



Урок 3

HTML5 Продвинутые веб-формы. Аудио и видео.

Узнаем, какие элементы добавлены, в чём его отличие от предыдущих версий. Рассмотрим и подробно разберём на практических примерах данные особенности.

Усовершенствование традиционных форм (добавление подсказок, фокусировка на элементе и т.д.). Проверка форм на ошибки заполнения. Новые типы элементов (адреса электронной почты, URL-адреса и т.д.).

Рассмотрим и реализуем на практике новые возможности HTML5. Увидим, как работают новые элементы форм и протестируем их в работе.

Добавим в наш проект видео и аудио информацию, посмотрим как все это реализуется на практике, взвесим все плюсы и минусы нового подхода к воспроизведению аудио-видео информации. Рассмотрим работу тегов `<VIDEO>` `<AUDIO>` `<TRACK>`

[Введение](#)

[HTML форма. Базовые понятия.](#)

[Модернизация стандартной HTML-формы](#)

[Добавление подсказок](#)

[Фокусировка на элементе](#)

[Проверка ошибок](#)

[Отключение проверки](#)

[Оформление результатов проверки](#)

[Проверка с помощью регулярных выражений](#)

[Новые типы элементов](#)

[Подсказки ввода <datalist>](#)

[Индикатор выполнения <progress> и счётчик <meter>](#)

[HTML5 Аудио и Видео](#)

[Аудио HTML5](#)

[Воспроизведения видео с помощью элемента <video>](#)

[Элемент <source>](#)

[Элемент <track>](#)

[Атрибут src](#)

[Структура файла .srt](#)

[Атрибут srclang](#)

[Атрибут kind](#)

[Субтитры \(subtitles\)](#)

[Подписи \(captions\)](#)

[Описания \(descriptions\)](#)

[Метаданные \(metadata\)](#)

[Главы \(chapters\)](#)

[Атрибут label](#)

[Атрибут default](#)

[Полезность медиа-дорожек для поисковой оптимизации](#)

[Использование flowplayer \(бонус для тех, кто хочет поменять внешний вид плеера\)](#)

[Программирование видеопроигрывателя своими руками](#)

[Совместимость с предыдущими версиями браузеров](#)

[Совместимость разметки с предыдущими версиями](#)

[Домашнее задание](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Введение

HTML-формы — это простые элементы управления HTML, которые применяются для сбора информации от посетителей веб-сайта. К ним относятся текстовые поля для ввода данных с клавиатуры, списки для выбора predetermined данных, флажки для установки параметров и т. п.

HTML-формы существовали с самых ранних времен языка HTML, и с тех пор они практически не изменились.

Стандарт HTML5 совершенствует уже существующую модель HTML-форм. Это означает, что HTML5-формы могут работать и на старых браузерах, только без нововведений. HTML5-формы также позволяют применять новые возможности, которые уже используются разработчиками в настоящее время. Эти возможности более доступны, не требуют написания страниц сценариев JavaScript или применения инструментариев JavaScript сторонних разработчиков.

На этом уроке мы рассмотрим все новые возможности HTML5-форм.

HTML форма. Базовые понятия.

Веб-форма — это набор текстовых полей, списков, кнопок и других активизируемых щелчком мыши элементов управления, посредством которых посетитель страницы может предоставить ей тот или иной вид информации.

Все основные веб-формы работают одинаково. Пользователь вводит определённую информацию, а потом нажимает кнопку, чтобы отправить введенную информацию на веб-сервер. По прибытию на веб-сервер эта информация обрабатывается каким-либо приложением, которое потом предпринимает соответствующий очередной шаг.

Модернизация стандартной HTML-формы

Элемент `<form>` удерживает вместе все элементы управления формы, которые также называются полями. Кроме этого, он также указывает браузеру, куда отправить данные после нажатия пользователем кнопки отправки, предоставляя URL в атрибуте `action`. Но если вся работа будет выполняться на стороне клиента сценариями JavaScript, то для атрибута `action` можно просто указать значение `#`.

Форма разделяется на логические фрагменты с помощью элемента `<fieldset>`. Каждому разделу можно присвоить название, для чего используется элемент `<legend>`. В листинге 1 приводится разметка формы:

```

<form action="#">
  <p><i>Пожалуйста, заполните форму регистрации пользователей. Обязательные поля
помечены </i><em>*</em></p>
  <fieldset>
    <legend>Контактная информация</legend>
    <label for="name">Фамилия <em>*</em></label>
    <input id="name"><br>
    <label for="name">Имя <em>*</em></label>
    <input id="name"><br>
    <label for="name">Отчество</label>
    <input id="name"><br>
    <label for="telephone">Телефон <em>*</em></label>
    <input id="telephone"><br>
    <label for="email">Email <em>*</em></label>
    <input id="email"><br>
  </fieldset>
  <fieldset>
    <legend>Персональная информация</legend>
    <label for="age">Возраст<em>*</em></label>
    <input id="age"><br>
    <label for="gender">Пол</label>
    <select id="gender">
      <option value="female">Мужской</option>
      <option value="male">Женский</option>
    </select><br>
    <label for="comments">Перечислите дополнительную информацию, которую вы хотели бы
сообщить о себе</label>
    <textarea id="comments"></textarea>
  </fieldset>
  <fieldset>
    <legend>Выберите ваши увлечения</legend>
    <label for="sport"><input id="sport" type="checkbox"> Спорт</label>
    <label for="tunist"><input id="tunist" type="checkbox"> Туризм</label>
    <label for="padi"><input id="padi" type="checkbox"> Дайвинг</label>
    <label for="travels"><input id="travels" type="checkbox"> Путешествия</label>
    <label for="auto"><input id="auto" type="checkbox"> Автомобили</label>
    <label for="it"><input id="it" type="checkbox"> IT</label>
  </fieldset>
  <p><input type="submit" value="Отправить информацию"></p>
</form>

```

Файл forms.css находится в приложении к занятию.

Большая часть работы в нашем примере выполняется универсальным элементом **<input>**, который собирает данные и создает флажки, переключатели и списки. Для ввода одной строки текста применяется элемент **<input>**, а для нескольких — элемент **<textarea>**; элемент **<select>** создает выпадающий список. Краткое обозрение этих и других элементов управления форм приведено в таблице:

Элементы управления формы:

Элемент управления	HTML-элемент	Описание
Однострочное текстовое поле	<code><input type="text"></code> <code><input type="password"></code>	Выводит однострочное текстовое поле для ввода текста. Если для атрибута <code>type</code> указано значение <code>password</code> , вводимые пользователем символы отображаются в виде звездочек (*) или маркеров-точек (•)
Многострочное текстовое поле	<code><textarea>...</textarea></code>	Текстовое поле, в которое можно ввести несколько строчек текста
Флажок	<code><input type="checkbox"></code>	Выводит поле флажка, который можно установить или очистить
Переключатель	<code><input type="radio"></code>	Выводит переключатель — элемент управления в виде небольшого кружка, который можно включить или выключить. Обычно создается группа переключателей с одинаковым значением атрибута <code>name</code> , вследствие чего можно выбрать только один из них
Кнопка	<code><input type="submit"></code> <code><input type="image"></code> <code><input type="reset"></code> <code><input type="button"></code>	Выводит стандартную кнопку, активируемую щелчком мыши. Кнопка типа <code>submit</code> всегда собирает информацию с формы и отправляет ее для обработки. Кнопка типа <code>image</code> делает то же самое, но позволяет вместо текста на кнопке вставить изображение. Кнопка типа <code>reset</code> очищает поля формы от введенных пользователем данных. А кнопка типа <code>button</code> сама по себе не делает ничего. Чтобы ее нажатие выполняло какое-либо действие, для нее нужно добавить сценарий JavaScript
Список	<code><select>...</select></code>	Выводит список, из которого пользователь может выбирать значения. Для каждого значения списка добавляем элемент <code><option></code>

Теперь, когда есть форма, с которой можно работать, настало время улучшить её с помощью HTML5.

Добавление подсказок

Обычно поля новой формы не содержат никаких данных. Для некоторых пользователей такая форма может быть не совсем понятной, в частности, какую именно информацию нужно вводить в конкретное поле. Поэтому часто поля формы содержат пример данных, которые нужно ввести в них. Этот подстановочный текст также называется "водяным знаком", так он часто отображается шрифтом светло-серого цвета, чтобы отличить его от настоящего, введенного содержимого.

Подстановочный текст для поля создается с помощью атрибута `placeholder`

```
<fieldset>
  <legend>Контактная информация</legend>
  <label for="name">Фамилия <em>*</em></label>
  <input id="name" placeholder="Иванов"><br>
  <label for="name">Имя <em>*</em></label>
  <input id="name" placeholder="Иван"><br>
  <label for="name">Отчество</label>
  <input id="name" placeholder="Сидорович"><br>
</fieldset>
```

Фокусировка на элементе

Вводить информацию в форму пользователи не смогут до тех пор, пока не щелкнут мышью по необходимому полю или выделят его с помощью клавиши `<Tab>`, установив, таким образом, *фокус* на этом поле.

Возможно установить фокус на нужном начальном поле автоматически. На этой идее основан новый HTML5-атрибут `autofocus`, который можно вставить в элемент `<input>`

```
<label for="name">Фамилия <em>*</em></label>
<input id="name" placeholder="Иванов" autofocus><br>
```

Проверка ошибок

Поля в форме предназначены для сбора информации от посетителей страницы.

Серьёзной веб-странице требуется проверка данных, т. е. какой-либо способ обнаружения ошибок во введённых данных, а ещё лучше — способ, не допускающий ошибок вообще. На протяжении многих лет веб-разработчики делали это с помощью процедур JavaScript собственной разработки или посредством профессиональных библиотек JavaScript.

Создатели HTML5 разработали систему проверки на *стороне клиента*

Допустим, что определённое поле нельзя оставлять пустым, и посетитель должен ввести в него хоть что-то. В HTML5 это осуществляется с помощью атрибута `required` в соответствующем поле.

Когда пользователь нажмет кнопку для отправки формы, браузер начнёт проверять поля сверху вниз. Встретив первое некорректное значение, он прекращает дальнейшую проверку, отменяет отправку формы и выводит сообщение об ошибке рядом с полем, вызвавшим эту ошибку. (Кроме этого, если при заполнении формы область с полем ошибки вышла за пределы экрана, браузер прокручивает экран, чтобы это поле находилось вверху страницы.) После того как пользователь исправит данную

ошибку и опять нажмёт кнопку для отправки формы, браузер остановится на следующей ошибке ввода.

Отключение проверки

В некоторых случаях может потребоваться отключить проверку. Например, вы хотите протестировать свой серверный код на правильность обработки поступивших некорректных данных. Чтобы отключить проверку для всей формы, в элемент `<form>` добавляется атрибут `novalidate`

Оформление результатов проверки

Хотя веб-разработчики не могут оформлять сообщения об ошибках проверки, они могут изменять внешний вид полей в зависимости от *результатов* их валидации. Например, можно выделить поле с неправильным значением цветным фоном сразу же, когда браузер обнаружит неправильные данные. Для этого требуется добавить несколько простых псевдоклассов

Доступны следующие опции:

Всё, что для этого требуется — это добавить несколько простых CSS3-псевдоклассов. Доступны следующие опции:

- `required` и `optional` Применяют форматирование к полю в зависимости от того, использует ли это поле атрибут `required` или нет;
- `valid` и `invalid` Применяют форматирование к полю в зависимости от правильности введенного в него значения. Но не забывайте, что большинство браузеров не проверяет данные, пока пользователь не попытается отправить форму, поэтому форматирование полей с некорректными значениями не выполняется сразу же при введении такого значения;
- `in-range` и `out-of-range` Форматирование к полям, для которых используется атрибут `min` или `max`, чтобы ограничить их значение определенным диапазоном значений.

```
form input:required
{
    background-color: lightyellow;
}
```

Проверка с помощью регулярных выражений

Самым мощным (и самым сложным) поддерживаемым HTML5 типом проверки является проверка на основе регулярных выражений.

Регулярное выражение — это шаблон для сопоставления с образцом, закодированный согласно определённым синтаксическим правилам. Регулярные выражения применяются для поиска в тексте строк, которые отвечают определённому шаблону. Например, с помощью регулярного выражения можно проверить, что почтовый индекс содержит правильное число цифр, или в адресе электронной почты присутствует знак `@`, а его доменное расширение содержит, по крайней мере, два символа. Возьмем, например, следующее выражение:

```
[A-Z]{3}-[0-9]{3}
```

Квадратные скобки в начале строки определяют диапазон допустимых символов. Иными словами, группа `[A-Z]` разрешает любые прописные буквы от A до Z. Следующая за ней часть в фигурных скобках указывает множитель, т.е. `{3}` означает, что нужны три прописные буквы. Следующее тире не

имеет никакого специального значения и означает само себя, т.е. указывает, что после трёх прописных букв должно быть тире. Наконец, [0-9] означают цифры в диапазоне от 0 до 9, а {3} требует три таких цифры.

Регулярные выражения полезны для поиска в тексте строк, отвечающих условиям, заданных в выражении, и проверки, что определённая строка отвечает заданному регулярным выражением шаблону. В формах HTML5 регулярные выражения применяются для валидации.

Для обозначения начала и конца значения в поле символы ^ и \$, соответственно, не требуются. HTML5 автоматически предполагает наличие этих двух символов. Это означает, что значение в поле должно полностью совпасть с регулярным выражением, чтобы его можно было считать корректным.

Таким образом следующие значения будут допустимыми для этого регулярного выражения:

- QDR-001
- WES-205
- LOG-104

А вот эти нет:

- qdr-001
- TTT-0259
- 5SD-000

Но регулярные выражения очень быстро становятся более сложными, чем рассмотренный нами пример. Поэтому создание правильного регулярного выражения может быть довольно трудоёмкой задачей, что объясняет, почему большинство разработчиков предпочитает использовать для проверки данных на своих страницах готовые регулярные выражения.

Чтобы применить полученное тем или иным путем регулярное выражение для проверки значения поля `<input>` или `<textarea>`, его следует добавить в этот элемент в качестве значения атрибута `pattern`:

Новые типы элементов

В HTML5 в элемент `<input>` было добавлено несколько новых типов, и если какой-либо браузер не поддерживает их, он будет обрабатывать их как обычные текстовые поля.

В таблице 1 приведены основные типы

Тип данных	Назначение
image	создает кнопку, позволяя вместо текста на кнопке вставить изображение.
email	Используется для полей, предназначенных для ввода адресов электронной почты.
URL	применяется для полей ввода URL-адресов.
tel	применяется с целью обозначения полей для ввода телефонных номеров, которые могут быть представлены в самых разных форматах. HTML5 не требует от браузеров выполнения проверки телефонных номеров.
number	при вводе данных в поле типа number браузер автоматически игнорирует все символы, кроме цифр. Можно использовать атрибуты min, max и step атрибут step, который указывает шаг изменения числа (в большую или меньшую сторону). Например, установив значение step в 0.1, можно вводить такие значения, как 0,1, 0,2 0,3 и т. д. По умолчанию значение шага равно 1.
range	Подобно типу number, этот тип может представлять целые и дробные значения. Также поддерживает атрибуты min и max для установки диапазона значений. Разница состоит в том, каким образом поле типа range представляет свою информацию. Вместо счетчика, как для поля типа number, интеллектуальные браузеры отображают ползунок
date	Дата по шаблону ГГГГ-ММ-ДД
time	Время в 24-часовом формате с необязательными секундами, по шаблону чч:мм:сс.сс
datetime-local	Дата и время, разделенные прописной английской буквой Т (по шаблону ГГГГ-ММ-ДДТчч:мм:сс)
datetime	Дата и время, как и для типа datetimelocal, но со смещением временного пояса. Используется такой же шаблон (ГГГГ-ММ-ДДТчч:мм:сс-чч: мм), как и для элемента <time>
month	Год и номер месяца по шаблону ГГГГ-ММ
week	Год и номер недели по шаблону ГГГГ-Изнн. Обратите внимание, что в зависимости от года может быть 52 или 53 недели
color	применяется для полей, предназначенных для ввода цвета. Тип данных color — это интересная, хотя редко используемая второстепенная возможность, позволяющая посетителю веб-страницы выбирать цвет из выпадающей палитры, похожей на палитру графического редактора MS Paint.

В стандарте также вводится несколько совершенно новых элементов. С их помощью в форму можно добавить список предложений, индикатор выполнения задания, панель инструментов

Подсказки ввода <datalist>

Элемент <datalist> предоставляет способ присоединить выпадающий список возможных вариантов ввода к обыкновенному текстовому полю. Заполняющим форму пользователям он даёт возможность либо выбрать вариант ввода из списка значений, либо ввести требуемое значение вручную.

Чтобы использовать элемент datalist, сначала нужно создать обычное текстовое поле. Допустим, мы создали обычный элемент <input>:

```
<fieldset>
  <legend>Выберите любимый фрукт</legend>
  <label for="fruit">Фрукт</label>
  <input id="fruit" list="dl_fruits">
</fieldset>
```

Чтобы добавить к этому полю выпадающий список возможных значений, для него нужно создать элемент `<datalist>`. Технически этот элемент можно разместить в любом месте разметки, т.к. он не отображается браузером, а просто предоставляет данные для использования в текстовом поле ввода. Но логично поместить этот элемент либо непосредственно перед тем элементом `<input>`, для которого он предоставляет свои данные, либо сразу же после него. Далее показан пример кода для создания списка `<datalist>`:

Как и традиционное поле `<select>`, список `<datalist>` использует элементы `<option>`. Каждый элемент `<option>` представляет собой отдельное возможное значение, которое браузер может предложить заполняющему форму. Значение атрибута `label` содержит текст, который отображается в текстовом поле, а атрибут `value` — текст, который будет отправлен на сервер, если пользователь выберет данную опцию.

Сам по себе список `<datalist>` не отображается в браузере. Для того чтобы подключить его к текстовому полю, нужно установить значение атрибута `list` равным значению параметра `id` соответствующего списка `<datalist>`:

```
<datalist id="dl_fruits">
  <option label="Яблоко" value="apple">
  <option label="Ананас" value="anas">
  <option label="Абрикос" value="apricot">
  <option label="Арбуз" value="watermelon">
  <option label="Авокадо" value="avocado">
  <option label="Апельсин" value="Orange">
</datalist>
```

В браузерах, которые поддерживают `<datalist>`, посетители увидят результат, другие браузеры будут игнорировать атрибут `list` и разметку `<datalist>`, делая все предложения возможного ввода бесполезными.

Индикатор выполнения `<progress>` и счётчик `<meter>`

Новые графические элементы `<progress>` и `<meter>` внешне похожи друг на друга, но имеют разные назначения.

Элемент `<progress>` отображает ход выполнения задания посредством зеленой пульсирующей полосы на сером фоне.

Элемент `<meter>` указывает значение в диапазоне известных значений. Внешне он похож на элемент `<progress>`, но зеленая полоска имеет другой оттенок и не пульсирует.

Элемент `<progress>` использует атрибут `value`, который обозначает ход выполнения задания в виде дробной величины от 0 до 1. Графически это отображается соответствующей шириной полосы

индикатора. Например, чтобы показать, что задание выполнено на 25%, атрибуту value присваивается значение 0,25:

```
<progress value="0.25"></progress>
```

Альтернативно можно использовать атрибут max, чтобы установить максимальное значение и изменить масштаб индикатора. Например, при значении max, равном 200, значение value должно быть между 0 и 200. Если сделать значение value равным 50, то получим те же самые 25% заполнения индикатора, как и в предыдущем примере:

```
<progress value="50" max="200"></progress>
```

Элемент <meter> имеет подобную модель, но отображает любой вид измерений. Иногда его ещё называют шкалой. Часто значение атрибута value этого элемента отображает какую-то действительную величину, например, денежную сумму на счету, количество дней, вес в килограммах и т. п. Отображение этой информации управляется установкой значений атрибутов min и max:

```
Пройденная дистанция: <meter min="5" max="70" value="28"> 28 метров</meter>
```

После всех модификаций листинг вывода формы страницы выглядит следующим образом:

```

<form action="#">
  <p><i>Пожалуйста, заполните форму регистрации пользователей. Обязательные поля
помечены </i><em>*</em></p>
  <fieldset>
    <legend>Контактная информация</legend>
    <label for="name">Фамилия <em>*</em></label>
    <input id="name" placeholder="Иванов" autofocus required pattern="\S+[А-яа-я]"><br>
    <label for="name">Имя <em>*</em></label>
    <input id="name" placeholder="Иван" required pattern="\S+[А-яа-я]"><br>
    <label for="name">Отчество</label>
    <input id="name" placeholder="Сидорович" pattern="\S+[А-яа-я]"><br>
    <label for="telephone">Телефон <em>*</em></label>
    <input id="telephone" placeholder="+7-903-955-55-55" required"><br>
    <label for="email">Email <em>*</em></label>
    <input id="email" placeholder="email@server.com" required pattern="\S+@[a-z]+\.[a-z]+"><br>
  </fieldset>
  <fieldset>
    <legend>Персональная информация</legend>
    <label for="age">Возраст<em>*</em></label> <input id="age" type="number" required min="14"
max="100"><br>
    <label for="gender">Пол</label>
    <select id="gender">
      <option value="female">Мужской</option>
      <option value="male">Женский</option>
    </select><br>
    <label for="comments">Перечислите дополнительную информацию, которую вы хотели бы
сообщить о себе</label>
    <textarea id="comments"></textarea>
  </fieldset>
  <fieldset>
    <legend>Выберите ваши увлечения</legend>
    <label for="sport"><input id="sport" type="checkbox"> Спорт</label>
    <label for="turist"><input id="turist" type="checkbox"> Туризм</label>
    <label for="padi"><input id="padi" type="checkbox"> Дайвинг</label>
    <label for="travels"><input id="travels" type="checkbox"> Путешествия</label>
    <label for="auto"><input id="auto" type="checkbox"> Автомобили</label>
    <label for="it"><input id="it" type="checkbox"> IT</label>
  </fieldset>
  <p><input type="submit" value="Отправить информацию"></p>
</form>

```

Полный листинг кода находится в приложении к занятию.

HTML5 Audio и Видео

Аудио HTML5

В следующем коде приведён пример использования элемента <audio>:

```

<p>Ознакомительный фрагмент аудио. :</p>
<audio src="audio.mp3" controls></audio>

```

Атрибут `src` содержит имя аудиофайла для воспроизведения, а атрибут `controls` указывает браузеру, что нужно отобразить базовые элементы управления воспроизведением. Своим внешним видом эти элементы управления слегка отличаются от браузера к браузеру, но все они имеют одинаковое назначение: разрешают пользователю начинать и останавливать воспроизведение, переходить в другое место записи и регулировать громкость (рисунок 1)

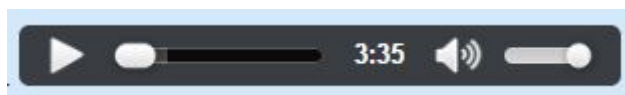


Рисунок 1 – Аудио проигрыватель

Кроме атрибута `controls` элемент `<audio>` поддерживает ещё три атрибута: `preload`, `autoplay` и `loop`. Атрибут `preload` указывает браузеру способ загрузки аудиофайла. Значение `auto` этого атрибута указывает браузеру загружать аудио-файл полностью, чтобы он был доступен, когда пользователь нажмет кнопку воспроизведения.

Загрузка осуществляется в фоновом режиме, чтобы посетитель веб-страницы мог перемещаться по странице и просматривать её, не дожидаясь завершения загрузки аудиофайла.

Атрибут `preload` может принимать два значения. Значение `metadata` указывает браузеру загрузить первую небольшую часть файла, достаточную, чтобы определить некоторые его основные характеристики (например, общий размер файла). Атрибут `none` указывает браузеру не загружать аудио-файл автоматически. Эти опции можно использовать для того, чтобы сэкономить пропускную способность подключения.

```
<audio src="rubberduckies.mp3" controls preload="metadata"></audio>
```

Когда атрибуту `preload` задано значение `none` или `metadata`, загрузка аудиофайла начинается после того, как пользователь нажмет кнопку воспроизведения.

Если значение атрибута `preload` не установлено, то браузеры действуют по своему индивидуальному усмотрению. Большинство браузеров предполагает `auto` в качестве значения по умолчанию. Кроме этого, важно отметить, что атрибут `preload` не является обязательным для выполнения правилом, а рекомендацией браузеру желаемого действия, которую он может игнорировать в зависимости от других обстоятельств.

Атрибут `autoplay` указывает браузеру начать воспроизведение сразу же после завершения загрузки страницы:

```
<audio src="audio.mp3" controls autoplay></audio>
```

Если этот атрибут не используется, пользователь должен нажать кнопку запуска, чтобы начать воспроизведение. Элемент `<audio>` можно использовать для того, чтобы проигрывать фоновую музыку.

Атрибут `loop` указывает браузеру повторять воспроизведение:

```
<audio src="audio.mp3" controls loop></audio>
```

Воспроизведения видео с помощью элемента <video>

С элементом <audio> хорошо идёт в паре элемент <video>. Он применяет такие же атрибуты src, controls, autoplay и loop. Пример использования этого элемента показан в следующем коде:

```
<p>Демонстрационное видео!</p>
<video src="video.mp4" controls></video>
```

Как и в случае с элементом <audio>, атрибут controls указывает браузеру создать набор элементов управления воспроизведением (рисунок 2). В большинстве браузеров эти элементы скрываются при щелчке где-нибудь в области страницы и отображаются опять при наведении курсора мыши на область видеоплеера.



Рисунок 2 - видеопроигрыватель

Кроме общих с элементом <audio> атрибутов, элемент <video> имеет три своих собственных атрибута: height, width и poster.

Атрибуты height и width устанавливают высоту и ширину окна воспроизведения в пикселах, соответственно. Следующий код показывает пример создания области воспроизведения размером 400 300 пикселей:

```
<video src="video.mp4" controls width="400" height="300"></video>
```

Размеры окна воспроизведения должны совпадать с размером видео, но лучше явно указать их, чтобы оформление страницы не искажалось до того, как видео-файл загрузится (или если видеофайл вовсе не загружается).

Атрибут poster позволяет указать изображение, которое можно использовать вместо видео:

```
<video src="video.mp4" controls poster="pict.jpg"></video>
```

Браузеры показывают это изображение в трёх ситуациях: когда первый кадр видео ещё не загрузился, атрибуту preload присвоено значение none или указанный видеофайл отсутствует.

Элемент <source>

Для совместимости с различными браузерами у нас есть не один видеофайл, а три. Один это OGV-файл, второй MP4- файл, третий WebM-файл. HTML5 обеспечивает способ сделать ссылки на все три файла с помощью элемента <source>. Каждый элемент <video> может содержать более одного тега <source>. Ваш браузер пройдётся по списку источников видео по порядку и выберет первым то, что он в состоянии воспроизвести.

Браузер проводит проверку, какое видео он сможет воспроизвести. В худшем случае он загружает каждое видео и пытается его сыграть, однако это большая трата пропускной способности. Вы сэкономите много трафика, если сообщите браузеру информацию о каждом видео. Это можно сделать атрибутом type тега <source>.

```
<video width="320" height="240" controls>
<source src="video.mp4" type="video/mp4; codecs="avc1.42E01E, mp4a.40.2">
<source src="video.webm" type="video/webm; codecs="vp8, vorbis">
<source src="video.ogv" type="video/ogg; codecs="theora, vorbis">
</video>
```

Элемент <video> определяет ширину и высоту видео, но не ссылку на видеофайл. Внутри <video> три элемента <source>. Каждый элемент <source> ссылается на отдельный видеофайл (с атрибутом src), а также даёт информацию о видеоформате (в атрибуте type).

Тип атрибута - комбинация из трех блоков информации: формат файла, видеокодек и аудиокодек. Для видеофайла .ogv формат контейнера это Ogg, представленный здесь как video/ogg, строго говоря, это MIME-тип для видео-файлов Ogg. Видео-кодек Theora и аудио-кодек Vorbis. Само значение должно быть заключено в кавычки, поэтому вы должны использовать различные виды кавычек, чтобы окружить значение целиком.

```
<source src="pr6.ogv" type="video/ogg; codecs="theora, vorbis">
```

С WebM почти так же, но с другим MIME-типом (video/webm вместо video/ogg) и другим видео-кодеком (vp8 вместо theora) написанным в параметре codecs.

```
<source src="pr6.webm" type="video/webm; codecs="vp8, vorbis">
```

Видео H.264 является более сложным.

```
<source src="pr6.mp4" type="video/mp4; codecs="avc1.42E01E, mp4a.40.2">
```

Преимуществом является то, что браузер проверяет атрибут type первым и смотрит, может ли он воспроизвести видеофайл. Если браузер решает, что он не может этого сделать, то не будет скачивать файл даже частично.

Элемент <track>

Элемент <track> задаёт любой текст, который бы вы хотели показывать во время проигрывания медиафайлов. Текст может включать субтитры, подписи, описания, главы или метаданные.

Элемент <track> позволяет указать дополнительные ресурсы с текстом, который будет отображаться синхронно с временной шкалой вашего видео или аудио файла.

<track> - у него нет парного закрывающего тега (то есть, это пустой элемент). Он должен располагаться внутри тега <video> или <audio>. К тому же, если внутри аудио или видео тегов присутствует элемент <source>, то элемент <track> должен располагаться после него.

```
<video width="640" height="320" controls>
<source src="some_video.mp4" type="video/mp4">
<source src="some_video.ogg" type="video/ogg">
<track src="some_video_subtitles.srt"
      kind="subtitles"
      srclang="en"
      label="English_subs">
</video>
```

Атрибут src

Этот атрибут определяет адрес текстового файла, содержащего данные дорожки, и он является обязательным. Его значением должен быть абсолютный или относительный URL. Это означает, что файлы должны быть расположены на сервере.

Например:

```
<track src="video_captions.srt">
```

Структура файла .srt

Атрибут элемента отслеживания src указывает на текстовый файл, содержащий данные о хронометрированных метках отслеживания в любом формате, который распознается браузером. Chrome поддерживает формат WebVTT, который имеет описанный ниже вид.

```
WEBVTT FILE
railroad
00:00:10.000 --> 00:00:12.500
Left uninspired by the crust of railroad earth
manuscript
00:00:13.200 --> 00:00:16.900
that touched the lead to the pages of your manuscript.
```

Атрибут srclang

Атрибут srclang указывает язык прикрепленного текста. Этот атрибут необходимо указывать, если значение атрибута kind выставлено в subtitles. Значение атрибута srclang должно быть валидным языковым тегом в формате BCP47. Например значение en используется для Английского.

```
<track src="video_subtitles.srt" kind="subtitles" srclang="en">
```

Этот пример указывает, что файл субтитров представлен для английского языка.

Атрибут kind

Этот атрибут указывает тип дорожки, которую мы добавляем к медиа-файлу. Он может принимать одно значение из некоторого набора:

- Subtitles;
- Captions;
- Description;
- Metadata;
- Chapters

Субтитры (subtitles)

Обычно представляет из себя перевод диалогов, которые звучат в видео или аудио файле. Они бывают полезны, когда пользователь не понимает языка оригинала, но у него есть возможность читать перевод диалогов. Указание языка источника обязательно. Это делается путем указания соответствующего значения атрибуту srclang:

```
<track src="video_subtitles.srt" kind="subtitles" srclang="en">
```

Подписи (captions)

Подпись - краткое описание, сопровождающее видео во время проигрывания. Бывает полезно в ситуациях, когда пользователю нужно сообщить важную информацию, или если звук плохо различим или не слышен. Простой пример:

```
<track src="video_captions.srt" kind="captions">
```

Описания (descriptions)

Как понятно из наименования, данный тип трека используется для описания медиаконтента. Синхронные дорожки, помеченные как описания, обычно представляют собой отдельную аудиодорожку. Например:

```
<track src="video_descriptions.srt" kind="descriptions">
```

Метаданные (metadata)

Предназначен для треков, которые предоставляют метаданные. Они не отображаются браузерами. Значения метаданных предназначены для использования скриптами, вроде JavaScript. Например:

```
<track src="video_metadata.srt" kind="metadata">
```

Главы (chapters)

Представляет собой заголовок, предназначен для использования в навигации по медиа-ресурсу. Дорожки, обозначенные как главы, обычно отображаются пользователю в виде интерактивного списка.

```
<track src="video_chapters.srt" kind="chapters">
```

Атрибут label

Атрибут предназначен для указания названия текстовой дорожки, подключенной к аудио/видео файлу. Используется браузерами при отображении пользователю списка доступных дорожек, и представляет собой название дорожки в удобном пользователю виде.

Если вы добавите атрибут label элементу <track>, его значение нельзя оставлять пустым. Значение должно быть строковым параметром. Если атрибут отсутствует, браузер подставит значение по-умолчанию, вроде “без названия”.

```
<track src="video_subtitles.srt"
      kind="subtitles"
      srclang="en"
      label="Английские субтитры">
```

В этом примере можно видеть, что атрибут label содержит значение “Английские субтитры”

Атрибут default

Используется для указания дорожки, которая будет выбрана по-умолчанию. Атрибут default может быть задан только одному элементу <track> для одного медиафайла. Такой элемент будет выбран по-умолчанию, если пользователь не выберет другую дорожку.

Следующий пример показывает, что из предложенных субтитров на разных языках (английский, японский, русский), по-умолчанию будут выбраны и отображены вместе с видео субтитры на русском:

```
<track kind="subtitles" src="video_subtitles_ru.srt" srclang="ru" default>
<track kind="subtitles" src="video_subtitles_en.srt" srclang="en">
<track kind="subtitles" src="video_subtitles_ja.srt" srclang="ja">
```

Теперь, когда мы рассмотрели основные атрибуты, которые могут использоваться с тегом <track>, мы распишем небольшой пример использования нескольких типов дорожек в рамках элемента video и source:

```
<video src="sample.ogv">
  <source src="video.mp4" type="video/mp4">
  <source src="video.ogv" type="video/ogg">
  <track kind="captions" src="video_captions.srt" srclang="en">
  <track kind="descriptions" src="video_descriptions.srt" srclang="en">
  <track kind="chapters" src="video_chapters.srt" srclang="en">
  <track kind="subtitles" src="video_subtitles_en.srt" srclang="en" default>
  <track kind="subtitles" src="video_subtitles_oz.srt" srclang="oz">
  <track kind="metadata" src="video_metadata1.srt" srclang="en" label="Метаданные 1">
  <track kind="metadata" src="video_metadata2.srt" srclang="en" label="Метаданные 2">
</video>
```

Полезность медиа-дорожек для поисковой оптимизации

Элемент `<track>` открыл дорогу к оптимизации под поисковые системы для видео, а также дал возможность использовать более эффективный способ сделать ваши видео более понятными поисковым системам.

Поисковые системы всё лучше понимают различное мультимедиа-содержимое, и чем большим количеством информации вы можете сопроводить свои файлы, тем более таргетированный трафик, большего объема, сможете получить вы.

Некоторые ключевые преимущества поисковой оптимизации:

- Улучшает ваше присутствие в поисковой выдаче: поисковые системы при поиске проходятся по всем текстовым данным, ассоциированным с вашим видео
- Лучшая связность: поисковые системы возвращают результаты поиска, которые указывают на конкретные части видео, ассоциированные с временными кодами
- Связанное содержимое: текстовые файлы могут быть легко включены в связанное текстовое содержимое на той же странице
- Доступность и пользовательское взаимодействие: субтитры и подписи улучшают юзабилити и доступность для людей с инвалидностью
- Миниатюры в результатах поиска: у страницы с видео есть преимущество того, что она может быть отображена в результатах поиска в виде фрагмента с миниатюрой, что может повысить количество переходов по ссылкам

Использование flowplayer (бонус для тех, кто хочет поменять внешний вид плеера)

Установка происходит в несколько шагов.

Предварительно необходимо загрузить с официального сайта (<https://flowplayer.org/latest/>) дистрибутив с проигрывателем.

1. Распаковать дистрибутив проигрывателя в корень сайта.
2. В заголовок страницы вставить строки:

```
<!-- player skin -->
<link rel="stylesheet" href="/flowplayer/skin/functional.css">
<!-- for video tag based installs flowplayer depends on jQuery 1.7.2+ -->
<script src="https://code.jquery.com/jquery-1.11.2.min.js"></script>
<!-- include flowplayer -->
<script src="flowplayer/flowplayer.min.js"></script>
<!-- the player -->
```

А в тело документа, где необходимо отобразить видео, вставить код для воспроизведения:

```
<div class="flowplayer" data-swf="flowplayer/flowplayer.swf" data-ratio="0.4167">
  <video width="400" height="300">
    <source type="video/mp4" src="media/video1.mp4">
  </video>
</div>
```

В результате получается видеопроигрыватель, изображенный на рисунке 3

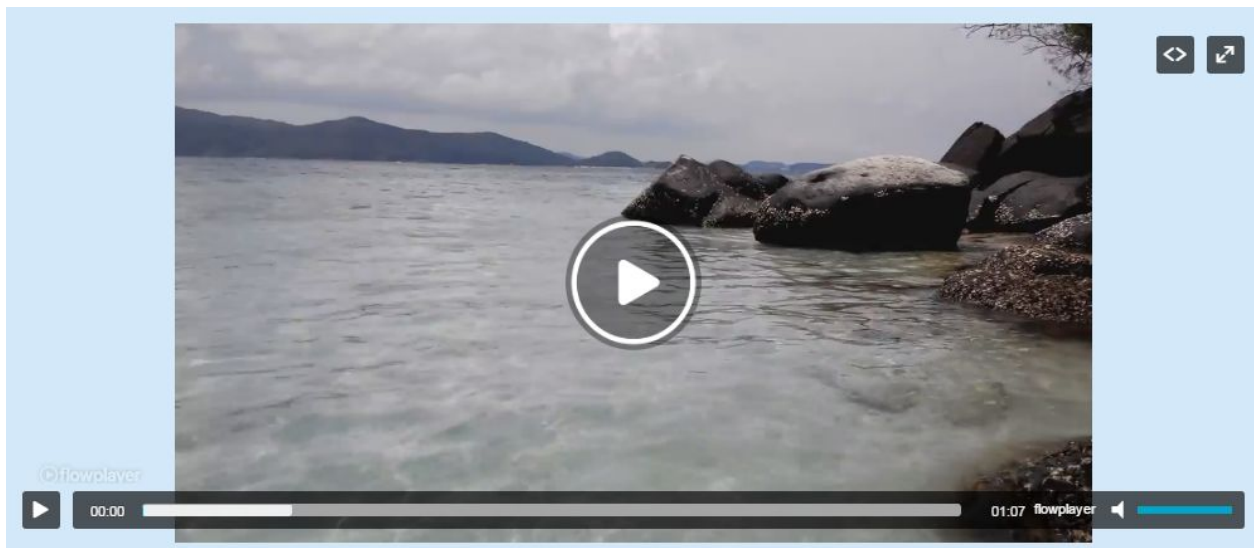


Рисунок 3 – видео-проигрыватель flowplayer

Программирование видеопроигрывателя своими руками

Создадим HTML шаблон для проигрывателя.

Для этого в заголовок страницы вставим строки ссылки на player.css файл и player.js файл. Данные файлы будут соответственно отвечать за внешний вид проигрывателя и его функциональность.

```
<link rel="stylesheet" href="player.css">
<script src="player.js"></script>
```

В тело документа вставим следующий код:

```
<section id="player">
<div class="clear"> </div>
    <video id="media" width="720" height="400">
        <source src="media/video3.mp4">
    </video>
    <div id="nav_panel">
    <div id="buttons">
        <input type="button" id="play" value="Play">
        <input type="button" id="mute" value="Mute">
    </div>
    <div id="bar">
        <div id="progress"> </div>
    </div>
    <div id="control">
        <input type="range" id="volume" min="0" max="1" step="0.1" value="0.6">
    </div>
    <div class="clear"> </div>
</div>
</section>
```

Визуальное оформление содержится в файле "player.css"

```
@charset "utf-8";
/* CSS Document */
#player
{
    text-align: center;
    width: 720px;
    margin: 20px auto;
    padding: 10px 5px 5px 5px;
    background: #999999;
    border: 1px solid #666666;
    border-radius: 10px;
}
#play, #mute
{
    padding: 2px 10px;
    width: 65px;
    border: 1px solid #000000;
    background: #DDDDDD;
    font-weight: bold;
    border-radius: 10px;
}
#nav_panel
{
    margin: 5px 0px;
}
#buttons
{
    float: left;
    width: 135px;
    height: 20px;
    padding-left: 5px;
}
#bar
{
    float: left;
    width: 400px;
    height: 16px;
    padding: 2px;
    margin: 2px 5px;
    border: 1px solid #CCCCCC;
    background: #EEEEEE;
}
#progress
{
    width: 0px;
    height: 16px;
    background: rgba(0,0,150,.2);
}
.clear
{
    clear: both;
}
```

Запрограммируем JavaScript для нашего проигрывателя. Используем несколько функций таких как: остановка, воспроизведение, отображение индикатора, а также переход по навигатору.

Используемые события.

В HTML5 есть события, относящиеся к конкретным API. Для обработки аудио и видео добавлены события, информирующие о состоянии мультимедиа.

Наиболее часто используемые события:

Progress – событие срабатывает периодически и обновляет информацию о состоянии загрузки мультимедиа;

Canplaythrough – событие срабатывает, когда становится известно, что весь файл мультимедиа может быть переведен без перерывов.

Ended – срабатывает, когда мультимедиа воспроизводится до конца.

Pause – срабатывает, когда воспроизведение приостанавливается.

Play – срабатывает при запуске воспроизведения мультимедиа.

Error – срабатывает, когда происходит ошибка.

Разберем файл `player.js`

Функция `initiate()` инициализирует приложение (запускает выполнение после загрузки окна)

В данной функции устанавливаются все переменные для настройки проигрывателя. Используя селектор `getElementById` определяются все ссылки на элементы проигрывателя. Позднее мы сможем к ним обращаться в коде. Переменная `maxim` задает максимальный размер индикатора прогресса.

Обрабатываются два события. Пользователь щелкает на кнопке воспроизведения и на индикаторе процесса.

Для этого прослушаем следующие события.

У элемента `play`, `mute`, `move` прослушаем события `click`. Этот прослушиватель запускает соответствующую функцию, каждый раз, когда возникает данное событие.

`function push()` – запускает и приостанавливает воспроизведение видео

```
function push()
{
    if(!mmedia.paused && !mmedia.ended)
    {
        mmedia.pause();
        play.value = 'Play';
        clearInterval(loop);
    }
    else
    {
        mmedia.play();
        play.value = 'Pause';
        loop = setInterval(status, 1000);
    }
}
```

function status() – обновляет вид индикатора прогресса

```
function status()
{
    if(!mmedia.ended)
    {
        var size = parseInt(mmedia.currentTime * maxim / mmedia.duration);
        progress.style.width = size + 'px';
    }
    else
    {
        progress.style.width = '0px'; play.innerHTML = 'Play';
        clearInterval(loop);
    }
}
```

Function move() – начинает воспроизведение с позиции выбранной пользователем

```
function move(e)
{
    if(!mmedia.paused && !mmedia.ended)
    {
        var mouseX = e.pageX - bar.offsetLeft;
        var newtime = mouseX * mmedia.duration / maxim; mmedia.currentTime = newtime;
        progress.style.width = mouseX + 'px';
    }
}
```

Function sound() – включает и отключает звук воспроизведения

```
function sound()
{
    if(mute.value == 'Mute')
    {
        mmedia.muted = true;
        mute.value = 'Sound';
    }
    else
    {
        mmedia.muted = false;
        mute.value = 'Mute';
    }
}
```

Function level() – устанавливает громкость воспроизведения воспроизведения

```
function level()
{
    mmedia.volume = volume.value;
}
```

Прослушивание события «load»


```
addEventListener('load', initiate);
```

В результате получим проигрыватель (рисунок 4)



Рисунок 4 – Запрограммированный видеопроигрыватель

Совместимость с предыдущими версиями браузеров

Добавление на веб-страницу видео без подключаемых модулей и внешних проигрывателей удобно для пользователей браузеров Internet Explorer 9 или Internet Explorer 10 с новым интерфейсом Windows, а также для пользователей мобильных устройств, которые не поддерживают подключаемые модули. Однако в таком случае ваша аудитория будет ограничена пользователями, использующими современные браузеры. Элементы video и audio в HTML5 позволяют размещать между тегами текст или код, который будет выполняться, только если браузер пользователя не поддерживает HTML5.

Следующий пример аналогичен предыдущему, но добавляется тег object для запуска проигрывателя Adobe Flash, поддерживающего браузеры прежних версий.

```
<video controls poster="demo.jpg">
  <source src="video.mp4" type="video/mp4" />
  <source src="video.webm" type="video/webm"/>
  <source src="video.ogv" type="video/ogg"/>
  <object>
    <embed src="video.mp4" type="application/x-shockwave-flash" allowfullscreen="false"
allowsriptaccess="always" />
  </object>
  HTML5 Video is required for this example
</video>
```

В этом примере, если браузер поддерживает видео HTML5, то последовательно проверяется возможность воспроизведения представленных форматов видео. Если видео HTML5 не поддерживается, то загружается проигрыватель Flash с помощью тегов `object` и `embed`.

Другим вариантом реализации резервного проигрывателя является размещение ссылки на видеоматериал, как показано в следующем примере:

HTML5 Видео не поддерживается вашим браузером.

[Скачать видео](video.mp4) file.

Совместимость разметки с предыдущими версиями

Семантические элементы по сути ничего не делают. Все, что нужно браузеру делать для их поддержки, — это обрабатывать их как обычный элемент `<div>`. Для этого нужно решить две проблемы.

Большинство семантических элементов HTML5 (за исключением элемента `<time>`) являются блочными элементами. Это означает, что их нужно отображать в отдельной строке с небольшим расстоянием между ними и предшествующими и последующими элементами.

Браузеры, которые не распознают элементы HTML5, не знают, что некоторые из этих элементов следует отображать как блочные, вследствие чего они, скорее всего, сбросят эти элементы в одну беспорядочную кучу. Чтобы решить эту проблему, нужно просто добавить одно новое правило в таблицу стилей:

```
article, aside, figure, figcaption, footer, header, hgroup, nav, section, summary
{
  display: block;
}
```

Это правило таблицы стилей никаким образом не будет влиять на браузеры, которые уже понимают HTML5. Также оно не будет влиять ни на какие правила, которые уже используются для форматирования этих элементов.

Этот метод решает проблему распознавания семантических элементов в большинстве браузеров, но в это большинство не входят Internet Explorer 8 и более старые его версии. Здесь нужно решить вторую проблему, заключающуюся в том, что эти версии IE отказываются применять форматирование таблицы стилей к элементам, которые они не распознают. Эта проблема решается, если зарегистрировать их с помощью команды JavaScript. Например, в следующем JavaScript-коде браузер IE наделяется способностью понимать и применять стиль к элементу `<header>`

```
<script>
document.createElement('header')
</script>
```

Вместо того чтобы самому разрабатывать такой код для всех элементов, можно воспользоваться уже готовым сценарием.

Домашнее задание

Основываясь на изученном материале, создать HTML5-документ следующего содержания:

1. В выбранном макете добавить видео, если нет поля для видео, заменить изображение из макета на блок с видео.
2. Проверить, как будет отображаться видео-плеер в разных браузерах.
3. Добавить в макет новые возможности форм, пройденные на уроке, если форма отсутствует в вашем макете, создать отдельную страничку (скачать любую из представленных <http://365psd.ru/psd/form>).
4. * Задание повышенной сложности (по желанию!) Создать страничку с фоновым видео, которое растягивается на весь экран и используется как фон. (пример <http://timflach.com/>)

Дополнительные материалы

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <http://www.wisdomweb.ru/>
2. <http://html5book.ru/>
3. Гоше Х. HTML5. Для профессионалов. СПб.: Питер, 2013. — 496 с.: ил. ISBN 978-5-496-00099-4.
4. Брайан Хоган. HTML5 и CSS3. Веб-разработка по стандартам нового поколения. Год выпуска 2012, ISBN 978-5-459-00592-9, 978-1934356685, Издательство Питер.
5. Дэвид Макфарланд. Большая книга CSS3