*Лекция 7: Работа с массивами данных*

### **Массивы**

В одной из первых лекций мы рассказывали о том, как можно *создать массив* данных. Напомним, что *массив* можно *создать* двумя способами:

1. С помощью конструкции array
2. $array\_name = array("key1"=>"value1",

"key2"=>"value2");

1. Непосредственно задавая *значения* элементам *массива*

$array\_name["key1"] = value1;

Например, нам нужно хранить *список* документов, которые будут удалены из *базы данных*. Естественно хранить его в виде *массива*, *ключом* в котором будет *идентификатор* документа (его уникальный номер), а *значением* – название документа. Этот *массив* можно *создать* таким образом:

<?

$del\_items = array("10"=>"Наука и жизнь",

"12"=>"Информатика");

$del\_items["13"] = "Программирование на Php";

// добавляем элемент в массив

?>

#### **Операции с массивами**

*Массив* – это тип данных, с данными этого типа должны быть определены *операции*. Какие же *операции* можно производить с *массивами*? *Массивы* можно *складывать* и *сравнивать*.

*Складывают массивы* с помощью стандартного оператора " + ". Вообще говоря, эту *операцию* по отношению к *массивам* точнее назвать объединением. Если у нас есть два *массива*, $a и $b, то результатом их *сложения* (объединения) будет *массив* $c, состоящий из элементов $a, к которым справа дописаны элементы *массива* $b. Причем, если встречаются совпадающие *ключи*, то в результирующий *массив* включается элемент из первого *массива*, т.е. из $a. Таким образом, если *складываются массивы* в языке PHP, от перемены мест слагаемых сумма меняется.

<?

$a = array("и"=>"Информатика",

"м"=>"Математика");

$b = array("и"=>"История","м"=>"Биология",

"ф"=>"Физика");

$c = $a + $b;

$d = $b + $a;

print\_r($c);

/\* получим: Array([и]=>Информатика

[м]=>Математика [ф]=>Физика) \*/

print\_r($d);

/\* получим: Array([и]=>История

[м]=>Биология [ф]=>Физика) \*/

?>

Пример 7.1. Сложение массивов

*Сравнивать массивы* можно, проверяя их равенство или неравенство либо эквивалентность или неэквивалентность. Равенство *массивов*– это когда совпадают все пары *ключ* / *значение* элементов *массивов*. Эквивалентность – когда кроме равенства *значений* и *ключей*элементов требуется еще, чтобы элементы в обоих *массивах* были записаны в одном и том же порядке. Равенство *значений* в PHP обозначается символом " == ", а эквивалентность – символом " === ".

<?

$a = array("и"=>"Информатика",

"м"=>"Математика");

$b = array("м"=>"Математика",

"и"=>"Информатика");

if ($a == $b) echo "Массивы равны и";

else echo "Массивы НЕ равны и ";

if ($a === $b) echo " эквивалентны";

else echo " НЕ эквивалентны";

// получим echo "Массивы равны и

НЕ эквивалентны"

?>

Пример 7.2. Сравнение массивов

Далее рассмотрим еще одну важную *операцию с массивом* – подсчет *количества его элементов*. Для ее реализации в PHP есть специальная функция.

##### **Функция count**

Не раз уже мы использовали функцию *count()* , чтобы вычислить *количество элементов массива*. На самом деле эта функция вычисляет число элементов в переменной вообще. Если применить ее к любой другой переменной, она возвратит 1. Исключение составляет переменная типа NULL – *count(NULL)* есть 0. Кроме того, применяя эту функцию к многомерному *массиву*, чтобы получить *число его элементов*, нужно использовать дополнительный параметр COUNT\_RECURSIVE.

<?

$del\_items = array("langs" => array(

"10"=>"Python", "12"=>"Lisp"),

"other"=>"Информатика");

echo count($del\_items) . "<br>";

// выведет 2

echo count($del\_items,COUNT\_RECURSIVE);

// выведет 4

?>

Пример 7.3. Применение функции count()

Мы не будем повторять все, что было сказано о *массивах* в предыдущих лекциях. В этой лекции мы рассмотрим некоторые встроенные функции для работы с *массивами*. И начнем мы с функций для *поиска значений в массиве*.

##### **Функция in\_array**

in\_array("искомое значение","массив",

["ограничение на тип"]);

позволяет установить, содержится ли в заданном *массиве* искомое *значение*. Если третий аргумент задан как true, то в *массиве*нужно найти элемент, совпадающий с искомым не только по *значению*, но и по типу. Если искомое *значение* – строка, то сравнение *чувствительно к регистру*.

Например, имеется *массив* неизученных нами языков программирования. Мы хотим узнать, содержится ли в этом *массиве* язык PHP. Напишем следующую программу:

<?php

$langs = array("Lisp","Python","Java",

"PHP","Perl");

if (in\_array("PHP",$langs,true))

echo "Надо бы изучить PHP<br>";

// выведет сообщение "Надо бы изучить PHP"

if (in\_array("php",$langs))

echo "Надо бы изучить php<br>";

// ничего не выведет, поскольку в массиве

// есть строка "PHP", а не "php"

?>

В качестве искомого *значения* этой функции может выступать и *массив*. Правда, это свойство было добавлено только начиная с PHP 4.2.0.

Например:

<?php

$langs = array("Lisp","Python",array("PHP","Java"),"Perl");

if (in\_array(array("PHP","Java"),$langs))

echo "Надо бы изучить PHP и Java<br>";

?>

##### **Функция array\_search**

Это еще одна функция для *поиска значения в массиве*. В отличие от *in\_array* в результате работы ***array\_search*** возвращает *значение* *ключа*, если элемент найден, и ложь – в противном случае. А вот синтаксис у этих функций одинаковый:

array\_search("искомое значение","массив",

["ограничение на тип"]);

Сравнение строк *чувствительно к регистру*, а если указан опциональный аргумент, то сравниваются еще и типы *значений*. До PHP 4.2.0, если искомое *значение* не было найдено, эта функция возвращала ошибку или пустое *значение* NULL.

**Пример 7.4**.Теперь, наоборот, пусть у нас есть *массив* языков программирования, которые мы знаем. Причем *ключом* каждого элемента является номер, указывающий, каким по счету был изучен этот язык.

<?php

$langs = array("","Lisp","Python","Java",

"PHP","Perl");

if (!array\_search("PHP",$langs))

echo "Надо бы изучить PHP<br>";

else {

$k = array\_search("PHP",$langs);

echo "PHP я изучила ."$k."-м";

}

?>

Пример 7.4. Применение функции array\_search()

В результате мы получим строчку:

PHP я изучила 4-м

Очевидно, что эта функция более функциональна, чем *in\_array* , поскольку мы не только получаем информацию о том, что искомый элемент в *массиве* есть, но и узнаем, где именно в *массиве* он находится. А что будет, если искомых элементов в *массиве* несколько? В таком случае функция *array\_search()* вернет *ключ* первого из найденных элементов. Чтобы получить *ключи* всех элементов, нужно воспользоваться функцией ***array\_keys()*** .

##### **Функция array\_keys**

Функция *array\_keys()* выбирает все *ключи* *массива*. Но у нее имеется дополнительный аргумент, с помощью которого можно получить список *ключей* элементов с конкретным *значением*. Синтаксис этой функции таков:

array\_keys ("массив",

["значение для поиска"])

Функция *array\_keys()* возвращает как строковые, так и числовые *ключи* *массива*, организуя все *значения* в виде нового *массива* с числовыми индексами.

**Пример 7.5**.Мы записали *массив* языков, которые изучили. Список был длинным, и некоторые языки были записаны несколько раз. У нас возникло подозрение, что один из таких языков – Lisp. Давайте это проверим:

<?php

$langs =

array("Lisp","Python","Java","PHP",

"Perl","Lisp");

$lisp\_keys = array\_keys($langs,"Lisp");

echo "Lisp входит в массив ".

count($lisp\_keys) ." раза:<br>";

foreach ($lisp\_keys as $val){

echo "под номером $val <br>";

}

?>

Пример 7.5. Применение функции array\_keys()

В результате получим:

Lisp входит в массив 2 раза:

под номером 0

под номером 5

Функция *array\_keys()* , как и две предыдущие, зависит от регистра, т.е. элементов LISP в *массиве* она не обнаружит. *array\_keys()*появилась только в *PHP4*. В *PHP3* для реализации ее функциональности нужно придумывать свою функцию.

Если есть функция для получения всех *ключей* *массива*, то можно предположить, что существует и функция для получения всех *значений* *массива*. Действительно, она существует. Это функция *array\_values*(массив). Все *значения* переданного ей *массива*записываются в новый *массив*, проиндексированный целыми числами, т.е. все *ключи* *массива* теряются, остаются только *значения*. Но вернемся к нашему примеру.

Итак, мы выяснили, что язык Lisp случайно упомянут в нашем *массиве* дважды. Поскольку изучить один язык дважды нельзя ("учил, но забыл" не считается), то нужно как-то избавиться от повторяющихся языков. Сделать это довольно просто с помощью функции *array\_unique()* .

##### **Функция array\_unique**

Функция *array\_unique(массив)* возвращает новый *массив*, в котором повторяющиеся элементы фигурируют в одном экземпляре. Таким образом, вместо нескольких одинаковых *значений* и их *ключей* мы имеем одно *значение*. Какой у него будет *ключ*? Как из нескольких *ключей* одинаковых элементов выбирается тот, который будет сохранен в новом *массиве*? Происходит следующее. Все элементы *массива* преобразуются в строки и *сортируются*. Затем обработчик запоминает первый *ключ* для каждого *значения*, а остальные *ключи* игнорирует.

Попробуем избавиться от повторяющихся языков в списке изученных.

<?php

$langs =

array("Lisp","Java","Python","Java",

"PHP","Perl","Lisp");

print\_r(array\_unique($langs));

?>

Получим следующее:

Array ( [0] => Lisp [1] => Java [2] => Python [3]

=> PHP [4] => Perl )

Далее рассмотрим задачу *сортировки массива*.

### **Сортировка массивов**

Необходимость *сортировки* данных, в том числе и данных, хранящихся в виде *массивов*, очень часто возникает при решении самых разнообразных задач. Если в языке *Си* для того, чтобы решить эту задачу, нужно написать десятки строк кода, то в *PHP* это делается одной простой командой.

##### **Функция sort**

Функция *sort* имеет следующий синтаксис

sort (массив [, флаги])

и *сортирует массив*, т.е. упорядочивает его *значения* по возрастанию. Эта функция удаляет все существовавшие в *массиве* *ключи*, заменяя их числовыми индексами, соответствующими новому порядку элементов. В случае успешного завершения работы она возвращает true, иначе – false.

**Пример 7.6**.Пусть у нас есть два *массива*: *цены товаров* – их названия и, наоборот, названия товаров – их цены. Упорядочим эти *массивы* по возрастанию:

<?

$items = array(10 => "хлеб", 20 => "молоко",

30 => "бутерброд");

sort($items);

// строки сортируются в алфавитном

// порядке, ключи теряются

print\_r($items);

$rev\_items = array("хлеб" => 10,

"бутерброд" => 30, "молоко" => 20);

sort($rev\_items);

// числа сортируются по возрастанию,

// ключи теряются

print\_r($rev\_items);

?>

Пример 7.6. Применение функции sort()

Получим:

Array ( [0] => бутерброд [1] =>

молоко [2] => хлеб )

Array ( [0] => 10 [1] => 20 [2] => 30 )

В качестве дополнительного аргумента может использоваться одна из следующих констант:

* *SORT\_REGULAR* – автоматический выбор метода;
* *SORT\_NUMERIC* – сравнивать элементы *массива* как числа;
* *SORT\_STRING* – сравнивать элементы *массива* как строки.

##### 

##### **Функции asort, rsort, arsort**

Если требуется сохранять индексы элементов *массива* после *сортировки*, то нужно использовать функцию *asort (массив [, флаги])*. Если необходимо *отсортировать массив* в обратном порядке, т.е. от наибольшего *значения* к наименьшему, то можно задействовать функцию *rsort (массив [, флаги])* . А если при этом нужно еще и сохранить *значения* *ключей*, то следует использовать функцию *arsort(массив [, флаги])* . Как вы, наверное, заметили синтаксис у этих функций абсолютно такой же, как у функции *sort* . Соответственно и *значения* флагов могут быть такими же, как у *sort* : *SORT\_REGULAR* , *SORT\_NUMERIC* , *SORT\_STRING* . Кстати говоря, флаг *SORT\_NUMERIC* появился только в *PHP4*.

<?php

$books = array("Пушкин"=>"Руслан и Людмила",

"Толстой"=>"Война и мир",

"Лермонтов"=>"Герой нашего времени");

asort($books);

// сортируем массив,

// сохраняя значения ключей

print\_r($books);

echo "<br>";

rsort($books);

// сортируем массив в обратном порядке,

// ключи будут заменены

print\_r($books);

?>

Пример 7.7. Применение функций asort, rsort, arsort

В результате работы этого скрипта получим:

Array ( [Толстой] => Война и мир

[Лермонтов] => Герой нашего времени

[Пушкин] => Руслан и Людмила )

Array ( [0] => Руслан и Людмила

[1] => Герой нашего времени

[2] => Война и мир )

**Пример 7.8**. Допустим, мы создаем каталог описаний документов. У каждого документа есть автор, название, дата публикации и краткое содержание. Мы уже не раз отображали описания, составленные из этих характеристик. Каждый раз порядок отображения этих элементов зависел от созданной нами программы. Теперь же мы хотим иметь возможность изменять порядок отображения элементов по желанию пользователя. Составим для этого следующую форму:

<form action=task.php>

<table border=1>

<tr><td>Название </td><td><input type=text

name=title size=5> </td></tr>

<tr><td>Краткое содержание </td><td><input

type=text name=description size=5>

</td></tr>

<tr><td>Автор </td><td><input type=text

name=author size=5> </td></tr>

<tr><td>Дата публикации </td><td><input

type=text name=published size=5></td></tr>

</table>

<input type=submit value="Отправить">

</form>

Пример 7.8a. Форма для примера 7.8

Будем упорядочивать данные, переданные этой формой, по убыванию их *значений*, сохраняя при этом *значения* *ключей*. Для этого удобно воспользоваться функцией *arsort()* . Поскольку нам важен только новый порядок элементов, сохраним в новом *массиве**ключи* исходного *массива* в нужном порядке. Мы сохраняем *ключи* исходного *массива*, поскольку они являются именами элементов, из которых конструируется описание документа, а помнить их важно. Итак, получаем такой скрипт:

<?php

print\_r($\_GET); echo "<br>";

arsort ($\_GET);

// сортируем массив в обратном порядке,

// сохраняя ключи

print\_r($\_GET); echo "<br>";

$ordered\_names = array\_keys($\_GET);

// составляем новый массив

foreach($ordered\_names as $key => $val)

echo "$key :$val <br>";

// выводим элементы нового массива

?>

Пример 7.8b. Программа обработки формы из примера 7.8

#### **Сортировка массива по ключам**

Очевидно, что может возникнуть необходимость в *сортировке массива по значениям ключей*. Например, если у нас есть *массив* данных о книгах, как в приведенном выше примере, то вполне вероятно, что мы захотим *отсортировать* книги по именам авторов. Для этого в PHP также не нужно писать много строк кода – можно просто воспользоваться функцией *ksort()* для *сортировки* по возрастанию (прямой порядок *сортировки* ) или *krsort()* – для *сортировки* по убыванию (обратный порядок *сортировки* ). Синтаксис этих функций опять же аналогичен синтаксису функции *sort()* .

<?php

$books = array("Пушкин"=>"Руслан и Людмила",

"Толстой"=>"Война и мир",

"Лермонтов"=>"Герой нашего времени");

ksort($books);

// сортируем массив,

// сохраняя значения ключей

print\_r($books);

?>

Пример 7.9. Сортировка массива по ключам

Получим:

Array ( [Лермонтов] => Герой нашего времени

[Пушкин] => Руслан и Людмила

[Толстой] => Война и мир )

#### **Сортировка с помощью функции, заданной пользователем**

Кроме двух простых способов *сортировки значений массива* (по убыванию или по возрастанию) PHP предлагает пользователю возможность самому задавать критерии для *сортировки* данных. Критерий задается с помощью функции, имя которой указывается в качестве аргумента для специальных функций *сортировки* *usort()* или *uksort()* . По названиям этих функций можно догадаться, что *usort()**сортирует* *значения* элементов *массива*, а *uksort()* – *значения* *ключей**массива* с помощью определенной пользователем функции. Обе функции возвращают true, если *сортировка* прошла успешно, и false – в противном случае. Их синтаксис выглядит следующим образом:

usort (массив , сортирующая функция)

uksort (массив , сортирующая функция)

Конечно же, нельзя *сортировать массив* с помощью любой пользовательской функции. Эта функция должна удовлетворять определенным критериям, позволяющим сравнивать элементы *массива*. Как должна быть устроена сортирующая функция? Во-первых, она должна иметь два аргумента. В них интерпретатор будет передавать пары *значений* элементов для функции *usort()* или *ключей**массива* для функции *uksort()* . Во-вторых, сортирующая функция должна возвращать:

* целое число, меньшее нуля, если первый аргумент меньше второго;
* число, равное нулю, если два аргумента равны;
* число большее нуля, если первый аргумент больше второго.

Как и для других функций *сортировки*, для функции *usort()* существует аналог, не изменяющий *значения* *ключей*, – функция *uasort()* .

**Пример 7.10**. Допустим, у нас есть *массив*, содержащий такие сведения о литературных произведениях, как название, автор и год создания. Мы хотим упорядочить книги по дате создания.

<?php

// массив выглядит таким образом:

$books = array("Герой нашего времени" =>

array ("Лермонтов", 1840),

"Руслан и Людмила" => array("Пушкин",1820),

"Война и мир" => array ("Толстой",1863),

"Идиот" => array("Достоевский",1868));

/\* можно, конечно переписать этот массив

по-другому, сделав год издания, например,

индексом, но гораздо удобнее написать свою

функцию для сортировки \*/

uasort($books,"cmp");

// сортируем массив с помощью функции cmp

foreach ($books as $key => $book) {

echo "$book[0]: \"$key\"<br>";

}

function cmp($a,$b){

// функция, определяющая способ сортировки

if ($a[1] < $b[1]) return -1;

elseif ($a[1]==$b[1]) return 0;

else return 1;

}

?>

Пример 7.10. Сортировка с помощью пользовательских функций

В результате получим:

Пушкин: "Руслан и Людмила"

Лермонтов: "Герой нашего времени"

Толстой: "Война и мир"

Достоевский: "Идиот"

Мы применили нашу собственную функцию *сортировки* ко всем элементам *массива*. Далее рассмотрим, как *применить к элементам массива любую другую пользовательскую функцию*.

### **Применение функции ко всем элементам массива**

Функция ***array\_walk(массив, функция [, данные])*** применяет созданную пользователем функцию ко всем элементам *массива*массив и возвращает true в случае успешного выполнения операции и false – в противном случае.

Пользовательская *функция*, как правило, имеет два аргумента, в которые поочередно передаются *значение* и *ключ* каждого элемента *массива*. Но если при вызове функции *array\_walk()* указан третий *аргумент*, то он будет рассмотрен как *значение* третьего аргумента пользовательской функции, смысл которого определяет сам *пользователь*. Если *функция пользователя* требует больше аргументов, чем в нее передано, то при каждом вызове *array\_walk()* будет выдаваться предупреждение.

Если необходимо работать с реальными *значениями* *массива*, а не с их копиями, следует передавать *аргумент* в функцию по ссылке. Однако нужно иметь в виду, что нельзя добавлять или удалять элементы *массива* и производить действия, изменяющие сам *массив*, поскольку в этом случае результат работы *array\_walk()* считается неопределенным.

<?php

$books1 = array(

"А.С. Пушкин"=>"Руслан и Людмила",

"Л.Н. Толстой"=>"Война и мир",

"М.Ю. Лермонтов"=>"Герой нашего времени");

// создаем функцию, которую хотим

// применить к элементам массива

function try\_walk($val,$key,$data){

echo "$data \"$val\" написал $key<br>";

}

// применяем ко всем элементам массива

// $books1 функцию try\_walk

array\_walk($books1,"try\_walk","Роман");

?>

Пример 7.11. Применение функции ко всем элементам массива

В результате работы скрипта получим:

Роман "Руслан и Людмила" написал А.С. Пушкин

Роман "Война и мир" написал Л.Н. Толстой

Роман "Герой нашего времени"

написал М.Ю. Лермонтов

Заметим, что мы не изменили *значений* у элементов *массива*. Чтобы их изменить, надо было передавать *значения* в переменную $*val*функции try\_walk по ссылке.

<?php

$books1 = array(

"А.С. Пушкин"=>"Руслан и Людмила",

"Л.Н. Толстой"=>"Война и мир",

"М.Ю. Лермонтов"=>"Герой нашего времени");

// создаем функцию, которую хотим

// применить к элементам массива

function try\_walk(&$val,$key){

$key = "<p>Автор: " .$key ."<br>";

$val = "Название: \"" . $val ."\"</p>";

echo $key.$val;

}

// применяем ко всем элементам массива

// $book1 функцию try\_walk

array\_walk($books1,"try\_walk");

print\_r($books1);

?>

Пример 7.12. Применение функции ко всем элементам массива. Вариант 2

В результате работы скрипта получим:

Автор: А.С. Пушкин

Название: "Руслан и Людмила"

Автор: Л.Н. Толстой

Название: "Война и мир"

Автор: М.Ю. Лермонтов

Название: "Герой нашего времени"

Array ( [А.С. Пушкин] =>

Название: "Руслан и Людмила"

[Л.Н. Толстой] =>

Название: "Война и мир"

[М.Ю. Лермонтов] =>

Название: "Герой нашего времени")

### **Выделение подмассива**

#### **Функция array\_slice**

Поскольку *массив* – это набор элементов, вполне вероятно, потребуется выделить из него какой-нибудь поднабор. В PHP для этих целей есть функция *array\_slice* . Ее синтаксис таков:

array\_slice (массив,

номер\_элемента [, длина])

Эта функция *выделяет подмассив* длины длина в *массиве* массив, начиная с элемента, номер которого задан параметром номер\_элемента. Положительный номер\_элемента указывает на порядковый номер элемента относительно начала *массива*, отрицательный – на номер элемента с конца *массива*.

<?php

$arr = array(1,2,3,4,5);

$sub\_arr = array\_slice($arr,2);

print\_r($sub\_arr);

/\*

выведет Array ( [0] => 3 [1] =>4 [2] => 5 ),

т.е. подмассив, состоящий из элементов

3, 4, 5 \*/

$sub\_arr = array\_slice($arr,-2);

print\_r($sub\_arr);

// выведет Array ( [0] => 4 [1] => 5 ),

// т.е. подмассив, из элементов 4, 5

?>

Пример 7.13. Использование функции array\_slice()

Если задать параметр длина при использовании *array\_slice* , то будет *выделен подмассив*, имеющий ровно столько элементов, сколько задано этим параметром. Длину можно указывать и отрицательную. Если в эту функцию передан отрицательный параметр length, в последовательность войдут все элементы исходного массива, начиная с позиции offset и заканчивая позицией, отстоящей на length элементов от конца array.

<?php

$arr = array(1,2,3,4,5);

$sub\_arr = array\_slice($arr, 2, 2);

// содержит массив из элементов 3, 4

$sub = array\_slice($arr,-3, 2);

// тоже содержит массив из элементов 3, 4

$sub1 = array\_slice($arr,0, -1);

// содержит массив из

// элементов 1, 2, 3, 4

$sub2 = array\_slice($arr,-4, -2);

// содержит массив из элементов 2, 3

?>

Пример 7.14. Использование функции array\_slice(). Вариант 2

#### **Функция array\_chunk**

Есть еще одна функция, похожая на *array\_slice()* – это *array\_chunk()* . Она разбивает *массив* на несколько подмассивов заданной длины. Синтаксис ее такой:

array\_chunk ( массив, размер

[, сохранять\_ключи])

В результате работы *array\_chunk()* возвращает многомерный *массив*, элементы которого представляют собой полученные подмассивы. Если задать параметр сохранять *ключи* как true, то при разбиении будут сохранены *ключи* исходного *массива*. В противном случае *ключи* элементов заменяются числовыми индексами, которые начинаются с нуля.

**Пример 7.15**.У нас есть список приглашенных, оформленный в виде *массива* их фамилий. У нас имеются столики на три персоны. Поэтому нужно распределить всех приглашенных по трое.

<?php

$persons = array("Иванов", "Петров",

"Сидорова","Зайцева", "Волкова");

$triples = array\_chunk($persons,3);

// делим массив на подмассивы

// по три элемента

foreach ($triples as $k => $table){

// выводим полученные тройки

echo "За столиком номер $k сидят: <ul>";

foreach ($table as $pers)

echo "<li>$pers";

echo "</ul>";

}

?>

Пример 7.15. Использование функции array\_chunk()

В результате получим:

за столиком номер 0 сидят:

• Иванов

• Петров

• Сидорова

за столиком номер 1 сидят:

• Зайцева

• Волкова

### **Сумма элементов массива**

В этом разделе мы познакомимся с функцией, вычисляющей *сумму всех элементов массива*. Сама задача вычисления *суммы значений массива* предельно проста. Но зачем писать лишний раз один и тот же код, если можно воспользоваться специально созданной и всегда доступной функцией. *Функция* эта называется, как можно догадаться, *array\_sum()* . И в качестве параметра ей передается только имя *массива*, *сумму значений элементов* которого нужно вычислить.

В качестве примера использования этой функции приведем решение более сложной задачи, чем просто *вычисление* *суммы элементов*. Этот пример также иллюстрирует применение функции *array\_slice()* , которую мы обсуждали чуть раньше.

**Пример 7.16**.Пусть дан *массив* натуральных чисел. Нужно найти в нем такое число, что *сумма элементов* справа от него равна *сумме элементов* слева от него.

<?php

//массив задается функцией array

$arr = array(2,1,3,4,5,6,4);

// перебираем каждый элемент массива $arr.

// Внутри цикла текущий ключ массива

// содержится в переменной $k,

// текущее значение – в переменной $val

foreach ($arr as $k => $val){

$p = $k + 1;

// синтаксис array array\_slice (

// array array,int offset [,int length])

// array\_slice выделяет подмассив

// длины length в массиве array,

// начиная с элемента offset.

$out\_next = array\_slice($arr,$p);

// получаем массив элементов,

// идущих после текущего

$out\_prev = array\_slice($arr,0,$k);

// получаем массив элементов,

// идущих перед текущим

// функция mixed array\_sum (array array)

// подсчитывает сумму элементов массива array

$next\_sum = array\_sum($out\_next);

$prev\_sum = array\_sum($out\_prev);

// если сумма элементов до текущего равна

// сумме элементов после, то выводим

// значение текущего элемента

if ($next\_sum==$prev\_sum)

echo "value:$val";

// можно посмотреть, что представляют собой

// рассмотренные массивы на каждом шаге

// print\_r($out\_next); echo "<br>";

// print\_r($out\_prev);

// echo "$next\_sum, $prev\_sum<br>";

echo "<hr>";

}

?>

Пример 7.16. Программа поиска числа, такого что сумма элементов справа от него равна сумме элементов слева от него

### **Заключение**

Итак, подведем итоги. В этой лекции мы изучили ряд функций, упрощающих работу с *массивами* данных. Мы рассмотрели функции для *поиска значения среди элементов массива* ; функции для *сортировки элементов массива*, как по их *значениям*, так и по *ключам* ; функции, позволяющие *применять ко всем элементам массива функцию, созданную пользователем*. Кроме того, мы изучили функцию, *выделяющую подмассивы* из элементов *массива*, и функцию, вычисляющую *сумму всех элементов массива*. Использование всех этих функций было продемонстрировано на примерах. Все функции для работы с *массивами* доступны без каких-либо конфигурационных настроек *PHP*, и пользоваться ими можно абсолютно свободно. Так что, прежде чем изобретать велосипед, загляните в руководство по языку *PHP*: – вполне возможно, что кто-то сделал это до вас.