Лабораторна робота №2

Масиви у PHP. Багатократне використання коду. Створення функцій.

Тривалість: 2 акад. години

Мета: набути практичних навичок з використання масивів та багатократного використання програмного коду.

Завдання: дослідити використання масивів, створення функцій користувача та можливості включення у програмний код раніше розроблених файлів.

Обладнання: EOM, програмне забезпечення з відкритим кодом (opensource): PHP7, Apache 2.

2.1 Теоретичні відомості

2.1.1. Використання масивів у PHP

У більшості мов програмування масиви мають числові індекси, котрі, як правило, починаються з нуля чи одиниці.

PHP дозволяє використовувати як індекси числа або рядки.

Масив створюється наступним чином:

$a = array('data1','data2','data3');

$b = array(1,2,3);

Відповідно, доступ до елементів масиву, отримують так:

echo '$a[0]='.$a[0].'<br />';

echo '$a[1]='.$a[1].'<br />';

echo '$a[2]='.$a[2].'<br />';

echo '$b[0]='.$b[0].'<br />';

echo '$b[1]='.$b[1].'<br />';

echo '$b[2]='.$b[2].'<br />';

Якщо у масиві необхідно зберегти зростаючу послідовність чисел, використовують функцію range(). Аналогічно функція утворює послідовність символів:

<?

$nombers = range(1,10);

$symbols=range('a','j');

for($i=0; $i<10; $i++)

echo $nombers[$i].'---'.$symbols[$i].'<br/>';

?>

Даний приклад генерує масив із десяти чисел у діапазоні від 1 до 10 та послідовність символів від “a” до “j”.

Для доступу до елементів масиву зручно використовувати цикл foreach, який призначений, власне, для роботи з масивами.

<?php

$arr = range(1,20);

foreach ($arr as $current)

echo $current.'<br />';

?>

Наведений код почергово зберігає кожний елемент масиву у змінній $current і потім виводить її значення у вікно браузера.

При створенні масивів у наведених прикладах PHP було надано можливість присвоїти кожному елементові масиву індекс по замовчуванню, тобто перший доданий елемент став 0-м, другий – 1-м і так далі. PHP також підтримує масиви, у яких з кожним елементом можна пов’язати (асоціювати) довільний ключ (індекс).

У наступному прикладові символьні назви слугують як ключі, а числа – як значення.

<?php

$arr = array('the\_first'=>1,'the\_second'=>2,'the\_third'=>3);

echo $arr['the\_first'].'<br />';

echo $arr['the\_second'].'<br />';

echo $arr['the\_third'].'<br />';

?>

Створити масив з багатьма елементами можна також, оголосивши масив з одним елементом і додаючи до нього пізніше решту потрібних елементів:

<?php

$arr = array('the\_first'=>1);

$arr['the\_second']=2;

$arr['the\_third']=3;

foreach ($arr as $current)

echo $current.'<br />';

?>

Оскільки в асоціативних масивах індекси не є числами, то для роботи з такими масивами неможливо скористатися лічильником у циклі for. У цьому випадкові слід застосувати цикл foreach або конструкції list()та each().

<?php

$arr = array('the\_first'=>1,'the\_second'=>2,'the\_third'=>3);

foreach ($arr as $current)

echo $current.'<br />';

?>

Якщо необхідно отримати доступ до індексів масиву, використовують конструкцію:

<?php

$arr = array('the\_first'=>1,'the\_second'=>2,'the\_third'=>3);

foreach ($arr as $key => $value)

echo '$arr['.$key.']='.$value.'<br />';

?>

Результат:

$arr[the\_first]=1  
$arr[the\_second]=2  
$arr[the\_third]=3.

Наведений нижче код виводить вміст масиву за допомогою конструкції each():

<?php

$arr = array('the\_first'=>1,'the\_second'=>2,'the\_third'=>3);

while( $element = each($arr))

{

echo '$arr['.$element['key'].']='.$element['value'].'<br />';

}

?>

У цьому прикладі змінна $element– це масив. При викликові функції each() вона повертає масив з чотирма значеннями і чотирма індексами. Комірки key та 0 містять ключ поточного елементу, value та 1 – його значення.

Важливо: при використанні функції each() слід пам’ятати, що масив відстежує поточний елемент. Якщо в одному і тому ж сценарії необхідно двічі скористатися значеннями одного і того ж масиву, то потрібно за допомогою функції reset() встановити поточний елемент на початок масиву.

Наведемо ряд корисних функцій для роботи з масивами.

sort($array) – впорядковує елементи за алфавітом чи порядком зростання;

asort($array) – те саме, що і sort(), але для масивів з описовими індексами;

ksort($array) – впорядковує ключі у масиві з описовими індексами за зростанням;

rsort($array) – впорядковує елементи за алфавітом чи порядком спадання;

rasort($array) – те саме, що і sort(), але для масивів з описовими індексами;

rksort($array) – впорядковує ключі у масиві з описовими індексами за спаданням.

У PHP відсутня можливість порівняння двох масивів, тому для сортування багатовимірного масиву необхідно створювати деякий ручний метод.

shuffle($array) – розташовує елементи масиву випадковим чином;

array\_push($array, $data) – додає елемент $data у кінець масиву $array;

$array = array\_reverse($array) – змінює порядок розташування елементів на зворотній;

next($array)або each($array) – переміщують показник на елемент вперед на один елемент; next – спочатку змінює покажчик, потім – повертає значення, each–навпаки;

end($array) – встановлює покажчик на елемент у кінець масиву;

prev($array) – встановлює покажчик на елемент на одну позицію ближче до початку масиву;

current  ($array) — повертає поточний елемент масиву

reset($array) – встановлює покажчик масиву на його початок.

sizeof($array) – підраховує кількість елементів у масиві.

2.1.2 Багатократне використання коду

Одне із завдань, які зазвичай ставлять перед собою розробники програмного забезпечення – повторне використання існуючого програмного коду замість написання нового. Таким чином вдається знизити вартість розробки, підвищити надійність за забезпечити певну уніфікацію.

PHP надає два оператори require()та include() для забезпечення повторного використання програмного коду.

Оператор require(‘шлях до файлу’) слугує для включення повного коду файлу-параметру оператора у тому місці файла реципієнта, в якому знаходиться сам оператор.

При цьому розширення імені файла, який включається, ігнорується, тобто він може бути названий довільним чином. Файл, що включається, повинен мати PHP-дескриптори. Інакше вміст файла буде розглянуто як простий текст.

Оператор require()часто застосовується для створення шаблонів сторінок.

Оператор include()практично не відрізняється від оператора require() за винятком того, що require() у випадку невдалого виконання видає повідомлення про непоправну помилку, а include() – тільки попередження.

2.1.3 Створення функцій

Функцією називають незалежний модуль коду, який встановлює інтерфейс введення, виконує певну задачу і за необхідності повертає результат.

Імена функцій у PHP не чутливі до регістру.

Оголошення функції створює нову функцію. Оголошення починається ключовим словом function, воно присвоює функції ім’я, задає необхідні параметри і містить код, котрий виконується при кожному викликові функції.

Оголошення функції має вигляд:

Function my\_function()

{

Echo‘викликано функцію’;

}

Виклик функції реалізується за допомогою оператора my\_function().

Вбудовані функції PHP доступні для усіх сценаріїв, а функції користувача – тільки у тому сценарії, у якому вони оголошені. Є сенс додати усі функції, що часто використовуються, в окремий файл і за допомогою оператора require()надати доступ до них.

Імена функцій користувача вибираються, враховуючи обмеження:

– функція не може мати ім’я таке саме, як у вже існуючої функції;

– ім’я функції може складатися тільки з букв, цифр та символів підкреслювання;

– ім’я функції не може починатися з цифри.

Для нормального функціонування більшість функцій вимагають передачі у них параметрів. Наприклад (виводиться таблиця з 3 рядків і 1 стовпчика):

<?php

Function create\_table($data)

{

echo '<table border=1>';

reset($data); // вказуємо початок

$value=current($data);

while($value)

{

echo "<tr><td>$value</td></tr>\n";

$value = next($data);

}

echo '</table>';

}

$my\_array = array('Рядок 1', 'Рядок 2','Рядок 3');

create\_table($my\_array);

?>

Як і вбудовані функції, функції користувача можуть мати обов’язкові та необов’язкові параметри. Модифікуємо попередній приклад:

<?php

Function create\_table2($data, $border=1, $cellpadding=4, $cellspacing=4)

{

echo "<table border=$border cellpadding=$cellpadding"

"cellspacing=$cellspacing>\n";

reset($data); // вказуємо початок

$value=current($data);

while($value)

{

echo "<tr><td>$value</td></tr>\n";

$value = next($data);

}

echo '</table>';

}

$my\_array = array('Рядок 1', 'Рядок 2','Рядок 3');

create\_table2($my\_array,3,8,8);

?>

Перший параметр функції create\_table2() як і раніше – обов’язковий. Наступні три параметри – не обов’язкові, оскільки для них визначені значення по замовчуванню. Необов’язкові параметри можна передавати не всі, а тільки деякі з них. Параметри присвоюються зліва-направо.

Слід пам’ятати, що пропуск необов’язкового параметру і задавання наступного після нього параметру не допускається. У наведеному прикладі це означає, що за необхідності задати $cellpadding пропустити $border не можна.

Існує можливість дізнатися, скільки та які саме параметри передано.

<?php

Function create\_table2($data, $border=1, $cellpadding=4, $cellspacing=4)

{

echo "<table border=$border cellpadding=$cellpadding"

"cellspacing=$cellspacing>\n";

reset($data); // вказуємо початок

$value=current($data);

while($value)

{

echo "<tr><td>$value</td></tr>\n";

$value = next($data);

}

echo '</table>';

echo'Кількість параметрів: ';

echo func\_num\_args().'<br />';

$args = func\_get\_args();

foreach ($args as $arg)

echo $arg.'<br/>';

}

$my\_array = array('Рядок 1', 'Рядок 2','Рядок 3');

create\_table2($my\_array,3,8,8);

?>

Функція func\_num\_args() визначає, скільки аргументів було передано функції користувача, а func\_get\_args() повертає масив, який містить ці аргументи.

Звичний спосіб передачі параметрів називається передачею за значенням. При цьому зміна значення глобального параметра функції з тіла функції неможлива. Для вирішення проблеми використовується так звана передача за посиланням, яка реалізується шляхом додавання символу амперсанда (&) перед іменем параметра у визначенні функції.

Ключове слово return завершує виконання функції і передає виконання наступному операторові після виклику функції. Також функція завершує свою роботу, коли усі оператори виконані.

Найпоширеніша причина використання оператора return з метою передчасного завершення функції – умова виникнення помилки. Також return використовують, коли функція повинна повертати результат.

PHP підтримує рекурсивні функції. Рекурсивною функцією називають функцію, котра викликає сама себе. Такі функції особливо корисні для переміщення по динамічних структурах даних, таких як зв’язані списки та дерева.

Слід пам’ятати, що рекурсивна функція створює у пам’яті копії самої себе і, відповідно, веде до непродуктивних витрат ресурсів.

Детальнішу інформацію шукайте на сайті <http://www.php.su/> чи будь-якому іншому.