**Лабораторная работа №20**

**Пользовательские функции в PHP (часть 2)**

**Всего заданий: 19**

**Флаги в функциях**

Пусть у нас есть код, проверяющий число на простоту:

<?php

$num = 31;

$flag = true;

for ($i = 2; $i < $num; $i++) {

if ($num % $i === 0) {

$flag = false;

break;

}

}

var\_dump($flag);

?>

Давайте оформим этот код в виде функции:

<?php

function isPrime($num) {

$flag = true;

for ($i = 2; $i < $num; $i++) {

if ($num % $i === 0) {

$flag = false;

break;

}

}

return $flag;

}

?>

Проверим работу нашей функции:

<?php

var\_dump( isPrime(10) );

var\_dump( isPrime(31) );

?>

**Упростим код**

Давайте теперь посмотрим, как можно упростить работу с флагами, если мы используем их в функции. На самом деле в случае с функцией переменная $flag и команда break нам не понадобятся - вместо них все сделает команда return.

Делается это следующим образом: внутри функции запускаем наш цикл с ифом и, если мы попали в иф - возвращаем false, тем самым выходя из цикла и из функции. А если цикл прокрутился, но выхода из функции не случилось - после цикла возвращаем true:

<?php

function isPrime($num) {

for ($i = 2; $i < $num; $i++) {

if ($num % $i === 0) {

return false;

}

}

return true;

}

?>

Еще раз как это работает: если в массиве найдется нужный элемент - мы выйдем из функции (и из цикла тоже) с помощью return. Но если в массиве не найдется нужный элемент - выхода из функции не произойдет и выполнение дойдет до команды return true. И получится, что функция вернет true в знак того, что число простое.

**Проверка по массиву**

Давайте теперь сделаем функцию, которая параметром будет принимать массив, и проверять, что все элементы в этом массиве положительные.

Пусть функция возвращает true, если это так, и false, если хотя бы один из элементов массива отрицательный. Вот примеры того, как должна работать наша функция:

<?php

var\_dump( isPositive([1, -2, 3, 4, -5]) ); // выведет false

var\_dump( isPositive([1, 2, 3, 4, 5]) ); // выведет true

?>

Реализуем описанную функцию с помощью неявного использования флагов:

<?php

function isPositive($arr) {

foreach ($arr as $elem) {

if ($elem < 0) {

return false;

}

}

return true;

}

?>

**Задача №1**

Сделайте функцию, которая параметром будет принимать массив с числами, и проверять, что все элементы в этом массиве являются четными числами.

**Задача №2**

Сделайте функцию, которая параметром будет принимать число и проверять, что все цифры это числа являются нечетными.

**Задача №3**

Сделайте функцию, которая параметром будет принимать массив и проверять, есть ли в этом массиве два одинаковых элемента подряд.

**Правильное использование функций**

Пусть у нас дан вот такой массив с четырехзначными числами:

<?php

$arr = ['1524', '1321', '4563', '7144', '2879'];

?>

Давайте запишем в новый массив только те элементы, сумма первых двух цифр в которых равна сумме вторых двух цифр.

Очевидно, что для решения задачи необходимо перебирать циклом наш массив и проверять каждый элемент. Давайте вынесем проверку в отдельную функцию, назвав ее checkDigitsPairsSum (*проверить суммы пар цифр*). Пусть эта функция параметром принимает четырехзначное число и проверяет суммы пар цифр. Если суммы равны, пусть функция возвращает true, а если не равны - false.

Не будем пока реализовывать описанную функцию, а будем писать дальнейший код так, как будто у нас уже есть эта функция, а затем ее реализуем. Итак, если бы у нас была описанная функция, мы бы легко решили поставленную задачу вот так:

<?php

$result = [];

$arr = ['1524', '1321', '4563', '7144', '2879'];

foreach ($arr as $elem) {

if (checkDigitsPairsSum($elem)) { // используем нашу функцию

$result[] = $elem;

}

}

var\_dump($result);

?>

Напишем теперь реализацию нашей функции:

<?php

function checkDigitsPairsSum($num) {

$sum1 = $num[0] + $num[1];

$sum2 = $num[2] + $num[3];

if ($sum1 === $sum2) {

return true;

} else {

return false;

}

}

?>

Упростим код функции, используя вместо конструкции if сокращенную логическую операцию:

<?php

function checkDigitsPairsSum($num) {

$sum1 = $num[0] + $num[1];

$sum2 = $num[2] + $num[3];

return $sum1 === $sum2;

}

?>

Упростим еще больше:

<?php

function checkDigitsPairsSum($num) {

return $num[0] + $num[1] === $num[2] + $num[3];

}

?>

Окончательный код будет выглядеть так:

<?php

$result = [];

$arr = ['1524', '1321', '4563', '7144', '2879'];

foreach ($arr as $elem) {

if (checkDigitsPairsSum($elem)) {

$result[] = $elem;

}

}

var\_dump($result);

function checkDigitsPairsSum($num) {

return $num[0] + $num[1] === $num[2] + $num[3];

}

?>

**Задача №4**

Дана функция, которая параметром принимает целое число и возвращает сумму его цифр:

<?php

function getDigitsSum($num) {

return array\_sum(str\_split($num, ''));

}

?>

С помощью приведенной функции найдите все года от 1 до 2030, сумма цифр которых равна 13.

**Преимущества**

Вынесение проверки в отдельную функцию сделало наш код более понятным и очевидным. Для того, чтобы понять, что имеется ввиду, посмотрите на следующий цикл:

<?php

foreach ($arr as $elem) {

if (checkDigitsPairsSum($elem)) {

$result[] = $elem;

}

}

?>

При чтении кода сразу становится более-менее понятно, что именно проверяется в условии - находится и проверяется какая-то сумма пар чисел.

Очевидность нашего кода достигается удачно подобранным названием функции. Представьте себе, что мы бы назвали нашу функцию просто func:

<?php

foreach ($arr as $elem) {

if (func($elem)) {

$result[] = $elem;

}

}

?>

Как вы видите, код теперь далеко не такой очевидный! Понадобится детально изучить код функции func, чтобы понять, что же она такое делает.

**Комбинация вспомогательных функций**

Пусть у нас дана функция getDigits, возвращающая цифры переданного числа:

<?php

function getDigits($num) {

return str\_split($num, 1);

}

?>

Пусть также дана функция getAvg, возвращающая среднее элементов переданного массива:

<?php

function getAvg($arr) {

if (!empty($arr)) {

return array\_sum($arr) / count($arr);

} else {

return 0;

}

}

?>

Давайте найдем сумму цифр какого-нибудь числа, используя комбинацию приведенных выше функций:

<?php

echo getAvg(getDigits(12345));

?>

**Задача №5**

Напишите код, который будет находить среднее от делителей заданного числа.

**Вспомогательные функции внутри других функций**

В данном уроке я покажу вам, как решать более сложные задачи, используя комбинации вспомогательных функций. Рассмотрим на примере задачи.

Пусть у нас дано число. Давайте получим все собственные делители этого числа, являющиеся простыми числами.

Пусть наше решение задачи будет представлять собой реализацию вот этой функции:

<?php

function getPrimeDivisors($num) {

}

?>

Писать весь код решения в нашей функции - это не очень удачная идея. Так как наша задача достаточно сложная, то ее можно разбить на подзадачи, сделав для решения каждой подзадачи отдельную вспомогательную функцию.

Немного подумав, становится понятно, что было бы неплохо иметь функцию getOwnDivisors для нахождения делителей числа и функцию isPrime для проверки числа на простоту. Если бы у нас были такие функции, то мы бы решили нашу задачу следующим образом:

<?php

function getPrimeDivisors($num) {

$result = [];

$divs = getOwnDivisors($num);

foreach ($divs as $div) {

if (isPrime($div)) {

$result[] = $div;

}

}

return $result;

}

?>

**Задача №6**

Напишете реализацию вспомогательных функций и решите поставленную задачу.

**Работа с рекурсией в PHP**

В программировании есть такое понятие, как *рекурсия* - это когда функция вызывает сама себя. Давайте посмотрим на примере. Выведем с помощью рекурсии числа от 1 до 10:

<?php

$i = 1;

function func(){

echo $i;

$i++;

if ($i <= 10){

func(); // здесь функция вызывает сама себя

}

}

func();

?>

Давайте обсудим, как работает этот код.

У нас есть глобальная переменная $i и функция func, внутри которой в консоль выводится содержимое переменной $i, а затем делается ++.

Если наша переменная $i меньше или равна 10, то функция вызывается повторно. Так как переменная $i - глобальная, то при каждом новом вызове функции в ней будет заданное при предыдущем вызове значение переменной $i.

Получится, что функция будет вызывать сама себя до тех пор, пока $i не станет больше 10.

Учтите, что в нашем случае нельзя функцию запустить без if - если это сделать, то получится бесконечный вызов функций.

**Пример с параметром**

Давайте, с помощью рекурсии последовательно выведем элементы массива. Пусть массив изначально передается параметрам функции:

<?php

func([1, 2, 3]);

?>

Давайте пока без рекурсии используя функцию [array\_shift](http://code.mu/ru/php/manual/array/array_shift/) выведем все элементы массива по очереди:

<?php

function func($arr) {

var\_dump(array\_shift($arr)); // выведет 1

var\_dump($arr); // выведет [2, 3] - массив уменьшился

var\_dump(array\_shift($arr)); // выведет 2

var\_dump($arr); // выведет [3] - массив уменьшился

var\_dump(array\_shift($arr)); // выведет 3

var\_dump($arr); // выведет [] - массив пуст

}

func([1, 2, 3]);

?>

Как вы видите, функция array\_shift вырезает и возвращает первый элемент массива, при этом сам массив уменьшается на этот элемент.

Давайте теперь используем рекурсию:

<?php

function func($arr) {

var\_dump(array\_shift($arr));

var\_dump($arr);

if (count($arr) !== 0) {

func($arr);

}

}

func([1, 2, 3]);

?>

На самом деле, конечно же, проще всего перебрать элементы массива циклом. Приведенные примеры пока просто демонстрируют работу рекурсии на простых примерах (не жизненных). Более полезные примеры применения рекурсии просто более сложные, мы их разберем чуть ниже.

**Задача №7**

Дан массив:

<?php

$arr = ['a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5];

?>

С помощью рекурсии выведите элементы этого массива на экран.

**Сумма элементов массива**

Давайте теперь не будем выводить элементы массива на экран, а найдем сумму элементов этого массива:

<?php

function getSum($arr) {

$sum = array\_shift($arr);

if (count($arr) !== 0) {

$sum += getSum($arr);

}

return $sum;

}

var\_dump(getSum([1, 2, 3]));

?>

**Задача №8**

Дан массив:

<?php

$arr = ['a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5];

?>

С помощью рекурсии найдите сумму элементов этого массива.

**Рекурсия и многомерные структуры в PHP**

Дан массив многомерный произвольного уровня вложенности, например, такой:

<?php

$arr = [

1,

[

2, 7, 8

],

[

3, 4, [5, [6, 7]],

]

];

?>

Как вы видите, данный массив имеет сложную структуру, причем предполагается, что эта структура может быть произвольной и уровни вложенности могут быть сколь угодно глубоко.

Пусть мы хотим вывести на экран все примитивные (то есть не массивы) элементы нашего массива. В этом случае для перебора такого массива у нас просто не получится использовать циклы, так как массив имеет неправильную структуру и неизвестный уровень вложенности.

Зато для перебора такого массива очень удобно будет использовать рекурсию.

Для начала сделаем функцию, в которую параметром будем передавать наш массив, а в функции сделаем цикл для перебора нашего массива:

<?php

function func($arr) {

foreach ($arr as $elem) {

echo $elem;

}

}

func([1, [2, 7, 8], [3, 4, [5, [6, 7]]]]);

?>

Сделанный нами цикл будет перебирать только элементы основного массива. То есть вначале он выведет 1, потом [2, 7, 8], а потом [3, 4, [5, [6, 7]].

Давайте теперь будем разделять в цикле элементы-примитивы и элементы-массивы:

<?php

function func($arr) {

foreach ($arr as $elem) {

if (is\_array($elem)) {

// элемент - массив

} else {

// элемент - примитив

echo $elem;

}

}

}

func([1, [2, 7, 8], [3, 4, [5, [6, 7]]]]);

?>

А теперь сделаем так, чтобы если наш элемент - массив, функция вызывала сама себя, передавая параметром этот массив:

<?php

function func($arr) {

foreach ($arr as $elem) {

if (is\_array($elem)) {

func($elem);

} else {

echo $elem;

}

}

}

func([1, [2, 7, 8], [3, 4, [5, [6, 7]]]]);

?>

**Задача №9**

Дан многомерный массив произвольного уровня вложенности, например, такой:

<?php

$arr = [1, 2, 3, [4, 5, [6, 7]], [8, [9, 10]]];

?>

С помощью рекурсии выведите все примитивные элементы этого массива на экран.

Сумма элементов массива

Давайте найдем сумму примитивных элементов нашего массива:

<?php

function func($arr) {

$sum = 0;

foreach ($arr as $elem) {

if (is\_array($elem)) {

$sum += func($elem);

} else {

$sum += $elem;

}

}

return $sum;

}

var\_dump(func([1, [2, 7, 8], [3, 4, [5, [6, 7]]]]));

?>

**Задача №10**

Дан многомерный массив произвольного уровня вложенности, например, такой:

<?php

$arr = [1, 2, 3, [4, 5, [6, 7]], [8, [9, 10]]];

?>

С помощью рекурсии найдите сумму элементов этого массива.

**Задача №11**

Дан многомерный массив произвольного уровня вложенности, содержащий внутри себя строки, например, такой:

<?php

$arr = ['a', ['b', 'c', 'd'], ['e', 'f', ['g', ['j', 'k']]]];

?>

С помощью рекурсии слейте элементы этого массива в одну строку:

'abcdefgjk'

**Манипуляции с элементами**

Давайте что-нибудь сделаем с перебираемыми элементами массива, к примеру, запишем им в конец знак '!':

<?php

function func($arr) {

$length = count($arr);

for ($i = 0; $i < $length; $i++) {

if (is\_array($arr[$i])) {

$arr[$i] = func($arr[$i]);

} else {

$arr[$i] = $arr[$i] . '!';

}

}

return $arr;

}

var\_dump(func([1, [2, 7, 8], [3, 4, [5, 6]]]));

?>

**Задача №12**

Дан многомерный массив произвольного уровня вложенности, например, такой:

<?php

$arr = [1, [2, 7, 8], [3, 4], [5, [6, 7]]];

?>

Возведите все элементы-числа этого массива в квадрат.

**Практические задания по функциям**

**Задача №13**

Сделайте функцию, которая будет проверять пару чисел на дружественность. Дружественные числа - два числа, для которых сумма всех собственных делителей первого числа равна второму числу и наоборот, сумма всех собственных делителей второго числа равна первому числу.

**Задача №14**

Используя созданную вами функцию из предыдущей задачи найдите все пары дружественных чисел в промежутке от 1 до 1000.

**Задача №15**

Сделайте функцию, которая будет проверять число на совершенность. Совершенное число - это число, сумма собственных делителей которого равна этому числу.

**Задача №16**

Найдите все счастливые билеты. Счастливый билет - это билет, в котором сумма первых трех цифр его номера равна сумме вторых трех цифр его номера.

**Задача №17**

Сделайте функцию, которая параметром будет принимать два числа и возвращать массив их общих делителей.

**Задача №18**

Сделайте функцию, которая будет принимать строку на русском языке, а возвращать ее транслит.

**Задача №19**

Сделайте функцию, которая будет принимать число, а возвращать это число прописью. Пусть функция работает с числами до 999. Смотрите пример:

<?php

func(123); // выведет 'сто двадцать три'

?>