



# РАБОТА С HTML-ФОРМАМИ



МИХАИЛ КУЗНЕЦОВ



# МИХАИЛ КУЗНЕЦОВ

Разработчик в ING Bank



[@mkuznetcov](https://www.telegram.me/mkuznetcov)

# ПЛАН ЗАНЯТИЯ

1. Чтение и запись значений элементов форм. События полей форм.
2. Формы на странице `document.forms`. События форм
3. Валидация полей форм при помощи `checkValidity`



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Вопрос:** какой новое свойство, что позволяет работать с классами, вы помните?



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Ответ:** свойство для работы с классами - `Element.classList`.



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Вопрос:** через какое свойство можно обращаться к стилям элемента?



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Ответ:** свойство для работы со стилями - `Element.style`.



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Вопрос:** все ли атрибуты одинаковы?



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Ответ:** не все, существуют отдельный тип атрибутов - пользовательские data-атрибуты, для которых существует даже отдельное свойство `Element.dataset`.



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Вопрос:** сколько событий, что говорят о разных степенях готовности страницы, вы знаете?

---

## ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Ответ:** Пока мы знаем два события: `DOMContentLoaded` - загружен DOM и `window.onload` - обработан DOM и загружены все ресурсы страницы.

# ЗАЧЕМ НУЖНО УМЕТЬ РАЗРАБАТЫВАТЬ ФОРМЫ НА JAVASCRIPT?

Формы с давних времен являются стандартом для получения данных от пользователя.

Единственная альтернатива формам - это встроенные возможности браузера в виде диалоговых окон `prompt` и `confirm`, которые использовались нами ранее для взаимодействия с пользователем.

```
const years = prompt('Сколько вам лет?', 100);  
alert('Вам ' + years + ' лет!')
```

```
const isStudent = confirm("Вы студент?");  
alert(isStudent);
```

## НЕДОСТАТКИ `prompt`

Давайте подумаем, какие же недостатки у `prompt` ?

1. Синхронная работа функций: программа возобновит свою работу только после закрытия такого окна. Поведение диалоговых окон, встроенных в браузер, вызывает блокировку выполнения всего кода на JavaScript, в том числе таймеров, обработчиков событий, которые были назначены ранее и наступили, и всего последующего кода.
2. Невозможность стилизации: CSS-стили недоступны для элементов интерфейса браузера.
3. Задав дополнительную опцию в настройках браузера, пользователь вообще может отключить показ диалоговых окон, что может сделать невозможной дальнейшую работу с вашим сайтом.
4. Политика производителей браузеров не рекомендует пользоваться данными методами 1.

## НЕДОСТАТКИ `confirm`

У `confirm` аналогичные недостатки, плюс в ответ можно получить только `true` / `false`.

Давайте же рассмотрим, каким образом можно взаимодействовать с HTML-формами посредством JavaScript.

# ЗАДАЧА: ФОРМА РЕГИСТРАЦИИ

Допустим, перед нами стоит задача создать простую форму регистрации пользователя для сайта по управлению семейными финансами, приведенная на рисунке:

ФИО:

E-mail:

Выберите страну проживания:

Россия  
Азербайджан  
Армения  
Белоруссия

Тип аккаунта:

☐ Личный ☐ Семейный ☒ Бизнес

☒ согласен на обработку персональных данных

Комментарии:

Зарегистрироваться

# ЗАДАЧА: ФОРМА РЕГИСТРАЦИИ

Как мы видим, на ней представлены следующие поля:

- ФИО, текстовое поле;
- E-mail, текстовое поле;
- Выберите страну проживания, `select` ;
- Тип аккаунта, `radio` ;
- Согласен на обработку персональных данных, `checkbox` ;
- Комментарии, `textarea` ;
- Кнопка «Зарегистрироваться».

В итоге по нажатию на кнопку мы должны отправить на сервер полученные от пользователя данные.

Давайте вместе разберем работу со всеми этими типами полей.



# ТИПЫ ПОЛЕЙ И СОБЫТИЯ

В таблице приведены поля и их основные события, которые мы сегодня рассмотрим.

Типы полей	<code>input text, textarea</code>	<code>select, radio, checkbox</code>
События	<code>input, change, focus, blur</code>	<code>change, (focus, blur)</code>

---

# ЧТЕНИЕ И ЗАПИСЬ ЗНАЧЕНИЙ ЭЛЕМЕНТОВ ФОРМ. СОБЫТИЯ ПОЛЕЙ ФОРМ

# ТЕКСТОВОЕ ПОЛЕ `type="text"`. ЧТЕНИЕ ЗНАЧЕНИЯ

Рассмотрим пример получения введенного значения из поля ФИО.

Получение данных, введенных в поле, происходит в два этапа:

1. Получить ссылку на элемент поля ввода в дереве DOM.
2. Получить значение, введенное в это поле.

Мы знаем, как реализовать первый пункт из предыдущих занятий. С помощью методов объекта `document`: `querySelector`, `getElementById` и т.д. А второй пункт решается обращением к свойству `value` полученного узла.

# ТЕКСТОВОЕ ПОЛЕ `type="text"`. ЧТЕНИЕ ЗНАЧЕНИЯ

HTML-разметка поля ФИО:

```
1 <label>
2   ФИО: <input type="text" id="fio" name="fio">
3 </label>
4 <button id="registerButton">Зарегистрироваться</button>
```

Это пригодится при выполнении домашнего задания.

JavaScript-код, в котором в консоль выводится ФИО:

```
1 const button = document.getElementById('registerButton');
2
3 button.addEventListener('click', e => {
4   const name = document.getElementById('fio');
5   const user = name.value;
6   console.log(`Пользователь ${user} зарегистрирован`);
7 });
```

## ТЕКСТОВОЕ ПОЛЕ `type="text"`. ЗАПИСЬ ЗНАЧЕНИЯ

Рассмотрим, каким образом можно программно изменять значение текстового поля. Представим, что нам необходимо, чтобы e-mail пользователя при нажатии на кнопку «Зарегистрироваться» очищался от пробелов в начале и в конце.

```
1  const button = document.getElementById('registerButton');
2
3  button.addEventListener('click', e => {
4    const email = document.getElementById('email');
5    email.value = email.value.trim();
6    console.log(`Пользователь ${email.value} зарегистрирован`);
7  });
```

Метод `trim()` в строке 5 удаляет пробельные символы с начала и конца строки, и мы записываем новое значение в поле e-mail.

# СОБЫТИЯ ТЕКСТОВОГО ПОЛЯ

## `type="text"`

Рассмотрим наиболее часто использующиеся события текстового поля.

## СОБЫТИЕ `input`

На всех текстовых полях ввода событие `input` возникает каждый раз, когда мы вводим новый символ, удаляем символ, или еще как-то меняем введенное значение. В том числе с помощью вставки, вырезания и так далее.

В примере ниже при вводе значения в текстовое поле оно отображается в элементе `result`.

```
1 | const fio = document.getElementById('fio');  
2 |  
3 | fio.oninput = () =>  
4 |   document.getElementById('result').innerHTML = fio.value;
```

То же самое с использованием `addEventListener`:

```
1 | onInput = (e) => document.getElementById('result').innerHTML = e.target.value;  
2 |  
3 | fio.addEventListener("input", onInput);
```

## СОБЫТИЕ `change`

Есть также событие `change`, которое доступно и на текстовых полях, и на всех остальных (чекбоксы, радиокнопки, списки выбора). Событие `change` на текстовых полях возникает при потере фокуса.

В примере ниже при изменении значения и последующей потере фокуса в поле ФИО его значение отображается в элементе `result`.

```
1  const fio = document.getElementById('fio');  
2  
3  fio.onchange = () =>  
4    document.getElementById('result').innerHTML = fio.value;
```



## СОБЫТИЯ `focus`, `blur`

В проектировании интерфейсов есть понятие «фокус ввода». Пользователь одновременно может вводить данные только в одно поле. Считается, что поле находится в фокусе, если пользователь сейчас имеет возможность вводить данные в это поле.

Признаком фокуса может служить наличие курсора в этом поле, а так же изменение внешнего вида этого поля. Переходя к заполнению следующего поля, текущее поле теряет фокус. Именно фокус ввода переключается с помощью клавиши `Tab` в большинстве интерфейсов.

У элементов HTML-форм есть специализированные события для работы с фокусом:

- `focus` — когда поле ввода становится активным (в фокусе) с помощью клика мышки по полю или по метке поля либо при переключении фокуса клавишей `Tab`;
- `blur` — когда поле ввода теряет фокус.

# ЗАДАЧА: ВЫВОД ПОДСКАЗКИ НА ПОЛЕ ПРИ ФОКУСЕ

Необходимо показывать подсказку, как заполнять поле именно в тот момент, когда пользователь его заполняет.

HTML-разметка для нашего примера следующая:

```
1 <div>
2   <label>E-mail:</label>
3   <input type="text" placeholder="E-mail" name="e-mail" id="e-mail">
4   <div class="hint hidden">Адрес электронной почты должен содержать "@"</div>
5 </div>
```

# ЗАДАЧА: ВЫВОД ПОДСКАЗКИ НА ПОЛЕ ПРИ ФОКУСЕ

Подсказка в элементе `<div>` по умолчанию скрыта. И нам необходимо показать её, убрав класс `hidden` в тот момент, когда пользователь начнет вводить сообщение. И скрывать, вернув `hidden`, когда пользователь закончит ввод.

Решение:

```
1  const email = document.getElementById('e-mail');
2  const hint = document.querySelector('.hint');
3
4  showHint() => hint.classList.remove('hidden');
5
6  hideHint() => hint.classList.add('hidden');
7
8  email.addEventListener('focus', showHint);
9  email.addEventListener('blur', hideHint);
```

# МНОГОСТРОЧНОЕ ТЕКСТОВОЕ ПОЛЕ

## `textarea`

Запись и чтение из `textarea` производится аналогично тому, как это происходит для текстового поля.

### СОБЫТИЯ `textarea`

События `input`, `change`, `focus`, `blur` на поле `textarea` обрабатываются так же, как мы уже рассмотрели для текстового поля.

## ПОЛЕ ДЛЯ ВЫВОДА `output`

Поле `output` определяет не редактируемую для пользователя область, в которую выводится информация. Чтение и программная запись производится аналогично тому, как это происходит для текстового поля.

### СОБЫТИЯ `output`

Поскольку `output` не изменяемо, то и событий `input`, `change` оно не имеет. События `focus`, `blur` на поле `output` обрабатываются так же, как мы уже рассмотрели для текстового поля.

## СПИСОК `select`. ЧТЕНИЕ ЗНАЧЕНИЯ

Далее перейдем к выбору пользователем страны проживания с формы регистрации.

HTML-разметка поля:

```
1 <label>
2   Выберите страну проживания:
3 </label>
4 <select id="country">
5   <option value="RUS" selected>Россия</option>
6   <option value="AZE">Азербайджан</option>
7   <option value="ARM">Армения</option>
8   <option value="BLR">Белоруссия</option>
9   <option value="KAZ">Казахстан</option>
10  <option value="KGZ">Киргизия</option>
11 </select>
12 <button id="registerButton">Зарегистрироваться</button>
```

Обратите внимание, что для выбора значения по умолчанию используется атрибут `selected`.

## СПИСОК `select`. ЧТЕНИЕ ЗНАЧЕНИЯ

Это пригодится при выполнении домашнего задания.

Рассмотрим JavaScript-код, в котором в консоль выводится выбранная пользователем страна проживания при ее изменении (`change`):

```
1  const countryList = document.getElementById('country');
2
3  countryList.addEventListener('change', event => {
4    console.log(countryList.value);
5    // значение value выбранного элемента (RUS)
6    console.log(countryList.selectedIndex);
7    // порядковый номер выбранного элемента
8    console.log(countryList.options[countryList.selectedIndex].text);
9    // текст выбранной опции (Россия)
10 });
```

## СПИСОК `select`. ЧТЕНИЕ ЗНАЧЕНИЯ

- Свойство `value`, как и в случае с текстовым полем, содержит значение свойства `value` выбранной пользователем опции (в примере RUS, AZE, ARM...).
- Свойство `selectedIndex` показывает порядковый номер выбранной опции `option` (начиная с 0).
- Список элементов-опций доступен через `select.options`
- Выбранные опции имеют свойство `option.selected = true`.
- Свойство `text` выбранной пользователем опции содержит его текст (в примере Россия, Азербайджан...), строка 6.

Аналогично мы могли бы получить выбранное значение и при нажатии на кнопку «Зарегистрироваться», как мы это делали в примере с получением ФИО и email.



## СПИСОК `select`. ЗАПИСЬ ЗНАЧЕНИЯ

Представим, что нам необходимо программно проставить значение в списке. Например, по геолокации, доступной в браузере, мы можем предположить и проставить страну проживания по умолчанию.

Сделать это можно двумя способами: поставив значение `select.value`, либо установив свойство `select.selectedIndex` в номер нужной опции.

```
1  const countryList = document.getElementById( 'country' );
2  countryList.selectedIndex = 2; // по порядковому номеру с 0
3  countryList.value = "ARM"; // ИЛИ по значению value
```

## СПИСОК `select`. ВЫБОР НЕСКОЛЬКИХ ЗНАЧЕНИЙ

При помощи атрибута `multiple` можно создать список с возможностью множественного выбора.

На современных сайтах / веб-приложениях этот вид HTML-элемента практически не встречается. Как правило, если необходим список с возможностью выбора нескольких значений, используют различные JavaScript-компоненты, совместимые с используемыми в проекте библиотекой / фреймворком (React, Angular, Vue...). Их преимущества: кастомизация внешнего вида компонента, а также возможны дополнительные функции (типа поиска по всем элементам).

Для чтения выбранных элементов и их записи в таком списке приходится работать с массивом, что несколько сложнее. При желании вы можете разобраться со списком самостоятельно, см. ссылку из доп. материалов 2.

## СОБЫТИЯ `select`

На списках (а также чекбоксах и радиокнопках) событие `change` возникает при выборе нового значения. Обрабатывается оно так же, как мы уже рассмотрели для текстового поля. Рассмотрим пример для списка с возможностью выбора одного значения:

```
1  const country = document.getElementById( 'country' );  
2  
3  country.onchange = () =>  
4    document.getElementById( 'result' ).innerHTML = country.value;
```

События `focus`, `blur` для данного типа поля не являются такими популярными для обработки, как для `input text`, обычно достаточно обработки `change`.

# ЧТО БУДЕТ ВЫВЕДЕНО?

HTML-разметка поля:

```
1 <label>
2   Выберите способ оплаты:
3 </label>
4 <select id="payMethod">
5   <option value="cardSite" selected>Картой на сайте</option>
6   <option value="cardDelivery">Картой курьеру</option>
7   <option value="cashDelivery">Наличными курьеру</option>
8 </select>
9 <button id="send">OK</button>
```

```
1 const payMethods = document.getElementById('payMethod');
2
3 payMethods.addEventListener('change', event => {
4   const { value, options, selectedIndex } = event.currentTarget;
5   console.log(value);
6   console.log(options[selectedIndex].text);
7 });
```

Что будет выведено при клике на «Картой курьеру» в строках 5 и 6?

---

## ЧТО БУДЕТ ВЫВЕДЕНО?

Ответ: `cardDelivery` и Картой курьеру.

Обратите внимание, как внутри обработчика событий используется `event`.

## РАДИО-ГРУППА `radio`. ЧТЕНИЕ ЗНАЧЕНИЯ

Перейдем к выбору пользователем типа аккаунта с формы регистрации.  
HTML-разметка поля:

```
1  Тип аккаунта:
2  <label>
3      <input type="radio" name="type" value="Личный" checked id="private">
4      Личный
5  </label>
6  <label>
7      <input type="radio" name="type" value="Семейный" id="family">
8      Семейный
9  </label>
10 <label>
11     <input type="radio" name="type" value="Бизнес" id="business">
12     Бизнес
13 </label>
```

## РАДИО-ГРУППА `radio`. ЧТЕНИЕ ЗНАЧЕНИЯ

Обратите внимание, что если элементы имеют один и тот же `name`, тогда они считаются группой radio-кнопок. Иначе можно будет выбирать каждую radio-кнопку по отдельности (при выборе одной не будет сбрасываться выбор с других).

Для выбора значения по умолчанию используется атрибут `checked`.

# РАДИО-ГРУППА `radio`. ЧТЕНИЕ ЗНАЧЕНИЯ

JavaScript-код, в котором в консоль выводится выбранный пользователем тип:

```
1  const button = document.getElementById('registerButton');
2
3  button.addEventListener('click', e => {
4      const typeRadios = document.getElementsByName('type');
5      for (let i=0; i < typeRadios.length; i++) {
6          if (typeRadios[i].checked) {
7              console.log(`Выбран тип ${typeRadios[i].value}`);
8          }
9      }
10 });
```



## РАДИО-ГРУППА `radio`. ЗАПИСЬ ЗНАЧЕНИЯ

Самый простой способ для программной простановки значения группы radio-кнопок использовать `id` элемента:

```
document.getElementById("family").checked = true;
```

## СОБЫТИЯ `radio`

На радиокнопках событие `change` возникает при выборе нового значения. Рассмотрим пример, в котором при изменении значения оно выводится в элемент `result`:

```
1  conts typeRadios = document.getElementsByName('type');  
2  
3  for (var i = 0; i < typeRadios.length; i++) {  
4      typeRadios[i].addEventListener('change', (evt) => {  
5          const { value } = evt.target;  
6          document.getElementById('result').innerHTML = value;  
7      });  
8  }
```

События `focus`, `blur` для данного типа поля не являются такими популярными для обработки, как для `input text`, обычно достаточно обработки `change`.

## ЧЕКБОКС `checkbox` . ЧТЕНИЕ ЗНАЧЕНИЯ

Перейдем к обработке чекбокса «Согласен на обработку персональных данных» с формы регистрации.

HTML-разметка поля:

```
<label>  
  <input type="checkbox" name="isAgree" id="isAgree" checked> согласен на обработку персональных данных  
</label>
```

Для выбора значения по умолчанию используется атрибут `checked`.

## ЧЕКБОКС `checkbox`. ЧТЕНИЕ ЗНАЧЕНИЯ

Для проверки, установлен ли чекбокс, используется свойство `checked`.

**Это пригодится при выполнении домашнего задания.**

```
const checkbox = document.getElementById("isAgree");  
console.log(checkbox.checked); // true / false
```

У чекбоксов определен CSS псевдокласс для неопределенного состояния, `:indeterminate`, предлагаем ознакомиться с ним самостоятельно, см. ссылку из доп. материалов 3.

## ЧЕКБОКС `checkbox` . ЗАПИСЬ ЗНАЧЕНИЯ

Это пригодится при выполнении домашнего задания.

Для программной простановки значения чекбокса можно использовать следующий код:

```
document.getElementById("isAgree").checked = true; // или false
```

## СОБЫТИЯ `checkbox`

На чекбоксах событие `change` возникает при выборе нового значения. Рассмотрим пример, в котором при изменении состояния оно выводится в элемент `result`:

```
1 | const isAgree = document.getElementById('isAgree');  
2 |  
3 | isAgree.onchange = () => document.getElementById('result').innerHTML = isAgree.checked;  
4 | // true / false
```

События `focus`, `blur` для данного типа поля не являются такими популярными для обработки, как для `input text`, обычно достаточно обработки `change`.

## АТТРИБУТ `disabled`

При помощи атрибута `disabled` можно заблокировать поля формы разных типов, чтобы они были недоступны для изменения пользователем. При этом значение в поле все еще можно считать. Пример задизэйбленного текстового поля:

```
<label>  
  ФИО: <input type="text" id="fio" disabled>  
</label>
```

## АТРИБУТ disabled

При помощи следующего кода можно проверить, является ли элемент задизейбленным:

```
console.log(document.getElementById(fio).disabled);  
// true или false
```


При помощи следующего кода можно проставить атрибут disabled:

```
document.getElementById(fio).disabled = true  
// или false
```



# ИТОГО: ТИПЫ ПОЛЕЙ И СОБЫТИЯ

Типы полей	<code>input text, textarea</code>	<code>select, radio, checkbox</code>
События	<code>input, change, focus, blur</code>	<code>change, (focus, blur)</code>



**ФОРМЫ НА СТРАНИЦЕ**  
**document.forms.**  
**СОБЫТИЯ ФОРМ**

# ФОРМЫ HTML-СТРАНИЦЫ:

## `document.forms`

К конкретной форме на странице можно получить доступ несколькими способами:

1. Через ее `id`;

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

2. Через `document.forms`. Если форме добавить атрибут `name`, то к ней можно получить доступ через свойство `forms` элемента `document`, в котором хранится коллекция всех форм на странице.

```
<form name="register-form">  
  <!-- ... -->  
</form>
```

```
const form = document.forms['register-form'];  
// ...
```

# ОТПРАВКА ФОРМЫ НА СЕРВЕР, submit

В итоге у нас получилась форма регистрации со следующей разметкой (представлена в сокращенном виде):

```
1 <form name="register-form" method="get" action="">
2   <div>
3     <label>ФИО:</label>
4     <input type="text" placeholder="ФИО" name="fio" id="fio">
5   </div>
6   <div>
7     <label>Выберите местонахождение:</label>
8     <select name="country" id="country">
9       <option value="RUS">Россия</option>
10    <!-- ... -->
11  </select>
12 </div>
13 <!-- еще поля -->
14 <div>
15   <label><input type="checkbox" name="isAgree" id="isAgree" checked> согласен на обработку персональных данных </label>
16 </div>
17 <button name="register-button" id="register-button" type="submit">Зарегистрироваться</button>
18 </form>
```

Обработаем отправку данных:

```
1 const form = document.forms['register-form'];
2
3 form.addEventListener('submit', event => {
4   // тут может быть обработка данных до отправки формы
5 });
```

# МОЖНО ЛИ ОБРАБАТЫВАТЬ НАЖАТИЕ НА КНОПКУ ДЛЯ САБМИТА ФОРМЫ?

```
1  const registerButton = form["register-button"];
2
3  registerButton.addEventListener('click', e => {
4    // тут может быть обработка данных до отправки формы
5  });
```

## Варианты ответов:

1. Нет, можно обрабатывать только `submit` формы;
2. Можно, если кнопка имеет тип `submit`.



## МОЖНО ЛИ ОБРАБАТЫВАТЬ НАЖАТИЕ НА КНОПКУ ДЛЯ САБМИТА ФОРМЫ?

Ответ: 2.

## ПРИ ОТПРАВКЕ ФОРМЫ НА СЕРВЕР ОБРАТИТЕ ВНИМАНИЕ:

- чтобы отработало событие `submit` формы, `button` должен иметь `type="submit"` или тип должен отсутствовать (у кнопки тип `submit` по умолчанию). Если же у кнопки другой тип, например, `button`, то форму все равно можно засабмитить вручную:

```
1 registerButton.addEventListener('click', e => {  
2   // тут может быть обработка данных до отправки формы  
3   form.submit(); // сабмитим так, если у кнопки type="button"  
4 });
```

- форме нужно указать `action`, это URI программы на сервере, которая будет обрабатывать запрос и возвращать ответ. В нашем случае он пустой.

## ПРИ ОТПРАВКЕ ФОРМЫ НА СЕРВЕР ОБРАТИТЕ ВНИМАНИЕ:

- форме можно указать метод, которым будет происходить отправка на сервер (в нашем примере `method="get"`). Может принимать значения `get` / `post` (соответствуют одноименным HTTP методам). В случае `get` данные из формы добавляются к URI атрибута `action`, их разделяет `?`, и полученный URI посылается на сервер (можем его видеть в адресной строке браузера). В случае `post` данные из формы включаются в тело формы и посылаются на сервер.
- после отправки формы **страница перезагружается**.



# ОБРАБОТКА ФОРМЫ НА КЛИЕНТСКОЙ СТОРОНЕ

А что, если нам не нужно отправлять форму на сервер для обработки? Или мы хотим, чтобы данные формы передавались без перезагрузки страницы, при помощи AJAX?

Тогда нужно изменить наш пример следующим образом:

- `button` должен иметь `type="button"`;
- убираем атрибуты формы `action` и `method`;
- обрабатываем событие нажатия на кнопку, а не `submit` формы.

# ОБРАБОТКА ФОРМЫ НА КЛИЕНТСКОЙ СТОРОНЕ

Это пригодится при выполнении домашнего задания.

```
1  const form = document.forms['register-form'];
2
3  button.addEventListener('click', e => {
4    // тут может быть отправка формы через AJAX
5    const fio = document.getElementById('fio');
6    console.log(`Пользователь ${fio} зарегистрирован`);
7  });
```

# ОБРАБОТКА ФОРМЫ НА КЛИЕНТСКОЙ СТОРОНЕ

Если по какой-то причине кнопка имеет тип `submit`, то можно добиться такого же результата как у кода выше, отменив в обработке события `submit` действия браузера по умолчанию:

```
1  const form = document.forms['register-form'];
2
3  form.addEventListener('submit', event => {
4    event.preventDefault(); // из-за этой строки форма не сабмитится!
5    // тут может быть отправка формы через AJAX
6    const fio = document.getElementsByName('fio');
7    console.log(`Пользователь ${fio} зарегистрирован`);
8  });
```

Через одну лекцию мы научимся отправлять запросы посредством `XMLHttpRequest` (в т.ч. асинхронно), тогда у нас появится гораздо больше возможностей.

## СБРОС ФОРМЫ `reset`

Помимо события `submit` на форме наступает событие `reset`, которое браузер тоже обрабатывает сам. Он возвращает форму в то состояние, которое задано в HTML разметке изначально. Допустим, у нас в форме уже задано имя пользователя (например, на сервере):

```
1 <form name="register-form" method="get" action="">
2   <div>
3     <label>ФИО:</label>
4     <input type="text" placeholder="ФИО" name="fio" value="Василий">
5   </div>
6   <div>
7     <label>E-mail:</label>
8     <input type="text" placeholder="E-mail" name="e-mail">
9   </div>
10  <!-- еще поля -->
11  <button name="register-button" type="submit">Зарегистрироваться</button>
12</form>
```

## СБРОС ФОРМЫ `reset`

Дальше мы поменяем имя и e-mail. После наступления события `reset` поле `e-mail` будет очищено, а вот поле `fio` будет сброшено и будет содержать имя Василий. Т.е. сброс — это именно возврат к исходному состоянию, а не просто очистка.

Событие `reset` возникает в следующих случаях:

- Нажатие на кнопку `<input type="reset">` или `<button type="reset"></button>`
- Вызов метода `reset` у элемента формы.

## СБРОС ФОРМЫ `reset`

Нам как раз подходит второй вариант. Воспользуемся встроенным методом `reset` формы:

```
1 document.forms['register-form'].addEventListener('submit', event => {  
2   event.preventDefault();  
3   // обработка полей формы, передача на сервер при необходимости  
4   const form = event.currentTarget;  
5   form.reset();  
6 });
```



# ВАЛИДАЦИЯ ПОЛЕЙ ФОРМ ПРИ ПОМОЩИ `checkValidity`

# СТАНДАРТНАЯ ВАЛИДАЦИЯ ПОЛЕЙ

## `checkValidity`

С помощью метода `checkValidity`, который является частью JavaScript Validation API, можно проверять текстовые поля на соответствия ограничениям и выводить соответствующее сообщение.

Ограничения могут быть следующими:

- `minlength`, минимальная длина значения в символах;
- `maxlength`, максимальная длина значения в символах;
- `min`, минимальное значение для `input type="number"`;
- `max`, максимальное значение для `input type="number"`;
- `required`, является ли поле обязательным для заполнения;
- `pattern`, проверка значения на соответствие регулярному выражению.

Подробнее с Validation API предлагаем ознакомиться самостоятельно, см. ссылку из доп. материалов 4.



# СТАНДАРТНАЯ ВАЛИДАЦИЯ ПОЛЕЙ

## checkValidity

Это пригодится при выполнении домашнего задания.

Рассмотрим пример, в котором поставим минимальную и максимальную длину для поля E-mail (5-30 символов), провалидируем введенное значение после нажатия на кнопку «Проверить» и выведем соответствующее сообщение:

```
1 <input id="email" minlength="5" maxlength="30">
2 <button onclick="validate()">Проверить</button>
3 <p id="demo"></p>
```

```
1 const validate = () => {
2   let txt = "";
3   if (!document.getElementById("email").checkValidity()) {
4     txt = "E-mail должен быть длиной от 5 до 30 символов";
5   } else {
6     txt = "Валидация прошла успешно";
7   }
8   document.getElementById("demo").innerHTML = txt;
9 }
```

# ЧТО НАДО ДОБАВИТЬ ДЛЯ ВАЛИДАЦИИ?

Что надо дописать в стр. 1 в HTML-коде для валидации поля `numberOfProducts`, чтобы валидировалось количество товаров (должно быть от 3 до 6)?

```
1 <input type="number" id="numberOfProducts" <!--????--> >
2 <button onclick="validate()">OK</button>
3 <p id="demo"></p>
```

```
1 const validate = () => {
2   let txt = "";
3   if (!document.getElementById("numberOfProducts").checkValidity()) {
4     txt = "Для участия в акции количество товаров должно быть от 3 до 6";
5   } else {
6     txt = "Валидация прошла успешно";
7   }
8   document.getElementById("demo").innerHTML = txt;
9 }
```

# ЧТО НАДО ДОБАВИТЬ ДЛЯ ВАЛИДАЦИИ?

Ответ: `min="3" max="6"`

Обратите внимание, что в данном случае `input` имеет тип `type="number"` (ввод только цифр).

## ИТОГО: РАБОТА С ФОРМОЙ

- для сабмита формы используется событие `submit`;
- для сброса формы к начальному состоянию используется событие `reset`;
- с помощью метода `checkValidity`, который является частью JavaScript Validation API, можно проверять текстовые поля на соответствия ограничениям.

# КРАТКИЕ ИТОГИ ЛЕКЦИИ

Итак, сегодня мы узнали, что:

- основными элементами форм являются:
  - `input text`;
  - `checkbox`;
  - `select`;
  - `textarea`;
  - `radio`;
  - `output`;
- основными событиями полей форм являются `input`, `change`, `focus`, `blur`;
- событие формы `submit` используется при отправке формы на сервер, `reset` - для сброса формы к начальному состоянию;
- к форме на странице можно обращаться с помощью `document.forms`;
- производить валидацию формы можно при помощи `checkValidity`.

# ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ

1. <https://developers.google.com/web/updates/2017/03/dialogs-policy>
2. <https://learn.javascript.ru/form-elements>
3. <https://developer.mozilla.org/en-US/docs/Web/CSS/:indeterminate>
4. [https://developer.mozilla.org/ru/docs/Learn/HTML/Forms/Валидация\\_формы](https://developer.mozilla.org/ru/docs/Learn/HTML/Forms/Валидация_формы)

# МАТЕРИАЛЫ, ИСПОЛЬЗОВАННЫЕ ПРИ ПОДГОТОВКЕ

- <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input>
- <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form>
- <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/select>
- <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/textarea>
- <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/checkbox>
- <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/radio>
- [https://www.w3schools.com/js/js\\_validation\\_api.asp](https://www.w3schools.com/js/js_validation_api.asp)



# ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаем в Slack!
- Работы должны соответствовать принятому [стилю оформления кода](#).
- Зачет по домашней работе проставляется после того, как приняты все **3 задачи**.





**Задавайте вопросы и напишите отзыв о лекции!**

**МИХАИЛ КУЗНЕЦОВ**

