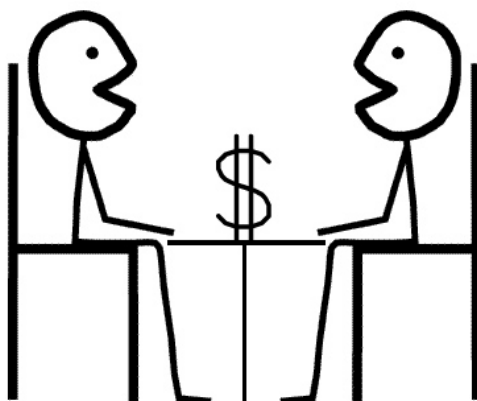


# PHP, JavaScript, JQuery, CSS, HTML, MySQL, ООП.



**Краткий сборник возможных  
вопросов и ответов  
на собеседовании.**

*Андрей Шевченко*

## Что говорят об этой книге читатели:

“В целом все очень хорошо, я даже знаю одного человека, который уже проводил собеседование по вашим вопросам”.

“Хорошая книга, несколько полезных вещей я для себя почерпнул”.

“Спасибо за статью и за книгу. Мне как студенту который только начинает познавать языки программирования легко прочитать такой простой материал и даже узнать много нового”.

“Удачный хендбук, в избранное однозначно!”.

“Мав проводит свою першу співбесіду у якості інтерв'юера, пішов нагуглити питань і натрапив на вашу книгу. Дуже вчасно!”.

“Потрібна книжка, ще й безкоштовна”.

“Хорошо, что автор не ограничился одним PHP, за вопросы по Javascript у спасибо!”.


“Легкий для понимания сборник, распечатал себе несколько страниц с примерами SQL-запросов”.

“Только начинаю изучать веб-разработку, с удовольствием прочитал эту книгу, особенно рад что в ней есть главы по ООП и MySQL. Понравился стиль изложения – простой и понятный начинающему”.

“Добавил в список книжек по программированию, которые собираюсь прочитать”.

“А есть что-то похожее по джаве?”.

**Особая благодарность за помощь при написании книги и конструктивную критику Олегу Махине и Виталию Савченко.**

Произведение «PHP. Собеседование в вопросах и ответах» созданное автором по имени Андрей Шевченко, публикуется на условиях лицензии Creative Commons Attribution-NonCommercial-ShareAlike (Атрибуция – Некоммерческое использование – С сохранением условий) 3.0 Непортированная. 

# ОГЛАВЛЕНИЕ

- 4** ВСТУПЛЕНИЕ
- 5** PHP
- 21** Общие принципы построения программ
- 24** ООП
- 28** JavaScript
- 32** MySQL
- 41** CSS
- 45** JQuery
- 48** HTML
- 52** Разное

# ВСТУПЛЕНИЕ

Идея написания этой книги появилась у меня после того, как я, готовясь к своему первому собеседованию на должность PHP-девелопера, не смог найти толкового "тutorials", в котором были бы собраны наиболее часто встречающиеся вопросы и задачи, которые задают на таких собеседованиях. Но готовиться же как-то надо? Надо. Иначе спросят потом чем отличается MySQL от InnoDB или что выберет вот такой jQuery-фильтр `$( "a[rel~='external']" )` и ты будешь сидеть и вспоминать чем же они еще отличаются кроме транзакций и что это за штука такая "~". Безусловно, отдельные вопросы без труда можно найти в Сети, но, во-первых, их далеко не так много, как хотелось бы, во-вторых, вопросы разбросаны по разным сайтам и блогам, что неудобно, ведь куда приятнее, когда все собрано в одном месте. Кроме того, зачастую это просто вопросы, без ответов, которые потом приходится искать самостоятельно. Конечно, можно заново перечитать книги по HTML, JavaScript, PHP, ООП, MySQL, пересмотреть tutorials по JQuery и CSS... но будет ли у вас время сделать все это перед интервью?

Вот почему я составил для себя почти 100-страничный "тutorials", который дополнил новыми вопросами уже "в боевых условиях", т.е. услышанными на реальных собеседованиях. Со временем этот разросшийся "тutorials" стал мне уже не нужным, т.к. я неплохо выучил все что мне было нужно, но, не пропадать же добру! Поэтому я убрал из него все лишнее, оставив только те вопросы и задачи, которые реально задают на собеседованиях ну и плюс еще некоторые, интересные на мой взгляд. В итоге получился сборник в стиле "вопрос – ответ", который удобно быстро пролистать, готовясь к собеседованию и который охватывает все основные концепции и ключевые области PHP и веб-разработки. Никакой лишней информации вроде "во время интервью держите спину прямо!" тут нет, только вопросы, многие из которых вы обязательно услышите и ответ, который устроит того, кто вас собеседует.

Для кого предназначена эта книга? В первую очередь, безусловно, для начинающих PHP- и WEB-девелоперов, которые хотят качественно подготовиться к собеседованию, либо же просто освежить свои знания или проверить себя. Но она также может быть полезна и тем, кто только начинает проводить собеседования, т.к. помимо вопросов, здесь приведены и задачи, которые:

- Не требуют много времени на решение, что оптимально для формата собеседования;
- С одной стороны не тривиальны, а с другой не слишком сложны;
- Понятны без продолжительной вводной.

Книга построена в виде реального собеседования, только очень длинного. Все вопросы разбиты на 9 разделов: PHP, Общие принципы построения программ, ООП, JavaScript, MySQL, CSS, JQuery, HTML и Разное.

Приятного чтения!

e-mail: [jeffjoellennox@mail.ru](mailto:jeffjoellennox@mail.ru)  
Skype: AndriiShevchenko

# 1. PHP

## 1. Какая разница между \$this и self в PHP?

\$this – это ссылка на сам объект, а self – на текущий класс.

## 2. Сколько типов данных в PHP?

PHP поддерживает 8 базовых типов данных.

### 4 скалярных типа:

**boolean.** Логический тип данных, переменные данного типа могут принимать значения true или false.

**integer.** Целочисленный тип данных, переменные могут принимать целые значения (...-2, -1, 0, 1, 2...) в диапазоне от  $-2^{31}$  до  $+2^{31}$ . Если значение превышает данный порог – оно автоматически переводится в тип float.

**float.** Числовой тип данных с плавающей точкой, может содержать как целые, так и дробные величины.

**string.** Строковый тип данных. Содержит нефиксированное количество различных символов. PHP не накладывает никаких ограничений на длину строки, поэтому можно смело работать даже с ОЧЕНЬ большими строками.

### 2 комплексных (составных) типа:

**array.** Массив, содержит упорядоченный список элементов.

**object.** Объект, содержит некий объект (экземпляр класса).

### 2 специальных типа:

**resource.** Ссылка на абстрактный элемент, т.н. внешний ресурс. Примеры внешних ресурсов - ссылка на файл и ссылка на результат выполнения запроса.

**NULL.** Пустой тип данных, обозначающий отсутствие какого-либо значения. О таких значениях обычно говорят “не определено”. Пустым значением можно инициализировать переменные любого другого типа.

## 3. Что можешь сказать про типизацию данных в PHP?

PHP является языком программирования с динамической типизацией, не требующим указания типа при объявлении переменных, равно как и самого объявления переменных. Преобразования между скалярными типами зачастую осуществляются неявно без дополнительных усилий. Впрочем, PHP предоставляет широкие возможности и для явного преобразования типов.

#### 4. Будет ли \$a == \$b? Будет ли \$b == \$c? Будет ли \$a == \$c?

```
<?php
$a = 0;
$b = null;
$c = "0";
?>
```

Ответ:

```
$a == $b
$a == $c
a вот $b != $c.
```

#### 5. Что такое static функция и чем она отличается от “обычной” (не static)?

Static принадлежит классу, а не экземпляру класса. И вызывается у класса, а не у объекта, т.е. напрямую. Объявление свойств и методов класса статическими позволяет обращаться к ним без создания экземпляра класса. Атрибут класса, объявленный статическим, не может быть доступен посредством экземпляра класса (но статический метод может быть вызван). Так как статические методы вызываются без создания экземпляра класса, то псевдопеременная \$this не доступна внутри метода, объявленного статическим. Доступ к статическим свойствам класса не может быть получен через оператор ->.

Пример. Static-члены класса доступны даже если объект этого класса не создан:

```
<?php
class A {
    public static $static_item = 'hello';
    public static function hello() {
        echo 'hello_function';
    }
}
echo A::$static_item; // выведет hello, хотя объекты класса А не создавались.
A::hello(); // выведет hello_function
?>
```

#### 6. Что такое конструктор?

Конструктор – это метод \_\_construct(), вызываемый при создании экземпляра класса (объекта) при помощи ключевого слова new.

#### 7. Приведи пример конструктора.

```
<?php
class MyClass {
    public function __construct() {
        echo "Привет из конструктора!";
    }
}
$myObject = new MyClass();
?>
```

## 8. Обязательно ли писать закрывающий тег `?>` в конце скрипта?

В PHP 4 – было обязательно. В PHP 5 не обязательно.

## 9. В каких случаях `?>` лучше не использовать?

Для файлов, содержащих только PHP-код, закрывающий тег `?>` лучше не использовать. Он не требуется синтаксисом PHP и его пропуск предотвращает случайное включение в вывод конечных пробелов.

## 10. Поддерживает ли PHP множественное наследование?

Нет, PHP не поддерживает множественное наследование. То есть у производного класса может быть только один родительский. Но с помощью “магической” функции `__call()` его можно эмулировать. А, начиная с версии 5.4.0 PHP вводит инструментарий для повторного использования кода, называемый трейтом (traits). Трейт предназначен для уменьшения некоторых ограничений единого наследования, позволяя разработчику повторно использовать наборы методов свободно, в нескольких независимых классах и реализованных с использованием разных архитектур построения классов.

Трейт очень похож на класс, но предназначен для группирования функционала хорошо структурированным и последовательным образом. Невозможно создать самостоятельный экземпляр трейта. Это дополнение к обычному наследованию и позволяет сделать горизонтальную композицию поведения, то есть применение членов класса без необходимости наследования.

Также стоит отметить, что один класс может реализовывать несколько интерфейсов.

## 11. Какая разница между `require()`, `require_once()`, `include()` и `include_once()`?

`require()` подключает в сценарий дополнительный файл, в то время как `require_once()` делает это только в том случае, если этот файл не был включен ранее.

Таким образом, `require_once()` лучше использовать, если нужно включить файл с большим количеством функций. Тогда можно быть уверенным, что файл не будет включен многократно и не возникнет ошибка “объявление функции дублируется”.

Отличие между `require()` и `include()` следующее: `require()` возвращает FATAL ERROR, если файл не найден, `include()` же возвращает только WARNING.

Функция `include_once()` работает почти так же, как и `include()`, а отличия те же, что и между `require()` и `require_once()`.

## 12. Какая разница между функциями `echo` и `print` в PHP?

Во-первых, `echo` может принимать и выводить любое количество аргументов, а `print` - только один. Во-вторых, `print` всегда возвращает 1, поэтому может быть использован в контексте выражения.

### 13. Чем отличается цикл `while` от `do while`?

`do-while` всегда выполняет тело цикла хотя бы один раз, поскольку его условное выражение проверяется в конце цикла.

### 14. Как перевернуть массив? Есть массив `array('h', 'e', 'l', 'l', 'o')`, как из него получить `array('o', 'l', 'l', 'e', 'h')`?

Для этого в PHP есть функция `array_reverse()`.

### 15. А как перевернуть массив без нее?

Например, так:

```
<?php
$arr = array ('h', 'e', 'l', 'l', 'o');
$reversed = array();
for ($i=count($arr)-1; $i>=0; $i--) $reversed[] = $arr[$i];
for ($i=0; $i<count($reversed); $i++) echo "$reversed[$i]";
?>
```

Или так:

```
<?php
$arr = array('h', 'e', 'l', 'l', 'o');
$reversed = array();
for ($i=0; $i<count($arr); $i++) array_unshift($reversed, $arr[$i]);
for ($i=0; $i<count($reversed); $i++) echo "$reversed[$i]";
?>
```

Или вот так:

```
<?php
$arr = array('h', 'e', 'l', 'l', 'o');
$reversed = array();
foreach ($arr as $v) array_unshift($reversed, $v);
for ($i=0; $i<count($reversed); $i++) echo "$reversed[$i]";
?>
```

Или же вот так:

```
<?php
$arr = array('h', 'e', 'l', 'l', 'o');

for ($i = 0; $i < floor(count($arr)/2); $i++)
{
    $tmp = $arr[$i];
    $arr[$i] = $arr[count($arr)-$i-1];
    $arr[count($arr)-$i-1] = $tmp;
}
for ($i=0; $i<count($arr); $i++) echo "$arr[$i]";
?>
```



И даже так:

```
<?php
$arr = array('h', 'e', 'l', 'l', 'o');
$pieces = count($arr)-1;
$reversed = array();
while($pieces >= 0) {
    $reversed[] = $arr[$pieces--];
}
for ($i=0; $i<count($reversed); $i++) echo "$reversed[$i]";
?>
```

Да хоть так:

```
<?php
$arr = array('h', 'e', 'l', 'l', 'o' );
$reversed = array();
while (count($arr)) $reversed[] = array_pop($arr);
for ($i=0; $i<count($reversed); $i++) echo "$reversed[$i]";
?>
```

## 16. Как перевернуть строку?

Функцией `strrev()`, а если без нее, то проще всего так:

```
<?php
$str = "Turn me baby";
for ($i = strlen($str); $i>=0; $i--) $rev[] = $str[$i];
$revstr = implode ("", $rev);
echo $revstr;
?>
```

А если это слишком просто, то можно и так:

```
<?php
$str = "Turn me baby";
function myrev($src) {
    $length = strlen($src);
    for ($i = 0; $i < $length / 2; $i++) {
        $a = $src[$i];
        $src[$i] = $src[$length - $i - 1];
        $src[$length - $i - 1] = $a;
    }
    return $src;
}
echo myrev($str);
?>
```

Или вот еще вариант:

```
<?php
$a = 'Turn me baby';
$b = '';
for ($i = strlen($a)-1; $i>=0; $i--)
    $b .= $a[$i];
$a = $b;
echo $a;
?>
```

## 17. Что такое рекурсия?

Рекурсия – это вызов функции из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция А вызывает функцию В, а функция В – функцию А. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

## 18. Напиши пример рекурсивной функции, которая вычисляет факториал числа.

```
<?php
function fac($x) {
    if ($x === 0)
        return 1;
    else
        return $x*fac($x-1);
    }
echo fac(4);
?>
```

## 19. Как вывести ряд чисел Фибоначчи?

```
<?php
function fibonacci($n)
{
    if ($n < 3) {
        return 1;
    }
    else {
        return fibonacci($n-1) + fibonacci($n-2);
    }
}
for ($n = 1; $n <= 16; $n++) {
    echo(fibonacci($n) . ", ");
}
echo("...\n");
?>
```

## 20. Что выведет этот скрипт?

```
<?php
$str = "0";
if (!$str) echo "EMPTY</br>";
else echo 'NOT EMPTY';

if (empty($str)) echo "EMPTY</br>";
else echo "NOT EMPTY";

if (!strlen($str)) echo "EMPTY</br>";
else echo "NOT EMPTY";
?>
```

Ответ: EMPTY  
EMPTY  
NOT EMPTY

## 21. А этот?

```
<?php
$str = "7";
if (!empty($str)) echo "EMPTY\n";
else echo "NOT EMPTY!\n";
?>
```

Ответ: EMPTY

## 22. Что получим?

```
<?php
$a=0;
if ($b=$a)
    echo "One";
else
    echo "Two";
?>
```

Ответ: Two

## 23. Преинкремент и постинкремент. В чем между ними разница?

Преинкремент ( $++\$i$ ) – сначала увеличивает, потом возвращает значение.  
Постинкремент ( $\$i++$ ) – сначала возвращает, потом увеличивает.

## 24. Что работает быстрее: преинкремент или постинкремент?

Преинкремент быстрее, т.к. постинкремент создает временную переменную, в то время как преинкремент изменяет саму переменную непосредственно.

## 25. Есть ли разница в сложении между PHP и JavaScript? "123" + "abc". Что будет? А если 123 + "abc"?

В JavaScript "+" это конкатенация, т.е. строки просто соединятся. В PHP в обоих случаях результат будет 123. А если в PHP сложить, например,  $10 + "20"$ , то, несмотря на кавычки результат будет 30.

## 26. Есть ли разница между одинарными и двойными кавычками в PHP?

В двойных кавычках данные "парсятся", а в одинарных – нет. Двойные кавычки в данном случае приведут к результату Chimay, а одинарные к \$beer.

```
<?php
$beer = 'Chimay';
echo "$beer";
?>
```

## 27. Нужно определить длину строки функцией `strlen()`. Откуда она узнает сколько в строке символов?

`strlen()` не выполняет подсчет символов. Она берет уже готовый результат из структуры `zval` (внутренняя C-структура, которая используется для хранения переменных в PHP).

## 28. Проход массива. Как вывести все элементы массива на экран?

Вывести с ключами: `print_r`, а пройти и вывести массив:

```
<?php
$cars = array("BMW", "Audi", "Mercedes", "Porsche");
foreach ($cars as $car) {
    echo $car . "<br />";
}
?>
```

либо так:

```
<?php
$cars = array("BMW", "Audi", "Mercedes", "Porsche");
for ($i=0; $i<count($cars); $i++) echo $cars[$i]."<br/>";
?>
```

## 29. Что такое ассоциативный массив?

Массивы, индексами которых являются строки, называются ассоциативными. Индексы ассоциативных массивов называются ключами. Например:

```
$price = array("яблоки" => 10, "груши" => 15, "бананы" => 24);
```

## 30. Нарисуй форму для отправки файла:

Для реализации возможности загрузки файлов на сервер можно использовать простую форму:

```
<form action="upload.php" method="post" enctype="multipart/form-data">
<input type="file" name="uploadfile">
<input type="submit" value="Загрузить">
</form>
```

Этот код выводит в браузер элемент `input` с кнопкой "Обзор" и кнопку "Загрузить". По нажатию на эту кнопку происходит обращение к файлу `upload.php`, который содержит следующий код:

```
<?php
$uploadfile = './upload/'.basename($_FILES['uploadfile']['name']);
// Копируем файл из каталога для временного хранения файлов:
if (copy($_FILES['uploadfile']['tmp_name'], $uploadfile))
{
    echo "<h3>Файл успешно загружен на сервер</h3>";
}
else { echo "<h3>Ошибка! Не удалось загрузить файл на сервер!</h3>"; exit;
}
?>
```

**31. Пусть имеем HTML-форму, которая содержит одно поле ввода text и одно поле ввода textarea. Требуется создать для данной HTML-формы скрипт-обработчик, который заносит построчно в файл result.txt полученные данные. В итоге структура файла result.txt должна быть следующая:**

**Name:** текст, введенный в поле text.

**Message:** текст, введенный в поле textarea.

Форма:

```
<form action="upload.php" method="post">
Name: <input type="text" name="name" /><br>
Message: <textarea rows="10" cols="20" name="message"></textarea>
<input type="submit" value="Загрузить" />
</form>
```

Файл обработки:

```
<?php
$all="Name: \"$_POST['name'].\"\\r\\n\".\"Message: \"$_POST['message'].\"\\r\\n\";
$files="result.txt";
if (!$handle = fopen($files, 'a')) {
echo "Невозможно открыть файл ($filename)";
exit;
}
if (fwrite($handle, $all) === FALSE) {
echo "Невозможно произвести запись в файл ($files)";
exit;
} else {
echo "Информация успешно записана в файл!";
}
?>
```

**32. Используя конструкцию switch, написать функцию foo(), принимающую одно число в качестве аргумента. Если это число равно 2, функция должна вывести слово "Двойка", если 3 – "Тройка", в остальных случаях – "Другое число".**

```
<?php
function foo($num) {
    switch($num) {

        case "2":
            echo "Двойка";
            break;
        case "3":
            echo "Тройка";
            break;

        default:
            echo "Другое число";
            break;
    }
}
foo(2);
?>
```

**33. Дан массив `$arr = array(3,8,15,25,16,11,10,5,7,30)`. Вывести те его элементы, которые делятся на 5.**

```
<?php
$arr = array(3,8,15,25,16,11,10,5,7,30);
foreach($arr as $v)
if($v % 5 == 0) echo $v.'

```

А можно и так:

```
<?php
$arr = array(3,8,15,25,16,11,10,5,7,30);
array_walk($arr, function($i){ if ($i%5===0)echo "$i</br>";});
?>
```

**34. Написать программу, которая выводит простые числа, т.е. делящиеся без остатка только на себя и на 1.**

```
<?php
$lst = array();
$k = "prostoe";
for($i = 2; $i<100; $i++) {
for($j = 2; $j < $i; $j++) {
if( ($i % $j) == 0) {
$k="ne prostoe";
}
}
if ($k == "prostoe")
$lst[] = $i;
else
$k = "prostoe";
}
foreach ($lst as $list) echo $list."

```

В качестве флага (`$k`) корректнее было бы использовать `true/false`, но я оставил так как есть для наглядности, чтобы было легче понять как работает программа. И еще. Скрипт можно ускорить, т.к. мы ищем простые числа и, например, ловим в первом цикле число 60 и поочередно делим его от 2 до 59, а есть смысл делить его до 31, дальше все впустую. Для этого изменяем пятую строку скрипта таким образом: `for($j = 2; $j < ($i/2)+1; $j++)`

**35. Сгенерировать 3 случайных числа в диапазоне от 0 до 10. Если сумма этих чисел меньше 14, сгенерировать новую тройку.**

```
<?php
do {
$a = mt_rand(0, 10);
$b = mt_rand(0, 10);
$c = mt_rand(0, 10);
$result = $a+$b+$c;
echo $result."

```

### 36. Чему будет равно \$a?

```
$a = "1";
$a[$a] = "2";
echo $a;
```

Ответ: 12

### 37. Есть массив `a = array(тут много элементов)`. Проходим по массиву циклом `for (i=0; i<=count(a); i++)`. Можно ли как-нибудь ускорить цикл?

Да.

- 1) Вынести `count(a)` в отдельную переменную;
- 2) Считать массив с конца циклом `for (i=count(a); i>=0; i--)`.

### 38. Вывести максимальное значение элемента массива `array(1,2,3,4,10,100,3,4987,6,7,8,9)`.

С использованием стандартной функции `max()`:

```
$arr = array(1,2,3,4,10,100,3,4987,6,7,8,9);
echo max($arr);
```

Без использования стандартной функции:

```
<?php
$arr = array(5,45,3,4,5,490,62);
$max = $arr[0];
foreach ($arr as $val)
if ($max < $val) $max = $val;
echo $max;
?>
```

Или так:

```
<?php
$arr = array(5,45,3,4,5,490,62);
$max = 0; array_walk($arr, function($i) use (&$max){
if ($max<$i) $max=$i;});
echo $max;
?>
```

### 39. Напиши программу-цензор, которая бы заменяла вводимые пользователем в форму слова "fuck", "idiot" и "bitch" на "f\*\*k", "id\*\*t" и "bi\*\*h".

```
<?php
$find = array('fuck', 'idiot', 'bitch');
$replace = array('f**k', 'id**t', 'bi**h');
If (isset($_POST['user_input']) && !empty($_POST['user_input'])) {
```

```

$user_input = $_POST['user_input'];
$user_input_new = str_ireplace($find, $replace, $user_input);
echo $user_input_new;
}
?>

<html>
<head></head>
<body>
<form action="a.php" method="POST">
<textarea name = "user_input" rows = "6" cols = "30"></textarea>
<input type = "submit" value = "Submit!">
</form>
</body>
</html>

```

## 40. Какие магические методы знаешь? Что это вообще такое?

Это методы зарезервированные в php, которые начинаются с двойного подчеркивания “\_”.

### Список всех магических методов:

```

__construct
__destruct
__call
__callStatic
__get
__set
__isset
__unset
__sleep
__wakeup
__toString
__set_state
__clone

```

`__construct` и `__destruct` – самые популярные методы, которые реализуют базовые понятия объектно-ориентированного программирования: конструктор и деструктор;

`__call`, `__callStatic`, `__get` и `__set` – методы, связанные с перегрузкой обращений как к свойствам, так и к методам. Методы `__get()` и `__set()` вызываются при установке и получении значения свойства, а методы `__call()` и `__callStatic` – при вызове метода. Стоит заметить, что эти магические функции будут вызываться только и исключительно в том случае, если запрошенные метод или свойство не существуют;

`__isset` – метод, срабатывающий при вызове функций `empty()` или `isset()` на несуществующем или недоступном свойстве класса;

`__unset` – срабатывает при вызове функции `unset()` на несуществующем или недоступном свойстве класса;

`__sleep` и `__wakeup` – методы, которые вызываются только из функций `serialize` и `unserialize` соответственно. Метод `__sleep` будет вызван сразу при применении к объекту функции `serialize`, а метод `__wakeup` – при применении `unserialize`. В настоящий момент методы применяются для



сохранения текущего состояния системы с последующим восстановлением данного состояния (например, коннект к базе);

`__toString` – метод, с помощью которого можно обращаться к классу как к строке (например, с помощью `print` или `echo`);

`__set_state` – метод, который вызывается для классов, экспортирующих значения свойств функцией `var_export()`;

`__clone` – вызывается при клонировании объекта (введен для использования из-за того, что объекты в php5 и выше передаются по ссылке);

`__invoke` – вызывается при попытке использовать объект в качестве функции.

## 41. Назови по памяти любые функции для работы со строками, массивами и файлами.

Например, такие:

### Строки:

**echo** – выводит одну или более строк.

**explode** – разбивает строку на подстроки.

**htmlentities** – преобразует символы в соответствующие HTML сущности.

**htmlspecialchars** – преобразует специальные символы в HTML сущности.

**implode** – объединяет элементы массива в строку.

**ltrim** – удаляет пробелы из начала строки.

**rtrim** – удаляет пробелы из конца строки.

**trim** – удаляет пробелы из начала и конца строки.

**md5** – возвращает MD5 хэш строки.

**str\_repeat** – возвращает повторяющуюся строку.

**str\_replace** – заменяет строку поиска на строку замены.

**str\_split** – преобразует строку в массив.

**strlen** – возвращает длину строки.

**strrev** – переворачивает строку.

### Массивы:

**array\_flip** – возвращает массив в обратном порядке, то есть ключи массива исходный\_массив становятся значениями, а значения массива исходный\_массив становятся ключами.

**shuffle** – перемешать массив

**ksort** – отсортировать массив по ключам

**array\_unique** – убрать повторяющиеся значения из массива

**array\_unshift** – добавить один или несколько элементов в начало массива

**array\_sum** – вычислить сумму значений массива

**array\_shift** – извлечь первый элемент массива

**array\_reverse** – возвращает массив с элементами в обратном порядке

**array\_pop** – извлечь последний элемент массива

**array\_push** – добавить один или несколько элементов в конец массива

**array\_keys** – выбрать все ключи массива

**array\_count\_values** – подсчитать количество всех значений массива

**Файлы:****filesize** – получить размер файла**filetype** – получить тип файла**fopen** – открывает файл или URL**fwrite** – бинарно-безопасная запись в файл**is\_executable** – определяет, является ли файл исполняемым**if\_file** – определяет, является ли файл обычным файлом**is\_readable** – определяет, доступен ли файл для чтения**is\_uploaded\_file** – определяет, был ли файл загружен при помощи HTTP POST**is\_writable** – определяет, доступен ли файл для записи**mkdir** – создаёт директорию**move\_uploaded\_file** – перемещает загруженный файл в новое место**pathinfo** – возвращает информацию о пути к файлу**readfile** – выводит файл**rename** – переименовывает файл или директорию**unlink** – удаляет файл**42. Какая функция возвращает количество рядов результата MySQL запроса?**

`mysql_num_rows()`. Она работает только с запросами SELECT. Чтобы получить количество рядов, обработанных функциями INSERT, UPDATE, DELETE, нужно использовать функцию `mysql_affected_rows()`.

Пример:

```
$result = mysql_query("SELECT * FROM table1", $link);
$num_rows = mysql_num_rows($result);

echo "$num_rows Rows\n";
```

**43. Какая разница между функциями `sort()`, `asort()` и `ksort()`?**

1) `sort()` сортирует массив элементов. В отсортированном массиве элементы размещаются по возрастанию. Это функция сортировки по умолчанию.

2) `asort()` сортирует ассоциативный массив так, что отсортированными оказываются элементы-значения ассоциаций. Используется, если важен порядок самих элементов, а не ключей.

Например:

```
$capitals = array("US" => "Washington", "UK" => "London", "Austria" => "Vienna");
asort($capitals);
```

```
// $capitals = {"UK" => "London", "Austria" => "Vienna", "US" => "Washington"}
```

3) `ksort()` сортирует ассоциативный массив по значению ключей. Для предыдущего примера отсортированные значения были бы такими:

```
ksort($capitals);
```

```
// $capitals = {"Austria" => "Vienna", "UK" => "London", "US" => "Washington"}
```

## 44. Что такое динамические переменные?

Динамической переменной считается та, имя которой хранится в самой переменной. Это так называемая “переменная переменная”.

Например:

```
$var = "first";
$$var = "Second";

// $$var == $first == "Second"
```

`$$var` – динамическая, ее имя может меняться вместе с изменением `$var`. Также, можно связать имя переменной с содержимым другой переменной неявно:

```
$first = "second";
$second = "third";
print $first; // напечатает "second"
print $$first; // напечатает "third"
```

## 45. Как можно переадресовать пользователя на другую страницу?

1. Используя функцию PHP `header()`

```
header('Location: '.$url);
```

2. Используя мета-тег `refresh`. Вот пример для перевода посетителя на другую страницу сайт, через определенное время:

```
<meta http-equiv="refresh" content="3;URL= http://phpbook.comli.com">
```

`content="3"` указывает на время задержки между обновлениями страницы в секундах. Тег заставит обновить страницу в браузере через 3 секунды, заменив адрес страницы на новый, указанный в теге. Если нужно перевести посетителя на другую страницу сразу, без задержки, просто устанавливаем вместо трех секунд ноль.

## 46. Для чего в PHP 5 используется тип данных “указатель”?

Такого типа данных в PHP 5 нет.

## 47. Какой будет результат выполнения этого скрипта?

```
<?php
function byRef(&$dollars) {
    $dollars++;
}
$dollars = 300;
$euros = 100;
byRef($euros);
```

```
echo "I have $dollars dollars and $euros euros.";
?>
```

Ответ: I have 300 dollars and 101 euros.

### 48. Какой будет результат выполнения этого скрипта?

```
<?php
$a = 10;
$b = 4;
echo (int)$a / (int)$b;
?>
```

Ответ: 2,5

### 49. Какой будет результат выполнения этого скрипта?

```
<?php
$arr = array(1,2,3,4,5,6,7,8,9);
$count = count($arr);
if ($count = 0) {
    echo "Array is empty.";
} else {
    echo "Array contains $count elements.";
}
?>
```

Ответ: Array contains 0 elements.

### 50. Какой будет результат выполнения этого скрипта?

```
<?php
function foo() {
    static $count = 4;
    return ++$count;
}
print foo();
print foo();
print foo();
?>
```

Ответ: 567

# 2. Общие принципы построения программ

## 1. Что такое MVC?

*Model-view-controller* – это схема использования шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента так, что модификация одного из компонентов оказывает минимальное воздействие на остальные. Данная схема проектирования часто используется для построения архитектурного каркаса, когда переходят от теории к реализации в конкретной предметной области.

## 2. Что за что отвечает в MVC?

В шаблоне MVC, как следует из названия, есть три основных компонента: Модель, Представление, и Контроллер.

**Модель** является “сутью” системы и отвечает за непосредственные алгоритмы, расчёты и тому подобное внутреннее устройство системы.

**Представление** отвечает за отображение информации, поступающей из системы или в систему.

**Контроллер** является связующим звеном между “моделью” и “представлением” системы, посредством которого и существует возможность произвести разделение между ними. Контроллер получает данные от пользователя и передаёт их в “модель”. Кроме того, он получает сообщения от модели, и передаёт их в “представление”.

## 3. Что такое шаблоны (паттерны) проектирования?

Паттерн проектирования – это общее типовое решение некоторой проблемы, многократно повторяемое в процессе проектирования архитектуры программного продукта. Они показывают отношения и взаимодействия между классами, позволяют сделать систему гибкой и легко изменяемой. За счет их правильного использования повышается коэффициент использования готовых решений.

## 4. Какие паттерны знаешь?

### 1. Порождающие

**Abstract Factory (Абстрактная фабрика)**. Его основное назначение – предоставить интерфейс для создания семейства взаимосвязанных объектов, не специфицируя их классы.

**Factory Method (Фабричный метод)**. Используется для определения и поддержания отношений между объектами. Фабричные методы избавляют проектировщика от необходимости встраивать в код зависящие от приложения классы.

**Singleton (Одиночка).** Используется для создания всего одного экземпляра класса и гарантирует, что во время работы программы не появится второй. Например, в схеме MVC, зачастую этот паттерн используется для порождения главного (фронтového) контроллера.

**Prototype (Прототип).** Используется для задания вида создаваемых объектов на основе объекта прототипа, от которого происходит передача внутреннего состояния. Prototype позволяет избавиться от жесткой привязки к классам и, как следствие, вязкости кода.

**Builder (Строитель).** Используется для отделения процесса конструирования сложного объекта от его представления, так что в результате одного и того же конструирования могут получаться различные объекты.

## 2. Структурные

**Adapter (Адаптер).** Унифицирует классы и объекты. Используется для преобразования одного интерфейса в другой, необходимый клиенту.

**Bridge (Мост).** Используется для отделения абстракции от ее реализации так, чтобы то и другое можно было изменять независимо.

**Composite (Компоновщик).** Объединяет объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково. Основным назначением паттерна, является обеспечение единого интерфейса как к составному так и конечному объекту, что бы клиент не задумывался над тем, с каким объектом он работает. Общеизвестными примерами этого паттерна является SimpleXML и jQuery.

**Decorator (Декоратор).** Используется для динамического расширения функциональности объекта. Является гибкой альтернативой наследованию.

**Facade (Фасад).** Определяет интерфейс более высокого уровня, который упрощает использование подсистем. Представляет собой унифицированный интерфейс вместо набора интерфейсов некоторой подсистемы. Разбиение на подсистемы сложной системы позволяет упростить процесс проектирования, а также помогает максимально снизить зависимости одной подсистемы от другой. Однако это приводит к тому, что использовать такие подсистемы вместе становится довольно сложно. Один из способов решения этой проблемы является ввод паттерна фасад.

**Flyweight (Приспособленец).** Он используется для эффективной поддержки множества мелких объектов и позволяет повторно использовать мелкие объекты в различном контексте.

**Proxy (Прокси).** Предоставляет объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).

## 3. Паттерны поведения

**Chain of Responsibility (Цепочка обязанностей).** Предназначен для организации в системе уровней ответственности.

**Command (Команда).** Инкапсулирует различные алгоритмы в единую сущность.

**Interpreter (Интерпретатор).** Реализует динамические алгоритмы с помощью декларативного описания.

**Iterator (Итератор).** Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из объектов, входящий в состав агрегации.

**Mediator (Посредник).** Предоставляет собой единый центр взаимодействия определенной группы объектов, которые должны быть взаимосвязаны друг с другом.

**Memento (Хранитель).** Паттерн поведения объектов, сохраняющий состояния.

**Observer (Наблюдатель).** Определяет зависимость типа “один ко многим” между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.

**State (Состояние).** Используется в тех случаях, когда во время выполнения программы объект должен менять свое поведение в зависимости от своего состояния.

**Strategy (Стратегия).** Предназначен для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путем определения соответствующего класса. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.

**Template Method (Шаблонный метод).** Определяет основу алгоритма и позволяет наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.

**Visitor (Посетитель).** Описывает операцию, которая выполняется над объектами других классов. При изменении Visitor нет необходимости изменять обслуживаемые классы.

## 5. Напиши на PHP пример реализации паттерна Singleton.

```
<?php
class Database {
    private static $connection;
    private function __construct() {
        echo "Hello from construct!";
    }
    public static function Connect() {
        if (!isset(self::$connection)) {
            self::$connection = new Database;
        }
        return self::$connection;
    }
}

Database::Connect();
```

# 3. ООП

## 1. ООП знаешь? Что это такое?

Объектно-ориентированное программирование – это парадигма программирования, в которой основными концепциями являются понятия объектов и классов. В языках с прототипированием (например, в JavaScript) вместо классов используются объекты-прототипы.

## 2. Расскажи основные принципы ООП.

1) **Инкапсуляция.** Это механизм, который объединяет данные и методы, манипулирующие этими данными, и защищает их от внешнего вмешательства или неправильного использования. Когда методы и данные объединяются таким способом, создается объект. Т.е. переменные состояния объекта скрыты от внешнего мира. Изменение состояния объекта (его переменных) возможно ТОЛЬКО с помощью его собственных методов. Можно сказать, что инкапсуляция подразумевает под собой сокрытие данных, что позволяет эти данные защитить.

2) **Наследование.** Это процесс, посредством которого, один объект может наследовать свойства другого объекта и добавлять к ним черты, характерные только для него.

3) **Полиморфизм.** Это свойство, которое позволяет одно и тоже имя использовать для решения нескольких технически разных задач. Проще говоря, концепцией полиморфизма является идея “один интерфейс, множество реализаций”. Это означает, что можно создать общий интерфейс для группы близких по смыслу действий.

## 3. Напиши пример реализации полиморфизма.

```
<?php
Class One {
    function foo() {
        echo "Hello from class One!";
    }
    function callMe() {
        $this->foo();
    }
}
Class Two extends One {
    function foo() {
        echo "Hello from class Two";
    }
}
$Two = new two();
$Two->callMe();
?>
```

## 4. Что такое виртуальный метод?

Виртуальный метод в объектно-ориентированном программировании – это метод класса, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения. Виртуальные методы – один из важней-



ших приёмов реализации полиморфизма.

## **5. А зачем такое нужно?**

Чтобы программисту необязательно было знать точный тип объекта для работы с ним через виртуальные методы, достаточно лишь знать, что объект принадлежит классу или наследнику класса, в котором метод объявлен.

Виртуальные методы позволяют создавать общий код, который может работать как с объектами базового класса, так и с объектами любого его класса-наследника. При этом базовый класс определяет способ работы с объектами и любые его наследники могут предоставлять конкретную реализацию этого способа.

## **6. Зачем нужна инкапсуляция?**

Никто не застрахован от ошибок, а человеку тем более свойственно ошибаться. Применяя инкапсуляцию, мы, как бы, возводим купол, который защищает данные, принадлежащие объекту, от возможных ошибок, которые могут возникнуть при прямом доступе к этим данным. Кроме того, применение этого принципа очень часто помогает локализовать возможные ошибки в коде программы, а это намного упрощает процесс поиска и исправления этих ошибок.

## **7. Как называется способность объекта скрывать свои данные и реализацию от других объектов системы?**

Инкапсуляция.

## **8. Какие механизмы в ОО языках обычно позволяют обеспечить инкапсуляцию объектов?**

Модификаторы доступа.

## **9. Может ли быть конструктор виртуальным?**

Нет, конструкторы не могут быть виртуальными.

## **10. Что такое класс?**

Класс – это модель ещё не существующей сущности (объекта). Класс фактически описывает устройство объекта, являясь своего рода чертежом.

## **11. А объект?**

Объект – это совокупность данных и методов для их обработки. Данные и методы называются членами класса. Вообще, объектом является все то, что поддерживает инкапсуляцию.

## 12. Чем отличается класс от объекта?

Класс – это тип данных, а объект – экземпляр типа класс.

## 13. Что такое область видимости переменной?

Область видимости переменной – это место в программе, в котором доступно значение переменной. Каждая переменная имеет свою область видимости (есть локальные переменные и глобальные переменные)

**Public.** Метод/переменная доступны из любого места в коде.

**Protected.** Защищённые методы или переменные доступны только внутри класса, где они были объявлены и из его производных классов.

**Private.** Закрытые методы или переменные доступны только внутри класса.

## 14. Чем локальные переменные отличаются от глобальных?

Локальные доступны только конкретной подпрограмме, глобальные – всей программе. Ограничение зоны видимости придумали как для возможности использовать одинаковые имена переменных (что разумно, когда в разных подпрограммах переменные выполняют похожую функцию), так и для защиты от ошибок, связанных с неправомерным использованием переменных.

## 15. Чем отличается процедурный подход от объектно-ориентированного?

Процедурный подход предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Задачи разбиваются на шаги и решаются шаг за шагом. Кроме того данные лежат отдельно от функций, для каждой новой сущности приходится писать свой набор функций с немного другими именами.

А ООП предполагает заключение внутри одного класса, как данных, так и методов их обработки. При этом создание новой сущности не вызывает необходимости переписывать все методы, а только нужные (это называется “наследование”).

## 16. Какие еще есть парадигмы (модели, подходы) программирования кроме ООП?

Из тех, что наиболее часто встречаются: функциональная, аспектно-ориентированная и процедурная.

## 17. Что такое абстрактный класс?

Абстрактный класс в объектно-ориентированном программировании – это базовый класс, который не предполагает создания экземпляров. Абстрактные классы реализуют на практике один из принципов ООП - полиморфизм. Абстрактный класс может содержать абстрактные методы и свойства. Абстрактный метод не реализуется для класса, в котором описан, однако должен быть

реализован для его неабстрактных потомков. Абстрактные классы представляют собой наиболее общие абстракции, то есть имеющие наибольший объем и наименьшее содержание.

## 18. Можно ли создать экземпляр абстрактного класса?

Если язык программирования позволяет, то можно, например в Delphi. Но, поскольку, мы говорим о PHP, то создать экземпляр абстрактного класса нельзя.

## 19. Какая разница между абстрактным классом и интерфейсом?

*Один из самых любимых вопросов!*

Абстрактный класс – это класс, который имеет хотя бы 1 абстрактный (не определенный) метод и обозначается как `abstract`. Интерфейс – такой же абстрактный класс, только в нем не может быть свойств и не определены тела у методов.

Кроме того, что абстрактный класс наследуется (`extends`), а интерфейс реализуется (`implements`). Вот и возникает разница между ними, что наследовать мы можем только 1 класс, а реализовать сколько угодно.

## 20. Зачем нужен интерфейс, если есть абстрактный класс?

Затем, что можно унаследоваться только от одного абстрактного класса, но реализовать множество интерфейсов. Плюс, в качестве приятного дописка, ВСЕ методы, описанные в интерфейсе, ДОЛЖНЫ быть реализованы в классе, а в абстрактном классе их нужно для этой цели специально отмечать.

# 4. JAVASCRIPT

## 1. Как переадресовать страницу в JavaScript?

```
<script type="text/javascript">
window.location.href = "http://www.phpbook.comli.com"
</script>
```

## 2. Сколько параметров можно передать функции?

Сколько угодно.

## 3. Нужно алертом вывести какое-то сообщение, спустя 3 секунды после запуска скрипта. Как это сделать?

Так:

```
setTimeout(alert("Hello", 3000));
```

Или так:

```
setTimeout(function() {alert("Hello")}, 3000);
```

## 4. Чем отличается наследование в JavaScript от наследования в PHP?

В отличие от PHP, где наследование можно делать одним способом, в JavaScript таких способов много. На уровне языка реализовано наследование на прототипах.

В JavaScript каждый объект может иметь ассоциацию с другим объектом – так называемый “прототип” (prototype). В случае, если поиск некоторого свойства (или метода) в исходном объекте заканчивается неудачно, интерпретатор пытается найти одноименное свойство (метод) в его прототипе, затем – в прототипе прототипа и т. д. К примеру, если мы затребовали обращение к obj.pgor (или, что абсолютно то же самое, obj[‘pgor’]), JavaScript начнет искать свойство pgor в самом объекте obj, затем – в прототипе obj, прототипе прототипа obj, и так до конца.

## 5. Приведи пример наследования в JavaScript.

Например, пусть объект “wolf” наследуется от объекта “animal”.

В наследовании на прототипах это реализуется как ссылка

```
wolf.prototype = animal;
```

Или вот чуть более развернутый пример. MyType наследуется от Obj:

```
Obj = {
  x: "1"
} // создаем объект Obj и записуем в него свойство x = 1.

MyType = function() {
} // создаем пустой объект MyType.

MyType.prototype = Obj; // наследуем MyType от Obj.
newObj = new MyType();

document.write(newObj.x);
```

## 6. Пара слов об объектах в JavaScript?

Объекты (они же – ассоциативные массивы, хэши) и работа с ними в JavaScript реализованы не так, как в большинстве языков. Объект в JavaScript представляет собой обычный ассоциативный массив или, иначе говоря, “хэш”. Он хранит любые соответствия “ключ => значение” и имеет несколько стандартных методов.

## 7. Что представляет из себя метод объекта в JavaScript?

Метод объекта в JavaScript – это просто функция, которая добавлена в ассоциативный массив.

## 8. Зачем в JavaScript перед переменной писать var?

Если создавать переменную через обычное присваивание – будет создана “глобальная переменная”.

Пример:

```
max = 100;
```

Если создавать переменную с использованием слова var, тогда будет создана “локальная переменная”, которая перестает существовать после завершения работы функции.

Пример:

```
var max = 100;
```

## 9. Есть две функции:

```
function f(a,b) { return a+b }
```

**И**

```
var f = function(a,b) { return a+b }
```

## Есть ли между ними разница? Если есть то какая?

Есть, разница в видимости функции. Вариант функции без `var` виден везде в текущей области видимости. В том числе и до самого определения функции. Вариант с `var` присваивает функцию переменной, поэтому такая функция видна только после определения.

## 10. Как создать массив в JavaScript?

```
var array = [elem0, elem1, elem2];

var empty = [];

var array = new Array(elem0, elem1, elem2);

var empty = new Array();
```

## 11. Можно ли в JavaScript использовать функцию в качестве конструктора?

Да. Вот так:

```
var A = function() {
  something here
}

var myA = new A();
```

## 12. Сколько и какие конструкции для циклов есть в JavaScript?

Три: `for`, `while` и `do...while`.

## 13. Что сделает код: `break me_baby; ?`

Выйдет из текущего блока цикла или `switch` на метку `"me_baby"`.

## 14. Можно ли задать массив таким образом: `var a = "a,b".split(',')`?

Да, можно.

## 15. Что выведет `alert(typeof null); ?`

Выведет сообщение `'object'`.

## 16. А это: `alert(null instanceof Object); ?`

Выведет сообщение `'false'`.

**17.  $0.1 + 0.2 == 0.3$  ?**

Нет, т.к. вычисленное значение будет равно  $0.30000000000000004$ . Это действие точности вычислений и проявляется она не только в JavaScript.

**18. Что выведет `alert(typeof NaN)`; ?**

'Number'

**19. Что выведет `alert(NaN === NaN)`; ?**

'false'

# 5. MYSQL

## 1. Что такое реляционная база данных?

Реляционная база данных – это база данных, основанная на реляционной модели данных. Реляционная модель ориентирована на организацию данных в виде двумерных таблиц. Каждая реляционная таблица представляет собой двумерный массив и обладает следующими свойствами:

- каждый элемент таблицы – один элемент данных;
- все ячейки в столбце таблицы однородные, то есть все элементы в столбце имеют одинаковый тип (числовой, символьный и т. д.);
- каждый столбец имеет уникальное имя;
- одинаковые строки в таблице отсутствуют;
- порядок следования строк и столбцов может быть произвольным.

## 2. Что такое первичный ключ?

Первичный ключ (primary key) – столбец, значения которого во всех строках различны. Первичные ключи могут быть логическими (естественными) и суррогатными (искусственными). Так, для воображаемой таблицы “Users” первичным ключом может стать столбец e-mail (ведь теоретически не может быть двух пользователей с одинаковым e-mail). Но на практике лучше использовать суррогатные ключи, т.к. их применение позволяет абстрагировать ключи от реальных данных. Кроме того, первичные ключи менять нельзя, но что если у пользователя сменится e-mail?

Суррогатный ключ представляет собой дополнительное поле в базе данных. Как правило, это порядковый номер записи (хотя вы можете задавать их на свое усмотрение, контролируя, чтобы они были уникальны).

## 3. Что такое нормализация и денормализация?

Нормализация – это процесс приведения базы данных к виду, в котором она будет соответствовать правилам нормальных форм. Нормализация сводит к минимуму количество избыточной информации. Ее целью является сохранять данные только один раз, но в нужном месте.

Нормализованная база данных исключает дублирование и многократное обслуживание данных, а также появление проблем с целостностью данных, возникающих при повторном вводе одинаковых данных.

Денормализация – это процесс осознанного приведения базы данных к виду, в котором она не будет соответствовать правилам нормализации. Обычно это необходимо для повышения производительности и скорости извлечения данных за счет увеличения избыточности данных.

Если приложению необходимо часто выполнять выборки, которые занимают слишком много времени (например, объединение данных из множества таблиц), то следует рассмотреть воз-



возможность проведения денормализации.

Возможное решение следующее: вынести результаты выборки в отдельную таблицу. Это позволит увеличить скорость выполнения запросов, но также означает появление необходимости в постоянном обслуживании этой новой таблицы.

Прежде чем приступить к денормализации, необходимо убедиться, что ожидаемые результаты оправдывают издержки, с которыми придется столкнуться.

#### 4. Что такое `mysql_pconnect`? Чем он отличается от `mysql_connect`?

При использовании функции `mysql_connect()` каждый раз открывается новое соединение с базой данных. После вызова `mysql_close()` или после завершения работы скрипта соединение закрывается.

Отличия `mysql_pconnect()` заключаются в том, что, во-первых, при вызове функции сначала ищется уже открытое (постоянное) соединение с базой (`persistent connection`), если его нет – создается новое. Во-вторых, после завершения работы скрипта и при вызове `mysql_close()` соединение с базой MySQL не закрывается, а остается открытым для последующего использования.

#### 5. Что такое MyISAM и InnoDB?

MyISAM и InnoDB – это типы таблиц.

#### 6. Чем они отличаются?

##### MyISAM

- не поддерживает транзакции и с этим связаны его основные недостатки и преимущества;
- в большинстве случаев он быстрее, так как нет расходов на транзакции;
- занимает меньше дискового пространства;
- меньше расход памяти на обновления;
- полнотекстовый индекс;
- быстрый INSERT, SELECT.

##### InnoDB

- поддержка транзакций;
- построчная блокировка. UPDATE не блокирует всю таблицу;
- отлично ведет себя при смешанной нагрузке (`insert|select|update|delete`).

## 7. Что такое SQL-инъекция?

SQL-инъекции – встраивание вредоносного кода в запросы к базе данных. С использованием SQL-инъекций злоумышленник может не только получить закрытую информацию из базы данных, но и, при определенных условиях, внести туда изменения.

Уязвимость по отношению к SQL-инъекциям возникает из-за того, что пользовательская информация попадает в запрос к базе данных без должной обработки: чтобы скрипт не был уязвим, требуется убедиться, что все пользовательские данные попадают во все запросы к базе данных в экранированном виде.

## 8. Есть две таблицы:

**users** - таблица с пользователями (**users\_id, name**)

**orders** - таблица с заказами (**orders\_id, users\_id, status**)

**1) Выбрать всех пользователей из таблицы users, у которых ВСЕ записи в таблице orders имеют status = 0**

**2) Выбрать всех пользователей из таблицы users, у которых больше 5 записей в таблице orders имеют status = 1**

1) Выбор пользователей с использованием вложенного запроса:

```
SELECT * FROM users WHERE users_id NOT IN (
SELECT users_id FROM orders WHERE status <> 0)
```

2) С использованием JOIN и HAVING:

```
SELECT u.* FROM orders o
JOIN users u ON u.users_id = o.users_id
WHERE o.status = 1 GROUP BY o.users_id
HAVING COUNT(o.status) > 5
```

## 9. Какая разница между LEFT, RIGHT и INNER JOIN?

Основное различие в том, как соединяются таблицы, если нет общих записей.

Простой JOIN – это то же самое, что INNER JOIN, он показывает только общие записи обеих таблиц. Каким образом записи считаются общими, определяется полями в join-выражении. Например, следующая запись:

```
FROM t1 JOIN t2 on t1.id = t2.id
```

означает что будут показаны записи с одинаковыми id, существующие в обеих таблицах.

**LEFT JOIN** (или **LEFT OUTER JOIN**) означает показывать все записи из левой таблицы (той, которая идет первой в join-выражении) независимо от наличия соответствующих записей в правой

таблице.

**RIGHT JOIN** (или **RIGHT OUTER JOIN**) действует в противоположность **LEFT JOIN** - показывает все записи из правой (второй) таблицы и только совпавшие из левой (первой) таблицы.

#### **LEFT JOIN:**

- при выполнении условия сцепления таблиц, к ячейкам из первой таблицы присоединяются ячейки второй;
- если условие не выполняется, присоединяются пустые ячейки.

#### **INNER JOIN:**

- при выполнении условия тоже, что и с **LEFT JOIN**;
- если условие не выполняется, строка вообще игнорируется (не будет ячеек даже из первой таблицы).

Проще говоря, **LEFT JOIN** выберет из первой таблицы все записи, даже если во второй таблице нет совпадений по какому-то условию. **INNER JOIN** выберет только те, что полностью соответствуют условию.

## **10. Чем отличается WHERE от HAVING?**

При помощи **HAVING** отражаются все предварительно сгруппированные посредством **GROUP BY** блоки данных, удовлетворяющие заданным в **HAVING** условиям. Это дополнительная возможность "профильтровать" выходной набор.

Условия в **HAVING** отличаются от условий в **WHERE**:

- В условии поиска **WHERE** нельзя задавать агрегатные функции;
- **HAVING** исключает из результирующего набора данных группы с результатами агрегированных значений;
- **WHERE** исключает из расчета агрегатных значений по группировке записи, не удовлетворяющие условию.

## **11. Что можешь сказать про команду GROUP BY?**

**GROUP BY** используется для группировки результата одного или нескольких столбцов.

Синтаксис:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

## 12. Приведи пример использования GROUP BY.

Есть следующая таблица "Orders":

Id	Date	Price	Customer
1	2012/02/12	1000	Vasya
2	2012/03/17	1600	Petro
3	2012/01/25	700	Vasya
4	2012/01/03	300	Vasya
5	2012/02/21	2000	Ivan
6	2012/02/18	100	Petro

Теперь мы хотим найти общую сумму заказа для каждого клиента. Выполним запрос:

```
SELECT Customer, SUM(Price) FROM Orders GROUP BY Customer
```

Результат запроса:

Customer	SUM(Price)
Vasya	2000
Petro	1700
Ivan	2000

Теперь давайте посмотрим, что произойдет, если мы не используем запрос GROUP BY:

```
SELECT Customer, SUM(Price) FROM Orders
```

Результат запроса:

Customer	SUM(Price)
Vasya	5700
Petro	5700
Vasya	5700
Vasya	5700
Ivan	5700
Petro	5700

## 13. Допустим, у тебя есть Интернет-магазин. Составь запрос, который покажет сколько денег принес каждый отдельно взятый покупатель в общей сложности за всё время существования магазина.

```
SELECT customer_name, SUM(order_price) FROM orders
GROUP BY customer_name;
```

## 14. А теперь пусть этот же запрос показывает только тех, кто купил товаров в общей сложности минимум на 10 тысяч евро.

```
SELECT customer_name, SUM(order_price) FROM orders
GROUP BY customer_name HAVING SUM(order_price) >= 10000;
```

## 15. Что делает команда EXPLAIN?

EXPLAIN может в точности рассказать, что происходит, когда мы выполняем запрос. Эта информация позволит нам обнаружить медленные запросы и сократить время, затрачиваемое на обработку запроса, что впоследствии может значительно ускорить работу вашего приложения.

Вот пример использования этой команды:

```
EXPLAIN SELECT * FROM users WHERE id='42'
```

Если оператор SELECT предваряется ключевым словом EXPLAIN, MySQL сообщит о том, как будет производиться обработка SELECT, и предоставит информацию о порядке и методе связывания таблиц.

При помощи EXPLAIN можно выяснить, когда стоит снабдить таблицы индексами, чтобы получить более быструю выборку, использующую индексы для поиска записей. Кроме того, можно проверить, насколько удачный порядок связывания таблиц был выбран оптимизатором. Заставить оптимизатор связывать таблицы в заданном порядке можно при помощи указания STRAIGHT\_JOIN.

Для непростых соединений EXPLAIN возвращает строку информации о каждой из использованных в работе оператора SELECT таблиц. Таблицы перечисляются в том порядке, в котором они будут считываться. MySQL выполняет все связывания за один проход (метод называется “single-sweep multi-join”). Делается это так: MySQL читает строку из первой таблицы, находит совпадающую строку во второй таблице, затем - в третьей, и так далее. Когда обработка всех таблиц завершается, MySQL выдает выбранные столбцы и обходит в обратном порядке список таблиц до тех пор, пока не будет найдена таблица с наибольшим совпадением строк. Следующая строка считывается из этой таблицы и процесс продолжается в следующей таблице.

## 16. Как вывести все поля из таблицы my\_table?

```
SELECT * FROM my_table
```

## 17. Как вывести только поля name\_first, name\_last, salary из таблицы my\_table?

```
SELECT name_first, name_last, salary FROM my_table
```

## 18. Таблице my\_table задать псевдоним t и вывести всех, у кого salary выше 3800

```
SELECT * FROM my_table AS t WHERE t.salary>3800
```

## 19. Выбрать страны, из которых поставляют продукцию производители, так, чтобы страны не повторялись по 2 и более раз.

```
SELECT DISTINCT country FROM manufacturers
```

**20. Вывести всех украинских производителей.**

```
SELECT * FROM manufacturers WHERE country="Ukraine"
```

**21. Вывести только тех производителей, которые находятся во Львове и Харькове.**

```
SELECT * FROM manufacturers WHERE area="Львов" OR area="Харьков"
```

**22. Вывести все модели автобусов ЛАЗ, вместимостью не менее 15 пассажиров.**

```
SELECT * FROM buses WHERE brand="ЛАЗ" AND NOT seats<15
```

**23. Вывести все автобусы в порядке возрастания количества мест.**

```
SELECT * FROM buses ORDER BY seats
```

**24. Вывести все автобусы в порядке уменьшения количества мест.**

```
SELECT * FROM buses ORDER BY seats DESC
```

**25. Какие знаешь команды для подсчета значений поля?**

```
SELECT MAX(seats) FROM buses // Выведет автобус с максимальным количеством сидений
SELECT MIN(seats) FROM buses // Выведет автобус с минимальным количеством сидений
SELECT SUM(seats) FROM buses // Выведет общее количество сидений во всех автобусах
SELECT AVG(seats) FROM buses // Выведет среднее количество сидений
SELECT COUNT(*) FROM buses // Выведет общее количество автобусов в таблице
SELECT COUNT(*) FROM buses WHERE brand="ЛАЗ" // Выведет количество автобусов ЛАЗ
```

**26. Предположим, у нас есть таблица в которой есть поля name и id. Нужно вывести имя с наибольшим id, не используя при этом команду MAX. Как это можно сделать?**

Отсортировать по id в сторону уменьшения, но вывести только первый id. Он и будет наибольшим:

```
SELECT name, id FROM customers ORDER BY id DESC LIMIT 1
```

**27. С помощью конструкции IN вывести производителей из Украины, Германии и США.**

```
SELECT * FROM manufacturer WHERE country IN
("Украина", "Германия", "США")
```

## 28. Вывести всех производителей за исключением тех, которые находятся в Китае, Таджикистане и России.

```
SELECT * FROM manufacturer WHERE country NOT IN
("Китай", "Таджикистан", "Россия")
```

## 29. Вывести пустые / не пустые значения.

пустые:

```
SELECT * FROM manufacturer WHERE location IS NULL
```

не пустые:

```
SELECT * FROM manufacturer WHERE location IS NOT NULL
```

## 30. Вывести только те автобусы, названия которых начинаются на букву М.

```
SELECT * FROM buses WHERE brand LIKE "M%"
```

## 31. Мы не помним как точно пишется "Mercedes" или "Mersedes", но нужно из таблицы выбрать автобусы именно этой марки. Как быть?

Воспользоваться знаком подчеркивания, который означает "любой символ":

```
SELECT * FROM buses WHERE brand LIKE "Mer_edes"
```

## 32. Выбрать только те автобусы, цена которых лежит в пределах от 100000 до 180000 долларов включительно.

```
SELECT * FROM buses WHERE price BETWEEN 100000 AND 180000
```

## 33. Подсчитать количество автобусов в таблице, у которых 45 мест.

```
SELECT COUNT(brand) FROM buses WHERE seats=45
```

## 34. Приведи пример вложенного запроса.

```
SELECT * FROM buses WHERE price=(SELECT MAX(price) FROM buses)
```

## 35. Можно ли выбрать данные из нескольких таблиц?

Да, например вот так:

```
SELECT o.order_no, o.amount_paid, c.company
FROM orders AS o
LEFT JOIN customer AS c ON (c.custno=o.custno)
```

### 36. Обязательно ли писать команды прописными буквами? Сработает ли запрос, если его написать строчными буквами?

Работать будет, т.к. команды MySQL регистронезависимы. Но, тем не менее, команды традиционно принято писать прописными буквами для удобства чтения запроса.

### 37. А будет ли работать такой запрос: `SELECT city FROM customers?`

Будет. MySQL не обращает внимания на пробелы и разрывы строки между командами.

### 38. Что выберет такой запрос: `SELECT brand FROM buses WHERE brand REGEXP 'lvo|ool'?`

Выберет все автобусы, в названиях которых встречается "lvo" или "ool". Например, Volvo и Van Hool.

### 39. У нас есть таблица `usa`, в которой есть колонка `city` с названиями городов и колонка `state`, с названиями штатов. Нужно вывести новую колонку `new_column`, которая будет содержать название города и через запятую название штата, в котором находится этот город. Например вот так: `Raleigh, NC`.

```
SELECT CONCAT(city, ', ', state) AS new_column FROM usa
```

### 40. Есть таблица `one`, к ней нужно добавить таблицу `two`, которая содержит такие же поля (`id`, `name`, `seller`, `price`). Как это сделать?

```
INSERT INTO one(id, name, seller, price) SELECT id, name, seller, price FROM two
```



# 6. CSS

## 1. В чем разница между записью #my и .my?

#my – селектор ай-ди.

.my – селектор класса.

## 2. В чем разница между margin и padding?

margin – внешний отступ (снаружи от границ блока до остальных элементов страницы);

padding – внутренний отступ (внутри от границ блока до контента).

Допустим, у нас есть некий объект с рамкой вокруг него. Вот margin – это внешнее поле, а padding внутреннее. Или есть у нас абзац текста. Следовательно, margin – это отступ от других соседних объектов (левой, правой стороны экрана, соседних абзацев, разных окружающих картинок). А padding – это внутренние отступы. Если мы зададим padding: 50px для этого абзаца, то значит, сами буквы будут начинаться не от края абзаца, а с отступом в 50 пикселей.

## 3. Почему таблицы стилей CSS называются каскадными?

Слово “каскадные” говорит о том, что на вывод каждого тега в документе могут оказывать влияние сразу несколько стилевых спецификаций, образующих иерархическую систему. Например, поверх спецификаций, относящихся к конкретному документу, может действовать стилевой файл, общий для всех документов на сервере.

## 4. Что такое альтернативная таблица стилей?

Альтернативная таблица стилей – это таблица, определяющая стили, которые будут использованы взамен стилей, использующихся по умолчанию.

Например, пользователь может сделать выбор, в зависимости от своих предпочтений, если мы заранее подготовим одну таблицу стилей для маленького экрана, а другую – для слабовидящих (с крупными шрифтами). Альтернативные стили позволяют пользователю сделать выбор на более подходящего из них.

## 5. Какая разница между значениями 0 и auto в свойстве margin?

В вертикальных полях, auto всегда означает 0. В горизонтальных полях, auto означает 0 только тогда, если свойство width также auto.

## 6. Для чего применяются свойства border-position и border-all?

Таких свойств в CSS нет.

## 7. Какое свойство задает цвет фона?

`background-color.`

## 8. Как в CSS обозначаются комментарии?

```
/* вот так */
```

## 9. Как задать красный цвет для всех элементов, имеющих класс red?

```
.red { background-color:red; }
```

## 10. Как убрать подчеркивание для всех ссылок на странице?

```
a { text-decoration:none; }
```

## 11. Как сделать жирным текст во всех элементах <p>?

```
p { font-weight: bold; }
```

## 12. Что делает свойство clear?

Устанавливает, с какой стороны элемента запрещено его обтекание другими элементами. Если задано обтекание элемента с помощью свойства `float`, то `clear` отменяет его действие для указанных сторон.

## 13. Нарисуй лимон средствами CSS3!

```
<div class="lemon"></div>
<style>
.lemon {
width: 200px; height: 200px;
background: #F5F240;
border: 1px solid #F0D900;
border-radius: 10px 150px 30px 150px;
}
</style>
```

**13. Есть ячейка таблицы, в ней 3 дива, в каждом диве по слову. На первый див применен стиль `float: left`; на третий: `float: right`; что нужно применить на второй див, чтобы все три надписи/дивы были в одной горизонтальной строке, т.е. первый имел бы выравнивание по левому краю, третий по правому, а второй по центру?**

Можно сделать так:

```
<t:div style="width:33%; float:left; text-align: left;">AM </t:div>
<t:div style="width:34%; float:left;">NOON </t:div>
<t:div style="width:33%; float:left; text-align: right;">PM </t:div>
```

## 14. Что такое псевдоклассы и псевдостили?

Псевдоклассы – это то, что обычно пишут после селектора через двоеточие чтобы определить реакцию или состояние для данного селектора. Самые известные псевдоклассы это `:link`, `:hover`, `:visited` и `:active`. Псевдоклассы обладают следующей структурой: `selector:pseudo class { property: value; }`, т.е. нужно всего лишь поместить двоеточие между селектором и псевдоклассом.

```
a.snowman:link {
    color: blue;
}

a.snowman:visited {
    color: purple;
}

a.snowman:active {
    color: red;
}

a.snowman:hover {
    text-decoration: none;
    color: blue;
    background-color: yellow;
}
```

## 15. Название и назначение псевдоклассов.

**:link** – отвечает за стили непосещенной ссылки;

**:visited** – состояние посещенной ссылки;

**:active** – состояние активного объекта;

**:hover** – состояние объекта при наведении на него мышкой;

**:focus** – фокус на объекте;

**:first-child** – первый дочерний элемент текущего элемента;

**:last-child** – последний дочерний элемент;

**:only-child** – применяет стиль к элементу, если он единственный дочерний элемент;

**:nth-child(5)** – в данном случае пятый по счету с начала дочерний элемент;

**:nth-last-child(3)** – аналогично, только отчет с конца. В этом, как и в предыдущих селекторах можно задавать не только конкретные цифры, но и писать, например, что-то типа такого: `ul li:nth-last-child(2n+1);`

**:root** – дает указание применить стиль к корневому элементу (в html документе это тег `<html>`);

**:not()** – дает ограничение на применение стилей по селектору (то есть селектор `.white-block:not(div)` применит указанный стиль ко всем элементам с классом `.white-block`, только если этот элемент не `<div>`);

**:empty** – выбирает пустые элементы;

**:first-of-type** – применяет стиль к первому элементу этого типа, то есть если у вас есть два `div`, стиль будет работать только для первого из них;

**:last-of-type** – аналогично предыдущему, только для последнего элемента;

**:only-of-type** – применяет стили к элементу, если он имеет уникальный тип внутри своего родителя;

**:nth-of-type()** – выбирает указанный по счету с начала элемент текущего типа;

**:nth-last-of-type()** – тоже самое, но с конца;

**:target** – например, если у вас адрес имеет вид `index.html#anchor`, то этот псевдокласс задаст правило для элемента с `id="anchor"`;

**:enabled** – выбирает активные инпуты;

**:disabled** – выбирает неактивные инпуты;

**:checked** – отмеченные чекбоксы и выбранные радиокнопки;

**:default** – элемент по умолчанию, например кнопка отправки формы;

**:valid** – стиль для правильного инпута (когда указана `data type` в HTML 5);

**:invalid** – когда, соответственно, инпут невалиден;

**:in-range** – когда значение инпута находится в заданных границах (`type="range"`, задан `min` и `max`, но это все только в HTML 5);

**:out-of-range** – когда не попадает в границы;

**:required** – все обязательные поля;

**:optional** – все необязательные;

**:read-only** – те элементы, которые доступны только для чтения;

**:read-write** – для чтения и записи.

# 7. JQUERY

## 1. Как подключить JQuery к веб-странице?

```
<head>
<script type='text/javascript' src='jquery.js'></script>
</head>
```

## 2. В чем вообще смысл jQuery? Зачем оно надо?

Суть jQuery в том, чтобы отбирать элементы HTML-страниц и выполнять над ними определенные действия.

## 3. Выбрать все элементы с id = idname

```
$('#idname');
```

## 4. Выбрать все элементы div с id = idname

```
$('#div#idname');
```

## 5. Выбрать все элементы с class = classname

```
$('.classname');
```

## 6. Выбрать все элементы div с class = classname

```
$('#div.classname');
```

## 7. Выбрать все span элементы в элементах div

```
$('#div span');
```

или так:

```
$('#div').find('span');
```

## 8. Выбрать все div и span элементы

```
$('#div, span');
```

## 9. Выбрать предыдущий элемент от найденного

```
$('#banner').prev();
```

## 10. Выбрать следующий элемент от найденного

```
$('#banner').next();
```

## 11. Выбрать все span элементы в элементах div, где span является прямым потомком div'a

```
$('#div > span');
```

## 12. Выбрать все span после первого элемента div

```
$('#div ~ span');
```

## 13. Выбрать первый li в ul

```
$('#ul li:first-child');
```

## 14. Выбрать div`ы у которых нет класса cls

```
$('#div:not(.cls)');
```

## 15. Выбрать элементы с активной анимацией

```
$('#div:animated');
```

## 16. Выбрать div`ы которые содержат класс firstclass и класс secondclass

```
$('#div.firstclass').filter('.secondclass');
```

## 17. Выбрать все div`ы с атрибутом title = test

```
$('#div[title='test']');
```

## 18. Выбрать все input с type = radio

```
$('#input:radio');
```

## 19. Выбрать видимый div с именем red, который содержит тег span

```
$("#div[name=red]:visible:has(span)");
```

## 20. Что выберет этот фильтр? \$("a[rel~='external']");

Все <a> с атрибутом rel содержащим external в списке значений разделенных пробелом.

## 21. А этот? \$("div[name=apple]:visible:has(p)");

Выберет видимый div с именем apple, который содержит тег p.

## 22. Найти все элементы div с классом one, а также все элементы p с классом two, затем добавить им всем класс three и визуально плавно спустить вниз.

```
$("#div.one").add("p.two").addClass("three").slideDown("slow");
```

## 23. Сделать так, чтобы при нажатии на элемент <a> алертом выводилось "Hello world!".

```
jQuery(function($) {  
    $("a").click(function() {  
        alert("Hello world!");  
    });  
});
```

# 8. HTML

## 1. HTML5 знаешь? Нарисуй что-нибудь.

```
<rect
x="100" y="100"
width = "300" height = "300"
fill = "blue" stroke = "red"
stroke-width = "50px"
rx = "8" ry = "8"
id = "myRect" class = "chart" />
```

Нарисует синий квадрат с красной рамкой с закругленными краями.

## 2. Чем HTML отличается от XHTML?

XHTML представляет собой словарь XML, в то время как HTML – это лишь предшествующий XHTML язык разметки.

Основное различие между ними заключается в обработке документа. Документы XHTML обрабатываются своим модулем (парсером) аналогично документам XML. В процессе этой обработки ошибки, допущенные разработчиками, не исправляются.

Более подробно о различиях:

- Все элементы должны быть закрыты. Теги, которые не имеют закрывающего тега (например, `<img>` или `<br>`) должны иметь на конце `/` (например, `<br />`).
- Булевы атрибуты записываются в развёрнутой форме. Например, следует писать `<option selected="selected">` или `<td nowrap="nowrap">`.
- Имена тегов и атрибутов должны быть записаны строчными буквами (например, `<img alt="" />` вместо `<IMG ALT="" />`).
- XHTML гораздо строже относится к ошибкам в коде; `<` и `&` везде, даже в URL, должны замещаться `&lt;` и `&amp;` соответственно. По рекомендации W3C браузеры, встретив ошибку в XHTML, должны сообщить о ней и не обрабатывать документ. Для HTML браузеры должны были попытаться понять, что хотел сказать автор.
- Кодировкой по умолчанию является UTF-8 (в отличие от HTML, где кодировкой по умолчанию является ISO 8859-1).

## 3. Что такое DOCTYPE и зачем он нужен?

DOCTYPE – это определение типа документа (Document Type Definition (DTD)), правила, в соответствии с которыми осуществляется проверка конкретного документа (веб-страницы) XML или (X)HTML. Благодаря этой записи, браузер определяет, какая в данном документе используется версия DTD.

А нужен он для того, чтобы браузеры правильно отображали разметку документа. Если не ука-



зать DOCTYPE, то браузер будет добавлять “отсебятину”, причём каждые браузеры будут добавлять свою “отсебятину”. В результате, ни о какой кроссбраузерности не может и речи идти.

#### 4. Чем отличается div от span?

div – это блочный контейнер, а span - линейный.

Контейнеры – это то, во что может быть вложено что-нибудь еще.

div используется для разметки блоков, а span – для текста.

div формирует блок из того, что в нем с новой строки и после него элемент идет с новой строки (если не поменять поведение с помощью CSS), а span не переносит и “обтягивает”

#### 5. Как обозначаются комментарии в HTML?

```
<!-- вот так-->
```

#### 6. Ссылки. Как задать адрес документа, на который следует перейти?

```
<a href="http://vottak.com/example/example.html">Абсолютная ссылка</a>
```

```
<a href="../../../example/example.html">Относительная ссылка</a>
```

#### 7. Как сделать ссылку на e-mail?

```
<a href="mailto:jeffjoellennox@mail.ru">E-mail me</a>
```

#### 8. Что делают теги <em>?

Выводят заключенный в них текст курсивом.

#### 9. Что такое <ol>, <ul> и <li>

Тег <ol> устанавливает нумерованный список. Каждый элемент списка должен начинаться с тега <li>. Если к тегу <ol> применяется таблица стилей, то элементы <li> наследуют эти свойства.

Тег <ul> устанавливает ненумерованный (маркированный) список.

#### 10. Зачем нужны теги <dl>, <dt>, <dd> ?

Теги <dl>, <dt>, <dd>, предназначены для создания списка определений. Каждый такой список начинается с контейнера <dl>, куда входит тег <dt> создающий термин и тег <dd> задающий определение этого термина. Закрывающий тег </dd> не обязателен, поскольку следующий тег сообщает о завершении предыдущего элемента. Тем не менее, хорошим стилем является закрывать все теги.

## 11. Зачем нужны теги <tr>, <th>, <td> ?

Тег <tr> служит контейнером для создания строки таблицы. Каждая ячейка в пределах такой строки может задаваться с помощью тега <th> или <td>.

<th> предназначен для создания одной ячейки таблицы, которая обозначается как заголовочная. Текст в такой ячейке отображается браузером обычно жирным шрифтом и выравнивается по центру. Тег <th> должен размещаться внутри контейнера <tr>, который в свою очередь располагается внутри тега <table>.

<td> предназначен для создания одной ячейки таблицы. Тег <td> должен размещаться внутри контейнера <tr>, который в свою очередь располагается внутри тега <table>.

## 12. В каком регистре лучше писать HTML-код?

Раньше считалось, что это безразлично. Но с приходом XHTML на этот вопрос появился однозначный ответ - все теги, атрибуты и предопределенные значения пишем в нижнем регистре.

## 13. Как убрать синюю рамку вокруг картинки-ссылки?

Вот так:

```
<img border="0" ... />
```

## 14. Обязательно ли писать alt в <img>?

Да. Этого требует стандарт, да и здравый смысл, т.к. некоторые пользователи все еще отключают картинки, а некоторые используют текстовые или даже голосовые браузеры. Если картинка не несет никакой смысловой нагрузки (элемент дизайна, “распорка” и т.п.), то можно поставить в тег <img> значение alt="".

## 15. Что такое entities?

Entities – это комбинации знака & и буквенного или цифрового кода после нее, предназначенные для замещения символов, которые не могут встречаться в “чистом” виде в HTML-тексте, например, символа “<”.

## 16. Как сделать чтобы все гиперссылки сайта открывались в новом окне, т.е чтобы по умолчанию использовался target="\_blank"?

Нужно в области head прописать тег base с атрибутом target="\_blank":

```
<head>
<base target="_blank">
</head>
```

## **17. А как теперь быть, если какую-то из гиперссылок я захочу открыть в этом же окне, т.е. не создавая новое?**

В тег `<a>` этой ссылки вам уже нужно будет добавить атрибут `target="_self"`, ибо по умолчанию сейчас у нас используется `target="_blank"`..

# 9. РАЗНОЕ

## 1. Что такое сериализация?

Это процесс перевода какой-либо структуры данных в последовательность битов.

## 2. А десериализация?

Операция, обратная сериализации, т.е. восстановление начального состояния структуры данных из битовой последовательности.

## 3. Зачем они нужны?

Сериализация используется для передачи объектов по сети и для сохранения их в файлы. Например, нужно создать распределённое приложение, разные части которого должны обмениваться данными со сложной структурой. В таком случае для типов данных, которые предполагается передавать, пишется код, который осуществляет сериализацию и десериализацию. Объект заполняется нужными данными, затем вызывается код сериализации, в результате получается, например, XML-документ. Результат сериализации передаётся принимающей стороне, например, по HTTP. Приложение-получатель создаёт объект того же типа и вызывает код десериализации, в результате получая объект с теми же данными, что были в объекте приложения-отправителя. По такой схеме работает, например, сериализация объектов через SOAP в Microsoft .NET.

## 4. Что такое Apache?

Apache HTTP – бесплатный веб-сервер.

## 5. А mod\_rewrite?

Для веб-сервера Apache существует мощный модуль mod\_rewrite, который включается в стандартный дистрибутив. Данный модуль позволяет выполнять на лету URL преобразования. Механизм преобразования основывается на использовании правил, а правила в свою очередь представляют из себя регулярные выражения.

Модуль mod\_rewrite поддерживает неограниченное количество правил и связанных с каждым правилом условий, реализуя действительно гибкий и мощный механизм управления URL.

Для получения URL нужного вида могут использоваться разные источники данных, например переменные сервера, переменные окружения, HTTP заголовки, время и даже запросы к внешним базам данных в разных форматах.

Модуль оперирует с полными URL (включая path-info) и в контексте сервера (httpd.conf) и в контексте каталога (.htaccess) и даже может генерировать части строки запроса в качестве результата.

## 6. Аббревиатуры SVN и CVS о чем-то говорят?

SVN (Subversion) и CVS (Concurrent Versions System) – это системы управления версиями.

## 7. А что это за системы такие?

Система управления версиями – это программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости, возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение и многое другое.

## 8. Существуют ли еще какие-то системы управления версиями?

Да. Git, Mercurial, Bazaar...

## 9. Что такое сессии? Где хранятся сессии: на стороне клиента или на стороне сервера?

Сессия (http session) – это логический объект, который позволяет передавать данные между последовательными HTTP запросами от одного пользователя. Сессии создаются в PHP на стороне сервера и там же хранятся. Собственно, сессии, если в двух словах – это механизм, позволяющий однозначно идентифицировать браузер и создающий для этого браузера файл на сервере, в котором хранятся переменные сеанса.

По возможности лучше не хранить большое количество информации в сессии. Если информация не критичная (например, настройки пользовательского интерфейса), то лучше хранить ее в cookies на стороне клиента.

## 10. Как долго "живет" сессия?

Время жизни куки, которая хранит идентификатор сесии по умолчанию составляет ноль секунд, т.е. кука храниться до того момента, как браузер будет закрыт.

## 11. А при открытом браузере?

У томката по дефолту 30 минут, но это можно настроить.

## 12. Возможно ли как-то продлить время жизни сессии?

Да. Это можно изменить при помощи параметра `session.cookie_lifetime` в файле `php.ini`.

PHP, как правило, хранит данные сессии в файлах. Для того, чтобы не засорять файловую систему периодически осуществляется очистка устаревших данных, `garbage collection` – уборка мусора. В файле `php.ini` существует параметр `session.gc_maxlifetime` который указывает время в секундах, по истечению которого все устаревшие данные будут удалены из файловой системы.

Допустим, мы хотим задать время жизни сессии – 3 часа. Для этого в файле `php.ini` устанавливаем следующие параметры:

```
session.gc_maxlifetime = 10800  
session.cookie_lifetime = 10800
```

### 13. Зачем нужны сессии?

Дело в том, что веб-сервер не поддерживает постоянного соединения с клиентом, и каждый запрос обрабатывается как новый, безо всякой связи с предыдущими. То есть, нельзя ни отследить запросы от одного и того же посетителя, ни сохранить для него переменные между просмотрами отдельных страниц. Вот для решения этих двух задач и были изобретены сессии.

### 14. Как устроены, и как работают сессии?

Для начала надо как-то идентифицировать браузер. Для этого надо выдать ему уникальный идентификатор и попросить передавать его с каждым запросом.

Идентификатор – это обычная переменная. Для PHP по умолчанию ее имя – `PHPSESSID`. Задача PHP отправить ее браузеру, чтобы тот вернул ее со следующим запросом.

Переменную можно передать только двумя способами: в куках или `POST/GET` запросом. PHP использует оба варианта.

За это отвечают две настройки в `php.ini`:

`session.use_cookies` - если равно 1, то PHP передает идентификатор в куках, если 0 - то нет.

`session.use_trans_sid` если равно 1, то PHP передает его, добавляя к URL и формам, если 0 - то нет.

### 15. Поясни разницу между HTTP методами GET и POST.

GET передает данные серверу используя URL, а POST передает данные, используя тело HTTP запроса.

Длина URL'a ограничена 1024 символами, что и будет верхним пределом для данных, которые можно отослать GET'ом. POST может отправлять гораздо большие объемы данных.

Кроме того, передача данных методом POST более безопасна, чем методом GET, так как секретные данные (например пароль) не отображаются напрямую в web-клиенте пользователя (в отличие от URL, который виден почти всегда).

### 16. Что такое стандарты W3C?

Консорциум всемирной паутины (World Wide Web Consortium, W3C) – организация, разрабатывающая и внедряющая технологические стандарты для Всемирной паутины.

W3C разрабатывает для Интернета единые принципы и стандарты, называемые “рекомендаци-

ями”, которые затем внедряются производителями программ и оборудования. Таким образом достигается совместимость между программными продуктами и аппаратурой различных компаний, что делает Всемирную сеть более совершенной, универсальной и удобной.

## 17. По какому протоколу осуществляется передача данных в сети Интернет?

Стек протоколов TCP/IP – это два протокола нижнего уровня, являющиеся основой связи в сети Интернет. Протокол TCP (Transmission Control Protocol) разбивает передаваемую информацию на порции и нумерует все порции. С помощью протокола IP (Internet Protocol) все части передаются получателю. Далее с помощью протокола TCP проверяется, все ли части получены. При получении всех порций TCP располагает их в нужном порядке и собирает в единое целое.

## 18. Назови наиболее известные протоколы, используемые в сети Интернет?

**HTTP** (Hyper Text Transfer Protocol) – это протокол передачи гипертекста. Протокол HTTP используется при пересылке Web-страниц с одного компьютера на другой.

**FTP** (File Transfer Protocol) – это протокол передачи файлов со специального файлового сервера на компьютер пользователя. FTP дает возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удаленным компьютером, пользователь может скопировать файл с удаленного компьютера на свой или скопировать файл со своего компьютера на удаленный.

**POP** (Post Office Protocol) – это стандартный протокол почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ.

**SMTP** (Simple Mail Transfer Protocol) – протокол, который задает набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приеме, либо сообщение об ошибке, либо запрашивает дополнительную информацию.

## 19. Что такое Document Object Model (DOM) ?

Это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

## 20. Что такое AJAX и как он работает?

AJAX – это подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в “фоновом” обмене данными браузера с веб-сервером. В результате, при обновлении данных, веб-страница не перезагружается полностью, и веб-приложения становятся более быстрыми и удобными.

AJAX – не самостоятельная технология, а концепция использования нескольких смежных технологий. AJAX базируется на двух основных принципах:

1) Использование технологии динамического обращения к серверу “на лету”, без перезагрузки всей страницы полностью, например:

- с использованием XMLHttpRequest (основной объект);
- через динамическое создание дочерних фреймов;
- через динамическое создание тега <script>.

## 2) Использование DHTML для динамического изменения содержания страницы;

В качестве формата передачи данных обычно используются JSON или XML.

## 21. Что такое JSON?

Это текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми.

## 22. А XML?

XML – это расширяемый язык разметки, фактически представляющий собой свод общих синтаксических правил. Текстовый формат, предназначенный для хранения структурированных данных для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки (например, XHTML).

## 23. Приведи пример XML-документа.

```
<?xml version="1.0" encoding="utf-8"?>
<book>
<title>"PHP. Собеседование в вопросах и ответах"</title>
<author>Шевченко А.А.</author>
<description>В этой книге собраны наиболее часто встречающиеся вопросы на собеседовании на должность PHP-разработчика</description>
</book>
```

## 24. Что такое XML Schema?

XML Schema – это язык описания структуры XML-документа. Как большинство языков описания XML, XML Schema была задумана для определения правил, которым должен подчиняться документ. Но, в отличие от других языков, XML Schema была разработана так, чтобы её можно было использовать в создании программного обеспечения для обработки документов XML.

## 25. Что такое JSON Schema?

JSON Schema – один из языков описания структуры JSON документа. Использует синтаксис JSON. JSON Schema – самоописательный язык: при его использовании для обработки данных и описания их допустимости могут использоваться одни и те же инструменты сериализации/десериализации.

## 26. Что такое XMLHttpRequest?

Объект XMLHttpRequest дает возможность браузеру делать HTTP-запросы к серверу без перезагрузки страницы. Несмотря на слово XML в названии, XMLHttpRequest может работать с данными



в любом текстовом формате.

## **28. Что такое API?**

API (Application Programming Interface) – это интерфейс программирования приложений, набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах.