PHP

PHP препроцессор гипертекста

Ф.Филиппов, доцент СПбГУТ им. проф. М.А.Бонч-Бруевича

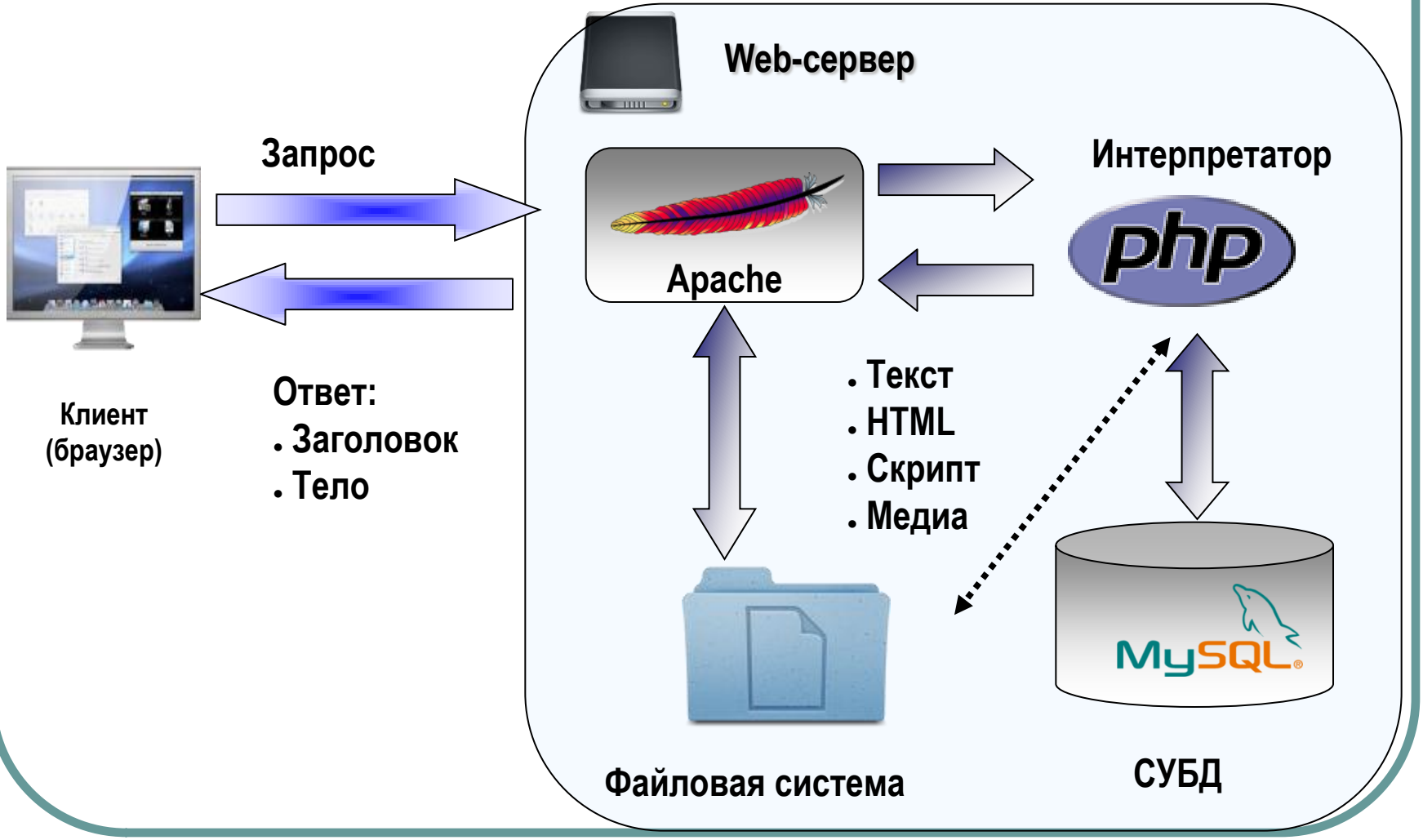
Разделы:

1. ***Клиент-серверные взаимодействия***
2. ***Типы данных***
3. ***Операции и операторы***
4. ***Управляющие конструкции***
5. ***Работа с файлами***
6. ***Регулярные выражения***
7. ***Функции***
8. ***Объекты и классы***
9. ***Практические примеры***

Ф.Филиппов, доцент СПбГУТ им. проф. М.А.Бонч-Бруевича

1 Клиент-серверные взаимодействия

Клиент - WEB-сервер



WEB-серверы

В качестве примеров *web-серверов* можно привести *сервер:*

- ***Apache*** группы Apache
- *IIS* (Internet Information Server) компании Microsoft
- *SunOne* фирмы Sun Microsystems
- *WebLogic* фирмы BEA Systems
- *IAS* (Inprise Application Server) фирмы Borland
- *WebSphere* фирмы IBM
- *OAS* (Oracle Application Server)

Передача данных

HTTP (HyperText Transfer Protocol, *протокол передачи гипертекста*) – это протокол прикладного уровня, разработанный для обмена гипертекстовой информацией в Internet.

Запрос клиента содержит:

- метод доступа;
- адрес URI;
- версию протокола;
- сообщение с информацией о типе передаваемых данных, информацией о *клиенте*, пославшем запрос, и, возможно, с содержательной частью (телом) сообщения.

Ответ сервера содержит:

- строку состояния, в которую входит версия протокола и код возврата (успех или ошибка);
- сообщение, в которое входит информация *сервера*, информация о содержании сообщения и тело сообщения.

Формы запроса клиента

Простой запрос содержит метод доступа и адрес ресурса:

<Простой запрос> := <Метод> <символ пробел>
<Запрашиваемый-URI> <символ новой строки>

GET http://phpbook.info/

Полный запрос содержит строку состояния, несколько заголовков (заголовок запроса, общий заголовок или заголовок содержания) и, возможно, тело запроса.

Формально общий вид *полного запроса* можно записать так:

<Полный запрос> := <Строка Состояния>
(<Общий заголовок>|<Заголовок запроса>|
<Заголовок содержания>) <символ новой строки>
[<содержание запроса>]

POST http://phpbook.info/cgi-bin/test HTTP/1.0

Методы протокола HTTP

Протокол HTTP поддерживает достаточно много методов, но реально используются только три: *POST*, *GET* и *HEAD*.

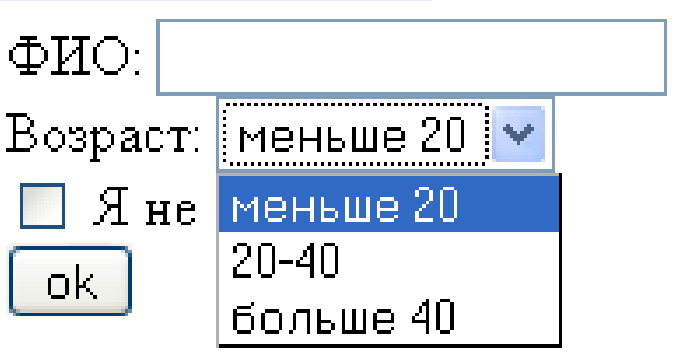
Метод *GET* позволяет получить данные, идентифицированные с помощью URI в запросе ресурса. Если URI указывает на программу, то возвращается результат работы программы (а не ее текст). Информация, необходимая для обработки запроса, встраивается в сам запрос (в строку статуса).

Метод *HEAD* аналогичен методу *GET*, только не возвращает тело ресурса и не имеет условного аналога.

Метод *POST* разработан для передачи на *сервер* такой информации, как данные о клиенте, новостные и почтовые сообщения, данные для добавления в базу данных, т.е. для передачи информации большого объема и достаточно важной.

Использование HTML-форм

```
<form method="get" action="result.php">
        ФИО:
        <input type="text" name="fio"><br>
        Возраст:
        <select name="age">
                <option value="1">меньше 20</option>
                <option value="2">20-40</option>
                <option value="3">больше 40</option>
        </select><br>
        <input type="checkbox" name="human"> Я не "бот"!<br>
        <input type="submit" name="submit" value="ok">
</form>
```



Обработка HTML-формы

(скрипт *result.php*)

```
<?php

    if (!$_REQUEST["human"]) {
        echo "Посторонним вход воспрещён!!!\n";
        exit();
    }
    echo "Уважаемый, $_REQUEST[fio]!\n";
    if ($_REQUEST["age"]==1) {
        echo «Замечательный возраст...\n";
    }
    elseif ($_REQUEST["age"]==2) {
        echo "А оно вам надо?\n";
    }
    elseif ($_REQUEST["age"]==3) {
        echo "Делитесь опытом!\n";
    }
```

Обработка HTML-формы
(скрипт *result.php* – другой вариант)

```
<?php

    if (!$_REQUEST["human"]) {
        echo "Посторонним вход воспрещён!!!\n";
        die; // аналог функции exit()
    }
    echo "Уважаемый, $_REQUEST[fio]!\n";

    $message = array(
        1=>«Замечательный возраст...\n“,
        2=>"А оно вам надо?\n“,
        3=>"Делитесь опытом!\n"
    );

    echo $message[$_REQUEST["age"]];
```

Переменные окружения

При отправке данных на *сервер* любым методом передаются не только сами данные, введенные пользователем, но и ряд переменных, называемых *переменными окружения*, характеризующих *клиента*, историю его работы, пути к файлам и т.п. Вот некоторые из *переменных окружения*:

REMOTE_ADDR – IP-адрес хоста (компьютера), отправляющего запрос;
REMOTE_HOST – имя хоста, с которого отправлен запрос;
HTTP_REFERER – адрес страницы, ссылающейся на текущий скрипт;
REQUEST_METHOD – метод, который был использован при отправке запроса;
QUERY_STRING – информация, находящаяся в *URL* после знака вопроса;
SCRIPT_NAME – виртуальный путь к программе, которая должна выполняться;
HTTP_USER_AGENT – информация о браузере, который использует *клиент*

Доступ к переданным данным

до версии PHP 4.1.0

До версии PHP 4.1.0
first_name=Вася - передавалась
$first_name - появлялась внутри скрипта со значением Вася.

Для определения метода передачи данных использовались *ассоциативные массивы* *$HTTP_POST_VARS* и *$HTTP_GET_VARS*, ключами которых являлись имена переданных переменных, а значениями – соответственно значения этих переменных.

Таким образом:
если пара first_name=Вася передана методом *GET*, то $HTTP_GET_VARS["first_name"]="Вася".
если пара first_name=Вася передана методом *POST*, то $HTTP_POST_VARS["first_name"]="Вася".

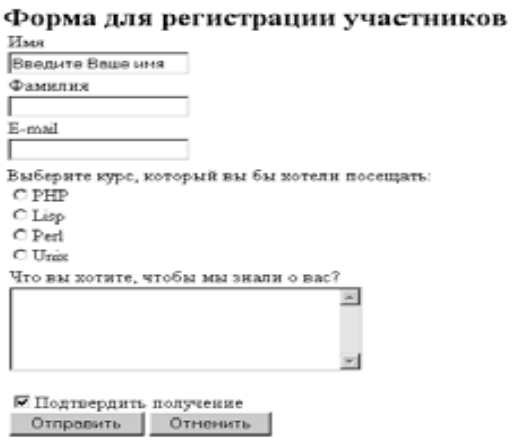
Доступ к переданным данным

версия PHP 4.1.0 и далее

Начиная с PHP 4.1.0

Для обращения к переменным, переданным с помощью HTTP-запросов, используется специальный массив – *$_REQUEST*. Этот массив содержит данные, переданные методами *POST* и *GET*, а также с помощью *HTTP* cookies.

Это *суперглобальный ассоциативный массив*, т.е. его значения можно получить в любом месте программы, используя в качестве ключа имя соответствующей переменной (элемента формы).




```
<?php
$str = "Здравствуйте,
    ".$_REQUEST["first_name"]. «
    ".$_REQUEST["last_name"]."! <br>";
$str .="Вы выбрали для изучения курс по
    ".$_REQUEST["kurs"];
echo $str;
?>
```

Доступ к переданным данным

После введения массива *$_REQUEST* массив
$HTTP_POST_VARS был переименован в *$_POST*
$HTTP_GET_VARS - *$_GET*

$_POST и *$_GET* стали *суперглобальными*, т.е. доступными напрямую и внутри функций и методов.

```
<?php
$str = "Здравствуйте,
    ".$_POST ["first_name"]."
    ".$_POST ["last_name"] ."! <br>";
$str .= "Вы выбрали для изучения курс по ".$_POST["kurs"];
echo $str;
?>
```

Здравствуйте, Вася Петров!
Вы выбрали для изучения курс по PHP

Функция *getenv()*

Возвращает значение *переменной окружения*, имя которой передано ей в качестве параметра.

```
<?
getenv("REQUEST_METHOD");
    // возвратит использованный метод
echo getenv ("REMOTE_ADDR");
      // выведет IP-адрес пользователя, пославшего запрос
getenv("QUERY_STRING");
      // возвратит все, что записано в URL после знака вопроса
getenv("HTTP_REFERER");
      // возвратит адрес страницы, с которой были посланы данные
?>
```

2

Синтаксис языка PHP (типы данных)

Переменные и константы

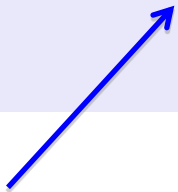
```
<?php echo "Hello, world!" ; ?>
```

Переменные:

```
$var_name = $value;
…
$var1 = 1;
$var2 = 1.0;
$var3 = “str”;
$var4 = true;
$var5 = $object;
```

Константы:

```
define($name, $value, $case_sen);
…
define(“pi”, 3.14, true);
define(“cty”, “St-Petersburg”, false);
…
```



с учетом регистра

Типы данных

PHP поддерживает ***восемь*** простых *типов данных*.

Четыре скалярных *типа*:

- *boolean (логический)*;
- *integer (целый)*;
- *float (с плавающей точкой)*;
- *string (строковый)*.

Два смешанных *типа*:

- *array (массив)*;
- *object (объект)*.

Два специальных *типа*:

- *resource (ресурс)*;
- *NULL*.

Тип *integer*

```
<?PHP
$a = 1234; // десятичное число
$b = -123; // отрицательное число
$c = 0123; // восьмеричное число
$d = 0x1A; // шестнадцатеричное число
?>
```

Тип *string*

Строка в PHP может быть определена тремя различными *способами*:

- с помощью ***одинарных кавычек***;
- с помощью ***двойных кавычек***;
- *с использованием* ***heredoc-синтаксиса***.

Одинарные кавычки:

```
<?PHP
echo ‘Это простая строка';
echo 'Это: \n - не вставит новую строку';
echo 'Переменная $expand также не подставится';
?>
```

Тип *string*

Двойные кавычки:

```
<?PHP
echo "Это: \n - вставит новую строку";
echo "Переменная $expand - подставится";
?>
```

Heredoc синтаксис:

```
<?PHP
$name = 'Вася';
echo <<< HEREDOC
Меня зовут "$name".
HEREDOC;                    // это выведет "Меня зовут Вася“
?>
```

Тип *string* в двойных кавычках

Внутри двойных кавычек:

\n – перевод строки (LF),
\r – возврат каретки (CR),
\t – табуляция,
\\ - обратный слеш,
\$ - доллар,
\" – двойная кавычка,
\000 - \377: символы с ASCII-кодами 0 – 377_8,
\x00 - \xFF: символы с ASCII-кодами 0 – FF_{16}

Тип *array*

```
<?PHP
$array = Array(25, -7 , 93, -567, 90);
?>

<?PHP
$food = Array('Сыр','Колбаса',‘Апельсины'); // Определяем массив с
    тремя элементами

echo $food[0]; // Обращение к нулевому элементу - выведет Сыр.

$food[1] = 'Мороженое'; // Присвоение первому элементу - заменяем
    колбасу на мороженое

echo $food[1]; // Выведет Мороженое.
?>
```

Ассоциативные массивы

```
<?PHP

$arr = Array(’abc'=>’мяч' , 'key2'=>'val2'); // Определяем массив с двумя
    элементами

echo $arr[’abc']; // Обращение к элементу с ключом abc. Выведет мяч

echo $arr[0]; // Обращение к нулевому элементу. Ошибка. Нет такого
    элемента

$arr['key2'] = 'Мороженое'; // Присвоение элементу с ключом key2 нового
    значения.

echo $arr['key2']; // Выведет Мороженое.

?>
```

Многомерные массивы

```
<?PHP
$ARRAY = Array                          // Числовой. Ключи 0,1 и 2
(
 Array('Сыр','Колбаса') ,               // Числовой. Ключами являются 0 и 1
 Array('key'=>'val' , 'key2'=>'val2') , // Ассоциативный. Ключи key и key2
 Array('key3'=>'val3', 'Чипсы')         // Ассоциативный. Ключи key3 и 0
);

echo $ARRAY[2]['key3'];                 // Обращение к val3


$arr = array("somearray" => array(6 => 5, 13 => 9, "a" => 42));
echo $arr["somearray"][6];               // 5
echo $arr["somearray"][13];              // 9
echo $arr["somearray"]["a"];             // 42
?>
```

Тип *object*

```
<?php
// создаем класс людей
class Person
{
// определяем свойства (атрибуты)
var $eye_color;
var $weight;
// определяем метод, который обучает человека PHP
    function know_php()
        {
            echo "Теперь я знаю PHP";
        }
    }
$bob = new Person;          // создаем объект bob класса человек
$bob->eye_color = ‘blue’;   // задаем значение свойства eye_color
$bob->weight = 86.5;        // задаем значение свойства weight
$bob -> know_php();         // обучаем его PHP
?>
```

Тип *resource*

Ресурс – это специальная *переменная*, содержащая ссылку на внешний *ресурс* (например, соединение с базой данных).

Ресурсы создаются и используются специальными функциями (например, mysql_connect(), pdf_new() и т.п.)

Тип *NULL*

Специальное значение *NULL* говорит о том, что *переменная* не имеет значения.

Переменная считается *NULL*, если:

- ей была присвоена *константа NULL* ($var = NULL);
- ей еще не было присвоено какое-либо значение;
- она была удалена с помощью *unset*().

Существует только одно значение *типа NULL* – регистронезависимое ключевое слово NULL.

Работа с типами

- gettype() – определяет тип параметра
- is_float(), is_int(), is_string(), is_object(), is_array() – определяют является ли параметр переменной данного типа
- intval() - возвращает аргумент в виде целого числа integer
- floatval() - возвращает аргумент в виде дробного числа float
- strval() - возвращает аргумент в виде строки string
- settype() - превращает первый аргумент в указанный во втором аргументе тип

3

Синтаксис языка PHP (операции и операторы)

Арифметические операции

- -$a Отрицание - смена знака $a.
- $a + $b Сложение - сумма $a и $b.
- $a - $b Вычитание - разность $a и $b.
- $a * $b Умножение - произведение $a и $b.
- $a / $b Деление - частное от деления $a на $b.
- $a % $b Деление по модулю - целочисленный остаток от деления $a на $b.

Операции ++ и --

- ++$a Префиксный инкремент - увеличивает $a на и возвращает значение $a.
- $a++ Постфиксный инкремент - возвращает значение $a, а затем увеличивает $a на 1.
- --$a Префиксный декремент - уменьшает $a на 1 и возвращает значение $a.
- $a-- Постфиксный декремент - возвращает значение $a, а затем уменьшает $a на 1

Строковые операторы

```
<?PHP

$a = "Hello ";
$b = $a . "World!"; // $b содержит строку "Hello World!"

$a = "Hello ";
$a .= "World!";     // $a содержит строку "Hello World!"

?>
```

Операторы работы с массивами

- $a + $b Объединение массива $a и массива $b.
- $a == $b Равно TRUE в случае, если $a и $b содержат одни и те же элементы.
- $a === $b Тождественно равно TRUE в случае, если $a и $b содержат одни и те же элементы в том же самом порядке.
- $a != $b Не равно TRUE если массив $a не равен массиву $b.
- $a <> $b Не равно TRUE если массив $a не равен массиву $b.
- $a !== $b Тождественно не равно TRUE если массив $a не равен тождественно массиву $b.

Операторы сравнения

Оператор	Название	Описание
==	Равенство	$a == $b
===	Эквивалентность	$a === $b
!=	Неравенство	$a != $b
<>	Неравенство	$a <> $b
!==	Неэквивалентность	$a !== $b
<	Меньше	$a < $b
>	Больше	$a > $b
<=	Меньше или равно	$a <= $b
>=	Больше или равно	$a >= $b

Логические операторы

Оператор	Название	Пример
and	И	$a and $b
&&	И	$a && $b
or	Или	$a or $b
||	Или	$a || $b
xor	Исключающее или	$a xor $b
!	Инверсия (NOT)	! $a

4

Синтаксис языка PHP (управляющие конструкции)

if –elseIf-else

if(логическое выражение)
  оператор1;
elseIf(логическое выражение)
  оператор2;
else
  оператор3;

Примеры использования *if*

```
if($name == "sasha")  echo "Привет, Саша!";
                      else    echo "Вы кто?";
```

```
<?php
        if($s == "yes") :
?>
<h1>Поздравляем!</h1>
<?php
        endif;
?>
```

PHP

HTML

PHP

switch-case

```
switch(выражение)
{
   case знач1: команда; [break;]
   …
   case значN: команда; [break;]
}
```

Примеры использования *switch-case*

```
switch($what) {
        case 33:        echo "what is 33"; break;
        case "sun":     echo "what is sun"; break;
        default:         echo "what is unknown"; }
```

```
switch ($var) {
        case $a: echo "значение var совпадает с a"; break;
        case $b: echo "значение var совпадает с b"; break; }
```

switch-case u if –elseIf

```
switch ($a) {
  case "run":
            func1();
            break;
  case "stop":
            func2();
            break;
  case "pause":
            func3();
            break;
}
```

```
if ($a=="run") {
            func1();
}
elseif ($a=="stop") {
            func2();
}
elseif ($a=="pause") {
            func3();
}
```

Циклы *while*, *do-while* и *for*

while(логическое выражение)
  {тело;}

do {тело;}
while(условие) ;

for(инициализация; условие; итерация)
  {тело;}

Примеры использования *while*

```
$a=0;
while($a <100) {
        $a++;
        echo $a; }
```

```
$a=0;
while($a <100):
        $a++;
        echo $a;
        endwhile;
```

Примеры использования *for*

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

```
for($i = 10; $i > 0; $i--):
        echo ($i);
endfor;
```

Цикл *foreach*

```
<?php
$names["Иванов"] = "Андрей";
$names["Петров"] = "Борис";
$names["Волков"] = "Сергей";
$names["Макаров"] = "Федор";
foreach ($names as $key => $value) {
echo "<b>$value $key</b><br>";
}
?>
```

Андрей Иванов
Борис Петров
Сергей Волков
Федор Макаров

5

Синтаксис языка PHP (работа с файлами)

Открытие файла *fopen()*

```
$fp = fopen('counter.txt', 'r');
```

r – открытие файла только для чтения.

r+ - открытие файла одновременно на чтение и запись.

w – создание нового пустого файла. Если на момент вызова уже существует такой файл, то он уничтожается.

w+ - аналогичен r+, только если на момент вызова файла такой существует, его содержимое удаляется.

a – открывает существующий файл в режиме записи, при этом указатель сдвигается на последний байт файла (на конец файла).

a+ - открывает файл в режиме чтения и записи при этом указатель сдвигается на последний байт файла (на конец файла). Содержимое файла не удаляется.

Запись в файл *fwrite()*

```
<?php
 $fp = fopen("counter.txt", "a"); // Открываем файл в режиме записи

 $mytext = "Эту строку нужно записать в файл\r\n"; // Исходная
     строка

 $test = fwrite($fp, $mytext); // Запись в файл

 if ($test) echo 'Данные в файл успешно занесены.';
 else echo 'Ошибка при записи в файл.';

 fclose($fp); //Закрытие файла
 ?>
```

Чтение из файла *fgets()*

```
<?php
 $fp = fopen("counter.txt", "r"); // Открываем файл в режиме чтения
 if ($fp)
 {
 while (!feof($fp))
 {
 $mytext = fgets($fp, 999);       // Читаем строку из файла
 echo $mytext."<br />";           // Выводим на дисплей
 }
 }
 else echo "Ошибка при открытии файла";
 fclose($fp);
 ?>
```

Чтение из файла *fread()*

```
<?php
// получает содержимое файла в строку
$filename = "/usr/local/something.txt";
$handle = fopen($filename, "r");
$contents = fread($handle, filesize($filename));
fclose($handle);
?>
```

Функция *file()*

```
<?php
  $file_array = file("counter.txt");
  // Считывание файла в массив $file_array
?>
```

```
$lines = file('http://www.example.com/');

// выведем номера строк и их содержимое
foreach ($lines as $line_num => $line) {
    echo "Строка  #{$line_num}: " .$line . "\n";
```

Функция *fgetc()*

```
<?php
 $fp = fopen("counter.txt", "r"); // Открываем файл в режиме чтения
 if ($fp)
 {
   while(!feof($fp))
   {
   $char = fgetc($fp);
   if ($char == 'c') $i = $i + 1;// Находим символ «c»
   }
 echo 'Количество букв "c" в файле: '. $i;
 }
 else echo "Ошибка при открытии файла";
 ?>
```

Закрытие файла *fclose()*

```
<?php
 $fp = fopen("counter.txt", "r");
 if ($fp)
 {
 echo 'Файл открыт';
 fclose($fp); // Закрытие файла
 }
 ?>
```

Функция *file_get_contents()*

Пример #1 Получить и вывести код страницы вебсайта

```
<?php
$homepage = file_get_contents('http://www.example.com/');
echo $homepage;
?>
```

Пример #2 Чтение секции файла

```
<?php
// Читаем 14 символов, начиная с 21 символа
$section = file_get_contents('./people.txt', NULL, NULL, 20, 14);
var_dump($section);
?>
```

Функция *var_damp()*

```
<?php
$a = array (1, 2, array ("a", "b", 3.14));
var_dump ($a);
?>
```

```
array(3) {
 [0]=> int(1)
 [1]=> int(2)
 [2]=> array(3) {
        [0]=> string(1) "a"
        [1]=> string(1) "b"
        [2]=> float(1)  (3.14)
} }
```

Работа с файловой системой

mkdir("mydir"); - создание диретория (bool)
rmdir("mydir"); - удаление директория (bool)

chmod("myfile.txt", 0666); - изменение атрибутов (bool)
copy("myfile1.txt", "myfile2.txt"); - копирование файла (bool)
rename("myfile1.txt", "myfile2.txt"); - переименование файла (bool)
unlink("myfile.txt"); - удаление файла (bool)

if (file_exists("myfile.txt")) { …} – проверка на существование (bool)

file_put_contents("myfile.txt", $data); - перезапись файла указанной строкой PHP 5 (int)

6

Синтаксис языка PHP (регулярные выражения)

Regular expressions (reges)

Регулярные выражения — можно представить как мини-язык программирования, имеющий специфическое назначение: находить подстроки в больших строковых выражениях.

Эта технология изначально появилась в среде **UNIX** и обычно использовалась в языке программирования Perl.

Регулярное выражение (*regular expression*) — средство для обработки строк или последовательность символов, определяющее так называемый шаблон (pattern) текста.

Паттерн (*pattern*) — задает правило поиска, также иногда называют шаблоном, маской.

Метасимволы или спецсимволы — операции специального синтаксиса использующегося при составлении шаблонов. Это так называемые команды регулярного выражения.

Модификаторы — задают дополнительные условия поиска, такие как учет регистра и области поиска.

Функции работы с текстом

$arr = **explode**(".", "28.01.2008"); - разбить строку на подстроки по указанному разделителю (array)
$str = **implode**("-", array(2008,1,28)); - склеить значения элементов массива в одну строку с указанным разделителем (string)
$str = **ltrim**(" Строка"); - убрать пробельные символы слева (string)
$str = **rtrim**("Строка "); - убрать пробельные символы справа (string)
$str = **trim**(" Строка "); - убрать пробельные символы с двух сторон (string)
$str = **sprintf**("%02d.%02d.%04d", $d, $m, $y); - формирование строки из указанных аргументов по формату (string)
$a = **strlen**("qwerty"); - длина строки (int)
if (**strpos**($str, "q")===FALSE) {…} – поиск подстроки в в строке (int)
$s = **strstr**($str, "q"); - поиск подстроки в строке (string)
$s = **substr**("qwerty", 2, 3); - получить подстроку (string)
$str = **str_replace**("banana", "a", "@"); - замена подстрок (string)
if (**strcasecmp**($str1, $str2)==0) {…} – сравнение строк без учёта регистра (int)
if (**strcmp**($str1, $str2)==0) {…} – сравнение строк с учётом регистра
unset($str) – аннулировать строку

Пример

```
<?php
$stroka = 'php';
$file = file("php_text.txt");
$i = 0;
while ($i < sizeof($file)) {
    if (strstr($file[$i], $stroka) <> FALSE){
        unset($file[$i]);
    }
    $i++; }
// сохраняем в php_text_out всё что осталось
$f = fopen("php_text_out.txt", 'w+');
foreach($file as $string) {
        fwrite($f, $string); }
fclose($f);
?>
```

Регулярное выражение

Регулярное выражение - *это формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов.*

Карл у Клары украл кораллы, а Клара у Карла украла кларнет.

/(Клар.*?)\s/i → “Клары” “Клара” “кларнет”.

/ (Клар.*?)\ /i
~(Клар.*?)\s~i
#(Клар.*?)\s#i

Разделитель | Шаблон | Разделитель Модификатор

(Клар.*?)\s
- **\s** - пробел
- **.*** - любые символы
- **?** - нет символов

i - регистронезависимый

/(Клар.*?)\s/ → “Клары” “Клара”.

Примеры соответствий

Регулярное выражение	Соответствует	Не соответствует
'a.c'	abc, aWc, a+c	ac, a..c, cba
'F.*d'	FddFd, Freud, F.*d	feed, Sigmund, dF
'u..[p-t]'	user, u##s, uppp	undo, uncut, u=t
'wuff(-wuff)*'	wuff, wuff-wuff-wuff	wuffwuff, wuff-
'[+-]?[0-9]+'	+1, -0932, 333	+-7, -, 0+0
'[fit|plug](in|out)+'	fitin, pluginoutin	infit, plug, outin

Метасимволы

метасимволы-квантификаторы

метасимволы-модификаторы

Метасимволы

Метасимвол	Значение
\d	Цифра (0-9)
\D	Не цифра (любой символ кроме символов 0-9).
\s	Пустой символ (обычно пробел и символ табуляции).
\S	Непустой символ (все, кроме символов, определяемых метасимволом \s).
\w	Все буквы, все цифры и знак подчеркивания (‘_’).
\W	Все, кроме символов, определяемых метасимволом \w.
\n	Символ перевода строки.
\r	Символ возврата каретки.
\t	Символ табуляции
\xhh	Вставка символа с шестнадцатиричным кодом 0xhh
^	Начало строки.
$	Конец строки.
|	Метасимвол выбора (или).
.	Точка. Любой символ.
*****	Ноль или более символов.
+	Один или более символов.
?	Ноль или один символ.
{ }	Указывает количество повторений метасимволов. {4,6} - от 4 до 6 повторений.
\A	Начало строки.
\Z	Конец строки.
\z	Конец текста.
\b	Граница слова.
\B	Не граница слова.
\G	Предыдущий успешный поиск.

Квантификаторы

Квантификаторы

Квантификатор - это специальная конструкция, определяющая, сколько раз должен встретиться символ или группа символов. Квантификатор записывается в фигурных скобках "{}".

Возможны два формата записи: **точный и диапазонный**.

\d{4} - ровно четыре последовательно записанные цифры.
\d{2,4} - от двух до четырёх последовательно записанных цифр.
\w{3,} - три и более букв.
\d{,5} - цифр нет вообще, либо есть, но не более пяти.
[А-Яа-я]{1,3} - все русские слова из одной, двух или трёх букв.

Квантификаторы

Кроме фигурных скобок существует ещё три метасимвола-квантификатора: **"*"** (звёздочка), **"+"** (плюс) и **"?"** (вопрос).

Они используются в случаях, когда заранее неизвестно количество необходимых повторений. Например, при поиске адресов электронной почты нельзя заранее сказать, сколько символов будет в имени пользователя (до @), а сколько - в имени домена (после @).

\w* - любое количество последовательных букв, в том числе и 0.
\d+ - любая цифровая последовательность, где цифр одна или более.
\d? - любая цифровая последовательности, где цифр одна или две.

Жадность "*" и "+":

"мама мыла раму" → **\.+a** → "мама мыла ра"

"мама мыла раму" → **\.+a?** → "ма"

“?" – ограничитель «жадности»

Модификаторы

Модификаторы вставляются за разделителем – их четыре.

i	Включает режим case-insensitive, т.е. большие и маленькие буквы в выражении не различаются.
m	Указывает на то, что текст, по которому ведется поиск, должен рассматриваться как состоящий из нескольких строк. По умолчанию механизм регулярных выражений рассматривает текст как одну строку вне зависимости от того, чем она является на самом деле. Соответственно метасимволы '^' и '$' указывают на начало и конец всего текста. Если же этот модификатор указан, то они будут указывать соответственно на начало и конец каждой строки текста.
s	По умолчанию метасимвол '.' не включает в свое определение символ перевода строки. Т.е. для многострочного текста выражение /.+/ вернет только первую строку, а не весь текст, как ожидается. Указание этого модификатора снимает это ограничение.
g	Делает все количественные метасимволы "не жадными" по умолчанию ("g" - сокращение от английского "greedy", "жадный").

Основные правила

Выражение	Означает
foo	Строка “foo”
^foo	Строка начинается с “foo”
foo$	Строка заканчивается на “foo”
^foo$	«foo» встречается в строке только один раз
[abc]	a, b, или c
[a-z]	любой символ в нижнем регистре
[^A-Z]	любой символ, не находящийся в верхнем регистре
(gif | jpg)	Означает как «gif” так и “jpeg”
[a-z]+	Один или более символов нижнего регистра
[0-9.-]	Любая цифра, точка или знак минус
^[a-zA-Z0-9_]{1,}$	Любое слово, хотя бы одна буква, число или _
([wx])([yz])	wy, wz, xy, или xz
(^A-Za-z0-9)	Любой символ (не число и не буква)
([A-Z]{3}|[0-9]{4})	Означает три буквы или 4 цифры

Примеры регулярных выражений

Что анализируем	Регулярное выражение
Адрес e-mail	[\.\-_A-Za-z0-9]+?@[\.\-A-Za-z0-9]+?[\ .A-Za-z0-9]{2,}
Дата	^\d{1,2}([-. /])\d{1,2}\1\d{2,4}$
Дробное число	[\d]{4}\.[\d]{2}
Номер мобильного телефона	(?:8|\+7)? ?\(?(\d{3})\)? ?(\d{3})[-]?(\d{2})[-]?(\d{2})
Выделение текста между тегами	/<tag>(.*)</tag>/is

Работа с регулярными выражениями

Для работы с регулярными выражениями в PHP предназначены специальные функции:

Функция	Описание
preg_match()	Функция preg_match() ищет строку по заданному шаблону, возвращает true, если строка находится и false, в остальных случаях
preg_match_all()	Функция preg_match_all() находит все вхождения строки, заданной по шаблону
preg_replace()	Функция preg_replace(), действует по тому же принципу, что и str_replace(), за исключением того, что регулярные выражения можно использовать как для задания шаблона поиска, так и для строки, на которую следует заменить, найденное значение.
preg_split()	Функция preg_split(), действует так же как split(), за исключением того, что регулярное выражение можно использовать в качестве параметра для шаблона поиска.
preg_grep()	Функция preg_grep() ищет все элементы входного массива, возвращая все элементы, соответствующие шаблону регулярного выражения.
preg_quote()	Экранирует символы регулярного выражения

http://on-line-teaching.com/php/regexp.htm

Пример

```
function checkmail($mail) {
    // убираем пробелы и табуляторы
    $mail=trim($mail);
    if (strlen($mail)==0) return 1;

    if (!preg_match("[\.\-_A-Za-z0-9]+?@[\.\-A-Za-z0-9]+
    ?[\.A-Za-z0-9]{2,}",$mail))
    return -1;

    return $mail;
}
```

Домашнее задание

Объяснить работу следующих шаблонов регулярных выражений:

Шаблон	Описание
/^(http|https|ftp)://([A-Z0-9][A-Z0-9_-]*(?:.[A-Z0-9][A-Z0-9_-]*)+):?(d+)?/?/i	Проверка доменного имени.
/A(?=[-_a-zA-Z0-9]*?[A-Z])(?=[-_a-zA-Z0-9]*?[a-z])(?=[-_a-zA-Z0-9]*?[0-9])[-_a-zA-Z0-9]{6,}z/	Комплексная проверка пароля. Строка не менее шести символов, цифры, дефисы и подчеркивания, как минимум один символ верхнего регистра, один нижнего регистра и одна цифра.
/<title>(.*)</title>/	Поиск заголовка страницы.
^\d{1,2}([-. /])\d{1,2}\1\d{2,4}$	Разбиваем дату на числа.
/.*?\./	Получить расширение файла.
/\.(?:exe|msi|dmg|bin|xpi|iso)$/i	Проверка расширения файла.
/<a [^<>]*href=[\'"]([^\'"]+)[\'"][^<>]*>(((?!/si	Внешние ссылки.
/^[0-9]{1,55}$/	Является ли строка числом до 55 цифр.
/([a-zA-Za-яА-Я]+)/	Разбирает текст на отдельные слова.
/^.{1,10}$/	Любая строка, содержащая от 1 до 10 символов.
/(.*)/	Произвольная последовательность символов, заключенная между тегами.

Привести примеры строк удовлетворяющих и неудовлетворяющих условиям накладываемым приведенными шаблонами.

7

Синтаксис языка PHP (функции)

Пользовательские функции

```
<?php
        function myFunction($parameter1, $parameter2)
            {
              // набор выражений (тело функции)
              // return возвращаемое_значение
            }
?>
```

1. В теле функции используются копии параметров, поэтому все изменения переменных будут потеряны при выходе из функции.
2. Если параметр должен быть изменён в функции - необходимо передавать его по ссылке, т.е. в описании функции надо перед именем параметра добавить "&": function func($p1, &$p2) { }
3. В старых версиях PHP функция должна была быть объявлена до первого использования, но в PHP версии 4.3 и выше порядок объявления и использования функции может быть любым.

Функции работы с массивами

$arr = array("a"=>"lemon", "b"=>"apple", "c"=>"orange");

$i = **count**($arr); - количество элементов в массиве
$k = **array_keys**($arr); - получить массив ключей (array)
$v = **array_values**($arr); - получить массив значений (array)
list($a1, $a2, $a3) = $v; - задать значения переменным, как если бы они были элементами массива
if (**in_array**("lemon", $arr)) {…} – проверить наличие значения в массиве
$idx = **array_search**("lemon", $arr); - определить ключ, соответствующий указанному значению из массива (найти элемент)
$s = **array_sum**(array(0,1,2,3,4,5)); - найти сумму значений элементов массива (int)
$a = **array_merge**($k, $v, …); - слияние двух или нескольких массивов (array)
$a = **array_unique**($a); - формирование массива с уникальными значениями элементов (удаление дубликатов) (array)
$a = **asort**($arr); - сортировка по значениям (array)
$a = **ksort**($arr); - сортировка по ключам (array)
$a = **sort**($arr); - простая сортировка по значениям (array)

Функции работы с датой/временем

$stamp = **time**(); - текущее время (сек. от начала эпохи UNIX) (int)

$str = **date**("d.m.Y"); - текущее время по формату (string): D – день недели (Mon-Sun), H – час (00-23), i – минуты (00-59), M – месяц (Jan-Dec), O – часовой пояс от GMT (+0400), r – дата в формате RFC822, s – секунды (00-59), W – номер недели (1-53), z – день года (1-366)

$stamp = **mktime**($hr, $min, $sec, $mon, $day, $y); - вычислить время UNIX для указанного момента (int)

if (**checkdate**($mon, $day, $y)) {…} – проверка корректности даты (bool)

Математические функции

acos($x), **asin**($x), **atan**($x),
pi(), **cos**($x), **sin**($x), **tan**($x),
exp($x), **log**($x), **log10**($x), **pow**($x, $y), **sqrt**($x),
abs($x), **ceil**($x), **floor**($x), **round**($x),
max($v1, $v2, …), **min**($v1, $v2, …),
rand($min, $max), **srand**(time())

$val2 = **base_convert**($val10, 10, 2); - перевод системы счисления (str)

Функции работы с переменными

if (**empty**($var)) {…} – возвращает TRUE, если переменная $var имеет значение 0, "0", "", NULL, FALSE, array() или не определена (bool)

if (**isset**($var)) {…} – возвращает TRUE, если переменная определена

$str = **gettype**($var); - текущий тип переменной (string)

if (**is_array**($arr)) {…} – является ли $arr массивом (bool)

$str = **print_r**($arr, TRUE); - содержимое $arr в читабельном виде (string)

settype($a, "integer"); - установить тип переменной (bool)

unset($a, $b, …); - аннулировать переменные

$str = **serialize**($obj); - представить объект/массив/переменную в строковом виде (string)

$obj = **unserialize**($str); - перевести строковое представление в объект/ массив/переменную

Функции, манипулирующие URL

```
$url = parse_url("http://mp3:qwerty@mp3.int.ru/search.php?query=1");
// Разобрать URL на поля (array):
Array (
   [scheme] => http
   [host] => mp3.int.ru
   [user] => mp3
   [pass] => qwerty
   [path] => /search.php
   [query] => query=1
)
```

$str = **urlencode**(«any_string"); - URL-кодирование (string)

```
<?php
echo '<a href="mycgi?foo=', urlencode($userinput), '">';
?>
```

$str = **urldecode**("%41%46+%34%33"); - URL-декодирование (string)

!! Переменные в суперглобальных массивах *$_GET* и *$_REQUEST* уже декодированы.

Функции работы с изображениями

$info = **gd_info**(); - получить информацию о библиотеке GD (array)

```
Array (
    [GD Version] => bundled (2.0.28 compatible)
    [FreeType Support] => 1
    [FreeType Linkage] => with freetype
    [T1Lib Support] =>
    [GIF Read Support] => 1
    [GIF Create Support] => 1
    [JPG Support] => 1
    [PNG Support] => 1
    [WBMP Support] => 1
    [XBM Support] => 1
    [JIS-mapped Japanese Font Support] =>
)
```

$types = **imagetypes**(); - какие форматы поддерживаются (int)

if ($types) {…}

константы: IMG_GIF, IMG_JPG, IMG_PNG, IMG_WBMP.

Функции работы с изображениями

$img = **imagecreate**(200, 100); - создать картинку 200x100 с палитрой
$img = **imagecreatetruecolor**(300, 200); - создать полноцветную (16M) картинку 300x200 (resource)
$img = **imagecreatefromjpeg**("my.jpg"); - создать картинку на основе JPEG-файла (resource)
$img = **imagecreatefrompng**("my.png"); - создать картинку на основе PNG-файла (resource)
imagedestroy($img); - освободить память от картинки
imagejpeg($img); - вывод картинки в формате JPEG …
imagepng($img); - вывод картинки в формате PNG
$info = **getimagesize**("my.jpg"); - получить информацию о картинке

```
Array (
    [0] => 593                                  // ширина (также imagesx($img))
    [1] => 639                                  // высота (также imagesy($img))
    [2] => 3                                    // тип картинки для image_type_to_mime_type()
    [3] => width="593" height="639"             // строка для тега <IMG>
    [bits] => 8                                 // бит на цвет
    [mime] => image/png                         // MIME-тип
)
```

Функции работы с изображениями

$white = **imagecolorallocate**($img, 255, 255, 255); - получить идентификатор RGB-цвета (int). Первый вызов устанавливает фоновый цвет картинки.
imagesetpixel($img, $x, $y, $clr); - установить пиксел
imageline($img, $x1, $y1, $x2, $y2, $clr); - нарисовать линию
imagearc($img, $cx, $cy, $w, $h, $s, $e, $clr); – нарисовать дугу эллипса
imagefilledarc($img, $cx, $cy, $w, $h, $s, $e, $clr, IMG_ARC_PIE); - нарисовать закрашенный сектор эллипса
imageellipse($img, $cx, $cy, $w, $h, $clr); - нарисовать эллипс
imagefilledellipse($img, $cx, $cy, $w, $h, $clr); - закрашенный эллипс
imagerectangle($img, $x1, $y1, $x2, $y2, $clr); - прямоугольник
imagefilledrectangle($img, $x1, $y1, $x2, $y2, $clr); - нарисовать закрашенный прямоугольник
imagestring($img, $font, $x, $y, "qwerty", $clr); - нарисовать строку растровым шрифтом (встроенные: 1-5)
imagettftext($img, $sz, $angle, $x, $y, $clr, "arial.ttf", "qwerty"); - нарисовать строку (UTF8) векторным шрифтом (TTF)

Функции работы с СУБД MySQL

$link = **mysql_connect**("localhost", "user", "pass"); - установить соединение с MySQL-сервером (resource);
mysql_close($link); - закрыть соединение с сервером (bool)
mysql_select_db("test"); - открыть базу данных “test” (bool)
$req = **mysql_query**("select * from users where uid=1000"); - выполнить SQL-запрос (resource)
$n = **mysql_num_rows**($req); - кол-во строк в ответе на SELECT (int)
$n = **mysql_affected_rows**($link); - кол-во строк изменённых запросами DELETE/INSERT/REPLACE/UPDATE (int)
mysql_free_result($req); - освободить память от рез-тов запроса (bool)
$arr = **mysql_fetch_assoc**($req); - получить очередную строку ответа
$arr = **mysql_fetch_row**($req); - получить очередную строку ответа

$str = **mysql_escape_string**("Alex's PC"); - экранировать кавычки в строке для SQL-запроса

8

Синтаксис языка PHP (объекты и классы)

Определения

Объект – это структурированная переменная, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии.

Класс – это описание таких *объектов* и действий, которые можно с ними выполнять.

```
class Articles { // создаем класс Статей
  var $title;
  var $author;
  var $description;
// метод для присваивания значения атрибутам класса
  function make_article($t, $a, $d){
    $this->title = $t;
    $this->author = $a;
    $this->description = $d;
  }
//метод для отображения экземпляров класса
  function show_article(){
    $art = $this->title . "<br>" .
              $this->description .
            "<br>Автор: " . $this->author;
    echo $art;
  }
 }
```

Концепции ООП

Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Наследование – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом.

9

Синтаксис языка PHP (Практические примеры)

Переменная *$GLOBALS*

$GLOBALS — ссылки на все переменные глобальной области видимости

Пример #1 Пример *$GLOBALS*

```
<?php
    function test() {
        $foo = “Local variable";
        echo '$foo in global scope: ' . $GLOBALS["foo"] . "\n";
        echo '$foo in current scope: ' . $foo . "\n";
    }
    $foo = "Example content";
    test();
    ?>
```

Результатом выполнения данного примера будет:

```
$foo in global scope: Example content
$foo in current scope: Local variable
```

Суперглобальные переменные

$_GET

$_POST

$_COOKIE

$_GET

```
<?php
echo 'Привет ' . htmlspecialchars($_GET["name"]) . '!';
?>
```

Привет, Вася!

GET

Плюсы GET

1. Страницу всегда можно сохранить в закладках.
2. Он быстрее POST, так как вся информация находится в заголовках.
3. Информация, посылаемая на сервер, всегда видима (в адресной строке).

Минусы GET

1. Иноформация, посылаемая на сервер, всегда видима (в адресной строке).
2. Объем информации, которую можно отправить, ограничен.

$_POST

```
<?php
echo 'Привет ' . htmlspecialchars($_POST["name"]) . '!';
?>
```

Привет, Вася!

POST

Плюсы POST

1. Можно отправить много информации на сервер, объем неограничен.
2. Отправляемая информация не показывается в адресной строке. Удобно, если нужны красивые URL.

Но ее все равно можно легко увидеть. Не используйте POST как способ защиты сайта!

Минусы POST

1. Медленнее, чем GET, так как анализируются заголовки и тело запроса.
2. Страницы, сгенерированные как результат запроса POST, нельзя добавить в закладки.

$_COOKIE

```
<?php
echo 'Привет, ' . htmlspecialchars($_COOKIE["name"]) . '!';
?>
```

Привет, Вася!

Школа *www* по PHP

http://www.w3schools.com/php/

HTML

- HTML Tutorial
- HTML Tag Reference

CSS

- CSS Tutorial
- CSS Reference

JavaScript

- JavaScript Tutorial
- JavaScript Reference

SQL

- SQL Tutorial
- SQL Reference

PHP

- PHP Tutorial
- PHP Reference

JQuery

- JQuery Tutorial
- JQuery Reference

http://www.w3schools.com/

http://www.w3schools.com/

HTML/CSS

- Learn HTML
- Learn HTML5
- Learn CSS
- Learn CSS3

JavaScript

- Learn JavaScript
- Learn jQuery
- Learn jQueryMobile
- Learn AJAX
- Learn JSON
- Learn Google Maps

Server Side

- Learn SQL
- Learn PHP
- Learn ASP
- Learn ADO
- Learn ASP.NET
- Learn VBScript
- Learn AppML

XML Tutorials

- Learn XML
- Learn DTD
- Learn Schema
- Learn XML DOM
- Learn XPath
- Learn XSLT
- Learn XQuery
- Learn XSL-FO
- Learn SVG
- Learn RSS
- Learn WSDL

WEB Building

- Web Primer
- Web Building
- Web Statistics
- Web Validation
- Web Certificates

References

- HTML/HTML5 Tags
- HTML Colors
- HTML Characters
- HTML Symbols

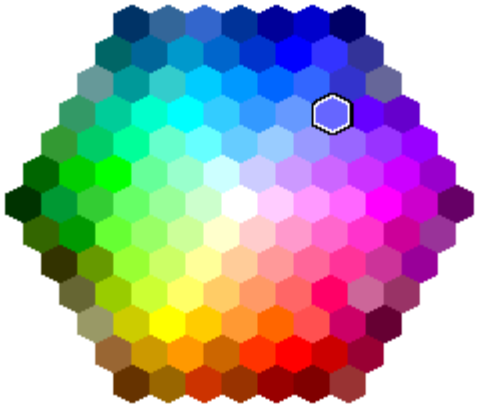
- CSS 1,2,3
- CSS3 Support

- JavaScript
- HTML DOM
- jQuery
- jQuery Mobile
- Google Maps

- PHP
- SQL
- ASP.NET

- XML DOM
- XSLT
- XPath
- SVG

Select color:



#6666FF

Selected color: #6666FF Submit

Вопросы

? ? ?