

## 1. Введение в веб-программирование. HTML-разметка

В ходе данного краткого курса мы изучим основы веб-программирования, которых, тем не менее, будет достаточно для полноценной разработки сайтов Интернета. Будут рассмотрены: HTML разметка веб-страниц и каскадные таблицы стилей (CSS), программирование клиентской части сайтов на JavaScript, программирование серверной части сайтов на PHP, использование системы управления базами данных MySQL, построение сайтов на базе систем управления содержимым (CMS) на примере CMS Joomla.

Курс рассчитан на слушателя, который уже знает один из языков программирования (например, C/C++, Pascal, Matlab) и понимает, что такое массивы, циклы и операторы условного перехода, а также какие бывают типы переменных.

В связи с тем, что данный курс является комплексным, нет необходимости рассказать вначале все о HTML, затем все о JavaScript и т.д. Поэтому, несмотря на то, что в целом содержание лекций является достаточно жестко структурированным, о многих вещах будет рассказываться именно в тот момент, когда слушатель будет наиболее готов к этому. Например, элементы форм HTML будут рассмотрены на 4-й лекции одновременно с основами PHP, которые позволят сразу же продемонстрировать обработку полученных от HTML-страницы данных формы на сервере.

**Первая лекция** посвящена вступлению в веб-программирование и основам HTML.

На **второй лекции** будут рассмотрены каскадные таблицы стилей, которые позволят нам отделить содержание веб-страницы (HTML) от ее оформления (CSS).

На **третьей лекции** мы изучим основы языка программирования JavaScript, узнаем, как с его помощью можно автоматически генерировать текст веб-страницы, обрабатывать события и “на лету” изменять текст веб-страницы.

**Четвертая лекция** посвящена основам языка PHP. Мы узнаем, в чем его преимущество перед JavaScript при генерировании текста страниц, а также как обрабатывать запросы HTML-страницы к серверу, рассмотрим отладку PHP-программ на локальном компьютере.

На **пятой лекции** мы узнаем, как пользоваться базами данных с помощью MySQL, как ими управлять, как сохранять в них данные, искать и извлекать по запросам.

На **шестой лекции** будет несколько глубже рассмотрен язык программирования JavaScript, в частности, работа с cookies, формирование запросов к серверу без перегрузки веб-страницы, библиотека JQuery.

**Седьмая лекция** будет посвящена некоторым тонкостям программирования на PHP, в частности, работе с регулярными выражениями, графикой, криптографическими функциями.

На **восьмой лекции** будут рассмотрены CMS и построение сайта на примере CMS Joomla.

Последняя, **девятая**, лекция будет посвящена тонкостям и трюкам веб-программирования, объектно-ориентированному программированию на PHP, вопросам продвижения сайтов и заработка на рекламе.

Все примеры страниц и программ в лекциях для удобства использования дублируются в отдельных файлах.

### 1.1. Базовые знания о сети Интернет

На рисунке 1.1 приведена упрощенная структурная схема взаимодействия компьютера пользователя с Интернетом или с локальной сетью.

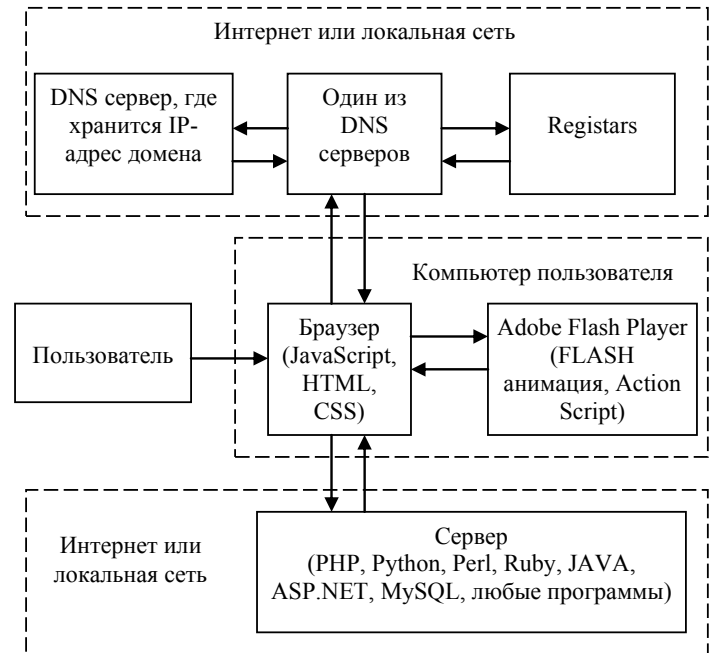


Рисунок 1.1 - Взаимодействие компьютера пользователя с сетью при загрузке и просмотре веб-страницы

Для просмотра веб-страниц в Интернете пользователи используют специальные программы, называемые **браузерами**. К наиболее распространенным относятся Mozilla FireFox, Opera, Internet Explorer (не рекомендуется использовать), Google Chrome.

Пользователь набирает в своем браузере адрес веб-страницы интернета. Если он набирает его в цифровом виде (**IP-адрес** вида 74.125.232.248), то браузер непосредственно связывается с сайтом Интернета, который находится по этому адресу. Если же адрес задан в текстовом виде, например «google.ru», то браузер связывается с **DNS-сервером** (прописанным в настройках сети компьютера), который заменяет текстовое имя на соответствующий ему IP-адрес.

Имя сайта в текстовом виде еще называют **доменным именем**. Так «google.ru» является доменом второго уровня в домене первого уровня «.ru». К наиболее распространенным доменным именам первого уровня относятся «.com», «.org», «.net», «.ru».

Каждый DNS сервер хранит данные (таблицу соответствия всех известных текстовых доменных имен цифровым IP-адресам). Это большой массив информации. Поэтому DNS сервера разбиты на несколько уровней, каждый из которых регулярно (примерно 2 раза в сутки) получает обновление от DNS-сервера старшего уровня. DNS сервера самого старшего уровня получают данные от **Registars** (компаний, отвечающих за регистрацию доменных имен). Registars за плату хранят информацию о том, какое доменное имя принадлежит какому человеку (или компании), а также регистрируют новые (незанятые) доменные имена.

Стоимость регистрации доменных имен второго уровня может составлять от 2 долларов за год для домена .info до 7-10 долларов за год для доменов .com, .org, .net. Доменные имена третьего уровня (например, ponom.pusku.com) не требуют регистрации в компаниях Registars и иногда раздаются бесплатно. Каждый год требуется продлевать регистрацию доменного имени. Такая процедура называется **renew** и обычно стоит дороже, чем регистрация (плата за первый год).

Процедура регистрации доменного имени состоит в выполнении следующих шагов. Среди компаний Registrars выбираем ту, чьи расценки на регистрацию и продление доменных имен в выбранной зоне являются наиболее подходящими. На сайте этой компании проверяем, чтобы то доменное имя, которое мы регистрируем, было не занято в данный момент. Платим деньги (из Украины можно заплатить через Интернет с помощью любой зарплатой карточки). На сайте компании Registrars указываем IP-адрес DNS-сервера (или серверов), который знает, где физически находится веб-страница, соответствующая нашему домену. При условии, что DNS-сервер правильно настроен и страница существует, она станет доступной для пользователей всего мира (которые наберут в своем браузере наше только что зарегистрированное доменное имя) приблизительно через день.

Человек, которому нужно создать свой сайт, может организовать сервер, где будут находиться веб-страницы и данные сайта, самостоятельно. Для этого требуется выделенный ip-адрес, зарегистрированное доменное имя и компьютер, подключенный к сети Интернет. Другим путем создания сайта является заказ платного или бесплатного **хостинга** у компаний, предоставляющих такие услуги (в этом случае требуется только зарегистрированное доменное имя или же бесплатно предоставляемое некоторыми хостингами доменное имя третьего уровня).

Хостинг может быть виртуальным (это дешевле). В таком случае на одном компьютере одновременно работают несколько сайтов разных владельцев. Хостинг может быть **выделенным** (это намного дороже). В этом случае для сайта выделяется отдельный компьютер. Для **виртуального хостинга** на веб-страницу накладываются ограничения по использованию ресурсов процессора (чтобы она не замедляла работу чужих сайтов).

Основные характеристики хостинга:

- Выделенный или виртуальный хостинг;
- Наличие и величина ограничений на объем **трафика**;
- Объем выделяемого места на диске;
- Наличие поддержки PHP (важно какая именно версия PHP установлена) и других языков программирования;
- Наличие поддержки MySQL и сколько баз данных разрешено создать;
- Количество доменов, которые можно привязать к своему **аккаунту** на хостинге.

Большинство хостингов предоставляют доступ к сайту и управление им как через **веб-интерфейс**, так и через **ftp**.

После того, как хостинг настроен, к нему привязано доменное имя, стартовая веб-страница сайта создана, пользователи могут заходить на этот сайт. Если набрать в браузере только имя сайта без указания конкретной веб-страницы, например, <http://ponomarenko.info>, то по умолчанию будет загружена страница index.html или index.htm или index.php (если хотя бы одна из страниц с таким именем есть на сайте).

## 1.2. Основы HTML

Цель данной лекции - рассказать о основах HTML, не претендуя на полную справочную информацию (в качестве справочника по тегам HTML можно порекомендовать, например, сайт <http://htmlbook.ru/html>).

**HTML** (HyperText Markup Language) переводится как “язык разметки гипертекста”. Под разметкой здесь понимается структурирование, форматирование и оформление документов.

HTML-страница представляет собой текстовый файл, который можно набрать в любом текстовом редакторе (например, в редакторе notepad++) и, который имеет расширение \*.html или \*.htm.

HTML-страница состоит из элементов, которые начинаются и заканчиваются специальными пометками - тегами. **Теги** - это ключевые слова, взятые в угловые скобки (символы меньше “<” и “больше” “>”). Например, тег <HR> означает, что в этом месте веб-страницы нужно вставить горизонтальную линию. Большинство тегов в отличие от тега <HR> применяются парами (обрамляют фрагмент текста). Например “<B>Привет, мир!</B>” означает, что текст “Привет мир” должен быть выделен жирным шрифтом. Здесь тег <B> означает включение жирного шрифта, а тег </B> - окончание действия тега <B>. В данном случае говорят, что тег <B> - открывающий, а тег </B> - закрывающий.

В общем случае открывающий тег заключается в угловые скобки вот так: <тег>, а закрывающий тег заключается в такие же скобки, но перед ключевым словом ставится еще наклонная черта “/” вот так: </тег>.

Веб-страница просматривается в браузере, который интерпретирует встречающиеся теги и выполняет нужные действия по разметке текста. Если какой-то тег неизвестен данному браузеру, то он просто игнорирует его. Язык HTML является стандартизированным и в теории действия всех браузеров должны соответствовать стандарту, однако на практике одна и та же веб-страница в разных браузерах может выглядеть по-разному. Причем отличия могут быть очень сильными и, иногда, даже противоречить стандарту (особенно это касается браузера Internet Explorer). Поэтому разработчик веб-страницы должен тестировать ее (предварительно просматривать) по возможности на всех основных браузерах.

На рисунке 1.2 приведен пример текста самой простой веб-страницы (файл ex1\_01.html) и окна браузера при просмотре данной веб-страницы (в любом файловом менеджере кликните по файлу страницы мышкой и она откроется в Интернет-браузере).

```
<html>
<body>
Здравствуй, цифровой мир!
</body>
```

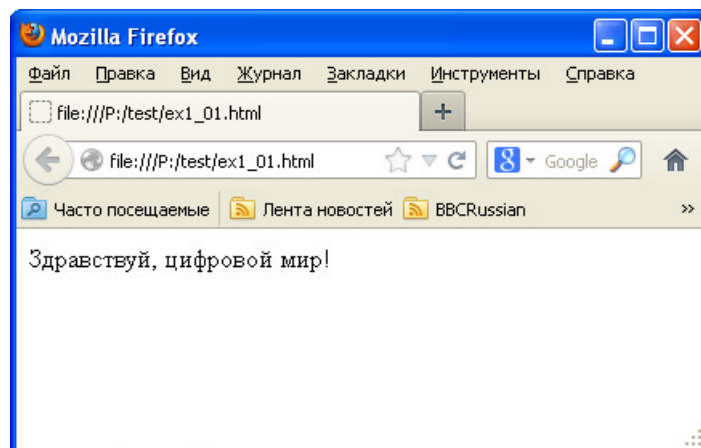


Рисунок 1.2 - простая веб-страница и окно браузера для нее

В данном примере есть только самые необходимые теги: <html> - должен идти самым первым тегом веб-страницы, и

`<body> .. </body>` - тег, который обрамляет текст веб-страницы.

Пока все очень просто. Далее будем постепенно добавлять на нашу страницу новые элементы. Но вначале проиллюстрируем одну важную особенность HTML. Изменим текст страницы следующим образом (файл ex1\_02.html) и посмотрим, что будет (рисунок 1.3).

```
<html>
<body>
Здравствуй,
цифровой мир!
</body>
```

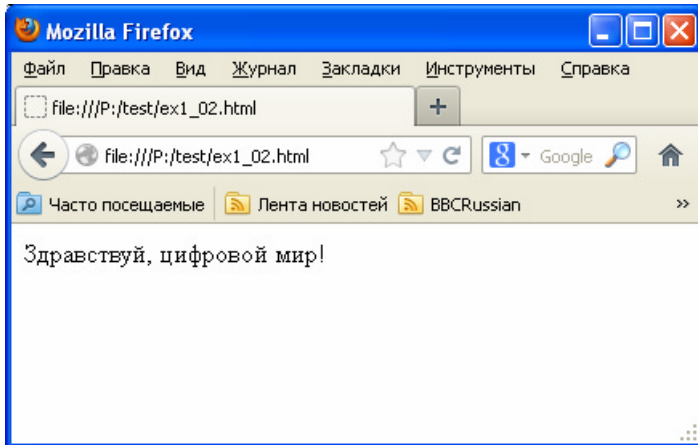


Рисунок 1.3 - измененная веб-страница и окно браузера

Как видно, в тексте страницы фразу “Здравствуй, цифровой мир!” мы разделили на две строки и вставили внутри нее несколько пробелов, однако браузер все эти изменения проигнорировал. Страница в окне браузера выглядит точно так же, как и раньше.

Дело в том, что HTML игнорирует все переводы строки внутри веб-страницы и все дополнительные (свыше одного) пробелы между словами. Если мы хотим перенести часть текста на другую строку, то должны использовать одиночный тег **<BR>**.

```
<html>
<body>
Здравствуй, <br> цифровой
мир!
</body>
```

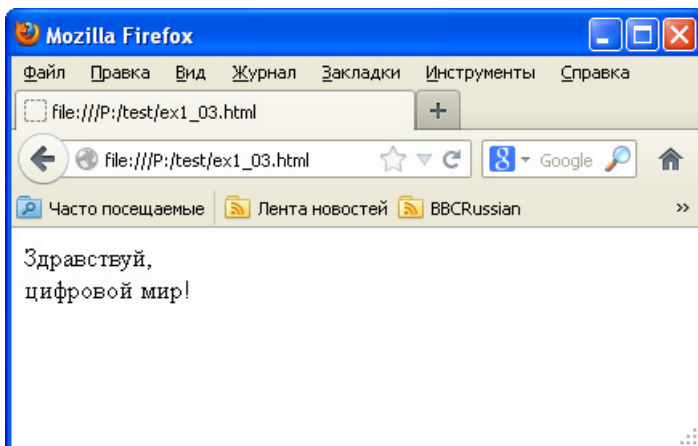


Рисунок 1.4 - страница с тегом <BR>

Например, так, как это показано на рисунке 1.4 (файл `ex1_03.html`).

Как видим, текст в HTML-странице может находиться и на одной строке, но, если браузер встречает тег `<BR>`, то переносит текст, следующий за этим тегом, на следующую строку.

Чтобы добавить лишние пробелы в текст, нужно использовать **спецсимволы** HTML. Спецсимволы HTML начинаются всегда с “&” и заканчиваются точкой с запятой “;”. Так, чтобы вставить в текст дополнительный пробел, нужно добавить в текст спецсимвол **&nbsp;**. Если же мы хотим добавить в текст, например, греческую букву кси, то нужно добавить в текст спецсимвол **&xi;**. Таблицы всех спецсимволов HTML можно найти в справочниках в Интернете. Добавим в нашу страницу (файл ex1\_04.html) несколько дополнительных пробелов и текст с буквой кси (см. рисунок 1.5).

```
<html>  
<body>  
Здравствуй, <br> цифровой  
    &nbsp;&nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&мир! <br>  
А вот так выглядит греческая буква  
кси: &x_i;  
</body>
```

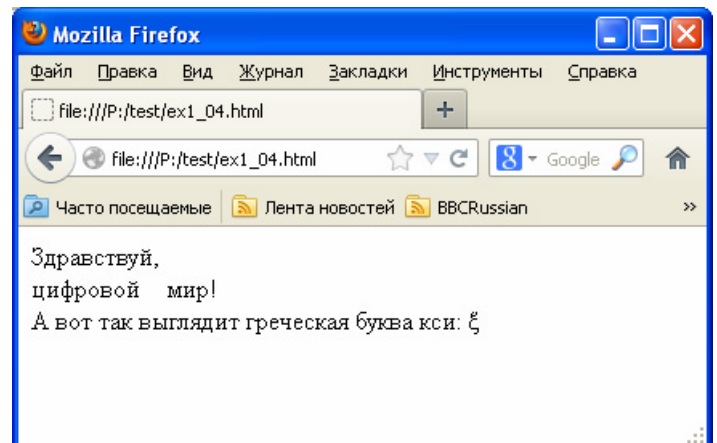


Рисунок 1.5 - страница с дополнительными пробелами и греческой буквой кси

Обратите внимание, что в браузере увеличился промежуток между словами “цифровой” и “мир” - это сказались вставленные нами на страницу дополнительные пробелы \$nbsp\$;

Обратите также внимание на то, что на странице теперь два перевода строки (тега `<br>`). Причем первый раз тег набран заглавными буквами, а второй раз - прописными. Оба раза браузер правильно его воспринял. Это иллюстрация того, что HTML инвариантен к регистру букв в тегах (все равно, маленькими или большими буквами они набраны). Поэтому пишите теги как удобнее лично Вам - маленькими или большими буквами.

Вот, собственно, и все особенности HTML - отсутствие перевода строк без специальных тегов, игнорирование повторяющихся пробелов и инвариантность к регистру букв в названиях тегов.

Рассмотрим теперь несколько тегов, чаще всего используемых на практике: `<head>`, `<title>`, `<a>`, `<img>`, `<table>`, `<tr>`, `<td>`, `<h1>`, `<p>`, `<font>`.

На рисунке 1.6 приведен пример использования тегов `<head>` и `<title>` (файл `ex1_05.html`).

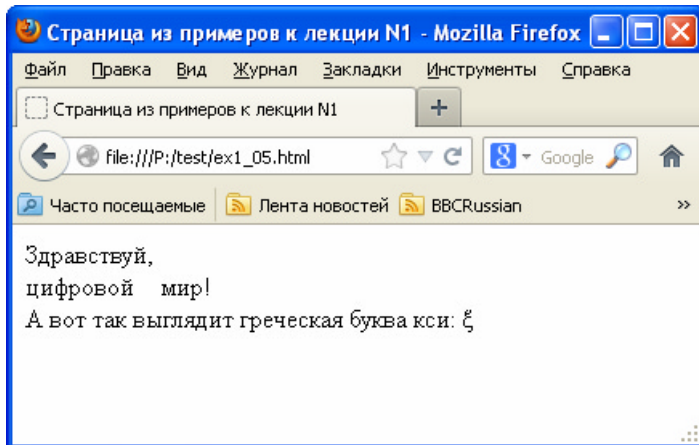
[illegible]

Рисунок 1.6 - Иллюстрация тегов `<head>` и `<title>`

Тег `<head> .. </head>` предназначен для хранения служебных элементов веб-страницы, цель которых — помочь браузеру в работе с данными. Например, это может быть информация о кодировке страницы или информация для поисковых систем, облегчающая индексацию страницы в их базах данных.

Внутри тега `<head>` могут использоваться другие теги. В данном примере это тег `<title>..</title>` , который задает заголовок страницы, который отображается в заголовке окна браузера (см. рисунок 1.6).

Обратите также внимание, что в данном примере для удобства чтения человеком текста нашей веб-страницы мы вставили несколько пустых строк (после тега `<html>`, перед и после тега `<body>`). Браузер просто игнорирует эти пустые строки и их может быть сколько угодно много.

В следующем примере (файл ex1\_06.htm) в заголовок добавлена информация еще и об иконке страницы (отображается в заголовке вкладки браузера, в закладках, в журнале посещений). Иконка должна иметь размеры 16x16 пикселей и может быть как статичной (в формате \*.ico), так и анимированной (в формате \*.gif). В данном примере в заголовок добавлены иконки обоих типов (см. рисунок 1.7).

Информация об иконках страницы передается браузеру внутри тега `<head> .. </head>` с помощью одиночного тега `<link>`. Этому тегу не нужно закрывающего тега, однако впервые мы сталкиваемся с необходимостью устанавливать **атрибуты** тега, которые задаются внутри него.

```
<html>

<head>

<title>Страница из примеров к лекции
N1</title>

<link rel="shortcut icon"
href="favicon.ico">

<link rel="icon"
href="animated_favicon.gif"
type="image/gif" >

</head>

<body>

Здравствуй, <BR> цифровой
      мир! <br>
А вот так выглядит греческая буква
кси: &xi;

</body>
```

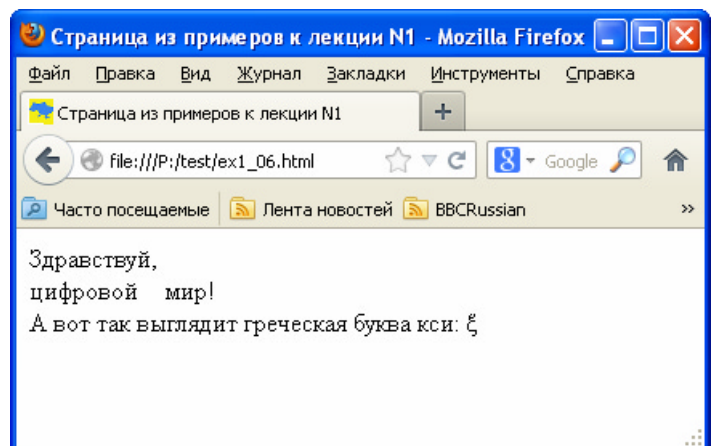


Рисунок 1.7 - Иллюстрация использования тега <link>

Стандартный формат атрибутов любого тега следующий:

```
<тег имя_атрибута1="значение атрибута"  
имя_атрибута2="значение атрибута">
```

В данном примере тег `<link>`, который используется нами, чтобы сообщить браузеру ссылку на иконку, имеет три атрибута: **rel**, **href** и **type**. Значение "icon" атрибута **rel** сообщает браузеру, что данный линк указывает на иконку, значение "animated\_favicon.gif" атрибута **href** задает путь к файлу иконки и имя этого файла, а значение "image/gif" атрибута **type** указывает браузеру, что изображение иконки представлено в формате gif.

В следующем примере рассмотрим, как добавлять на веб-страницы то главное, что отличает их от обычного текста - ссылки на другие страницы. Для этого используется тег `<a> .. </a>`.

Выбросим из нашего примера все лишнее и оставим только две ссылки: на сайт википедии и на веб-страницу с предыдущим примером (файл `ex1_07.html`). Результат показан на рисунке 1.8.



```
<html>

<body>

<a href="http://www.wikipedia.org">
Ссылка на сайт википедии
</a> <br>

А это ссылка на <a
href="ex1_06.html" target="_blank">
предыдущий пример</a>

</body>
```

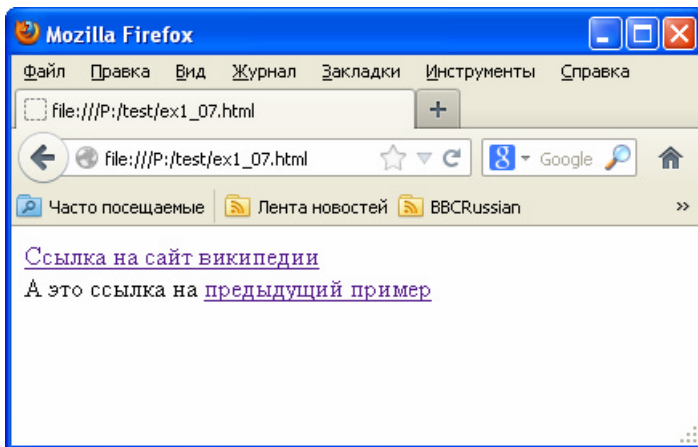


Рисунок 1.8 - Иллюстрация тега &lt;a&gt;

Тег <a> .. </a> предназначен для создания ссылок на другие веб-страницы, ссылок на файлы данных и ссылок на позиции внутри веб-страниц (**якоря**).

В данном примере обе ссылки ведут на веб-страницы. При этом адрес страницы задается в уже знакомом нам атрибуте **href**.

Ссылки могут быть абсолютными (в href указывается полный путь к веб-странице) или относительными (указывается путь относительно места, где лежит веб-страница, в которую встроена эта ссылка). В данном примере к сайту википедии указан полный путь, а к веб-странице с предыдущим примером - относительный.

Допустим, в папке, где лежит наша веб-страница, создана еще одна папка с названием "mydocs", в которой содержится файл iktm2012.pdf. Чтобы включить в веб-страницу ссылку на этот документ, нужно указать относительный путь следующим образом: href="mydocs/iktm2012.pdf".

Обратите внимание, что в примере используется еще один важный атрибут тега <a> - **target**. Этот атрибут определяет, где будет открываться данная ссылка: в этом же окне браузера или в новом окне браузера (есть и другие варианты, которые изучите самостоятельно). В данном примере значение "\_blank" означает, что ссылка на веб-страницу с предыдущим примером будет открыта в новом окне.

Чтобы организовать ссылку на **якорь** (на какое-то место в середине веб-страницы), нужно в атрибуте href вместо ссылки указать имя якоря, предваряемое символом "#":

```
<a href="#pub">Publications</a>
```

В том же месте страницы, куда должен быть осуществлен переход по ссылке, должен быть помещен сам якорь - ссылка без атрибута href, но с атрибутом **name**:

```
<a name=pub>Publications</a>
```

В данном примере именем якоря является "pub". Якорь ничем не выделяется на веб-странице. В данном примере это будет просто текстовое слово "Publications" - оно не подчеркивается, подобно ссылке.

Рассмотрим теперь тег <img> (файл ex1\_08.html). Это одиночный тег и он используется, чтобы вставить изображение на веб-страницу (см. рисунок 1.9).

```
<html>

<body>

 ХАИ

<br>
Вы видели когда-нибудь главный
корпус ХАИ?

</body>
```

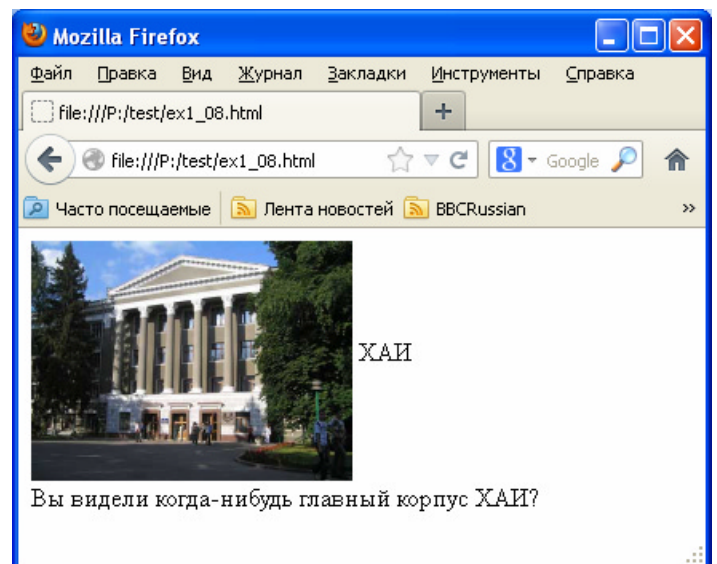


Рисунок 1.9 - Иллюстрация тега &lt;img&gt;

Атрибут **src** задает путь к файлу изображения (он может находиться как в той же папке, где лежит страница, так и в любом месте Интернета). Обращаем внимание, что для файлов Интернета важно, какие буквы используются в названии файла, большие или маленькие. Если файл назван "myFoto.JPG", то и в пути к файлу нужно писать "myFoto.JPG". Если написать "myfoto.jpg" или "myFoto.jpg", то браузер не найдет этот файл.

Атрибуты **width** и **height** задают соответственно ширину и высоту изображения (не обязательно сохранять пропорции). Можно задать эти атрибуты в процентах, тогда размеры изображения будут пропорциональны размерам всего окна браузера или родительского элемента (если изображение, например, находится внутри тега <div>, который мы рассмотрим на следующей лекции).

Атрибут **alt** задает текст, который появится на странице, пока изображение грузится.

Атрибут **title** (один из универсальных атрибутов, имеющихся у большинства тегов) задает текст, который будет

показан при наведении на изображение курсора мышки.

И, наконец, атрибут **align** задает выравнивание последующего текста (в данном примере - "ХАИ") относительно изображения.

На рисунке 1.10 приведен пример использования тегов `<table>`, `<tr>` и `<td>`, которые позволяют добавлять таблицы на веб-страницу (файл ex1\_09.html).

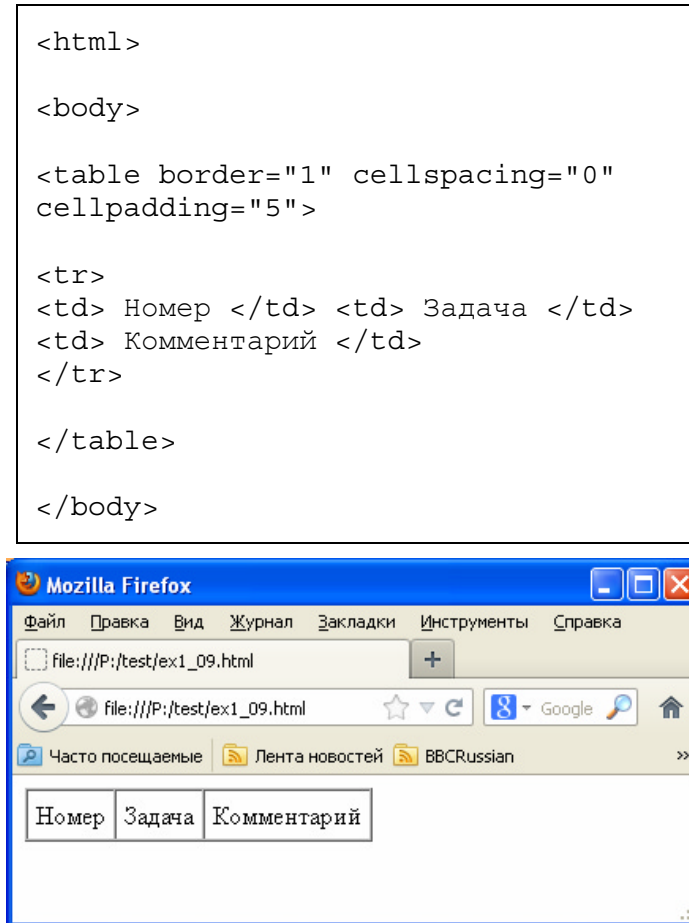


Рисунок 1.10 - Иллюстрация тегов `<table>`, `<tr>` и `<td>`

Тег `<table> .. </table>` обрамляет таблицу. В его атрибутах указываются параметры таблицы. В частности, в данном примере **border="1"** задает толщину рамки таблицы в пикселях (при **border="0"** рамка будет невидимой). Атрибут **cellspacing** задает отступ между ячейками таблицы, а атрибут **cellpadding** - отступ от краев ячейки до текста.

Каждая строка таблицы должна обрамляться тегом `<tr> .. </tr>`. Внутри этого тега вставляются ячейки таблицы, каждая из которых должна обрамляться тегом `<td> .. </td>`.

В приведенном простом примере таблица состоит из одной строки, в которую вставлены три ячейки.

Приведем пример более сложной таблицы. На рис. 1.11 приведена таблица, состоящая из нескольких строк, причем некоторые соседние ячейки таблицы объединены в более крупные ячейки (файл ex1\_10.html).

Эффект объединения ячеек достигается за счет использования атрибутов **rowspan** и **colspan** тега `<tr>`.

Атрибут **rowspan** задает число ячеек таблицы, объединяемых по вертикали, а атрибут **colspan** задает число клеточек таблицы, объединяемых по горизонтали.

Если, например, задано значение **rowspan="2"**, то в следующей строке таблицы должно быть на одну ячейку меньше. Аналогично и для **colspan="3"**, только в данной строке указывается на две ячейки меньше, чем в предыдущей и следующей строках.

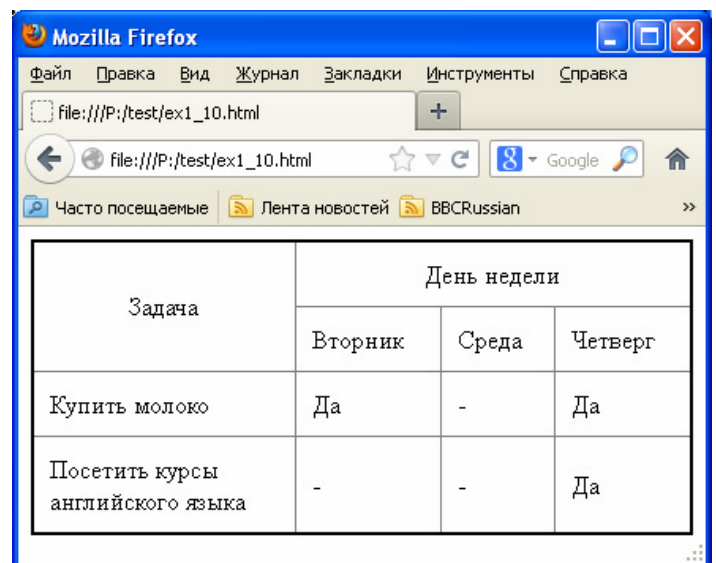
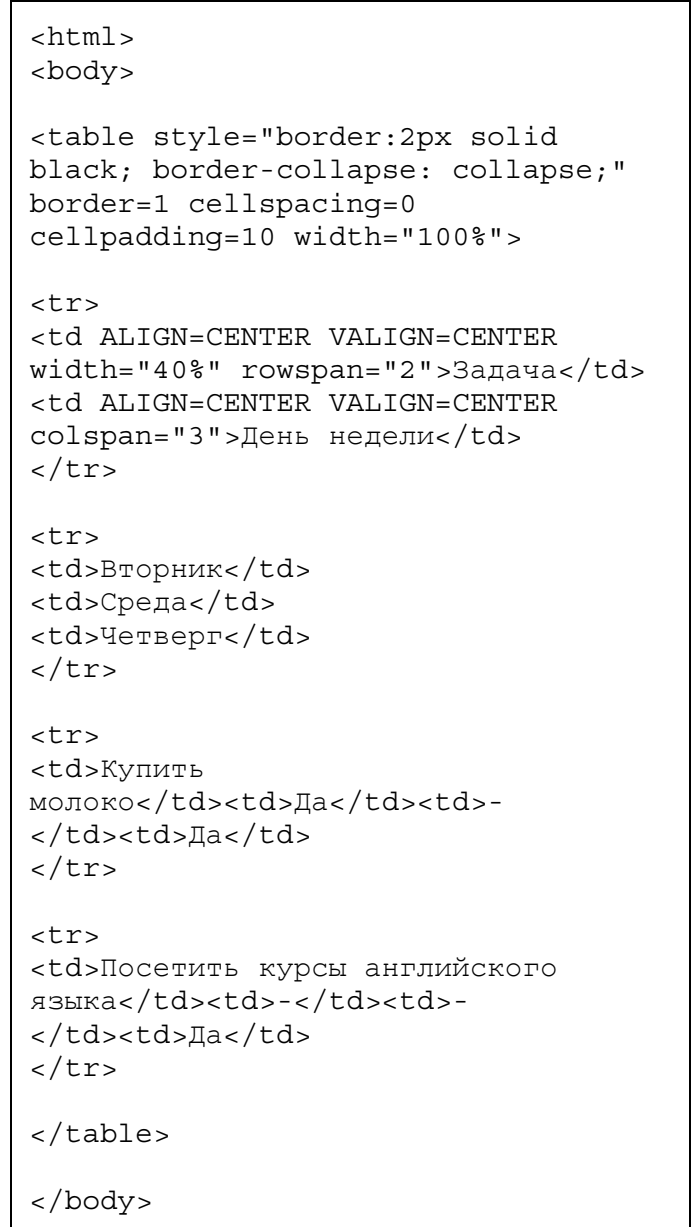


Рисунок 1.11 - Сложная таблица

В этом примере есть еще несколько атрибутов, ранее нами не использовавшихся. Атрибут **width="100%"** тега `<table>` растягивает таблицу на всю ширину страницы. Этот

же атрибут в теге <td> с текстом “Задача” задает для этой ячейки ширину в 40% от ширины всей таблицы.

Атрибуты **align** и **valign** тега <td> задают горизонтальное и вертикальное выравнивание текста в данной ячейке. Для тех ячеек, где эти атрибуты не установлены, текст выравнивается по левому краю ячейки.

И, наконец, атрибут **style** (его мы подробно рассмотрим на следующей лекции) задает стиль оформления таблицы (жирную рамку и одинарные линии внутри таблицы вместо двойных).

Последним примером лекции (file ex1\_11.html) проиллюстрируем действие некоторых тегов, использующихся при оформлении текста (см. рисунок 1.12).

```
<html>
<body>

<h1>
Цифровой мир
<hr> </h1>

<p>Интернет является <font
color="#ee0000" face="Verdana"
size="6">самым большим</font>
сегментом телекоммуникаций.</p>

<p style="background-
color:#aaffaa">Каждый день мы
пользуемся Интернетом, мобильной
связью, смотрим телевизор.
Телекоммуникации делают нашу жизнь
качественнее.</p>

</body>
```

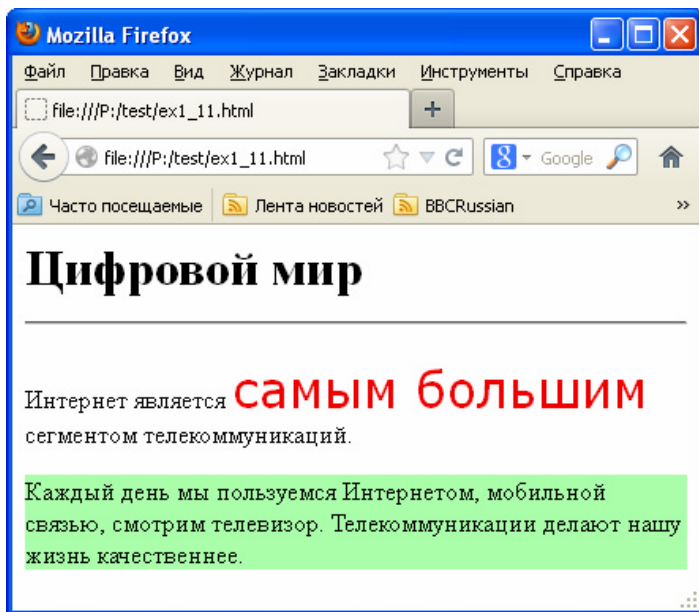


Рисунок 1.12 - Иллюстрация тегов <h1>, <hr>, <font>, <p>

В данном примере тег <h1> .. </h1> сообщает браузеру, что данный текст является заголовком с самым большим размером букв (есть еще теги <h2>, <h3> и т.д.).

Тег <hr> вставляет горизонтальную линию на следующей строке.

Тег <p> .. </p> выделяет какой-то текст в виде одного абзаца. Для этого абзаца можно задать, например, общий стиль (зеленый фон для второго абзаца).

Тег <font> .. </font> меняет шрифт для заданного фрагмента текста. Атрибут **face** задает имя шрифта, атрибут **color** задает цвет шрифта, а атрибут **size** - размер шрифта (число от 1 до 7).

Крайне не рекомендуется пользоваться тегом <font> на практике. Этого же самого можно добиться с помощью каскадных таблиц стилей (почему так будет правильнее и целесообразнее, рассмотрим на следующей лекции).

Последним примером лекции (file ex1\_12.html) проиллюстрируем действие тега <pre>, использующегося, в частности, при оформлении текста программ (см. рисунок 1.13).

```
<html>
<body>
<pre>
for i:=1 to 50 do
begin
  x:=x+i;
  writeln(x);
end;
</pre>
</body>
```

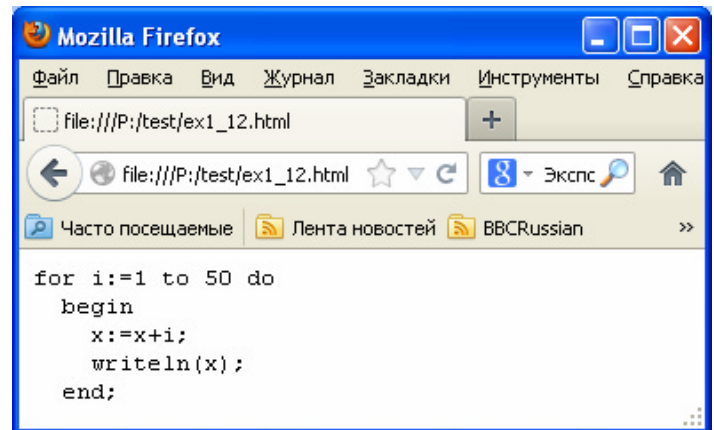


Рисунок 1.13 - Иллюстрация тега <pre>

Как видим, внутри тега <pre> .. </pre> браузер “забывает” про HTML-разметку и выводит текст с переводами строки, всеми пробелами и шрифтом, в котором у всех букв одинаковая ширина (**моноширинный** шрифт).

На этом первая лекция закончена.

Пожалуйста, самостоятельно разберитесь по справочнику (например, сайту <http://htmlbook.ru>) с назначением и особенностями использования следующих тегов: <meta>, <base>, <frame>, <iframe>, <ul>, <li>, <span>. Выясните, чем отличаются теги <div> и <span>.

Для изученных в ходе лекции тегов самостоятельно ознакомьтесь с возможными атрибутами этих тегов. Например, у тега <br> имеется атрибут **clear**. Самостоятельно изучите его. При изучении не стесняйтесь создавать разные варианты тестовых страниц с разными значениями атрибутов и смотреть, что получится.

Теги <div>, а также теги, относящиеся к элементам форм (<form>, <input>), мы рассмотрим на следующих лекциях по CSS, JavaScript и PHP.



## 2. Каскадные таблицы стилей

В данной лекции мы рассмотрим основы CSS. Полный перечень всех свойств CSS с примерами их использования Вы изучите сами, по мере того, как они Вам понадобятся на практике при верстке веб-страниц. В этом Вам поможет любой из справочников по CSS в сети Интернет (можно порекомендовать, например, сайт <http://htmlbook.ru/css>).

Каскадные таблицы стилей (Cascading Style Sheets, CSS) предназначены для облегчения оформления web-страниц. Основное их назначение - отделение содержания веб-страниц (HTML) от их оформления (CSS), которое может быть единым для нескольких веб-страниц, и изменения в котором легко вносить.

Рассмотрим пример (файл ex2\_01.html), который поможет нам понять вышесказанное (см. рисунок 2.1).

```
<html>
<body>
<font face="Verdana" size="4">
<b><i>Браузер</i></b></font> -
программное обеспечение для просмотра
веб-сайтов.<br>

<font face="Verdana" size="4">
<b><i>Веб-сервер</i></b></font> -
Веб-сервер - это сервер, принимающий
HTTP-запросы от клиентов, обычно веб-
браузеров, и выдающий им HTTP-
ответы.<br>

<font face="Verdana" size="4">
<b><i>JavaScript</i></b></font> -
язык сценариев для придания
интерактивности веб-страницам.<br>
</body>
```

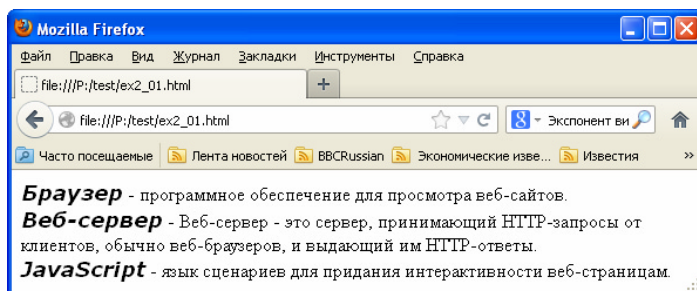


Рисунок 2.1 - Пример веб-страницы без CSS

Часть текста на страничке выделено жирным наклонным шрифтом. Теги `<font>` и `<b>` нам уже встречались, а новый тег `<i> .. </i>` делает текст наклонным (*italic*).

У приведенного примера два недостатка. Во-первых, это громоздкость текста. Одни и те же теги с одними и теми же атрибутами (`<font face="Verdana" size="4">` `<b>` `<i>`) повторяются три раза. Причем, если на страничке будут определения не трех, а двухсот терминов, то эти теги придется повторять 200 раз. Все это приведет к существенному увеличению размера файла (а, значит к замедлению загрузки этой страницы в браузере), а также к ухудшению читаемости и редактируемости текста страницы.

Вторым недостатком является сложность внесения правок в такую страницу. Например, мы хотим заменить шрифт Verdana на шрифт Arial. Нам придется менять его в

трех местах. Если же таких определений терминов на странице будет не три, а двести, то нам придется вносить уже двести изменений. Это утомительно и, к тому же, чревато внесением опечаток в текст страницы.

Избежать обоих недостатков помогает использование CSS. Все элементы оформления (шрифт, цвет шрифта, жирность шрифта, цвет фона, вид границ и т.п.) выносятся в отдельное место веб-страницы и им присваивается какое-то имя. А в том месте веб-страницы, где нужно вывести текст с таким стилем оформления, помещается, например, тег `<span> .. </span>` или `<div> .. </div>`, в атрибутах которого указывается имя стиля и внутри которого помещается выводимый на экран текст.

### 2.1. Способы вставки стилей на веб-страницу

На рисунке 2.2 приведена веб-страница, выводящая такой-же текст, как и в предыдущем примере, но оформленная с помощью стилей (файл ex2\_02.html).

```
<html>
<body>

<style>
.papa {
  font-family:Verdana;
  font-weight:bold;
  font-size:large;
  font-style:italic;
  display:inline;
}
</style>

<div class="papa">Браузер</div> -
программное обеспечение для просмотра
веб-сайтов.<br>

<div class="papa">Веб-сервер</div> -
Веб-сервер - это сервер, принимающий
HTTP-запросы от клиентов, обычно веб-
браузеров, и выдающий им HTTP-
ответы.<br>

<div class="papa">JavaScript</div> -
язык сценариев для придания
интерактивности веб-страницам.<br>
</body>
```

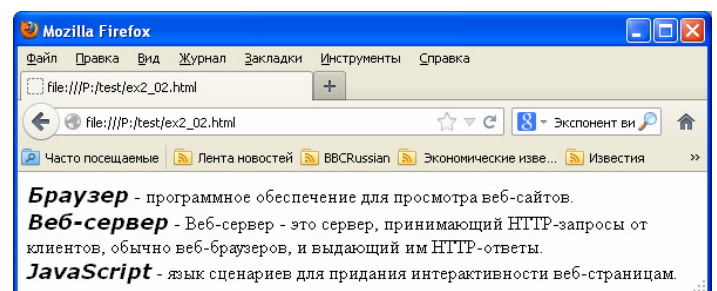


Рисунок 2.2 - Такая же веб-страница, оформленная с CSS

Существует три разных способа добавления информации о стилях на веб-страницу, которые будут рассмотрены в



этом параграфе: внутри тега `<style>..</style>`, в атрибуте **style** нужного тега, в отдельном файле с описанием стилей.

В примере на рисунке 2.2 используется первый способ. Внутри тега `<style>..</style>` описываются стили (один или несколько), которые затем используются в оформлении элементов веб-страницы. Для этого в теге, который требуется оформить в соответствии с данным стилем, в атрибуте **class** указывается имя стиля.

В данном примере именем стиля является “papa” (любое слово, выбранное нами). В фигурных скобках перечисляются **свойства** стиля, которые мы хотим задать, и их **значения**. В общем виде формат описания стиля выглядит так:

```
.ИМЯ_СТИЛЯ {
    свойство_1 : значение;
    свойство_2 : значение;
}
```

В данном примере **font-family**, **font-weight**, **font-size**, **font-style**, **display** - это свойства стиля, а **Verdana**, **bold**, **large**, **italic**, **inline** - их значения.

Если имя стиля - это любое имя, придуманное нами, то для имен свойств и их значений существуют специальные имена. В данной лекции мы рассмотрим некоторые из них, а полный перечень служебных имен свойств стилей и их значений Вы можете найти самостоятельно в справочнике. Не старайтесь сразу изучить весь справочник. Изучайте новые свойства стилей постепенно, по мере практической необходимости в них.

В данном примере **font-family** задает имя шрифта, **font-weight** - толщину шрифта, **font-size** - размер шрифта, **font-style** - наклонность. Смысл свойства **display** рассмотрим чуть позже.

Итак, в тегах `<style>..</style>` мы описали стиль, который назвали “papa”. Каким образом назначить его какому-либо тексту? Для этого чаще всего используется тег `<div>..</div>`. Тег задает прямоугольную область (блок текста), внутри которой, кроме текста, может находиться что угодно - изображения, ссылки, таблицы и даже другие блоки, ограниченные тегами `<div>..</div>`.

Чтобы на весь текст внутри тегов `<div>..</div>` распространялся требуемый нам стиль оформления, нужно в атрибуте **class** тега `<div>` указать имя стиля (см. рисунок 2.2). В данном случае это выглядит так:

```
<div class="papa">
```

Стиль будет распространяться на весь текст до тех пор, пока не встретится закрывающий тег `</div>`.

Данный пример сразу показывает нам преимущества использования стилей. Во-первых, если мы захотим внести изменения в оформление страницы, то достаточно это сделать только в одном месте, а не во многих, как раньше. Кроме того, текст веб-страницы стал более структурированным и читаемым.

Добавим в оформление зеленый фон (см. рисунок 2.3) и изменим шрифт с Verdana на Helvetica (файл ex2\_03.html).

Как видим, изменения были внесены только в описание стиля .papa, а визуально проявились в трех разных местах веб-страницы.

Здесь **background-color** - свойство стиля, задающего цвет фона веб-страницы. В приведенном примере его значение было задано специальным служебным словом **lightgreen** (светло-зеленый).

```
<html>
<body>

<style>
.papa {
    font-family: Helvetica;
    font-weight: bold;
    font-size: large;
    font-style: italic;
    display: inline;
    background-color: lightgreen;
}
</style>

<div class="papa">Браузер</div> -
программное обеспечение для просмотра
веб-сайтов.<br>

<div class="papa">Веб-сервер</div> -
Веб-сервер - это сервер, принимающий
HTTP-запросы от клиентов, обычно веб-
браузеров, и выдающий им HTTP-
ответы.<br>

<div class="papa">JavaScript</div> -
язык сценариев для придания
интерактивности веб-страницам.<br>
</body>
```

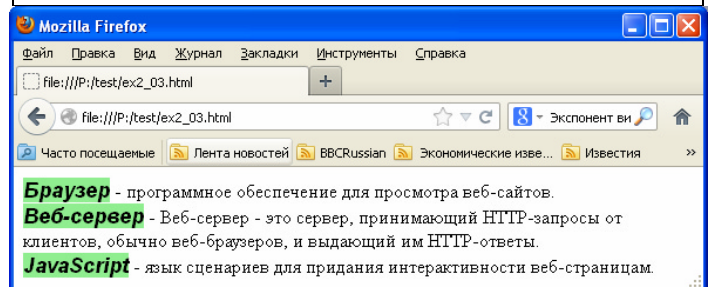


Рисунок 2.3 - Веб-страница с изменениями в стиле

Кроме служебных слов могут использоваться и цвета, заданные тремя двухзначными числами (задающими яркость красной, зеленой и синей составляющей от 0 до 255) в шестнадцатеричном формате. Например:

```
background-color: #cc6500;
```

означает коричневый цвет. Первые две цифры “cc” задают значение красной составляющей (#cc = 204), вторые две цифры “65” задают значение зеленой составляющей (#65 = 101) и третьи две цифры “00” задают значение синей составляющей (#00 = 0).

Количество служебных слов, означающих различный цвет, является ограниченным, а использование числовой формы для значения цвета позволяет использовать на странице любой оттенок цвета. Если по какой-либо причине использовать шестнадцатеричное представление неудобно, то можно пользоваться и десятичным представлением, например, так:

```
background-color: RGB(204,101,0);
```

Помните, что в веб-программировании, как и в любом другом программировании, поставленную задачу обычно можно решить несколькими различными путями с использованием различных средств языка программирования. Всегда выбирайте из них тот вариант, который удобнее лично для Вас.

Чтобы закончить со всеми непонятными вещами в этом примере, рассмотрим свойство **display** стиля. Дело в том, что по умолчанию каждый блок текста, ограниченный тегами `<div> .. </div>`, выводится с новой строки веб-страницы. Значение **inline** свойства **display** позволяет убрать этот перенос строк. Если бы мы не задали это свойство в предыдущем примере, то увидели бы веб-страницу такой, какой она показана на рис. 2.4 (файл ex3\_04.html).

```
<html>
<body>

<style>
.papa {
  font-family: Helvetica;
  font-weight:bold;
  font-size:large;
  font-style:italic;
  background-color:lightgreen;
}
</style>

<div class="papa">Браузер</div> -
программное обеспечение для просмотра
веб-сайтов.<br>

<div class="papa">Веб-сервер</div> -
Веб-сервер - это сервер, принимающий
HTTP-запросы от клиентов, обычно веб-
браузеров, и выдающий им HTTP-
ответы.<br>

<div class="papa">JavaScript</div> -
язык сценариев для придания
интерактивности веб-страницам.<br>
</body>
```

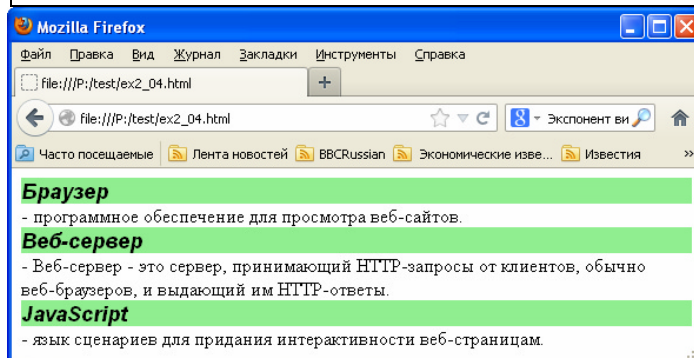


Рисунок 2.4 - Иллюстрация переноса строк при использовании тега `<div>` без установки специальных свойств стиля

Как видно, по умолчанию тег `<div> .. </div>` занимает целую строку веб-страницы без остатка. Чтобы изменить ситуацию, нужно задавать значения свойства **display** (что

было сделано) или же свойства **float** (изучите его по справочнику самостоятельно) или же задавать свойство **width** (ширина тега - изучите его самостоятельно) или вообще использовать тег `<span> .. </span>`.

Используемый в приведенных выше примерах способ описания стилей внутри тега `<style> .. </style>` хорош, если стиль затем часто используется на странице. Если же стиль используется всего раз, то его можно задать прямо внутри тега с помощью атрибута **style** (см. рисунок 2.5, файл ex2\_05.html).

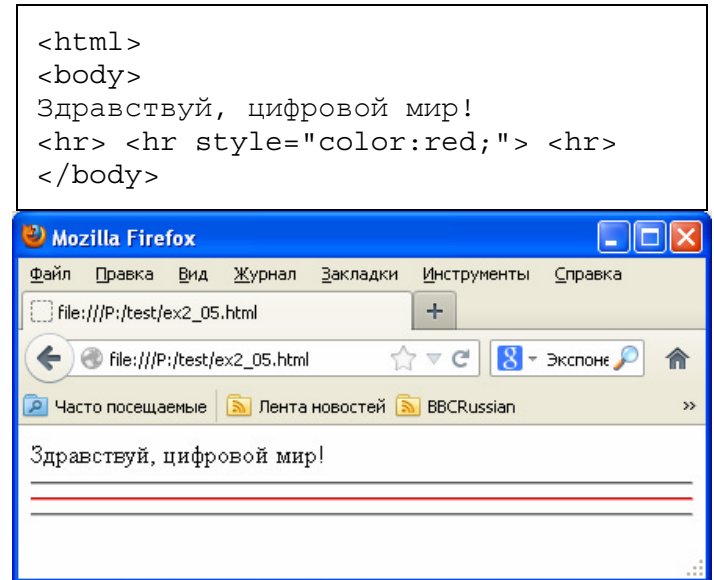


Рисунок 2.5 - Иллюстрация указания стиля внутри тега `<hr>`

В данном примере с помощью одиночных тегов `<hr>` в веб-страницу вставлены три горизонтальные линии. Для второй из них с помощью атрибута **style** задан красный цвет (свойство **color**). Так как красный цветом выделена только одна строка, нет необходимости выносить описание этого стиля в отдельное место веб-страницы.

Если бы в этом примере нам захотелось использовать тег `<style> .. </style>` для описания стиля, то нам пришлось бы придумать ему имя, например, "mama", затем описать его:

```
<style>
.mama {
  color:red;
}
</style>
```

Затем в теге `<hr>` пришлось бы указать атрибут **class**:

```
<hr class="mama">
```

Текст веб-страницы бы от этого только увеличился и стал менее понятным. Именно поэтому в данном случае целесообразнее (проще) указать значения свойств стиля прямо внутри тега.

Как видим, какой способ описания стилей на веб-странице использовать, определяется только целесообразностью. Следует использовать тот способ, который больше нравится и который удобнее использовать в данном конкретном случае.

Все это относится и к последнему, третьему способу описания стилей - в отдельном файле. Представим, что одними и теми же стилями мы пользуемся для нескольких веб-страниц. Тогда разумно описать их в отдельном файле, а

в каждой из веб-страниц просто указать имя этого файла.

При описании стилей в отдельном файле теги `<style> .. </style>` не нужны. Стили просто описываются один под другим. При этом в текст веб-страницы вставляется ссылка на этот файл (одинарный тег `<link>`), которая должна выглядеть примерно так:

```
<link rel=stylesheet type=text/css
href="styles.css">
```

Здесь “styles.css” имя файла, в котором описаны стили. На рисунке 2.6 приведен текст файла описания стилей (файл styles.css), текст веб-страницы (файл ex2\_06.html) и скриншот окна браузера с выведенной страницей.

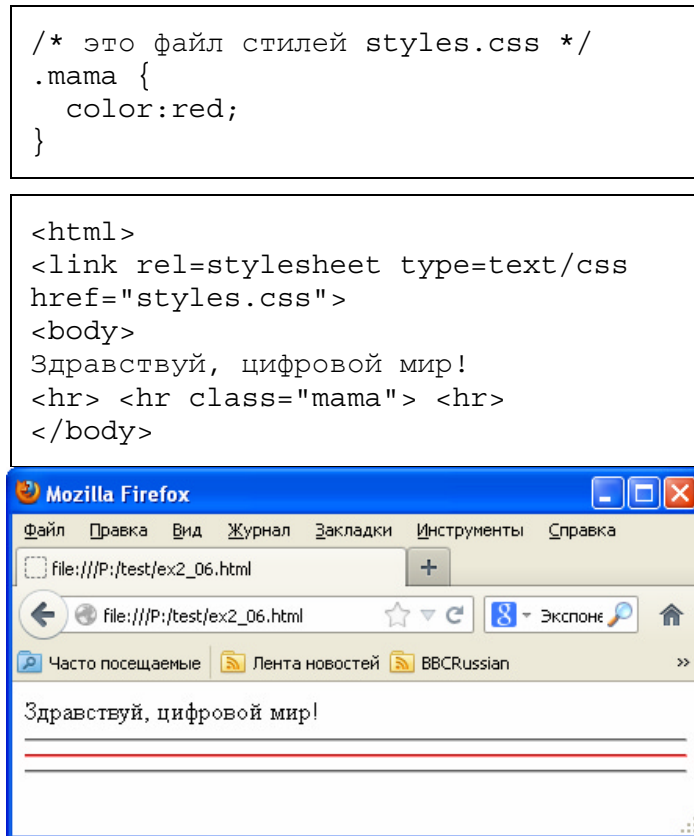


Рисунок 2.6 - Пример описания стилей в отдельном файле

Обратите внимание на то, что в данном примере в файле стилей вставлен комментарий (текст, который ни на что не влияет). Комментарии в стилях ограничиваются с обеих сторон косыми чертами со звездочками вот так:

```
/* Текст комментария */
```

## 2.2. Объявление нескольких стилей на веб-странице

В предыдущем параграфе мы с Вами разобрались, для чего нужны стили (кроме удобства использования, они предоставляют еще и больше возможностей для оформления веб-страницы, чем HTML разметка) и какими тремя способами их можно описать на веб-странице. Однако во всех примерах был описан только один стиль (для простоты). Каким же образом на одной странице описать несколько стилей?

Рассмотрим простой пример (рисунок 2.7, файл ex2\_07.html). В данном примере описаны три стиля с именами “.papa”, “.mama” и “.derevo”.

```
<html>
<body>
<style>
.papa, .mama {
    font-family:Helvetica;
    font-weight:bold;
    font-size:large;
    font-style:italic;
    background-color:blue;
    display:inline;
}

.mama {
    font-size:medium;
    color:RGB(255,255,0);
}

.derevo {
    font-family:Arial;
    font-size:20pt;
}
</style>
<div class="papa">Браузер</div> -
программное обеспечение для просмотра
веб-сайтов.<br>
<div class="mama">Веб-сервер</div> -
Веб-сервер - это сервер, принимающий
HTTP-запросы от клиентов, обычно веб-
браузеров, и выдающий им HTTP-
ответы.<br>
<div class="derevo">JavaScript</div>
JavaScript - язык сценариев для
придания интерактивности веб-
страницам.<br>
</body>
```

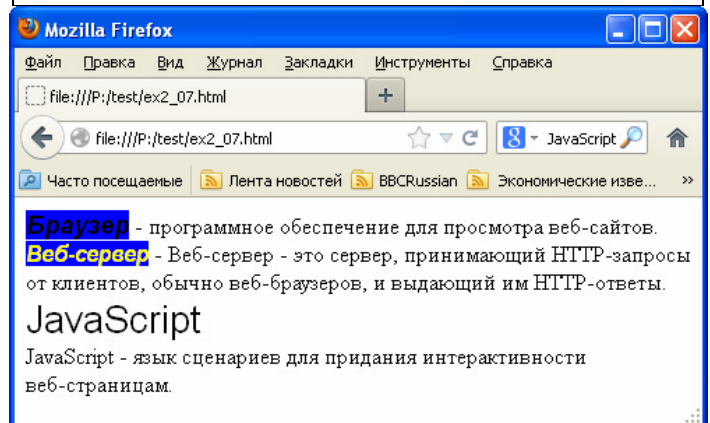


Рисунок 2.7 - Иллюстрация описания нескольких стилей

Стили описываются один под другим. В данном примере для удобства чтения текста веб-страницы между описаниями стилей вставлены пустые строки, но это не обязательно.

Обратите внимание, что в описании первого стиля через запятую указаны сразу два имени: “.papa, .mama”. Это значит, что перечисленные ниже значения свойств будут заданы для обоих стилей.

Обратите также внимание, что в описании второго стиля снова указано имя “.mama”. Это описание не отменяет предыдущее описание для `.mama`, а дополняет и частично

замещает его.

Так, в описании второго стиля для стиля .mama переопределяется значение свойства **font-size** (с large на medium) и добавляется новое свойство **color**, ранее не заданное.

Такая гибкость в описании CSS позволяет экономить текст страницы и время разработчика при описании стилей, у которых много свойств с похожими значениями.

У стиля .derevo в данном примере нет общих свойств с стилями .para и .mama, поэтому он описан отдельно от них.

## 2.2. Способы указания имени стиля

В приведенных выше примерах мы работали со стилями, идентифицируемыми произвольно заданным именем стиля. Чтобы оформить какой-то тег в данном стиле, нужно в атрибуте **class** этого тега указать имя этого стиля (для имени стиля существует специальный термин: **селектор**).

Что делать, если мы хотим, например, все теги <hr> в веб-странице оформить в определенном стиле? Решение приведено на рисунке 4.8 (файл ex2\_08.html).

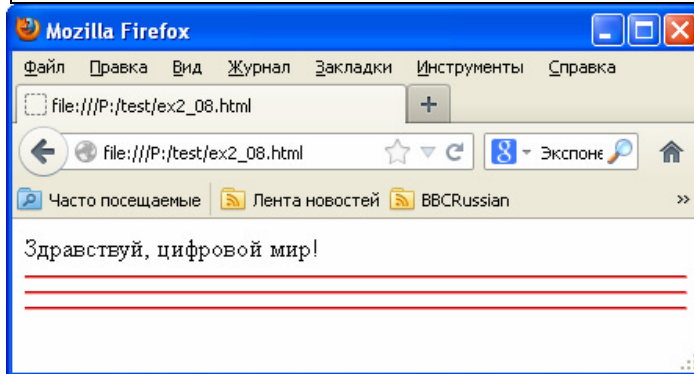
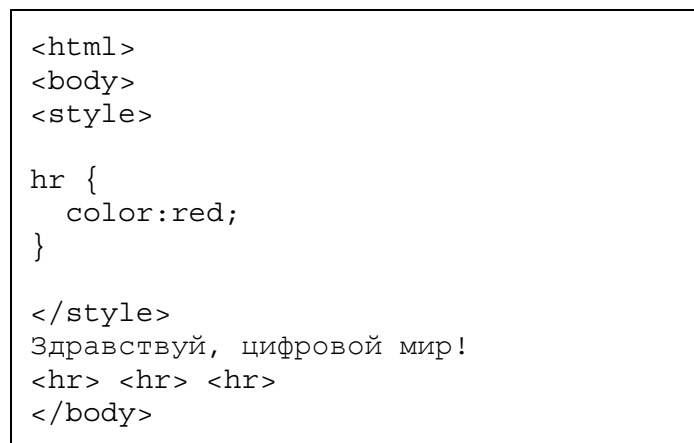


Рисунок 2.8 - Иллюстрация задания стиля для всех тегов <hr> на веб-странице

Как видно, в данном случае вместо имени стиля указывается ключевое слово тега “**hr**”, причем точка перед ним не ставится.

Все теги <hr> на странице теперь выводятся красным цветом. Обратите внимание, что внутри этих тегов в данном случае не нужно указывать атрибут **class**.

Аналогично можно задавать стили и для других тегов, например, <h1>, <table> и т.д.

На рисунке 2.9 приведен еще один пример (файл ex2\_09.html), в котором стиль оказывается привязанным к тегу <hr> с конкретным значением атрибута **id** (со значением “moi”).

В данном случае в качестве имени стиля задается

“hr#moi”, состоящее из названия тега, знака “#” и значения атрибута **id**. Таким же образом (через знак “#”) можно указывать и любые другие теги и их **id**.

Как видно из рисунка 2.9, в этом случае действие стиля оформления распространяется только на один тег из множества тегов <hr> (только на тот, у которого **id**=“moi”).

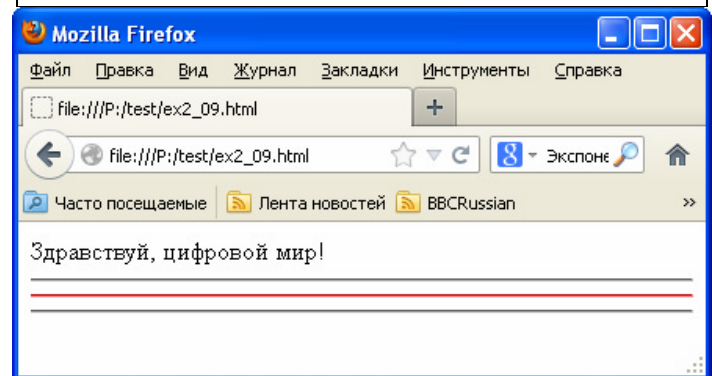
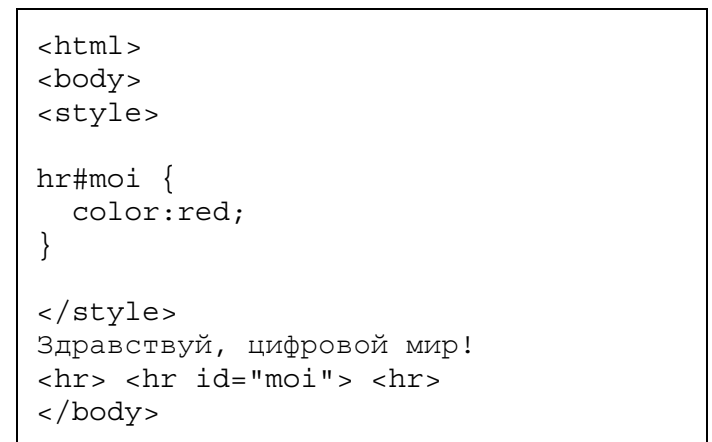


Рисунок 2.9 - Иллюстрация задания стиля для тега с определенным названием и именем

Возможна ситуация, когда стиль не соответствует ни одному из тегов веб-страницы и, таким образом, никак не влияет на ее внешний вид.

И, наконец, в следующем примере (файл ex2\_10.html, рисунок 2.10) покажем, как сделать, чтобы стиль распространялся на тег только при условии, что он является вложенным в другой тег.

В этом примере для всех тегов <hr> (горизонтальная черта) задан красный цвет. Однако, если горизонтальная черта выводится внутри какого-либо тега со значением атрибута **class**=“mama”, то она выводится синим цветом.

Чтобы добиться такого эффекта, вместо названия стиля ставится название стиля тега родителя (внутри которого будет выводиться заданный стиль), затем пробел и имя описываемого стиля:

```

имя_родителя имя_стиля {
}

```

Внимательно набирайте текст веб-страницы и читайте текст чужой веб-страницы. От того, будет ли между именами стилей запятая или просто пробел, смысл написанного существенно меняется. Запятая, как было сказано раньше, означает, что одинаковые свойства будут заданы обоим стилям. Пробел же означает, что стиль является зависимым от контекста и действует на тег только при условии его вложенности в другой тег (если он является “**потомком**” другого тега).



```

<html>
<body>
<style>

hr {
  color:red;
}

.mama {
width:70%;
background-color:RGB(180,90,0);
color:white;
}

.mama hr {
  color:blue;
}

</style>
Здравствуй, цифровой мир!
<hr>
<div class="mama">
У этого текста белый цвет<hr><br>
</div>
<hr>
</body>

```

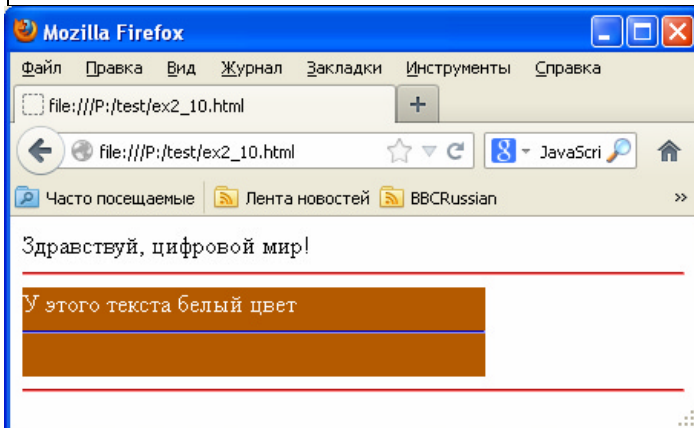


Рисунок 2.10 - Иллюстрация стиля тега, зависящего от контекста (тега, в который он вложен)

Обратите также внимание, что в данном примере в стиле “mama” задана ширина блока в процентах от ширины экрана браузера (самостоятельно изучите другие способы задания ширины блока). Горизонтальная линия, находящаяся внутри тега `<div> .. </div>`, занимает всю ширину блока (обычно занимает всю ширину экрана).

Если в странице присутствует многократная вложенность блоков один в один, иногда бывает необходимо, чтобы стиль действовал не на всех потомков определенного тега, а только на его “детей” (потомков первого уровня вложенности). В этом случае вместо записи “.mama hr” используют запись “.mama > hr”. Имена стилей в данном случае разделяются знаком “>”.

Еще бывает нужно, чтобы условием были связаны не родитель и потомок, а два соседних блока веб-страницы (идущие в тексте страницы один за другим). В этом случае используют запись вида “.mama + hr”. Имена стилей в данном случае разделяются знаком “+”.

Самостоятельно подготовьте и проанализируйте примеры веб-страниц с вышеперечисленными условиями для стилей.

### 2.3. Почему стили называются каскадными?

Стили называются каскадными, потому что их действие распространяется и на вложенные блоки, если это не переопределено стилями этих блоков. В таком случае говорят, что свойства стиля **родительского** блока были **наследованы** стилем вложенного блока.

```

<html>
<body>
<style>
.papa {
  font-family:Verdana;
  font-weight:bold;
  font-size:large;
  width:500px;
  height:350px;
  border-style:solid;
  border-width: 1px;
  padding: 15px;
}
.mama {
width:400px;
height:200px;
background-color:RGB(180,90,0);
padding: 5px;
}
</style>
<div class="papa">
Здравствуй, цифровой мир!<br><br>
<div class="mama">
Здесь вложенный текст
</div></div>
</body>

```

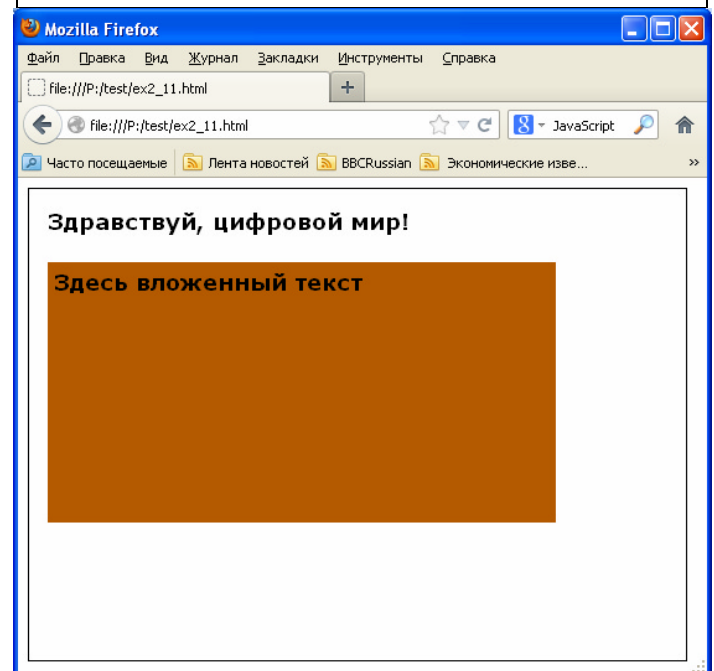


Рисунок 2.11 - Иллюстрация наследования свойств стилей

Эту важную особенность CSS иллюстрирует пример на рисунке 2.11 (файл ex2\_11.html).

Тег с классом “mama” в данном примере является вложенным в тег с классом “papa” и наследует свойства стилей родителя, в частности, размер и тип шрифта (которые явно в стиле “.mama” не прописаны).

Наследование в CSS является выборочным - наследуются далеко не все свойства. Например, разработчики CSS посчитали нецелесообразным наследовать свойства рамки (border). Именно поэтому рамочка, которая задана вокруг блока с классом “papa” отсутствует вокруг блока с классом “mama”. Самостоятельно изучите по справочникам, какие свойства стилей наследуются, а какие - нет.

Для многих свойств стилей предусмотрено значение “inherit”, которое прямо указывает стилю, что он должен наследовать это свойство у родителя (по умолчанию он может не наследовать это свойство).

#### 2.4. Примеры использования некоторых свойств стилей

Рассмотрим несколько важных свойств стилей. В первых, это свойство **position**. Это свойство позволяет, например, добиться перекрытия нескольких блоков на веб-странице. Пример такого перекрытия приведен на рисунке 2.12 (файл ex2\_12.html).

Значение **absolute** для свойства **position** приводит к тому, что блок “забывает” о своем стандартном местоположении на веб-странице и перемещается в то место, которое задают свойства **left**, **right**, **top** или **bottom** (смещения от края).

При этом позиция блока на экране вычисляется относительно краев веб-страницы. Если же у блока есть родители (он является вложенным в другой блок) и у блока-родителя свойство **position** также установлено в **absolute** или **fixed** или **relative** (рассмотрите значения **fixed** и **relative** самостоятельно)), то позиция вычисляется относительно краев родительского блока.

В данном примере и у зеленого и у синего блоков заданы абсолютные позиции, а у белого блока, вложенного в синий, позиция задана относительно границ синего блока.

Обратите также внимание на свойство **z-index**. Это полезное свойство позволяет указать браузеру, какой из двух пересекающихся блоков будет выведен поверх другого (тот, у которого установлено более высокое значение **z-index**).

Еще одной важной особенностью CSS является вычисление размеров блоков. Казалось бы, у синего блока задан размер в ширину 400 пикселей (width: 400). У маленького белого блока задан размер в ширину 90 пикселей и отступ от левого края в 370 пикселей. Несложная арифметика говорит нам, что  $370 + 90 = 460$ . Таким образом, белый блок должен выступать за правый край синего родительского блока, однако, он не выступает! В чем же дело?

Дело в свойстве **padding** (задает отступ от краев блока до текстовой области внутри блока) и в том, как CSS интерпретирует свойство **width** (это ширина текстовой области внутри блока, не считая отступы). Реальная ширина блока складывается из суммы **width** и двух **padding** (отступ слева и отступ справа). В данном случае  $40 + 400 + 40 = 480$  пикселей. Теперь становится понятно, почему при отступе от левого края в 370 пикселей и длине в 100 пикселей (width 90 и два отступа по 5) белый блок не выступает за край синего.

```
<html>
<body>
<style>
.niz, .verh, .small {
  width:400px;
  height:200px;
  border-style:solid;
  border-width: 1px;
  padding: 40px;
  position: absolute;
  background-color:#99ff77;
  font-size: large;
  z-index: 1;
  left:20;
  top:20;
}
.verh {
  background-color:#5555ff;
  z-index: 2;
  left:100; top:130;
}
.small {
  background-color:white;
  left:370px;
  width:90px;height:22px;
  padding:5px;
}
</style>
<div class="verh">Этот блок
расположен выше
<div class="small">13-02-
2013</div></div>
<div class="niz">Этот блок расположен
ниже</div>
</body>
```

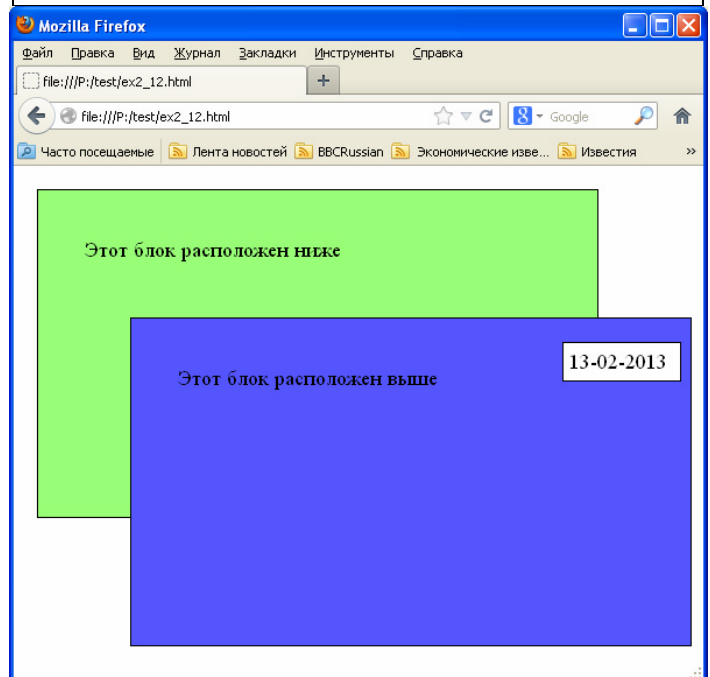


Рисунок 2.12 - Иллюстрация свойства “position”

Будьте внимательны с использованием свойств **width**, **height**

и **padding**, проверяйте, как выглядит страница на разных браузерах. Дело в том, что браузер Internet Explorer вопреки спецификации CSS считает, что ширина блока - это не сумма **width** и отступов, а просто **width**. Отступы он обеспечивает за счет уменьшения внутренней области вывода текста. Приведенная на рисунке 2.12 веб-страница при просмотре в Internet Explorer будет выглядеть так, как это показано на рисунке 2.13.

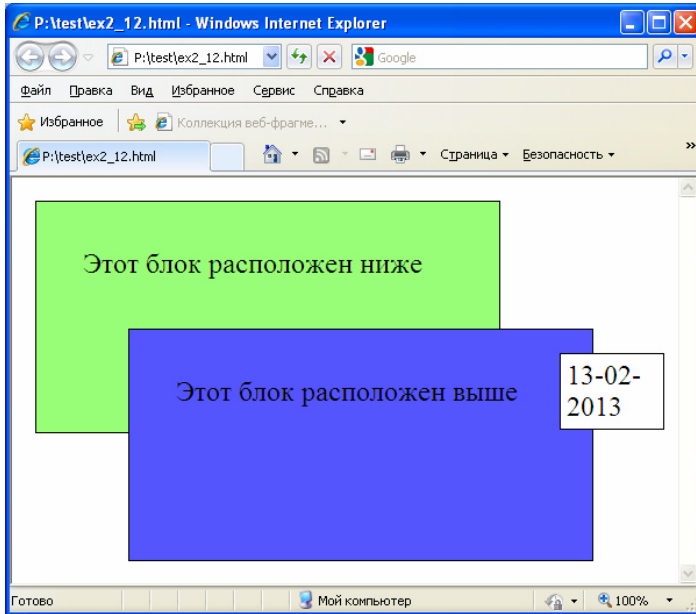


Рисунок 2.13 - Эта же веб-страница в Internet Explorer

Как видим, белый блок выехал за пределы синего, а надпись на нем из-за уменьшившейся ширины области текста частично переехала на вторую строку.

Чтобы обойти данную проблему, можно попытаться обойтись в нашей странице без свойства **padding**, или же задать отступ для белого блока не от левой границы, а от правой, или найти еще какое-то другое решение, которое покажется лично Вам наиболее приемлемым из множества возможных вариантов. Можно и вообще проигнорировать браузер Internet Explorer.

Рассмотрим еще одно важное свойство стилей - **background** и связанное с ним свойство **background-position**. На рисунке 2.14 приведено изображение ui.png, содержащее альфа-канал (прозрачность, обозначенная на рисунке шахматными клеточками).

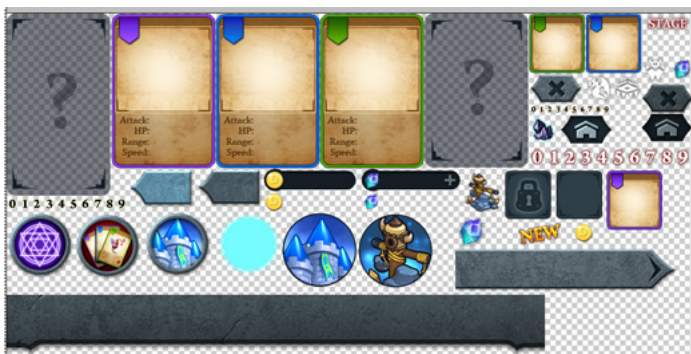


Рисунок 2.14 - Изображение ui.png

На рисунке 2.15 приведен пример веб-страницы, созданной с использованием изображения ui.png в качестве фонового (файл ex2\_13.html), а также свойства позиционирования (сдвига) фона - **background-position**.

```
<html>
<body style="background-
color:green;">

<style>
.qq, .bb, .uu {
    position:absolute;
    background:url('ui.png');
}
.qq {
    top:20px;left:20px;
    width:156px;
    height:230px;
    background-position:-154px -2px;
}
.bb {
    top:100px;left:100px;
    width:110px;
    height:110px;
    background-position:-410px -298px;
}
.uu {
    top:80px;left:40px;
    width:62px;
    height:65px;
    background-position:-679px -236px;
}
</style>

<div class="qq"></div>
<div class="bb"></div>
<div class="uu"></div>

</body>
```



Рисунок 2.15 - Иллюстрация свойства background



Как видно, три разных блока за счет абсолютного позиционирования были наложены друг поверх друга. Использование же в качестве фона изображения с прозрачностью позволило создать сложное составное изображение непрямоугольной формы.

Здесь каждый блок является своеобразным окном, сквозь которое мы смотрим на фоновое изображение (имя изображения задается в свойстве **background**):

```
background:url('ui.png');
```

При этом свойство **background-position** позволяет сдвинуть фоновое изображение на требуемое число пикселей по вертикали и горизонтали так, что в наше “окно” мы начинаем видеть другой участок фонового изображения.

Поэкспериментируйте с этой страницей. Попробуйте изменять размеры окон, задавать другие величины для сдвига фонового изображения.

Обратите внимание, что общий зеленый фон страницы задан прямо в стиле тега **<body>**. Все стили оформления для этого тега распространяются на все элементы веб-страницы.

Также обратите внимание, что свойства **position** и **background** мы задали одинаковыми сразу для трех стилей, перечислив имена этих стилей через запятую. Ниже задаются остальные свойства для этих стилей (при желании некоторым свойствам).

## 2.5. Визуальный анализ стилей веб-страниц

Бывает тяжело анализировать стили веб-страниц (особенно чужих) в текстовом виде. Поэтому полезными являются утилиты и **плагины** к браузерам, позволяющие анализировать стили визуально. Одним из таких плагинов является “Инспектор” в браузере Mozilla Firefox. Для запуска этого браузера нужно войти в подменю “Инструменты”, затем - в подменю “Веб-разработка” и там выбрать пункт “Инспектор”. Запускается анализ текущей веб-страницы и при нажатии ее элементов можно увидеть окно, как на рисунке 2.16.

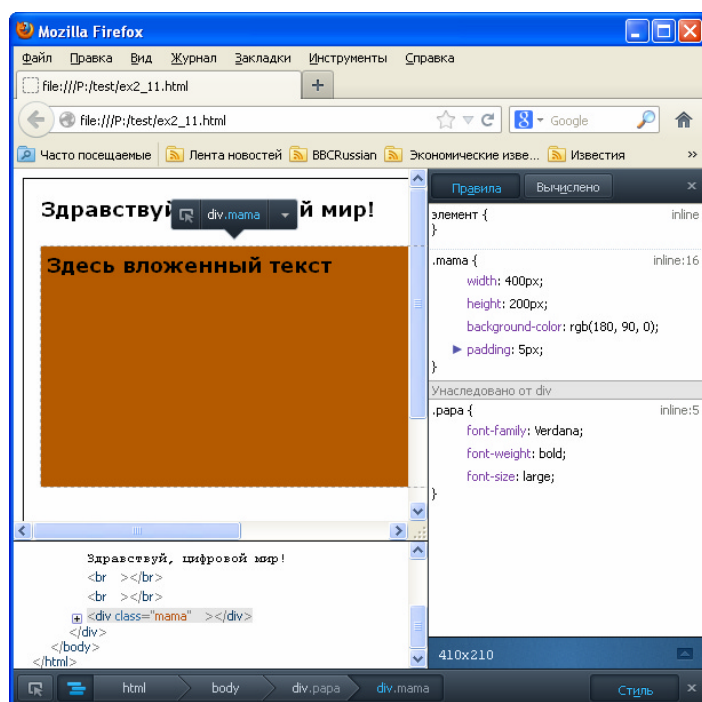


Рисунок 2.16 - Пример использования режима “Инспектор”

В данном примере анализировалась страница, приведенная на рисунке 2.11. Видно, что при нажатии на блок со стилем “.mama” инспектор показывает значения свойств этого стиля, а также, какие свойства стилей были наследованы от родительского блока со стилем “.para”.

Более того “Инспектор” позволяет на лету изменять свойства стилей и смотреть, что из этого получится. Для этого нужно кликнуть мышкой на значение свойства и ввести новое значение. На рис. 2.17 показан этот же пример после изменения в “Инспекторе” свойства **background-color**.

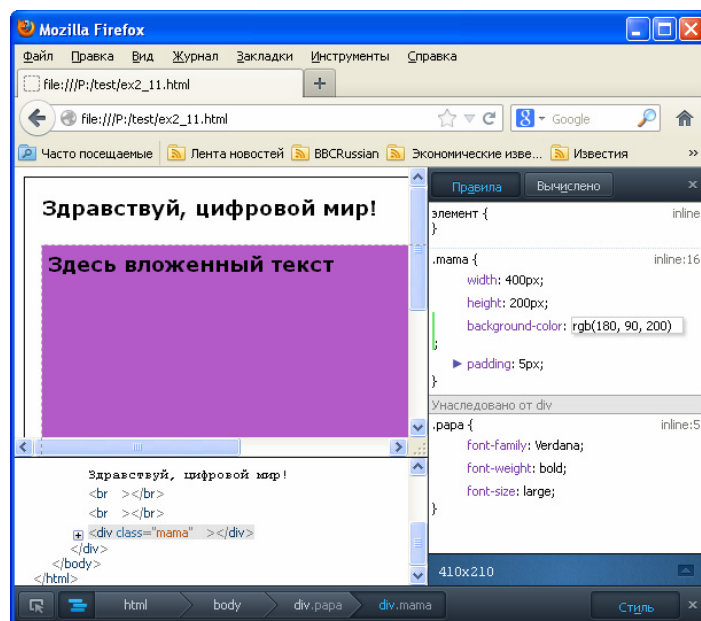


Рисунок 2.17 - Изменение в “Инспекторе” свойств стилей

На этом данная лекция заканчивается. Пожалуйста, самостоятельно рассмотрите следующие свойства стилей: **border**, **float**, **clear**, **cursor**, **margin**, **max-width**, **opacity**, **outline**, **text-align**, **transform**, **vertical-align**, **visibility**. Поэкспериментируйте с разными их значениями, с разными вариантами наследования свойств стилей (при вложении одних блоков текста в другие). Помните, что в веб-программировании многое легче узнать на конкретном примере, чем изучить в теории. Создавайте веб-страницы, смотрите, как они выглядят в браузере. Если что-то в поведении веб-страницы или браузера непонятно, то кроме справочников информацию можно найти по поиску в Google или спросить у знакомого, который разобрался в этом вопросе раньше вас.



### 3. Основы языка программирования JavaScript

**JavaScript** - язык программирования, который применяется в основном для разработки клиентской части сайтов (**front-end**).

JavaScript является интерпретатором и выполняется внутри браузера (программы, которая визуализирует веб-страницу). Браузер вначале анализирует текст программы, интерпретирует его, и только затем выполняет нужные действия. Это медленно. Поэтому программы на JavaScript работают медленно и не всегда одинаково для разных браузеров.

Почему тогда интерпретатор, а не компилятор? Язык JavaScript был специально разработан как интерпретатор, чтобы была возможность исключить из него все операции работы с файлами (чтение, запись и т.п.). Это необходимо для того, чтобы веб-страница, загруженная из сети Интернет и содержащая программу JavaScript, не могла повредить компьютеру пользователя (например, заразить его вирусом). Если заражение вирусом и происходит, то из-за каких-то других причин, но не по вине JavaScript.

Итак, писать и читать файлы программа на JavaScript не может. А что же она может? Для чего в основном используется JavaScript?

JavaScript предназначен для:

- 1) Автоматизации процесса набора текста страниц;
- 2) Обработки событий, происходящих на странице (придания странице интерактивности);
- 3) Изменения содержимого веб-страниц во время их просмотра, анализ данных веб-страниц;
- 4) Формирование запросов к серверу и получение ответов без перегрузки веб-страницы;
- 5) Работа с cookies (данные веб-страницы, хранящиеся в браузере на компьютере пользователя).

Ничего из этого нельзя добиться с помощью HTML и CSS, поэтому JavaScript дополняет эти инструменты веб-программирования, а не заменяет или дублирует их.

Перед тем, как приступить к описанию JavaScript, отметим, что здесь будут рассмотрены лишь необходимые основы, достаточные для того, чтобы начать программировать на JavaScript. Все остальное вы сможете изучить самостоятельно по мере практической надобности.

Мы здесь не будем останавливаться на перечислении конструкций и типов языка программирования JavaScript, а также подробном описании его синтаксиса. Будем считать, что вы уже знаете какой-либо из языков программирования, например, C/C++. В этом случае вы легко поймете синтаксис JavaScript, а основное знание, которое вам потребуется приобрести, это как с помощью JavaScript решать его основные задачи, перечисленные выше. Этому и конкретным примерам мы и посвятим основной объем данной лекции.

#### 3.1. Автоматическое генерирование текста страниц

Чтобы начать программировать на JavaScript не нужно ничего, кроме одного (любого) из браузеров Интернета, установленного на нашем компьютере. Текст программы на JavaScript вставляется прямо в HTML-страницу внутри тега `<script> .. </script>`. Таких тегов может быть несколько и браузер выполняет помещенные внутри них куски JavaScript-программ сразу же, как только доходит до этого места в процессе загрузки веб-страницы.

Рассмотрим текст простой веб-страницы, который показан на рисунке 3.1 (файл ex3\_01.html).

```
<html>
<body>
Первая программа на JavaScript.<BR>

<script type="text/javascript">

    document.writeln('Hello,    digital
world!');

</script>

</body>
```

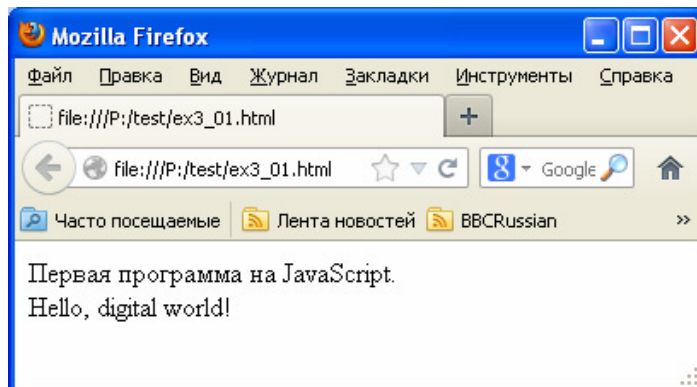


Рисунок 3.1 - Пример простой страницы с JavaScript

Атрибут **type** тега `<script>` в данном случае сообщает браузеру, что он имеет дело с JavaScript (теоретически здесь может быть скрипт на другом языке программирования). Однако для всех браузеров языком программирования по умолчанию является именно JavaScript. Поэтому вместо строки

```
<script type="text/javascript">
```

можно было бы написать вот так:

```
<script>
```

Функция

```
document.writeln('Hello, digital world')
```

выводит надпись 'Hello, digital world' на **document** (этим ключевым словом на JavaScript обозначается вся веб-страница, в которую вставлен скрипт).

Здесь **document** - это объект JavaScript, а **writeln** через точку от **document** - функция. Точка между ними указывает, что функция принадлежит этому объекту и, в данном конкретном случае выводит текст на этот объект.

Надпись в данном примере выводится в том месте страницы, где был вставлен скрипт.

Вышеприведенный пример просто иллюстрирует способ вставки программы на JavaScript на веб-страницу. Вообще же вставить эту надпись на веб-страницу можно было бы и средствами HTML. Рассмотрим более сложный пример, в котором при попытке обойтись средствами HTML нас бы ждал серьезный объем работы, в то время как программа на JavaScript делает все автоматически.

Пусть требуется вывести на веб-страницу 200 чисел, отмечая простые числа (которые ни на что не делятся) красным цветом. Программа на JavaScript будет содержать

функцию, возвращающую 1, если заданное число - простое, а также условные операторы и операторы цикла (подобные соответствующим конструкциям других языков программирования).

На рисунке 3.2 приведена такая программа (файл ex3\_02.html).

```
<html>
<body>
Простые числа выделены красным
цветом.<BR>

<style>
.myred { color:red; }
</style>

<script>

function prost(n)
{
    for (i=2;i<n;i++)
        if ((n % i)==0) return 0;
    return 1;
}

for (papa=1;papa<=200;papa++)
    if (prost(papa)==0)
        document.writeln(papa+' ');
    else
        document.writeln('<span
class="myred"> '+papa+' </span>');
</script>

</body>
```

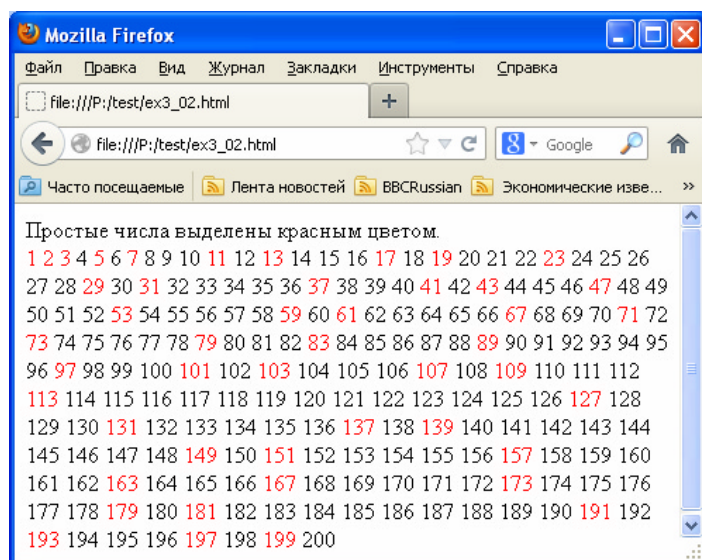


Рисунок 3.2 - Программа на JavaScript, которая отмечает простые числа красным цветом

Функции на JavaScript похожи на функции C/C++ за тем исключением, что не нужно объявлять тип входных переменных и тип результата функции. Браузер, который выполняет скрипт, по присваиваемым значениям сам поймет, что имеется ввиду.

Данная функция проверяет число n на простоту. Для этого в цикле **for** она находит остатки от деления n на все числа от 2 до n-1 (оператор %). Если хоть один из этих остатков равен нулю, то функция возвращает (оператор **return**) 1. Иначе (после выхода из цикла) функция возвращает 0 (признак того, что число - простое).

Основная программа с помощью цикла **for** (переменная "papa" названа так, чтобы показать, что имена переменных мы можем выбирать такие, какие нам нравятся) и функции prost(papa).

Не простые числа выводятся на веб-страницу без всякого оформления (оператор **document.writeln**).

Простые числа выводятся на веб-страницу, обрамленные тегом **<span>..</span>**, у которого выставлен атрибут **class="myred"**. Стиль ".myred" в свою очередь был описан выше в теге **<style>..</style>**. У него прописано единственное свойство - "color:red;" (красный цвет). Поэтому все простые числа выводятся на веб-страницу красным цветом.

Обратим внимание также на оператор "+", который используется в программе для объединения строковых переменных. Если этим оператором складываются два числа, то результатом будет их сумма. Если же складывают две строки (или число и строку), то результатом будет строка. Например, в результате выполнения кода **x=57;s='Nikolay'+x;** в переменной s будет содержаться строка 'Nikolay57'.

Приведенный на рисунке 3.2 пример иллюстрирует ситуацию, когда вручную набирать текст страницы гораздо дольше, чем написать программу. Чаще же бывает, что страница содержит какие-то переменные данные и должна каждый раз выглядеть по другому. JavaScript в данной ситуации незаменим.

### 3.2. Обработка событий

То, ради чего JavaScript и был создан - это обработка событий на веб-страницах, например, нажатий мышкой на какие-то элементы страницы. На рисунке 3.3 приведен пример простой страницы с кнопкой (файл ex3\_03.htm).

```
<html>
<body>
Это пример кнопки: <br><br>

<button>Кнопка</button>

</body>
```

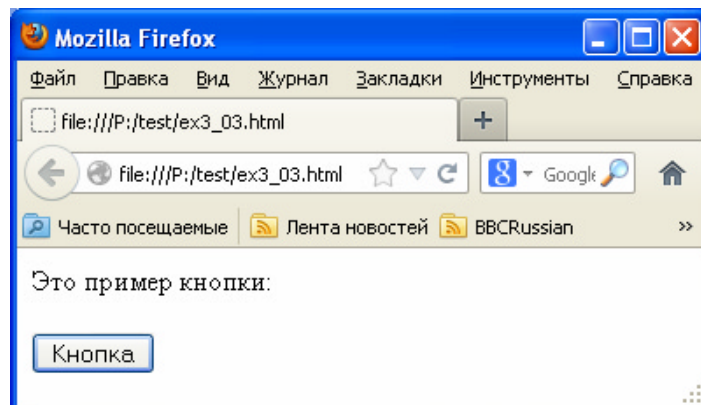


Рисунок 3.3 - Веб-страница с кнопкой

Здесь мы видим новый для нас тег `<button> .. </button>`, который добавляет на веб-страницу кнопку. Внутри тега может быть вставлена какая-то надпись, а может, например, быть вставлено и изображение (тег `<img>`).

Эта кнопка ничего не делает. Можно на нее нажать (она нажмется), но при этом ничего не произойдет. Чтобы что-то произошло, нужно использовать JavaScript.

У тега `<button>`, как и у большинства других тегов, есть специальные атрибуты, назначение которых - обработка событий. За нажатие мышкой отвечает атрибут `onClick`. В качестве значения этого атрибута должен быть вставлен какой-то текст на JavaScript примерно вот так:

```
<button onClick="Текст JavaScript">
```

В этом случае при нажатии на данную кнопку будет выполнен вставленный текст программы.

На рисунке 3.4 приведен текст этой же веб-страницы с обработкой нажатий на кнопку (файл `ex3_04.html`).

```
<html>
<body>
Это пример кнопки: <br><br>

<button onClick="alert('Кнопка
нажата') ">Кнопка</button>

</body>
```

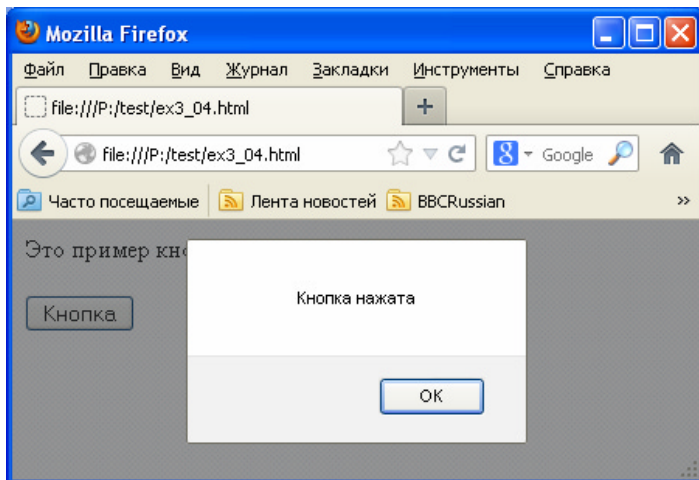


Рисунок 3.4 - Веб-страница с кнопкой и обработкой события `onClick`

Здесь в качестве текста программы указано:

```
alert('Кнопка нажата')
```

Функция **alert** является стандартной функцией JavaScript и выдает на экран окно с надписью (см. рисунок 3.4).

Давайте рассмотрим чуть более сложный пример, чтобы показать, что программа, реагирующая на событие, может состоять из многих команд и взаимодействовать с другими частями JavaScript на этой веб-странице.

Веб-страница на рисунке 3.5 подсчитывает, сколько раз мы нажали на эту кнопку (файл `ex3_05.html`) и выдает результат подсчетов внутри всплывающего окна с помощью функции **alert**.

```
<html>
<body>

<script>
var n=0;
</script>

Это пример кнопки: <br><br>

<button onClick="n=n+1;alert('Число
нажатий кнопки:
'+n) ">Кнопка</button>

</body>
```

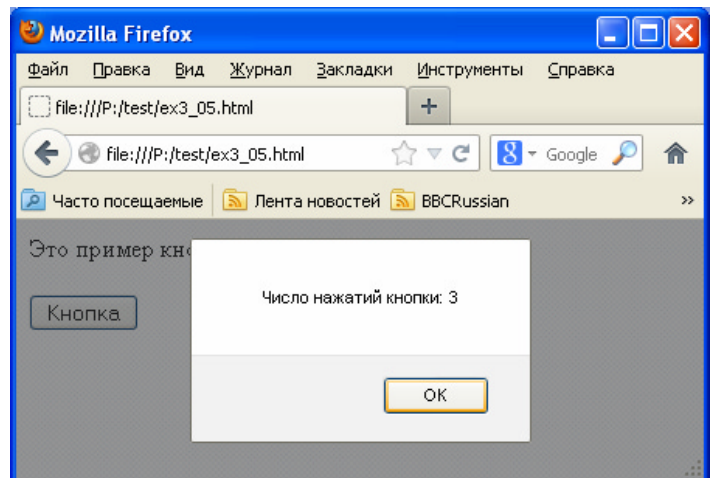


Рисунок 3.5 - еще одна веб-страница с кнопкой и обработкой события `onClick`

Здесь по нажатию кнопки выполняется следующий текст программы:

```
n=n+1;
alert('Число нажатий кнопки: '+n)
```

В переменной `n` мы храним число нажатий кнопки. При загрузке страницы в теге `<script> .. </script>` мы объявили эту переменную (ключевое слово **var**) и присвоили ей ноль. Далее при каждом нажатии ее значение увеличивается на единицу.

Обратим внимание, что писать при объявлении переменных ключевое слово **var** рекомендуется, но не обязательно (браузеры понимают и так). В частности, вместо текста

```
var n=0;
```

можно было написать просто

```
n=0;
```

Кстати, ставить точку с запятой после каждой команды на JavaScript тоже не обязательно. Но лучше это делать.

Если текст программы обработки события нажатия на кнопку будет длинным, то он порядком затруднит понимание текста веб-страницы, если помещать его внутрь атрибута `onClick`. Лучшим решением будет выносить его в отдельную функцию. Текст веб-страницы будет чуть длиннее, но при



этом будет выглядеть аккуратнее и понятнее (см. рисунок 3.6, файл ex3\_06.html).

```
<html>
<body>

<script>
var n=0;
function misha()
{
  n=n+1;
  alert('Число нажатий кнопки: '+n);
}
</script>

Это пример кнопки: <br><br>

<button onClick="misha()">Кнопка
</button>

</body>
```

Рисунок 3.6 - Обработка нажатия на кнопку вынесена в отдельную функцию misha

Как видим, внутри атрибута **onClick** остался только вызов функции “misha”, а текст самой функции находится в другом месте веб-страницы внутри тега **<script> .. </script>**.

Рассмотрим теперь пример обработки событий не для кнопки, а какого-нибудь другого тега HTML, например, **<div>** (см. рисунок 3.7, файл ex3\_07.html).

```
<html>
<body>

<div style="width:200px;
height:100px;
background-color:lightgreen;"
onClick="alert('Нажато')">Текст</div>

</body>
```

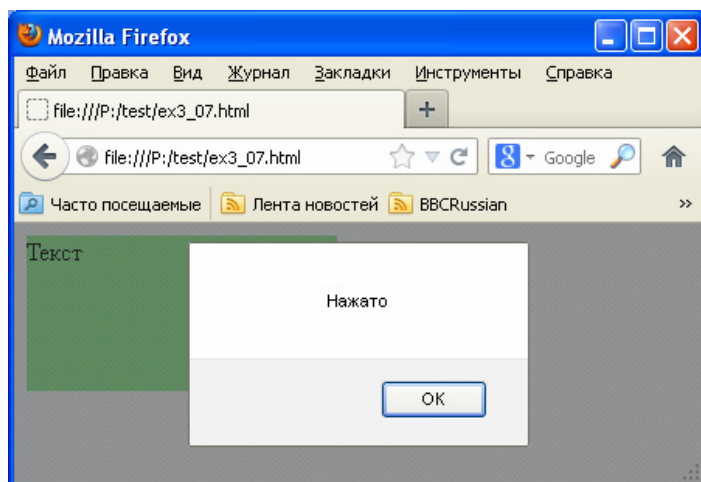


Рисунок 3.7 - Обработка нажатия на прямоугольник, заданный тегом **<div> .. </div>**

Как видим, сделано все точно так же, как и для тега **<button>**. При нажатии на элемент **<div>** (зеленый прямоугольник) выскакивает окошко с надписью. Точно так же атрибут **onClick** можно устанавливать для большинства видимых тегов HTML.

А какие еще есть атрибуты тегов, отвечающие за обработку событий? Перечислим наиболее часто используемые из них:

<b>onblur:</b>	потеря фокуса;
<b>onchange:</b>	изменение значения элемента формы;
<b>onclick:</b>	щелчок левой кнопкой мыши на элементе;
<b>ondblclick:</b>	двойной щелчок левой кнопкой мыши на элементе;
<b>onfocus:</b>	получение фокуса;
<b>onkeydown:</b>	клавиша нажата, но не отпущена;
<b>onkeypress:</b>	клавиша нажата и отпущена;
<b>onkeyup:</b>	клавиша отпущена;
<b>onload:</b>	документ загружен;
<b>onmousedown:</b>	нажата левая кнопка мыши;
<b>onmousemove:</b>	перемещение курсора мыши;
<b>onmouseout:</b>	курсор покидает элемент;
<b>onmouseover:</b>	курсор наводится на элемент;
<b>onmouseup:</b>	левая кнопка мыши отпущена;
<b>onreset:</b>	форма очищена;
<b>onselect:</b>	выделен текст в поле формы;
<b>onsubmit:</b>	форма отправлена;
<b>onunload:</b>	заккрытие окна.

Разберитесь с этими атрибутами самостоятельно (практическим путем и с помощью справочников), а здесь мы рассмотрим еще один пример, который позволит нам лучше понимать поведение программы на JavaScript (рисунок 3.8, файл ex3\_08.html).

```
<html>
<body>
<button
onClick="more()">Кнопка</button>
<script> more(); </script>
<br>Здесь какой-то текст<br>
<script>
  function more()
  {
    alert('Мопе!');
  }
</script>
</body>
```

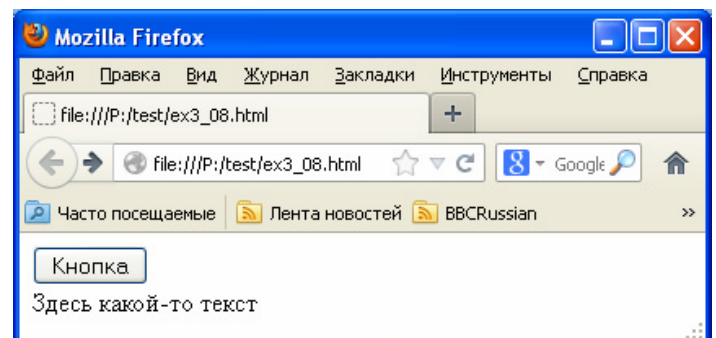


Рисунок 3.8 - Иллюстрация к особенностям размещения текста JavaScript на веб-странице



Попробуйте открыть в браузере эту страницу. Она нарисует кнопку, текст, но ничего не выведет на экран. А ведь в тексте этой веб-страницы вызывается функция `more()`:

```
<script> more(); </script>
```

При вызове этой функции по идее должно всплыть окошко с надписью “Море”. Но не всплывает.

А еще на страничке есть кнопка, при нажатии на которую тоже вызывается функция `more()`. Попробуйте нажать на нее. А сейчас окно с надписью море появилось!

Почему? Почему в одном случае функция не сработала, а в другом - сработала?

Потому что браузер пытается выполнить JavaScript сразу же, как только его встречает (прямо в процессе загрузки страницы). И в тот момент, когда браузер частично загрузит текст страницы и увидит команду:

```
<script> more(); </script>
```

остальная часть страницы еще не будет загружена. И в том числе еще не будет загружена та часть страницы, где объявлена функция `more()`. Поэтому браузер проигнорирует вызов этой функции (он такой функции пока не знает).

В тот же момент, когда мы нажимаем на кнопку, страница уже полностью загружена, браузер уже загрузил то место веб-страницы, где объявлена `more()` и поэтому может вызвать эту функцию.

Учитывайте этот нюанс при разработке своих страниц. Пока веб-страница не загрузилась полностью, нельзя из JavaScript обращаться к тем кускам программы, которые вставлены в страницу дальше по тексту. Иногда для проверки загруженности страницы бывает удобно использовать событие **onload** (документ загружен). Его можно привязать к тегу `<body> .. </body>` (вся веб-страница), например, как это показано на рисунке 3.9 (файл `ex3_09.html`).

```
<html>
<body onload="more()">
Здесь какой-то текст
<script>
function more() { alert('Море!'); }
</script>
</body>
```

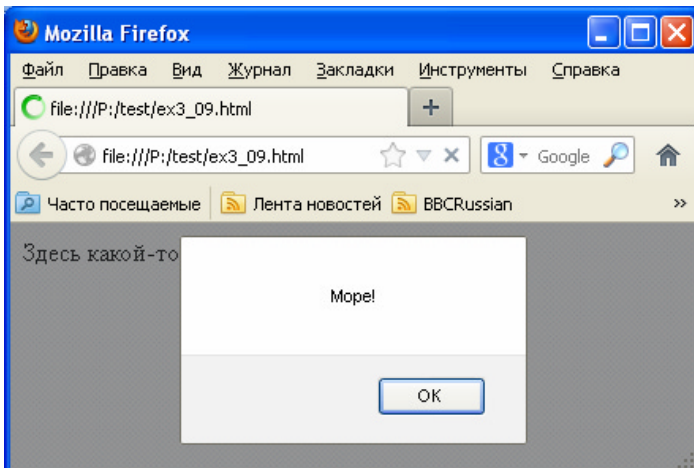


Рисунок 3.9 - Иллюстрация использования события **onload**

### 3.3. Изменение содержимого веб-страниц

Объект `document` имеет ряд полезных функций, например, `getElementById(id)`, которые позволяют получить доступ к элементам веб-страницы и менять их содержание. На рисунке 3.10 приведен пример использования этой функции (файл `ex3_10.html`).

```
<html>
<body>

<style>
  div {
    background-color:yellow;
    padding:15px;
  }
</style>

<script>
function primer()
{
  s=document.getElementById('olga');
  s.innerHTML='Меня переехали!';
}
</script>
```

Пример изменения текста  
`<div id="olga" onmouseover="primer()">`  
 Желтый прямоугольник `</div>`  
  
`</body>`

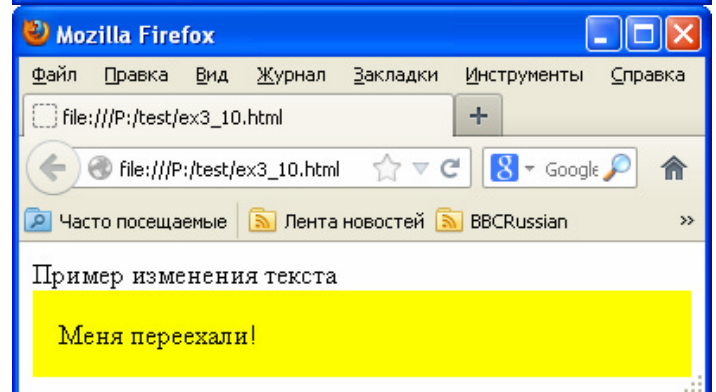
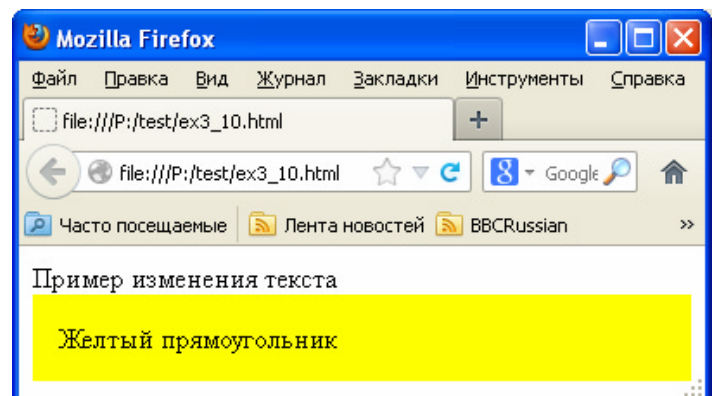


Рисунок 3.10 - Пример использования события **onload**

Если навести мышку на желтый прямоугольник (событие **mouseover**), то вызывается функция `primer()`.

Эта функция находит элемент веб-страницы, у которого `id="olga"` (с помощью функции `getElementById('olga')`) и

записывает указатель на этот элемент в переменную **s** (название из одной буквы мы дали этой переменной, чтобы текст программы был более кратким).

Далее мы заменяем значение поля **innerHTML** (**innerHTML** - это весь текст веб-страницы, который находится внутри данного тега `<div>..</div>`) на текст "Меня переехали!". В результате вместо надписи "Желтый прямоугольник" появится надпись "Меня переехали!".

По аналогии можно менять текст страницы, значения элементов страницы, значения свойств стилей. Алгоритм состоит всего из двух шагов: 1) Найти элемент по его имени или id; 2) Указать, какое поле этого элемента нужно поменять и указать новое значение для него.

```
<html>
<body>

<style>
  div {
    background-color:yellow;
    padding:15px;
  }
</style>

<script>
function primer()
{
  s=document.getElementById('olga');
  s.style.backgroundColor='#ff8888';
  if (s.innerHTML.length<25)
    s.innerHTML=s.innerHTML+'<br>Меня
переехали!';
  else
    s.innerHTML=s.innerHTML+'<br>Меня
опять переехали!';
}
</script>
```

Пример изменения текста  
 <div id="olga" onmouseover="primer()">  
 Желтый прямоугольник </div>  
 </body>

На рисунке 3.11 приведен предыдущий пример, но немного дополненный: при наведении мышки меняется цвет прямоугольника, а надпись не замещает старый текст, а добавляется к нему (файл ex3\_11.html).

Здесь по событию мы меняем цвет фона прямоугольника `<div>` с желтого на светло-красный (`#ff8888`). Для этого изменяется поле **backgroundColor** поля **style** найденного по id элемента **s** веб-страницы. Любое другое свойство стиля можно менять по этой же схеме:

переменная.style.свойство = значение;

Например, можно было бы написать так:

```
s.style.padding='25px';
      или
s.style.fontFamily='Verdana';
```

Обратите внимание, что свойства стилей в CSS называются "**background-color**" и "**font-family**", а в JavaScript они называются "**backgroundColor**" и "**fontFamily**". Это из-за того, что в названиях переменных и полей в JavaScript нельзя использовать знак "-" (если написать `font-family`, то браузер подумает, что мы хотим из значения переменной `font` вычесть значение переменной `family`). Общее правило перевода названий свойств из CSS в JavaScript таково: выбрасываем знак "-", а следующую после него букву делаем большой. Например, если нам потребуется поменять значение свойства **background-position**, то в JavaScript нужно писать **backgroundPosition**.

Дописать текст вместо замещения просто. Вместо

s.innerHTML='новый текст';

мы пишем

s.innerHTML= s.innerHTML+'новый текст';

И последнее пояснение к этому примеру. Поле `length` любой строки содержит ее длину. Например, если мы напишем так:

```
s='Nikolay';
u=s.length;
```

то в переменной `u` будет находиться число 7.

В нашем следующем примере мы рассмотрим событие **onChange** и попробуем изменить атрибут **value** тега `<input>` HTML.

Тег `<input>` относится к элементам форм HTML, которые подробнее мы рассмотрим на следующей лекции вместе с основами PHP. Здесь же нам потребуется две строки ввода, которые можно организовать, если тегу `<input>` задать значение атрибута `type="text"`. Значение атрибута **value** тега `<input>` будет при этом содержать текущее значение введенной строки.

Наша веб-страница будет вычислять факториал заданного числа. В одну из двух строк будем вводить число, а во второй выводить факториал этого числа.

Чтобы узнать, что пользователь ввел какое-то число, будем отслеживать событие **onChange** (данные изменились) для строки, в которую пользователь вводит число.

На рисунке 3.12 приведен текст этой веб-страницы (файл ex3\_12.html) и результат ввода пользователем числа 7.

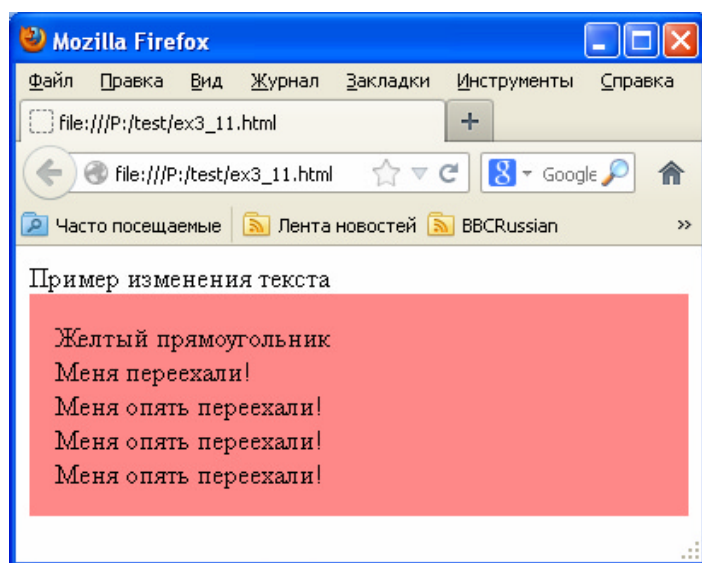


Рисунок 3.11 - Окно браузера после нескольких наведений курсора мышки на цветной прямоугольник с текстом

```

<html>

<script>
function faktorial(n)
{
    var k=1;
    for (var i=1;i<=n;i++)
        k=k*i;
    return k;
}

function obrabotka()
{
    n=document.getElementById("mama").value;
    document.getElementById("otvet").value =
    faktorial(n);
}
</script>

<body>
Число: <input id="mama"
onChange="obrabotka()" type="text"
value="1"></input> <br>

Факториал: <input id="otvet" type="text"
value="1"></input>
</body>

```

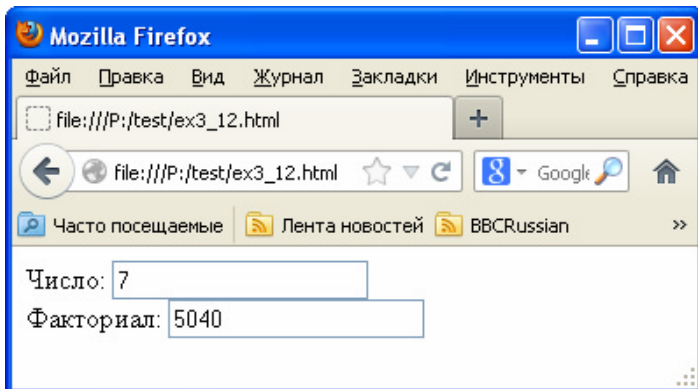


Рисунок 3.12 - Отслеживание события **onChange** и изменение значений поля **value** элементов формы

Попробуйте запустить эту страницу. Обратите внимание, что событие возникает не сразу же, после того, как мы кликнули на строку ввода мышкой и нажали на клавиатуре цифру 7, а только после того, как мы ушли с измененной строки ввода (кликнули мышкой где-то в другом месте веб-страницы или нажали клавишу “Enter”). Нужно учитывать это при использовании события **onChange**.

### 3.4. Отслеживание координат мыши

Иногда при обработке события нужно знать, где в этот момент находился курсор мышки. Для этого нужно не только знать о событии (движении мышки), но и знать его параметры (координаты курсора мышки). Для этого нужно перехватить событие движения мышки и назначить функцию, которая будет получать параметры события. В общем виде эта процедура выглядит так:

```

document.captureEvents(Event.Событие);
document.onmousemove=функция;

```

Здесь “событие” - название того события, которое нам нужно перехватить, а “функция” - название нашей функции, которая будет обрабатывать событие.

На рисунке 3.13 приведена короткая веб-страница, показывающая, как это сделать (файл `ex3_13.html`).

```

<html>
<body>

<script>
    function khai(eve)
    {
        s=document.getElementById('taras');
        s.innerHTML='X='+eve.pageX+'
Y='+eve.pageY;
    }

    document.captureEvents(Event.MOUSEMOVE);
    document.onmousemove=khai;
</script>

<div id="taras"></div>
</body>

```

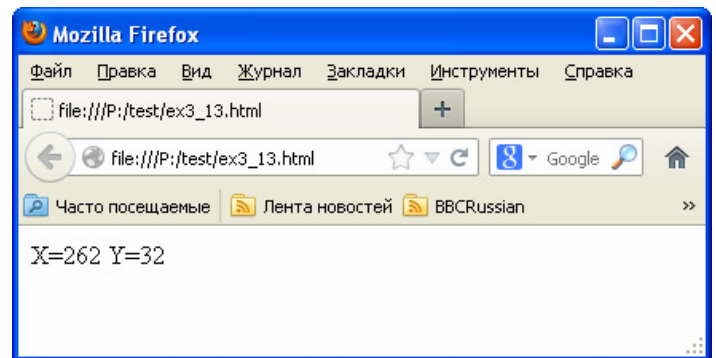


Рисунок 3.13 - Простая веб-страница с отслеживанием координат курсора мышки

В данном примере мы перехватили событие **MOUSEMOVE**:

```
document.captureEvents(Event.MOUSEMOVE);
```

и назначили функцию `khai` для его обработки:

```
document.onmousemove=khai;
```

В функцию `khai` передается параметр `eve` (все данные о событии), у которого есть интересующие нас поля `eve.pageX` (координата X курсора мышки) и `eve.pageY` (координата Y курсора мышки).

Функция `khai` получает доступ к содержимому прямоугольника `<div>` с именем “taras”

```
s=document.getElementById('taras');
```

и выводит на него полученные в `eve` координаты курсора мышки:

```
s.innerHTML='X='+eve.pageX+' Y='+eve.pageY;
```

Возьмите себе за основу этот пример. А мы рассмотрим более сложную веб-страницу, где нам понадобятся все наши знания (рисунок 3.14, файл `ex3_14.html`).

```

<html>
<style>
.qq, .ima, .big {
    position: absolute;
}
.qq, .ima { background:url('griv.png'); }
.qq, .big {
    border: 1px solid black;
    border-radius:15px;
    visibility:hidden;
}
.qq {
    top:9px;left:9px;
    cursor:crosshair;
    width:80;height:80;
}
.ima {
    width:512;height:377;
    top:10px;left:10px
}
.big {
    background:url('grivb.jpg');
    top:10px;left:550px;
    width:320;height:320;
}
</style>

<script>
var posX=10; posY=10; LoY=377; LoX=512;
function khai(eve)
{
    mx=eve.pageX;my=eve.pageY;
    if ((mx>posX+40) & (mx<posX+LoX-40) & (my>posY+40) & (my<posY+LoY-40) )
    {
        lup.top=my-41;lup.left=mx-41;
        lup.backgroundPosition="- "+(mx-50)+"px - "+(my-50)+"px";
        bol.backgroundPosition="- "+(mx-10)*4-160)+"px - "+(my-10)*4-160)+"px";
        lup.visibility="visible";
        bol.visibility="visible";
    }
    else
    {
        lup.visibility="hidden";
        bol.visibility="hidden";
    }
}
function myst()
{
    lup=document.getElementById('lupa').style;
    bol=document.getElementById('uvel').style;
    document.captureEvents(Event.MOUSEMOVE);
    document.onmousemove=khai;
}
</script>
<body>
<div class=ima id="small"></div>
<div class=qq id="lupa"></div>
<div class=big id="uvel"></div>
<script> myst() </script>
</body>

```

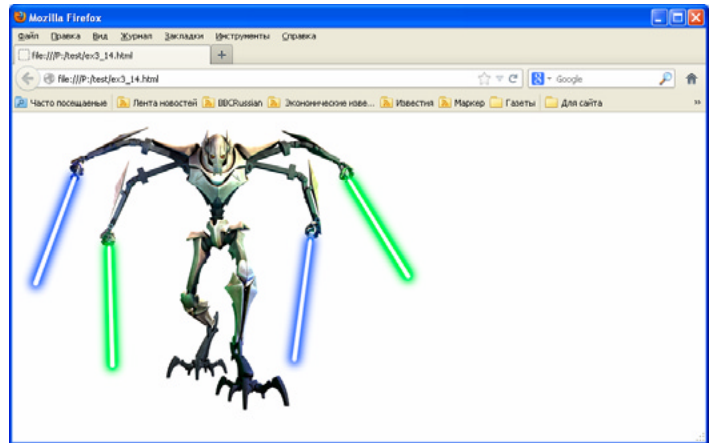


Рисунок 3.14 - Страница с масштабированием изображения

Попробуйте самостоятельно разобраться в этом тексте веб-страницы. Все, что здесь используется, мы уже знаем.

Анимация в этой веб-странице (при наведении курсора на изображение справа появляется его увеличенный фрагмент, см. рисунок 3.15) осуществляется с помощью изменения свойств стилей из JavaScript.

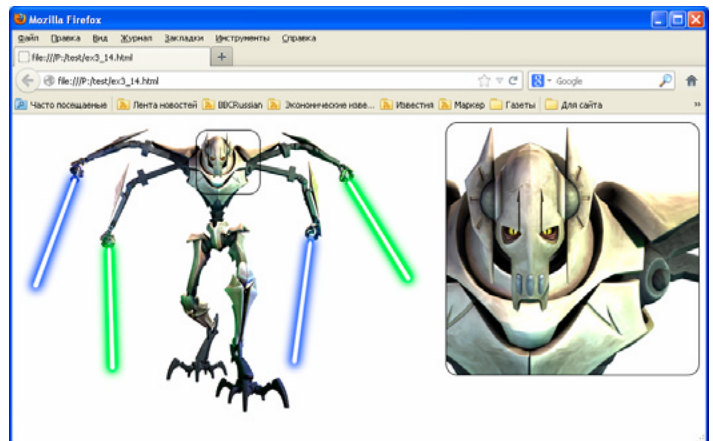


Рисунок 3.15 - При наведении курсора мышки на изображение справа появляется его увеличенный фрагмент

Стиль “qq” отвечает за лупу на уменьшенном изображении, стиль “ima” - за уменьшенное изображение, а стиль “big” - за увеличенный фрагмент изображения, на который наведена лупа.

### 3.5. Объект window и события, привязанные к таймеру

Мы уже не раз использовали стандартный для JavaScript объект **document**. Рассмотрим еще один стандартный объект - **window**.

Объект **window** дает доступ к функциям браузера. Например, можно открыть новое окно браузера (вызвав функцию **window.open()**) или закрыть существующее (вызвав функцию **window.close()**). Можно также менять какие-то свойства окна браузера, например, задать текст, который в данный момент выводится в строке статуса (**window.status**=’Текст, который нужно вывести’). Рассмотрите методы и свойства объекта **window** самостоятельно по мере практической необходимости. Возможно, на практике Вам потребуется создавать свои собственные объекты. С помощью любого справочника посмотрите, как это делать.



Последний тип событий, который мы здесь рассмотрим - это события времени (таймера). Чтобы привязать функцию к таймеру, нужно в программе написать:

Здесь myfun - имя функции, которая будет вызываться один раз в 3000 миллисекунд (раз в 3 секунды), а m - переменная, которая будет нужна, если мы захотим выключить таймер. Таймер в этом случае выключается командой:

```
window.clearInterval(m);
```

На рисунке 3.16 приведен пример веб-страницы с обработкой событий времени файл ex3\_15.html).

```
<html>
<body>
<script>
  n=0;
  function ok()
  {
    n=n+1;
    s=document.getElementById('ua');
    s.innerHTML=n+' sec';
  }
  m=window.setInterval(ok,1000);
</script>
<div id="ua"></div>
</body>
```

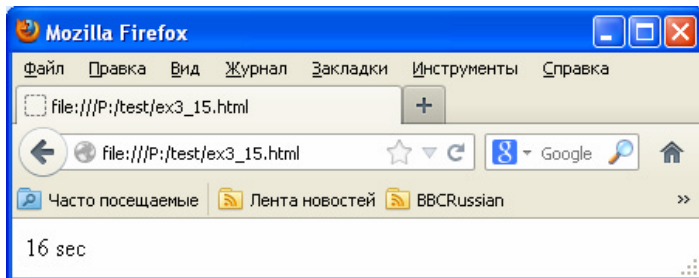


Рисунок 3.16 - Обработка событий таймера

На этом данная лекция заканчивается. Дальше вы будете изучать JavaScript самостоятельно, усваивая новый материал по мере того, как это потребует при решении той или иной практической задачи.

К сожалению, таких хороших справочников, как для HTML и CSS, для JavaScript нет. Можно пользоваться, например, такими справочниками, как <http://javascript.narod.ru> или <http://javascript.ru>, но они не очень удобны, понятны и самодостаточны. Придется часто использовать поиск в Google, например вот по таким запросам: “как узнать время из JavaScript”, “как сгенерировать случайное число на JavaScript” и т.д. Так будет проще найти нужную информацию, чем в справочнике.

Позже (на лекции №6) мы рассмотрим библиотеку JQuery - набор функций, написанных на JavaScript и облегчающих работу с JavaScript (обработку событий, анимацию, отправку и получение запросов к серверу, работу с cookies).

## 4. Основы языка программирования PHP

PHP - расшифровывается как язык гипертекстовой обработки.

Основными функциями языка PHP являются:

- Реализация серверной части сайтов (хранение, анализ и обработка данных) - **back-end** приложения;
- Автоматическая генерация текстов веб-страниц на сервере;
- Обработка запросов веб-страниц;
- Загрузка и анализ веб-страниц других сайтов;
- Создание скриптов для выполнения в командной строке.

В отличие от JavaScript это практически полноценный язык программирования с возможностью работы с файлами и с операционной системой. В то же время это интерпретатор (как и JavaScript), а не компилятор.

Как видно, для автоматического генерирования веб-страниц можно использовать и JavaScript и PHP. Преимуществом PHP является то, что он исполняется на сервере и имеет доступ к файлам, хранящимся на сервере. Кроме того, пользователь после загрузки веб-страницы может посмотреть текст JavaScript в браузере, а текст PHP удаляется сервером из кода страницы после исполнения.

### 4.1. Установка веб-сервера на локальный компьютер

Если для запуска программ, написанных на JavaScript, достаточно лишь веб-браузера, то для запуска программ, написанных на языке PHP, понадобится веб-сервер.

Если имеется **ftp**-доступ к **хостингу** с реальным Интернет-сайтом, то тестируемые PHP-программы можно выкладывать на этом сайте и запускать оттуда. Но такой подход далеко не всегда удобен и приемлем, особенно, когда разрабатывается сложное серверное приложение, состоящее из многих веб-страниц, PHP-файлов и, возможно, интегрированное с базами данных MySQL. В этом случае наиболее подходящим решением является установка веб-сервера на локальный компьютер и отладка PHP-программ на локальном компьютере.

Здесь мы будем использовать для отладки PHP-программ бесплатный набор программного обеспечения Denver, который включает в себя веб-сервер Apache, PHP, MySQL и все необходимое.

Скачать Denver можно с сайта <http://www.denver.ru> (дистрибутив занимает меньше 10 Мбайт в архиве). При установке Denver-а соглашайтесь на все предлагаемые по умолчанию настройки. Процесс установки займет не больше 10 минут.

После того, как Denver установлен на компьютер, на рабочем столе появятся иконки “Start Denver”, “Stop Denver” и “Restart Denver”. Для запуска Denver-а нужно кликнуть по иконке “Start Denver”. При этом он загрузится в память, но никаких окон на экране не останется, кроме иконки в панели задач. Для выгрузки Denver-а из памяти, нужно нажать на иконку “Stop Denver”.

Внимание, Denver может конфликтовать с программой SKYPE (использовать один и тот же порт 80). Поэтому перед запуском Denver-а может потребоваться временно выключить SKYPE. Или же уберите галочку в настройках SKYPE в пункте “Использовать порты 80 и 443 в качестве входящих альтернативных”.

Denver создаст на компьютере виртуальный диск (обычно это диск Z). Это делается для нашего удобства (физически файлы лежат на диске C в папке “C:\WebServers”). На этом диске в папку “Z:\home\localhost\www\” мы будем помещать разрабатываемые нами PHP-программы и оттуда их запускать на исполнение.

## 4.2. Настройка Apache

По умолчанию в Denver PHP-код в файлах с расширением “.html” не исполняется (а нам это потребуется). Поэтому необходимо будет выполнить небольшую (и единственную) правку файла конфигурации ‘Z:\usr\local\apache\conf\httpd.conf’:

- 1) Находим в файле конфигурации строку, начинающуюся на “AddType application/x-httpd-php” и дописываем в ее конец “.html” и “.htm” (с пробелами впереди).
- 2) Перед этой строкой вставляем строку: “RemoveHandler .html .htm”
- 3) Удостоверяемся, что в директиве “AddHandler server-parsed” (если такая строка есть) нет расширений “.html” и “.htm” (если есть - удаляем их).

## 4.3. Первая программа на PHP

Этот материал рассчитан на тех, кто уже знаком с каким-либо языком программирования (например, C/C++ или Pascal). Мы не будем подробно рассматривать синтаксис языка программирования PHP, а сосредоточимся на примерах, которые позволят нам как можно быстрее начать писать программы на языке PHP. Тонкости языка программирования и его синтаксиса будем изучать по мере того, как они нам понадобятся на практике. Можно делать это и самостоятельно с помощью справочника (например, сайт <http://php.ru>).

Программа на языке PHP может либо находиться в текстовом файле с расширением \*.php, либо быть интегрированной прямо в текст HTML-страницы (файл с расширением \*.html). Редактировать PHP-программу можно в любом текстовом редакторе, например в notepad++.

На рисунке 4.1 приведен текст HTML-страницы с внедренной в нее PHP-программой и результат ее загрузки в веб-браузер (файл ex4\_01.html).

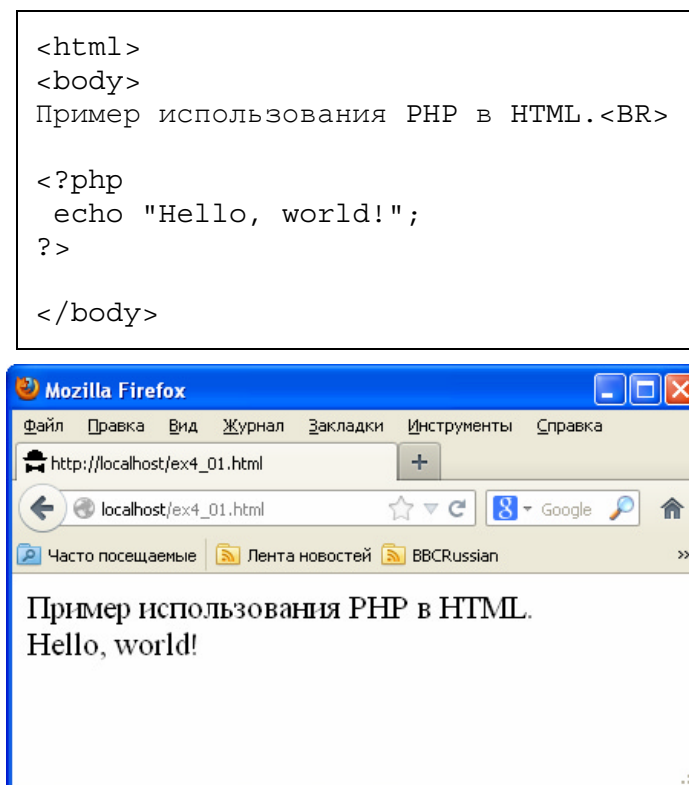


Рисунок 4.1 - Пример простой страницы с PHP-кодом

Данную веб-страницу, как говорилось выше, мы предварительно должны положить в папку “Z:\home\localhost\www\”, а в браузере написать не просто “ex4\_01.html”, а “http://localhost/ex4\_01.html”. Это все, что нужно запомнить при запуске PHP-программ и HTML-страниц с PHP-кодом с помощью Denver.

Теперь посмотрим, собственно, на текст HTML-страницы. PHP-код в ней помещается внутри специального HTML-тега “<?php ?>”. Между знаками вопроса здесь нужно вставлять текст PHP-программы, которая будет выполнена на сервере. При этом результат выполнения программы (весь текст, что программа выводит с помощью команды **echo**) сервер помещает прямо в текст веб-страницы в то место, где была эта программа.

В данной программе нет ничего, кроме вывода на экран (в тексте HTML-страницы) строки “Hello, world!” командой **echo**. Текстовые строки на языке PHP нужно брать в кавычки (в одинарные или в двойные - все равно).

На рисунке 4.2 приведен экран браузера при просмотре в нем исходного текста веб-страницы.

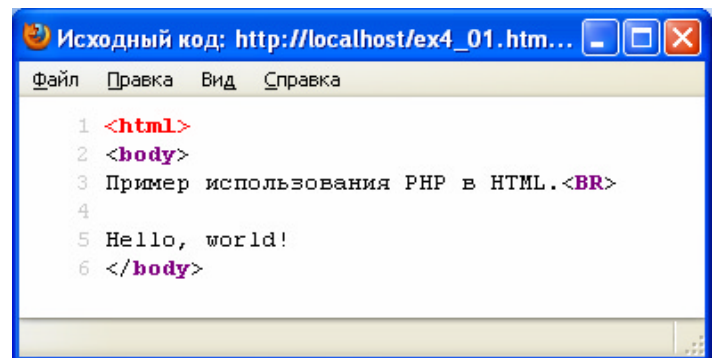


Рисунок 4.2 - Попытка увидеть PHP-код при просмотре исходного текста веб-страницы в браузере

На рисунке мы действительно видим исходный текст страницы, однако вместо текста вставленной в эту страницу PHP-программы мы видим лишь результат ее выполнения - надпись “Hello, world!”.

Этот пример иллюстрирует одно из наиболее важных свойств PHP-программы. Сервер никогда не передает текст PHP-кода браузеру. Сервер исполняет PHP-код, удаляет его из страницы, а на его место вставляет результат всех команд **echo** кода. Таким образом, можно не опасаться, что текст PHP-программы будет кем-либо похищен - она никогда не покидает сервера. В то же время текст программы на языке JavaScript, на котором пишут клиентские приложения (**front-end**), доступен для просмотра в браузере и его можно похитить (воровство кода программ можно лишь затруднить с помощью **обфускации** текста программы).

На рисунке 4.3 приведен текст этой же страницы (в браузере выглядит точно так же), но оформленной в виде файла с расширением \*.php (файл ex4\_01.php). Браузеры понимают расширение \*.php и интерпретируют такие веб-страницы так же, как и HTML-страницы.

Как видим, здесь уже нельзя просто вставлять HTML-текст. Все, что будет содержать веб-страница, нужно выводить на нее командами **echo**, а текст обязательно брать в кавычки.

Так же видно, что в данном случае теги <html>, <body> не обязательны (но, при желании и их можно добавить на страницу с помощью команды **echo**).

```
<?php
echo "Пример использования PHP в
HTML.<BR>";
echo "Hello, world!";
?>
```

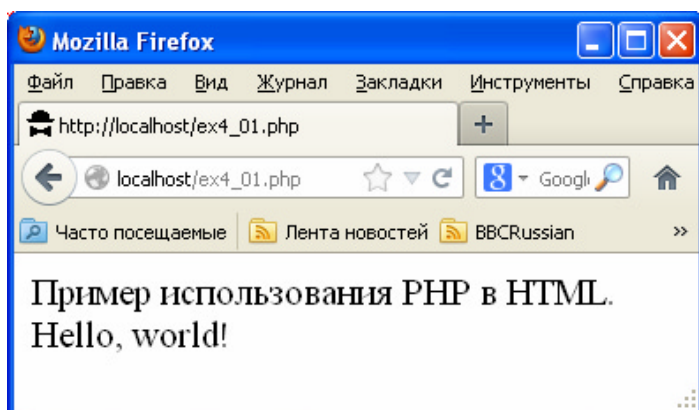


Рисунок 4.3 - Эта же страница, только оформленная в виде файла с расширением \*.php

На рисунке 4.4 показан исходный текст этой веб-странице при попытке его просмотреть в браузере.

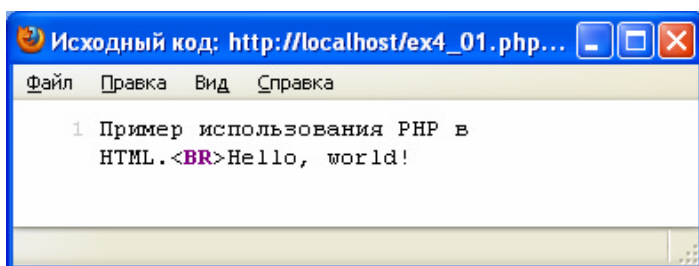


Рисунок 4.4 - Попытка увидеть PHP-код программы с рисунка 4.3 при просмотре исходного текста веб-страницы в браузере

Видно, что в браузер передаются только результаты работы команды **echo** PHP-программы.

Оба способа написания PHP-программ (в отдельных PHP-файлах или внутри HTML-страницы) эквивалентны друг другу. Но написание PHP-кода в файле с расширением \*.php позволяет нам проверить текст программы на наличие ошибок. Для этого можно использовать файл php.exe, находящийся в папке "Z:\usr\local\php5\\" Denvera. Чтобы проверить на наличие ошибок, например, программу "ex4\_01.php", нужно в командной строке файлового менеджера набрать:

```
php.exe -l ex4_01.php
```

Если в программе есть ошибки, то php.exe сообщит нам об этом с указанием номеров строк, в которых содержатся ошибки.

#### 4.4. Автоматическое генерирование текста страницы

В вышеприведенном примере программы всего лишь выводит текстовую строку. Но для этого не нужен язык программирования.

Приведем более сложный текст программы, в котором она генерирует такой текст страницы, который сложно создать вручную. Программа выведет на экран все целые числа с 1 до 200, выделив красным цветом простые числа.

```
<html>
<body>
Простые числа выделены красным
цветом.<BR>

<style>
.myred {
    color:red;
    display:inline;
}
</style>

<?php
function prost($n)
{
    for ($i=2;$i<$n;$i++)
        if (($n % $i)==0) return 0;
    return 1;
}

for ($papa=1;$papa<=200;$papa++)
    if (prost($papa)==0)
        echo $papa, ' ';
    else
        echo '<div class="myred"> ',
        $papa, ' </div>';
?>

</body>
```

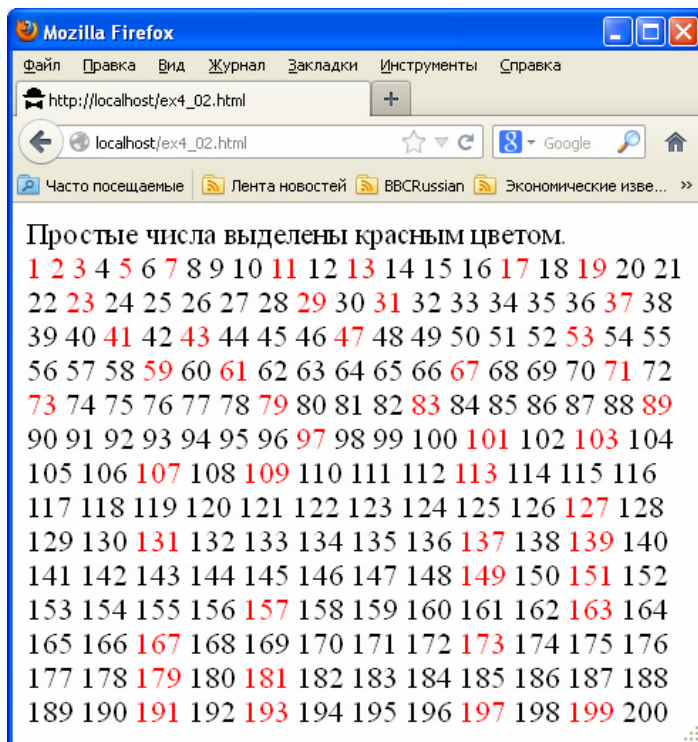


Рисунок 4.3 - PHP-программа выделяет красным цветом простые числа

На рисунке 4.3 приведен текст веб-страницы (файл ex4\_02.html) и окно браузера при выводе этой страницы на экран.

Для полноты картины приведем и исходный текст страницы (рисунок 4.4) при попытке просмотра его из окна



браузера (так можно увидеть результат выполнения PHP-программы).

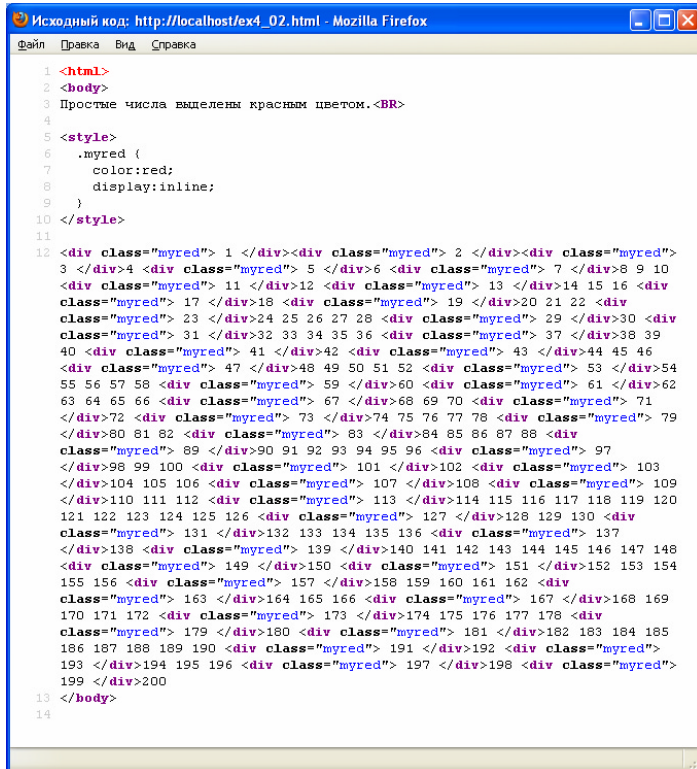


Рисунок 4.4 Результат выполнения PHP-программы на рисунке 4.3

Здесь мы уже видим достаточно сложную программу, в которой есть функция, циклы, условные операторы, переменные. Видно, что синтаксис языка PHP очень похож на синтаксис языка C, однако есть и отличия.

Во-первых, все имена переменных в PHP-программах должны начинаться со знака доллара “\$”. Во-вторых, типы переменных не обязательно объявлять – PHP сам поймет, что это за переменная по тому значению, которое мы в нее записываем.

Еще одной особенностью языка PHP по сравнению с C/C++ является требование, чтобы все условия в условных операторах обязательно брались в скобки. Нельзя написать так:

```
if $k>5 { $k=$k-3;$x=$x*2; }
```

Нужно обязательно писать так:

```
if ($k>5) { $k=$k-3;$x=$x*2; }
```

Попробуйте разобраться сами, как работает программа на рисунке 4.3. Скажем лишь, что функция “prost” возвращает 0, если число не простое, и 1, если число простое (ни на что не делится, кроме единицы и самого себя). Также отметим, что “%” в выражении “\$n % \$i” означает “остаток от деления числа \$n на число \$i”.

На рисунке 4.5 приведена программа (файл ex4\_03.php), которая выводит на экран список поселков городского типа Харьковской области, хранящийся на сервере в файле “kh.txt”.

В этом примере мы знакомимся с некоторыми функциями PHP для работы с файлами (**fopen** открывает файл, **fgets** читает из файла одну строку, **fclose** закрывает файл, **feof** возвращает значение “истина”, если все строки из файла уже прочитаны).

```
<html>
<body>
Посёлки городского типа Харьковской
области:<HR>

<?php
    $f=fopen('kh.txt','r');
    while (!feof($f))
    {
        $s=fgets($f);
        echo $s,'<br>';
    }
    fclose($f);
?>

</body>
```

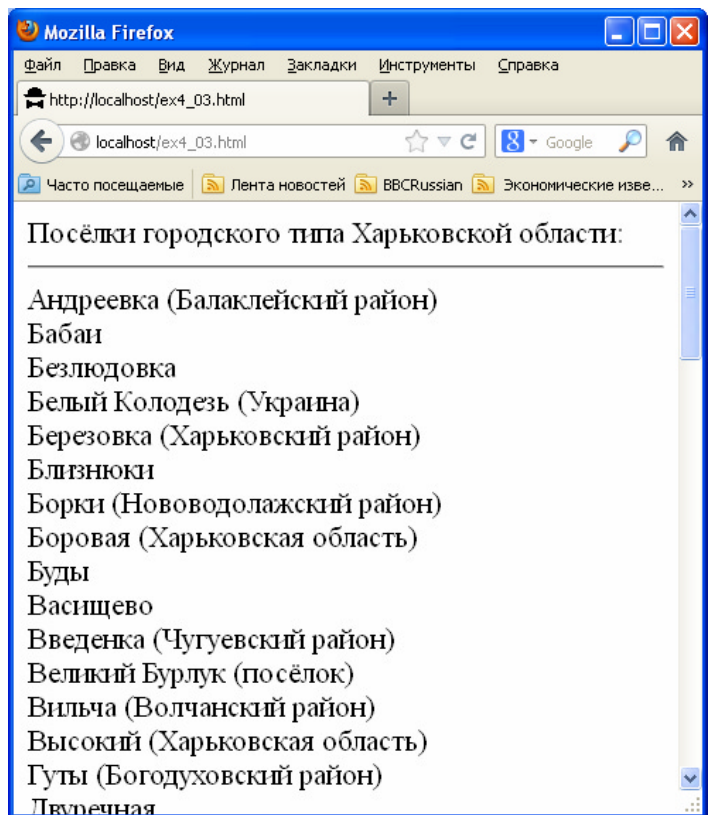


Рисунок 4.5 - PHP-программа, которая читает данные из файла и выводит их на веб-страницу

Здесь специально используются простые конструкции языка PHP, которые есть и в других языках программирования. Более сложные и красивые функции PHP, которые позволяют сделать текст программы еще короче, изучите сами в процессе приобретения опыта в программировании на PHP.

Еще в этой программе мы видим цикл **while** (такой же, как и в C/C++), а также восклицательный знак перед условием, который на PHP означает отрицание. Если у условия значение “истина”, то после отрицания оно изменится на “ложь”. И наоборот, если у условия значение “ложь”, то после отрицания оно изменится на “истина”.

Давайте модифицируем программу на рисунке 4.5 так, чтобы она выводила на экран только поселки, начинающиеся с буквы “Б” (файл ex4\_04.html, рисунок 4.6).



```

<html>
<body>
Посёлки городского типа Харьковской
области:<HR>

<?php
    $f=fopen('kh.txt','r');
    while (!feof($f))
    {
        $s=fgets($f);
        if ($s[0]=='Б')
            echo $s,'<br>';
    }
    fclose($f);
?>

</body>

```

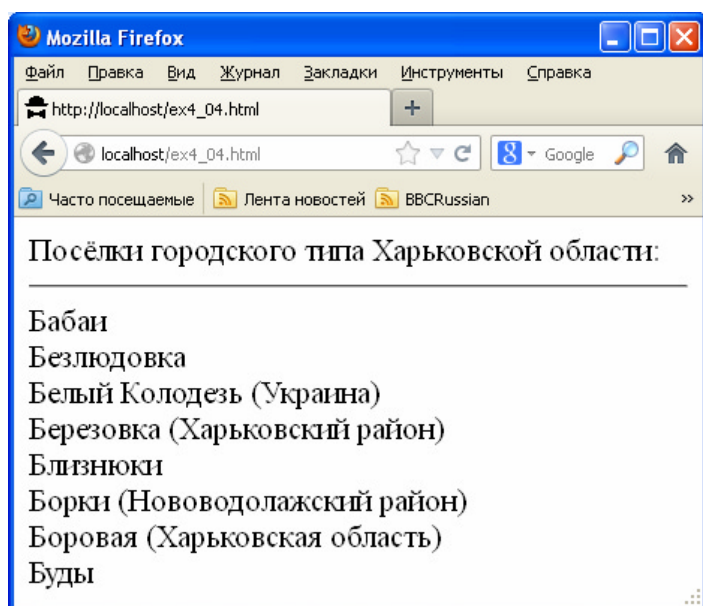


Рисунок 4.6 - Модификация PHP-программы с рисунка 4.5, которая выводит на веб-страницу только поселки, начинающиеся на букву “Б”

Поясним новые особенности языка программирования PHP, которые встретились нам в этом примере.

Переменная `$s` содержит текстовую строку. Текстовая строка на языке PHP - это массив букв. У массива есть длина (ее можно получить функцией `strlen($s)`). Индексы любого массива на языке PHP начинаются с нуля. Поэтому, чтобы обратиться к первой букве строки `$s`, нужно написать `$s[0]`. Чтобы обратиться к второй букве строки `$s`, нужно написать `$s[1]` и т.д. Чтобы обратиться к последней букве строки `$s`, нужно написать `$s[strlen($s)]`. Если в программе нужно последовательно перебрать все буквы строки `$s` и что-то с ними сделать, то можно организовать цикл от 0 до `strlen($s)-1`, примерно вот так:

```
for ($n=0;$n<strlen($s);$n++)
```

И уже в теле цикла можно обращаться к буквам строки `$s` как `$s[$n]`.

В вышеприведенном примере мы просто проверяем, равна ли первая буква строки `$s` букве “Б”. Здесь “==” - логический оператор, который возвращает “истина”, если

`$s[0]` равно “Б” (точно так же равенство проверяется и в языке C/C++).

Приведем пример еще одной программы (см.рисунок 4.7), которая выводит на веб-страницу название текущего дня недели. Для разнообразия пусть это будет не \*.html, а \*.php файл (файл `ex4_05.php`).

```

<?php

    $myday=array("воскресенье",
    "понедельник", "вторник", "среда",
    "четверг", "пятница", "суббота");
    $today=getdate();
    echo "Сегодня ";
    $day=$today[wday];
    echo $myday[$day];

?>

```

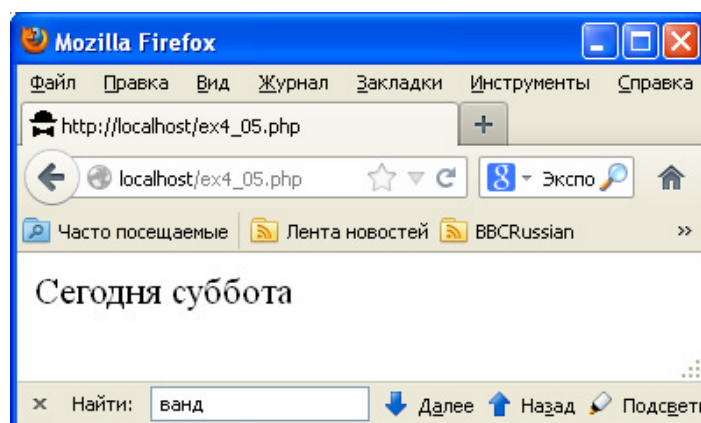


Рисунок 4.7 - PHP-программа, которая выводит на веб-страницу текущую дату

Рассмотрим текст этой программы. Вначале объявлен массив с именем `$myday`, содержащий 7 строк:

```
$myday=array("воскресенье", "понедельник",
"вторник", "среда", "четверг", "пятница",
"суббота");
```

Как уже было сказано, нумерация элементов массивов на языке PHP начинается с нуля. Чтобы обратиться к элементу массива, нужно написать его имя, а затем в квадратных скобках - значение индекса (номера элемента). Если написать `$s = $myday[0]`, то в `$s` будет записано слово “воскресенье”. Если же написать, например, `$s=$myday[3]`, то в `$s` будет записано слово “среда”. Если в переменной `$k` находится число 6, и, если написать `$s=$myday[$k]`, то в `$s` будет записано слово “суббота”.

Далее в тексте программы в переменную, названную нами `$today`, заносится результат выполнения функции `getdate()`. Функция `getdate()` является стандартной функцией PHP (пустые скобки свидетельствуют о том, что у этой функции нет входных параметров), а результатом ее работы является ассоциативный массив данных о текущей дате.

По мере практической необходимости Вы самостоятельно разберетесь с особенностями работы с простыми, ассоциативными и многомерными массивами на PHP (в PHP больше разных возможностей для работы с массивами, чем в C/C++). Здесь же скажем лишь, что

ассоциативный массив отличается от обычного тем, что в качестве индексов массива используются не цифры, а ключевые слова, заданные при объявлении массива. Например, ассоциативный массив можно объявить вот так:

```
$a["imya"]="Nikolay";
$a["familiya"]="Ponomarenko";
$a["gorod"]="Kharkov";
$a["denxr"]=1970;
```

В данном случае в массиве \$a есть четыре ячейки. В первые три ячейки с именами imya, familiya и gorod мы занесли строки текста, а в четвертую ячейку с именем denxr мы занесли число. На PHP так можно делать.

Есть и чуть более компактный способ объявить этот массив:

```
$a=array("imya"=>"Nikolay",
"familiya"=>"Ponomarenko",
"gorod"=>"Kharkov", "denxr"=>1970);
```

Результат будет таким же. Вообще же PHP, как правило, предоставляет несколько альтернативных возможностей добиться нужного результата. Не обязательно знать все из них.

Теперь, обратиться к полям объявленного массива \$a можно, например, вот так:

```
echo $a["familiya"], " ", $a["denxr"];
```

На веб-страницу будет выведена надпись "Ponomarenko 1970". А можно и так (без кавычек в именах индекса):

```
echo $a[familiya], " ", $a[denxr];
```

Результат будет тем же самым.

Вернемся теперь к программе на рисунке 4.7. Строка "\$day=\$today[wday];" означает, что в переменную \$day мы заносим значение ячейки массива \$today с индексом "wday". Это номер дня недели, число с возможными значениями от 0 до 6 (см. описание функции **getdate()** в справочнике).

В последней строке программы командой **echo** выводится на веб-страницу значение ячейки массива \$myday (туда мы занесли названия всех дней недели) с номером ячейки, равным \$day (там у нас номер дня недели).

В языке PHP есть функция **print\_r**, которая позволяет вывести на экран заданный массив с названиями и значениями его ячеек. На рисунке 4.8 приведена программа (файл ex4\_06.php), которая выводит на экран значения массивов \$myday и \$today из предыдущего примера.

Здесь **<pre>** .. **</pre>** - это тег HTML, который выводит текст моноширинным шрифтом с сохранением всех пробелов и переводов строки.

Как видно, у простого массива \$myday, объявленного нами, PHP автоматически пронумеровал индексы от 0 до 6. Видны также все названия индексов (имен ячеек) массива \$today, который содержит результат работы функции **getdate()**.

```
<?php

$myday=array("воскресенье",
"понедельник", "вторник", "среда",
"четверг", "пятница", "суббота");
$today=getdate();
echo "<pre>";
print_r($myday);
print_r($today);
echo "</pre>";

?>
```

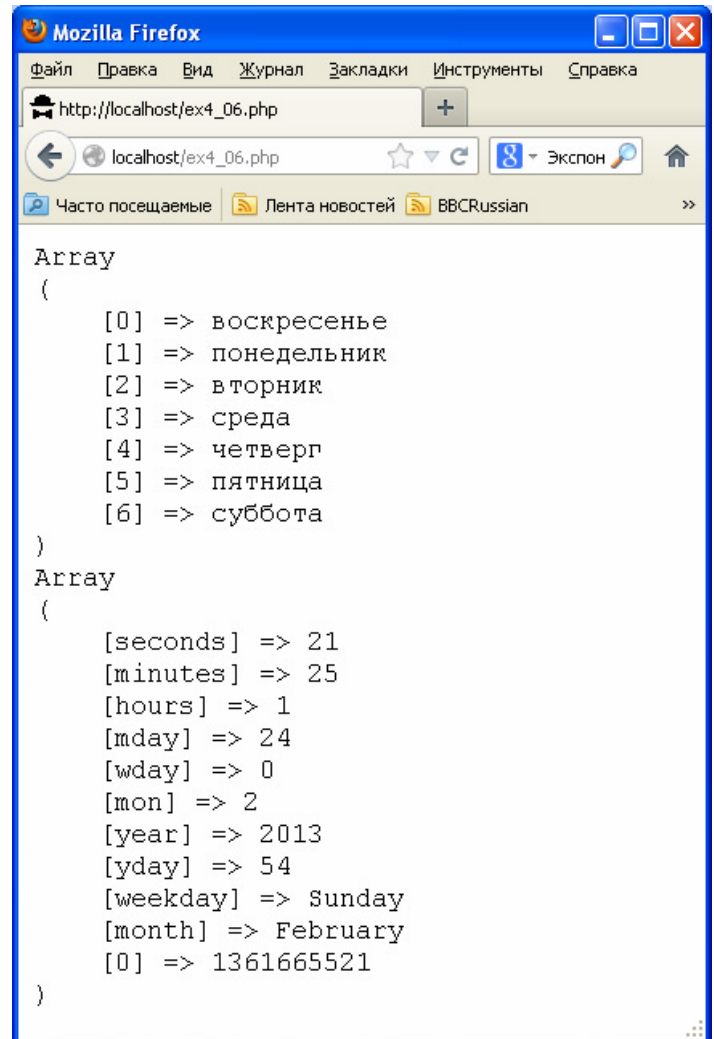


Рисунок 4.8 - Иллюстрация функции **print\_r**

Обращаться из программы к ячейкам ассоциативного массива \$today можно как с использованием кавычек: \$today["year"], так и без использования кавычек: \$today[year].

#### 4.5. Обработка запросов к серверу на PHP

Одно из основных назначений PHP - обработка запросов к серверу, поступающих от веб-страниц или других серверов. На рисунке 4.9 приведен пример такой страницы (файл ex4\_07.html) с новыми для нас тегами **<form>** .. **</form>** и **<input>**.

```

<html>
<body>

<form action="primer.php"
method="POST">
  <p>Ваше имя?</p>
  <p><input type="text" name="nam"
size="50"></p>
  Придумайте пароль:
  <input type="password" name="oleg"
size="20">
  <br><br>Сформируйте заказ:
  <p><input type="checkbox"
name="cofe">Кофе
  <input type="checkbox" name="tea"
checked>Чай
  <input type="checkbox" name="beer"
checked>Пиво</p>
  <br>Выберите способ оплаты:
  <p><input type="radio"
name="money" value="var1"
checked>Наличными
  <input type="radio" name="money"
value="var2">Кредитная карта</p>
  <p><input type="submit"></p>
</form>

</body>

```

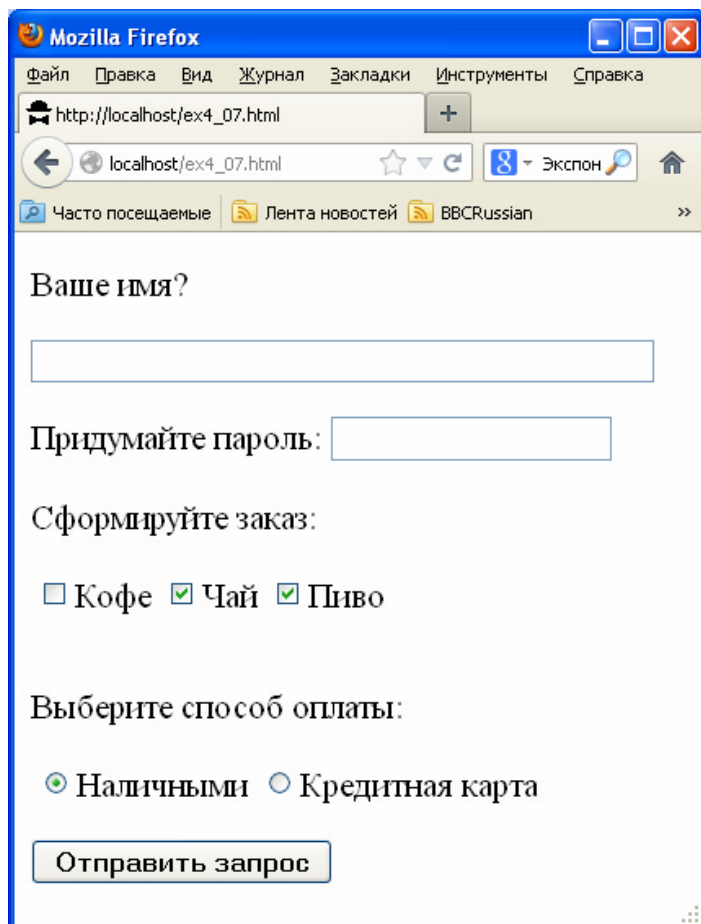


Рисунок 4.9 - Веб-страница с элементами формы

Тег **<form>** .. **</form>** обрамляет те данные, которые будут отправлены на сервер. В атрибуте **action** указывается адрес в Интернете той программы, которая будет обрабатывать эти данные. В данном случае это "primer.php" (рассмотрим его позже). В атрибуте **method** указывается метод отправки (это может быть "POST" или "GET"). В данном примере указан метод POST. Методы POST и GET отличаются тем, что для метода POST данные отправляются в теле HTTP-запроса (их не видно пользователю), а в методе GET данные отправляются в строке URL (их видно в адресной строке в браузере). Рассмотрим подробнее POST и GET позже, когда будем рассматривать текст программы primer.php, которая будет на сервере обрабатывать наши данные.

Внутри тега **<form>** .. **</form>** мы видим несколько тегов **<input>** (элемент формы), отличающихся разным значением атрибута **type**. Именно значение атрибута **type** определяет внешний вид элемента формы и его назначение.

Значение **type="text"** задает текстовую строку ввода. В этом случае атрибут **size="50"** означает, что в строке будет максимум 50 букв. Атрибут **name** понадобится нам при обработке данных на сервере и мы вернемся к нему позже.

Значение **type="password"** также задает строку ввода. Однако буквы, которые пользователь будет вводить, будут отображаться на экране звездочками (символами \*).

Значение **type="checkbox"** задает элементы с "галочками", которые можно устанавливать независимо друг от друга. При этом наличие атрибута **checked** (без значения) указывает браузеру, что "галочка" должна стоять в элементе уже при загрузке веб-страницы.

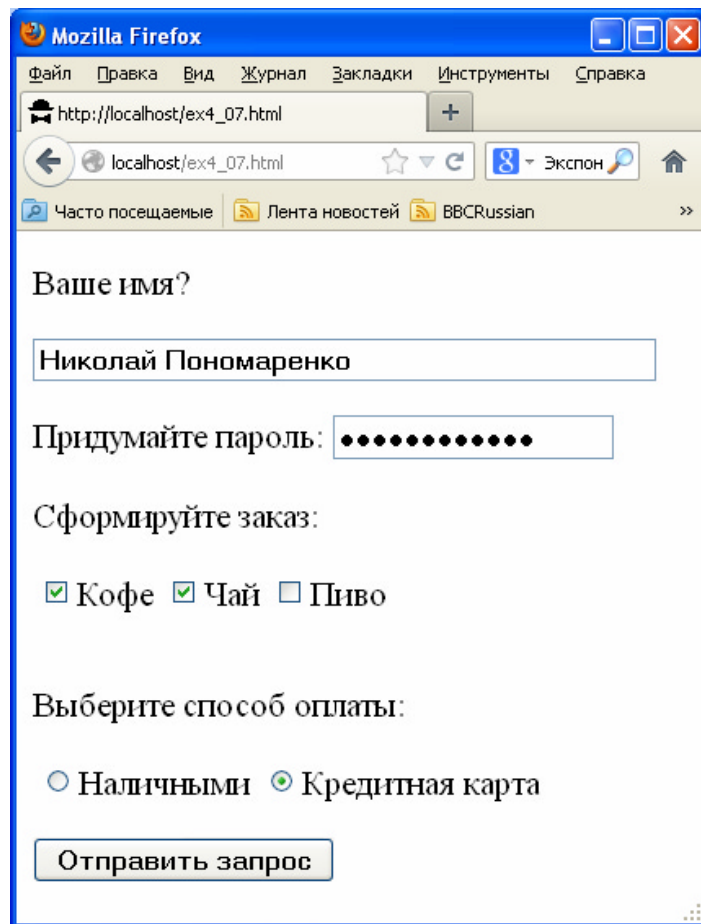


Рисунок 4.10. Заполненные данные формы

И, наконец, **type="submit"** задает кнопку, при нажатии на которую данные формы (значения ее элементов) будут отправлены на сервер.



Значение `type="radio"` задает элемент переключатель (кружочек с точечкой или без). Среди всех элементов типа `radio` с одинаковыми атрибутами **name** только один может быть установлен. Чтобы выбрать, какой из них будет установлен по умолчанию, используется атрибут **checked** (как и для `checkbox`).

На рис. 4.10 приведен пример заполненных данных для формы на рис. 4.9.

Рассмотрим теперь программу `primer.php`, которая будет обрабатывать данные формы (рисунок 4.11, файл `primer.php`).

```
<?php
echo "Здравствуйте, ",
$_POST['nam'], "!\n";
echo "Ваш пароль: ", $_POST['oleg'],
"\n";

if ($_POST['cofe']=='on')
    echo "Извините, кофе нет!\n";

if ($_POST['money']=="var1")
    echo "Оплата наличными\n";

$f=fopen('zakazy.txt','a');
$st=print_r($_POST, true);
fwrite($f,$st);
fclose($f);

echo "Заказ принят! Спасибо!";
?>
```

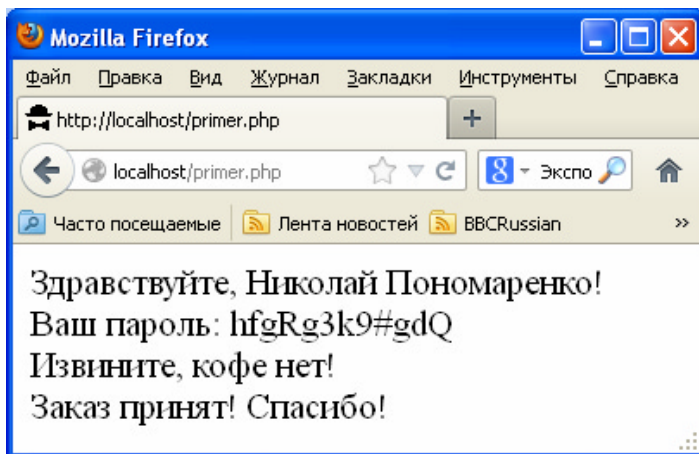


Рисунок 4.11 - PHP-код для обработки запроса и результат работы этого кода

Эта программа проверяет данные формы, анализирует их, выводит сообщения на веб-страницу и сохраняет данные формы в текстовом файле.

PHP - очень удобный язык для обработки данных форм. Все, что можно, делается автоматически и все полученные методом `POST` от веб-страницы данные помещаются в специальный ассоциативный массив с именем `$_POST`. Названия индексов этого массива соответствуют именам тегов `<input>` (атрибут **name**). Чтобы проверить, что передано из веб-страницы в элемент формы с значением тега `name="oleg"` нужно просто обратиться к ячейке

`$_POST['oleg']`. Точно так же и со всеми остальными элементами формы.

В приведенном примере на экран командами **echo** выводятся ФИО и пароль. Затем проверяется, была ли установлена галочка в элементе формы с именем `'cofe'` и, если да, то выводится надпись "Извините, кофе нет!".

Далее проверяется, был ли выбран элемент **radio** со значением `"var1"` (см. рисунок 4.9) и, если да, то выводится надпись "Оплата наличными".

Как видим, все просто.

И в завершение примера все данные, полученные от веб-страницы, сохраняются в текстовом файле `zakazy.txt`. Параметр `'a'` в функции **fopen** (`'zakazy.txt','a'`) означает, что файл открывается для дозаписи (добавления информации к уже записанной ранее в этот файл).

Можно было бы сохранить каждое данное отдельно, но для красоты и краткости текста программы здесь использована функция **print\_r**. Второй параметр этой функции, равный `TRUE`, сообщает ей, что нужно не выводить текст на веб-страницу, а передать его на выход функции (в данном случае в переменную `$st`). На рисунке 4.12 приведено содержимое файла `zakazy.txt`, соответствующие случаю на рисунке 4.11.

```
Array
(
    [nam] => Николай Пономаренко
    [oleg] => hfgRg3k9#gdQ
    [cofe] => on
    [tea] => on
    [money] => var2
)
```

Рисунок 4.12 - Содержимое файла `zakazy.txt`

Хорошо видно, что для текстовых элементов формы и паролей на сервер передаются текстовые строки, для элементов типа `"checkbox"` - текстовая строка `"on"` (для случая, если галочка установлена), для элементов типа `"radio"` - значение того элемента, который был выбран.

В заключение по этому примеру отметим, что, если данные с веб-страницы по какой-то причине не были переданы, то массив `$_POST` окажется пустым. Проверить, не пустая ли та или иная ячейка массива `$_POST` можно с помощью функции **isset** (`$s`), которая возвращает `TRUE`, если переменная `$s` существует. Чтобы проверить, например, было ли передано на сервер данное с именем `"oleg"`, нужно проверить `isset($_POST['oleg'])`.

Итак, мы рассмотрели пример отправки данных на сервер методом `POST` и их обработки там. Рассмотрим теперь простой пример отправки данных на сервер методом `GET`.

Чтобы отправить данные на сервер методом `GET`, можно точно так же использовать элементы формы, только в теге `<form>` присвоить атрибуту **method** значение `"GET"`.

На рисунке 4.13 приведен пример простой веб-страницы, использующей метод `GET` (файл `ex4_08.html`), и ее внешний вид в окне браузера после ввода данных. Пока все выглядит точно так же, как и в предыдущем примере. Существенное отличие будет лишь в PHP-коде и том, как выглядит в браузере ответ сервера.



```
<html>
<body>

<form action="pr.php" method="GET">
  <p>Ваше имя? <input type="text"
name="nam" size="15"></p>
  <p>Год рождения? <input
type="text" name="year"
size="7"></p>
  <p><input type="submit"></p>
</form>

</body>
```

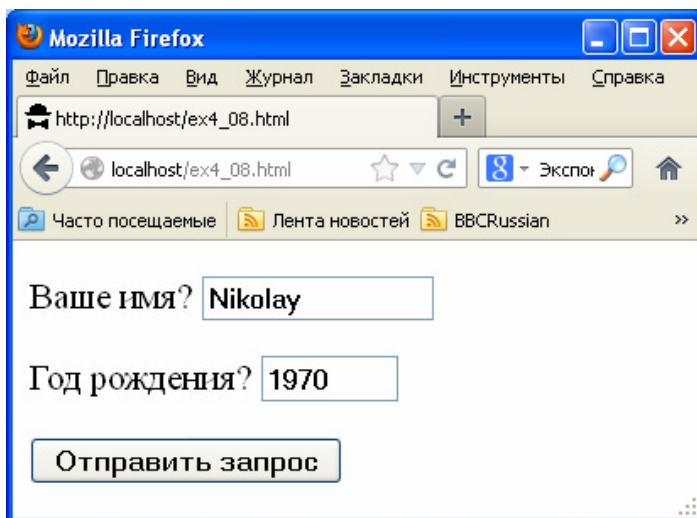


Рисунок 4.13 - Веб-страница с использованием метода GET

На рисунке 4.14 приведен текст программы pr.php, которая обрабатывает запрос, и окно браузера с ответом.

```
<?php
echo $_GET['nam'], " г.р.",
$_GET['year'];
?>
```

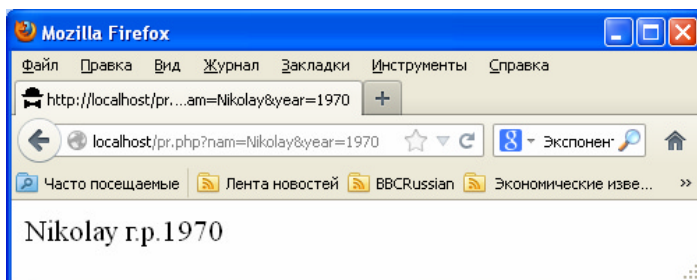


Рисунок 4.14 - Веб-страница с использованием метода GET

Казалось бы, никакого отличия, кроме того, что данные веб-страницы для метода **GET** нужно читать из ассоциативного массива `$_GET`. Однако обратите внимание на адресную строку в браузере. Там значится "localhost/pr.php?nam=Nikolay&year=1970". Все, что находится после знака "?" и есть данные, переданные от веб-страницы серверу методом **GET**.

Данные для метода **GET** передаются на сервер прямо в строке URL и разделяются символом "&". При этом русские буквы могут кодироваться в кодировке **UTF8** и быть

нечитаемыми, однако PHP делает извлечение этих данных легкой задачей благодаря массиву `$_GET`.

Недостатком метода **GET** по сравнению с методом **POST** является меньший объем данных, которые можно передать на сервер. Однако, иногда метод **GET** является более удобным, так как позволяет формировать запросы на сервер даже без использования элементов формы.

На рисунке 4.15 приведен пример такой веб-страницы, которая отправляет на сервер данные методом GET (файл ex4\_09.html) без использования форм. В данном примере используются просто ссылки с интегрированными в них запросами к серверу.

```
<html>
<body>

<a href="prim.php?ponom=1">Первая
ссылка</a><BR>
<a href="prim.php?ponom=2">Вторая
ссылка</a>

</body>
```

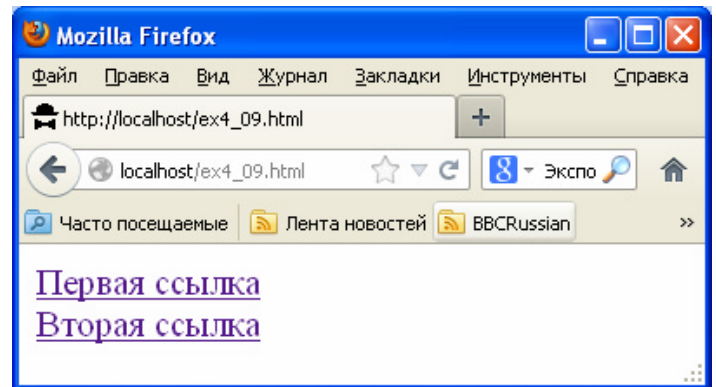


Рисунок 4.15 - Веб-страница с GET запросами, оформленными в виде ссылок

Данные здесь помещены в значения атрибутов **href** тега `<a> .. </a>` через символ "?" после пути к файлу PHP-программы.

На рисунке 4.16 приведена программа prim.php и результат ее работы.

```
<?php
$a=$_GET['ponom'];
if ($a=="1")
echo "Нажата первая ссылка";
else echo "Нажата вторая ссылка";
?>
```

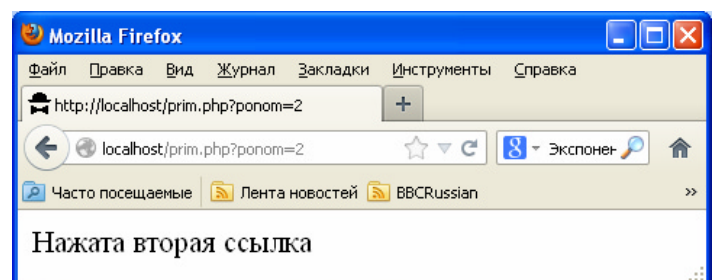


Рисунок 4.15 - Веб-страница с GET запросами, оформленными в виде ссылок

В данном примере обе ссылки ведут к одному и тому же PHP-коду (иногда еще говорят “PHP-скрипт”). Но результат работы этого PHP-кода будет разным в зависимости от того, какая из двух ссылок нажата.

Такого же эффекта можно добиться и при отправке данных методом **POST**, но придется задействовать JavaScript и текст веб-страницы будет более сложным.

Кроме массивов `$_POST` и `$_GET`, содержащих данные, переданные с веб-страницы, в PHP есть еще ассоциативный массив `$_SERVER`, содержащий данные о компьютере пользователя (например, IP-адрес). Изучите содержание этого массива самостоятельно с помощью справочников и функции **print\_r()**.

В заключении вернемся к примеру на рисунке 4.11 и обратим внимание на следующий момент. А что будет, если два пользователя почти одновременно отправят данные на сервер? Запустится два экземпляра этого PHP-кода и они могут одновременно пытаться что-то записать в файл 'zakazy.txt', что может привести к сбою и потере данных. Для решения этой проблемы в PHP имеется функция **flock()**, на логическом уровне запирающая файл. Если кто-то уже запер этот файл, функция **flock()** ждет, пока файл освободится и только тогда запирает его.

Можно завести вспомогательный служебный файл, (например, назовем его 'flag.txt') и использовать как семафор. Перед записью в файл 'zakazy.txt' нужно запереть файл 'flag.txt' вот так:

```
$q=fopen('flag.txt','r');
flock($q, LOCK_EX);
```

После же того, как запись в файл 'zakazy.txt' завершена и он закрыт, нужно отпереть файл 'flag.txt' вот так:

```
flock($q, LOCK_UN);
fclose($q);
```

Почему нельзя запереть и потом отпереть непосредственно файл 'zakazy.txt'? Попробуйте сами ответить на данный вопрос, рассуждая логически. Затем проверьте себя, прочитав ответ в следующем абзаце.

Дело в том, что чтобы выполнить функцию **flock()**, нужно сначала открыть файл. Если две PHP-программы открывают файл для чтения (параметр 'r'), то ничего страшного не происходит - это допустимо. Затем одна из программ выполняет **flock()**, а вторая ждет, пока файл освободится. Если же две программы попытаются одновременно открыть файл для записи или дозаписи (а именно для этого открывается файл 'zakazy.txt'), то может произойти сбой и до вызова функции **flock()** просто не дойдет дело. Именно поэтому в данном примере в качестве семафора используется еще один файл ('flag.txt'), который открывается для чтения. И, видя, что он закрыт, вторая программа знает, что кто-то в это время пишет в файл 'zakazy.txt'. Когда же файл 'flag.txt' откроется, вторая программа закрывает его и начинает запись в файл 'zakazy.txt'.

На этом данная лекция заканчивается. Формирование запросов из PHP к другим серверам, работу с cookies, с графикой, с регулярными выражениями и некоторые другие тонкости мы рассмотрим в лекции №7.

## 5. PHP и MySQL

MySQL - свободно распространяемая система управления базами данных. Обычно это сервер, к которому обращаются с запросами (командами) удаленные и локальные клиенты.

MySQL позволяет облегчить работу с базами данных для программиста (доступ к базе данных одновременно многих клиентских приложений, удобный поиск в базе данных, составление сводок и т.д.). Однако за удобство приходится платить меньшей надежностью - сайт, использующий MySQL легче взломать (или нарушить его работу), чем сайт, в котором данные сохраняются в обычных файлах.

MySQL реализован для большинства известных операционных систем. Для MySQL существует **API** (наборы функций для работы с MySQL) для большинства известных языков программирования. В этом разделе мы научимся использовать MySQL из PHP.

### 5.1. Установка MySQL и создание базы данных

Чтобы пользоваться MySQL на локальном компьютере, нужно установить на него Denver (установка Denver была рассмотрена в лекции о PHP). Настраивать MySQL не нужно, сразу после установки Denver-а он готов к работе.

Вместе с Denver-ом кроме MySQL автоматически устанавливается еще и программа phpMyAdmin, которая позволяет удобно администрировать MySQL сервер. Чтобы запустить phpMyAdmin, нужно запустить Denver, а затем в командной строке браузера набрать:

<http://localhost/Tools/phpMyAdmin>

Появится окно управления сервером MySQL (рисунок 5.1).

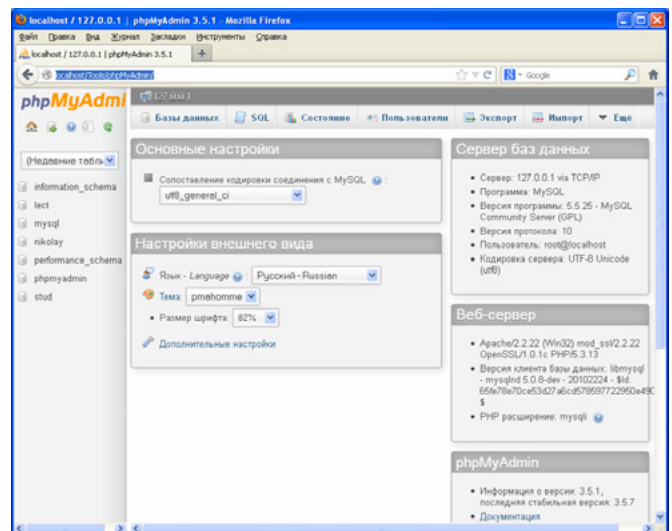


Рисунок 5.1 - Окно программы phpMyAdmin в браузере

Слева на рисунке можно видеть столбец с перечнем баз данных, созданных на этот момент, сверху - строку меню. Постепенно, по мере необходимости, вы разберетесь с этим меню самостоятельно. Для начала же нам потребуется лишь создать себе базу данных, а также придумать для себя логин и пароль для доступа к этой базе.

Чтобы создать базу данных, нажимаем на пункт меню “Базы данных”, в соответствующую строку вбиваем имя базы данных (используйте латинские буквы и цифры) и нажимаем на кнопку “Создать”. Например, создадим базу данных “lection5”. После этого она появляется в меню баз данных (см. рисунок 5.2).

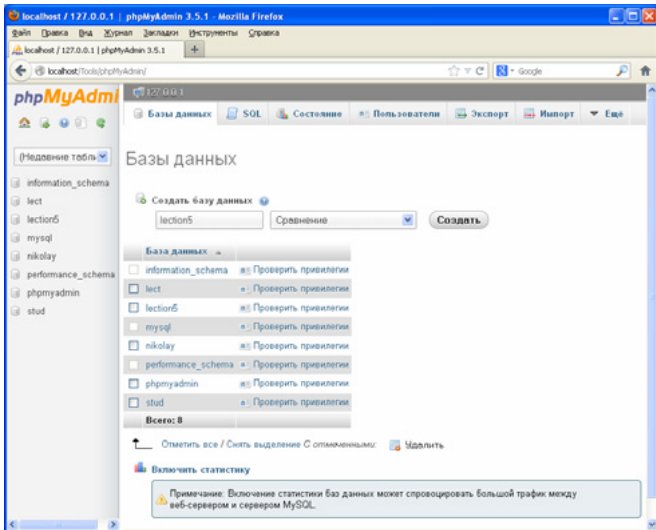


Рисунок 5.2 - Результат создания новой базы данных

Следующий шаг - придумать логин и пароль для пользователя, которому сервер MySQL будет разрешать работать с этой базой данных.

Нажмем на название нашей базы данных "lection5" в списке баз данных и в открывшемся окне на пункт меню "Привилегии". В открывшемся окне (рисунок 5.3) нажмем на "Добавить пользователя".

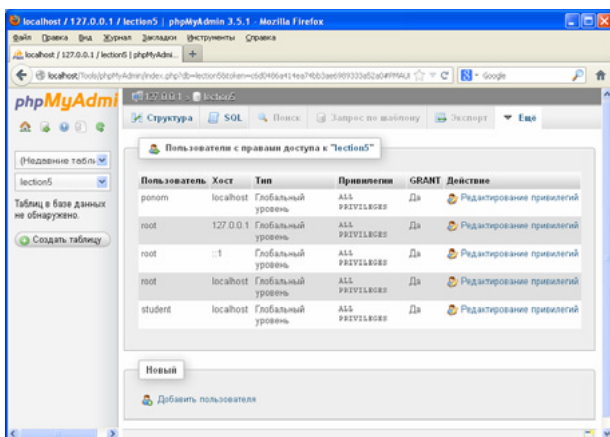


Рисунок 5.3 - Окно редактирования привилегий для базы lection5

В открывшемся окне (рисунок 5.4) вписываем придуманное нами имя пользователя (vladimir), придуманный нами пароль (vladimir2013), в качестве хоста указываем 'Локальный' и выбираем 'Выставить полные привилегии на базу данных "lection5"'.  

Пользователь	Хост	Тип	Привилегии	GRANT	Действие
root	localhost	Глобальный уровень	ALL PRIVILEGES	Да	Редктирование привилегий
root	127.0.0.1	Глобальный уровень	ALL PRIVILEGES	Да	Редктирование привилегий
root	localhost	Глобальный уровень	ALL PRIVILEGES	Да	Редктирование привилегий
student	localhost	Глобальный уровень	ALL PRIVILEGES	Да	Редктирование привилегий

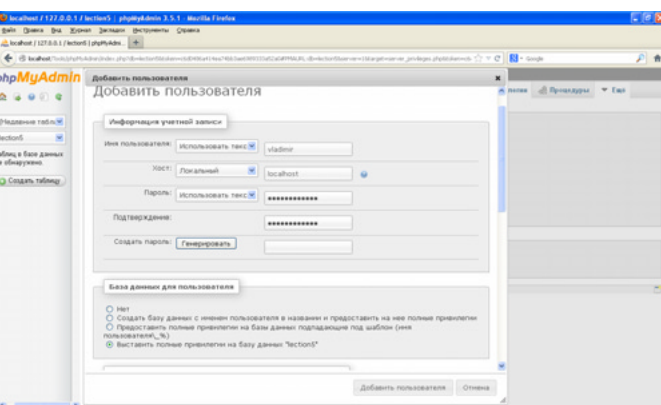


Рисунок 5.4 - Создание нового пользователя

После нажатия на "Добавить пользователя" пользователь будет создан и теперь можно приступать к программированию на PHP и работе с этой базой данных.

В этом параграфе мы научились создавать на локальном сервере базу данных MySQL и пользователя для нее. Если нужно создать базу данных MySQL не на локальном сервере, а на Интернет-хостинге, то там все еще проще. Как правило, нам достаточно вбить имя базы данных, имя пользователя и пароль в соответствующие строки ввода (нужно предварительно почитать инструкции на конкретном хостинге) и все будет создано автоматически.

## 5.2. Базовые действия PHP-программы при работе с MySQL

Любая наша программа на языке PHP, которая собирается работать с MySQL базой данных, должна состоять из следующих обязательных базовых шагов:

1. Установить связь с MySQL сервером.
2. Выбрать базу данных для работы.
3. Посылать команды MySQL серверу и получать ответы.
4. Закрыть связь с MySQL сервером.

Начнем с первого шага. На рисунке 5.5 приведена программа (файл ex5\_01.php), которая устанавливает связь с нашим локальным сервером MySQL и больше ничего не делает.

```
<?php

$link = mysql_connect("localhost",
"vladimir", "vladimir2013")
or die("Не получилось: ".mysql_error());

echo "Соединение установлено";

mysql_close($link);

?>
```

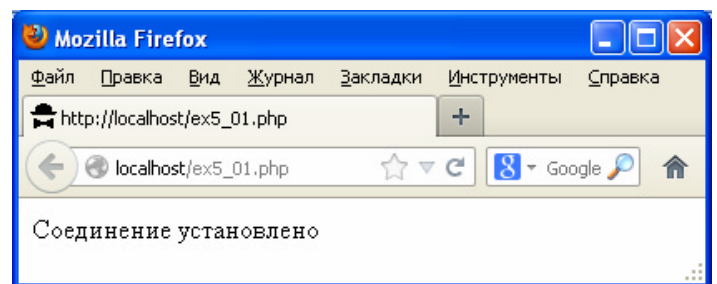


Рисунок 5.5 - Веб-страница, которая устанавливает и закрывает соединение с MySQL сервером

Эта программа выполняет сразу два обязательных пункта - 1-й и 4-й.

Функция `mysql_connect` устанавливает связь с сервером. Первый параметр функции - имя сервера (в нашем случае - "localhost"). Второй параметр - логин пользователя ("vladimir"). Третий параметр - пароль пользователя ("vladimir2013").

Если соединение не установлено (например, сервер не разрешает этому пользователю доступ к базам данных), то



PHP выполнит функцию `die()` (завершение программы).

Если соединение успешно установлено, то в служебной переменной `$link` (имя этой переменной придумано нами) возвращаются параметры соединения с сервером, которые в дальнейшем нам потребуются.

После этого программа выдает на веб-страницу надпись “Соединение установлено” и закрывает соединение с сервером командой `mysql_close($link)`;

### 5.3. Создание таблицы MySQL

Любая база данных MySQL состоит из таблиц и все данные в MySQL сохраняются в виде таблиц. Поэтому, если мы хотим писать что-то в базу данных MySQL, наша программа должна сначала создать таблицу. На рисунке 5.6 приведен пример такой программы (файл `ex5_02.php`).

```
<?php

$link = mysql_connect("localhost",
"vladimir", "vladimir2013") or
die(mysql_error());

mysql_select_db('lection5', $link) or
die (mysql_error());

$s="CREATE TABLE studata (god int,
grup char(7), fio char(50));";
mysql_query($s) or
die(mysql_error());

echo "Сделано.";

mysql_close($link);
?>
```

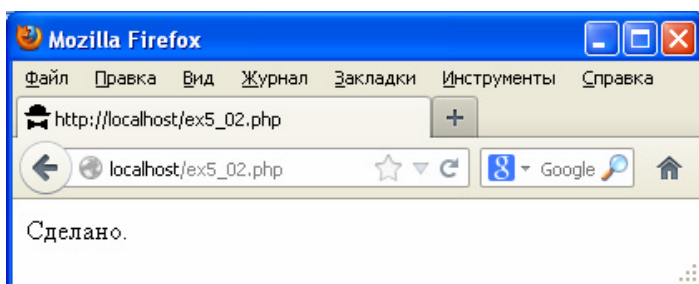


Рисунок 5.6 - Веб-страница, которая таблицу в базе MySQL

Что нового в этой программе? Команды `mysql_connect()` и `mysql_close()` уже были в прошлом примере. Новыми для нас командами здесь являются `mysql_select_db` (выбор базы данных) и `mysql_query` (запрос к серверу). Поясим их назначение.

Команда `mysql_connect()` соединяет нас (пользователя) с сервером. Однако на сервере может лежать несколько баз данных и пользователю может быть разрешен доступ к нескольким из них. Поэтому необходимо уточнить, с какой же базой данных мы хотим работать, что и делает команда `mysql_select_db()`. Первым параметром выступает название базы данных (“lection5”). Вторым параметром нужно указать переменную, которую мы получили в результате выполнения `mysql_connect()` - `$link`.

Как только мы выбрали базу данных, можно создавать в ней таблицу. Заметим, что все действия над базой данных (создание таблиц, занесение в них данных, изменение данных, поиск данных, удаление данных и т.д.) выполняются в виде запросов (текстовых команд) к MySQL серверу. Сначала формируется текстовая переменная, в которую помещается текст запроса. Затем содержимое этой текстовой переменной отправляется на сервер командой `mysql_query()`.

Запросом в данном случае является строка:

```
CREATE TABLE studata (god int, grup char(7), fio
char(50));
```

Здесь “CREATE TABLE” - ключевые слова, по которым сервер поймет, что нужно создать таблицу. “studata” - название для таблицы. Далее в круглых скобках через запятую перечисляются столбцы, которые должны быть у таблицы. Для каждого столбца обязательно через пробел указывается его название и тип. В данном случае мы указываем серверу, что в таблице должно быть 3 столбца с названиями “god”, “grup” и “fio”. Столбец “god” будет содержать данные типа “int”, то есть целые числа. Столбец “grup” будет содержать строки с максимально возможной длиной в 7 букв каждая. Столбец “fio” будет содержать строки с максимально возможной длиной в 50 букв каждая.

Точка с запятой в конце строки запроса - обязательный элемент синтаксиса.

В создаваемой таблице может быть сколько угодно столбцов, но в данном примере их лишь три. Кроме двух обязательных параметров (имя и тип) у каждого столбца могут быть и необязательные, но для экономии времени мы их здесь рассматривать не будем. Они нам пока не понадобятся. По мере необходимости изучите их самостоятельно, в частности, рассмотрите с помощью какого-либо справочника (например, <http://www.php.ru/mysql/>), назначение параметра “NOT NULL”.

Зайдем снова в phpMyAdmin и посмотрим, создалась ли заказанная нами таблица? Да, создалась (рисунок 5.7).

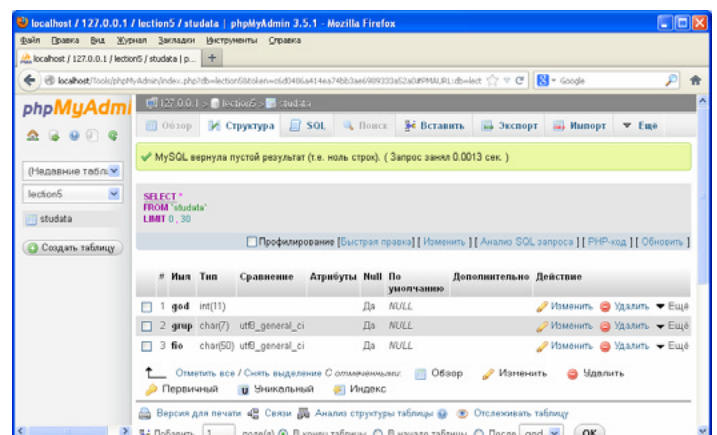


Рисунок 5.7 - Результат добавления в базу “lection5” таблицы “studata”

Эта таблица пока пуста.

### 5.4. Добавление строки в таблицу

Строки в таблицу добавляются такими же запросами-командами к серверу с помощью функции `mysql_query()`, только с другим текстом запроса.



На рисунке 5.8 приведена программа (файл ex5\_03.php), которая добавляет в таблицу “studata” строку, в которой god=1996, grup=“519-Б”, а fio=“Иванов Иван Петрович”.

```
<?php

$link = mysql_connect("localhost",
"vladimir", "vladimir2013") or
die(mysql_error());

mysql_select_db('lection5', $link) or
die (mysql_error());

$s="INSERT INTO studata VALUES (1996,
'519-Б', 'Иванов Иван Петрович');";
mysql_query($s) or
die(mysql_error());

echo "Строка в таблицу добавлена.";

mysql_close($link);

?>
```

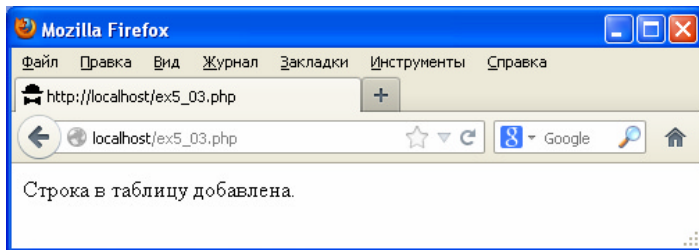


Рисунок 5.8 - Программа, которая добавляет строку в таблицу MySQL

Здесь запросом является строка:

```
INSERT INTO studata VALUES (1996, '519-Б',
'Иванов Иван Петрович');
```

“INSERT INTO” - ключевые слова, которые говорят серверу, что нужно добавить строку в таблицу. “studata” - название таблицы, в которую нужно добавлять данные. После ключевого слова “VALUES” в круглых скобках через запятую перечисляются значения столбцов. Заканчивается запрос точкой с запятой.

Как видим, все несложно. Установили связь с сервером. Выбрали базу данных. Послали команду серверу. Закрыли сеанс связи. Если мы уверены, что ошибки проверять не нужно, то текст программы можно упростить, например, до такого, который приведен на рисунке 5.9 (файл ex5\_04.php).

```
<?php
$link=mysql_connect("localhost",
"vladimir", "vladimir2013");
mysql_select_db('lection5', $link);
$s="INSERT INTO studata VALUES
(1995,'519','Нос Олег');";
mysql_query($s);
mysql_close($link);
?>
```

Рисунок 5.9 - Очень короткая программа, которая добавляет строку в таблицу MySQL

Если после выполнения этих двух программ зайти в phpMyAdmin и посмотреть там базу studata, то можно увидеть добавленные строки (рисунок 5.9).

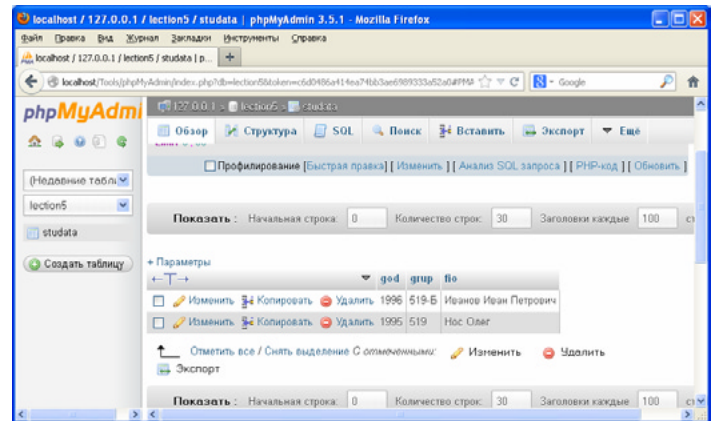


Рисунок 5.9 - База данных MySQL после добавления строк

Вот мы и научились создавать таблицу и добавлять туда данные из программы. Теперь мы уже можем создать веб-страницу, где пользователь будет вводить данные, которые затем передадутся в php-программу на сервере, которая добавит эти данные в базу MySQL.

На рисунке 5.10 приведена веб-страница, на которой пользователь вводит группу, фамилию и год рождения студента (файл ex5\_05.html).

```
<html>
<body>

<form action="ex5_06.php" method="POST">
  <p>Номер группы?</p>
  <p><input type="text" name="gru"
size="10"></p>
  <p>ФИО: <input type="text" name="fam"
size="50"></p>
  <p>Год рождения: <input type="text"
name="fam" size="50"></p>
  <p><input type="submit"></p>
</form>

</body>
```

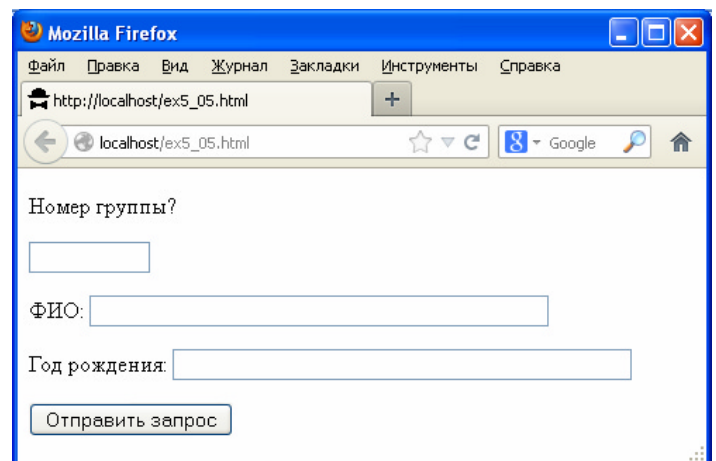


Рисунок 5.10 - Веб-страница для ввода данных

Эта веб-страница при нажатии на кнопку методом POST отправляет данные на сервер программе “ex5\_06.php”.

Эта программа и результат ее работы, в свою очередь, приведены на рисунке 5.11.

```
<?php
    $f=$_POST['fam'];
    $gr=$_POST['gru'];
    $go=$_POST['gd'];
    $link=mysql_connect("localhost",
"vladimir", "vladimir2013");
    mysql_select_db('lection5', $link);
    $s="INSERT INTO studata VALUES
('.$go.', ' ".$gr.', ' '.$f.'');";
    mysql_query($s);
    mysql_close($link);
    echo "Запись добавлена. <a
href=\"ex5_05.html\">Сделать еще одну
запись</a>";
?>
```

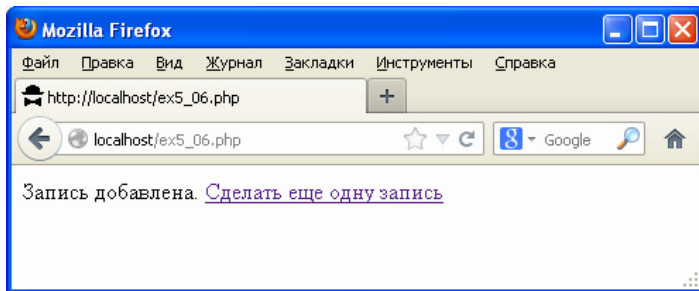


Рисунок 5.11 - Веб-страница для ввода данных

Данная программа получает пришедшие от веб-страницы данные из стандартного массива \$\_POST и затем формирует из них запрос для MySQL сервера на вставку в таблицу новой строки.

Обращаем внимание, что логин и пароль пользователя для доступа к MySQL серверу и базе данных находится в PHP-программе "ex5\_06.php", которая лежит на сервере. Пользователь может видеть только результат выполнения этой программы (результат команд echo), но не ее исходный текст. Поэтому пароль и логин находятся в полной безопасности, хотя и используются в программе.

### 5.5. Поиск и извлечение данных из таблицы

Проще всего извлечь из базы данных всю таблицу целиком (см. рисунок 5.12, файл ex5\_07.php).

В данном примере запросом к серверу является строка

```
SELECT * FROM studata;
```

Здесь ключевое слово "SELECT" сообщает серверу, что необходимо искать данные в таблице, а служебный символ "\*" означает, что искать нужно во всех столбцах.

Результат запроса поступает в переменную \$r. Если ничего не найдено (например, такой таблицы нет или она пуста), то в \$r будет записан ноль. Иначе же там будет двумерный массив, содержащий найденные строки таблицы.

Для работы с результатами поиска данных в таблице в PHP есть несколько похожих функций. Пока вполне достаточно рассмотреть лишь одну из них - \$q=mysql\_fetch\_row(\$r). Она последовательно извлекает из результатов поиска \$r по одной строке и заносит их в переменную \$q.

```
<?php
    $link=mysql_connect("localhost",
"vladimir", "vladimir2013");
    mysql_select_db('lection5', $link);
    $s="SELECT * FROM studata;";
    $r=mysql_query($s);
    mysql_close($link);
    while ($q=mysql_fetch_row($r))
        echo $q[0], " ", $q[1], "
", $q[2], "<br>";
?>
```

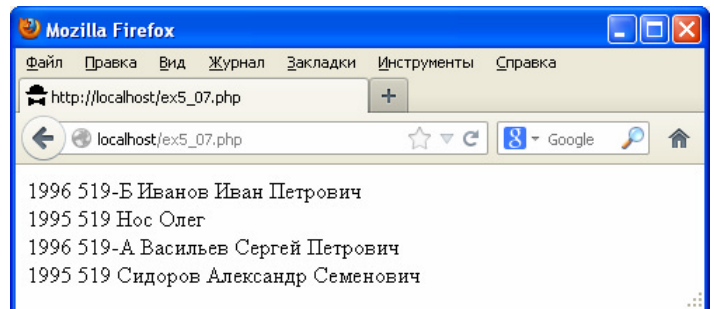


Рисунок 5.12- Веб-страница выводит содержимое таблицы

Когда строки закончатся, в \$q будет занесен ноль и цикл while в примере остановится.

\$q представляет собой массив, чьи индексы являются номерами, начинающимися с нуля. Поэтому, чтобы обратиться к элементу из первого столбца таблицы, нужно написать \$q[0] (год рождения) и т.д.

Чуть более сложный вариант, чем извлечь всю базу целиком, это поиск с условиями. Чтобы программа на рисунке 5.12 выводила не всю таблицу, а лишь тех, кто родился в 1996-м году, нужно в запрос добавить ключевое слово "WHERE" и условие поиска, вот так:

```
SELECT * FROM studata WHERE god=1996;
```

Тогда результат работы программы (ex5\_08.php) будет выглядеть, как это показано на рисунке 5.13.

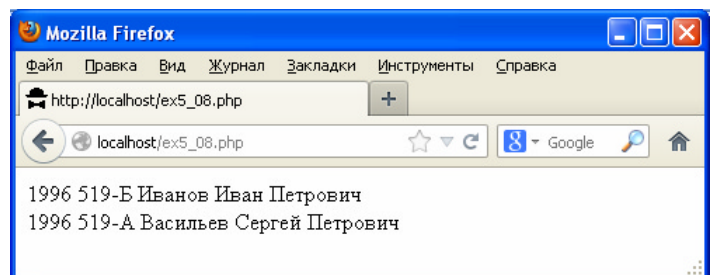


Рисунок 5.13 - Результаты поиска по условию

Вместе с ключевым словом "WHERE" можно использовать, например, модификатор "LIKE" (рисунок 5.14, файл ext\_09.php).

Здесь строка запроса:

```
SELECT * FROM studata WHERE fio LIKE '%Сергей%';
```

Модификатор "LIKE" задает шаблон для столбца "fio". Символ "%" в начале и конце шаблона означает, что перед именем "Сергей" может быть что угодно и после имени "Сергей" может быть, что угодно. Если бы мы хотели, чтобы

имя Сергей обязательно завершало текст, то шаблон выглядел бы так: “%Сергей”.

```
<?php
$link=mysql_connect("localhost",
"vladimir", "vladimir2013");
mysql_select_db('lection5', $link);

$s="SELECT * FROM studata WHERE fio
LIKE '%Сергей%';";

$r=mysql_query($s);
mysql_close($link);
while ($q=mysql_fetch_row($r))
    echo $q[0], " ", $q[1], "
", $q[2], "<br>";
?>
```

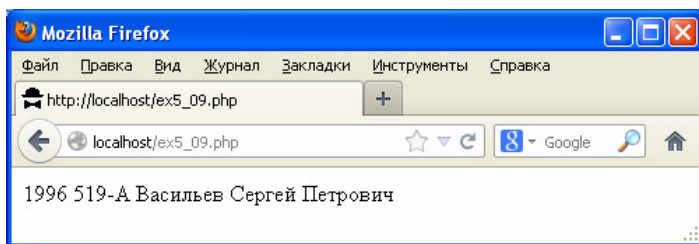


Рисунок 5.14 - Поиск по шаблону

Условия при поиске могут быть сколь угодно сложными (можно использовать логические связки “or”, “and”). Можно извлекать из таблицы не все столбцы, а лишь указанные нами. Как это делать - Вы постепенно изучите по мере практической необходимости. Сейчас же главное для нас - изучить принцип поиска и извлечения данных из баз MySQL, что мы и сделали.

## 5.6. Изменение и удаление данных в таблице

На рисунке 5.15 приведен текст программы, которая изменяет данные в таблице (файл ex5\_10.php).

```
<?php
$link=mysql_connect("localhost",
"vladimir", "vladimir2013");
mysql_select_db('lection5', $link);

$s="UPDATE studata SET god=1994,
grup='529' WHERE god<1996;";

$r=mysql_query($s);
mysql_close($link);
?>
```

Рисунок 5.15 - Программа изменяет данные в таблице

Здесь запросом является:

```
UPDATE studata SET god=1994, grup='529'
WHERE god<1996;
```

Ключевое слово “UPDATE” говорит серверу, что нужно изменить данные, после ключевого слова “SET” через запятую перечисляются названия столбцов, которым нужно изменить данные и их новые значения. После ключевого

слова “WHERE” следует условие. Данные будут изменены для **всех** строк таблицы “studata”, для которых это условие выполняется.

Если после запуска программы ex5\_10.php запустить программу ex5\_07.php (которая визуализирует всю таблицу), то мы увидим, что поменялись сразу две строки таблицы (см. рисунок 5.16)

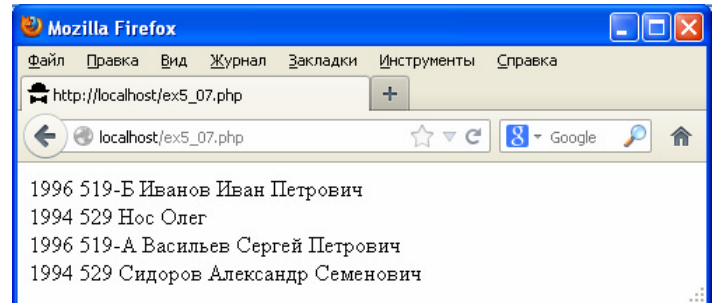


Рисунок 5.16 - Результат изменения таблицы

Если неосторожно и криво написать запрос, то можно поменять сразу все строки таблицы.

Чтобы поменять только одну строку, нужно так составить условие, чтобы ему соответствовала только эта строка.

Удаление данных происходит по запросам, похожим на запросы поиска, только ключевым словом является “DELETE”. Например, следующий запрос удалит из таблицы строку, где столбец “fio” содержит строку “Нос Олег”:

```
DELETE FROM studata WHERE fio='Нос Олег';
```

Будьте осторожны. Если криво задать условие WHERE, то можно одной командой удалить из таблицы все строки.

## 5.7. Кодировки русского текста при работе с MySQL

Если Вы написали программу для работы с базой MySQL и видите, что в базу вместо русских букв заносятся пустые строки или сервер в ответ на запросы возвращает кракозябры, то, скорее всего ваша веб-страница и php-программа работают в одной кодировке, а сервер MySQL - в другой.

Мы можем указать серверу в какой кодировке будут присылаться данные, в какой кодировке их хранить, в какой кодировке возвращать ответы сервера. Изучите этот вопрос самостоятельно, а здесь покажем лишь наиболее простой вариант решения проблемы.

Пусть мы хотим, чтобы сервер принимал, хранил и возвращал данные в кодировке “Windows-1251”. Тогда при создании таблицы вместо, например, запроса

```
CREATE TABLE stuff (fi char(10), tel char(10));
```

пишите:

```
CREATE TABLE stuff (fi char(10), tel char(10))
DEFAULT CHARACTER SET cp1251 COLLATE
cp1251_general_ci;
```

Это заставит сервер сохранять и возвращать данные в кодировке “Windows-1251”.

Еще в тексте программы, которая будет посылать запросы к серверу, нужно после выбора базы данных командой mysql\_select\_db() добавить запрос:

```
mysql_query("SET NAMES 'cp1251'");
```

После этого мы можем быть уверены, что вся работа с базой данных MySQL осуществляется с использованием кодировки “Windows-1251”.

### 5.8. Проверка существования таблицы

Как узнать, существует ли таблица, например, с именем “olga” в базе данных MySQL? Например, можно отправить запрос на сервер командой `mysql_query()`:

```
SHOW TABLES LIKE "olga";
```

Здесь “SHOW TABLES” говорит серверу, что нужно вернуть названия таблиц, а после “LIKE” нужно указать имя таблицы. Если таблица не существует, то сервер вернет в ответ ноль.

### 5.9. Первичный и внешний ключи и некоторые другие тонкости

При создании таблиц кроме имени и типа столбцов для них можно указывать еще и модификаторы:

NOT NULL - поле не может быть пустым;  
PRIMARY KEY - поле будет первичным ключом (иметь уникальное значение);  
AUTO\_INCREMENT - при вставке новой записи значение этого поля будет автоматически увеличено на единицу;  
DEFAULT - задает значение, которое будет использовано по умолчанию.

Эти идентификаторы на практике понадобятся нам еще не скоро, но необходимо понимать, для чего они нужны.

Если мы скажем MySQL, что столбец с именем “fio” является первичным ключом, то это приведет к тому, что сервер не будет нам позволять записывать две полностью одинаковые фамилии в две разные строки таблицы. Это может быть полезно, ведь в этом случае при удалении и модификации строк с указанием в WHERE конкретной фамилии, мы можем быть уверены, что изменим лишь одну строку таблицы.

В MySQL предусмотрены еще внешние ключи для связи таблиц между собой. Если столбец назначен внешним ключом (FOREIGN KEY), то указывается связанный с ним столбец другой таблицы, и сервер не будет разрешать добавлять в таблицу данные со значениями, которых нет в том связанном столбце. Можете разобраться в этом вопросе самостоятельно, хотя на практике это Вам не скоро понадобится, если понадобится вообще.

### 5.10. Заключение

Подведем итог. Основными этапами работы любой PHP программы, которая обращается к серверу MySQL, являются:

- 1) Связаться с сервером:

```
$link = mysql_connect("localhost",  
"vladimir", "vladimir2013");
```

- 2) Выбрать конкретную базу данных:

```
mysql_select_db('lection5', $link);
```

- 3) Посылать запросы к базе данных

```
$r=mysql_query($s);
```

- 4) Основные виды запросов:

Добавить строку в таблицу:

```
INSERT INTO studata VALUES (5,'Петров');
```

Узнать количество строк в таблице:

```
SELECT count(*) FROM studata;
```

проверить существование таблицы:

```
SHOW TABLES LIKE "studata";
```

Извлечь данные из таблицы:

```
SELECT * FROM studata ORDER by fio;
```

Извлечь данные по условию:

```
SELECT * FROM studata WHERE gr = '539';
```

Изменить ячейку:

```
UPDATE studata SET fam="Петров" WHERE  
id=50071;
```

Удалить ячейку:

```
DELETE FROM studata WHERE god<1900;
```

- 5) Обработать результаты запроса:

```
while ($q=mysql_fetch_row($r)) { текст  
программы }
```

- 6) Завершить работу с сервером

```
mysql_close($link);
```

На этом лекция по MySQL заканчивается. Дальше будете изучать тонкости MySQL самостоятельно по мере практической необходимости. Не старайтесь сразу прочитать в справочниках и запомнить наизусть все возможные сведения о MySQL - в этом нет никакого смысла. Если понадобится ответ на какой-нибудь вопрос по синтаксису запросов или еще чему-то, всегда можно быстро найти его в google. Те же вещи, которые Вы часто используете, запомнятся сами собой.



## 6. Библиотека JQuery

Библиотека JQuery создана с единственной целью - облегчить и упростить программирование на языке JavaScript. Поэтому ничего сложного и заумного в этой лекции не будет - иначе какое же это получилось бы упрощение программирования?

Библиотека JQuery написана на языке JavaScript - это достаточно большой файл с расширением ".js", который подключается к веб-странице, как кусок скрипта, и который загружается в браузер вместе с веб-страницей и выполняется браузером. При желании можно посмотреть и попытаться понять текст исходного кода библиотеки JQuery. Но делать это совсем не обязательно.

Кроме того, что текст программы на JQuery выглядит короче и проще, чем на JavaScript, JQuery еще и обеспечивает нашей программе совместимость со всеми браузерами. Разные браузеры могут по-разному реагировать на один и тот же текст программы на JavaScript, однако JQuery специально разрабатывался так, чтобы одинаково выполняться во всех браузерах. Он обеспечивает нам кросс-браузерность.

JQuery - одна из наиболее известных, но не единственная из существующих библиотек для JavaScript. Например, очень популярной является библиотека MooTools (полный список широко используемых библиотек можно найти на: <https://developers.google.com/speed/libraries/devguide>).

### 6.1. Подключение библиотеки JQuery

Чтобы включить JQuery в свою веб-страницу, достаточно скачать последнюю версию библиотеки с сайта jquery.com (например, это оказался файл "jquery-1.9.1.js"), положить в ту же папку, где лежит текст веб-страницы, а в текст веб-страницы вставить:

```
<script src="jquery-1.9.1.js"></script>
```

Файл "jquery-1.9.1.js" при этом имеет объем более 200 килобайт, что может замедлять загрузку нашей веб-страницы в браузере пользователя.

Если ваша веб-страница - маленькая и для вас важно, чтобы все «летало» и загрузка библиотеки JQuery ничего не замедляла, то существует альтернативный метод загрузки JQuery - с сайта Google:

```
<script
src="//ajax.googleapis.com/ajax/libs/jquery
/1.9.1/jquery.min.js"> </script>
```

Чем хорош этот способ? Он используется на большом количестве веб-страниц, и на момент, когда ваша веб-страница будет загружаться, скорее всего браузер пользователя уже закешировал эту ссылку, вследствие чего загрузка JQuery произойдет мгновенно. Недостаток этого способа состоит в том, что при использовании вашей веб-страницы в оффлайне (без Интернета) браузер не сможет добраться до сайта [ajax.googleapis.com](http://ajax.googleapis.com) и, следовательно, JQuery не будет загружена.

В данной лекции мы будем использовать первый способ подключения JQuery, чтобы можно было запускать все примеры в оффлайне. Файлы с библиотекой JQuery ("jquery-1.9.1.js") и плагином к JQuery для работы с cookies ("jquery.cookie.js"), который мы тоже будем использовать, находятся в архиве с примерами к данной лекции.

### 6.2. Основной принцип использования JQuery

Синтаксис команд JQuery в общем виде выглядит так:

```
jQuery ("селектор") .действие (параметры)
или
$ ("селектор") .действие (параметры)
```

Например, если мы в программе напишем:

```
jQuery ('#khai') .fadeOut (3000)
```

то это будет означать, что к элементу HTML страницы с id="khai" нужно применить действие fadeOut (эффект плавного растворения) и растянуть это действие на 3 секунды.

Вместо "#khai" можно подставлять id или class других элементов HTML (чуть позже мы рассмотрим возможные варианты), а вместо "fadeOut" - другие действия (чуть позже мы перечислим наиболее интересные и полезные из них)

Вместо слова "jQuery" в программе для краткости записи можно ставить просто знак "\$" - JavaScript нас поймет. Предыдущий пример при этом будет выглядеть совсем коротко:

```
$ ('#khai') .fadeOut (3000)
```

На рисунке 6.1 приведен пример программы (файл ex6\_01.html) с использованием вышеприведенного действия JQuery.

```
<html>

<head>

<script type="text/javascript"
src="jquery-1.9.1.js"></script>

</head>

<body>

<script>

function katya()
{
    $('#khai') .fadeOut (3000)
}

</script>



ХАИ <br clear="all">
Вы видели когда-нибудь главный корпус
ХАИ?<br>

<button onclick="katya()">Fade</button>

</body>
```

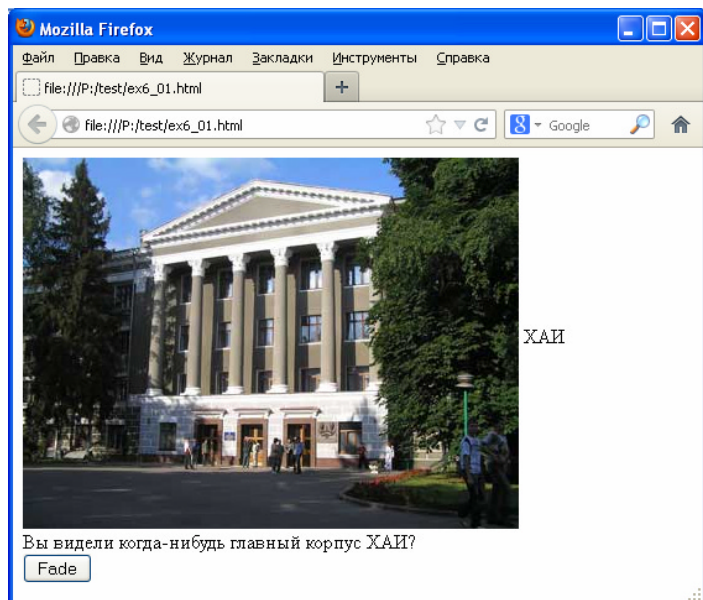


Рисунок 6.1 - Пример простой программы с JQuery

При нажатии на кнопку “Fade” изображение главного корпуса ХАИ плавно исчезнет. Чтобы это реализовать, мы для этой кнопки назначили обработку события onClick - функцию `katya()`, в которой и осуществили вызов действия `fadeOut` JQuery. Можно было все это написать еще короче, вызвав необходимое действие прямо в тексте кнопки:

```
<button onclick="$('#khai').fadeOut(3000)">
```

Здесь “`#khai`” - селектор, указывающий на элемент HTML с атрибутом `id="khai"`. Элементом HTML с таким значением атрибута `id` в тексте данной веб-страницы является тег `<img>` (изображение, которое по нашему замыслу должно плавно исчезать при нажатии кнопки).

### 6.3. Селекторы JQuery

Какие бывают виды селекторов? Перечислим их:

Селектор	Действие распространяется на
<code>#mom</code>	Первый элемент веб-страницы с <code>id="mom"</code>
<code>.mom</code>	Все элементы веб-страницы, у которых атрибут <code>class="mom"</code>
<code>elements</code>	Все теги “elements” веб-страницы. Например, <code>\$('div').fadeOut(100)</code> заставит исчезнуть с веб-страницы все теги <code>&lt;div&gt;</code> .
<code>*</code>	Вообще все элементы веб-страницы

Разобраться здесь несложно. Если нам нужно обратиться к элементу веб-страницы по его `id`, то перед значением `id` нужно поставить “`#`”. Если нужно обратиться по названию класса, то перед названием класса нужно поставить “`.`”.

Можно перечислить несколько селекторов через запятую, например,

```
$('#khai, .mom').fadeOut(500)
```

В этом конкретном случае действие `fadeOut` распространится на элемент с `id="khai"` и на все элементы с `class="mom"`.

Перейдем теперь к описанию того, что можно быстро и просто сделать с помощью JQuery.

### 6.4. Изменение свойств стилей с помощью JQuery

Часто программа на JavaScript должна что-то делать со значениями свойств стилей каких-то элементов веб-страницы - считывать их значения или изменять их. JQuery позволяет это делать проще.

Для доступа к свойствам стилей в JQuery есть действие (метод) `css`, которое в общем виде вызывается так:

```
$('селектор').css('параметры')
```

Если нужно просто узнать значение, например, отступа сверху для элемента веб-страницы с `id="papa"`, и записать это значение в переменную `misha`, то нужно написать так:

```
misha = $('#papa').css('top')
```

Здесь “`top`” - имя свойства стиля. Если бы нам нужно было другое свойство стиля, например, отступ слева, мы написали бы `css('left')`.

Если нам нужно присвоить свойству стиля новое значение, то нужно написать так:

```
$('#papa').css('top', '15px')
```

Здесь “`15px`” - новое значение для отступа сверху элемента веб-страницы, у которого `id="papa"`.

Если нужно поменять сразу несколько свойств стиля, можно писать так:

```
$('#papa').css({'top': '3px', 'left': '7px'})
```

В общем виде это будет выглядеть так:

```
.css({styleName1:value1,styleName2:value2,...})
```

```
<html>
<head>
<script type="text/javascript"
src="jquery-1.9.1.js"></script>
</head>
<body>

<style>
.ppp {
width:400px;
height:150px;
background-color:#ff0000; }
</style>

<script>
function myfun()
{
$('#.ppp').css("background-color",
"#00ff00");
}
</script>

<div class="ppp"> Текст </div>
<br>
<button onclick="myfun()">Изменить
цвет</button>

</body>
```

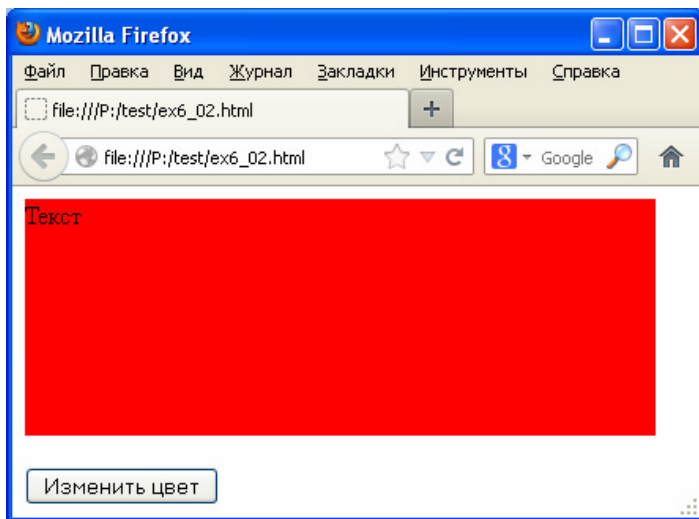


Рисунок 6.2 - Программа изменяет свойство стиля

На рис. 6.2 приведен пример веб-страницы, где JQuery используется для изменения свойств стилей (файл ex6\_02.html).

При нажатии на кнопку все `<div>` с атрибутом `class="ppp"` (а такой на странице только один) поменяют цвет (свойство `background-color`) на зеленый.

### 6.5. Методы для изменения содержимого элементов HTML

Перечислим основные команды (действия, методы) JQuery, позволяющие манипулировать с содержимым (текстом) элементов HTML веб-страницы.

Команда	Результат
<code>html(val)</code>	Код указанного элемента веб-страницы заменяется на <code>val</code>
<code>text(val)</code>	Текст указанного элемента веб-страницы заменяется на <code>val</code>
<code>text()</code>	Возвращает текст указанного элемента веб-страницы (позволяет проверить, что там сейчас находится)
<code>append(val)</code>	Добавляет <code>val</code> к коду указанного элемента веб-страницы после существующего кода
<code>prepend(val)</code>	Добавляет <code>val</code> к коду указанного элемента веб-страницы, вставляя его до существующего кода
<code>wrap(val)</code>	Оборачивает указанный элемент в обертку. Например, <code>\$('#ja').('&lt;span&gt;&lt;/span&gt;')</code> поместит элемент с <code>id="ja"</code> внутрь тега <code>&lt;span&gt; .. &lt;/span&gt;</code> .
<code>empty()</code>	Удаляет из указанного элемента HTML все элементы-потомки (вложенные элементы HTML)
<code>remove()</code>	Удаляет указанный элемент из текста HTML-страницы
<code>clone()</code>	Клонирует (повторяет) указанный элемент HTML

На рисунке 6.3 приведен простой пример с использованием команды `append` (файл ex6\_03.html) и результатом нажатия на кнопку.

```
<html>
<head>
<script type="text/javascript"
src="jquery-1.9.1.js"></script>
</head>
<body>

<script>
function otv()
{
    $('#maksim').append('<br> Спасибо,
нормально. ');
}
</script>

<div id="maksim"> Здравствуйте, как
дела? </div>
<br>
<button onclick="otv()" ">
Ответить</button>

</body>
```

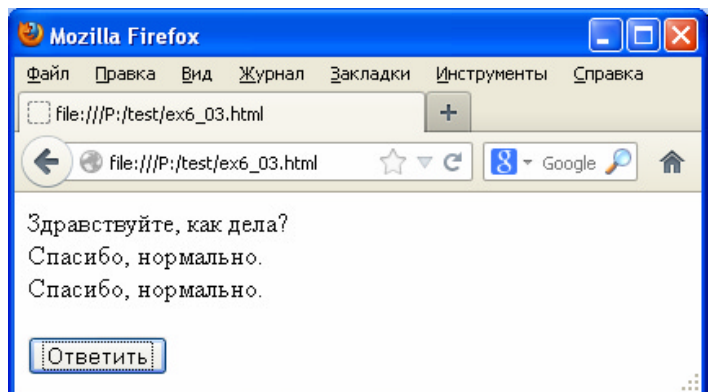
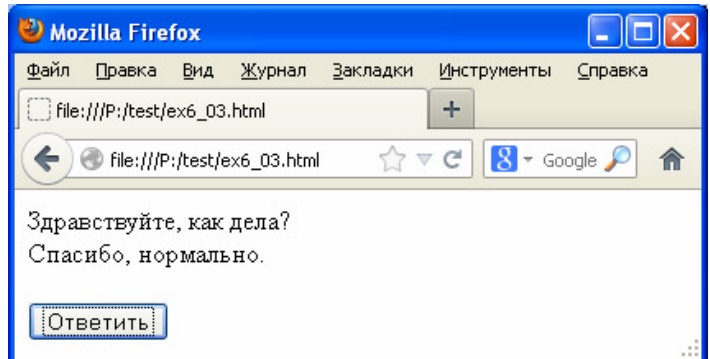
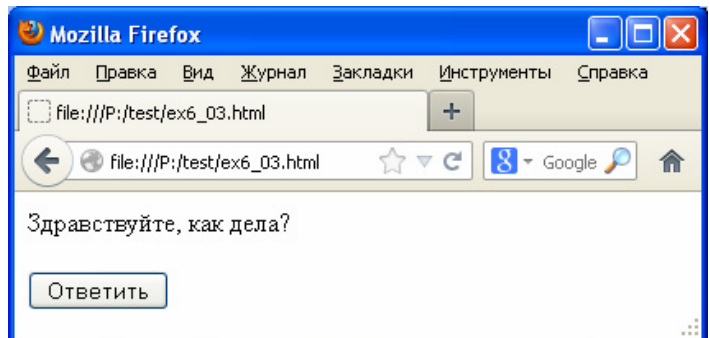


Рисунок 6.3 - Программа, которая добавляет код в указанный элемент, и результат ее работы после первого и второго нажатий на кнопку



## 6.6. Обработка событий с помощью JQuery

Перечислим основные методы JQuery для обработки событий и приведем несколько примеров.

Команда	Результат
<code>ready(fn)</code>	Функция с именем <code>fn</code> выполнится один раз, когда браузер полностью загрузит содержимое веб-страницы
<code>bind(tip,fn)</code>	Функция с именем <code>fn</code> теперь будет всегда вызываться при возникновении события <code>tip</code> . Например: <code>\$('#pap').bind('click',myclick)</code> Можно указать несколько событий через пробел. Вот лишь некоторые из возможных событий: <code>click</code> , <code>dblclick</code> , <code>change</code> , <code>keydown</code> , <code>keyup</code> , <code>keypress</code> , <code>mousedown</code> , <code>mouseover</code> , <code>mouseout</code> , <code>mousemove</code> , <code>submit</code> , <code>select</code> , <code>focus</code> , <code>blur</code> .
<code>one(tip,fn)</code>	То же, что и <code>bind</code> , но функция <code>fn</code> выполнится только один раз
<code>hover(fover, fout)</code>	Функция <code>fover</code> будет выполняться, когда курсор мыши наезжает на указанный элемент HTML, а функция <code>fout</code> будет выполняться, когда курсор мыши покидает указанный элемент HTML
<code>toggle(f1,f2)</code>	Первый клик по указанному элементу вызовет функцию <code>f1</code> . Второй клик вызовет функцию <code>f2</code> . Таких функций через запятую можно указать несколько.
<code>click(fn)</code>	При нажатии на указанный элемент всегда будет вызываться функция <code>fn</code> .
<code>click()</code>	Обманывает браузер, заставляя его подумать, что пользователь кликнул по указанному элементу

Здесь перечислены не все методы для обработки событий. Например, для метода `bind('click',fn)` аналогом является `click(fn)` и имеется `click()`, обманывающая браузер. Точно такие же аналоги имеются и для других событий. Например для `bind('focus', fn)` аналогом является `focus(fn)`, а для обмана браузера существует `focus()`.

Фактически, метод `bind` удобнее только, если есть несколько событий, для обработки которых нужно назначить одну и ту же функцию. Например, вместо:

```
$('#mam').click(obrabortka)
$('#mam').dblclick(obrabortka)
```

можно написать более короткий текст:

```
$('#mam').bind('click,dblclick',obrabortka)
```

Не забываем, что основное назначение JQuery - облегчить нашу работу по написанию программ JavaScript, чтобы мы нажимали поменьше клавиш в процессе написания программы.

Рассмотрим простой пример использования метод `hover` (рисунок 6.4, файл `ex6_04.html`).

```
<html>
<head>
<script type="text/javascript"
src="jquery-1.9.1.js"></script>
</head>
<body>

<style>
.ppp {
width:400px;
height:150px;
background-color:#ff0000;
}
</style>

<script>
function myf1()
{
$('#kk').css("background-color",
"#00ff00");
$('#kk').text("Курсор внутри");
}
function myf2()
{
$('#kk').css("background-color",
"#ff0000");
$('#kk').text("Курсор снаружи");
}
</script>

<div id="kk" class="ppp"> Текст </div>

<script> $('#kk').hover(myf1,myf2)
</script>

</body>
```

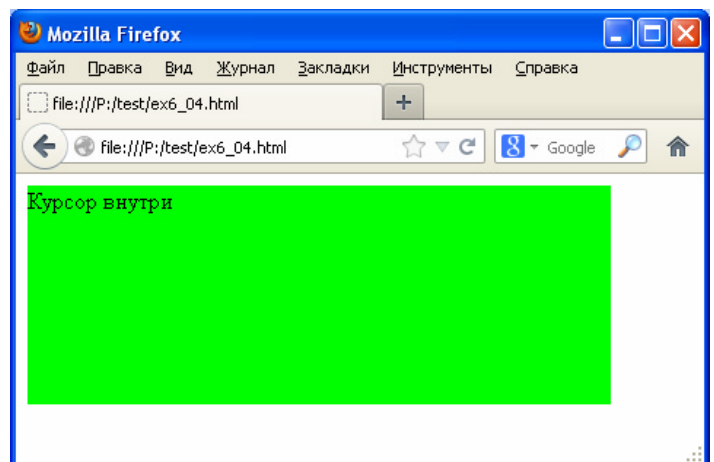


Рисунок 6.4 - Программа, реагирующая на два события, когда курсор мыши наезжает и покидает элемент

Обратите внимание, что функции для обработки событий мы назначили с помощью метода `hover` уже после того, как вставили в текст страницы сам элемент `<div> .. </div>` с `id="kk"`. Браузер пытается выполнять команды JavaScript сразу же, после того, как их увидит, даже, если еще не вся страница загружена. В данном же случае на момент, когда браузер обрабатывает команду `hover`, он уже знает про элемент `<div> .. </div>` с `id="kk"` и корректно отработает команду.

А вот в этом случае обработчик не был бы назначен:

```
<script> $('#kk').click(myf); </script>
<div id="kk" class="ppp"> Текст </div>
```

Почему? Браузер попытался бы выполнить скрипт с назначением функции myf обработчиком события onClick для тега с id="kk", а такой тег ему еще неизвестен (не загружен на данный момент). В итоге при нажатии на этот div ничего происходить не будет. Корректно написать вот так:

```
<div id="kk" class="ppp"> Текст </div>
<script> $('#kk').click(myf); </script>
```

Вроде бы тоже самое, но порядок действий здесь важен. Все, к чему мы пытаемся получить доступ из JavaScript, должно находиться по тексту HTML-страницы раньше данного фрагмента кода JavaScript или же мы должны быть уверены, что на данный момент (когда выполняется этот фрагмент кода) страница уже полностью загрузилась.

Рассмотрим еще один пример - обработка движения мыши (рисунок 6.5, файл ex6\_05.html).

```
<html>
<head>
<script type="text/javascript"
src="jquery-1.9.1.js"></script>
</head>
<body>

<script>
function myf(event)
{
    $('#uu').text("Координаты
X="+event.pageX+" Y="+event.pageY);
}
</script>

<div id="uu" style="width:400px;
height:100px; background-
color:lightblue"></div>

<script> $('#uu').mousemove(myf)
</script>
</body>
```

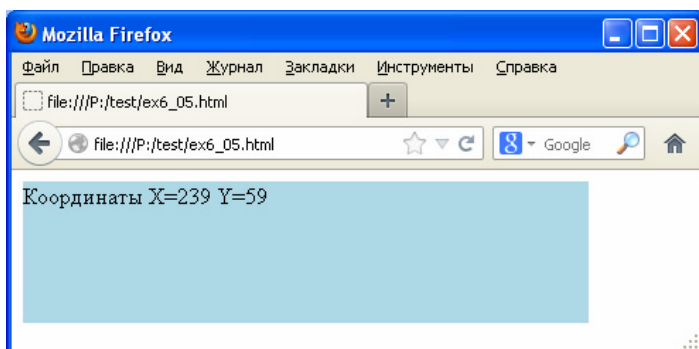


Рисунок 6.5 - Программа показывает координаты мыши

Чтобы наша программы выдавала координаты курсора мыши (внутри выбранного нами <div> с id="uu"), нужно обрабатывать параметры события, которые JQuery передаст в нашу функцию в переменной event (в частности, поля

event.pageX и event.pageY содержат координаты курсора мыши по X и Y).

Откуда мы это узнали и как узнать названия других переменных, если нам понадобится обработать какое-то другое сложное событие? Не стесняйтесь использовать поисковые системы в Интернете. Запомнить наизусть все невозможно. В данном случае поиск в Google по фразе "jquery mousemove coordinates" или "jquery mousemove x y" приведет вас в нужное место, где все объяснено на примерах.

## 6.7. Визуальные эффекты и анимация на JQuery

Перечислим основные методы JQuery для анимации и визуальных эффектов.

Команда	Результат
hide(tim)	Скрывает заданный элемент HTML за заданное в миллисекундах время tim.
show(tim)	Ранее невидимый заданный элемент HTML появляется на экране за время tim.
toggle(tim)	Если элемент - невидим, то показать его. Иначе - скрыть.
slideUp(tim)	Плавно свернуть элемент
slideDown(tim)	Плавно развернуть элемент
slideToggle(tim)	Плавно свернуть или развернуть элемент
FadeIn(tim)	Уменьшает прозрачность элемента
FadeOut(tim)	Увеличивает прозрачность элемента
animate(par,tim)	Плавно меняет значения свойств стилей до указанных в par
stop()	Останавливает анимацию (которая началась после вызова animate и еще не успела закончиться)

Пример с использованием FadeOut был рассмотрен в самом начале этой лекции. А здесь рассмотрим более сложный пример, с использованием метода animate.

Если написать, например, так:

```
$('#ni').animate({width:'10',height:'30'}, 100)
```

то это будет означать, что элементу HTML-страницы с id="ni" мы говорим плавно за 100 миллисекунд изменить свою ширину до 10 пикселей, а высоту - до 30 пикселей.

В общем виде это выглядит так:

```
animate( {свойство1:'значение1', свойство2:
'значение2', свойство3:'значение3'}, время)
```

Может быть изменено сколько угодно свойств (они перечисляются через запятую). Если в названии свойства имеется тире, то правило написания имени такое же - как и в JavaScript (см. лекцию N3). Например, если свойство стиля называется "background-position", то при вызове animate нужно писать "backgroundPosition".

На рисунке 6.6 приведен пример программы с JQuery плавной анимацией (файл ex6\_06.html) свойств одновременно двух элементов HTML.

```

<html>
<head>
<script type="text/javascript"
src="jquery-1.9.1.js"></script>
</head>
<body>

<style>
#bar1 { background-color:red;
width:400; height:100; }
#bar2 { background-color:green;
width:200; height:20; }
</style>

<script>
function nnn()
{
$('#bar1').animate({width:"100",
height:"55"},500);
$('#bar2').animate({width:"500",
height:"200"},500);
}
</script>

<div id="bar1"></div><br>
<div id="bar2"></div><br>
<button
onclick="nnn()">Изменить</button>

</body>

```

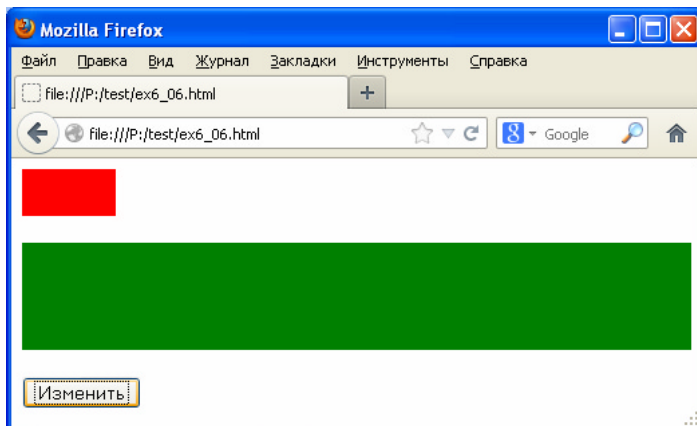
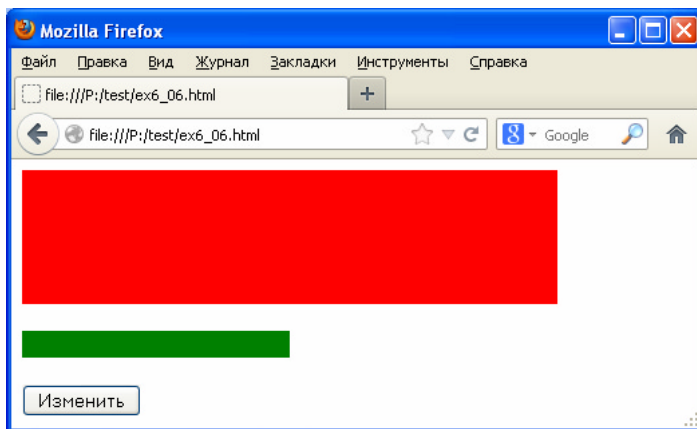


Рисунок 6.6 - Результат анимации свойств стилей

Использовать метод `animate` для преобразования одного графического изображения в другое или для преобразования одного цвета в другой JQuery не умеет.

## 6.7. Отправка запросов к серверу без перезагрузки страницы

Традиционное взаимодействие браузера и сервера подразумевает отправку браузером запроса на сервер (например, в результате нажатия пользователем на кнопку), после чего сервер возвращает ответ в виде новой веб-страницы и браузер выводит эту страницу на экран.

Однако часто мы сталкиваемся с ситуацией, когда браузер вроде бы не перегружал страницу, но информация с сервера на нее явно попадает (рисунок 6.7).

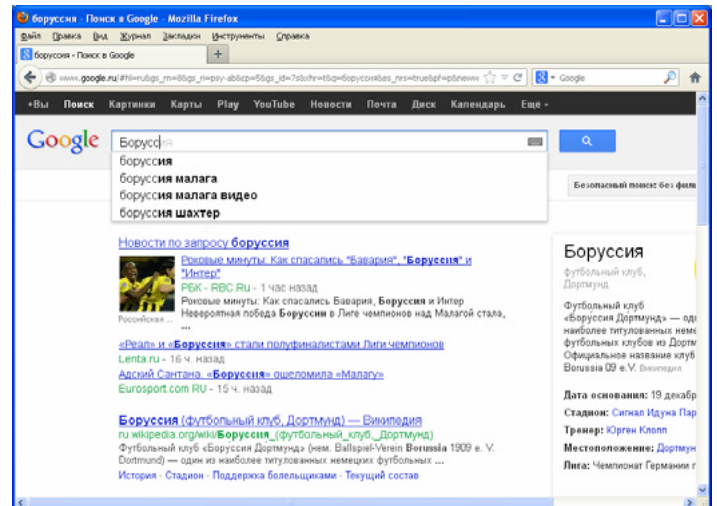


Рисунок 6.7 - Иллюстрация загрузки информации с сервера "на лету" без перезагрузки веб-страницы браузером

На данном рисунке показана ситуация, когда мы только начали вводить слово в поисковой строке на страничке Google, а нам уже показывают варианты найденного! Понятно, что обрабатывать события и на лету менять содержимое страницы не сложно - в этом нам поможет JavaScript. Но откуда взялся весь этот текст? Ведь не был же он заранее загружен с сервера вместе с веб-страницей? Для этого сервер Google должен уметь читать наши мысли на расстоянии.

На самом деле ничего сверхъестественного не происходит. Просто веб-страница умудряется посылать запросы на сервер и получать ответы с сервера без перезагрузки браузером текста страницы. Как? Есть несколько способов, например, с использованием тега `<iframe>`, к тому же для разных браузеров это могут быть разные способы, поэтому рассматривать их здесь не будем. Скажем лишь, что эта трудоемкая технология носит название **Ajax** и в JQuery она реализована так, что пользоваться ей легко и удобно.

Прежде, чем переходить к примерам, которые расскажут нам, как пользоваться технологией Ajax, перечислим основные достоинства и недостатки ее использования. Достоинствами использования Ajax являются:

- Экономия трафика, уменьшение нагрузки на сервер;
- Ускорение реакции интерфейса на действия пользователя, что, безусловно, нравится всем пользователям;
- Повышение функциональных возможностей программы, так как не нужно заставлять пользователя нажимать на что-то, если нужно отправить запрос к серверу.

Недостатки Ajax:

- Ухудшение индексации содержимого сайта поисковыми системами;
- Сложнее учитывать статистику.



Итак, первое, что нам понадобится в JQuery, это метод `serialize()`, который собирает данные формы в некое подобие GET запроса (см. рисунок 6.8, файл `ex6_07.html`).

```
<html>
<head>
<script type="text/javascript"
src="jquery-1.9.1.js"></script>
</head>
<body>

<script>
function my()
{
    var data = $('#nik').serialize();
    alert('Результат: '+data);
}
</script>

<form id="nik" action="primer.php"
method="POST">
    <p>Ваше имя? <input type="text"
name="nam" size="15"></p>
    <p>Год рождения? <input type="text"
name="year" size="7"></p>
    <p><input type="submit"></p>
</form>
<button onclick="my()">
Посмотреть</button>
</body>
```

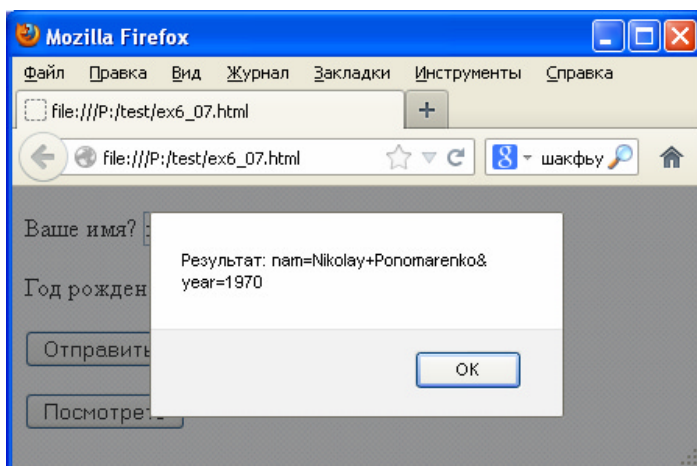
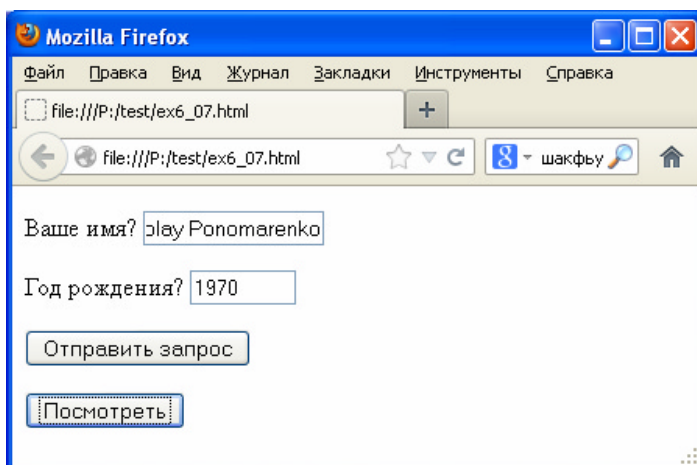


Рисунок 6.8 - Программа при нажатии кнопки “Посмотреть” показывает результаты сериализации данных формы

Как видим, команда `$('#nik').serialize()` подготовила данные формы с `id="nik"` к отправке на сервер. А именно преобразовала содержимое строк “Nikolay Ponomarenko” и “1970” к виду, используемому в GET запросах: “`nam=Nikolay+Ponomarenko&year=1970`”.

Здесь “`nam`” и “`year`” - значения атрибута “`name`” тегов `<input>` формы, данные которой сериализуются.

Отправить данные на сервер технологией Ajax и получить ответ можно, используя метод `post`:

```
$.post(адрес, данные, обработчик_ответа)
```

Рассмотрим это на примере. Пусть у нас есть PHP-программа, которая проверяет, совпадает ли присланная с веб-страницы строка с названием одного из поселков Харьковской области (рисунок 6.9, файл `ex6_08.php`).

```
<?php
$f=fopen('kh.txt','r');
$b=false;
while (!feof($f))
{
    $s=fgets($f);
    $s=trim($s);
    if ($s==$_POST['nam'])
        $b=true;
}
fclose($f);
if ($b)
    echo "Есть такой поселок в
    Харьковской области: ",$_POST['nam'];
else
    echo "Такого поселка нет в
    Харьковской области: ",$_POST['nam'];
?>
```

Рисунок 6.9 - Программа проверяет, есть ли название среди названий поселков Харьковской области

Текстовая строка поступает в программу методом POST. Переменная, содержащая эту строку, должна иметь название “`nam`”. Допустим, эта переменная содержит строку “Васищево”. Программа функцией `echo` возвращает либо строку “Есть такой поселок в Харьковской области: Васищево”, либо строку “Такого поселка нет в Харьковской области: Васищево” в зависимости от того, найдет ли она такое название в файле `kh.txt`.

Создадим теперь веб-страницу, которая будет запрашивать от пользователя название, посылать Ajax запрос серверу и выводить полученный результат на этой же странице без ее перезагрузки.

Нам понадобится форма со строкой ввода. Далее мы назначим функцию на какое-либо событие, связанное с изменением этой строки, например, `keyup`. В функции, которая это событие обрабатывает, мы будем сериализовать данные формы, отправлять их с помощью технологии Ajax на сервер, получать ответ и вставлять этот ответ на веб-страницу, опять же используя JQuery. Программа, которая при написании ее с нуля на JavaScript заняла бы у нас несколько сотен строк, со всеми пустыми строками займет не более тридцати.

На рисунке 6.10 приведен текст этой веб-страницы (файл `ex6_09.html`) и результата ее работы.

```

<html>
<head>
<script type="text/javascript"
src="jquery-1.9.1.js"></script>
</head>
<body>

<script>
function obrabotka(otklik, status)
{
    if (status=='success')
        $('#otvet').html(otklik);
    else
        alert('Ошибка!');
}
function lena()
{
    data = $('#nik').serialize();
    $.post('ex8_14.php',data, obrabotka)
}
</script>

<form id="nik">
    Название поселка? <input id="ira"
type="text" name="nam">
</form>
<br>

<div id="otvet"></div>
<script> $('#ira').keyup(lena) </script>
</body>

```

Здесь “lena” - название функции, которая будет запускаться, если событие произошло.

В функции lena() первым делом мы сериализуем (подготавливаем к отправке) данные формы. Так как форме мы назначили id=“nik”, то выглядит это следующим образом:

```
data = $('#nik').serialize
```

Теперь переменная data содержит строку, которую нужно отправить на сервер.

Отправляем запрос на сервер следующей командой:

```
$.post('ex8_14.php', data, obrabotka)
```

Здесь 'ex8\_14.php' - адрес PHP-программы, которая будет обрабатывать запрос, data - отправляемые данные, obrabotka - название функции, которая примет ответ сервера.

Осталось рассмотреть текст функции obrabotka. У этой функции две входных переменных. В переменной otklik сервер возвращает текст веб-страницы (то, что в программе ex8\_14.php выводится функцией echo). В переменной status JQuery возвращает нам ‘success’, если с запросом все прошло нормально.

Поэтому в функции obrabotka для начала проверяется, содержит ли переменная status значение ‘success’. Если да, то присланный сервером текст страницы (переменная otklik) выводится на веб-страницу:

```
$('#otvet').html(otklik)
```

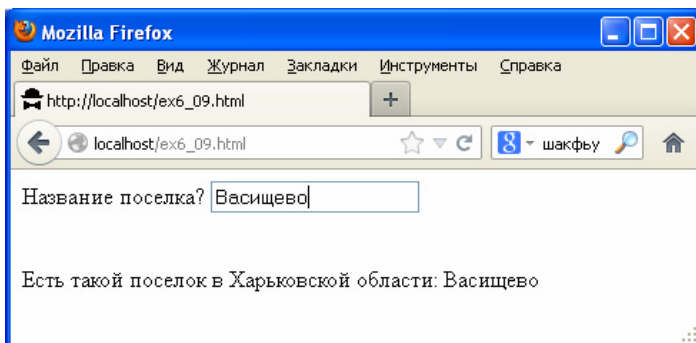
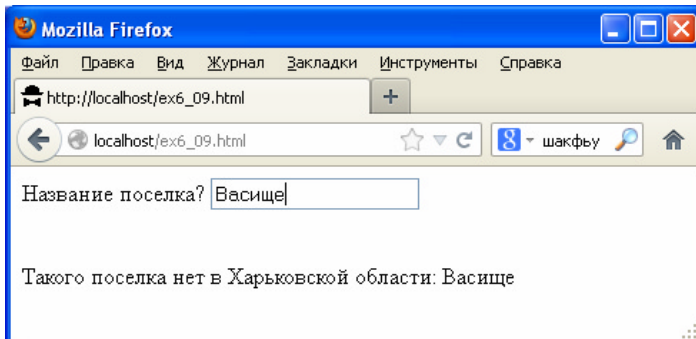


Рисунок 6.10 - Программа посылает на сервер запросы с помощью технологии Ajax

Поясним текст этой программы. Строке ввода мы дали id=“ira”, поэтому назначаем обработку события keyup для нее следующим образом:

```
$('#ira').keyup(lena)
```

Здесь уже знакомый нам метод html используется для того, чтобы записать строку otklik в тег <div>, которому мы дали id=“otvet” (см. рисунок 6.10).

Рассмотренного примера вполне достаточно для начала работы с Ajax на JQuery. Сервер может пересылать в ответ любые данные и наша программа может анализировать их любым способом, который мы придумаем, как разработчик программы.

Данные формы не обязательно должны быть видимыми на экране (можно использовать тип “hidden”). Не обязательно вообще использовать формы - можно подготовить данные для отправки на сервер как-нибудь иначе. Например, так:

```
dat=' fio=Ivanoff';
$.post('obr.php', dat, mama);
```

Далее будете изучать тонкости использования Ajax на JQuery самостоятельно с помощью Интернета и справочников по мере практической необходимости.

## 6.8. Работа с cookies на JQuery

Представим, что нашему сайту нужно, чтобы пользователь вводил какую-то информацию о себе, например, логин и пароль. Но мы хотим, чтобы пользователь один раз залогинился, а при следующих заходах на наш сайт мы его каким-то образом бы узнавали и не спрашивали пароль заново. Как вариант, можно запомнить его IP-адрес, когда он логинился и потом, если зайдет еще один человек с этим же IP-адресом, то пускать его без пароля. Но, во-первых, у нескольких разных людей может быть один и тот

же IP-адрес. И, во-вторых, человек может выходить в Интернет с ноутбука с динамически получаемым IP-адресом, который у него каждый раз будет новым.

И здесь помогает механизм, называемый **cookies** и позволяющий нашему сайту хранить информацию на компьютере пользователя, в браузере (браузер сохраняет эту информацию на жестком диске, поэтому она не теряется с выключением компьютера).

Механизм **cookies** предусматривает, что сайт может хранить в браузере сколько угодно (в разумных пределах) переменных и их значений, где для каждой переменной хранятся:

имя значение время\_хранения

Когда время хранения переменной заканчивается (а устанавливает его наш сайт), то браузер удаляет ее из своей памяти.

Сайт может проверять значения только тех переменных в **cookies**, которые сам же ранее установил. Проверить значения чужих переменных нельзя.

Вот и все, что нужно знать о **cookies**, а теперь на примере (рисунок 6.11, файл ex6\_10.html) рассмотрим, как работать с **cookies** на JQuery.

```
<html>
<head>
<script type="text/javascript"
src="jquery-1.9.1.js"></script>
<script type="text/javascript"
src="jquery.cookie.js"></script>
</head>

<body>

<script>

$.cookie('fio','Пономаренко Николай
Николаевич',{ expires: 7 });

</script>

Куки добавлены!

</body>
```

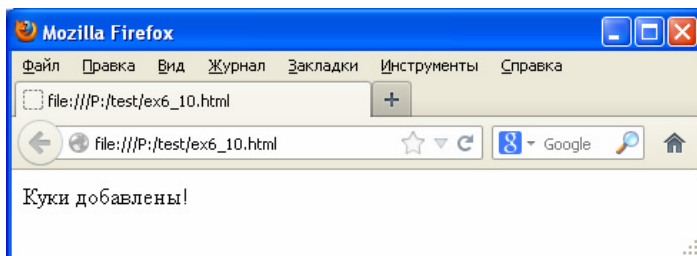


Рисунок 6.11 - Программа сохраняет информацию в cookies

В этой программе используется специальный плагин к JQuery, называющийся "JQuery.Cookie". Его текст нужно подключить к программе точно так же, как и текст самого JQuery:

```
<script type="text/javascript"
src="jquery.cookie.js"></script>
```

Команда

```
$.cookie('fio','Пономаренко Николай
Николаевич',{ expires: 7 });
```

добавляет в **cookies** переменную fio, присваивает ей значение 'Пономаренко Николай Николаевич' и задает срок хранения для этой переменной в 7 дней.

Таким образом, программа в вышеприведенном примере ничего не делает, а только записывает в **cookies** эту переменную.

В следующем примере (рисунок 6.12, файл ex6\_11.html) показано, как прочитать значение переменной из **cookies**.

```
<html>
<head>
<script type="text/javascript"
src="jquery-1.9.1.js"></script>
<script type="text/javascript"
src="jquery.cookie.js"></script>
</head>

<body>

<script>
pa=$.cookie('fio');
document.writeln('Здравствуйте, '+pa);
</script>

</body>
```

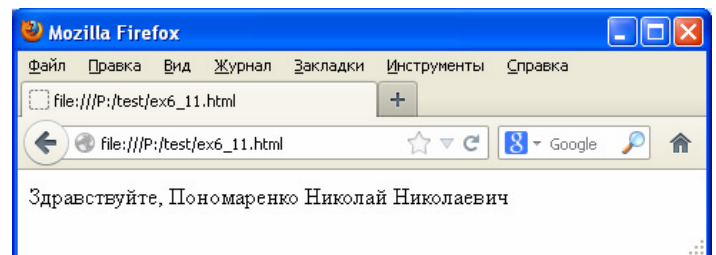


Рисунок 6.12 - Программа читает информацию из cookies

Как видим, чтобы прочитать информацию из **cookies**, нужно написать:

```
pa=$.cookie('fio')
```

и информация из переменной fio **cookies** будет записана в переменную pa JavaScript.

Чтобы досрочно удалить переменную fio из **cookies**, нужно написать:

```
$.cookie('fio','')
```

Как видим, все очень просто.

На этом лекция по JQuery заканчивается. Относительно длинной она получилась не из-за сложности ее использования, а из-за богатств возможностей, перечисление которых и заняло столько места.

В качестве самостоятельной работы можете рассмотреть фильтры JQuery, но это совсем не обязательно. Правильный подход здесь - изучать какие-то редко используемые возможности языка программирования по мере возникновения в них практической необходимости.