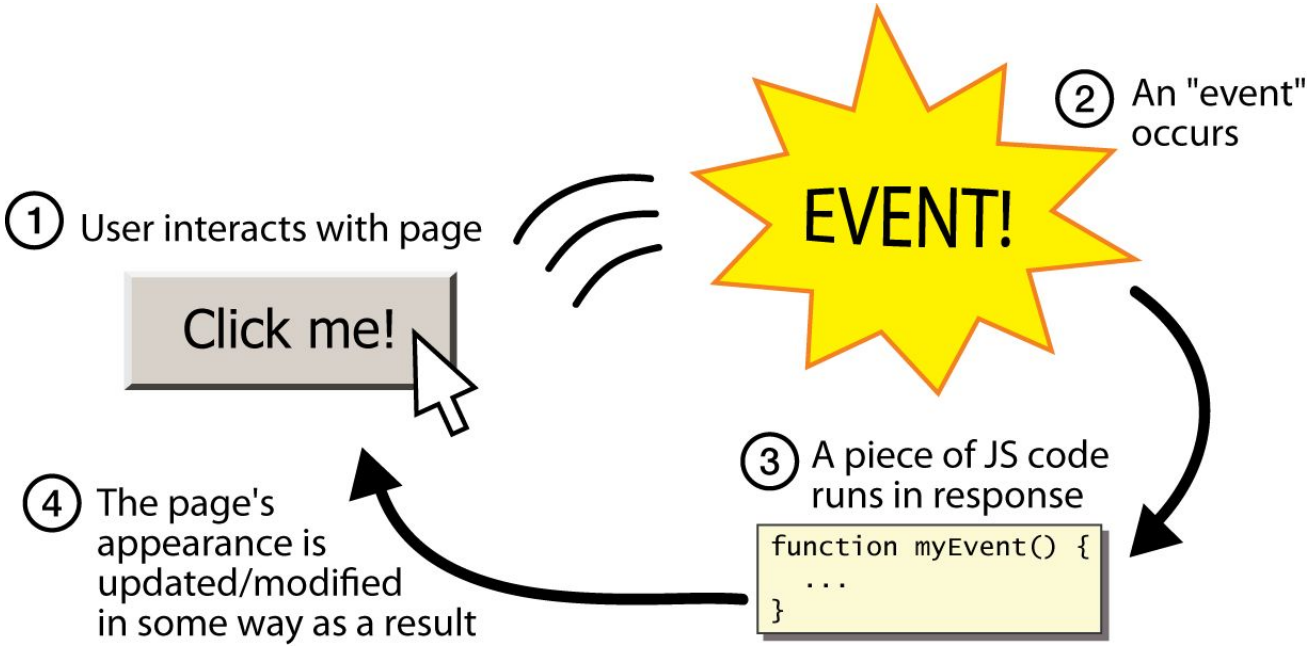
Пример использования: удаление элемента с id=”register”

```
document.getElementById("register").remove();
```

Events, event listeners

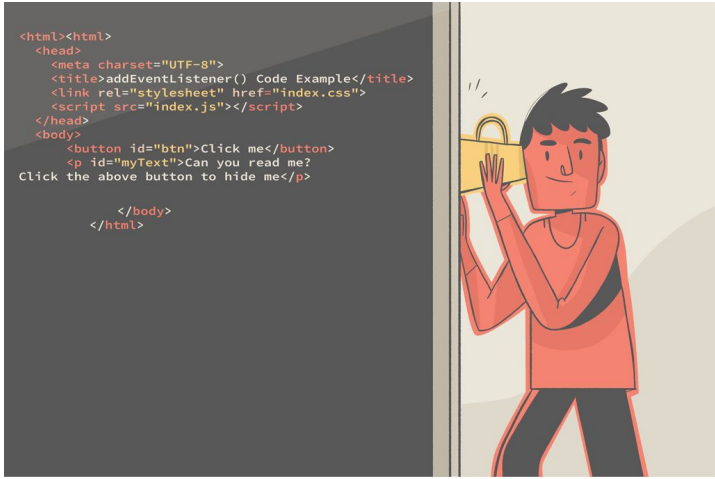


Любой DOM элемент запускает событие, когда мы с ним как-то взаимодействуем (кликаем, наводим мышь и др.). Обработчики событий в JS используются для того, чтобы реагировать на эти события.



Чтобы "повесить" обработчик событий на элемент, нужно использовать специальный метод - **addEventListener**. Этот метод принимает 2 аргумента:

1. **Тип события** (например "click").
2. Так называемую колбэк (**callback**) функцию, которая запускается после срабатывания нужного события.




```
element.addEventListener('click', handleClickFunction)
```

Пример

Найдём кнопку на странице и будем выводить сообщение в консоль, когда произошёл клик по этой кнопке.

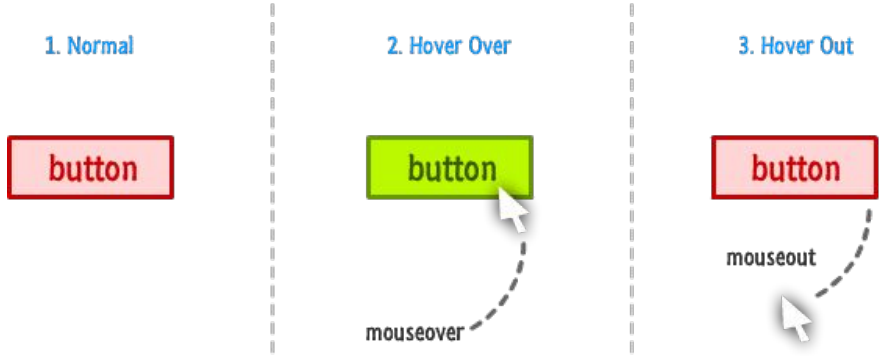
```
1 const element = document.querySelector('button')
2
3 element.addEventListener('click', function (event) {
4   console.log('Произошло событие', event.type)
5 })
```

Типы событий

eventType (первый аргумент addEventListener) - строка, содержащая название события.

Наиболее популярные события:

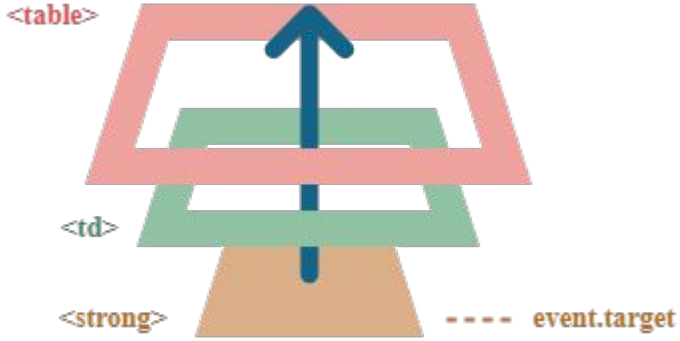
- 'click'
- 'change'
- 'submit'
- 'keydown'
- 'keyup'
- 'mousemove'
- 'mouseenter'
- 'mouseleave'



Объект Event

Когда происходит событие, браузер создаёт объект события - event, записывает в него детали и передаёт его в качестве аргумента функции-обработчику (второму аргументу addEventListener).

Объект Event - основные свойства



- **defaultPrevented** — отменено ли поведение события по умолчанию.
- **target** — ссылка на объект, которым было инициировано событие. Например, если событие произошло на поле ввода, мы получим ссылку на этот DOM элемент.
- **type** — тип события.

Объект Event - метод preventDefault

`event.preventDefault()` — предотвращает дефолтное поведение события.

Например, при нажатии на ссылку, отменить переход по адресу ссылки

Работа с формой

```
<form id="myForm">
    <label for="username">Username:</label>
    <input type="text" id="username"
name="username">

    <label for="password">Password:</label>
    <input type="password" id="password"
name="password">

    <button type="button"
onclick="getData()">Submit</button>
 </form>
```

```
function getData() {
 const form = document.getElementById('myForm');

 // Получение данных по имени
 // два варианта
 let username = form.elements["username"].value;
 let password = form.password.value;
  // Действия с данными
 console.log('Username:', username);
 console.log('Password:', password);
}
```

Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH

AIT
Hi-tech team