

# Тема 14. События и пользовательское взаимодействие .

хекслет колледж



## Цель занятия:

Научить студентов обрабатывать действия пользователя на странице, используя события JavaScript, и создавать интерактивные элементы интерфейса.

# Учебные вопросы:

1. Понятие события в JavaScript
2. Обработчики событий
3. Метод `addEventListener`
4. Объект события (Event)
5. События мыши
6. События клавиатуры
7. События формы и элементов ввода
8. Всплытие и перехват событий (введение)

# 1. Понятие события в JavaScript

Событие (event) — это действие или изменение состояния на веб-странице, на которое можно отреагировать с помощью JavaScript.

Примеры событий:

- Действия пользователя: клик мышью (click), двойной клик (dblclick), наведение курсора (mouseover)
- Ввод данных: изменение текста в поле (input), выбор из списка (change)
- Работа с формой: отправка формы (submit), фокус на поле (focus)
- Изменения страницы: загрузка документа (DOMContentLoaded), прокрутка (scroll)

# Для чего нужны события?

События позволяют создавать интерактивные страницы:

- реагировать на действия пользователя;
- изменять содержимое и стиль элементов;
- валидировать формы;
- управлять поведением интерфейса без перезагрузки страницы.

# Как работает событие?

- Пользователь выполняет действие (например, кликает по кнопке).
- Браузер фиксирует событие и помещает его в очередь событий.
- JavaScript обрабатывает событие через обработчик — функцию, которая выполняется при наступлении события.
- Страница может изменить содержимое, стиль или поведение элемента.

# Важные моменты:

- Одно и то же событие может иметь несколько обработчиков.
- Обработчики событий можно назначать через HTML, через свойства элемента или с помощью `addEventListener` (предпочтительный способ).
- Каждое событие сопровождается объектом события, который содержит информацию о произошедшем действии (какой элемент, тип события, координаты мыши и т.д.).

# Пример простого события

HTML:

```
<button id="btn">Нажми меня</button>
```

JavaScript:

```
const button = document.getElementById('btn');

button.onclick = function() {
  console.log('Кнопка нажата');
};
```



## Вывод:

- События — это основной способ взаимодействия пользователя с веб-страницей.
- Понимание событий необходимо для создания динамических, интерактивных интерфейсов и управления поведением элементов через JavaScript.

## 2. Обработчики событий

Обработчик события — это функция, которая выполняется в ответ на определённое событие на элементе страницы.

Используя обработчики, JavaScript позволяет веб-странице реагировать на действия пользователя.

# Способы назначения обработчиков

## 1. Через HTML-атрибут.

Можно прямо в HTML указать обработчик события с помощью атрибута, например onclick.

```
<button onclick="alert('Кнопка нажата')">Нажми меня</button>
```

Особенности:

- Простой способ для небольших страниц;
- Не рекомендуется для крупных проектов, так как смешивает разметку и логику.

Как работает:

- В атрибуте **onclick** указываем код JavaScript, который выполнится при клике по кнопке.
- Код находится прямо в HTML-разметке.

## 2. Через свойство элемента в JavaScript

Обработчик можно назначить через свойство элемента, соответствующее событию:

```
<button id="btn">Нажми меня</button>
```

```
const button = document.getElementById('btn');  
  
button.onclick = function() {  
  console.log('Кнопка нажата');  
};
```

Как работает:

- Сначала находим элемент через DOM (getElementById, querySelector и т.д.).
- Присваиваем функцию обработчика свойству события (onclick, oninput, onchange и т.д.).
- При срабатывании события выполняется присвоенная функция.

Особенности:

- Логика отделена от HTML → код более читаем.
- Можно назначить только один обработчик на одно событие.

### 3. Через addEventListener (рекомендуемый способ)

Позволяет назначать несколько обработчиков на один элемент и управлять фазами события.

```
const button = document.getElementById('btn');

button.addEventListener('click', function() {
  console.log('Первый обработчик');
});

button.addEventListener('click', function() {
  console.log('Второй обработчик');
});
```

Как работает:

- Используется метод `addEventListener`, который принимает:
- Название события ('click', 'input', 'submit' и т.д.).
- Функцию-обработчик, которая выполнится при событии.
- (Опционально) объект с настройками { `once: true`, `capture: true/false`, `passive: true/false` }.

Особенности:

- Можно назначать несколько обработчиков на одно событие (в отличие от `onclick`).
- Можно удалять обработчик через `removeEventListener`.
- Поддерживает фазу события (`capture`) и опцию `once`.



# Отличие обработчика от события

- Событие — это действие или изменение состояния (например, клик, ввод текста).
- Обработчик — функция, которая реагирует на событие, выполняя код в ответ на него.

# Пример

HTML:

```
<input id="name" type="text" placeholder="Введите имя">  
<button id="greet">Приветствие</button>
```

JavaScript:

```
const input = document.getElementById('name');  
const button = document.getElementById('greet');  
  
button.addEventListener('click', function() {  
  alert('Привет, ' + input.value + '!');  
});
```

## Вывод:

- Обработчики событий — это функции, которые выполняются при наступлении события.
- Современный подход — использовать `addEventListener` для гибкости и чистоты кода.
- Понимание назначения и работы обработчиков — ключ к созданию интерактивных веб-страниц.

### 3. Метод `addEventListener`

**`addEventListener`** — это основной и современный способ назначения обработчиков событий на элементы DOM. Он позволяет подключать несколько обработчиков на одно событие и управлять поведением событий более гибко, чем через HTML-атрибуты или свойства элемента.

Синтаксис:

**`element.addEventListener(event, handler, options);`**

# Параметры

Параметр	Описание
event	Название события (например, 'click', 'input', 'submit')
handler	Функция, которая будет выполнена при наступлении события
options	(необязательно) объект с дополнительными настройками: <ul style="list-style-type: none"><li>• capture — использование фазы захвата (true / false)</li><li>• once — обработчик выполнится только один раз (true / false)</li><li>• passive — оптимизация для событий прокрутки</li></ul>

# Пример использования:

HTML:

```
<button id="btn">Нажми меня</button>
```

JavaScript:

```
const button = document.getElementById('btn');  
  
button.addEventListener('click', function() {  
  console.log('Кнопка была нажата!');  
});
```

- При каждом клике по кнопке в консоли появится сообщение.
- Можно добавить ещё один обработчик на тот же элемент и событие:

```
button.addEventListener('click', function() {  
  console.log('Второй обработчик!');  
});
```

# Особенности метода:

- Несколько обработчиков
- В отличие от **element.onclick**, **addEventListener** позволяет назначить несколько функций на одно событие.



# Удаление обработчика

Для удаления обработчика используется метод **removeEventListener**:

```
const button = document.getElementById('btn');

function greet() {
  console.log('Привет!');
}

button.addEventListener('click', greet);
button.removeEventListener('click', greet); // обработчик удалён
```

## Вывод:

- **addEventListener** — современный и гибкий способ назначать обработчики событий.
- Позволяет подключать несколько обработчиков, управлять фазами событий и удалять обработчики.
- Является стандартным инструментом для создания интерактивных и динамичных веб-страниц.

## 4. Объект события (Event)

Когда происходит событие на элементе, JavaScript автоматически создаёт объект события и передаёт его в обработчик. Этот объект содержит всю информацию о произошедшем событии и позволяет управлять его поведением.

# Получение объекта события

```
const button = document.getElementById('btn');  
  
button.addEventListener('click', function(event) {  
  console.log(event);  
});
```

- Параметр **event** — объект события.
- Его можно назвать как угодно (e, evt, event), главное — получать его в функции обработчика.

# Основные свойства объекта события

Свойство	Описание	Пример
type	Тип события (например, 'click', 'input')	<code>console.log(event.type)</code>
target	Элемент, на котором произошло событие	<code>console.log(event.target)</code>
currentTarget	Элемент, на котором сработал текущий обработчик	<code>console.log(event.currentTarget)</code>
bubbles	Логическое значение: всплывает ли событие	<code>console.log(event.bubbles)</code>
defaultPrevented	Проверяет, вызван ли <code>preventDefault()</code>	<code>console.log(event.defaultPrevented)</code>
timeStamp	Время, когда произошло событие	<code>console.log(event.timeStamp)</code>

# Основные методы объекта события

Метод	Описание	Пример
<code>preventDefault()</code>	Отменяет действие браузера по умолчанию (например, отправку формы или переход по ссылке)	<code>event.preventDefault()</code>
<code>stopPropagation()</code>	Останавливает дальнейшее всплытие события	<code>event.stopPropagation()</code>
<code>stopImmediatePropagation()</code>	Останавливает всплытие и другие обработчики на этом элементе	<code>event.stopImmediatePropagation()</code>

# Примеры использования

## Пример 1: Узнаём целевой элемент

```
document.getElementById('btn').addEventListener('click', function(event) {  
  console.log('Вы нажали на элемент:', event.target.tagName);  
});
```

## Пример 2: Отмена действия по умолчанию

```
const link = document.querySelector('a');  
link.addEventListener('click', function(event) {  
  event.preventDefault(); // ссылка не перейдёт по адресу  
  console.log('Переход отменён');  
});
```



### Пример 3: Использование currentTarget

```
<div id="container">
  <p id="p1">container</p>
  <button id="btn1">button1</button>
</div>
```

```
const container = document.getElementById('container');
container.addEventListener('click', function(event) {
  console.log('Обработчик сработал на:', event.currentTarget.id);
  console.log('Элемент, по которому кликнули:', event.target.id);
});
```

event.target — конкретный элемент, на котором произошло событие.  
event.currentTarget — элемент, к которому привязан обработчик.

# Вывод:

- Объект события — основной источник информации о событии.
- Позволяет узнать, что произошло, где произошло, когда и как.
- С его помощью можно отменять стандартное поведение браузера и управлять порядком выполнения обработчиков.
- Освоение объекта события — ключ к созданию интерактивных и динамичных веб-страниц.

## 5. События мыши

События мыши позволяют реагировать на действия пользователя с помощью мыши — клики, наведение, перемещение и т.д. Это основа интерактивного интерфейса.

# Основные события мыши

Событие	Описание
click	Клик по элементу мышью (левая кнопка)
dblclick	Двойной клик по элементу
mousedown	Нажатие кнопки мыши на элементе
mouseup	Отпускание кнопки мыши на элементе
mousemove	Перемещение мыши над элементом
mouseover	Наведение курсора на элемент
mouseout	Уход курсора с элемента

# Примеры использования

## Пример 1: Клик по кнопке.

HTML:

```
<button id="btn">Нажми меня</button>
```

JS:


```
const button = document.getElementById('btn');

button.addEventListener('click', () => {
  alert('Кнопка нажата!');
});
```

Срабатывает при каждом клике по кнопке.

## Пример 2: Наведение мыши на элемент

HTML:

```
<div id="box" style="width:100px;height:100px;background:  lightblue;">
</div>
```

JS:

```
const box = document.getElementById('box');

box.addEventListener('mouseover', () => {
  box.style.backgroundColor = 'lightgreen';
});

box.addEventListener('mouseout', () => {
  box.style.backgroundColor = 'lightblue';
});
```

При наведении цвет меняется на зелёный, при уходе возвращается.

## Пример 3: Отслеживание движения мыши

```
document.addEventListener('mousemove', function(event) {  
  console.log(`X: ${event.clientX}, Y: ${event.clientY}`);  
});
```

Выводит координаты курсора в консоль при движении мыши.

# Особенности:

- `mouseover/mouseout` — учитывают наведение на дочерние элементы, иногда лучше использовать `mouseenter/mouseleave` для игнорирования детей.
- `click` — удобен для кнопок, ссылок, интерактивных элементов.
- Можно комбинировать с объектом события (event) для более сложного поведения (координаты, кнопка мыши, целевой элемент).



## Вывод:

- События мыши позволяют отслеживать действия пользователя и создавать интерактивный интерфейс.
- Наиболее часто используются: `click`, `mouseover`, `mouseout`.
- В сочетании с объектом события можно точно управлять поведением элементов и реагировать на действия пользователя.

## 6. События клавиатуры

События клавиатуры позволяют реагировать на действия пользователя при нажатии клавиш клавиатуры.

Они широко используются для обработки ввода текста, управления интерфейсом, валидации данных и реализации горячих клавиш.

В JavaScript используются три основных события клавиатуры:

## 1. **keydown**

Срабатывает в момент нажатия клавиши.

```
document.addEventListener('keydown', function(event) {  
  console.log('Клавиша нажата');  
});
```

Используется, когда важно отреагировать сразу (например, управление персонажем в игре).

## 2. keyup

Срабатывает в момент отпускания клавиши.

```
document.addEventListener('keyup', function(event) {  
  console.log('Клавиша отпущена');  
});
```

Удобно применять, когда требуется обработать уже введённые данные.

# Объект события клавиатуры

В обработчиках доступен объект **event**, содержащий информацию о нажатой клавише.

Основные свойства:

- `event.key` — символ или название клавиши
- `event.code` — физическая клавиша на клавиатуре
- `event.ctrlKey`, `event.shiftKey`, `event.altKey` — были ли нажаты специальные клавиши

Пример:

```
document.addEventListener('keydown', function(event) {  
  console.log(event.key);  
});
```

# Примеры использования

## Определение конкретной клавиши:

```
document.addEventListener('keydown', function(event) {  
  if (event.key === 'Enter') {  
    console.log('Нажата клавиша Enter');  
  }  
});
```

# Проверка сочетания клавиш:

```
document.addEventListener('keydown', function(event) {  
    if (event.ctrlKey && event.key === 's') {  
        event.preventDefault();  
        console.log('Ctrl + S перехвачено');  
    }  
});
```

# Где применяются события клавиатуры:

- обработка ввода в формах;
- создание горячих клавиш;
- управление интерфейсом без мыши;
- игры и интерактивные элементы.

Важно помнить:

- События клавиатуры чаще всего вешают на `document` или `input`, `textarea`.
- Не стоит перехватывать стандартные сочетания клавиш без необходимости.
- Используйте `keydown` и `keyup`, а не `keypress`.



## Вывод:

- События клавиатуры позволяют JavaScript реагировать на нажатия клавиш и получать подробную информацию о действиях пользователя, делая веб-интерфейс интерактивным и удобным.

## 7. События формы и элементов ввода

События формы и элементов ввода позволяют отслеживать действия пользователя при работе с формами: ввод текста, изменение значений, отправку формы.

Они являются основой валидации данных и интерактивного взаимодействия с пользователем.

# Основные события форм

1. **submit**. Срабатывает при попытке отправки формы.

```
<form id="myForm">
  <input type="text" name="username">
  <button type="submit">Отправить</button>
</form>
```

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

form.addEventListener('submit', (event) => {
  event.preventDefault(); // Останавливает стандартную отправку и перезагрузку
  // Теперь мы можем собрать данные вручную через FormData
  const formData = new FormData(form);
  console.log(formData.get('username'));
});
```

- Чаще всего используется для проверки данных перед отправкой.

## 2. reset

Срабатывает при нажатии кнопки сброса формы.

```
<form id="myForm">
  <input type="text" name="username">
  <button type="submit">Отправить</button>
  <button type="reset">Сброс</button>
</form>
```

```
form.addEventListener('reset', function() {
  console.log('Форма сброшена');
});
```

# События элементов ввода

## 3. input

Срабатывает каждый раз, когда изменяется значение поля.

```
<input type="text" id="nameInput">
```

```
const input = document.getElementById('nameInput');  
  
input.addEventListener('input', function() {  
  console.log(input.value);  
});
```

Подходит для моментальной проверки или отображения введённых данных.

## 4. change

Срабатывает, когда значение изменилось и элемент потерял фокус.

```
input.addEventListener('change', function() {  
  console.log('Значение изменено');  
});
```

Используется, когда не требуется реагировать на каждый ввод символа.

## 5. focus

Срабатывает, когда элемент получает фокус.

```
const input = document.getElementById('nameInput');  
  
input.addEventListener('focus', function() {  
  console.log('Поле в фокусе');  
});
```

## 6. blur

Срабатывает, когда элемент теряет фокус.

```
const input = document.getElementById('nameInput');  
  
input.addEventListener('blur', function() {  
  console.log('Поле потеряло фокус');  
});
```

Часто применяется для проверки корректности введённых данных.



## Пример. Простая проверка заполнения поля

```
input.addEventListener('blur', function() {  
  if (input.value === '') {  
    console.warn('Поле не заполнено');  
  }  
});
```

# Где чаще всего используются эти события?

- формы авторизации и регистрации;
- поиск и фильтрация;
- валидация данных;
- интерактивный ввод.

Важно помнить

- `submit` необходимо отменять через `event.preventDefault()`, если форма не должна отправляться сразу.
- Событие `input` срабатывает чаще, чем `change`.
- События `focus` и `blur` не всплывают.

## Вывод:

- События формы и элементов ввода позволяют контролировать процесс ввода данных пользователем и обеспечивать корректную работу форм и интерактивных элементов интерфейса.

## 8. Всплытие и перехват событий (введение)

При возникновении события в браузере оно не обрабатывается мгновенно только одним элементом. Событие проходит несколько этапов, что позволяет обрабатывать его на разных уровнях DOM.

В рамках вводной части достаточно понимать идею, без глубокого погружения.

# Что происходит при событии?

Когда пользователь, например, нажимает кнопку:

```
<div>  
  <button>click</button>  
</div>
```

Событие связано с кнопкой, но браузер «сообщает» о нём и другим элементам.

# Всплытие событий (bubbling)

Всплытие — это стандартное поведение событий в JavaScript.

Событие возникает на целевом элементе

Затем «всплывает» вверх по DOM-дереву:

button → div → body → html → document

Простой пример:

```
document.body.addEventListener('click', function() {  
  console.log('Клик по body');  
});
```

При клике на кнопку сообщение появится, потому что событие всплыло до body.

# Перехват событий (capturing)

Перехват — это обратный порядок:

document → html → body → ... → button

Используется редко и включается явно:

```
element.addEventListener('click', handler, true);
```

На данном этапе достаточно знать:

- перехват существует;
- по умолчанию обработчики работают на этапе всплытия.

Когда это важно:

- обработка событий у родительских элементов;
- понимание, почему срабатывают несколько обработчиков;
- основа для изучения делегирования событий позже.



Основная идея:

- Всплытие — событие идёт снизу вверх (по умолчанию).
- Перехват — сверху вниз (включается редко).
- Один клик может быть обработан несколькими элементами.

Вывод:

- Всплытие и перехват определяют порядок обработки событий в DOM. На начальном этапе достаточно понимать, что события могут «подниматься» к родительским элементам и вызывать их обработчики.

# Выводы по лекции:

- События — это реакция браузера на действия пользователя (клик, ввод с клавиатуры, отправка формы и др.).
- JavaScript позволяет создавать интерактивные веб-страницы, реагируя на эти действия без перезагрузки страницы.
- Основной и рекомендуемый способ работы с событиями — метод `addEventListener`.
- При обработке события используется объект события (`Event`), который содержит важную информацию: что произошло; на каком элементе; дополнительные данные.
- Методы объекта события, такие как `preventDefault()`, позволяют управлять стандартным поведением браузера.
- Существуют разные группы событий: события мыши (`click`, `dblclick`); события клавиатуры (`keydown`, `keyup`); события форм и элементов ввода (`input`, `change`, `submit`).

# Контрольные вопросы:

- Что называется событием в JavaScript?
- Какие действия пользователя могут вызывать события в браузере?
- Какие способы назначения обработчиков событий существуют?
- Почему метод `addEventListener` считается предпочтительным?
- Что такое обработчик события?
- Что представляет собой объект события (Event)?
- Какие полезные свойства объекта события вы знаете?
- Для чего используется метод `event.preventDefault()`?
- Какие события относятся к событиям клавиатуры?
- Какие события используются при работе с формами и элементами ввода?
- Что такое всплытие событий?
- В каком порядке по умолчанию обрабатываются события в DOM?
- В каких случаях знание механизма событий особенно важно при разработке интерфейса?

# Домашнее задание:

<https://ru.hexlet.io/courses/js-basics>

хекслет колледж

@HEXLY.KZ