Тема 16. Формы и пользовательский ввод.

хекслет колледж

Цель занятия:

Познакомить студентов с принципами работы HTML-форм и обработкой пользовательского ввода с помощью JavaScript без использования базы данных и серверной логики.

Учебные вопросы:

1. Назначение форм в веб-приложениях
2. Основные элементы HTML-форм
3. Атрибуты элементов формы
4. События форм и элементов ввода
5. Получение данных из формы с помощью JavaScript
6. Клиентская проверка данных (валидация)
7. Управление состоянием формы. Обратная связь пользователю при работе с формами.

1. Назначение форм в веб-приложениях

Форма — это часть веб-страницы, предназначенная для ввода и передачи данных пользователем.

Формы позволяют пользователю взаимодействовать с сайтом, вводя информацию и отправляя её на обработку.

Registration

Full Name

Enter your name

Username

Enter your username

Email

Enter your email

Phone Number

Enter your number

Password

Enter your password

Confirm Password

Confirm your password

Gender

Male Female Prefer not to say

Register

Зачем нужны формы?

Формы используются для:

- ввода текстовой информации;
- выбора вариантов;
- отправки данных;
- управления пользовательским взаимодействием.

Роль JavaScript при работе с формами

JavaScript позволяет:

- получать данные из полей формы;
- проверять корректность ввода;
- выводить сообщения пользователю;
- управлять поведением формы без перезагрузки страницы.

Поведение формы по умолчанию

По умолчанию HTML-форма имеет встроенное стандартное поведение, которое выполняется браузером без участия JavaScript.
При нажатии кнопки отправки (submit) браузер:

- Собирает данные из всех полей формы;
- Отправляет их на сервер;
- Перезагружает страницу или открывает страницу ответа сервера.

Пример стандартной формы (по умолчанию)

```
<form action="/send" method="POST">
  <input type="text" name="username" placeholder="Введите имя">
  <button type="submit">Отправить</button>
</form>
```

Что происходит:

- action="/send" — адрес сервера, куда отправляются данные;
- method="POST" — способ отправки данных;
- браузер отправляет данные в формате: username=Иван
- страница перезагружается.

⚠ Если сервер отсутствует, браузер покажет ошибку или пустую страницу.

Пример отправки данных через URL (метод GET)

```
<form action="https://example.com">
  <input type="text" name="search" placeholder="Поиск">
  <button>Искать</button>
</form>
```

Если ввести "js" и отправить форму, браузер перейдёт по адресу:

https://example.com/?search=js

Почему это поведение часто не подходит?

Для современных интерфейсов стандартное поведение формы неудобно, потому что:

- происходит перезагрузка страницы;
- теряется состояние интерфейса;
- невозможно показать ошибки или сообщения без новой загрузки;
- сложно управлять интерактивностью.

Формы без базы данных и без сервера

В учебных и клиентских проектах формы часто не отправляют данные на сервер вообще.
Вместо этого:

- данные обрабатываются JavaScript;
- результат сразу показывается пользователю;
- форма работает как часть интерфейса.

Отмена стандартного поведения формы

Чтобы форма не отправлялась и не перезагружала страницу, используется:
event.preventDefault();
Пример:
HTML:

```
<form id="userForm">
  <input type="text" id="username" placeholder="Введите имя" />
  <button>Отправить</button>
</form>

<p id="result"></p>
```

JS:

```
const form = document.getElementById('userForm');
const result = document.getElementById('result');

form.addEventListener('submit', function(event) {
  event.preventDefault(); // отмена перезагрузки страницы

  const name = document.getElementById('username').value;
  result.textContent = 'Здравствуйте, ' + name + '!';
});
```

В этом примере:

- данные не отправляются на сервер;
- имя обрабатывается сразу после ввода;
- результат отображается на странице.

Важно запомнить:

- Форма всегда отправляется по умолчанию.
- submit — это событие формы.
- preventDefault() — ключевой инструмент для форм без БД.
- Современные интерфейсы почти всегда отменяют стандартное поведение формы.

Вывод:

- Формы предназначены для ввода и передачи данных.
- По умолчанию форма отправляет данные на сервер и перезагружает страницу.
- JavaScript позволяет изменить это поведение.
- Формы без базы данных — это формы, обрабатываемые полностью на стороне клиента.

2. Основные элементы HTML-форм

HTML-форма состоит из контейнера <form> и элементов управления вводом, с помощью которых пользователь вводит данные.

Элемент <form>

Основной контейнер формы.

```
<form action="/send" method="post">
  ...
</form>
```

Назначение:

- объединяет поля ввода;
- определяет, куда и как отправляются данные.

Основные атрибуты:

- action — URL, на который отправляются данные;
- method — способ отправки (GET или POST);
- name / id — идентификация формы (часто используется в JS).

Поле ввода <input>

Самый универсальный элемент формы. Тип поля определяется атрибутом type.

<input type="text" name="username">

Распространённые типы:

type	Назначение
text	Текстовая строка
password	Скрытый ввод
email	Email (с базовой проверкой)
number	Числа
checkbox	Флажок
radio	Переключатель
submit	Кнопка отправки
reset	Сброс формы

📌 Атрибут name обязателен для отправки данных на сервер.

Многострочное поле <textarea>

Используется для ввода большого текста.

```
<textarea name="comment" rows="4"></textarea>
```

Особенности:

- подходит для комментариев и описаний;
- текст располагается между тегами.

Выпадающий список <select>

Позволяет выбрать значение из списка.

```
<select name="city">
  <option value="msk">Москва</option>
  <option value="spb">Санкт-Петербург</option>
</select>
```

Составные элементы:

- <select> — список;
- <option> — вариант выбора.

Элемент <label>

Описание поля формы.

```
<label for="email">Email:</label>
<input id="email" type="email">
```

Зачем нужен:

- улучшает доступность;
- при клике на текст фокус переходит на поле ввода.

Кнопки формы <button>

```
<button type="submit">Отправить</button>
<button type="reset">Очистить</button>
```

Типы кнопок:

- submit — отправляет форму;
- reset — сбрасывает значения;
- button — обычная кнопка (используется с JavaScript).

Атрибуты, связанные с вводом данных

Наиболее часто используемые:

Атрибут	Назначение
placeholder	Подсказка в поле
required	Обязательное поле
value	Значение по умолчанию
disabled	Поле недоступно
checked	Активный чекбокс / radio

Пример:

```
<input type="email" name="email" required placeholder="Введите email">
```

Пример:

```
<form action="/send" method="post">
  <label>
    Имя:
    <input type="text" name="name" required />
  </label>

  <label>
    Сообщение:
    <textarea name="message"></textarea>
  </label>

  <button type="submit">Отправить</button>
</form>
```

Вывод:

HTML-форма — это:

- контейнер <form>;
- элементы ввода (input, textarea, select);
- кнопки управления;
- атрибуты, которые задают поведение формы.

Все эти элементы работают даже без JavaScript, используя стандартный механизм браузера.

3. Атрибуты элементов формы

Атрибуты элементов формы управляют:

- внешним видом полей;
- допустимыми данными;
- поведением формы при отправке;
- взаимодействием с пользователем.

Атрибут name

<input type="text" name="login">

Назначение:

- задаёт имя поля;
- используется при отправке данных на сервер;
- без name значение поля не передаётся.

 Ключевой атрибут для любой формы.

Атрибут value

<input type="text" name="city" value="Москва">

Назначение:

- задаёт начальное значение поля;
- используется также для кнопок, radio и checkbox.

Пример с radio:

<input type="radio" name="gender" value="male">

Атрибут placeholder

```
<input type="email" placeholder="Введите email">
```

Назначение:

- отображает подсказку;
- исчезает при вводе текста;
- не является значением поля.

Атрибут required

<input type="text" required>

Назначение:

- делает поле обязательным;
- браузер не отправит форму, если поле пустое;
- встроенная валидация без JavaScript.

Атрибут type

```
<input type="password">
<input type="number">
<input type="email">
```

Назначение:

- определяет тип вводимых данных;
- влияет на проверку и внешний вид;
- для email, number есть базовая валидация.

Атрибуты min, max, step

```
<input type="number" min="1" max="10" step="1">
```

Назначение:

- задают допустимый диапазон чисел;
- учитываются браузером при вводе.

Атрибут checked

<input type="checkbox" checked>

Назначение:

- задаёт начальное состояние checkbox или radio.

Атрибут disabled

<input type="text" disabled>

Назначение:
поле недоступно для ввода;
значение не отправляется;
часто используется для блокировки формы.

Атрибут readonly

<input type="text" readonly>

Отличие от disabled:

- поле нельзя изменить;
- значение отправляется при отправке формы.

Атрибут maxlength

```
<input type="text" maxlength="20">
```

Назначение:

- ограничивает количество символов;
- полезен для логинов, кодов, комментариев.

Атрибут pattern

```
<input type="text" pattern="\d{4}">
```

Назначение:

- задаёт регулярное выражение;
- браузер проверяет значение перед отправкой.

Атрибут autocomplete

```
<input type="email" autocomplete="off">
```

Назначение:

- включает или отключает автозаполнение;
- часто используется для паролей и тестовых форм.

Атрибуты формы (<form>)

<form action="/send" method="post" novalidate>

Атрибут	Назначение
action	Адрес отправки
method	GET или POST
novalidate	Отключение проверки

Итог по атрибутам форм:

Атрибуты позволяют:

- ограничивать ввод данных;
- проверять форму без JavaScript;
- управлять доступностью элементов;
- влиять на отправку данных.

4. События форм и элементов ввода

Формы и поля ввода в JavaScript реагируют на действия пользователя с помощью событий. Эти события позволяют:

- контролировать ввод данных;
- предотвращать стандартную отправку формы;
- выполнять проверку и обратную связь.

Событие submit

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

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

form.addEventListener('submit', function (event) {
  event.preventDefault(); // отмена стандартной отправки
  console.log('Форма отправлена');
});
```

Назначение:

- срабатывает при отправке формы;
- используется для проверки данных;
- чаще всего сопровождается event.preventDefault().

 Основное событие при работе с формами.

Событие input

```
const input = document.querySelector('input');

input.addEventListener('input', function () {
  console.log(input.value);
});
```

Назначение:

- срабатывает при каждом изменении значения;
- реагирует на ввод, удаление, вставку текста;
- подходит для онлайн-проверок и подсказок.

Событие change

```
const input = document.querySelector('input');

input.addEventListener('input', function () {
  console.log(input.value);
});
```

Назначение:

- срабатывает после потери фокуса;
- актуально для select, checkbox, radio;
- не реагирует на каждую клавишу.

📌 Часто используется там, где не нужен постоянный контроль.

События фокуса: focus и blur

```
input.addEventListener('focus', function () {
  console.log('Поле активировано');
});

input.addEventListener('blur', function () {
  console.log('Поле потеряло фокус');
});
```

Назначение:

- focus — пользователь начал ввод;
- blur — пользователь покинул поле;
- удобно для подсказок и валидации.

Событие reset

```html
<form id="form">
  <input type="text">
  <button type="reset">Очистить</button>
</form>
```

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

Назначение:

- реагирует на очистку формы;
- используется для сброса состояний интерфейса.

События для checkbox и radio

```
checkbox.addEventListener('change', function () {
  console.log(checkbox.checked);
});
```

Особенности:

- используется checked;
- чаще всего применяется change, а не input.

Сравнение основных событий:

Событие	Когда срабатывает
submit	Отправка формы
input	Каждый ввод
change	После изменения
focus	Получение фокуса
blur	Потеря фокуса
reset	Сброс формы

События форм позволяют:

- управлять вводом пользователя;
- проверять данные до отправки;
- реализовывать интерактивную обратную связь;
- заменять серверную логику на клиентскую.

5. Получение данных из формы с помощью JavaScript

JavaScript позволяет получать значения, введённые пользователем в форму, и использовать их:

- для проверки данных;
- для обработки формы без сервера;
- для отображения результата на странице;
- для подготовки данных к отправке.

Получение значения поля через value

Самый простой и распространённый способ.

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

const loginInput = document.getElementById('login');
console.log(loginInput.value);
```

📌 Свойство value содержит текущее значение поля ввода.

Получение данных при отправке формы

```html
<form id="form">
  <input type="text" name="username">
  <input type="password" name="password">
  <button type="submit">Войти</button>
</form>
```

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

form.addEventListener('submit', function (event) {
  event.preventDefault();
  const username = form.username.value;
  const password = form.password.value;
  console.log(username, password);
});
```

Пояснение:

- **form.nameПоля.value** — быстрый доступ к полям формы;
- имена берутся из атрибута name.

Получение данных из нескольких полей

```
<input type="email" name="email">
<input type="number" name="age">

const email = form.email.value;
const age = form.age.value;
```

Все значения из формы получаются в виде строк.

Получение значения checkbox

```html
<input type="checkbox" id="agree">
```

```js
const agree = document.getElementById('agree').checked;
```

Особенность:

- используется свойство checked, а не value;
- возвращает true или false.

Получение значения radio-кнопок

```html
<form id="form">
  <input type="radio" name="gender" value="male">
  <input type="radio" name="gender" value="female">
  <button type="submit">Send</button>
</form>
```

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

form.addEventListener('submit', (e) => {
  e.preventDefault(); // Чтобы страница не перезагрузилась
  const gender = form.gender.value;
  console.log('Выбрано:', gender);
});
```

value вернёт значение выбранной radio-кнопки.

Получение значения select

```html
<select name="city">
  <option value="spb">Санкт-Петербург</option>
  <option value="msk">Москва</option>
</select>
```

```
const city = form.city.value;
```

Получение всех данных формы

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

form.addEventListener("submit", function(event){
  event.preventDefault();
  const formData = new FormData(form);
  // FormData нужно преобразовать в обычный объект
  const data = Object.fromEntries(formData.entries());
  console.log(data) // {login: 'user', passwd: '1234'}
})
```

Объект FormData автоматически собирает все пары «имя-значение» из полей формы.

Итог:

JavaScript позволяет:

- получать данные из любых полей формы;
- работать с данными до отправки;
- реализовывать интерактивные сценарии без сервера;
- подготовить форму к реальной серверной обработке.

6. Клиентская проверка данных (валидация)

Клиентская валидация — это проверка данных формы на стороне браузера до отправки формы.
Цели валидации:

- предотвратить отправку некорректных данных;
- подсказать пользователю, что введено неверно;
- улучшить удобство и качество ввода.

Валидация средствами HTML

HTML предоставляет встроенные механизмы проверки:

```
<input type="text" required>
<input type="email">
<input type="number" min="18" max="99">
```

Основные атрибуты:

- required — обязательное поле;
- type="email" — проверка формата email;
- min, max, maxlength, pattern.

При использовании этих атрибутов браузер сам блокирует отправку формы.

Зачем нужна JavaScript-валидация?

JavaScript используется, когда требуется:

- сложная логика проверки;
- проверка нескольких полей одновременно;
- собственные сообщения об ошибках;
- динамическая обратная связь.

Пример простой проверки при отправке формы

```
form.addEventListener('submit', function (event) {
  event.preventDefault();

  if (form.login.value === '') {
    alert('Введите логин');
    return;
  }

  console.log('Форма прошла проверку');
});
```

Если данные неверны — форма не обрабатывается дальше.

Проверка длины и формата данных:

```
form.addEventListener('submit', function (event) {
  event.preventDefault();

  if (form.password.value.length < 6) {
    alert('Пароль должен быть не менее 6 символов');
    return;
  }
  console.log('Форма прошла проверку');
});
```

Проверка checkbox:

```
form.addEventListener('submit', function (event) {
  event.preventDefault();

  if (!agree.checked) {
    alert('Необходимо согласие');
    return;
  }
  console.log('Форма прошла проверку');
});
```

Обратная связь вместо alert (рекомендуется):

```
<p id="error"></p>
```

```
error.textContent = 'Поле заполнено неверно';
error.style.color = 'red’;
```

Это более современный и удобный подход.

Пример

```
form.addEventListener('submit', function (event) {
  event.preventDefault();

  if (form.login.value === '' || form.email.value === '') {
    error.textContent = 'Заполните все поля';
    return;
  }

  error.textContent = '';
  result.textContent = 'Форма успешно отправлена';
});
```

Итог:

Клиентская валидация:

- выполняется до отправки формы;
- снижает количество ошибок;
- улучшает пользовательский опыт;
- не заменяет серверную проверку, но дополняет её.

7. Управление состоянием формы. Обратная связь пользователю при работе с формами.

Состояние формы — это текущее положение формы и её элементов в процессе взаимодействия с пользователем:

- ожидание ввода;
- наличие ошибок;
- корректное заполнение;
- успешная обработка данных.

JavaScript позволяет управлять этими состояниями и наглядно сообщать пользователю, что происходит.

Основные состояния формы

Типичные состояния:

- пустая форма;
- форма с ошибками;
- форма заполнена корректно;
- форма отправлена (успех);
- форма заблокирована (например, кнопка отключена).

Каждое состояние должно быть визуально понятно пользователю.

Управление классами состояния

Часто применяются CSS-классы:

- error
- valid
- disabled
- Hidden и пр.

```
form.classList.add('error');
form.classList.remove('error’);
```

Логика управления состоянием — в JavaScript, внешний вид — в CSS.

Пример, реакция на ввод пользователя:

```
input.addEventListener('input', function () {
  if (input.value === '') {
    input.classList.add('error');
  } else {
    input.classList.remove('error');
  }
});
```

Выводы по лекции:

- Форма — это элемент веб-страницы для ввода данных пользователем.
- По умолчанию форма отправляет данные на сервер и перезагружает страницу.
- JavaScript позволяет обрабатывать форму на стороне клиента без сервера.
- Основные элементы формы: <input>, <textarea>, <select> и <button>.
- Атрибуты формы (name, value, placeholder, required, checked, disabled) управляют поведением и вводом.
- События формы (submit, input, change, focus, blur, reset) помогают реагировать на действия пользователя.
- Значения полей можно получать через .value или .checked, а все данные формы — через FormData.
- Валидация позволяет проверять корректность данных до отправки формы.
- Сообщения об ошибках и успешной отправке показываются пользователю через текст и CSS-классы.
- Классы для состояния (error, success, disabled) делают интерфейс наглядным и удобным.
- Кнопка отправки может блокироваться до исправления ошибок.
- После успешной отправки форма может быть сброшена.
- Формы без сервера полезны для обучения и интерактивных интерфейсов.

Контрольные вопросы:

- Что такое HTML-форма и для чего она используется?
- Какое поведение формы по умолчанию при нажатии кнопки отправки?
- Какие основные элементы формы вы знаете? Приведите примеры.
- Назовите ключевые атрибуты полей формы и их назначение (name, value, required, placeholder, checked, disabled).
- Какие события формы и полей ввода существуют? Для чего они применяются?
- Как получить значение текстового поля, чекбокса и radio-кнопки с помощью JavaScript?
- Что такое клиентская проверка (валидация) и зачем она нужна?
- Как можно показывать пользователю ошибки или подтверждения при заполнении формы?
- Чем отличается форма с серверной отправкой от формы без базы данных?
- Почему лучше использовать CSS-классы для управления состоянием формы вместо прямого изменения стиля через element.style?

Домашнее задание:

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

хекслет колледж

@HEXLY.KZ