Практическое занятие №22 Массивы и функции в PHP. Часть 1.

Задание №1

Проверьте запущен ли у Вас Open Server. Папка с материалами урока (php22.loc) должна быть размещена в директории /domains сервера. Заходим на http://php22.loc и начинаем урок.

Задание №2

Как вы могли убедиться на лекции, управляющие конструкции if…else, switch, а также циклы практически не отличаются от таких в JS. Рассмотрим только некоторые отличающиеся элементы.

1. Elseif. В языке PHP помимо конструкции if и if…else, также есть if…elseif и if…elseif…else.

```php
$a = 5;
if ($a == 1) {
    //код
} elseif ($a == 2) {
    //код
} elseif ($a == 3) {
    //код
} elseif ($a == 4) {
    //код
} elseif ($a == 4) {
    //код
} else {
    //код
}
```

Впрочем, в большинстве случаев это можно заменить конструкцией switch.

2. Foreach. Конструкция foreach предоставляет простой способ перебора массивов. Существует два вида синтаксиса.

Первый только со значением:

```php
$a = [1,2,3,4,5,6,7,8];

foreach ($a as $value) {

    //в каждой новой итерации в $value будет следующее значение
    var_dump($value);

}
```

Второй со значением и ключом.

```php
$a = [1, 2, 3, 4, 5, 6, 7, 8];

foreach ($a as $key => $value) {

    //в каждой новой итерации в $value будет следующее значение,
    //а в $key будет ключ текущего значения
    var_dump("$key: " . $value);
}
```

Интересным моментом также может быть то, что следующие 2 варианта идентичны

```php
function f() {
    var_dump('call');
    return [1, 2, 3, 4, 5, 6, 7, 8];
}

$array = f();
foreach ($array as $value) {
    var_dump($value);
}
```

```php
function f() {
    var_dump('call');
    return [1, 2, 3, 4, 5, 6, 7, 8];
}

foreach (f() as $value) {
    var_dump($value);
}
```

Как вы могли заметить и в том и в ином случае функция f была вызвана 1 раз. Это может быть полезно, поскольку позволяет не создавать без необходимости лишние переменные.

Полное описание конструкции foreach - http://php.net/manual/ru/control-structures.foreach.php.

Задание №3

Массив в PHP - это упорядоченное отображение, которое устанавливает соответствие между значением и ключом. Ключ может быть либо типа integer, либо типа string.

Массив (тип array) может быть создан языковой конструкцией array(). В качестве параметров она принимает любое количество разделенных запятыми пар ключ => значение.

```php
$array1 = array(1,2,3,4);

$array2 = array(
    'a' => 1,
    'b' => 2,
    'c' => 3
);
```

Начиная с PHP 5.4 возможно использовать короткий синтаксис определения массивов, который заменяет языковую конструкцию array() на [].

```php
$array1 = [1,2,3,4];

$array2 = [
    'a' => 1,
    'b' => 2,
    'c' => 3
];
```

Для обращения к элементу массива можно использовать квадратные или фигурные скобки.

```php
$array = [1,2,3,4];

var_dump($array[0]);
var_dump($array{0});
```

Для добавления элемента в конец массива можно использовать конструкцию:

```php
$array = [1,2,3,4];

$array[] = 5;
var_dump($array);
```

Также для манипулирования элементами массива можно использовать функции array_pop, array_push, array_shift и array_unshift, которые работают аналогично методам Array.prototype.pop, Array.prototype.push, Array.prototype.shift и Array.prototype. unshift в JS. Вспомните как работают эти методы и поэкспериментируйте с функциями PHP.

Задание №4

В PHP есть множество функций для работы с массивами. Полный список находится тут - http://php.net/manual/ru/ref.array.php. Рассмотрим самые часто используемые из них:

1. Count - подсчитывает количество элементов массива.

```php
$array = [1,2,3,4,5];
var_dump(count($array));
```

Если в качестве второго параметра передать константу COUNT_RECURSIVE, функция будет рекурсивно подсчитывать количество элементов массива.

2. is_array - определяет, является ли данная переменная массивом.

```php
$array = array(1,2,2,3);
var_dump(is_array($array));
```

3. in_array - проверяет, присутствует ли в массиве значение.

```php
$array = [1,2,3,4,5];
var_dump(in_array('1', $array));
var_dump(in_array('1', $array, true));
var_dump(in_array(7, $array));
```

Если третьим параметром передать true, для сравнения значений будет использовано строгое сравнение ===.

4. array_values – выбирает все значения массива. Она также заново индексирует возвращаемый массив числовыми индексами.

```php
$array = array("size" => "XL", "color" => "gold");
print_r(array_values($array));
```

5. array_keys – возвращает все или некоторое подмножество ключей массива. Если указан необязательный второй параметр, функция возвращает только ключи, совпадающие с этим параметром. В обратном случае, функция возвращает все ключи массива array. Если третьим параметром передать true, для сравнения значений будет использовано строгое сравнение ===.

```php
$array = array(1,2,3,3,4,5);
var_dump(array_keys($array));
var_dump(array_keys($array, 3));
```

6. array_filter – Фильтрует элементы массива с помощью функции. Обходит каждое значение массива array, передавая его в функцию. Если функция возвращает true, данное значение из array возвращается в результирующий массив. Если функция не передана все значения массива array эквивалентные FALSE будут удалены. Если передать один из 2 возможных флагов в качестве третьего параметра, можно определить, что будет передаваться в функцию (по умолчанию передается только значение): ARRAY_FILTER_USE_KEY -

передавать только ключ массива как аргумент для callback вместо значения. ARRAY_FILTER_USE_BOTH - передавать и ключ, и значение в callback вместо только значения.

```php
$array=[0,1,1,12,3,4,5];

$filterResult = array_filter($array, function($value, $key){
    var_dump($key, $value);
    return $value ==1;
}, ARRAY_FILTER_USE_BOTH);

var_dump($filterResult);
```

7. array_merge — сливает один или большее количество массивов. Сливает элементы одного или большего количества массивов таким образом, что значения одного массива присоединяются к концу предыдущего. Результатом работы функции является новый массив. Если входные массивы имеют одинаковые строковые ключи, тогда каждое последующее значение будет заменять предыдущее. Однако, если массивы имеют одинаковые числовые ключи, значение, упомянутое последним, не заменит исходное значение, а будет добавлено в конец массива. В результирующем массиве значения исходного массива с числовыми ключами будут перенумерованы в возрастающем порядке, начиная с нуля.

```php
$array1 = array("color" => "red", 2, 4);
$array2 = array("a", "b", "color" => "green", "shape" => "trapezoid", 4);
var_dump($array1, $array2);

$result = array_merge($array1, $array2);
var_dump($result);
```

Задание №5

Для сортировки массива в PHP есть множество функций (полный список: http://php.net/manual/ru/array.sorting.php), однако чаще всего используются две из них:

1. sort. Эта функция сортирует массив. После завершения работы функции элементы массива будут расположены в порядке возрастания. Дополнительный второй параметр можно использовать для изменения поведения сортировки.

```php
$array1 = [65,43,64,487,54,23];
$array2 = ['fcd', 'jh', 'dsc', 'e'];

sort($array1);
sort($array2);

var_dump($array1, $array2);
```

2. usort. Сортирует массив по значениям используя пользовательскую функцию для сравнения элементов. Функция сравнения должна возвращать целое, которое меньше (если первый аргумент является меньшим), равно (если аргументы равны) или больше нуля (если первый аргумент является большим).

```php
$array = [65,43,64,487,54,23];


usort($array, function($a, $b){
    return (int) $b-$a;
});

var_dump($array);
```

Задание №6

В общем случае пользовательские функции выглядят следующим образом:

```
function f(/* args */){
    //body
}
```

Имена функций следуют тем же правилам, что и другие метки в PHP. Однако в отличии от переменных, функции в PHP не регистрозависимы.

```
function f(){
    var_dump(1);
}

f();
F();
```

Все функции и классы PHP имеют глобальную область видимости - они могут быть вызваны вне функции, даже если были определены внутри. Однако если функция определена в условном блоке или внутри другой функции, она недоступна до выполнения того блока в котором она определена.

```
function f(){

    function fInner(){
        echo 'hello. I\'m fInner()';
    }

}

//fInner();// недоступна

f();
fInner();
```

PHP не поддерживает перегрузку функции, также отсутствует возможность переопределить или удалить объявленную ранее функцию. Поэтому если вы по какой-то причине не уверены есть ли такая функция перед объявлением новой функции или перед ее вызовом можно проверить существование функции функцией function_exist. Модифицируем немного предыдущий пример.

```
function f() {

    if (!function_exists('fInner')) {

        function fInner() {
            echo 'hello. I\'m fInner()';
        }

    }
}

if (function_exists('fInner')) {
    fInner();
}

f();
f(); //теперь функцию f() можно вызвать любое кол-во раз

if (function_exists('fInner')) {
    fInner();
}
```

Это не лучшая практика, но в некоторых случаях необходимо.

Задание №7

Функция может принимать информацию в виде списка аргументов, который является списком разделенных запятыми выражений. Аргументы вычисляются слева направо.

```php
function f($a) {

   echo $a;

}

f('test');
```

По умолчанию аргументы в функцию передаются по значению (это означает, что если вы измените значение аргумента внутри функции, то вне ее значение все равно останется прежним). Если вы хотите разрешить функции модифицировать свои аргументы, вы должны передавать их по ссылке.

```php
function f(&$a) {

    $a += 5;
}

$var = 5;

f($var);
var_dump($var);
```

В старых версиях PHP переменную по ссылке можно было передать через вызов функции, даже если в объявлении функции это не было указано, однако в текущих версиях PHP это было отменено и вызовет ошибку.

```php
function f($a) {

    $a += 5;
}

$var = 5;

//сработает в старых версиях, но в актуальных вызовет ошибку
f(&$var);
var_dump($var);
```

Функция может определять значения по умолчанию. Обратите внимание, что все аргументы, для которых установлены значения по умолчанию, должны находиться правее аргументов, для которых значения по умолчанию не заданы, в противном случае Вам придется всегда указывать значения параметров даже если у них есть значение по умолчанию. PHP также позволяет использовать массивы, специальный тип NULL и константы качестве значений по умолчанию.

```php
define('MY_CONSTANT', 1);

function f($a, $b = MY_CONSTANT, $c = array(1, 2, 3), $d = NULL) {
    var_dump($a, $b, $c, $d);
}

f(1); //передаем только $a
f(1,2); //передаем $a и $b
//...
```

Объявления типов позволяют функциям строго задавать тип передаваемых параметров. Передача в функцию значений несоответствующего типа будет приводить к ошибке: в PHP 5 это будет обрабатываемая фатальная ошибка, а в PHP 7 будет выбрасываться исключение TypeError. Чтобы объявить тип аргумента, необходимо перед его именем добавить имя требуемого типа. Также можно объявить тип NULL, чтобы указать, что значением по умолчанию аргумента является NULL. Указывать примитивные типы (bool, int, string, float) можно только с 7 версии PHP.

```php
class C{}


function f(
        array $a, //если передать не массив вызовет ошибку, рабоает с 5 версии
        C $b, // переданных аргумент должен быть объектом класса C (об этих
        //сущностях в дальнейших занятиях).
        int $c, //работает только 7 версии, если тип не соответсвует PHP,
        //попытается преобразовать значение
        float $d //работает только 7 версии, если тип не соответсвует PHP,
        //попытается преобразовать значение
        ) {
            var_dump($a,$b,$c,$d);
}

f(array(1,2,3), new C(), '5.5', 5);
//f(array(1,2,3), new C(), 'error', 5); вызовет ошибку
```

Для работы с функциями с произвольным кол-вом параметров лучше всего воспользоваться функциями func_num_args, func_get_args, func_get_arg.

```php
function f(){

    $num = func_num_args(); //кол-во аргументов

    $argsAsArray = func_get_args(); //все аргументы в массиве

    $arg0 = func_get_arg(0); //первый аргумент

    var_dump($num, $argsAsArray, $arg0);


}

f(1,2,3,3,5);
```

Также в PHP 7 была введена возможность указывать возвращаемый функцией тип

```php
function f() : float {

}
```

Задание №8

Мы уже рассмотрели некоторые функции для работы с функциями: func_num_args, func_get_args, func_get_arg и function_exist. Весь список таких функций можно посмотреть тут http://php.net/manual/ru/ref.funchand.php.

Отдельно стоит упомянуть функции call_user_func и call_user_func_array. Они похожи на методы call и apply Function.prototype JS, то есть позволяют неявно вызвать функцию.

```php
function f($a, $b){
    var_dump($a + $b);
}

call_user_func('f', 5, 7);
call_user_func_array('f', [7,8]);
```

Задание №9

Найти сумму натуральных чисел от a до b, где a и b, произвольные переменные, заданные в начале скрипта. В случае некорректных a и b (например, a>b) вывести сообщение 'Сумма не существует'.

Задание №10

Напишите функцию, которая выводит все целые числа, меньшие n, у которых есть хотя бы одна цифра 3 и которые не делятся нацело на 5. n – аргумент функции.

45 -> 3, 13, 23, 31, 32, 33, 34, 36, 37, 38, 39, 43

43 -> 3, 13, 23, 31, 32, 33, 34, 36, 37, 38, 39

Задание №11

Вывести все нечисловые ключи массива.

[1, 3, 4, ‘a’ => 5, ‘b’ => 6] -> a, b

Задание №12

Напишите функцию, которая возвращает уникальные значения массива (без использования встроенной функции array_unique).

Задание №13

Написать функцию, которая принимает произвольный массив и возвращает его максимальный уровень вложенности.

Задание №14

Дан одномерный массив целых чисел. Найти число, которое повторяется чаще всего.

Задание №15 *

Дан массив размера n. После каждого отрицательного элемента массива вставить элемент с нулевым значением.