Лекция № 1. Введение. О языке HTML5, нововведения, возможности, теги и атрибуты.

HTML-документ — это обычный текстовый документ, может быть создан как в обычном текстовом редакторе (Блокнот), так и в специализированном, с подсветкой кода (Notepad++, Visual Studio Code и т.п.). HTML-документ имеет расширение .html.

HTML-документ состоит из дерева HTML-элементов и текста. Каждый элемент обозначается в исходном документе начальным (открывающим) и конечным (закрывающим) тегом.

Начальный тег показывает, где начинается элемент, конечный — где заканчивается. Закрывающий тег образуется путем добавления слэша / перед именем тега: <имя тега>…</имя тега>. Между начальным и закрывающим тегами находится содержимое тега — контент.

Одиночные теги не могут хранить в себе содержимого напрямую, оно прописывается как значение атрибута, например, тег <input type="button" value="Кнопка"> создаст кнопку с текстом Кнопка внутри.

Теги могут вкладываться друг в друга, например, Текст . При вложении следует соблюдать порядок их закрытия (принцип «матрёшки»), например, следующая запись будет неверной: Текст .

HTML-элементы могут иметь атрибуты (глобальные, применяемые для всех HTML-элементов, и собственные). Атрибуты позволяют изменять свойства и поведение элемента, для которого они заданы. Атрибуты прописываются в открывающем теге элемента и содержат имя и значение, указываемые в формате имя атрибута="значение".

Каждому элементу можно присвоить несколько значений class и только одно значение id. Множественные значения class записываются через пробел, . Значения class и id должны состоять только из букв, цифр, дефисов и нижних подчеркиваний и должны начинаться только с букв или цифр.

Браузер просматривает (интерпретирует) HTML-документ, выстраивая его структуру (DOM) и отображая ее в соответствии с инструкциями, включенными в этот файл (таблицы стилей, скрипты). Если разметка правильная, то в окне браузера будет отображена HTML-страница, содержащая HTML-элементы — заголовки, таблицы, изображения и т.д.

Процесс интерпретации (парсинг) начинается прежде, чем веб-страница полностью загружена в браузер. Браузеры обрабатывают HTML-документы последовательно, с самого начала, при этом обрабатывая CSS и соотнося таблицы стилей с элементами страницы.

HTML-документ состоит из двух разделов — заголовка — между тегами <head>…</head> и содержательной части — между тегами <body>…</body>.

Язык HTML следует правилам, которые содержатся в файле объявления типа документа (Document Type Definition, или DTD). DTD представляет собой XML-документ, определяющий, какие теги, атрибуты и их значения действительны для конкретного типа HTML. Для каждой версии HTML есть свой DTD.

DOCTYPE отвечает за корректное отображение веб-страницы браузером. DOCTYPE определяет не только версию HTML (например, html), но и соответствующий DTD-файл в Интернете.

Элементы, находящиеся внутри тега <html>, образуют дерево документа, так называемую объектную модель документа, DOM (document object model). При этом элемент <html> является корневым элементом.

Чтобы разобраться во взаимодействии элементов веб-страницы, необходимо рассмотреть так называемые «родственные отношения» между элементами. Отношения между множественными вложенными элементами подразделяются на родительские, дочерние и сестринские.

Предок — элемент, который заключает в себе другие элементы. На рисунке 1 предком для всех элементов является <html>. В то же время элемент <body> является предком для всех содержащихся в нем тегов: <h1>, <p>, , <nav> и т.д.

Потомок — элемент, расположенный внутри одного или более типов элементов. Например, <body> является потомком <html>, а элемент <p> является потомком одновременно для <body> и <html>.

Родительский элемент — элемент, связанный с другими элементами более низкого уровня, и находящийся на дереве выше их. На рисунке 1 <html> является родительским только для <head> и <body>. Тег <p> является родительским только для .

Дочерний элемент — элемент, непосредственно подчиненный другому элементу более высокого уровня. На рисунке 1 только элементы <h1>, <h2>, <p> и <nav> являются дочерними по отношению к <body>.

Сестринский элемент — элемент, имеющий общий родительский элемент с рассматриваемым, так называемые элементы одного уровня. На рисунке 1 <head> и <body> — элементы одного уровня, так же как и элементы <h1>, <h2> и <p> являются между собой сестринскими.

1.1. Элемент <html>

Является корневым элементом документа. Все остальные элементы содержатся внутри тегов <html>...</html>. Все, что находится за пределами тегов, не воспринимается браузером как код HTML и никак им не обрабатывается.

1.2. Элемент <head>

Раздел <head>...</head> содержит техническую информацию о странице: заголовок, описание, ключевые слова для поисковых машин, кодировку и т.д. Введенная в нем информация не отображается в окне браузера, однако содержит данные, которые указывают браузеру, как следует обрабатывать страницу. Для элемента доступны глобальные атрибуты.

1.2.1. Элемент <title>

Обязательным тегом раздела <head> является тег <title>. Текст, размещенный внутри этого тега, отображается в строке заголовка веб-браузера. Длина заголовка должна быть не более 60 символов, чтобы полностью поместиться в заголовке. Текст заголовка должен содержать максимально полное описание содержимого веб-страницы. Для элемента доступны глобальные атрибуты.

1.2.2. Элемент <meta>

Необязательным тегом раздела <head> является одинарный тег <meta>. С его помощью можно задать описание содержимого страницы и ключевые слова для поисковых машин,

автора HTML-документа и прочие свойства метаданных. Элемент <head> может содержать несколько элементов <meta>, потому что в зависимости от используемых атрибутов они несут различную информацию.

```
<meta name="description" content="Описание содержимого страницы">
<meta name="keywords" content="Ключевые слова через запятую">
```

Описание содержимого страницы и ключевые слова одновременно можно указывать на нескольких языках, например, на русском и английском:

```
<meta name="description" lang="ru" content="Описание содержимого страницы">
<meta name="description" lang="en" content="Description">
<meta name="keywords" lang="ru" content="Ключевые слова через запятую">
<meta name="keywords" lang="en" content="Keywords">
```

С помощью тега <meta> можно запретить или разрешить индексацию веб-страницы поисковыми машинами:

Индексация и переход по ссылкам разрешены:

```
<meta name="robots" content="index, follow">
```

Индексация разрешена, переход по ссылкам запрещен:

```
<meta name="robots" content="index, nofollow">
```

Индексация и переход по ссылкам запрещены:

```
<meta name="robots" content="noindex, nofollow">
```

Для автоматической перезагрузки страницы через заданный промежуток времени нужно воспользоваться значением refresh:

```
<meta http-equiv="refresh" content="30">
```

Страница будет перезагружена через 30 секунд. Чтобы перебросить посетителя на другую страницу, нужно указать URL-адрес в параметре url:

```
<meta http-equiv="refresh" content="0; url=http://yandex.ru/">
```

HTML

Для элемента <meta> доступны атрибуты charset, content, http-equiv, name, а также глобальные атрибуты.

1.2.3. Элемент <style>

Внутри этого элемента задаются стили, которые используются на странице. Для задания стилей в HTML-документе используется язык CSS. Таких элементов на странице может быть несколько.

Для элемента доступны атрибуты media, scoped, type, а также глобальные атрибуты.

Внутрь этого элемента можно записывать код форматирования как самих элементов веб-страницы, так и веб-страницы целиком.

```
<style type="text/css">
.paper {
width: 200px;
height: 300px;
background-color: #ef4444;
color: #666666;
}
```

Чтобы подключить к элементу заданный стиль, необходимо через атрибут class (или id) присвоить элементу соответствующее название:

```
<div class="paper">
...
</div>
```

CSS-код можно встраивать непосредственно в элемент разметки в виде значение атрибута style, например:

```
<p style="color: #666666; background-color: #ef4444; padding: 20px;">
```

1.2.4. Элемент <link>

Задать стили для документа можно также при помощи другого способа — записать их в отдельный файл с расширением .css, например, style.css.

1.2.5. Элемент <script>

Элемент <script> позволяет присоединять к документу различные сценарии. Закрывающий тег обязателен, при этом текст сценария может располагаться либо внутри этого элемента, либо во внешнем файле. Если текст сценария расположен во внешнем файле, то он подключается с помощью атрибутов элемента. Для элемента доступны атрибуты async, charset, defer, src, type, а также глобальные атрибуты.

Src Указывает на месторасположение файла со сценарием, значение атрибута — это url файла, содержащего JavaScript-программу.

1.3. Элемент <body>

В этом разделе располагается все содержимое документа. Для элемента доступны глобальные атрибуты.

Теги

<html></html>	корневой элемент html-документа
<head></head>	контейнер для метаданных html-документа
<title></title>	заголовок / имя html-документа
<link>	подключает внешние таблицы стилей
<meta>	мета-данные веб-страницы
<style></style>	подключает таблицы стилей
<body></body>	тело html-документа
-	заголовки шести уровней

<header></header>	секция для вводной информации сайта или группы навигационных ссылок
<footer></footer>	секция для нижнего колонтитула документа или раздела
	параграфы в тексте
<hr>	горизонтальная линия
<pre></pre>	выводит текст с пробелами и переносами
<blockquote></blockquote>	большая цитата
	упорядоченный нумерованный список
	маркированный список
	элемент списка
<dl></dl>	контейнер для термина и его описания
<dt></dt>	задаёт термин
<dd></dd>	расшифровывает термин
<figure></figure>	независимый контейнер для такого контента как изображения, диаграммы и т.п.
<figcaption></figcaption>	заголовок для элемента <figure>
<main></main>	контейнер для уникального основного содержимого в пределах одной страницы сайта
	контейнер для разделов html-документа, группирует блочные элементы
<table></table>	html-таблица
<caption></caption>	подпись к таблице
<colgroup></colgroup>	контейнер для одного или нескольких <col>
<col>	выбирает для форматирования столбцы
<thead></thead>	блок заголовков таблицы
<tbody></tbody>	тело таблицы
<tfoot></tfoot>	нижний колонтитул таблицы
<tr></tr>	строка таблицы
<th></th>	заголовок столбца таблицы
<td></td>	ячейка таблицы
<script></script>	подключает сценарии к странице
<details></details>	контейнер с дополнительными сведениями, который можно открыть или закрыть
<summary></summary>	видимый заголовок для элемента <details>
<dialog></dialog>	диалоговое окно, инспектор или окно
<picture></pictire>	контейнер для одного <img> и ноль или больше <source>
<source>	местоположение и тип альтернативных медиаресурсов для элементов <picture>, <video>, <audio>

<img>	html-изображения
<iframe></iframe>	создаёт встроенный фрейм
<audio></audio>	добавляет аудио-файлы
<video></video>	добавляет видео-файлы
<a>	гиперссылка
	выделяет важные фрагменты текста курсивом
	выделяет полужирным важный текст
	отображает текст шрифтом меньшего размера
<s></s>,<del></ del>	перечёркивает неактуальный текст
<cite></cite>	источник цитирования
<q></q>	краткая цитата
<dfn></dfn>	выделяет термин курсивом
<abbr></abbr>	аббревиатура или акроним
<time></time>	дата / время документа или статьи
	подстрочное написание символов
	надстрочное написание символов
	выделяет текст курсивом без акцента
	задает полужирное начертание отрывка текста, без дополнительного акцента
<u></u>	выделяет отрывок текста подчёркиванием, без дополнительного акцента
<mark></mark>	выделяет фрагменты текста желтым фоном
<bdo></bdo>	задаёт направление написания текста
 	перенос текста на новую строку
<form></form>	html-форма
<label></label>	текстовая метка для элемента <input>
<input>	многофункциональные поля формы
<button></button>	интерактивная кнопка
<select></select>	элемент управления с выбором значений из предложенных вариантов <option>
<datalist></datalist>	контейнер для выпадающего списка элемента <input> с <option>-значениями
<optgroup></ optgroup>	контейнер с заголовком для группы элементов <option>
<option></option>	вариант (опция) в раскрывающемся списке
<textarea>	многострочное поле формы
<output></output>	поле для вывода результата вычисления
<progress></ progress>	индикатор выполнения задачи
<meter></meter>	индикатор измерения в заданном диапазоне
<fieldset></fieldset>	группирует связанные элементы в форме

Глобальные атрибуты:

`accesskey`	Генерирует сочетания клавиш для доступа к текущему элементу. Состоит из разделенного пробелами списка символов. Браузер в первую очередь выбирает те клавиши, которые существуют на раскладке клавиатуры. Применяется к следующим элементам: `<a>`, `<area>`, `<button>`, `<input>`, `<label>`, `<legend>`, `<textarea>`. Принимаемые значения: перечень названий клавиш.
`class`	Определяет имя класса для элемента (используется для определения класса в таблице стилей). Принимаемые значения: имя класса.
`contenteditable`	Определяет, может ли пользователь редактировать содержимое (контент). Позволяет преобразовать любое поле HTML в редактируемый элемент. Принимаемые значения: `true/false`.
`contextmenu`	Добавляет к элементу контекстное меню, заданное тегом `<menu>`. Принимаемые значения: значение атрибута `id` элемента `<menu>`.
`dir`	Определяет направление текста контента в элементах `<bdo>` и `<bdi>`. Принимаемые значения: `ltr/rtl/auto`.
`draggable`	Определяет, может ли пользователь перетащить элемент. Принимаемые значения: `true/false/auto`.
`dropzone`	Определяет область для приема перемещаемых элементов, сообщая браузеру пользователя, какие действия совершить при перемещении. Принимаемые значения: `copy` — содержимое перемещаемого элемента будет скопировано в область. `move` — содержимое перемещаемого элемента будет перемещено в новую область. `link` — при перемещении будет создана ссылка на первоначальные данные элемента.
`hidden`	Указывает на то, что элемент должен быть скрыт. Принимаемые значения: `hidden`.
`id`	Определяет уникальный идентификатор элемента. Принимаемые значения: `id` — идентификатор элемента.
`lang`	Определяет код языка содержимого (контента) в элементе. Принимаемые значения: код языка.

`spellcheck`	Указывает, подлежит ли содержимое элемента проверке орфографии и грамматики. Принимаемые значения: `true/false`.
`style`	Указывает на код CSS, применяемую для оформления элемента. Принимаемые значения: код CSS.
`tabindex`	Определяет порядок перехода к элементу при помощи клавиши TAB. Принимаемые значения: порядковый номер.
`title`	Определяет дополнительную информацию об элементе, задавая всплывающую подсказку для страницы. Принимаемые значения: текст.
`translate`	Разрешает или запрещает перевод текста внутри элемента. Принимаемые значения: `yes/no`.

Псевдоэлемент :before применяется для отображения желаемого контента до содержимого элемента, к которому он добавляется. Работает совместно со свойством content.

Для :before характерны следующие особенности.

- При добавлении :before к блочному элементу, значение свойства display может быть только: block, inline, none, list-item. Все остальные значения будут трактоваться как block.
- При добавлении :before к встроенному элементу, display ограничен значениями inline и none. Все остальные будут восприниматься как inline.
- :before наследует стиль от элемента, к которому он добавляется.

Синтаксис

```
элемент:before { content: "текст" }
```

Значения

Нет.

Пример

HTML5 | CSS2.1 | IE | Cr | Op | Sa | Fx

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>before</title>
  <style>
   li:before {
     content: "¶ "; /* Добавляем желаемый символ перед элементом списка */
   }
   li {
     list-style: none; /* Убираем исходные маркеры */
   }
  </style>
 </head>
 <body>
   <ul>
     <li>Альфа</li>
     <li>Бета</li>
     <li>Гамма</li>
   </ul>
 </body>
</html>
```

Описание

Тег **<blockquote>** предназначен для выделения длинных цитат внутри документа. Текст, обозначенный этим тегом, традиционно отображается как выровненный блок с отступами слева и справа (примерно по 40 пикселов), а также с отбивкой сверху и снизу.

Синтаксис

```
<blockquote>Текст</blockquote>
```

Закрывающий тег

Обязателен.

Пример

HTML5 | IE | Cr | Op | Sa | Fx

```
<!DOCTYPE HTML>
<html>
 <head>
  <meta charset="utf-8">
  <title>Тег BLOCKQUOTE</title>
 </head>
 <body>

  <hr>
  <blockquote>
   <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
   sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat
   volutpat. Ut wisis enim ad minim veniam, quis nostrud exerci tution ullamcorper
   suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
  </blockquote>
  <hr>

 </body>
</html>
```

Атрибут rules

Описание

Сообщает браузеру, где отображать границы между ячейками. Цвет границы указывается с помощью атрибута bordercolor, толщина внешней рамки таблицы — через атрибут border. По умолчанию рамка рисуется вокруг каждой ячейки, образуя тем самым сетку; толщина таких линий составляет 1px.

Синтаксис

```
<table rules="значение">...</table>
```

Значения

All Линия рисуется вокруг каждой ячейки таблицы.

Groups Линия отображается между группами, которые образуются тегами **<thead>**, **<tfoot>**, **<tbody>**, **<colgroup>** или **<col>**.

Cols Линия отображается между колонками.

None Все границы скрываются.

Rows Граница рисуется между строками таблицы, созданых через тег **<tr>**.

Значение по умолчанию

none (если border="0");

all (если значение атрибута border отлично от нуля).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Тег TABLE, атрибут rules</title>
 </head>
 <body>

  <table cellspacing="0" cellpadding="10" rules="rows"
    border="1" width="80%">
   <tr>
    <td> ... </td>
    <td> ... </td>
    <td> ... </td>
   </tr>
  </table>

 </body>
</html>
```

Лекция 2. Работа с текстом, атрибуты, списки.

HTML- текст представлен в спецификации тегами для форматирования и группировки текста. Теги представляют собой контейнеры для текста и не имеют визуального отображения.

Теги для форматирования текста несут смысловую нагрузку и обычно задают для текста, заключенного внутрь, стилевое оформление, например, выделяют текст жирным начертанием или отображают его шрифтом другого семейства (свойство font-family).

Грамотно отформатированный текст дает понять поисковым системам, какие слова несут важную смысловую нагрузку, по каким из них предпочтительно ранжировать веб-страницу в поисковой выдаче. Вся текстовая информация, отображаемая на сайте, размещается внутри тега <body>.

Заголовки являются важными элементами веб-страницы, они упорядочивают текст, формируя его визуальную структуру. Теги <h1>...<h6> должны использоваться только для выделения заголовков нового раздела или подраздела. При использовании заголовков необходимо учитывать их иерархию, т.е. за <h1> должен следовать <h2> и т.д. Также не допускается вложение других тегов в теги <h1>...<h6>.

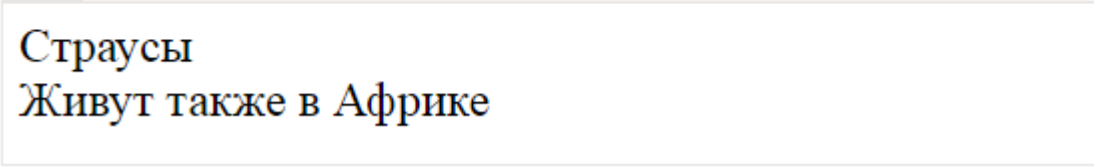
1.1. Тег <h1>

Заголовок самого верхнего уровня, на странице рекомендуется использовать только один раз, по возможности частично дублируя заглавие страницы. Тег <h1> должен быть уникальным для каждой страницы сайта. Рекомендуется прописывать тег в начале статьи, используя ключевое слово в тексте заголовка. Размер шрифта в браузере равен 2em, верхний и нижний отступ по умолчанию 0.67em.

Размеры в em – относительные, они определяются по текущему контексту.

Например, давайте сравним px с em на таком примере:

```
<div style="font-size:24px">
  Страусы
  <div style="font-size:24px">Живут также в Африке</div>
</div>
```



24 пикселей – и в Африке 24 пикселей, поэтому размер шрифта в одинаков.

А вот аналогичный пример с em вместо px:

```
<div style="font-size:1.5em">
  Страусы
  <div style="font-size:1.5em">Живут также в Африке</div>
</div>
```

Так как значение в em высчитывается относительно текущего шрифта, то вложенная строка в 1.5 раза больше, чем первая.

Выходит, размеры, заданные в em, будут уменьшаться или увеличиваться вместе со шрифтом. С учётом того, что размер шрифта обычно определяется в родителе, и может быть изменён ровно в одном месте, это бывает очень удобно.

1.2. Тег <h2>

Им обозначаются подзаголовки тега <h1>. Размер шрифта в браузере равен 1.5em, верхний и нижний отступ по умолчанию 0.83em.

1.3. Тег <h3>

Показывает подзаголовки тега <h2>. Размер шрифта в браузере равен 1.17em, верхний и нижний отступ по умолчанию 1em.

1.4. Теги <h4>, <h5>, <h6>

Обозначают подзаголовки четвёртого, пятого и шестого уровня. Размер шрифта в браузере равен 1em / 0.83em / 0.67em, верхний и нижний отступ по умолчанию 1.33em / 1.67em / 2.33em соответственно.

Для всех тегов доступны глобальные атрибуты.

2. Теги для форматирования текста

2.1. **Тег <b>**

Задаёт полужирное начертания шрифта. Выделяет текст без акцента на его важность.

Для тега доступны глобальные атрибуты.

2.2. **Тег **

Отображает шрифт курсивом, придавая тексту значимость.

Для тега доступны глобальные атрибуты.

2.3. **Тег <i>**

Отображает шрифт курсивом.

Для тега доступны глобальные атрибуты.

2.4. **Тег <small>**

Уменьшает размер шрифта на единицу по отношению к обычному тексту.

Для тега доступны глобальные атрибуты.

2.5. **Тег **

Задаёт полужирное начертание шрифта, относится к тегам логической разметки, указывая браузеру на важность текста.

Для тега доступны глобальные атрибуты.

2.6. **Тег <sub>**

Используется для создания нижних индексов. Сдвигает текст ниже уровня строки, уменьшая его размер.

Для тега доступны глобальные атрибуты.

2.7. **Тег <sup>**

Используется для создания степеней. Сдвигает текст выше уровня строки, уменьшая его размер.

Для тега доступны глобальные атрибуты.

2.9. **Тег <del>,<s>**

Перечёркивает текст. Используется для выделения текста, удаленного из документа.

Для тега доступны следующие атрибуты: cite, datetime.

3. Теги для ввода «компьютерного» текста

3.1. Тег <code>

Служит для выделения фрагментов программного кода. Отображается моноширинным шрифтом (все знаки имеют одинаковую ширину).

Для тега доступны глобальные атрибуты.

3.2. Тег <kbd>

Отмечает фрагмент как вводимый пользователем с клавиатуры. Отображается моноширинным шрифтом.

Для тега доступны глобальные атрибуты.

3.3. Тег <samp>

Применяется для выделения результата, полученного в ходе выполнения программы. Отображается моноширинным шрифтом.

Для тега доступны глобальные атрибуты.

3.4. Тег <var>

Выделяет имена переменных, отображая курсивом.

Для тега доступны глобальные атрибуты.

3.5. Тег <pre>

Позволяет вывести текст на экран, сохранив изначальное форматирование. Пробелы и переносы строк при этом не удаляются.

Для тега доступны глобальные атрибуты.

4. Теги для оформления цитат и определений

4.1. **Тег <abbr>**

Применяется для форматирования аббревиатур. Браузером обычно подчеркивается пунктирной линией. Расшифровка сокращения осуществляется с помощью атрибута title, она появляется при наведении курсора мыши на текст.

Для тега доступны глобальные атрибуты.

4.2. **Тег <bdo>**

Используется для замещения текущего направления текста, т.е. текст в теге отображается зеркально.

Для тега доступен атрибут dir.

4.3. **Тег <blockquote>**

Выделяет цитаты внутри документа, выделяя его отступами и переносами строк.

Для тега доступен атрибут cite.

4.4. **Тег <q>**

Используется для выделения коротких цитат. Браузерами заключается в кавычки.

Для тега доступен атрибут cite.

4.5. Тег <cite>

Применяется для выделения цитат, названий произведений, сносок на другие документы.

Для тега доступны глобальные атрибуты.

4.6. Тег <dfn>

Позволяет выделить текст как определение. Несмотря на наличие специального тега, рекомендуется выделять текст силами CSS.

Для тега доступен атрибут title.

5. Абзацы, средства переноса текста

5.1. **Тег <p>**

Разбивает текст на отдельные абзацы, отделяя друг от друга пустой строкой. Браузер автоматически добавляет верхний и нижний отступ, равный 1em, при этом отступы соседних абзацев «схлопываются».

Для тега доступны глобальные атрибуты.

5.2. **Тег
**

Переносит текст на следующую строку, создавая разрыв строки.

Для тега доступны глобальные атрибуты.

5.3. **Тег <hr>**

Используется для разделения контента на веб-странице. Отображается в виде горизонтальной линии.

Для тега доступны глобальные атрибуты.

Псевдоэлемент :before применяется для отображения желаемого контента до содержимого элемента, к которому он добавляется. Работает совместно со свойством content.

Для :before характерны следующие особенности.

- **При добавлении :before к блочному элементу, значение свойства display может быть только: block, inline, none, list-item. Все остальные значения будут трактоваться как block.**
- **При добавлении :before к встроенному элементу, display ограничен значениями inline и none. Все остальные будут восприниматься как inline.**
- **:before наследует стиль от элемента, к которому он добавляется.**

Синтаксис

```
элемент:before { content: "текст" }
```

Значения

Нет.

Пример

HTML5 | CSS2.1 | IE | Cr | Op | Sa | Fx

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>before</title>
  <style>
   li:before {
     content: "¶ "; /* Добавляем желаемый символ перед элементом списка */
   }
   li {
     list-style: none; /* Убираем исходные маркеры */
   }
  </style>
 </head>
 <body>
   <ul>
     <li>Альфа</li>
     <li>Бета</li>
     <li>Гамма</li>
   </ul>
 </body>
</html>
```

Описание

Тег <blockquote> предназначен для выделения длинных цитат внутри документа. Текст, обозначенный этим тегом, традиционно отображается как выровненный блок с отступами слева и справа (примерно по 40 пикселов), а также с отбивкой сверху и снизу.

Синтаксис

```
<blockquote>Текст</blockquote>
```

Закрывающий тег

Обязателен.

Пример

HTML5 | IE | Cr | Op | Sa | Fx

```
<!DOCTYPE HTML>
<html>
 <head>
  <meta charset="utf-8">
  <title>Тег BLOCKQUOTE</title>
 </head>
 <body>

  <hr>
  <blockquote>
   <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
   sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat
   volutpat. Ut wisis enim ad minim veniam, quis nostrud exerci tution ullamcorper
   suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
  </blockquote>
  <hr>

 </body>
</html>
```

Маркированный список

Тип списка	Код HTML
Список с маркерами в виде круга	<ul type="disc"> ...
Список с маркерами в виде окружности	<ul type="circle"> ...
Список с квадратными маркерами	<ul type="square"> ...

Атрибут rules

Описание

Сообщает браузеру, где отображать границы между ячейками. Цвет границы указывается с помощью атрибута bordercolor, толщина внешней рамки таблицы — через атрибут border. По умолчанию рамка рисуется вокруг каждой ячейки, образуя тем самым сетку; толщина таких линий составляет 1px.

Синтаксис

```
<table rules="значение">...</table>
```

Значения

All Линия рисуется вокруг каждой ячейки таблицы.

Groups Линия отображается между группами, которые образуются тегами **<thead>**, **<tfoot>**, **<tbody>**, **<colgroup>** или **<col>**.
Cols Линия отображается между колонками.
None Все границы скрываются.
Rows Граница рисуется между строками таблицы, созданых через тег **<tr>**.

Значение по умолчанию

none (если border="0");

all (если значение атрибута border отлично от нуля).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Тег TABLE, атрибут rules</title>
 </head>
 <body>

  <table cellspacing="0" cellpadding="10" rules="rows"
    border="1" width="80%">
   <tr>
    <td> ... </td>
    <td> ... </td>
    <td> ... </td>
   </tr>
  </table>

 </body>
</html>
```

Лекция 3. Таблицы, работа с таблицами

HTML-таблицы упорядочивают и выводят на экран данные с помощью строк или столбцов. Таблицы состоят из ячеек, образующихся при пересечении строк и столбцов. Ячейки таблиц могут содержать любые HTML-элементы, такие как заголовки, списки, текст, изображения, элементы форм, а также другие таблицы. Каждой таблице можно добавить связанный с ней заголовок, расположив его перед таблицей или после неё.

Таблицы больше не используются для вёрстки веб-страниц и компоновки отдельных элементов, потому что такой приём не обеспечивает гибкость структуры и адаптивность сайта, существенно увеличивая HTML-разметку.

Для всех элементов таблицы доступны глобальные атрибуты, а также собственные атрибуты.

1. Как создать таблицу

Таблица создаётся при помощи парного тега <table></table>. Данный тег является контейнером для элементов таблицы и все элементы должны находиться внутри него. Например, с помощью данной разметки можно создать таблицу, состоящую из двух столбцов и двух строк:

```
<table>
<tr><th>текст заголовка</th><th>текст заголовка</th></tr> <!--ряд с ячейками заголовков
-->
<tr><td>данные</td><td>данные</td></tr> <!--ряд с ячейками тела таблицы-->
</table>
```

По умолчанию таблица и ячейки не имеют видимых границ. Границы задаются с помощью свойства border:

((Промежутки между ячейками таблицы убираются с помощью свойства table {border-collapse: collapse;}.))

Ширина таблицы по умолчанию равна ширине её внутреннего содержимого. Чтобы установить ширину, нужно задать значение для свойства width:

```
/* сделает ширину таблицы равной ширине блока контейнера, в котором она находится */
table {width: 100%;}
/* задаст фиксированную ширину для таблицы */
table {width: 600px;}
```

((Если для ячеек таблицы заданы внутренние отступы и границы, то ширина таблицы будет включать в себя следующие значения:

padding-left и padding-right, ширина border-left плюс ширина border-right последней ячейки в ряду. Если заданы ширина и границы ячеек, то ширина таблицы будет складываться из ширины ячеек плюс ширина border-left и ширина border-right последней ячейки в ряду.))

2. Как создать строки (ряды) таблицы

Строки или ряды таблицы создаются с помощью тега <tr>. Количество горизонтальных строк таблицы определяется количеством парных тегов <tr></tr>.

3. Как сделать ячейку заголовка столбца таблицы

Элемент <th> создаёт заголовок столбца — специальную ячейку, текст в которой выделяется полужирным. Количество ячеек заголовка определяется количеством пар тегов <th></th>. Для элемента доступны атрибуты colspan, rowspan, headers.

Атрибут	Описание, принимаемое значение
`colspan`	Количество ячеек в строке для объединения по горизонтали. `<td colspan="3">` Возможные значения: число от 1 до 999.

`headers`	Задает список ячеек заголовка, содержащих информацию о заголовке текущей ячейки данных. Предназначен для речевых браузеров.
	`<th id="идентификатор">...</th>`
	`<th headers="идентификатор">...</th>`
	Принимаемые значения: список имен ячеек, разделенных пробелами; эти имена должны быть присвоены ячейкам через их атрибут `id`.

`rowspan`	Количество ячеек в столбце для объединения по вертикали. `<td rowspan="2">` Возможные значения: число от 1 до 999.
`span`	Количество колонок, объединяемых для задания единого стиля, по умолчанию равно 1. `<col span="2">` Принимаемые значения: любое целое положительное число.

```
<table>
<tr><th>ячейка заголовка</th><th>ячейка заголовка</th></tr>
</table>
```

4. Как сделать ячейку тела таблицы

Элемент <td> создаёт ячейки таблицы, внутрь которых помещаются данные таблицы. Парные теги <td></td>, расположенные в одном ряду, определяют количество ячеек в строке таблицы. Количество пар ячеек <td> должно быть равно количеству пар ячеек <th>.

```
<table>
<tr><th>ячейка заголовка</th><th>ячейка заголовка</th></tr>
<tr><td>ячейка тела таблицы</td><td>ячейка тела таблицы</td></tr>
</table>
```

5. Как добавить подпись (заголовок) к таблице

Элемент <caption> создает подпись таблицы. Добавляется непосредственно после тега <table>, вне строки или ячейки.

```
<table>
<caption>Перечень продуктов</caption>
  <tr>
    <th>№ п/п</th>
    <th>Наименование товара</th>
    <th>Ед. изм.</th>
    <th>Количество</th>
    <th>Цена за ед. изм., руб.</th>
    <th>Стоимость, руб.</th>
  </tr>
```

6. Группирование строк и столбцов таблицы

Элемент <colgroup> создает структурную группу столбцов, выделяя логически однородные ячейки. Группирует один или более столбцов для единого форматирования, позволяя применить стили к столбцам вместо того, чтобы повторять стили для каждой ячейки и для каждой строки. Добавляется непосредственно после тегов <table> и <caption>.

Элемент <col> формирует группы столбцов, которые делят таблицу на разделы, не относящиеся к общей структуре, т.е. не содержащие информацию одного типа. Позволяет задавать свойства столбцов для каждого столбца в пределах элемента <colgroup>. С помощью атрибута style можно изменить основной цвет фона ячеек. Для элемента <col> доступен атрибут span, задающий количество столбцов для объединения.

```
<table>
  <colgroup>
    <col span="2" style="background:Khaki"><!-- С помощью этой конструкции задаем цвет
    фона для первых двух столбцов таблицы-->
    <col style="background-color:LightCyan"><!-- Задаем цвет фона для следующего (
    одного) столбца таблицы-->
  </colgroup>
  <tr>
    <th>№ п/п</th>
    <th>Наименование</th>
    <th>Цена, руб.</th>
  </tr>
  <tr>
    <td>1</td>
    <td>Карандаш цветной</td>
    <td>20,00</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Линейка 20 см</td>
    <td>30,00</td>
  </tr>
</table>
```

7. Группировка разделов таблицы

Элемент <thead> создает группу заголовков для строк таблицы с целью задания единого оформления. Используется в сочетании с элементами <tbody> и <tfoot> для указания каждой части таблицы.

Элемент должен быть использован в следующем порядке: как дочерний элемент <table>, после <caption> и <colgroup>, и перед <tbody>, <tfoot> и <tr> элементами. В пределах одной таблицы можно использовать один раз.

Элемент <tbody> группирует основное содержимое таблицы. Используется в сочетании с элементами <thead> и <tfoot>.

Элемент <tfoot> создает группу строк для представления информации о суммах или итогах, расположенную в нижней части таблицы. Используется в таблице один раз. Располагается после тега <thead>, перед тегами <tbody> и <tr>.

```
<table>
 <thead>
  <tr>
    <th>№ п/п</th>
    <th>Наименование товара</th>
     <th>Ед. изм.</th>
     <th>Количество</th>
     <th>Цена за ед. изм., руб.</th>
     <th>Стоимость, руб.</th>
  </tr>
 </thead>
<tfoot>
  <tr>
    <td colspan="5" style="text-align:right">ИТОГО:</td><td>1168,80</td>
  </tr>
</tfoot>
<tbody>
  <tr>
    <td>1.</td>
    <td>Томаты свежие</td><td>кг</td><td>15,20</td><td>69,00</td><td>1048,80</td>
  </tr>
  <tr>
    <td>2.</td>
    <td>Огурцы свежие</td><td>кг</td><td>2,50</td><td>48,00</td><td>120,00</td>
  </tr>
</tbody>
</table>
```

8. Как объединить ячейки таблицы

Атрибуты colspan и rowspan объединяют ячейки таблицы. Атрибут colspan задает количество ячеек, объединенных по горизонтали, а rowspan — по вертикали.

Лекция 4. Вставка изображений, работа с мультимедийным содержимым

1.6. HTML-изображения

HTML-изображения добавляются на веб-страницы с помощью тега <img>. Использование графики делает веб-страницы визуально привлекательнее. Изображения помогают лучше передать суть и содержание веб-документа.

С помощью HTML-тегов <map> и <area> можно создавать карты-изображения с активными областями.

1. Тег <img>

Элемент <img> представляет изображение и его резервный контент, который добавляется с помощью атрибута alt. Так как элемент <img> является строчным, то рекомендуется располагать его внутри блочного элемента, например, <p> или .

Тег <img> имеет обязательный атрибут src, значением которого является абсолютный или относительный путь к изображению:

```
<img src="image.png" alt="Пример кода">
```

	Атрибут `alt` добавляет альтернативный текст для изображения.

`alt`	**Выводится на месте появления изображения до его загрузки или при отключенной графике, а также выводится всплывающей подсказкой при наведении курсора мыши на изображение.** **Синтаксис:** `alt="описание изображения"`.
`eight`	**Атрибут** `height` **задает высоту изображения в** `px`. **Синтаксис:** `height="300"`.
`idth`	**Атрибут** `width` **задает ширину изображения в** `px`. **Синтаксис:** `width="500"`.
`ngdesc`	**URL расширенного описания изображения, дополняющее атрибут** `alt`. **Синтаксис:** `longdesc="http://www.example.com/description.txt"`.
`src`	**Атрибут** `src` **задает путь к изображению.** **Синтаксис:** `src="flower.jpg"`.

1.1. Адрес изображения

Адрес изображения может быть указан полностью (абсолютный URL), например:
`url(http://anysite.ru/images/anyphoto.png)`
Или же через относительный путь от документа или корневого каталогасайта:
`url(../images/anyphoto.png)` — относительный путь от документа,
`url(/images/anyphoto.png)` — относительный путь от корневого каталога.
Это интерпретируется следующим образом:
`../` — означает подняться вверх на один уровень, к корневому каталогу,
`images/` — перейти к папке с изображениями,
`anyphoto.png` — указывает на файл изображения.

1.2. Размеры изображения

Без задания размеров изображения рисунок отображается на странице в реальном размере. Отредактировать размеры изображения можно с помощью атрибутов `width` и `height`. Если будет задан только один из атрибутов, то второй будет вычисляться автоматически для сохранения пропорций рисунка.

1.3. Форматы графических файлов

Формат JPEG *(Joint Photographic Experts Group)*. Изображения JPEG идеальны для фотографий, они могут содержать миллионы различных цветов. Сжимают изображения лучше GIF, но текст и большие площади со сплошным цветом могут покрыться пятнами.
Формат GIF *(Graphics Interchange Format)*. Идеален для сжатия изображений, в которых есть области со сплошным цветом, например, логотипов. GIF-файлы позволяют установить один из цветов прозрачным, благодаря чему фон веб-страницы может проявляться через часть изображения. Также GIF-файлы могут включать в себя простую анимацию. GIF-изображения содержат всего лишь 256 оттенков, из-за чего изображения выглядят пятнистыми и нереалистичного цвета, как плакаты.
Формат PNG *(Portable Network Graphics)*. Включает в себя лучшие черты GIF- и JPEG-форматов. Содержит 256 цветов и дает возможность сделать один из цветов прозрачным, при этом сжимает изображения в меньший размер, чем GIF-файл.

1. Элемент <video>

В простом варианте HTML-разметка для размещения видеофайла на странице имеет следующий вид:

```
<video src="video.ogv" controls></video>
```

HTML

Атрибут `controls` отвечает за появление элементов управления видеоплеером. Вы можете добавить изображение с помощью атрибута `poster`, которое браузер будет использовать, пока загружается видео или пока пользователь не нажмет на кнопку воспроизведения, а также задать высоту и ширину видео.

Как и в случае с аудиофайлами, рекомендуется перечислять в `<source>` все форматы, начиная с более предпочтительного. Также нужно указывать MIME-тип для каждого видеофайла.(MIME (Multipurpose Internet Mail Extension, Многоцелевые расширения почты Интернета) — спецификация для передачи по сети файлов различного типа: изображений, музыки, текстов, видео, архивов и др. Указание MIME-типа используется в HTML обычно при передаче данных форм и вставки на страницу различных объектов.)

```
<video controls width="400" height="300">
  <source src="video.mp4" type="video/mp4"><!-- MP4 для Safari, IE9, iPhone, iPad, Android, и Windows Phone 7 -->
  <source src="video.webm" type="video/webm"><!-- WebM/VP8 для Firefox4, Opera, и Chrome -->
  <source src="video.ogv" type="video/ogg"><!-- Ogg/Vorbis для старых версий браузеров Firefox и Opera -->
  <object data="video.swf" type="application/x-shockwave-flash"><!-- добавляем видеоконтент для устаревших браузеров, в которых нет поддержки элемента video -->
    <param name="movie" value="video.swf">
  </object>
</video>
```

`toplay`	**Автоматическое воспроизведение видеоофайла сразу же после загрузки страницы.**
`ntrols`	**Указывает браузеру, что нужно отобразить базовые элементы управления воспроизведением (воспроизведение, пауза, громкость).**
`ight`	**Задает высоту окна для отображения видеоданных, возможные значения: `px` или `%`**
`op`	**Циклическое воспроизведение видеофайла.**
`ted`	**Выключает звук при воспроизведении видеофайла.**
`ster`	**URL файла изображения, которое будет отображаться во время загрузки видеофайла или до тех пор, пока пользователь не нажмет на кнопку PLAY. Если атрибут не задан, то будет отображаться первый кадр видеофайла.**
`eload`	**Атрибут, отвечающий за предварительную загрузку видеоконтента. Не является обязательным, некоторые браузеры игнорируют его. Возможные значения:** `auto` **— браузер загружает видеофайл полностью, чтобы он был доступен, когда пользователь начнет его воспроизведение.** `metadata` **— браузер загружает первую небольшую часть видеофайла, чтобы определить его основные характеристики.** `none` **— отсутствие автоматической загрузки видеофайла.**
`c`	**Содержит абсолютный или относительный URL-адрес видеофайла.**
`dth`	**Задает ширину окна для отображения видеоданных, возможные значения: `px` или `%`**

1. Элемент <audio>

HTML5-элемент `<audio>` используется для внедрения звукового контента в веб-страницы. В общем виде HTML-разметка имеет следующий вид:

```
<audio src="name.ogg" controls></audio>
```

В настоящий момент не существует аудио формата, который бы работал во всех браузерах, поэтому для обеспечения доступности контента максимально широкой аудитории рекомендуется включать несколько источников звука, представленных с использованием атрибута `src` элемента `<source>`. Одновременно можно добавить резервный контент для браузеров, которые не поддерживают элемент `<audio>`.

```
<audio controls>
  <source src="name.ogg" type="audio/ogg">
```

```
  <source src="name.mp3" type="audio/mpeg">
  <a href="sounds/name.mp3">Скачать name.mp3</a>
</audio>
```
(Атрибуты такие же кроме высоты, ширины)

Лекция 5. Введение в каскадные таблицы стилей. Возможности, способы их подключения в документ.

CSS (Cascading Style Sheets) — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML). Обычно CSS-стили используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, но также могут быть применены к любому виду XML-документа Отделяя стиль представления документов от содержимого документов, CSS упрощает создание веб-страниц и обслуживание сайтов.

CSS поддерживает таблицы стилей для конкретных носителей, поэтому авторы могут адаптировать представление своих документов к визуальным браузерам, слуховым устройствам, принтерам, брайлевским устройствам, карманным устройствам и т.д.
Каскадные таблицы стилей описывают правила форматирования элементов с помощью свойств и допустимых значений этих свойств. Для каждого элемента можно использовать ограниченный набор свойств, остальные свойства не будут оказывать на него никакого влияния.
Объявление стиля состоит из двух частей: **селектора** и **объявления**. В HTML имена элементов нечувствительны к регистру, поэтому «h1» работает так же, как и «H1». Объявление состоит из двух частей: имя свойства (например, `color`) и значение свойства (`grey`). Селектор сообщает браузеру, какой именно элемент форматировать, а в блоке объявления (код в фигурных скобках) перечисляются форматирующие команды — свойства и их значения.

Внешняя таблица стилей представляет собой текстовый файл с расширением `.css`, в котором находится набор CSS-стилей элементов. Файл создаётся в редакторе кода, так же как и HTML-страница. Внутри файла могут содержатся только стили, без HTML-разметки. Внешняя таблица стилей подключается к веб-странице с помощью тега `<link>`, расположенного внутри раздела `<head></head>`. Такие стили работают для всех страниц сайта.
К каждой веб-странице можно присоединить несколько таблиц стилей, добавляя последовательно несколько тегов `<link>`

1.2.4. Элемент <link>

Элемент не требует закрывающего тега. Данный элемент определяет отношение между текущей страницей и другими документами.
```
<link rel="stylesheet" type="text/css" href="style.css">
```

Rel Атрибут определяет отношения между текущим документом и документом, на который идет ссылка.

Type Определяет MIME-тип документа(Задача MIME это идентификация типа содержимого документа по его заголовку.), на который идет ссылка. В данном случае он принимает значение "text/css".

Href Основной атрибут тега, в качестве значения выступает путь к файлу со стилями.

Атрибут `type="text/css"` не является обязательным по стандарту HTML5, поэтому его можно не указывать. Если атрибут отсутствует, по умолчанию используется значение `type="text/css"`.

1.2. Внутренние стили

Внутренние стили встраиваются в раздел `<head></head>` HTML-документа и определяются внутри тега `<style></style>`. Внутренние стили имеют приоритет над внешними, но уступают встроенным стилям (заданным через атрибут `style`).

1.3. Встроенные стили

Когда мы пишем **встроенные стили**, мы пишем CSS-код в HTML-файл, непосредственно внутри тега элемента с помощью атрибута `style`:
Такие стили действуют только на тот элемент, для которого они заданы.

Правило `@import` позволяет загружать внешние таблицы стилей. Чтобы директива `@import` работала, она должна располагаться в таблице стилей (внешней или внутренней) перед всеми остальными правилами:

Правило `@import` также используется для подключения веб-шрифтов:

Как задаются и работают CSS-стили

1) Стили могут наследоваться от родительского элемента (наследуемые свойства или с помощью значения `inherit`);
2) Стили, расположенные в таблице стилей ниже, отменяют стили, расположенные в таблице выше;
3) К одному элементу могут применяться стили из разных источников. Проверить, какие стили применяются, можно в режиме разработчика браузера. Для этого над элементом нужно щёлкнуть правой кнопкой мыши и выбрать пункт «Посмотреть код» (или что-то аналогичное). В правом столбце будут перечислены все свойства, которые заданы для этого элемента или наследуются от родительского элемента, а также файлы стилей, в которых они указаны, и порядковый номер строки кода.
4) При определении стиля можно использовать любую комбинацию селекторов — селектор элемента, псевдокласса элемента, класса или идентификатора элемента.

Порядок подключённых таблиц

Вы можете создать несколько внешних таблиц стилей и подключить их к одной веб-странице. Если в разных таблицах будут встречаться разные значения свойств одного элемента, то в результате к элементу применится правило, находящееся в таблице стилей, идущей в списке ниже.

Лекция 6. Селекторы, атрибуты, классы, идентификаторы, псевдоклассы и псевдоэлементы. Форматирование текста.

2. Виды селекторов

Селекторы представляют структуру веб-страницы. С их помощью создаются правила для форматирования элементов веб-страницы. Селекторами могут быть элементы, их классы и идентификаторы, а также псевдоклассы и псевдоэлементы.

Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя, а также положение в дереве документа. Примером такого состояния служит текстовая ссылка, которая меняет свой цвет при наведении на неё курсора мыши. При использовании псевдоклассов браузер не перегружает текущий документ, поэтому с помощью псевдоклассов можно получить разные динамические эффекты на странице.
Синтаксис применения псевдоклассов следующий.
Селектор:Псевдокласс { Описание правил стиля }

Псевдоэлементы позволяют задать стиль элементов не определённых в дереве элементов документа, а также генерировать содержимое, которого нет в исходном коде текста.
Синтаксис использования псевдоэлементов следующий.
Селектор:Псевдоэлемент { Описание правил стиля }

2.1. Универсальный селектор

Соответствует любому HTML-элементу. Например, `* {margin: 0;}` обнулит внешние отступы для всех элементов сайта. Также селектор может использоваться в комбинации с псевдоклассом или псевдоэлементом: `*:after {CSS-стили}`, `*:checked {CSS-стили}`.

2.2. Селектор элемента

Селекторы элементов позволяют форматировать все элементы данного типа на всех страницах сайта. Например, `h1 {font-family: Lobster, cursive;}` задаст общий стиль форматирования всех заголовков `h1`.

2.3. Селектор класса

Селекторы класса позволяют задавать стили для одного и более элементов с одинаковым именем класса, размещенных в разных местах страницы или на разных страницах сайта. Например, для создания заголовка с классом `headline` необходимо добавить атрибут `class` со значением `headline` в открывающий тег `<h1>` и задать стиль для указанного класса. Стили, созданные с помощью класса, можно применять к другим элементам, не обязательно данного типа.СЛАЙД

2.4. Селектор идентификатора

Селектор идентификатора позволяет форматировать **один** конкретный элемент.
Значение `id` должно быть уникальным, на одной странице может встречаться только один раз и должно содержать хотя бы один символ. Значение не должно содержать пробелов.
Нет никаких других ограничений на то, какую форму может принимать `id`, в частности, идентификаторы могут состоять только из цифр, начинаться с цифры, начинаться с подчеркивания, состоять только из знаков препинания и т. д.
Уникальный идентификатор элемента может использоваться для различных целей, в частности, как способ ссылки на конкретные части документа с использованием идентификаторов фрагментов, как способ нацеливания на элемент при создании сценариев и как способ стилизации конкретного элемента из CSS. СЛАЙД

2.5. Селектор потомка

Селекторы потомков применяют стили к элементам, расположенным внутри элемента-контейнера. Например, `ul li {text-transform: uppercase;}` — выберет все элементы `li`, являющиеся потомками всех элементов `ul`.

Если нужно отформатировать потомки определенного элемента, этому элементу нужно задать стилевой класс:

`p.first a {color: green;}` — данный стиль применится ко всем ссылкам, потомкам абзаца с классом `first`;

`p .first a {color: green;}` — если добавить пробел, то будут стилизованы ссылки, расположенные внутри любого тега класса `.first`, который является потомком элемента `<p>`;

`.first a {color: green;}` — данный стиль применится к любой ссылке, расположенной внутри другого элемента, обозначенного классом `.first`.

2.6. Дочерний селектор

Дочерний элемент является прямым потомком содержащего его элемента. У одного элемента может быть несколько дочерних элементов, а родительский элемент у каждого элемента может быть только один. Дочерний селектор позволяет применить стили только если дочерний элемент идёт сразу за родительским элементом и между ними нет других элементов, то есть дочерний элемент больше ни во что не вложен.

Например, `p > strong` — выберет все элементы `strong`, являющиеся дочерними по отношению к элементу `p`.

2.7. Сестринский селектор

Сестринские отношения возникают между элементами, имеющими общего родителя. Селекторы сестринских элементов позволяют выбрать элементы из группы элементов одного уровня.
`h1 + p` — выберет все первые абзацы, идущие непосредственно за любым тегом `<h1>`, не затрагивая остальные абзацы;
`h1 ~ p` — выберет все абзацы, являющиеся сестринскими по отношению к любому заголовку `h1` и идущие сразу после него.

2.8. Селектор атрибута

Селекторы атрибутов выбирают элементы на основе имени атрибута или значения атрибута:
`[атрибут]` — все элементы, содержащие указанный атрибут, `[alt]` — все элементы, для которых задан атрибут `alt`;
`селектор[атрибут]` — элементы данного типа, содержащие указанный атрибут, `img[alt]` — только картинки, для которых задан атрибут `alt`;
`селектор[атрибут="значение"]` — элементы данного типа, содержащие указанный атрибут с конкретным значением, `img[title="flower"]` — все картинки, название которых содержит слово `flower`;
`селектор[атрибут~="значение"]` — элементы частично содержащие данное значение, например, если для элемента задано несколько классов через пробел, `p[class~="feature"]` — абзацы, имя класса которых содержит `feature`;
`селектор[атрибут|="значение"]` — элементы, список значений атрибута которых начинается с указанного слова, `p[class|="feature"]` — абзацы, имя класса которых `feature` или начинается на `feature`;
`селектор[атрибут^="значение"]` — элементы, значение атрибута которых начинается с указанного значения, `a[href^="http://"]` — все ссылки, начинающиеся на `http://`;
`селектор[атрибут$="значение"]` — элементы, значение атрибута которых заканчивается указанным значением, `img[src$=".png"]` — все картинки в формате `png`;
`селектор[атрибут*="значение"]` — элементы, значение атрибута которых содержит в любом месте указанное слово, `a[href*="book"]` — все ссылки, название которых содержит `book`.

2.9. Селектор псевдокласса

Псевдоклассы — это классы, фактически не прикрепленные к HTML-тегам. Они позволяют применить CSS-правила к элементам при совершении события или подчиняющимся определенному правилу. Псевдоклассы характеризуют элементы со следующими свойствами:
`:link` — не посещенная ссылка;
`:visited` — посещенная ссылка;
`:hover` — любой элемент, по которому проводят курсором мыши;
`:focus` — интерактивный элемент, к которому перешли с помощью клавиатуры или активировали посредством мыши;
`:active` — элемент, который был активизирован пользователем;
`:valid` — поля формы, содержимое которых прошло проверку в браузере на соответствие указанному типу данных;
`:invalid` — поля формы, содержимое которых не соответствует указанному типу данных;
`:enabled` — все активные поля форм;
`:disabled` — заблокированные поля форм, т.е., находящиеся в неактивном состоянии;
`:in-range` — поля формы, значения которых находятся в заданном диапазоне;
`:out-of-range` — поля формы, значения которых не входят в установленный диапазон;

`:lang()` — элементы с текстом на указанном языке;
`:not(селектор)` — элементы, которые не содержат указанный селектор — класс, идентификатор, название или тип поля формы — `:not([type="submit"])`;
`:target` — элемент с символом `#`, на который ссылаются в документе;
`:checked` — выделенные (выбранные пользователем) элементы формы.

2.10. Селектор структурных псевдоклассов

Структурные псевдоклассы отбирают дочерние элементы в соответствии с параметром, указанным в круглых скобках:
`:nth-child(odd)` — нечётные дочерние элементы;
`:nth-child(even)` — чётные дочерние элементы;
`:nth-child(3n)` — каждый третий элемент среди дочерних;
`:nth-child(3n+2)` — выбирает каждый третий элемент, начиная со второго дочернего элемента `(+2)`;
`:nth-child(n+2)` — выбирает все элементы, начиная со второго;
`:nth-child(3)` — выбирает третий дочерний элемент;
`:nth-last-child()` — в списке дочерних элементов выбирает элемент с указанным местоположением, аналогично с `:nth-child()`, но начиная с последнего, в обратную сторону;
`:first-child` — позволяет оформить только самый первый дочерний элемент тега;
`:last-child` — позволяет форматировать последний дочерний элемент тега;
`:only-child` — выбирает элемент, являющийся единственным дочерним элементом;
`:empty` — выбирает элементы, у которых нет дочерних элементов;
`:root` — выбирает элемент, являющийся корневым в документе — элемент `html`.

2.11. Селектор структурных псевдоклассов типа

Указывают на конкретный тип дочернего тега:
`:nth-of-type()` — выбирает элементы по аналогии с `:nth-child()`, при этом берёт во внимание только тип элемента;
`:first-of-type` — выбирает первый дочерний элемент данного типа;
`:last-of-type` — выбирает последний элемент данного типа;
`:nth-last-of-type()` — выбирает элемент заданного типа в списке элементов в соответствии с указанным местоположением, начиная с конца;
`:only-of-type` — выбирает единственный элемент указанного типа среди дочерних элементов родительского элемента.

2.12. Селектор псевдоэлемента

Псевдоэлементы используются для добавления содержимого, которое генерируется с помощью свойства `content`:
`:first-letter` — выбирает первую букву каждого абзаца, применяется только к блочным элементам;
`:first-line` — выбирает первую строку текста элемента, применяется только к блочным элементам;
`:before` — вставляет генерируемое содержимое перед элементом;
`:after` — добавляет генерируемое содержимое после элемента.

Лекция 7. Отступы, границы, блоки и блочная модель. Свойство элементов display.

Выделяют две основные категории HTML-элементов, которые соответствуют типам их содержимого и поведению в структуре веб-страницы — **блочные** и **строчные элементы**. С помощью блочных элементов можно создавать структуру веб-страницы, строчные элементы используются для форматирования текстовых фрагментов

1. Модель визуального форматирования

HTML-документ организован в виде дерева элементов и текстовых узлов. Модель визуального форматирования CSS представляет собой алгоритм, который обрабатывает HTML-документ и выводит его на экран устройства.
Каждый блок в дереве представляет соответствующий элемент или псевдоэлемент, а текст (буквы, цифры, пробелы), находящийся между открывающим и закрывающим тегами, представляет содержимое текстовых узлов.
Чтобы создать дерево блоков, CSS сначала использует каскадирование и наследование, позволяющие назначить вычисленное значение для каждого css-свойства каждому элементу и текстовому узлу в исходном дереве.
Затем для каждого элемента CSS генерирует ноль или более блоков в соответствии со значением свойства `display` этого элемента. Как правило, элемент генерирует один основной блок, который представляет самого себя и содержит свое содержимое. Некоторые значение свойства `display`, например, `display: list-item;`, генерируют блок основного блока и блок дочернего маркера. Другие, например, `display: none;`, приводят к тому, что элемент и/или его потомки вообще не генерируют блоки.
Положение блоков на странице определяется следующими факторами:

- размером элемента (с учётом того, заданы они явно или нет);
- типом элемента (строчный или блочный);
- схемой позиционирования (нормальный поток, позиционированные или плавающие элементы);
- отношениями между элементами в DOM (родительский — дочерний элемент);
- внутренними размерами содержащихся изображений;
- внешней информацией (например, размеры окна браузера).

Свойство элементов Display

Многоцелевое свойство, которое определяет, как элемент должен быть показан в документе.

Синтаксис

```
display: block | inline | inline-block | inline-table | list-item | none | run-in |
table | table-caption | table-cell | table-column-group | table-column | table-
footer-group | table-header-group | table-row | table-row-group
```

Список возможных значений этого свойства, понимаемый разными браузерами очень короткий — block, inline, list-item и none.

block - Элемент показывается как блочный. Применение этого значения для встроенных элементов, например тега ****, заставляет его вести подобно блокам — происходит перенос строк в начале и в конце содержимого.

inline - Элемент отображается как встроенный. Использование блочных тегов, таких как ** **и **<p>**, автоматически создает перенос и показывает содержимое этих тегов с новой строки.
Значение inlineотменяет эту особенность, поэтому содержимое блочных элементов начинается с того места, где окончился предыдущий элемент.

list-item - Элемент выводится как блочный и добавляется маркер списка.

none - Временно удаляет элемент из документа. Занимаемое им место не резервируется и веб-страница формируется так, словно элемента и не было. Изменить значение и сделать вновь видимым элемент можно с помощью скриптов, обращаясь к свойствам через объектную модель. В этом случае происходит переформатирование данных на странице с учетом вновь добавленного элемента.

2. Блочные элементы и блочные контейнеры

Блочные элементы — элементы высшего уровня, которые форматируются визуально как блоки, располагаясь на странице в окне браузера вертикально. Значения свойства `display`, такие как `block`, `list-item` и `table` делают элементы блочными. Блочные элементы генерируют основной блок, который содержит только блок элемента. Элементы со значением `display: list-item` генерируют дополнительные блоки для маркеров, которые позиционируются относительно основного блока.
`<address>`, `<article>`, `<aside>`,
`<blockquote>`,
`<dd>`, `<div>`, `<dl>`, `<dt>`, `<details>`,
`<fieldset>`, `<figcaption>`, `<figure>`, `<footer>`, `<form>`,
`<h1>-<h6>`, `<header>`, `<hr>`,
`<li>`, `<legend>`,
`<nav>`, `<noscript>`,
`<ol>`, `<output>`, `<optgroup>`, `<option>`,
`<p>`, `<pre>`,
`<section>`, `<summary>`,
`<table>`,
`<ul>`
Блочные элементы могут размещаться непосредственно внутри элемента `<body>`. Они создают разрыв строки перед элементом и после него, образуя прямоугольную область, по ширине занимающую всю ширину веб-страницы или блока-родителя.
Блочные элементы могут содержать как строчные, так и блочные элементы, но не оба типа элементов сразу. При необходимости, строки текста, принадлежащие блочному контейнеру, могут быть обёрнуты анонимными контейнерами, которые будут вести себя внутри блока как элементы со значением `display: block;`, а строчные элементы обёрнуты элементом `<p>`.
Блочные элементы могут содержаться только в пределах блочных элементов.

Любой блочный элемент состоит из набора свойств, подобно капустным листам накладываемых друг на друга. Основой блока выступает его контент (это может быть текст, изображение и др.), ширина которого задается свойством width, а высота через height; вокруг контента идут поля (padding), они создают пустое пространство от контента до внутреннего края границ; затем идут собственно сами границы (border) и завершают блок отступы (margin), невидимое пустое пространство от внешнего края границ. Порядок влияния этих свойств на блок четко определён и не может быть нарушен.

Поля

Поле – это расстояние от внутреннего края границы или края блока до воображаемого прямоугольника, ограничивающего содержимое блока. Обозначение «поля» это одинаковое значение полей для всех сторон. Основное предназначение полей — создать пустое пространство вокруг содержимого блочного элемента, например текста, чтобы он не прилегал плотно к краю элемента. Использование полей повышает читабельность текста и улучшает внешний вид страницы.

Границы

Границы — это линии вокруг полей элемента на одной, двух, трёх или всех четырёх его сторонах. У каждой линии есть толщина, стиль и цвет. Для создания рамки применяется универсальное свойство border одновременно задающее все эти параметры, а для создания линий на отдельных сторонах элемента можно воспользоваться свойствами border-left, border-top, border-right и border-bottom, соответственно устанавливающих границу слева, сверху, справа и снизу. В примере 3.2 показано добавление линии слева от элемента.

Отступы

Отступ – пустое пространство от внешнего края границы, полей или содержимого блока. «отступы» - это одинаковое значение отступов для всех сторон.

Для отступов характерны следующие особенности.

- Отступы прозрачны, на них не распространяется цвет фона или фоновая картинка, заданная для блока. Однако если фон установлен у родительского элемента, он будет заметен и на отступах.
- Отступы в отличие от полей могут принимать отрицательное значение, это приводит к сдвигу всего блока в указанную сторону. Так, если задано margin-left: -10px, это сдвинет блок на десять пикселов влево.
- Для отступов характерно явление под названием «схлопывание», когда отступы у близлежащих элементов не суммируются, а объединяются между собой.
- Отступы, заданные в процентах, вычисляются от ширины контента блока. Это касается как вертикальных, так и горизонтальных отступов.

расстояние между блоками равно 20 пикселов, а не 40, которые получаются суммированием верхнего и нижнего отступа у блоков. Это происходит за счёт эффекта схлопывания, при котором близлежащие отступы объединяются. элемента и не оказывает влияние на отступы.

Ширина блока

Ширина блока это комплексная величина и складывается из нескольких значений свойств:

- width — ширина контента, т.е. содержимого блока;
- padding-left и padding-right — поле слева и справа от контента;
- border-left и border-right — толщина границы слева и справа;
- margin-left и margin-right — отступ слева и справа.

Как уже упоминалось, какие-то свойства могут отсутствовать и в этом случае на ширину не оказывают влияние. Общая ширина показана на рис. 3.6 в виде чёрной пунктирной линии.

Если значение width не задано, то оно по умолчанию устанавливается как auto. В этом случае ширина блока будет занимать всю доступную ширину при сохранении значений полей, границ и отступов. Под доступной шириной в данном случае подразумевается ширина контента у родительского блока, а если родителя нет, то ширина контента браузера.

Алгоритм блочной модели

Как уже упоминалось, ширина блока формируется из ширины контента и значений полей, границ и отступов. В браузере Internet Explorer в режиме совместимости (иными словами, когда не указан доктайп) алгоритм меняется автоматически и ширина всего блока устанавливается равной width. Остальные браузеры так просто не меняют алгоритм, к тому же вы знаете, что режим совместимости это зло. В CSS3 есть замечательное свойство box-sizing, которое нам и пригодится. При значении border-box ширина начинает включать поля и границы, но не отступы. Таким образом, подключая box-sizing со значением border-box к своему стилю, мы можем задавать ширину в процентах и спокойно указывать border и padding, не боясь, что изменится ширина блока. К сожалению, с этим свойством связана небольшая проблема, как обычно относящаяся к браузерам — не все браузеры его понимают. Радует, что браузеры хотя бы поддерживают специфические для каждого браузера свойства. В табл. 3.1 приведена поддержка браузерами.

```
<style type="text/css">
   div {
    width: 100%; /* Ширина */
    background: #fc0; /* Цвет фона */
    padding: 20px; /* Поля */
    -moz-box-sizing: border-box; /* Для Firefox */
    -webkit-box-sizing: border-box; /* Для Safari и Chrome */
    box-sizing: border-box; /* Для IE и Opera */
   }  </style>
```

Высота блока

На высоту блока действуют те же правила, что и на ширину. А именно, высота складывается из значений высоты контента (height), полей (padding), границ (border) и отступов (margin). Если свойство height не

указано, то оно считается как auto, в этом случае высота контента вычисляется автоматически на основе содержимого. На рис. 3.8 показаны свойства, дающие итоговую высоту, которая обозначена чёрной пунктирной линией.

Лекция 8. Отображение веб-страниц в различных браузерах. Приведение стилей к общему виду. Сброс стилей.

Каждый браузер устанавливает свои значения стилей по умолчанию для различных HTML-элементов. С помощью CSS Reset мы можем нивелировать эту разницу для обеспечения кроссбраузерности стилей.

Например, вы используете элемент **a** в вашем документе. Большинство браузеров, как Internet Explorer и Firefox, добавляют ссылке *синий цвет* и *подчёркивание*. Однако представьте, что через пять лет кто-то решил создать новый браузер (назовём его UltraBrowser). Разработчикам браузера не нравился синий цвет и раздражало подчёркивание, поэтому они решили выделять ссылки *красным цветом* и *полужирным шрифтом*. Именно исходя из этого, если вы установите базовое значение стилей для элемента **a**, то он гарантированно будет таким, каким вы хотите его видеть, а не как предпочитают его отображать разработчики UltraBrowser.

Как всё начиналось?

CSS Reset впервые был применён в далёком 2004 году, использовался универсальный селектор (*) в начале CSS-файла, чтобы задать всем элементам нулевые отступы (margin и padding).

```
*
{
  margin: 0;
  padding: 0;
}
```

Универсальный селектор работает как регулярное выражение, захватывая каждый элемент на своём пути, без разбора и пощады. Так как до него мы не указали никаких других селекторов, со всех элементов в документе (это лишь в теории, в действительности происходит несколько иначе) удаляются какие-либо отступы.

Но теперь у нас вообще нет никаких отступов, в том числе между отдельными параграфами! Что сделать? Ниже нашего сброса мы опишем нужное нам правило. Сделать это можно разными способами: указать отступ снизу или сверху параграфа, указать его в процентах, пикселях или в em.

```
* { margin: 0; padding: 0; }
p { margin: 5px 0 10px 0; }
```

И уже браузер отображает то, что хотим мы.

Вскоре после этого, CSS-гуру Эрик Мейер (Eric Meyer) производит дальнейшие исследования и делает файл в котором не только сбрасывались отступы, но и устанавливались базовые значения других атрибутов: стили шрифтов, стили списков.

После многочисленных переделок и уточнений, мы приходим к замечательному решению под названием CSS Reset. В нём сброс значений сделан аккуратнее: с применением

непосредственно имён элементов, а не универсального селектора. Он же устанавливает значения по умолчанию для «проблемных» элементов, например таблиц, в которых border-collapse обрабатывается некорректно некоторыми браузерами.

Применение CSS Reset

Давайте остановимся на некоторых моментах использования приёма в реальном мире.

1. Определите, как именно вы будете сбрасывать стили

Я показала два способа сброса стилей: простой, основанный на применении универсального селектора и комплексный, с применением стилей

2. Ваш CSS Reset — это первое, что должен увидеть браузер

Сброс стилей после установки ваших собственных стилей для элементов — это не правильно. В этом случае ничего хорошего от отображения браузером ждать не следует. Запомните, что сначала всегда следует подключать CSS Reset, а потом все остальные стили.
правила, записанные ниже по тексту CSS-файла (и даже ниже по их порядку подключения в документе), перезаписывают правила, объявленные ранее.

3. Используйте отдельный CSS-документ для CSS Reset

Использование отдельного файла для CSS Reset — это обычная практика, которую поддерживает большое число разработчиков.

4. Старайтесь избегать использование универсального селектора

Несмотря на то, что эта концепция работает, её применение чаще всего не является желательным из-за несовместимости с некоторыми браузерами (например, данный селектор некорретно обрабатывается в Internet Explorer). Вам следует использовать этот приём только для простых, небольших, статичных и предсказуемых страниц (если уж вам пришлось делать это).

Лекция 9. Позиционирование элементов на странице. Виды позиционирования. Взаимодействие элементов.

CSS рассматривает макет html-документа как дерево элементов. Уникальный элемент, у которого нет родительского элемента, называется **корневым** элементом. Модуль CSS-позиционирование описывает, как любой из элементов может быть размещен независимо от порядка документа (т.е. извлечен из «потока»).
Модуль CSS3 дополняет и расширяет схему позиционирования. Расположение этих блоков регулируется:

- размерами и типом элемента,
- схемой позиционирования (нормальный поток, обтекание и абсолютное позиционирование),
- отношениями между элементами в дереве документа,
- внешней информацией (например, размер области просмотра, внутренними размерами изображений и т.д.).

Схемы позиционирования

В CSS блок элемента может быть расположен в соответствии с тремя схемами позиционирования:

Нормальный поток

Нормальный поток включает блочный контекст форматирования (элементы с `display` `block`, `list-item` или `table`), строчный (встроенный) контекст форматирования (элементы с `display` `inline`, `inline-block` или `inline-table`), и относительное и «липкое» позиционирование элементов уровня блока и строки.

Обтекание

В обтекающей модели блок удаляется из нормального потока и позиционируется влево или вправо. Содержимое обтекает правую сторону элемента с `float: left` и левую сторону элемента с `float: right`.

Абсолютное позиционирование

В модели абсолютного позиционирования блок полностью удаляется из нормального потока и ему присваивается позиция относительно содержащего блока. Абсолютное позиционирование реализуется с помощью значений `position: absolute;` и `position: fixed;`.

Элементом «вне потока» может быть плавающий, абсолютно позиционированный или корневой элемент.

1. Содержащий блок

Положение и размер блока(ов) элемента иногда вычисляются относительно некоторого прямоугольника, называемого **содержащим блоком** элемента (containing block). В общих словах, содержащий блок — это блок, который содержит другой элемент. В случае нормального потока корневой элемент `html` является содержащим блоком для элемента `body`, который, в свою очередь, является содержащим блоком для всех его дочерних элементов и так далее. В случае позиционирования содержащий блок полностью зависит от типа позиционирования.
Содержащий блок элемента определяется следующим образом:

- Содержащий блок, в котором находится корневой элемент, представляет собой прямоугольник — так называемый **начальный содержащий блок**.
- Для некорневого элемента с `position: static;` или `position: relative;` содержащий блок формируется краем области содержимого ближайшего родительского блока уровня блока, ячейки таблицы или уровня строки.
- Содержащим блоком элемента с `position: fixed;` является окно просмотра.
- Для некорневого элемента с `position: absolute;` содержащим блоком устанавливается ближайший родительский элемент со значением `position: absolute/relative/fixed` следующим образом:
  — если предок — элемент уровня блока, содержащим блоком будет область содержимого плюс поля элемента `padding`;
  — если предок — элемент уровня строки, содержащим блоком будет область содержимого;
  — если предков нет, то содержащий блок элемента определяется как начальный содержащий блок.
- Для «липкого» блока содержащим блоком является ближайший предок с прокруткой или окно просмотра, в противном случае.

2. Выбор схемы позиционирования: свойство position

Свойство `position` определяет, какой из алгоритмов позиционирования используется для вычисления положения блока.
Свойство не наследуется.

position	
Значение:	
`static`	Блок располагается в соответствии с нормальным потоком. Свойства `top`, `right`, `bottom` и `left` не применяются. Значение по умолчанию.

`relative`	Положение блока рассчитывается в соответствии с нормальным потоком. Затем блок смещается относительно его нормального положения и во всех случаях, включая элементы таблицы, не влияет на положение любых следующих блоков. Тем не менее, такое смещение может привести к перекрытию блоков, а также к появлению полосы прокрутки в случае переполнения. Относительно позиционированный блок сохраняет свои размеры, включая разрывы строк и пространство, первоначально зарезервированное для него. Относительно позиционированный блок создает новый содержащий блок для абсолютно позиционированных потомков. Влияние `position: relative;` на элементы таблицы определяется следующим образом: Элементы с `table-row-group`, `table-header-group`, `table-footer-group` ид `table-row` смещаются относительно их обычной позиции в таблице. Если ячейки таблицы занимают несколько строк, смещаются только ячейки начальной строки. `table-column-group`, `table-column` не смещает соответствующий столбец и не оказывает визуального влияния. table-caption and table-cell смещаются относительно своего нормального положения в таблице. Если ячейка таблицы охватывает несколько столбцов или строк, то она смещается целиком.
`absolute`	Положение блока (и, возможно, размер) задается с помощью свойств `top`, `right`, `bottom` и `left`. Эти свойства определяют явное смещение относительно его содержащего блока. Абсолютно позиционированные блоки полностью удаляется из нормального потока, не влияя на расположение сестринских элементов. Отступы `margin` абсолютно позиционированных блоков не схлопываются. Абсолютно позиционированный блок создает новый содержащий блок для дочерних элементов нормального потока и потомков с `position: absolute;`. Содержимое абсолютно позиционированного элемента не может обтекать другие блоки. Абсолютно позиционированный блок могут скрывать содержимое другого блока (или сами могут быть скрыты), в зависимости от значения `z-index` перекрывающихся блоков.
`sticky`	Положение блока рассчитывается в соответствии с нормальным потоком. Затем блок смещается относительно своего ближайшего предка с прокруткой или окна просмотра, если ни у одного из предков нет прокрутки. «Липкий» блок может перекрывать другие блоки, а также создавать полосы прокрутки в случае переполнения. «Липкий» блок сохраняет свои размеры, включая разрывы строк и пространство, первоначально зарезервированное для него. «Липкий» блок создает новый содержащий блок для абсолютно и относительно позиционированных потомков.
`fixed`	Фиксированное позиционирование аналогично абсолютному позиционированию, с отличием в том, что для содержащим блоком устанавливается окно просмотра. Такой блок

	полностью удаляется из потока документа и не имеет позиции относительно какой-либо части документа. Фиксированные блоки не перемещаются при прокрутке документа. В этом отношении они похожи на фиксированные фоновые изображения. При печати фиксированные блоки повторяются на каждой странице, содержащим блоком для них устанавливается область страницы. Блоки с фиксированным положением, которые больше области страницы, обрезаются.
`initial`	Устанавливает значение свойства в значение по умолчанию.
`inherit`	Наследует значение свойства от родительского элемента.

Синтаксис

```
position: static;
position: relative;
position: absolute;
position: sticky;
position: fixed;
position: initial;
position: inherit;
```

CSS

3. Смещение блока: свойства top, right, bottom, left

Элемент считается позиционированным, если свойство `position` имеет значение, отличное от `static`. Позиционированные элементы генерируют позиционированные блоки и могут быть расположены в соответствии со следующими четырьмя физическими свойствами:

top	
Значение:	
`auto`	Влияние значения зависит типа элемента. Значение по умолчанию.
длина	Смещение на фиксированном расстоянии от указанного края. Отрицательные значения допускаются.
`%`	Процентные значения вычисляются относительно высоты содержащего блока. Для «липкого» блока — относительно высоты корневого элемента. Отрицательные значения допускаются.
`initial`	Устанавливает значение свойства в значение по умолчанию.
`inherit`	Наследует значение свойства от родительского элемента.

Синтаксис

```
top: 10px;
top: 2em;
top: 50%;
top: auto;
top: inherit;
```

```css
top: initial;
```

Свойство `top` задает расстояние, на которое верхний край абсолютно позиционированного блока (с учетом его `margin`) смещается ниже верхнего края содержащего блока. Для относительно позиционированных блоков определяет смещение относительно верхнего края самого блока (то есть блоку задается позиция в нормальном потоке, а затем смещение от этой позиции в соответствии с этим свойством).

right

Значение:

`auto`	Влияние значения зависит типа элемента. Значение по умолчанию.
длина	Смещение на фиксированном расстоянии от указанного края. Отрицательные значения допускаются.
`%`	Процентные значения вычисляются относительно ширины содержащего блока. Для «липкого» блока — относительно ширины корневого элемента. Отрицательные значения допускаются.
`initial`	Устанавливает значение свойства в значение по умолчанию.
`inherit`	Наследует значение свойства от родительского элемента.

Синтаксис

```css
right: -10px;
right: .5em;
right: -10%;
right: auto;
right: inherit;
right: initial;
```

Свойство `right` указывает расстояние, на которое правый край абсолютно позиционированного блока (с учетом его `margin`) смещен влево от правого края содержащего блока. Для относительно позиционированных блоков определяет смещение относительно правого края самого блока.

bottom

Значение:

`auto`	Влияние значения зависит типа элемента. Значение по умолчанию.
длина	Смещение на фиксированном расстоянии от указанного края. Отрицательные значения допускаются.

`%`	Процентные значения вычисляются относительно высоты содержащего блока. Для «липкого» блока — относительно высоты корневого элемента. Отрицательные значения допускаются.
`initial`	Устанавливает значение свойства в значение по умолчанию.
`inherit`	Наследует значение свойства от родительского элемента.

Синтаксис

```css
bottom: 50px;
bottom: -3em;
bottom: -50%;
bottom: auto;
bottom: inherit;
bottom: initial;
```

Свойство `bottom` указывает расстояние, на которое нижний край блока смещен вверх относительно нижнего края содержащего блока. Для относительно позиционированных блоков определяет смещение относительно нижнего края самого блока.

left

Значение:

`auto`	Влияние значения зависит типа элемента. Значение по умолчанию.
длина	Смещение на фиксированном расстоянии от указанного края. Отрицательные значения допускаются.
`%`	Процентные значения вычисляются относительно ширины содержащего блока. Для «липкого» блока — относительно ширины корневого элемента. Отрицательные значения допускаются.
`initial`	Устанавливает значение свойства в значение по умолчанию.
`inherit`	Наследует значение свойства от родительского элемента.

Синтаксис

```css
left: 50px;
left: 10em;
left: 20%;
left: auto;
left: inherit;
left: initial;
```

Свойство `left` указывает расстояние, на которое левый край смещен вправо от левого края содержащего блока. Для относительно позиционированных блоков определяет смещение относительно левого края самого блока.

Положительные значения смещают элемент внутрь содержащего блока, а отрицательные — за его пределы.

Лекция 10. Плавающие блоки. Очистка Float`ов.

4. Обтекание: свойство float

Обтекание позволяет блокам смещаться влево или вправо на текущей строке. «Плавающий блок» смещается влево или вправо до тех пор, пока его внешний край не коснется края содержащего блока или внешнего края другого плавающего блока. Если имеется линейный блок, внешняя верхняя часть плавающего блока выравнивается с верхней частью текущего линейного блока.

При использовании свойства `float` для элементов рекомендуется задавать ширину. Тем самым браузер создаст место для другого содержимого. Если для плавающего элемента недостаточно места по горизонтали, он будет смещаться вниз до тех пор, пока не уместится. При этом остальные элементы уровня блока будут его игнорировать, а элементы уровня строки будут смещаться вправо или влево, освобождая для него пространство и обтекая его.

Правила, регулирующие поведение плавающих боков, описываются свойством `float`.

Свойство не наследуется.

float	
`none`	Отсутствие обтекания. Значение по умолчанию.
`left`	Элемент перемещается влево, содержимое обтекает плавающий блок по правому краю.
`right`	Элемент перемещается вправо, содержимое обтекает плавающий блок по левому краю.
`inherit`	Наследует значение свойства от родительского элемента.

Синтаксис

```
float: left; float: right; float: none; float: inherit;
```
CSS

Плавающий блок принимает размеры своего содержимого с учетом внутренних отступов и рамок. Верхние и нижние отступы `margin` плавающих элементов не схлопываются.

Плавающие элементы могут использовать отрицательные отступы `margin`, чтобы перемещаться за пределы области содержимого их родительского элемента.

Свойство автоматически изменяет вычисляемое (отображаемое в браузере) значение свойства `display` на `display: block` для следующих значений: `inline`, `inline-block`, `table-row`, `table-row-group`, `table-column`, `table-column-group`, `table-cell`, `table-caption`, `table-header-group`, `table-footer-group`. Значение `inline-table` меняет на `table`.

Свойство не оказывает влияние на элементы с `display: flex` и `display: inline-flex`. Не применяется к абсолютно позиционированным элементам.

РАЗМЕТКА:

```
<div style="height:150px; border:1px dashed #8bc63e">БЛОК 1.</div>
<div style="height:200px; width:300px; border:3px dashed #e03c32; float:left">БЛОК 2.</div>
<div style="height:150px; border:1px solid #888888">БЛОК 3.</div>
```

5. Управление потоком рядом с плавающими элементами: свойство clear

Свойство `clear` указывает, какие стороны блока/блоков элемента не должны прилегать к плавающим блокам, находящемся выше в исходном документе. В CSS2 и CSS 2.1 свойство применяется только к неплавающим элементам уровня блока.
Свойство не наследуется.

clear	
`none`	Означает отсутствие ограничений на положение элемента относительно плавающих блоков. Значение по умолчанию.
`left`	Смещает элемент вниз относительно нижнего края любого плавающего слева элемента, находящемся выше в исходном документе.
`right`	Смещает элемент вниз относительно нижнего края любого плавающего справа элемента, находящемся выше в исходном документе.
`both`	Смещает элемент вниз относительно нижнего края любого плавающего слева и справа элемента, находящемся выше в исходном документе.
`inherit`	Наследует значение свойства от родительского элемента.

Синтаксис

```
clear: none; clear: left; clear: right; clear: both; clear: inherit;
```
CSS

Для предотвращения отображение фона или границ под плавающими элементами используется правило `{overflow: hidden;}`.

6. Определение контекста наложения: свойство z-index

В CSS каждый блок имеет позицию в трех измерениях. В дополнение к горизонтальному и вертикальному положению, блоки выкладываются вдоль оси Z друг над другом. Положение вдоль оси Z особенно важно, когда блоки визуально накладываются друг на друга.
СЛАЙД

Порядок, в котором дерево документа отрисовывается на экране, описывается с помощью **контекста наложения**. Каждый блок принадлежит одному контексту наложения. Каждый блок в данном контексте наложения имеет целочисленный уровень, который является его положением на оси Z относительно других блоков в том же контексте наложения. Блоки с более высокими уровнями всегда отображаются перед блоками с более низкими уровнями, а блоки с одинаковым уровнем располагаются снизу вверх в соответствии с порядком следования элементов в исходном документе. Блок элемента имеет ту же позицию, что и блок его родителя, если только ему не присвоен другой уровень свойством `z-index`.
Свойство `z-index` позволяет изменить способ наложения элементов друг на друга.
Свойство не наследуется.

z-index	
`auto`	Вычисляется в `0`. Если для блока задано `position: fixed;` или это корневой элемент, значение `auto` также устанавливает новый контекст наложения. Значение по умолчанию.
целое число	Определяет положение блока в текущем контексте наложения. Также устанавливает новый локальный контекст наложения. Можно использовать любое целое число, включая отрицательные числа. Отрицательные значения помещают элемент вглубь экрана.
`inherit`	Наследует значение свойства от родительского элемента.
`initial`	Устанавливает значение свойства в значение по умолчанию.

Синтаксис

```
z-index: auto; z-index: 0; z-index: 5; z-index: 999;
z-index: -1; z-index: inherit; z-index: initial;
```

Как сделать выпадающее меню с двумя уровнями вложения

Разметка горизонтального многоуровневого меню базируется на css-позиционировании. Всем элементам списка `li` задается относительное позиционирование, а выпадающему меню `ul` любого уровня — абсолютное позиционирование.

По умолчанию ширина выпадающего меню равна ширине элемента списка, в который оно вложено. Чтобы изменить ширину, нужно её задать при помощи свойства `width`, например, `.submenu {width: 300px;}`.

Выпадающее меню первого уровня вложения не требует задания смещения, абсолютное позиционирование делает его высоту равной нулю, чтобы не оставлять пустое место под элементом списка. Для меню второго уровня задаётся смещение относительно левой и верхней границы элемента списка `ul {left: 100%; top: 0;}`

Выпадающее меню скрывается с помощью `.submenu {visibility: hidden; opacity: 0;}`, показывается `li:hover > .submenu {visibility: visible; opacity: 1;}`.

Смещаем подменю `.submenu {transform: translateY(10px);}` по горизонтали, скрываем его при помощи `visibility: hidden; opacity: 0;`. При наведении на элемент списка меню становится видимым и перемещается вверх на `10px`. Для отображения галочки не забудьте подключить FontAwesome.

Лекция 11. Основные компоненты веб-сайтов. Виды сайтов, соблюдение семантики при верстке.

Сайт состоит из стандартных элементов, своего рода кирпичиков, которыми можно пользоваться при построении. Давайте посмотрим на эти элементы, чтобы понять, какой функциональностью они обладают и что могут включать в себя.

Шапка. Шапкой (header) называется фрагмент контента, появляющийся в верхней части сайта. Это практически универсальный компонент, хотя в некоторых ситуациях без него можно обойтись. Тем не менее обычно каждый сайт будет (или должен) его иметь.

С шапкой обычно связаны следующие вещи:

Top-Line Branding. Здесь, как правило, располагаются сведения об основной торговой марке. В этой области часто помещается суббренд или строка тегов. Логотип чаще всего занимает верхний левый угол, но это не обязательное правило, а лишь распространенный эталон, к которому привыкло большинство пользователей.

Основная система навигации. С шапкой сайта обычно связана основная система навигации. Именно она позволяет пользователям попасть в глубь сайта. В случае, когда количество страниц невелико, сюда можно поместить всю систему навигации в виде выпадающего меню.

Вспомогательная система навигации. Сайты часто имеют отдельную вторичную систему навигации, в которую входят элементы, не являющиеся ключевыми для процесса продаж. Сюда часто попадают такие разделы, как About Us (о компании) и Contact Us (контакты).

Поиск. Поле поиска чаще всего располагается в шапке. Это вполне логично, ведь именно здесь находится основная система навигации.

Учетная запись и авторизация. Если на сайте присутствует защищенный раздел со ссылкой на учетную запись или на возможность авторизации, он, как правило, помещается в шапку. Существует множество элементов, о которых вам следует помнить. Встречаются сайты, содержащие все эти элементы, но в большинстве случаев достаточно только части.

Навигация/Меню. это вполне самостоятельный компонент. Существует три основные

системы навигации на основе меню. Все остальные варианты в той или иной мере являются их побочным продуктом.

Вкладки. Вкладки обычно используются в качестве основного средства навигации для перехода пользователей в различные разделы сайта. Внутри вкладок отсутствует дополнительная система навигации, вместо этого перенаправление осуществляется средствами связанных с вкладками страниц.

Выпадающие меню. Выпадающие меню могут быть как горизонтальными, так и вертикальными, но чаще выбирают первый вариант. Обычно они функционируют весьма предсказуемым, понятным пользователям способом. Подобные меню позволяют сразу перейти к тому, что мы ищем. Некоторые сайты содержат такое количество параметров, что приходится создавать настраиваемые выпадающие меню.

Древовидные меню. Древовидные меню практически всегда делаются вертикальными и располагаются в боковых панелях. Такие меню могут отражать как весь контент сайта, так и подразделы текущих страниц. При этом иногда они разворачиваются, а иногда их приходится выбирать из статичного списка вариантов.

Нижний колонтитул. Нижние колонтитулы (footers) обычно относятся к одной из двух категорий. Это могут быть как функциональные навигационные инструменты, направляющие пользователя к дополнительному контенту, так и место для той информации, которая обязана присутствовать на сайте, но которую никто не хочет видеть. Но даже в первом случае вы все равно рано или поздно обнаружите, что часть нижнего колонтитула будет содержать информационный «мусор».

Функциональные нижние колонтитулы. Достаточно часто область нижнего колонтитула сайта превращают в важный навигационный инструмент. И это имеет смысл, если пользователь добрался до нижней части страницы, имеет смысл предоставить ему указатели на дополнительный контент.

Распространенные элементы нижних колонтитулов. В нижний колонтитул часто помещают:

– Сведения об авторских правах и ссылки на правовую информацию, например политику конфиденциальности.

– Копию основной системы навигации сайта.

– Полную или частичную карту сайта.

– Не самые важные ссылки на контент, который бывает востребован весьма небольшой частью посетителей (например, информация о том, как поместить свою рекламу на данный сайт).

Популярным подходом к оформлению нижних колонтитулов является включение в них карты сайта, что дает пользователям возможность легко находить нужный контент. Это намного лучше, чем просто завершить страницу, не предоставив никакой дополнительной информации. Зачастую функциональные нижние колонтитулы имеют значительный размер. Еще один распространенный подход: включение в нижний колонтитул динамического контента, например, относящихся к теме записей или последних комментариев. Это побуждает посетителей остаться на сайте.

ТИПЫ САЙТОВ

Для классификации сайтов в интернете обычно используют основные характеристики, по которым сайты разделяют на определенные типы.

САЙТ-ВИЗИТКА

Сайт визитка - это минимально необходимый, но достаточный набор информации, распространение которой полезно как для начинающих компаний, так и для уже состоявшихся организаций. Это своеобразная визитная карточка фирмы, которая содержит основную информацию о деятельности компании и ее контактные данные. В основу данного типа сайта лег принцип "краткость - сестра таланта".

КОРПОРАТИВНЫЙ САЙТ

По сравнению с сайтом-визиткой, корпоративный сайт является более серьезной структурной единицей. Сайты такого типа выступают в роли инструмента, помогающего привлекать новых клиентов и партнеров для сотрудничества.Данный сайт может служить рекламной площадкой в интернете, тут можно публиковать свою продукцию или услуги с возможностью оформить заказ прямо на сайте. Разработка корпоративного сайта выполняется относительно быстро и недорого в отличии от интернет магазина или портала.

ПРОМО-САЙТ

Сайт в виде интернет-проспекта или буклета выступает в основе рекламной кампании и

привлечет внимание многих потенциальных клиентов, которые смогут получить всю необходимую информацию о предлагаемом товаре, услуге, мероприятии. Видеосюжеты, аудио-сообщения, анимированные рекламные объявления, лента новостей и отзывов.

САЙТ-ВИТРИНА

Это логично организованный, структурированный каталог, главное назначение которого - привлечение клиентов. Здесь размещается подробная информация о товаре, или группе товаров, информацию о вашей компании в интернете.

ИНТЕРНЕТ-МАГАЗИН

Интернет-магазин – это сайт-витрина, где клиент может не только просмотреть товары на страницах каталога, но и купить их. Вы можете "положить" в "корзину" понравившиеся вам товары и оформить покупку тут же, выбрать удобный способ оплаты и доставки товара. Полный аналог обычного магазина только в интернете. Главные качества данного сайта - удобство и функциональность. Пользователи должны иметь возможность легко найти необходимый товар, отправить его в "корзину" и оформить покупку в несколько кликов.

ПОРТАЛ

Сайт-портал - это тип сайта, который имеет свою индивидуальность и определенную тематику. Структура и набор функциональных модулей сайта-портала устроены таким образом, чтобы посетители сайта могли максимально в полном объеме получать информацию. Как правило, на сайте-портале всегда есть ленты новостей и событий, форумы, где можно обсудить интересую тему, высказать свое мнение. Нередко сайты порталы имеют свойства и функциональность всех выше перечисленных типов сайтов.

Семантика — это ядро сайта, содержащий ключи, которые отвечают запросам пользователей. Ключи состоят из связанной структуры слов, которые обеспечивают максимальный объем запросов, которые пользователь вводит в поисковую систему.

Особенности семантики

Во время структурирования семантического ядра необходимо учитывать различные типы запросов, которые по сложности отличаются между собой, но помогают получить максимальный трафик с поисковых систем.

Семантика сайта основывается на грамотном применении SEO-тегов, позволяющие продвигать ресурс, не теряя качество в поисковой системе.

Зачем нужна семантика html

Что такое семантика html? Это создание страниц сайта с использованием тегов html, которые соответствуют семантике и обеспечивают последовательную и логичную иерархию веб-страниц. Элементы семантики способны детально описывать назначение и смысл содержания сайта для разработчиков и для поисковых систем.

Семантика html имеет определенные теги, которые могут принадлежать к одной или к нескольким категориям, таким образом прописывая в ядре структуру текста.

Семантика сайта основывается на грамотном применении SEO тегов

Наиболее важные html-теги, связанные с ранжированием страниц сайта в поисковых системах, это теги:

- title;
- description;
- keywords;
- заголовки h1, h2, h3, …

Зачем нужно правильно формировать теги title, description, keywords и h1?

1. **Для высокого ранжирования вашего сайта в поисковой выдаче.** Мета-теги прописываются в части <head> программного кода каждой из страниц интернет-магазина. Они предназначены для того, чтобы поисковые системы понимали, какая информация находится на страницах сайта, и рекомендовали ее пользователям, ищущим ваши товары.
2. Для формирования развернутого сниппета – текстовой информации, отбираемой поисковой машиной для презентации страницы вашего интернет-магазина в результатах поиска. Для сниппета может быть выбран отрывок из текста с ключевым словом или текст из тега description.
3. **Для повышения кликабельности вашего сайта в поисковиках**. Чем лучше и точнее вы опишите то, что встретит покупатель, перейдя на страницу вашего сайта, тем выше будет количество переходов и конверсия поискового трафика в продажи.
4. **Для привлекательного описания ссылки на товар в социальных сетях**. Если

вы продвигаете свои товары в социальных сетях, то к ним должны быть придуманы интригующие описания, которые подтягиваются к ссылкам и привлекают пользователей на ваш сайт.

5. **Для представления дополнительной информации.**

Правильное заполнение мета-тегов помогает страницам сайта не только адекватно восприниматься поисковыми системами и продвигаться в топе, но и конкурировать с другими похожими страницами

Как посмотреть мета-теги страниц интернет-сайта в WEB-браузере?

Для просмотра тегов в коде страницы, для браузеров Google Chrome, Mozilla Firefox и Opera, вам необходимо:

1. Открыть в ВЕБ-браузере интересующую вас страницу.
2. Нажать сочетание клавиш «CTRL» + «U».
3. В новой вкладке браузера откроется HTML-код страницы.
4. При необходимости, Нажать сочетание клавиш «CTRL» + «F» для поиска в коде страницы тега, например: <title>, <description>, <keywords> или <h1>.

Правила заполнения тега TITLE

Мета-тег title

Title – тег названия страницы, который выводится первой строчкой в результатах поиска Google и Yandex для каждого сайта и показывается в названии вкладки браузера. Он оказывает наибольшее влияние на ранжирование страницы, дает представление пользователям и самой поисковой системе о содержании страницы.

Чтобы поисковые системы правильно воспринимали содержание страниц вашего сайта, очень важно грамотно прописать тег title.

1. Тег title должен быть уникальным для каждой страницы

2. Тег title должен быть отличным от тегов description и h1

title очень часто отображается в поисковой выдаче вместе с содержимым тега description, поэтому они должны не дублироваться, а раскрывать и дополнять значение друг друга. Если же они идентичны, то поисковая машина выберет другой фрагмент текста с ключевым запросом.

3. Тег title должен быть оптимизирован под поисковые запросы, соответствующие тематике страницы

Title – это основной тег, который позволяет поисковым машинам понять какая информация находится на странице, поэтому в него необходимо включить поисковый запрос, соответствующий тому как сайт, будут искать.

4. Тег title должен отражать реальное содержимое страницы

Title как можно точнее должен описывать, что пользователь встретит на сайте и что сможет с этим сделать. К примеру, если у вас на сайте можно только ознакомиться с ассортиментом, но нет возможности купить товар, то в title ни в коем случае не стоит включать коммерческие запросы со словами «купить», «заказать», «доставка» и т. д.

5. Тег title должен иметь структуру законченного предложения

Title является визитной карточкой страницы в результатах поиска, поэтому он должен быть читаемым, понятным и интересным для потенциальных покупателей.

1. **Нельзя просто взять и через запятую перечислить все ключевые слова по данному запросу**: *электронные сигареты, купить электронную сигарету, электронные сигареты с доставкой в Минске*.
2. **Нужно сформулировать фразу грамотно**: *купить электронные сигареты с доставкой по Минску*.
3. **Нельзя допускать нагромождения несогласованных между собой ключевиков**: *купить стулья минск недорого.*
4. **Нужно согласовать все слова в title**: *купить стулья недорого в Минске*.

Обратите внимание: при формирования мета-тега title нельзя использовать «.» !

6. Тег title должен быть составлен грамотно

Не только тексты, то и теги не должны содержать грамматических ошибок, так как это одинаково не нравится ни пользователям, ни поисковым системам.

1. Не допускайте описок и ошибок в написании слов;
2. Не используйте жаргонизмы и другие слова низкого стиля;
3. Не злоупотребляйте спецсимволами, восклицательными знаками и заглавными буквами;

4. Не используйте точки и кавычки, так как они разбивают содержимое тега на пассажи;

5. Используйте кавычки и скобки, если этого требуют правила;

6. Используйте написание брендов кириллицей и латиницей.

7. Тег title должен содержать не более 70 символов

На самом деле, в этот тег можно «впихнуть» даже 1200 символов, но, в результатах поиска отображается не более 70 символов.

- **50–57 символов для Google**;
- **65–70 символов для Яндекс**.

Если title длинный, поисковая система сама выберет, какие 70 символов показать пользователю в соответствии с введенным запросом. И этот выбор непредсказуем. Лучше облегчить задачу поисковикам и стараться уместить все или самое важное в первые 50-70 символов, иначе все лишнее будет заменено троеточием. Причем поисковая система может сократить не только конец фразы, но и середину, и самое начало.

Правила заполнения мета-тега DESCRIPTION

Мета-тег description

Мета-тег description существует для описания страницы, которое помогает посетителю определиться, стоит ли переходить на вашу страничку или нет. Текст этого тега часто выводится поисковыми системами сразу под ссылкой на ваш сайт, а также подтягивается к ссылкам в социальных сетях.

Тег **description напрямую не влияет на ранжирование сайтов в поисковых системах**

Как правильно заполнить тег description?

1. Мета-тег description должен описывать содержимое страницы

Мета-тег description должен не только описывать, но и привлекать внимание, рекламировать и содержать заманчивое предложение.

2. Мета-тег description должен быть уникальным и отличным от тегов h1 и title

Если поисковая система не найдет в привычном для нее месте описания страницы сайта, она возьмет тот отрывок, который посчитает нужным.

3. Мета-тег description может содержать все связанные с содержимым страницы ключевые запросы

Тег description может быть более развернутым, чем title, поэтому можно составить четкое описание товара или раздела каталога с употреблением самых частотных ключевых слов.

Попытаемся вместить все ключевики в description, чтобы он был лаконичным, читабельным и привлекательным для покупателей.

Пример:

<meta name="Description" content="В магазине TEFAL.BY вы можете купить утюг Tefal с парогенератором в Минске по ценам производителя. Гарантия 3 года. Отзывы покупателей помогут вам выбрать и заказать утюг Tefal со скидкой 10% на первую покупку!**">**

4. Мета-тег description должен быть написан человеческим языком

Все правила о структуре предложения и грамотности высказывания также распространяются и на составление мета-тега description. Этот тег должен быть заполнен не простым перечислением ключевиков, а логичным высказыванием с грамотным употреблением и написанием запросов. В конце текста необходимо ставить точку или восклицательный знак, можно использовать спецсимволы.

5. Мета-тег description не должен превышать 150 символов

Это оптимальная длина описания, доступная для отображения в поисковиках. В среднем это две строчки текста с учетом пробелов. Также этот тег не должен быть слишком коротким (2-3 слова), иначе поисковику снова придется искать более развернутое описание для страницы вашего сайта.

Правила формирования тегов заголовков H1-H6

Как правильно заполнить тег H1?

1. Тег h1 должен встречаться на странице только один раз

Иногда сеошники пытаются продвинуть сайты, чрезмерно злоупотребляя заголовками первого уровня на страницах сайта. Но поисковые системы давно научились распознавать черные методы продвижения и не придают значения таким примитивным сеошным уловкам.

2. Тег h1 должен быть уникальным

3. Тег h1 должен содержать поисковый запрос, соответствующий названию товара

или раздела каталога

В рекомендациях для копирайтеров часто встречается совет употреблять в заголовках <h1> основной ключевой запрос в прямом вхождении. Но также приветствуется в заголовках использование синонимов и ключевиков, разбавленных тематическими словами. Например, для основного запроса футбольная форма можно придумать несколько вариантов заголовков.

4. Тег h1 должен быть информативным

Тег заголовка должен содержать максимум информации, поэтому вода и мусор тут не допустимы. Необходимо убрать из заголовков:

1. прилагательные: самый лучший, оптимальный, современный, креативный;
2. шаблонные фразы: быстрая доставка, низкие цены, отличный сервис;
3. спецсимволы, заглавные буквы и восклицательные знаки.

5. Тег h1 не должен превышать более 7-8 слов

Лучше всего, если после публикации заголовок уместится в одну строку. Это ускорит поиск информации для клиента.

Правила заполнения тега KEYWORDS

Как правильно заполнить тег keywords?

В мета-тег keywords записывают ключевые слова, которые встречаются на странице.

1. Мета-тег keywords должен содержать ключевые запросы

В теге можно перечислить все поисковые запросы, но нельзя одни и те же запросы повторять несколько раз:

2. Мета-тег keywords может содержать тематические слова и синонимы

Если вы продаете товар, который имеет несколько названий-синонимов, то это стоит перечислить в ключевых словах.

3. Мета-тег keywords лучше формировать из существительных

Для этого мета-тега лучше использовать существительные единственного числа. Изредка – глаголы (купить, заказать, оплатить), еще реже – прилагательные (свежемороженая рыба, резиновый коврик).

4. Мета-тег keywords не должен содержать никаких знаков пунктуации, кроме запятой

Можно ключевые слова перечислять через запятую или через пробел, но в конце нельзя ставить точек или каких-либо иных знаков препинания.

К чему приводит неправильное формирование SEO-тегов?

Если теги не соответствуют правилам их формирования, поисковые системы Google и Яндекс сами формируют сниппет (заголовок и описание страницы сайта) на страницах результатов поиска из содержимого других тегов, анкорных ссылок, заголовков Яндекс каталога и других участков текста, которые поисковым роботам покажутся более адекватными запросам пользователей. Это происходит в нескольких случаях:

1. Уменьшение или превышение оптимальной длины тегов;
2. Отсутствие точных вхождений ключевиков;
3. Повтор ключевых фраз в одном теге;
4. Грамматические и синтаксические ошибки;
5. Дубли на нескольких страницах сайта.

Если теги не соответствуют требованиям поисковых систем, сайт с такими тегами будет показан только после сайтов с идеально составленными тегами. **Поэтому формированию и заполнению тегов нужно отвести достаточно внимания, чтобы избежать дублей, переспама и, соответственно, падения позиций в поиске.**

Лекция №12. Введение в адаптивную верстку. Принципы адаптации сайта под различные устройства.

Адаптивная верстка сайта, или mobile-friendly, заключается в выполнении определенных функций, направленных на создание веб-страницы, способной подстроиться под любое разрешение экрана.

На заре своей деятельности верстальщики создавали несколько вариантов веб-страниц, чтобы сайт мог отображаться на устройствах с разным разрешением окна (это продолжалось вплоть до 2010 года). Позже для решения данных задач стали применять JavaScript (специализированный язык программирования).

Сегодня адаптивная верстка сайта проводится путем использования каскадных таблиц CCS3 и языка разметки HTML5.

Основы адаптивной верстки сайта

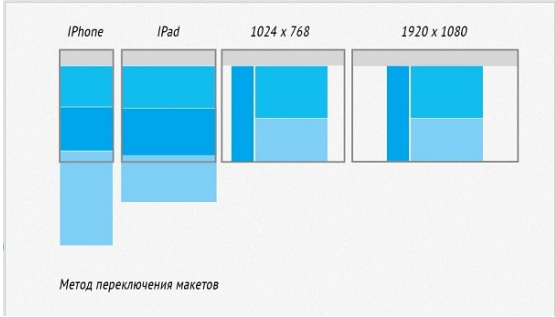
Первым этапом адаптивной верстки сайта является проектирование. В процессе работы дизайнерам необходимо грамотно передать суть и главные идеи, используя относительно маленький экран и всего одну колонку.

Адаптивные макеты могут быть следующих типов:

Резиновый

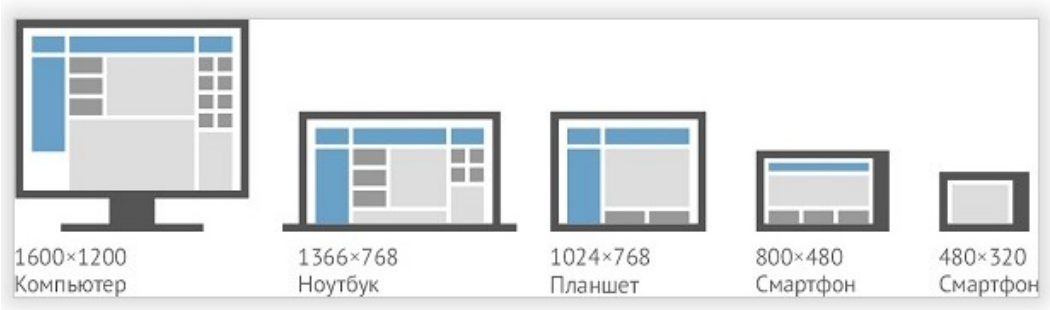
Данный тип просто реализовать, он не вызывает особых трудностей у пользователя. Ключевые блоки сайта сжимаются, пока не достигнут размера экрана мобильного устройства. Если сжатие применить невозможно, то блоки располагают друг за другом в виде ленты.

Перемещение блоков



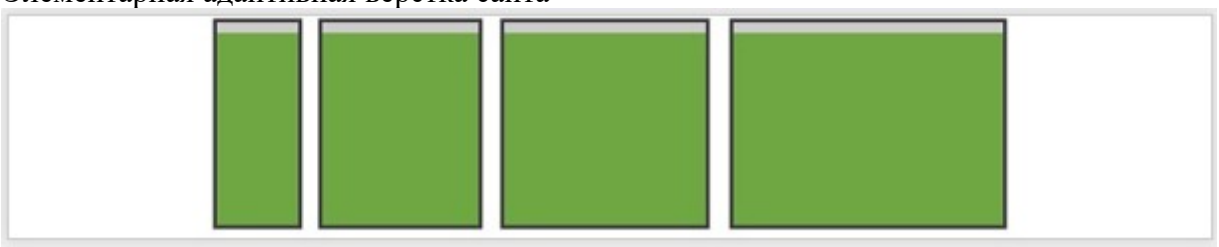
Этот способ подходит для сайта с большим количеством колонок. Расположение сайдбаров (дополнительных блоков) меняется в зависимости от ширины экрана: экран уменьшается – сайдбары перемещаются вниз макета.

Переключение макетов



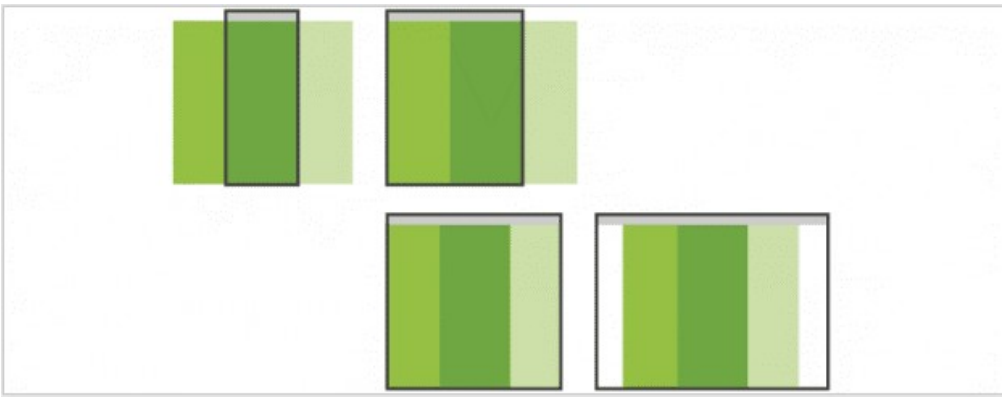
Это достаточно трудоемкий способ, который заключается в том, что каждому разрешению экрана соответствует свой, специально разработанный макет. Однако он облегчает ознакомление с сайтом, но сложность работы снижает популярность его применения.

Элементарная адаптивная верстка сайта



Способ, наиболее подходящий для простых сайтов. Верстальщик просто масштабирует изображение и типографику. Данный способ сложно назвать популярным из-за отсутствия гибкости.

Панели



Данный способ перешел из мобильных приложений, в которых дополнительное меню может появляться при любом положении экрана. Сегодня этот способ не очень популярен, так как мобильная навигация на веб-сайте непривычна и не совсем понятна пользователю.

Ни один из макетов нельзя назвать универсальным. Помните, что все зависит от реализовываемого проекта, способ должен соответствовать его возможностям и удовлетворять его потребности.

Типичный сайт, оптимизированный для мобильных устройств, содержит следующий мета-тег:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Свойство `width` определяет размер окна просмотра. Он может быть установлен на определенное количество пикселей, скажем,`width=600` или на специальное значение `device-width`, которое означает ширину экрана в пикселях CSS в масштабе 100%. Задав мета-тегу viewport значение **“device-width”**, мы говорим браузеру, что ширина области просмотра равняется ширине этого устройства, а не стандартной ширине в 980px, как он может предполагать по-умолчанию. (Есть также соответствующие значения `height` и `device-height`, которые могут быть полезны для страниц с элементами, которые изменяют размер или положение на основе высоты окна просмотра).
Свойство `initial-scale` контролирует уровень масштабирования при первой загрузке страницы.

Свойства `maximum-scale`, `minimum-scale` и `user-scalable` определяют, как пользователям разрешено увеличивать или уменьшать страницу.
На экранах с высоким разрешением экрана страницы с `initial-scale=1` будут эффективно масштабироваться браузерами. Их текст будет плавным и четким, но их растровые изображения, вероятно, не будут использовать полное разрешение экрана.
Сайты могут устанавливать свой viewport на определенный размер. Например, определение «`width=320, initial-scale=1`» может использоваться для точного размещения на маленьком дисплее телефона в портретном режиме. Это может вызвать проблемы, когда браузер не отображает страницу большего размера. Чтобы исправить это, браузеры, если необходимо, увеличат ширину окна просмотра, чтобы заполнить экран по заданной шкале. Это особенно полезно для устройств с большим экраном, таких как iPad.
Для страниц, задающих начальный или максимальный масштаб, это значит, что свойство `width` фактически переводит в *минимальную* ширину viewport. Например, если ваш макет должен иметь ширину не менее 500 пикселей, вы можете использовать следующую разметку. Когда экран шириной более 500 пикселей, браузер будет расширять область просмотра (а не увеличивать), чтобы она соответствовала экрану:

```
<meta name="viewport" content="width=500, initial-scale=1">
```

Другими доступными атрибутами являются `minimum-scale`, `maximum-scale`, и `user-scalable`. Эти свойства влияют на начальный масштаб и ширину, а также ограничивают изменения уровня масштабирования.
Не все мобильные браузеры обрабатывают изменения ориентации таким же образом. Например, Mobile Safari часто просто увеличивает масштаб страницы при смене с вертикальной ориентации на горизонтальный, вместо того, чтобы выкладывать страницу так, как если бы она была первоначально загружена в "ландшафт". Если веб-разработчики хотят, чтобы их настройки масштаба оставались неизменными при переключении ориентации на iPhone, они должны добавить значение `maximum-scale`, чтобы предотвратить это масштабирование, которое иногда имеет нежелательный побочный эффект, который мешает пользователям изменять масштаб:`<meta name="viewport" content="initial-scale=1, maximum-scale=1">`
Meta-тег **HandheldFriendly** определяет оптимизирована ли страница сайта под мобильные устройства на Palm и Blackberry, в таком браузере как AvantGo. Сейчас распознаётся и многими другими мобильными браузерами.
Пример:`<meta name="HandheldFriendly" content="true">`
Meta-тег **MobileOptimized** задаёт ширину области просмотра в мобильных браузеров IE Mobile или Pocket IE. Является аналогом атрибута **width** в meta-теге **viewport**.
Пример:

```
<!-- фиксированная ширина в 320 пикселей -->
<meta name="MobileOptimized" content="320">
<!-- ширина страницы в соответствии с размером экрана, аналог device-width -->
<meta name="MobileOptimized" content="width">
```

Meta-тег **apple-mobile-web-app-capable** позволяет странице работать в полноэкранном режиме, актуален для мобильных устройств Apple.
Пример:`<meta name="apple-mobile-web-app-capable" content="yes">`

Рекомендованный набор метатегов

Используемый мной набор meta-тегов для сайтов с адаптивным дизайном, заточенным под мобильные устройства:

```
<meta name='viewport' content='width=device-width,initial-scale=1'/>
<meta content='true' name='HandheldFriendly'/>
<meta content='width' name='MobileOptimized'/>
<meta content='yes' name='apple-mobile-web-app-capable'/>
```

Возможные параметры для мета-тега viewport

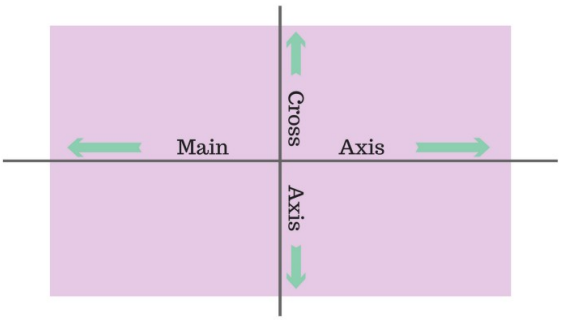
Атрибут	Возможное значение	Описание
idth	Целое неотрицательное значение (от 200px — 10,000px) или константа device-width.	Определяет ширину viewport. Если ширина не указана, в мобильном Safari устанавливается значение 980px, в Opera — 850px, в Android WebKit — 800px, а в IE — 974px.
eight	Целое неотрицательное значение (от 223px до 10,000px) или константа device-height	Определяет высоту viewport. В большинстве случаев на этот атрибут можно не обращать внимания

itial-scale	Число с точкой (от 0.1 до 10). Значение 1.0 — не масштабировать	Определяет начальный масштаб страницы. Чем больше число, тем выше масштаб.
ser-scalable	no или yes	Определяет, может ли пользователь изменять масштаб в окне. По-умолчанию “yes” в мобильном Safari.
inimum-scale	Число с точкой (от 0.1 до 10). 1.0 — не масштабировать	Определяет минимальный масштаб viewport. По-умолчанию “0.25” в мобильном Safari.
aximum-scale	Число с точкой (от 0.1 до 10). 1.0 — не масштабировать	Определяет максимальный масштаб viewport. По-умолчанию “1.6” в мобильном Safari.

Основная задача Flexbox — сделать слои гибкими, а работу с ними — интуитивно понятными. Для достижения этой цели он позволяет контейнерам самим решать, как обращаться со своими дочерними элементами, в том числе изменять их размер и расстояние между ними
У нас есть 4 разноцветных div’a разных размеров, которые находятся внутри серого div’a. У каждого div’a есть свойство `display: block`. Поэтому каждый квадрат занимает всю ширину строки.
Чтобы начать работать с Flexbox, нам нужно сделать наш контейнер flex-контейнером. Делается это так:

```
#container {
  display: flex;
}
```

У flex-контейнера есть две оси: главная ось и перпендикулярная ей.



По умолчанию все предметы располагаются вдоль главной оси: слева направо. Поэтому наши квадраты выровнялись в линию, когда мы применили `display: flex`. Однако `flex-direction` позволяет вращать главную ось.

```
#container {
  display: flex;
  flex-direction: column;
}
```

Важно заметить, что `flex-direction: column` не выравнивает квадраты по оси, перпендикулярной главной. Главная ось сама меняет свое расположение и теперь направлена сверху вниз.
Есть еще парочка свойств для flex-direction: `row-reverse` и `column-reverse`.

`Justify-content` отвечает за выравнивание элементов по главной оси.
Вернемся к `flex-direction: row`.

```
#container {
  display: flex;
  flex-direction: row;
  justify-content: flex-start;
}
```

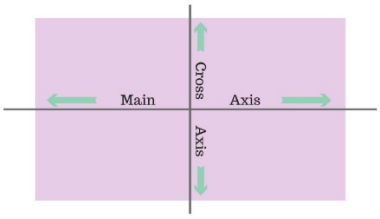
`Justify-content` может принимать 5 значений:

1. `flex-start`;
2. `flex-end`;
3. `center`;
4. `space-between`;
5. `space-around`.

`Space-between` задает одинаковое расстояние между квадратами, но не между контейнером и квадратами. `Space-around` также задает одинаковое расстояние между квадратами, но теперь расстояние между контейнером и квадратами равно половине расстояния между квадратами.

Align Items

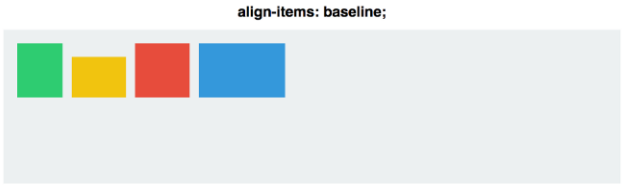
Если `justify-content` работает с главной осью, то `align-items` работает с осью, перпендикулярной главной оси.



Вернемся обратно к `flex-direction: row` и пройдемся по командам `align-items`:

1. `flex-start`;
2. `flex-end`;
3. `center`;
4. `stretch`;
5. `baseline`.

Стоит заметить, что для `align-items: stretch` высота квадратов должна быть равна `auto`. Для `align-items: baseline` теги параграфа убирать не нужно, иначе получится вот так:



Чтобы получше разобраться в том, как работают оси, давайте объединим `justify-content` с `align-items` и посмотрим, как работает выравнивание по центру для двух свойств `flex-direction`:

`Align-self` позволяет выравнивать элементы по отдельности.

```
#container {
  align-items: flex-start;
}
.square#one {
  align-self: center;
}
// Only this square will be centered.
```

Давайте для двух квадратов применим `align-self`, а для остальных применим `align-items: center` и `flex-direction: row`.

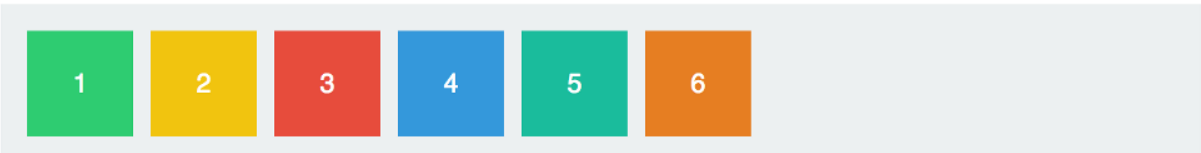
Flex-Basis

`Flex-basis` отвечает за изначальный размер элементов до того, как они будут изменены другими свойствами Flexbox:
`Flex-basis` влияет на размер элементов вдоль главной оси.
Давайте посмотрим, что случится, если мы изменим направление главной оси:
29
Заметьте, что нам пришлось изменить и высоту элементов. `Flex-basis` может определять как высоту элементов, так и их ширину в зависимости от направления оси.

Flex Grow

Это свойство немного сложнее.
Для начала давайте зададим нашим квадратикам одинаковую ширину в 120px:

width: 120px;



По умолчанию значение `flex-grow` равно 0. Это значит, что квадратам запрещено расти (занимать оставшееся место в контейнере).
Попробуем задать `flex-grow` равным 1 для каждого квадрата:

flex-grow: 1; width: 120px;

Квадраты заняли оставшееся место в контейнере. Значение `flex-grow` аннулирует значение ширины.
Но здесь возникает один вопрос: что значит `flex-grow: 1`?
Попробуем задать `flex-grow` равным 999:

flex-grow: 999;

И… ничего не произошло. Так получилось из-за того, что `flex-grow` принимает не абсолютные значения, а относительные.
Это значит, что не важно, какое значение у `flex-grow`, важно, какое оно по отношению к другим квадратам:

Вначале `flex-grow` каждого квадрата равен 1, в сумме получится 6. Значит, наш контейнер поделен на 6 частей. Каждый квадрат будет занимать 1/6 часть доступного пространства в контейнере.
Когда `flex-grow` третьего квадрата становится равным 2, контейнер делится на 7 частей (1 + 1 + 2 + 1 + 1 + 1).
Теперь третий квадрат занимает 2/7 пространства, остальные — по 1/7.
И так далее.
Стоит помнить, что `flex-grow` работает только для главной оси (пока мы не поменяем ее направление).

Flex Shrink

`Flex-shrink` — прямая противоположность `flex-grow`. Оно определяет, насколько квадрату можно уменьшиться в размере.
`Flex-shrink` используется, когда элементы не вмещаются в контейнер.
Вы определяете, какие элементы должны уменьшиться в размерах, а какие — нет. По умолчанию значение `flex-shrink` для каждого квадрата равно 1. Это значит, что квадраты будут сжиматься, когда контейнер будет уменьшаться.
Зададим `flex-grow` и `flex-shrink` равными 1:

Теперь давайте поменяем значение `flex-shrink` для третьего квадрата на 0. Ему запретили сжиматься, поэтому его ширина останется равной 120px:
Стоит помнить что `flex-shrink` основывается на пропорциях. То есть, если у квадрата `flex-shrink` равен 6, а у остальных он равен 2, то, это значит, что наш квадрат будет сжиматься в три раза быстрее, чем остальные.

Flex

`Flex` заменяет `flex-grow`, `flex-shrink` и `flex-basis`.
Значения по умолчанию: `0 (grow) 1 (shrink) и auto (basis)`.
Создадим два квадрата:

```
.square#one {
  flex: 2 1 300px;
}
.square#two {
  flex: 1 2 300px;
}
```

У обоих квадратов одинаковый `flex-basis`. Это значит, что они оба будут шириной в 300px (ширина контейнера: 600px плюс margin и padding).
Но когда контейнер начнет увеличиваться в размерах, первый квадрат (с большим `flex-grow`) будет увеличиваться в два раза быстрее, а второй квадрат (с наибольшим `flex-shrink`) будет сжиматься в два раза быстрее.

https://developer.mozilla.org/ru/docs/Mozilla/Mobile/Viewport_meta_tag

https://www.insales.com.ua/blogs/blog/adaptivnaya-versiya-sayta

https://sales-generator.ru/blog/adaptatsiya-sayta-pod-mobilnye-ustroystva/

https://o-es.ru/blog/nyuansy-adaptatsii-sajta-dlya-mobilnyh-ustrojstv/

Лекция №13. Введение в JavaScript. Подключение, выполнение скриптов. Типы данных, ввод и вывод данных. Порядок исполнения, внешние скрипты. Переменные.

Что такое JavaScript?
1) JavaScript — язык сценариев, или скриптов. Скрипт представляет собой программный код — набор инструкций, который не требует предварительной обработки (например, компиляции) перед запуском. Код JavaScript интерпретируется движком браузера во время загрузки веб-страницы. Интерпретатор браузера выполняет построчный анализ, обработку и выполнение исходной программы или запроса.
2) JavaScript — объектно-ориентированный язык с прототипным наследованием. Он поддерживает несколько встроенных объектов, а также позволяет создавать или удалять свои собственные (пользовательские) объекты. Объекты могут наследовать свойства непосредственно друг от друга, образуя цепочку объект-прототип.

1. Подключение сценариев к html-документу

Сценарии JavaScript бывают **встроенные**, т.е. их содержимое является частью документа, и **внешние**, хранящиеся в отдельном файле с расширением `.js`. Сценарии можно подключить в html-документ следующими способами:

В виде гиперссылки.
Для этого нужно разместить код в отдельном файле и включить ссылку на файл в заголовок

```
<head>
<script src="script.js"></script>
</head>
```

HTML

или тело страницы.

```
<body>
<script src="script.js"></script>
</body>
```

HTML

Этот способ обычно применяется для сценариев большого размера или сценариев, многократно используемых на разных веб-страницах.

В виде обработчика события.
Каждый html-элемент имеет JavaScript-события, которые срабатывают в определенный момент. Нужно добавить необходимое событие в html-элемент как атрибут, а в качестве значения этого атрибута указать требуемую функцию. Функция, вызываемая в ответ на срабатывание события, является **обработчиком события**. В результате срабатывания события исполнится связанный с ним код. Этот способ применяется в основном для коротких сценариев, например, можно установить смену цвета фона при нажатии на кнопку:

```
<script>
var colorArray = ["#5A9C6E", "#A8BF5A", "#FAC46E", "#FAD5BB", "#F2FEFF"]; //
создаем массив с цветами фона
var i = 0;

function changeColor(){
    document.body.style.background = colorArray[i];
    i++;
    if( i > colorArray.length - 1){
        i = 0;
    }
}
</script>
<button onclick="changeColor();">Change background</button>
```

Внутрь элемента `<script>`.
Элемент `<script>` может вставляться в любое место документа. Внутри тега располагается код, который выполняется сразу после прочтения браузером, или содержит описание функции, которая выполняется в момент ее вызова. Описание функции можно располагать в любом месте, главное, чтобы к моменту ее вызова код функции уже был загружен.
Обычно код JavaScript размещается в заголовке документа (элемент `<head>`) или после открывающего тега `<body>`. Если скрипт используется после загрузки страницы, например, код счетчика, то его лучше разместить в конце документа:

```
<footer>
<script>
document.write("Введите свое имя");
</script>
</footer>
</body>
```

2. Типы данных и переменные в JavaScript

Данные могут быть представлены в различных формах или типах. Большая часть функциональности JavaScript реализуется за счет простого набора объектов и типов данных. Функциональные возможности, связанные со строками, числами и логикой, базируются на строковых, числовых и логических типах данных. Другая функциональная возможность, включающая регулярные выражения, даты и математические операции, осуществляется с помощью объектов RegExp, Date и Math.
Литералы в JavaScript представляют собой особый класс типа данных, фиксированные значения одного из трех типов данных — строкового, числового или логического:

```
"это строка"
3.14
true
alert("Hellow"); // "Hellow" - это литерал
var myVariable = 15; // 15 - это литерал
```

JavaScript

Примитивный тип данных является экземпляром определенного типа данных, таких как строковый, числовой, логический, `null` и `undefined`.

2.1. Переменные в JavaScript

Данные, обрабатываемые сценарием JavaScript, являются **переменными**. Переменные представляют собой именованные контейнеры, хранящие данные (значения) в памяти компьютера, которые могут изменяться в процессе выполнения программы. Переменные имеют **имя**, **тип** и **значение**.

Имя переменной, или **идентификатор**, может включать только буквы `a-z`, `A-Z`, цифры `0-9` (цифра не может быть первой в имени переменной), символ `$` (может быть только первым символом в имени переменной или функции) и символ подчеркивания `_`, наличие пробелов не допускается. Длина имени переменной не ограничена. Можно, но не рекомендуется записывать имена переменных буквами русского алфавита, для этого они должны быть записаны в Unicode. В качестве имени переменной нельзя использовать ключевые слова JavaScript. Имена переменных в JavaScript чувствительные к регистру, что означает, что переменная `var message;` и `var Message;` — разные переменные.

Переменная создается (объявляется) с помощью ключевого слова `var`, за которым следует имя переменной, например, `var message;`. Объявлять переменную необходимо перед ее использованием.

Переменная **инициализируется** значением с помощью операции присваивания `=`, например, `var message="Hellow";`, т.е. создается переменная `message` и в ней сохраняется ее **первоначальное** значение `"Hellow"`. Переменную можно объявлять без значения, в этом случае ей присваивается значение по умолчанию `undefined`. Значение переменной может изменяться во время исполнения скрипта. Разные переменные можно объявлять в одной строке, разделив их запятой:

```
var message="Hellow", number_msg = 6, time_msg = 50;
```
JavaScript

2.2. Типы данных переменных

JavaScript является нетипизированным языком, тип данных для конкретной переменной при ее объявлении указывать не нужно. Тип данных переменной зависит от значений, которые она принимает. Тип переменной может изменяться в процессе совершения операций с данными (**динамическое приведение типов**). Преобразование типов выполняется автоматически в зависимости от того, в каком контексте они используются. Например, в выражениях, включающих числовые и строковые значения с оператором `+`, JavaScript преобразует числовые значения в строковые:

```
var message = 10 + " дней до отпуска"; // вернет "10 дней до отпуска"
```
JavaScript

Получить тип данных, который имеет переменная, можно с помощью оператора `typeof`. Этот оператор возвращает строку, которая идентифицирует соответствующий тип.

```
typeof 35; // вернет "number"
typeof "text"; // вернет "string"
typeof true; // вернет "boolean"
typeof [1, 2, 4]; // вернет "object"
typeof undefined; // вернет "undefined"
typeof null; // вернет "object"
```
JavaScript

Все типы данных в JavaScript делятся на две группы — **простые** типы данных *(primitive data types)* и **составные** типы данных *(composite data types)*.

К **простым** типам данных относят строковый, числовой, логический, `null` и `underfined`.

2.2.1. Строковый тип (string)

Используется для хранения строки символов, заключенных в двойные или одинарные кавычки. Пустой набор символов, заключенный в одинарные или двойные кавычки, является пустой строкой. Число, заключенное в кавычки, также является строкой.

```
var money = ""; // пустая строка, ноль символов
var work = 'test';
var day = "Sunday";
var x = "150";
```
JavaScript

В строку в двойных кавычках можно включить одиночную кавычку и наоборот. Кавычка того же типа отключается с помощью символа обратного слэша `\` (так называемая **escape-последовательность**):

```
document.writeln("\"Доброе утро, Иван Иваныч!\"\n"); // выведет на экран "Доброе утро, Иван Иваныч!"
```
JavaScript

Строки можно сравнивать, а также объединять с помощью операции конкатенации `+`.

Благодаря автоматическому приведению типов можно объединять числа и строки. Строки являются постоянными, после того, как строка создана, она не может быть изменена, но может быть создана новая строка путем объединения других строк.

2.2.2. Числовой тип (number)

Используется для числовых значений. Числа в языке JavaScript бывают двух типов: целые числа *(integer)* и числа с плавающей точкой *(floating-point number)*. Целочисленные величины могут быть положительными, например `1`, `2`, и отрицательными, например `-1`, `-2`, или равными нулю. `1` и `1.0` — одно и то же значение. Большинство чисел в JavaScript записываются в десятичной системе счисления, также может использоваться восьмеричная и шестнадцатеричная системы.

В **десятичной** системе значения числовых переменных задаются с использованием арабских цифр `1, 2, 3, 4, 5, 6, 7, 8, 9, 0`.

В **восьмеричном** формате числа представляет собой последовательность, содержащая цифры от `0` до `7` и начинающаяся с префикса `0`.

Для **шестнадцатеричного** формата добавляется префикс `0x` (`0X`), за которым следует последовательность из цифр от `0` до `9` или букв от `a` (`A`) до `f` (`F`), соответствующие значениям от `10` до `15`.

```
var a = 120; // целое десятичное числовое значение
var b = 012; // восьмеричный формат
var c = 0xfff; // шестнадцатеричный формат
var d = 0xACFE12; // шестнадцатеричный формат
```
JavaScript

Числа с плавающей точкой представляют собой числа с дробной десятичной частью, либо это числа, выраженные в экспоненциальном виде. Экспоненциальная запись чисел предполагает следующий вид: число с дробной десятичной частью, за ним следует буква `e`, которая может быть указана как в верхнем, так и в нижнем регистре, далее — необязательный знак `+` или `-` и целая экспонента.

```
var a = 6.24; // вещественное число
var b = 1.234E+2; // вещественное число, эквивалентно 1.234 X 10²
var c = 6.1e-2; // вещественное число, эквивалентно 6.1 X 10⁻²
```
JavaScript

2.2.3. Логический тип (boolean)

Данный тип имеет два значения, `true` (истина), `false` (ложь). Используется для сравнения и проверки условий.

```
var answer = confirm("Вам понравилась эта статья?\n Нажмите OK. Если нет, то
нажмите Cancel.");
if (answer == true)
{
alert("Спасибо!");
}
```
JavaScript

Также существуют специальные типы простых значений:
нулевой тип — данный тип имеет одно значение `null`, которое используется для представления несуществующих объектов.
неопределенный тип — тип переменной `underfined` означает отсутствие первоначального значения переменной, а также несуществующее свойство объекта.
Составные типы данных состоят из более чем одного значения. К ним относятся объекты и особые типы объектов — массивы и функции. Объекты содержат свойства и методы, массивы представляют собой индексированный набор элементов, а функции состоят из коллекции инструкций.

2.3. Глобальные и локальные переменные

Переменные по области видимости делятся на **глобальные** и **локальные**. **Область видимости** представляет собой часть сценария, в пределах которой имя переменной связано с этой переменной и возвращает ее значение. Переменные, объявленные внутри тела функции, называются **локальными**, их можно использовать только в этой функции. Локальные переменные создаются и уничтожаются вместе с соответствующей функцией.
Переменные, объявленные внутри элемента `<script>`, или внутри функции, но без использования ключевого слова `var`, называются **глобальными**. Доступ к ним может осуществляться на протяжении всего времени, пока страница загружена в браузере. Такие переменные могут использоваться всеми функциями, позволяя им обмениваться данными.
Глобальные переменные попадают в **глобальное пространство имен**, которое является местом взаимодействия отдельных компонентов программы. Не рекомендуется объявлять переменные таким способом, так как аналогичные имена переменных уже могут использоваться любым другим кодом, вызывая сбой в работе скрипта.
Глобальное пространство в JavaScript представляется глобальным объектом `window`.
Добавление или изменение глобальных переменных автоматически обновляет глобальный объект. В свою очередь, обновление глобального объекта автоматически приводит к обновлению глобального пространства имен.
Если глобальная и локальная переменная имеют одинаковые имена, то локальная переменная будет иметь преимущество перед глобальной.
Локальные переменные, объявленные внутри функции в разных блоках кода, имеют одинаковые области видимости. Тем не менее, рекомендуется помещать объявления всех переменных в начале функции.
Циклы JavaScript обеспечивают многократное выполнение повторяющихся вычислений. Они оптимизируют процесс написания кода, выполняя одну и ту же инструкцию или блок инструкций, образующих тело цикла, заданное число раз (используя переменную-счётчик) или пока заданное условие истинно. Циклы выполняют обход последовательности значений.
Однократное выполнение цикла называется **итерацией**.
На производительность цикла влияют количество итераций и количество операций, выполняемых в теле цикла каждой итерации.
В JavaScript существуют следующие операторы цикла:
1) `for` используется когда вы заранее знаете, сколько раз вам нужно что-то сделать;
2) `for...in` используется для обхода свойств объектов;
3) `while` используется когда вы не знаете, сколько раз нужно что-то сделать;
4) `do...while` работает аналогично с оператором `while`. Отличается тем, что `do...while` всегда выполняет выражение в фигурных скобках, по крайней мере один раз, даже если проверка условия возвращает `false`.

1. Цикл for

Цикл `for` используется для выполнения итераций по элементам массивов или объектов, напоминающих массивы, таких как `arguments` и `HTMLCollection`. Условие проверяется перед каждой итерацией цикла. В случае успешной проверки выполняется код внутри цикла, в противном случае код внутри цикла не выполняется и программа продолжает работу с первой строки, следующей непосредственно после цикла.
Следующий цикл выведет на консоль строчку `Hello, JavaScript!` пять раз.

```
for (var i = 0; i < 5; i++) {
  console.log(i + ": Hello, JavaScript!");
}
```
JavaScript

1.1. Как работает цикл for

Цикл `for` состоит из трёх разных операций:
Шаг 1. **инициализация** `var i = 0;` — объявление переменной-счётчика, которая будет проверяться во время выполнения цикла. Эта переменная инициализируется со значением `0`. Чаще всего в качестве счётчиков цикла выступают переменные с именами `i`, `j` и `k`.
Шаг 2. **проверка условия** `i < 5;` — условное выражение, если оно возвращает `true`, тело цикла (инструкция в фигурных скобках) будет выполнено. В данном примере проверка условия идёт до тех пор, пока значение счётчика меньше `5`.
Шаг 3. **завершающая операция** `i++` — операция приращения счётчика, увеличивает значение переменной `var i` на единицу. Вместо операции инкремента также может использоваться операция декремента.
По завершении цикла в переменной `var i` сохраняется значение `1`. Следующий виток цикла выполняется для `for (var i = 1; i < 5; i++) { }`. Условное выражение вычисляется снова, чтобы проверить, является ли значение счётчика `i` всё ещё меньше `5`. Если это так, операторы в теле цикла выполняются ещё раз. Завершающая операция снова увеличивает значение переменной на единицу. Шаги 2 и 3 повторяются до тех пор, пока условие `i < 5;` возвращает `true`.

1.2. Вывод значений массива

Чтобы вывести значения массива с помощью цикла `for`, нужно задействовать свойство массива `length`. Это поможет определить количество элементов в массиве и выполнить цикл такое же количество раз.
Приведённый ниже скрипт выведет на экран пять сообщений с названиями цветов:

```
var flowers = ["Rose", "Lily", "Tulip", "Jasmine", "Orchid"];
for (var i = 0; i < flowers.length; i++){
  alert(flowers[i] + " - это цветок.");
}
```
JavaScript

Если значение свойства `length` не изменяется в ходе выполнения цикла, можно сохранить его в локальной переменной, а затем использовать эту переменную в условном выражении. Таким образом можно повысить скорость выполнения цикла, так как значение свойства `length` будет извлекаться всего один раз за всё время работы цикла.

```
var flowers = ["Rose", "Lily", "Tulip", "Jasmine", "Orchid"], len = flowers.length;
for (var i = 0; i < len; i++){
  alert(flowers[i] + " - это цветок.");
}
```
JavaScript

2. Цикл for...in

Циклы `for...in` используются для обхода свойств объектов, не являющихся массивами. Такой обход также называется **перечислением**. При обходе рекомендуется использовать метод `hasOwnProperty()`, чтобы отфильтровать свойства, которые были унаследованы от прототипа.
Для примера создадим объект с помощью литерала объекта.

```
var user = {
  name: 'Alice',
  age: 25,
  country: 'Russia'
};

for (var prop in user) {
  console.log(prop + ": " + user[prop]);
}
```
JavaScript

Предположим, что в сценарии до или после создания объекта `user` прототип объекта `Object` был расширен дополнительным методом `clone()`.

```
if (typeof Object.prototype.clone === 'undefined') {
  Object.prototype.clone = function () {};
}
```
JavaScript

Так как цепочка наследования прототипа постоянно проверяется интерпретатором, то все объекты автоматически получают доступ к новому методу.

3. Цикл while

Цикл `while` - цикл с предварительной проверкой условного выражения. Инструкция внутри цикла (блок кода в фигурных скобках) будет выполняться в случае, если условное выражение вычисляется в `true`. Если первая проверка даст результат `false`, блок инструкций не выполнится ни разу.
После завершения итерации цикла условное выражение опять проверяется на истинность и процесс будет повторяться до тех пор, пока выражение не будет вычислено как `false`. В этом случае программа продолжит работу с первой строки, следующей непосредственно после цикла (если таковая имеется).
Данный цикл выведет на экран таблицу умножения для числа 3:

```
var i = 1;
var msg = '';
while (i < 10) {
  msg+= i + ' x 3 = ' + (i * 3) + '<br>';
  i++;
}
document.write(msg);
```
JavaScript

4. Цикл do...while

Цикл `do...while;` проверяет условие продолжения после выполнения цикла. В отличие от цикла `while`, в `do...while;` тело цикла выполняется как минимум один раз, так как условие проверяется в конце цикла, а не в начале. Данный цикл используется реже, чем `while`, так как на практике ситуация, когда требуется хотя бы однократное исполнение цикла, встречается редко.

```
var result = '';
var i = 0;
do {
   i += 1;
   result += i + ' ';
```

```
} while (i < 5);
document.write(result);
```
JavaScript

В следующем примере операторы внутри цикла выполняются один раз, даже если условие не выполняется.

```
var i = 10;
do {
    document.write(i + ' ');
    i++;
} while (i < 10);
```
JavaScript

5. Бесконечные циклы

При создании любого цикла можно создать бесконечный цикл, который никогда не завершится. Такой цикл может потенциально продолжать работать до тех пор, пока работает компьютер пользователя. Большинство современных браузеров могут обнаружить это и предложат пользователю остановить выполнение скрипта. Чтобы избежать создания бесконечного цикла, вы должны быть уверены, что заданное условие в какой-то момент вернёт `false`. Например, следующий цикл задаёт условие, которое никогда не возвращает ложь, так как переменная `i` никогда не будет меньше `10`:

```
for (var i = 25; i > 10; i++) {
document.write("Это предложение будет выводиться бесконечно...<br>");
}
```
JavaScript

6. Вложенные циклы

Цикл внутри другого цикла называется **вложенным**. При каждой итерации цикла вложенный цикл выполняется полностью. Вложенные циклы можно создавать с помощью цикла `for` и цикла `while`.

```
for (var count = 1; count < 3; count++) {
  document.write(count + ". Строка цикла<br>");
  for (var nestcount = 1; nestcount< 3; nestcount++) {
    document.write("Строка вложенного цикла<br>");
  }
}
```

7. Управление циклом

Циклом можно управлять с помощью операторов `break;` и `continue;`.

7.1. Оператор break;

Оператор `break;` завершает выполнение текущего цикла. Он используется в исключительных случаях, когда цикл не может выполняться по какой-то причине, например, если приложение обнаруживает ошибку. Чаще всего оператор `break;` является частью конструкции `if`.

Когда оператор `break;` используется без метки, он позволяет выйти из цикла или из инструкции `switch`. В следующем примере создаётся счётчик, значения которого должны изменяться от `1` до `99`, однако оператор `break` прерывает цикл после 14 итераций.

```
for (var i = 1; i < 100; i++) {
  if (i == 15) {
    break;
    }
  document.write(i);
  document.write(' <br>');
}
```

Для вложенных циклов оператор `break;` используется с меткой, с помощью которой завершается работа именованной инструкции. Метка позволяет выйти из любого блока кода.

Именованной инструкцией может быть любая инструкция, внешняя по отношению к оператору `break;`. В качестве метки может быть имя инструкции `if` или имя блока инструкций, заключенных в фигурные скобки только для присвоения метки этому блоку. Между ключевым словом `break;` и именем метки не допускается перевод строки.

```
outerloop:
  for(var i = 0; i < 10; i++) {
    innerloop:
      for(var j = 0; j < 10; j++) {
        if (j > 3) break; // Выход из самого внутреннего цикла
        if (i == 2) break innerloop; // То же самое
        if (i == 4) break outerloop; // Выход из внешнего цикла
        document.write("i = " + i + " j = " + j + "<br>");
      }
    }
document.write("FINAL i = " + i + " j = " + j + "<br>");
```

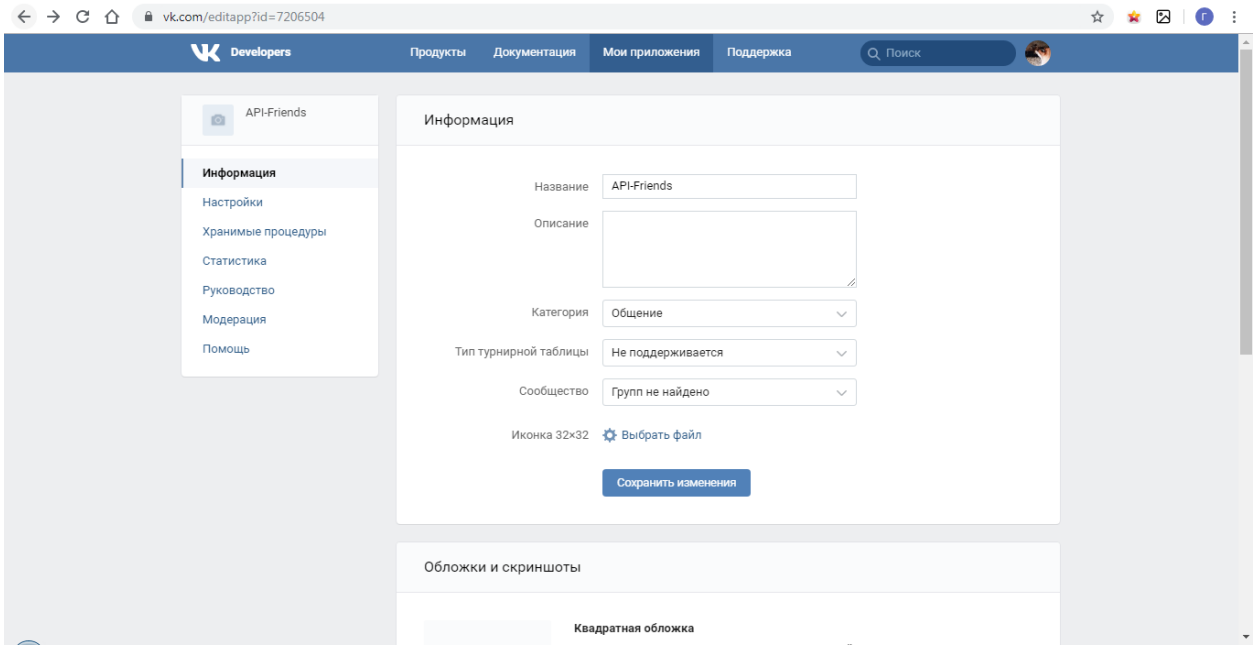
7.2. Оператор continue;

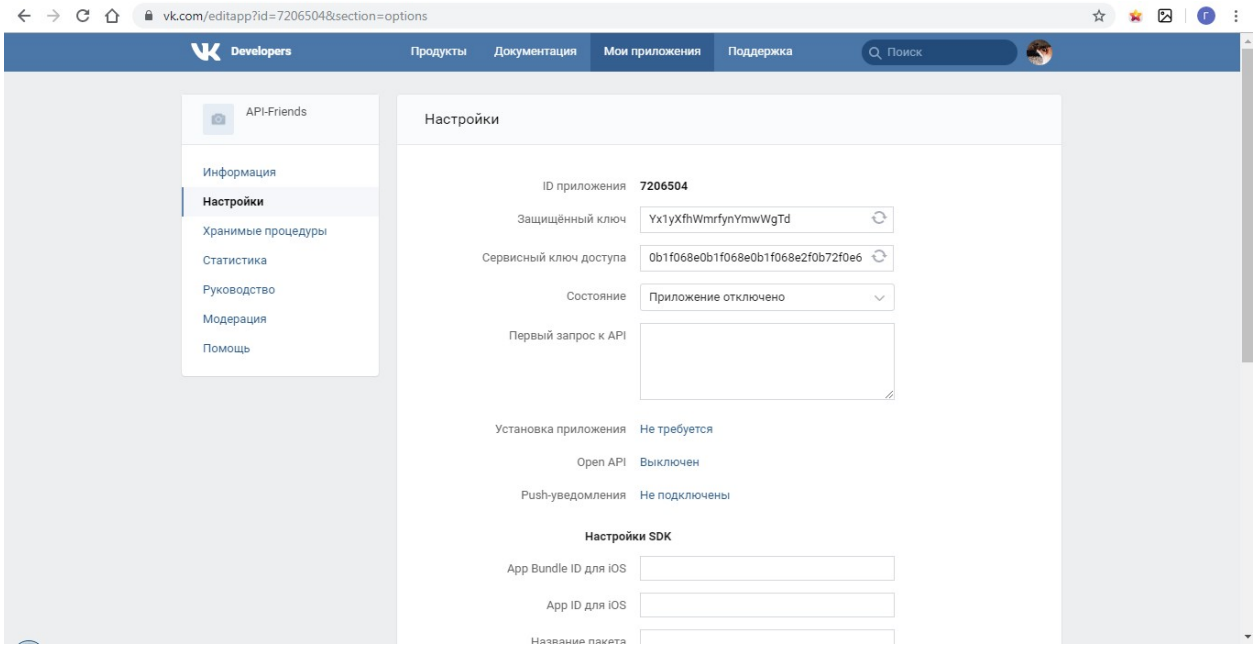
Оператор `continue;` останавливает текущую итерацию цикла и запускает новую итерацию. При этом, цикл `while` возвращается непосредственно к своему условию, а цикл `for` сначала вычисляет выражение инкремента, а затем возвращается к условию.
В этом примере на экран будут выведены все чётные числа:

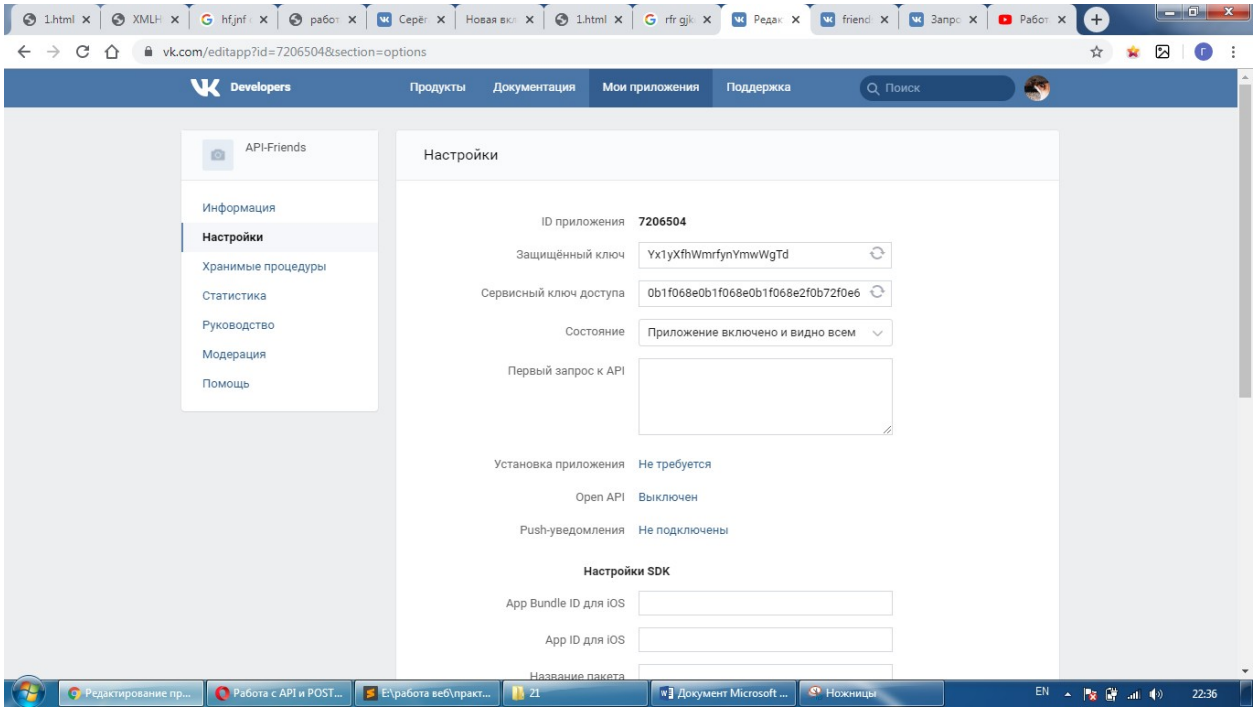
```
var i;
  for(i = 1; i <= 10; i++) {
    if (i % 2 !== 0) {
      continue;
    }
    document.write("<br><b>чётное число</b> = " + i);
  }
```

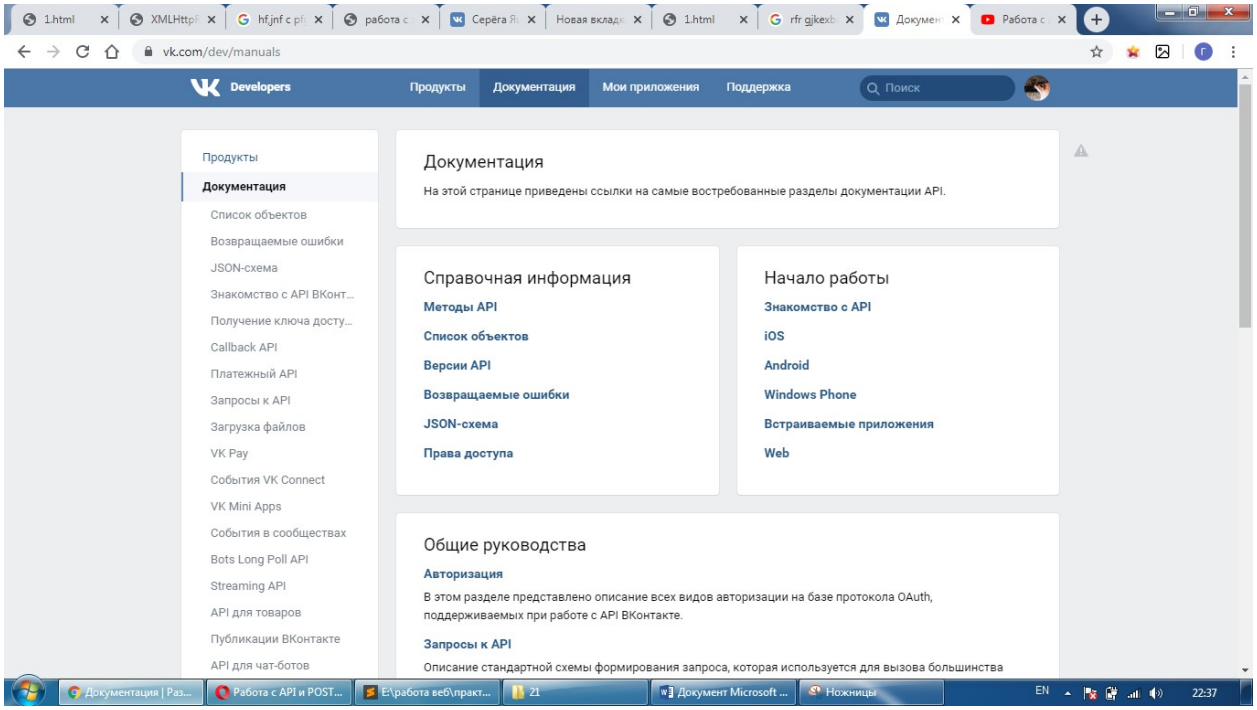
Оператор `continue;` также может применяться во вложенных циклах с меткой.

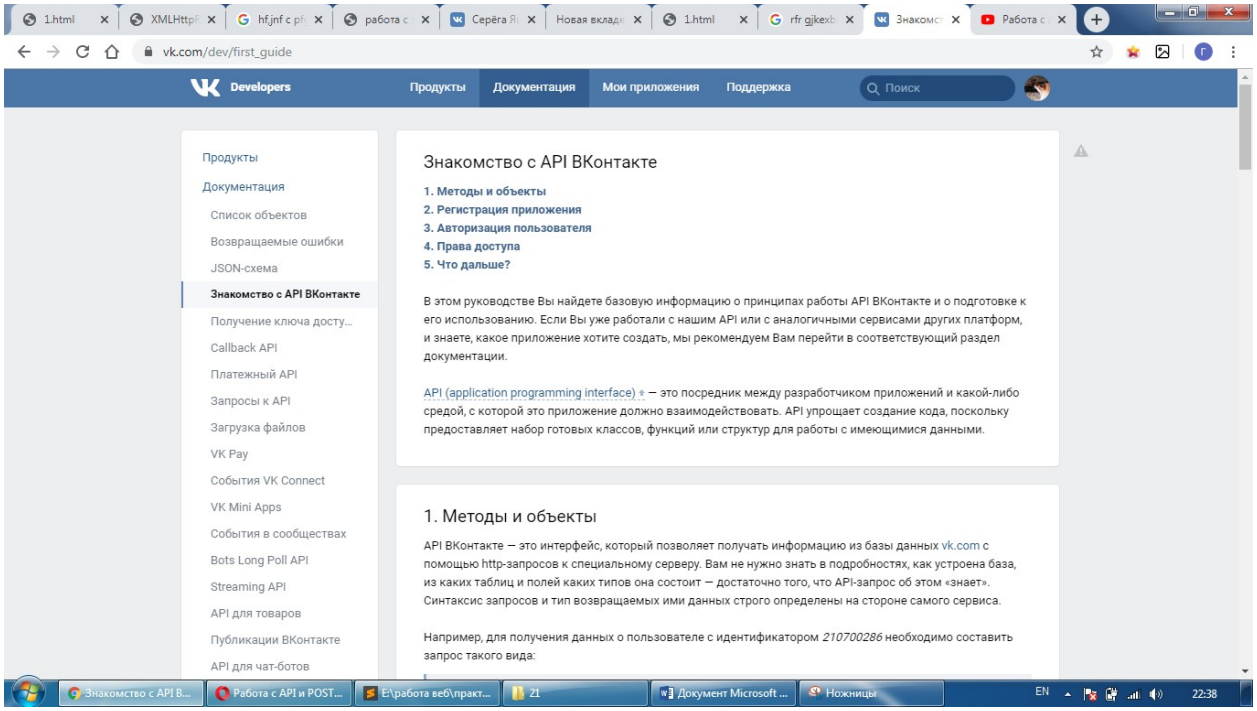
```
outerloop:
for (var i = 0; i < 3; i++)
{
  document.write("внешний цикл: "+i+"");
  for (var j = 0; j < 5; j++)
  {
    if (i == 1)
    break;
    if (j == 3)
    continue outerloop;
    document.write("вложенный цикл: "+j+"");
   }
}
document.write("Все циклы выполнены"+"");
```

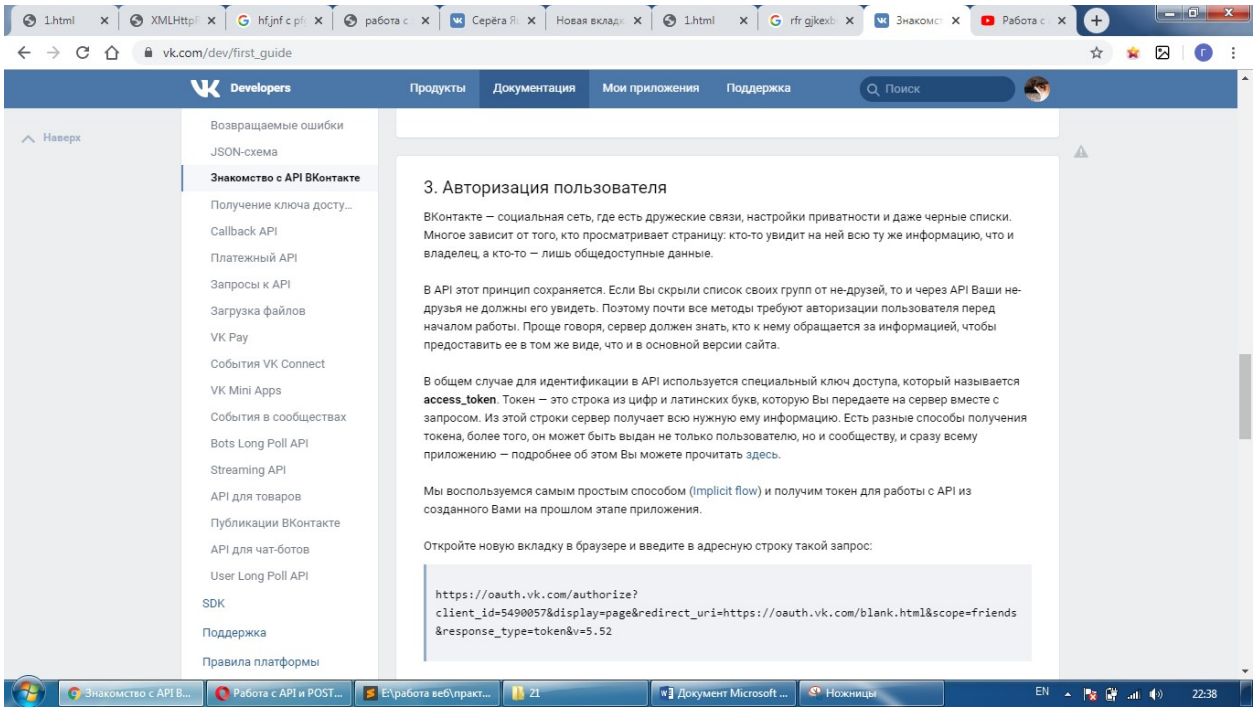

vk.com/editapp?id=7206504
Developers
Продукты
Документация
Мои приложения
Поддержка
Поиск
API-Friends
Информация
Настройки
Хранимые процедуры
Статистика
Руководство
Модерация
Помощь
Информация
Название
API-Friends
Описание
Категория
Общение
Тип турнирной таблицы
Не поддерживается
Сообщество
Групп не найдено
Иконка 32×32
Выбрать файл
Сохранить изменения
Обложки и скриншоты
Квадратная обложка

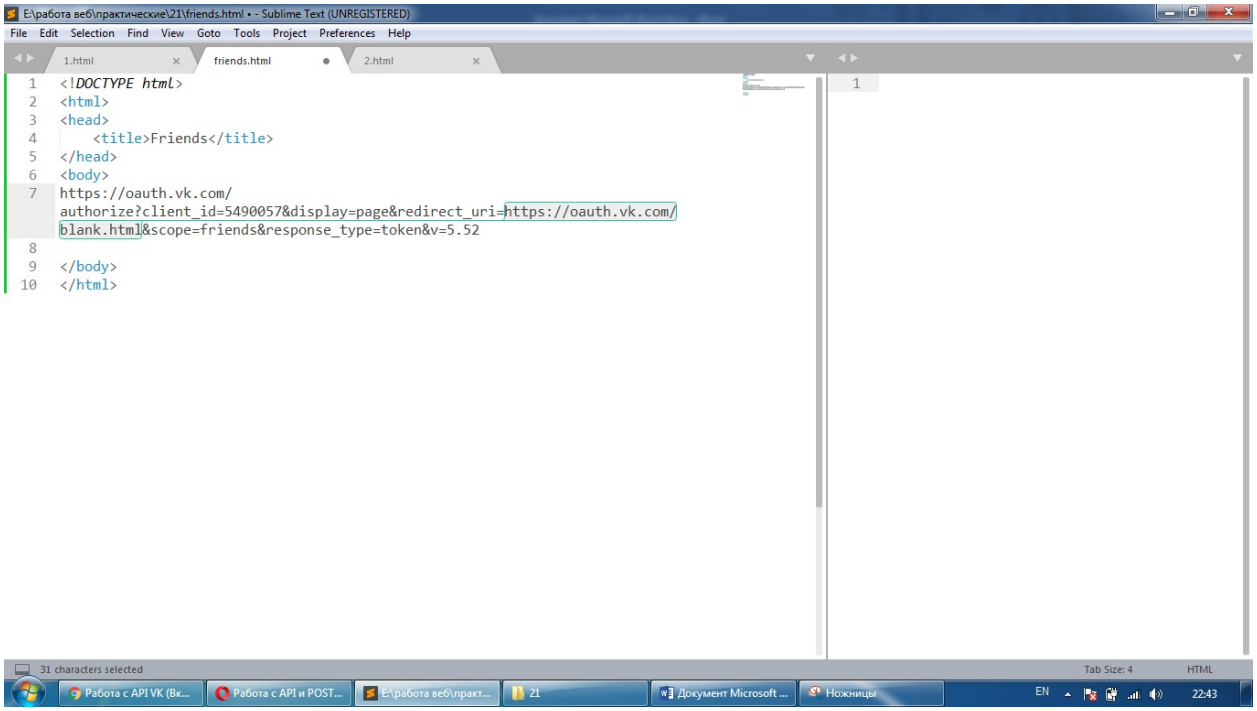

vk.com/editapp?id=7206504&section=options
Developers
Продукты
Документация
Мои приложения
Поддержка
Поиск
API-Friends
Информация
Настройки
Хранимые процедуры
Статистика
Руководство
Модерация
Помощь
Настройки
ID приложения
7206504
Защищённый ключ
Yx1yXfhWmrfynYmwWgTd
Сервисный ключ доступа
0b1f068e0b1f068e0b1f068e2f0b72f0e6
Состояние
Приложение отключено
Первый запрос к API
Установка приложения
Не требуется
Open API
Выключен
Push-уведомления
Не подключены
Настройки SDK
App Bundle ID для iOS
App ID для iOS

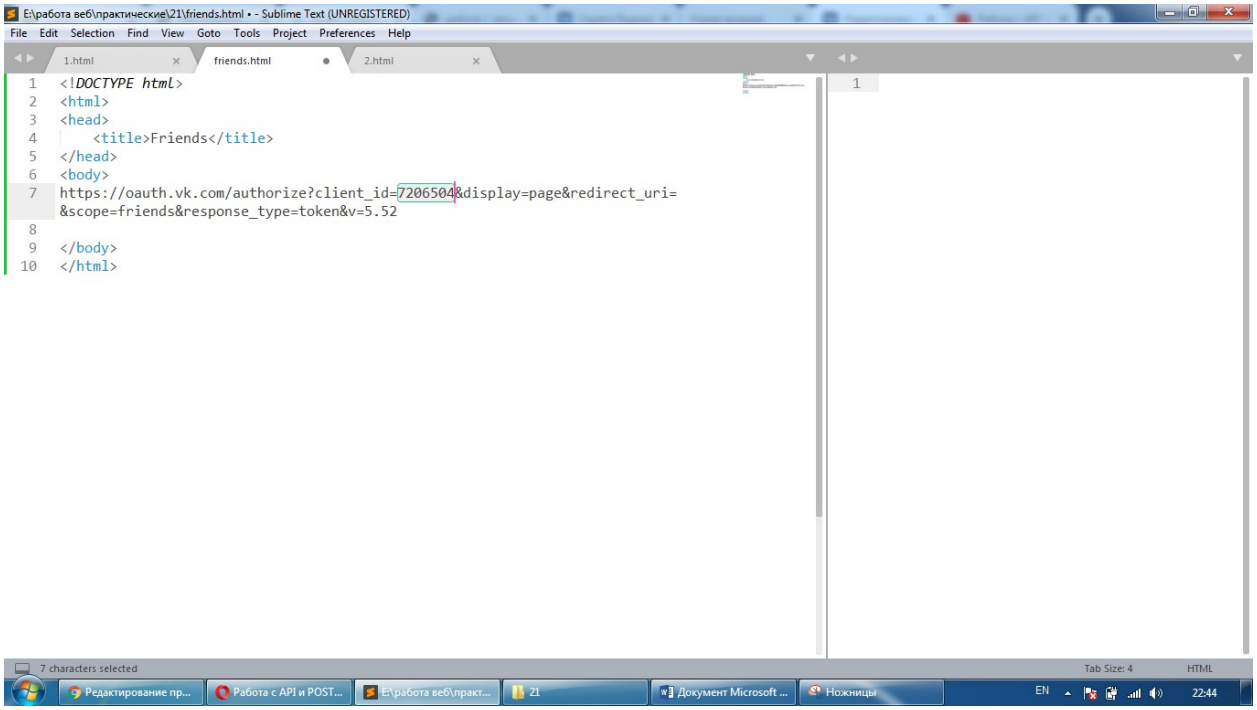

vk.com/editapp?id=7206504&section=options
Developers
Продукты
Документация
Мои приложения
Поддержка
Поиск
API-Friends
Информация
Настройки
Хранимые процедуры
Статистика
Руководство
Модерация
Помощь
Настройки
ID приложения 7206504
Защищённый ключ Yx1yXfhWmrfynYmwWgTd
Сервисный ключ доступа 0b1f068e0b1f068e0b1f068e2f0b72f0e6
Состояние Приложение включено и видно всем
Первый запрос к API
Установка приложения Не требуется
Open API Выключен
Push-уведомления Не подключены
Настройки SDK
App Bundle ID для iOS
App ID для iOS
Название пакета
EN
22:36

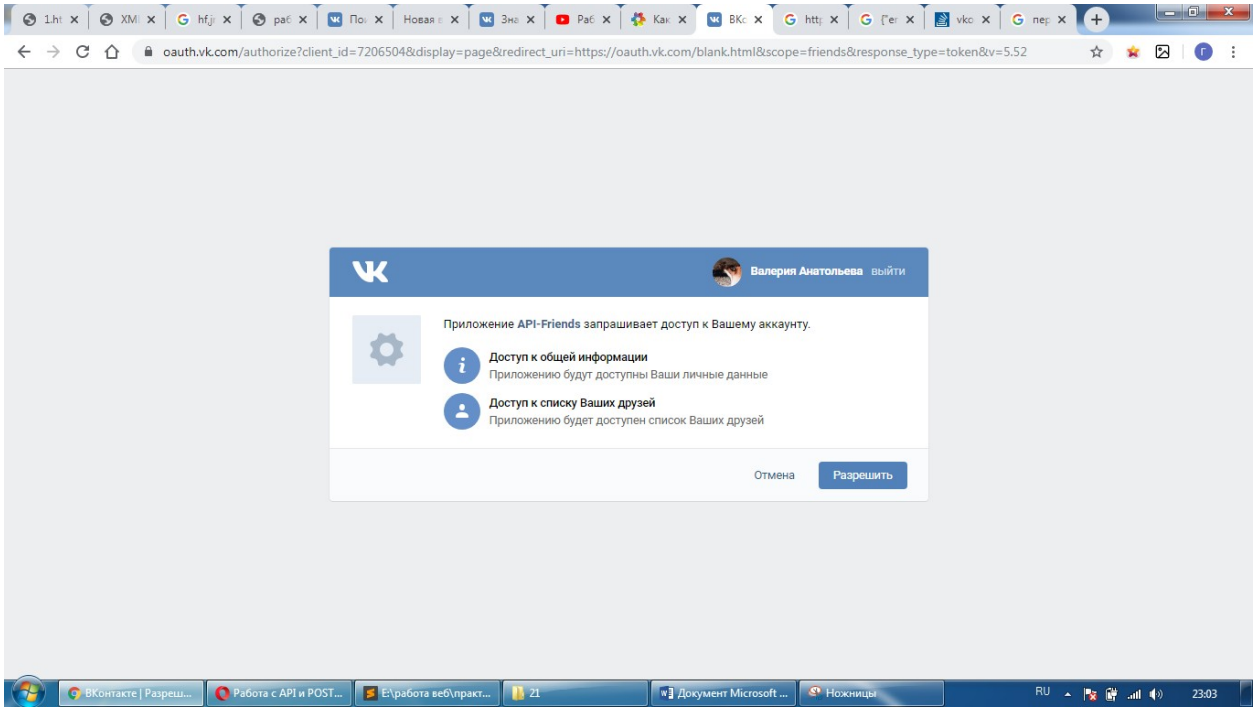

vk.com/dev/manuals
Developers
Продукты
Документация
Мои приложения
Поддержка
Поиск
Продукты
Документация
Список объектов
Возвращаемые ошибки
JSON-схема
Знакомство с API ВКонт...
Получение ключа досту...
Callback API
Платежный API
Запросы к API
Загрузка файлов
VK Pay
События VK Connect
VK Mini Apps
События в сообществах
Bots Long Poll API
Streaming API
API для товаров
Публикации ВКонтакте
API для чат-ботов
Документация
На этой странице приведены ссылки на самые востребованные разделы документации API.
Справочная информация
Методы API
Список объектов
Версии API
Возвращаемые ошибки
JSON-схема
Права доступа
Начало работы
Знакомство с API
iOS
Android
Windows Phone
Встраиваемые приложения
Web
Общие руководства
Авторизация
В этом разделе представлено описание всех видов авторизации на базе протокола OAuth, поддерживаемых при работе с API ВКонтакте.
Запросы к API
Описание стандартной схемы формирования запроса, которая используется для вызова большинства
EN
22:37


vk.com/dev/first_guide
Developers
Продукты
Документация
Мои приложения
Поддержка
Поиск
Продукты
Документация
Список объектов
Возвращаемые ошибки
JSON-схема
Знакомство с API ВКонтакте
Получение ключа досту...
Callback API
Платежный API
Запросы к API
Загрузка файлов
VK Pay
События VK Connect
VK Mini Apps
События в сообществах
Bots Long Poll API
Streaming API
API для товаров
Публикации ВКонтакте
API для чат-ботов
Знакомство с API ВКонтакте
1. Методы и объекты
2. Регистрация приложения
3. Авторизация пользователя
4. Права доступа
5. Что дальше?
В этом руководстве Вы найдете базовую информацию о принципах работы API ВКонтакте и о подготовке к его использованию. Если Вы уже работали с нашим API или с аналогичными сервисами других платформ, и знаете, какое приложение хотите создать, мы рекомендуем Вам перейти в соответствующий раздел документации.
API (application programming interface) — это посредник между разработчиком приложений и какой-либо средой, с которой это приложение должно взаимодействовать. API упрощает создание кода, поскольку предоставляет набор готовых классов, функций или структур для работы с имеющимися данными.
1. Методы и объекты
API ВКонтакте — это интерфейс, который позволяет получать информацию из базы данных vk.com с помощью http-запросов к специальному серверу. Вам не нужно знать в подробностях, как устроена база, из каких таблиц и полей каких типов она состоит — достаточно того, что API-запрос об этом «знает». Синтаксис запросов и тип возвращаемых ими данных строго определены на стороне самого сервиса.
Например, для получения данных о пользователе с идентификатором 210700286 необходимо составить запрос такого вида:
Знакомство с API В...
Работа с API и POST...
E:\работа веб\практ...
21
Документ Microsoft ...
Ножницы
EN
22:38

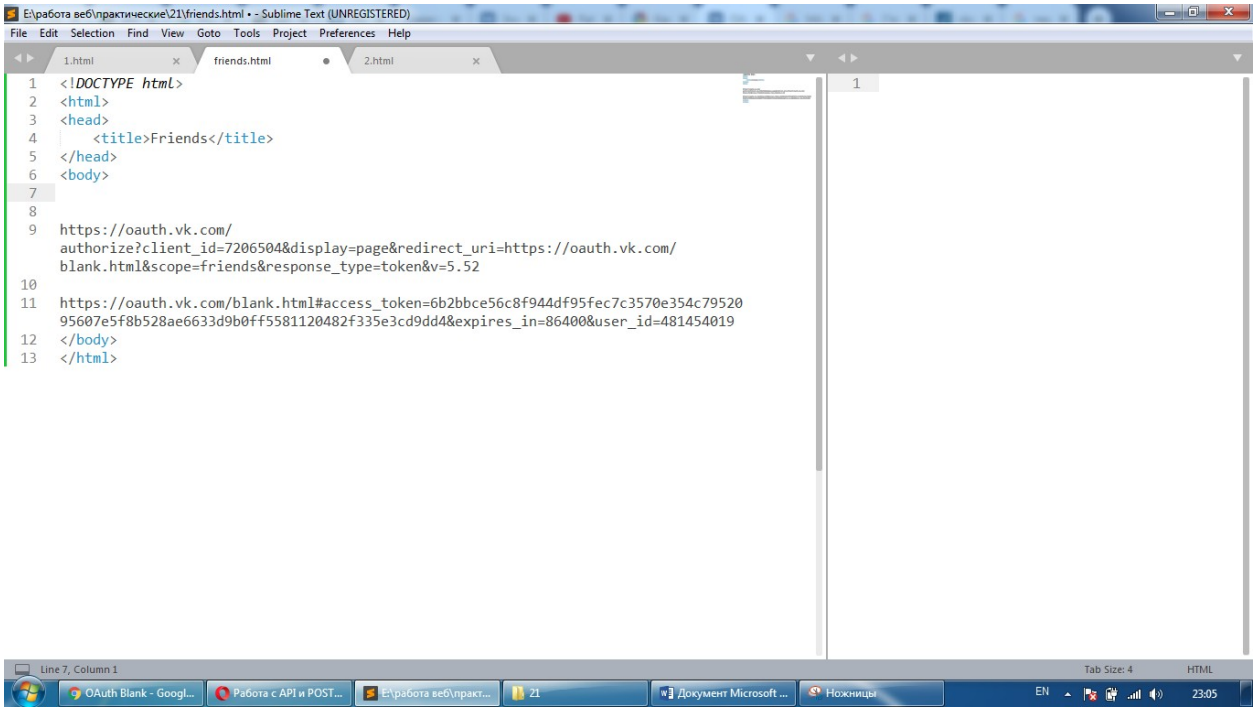

vk.com/dev/first_guide
Developers
Продукты
Документация
Мои приложения
Поддержка
Поиск
Наверх
Возвращаемые ошибки
JSON-схема
Знакомство с API ВКонтакте
Получение ключа досту...
Callback API
Платежный API
Запросы к API
Загрузка файлов
VK Pay
События VK Connect
VK Mini Apps
События в сообществах
Bots Long Poll API
Streaming API
API для товаров
Публикации ВКонтакте
API для чат-ботов
User Long Poll API
SDK
Поддержка
Правила платформы
3. Авторизация пользователя
ВКонтакте — социальная сеть, где есть дружеские связи, настройки приватности и даже черные списки. Многое зависит от того, кто просматривает страницу: кто-то увидит на ней всю ту же информацию, что и владелец, а кто-то — лишь общедоступные данные.
В API этот принцип сохраняется. Если Вы скрыли список своих групп от не-друзей, то и через API Ваши не-друзья не должны его увидеть. Поэтому почти все методы требуют авторизации пользователя перед началом работы. Проще говоря, сервер должен знать, кто к нему обращается за информацией, чтобы предоставить ее в том же виде, что и в основной версии сайта.
В общем случае для идентификации в API используется специальный ключ доступа, который называется access_token. Токен — это строка из цифр и латинских букв, которую Вы передаете на сервер вместе с запросом. Из этой строки сервер получает всю нужную ему информацию. Есть разные способы получения токена, более того, он может быть выдан не только пользователю, но и сообществу, и сразу всему приложению — подробнее об этом Вы можете прочитать здесь.
Мы воспользуемся самым простым способом (Implicit flow) и получим токен для работы с API из созданного Вами на прошлом этапе приложения.
Откройте новую вкладку в браузере и введите в адресную строку такой запрос:
https://oauth.vk.com/authorize?
client_id=5490057&display=page&redirect_uri=https://oauth.vk.com/blank.html&scope=friends
&response_type=token&v=5.52
Знакомство с API В...
Работа с API и POST...
E:\работа веб\практ...
21
Документ Microsoft ...
Ножницы
EN
22:38

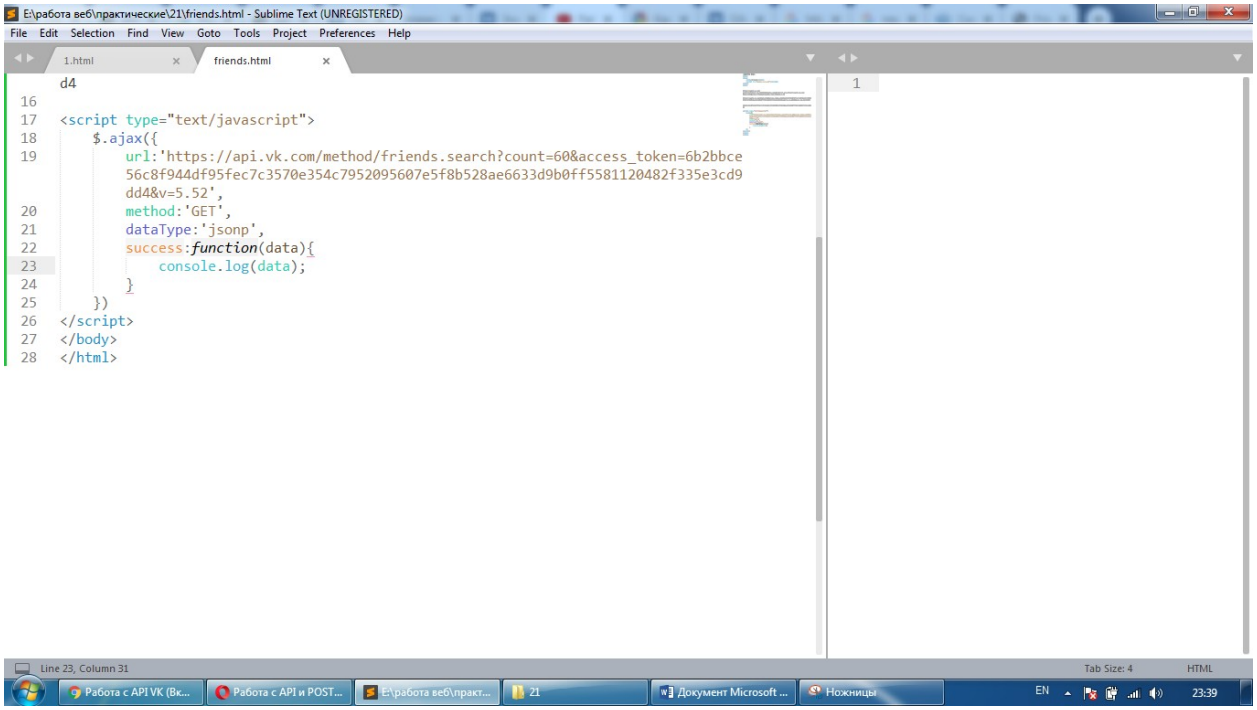

E:\работа веб\практические\21\friends.html • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1.html
friends.html
2.html
<!DOCTYPE html>
<html>
<head>
<title>Friends</title>
</head>
<body>
https://oauth.vk.com/
authorize?client_id=5490057&display=page&redirect_uri=https://oauth.vk.com/
blank.html&scope=friends&response_type=token&v=5.52
</body>
</html>
31 characters selected
Tab Size: 4
HTML
Работа с API VK (Вк...
Работа с API и POST...
E:\работа веб\практ...
21
Документ Microsoft ...
Ножницы
EN
22:43

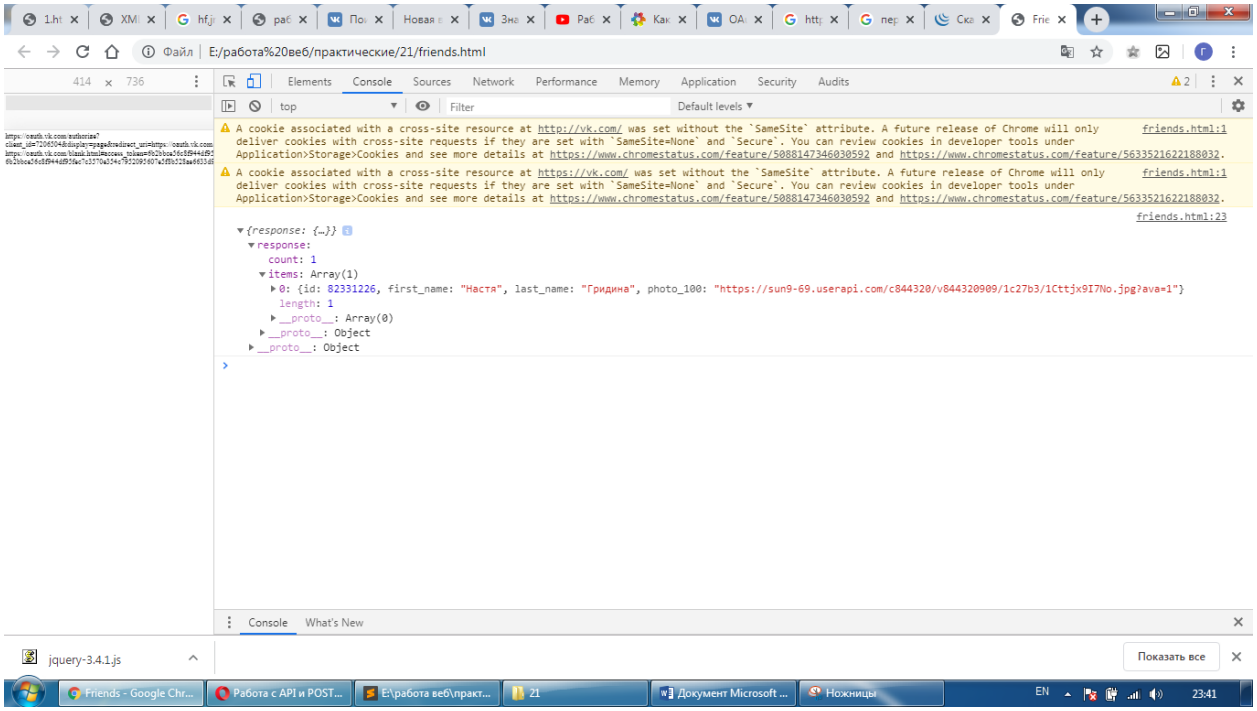

E:\работа веб\практические\21\friends.html • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1.html
friends.html
2.html
<!DOCTYPE html>
<html>
<head>
<title>Friends</title>
</head>
<body>
https://oauth.vk.com/authorize?client_id=7206504&display=page&redirect_uri=
&scope=friends&response_type=token&v=5.52
</body>
</html>
7 characters selected
Tab Size: 4
HTML
Редактирование пр...
Работа с API и POST...
E:\работа веб\практ...
21
Документ Microsoft ...
Ножницы
EN
22:44

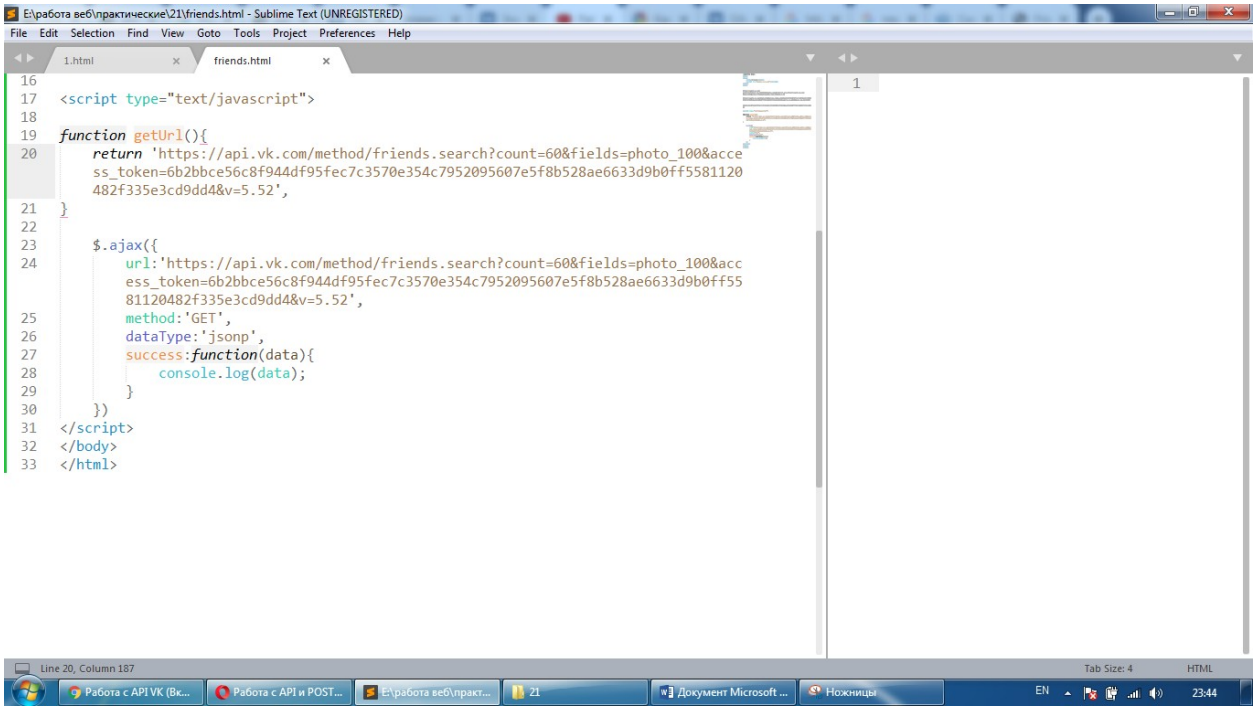


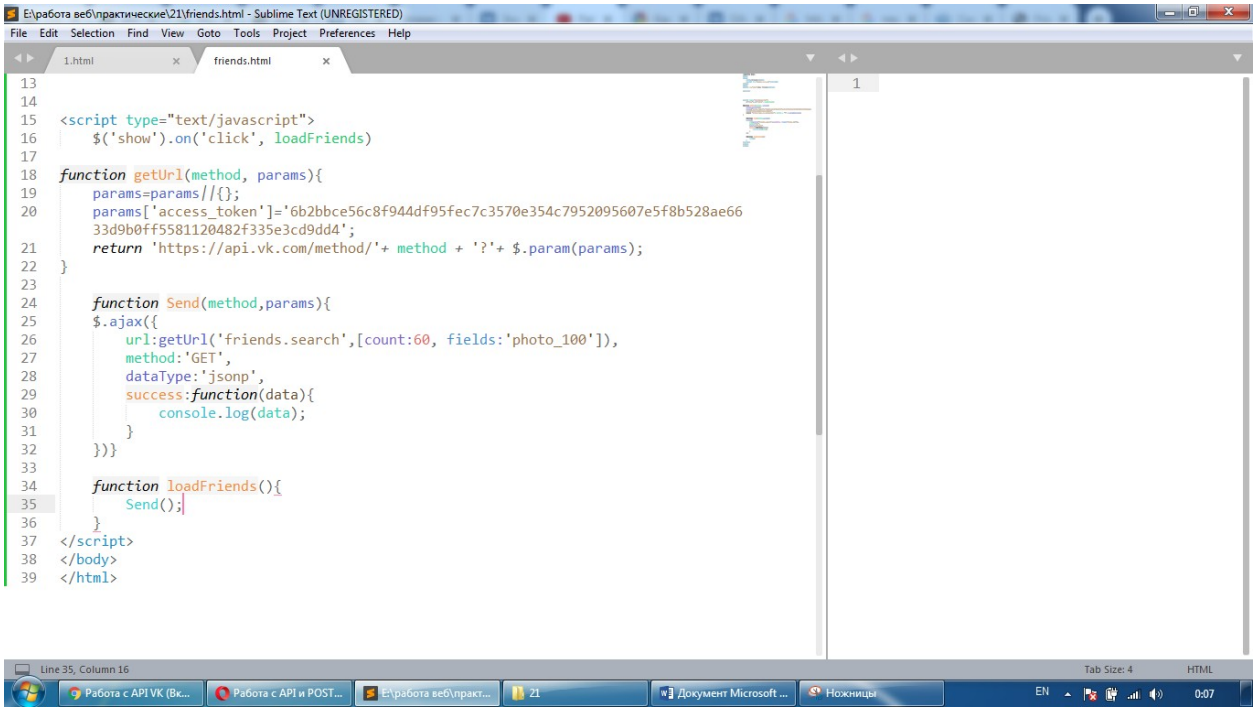
Пожалуйста, **не копируйте** данные из адресной строки для сторонних сайтов. Таким образом Вы можете **потерять доступ** к Вашему аккаунту.

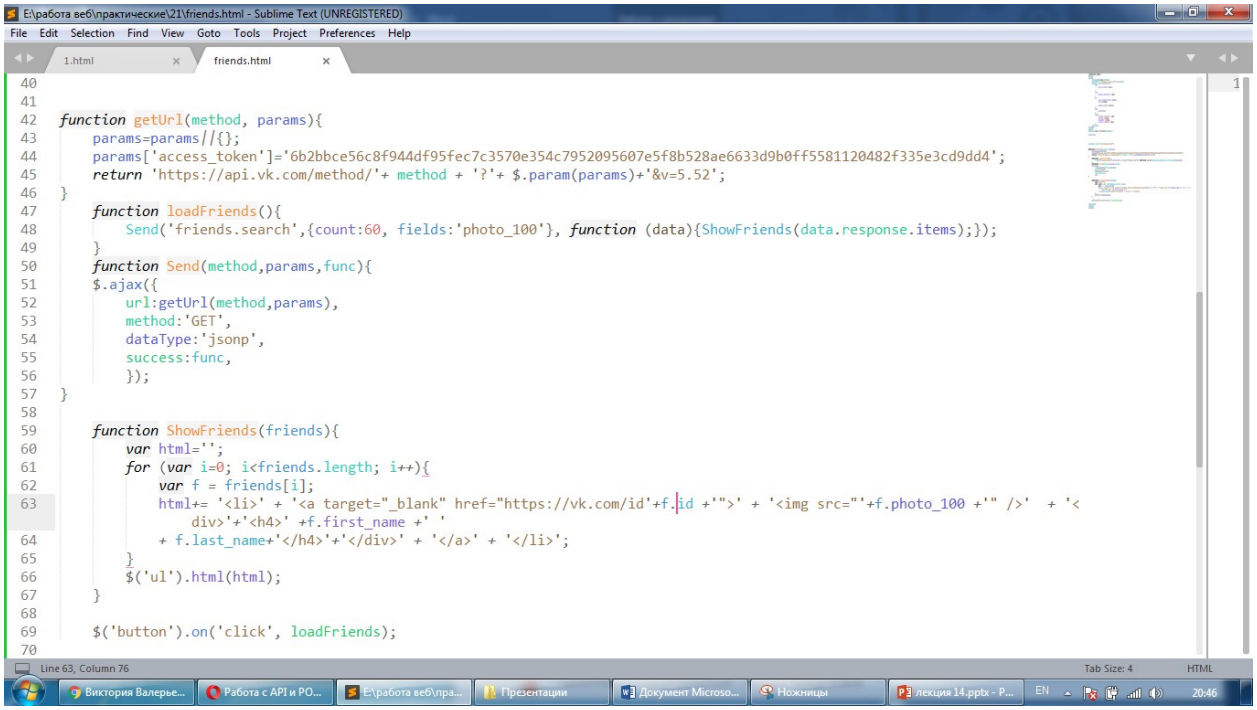

E:\работа веб\практические\21\friends.html • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1.html
friends.html
2.html
<!DOCTYPE html>
<html>
<head>
<title>Friends</title>
</head>
<body>
https://oauth.vk.com/
authorize?client_id=7206504&display=page&redirect_uri=https://oauth.vk.com/
blank.html&scope=friends&response_type=token&v=5.52
https://oauth.vk.com/blank.html#access_token=6b2bbce56c8f944df95fec7c3570e354c79520
95607e5f8b528ae6633d9b0ff5581120482f335e3cd9dd4&expires_in=86400&user_id=481454019
</body>
</html>
Line 7, Column 1
Tab Size: 4
HTML
OAuth Blank - Googl...
Работа с API и POST...
E:\работа веб\практ...
21
Документ Microsoft ...
Ножницы
EN
23:05

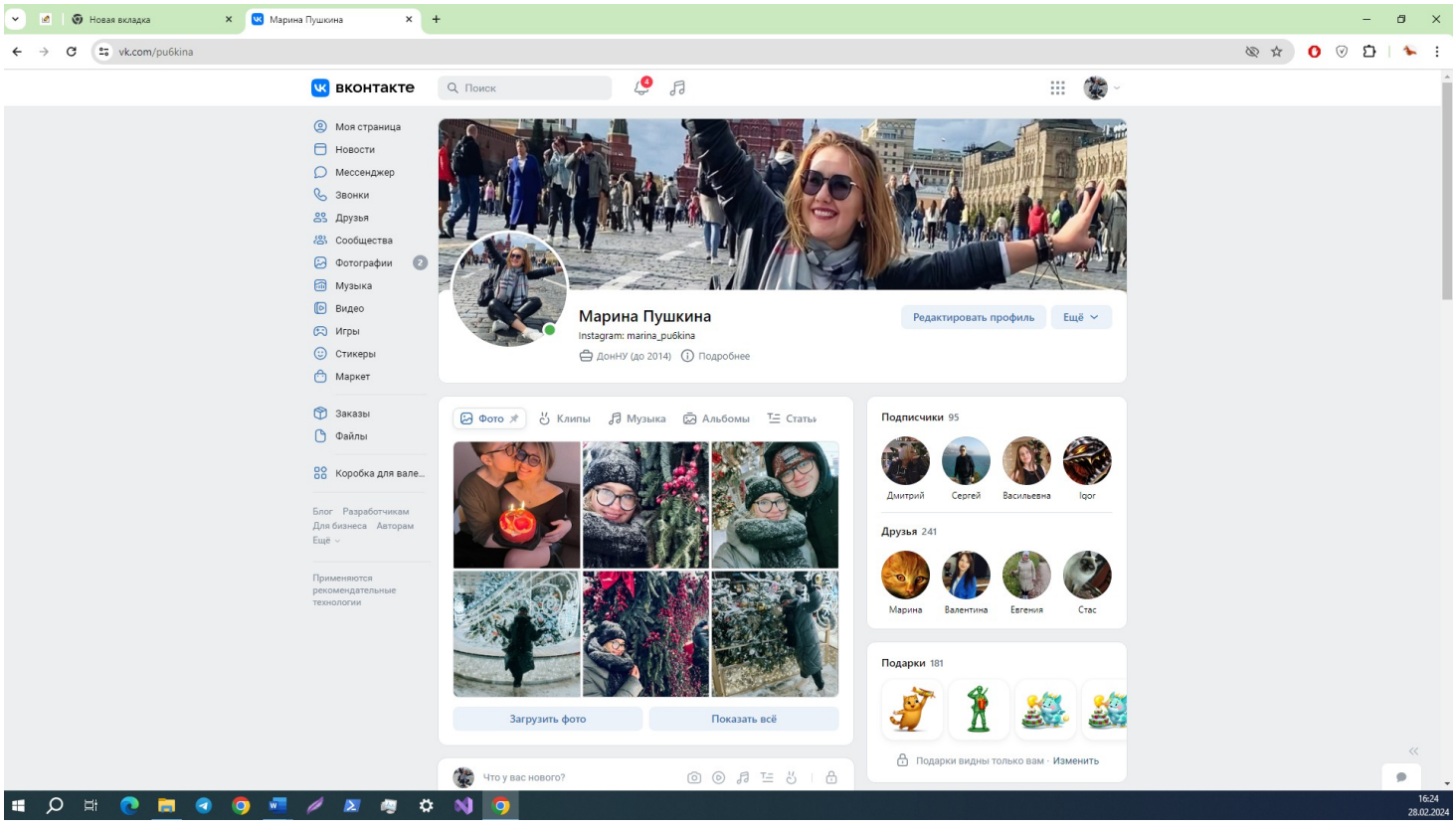

E:\работа веб\практические\21\friends.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1.html
friends.html
d4
<script type="text/javascript">
$.ajax({
url:'https://api.vk.com/method/friends.search?count=60&access_token=6b2bbce
56c8f944df95fec7c3570e354c7952095607e5f8b528ae6633d9b0ff5581120482f335e3cd9
dd4&v=5.52',
method:'GET',
dataType:'jsonp',
success:function(data){
console.log(data);
}
})
</script>
</body>
</html>
Line 23, Column 31
Tab Size: 4
HTML
Работа с API VK (Вк...
Работа с API и POST...
E:\работа веб\практ...
21
Документ Microsoft ...
Ножницы
EN
23:39


Файл | E:/работа%20веб/практические/21/friends.html
Elements Console Sources Network Performance Memory Application Security Audits
top Filter Default levels
A cookie associated with a cross-site resource at http://vk.com/ was set without the `SameSite` attribute. A future release of Chrome will only deliver cookies with cross-site requests if they are set with `SameSite=None` and `Secure`. You can review cookies in developer tools under Application>Storage>Cookies and see more details at https://www.chromestatus.com/feature/5088147346030592 and https://www.chromestatus.com/feature/5633521622188032.
friends.html:1
A cookie associated with a cross-site resource at https://vk.com/ was set without the `SameSite` attribute. A future release of Chrome will only deliver cookies with cross-site requests if they are set with `SameSite=None` and `Secure`. You can review cookies in developer tools under Application>Storage>Cookies and see more details at https://www.chromestatus.com/feature/5088147346030592 and https://www.chromestatus.com/feature/5633521622188032.
friends.html:1
friends.html:23
{response: {…}}
response:
count: 1
items: Array(1)
0: {id: 82331226, first_name: "Настя", last_name: "Гридина", photo_100: "https://sun9-69.userapi.com/c844320/v844320909/1c27b3/1Cttjx9I7No.jpg?ava=1"}
length: 1
__proto__: Array(0)
__proto__: Object
__proto__: Object
Console What's New
jquery-3.4.1.js
Показать все
Friends - Google Chr...
Работа с API и POST...
E:\работа веб\практ...
21
Документ Microsoft ...
Ножницы
EN
23:41


E:\работа веб\практические\21\friends.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1.html
friends.html
<script type="text/javascript">
function getUrl(){
return 'https://api.vk.com/method/friends.search?count=60&fields=photo_100&access_token=6b2bbce56c8f944df95fec7c3570e354c7952095607e5f8b528ae6633d9b0ff5581120482f335e3cd9dd4&v=5.52',
}
$.ajax({
url:'https://api.vk.com/method/friends.search?count=60&fields=photo_100&access_token=6b2bbce56c8f944df95fec7c3570e354c7952095607e5f8b528ae6633d9b0ff5581120482f335e3cd9dd4&v=5.52',
method:'GET',
dataType:'jsonp',
success:function(data){
console.log(data);
}
})
</script>
</body>
</html>
Line 20, Column 187
Tab Size: 4
HTML
Работа с API VK (Вк...
Работа с API и POST...
E:\работа веб\практ...
21
Документ Microsoft ...
Ножницы
EN
23:44


E:\работа веб\практические\21\friends.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1.html
friends.html
13
14
15 <script type="text/javascript">
16 $('show').on('click', loadFriends)
17
18 function getUrl(method, params){
19 params=params||{};
20 params['access_token']='6b2bbce56c8f944df95fec7c3570e354c7952095607e5f8b528ae66
33d9b0ff5581120482f335e3cd9dd4';
21 return 'https://api.vk.com/method/'+ method + '?'+ $.param(params);
22 }
23
24 function Send(method,params){
25 $.ajax({
26 url:getUrl('friends.search',[count:60, fields:'photo_100']),
27 method:'GET',
28 dataType:'jsonp',
29 success:function(data){
30 console.log(data);
31 }
32 })}
33
34 function loadFriends(){
35 Send();
36 }
37 </script>
38 </body>
39 </html>
1
Line 35, Column 16
Tab Size: 4
HTML
Работа с API VK (Вк...
Работа с API и POST...
E:\работа веб\практ...
21
Документ Microsoft ...
Ножницы
EN
0:07


E:\работа веб\практические\21\friends.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1.html
friends.html
40
41
42 function getUrl(method, params){
43 params=params||{};
44 params['access_token']='6b2bbce56c8f944df95fec7c3570e354c7952095607e5f8b528ae6633d9b0ff5581120482f335e3cd9dd4';
45 return 'https://api.vk.com/method/'+ method + '?'+ $.param(params)+'&v=5.52';
46 }
47 function loadFriends(){
48 Send('friends.search',{count:60, fields:'photo_100'}, function (data){ShowFriends(data.response.items);});
49 }
50 function Send(method,params,func){
51 $.ajax({
52 url:getUrl(method,params),
53 method:'GET',
54 dataType:'jsonp',
55 success:func,
56 });
57 }
58
59 function ShowFriends(friends){
60 var html='';
61 for (var i=0; i<friends.length; i++){
62 var f = friends[i];
63 html+= '<li>' + '' + '' + '<
div>'+'<h4>' +f.first_name +' '
64 + f.last_name+'</h4>'+' ' + '' + '
';
65 }
66 $('ul').html(html);
67 }
68
69 $('button').on('click', loadFriends);
70
1
Line 63, Column 76
Tab Size: 4
HTML
Виктория Валерье...
Работа с API и РО...
E:\работа веб\пра...
Презентации
Документ Microso...
Ножницы
Лекция 14.pptx - P...
EN
20:46



Лекция 16. Обработка ошибок. Исключения.

Поговорим об ошибках в JavaScript и о том, как их обрабатывать. Ошибки бывают в основном 2-х типов — это синтаксические и логические. К синтаксическим можно отнести ошибки в имени переменных, функций, ошибки в синтаксисе кода. В принципе такие ошибки легко отловить через консоль браузера.

А вот логические ошибки с ними все не так просто потому, что они приводят к неправильному выполнению кода программы. Поэтому для их устранения

потребуется отладка программы, чтобы понять, что собственно происходит на каждом шаге скрипта. Мы же с вами рассмотрим в основном локализацию синтаксических ошибок с помощью конструкции try...catch.

Конструкция перехвата ошибок try...catch

Конструкция **try..catch** состит из 2-х блоков: try, и затем catch.

```
try {
  // код ...
} catch (err) {
  // обработка ошибки

}
```

Работает эта конструкция таким образом:

1. Выполняется код внутри блока try, так называемой ловушки.
2. Если в нём не встречаются ошибки, то блок catch(err) игнорируется.
3. А вот, если в нём возникнет ошибка, то выполнение try будет прервано на ошибке, и управление передается в начало блока catch(err). При этом переменная err (можно выбрать любое другое название) будет содержать объект ошибки с подробнейшей информацией о произошедшей ошибке.

Поэтому при ошибке в try скрипт не останавливается, и даже более того мы имеем возможность обработать ошибку внутри блока catch.

Рассмотрим это на примерах.

Пример без ошибок:при запуске сработают alert(1)и(2): А вот пример с ошибкой:при запуске сработают (1)и(3):

```
try {

  alert('Начало блока try');  // (1) <--

  // ...код без ошибок

  alert('Конец блока try');   // (2) <--

} catch(err) {

  alert('Catch игнорируется, так как нет ошибок'); // (3)

}
```

```
try {

  alert('Начало блока try');  // (1) <--

  lalala; // ошибка, переменная не определена!

  alert('Конец блока try (никогда не выполнится)');  // (2)

} catch(err) {

  alert(`Возникла ошибка!`); // (3) <--

}
```

В случае, если грубо нарушена структура кода, не закрыта фигурная скобка или где-то стоит лишняя запятая, то вам никакой try..catch не поможет. Это синтаксические ошибки, интерпретатор такой код просто не понимает. JavaScript-движок сначала читает код, а затем исполняет его. Ошибки, которые возникают во время фазы чтения, называются ошибками парсинга. Их нельзя обработать (изнутри этого кода), потому что движок не понимает код.

Таким образом, try..catch может обрабатывать только ошибки, которые возникают в корректном коде. Такие ошибки называют «ошибками во время выполнения», а иногда «исключениями».

Важное замечание try..catch может работать только в синхронном коде

Ошибку, которая может произойти в коде,который будет выполняться через время например в setTimeout, try..catch не поймает:

На момент запуска функции, назначенной через setTimeout, этот код уже завершится, и интерпретатор выйдет из блока try..catch.

Для того чтобы Чтобы поймать ошибку внутри функции из setTimeout, и try..catch должен быть в той же функции.

```
try {
  setTimeout(function() {
    noSuchVariable; // скрипт упадёт тут
  }, 1000);
} catch (e) {
  alert( "не сработает" );
}
```

```
setTimeout(function() {
  try {
    noSuchVariable; // try..catch обрабатывает ошибку!
  } catch {
    alert( "ошибка поймана!" );
  }
}, 1000);
```

Объект ошибки

Когда возникает ошибка, JavaScript генерирует объект, содержащий её детали. Затем этот объект передаётся как аргумент в блок `catch`:
Для всех встроенных ошибок этот объект имеет два основных свойства:
`name`
Имя ошибки. Например, для неопределённой переменной это `"ReferenceError"`.
`message`
Текстовое сообщение о деталях ошибки.
В большинстве окружений доступны и другие, нестандартные свойства. Одно из самых широко используемых и поддерживаемых – это:**`stack`**
Текущий стек вызова: строка, содержащая информацию о последовательности вложенных вызовов, которые привели к ошибке. Используется в целях отладки.
Если нам не нужны детали ошибки, в `catch` можно её пропустить:

Генерация своих ошибок

Оператор throw

Данный оператор throw генерирует ошибку.
Синтаксис: throw <объект ошибки>.
Технически в качестве объекта ошибки вы можете передать что угодно, это может быть даже и не объект, а число или строка.
В качестве конструктора ошибок вы можете использовать встроенный конструктор: new Error(message) или любой другой.
В JavaScript также встроен ряд конструкторов для стандартных ошибок: SyntaxError, ReferenceError, RangeError и некоторые другие.
В данном примере мы используем конструктор new SyntaxError(message). Он создаёт ошибку того же типа, что и JSON.parse.
В нашем случае отсутствие свойства name – это ошибка, ведь пользователи должны иметь имена.
В строке (*) оператор throw генерирует ошибку SyntaxError с сообщением message. Точно такого же вида, как генерирует сам JavaScript. Выполнение блока try немедленно останавливается, и поток управления прыгает в catch.
Теперь блок catch становится единственным местом для обработки всех ошибок: и для JSON.parse и для других случаев.
В примере выше мы использовали try..catch для обработки некорректных данных. А что, если в блоке try {...}возникнет другая неожиданная ошибка?
В нашем случае try..catch предназначен для выявления ошибок, связанных с некорректными данными. Но по своей природе catch получает все свои ошибки из try. Здесь он получает неожиданную ошибку, но всё также показывает то же самое сообщение "JSON Error". Это неправильно и затрудняет отладку кода.
Есть простое правило:
Блок catch должен обрабатывать только те ошибки, которые ему известны, и «пробрасывать» все остальные.
Техника «проброс исключения» выглядит так:
Блок catch получает все ошибки.
В блоке catch(err) {...} мы анализируем объект ошибки err.
Если мы не знаем как её обработать, тогда делаем throw err.
В коде ниже мы используем проброс исключения, catch обрабатывает только SyntaxError:

Ошибка в строке (*) из блока catch «выпадает наружу» и может быть поймана другой внешней конструкцией try..catch (если есть), или «убьёт» скрипт.

Таким образом, блок catch фактически обрабатывает только те ошибки, с которыми он знает, как справляться, и пропускает остальные.

Секция finally

Конструкция try..catch может содержать ещё одну секцию: finally.
Если секция есть, то она выполняется в любом случае:
после try, если не было ошибок,
после catch, если ошибки были.
Расширенный синтаксис выглядит следующим образом:

```
try { .. пробуем выполнить код .. } catch(e) { .. перехватываем исключение .. } finally { .. выполняем всегда .. }
```

```
try {
  alert( 'try' );
  if (confirm('Сгенерировать ошибку?')) BAD_CODE();
} catch (e) {
  alert( 'catch' );
} finally {
  alert( 'finally' );
}
```

У кода есть два пути выполнения:
Если вы ответите на вопрос «Сгенерировать ошибку?» утвердительно, то try -> catch -> finally.
Если ответите отрицательно, то try -> finally.
Секцию finally используют, чтобы завершить начатые операции при любом варианте развития событий.

Итого

Конструкция `try..catch` позволяет обрабатывать ошибки во время исполнения кода. Она позволяет запустить код и перехватить ошибки, которые могут в нём возникнуть.

```
Синтаксис: try {
  // исполняем код
} catch(err) {
  // если случилась ошибка, прыгаем сюда
  // err - это объект ошибки
} finally {
  // выполняется всегда после try/catch
}
```

Секций catch или finally может не быть, то есть более короткие конструкции try..catch и try..finally также корректны.
Объекты ошибок содержат следующие свойства:

- message – понятное человеку сообщение.
- name – строка с именем ошибки (имя конструктора ошибки).
- stack (нестандартное, но хорошо поддерживается) – стек на момент ошибки.

Если объект ошибки не нужен, мы можем пропустить его, используя catch { вместо catch(err) {.
Мы можем также генерировать собственные ошибки, используя оператор throw.
Аргументом throw может быть что угодно, но обычно это объект ошибки, наследуемый от встроенного класса Error. Подробнее о расширении ошибок см. в следующей главе.
Проброс исключения – это очень важный приём обработки ошибок: блок catch обычно ожидает и знает, как обработать определённый тип ошибок, поэтому он должен пробрасывать дальше ошибки, о которых он не знает.

Работа с формами

Сейчас мы поговорим о различных приемах работы сценариев JavaScript с HTML-формами.

Если в HTML-документе определена форма, то она доступна сценарию JavaScript как объект, входящий в объект `document` с именем, заданным атрибутом `NAME` тега `FORM`.

Свойства форм

Форма имеет два набора свойств, состав одного из которых фиксированный, а состав другого зависит от того, какие элементы определены в форме.

Свойства первого набора

- **action**. Значение атрибута `ACTION` тега `FORM`.
- **encoding**. Значение атрибута `ENCTYPE` тега `FORM`.
- **method**. Значение атрибута `METHOD` тега `FORM`.
- **target**. Значение атрибута `TARGET` тега `FORM`.
- **elements**. Массив всех элементов формы.
- **length**. Размер массива elements.

Большинство свойств первого набора просто отражает значение соответствующих атрибутов тега `FORM`. Что же касается массива `elements`, то в нем находятся объекты, соответствующие элементам, определенным в форме. Эти объекты образуют второй набор свойств формы. Адресоваться к этим объектам можно как к элементам массива `elements`, причем первому элементу формы будет соответствовать элемент с индексом 0, второму - с индексом 1 и т.д. Однако удобнее обращаться к элементам формы по их именам, заданным атрибутом `NAME`.

Элементы форм

Кнопки (BUTTON, RESET, SUBMIT)

Свойства

- **name**. Имя объекта.
- **value**. Надпись на кнопке.

Метод

- **click( )**. Вызов этого метода тождественен щелчку мышкой по кнопке.

Пример

```
    <script>
        <!--
        function btnClick()
        {
            var Txt1 = "";
            var Txt2 = "";
            Txt1 = document.Test.bt.value;
            Txt2 = document.Test.bt.name;
            document.getElementById('ex1').innerHTML="<HR>"+
                "Вы нажали кнопку: " + Txt1.bold() +
                " с именем: "  + Txt2.bold() +"<HR>";
        }
         //-->
    </script>
</head>
<body>
<H1>Нажатие кнопки</H1>
<div id="ex1"></div>
<FORM NAME="Test">
    <INPUT TYPE="button" NAME="bt" VALUE="Щелкни здесь!"
        onClick="btnClick();">
</FORM>
```

Нажатие кнопки

Вы нажали кнопку: **Щелкни здесь!** с именем: **bt**

Щелкни здесь!

Флажок (CHECKBOX)

Свойства

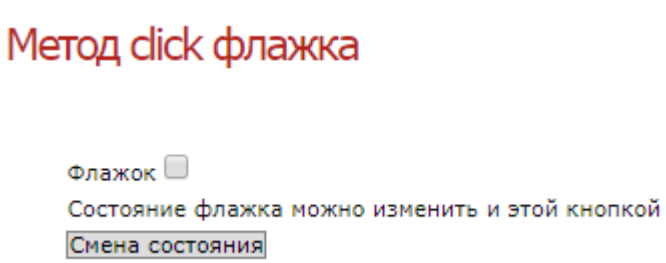
- **name**. Имя объекта.
- **value**. Надпись на кнопке.
- **checked**. Состояние флажка: `true` - флажок установлен, `false` - флажок не установлен.
- **defaultChecked**. Отражает наличие атрибута `CHECKED`: `true` - есть, `false` - нет.

Метод

- **click( )**. Вызов этого метода меняет состояние флажка.

Пример

```
<H1>Метод click флажка</H1>
<FORM NAME="Test">
    Флажок <INPUT TYPE="checkbox" NAME="ch">
    <BR>Состояние флажка можно изменить и этой кнопкой
    <INPUT TYPE="button" VALUE="Смена состояния"
           onClick="document.Test.ch.click();">
</FORM>
```



Переключатель (RADIO)

Свойства

- **name**. Имя объекта.
- **value**. Надпись на кнопке.
- **length**. Количество переключателей в группе.
- **checked**. Состояние переключателя: `true` - переключатель включен, `false` - выключен.
- **defaultChecked**. Отражает наличие атрибута `CHECKED`: `true` - есть, `false` - нет.

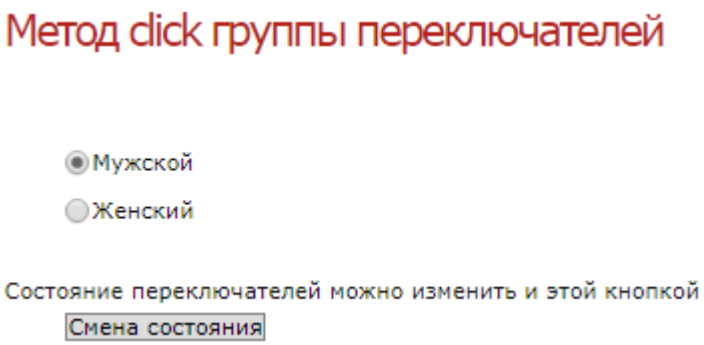
Метод

- **click( )**. Вызов этого метода включает переключатель.

Так как группа переключателей имеет одно имя `NAME`, то к переключателям надо адресоваться как к элементам массива.

Пример

```
    <script><!--
        function btnClick()
        {
            if(document.Test1.Sex[0].checked){
                document.Test1.Sex[1].click();
            }else{
                document.Test1.Sex[0].click();
            }
        }
         //-->
    </script>
</head>
<body>
<H1>Метод click группы переключателей</H1>
<FORM NAME="Test1">
    <INPUT TYPE="RADIO"  NAME="Sex" VALUE ="Man" CHECKED>Мужской
    <INPUT TYPE="RADIO"  NAME="Sex" VALUE ="Woman">Женский
    <BR>Состояние переключателей можно изменить и этой кнопкой
    <INPUT TYPE="button" VALUE="Смена состояния" onClick="btnClick();">
</FORM>
```



Список (SELECT)

Свойства

- **name**. Имя объекта.
- **selectedIndex**. Номер выбранного элемента или первого среди выбранных (если указан атрибут `MULTIPLE`).
- **length**. Количество элементов (строк) в списке.
- **options**. Массив элементов списка, заданных тегами `OPTION`.
  Каждый элемент массива **options** является объектом со следующими свойствами:
- **value**. Значение атрибута `VALUE`.
- **text**. Текст, указанный после тега `OPTION`.
- **index**. Индекс элемента списка.
- **selected**. Присвоив этому свойству значение `true`, можно выбрать данный элемент.
- **defaultSelected**. Отражает наличие атрибута `SELECTED`: `true` - есть, `false` - нет.

Методы

- **focus( )**. Передает списку фокус ввода.
- **blur( )**. Отбирает у списка фокус ввода.

Пример

Лекция 18. Использование фреймворков. Создание постраничной прокрутки на сайте. Использование fullPage.js

fullPage.js это jquery плагин, который позволяет создать постраничный скроллинг.

Начало работы с fullPage.js

Перед началом работы, у вас должны быть загружены следующие библиотеки:

- jQuery (≥1.6.0)
- файл `fullPage.css`
- файл `fullPage.js`

Вы можете скачать файлы fullPage.js с репозитория на Gthub или подключить их как CDN.
Файлы javascript подключим перед закрывающим тегом :

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.4/jquery.js"></
script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/fullPage.js/2.6.6/jquery.fullPage.
min.js"></script>
```

Теперь мы готовы использовать плагин!

Создадим полноэкранные секции

Сначала выделим секции, обернув их в тег section. Позже каждой секции будет присвоен уникальный id. По умолчанию, плагин считает активной первую секцию.

Чтобы сделать активной другую секцию, добавьте к ней класс action.
Пример HTML структуры:

```
<div id="fullPage">
   <section class="vertical-scrolling">
      <h2>fullPage.js</h2>
      <h3>Первая секция</h3>
      <div class="scroll-icon">
         <p>Перейти к последнему слайду</p>
         <a href="#fifthSection/1" class="icon-up-open-big"></a
      </div>
   </section>
   <section class="vertical-scrolling">
      <!-- контент здесь -->
   </section>
   <section class="vertical-scrolling">
      <!-- контент здесь -->
   </section>
   <section class="vertical-scrolling">
      <!-- контент здесь -->
   </section>
   <section class="vertical-scrolling">
      <!-- контент здесь -->
   </section>
</div>
```

Создадим горизонтальные слайды

Вертикально расположенные секции можно скроллить горизонтально. Применим для секций класс horizontal-scrolling.

```
<section class="vertical-scrolling">
   <div class="horizontal-scrolling">
      <h2>fullPage.js</h2>
      <h3>Это пятый раздел и содержит первый слайд</h3>
   </div>
   <div class="horizontal-scrolling">
      <h2>fullPage.js</h2>
      <h3>Это второй слайд</h3>
      <p class="end">Спасибо!</p>
   </div>
</section>
```

Настроим навигацию

Плагин предлагает множество встроенных опций для перемещения по разделам и слайдам. Некоторые из этих опций изначально включены. Мы хотим добавить в наш проект дополнительную навигацию в виде точек. Кроме того, мы скроем правую и левые стрелки, которые обычно появляются на слайде. После включения точек навигации, мы можем изменить их внешний вид путем перезаписи стилей по умолчанию. Вот новые стили:

```
#fp-nav ul li a span,
.fp-slidesNav ul li a span {
   background: white;
   width: 8px;
   height: 8px;
   margin: -4px 0 0 -4px;
}

#fp-nav ul li a.active span,
.fp-slidesNav ul li a.active span,
```

```
#fp-nav ul li:hover a.active span,
.fp-slidesNav ul li:hover a.active span {
   width: 16px;
   height: 16px;
   margin: -8px 0 0 -8px;
   background: transparent;
   box-sizing: border-box;
   border: 1px solid #24221F;
}
```

Ниже вы можете увидеть изменения:

Создадим ссылки для секций и слайдов

Зададим якорные ссылки для каждого раздела с помощью параметра anchors. Это массив который содержит ссылки:

```
anchors: ['firstSection', 'secondSection', 'thirdSection', 'fourthSection', 'fifthSection']
```

Лекция №19. Введение в анимацию свойств при помощи CSS3. Свойство transition

CSS-анимация - это возможность сделать вашу страницу интерактивной и добавить ей определенной привлекательности. Мы рассмотрим анимацию с помощью свойства `transition`, хотя есть еще такие свойства, как `@keyframes` и `animation`.

Свойство `transition` можно перевести, как **переход**, т.е. при анимации с помощью этого css-свойства в течение некоторого времени осуществляется плавный переход от начального к конечному значению выбранных для этого свойств.

hover-эффекты с помощью свойства transition

Чаще всего свойство `transition` применяется для создания **hover-эффектов**, или **эффектов при наведении курсора мыши**. Его суть заключается в том, что оно просчитывает изменения между свойствами, заданными для обычного состояния элемента и при наведении на него курсора мыши, которое задается с помощью **псевдокласса :hover**. Это позволяет создавать различные красивости от плавного изменения цвета текста и фона на кнопках-ссылках до наплывающих блоков и сменяющихся картинок. Очень интересные эффекты можно получить, если использовать псевдоэлементы ::before и/или ::after.

Составляющие свойства transition

Свойство `transition` является **составным** (обобщенным, или универсальным), что значит, что оно состоит из ряда свойств, которые можно задавать по отдельности. Синтаксис его таков:

```
transition: transition-property transition-duration transition-timing-function transition-delay;
```

Из кода видно, что вы можете использовать 4 свойства. Рассмотрим эти css-свойства по отдельности:

1. transition-property
2. transition-duration
3. transition-timing-function
4. transition-delay

Свойство transition-property

Возможные значения `transition-property`: `transition-property: свойство | all | none | initial | inherit`

Свойство `transition-property` задает **название css-свойства для анимации перехода**. Поскольку нужно просчитать разницу между начальным и конечным значением свойства, то и само свойство должно быть **рассчитываемым**. Например, можно изменить `border-width` с 1px до 6px, но нельзя преобразовать border-style из `solid` в `dashed`. Также можно уменьшить прозрачность элемента с помощью свойства `opacity` от значения 1 до 0, но невозможно анимировать переход от свойства `visibility: visible` к свойству `visibility: hidden` или от `display: block` к `display: none`. Для этого лучше воспользоваться методами `show()` или `hide()` библиотеки jQuery.

Ниже показан код, который меняет цвет фона у div с `id="tr-property"` с салатного на желтый за 0.8 секунды.

В свойстве `transition-property` можно указать несколько значений через запятую или использовать значение `all` (все свойства), которое является ***значением* по умолчанию**. Также свойство `transition-property` имеет значение `none` (ни одно из свойств), которое обычно применяют для отмены анимации, например на мобильных устройствах.

Рассмотрим пример, в котором нужно анимировать несколько свойств:

Обратите внимание, что для div-а с `id="tr-property2"` свойство `border-radius` изначально не было задано, оно получило значение в 8px только при наведении (псевдокласс `:hover`), тем не менее скругление углов тоже было анимированным, т.к. его включили в свойство transition-property, а его начальное значение (0) было взято из значений по умолчанию для `border-radius`.

Тот же самый пример можно было сократить, использовав в коде свойство `all` вместо перечисления нескольких свойств для анимации.

Свойство transition-duration

В примерах к свойству `transition-property` для создания анимации требовалось указать промежуток времени, за который как раз и отвечает свойство `transition-duration`. Этих двух свойств вполне достаточно для создания анимации типа `transition`. Можно вообще обойтись свойством `transition-duration`, т.к. `transition-property` по умолчанию имеет значение `all`, т.е. все измененные свойства будут анимированы.

Свойство `transition-duration` измеряется либо в секундах (`1s, 1.5s, 0.8s, .5s`) или в миллисекундах (`1000ms, 1500ms, 800ms, 500ms`) . По умолчанию это свойство имеет значение 0, т.е. время на анимацию фактически нет, поэтому, если вы забудете указать это значение, то переход свойств не произойдет.

Возможные значения transition-duration: время в s или ms | 0s (по умолчанию)| initial | inherit

Код примера для исследования свойства `transition-duration` таков:

Использование только свойства `transition-duration` при hover-эффекте:

Обратите внимание, что в качестве свойства, отвечающего за переход, использовано только `transition-duration` . Свойство `transition-property` принимает значение `all`, которое является значением по умолчанию, поэтому анимации подлежат все измененные свойства.

Свойство transition-timing-function

В соответствии с переводом это свойство задаёт временную функцию, которая описывает способ расчета скорости перехода свойств(а) html-элемента от одного значения к другому. По умолчанию свойство имеет значение `ease`, т.е. анимация происходит с некоторым замедлением.

Варианты свойства `transition-timing-function`:

transition-timing-function: ease| ease-in| ease-out| ease-in-out| linear | cubic-bezier |

step-start| step-end | steps

На примере ниже вы можете видеть действие большинства из этих вариантов. Можно заметить, что английское слово `ease`, которое присутствует в 4-х значениях свойства, отвечает за замедление движения, причем с добавкой `in` - в начале движения, с `out` - в конце, а с `in-out` - и в начале и в конце. График функции можно посмотреть в том же примере при наведении на любой блок. Суть использования графика заключается в том, что у нас **по вертикальной оси** отображен **прогресс**, т.е. разница в числовых значениях того свойства, которое изменяется в процессе анимации. **По горизонтальной оси** у нас показано **время**, в течение которого происходит переход.

На первом графике показан линейный переход (`linear`) - анимация происходит равномерно, без задержек в начале или в конце. *Второй график* отображает функцию `ease-in` - анимация происходит с замедлением в начале, т.к. именно в начале перехода за продолжительное время (2 клетки на графике) изменение свойства происходит очень незначительно (полклетки). Третий график отвечает за функцию `ease-out` - анимация с замедлением в конце. Для него характерно небольшое изменение значения анимируемого свойства за продолжительное время именно к концу анимации, т.е. ситуация и сам график противоположны функции `ease-in`.

Вариант `cubic-bezier` - это возможность управлять 2-мя точками графика с помощью маркеров для изменения точек на кривой Безье. чтобы управлять графиком функции нужно использовать **Инспектор свойств браузера** (клавиши `F12` или `Ctrl + Shift + I`). Скриншоты сделаны в браузере Chrome.

Для того чтобы увидеть график функции, управляющей скоростью перехода, нужно кликнуть по иконке графика рядом с названием функции - и вы получите не только его, но и визуальное отображение процесса перехода, и возможность поменять график на другой.

Потяните за точку в начале или в конце графика и задайте другой вариант управления скоростью анимации `transition`:

При изменении графика вы получите один из вариантов функции типа `cubic-bezier`, который отобразится в свойстве `transition-timing-function`. Если вы удовлетворены результатом, просто скопируйте код в свой редактор, т.к. изменения в Инспекторе свойств продержатся лишь до следующего обновления страницы.

Свойство transition-delay

Свойство `transition-delay` позволяет указать задержку в секундах или миллисекундах, после которой будет запущена анимация. Его необязательно использовать для всех анимаций и даже необязательно указывать, т.к. по умолчанию `transition-delay` равен 0s.

```
transition-delay: 0s | число s | число ms | initial | inherit
```

Это свойство удобно использовать, когда необходимо, чтобы анимация "сыграла" не сразу, а с некоторым отставанием. В примере ниже мы создадим эффект всплывающей подсказки, которая появляется через `0.3s` после наведения курсора на элемент с изображением. При наведении на блок с классом `.tr-holder` мы будем запускать анимацию для блока с классом `.tr-descr`. Обратите внимание на то, как записан код: свойства группы `transition`, в том числе и `transition-delay`, указаны для `.tr-descr`, а псевдокласс `:hover` использован для `.tr-holder`. При этом изменение свойства `bottom` все же записано в виде контекстного селектора:

.tr-holder:hover .tr-descr { bottom: 0; }

Кроме того, для блока `.tr-descr` указан курсор в виде руки, как подсказка, что при наведении на этот элемент что-то произойдет.

Еще одна особенность стилей `.tr-holder`, на которую стоит обратить внимание - это свойство `overflow: hidden` при заданных свойствах `width` и `height`, которое позволяет отсечь все, что выходит за пределы этих ширины и высоты. Именно благодаря этим свойствам мы не видим изначально блока с описанием `.tr-descr`, т.к. он спрятан внизу за счет использования свойства `bottom: -40%`.

Лекция №20. Фильтры изображений на CSS3.

Filter – это свойство в CSS3, которое может видоизменять ваши картинки. Накладывать размытость, увеличивать контраст и яркость, добавлять тень, менять цвета и многое, многое другое.

Всего у Filter есть 10 значений,

Браузеры обрабатывают страницу попиксельно применяя заданные эффекты и отрисовывают результат поверх оригинала. Таким образом, применяя несколько фильтров можно достигать различных эффектов, они как бы накладываются друг на друга. Чем больше фильтров, тем больше времени требуется браузеру, чтобы отрисовать страницу.

Можно применять несколько фильтров одновременно. Классический способ применения таких эффектов — при наведении на элемент `:hover`.

1. Фильтр размытие - blur

Если говорить простым языком, то это обычное размытие картинки. Фильтр подойдет, если вам нужно сделать края более мягкими. За счет размытия создается ощущение фона, который не в фокусе.

Давайте попробуем применить наш фильтр на лисичке, прописав вот такой код:

Фильтр со значением blur указывается именно в пикселях. Причем, чем больше это значение, тем больше проявляется размытость картинки.

Фильтр яркость — brightness

Этот фильтр напоминает изменение яркости экрана телевизора. В данном случае регулируется цвет между черным и оригинальным цветом по мере добавления параметров.

Регулировать вы можете от 0% и более. При 0% изображение будет черным, при 100% - оригинальным, а при 200% - станет ярче в два раза. Это очень хороший эффект, особенно для темных

изображений.

Фильтр brightness может задаваться 3 способами:

1. целые числа
2. дробные числа
3. проценты

Причем ограничений в принципе нет. В примере мы поставили значение 2 и увеличили яркость нашей картинки на 2 раза или на 200%.

3. Фильтр контрастность — contrast

Этот фильтр позволит вам повысить контраст картинки, регулируя разницу между светлыми и темным тонами изображения. Здесь значения также задаются тремя способами: целые числа, дробные числа и проценты. Таким образом, 100% - это значение по умолчанию. 0% - черное изображение. А все, что больше 100%, добавляет вам контраст.

4. Фильтр насыщенность — saturate

Это очень классный эффект, который сделает ваши изображения более яркими и насыщенными. Значения указываем в трех вариантах: целые и дробные числа, а также, проценты. Укажем значение - 200%. Повысим насыщенность нашей картиночки в 2 раза.

5. Фильтр прозрачность — opacity

Устанавливает прозрачность. На значения данного фильтра вводятся определенные ограничения:

- целые и дробные числа: от 0 до 1
- проценты: от 0% до 100

Давайте попробуем уменьшить прозрачность на 50% следующей картинке.

6. Фильтр Инверсия - invert

Он позволяет вам "переворачивать" цвета. На значения данного фильтра также вводятся ограничения:

- целые и дробные числа: от 0 до 1
- проценты: от 0% до 100

В нашем случае установим максимальное значение - 100 %

7. Фильтр сепия — sepia

Он позволит вам изменить цвет, добавив оттенок сепия. То есть вы получите имитацию старой фотографии. На значения фильтра ограничения указываются те же самые, что и в двух предыдущих.

- целые и дробные числа: от 0 до 1

- проценты: от 0% до 100

8. Фильтр оттенки серого — grayscale

Данный фильтр позволяет нам превращать цвета в оттенки серого. На значения фильтра также наложены ограничения:

- целые и дробные числа: от 0 до 1
- проценты: от 0% до 100

Таким образом, при 100% все изображение будет с оттенками серого, а при 0% останется неизменным. 0 приравнивается к 0%, а 1,0 - к 100%.

Зададим значение - 0.7 (или 70%)

9. Фильтр оттенок освещения — hue-rotate

На мой взгляд, это очень классный фильтр, при помощи которого можно например, менять цвет исходного изображения, изменяя угол освещения.

Мы зададим значение - 300 градусов:

10. Фильтр тень - drop-shadow

Фильтр задается сразу несколькими значениями. Сначала мы задаем значение по оси X, потом - по оси Y. Так мы обозначаем смещение тени по оси X и Y. Далее указывается радиус нашей тени и последним атрибутом - ее цвет.

В нашем случае укажем смещение тени на 3 пикселя, размер 5 и цвет тёмно-серый.

Лекция №20. Анимация по ключевым кадрам. Использование правила @keyframes

1. Ключевые кадры

Ключевые кадры используются для указания значений свойств анимации в различных точках анимации. Ключевые кадры определяют поведение одного цикла анимации; анимация может повторяться ноль или более раз.
Ключевые кадры указываются с помощью правила `@keyframes`, определяемого следующим образом: `@keyframes имя анимации { список правил }`
Создание анимации начинается с установки **ключевых кадров** правила `@keyframes`. Кадры определяют, какие свойства на каком шаге будут анимированы. Каждый кадр может включать один или более блоков объявления из одного или более пар свойств и значений.
Правило `@keyframes` содержит имя анимации элемента, которое связывает правило и блок объявления элемента.

```
@keyframes shadow {from {text-shadow: 0 0 3px black;}50% {text-shadow: 0 0 30px black;}to {text-shadow: 0 0 3px black;}}
```

Ключевые кадры создаются с помощью ключевых слов `from` и `to` (эквивалентны значениям `0%` и `100%`) или с помощью процентных пунктов, которых можно задавать сколько угодно. Также можно комбинировать ключевые слова и процентные пункты. Если кадры имеют одинаковые свойства и значения, их можно объединить в одно объявление:

```
@keyframes move {from,to {top: 0;left: 0;}25%, 75% {top: 100%;}50% {top: 50%;}}
```

Если `0%` или `100%` кадры не указаны, то браузер пользователя создает их, используя вычисляемые (первоначально заданные) значения анимируемого свойства.

Если несколько правил `@keyframes` определены с одним и тем же именем, сработает последнее в порядке документа, а все предыдущие проигнорируются.
После объявления правила `@keyframes`, мы можем ссылаться на него в свойстве `animation`:

```
h1{font-size:3.5em;color:darkmagenta;animation:shadow  2s infinite ease-in-out;}
```

Не рекомендуется анимировать нечисловые значения (за редким исключением), так как результат в браузере может быть непредсказуемым. Также не следует создавать ключевые кадры для значений свойств, не имеющих средней точки, например, для значений свойства `color: pink` и `color: #ffffff`, `width: auto` и `width: 100px` или `border-radius: 0` и `border-radius: 50%` (в этом случае правильно будет указать `border-radius: 0%`).

1.1. Временная функция для ключевых кадров

Правило стиля ключевого кадра также может объявлять временную функцию, которая должна использоваться при перемещении анимации к следующему ключевому кадру.

Пример

```
@keyframes bounce {
  from { top: 100px;    animation-timing-function: ease-out;  }
  25% {top: 50px;    animation-timing-function: ease-in;  }
  50% {    top: 100px;    animation-timing-function: ease-out;  }
  75% {    top: 75px;    animation-timing-function: ease-in;  }
  to {    top: 100px;  }}
```

Пять ключевых кадров указаны для анимации с именем «bounce». Между первым и вторым ключевым кадром (то есть между 0% и 25%) используется функция замедления. Между вторым и третьим ключевым кадром (то есть между 25% и 50%) используется функция плавного ускорения. И так далее. Элемент будет перемещаться вверх по страницу на `50px`, замедляясь по мере того, как он достигает своей наивысшей точки, а затем ускоряясь, когда он падает до `100px`. Вторая половина анимации ведет себя аналогичным образом, но только перемещает элемент на `25px` вверх по странице.

Временная функция, указанная в ключевом кадре `to` или `100%`, игнорируется.

```
<!DOCTYPE html>
<html>
<head>
<title>Пример использования CSS правила @keyframes</title>
<style>
.test {
width: 75px; /* устанавливаем ширину блока */
height: 75px;  /* устанавливаем высоту блока */
border-radius: 50px;  /* определяем форму углов элемента */
position:  relative;  /* элемент с относительным позиционированием (position:static - по умолчанию и, как следствие, значение свойства left не повлияли бы на позиционирование элемента) */
-webkit-animation: iliketomoveit 5s infinite;  /*  для поддержки ранних версий браузеров */
animation: iliketomoveit 5s infinite;  /* задаём имя анимации (соответствует имени назначенному в ключевых кадрах), продолжительность анимации и указываем, что анимация будет повторяться бесконечно  */
}
@-webkit-keyframes iliketomoveit { /* для поддержки ранних версий браузеров */
        0%   {left: 0px;} /* задаем начало анимации */
```

```
        25%  {left: 400px; background: red;} /* сдвигаем элемент на 400px от левого края при этом устанавливаем цвет заднего фона */
        75%  {left: 200px;}  /* уменьшаем сдвиг от левого края */
        100%  {left:  0px;  background:  green;}  /* возвращаем  элемент  на первоначальную точку и изменяем цвет заднего фона */
        }
@keyframes iliketomoveit {
        0%   {left: 0px;}
        25%  {left: 400px; background: red;}
        75%  {left: 200px;}
        100% {left: 0px; background: green;}
        }
</style>
</head>
<body>
<div class = "test"></div>
</body>
</html>
```

```
@keyframes animationName {
        from | %  {css-styles} // начало цикла
        to | %  {css-styles} // конец цикла
}
```

animationName - имя анимации
from | to | % - селектор ключевого кадра
css-styles - css стили

Лекция №22. Размещение сайта на хостинге. Введение в системы управление контентом. Установка виртуального сервера OpenServer.

Хостинг сайтов – это онлайн услуга, которая позволяет публиковать ваш веб-сайт или веб-приложение в интернете. Когда вы подписываетесь на услугу хостинга, вы обычно арендуете пространство на сервере, на котором вы можете хранить все файлы и данные, необходимые для правильного функционирования вашего сайта.

Сервер – это физический компьютер, который работает без перерывов, чтобы ваш сайт был доступен всё время для тех, кто хочет его посетить. Ваш хостинг отвечает за поддержание работы сервера, защиту его от вредоносных атак и передачу вашего контента (текста, изображений, файлов) с сервера в браузеры ваших посетителей.

Когда вы решите запустить новый сайт, вам понадобиться найти хостинговую компанию, которая предоставит вам ресурсы на сервере. Ваш хостинг провайдер хранит все ваши файлы, ресурсы и базы данных на сервере. Всякий раз, когда кто-то вводит ваше доменное имя в адресную строку своего браузера, ваш хост передаёт все файлы, необходимые для обслуживания запроса.

Фактически, веб-хостинг работает аналогично аренде жилья, вам нужно регулярно оплачивать арендную плату, чтобы поддерживать постоянную работу своего сайта.

Аккаунты хостинга имеют графический интерфейс пользователя, где вы можете управлять всеми настройками своего сайта. Например, вы можете загружать на

сервер HTML и другие файлы, устанавливать системы управления контентом, такие как WordPress, обращаться к своей базе данных и создавать бэкапы (резервные копии) для своего сайта.

Помимо предоставления серверного пространства для вашего сайта, хостинг провайдеры могут также предлагать другие услуги, связанные с управлением сайтов, такие как:

- SSL-сертификаты (для обеспечения безопасности сайтов используется протокол https://)
- Хостинг электронной почты Email
- Конструкторы страниц
- Инструменты для разработчиков
- Услуги поддержки клиентов (обычно с онлайн чатом)
- Автоматизированное создание бэкапов (резервных копий данных)
- Установщики программ в 1-клик мыши (например, CMS WordPress или Drupal)

Различные типы хостинга

Большинство провайдеров предлагают несколько типов хостинга для удовлетворения различных потребностей клиентов. Вот наиболее часто предоставляемые типы хостинга:

- Общий хостинг (Shared Hosting)
- VPS (**V**irtual **P**rivate **S**erver – виртуальный приватный сервер) хостинг
- Облачный хостинг (Cloud Hosting)
- WordPress хостинг
- Хостинг выделенных серверов

Общий хостинг (его ещё иногда называют виртуальным хостингом) является наиболее распространённым типом веб-хостинга, и это отличное решение для большинства небольших проектов и блогов. Общий хостинг предполагает, что вы совместно с другими клиентами вашего хостинга пользуетесь ресурсами сервера. Веб-сайты, расположенные на одном сервере разделяют свои ресурсы, такие как память, вычислительная мощность, дисковое пространство и другие.

Достоинства:

- Низкая стоимость
- Удобно для начинающих (не требует специальных технических знаний)
- Настроенный сервер
- Понятная панель управления
- Обслуживание и администрирование сервера выполняются службой поддержки

Недостатки:

- Ограниченная возможность конфигурации (настройки) сервера
- Потоки трафика на других сайтах могут замедлить работу вашего сайта

С хостингом VPS (Virtual Private Server – виртуальный приватный сервер) вы по-прежнему используете сервер совместно с другими пользователями, однако ваш провайдер выделяет вам отдельный раздел на сервере. Это означает, что вы получаете выделенное пространство на сервере и зарезервированное количество вычислительной мощности и памяти.

Достоинства:

- Выделенные ресурсы на сервере (без платы за выделенный сервер)
- Потоки трафика на другие сайты никак не влияют на производительность вашего сайта
- Root-доступ к сервера
- Лёгкая масштабируемость
- Высокая настраиваемость

Недостатки:

- Дороже общего хостинга
- Необходимы технические знания и знания по управлению серверами

Облачный хостинг в настоящее время является самым надёжным решением на рынке, поскольку он работает буквально бесперебойно. Облачный хостинг предоставляет вам кластер серверов. Ваши файлы и ресурсы реплицируются(копируются) на каждом сервере. Когда один из облачных серверов занят или имеет какие-либо проблемы, ваш трафик

автоматически направляется на другой сервер в кластере.

Достоинства:

- Сбои на сервере не сказываются на работе вашего сайта
- Ресурсы выделяются по требованию
- Оплата по мере использования (платите только за то, что используете)
- Более масштабируемый, чем VPS

Недостатки:

- Сложно заранее просчитать стоимость
- Не всегда предоставляется Root-доступ

WordPress Хостинг – это разновидность общего хостинга, специально созданная для размещения сайтов на WordPress. Ваш сервер настроен определённым образом наиболее подходящим для потребностей работы CMS WordPress, на вашем сайте сразу готовые заранее установленный плагины для таких важных моментов как, например, кэширование и безопасность. Благодаря высоко оптимизированной конфигурации ваш сайт загружается намного быстрее и работает с меньшим количеством проблем. Тарифные планы, ориентированные на размещение сайтов на WordPress часто включают дополнительные функции, связанные с WordPress, такие как дополнительные темы WordPress, конструкторы страниц простым перетаскиванием и специальные инструменты разработчика.

Достоинства:

- Низкая цена (обычно доступны по такой же цене, как и общий хостинг)
- Удобный для начинающих
- Установка WordPress в одно нажатие
- Отличная производительность для сайтов на WordPress
- Служба поддержки клиентов хорошо разбирается в вопросах, связанных с WordPress
- Заранее установленные плагины и темы для WordPress

Недостатки:

- Рекомендуется использовать только для сайтов на WordPress (может возникнуть проблема, если вы захотите разместить более одного сайта на своём аккаунте, и не все из них используют WordPress)

Выделенный хостинг означает, что у вас есть собственный физический сервер, который предназначен исключительно для вашего сайта. Таким образом, выделенный хостинг даёт вам невероятную гибкость. Вы можете настроить свой сервер по своему усмотрению, выбрать операционную систему и программное обеспечение, которые хотите использовать, и настроить всю среду размещения в соответствии с вашими потребностями.

Фактически, аренда выделенного сервера равносильна своему собственному локальному серверу, но поставляется с профессиональной поддержкой вашего хостинг провайдера.

Достоинства:

- Полное управление конфигурацией вашего сервера
- Высокая надёжность (вы ни с кем не делите ресурсы своего сервера)
- Полный Root-доступ
- Высокая безопасность

Недостатки:

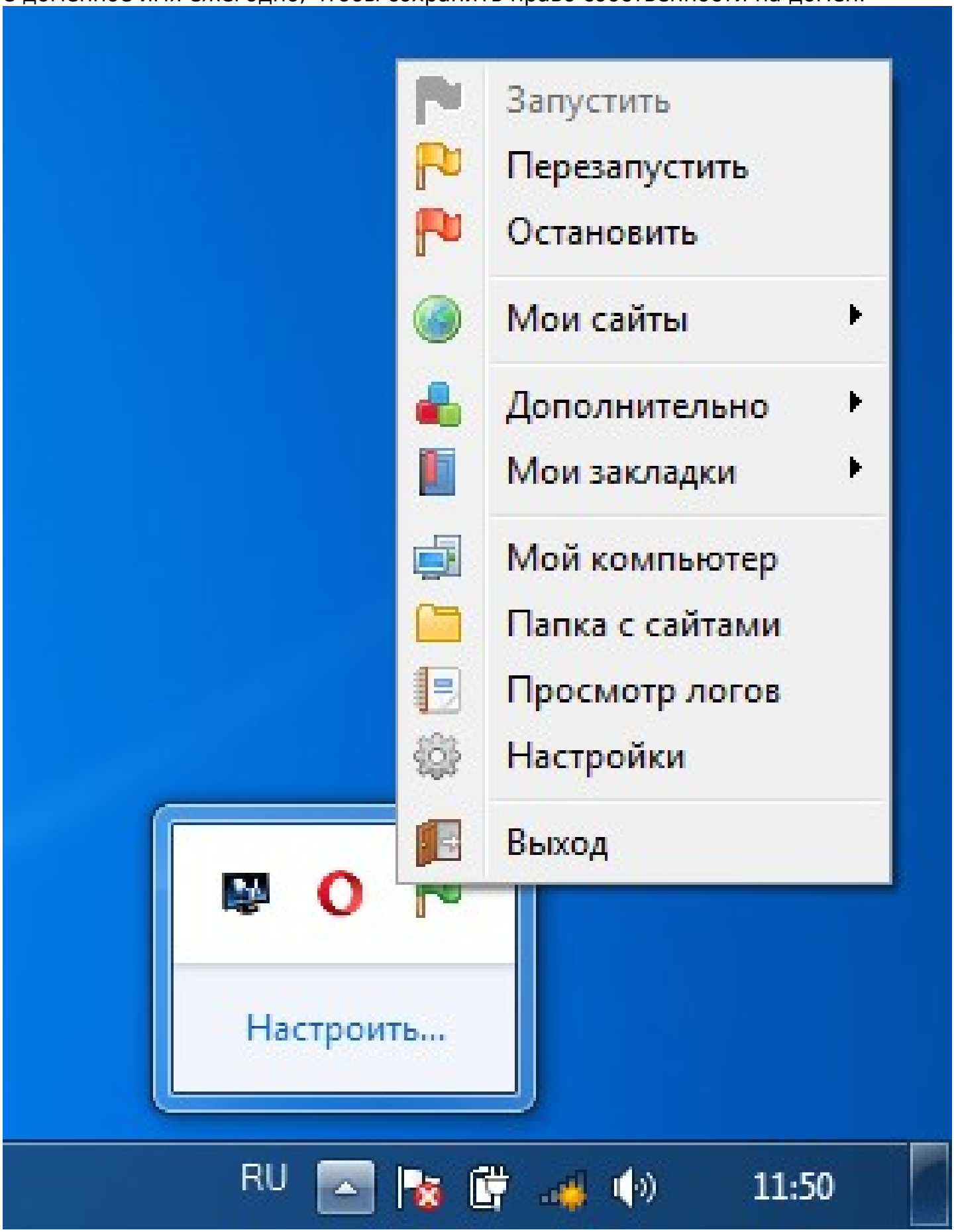
- Высокая стоимость
- Необходимы технические знания и знания по управлению серверами

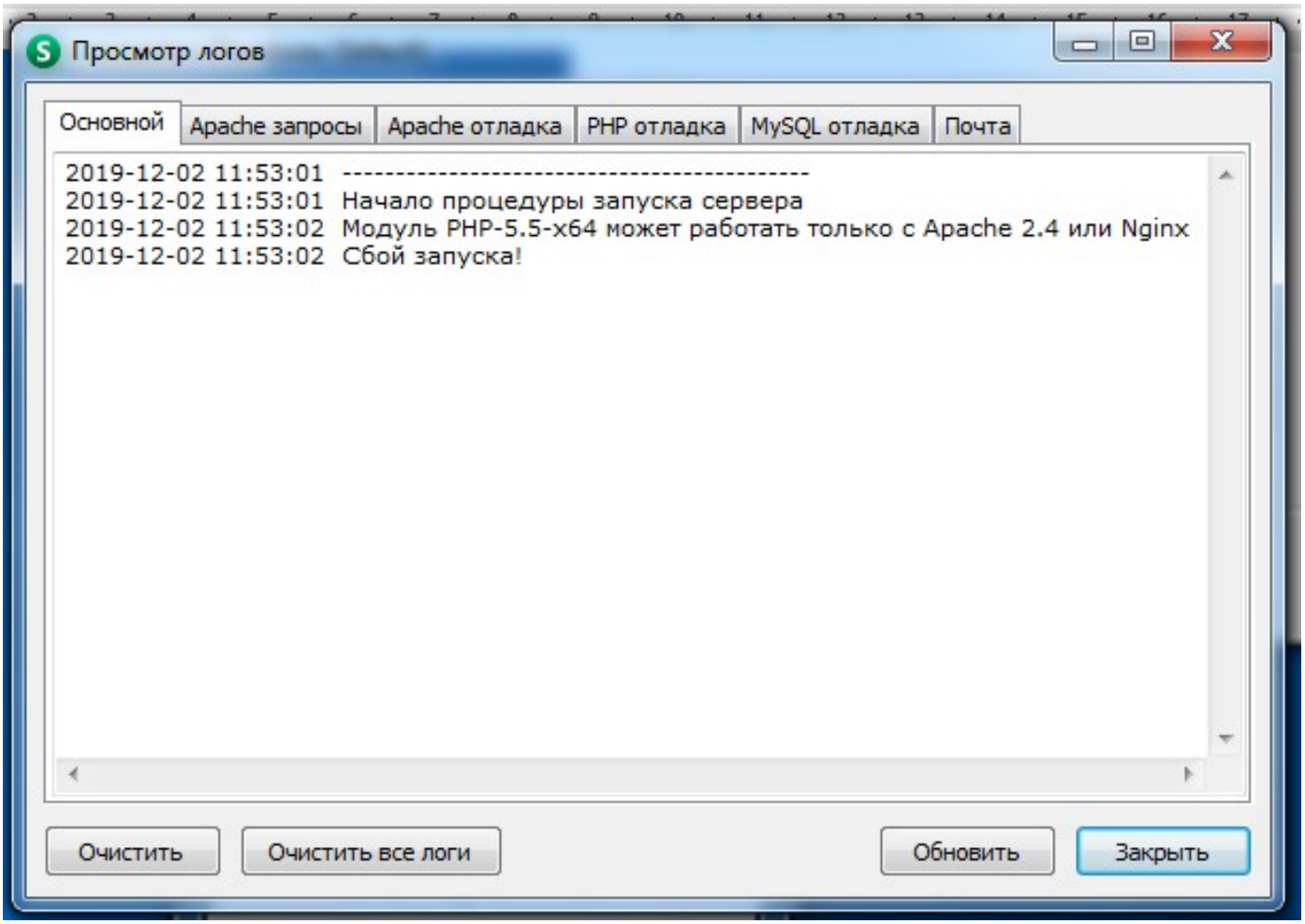
Разница между Хостингом сайтов и Доменными именами

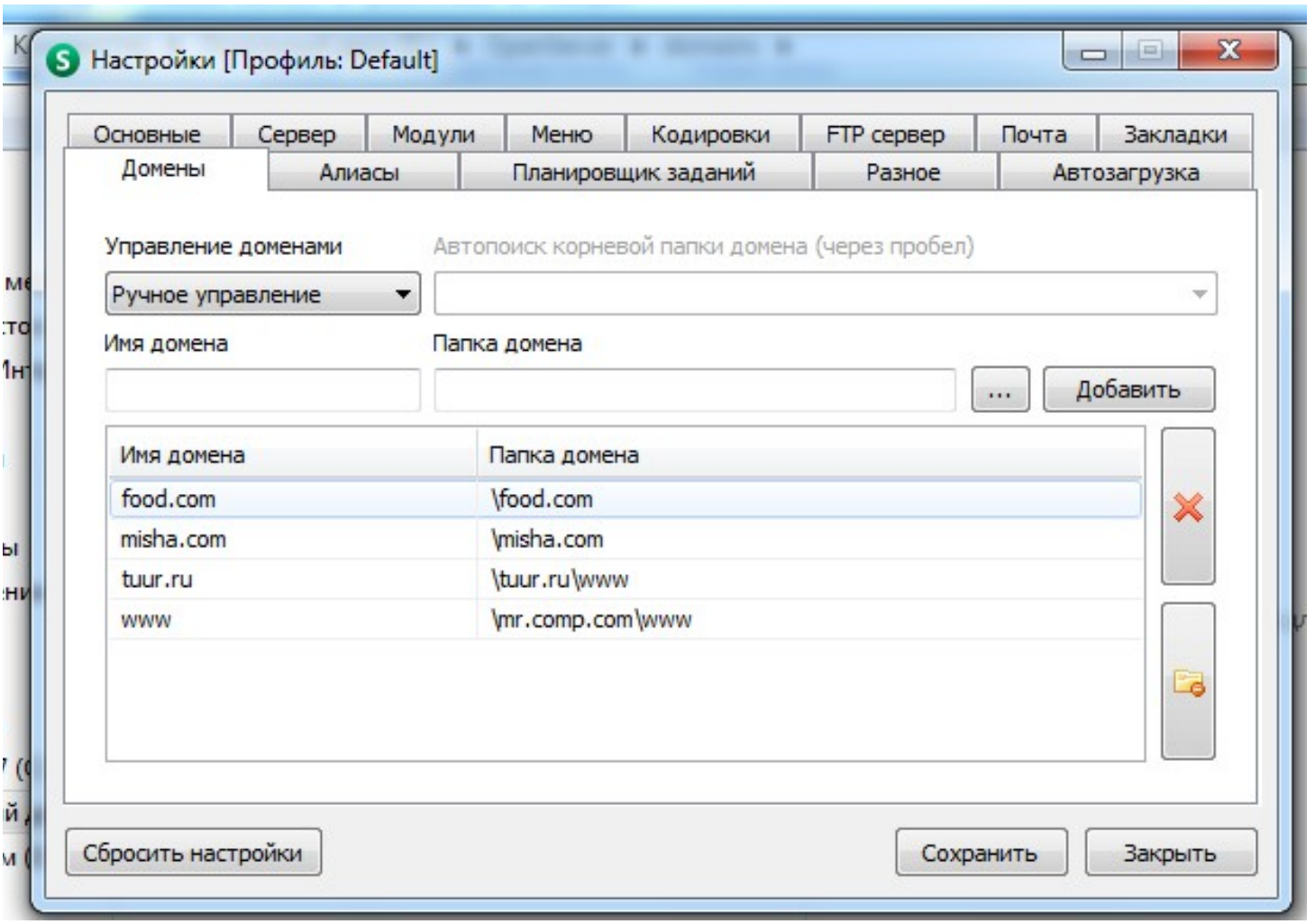
Кроме регистрации услуги хостинга, вам также необходимо приобрести домен. В чём разница? Хотя веб-хостинг позволяет арендовать серверное пространство для вашего сайта, домен является его адресом, например, наше доменное имя – hostinger.ru. Когда посетители хотят зайти на ваш сайт, они вводят имя домена в адресную строку своего браузера, и сервер передаёт содержимое, которое они запросили.

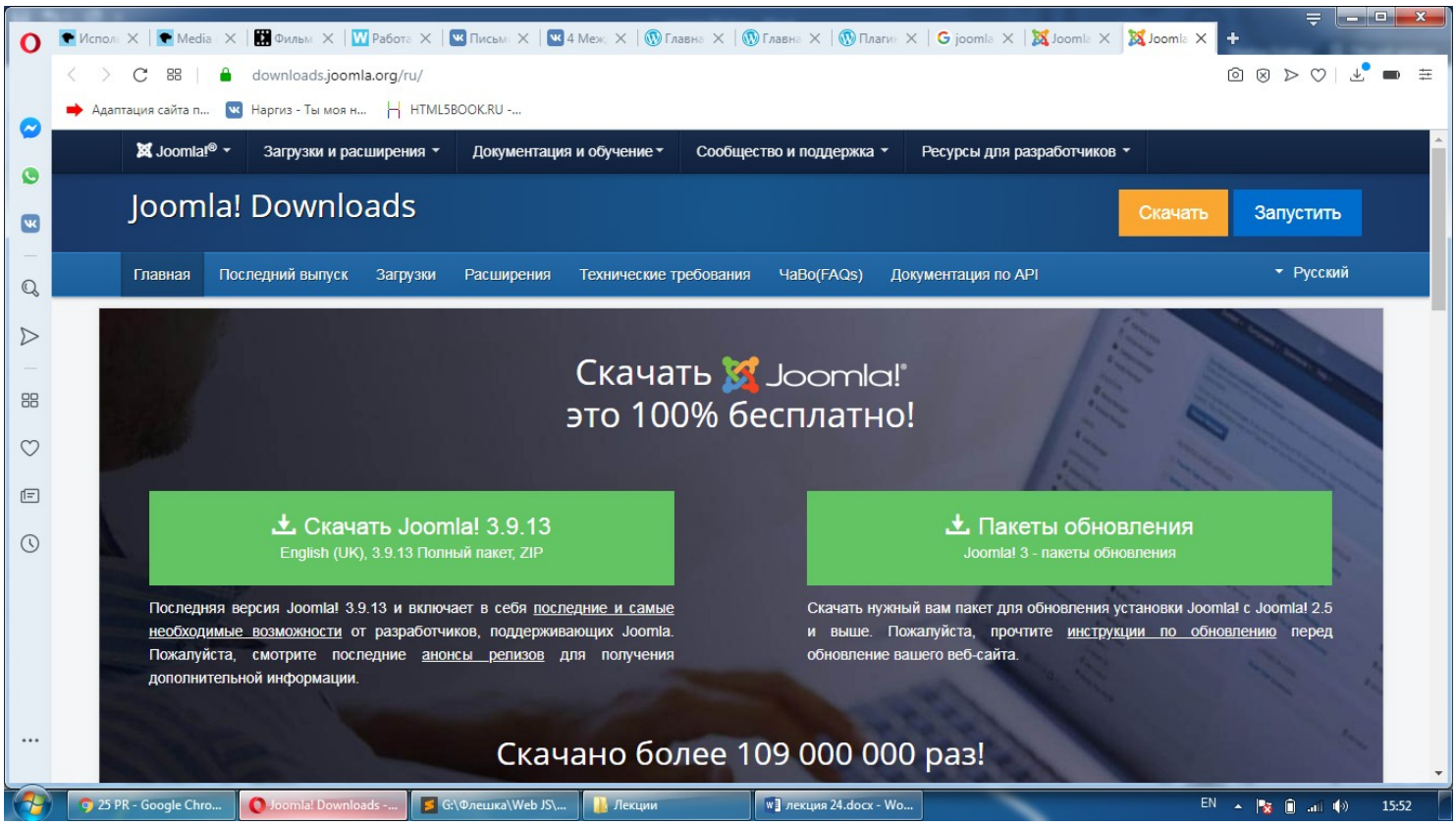
Большинство провайдеров хостинга предлагают отдельно приобрести доменное имя. Или, если у вас уже есть домен, вы также можете перенести его к текущему

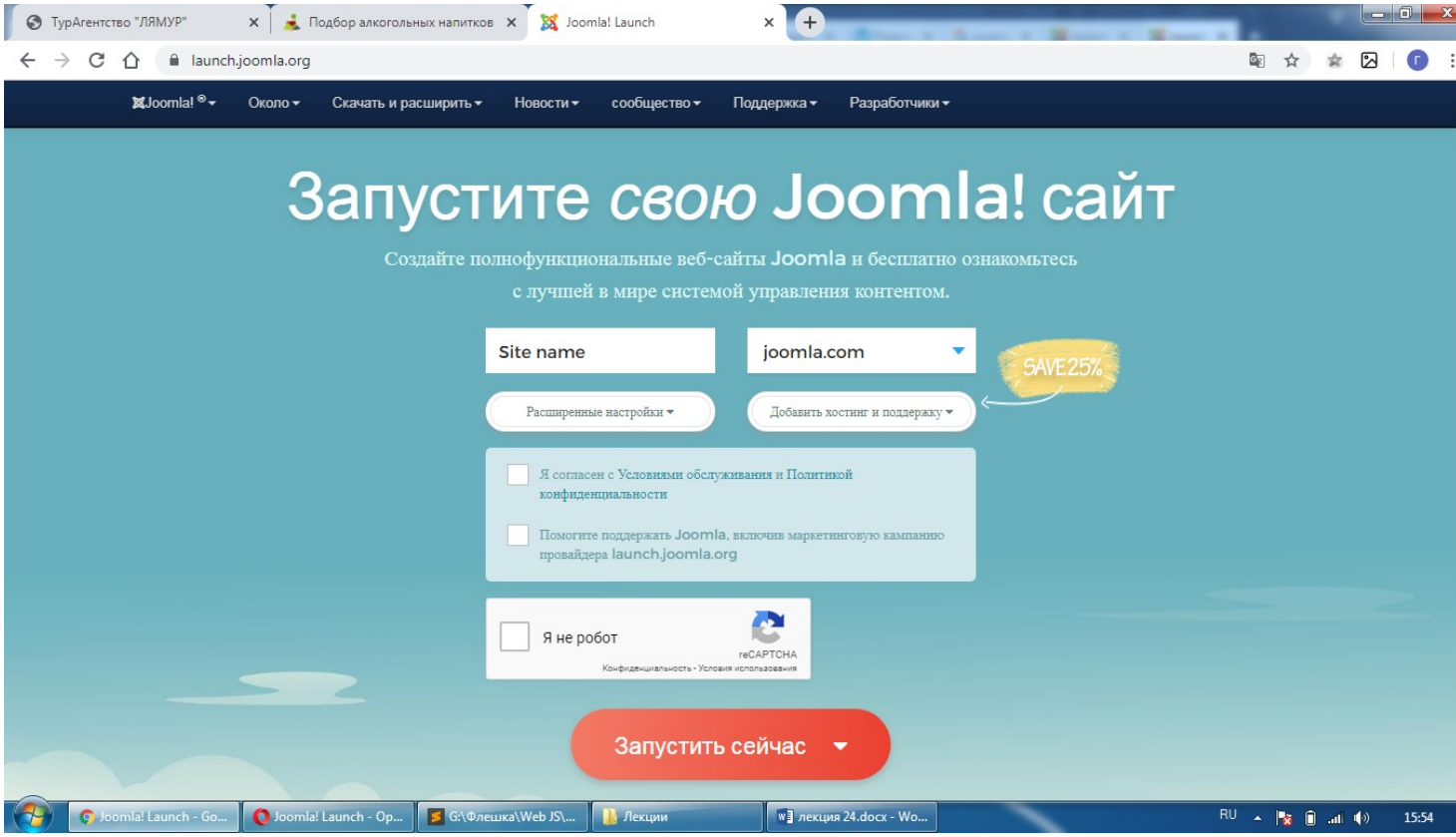
хостинг провайдеру. Подобно тарифным планам хостинга, вы должны платить за своё доменное имя ежегодно, чтобы сохранить право собственности на домен.

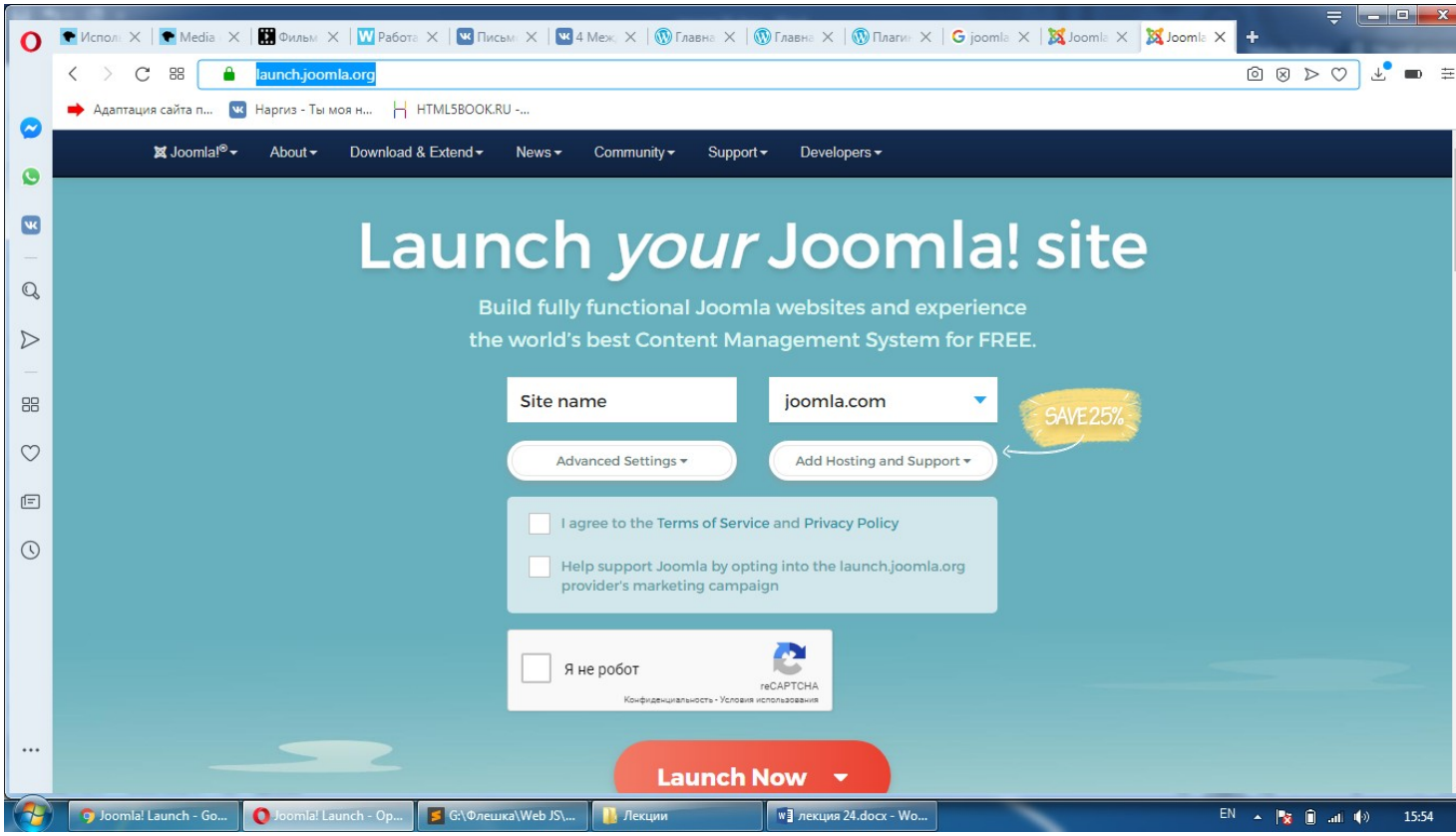


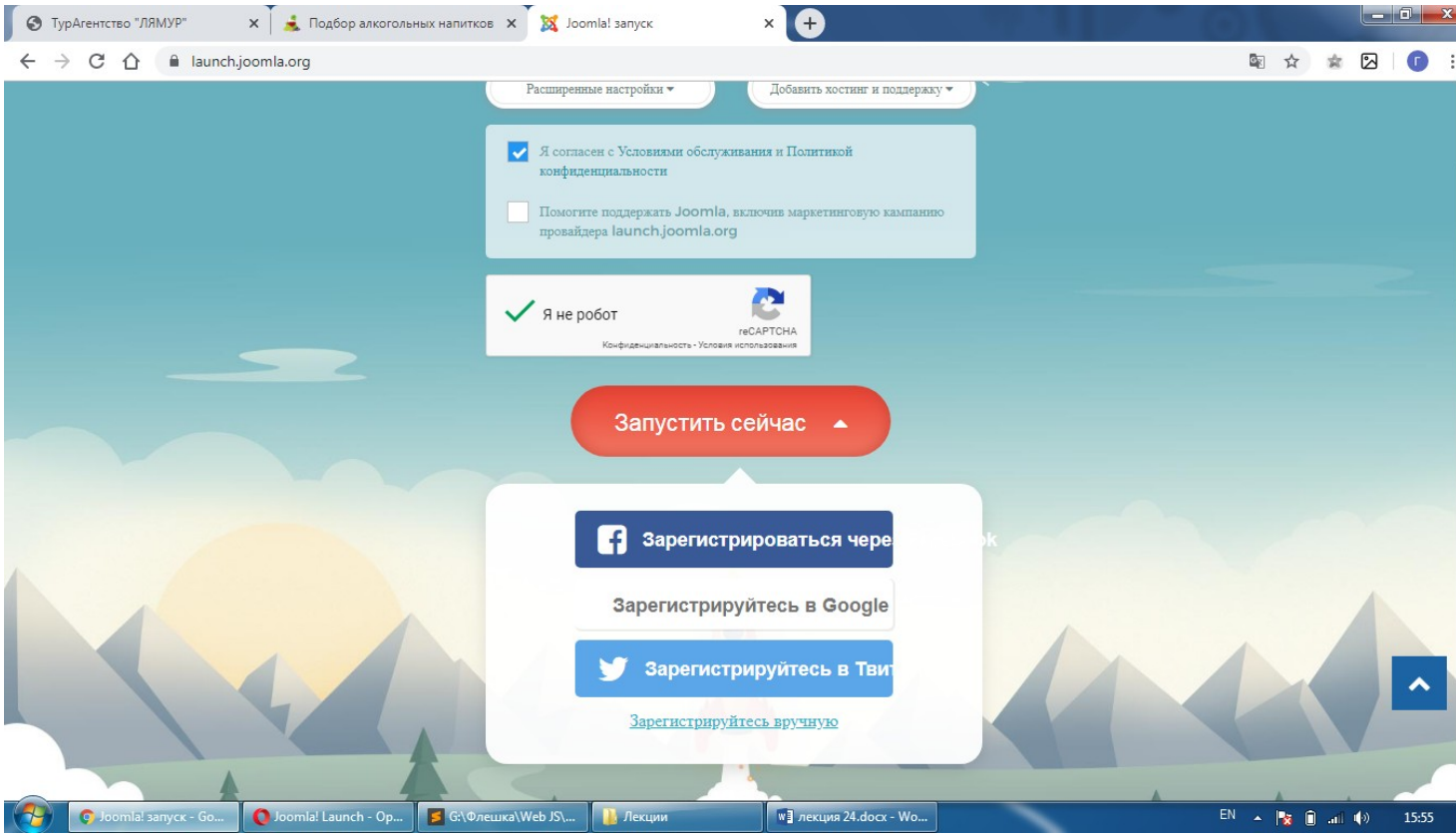

Просмотр логов
Основной
Apache запросы
Apache отладка
PHP отладка
MySQL отладка
Почта
2019-12-02 11:53:01 ----------
2019-12-02 11:53:01 Начало процедуры запуска сервера
2019-12-02 11:53:02 Модуль PHP-5.5-x64 может работать только с Apache 2.4 или Nginx
2019-12-02 11:53:02 Сбой запуска!
Очистить
Очистить все логи
Обновить
Закрыть

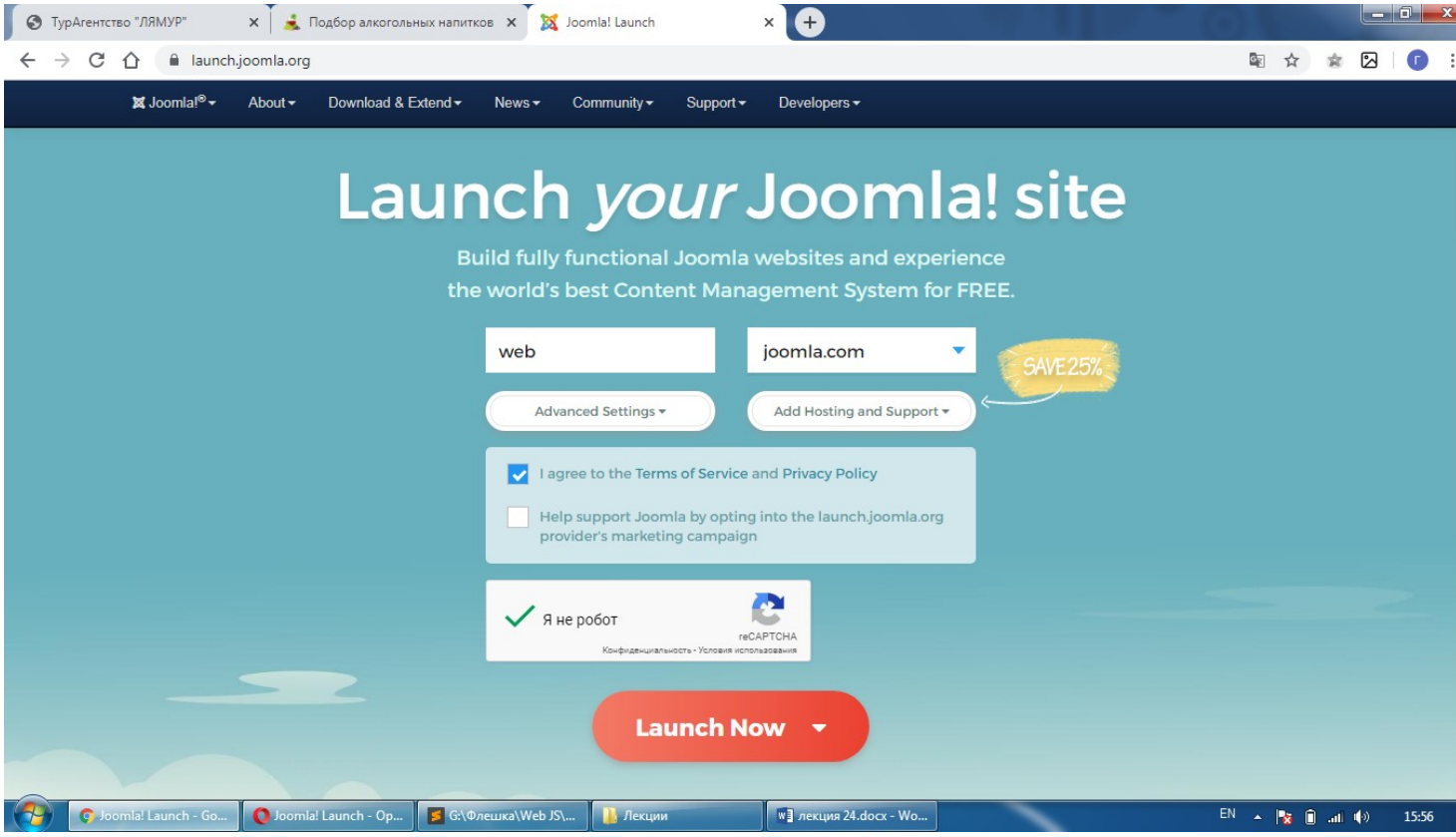

Настройки [Профиль: Default]
Основные
Сервер
Модули
Меню
Кодировки
FTP сервер
Почта
Закладки
Домены
Алиасы
Планировщик заданий
Разное
Автозагрузка
Управление доменами
Автопоиск корневой папки домена (через пробел)
Ручное управление
Имя домена
Папка домена
...
Добавить
Имя домена
Папка домена
food.com
\food.com
misha.com
\misha.com
tuur.ru
\tuur.ru\www
www
\mr.comp.com\www
Сбросить настройки
Сохранить
Закрыть

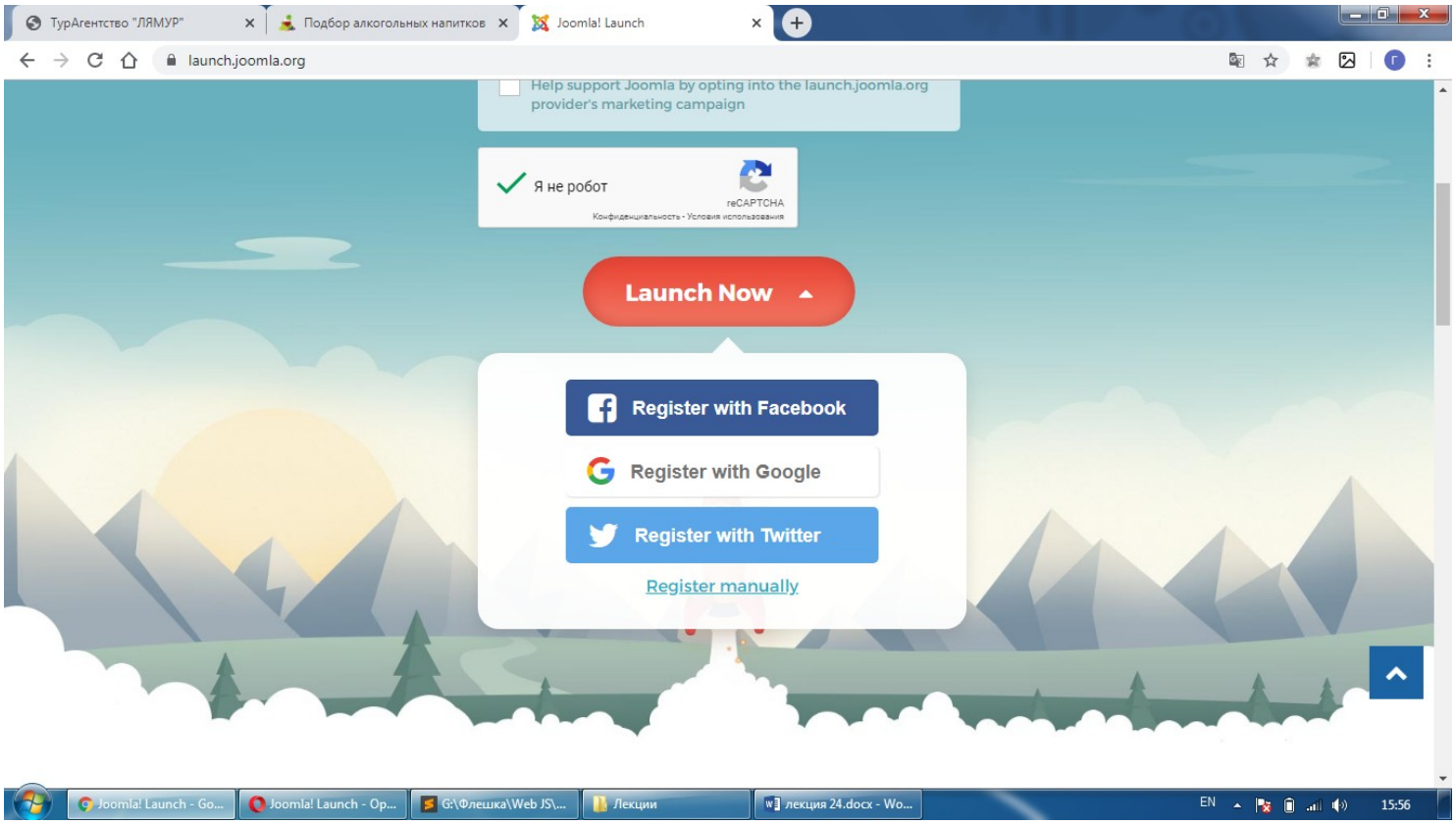

downloads.joomla.org/ru/
Joomla! Downloads
Скачать
Запустить
Главная
Последний выпуск
Загрузки
Расширения
Технические требования
ЧаВо(FAQs)
Документация по API
Русский
Скачать Joomla!
это 100% бесплатно!
Скачать Joomla! 3.9.13
English (UK), 3.9.13 Полный пакет, ZIP
Пакеты обновления
Joomla! 3 - пакеты обновления
Последняя версия Joomla! 3.9.13 и включает в себя последние и самые необходимые возможности от разработчиков, поддерживающих Joomla. Пожалуйста, смотрите последние анонсы релизов для получения дополнительной информации.
Скачать нужный вам пакет для обновления установки Joomla! с Joomla! 2.5 и выше. Пожалуйста, прочтите инструкции по обновлению перед обновление вашего веб-сайта.
Скачано более 109 000 000 раз!

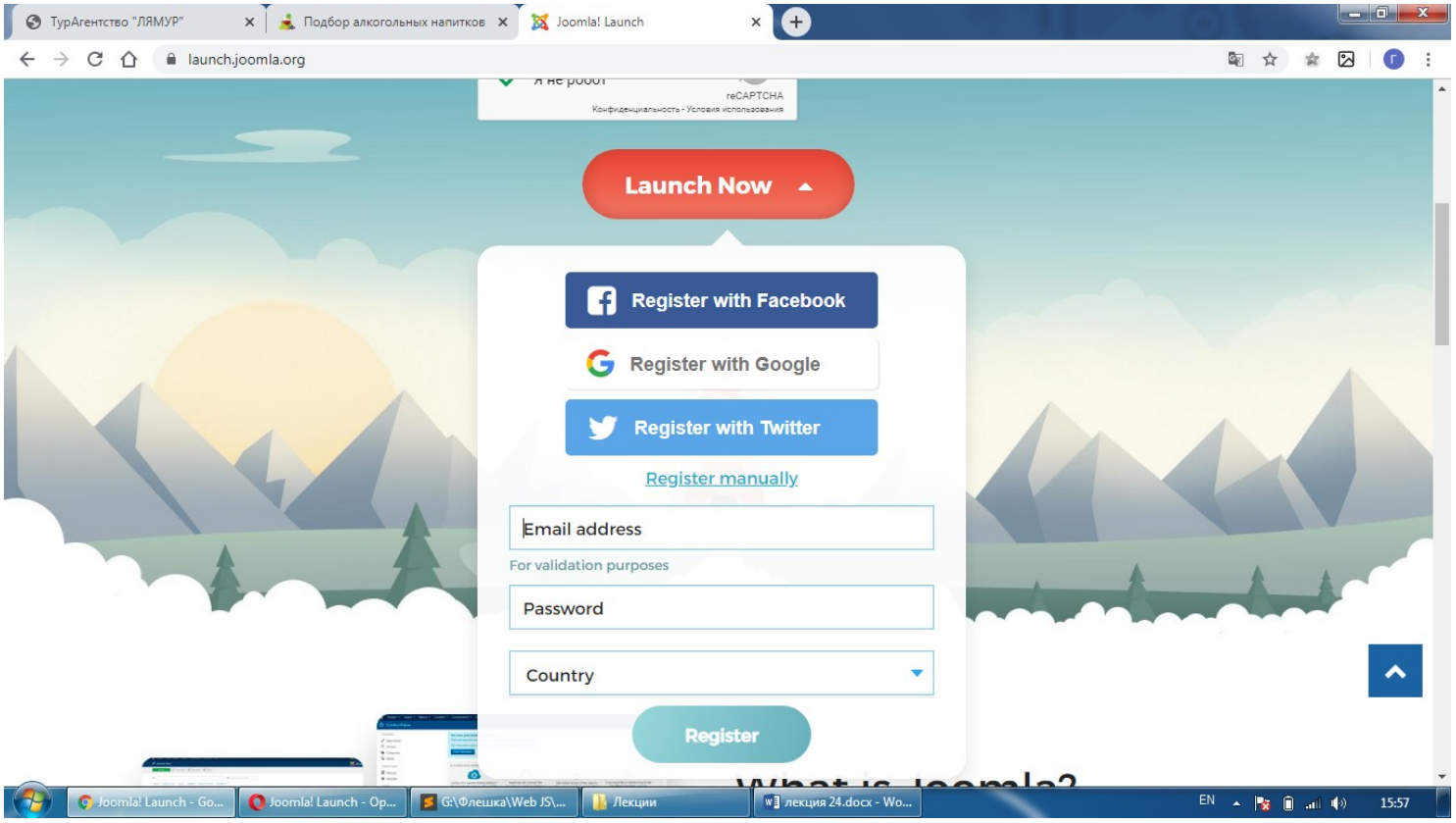

launch.joomla.org
Joomla!
Около
Скачать и расширить
Новости
сообщество
Поддержка
Разработчики
Запустите свою Joomla! сайт
Создайте полнофункциональные веб-сайты Joomla и бесплатно ознакомьтесь с лучшей в мире системой управления контентом.
Site name
joomla.com
SAVE 25%
Расширенные настройки
Добавить хостинг и поддержку
Я согласен с Условиями обслуживания и Политикой конфиденциальности
Помогите поддержать Joomla, включив маркетинговую кампанию провайдера launch.joomla.org
Я не робот
reCAPTCHA
Запустить сейчас


launch.joomla.org
Joomla!® About Download & Extend News Community Support Developers
Launch your Joomla! site
Build fully functional Joomla websites and experience
the world's best Content Management System for FREE.
Site name
joomla.com
SAVE 25%
Advanced Settings
Add Hosting and Support
I agree to the Terms of Service and Privacy Policy
Help support Joomla by opting into the launch.joomla.org provider's marketing campaign
Я не робот
reCAPTCHA
Launch Now

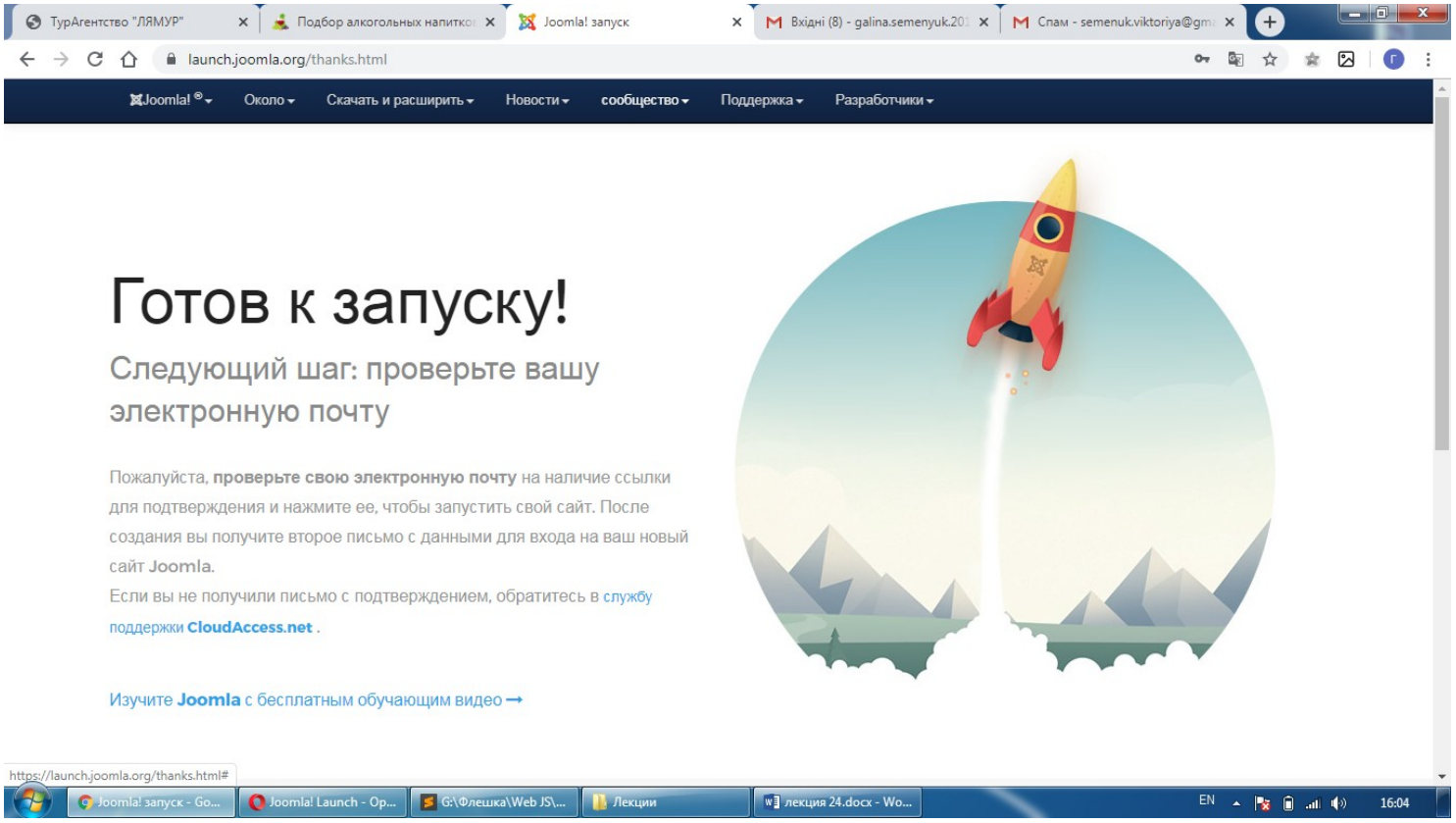

launch.joomla.org
Расширенные настройки
Добавить хостинг и поддержку
Я согласен с Условиями обслуживания и Политикой конфиденциальности
Помогите поддержать Joomla, включив маркетинговую кампанию провайдера launch.joomla.org
Я не робот
reCAPTCHA
Запустить сейчас
Зарегистрироваться чере
Зарегистрируйтесь в Google
Зарегистрируйтесь в Твит
Зарегистрируйтесь вручную

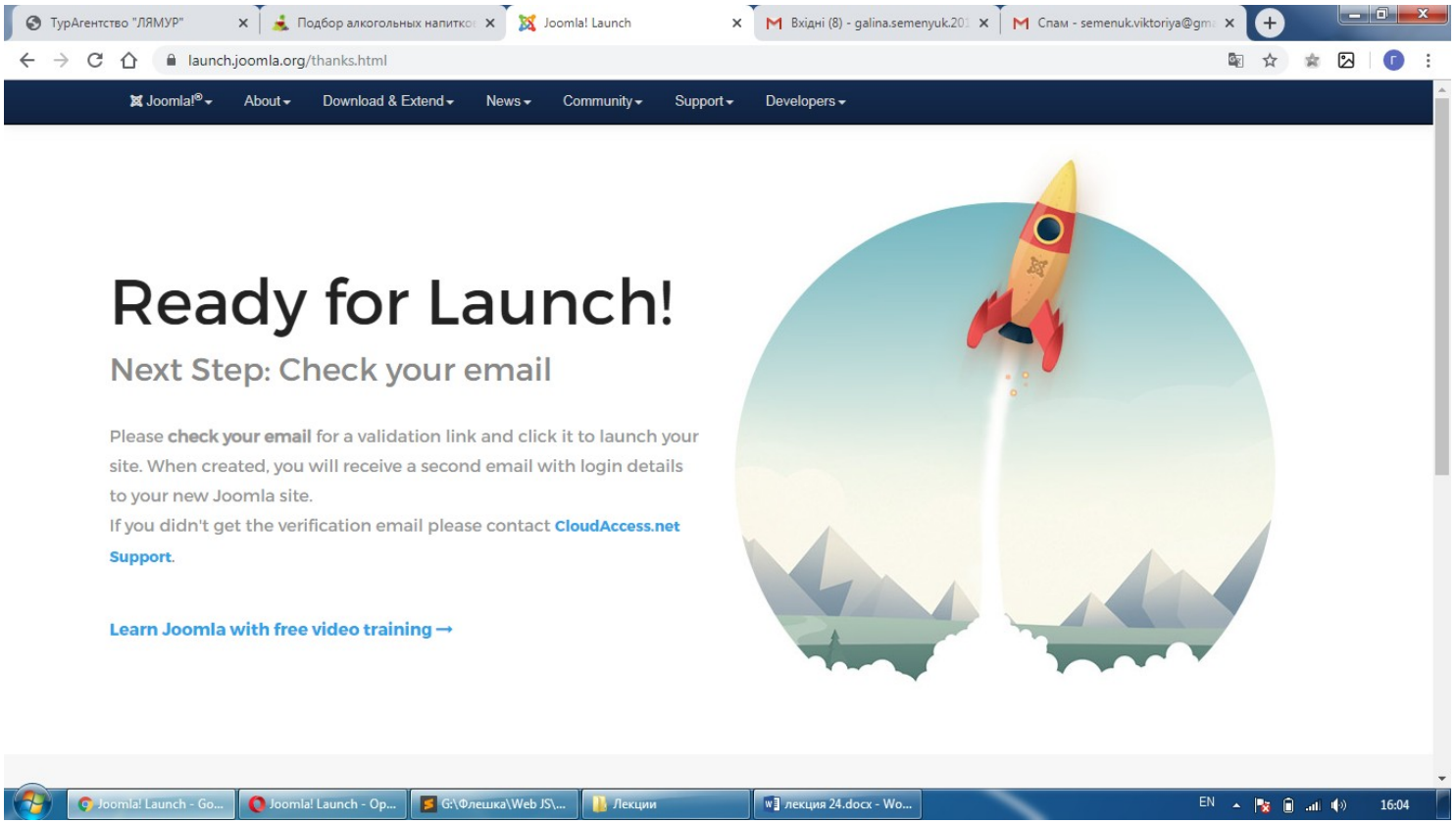

ТурАгентство "ЛЯМУР"
Подбор алкогольных напитков
Joomla! Launch
launch.joomla.org
Joomla!®
About
Download & Extend
News
Community
Support
Developers
Launch your Joomla! site
Build fully functional Joomla websites and experience
the world's best Content Management System for FREE.
web
joomla.com
SAVE 25%
Advanced Settings
Add Hosting and Support
I agree to the Terms of Service and Privacy Policy
Help support Joomla by opting into the launch.joomla.org
provider's marketing campaign
Я не робот
reCAPTCHA
Launch Now
Joomla! Launch - Go...
Joomla! Launch - Op...
G:\Флешка\Web JS\...
Лекции
лекция 24.docx - Wo...
EN
15:56


ТурАгентство "ЛЯМУР"
Подбор алкогольных напитков
Joomla! Launch
launch.joomla.org
Help support Joomla by opting into the launch.joomla.org
provider's marketing campaign
Я не робот
reCAPTCHA
Launch Now
Register with Facebook
Register with Google
Register with Twitter
Register manually
Joomla! Launch - Go...
Joomla! Launch - Op...
G:\Флешка\Web JS\...
Лекции
лекция 24.docx - Wo...
EN
15:56


launch.joomla.org
Launch Now
Register with Facebook
Register with Google
Register with Twitter
Register manually
Email address
For validation purposes
Password
Country
Register


launch.joomla.org
Launch Now
Register with Facebook
Register with Google
Register with Twitter
Register manually
semenuk.viktoriya@gmail.com
For validation purposes
Russian Federation
Register
What is Joomla?
Joomla is a user-friendly way for people all over the world to build
anything from basic websites to advanced web applications. It is


launch.joomla.org/thanks.html
Joomla! ®
Около
Скачать и расширить
Новости
сообщество
Поддержка
Разработчики
Готов к запуску!
Следующий шаг: проверьте вашу электронную почту
Пожалуйста, проверьте свою электронную почту на наличие ссылки для подтверждения и нажмите ее, чтобы запустить свой сайт. После создания вы получите второе письмо с данными для входа на ваш новый сайт Joomla.
Если вы не получили письмо с подтверждением, обратитесь в службу поддержки CloudAccess.net .
Изучите Joomla с бесплатным обучающим видео →
https://launch.joomla.org/thanks.html#


launch.joomla.org/thanks.html
Joomla!®
About
Download & Extend
News
Community
Support
Developers
Ready for Launch!
Next Step: Check your email
Please check your email for a validation link and click it to launch your site. When created, you will receive a second email with login details to your new Joomla site.
If you didn't get the verification email please contact CloudAccess.net Support.
Learn Joomla with free video training →


joomlaportal.ru
Joomlaportal.ru
Joomla! по-русски
Новости
Блоги
FAQ
Видео
Форум
О Joomla!®
Joomla! CMS — легкие решения, гибкие возможности
Скачать Joomla 3.9.13
Релиз:
Документация Joomla!
Искать...
Авторизация
Нажимая кнопку авторизации вы соглашаетесь на обработку персональных данных
Логин
Пароль
Запомнить меня
ВОЙТИ
Забыли логин?
Забыли пароль?
Горячая новость
Все новости
Скидки по зимней акции и релиз JURSSPublisher 3.33.20
Внимание! Представляем акцию по покупки лицензии, а также релиз JURSSPublisher 3.33.20. В новой версии мы поработали над улучшением интеграции с schema.org и оптимизацией сайта.
Новости Joomla!
JURSSPublisher
Используй потенциал социальных сетей во благо своего сайта
JUNewsUltra Pro 6
Best Joomla module for news, articles, YouTube and RSS with many options.
Fast performs and mostly optimization for over 100.000 articles!
Joomla-UA.org
Скидки по зимней акции и релиз RSSPublisher 3.33.20
Релиз модуля JUNewsUltra Pro 6.6.2
Релиз визуального редактора для Joomla CKEditor 4 (версия 5.16.4)
16:19