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.

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

Например, можно использовать для отладки PHP-программ бесплатный набор программного обеспечения Denver, который включает в себя веб-сервер Appache, PHP, MySQL и все необходимое:

1. Скачиваем Denver с сайта http://www.denver.ru и при установке Denver-а соглашайтесь на все предлагаемые по умолчанию настройки.

2. 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”.

3. 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 PHP-программы и оттуда их запускать на исполнение.

Настройка Appache

По умолчанию в Denver PHP-код в файлах с расширением “.html” не исполняется. 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).

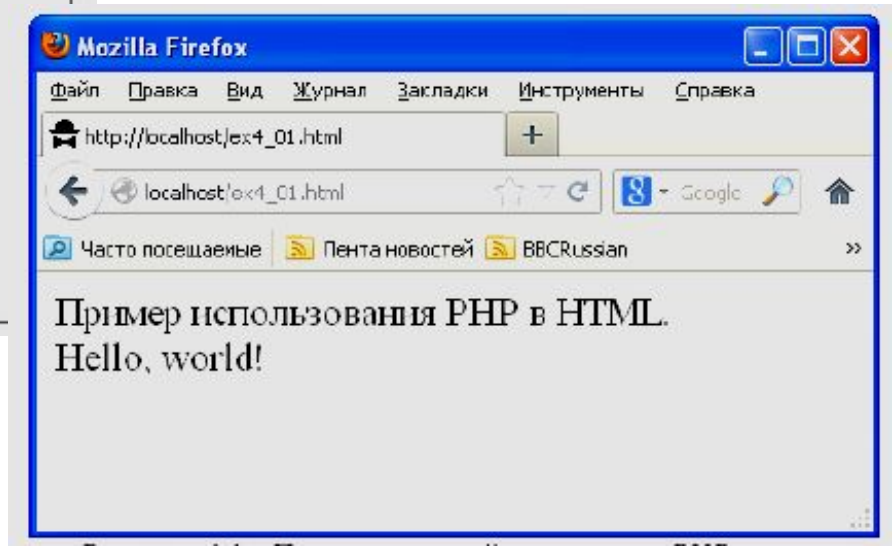
2) Перед этой строкой вставляем строку: **“RemoveHandler .html .htm**”

3) Удостоверяемся, что в директиве “**AddHandler server-parsed**” (если такая строка есть) нет расширений **“.html”** и “**.htm**” (если есть - удаляем их).

Для запуска веб-страницы необходимо предварительно положить её в папку “**Z:\home\localhost\www**”, а в браузере написать не просто “программа.html”, а “**http://localhost/программа.html**”.

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++.

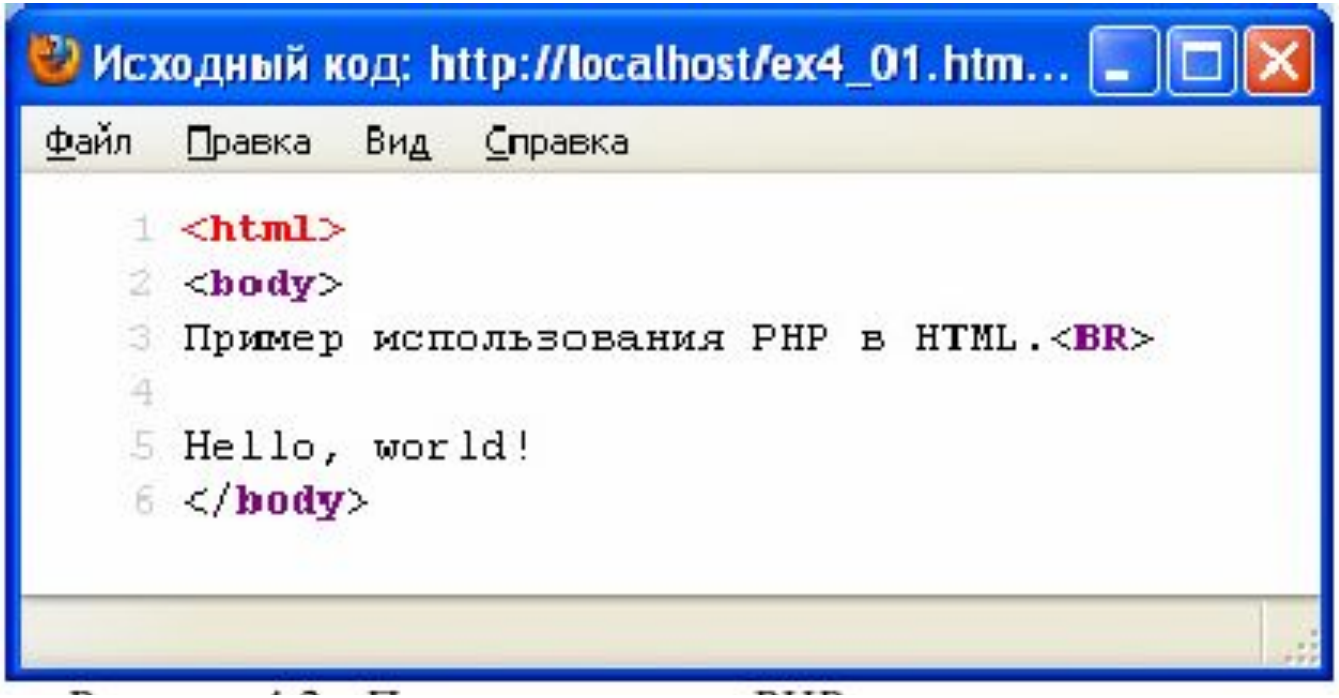
```
<html>
<body>
Пример использования PHP в HTML.<BR>

<?php
 echo "Hello, world!";
?>

</body>
```



PHP-код на HTML-странице 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).

Пример экрана браузера при просмотре в нем исходного текста веб-страницы.



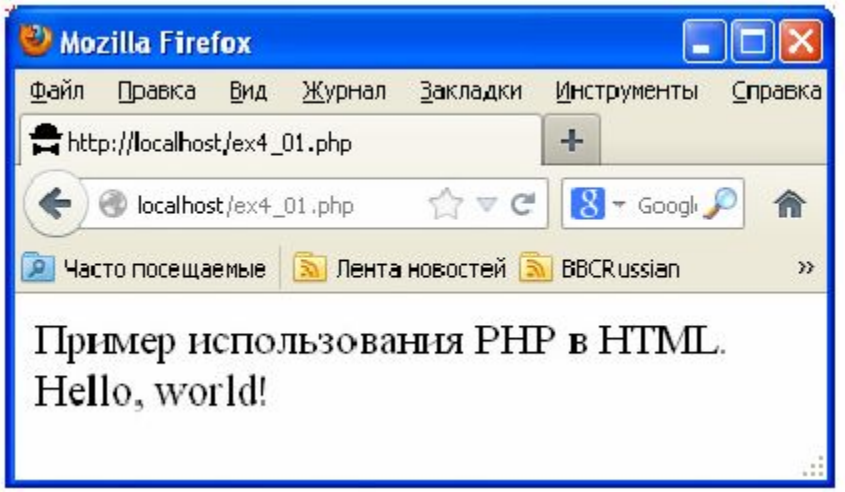
На рисунке мы действительно видим исходный текст страницы, однако вместо текста вставленной в эту страницу PHP-программы мы видим лишь результат ее выполнения - надпись “Hello, world!”.

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).

Эта же страница, только оформленная в виде файла с расширением ***.php**

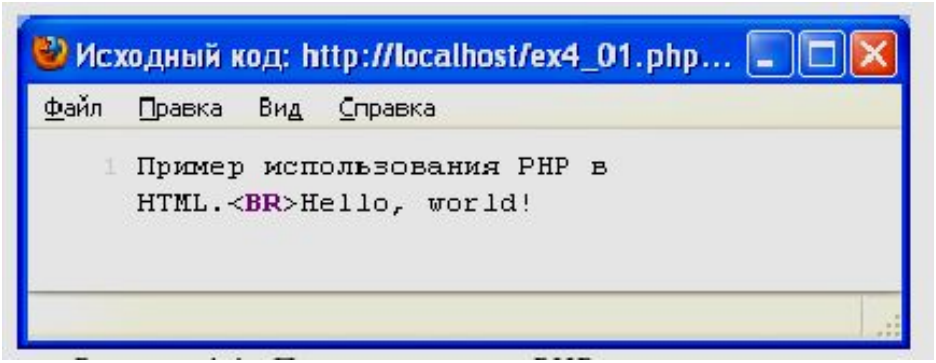
```
<?php
 echo "Пример использования PHP в
HTML.<BR>";
 echo "Hello, world!";
?>
```



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

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**).



Попытка увидеть PHP-код данной программы при просмотре исходного текста веб-страницы в браузере

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

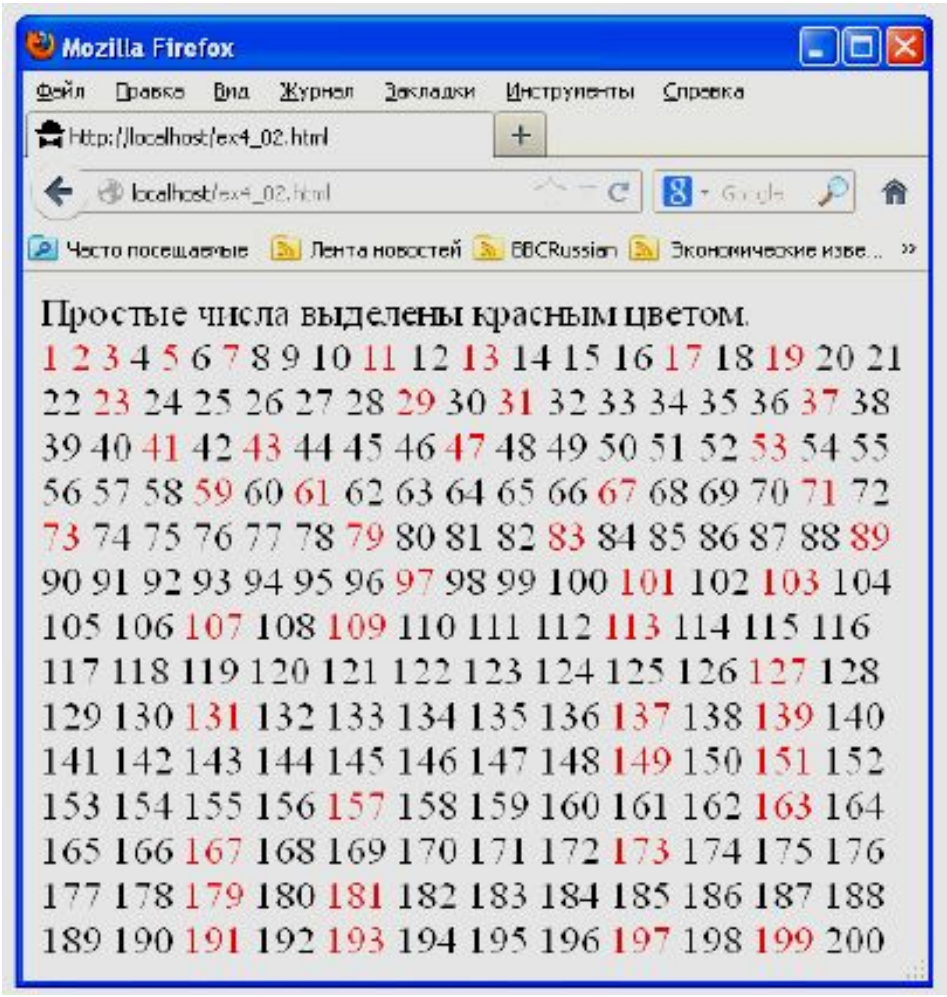
Оба способа написания PHP-программ (в отдельных PHP-файлах или внутри HTML-страницы) эквивалентны друг другу. Но написание PHP-кода в файле с расширением ***.php** позволяет нам проверить текст программы на наличие ошибок. Например, с помощью файла php.exe, находящийся в Denvera.

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

Автоматическое генерирование текста страницы

```
<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>
```

Программы, которая выведет на экран все целые числа с 1 до 200, выделив красным цветом простые числа.



Здесь мы уже видим пример программы, в которой есть функция, циклы, условные операторы, переменные.

Особенности синтаксиса PHP-програм:

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; }
```

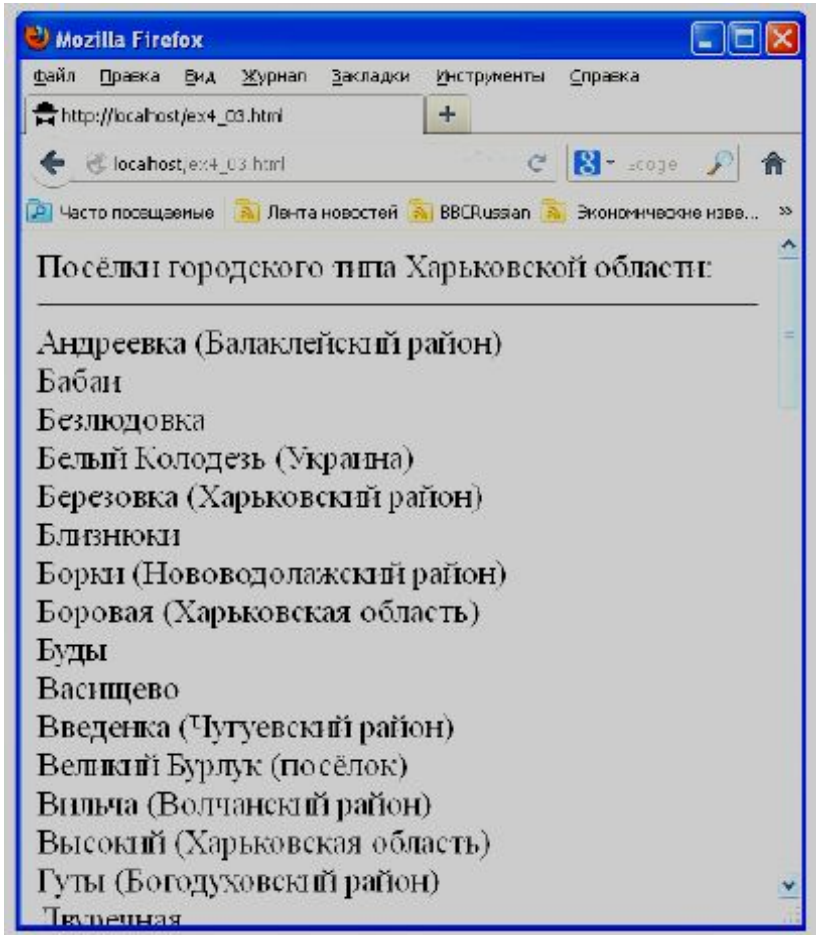
В данном примере функция “**prost**” возвращает 0, если число не простое, и 1, если число простое (ни на что не делится, кроме единицы и самого себя). Оператор **“%”** в выражении **“$n % $i**” означает “остаток от деления числа $n на число $i”.

Результат выполнения данной PHP-программы

Работа с файлами

Пример программы, которая выводит на экран список ПГТ Харьковской области, хранящийся на сервере в файле **“kh.txt”**.

```
<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>
```



Некоторые функции PHP для работы с файлами:

fopen открывает файл,

fgets читает из файла одну строку,

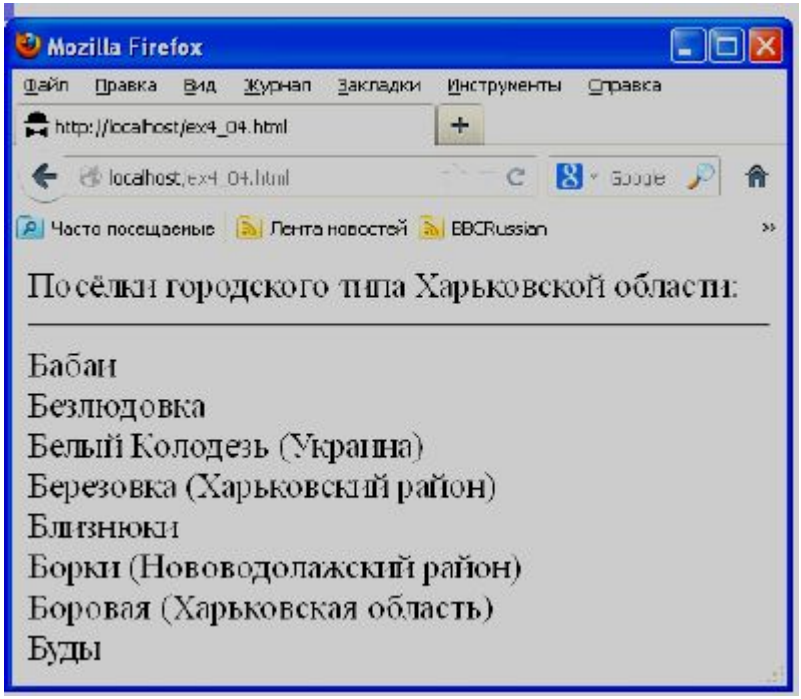
fclose закрывает файл,

feof возвращает значение “истина”, если все строки из файла уже прочитаны).

Восклицательный знак перед условием 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**”.

Программа, которая выводит на экран только поселки, начинающиеся с буквы “Б”

```
<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>
```



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:

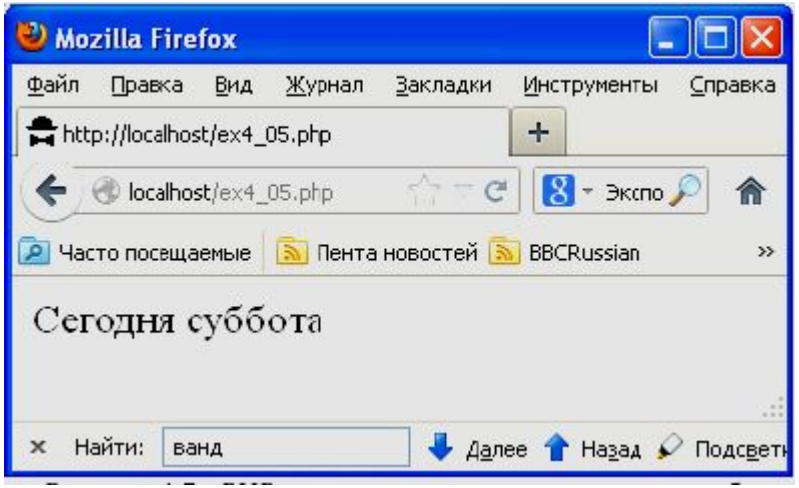
for ($n=0;$n<strlen($s);$n++) .

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

В 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++).

Программа, которая выводит на веб-страницу название текущего дня недели

```
<?php

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

?>
```



Вначале объявлен массив с именем **$myday**, содержащий 7 строк:

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

Нумерация элементов массивов, как и строк, 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 **$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.

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.

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];
```

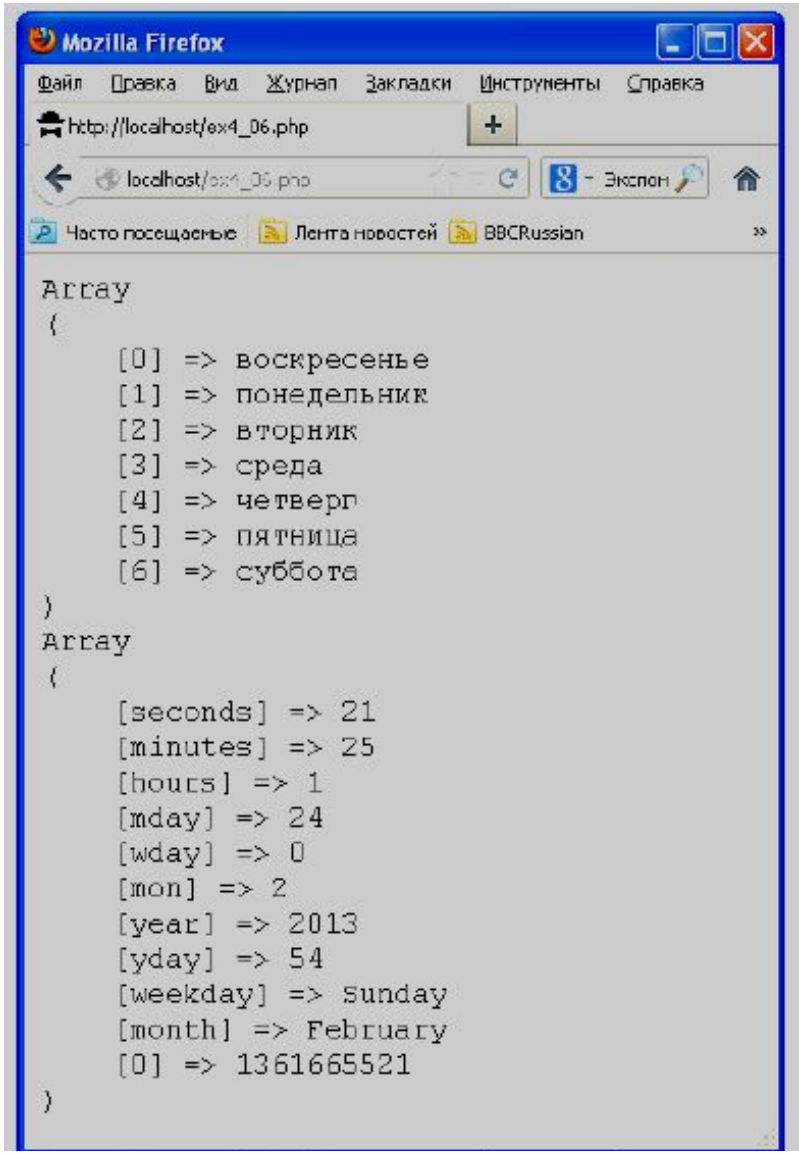
Результат будет тем же самым.

Согласно свойствам ассоциативных массивов в приведенном примере программы 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).

```
<?php

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

?>
```

```php
<?php

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

?>
```



В языке PHP есть функция **print_r,** которая позволяет вывести на экран заданный массив с названиями и значениями его ячеек. Данная программа выводит на экран значения массивов **$myday** и **$today** из предыдущего примера.

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

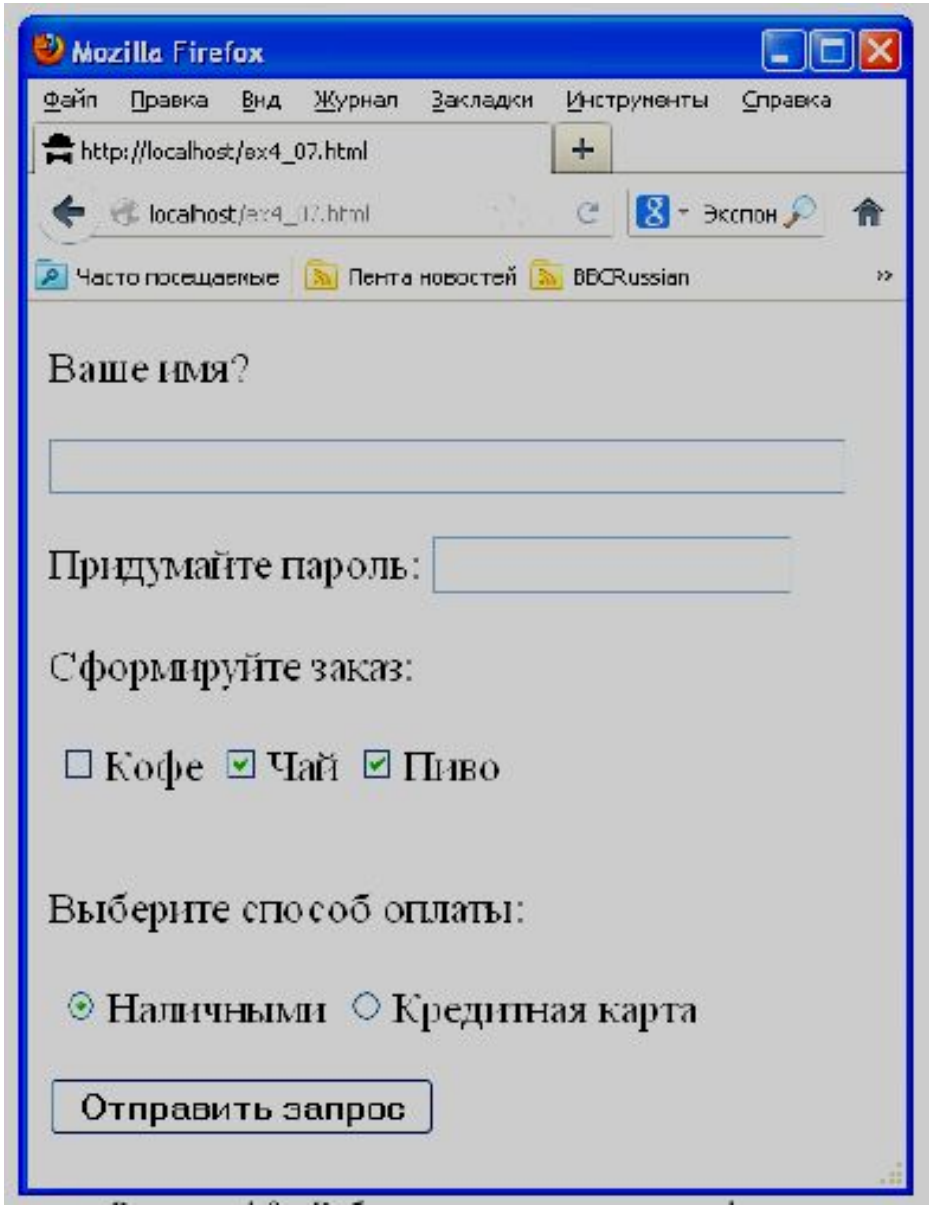
Обращаться из программы к ячейкам ассоциативного массива **$today** можно как с использованием кавычек: **$today[“year”],** так и без использования кавычек:

$today[year].

Обработка запросов к серверу на PHP

Одно из основных назначений PHP - обработка запросов к серверу, поступающих от веб-страниц или других серверов.

```
<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>
```



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”. 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).

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.

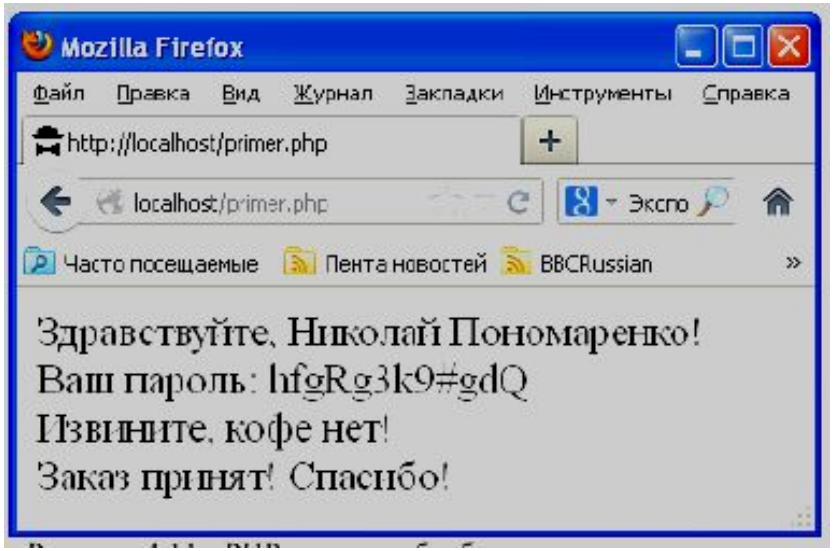
Значение **type=”text”** задает текстовую строку ввода. В этом случае атрибут **size=”50”** означает, что в строке будет максимум 50 букв. Атрибут **name** понадобится нам при обработке данных на сервере. Значение **type=”password”** также задает строку ввода. Однако буквы, которые пользователь будет 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. И **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.

Программа ***primer.php***, которая будет обрабатывать данные формы

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

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

if ($_POST['cofe']=='on')
  echo "Извините, кофе нет!<br>";

if ($_POST['money']=="var1")
  echo "Оплата наличными<br>";

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

echo "Заказ принят! Спасибо!";
?>
```



На PHP 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 элементе формы с значением тега **name=”oleg”** нужно просто обратиться к ячейке **$_POST[‘oleg’]**. Точно так же и со всеми остальными элементами формы.

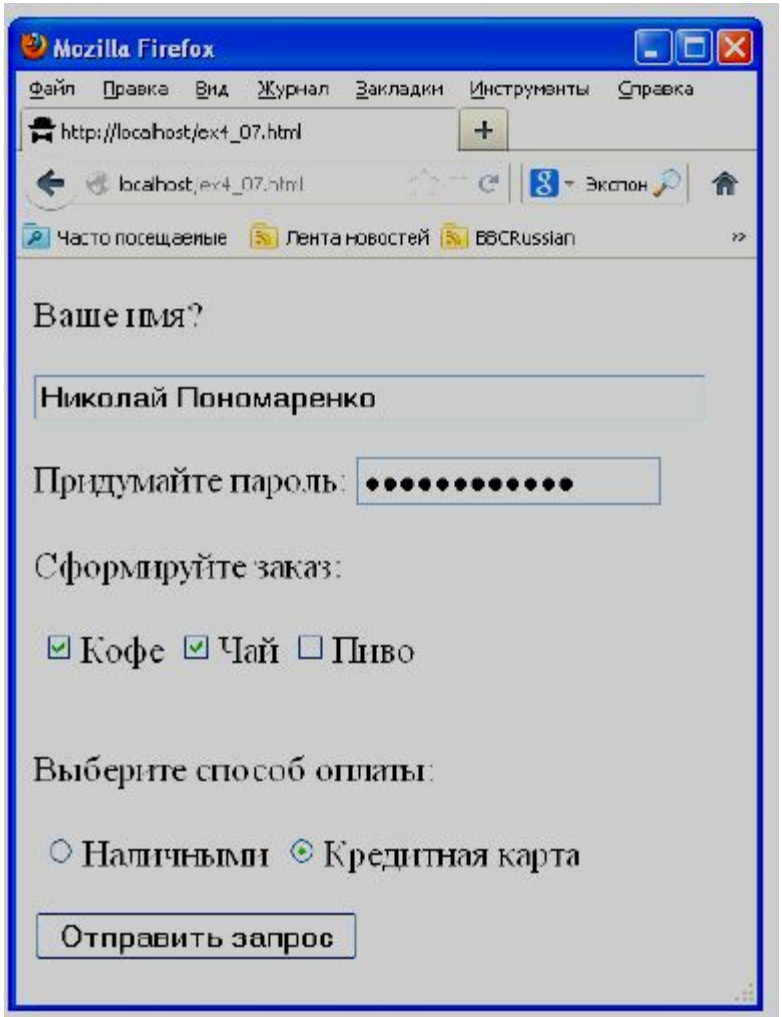
В приведенном примере на экран командами **echo** выводятся ФИО и пароль. Затем проверяется, была ли установлена галочка в элементе формы с именем ‘**cofe**’ и, если да, то выводится надпись “**Извините, кофе нет!”.**

Далее проверяется, был ли выбран элемент **radio** со значением “**var1**” и, если да, то выводится надпись “**Оплата наличными**”.

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 пременную **$st**).

Содержимое файла **zakazy.txt** для данной заполненной формы

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



На этом примере 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.

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

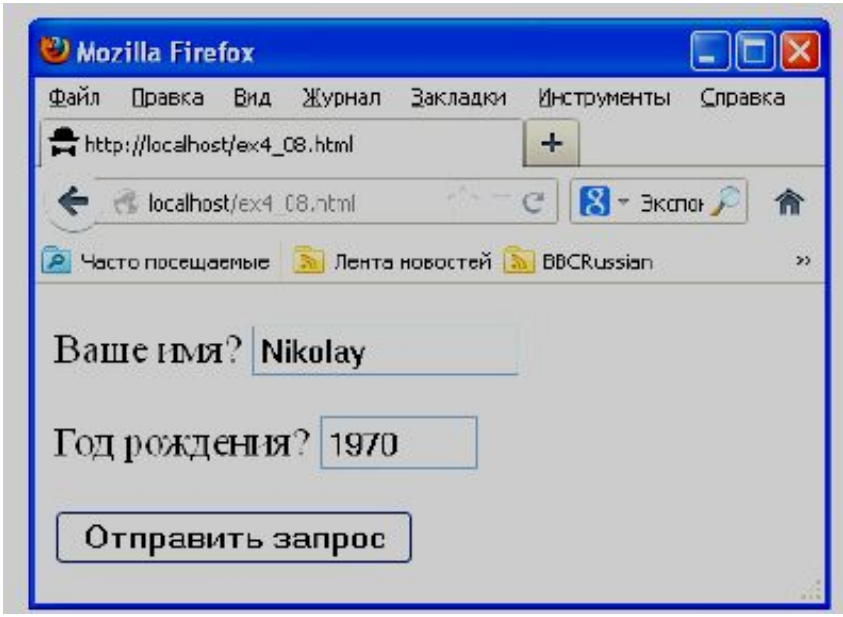
Если данные с веб-страницы по какой-то причине не были переданы, то массив **$_POST** окажется пустым.

Проверить, не пустая ли та или иная ячейка массива **$_POST** можно с помощью функции **isset($s)**, которая возвращает **TRUE**, если переменная **$s** существует. Чтобы проверить, например, были ли переданы на сервер данные с именем “**oleg**”, нужно проверить **isset($_POST[‘oleg’]).**

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**”.

```
<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>
```

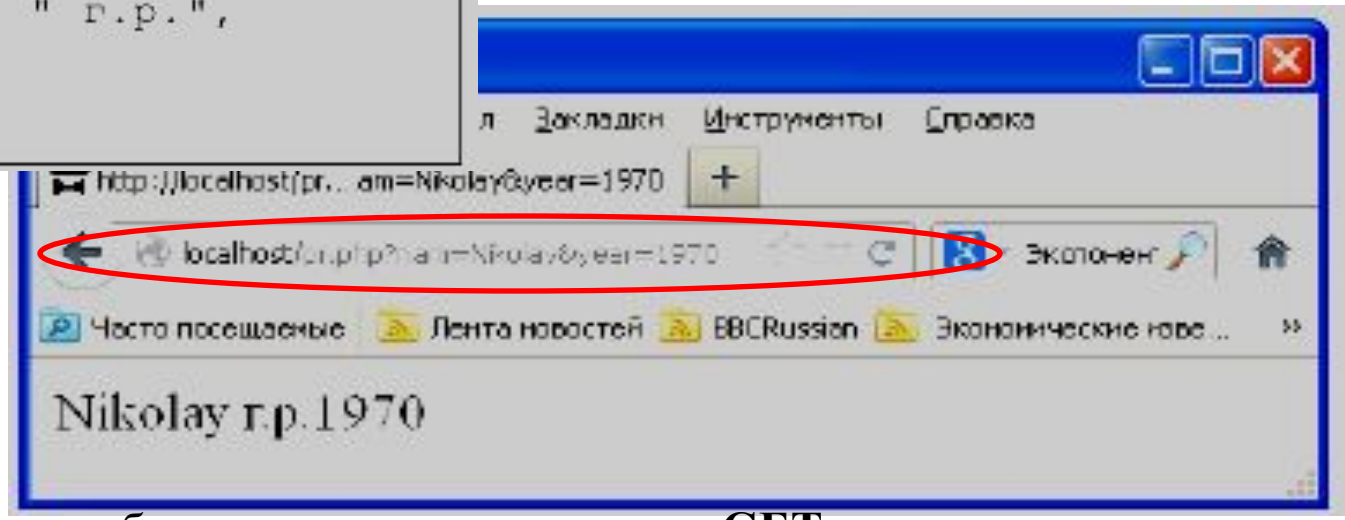


*Пример простой веб-страницы, использующей метод **GET**, и ее внешний вид в окне браузера после ввода данных.*

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.

Текст программы **pr.php**, которая обрабатывает запрос

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



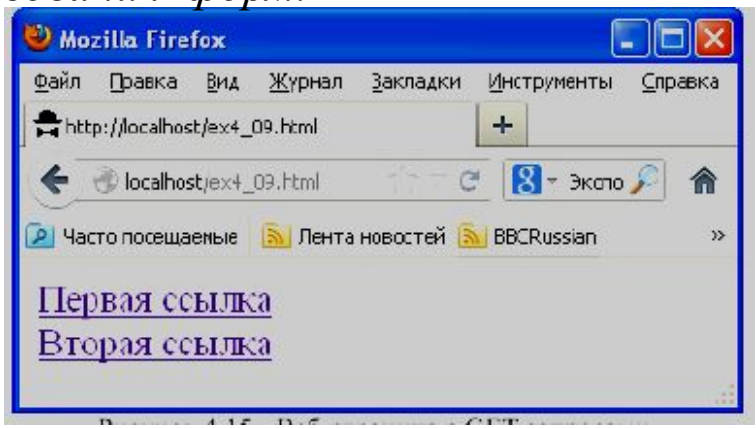
В этом случае данные веб-страницы для метода **GET** нужно читать из ассоциативного массива **$_GET**. Однако в адресной строке в браузере в этом случае значится **“localhost/pr.php?nam=Nikolay&year=1970**”. Все, что находится после знака “**?**” и есть данные, переданные от веб-страницы серверу методом **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.

*Пример веб-страницы, которая отправляет на сервер данные методом **GET** без использования форм.*

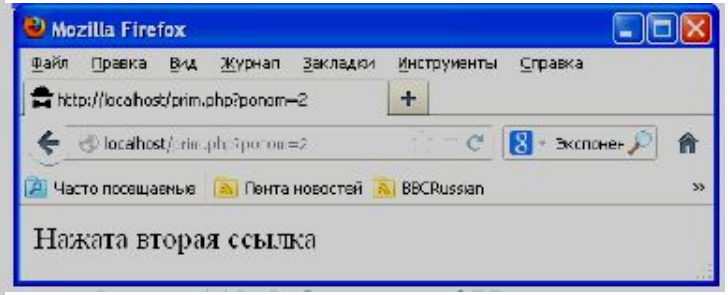
```
<html>
<body>

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

</body>
```



В данном примере используются просто ссылки с интегрированными в них запросами к серверу. Данные здесь помещены в значения атрибутов **href** тега **<a> .. ** через символ "**?**" после пути к файлу PHP-программы.

*Программа **prim.php** и результат ее работы*

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



В данном примере обе ссылки ведут к одному и тому же PHP-коду (или "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.

NOTE!!! В предыдущих примерах пользователю предлагалось заполнить форму, но не указано, 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, **'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 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**'?

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**'.

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).

PHP и MySQL

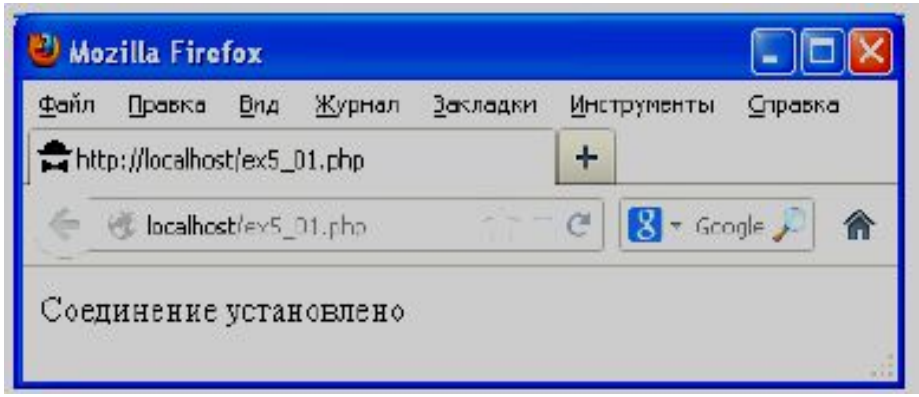
Базовые действия PHP-программы при работе с MySQL

Любая 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.

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

```
<?php

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

   echo "Соединение установлено";

   mysql_close($link);

?>
```



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).

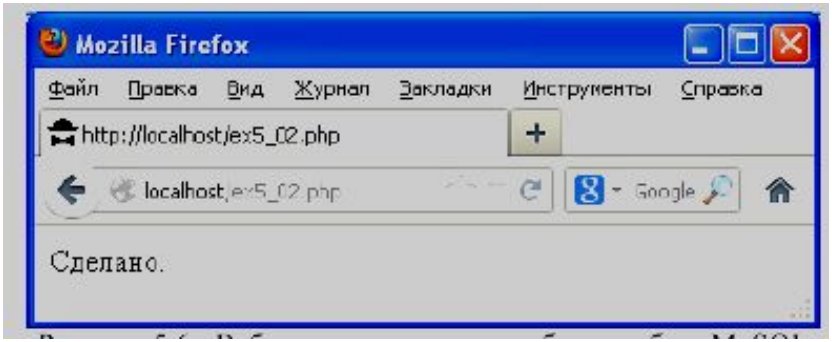
Если соединение успешно установлено, то в служебной переменной **$link** (имя этой переменной можно задать любое) возвращаются параметры соединения с сервером, которые в дальнейшем нам потребуются.

После этого программа выдает на веб-страницу надпись "**Соединение установлено**" и закрывает соединение с сервером командой **mysql_close($link).**

Создание таблицы MySQL

Веб-страница, которая таблицу в базе MySQLВеб-страница, которая таблицу в базе MySQL

```
<?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);
?>
```



Команда **mysql_select_db** это выбор базы данных и **mysql_query** - запрос к серверу.

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.**

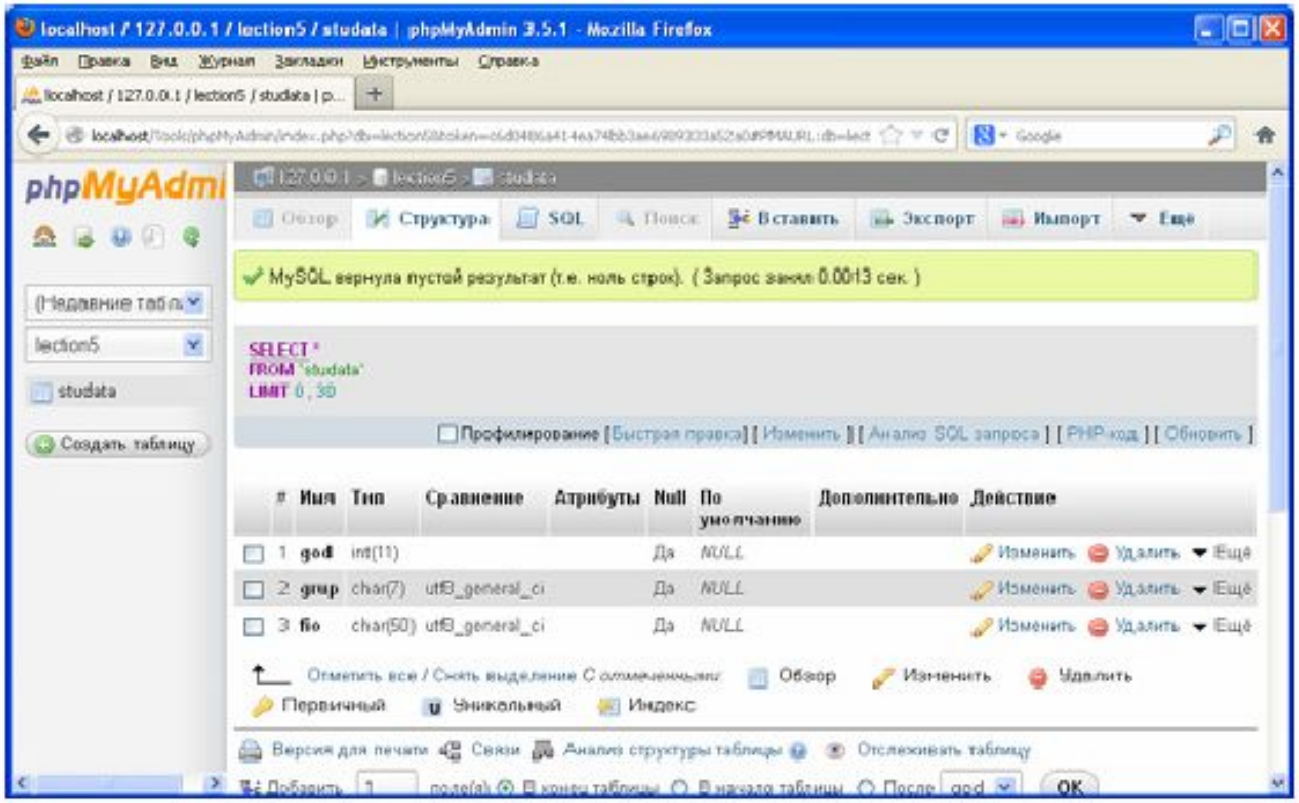
Как только мы выбрали базу данных, можно создавать в ней таблицу. Заметим, что все действия над базой данных (создание таблиц, занесение в них данных, изменение данных, поиск данных, удаление данных и т.д.) выполняются в виде запросов к MySQL серверу. Сначала формируется текстовая переменная, в которую 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.

Результат добавления в базу “lection5” таблицы “studata”

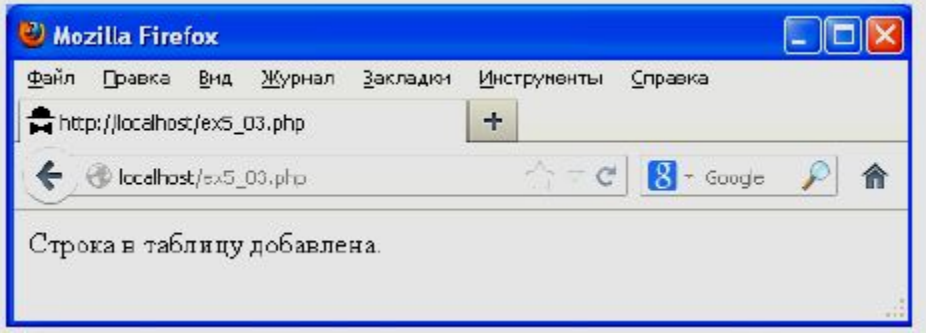


Добавление строки в таблицу

```
<?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);

?>
```

Программа, которая добавляет в таблицу “**studata**” строчку, в которой **god=1996**, **grup=”519-Б**”, а **fio=“Иванов Иван Петрович**”.

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



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.

Алгоритм действий остается одним и тем же:

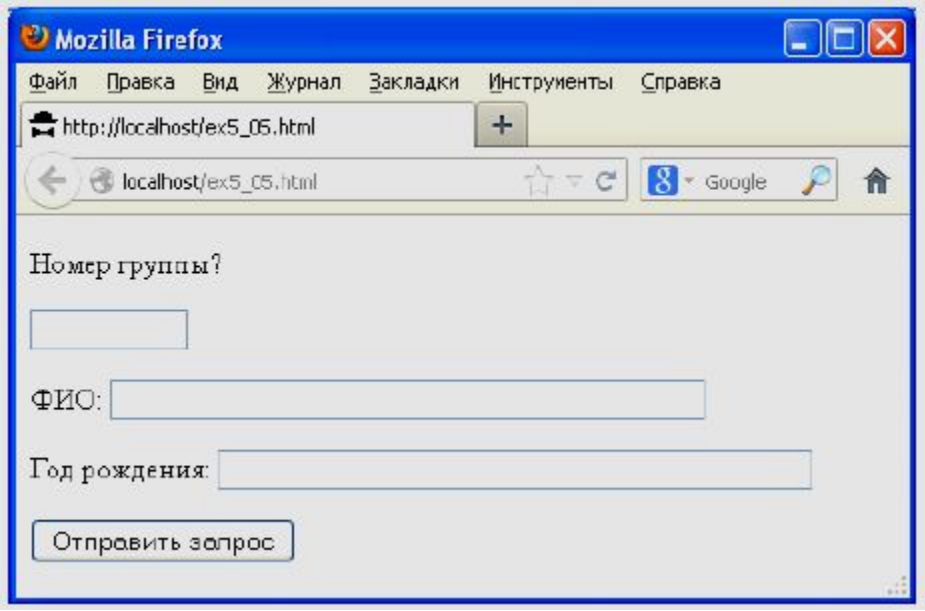
1. Установили связь с сервером.
2. Выбрали базу данных.
3. Послали команду серверу.
4. Закрыли сеанс связи.

Если ошибки проверять не нужно, то текст программы можно упростить, например, до такого

```
<?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);
?>
```

Создание веб-страницы с вводом данных, которые затем передадутся в PHP-программу на сервере, которая добавит эти данные в базу MySQL

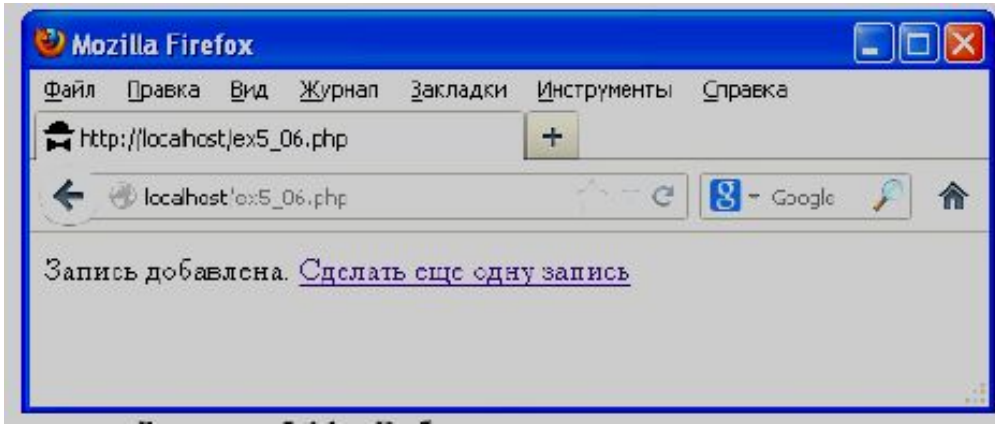
```
<html>
<body>

<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>
```



Эта веб-страница при нажатии на кнопку методом POST отправляет данные про группу, фамилию и год рождения студента на сервер программе “ex5_06.php”.

Программа “ex5_06.php” и результат ее работы

```
<?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>";
?>
```



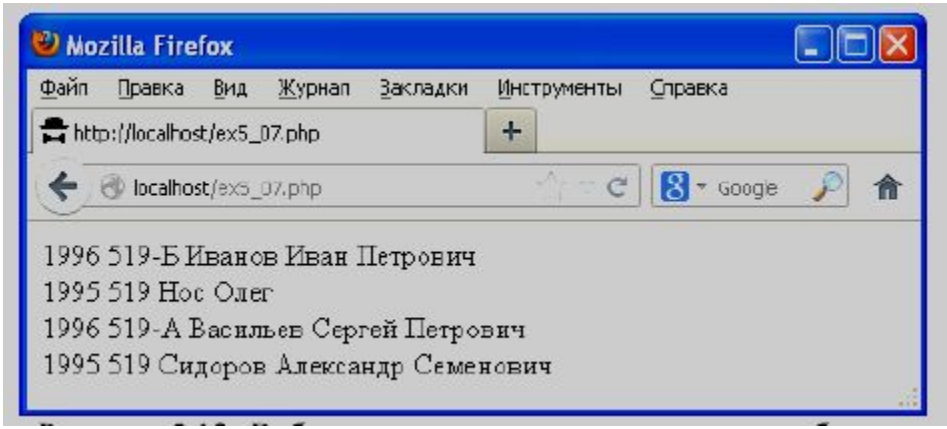
Данная программа получает пришедшие от веб-страницы данные из стандартного массива **$_POST** и затем формирует из них запрос для MySQL сервера на вставку в таблицу новой строки.

NOTE!!! 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.

Поиск и извлечение данных из таблицы

Проще всего извлечь из базы данных всю таблицу целиком

```
<?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>";
?>
```



В данном примере запросом к серверу является строка

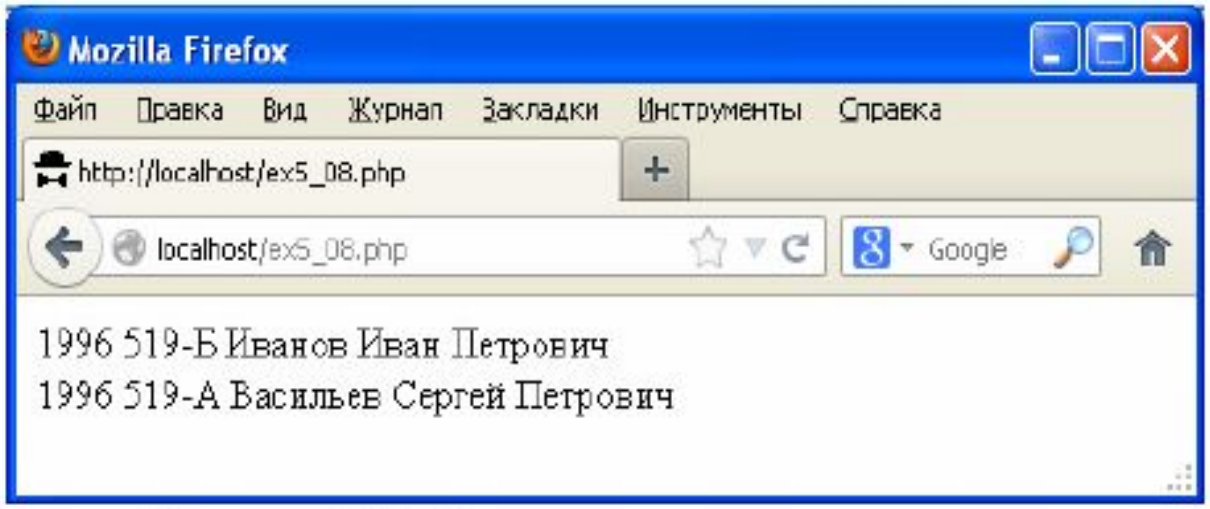
SELECT * FROM studata;

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. Например, **$q=mysql_fetch_row($r).** Она последовательно извлекает из результатов поиска **$r** по одной строке и заносит их в переменную **$q**.

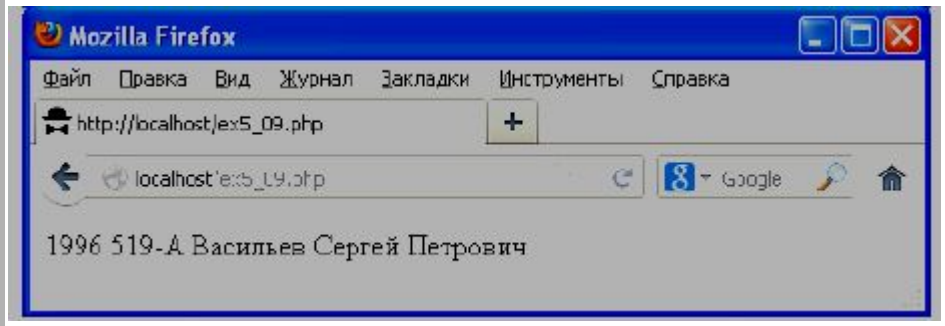
Когда строки закончатся, в **$q** будет занесен ноль и цикл **while** в примере остановится. $**q** представляет собой массив, чьи индексы являются номерами, начинающимися с нуля. Поэтому, чтобы обратиться к элементу из первого столбца таблицы, нужно написать **$q[0]** (год рождения) и т.д.

Чтобы программа 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;



```
<?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>";
?>
```



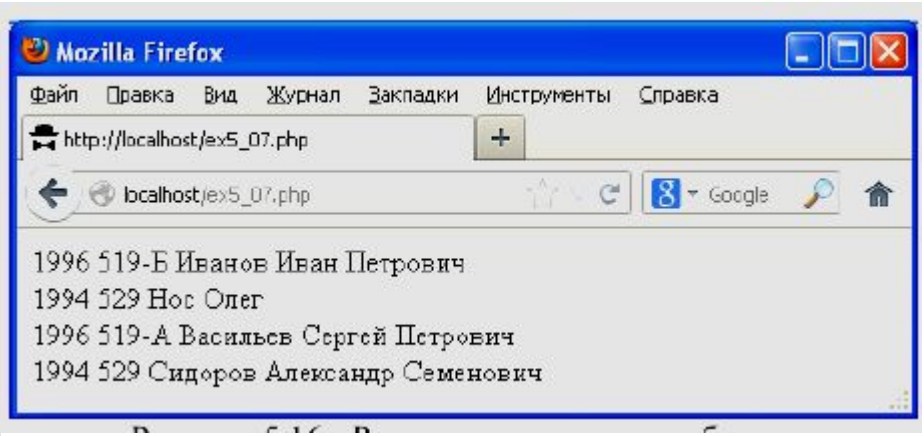
Вместе с ключевым словом “WHERE” можно использовать, например, модификатор “LIKE”. Здесь строка запроса:

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’.**

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.

```
<?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);
?>
```



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.

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.

Кодировки русского текста при работе с MySQL

Мы можем указать серверу в какой кодировке будут присылаться данные, в какой кодировке их хранить, в какой кодировке возвращать ответы сервера.

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**”.

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.

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.

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.

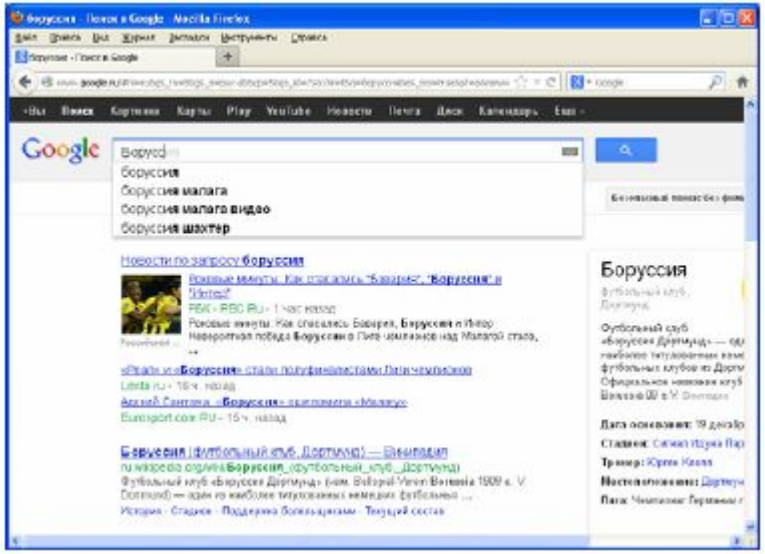
В MySQL предусмотрены еще внешние ключи для связи таблиц между собой. Если столбец назначен внешним ключом (**FOREIGN KEY**), то указывается связанный с ним столбец другой таблицы, и сервер не будет разрешать добавлять в таблицу данные со значениями, которых нет в том связанном столбце.

PhP и JQuery:
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. При этом веб-страница посылает запросы на сервер и получает ответы с сервера без перегрузки браузером текста страницы.

Это может быть реализовано несколькими способами, например, с использованием тега **<iframe>,** к тому же для разных браузеров это могут быть разные способы. В общем, эта технология носит название **Ajax** и в Jquery она реализована так, что пользоваться ей легко и удобно.



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 нажать на что-то, если нужно отправить запрос к серверу.

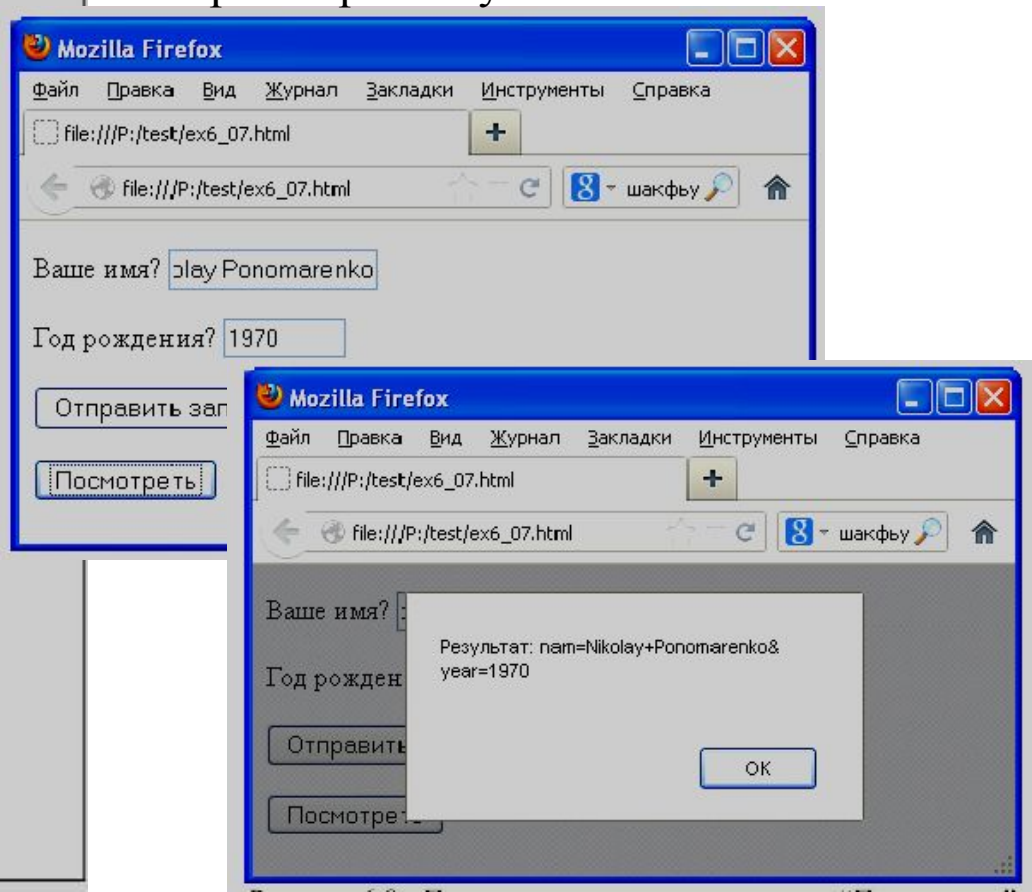
Недостатки Ajax:

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

В JQuery существует функция (или метод), **serialize()**, который собирает данные формы в некое подобие GET запроса

```
<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>
```

Команда **$('#nik').serialize()** подготовила данные формы с **id=”nik”** к отправке на сервер. Т.е. 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)

Пример

```
<?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'];
?>
```

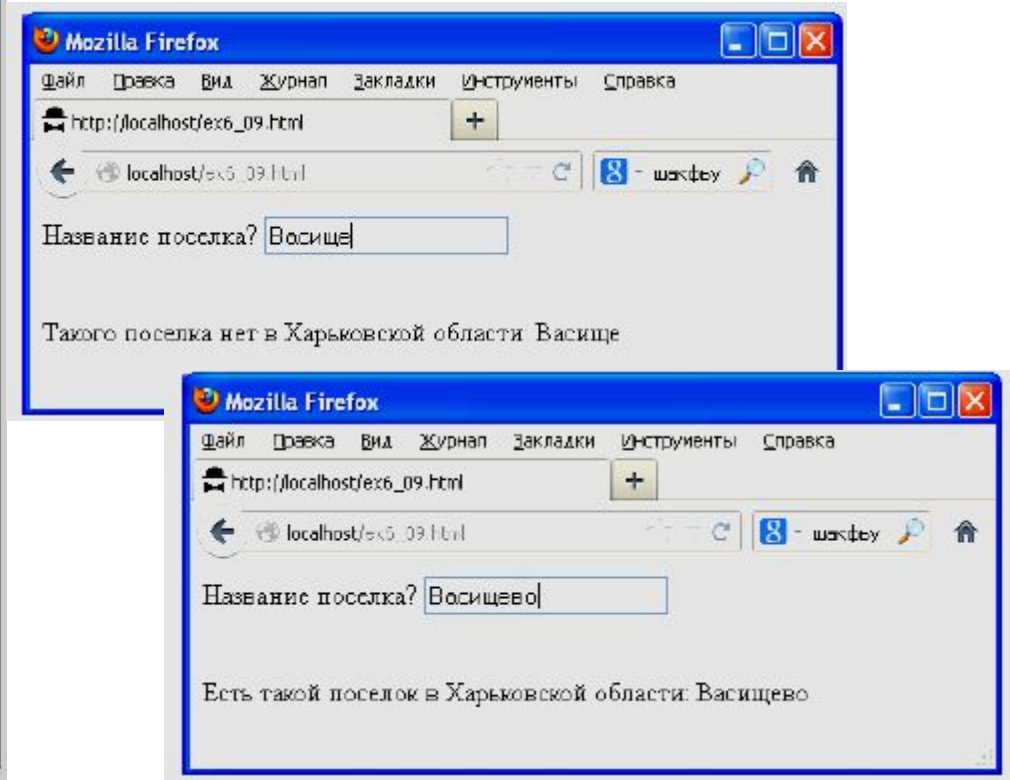
Есть PHP-программа, которая проверяет, совпадает ли присланная с веб-страницы строка с названием одного из поселков Харьковской области.

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.

Для создания веб-страницы, которая будет запрашивать от пользователя название, посылать Ajax запрос серверу и выводить полученный результат на этой же странице без ее перезагрузки, понадобится форма со строкой ввода.

```
<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>
```

Далее назначается функция на какое-либо событие, связанное с изменением этой строки, например, **keyup**. В функции, которая это событие обрабатывает, будут сериализироваться данные формы, отправляться с помощью технологии Ajax на сервер, получать ответ и вставлять этот ответ на веб-страницу, опять же используя JQuery.



На строку ввода с **id=”ira**” назначена обработка события **keyup** следующим образом:

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

Здесь “**lena**” - название функции, которая будет запускаться, если событие произошло. В функции **lena()** мы сериализируем (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.

У функции **obrabotka** две входных переменных. В переменной **otklik** сервер возвращает текст веб-страницы (то, что в программе 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)

Здесь метод **html** используется для того, чтобы записать строку **otklik** в тег <d**iv**>, которому мы дали **id=”otvet” .**

Данные формы не обязательно должны быть видимыми на экране (можно использовать тип “**hidden**”).

Работа с cookies на JQuery

Механизм **cookies** позволяет 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 значение чужих переменных нельзя.

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>

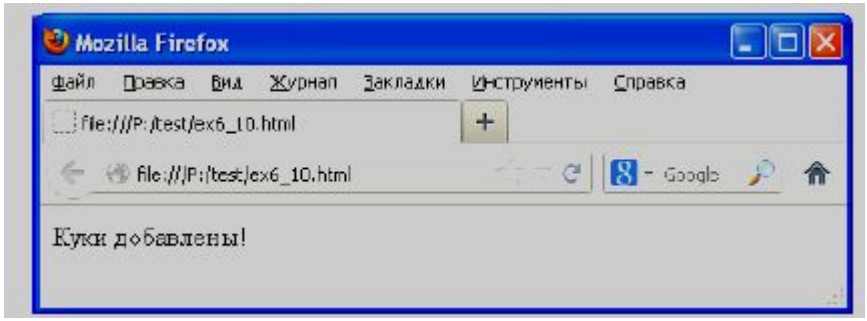
Команда

$.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.

```
<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>
```

Таким образом, программа в этом примере ничего не делает, а только записывает в **cookies** эту переменную



Как прочитать значение переменной из cookies.

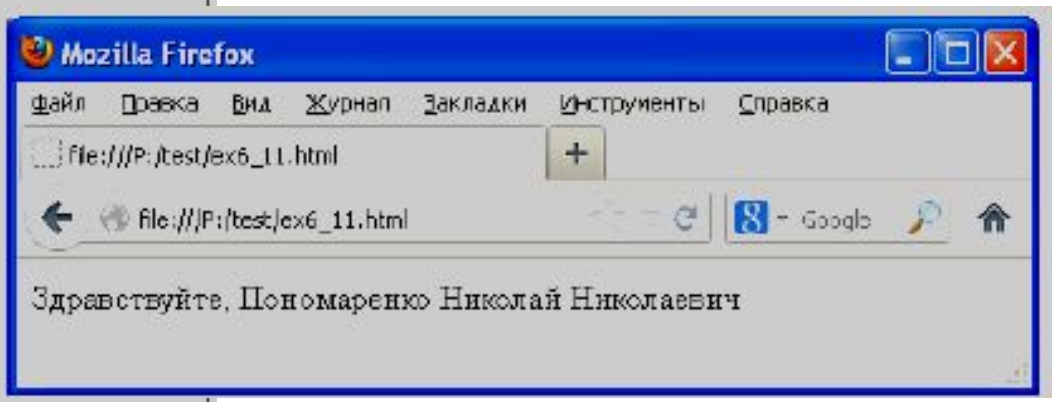
Чтобы прочитать информацию из **cookies**, нужно написать:

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','')

```
<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>
```



Web Design

УСПЕШНОЙ РАЗ ТКИ!