Функции для работы с БД MySql

1) Установка соединения

Для установки соединения используются две функции:

mysql_connect(<Имя хоста>, <Имя пользователя>, <Пароль>);

Постоянное соединение

mysql_pconnect(<Имя хоста>, <Имя пользователя>, <Пароль>);

Функция mysql_connect() устанавливает обычное соединение с сервером MySQL, которое закрывается, когда сценарий завершает работу или когда вызывается функция

mysql_close(<Идентификатор>);

Функция mysql_pconnect() устанавливает постоянное соединение с сервером MySQL. При вызове функция проверяет наличие уже открытого постоянного соединения. Если соединение существует, функция использует это соединение, а не открывает новое. По завершению работы сценария постоянное соединение не закрывается.

Пример

```
<?php
$conn = @mysql_connect("localhost", "root",
   "123456");
if (!$conn) {
   echo "Не удалось установить соединение ";
} else {
   // Выполняем работу с базой данных
   mysql_close($conn); // Закрываем
   соединение
}
?>
```

Выбор базы данных

mysql_select_db(<Имя базы данных>, [<Идентификатор соединения>]);

Параметр <Идентификатор> можно не указывать. По умолчанию будет использоваться последнее открытое соединение.

```
<?php
$conn=@mysql_connect("localhost", "root", "");
If(!$conn)
    die("соединение с сервером не выполнено");
else
    echo("Соединение с сервером успешно </br>");
$db=mysql_select_db("warehouse",$conn);
if(!$db)
    exit("Подключение БД не выполнено");
else
    echo("Успешно подключились к БД");
?>
```

Пример 19.php

Выполнение запроса к базе данных

resource mysql_query(<SQL-запрос>, [<Идентификатор соединения>]);

SQL-запрос не требует указания в конце точки с запятой.

Функция возвращает идентификатор результата.

Получить все записи таблицы City позволяет следующий код:

```
$res = mysql_query('SELECT * FROM `City`');
```

Для того чтобы записи возвращались в нужной кодировке, следует после выбора базы данных указать один из запросов:

```
mysql_query('SET NAMES cp1251'); // Для кодировки Windows-1251
mysql_query('SET NAMES utf8'); // Для кодировки UTF-8
```

Обработка результата запроса

int mysql_num_rows(<Идентификатор результата>) - количество записей в результате.

Чтобы выяснить число записей, на которые воздействовали операторы INSERT, UPDATE, DELETE, следует использовать ф-ю

int mysql_affected_rows(соединение).

int mysql_num_fields(<Идентификатор результата>)

количество полей в результате.

string mysql_result (resource result, int row [, mixed field])

Извлекает поле из одной записи запроса.

Параметры

- *result* – то что возвращено ф-ей mysql_query()
- *row* - номер строки. Отсчет от 0.
- *field* - Имя или номер извлекаемого поля. Если опущено, то поле номер 0.
- Возвращает значение поля или false.

string mysql_field_name (resource result, int index)

Возвращает имя поля с номером index.

string mysql_field_type (resource result, int index)

Возвращает тип поля с указанным номером index

array mysql_fetch_array (resource result [, int result_type])

Возвращает массив полей записи и сдвигает курсор вперед.

Параметры:

result – значение, возвращенное **mysql_query()**.

result_type – Тип извлекаемого массива. Это константа, принимающая значения: MYSQL_ASSOC (обращение к полю только по имени), MYSQL_NUM – по индексу, и по умолчанию - MYSQL_BOTH – возвращает оба типа результата

```
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
    printf("ID: %s Name: %s", $row[0], $row[1]);
}
```

array mysql_fetch_row (resource result)

Возвращает массив полей очередной записи и сдвигает курсор. Обращение к элементам массива по индексу.

Параметр *result* – то что вернула ф-я **mysql_query()**.

array mysql_fetch_assoc (resource result)

Аналогично предыдущей ф-ии, но возвращает ассоциативный массив.

bool mysql_free_result(resource result)

освобождает память, занятую ресурсом выполненного запроса.

All associated result memory is automatically freed at the end of the script's execution.

Типовой пример использования функций для работы с БД:

```
<?php
$conn= @mysql_connect ('localhost','root','');
if(!$conn)die('Нет соединения');

If(!mysql_select_db(“warehouse”,$conn)){
    die(‘не выбрана БД’);
}
$res= mysql_query('select * from Org', $conn);
$n=mysql_num_fields ( $res );
while(mysql_fetch_assoc($res)){
    echo('<br>' );
    for($i=0;$i<$n;$i=$i+1){
     $fld= mysql_result($res, $i);
     echo($fld.'  ');
    }
}
mysql_free_result($res);
?>
```

Пример 11.php

string mysql_error ([resource link_identifier])

Возвращает текст, содержащий ошибку в последней выполненной mysql – функции

int mysql_insert_id ([resource link_identifier])

Возвращает последнее значение автоинкрементного поля, полученного последним оператором INSERT в соединении **link_identifier.**

bool mysql_data_seek (resource result, int row_number)

Перемещает указатель курсора result в позицию row_number (отсчет от 0).

object mysql_fetch_field (resource result [, int field_offset])

Parameters

result – Результат mysql_query.

field_offset – номер поля. Если field offset не указано, извлекаются данные следующего, ещё не извлеченного поля. Нумерация от 0.

Возвращает объект, содержащий характеристики поля. Объект имеет следующие свойства:

- name – имя поля
- table – имя таблицы, в которой находится поле
- def – значение по умолчанию
- max_length – макс. Длина поля
- not_null - 1 если неопределенные значения недопустимы
- primary_key - 1 если поле – первичный ключ
- unique_key - 1 если поле – уникальный ключ
- multiple_key - 1 если поле – неуникальный ключ
- numeric - 1 если поле является числовым
- blob - 1 если поле BLOB
- type – тип поля
- unsigned - 1 если поле не имеет знака
- zerofill - 1 если поле заполняется нулями

Примеры и функции для работы с БД MySql

Функция, приведенная ниже помещает имена полей и их типы в массив:

```
function FieldNamesAndTypes($rs){
$r=array();
$nFld=mysql_num_fields($rs);
for($i=0;$i<$nFld;$i++){
    $fld=mysql_fetch_field ($rs ,$i);
    $name=$fld->name;
    $type=$fld->type;
    $r[$i][$name]=$type;
}
return $r;
}
```

- См пример 12NamesAndTypes.php

```
function Locate($rs,$FieldName,$FieldValue){
// в наборе записей $rs, полученных mysql_query,
// найти запись, где $FieldName=$FieldValue (аналог
    Locate)
$f=-1;
$iRow=0;
while ($row = mysql_fetch_array($result,
    MYSQL_ASSOC)) {
    if($row[$FieldName]==$FieldValue){
            $f=$iRow;
            break;
     }
    $iRow++;
}
return $f;
}
```

```
function FindPrefix($rs,$FieldName,$Prefix){
// найти запись, в которой поле начинается с $Prefix
// (для инкрементного поиска)
$f=-1;
$L=strlen($Prefix);
$iRow=-1;
while ($row = mysql_fetch_array($result, MYSQL_ASSOC)){
    $ss=substr($row[$FieldName],0,$L);
    if($Prefix==$ss){
     $f=$iRow;
     break;
    }
    $iRow++;
}
return $f;
}
```

```
function MakeSqlInsert($TableName, $Fields, $Values){
// построить оператор insert для таблицы
    $TableName
// $Fields и $Values - массивы имен полей и их
    значений

$sqlIns="insert into ".$TableName."(";
$Comma="";
$n=count($Fields);
for($i=0; $i<$n;$i++){
    $sqlIns.=$Comma.$Fields[$i];
    $Comma=",";
}
$sqlIns.=") values(";
```

```
$Comma="";
$m=count($Values);
if($m!=$n){
   throw new Exception("MakeSQLInsert():число
   полей не равно числу значений");
}
for($i=0; $i<$m;$i++){
   $sqlIns.=$Comma.$Values[$i];  $Comma=",";
}
$sqlIns.=")";
return $sqlIns;
}  // См пример 15MakeSqlInsert.php
```

```
function MakeSqlUpdate($TableName, $Fields, $Values,
    $PrimName, $PrimValue){
// Построить оператор  u
//update <$TableName> set
//  <$Fields[0]>=<$Values[0]>
//  ……
// where <первичный ключ (<$PrimName>)=<$PrimValue>
$sqlUpd="update ".$TableName." set ";
$Comma="";
$n=count($Fields);
for($i=0; $i<$n;$i++){
   $sqlUpd.=$Comma." ".$Fields[$i]."=".$Values[$i];
   $Comma=",";
}
$sqlUpd.=" where ".$PrimName."=".$PrimValue;
return $sqlUpd;
} // См пример 16MakeSqlUpdate.php
```

```
function
    FieldByKey($conn,$table,$KeyName,$KeyVal,$FieldName,
    $AddQuot){
// найти значение поля $FieldName в таблице $table по
    полю
// $KeyName,  имеющем значение $KeyVal;
// $conn - соединение
// $AddQuot - bool (заключать ли $KeyVal в кавычки)
```

```
$sql="select top 1 ".$ValName." from ".$table."
   where ".$KeyName."=";
$quot=($AddQuot ? "'" : "");
$sql.=$quot.$KeyVal.$quot; //echo $sql;
$rs=mysql_query($sql, $conn);
if($row = mysql_fetch_array($rs, MYSQL_ASSOC))){
   return $row[$FieldName];
} else {
   return NULL;
}
}
```

```
function SelectXML($sql,$conn,$RowName){
// $sql – оператор select
// $conn – соединение
// $RowName -тег для одной строки результата

/* формирует xml в виде:
<SostNakls>
   <SostNakl SostNakl_ID="62" Tovar_ID="28"
   Amount="33.0"   Price="22.0000" TovarName="Baton" />
   <SostNakl SostNakl_ID="40" Tovar_ID="35"
   Amount="16.8"   Price="45.8000" TovarName="Ананас"
   />
   <SostNakl SostNakl_ID="41" Tovar_ID="10" Amount="4.0"
    Price="12.0000" TovarName="Апельсин" />
</SostNakls>
```

```
$rs=mysql_query($sql, $conn);
$n=mysql_num_fields($rs);
$s="";
$s.= "<".$RowName."s>\n";
while($row=mysql_fetch_array($rs, MYSQL_ASSOC)){
    $s.="<".$RowName." ";
    foreach ($row as $FieldName => $FieldValue)
     $r=" ".$FieldName. "=\"".$FieldValue. "\"“;
     $s.=$r;
    }
    $s.= " />\n";
}
$s.="</".$RowName."s>\n";
$s=iconv("Windows-1251", "UTF-8",$s);
return @s;
} //пример 17SelectXML.php рез-т см в “исх. коде» browser-а
```

ФУНКЦИИ

Синтаксис:

```
function <Имя функции> ([<Параметры>]) {
   <Тело функции>
   [return <Значение>]
}
```

Имена функций регистронезависимы

Внутри функции может находиться любой PHP код, в том числе, другие функции и определения классов.

Параметры функции могут быть необязательными. Необязательный пар-р должен иметь значение по умолчанию

```
function f_Sum($x, $y=2) {
   return ($x + $y);
}
$var1 = 5;
$var3 = f_Sum($var1); // Переменной $var3 будет присвоено значение 7
$var4 = f_Sum($var1, 5); // Переменной $var4 будет присвоено значение 10
```

Глобальные переменные — это переменные, объявленные вне функции.

В PHP глобальные переменные видны в любой части программы, кроме функций.

(пример 21.php)

```
<?php
   $a=5;
   function f($x){
    return $x+$a;
   }
   echo(f(6));
?>
```

Переменные, объявленные внутри функции являются локальными.

- Передача параметров – по значению. Возможна передача по ссылке: f(&$SomeVar) (*Пример 22.php)*

```
?php
   function f_Sum(&$x) {
    $number = 2;
    $x += $number;
   }
   $var1 = 10;
   echo 'Глобальная переменная $var1 вне
   функции = ' . $var1 . '<br>';
   f_Sum($var1); // Вызов функции
    echo 'Значение переменной $var1 после
   функции = ' . $var1 . '<br>';
?>
```

Внутри функции к глобальной переменной можно обратиться через суперглобальный массив $GLOBALS: (пример 23.php)

```
<?php
function f_Sum() {
   $number = 2;
   $GLOBALS['var1'] += $number;
}
$var1=5;
f_Sum();
echo($var1);
?>
```

Если внутри функции объявлена статическая переменная, то после завершения работы функции она не будет удалена и сохранит свое значение.

Выведем все четные числа от 1 до 100. *(пример 24.php)*

```
<?php
function f_Sum() {
    static $var;
    $number = 2;
    $var += $number;
    echo $var . "<br>\n";
}
for ($i=0; $i<50; $i++) f_Sum();
?>
```

Переменное число параметров в функции.

Функция func_get_args() возвращает массив аргументов.

func_get_arg(номер)- аргумент с заданным индексом

func_num_args() – возвращает число аргументов

(Пример 25.php)

Этой же функцией можно просуммировать произвольное количество переменных: (пример 26.php)

```
<?php
function f_Sum($var1, $var2) {
$sum = 0;
$count = func_num_args();
for ($i=0; $i<$count; $i++) {
    $sum += func_get_arg($i);
}
return $sum;
}

echo "<h1>".f_Sum(5, 6, 7, 20)."</h1>"; // Выведет 38
?>
```

Классы

- Создание объекта класса:

<Экземпляр класса> = new <Имя класса>
([<Параметры>]);

- При обращении к свойствам используется следующий формат:

<Экземпляр класса>-><Имя свойства без знака $>;

- Обращение к методам осуществляется аналогично:

<Экземпляр класса>-><Имя метода>([параметры]);

- Для удаления экземпляра класса используется функция unset():

unset(<Экземпляр класса>);

- Экземпляр класса можно также удалить, если ему присвоить значение null:

<Экземпляр класса> = null;

Определение класса

```
class <Имя класса> {
  // свойства и методы класса
}
```

Для создания переменной (свойства) внутри класса применяется следующий синтактис:

```
class <Имя класса> {
  <Область видимости> <Имя переменной со знаком $>;
  // например public @x;
}
```

Область видимости может принимать значения

public, private, protected;

Для ссылки на свойство или метод класса внутри метода класса используется указатель $this.

При обращении к свойству класса символ ‘$’ опускается:

```
private $x;
//……….
$this->x=8;
```

Метод внутри класса создается:

```
class <Имя класса> {
  [<Область видимости>] function <Имя функции>([Параметры]) {
    // Тело функции
  }
}
```

Пример класса:

```
<?php
   class SimpleClass
   {
      // объявление члена класса
      public $var = 'a default value';

      // объявление метода
      public function displayVar() {
          echo $this->var;
      }
   }
?>
```

Объект класса создаётся оператором new:

```
$instance = new SimpleClass();
```

Производные классы

Класс может наследовать свойства и методы родительского класса (одного). Наследуемые методы могут быть переопределены. Ключевое слово «final» запрещает дальнейшее переопределение метода в производных классах.

```
final public function moreTesting() {
    echo "BaseClass::moreTesting() called\n";
}
```

Пример определения производного класса:

```
<?php
  class ExtendClass extends SimpleClass
  {
    // Переопределяем метод базового класса
    function displayVar()
    {
       echo "Extending class\n";
       parent::displayVar();
    }
  }

  $extended = new ExtendClass();
  $extended->displayVar();
?>
```

Если требуется обратиться к переопределённому методу базового класса, то употребляется уточнение `parent::`

PHP5 позволяет явно объявить конструктор.

```
class BaseClass {
    function __construct() {
        print "В конструкторе базового класса\n";
    }
}
```

Для совместимости с PHP4 можно воспользоваться конструктором, совпадающим с именем класса.

Родительский конструктор не вызывается неявно в дочернем классе. Для его вызова следует вызвать **parent::__construct()** в конструкторе дочернего класса.

Пример:

```
<?php
    class BaseClass {
      function __construct() {
          print " В конструкторе базового класса\n";
      }
    }

    class SubClass extends BaseClass {
      function __construct() {
          parent::__construct();
          print "В конструкторе производного класса\n";
      }
    }

    $obj = new BaseClass();
    $obj = new SubClass();
?>
```

Деструкторы.

Название деструктора по умолчанию:__destruct.

Пример:

```
<?php
    class MyClass {
      function __construct() {
          print "В конструкторе\n";
          $this->name = "MyClass";
      }

      function __destruct() {
          print "Уничтожаем объект типа " . $this->name .
    "\n";
      }
    }

    $obj = new MyClass();
?>
```

Деструктор автоматически вызывается, когда уничтожены все ссылки на объект или когда объект уничтожается явно. Деструктор вызывается и при закрытии скрипта.

Деструктор базового класса не вызывается неявно. Следует писать: **parent::__destruct().**

Свойства видимости public, private, protected аналогичны C++. Если не указано, то public является умолчанием. Оператор разрешения области видимости «::» и понятие «static»также аналогичны C++.

Помимо parent:: есть ещё self::

Если метод не определён, то класс является абстрактным:

```
abstract class AbstractClass  {
    // Производный класс должен
  определить
    //  эти методы
   abstract protected function getValue();
   abstract protected function prefixValue($prefix);
   ……..
```

Cookie

Web-браузеры позволяют сохранять небольшой объем информации в специальном текстовом файле на компьютере пользователя. Такая информация называется cookie.

Cookie отсылаются браузеру вместе с headers. Возможность использования cookie можно отключить в настройках Web-браузера. К cookie можно обратиться как на рабочей станции средствами Javascript, так и на стороне сервера (PHP).

Javascript

Для проверки возможности использования cookies следует использовать свойство cookieEnabled объекта navigator.

```
if (navigator.cookieEnabled) {
    window.alert("Использование cookies разрешено");
}
```

Запись cookies производится путем присвоения значения свойству cookie объекта document в следующем формате:

```
document.cookie = "<Имя>=<Значение>;
    [expires=<Дата>;] [domain=<Имя домена>;]
    [path=<Путь>;] [secure;]";
```

Параметры

<Имя>=<Значение>

задает имя сохраняемой переменной и ее значение. Это единственный обязательный параметр. Если не задан параметр expires, то по истечении текущего сеанса работы Web-браузера cookies будут автоматически удалены;

expires указывает дату удаления cookies в следующем формате:

Thu, 01 Jan 1970 00:00:01 GMT

Получить дату в этом формате можно с помощью методов setTime() и toGMTString() класса Date.

Методу setTime() нужно передать текущее время в миллисекундах плюс время хранения cookies в миллисекундах. Текущее время можно получить с помощью метода getTime().

```
var d = new Date();
d.setTime(d.getTime()+3600000); // Задан 1 час
var End_Date = d.toGMTString(); // Дата удаления cookies
```

- domain=<Имя домена> задает доменную часть URL-адреса, для которой действует данный cookies;
- path=<Путь> задает часть URL-адреса, определяющую путь к документам, для которых действует данный cookies.

Считывание cookies производится с помощью свойства cookie объекта document:

```
var cookies = document.cookie;
```

Переменная cookies будет содержать строку, в которой перечислены все установленные пары имя=значение через точку с запятой:

```
"имя1=значение1; имя2=значение2"
```

Для удаления cookies следует установить cookies с прошедшей датой.

В качестве примера рассмотрим ситуацию, когда пользователь регистрируется, и мы запоминаем его имя и пароль в cookie.

При последующих посещениях, пока не достигнуто время expire, мы больше не будем запрашивать его данные.

Для совместимости закодируем введенные данные с помощью метода escape(), а при выводе раскодируем их с помощью метода unescape(). Это позволяет безопасно сохранять значения, введенные кириллицей.

PHP

Cookies может быть установлен ф-ей **setcookie()**.

Cookies является частью HTTP заголовка (header), что означает, что функция **setcookie()** должна быть вызвана до того как что-либо будет отправлено браузеру, включая теги <html> и <head> и даже пробелы. Если это требование нарушено, то **setcookie()** возвращает false, а в случае успеха – true.

Пример 20.php

```
bool setcookie ( string name [, string value [, int expire [,
    string path [, string domain [, bool secure]]]]] )
```

Смысл параметров аналогичен рассмотренным для Javascript. В параметре expire время задаётся в секундах.

Пример:

```
<?php
$value = 'something';
setcookie("TestCookie", $value);
setcookie("TestCookie", $value, time()+3600); /* expire
    in 1 hour */
setcookie("TestCookie", $value, time()+3600,
    "/~rasmus/", ".example.com", 1);
?>
```

Переменные из cookie доступны через глобальные массивы $_COOKIE или $HTTP_COOKIE_VARS.

Например:

```
$_COOKIE["TestCookie"];
```

Сессии

- Протокол HTTP является протоколом "без сохранения состояния”
- Сессии и cookies предназначены для хранения сведений о пользователях при переходах между несколькими страницами.
- При использовании сессий данные сохраняются во временных файлах на сервере.
- Сессия (сеанс) по сути, представляет собой группу переменных, которые, в отличие от обычных переменных,

При работе с сессиями различают следующие этапы:

- открытие сессии
- регистрация переменных сессии и их использование
- закрытие сессии

Открытие сессии

- Самый простой способ открытия сессии заключается в использовании функции **session_start**, которая вызывается в начале PHP-сценария:

Синтаксис: **session_start*();***

Эта функция проверяет, существует ли идентификатор сессии, и, если нет, то создает его.

Если идентификатор текущей сессии уже существует, то загружаются зарегистрированные переменные

После инициализации сессии появляется возможность сохранять информацию в суперглобальном массиве **$_SESSION**.

Пусть имеется файл index.php в котором в массиве **$_SESSION** сохраняются переменная и массив.

```
<?php
session_start(); //Инициируем сессию
$_SESSION['name'] = "value"; // Помещаем значение в сессию
$arr = array("first", "second", "third");
$_SESSION['arr'] = $arr; // Помещаем массив в сессию
// Выводим ссылку на другую страницу
 echo "<a href='other.php'>другая страница</a>";
?>
```

На страницах, где происходит вызов функции **session_start()**, значения данных переменных можно извлечь из глобального массива **$_SESSION**. Ниже приводится содержимое страницы other.php, где извлекаются данные, ранее помещенные на странице index.php.

```
<?php // Инициируем сессию session_start();
// Выводим содержимое массива $_SESSION
echo "<pre>";
print_r($_SESSION);
echo "</pre>";
?>
```

Этот пример находится в Example2. Запустить index.php

Закрытие сессии

После завершения работы с сессией сначала нужно разрегистрировать все переменные сессии, а затем вызвать функцию **unset()**:

Синтаксис:

```
unset($_SESSION["username"]);
```

Можно сделать это проще функцией bool session_destroy ( ).

Глобальные объекты

Массив $GLOBALS

Содержит ссылки на все доступные глобальные переменные. Ключи этого массива суть имена глобальных переменных.

- $_SERVER
- $_SERVER – массив, содержащий headers, paths, местоположение текущего скрипта. Создаётся web-сервером, каждый по своему. Нет гарантии, что web-сервер предоставит всю нужную информацию в массиве.
- **'PHP_SELF'**
- $_SERVER['PHP_SELF'] – текущий скрипт
- **'argv' –** Массив аргументов, передаваемых Если скрипт вызывается методом GET, то содержит строку с аргуметами вроде “0=>w=7&s=asd
- **'argc'**
- **'SERVER_NAME' –** имя сервера
- ……..и так далее

См. пример 13.php

- $_FILES

Информация о загружаемых файлах

- $_ENV - оборудование, окружение

(показать пример, изменив 13.php)

- $_REQUEST

Переменные, получаемые через GET, POST и COOKIE.

- $_SESSION

Функции

Синтаксис ф-ии как везде:

```
function <Имя функции> ([<Параметры>]) {
   <Тело функции>
   [return <Значение>]
}
```

Имена ф-й регистронезависимы

Внутри функции может находиться любой PHP код, в том числе, другие функции и определения классов.

Параметры функции могут быть необязательными.

Необязательный параметр должен иметь значение по умолчанию

```
function f_Sum($x, $y=2) {
    return ($x + $y);
}

$var1 = 5;
$var3 = f_Sum($var1); // Переменной $var3 будет присвоено значение 7
$var4 = f_Sum($var1, 5); // Переменной $var4 будет присвоено значение 10
```

Глобальные переменные — это переменные, объявленные вне функции. В PHP глобальные переменные видны в любой части программы, кроме функций. Пример 21.php:

```
<?php
   $a=5;
   function f($x){
    return $x+$a;
   }
   echo(f(6));
?>
```

Будет выведено 6.

Переменные, объявленные внутри функции являются локальными.

Внутри функции к глобальной переменной можно обратиться через суперглобальный массив $GLOBALS:

```
<?php
   function Summa() {
    $number = 2;
    $GLOBALS['var1'] += $number;
   }

   $var1=5;
   Summa();
   echo($var1);
?>
```

Статическая переменная после завершения работы функции не будет удалена и сохранит свое значение.

Выведем все четные числа от 1 до 100.

```
<?php
function f_Sum() {
    static $var;
    $number = 2;
    $var += $number;
    echo $var . "<br>\n";
}
for ($i=0; $i<50; $i++) f_Sum();
?>
```

(пример 24.php)

Переменное число параметров в функции.

- func_get_args() возвращает массив аргументов.
- func_get_arg(номер)- аргумент с заданным индексом
- func_num_args() – возвращает число параметров

```
<?php
   function f_Sum($var1, $var2) {
    return func_get_arg(0)+func_get_arg(1);
   }
   echo "<h1>".f_Sum(5, 6)."</h1>"; // =11
?>
```

Пример 25.php

Этой же функцией можно просуммировать произвольное количество переменных: (пример 26.php)

```
<?php
function f_Sum($var1, $var2) {
    $sum = 0;
    $count = func_num_args();
    for ($i=0; $i<$count; $i++) {
     $sum += func_get_arg($i);
    }
return $sum;
}

echo "<h1>".f_Sum(5, 6, 7, 20)."</h1>"; // Выведет 38
?>
```

Классы

Создание объекта класса:

<Экземпляр класса> = new <Имя класса>
([<Параметры>]);

При обращении к свойствам используется следующий формат:

<Экземпляр класса> □ <Имя свойства без знака $>;

Обращение к методам осуществляется аналогично:

<Экземпляр класса>□<Имя метода>([параметры]);

Для удаления экземпляра класса используется функция unset():

unset(<Экземпляр класса>);

Экземпляр класса можно также удалить, если ему присвоить значение null:

<Экземпляр класса> = null;

Определение класса

```
class <Имя класса> {
// свойства и методы класса
}
```

Для создания переменной (свойства) внутри класса применяется следующий синтактис:

```
class <Имя класса> {
<Область видимости> <Имя переменной со знаком $>;
// например public @x;
}
```

Область видимости может принимать значения public, private, protected;

Для ссылки на свойство или метод класса внутри метода класса используется указатель $this.

При обращении к свойству класса символ ‘$’ опускается:

```
private $x;
//……..
$this->x=8;
```

Метод внутри класса создается:

```
class <Имя класса> {
    [<Область видимости>] function <Имя функции>
    ([Параметры]) {
    // Тело функции
}
```

Пример класса:

```php
<?php
    class SimpleClass
    {
        // объявление члена класса
        public $var = 'значение по умолчанию';

        // объявление метода
        public function displayVar() {
            echo $this->var;
        }
    }
?>
```

Объект класса создаётся оператором new:

```php
$instance = new SimpleClass();
```

Производные классы

Класс может наследовать свойства и методы родительского класса (одного). Наследуемые методы могут быть переопределены.

Пример определения производного класса:

```
<?php
   class ExtendClass extends SimpleClass
   {
      // Переопределяем метод базового класса
      function displayVar()
      {
         echo "Extending class\n";
         parent::displayVar();
      }
   }

   $extended = new ExtendClass();
   $extended->displayVar();
?>
```

Ключевое словр «final» запрещает дальнейшее переопределение метода в производных классах.

```
final public function moreTesting() {
        echo "BaseClass::moreTesting() called\n";
}
```

Если требуется обратиться к переопределённому методу базового класса, то употребляется уточнение parent:: .

PHP5 позволяет явно объявить конструктор.

```
class BaseClass {
    function __construct() {
        print " В конструкторе базового класса\n";
    }
}
```

Для совместимости с PHP4 можно воспользоваться конструктором, совпадающим с именем класса.

Родительский конструктор не вызывается неявно в дочернем классе. Для его вызова следует вызвать **parent::__construct()** в конструкторе дочернего класса.

Пример:

```
<?php
    class BaseClass {
      function __construct() {
         print " В конструкторе базового класса\n";
      }
}


class SubClass extends BaseClass {
      function __construct() {
         parent::__construct();
         print " В конструкторе производного класса\n";
      }


}

    $obj = new BaseClass();
    $obj = new SubClass();
?>
```

Аналогично C++ в PHP существуют деструкторы.
Название деструктора по умолчанию:__destruct.

Пример:

```
<?php
    class MyClass {
      function __construct() {
          print " В конструкторе\n";
          $this->name = "MyClass";
      }

      function __destruct() {
          print " Уничтожаем объект типа " . $this->name .
    "\n";
      }
}

    $obj = new MyClass();
?>
```

Деструктор автоматически вызывается, когда уничтожены все ссылки на него или когда объект уничтожается явно. Деструктор вызывается и при закрытии скрипта.

Деструктор базового класса не вызывается неявно. Следует писать: **parent::__destruct().**

Свойства видимости public, private, protected аналогичны C++. Если не указано, то public является умолчанием. Оператор разрешения области видимости «::» и понятие «static»также аналогичны C++.

Помимо parent:: есть ещё self::.

Если метод не определён, то класс является абстрактным:

```
abstract class AbstractClass
   {
     // Производный класс должен определить эти методы
   abstract protected function getValue();
   abstract protected function prefixValue($prefix);
   ……..
```

Все функции и классы PHP глобальны. Их можно вызвать извне функции, даже если они определены внутри неё.

PHP не поддерживает перегрузку функций и операторов.

```php
<?php
   class MyClass { //пример 28.php
      public $var1 = 'value 1';
      public $var2 = 'value 2';
      public $var3 = 'value 3';

      protected $protected = 'protected var';
      private   $private   = 'private var';

        function iterateVisible() {
        echo "    MyClass::iterateVisible:</br>";
        foreach($this as $key => $value) {
          print "$key => $value"."</br>";
        }
     }
    }
```

```
$class = new MyClass();

 foreach($class as $key => $value) {
    print "$key => $value"."</br>";
 }
 echo "</br></br>";

 $class->iterateVisible();

?>
```

Некоторые функции

Сериализация

string serialize(mixed)

возвращает строку, содержащую поток байтов, являющихся представлением любого объекта PHP. Функции не сохраняются и не восстанавливаются. Ф-я serialize() может работать с любыми типами, кроме resource-type

mixed unserialize (string str)

восстанавливает этот объект по строке. Для того чтобы было возможно unserialize() объект, класс объекта должен быть определён.

Это означает, что если вы имеете объект $a класса A на стр. page1.php и serialize его, то вы получите строку, ссылающуюся на класс A и содержащие переменные в объекте $a. Если Вы хотите восстановить этот объект на стр. page2.php, то определение класса A должно быть доступно на стр. page2.php. Определение класса можно сохранить в include файле.

Перед выполнением сериализации объекта, PHP will попытается вызвать функцию __sleep(). Это делается для того, чтобы объект успел выполнить необходимые действия пред сериализацией. Аналогично, когда объект восстанавливается с помощь. unserialize() вызывается ф-я __wakeup().