

# DOM

## Занятие 10



---

## Ход занятия

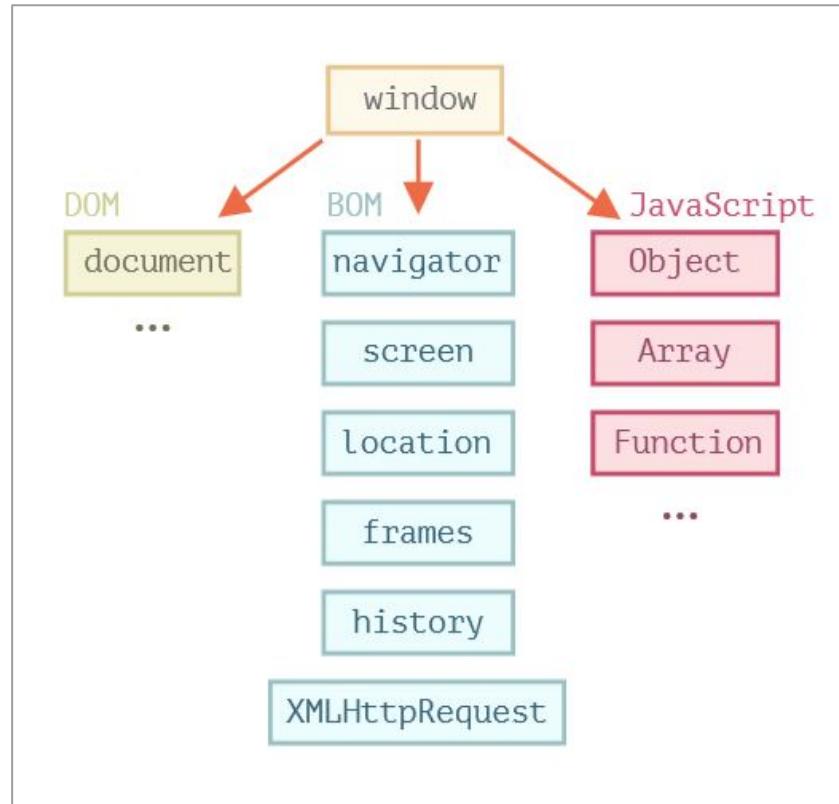
1. DOM.
2. События.

---

# **Document Object Model**

...

# Окружение



---

# Window

Корневой объект `window`, выступает в двух ролях:

- Во-первых, это глобальный объект для JavaScript-кода.
- Во-вторых, он также представляет собой окно браузера и располагает методами для управления им.

\* При обращении к ключевым объектам `Window` можно опустить дальше в примерах.

---

## Browser Object Model

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

---

## Navigator

Интерфейс Navigator представляет собой состояние и особенности(свойства) пользовательского агента. Это позволяет скриптам узнавать их и самостоятельно регистрироваться для выполнения некоторых действий. Объект Navigator работает только для чтения.

```
console.log(window.navigator.userAgent); // информация о  
браузере
```

```
console.log(window.navigator.platform); //информация о ОС
```

---

## ФУНКЦИИ

Функции `alert` / `confirm` / `prompt` / `setTimeout` тоже являются частью ВОМ: они не относятся непосредственно к странице, но представляют собой методы объекта окна браузера.

```
window.alert("Привет мир!");
```

---

## History

`window.history` — управляет историей просмотра веб-страниц. Для перемещения по журналу просмотра используются методы `back`, `forward`:

`window.history.back();` // вернет на предыдущую просмотренную страницу

`window.history.forward();` // передвинет на одну запись вперед в истории (если такая есть)

---

## Location

Объект `location` позволяет получить информацию о текущем URL и перенаправить браузер по новому адресу.

---

## Screen

Интерфейс screen представляет экран.

Обычно, это тот, на котором текущее окно визуализируется, может быть получен с использованием `window.screen`.

```
console.log(window.screen.width); //Возвращает ширину экрана в пикселях
```

```
console.log(window.screen.height); //Возвращает высоту экрана в пикселях.
```

---

# XMLHttpRequest

`XMLHttpRequest` – это встроенный в браузер объект, который даёт возможность делать HTTP-запросы к серверу без перезагрузки страницы.

На сегодняшний день не обязательно использовать `XMLHttpRequest`, так как существует другой, более современный метод `fetch`.

```
let xhr = new XMLHttpRequest();
```

```
xhr.open('GET', '/my/url');
```

```
xhr.send();
```

---

# Document Object Model

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.

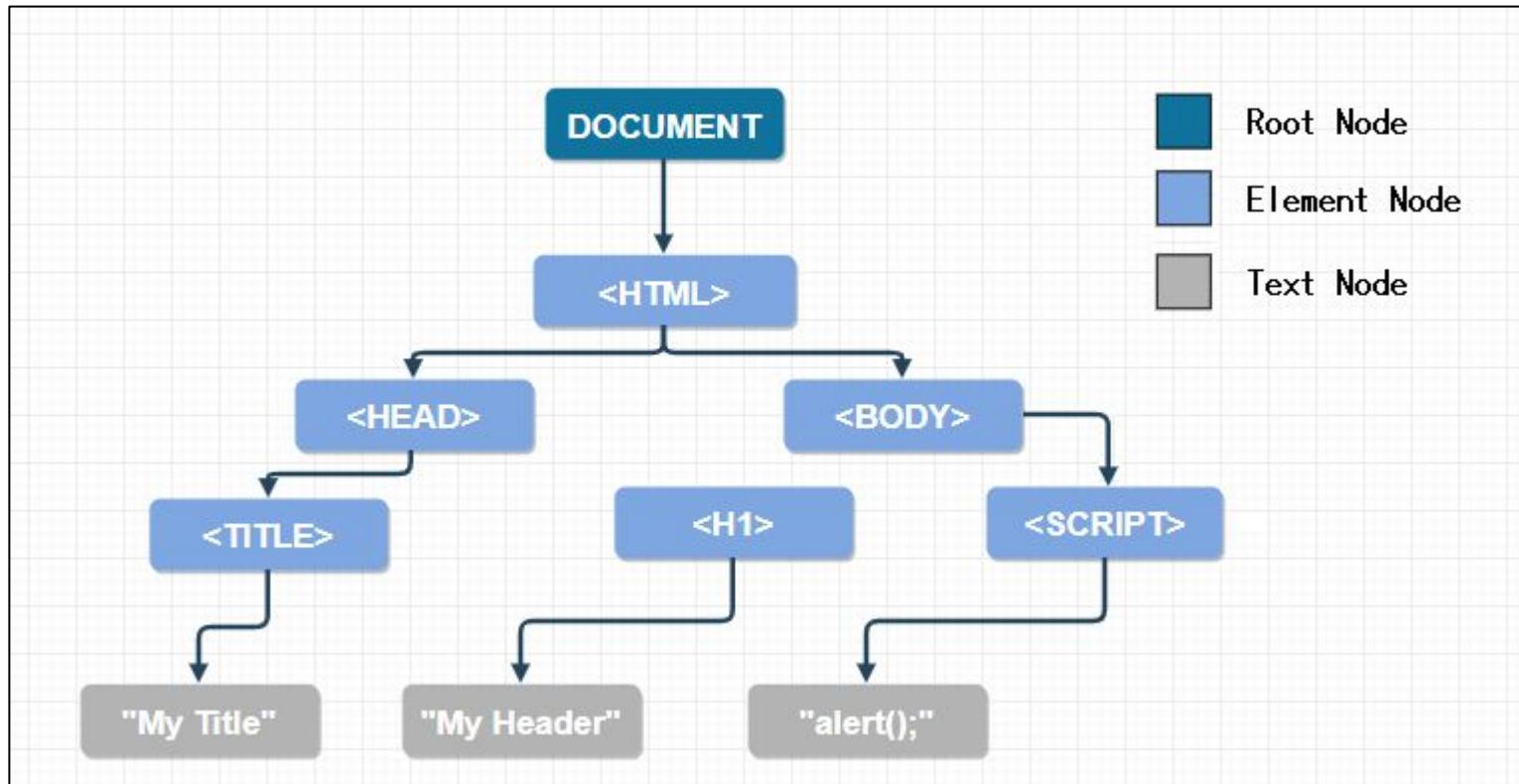
---

## DOM-дерево

В соответствии с объектной моделью документа («Document Object Model», коротко DOM), каждый HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом.

Например, `document.body` – объект для тега `<body>`.

# Схема

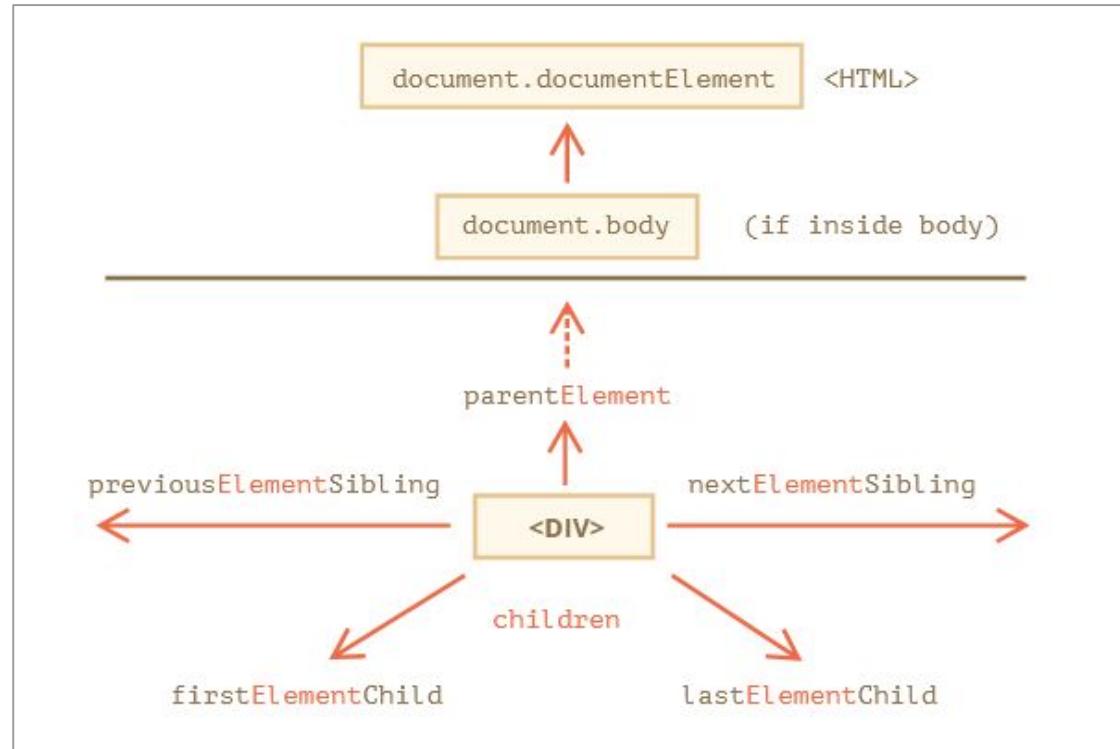


---

## Описание

- `document` – «входная точка» в DOM.
- узлы-элементы – HTML-теги, основные строительные блоки.
- текстовые узлы – содержат текст.
- комментарии – иногда в них можно включить информацию, которая не будет показана, но доступна в DOM для чтения JS.

# Навигация по DOM



---

## Важно

- В DOM значение null значит «не существует» или «нет такого узла».
- DOM-коллекции – все навигационные свойства, доступны только для чтения.
- Коллекции нужно перебирать циклом for..of. Цикл for..in перебирает все перечисляемые свойства. А у коллекций есть некоторые «лишние»
- Нельзя получить доступ к элементу, которого ещё не существует в момент выполнения скрипта. В частности, если скрипт находится в <head>, document.body в нём недоступен, потому что браузер его ещё не прочитал.

---

# Поиск элементов в DOM

---

## document.getElementById

```
<div id="elem">...</div>
```

```
<script>
```

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

```
        elem.style.background = 'red';
```

```
</script>
```

---

## getElementsBy\*

`document.getElementsByTagName(tag);` //ищет элементы с  
данным тегом и возвращает их коллекцию

`document.getElementsByClassName(className);` //ищет элементы  
с данным классом и возвращает их коллекцию

`document.getElementsByName(name);` //ищет элементы с данным  
атрибутом name и возвращает их коллекцию

\* возвращается коллекция [], нужно обращаться к индексам  
элементов

---

## document.querySelector(css)

Результат такой же, как при вызове `elem.querySelectorAll(css)[0]`, но он сначала найдет все элементы, а потом возьмёт первый, в то время как `elem.querySelector` найдёт только первый и остановится.

```
<ul>
```

```
  <li>Пункт 1</li>
```

```
  <li>Пункт 2</li>
```

```
</ul>
```

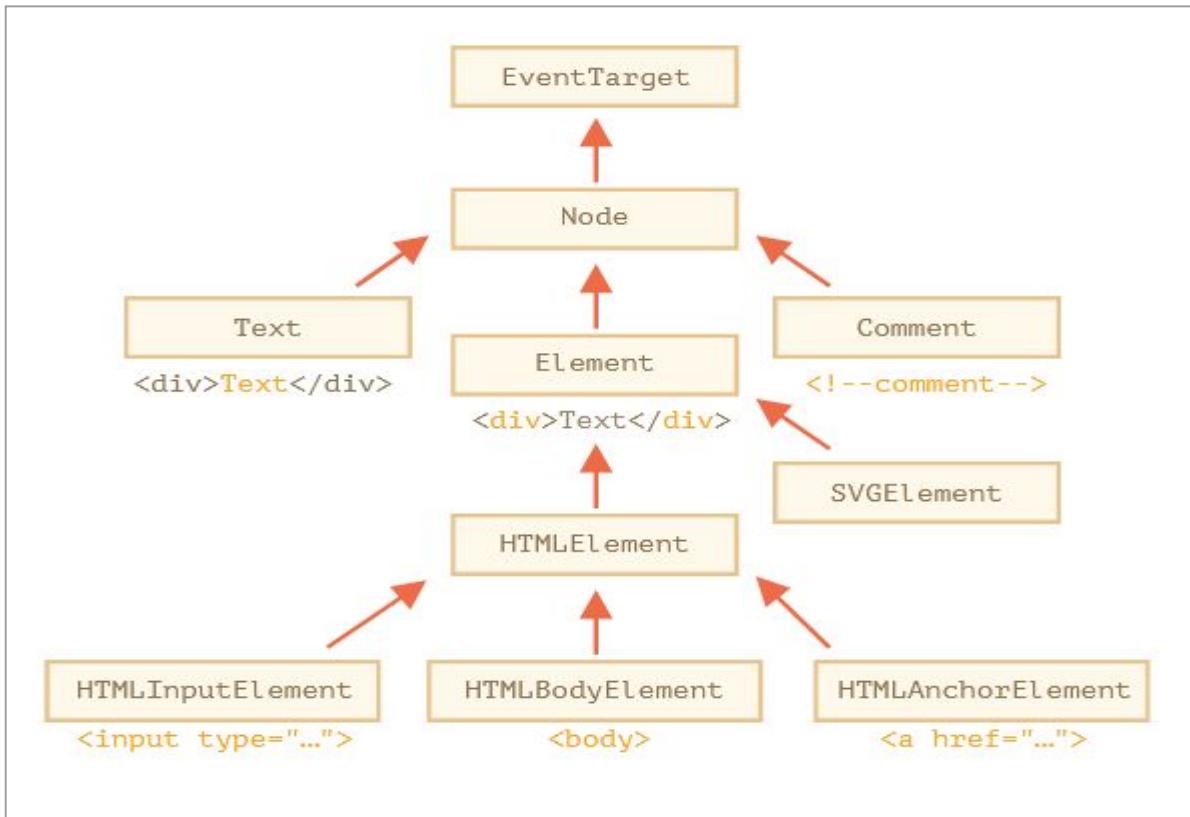
---

## Классы DOM-узлов

У разных DOM-узлов могут быть разные свойства. Например, у узла, соответствующего тегу `<a>`, есть свойства, связанные со ссылками, а у соответствующего тегу `<input>` – свойства, связанные с полем ввода и т.д. Текстовые узлы отличаются от узлов-элементов. Но у них есть общие свойства и методы, потому что все классы DOM-узлов образуют единую иерархию.

DOM-узлы – это обычные JavaScript объекты. Для наследования они используют классы, основанные на прототипах.

# Классы DOM-узлов



---

## Изменение содержимого innerHTML

Свойство `innerHTML` позволяет получить HTML - содержимое элемента в виде строки.

```
document.body.innerHTML = 'Новый BODY!'; // заменяем  
содержимое
```

```
elem.innerHTML += "Новый HTML";
```

---

## Изменение содержимого outerHTML

Свойство `outerHTML` содержит HTML элемента целиком. Это как `innerHTML` плюс сам элемент.

\*Будьте осторожны: в отличие от `innerHTML`, запись в `outerHTML` не изменяет элемент. Вместо этого элемент заменяется целиком во внешнем контексте.

---

## Изменение содержимого `textContent`

Свойство `textContent` предоставляет доступ к тексту внутри элемента за вычетом всех <тегов>.

Запись в него помещает текст в элемент, при этом все специальные символы и теги интерпретируются как текст. Можно использовать для защиты от вставки произвольного HTML кода.

---

## Изменение стилей

- Создать класс в CSS и использовать его: <div class="...">
- Писать стили непосредственно в атрибуте style: <div style="...">.

JavaScript может менять и классы, и свойство style.

Классы – всегда предпочтительный вариант по сравнению со style. Мы должны манипулировать свойством style только в том случае, если классы «не могут справиться».

```
elem.style.width="100px"; //работает так же, как наличие в атрибуте style  
строки width:100px
```

---

## Работа с классами

- `document.body.className`; // получить классы
- `elem.className` - при присвоении заменяет всю строку с классами.
- `elem.classList` – это специальный объект с методами для добавления/удаления одного класса.
  - `elem.classList.add/remove("class")` – добавить/удалить класс.
  - `elem.classList.toggle("class")` – добавить класс, если его нет, иначе удалить.
  - `elem.classList.contains("class")` – проверка наличия класса, возвращает true/false.

---

## Создать элемент

```
let test = document.createElement('div');
test.className = "test";
test.innerHTML = "text"; //div переменная dom object
```

Получаем

```
<div class="test">text</div>
```

---

# Методы вставки в DOM дерево

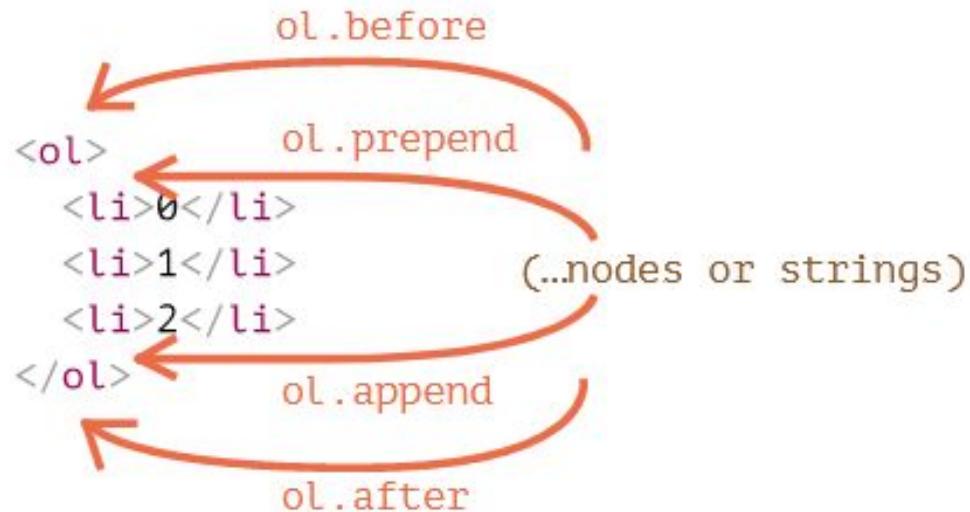
- `node.append(...nodes or strings)` – добавляет узлы или строки в конец node,
- `node.prepend(...nodes or strings)` – вставляет узлы или строки в начало node,
- `node.before(...nodes or strings)` – вставляет узлы или строки до node,
- `node.after(...nodes or strings)` – вставляет узлы или строки после node,
- `node.replaceWith(...nodes or strings)` – заменяет node заданными узлами или строками.

Пример:

```
document.body.append(test); // вставляем нашу переменную в body
```

# Методы вставки в DOM дерево

Наглядная иллюстрация того, куда эти методы вставляют:



---

## Практика

Создание To Do List, необходимо средствами JS создать страницу на которой будут элементы:

- header содержащий заголовок страницы
- контейнер с контентом страницы
- поле ввода input
- список элементов (число элементов > 1)
- кнопка добавления To Do

---

# **События**

---

## Самые популярные

- click – происходит, когда кликнули на элемент левой кнопкой мыши.
- change - срабатывает по окончании изменения элемента.
- input - срабатывает каждый раз при изменении значения.
- submit – пользователь отправил форму <form>.
- focus – пользователь фокусируется на элементе, например нажимает на <input>.
- keydown и keyup – когда пользователь нажимает / отпускает клавишу.

---

## addEventListener

Фундаментальный недостаток обработчиков это невозможность повесить несколько обработчиков на одно событие.

```
input.onclick = function() { alert(1); }
```

```
input.onclick = function() { alert(2); } // заменит предыдущий обработчик
```

Выход:

```
element.addEventListener(event, handler[, options]); // добавить
```

```
element.removeEventListener(event, handler[, options]); // удалить
```

\* удаление требует именно ту же функцию обработчик, ссылку на объект

---

## Контекст

Внутри обработчика события this ссылается на текущий элемент, то есть на тот, на котором, как говорят, «висит» (т.е. назначен) обработчик.

```
<button onclick="alert(this.innerHTML)">Нажми меня</button>
```

---

## Объект события

```
function(event) {};
```

Некоторые свойства объекта `event`:

`event.type` - Тип события, например "click".

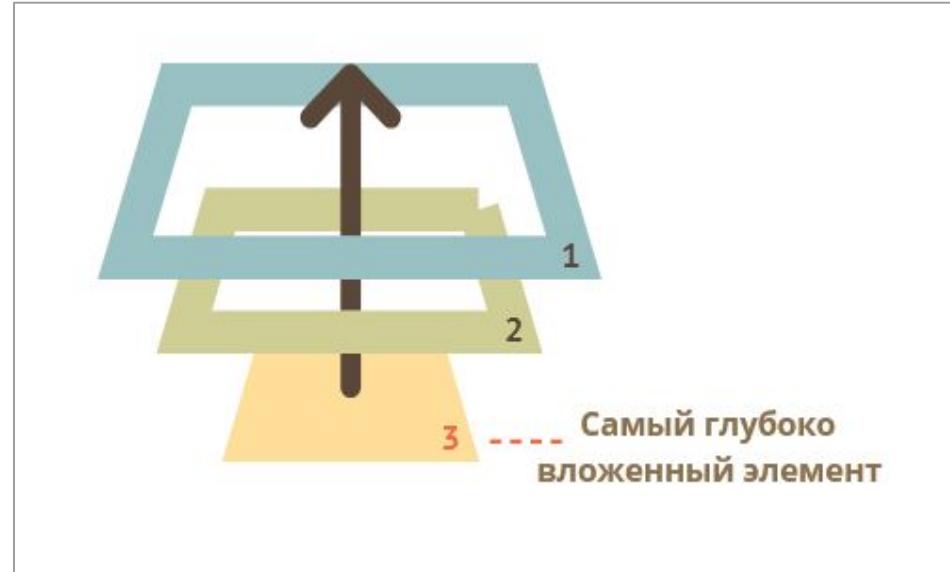
`event.currentTarget` - Элемент, на котором сработал обработчик.

`event.target` - Элемент, на котором возникло событие.

---

## Всплытие

Когда на элементе происходит событие, обработчики сначала срабатывают на нём, потом на его родителе, затем выше и так далее, вверх по цепочке предков.



---

## Прекращение всплытия

```
function(event) {  
    event.stopPropagation();  
}
```

Не прекращайте всплытие без необходимости!

Всплытие – это удобно. Не прекращайте его без явной нужды.

---

## Отмена действия браузера

Например: Клик по ссылке инициирует переход на новый URL.

- Основной способ – это воспользоваться объектом event. Для отмены действия браузера существует стандартный метод `event.preventDefault()`.
- Если же обработчик назначен через `on< событие >` (не через `addEventListener`), то также можно вернуть `false` из обработчика.

`<a href="/" onclick="event.preventDefault()">Нажми здесь</a>`

---

## Практика

Используя наработки первого задания, добавить возможность добавления To Do в список с помощью поля ввода input и кнопки создания нового To Do.