Module 7: Arrays

Manipulate array

associative arrays

Search Arrays

Sort, Merge Arrays

Multidimensional arrays

CS 85:
PHP PROGRAMMING

Santa Monica College
Computer Science &
Information Systems Dept.

This week module will focus on Advance Arrays concepts. As a refresher, arrays are sets of data that can be defined in a PHP script using a key/value pair. Array are essentially a variable that can store multiple values. A value in an Array can be accessed by referencing their key/index within the array variable. Arrays can also contain other arrays inside of them without any restriction (hence building multidimensional arrays). As you program with arrays, there will be a need to manipulate your array, either remove elements in the array, move around elements in an array, add new elements to an array. There is wide range of ways to manipulate elements in an array. That is what we will be studying this week.

From a real world perspective, when a user is visiting an ecommerce website they may add and remove item from their shopping cart. As items are being added to the shopping cart, the backend PHP code will add and remove the user selected items from an array.

PHP has three types of arrays:

- Indexed arrays - Arrays with a numeric index
- Associative arrays - Arrays with named keys
- Multidimensional arrays - Arrays containing one or more arrays

Syntax

Arrays can be created in two ways. The first involves using the function array. The second involves using square brackets.

The array function method

In the *array* function method, you create an array in the scheme of $aArray = array();

For example, to set up the array to make the keys sequential numbers (Example: "0, 1, 2, 3"), you use:

```
$foobar = array($foo, $bar);
```

This would produce the array like this:

```
$foobar[0] = $foo;
$foobar[1] = $bar;
```

Notice how indexed arrays will start with the numerical value 0. Index always start with an index of zero for the first array entry.

It is also possible to define the key value known as an Associative array where the index is now a named key such as foo or bar:

```
$foobar = array('foo' => $foo, 'bar' => $bar);
```

This would set the array like this:

```
$foobar['foo'] = $foo;
$foobar['bar'] = $bar;
```

The square brackets method

The square brackets method allows you to set up by directly setting the values. For example, to make $foobar[1] = $foo, all you need to do is:

```
$foobar[0] = $foo;
```

The same applies for setting the key value foo:

```
$foobar['foo'] = $foo;
```

Examples of arrays

Example #1

This example sets and prints arrays.

PHP Code:

```
<?php
$array =
array("name"=>"Toyota","type"=>"Celica","colour"=>"black","manufacture
d"=>"1991");
$array2 = array("Toyota","Celica","black","1991");
$array3 = array("name"=>"Toyota","Celica","colour"=>"black","1991");
print_r($array);
print_r($array2);
print_r($array3);
?>
```

PHP Output:

```
Array
(
  [name] => Toyota
  [type] => Celica
  [colour] => black
  [manufactured] => 1991
)
Array
(
  [0] => Toyota
  [1] => Celica
  [2] => black
  [3] => 1991
)
Array
(
  [name] => Toyota
  [0] => Celica
  [colour] => black
  [1] => 1991
)
```

HTML Render:

Array ([name] => Toyota [type] => Celica [colour] => black [manufactured] => 1991) Array ([0] => Toyota [1] => Celica [2] => black [3] => 1991) Array ([name] => Toyota [0] => Celica [colour] => black [1] => 1991)

Example #2

The following example will output the identical text as Example #1:

```
<?php
$array['name'] = "Toyota";
$array['type'] = "Celica";
$array['colour'] = "black";
$array['manufactured'] = "1991";

$array2[] = "Toyota";
$array2[] = "Celica";
$array2[] = "black";
$array2[] = "1991";
$array3['name'] = "Toyota";
$array3[] = "Celica";
$array3['colour'] = "black";
$array3[] = "1991";

print_r($array);
print_r($array2);
print_r($array3);
?>
```

Example #3

Using the *Example #1* and *Example #2* above, now you can try and use arrays the same way as normal variables:

PHP Code:

```
<?php
echo "Manufacturer: {$array['name']} \n";
echo "Brand: &lt;b&gt;{$array2['1']}&lt;/b&gt;&lt;br /&gt;\n";
echo "Colour: &lt;b&gt;".$array3['colour']."&lt;/b&gt;&lt;br /&gt;\n";
echo "Year Manufactured: &lt;b&gt;".$array3[1]."&lt;/b&gt;&lt;br
/&gt;\n"
?>
```

PHP Output:

Manufacturer: Toyota
Brand: Celica
Colour: black
Year Manufactured: 1991

HTML Render:

Manufacturer: **Toyota**
Brand: **Celica**
Colour: **black**
Year Manufactured: **1991**

Sample Code: echo each element in a array using the a for loop.

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
?>

</body>
</html>
```

Add and Remove Element to Array

PHP has many built-in functions to control the elements in the array. There is array_shift() and array_unshift() function that will allow the developer to either use array_shift() to remove the first element in the array or the use the function array_unshift() to add an element to the beginning of the a array. A possible use a company has an array of inventory to track and process. When new inventory comes in and it is being processed, use the array_unshift() function to add a new elements to the top of the inventory array.

Syntax: shift($arrayName);

```
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_shft($stack);
print_r($stack);
```

Output:

```
Array
(
    [0] => banana
    [1] => apple
    [2] => raspberry
```

```
)
```

```php
$a=array("a"=>"red","b"=>"green","c"=>"blue");
echo array_shift($a)."<br>";
print_r ($a);
```

Output:

```
red
Array ( [b] => green [c] => blue )
```

To remove array elements from the end of an array use the PHP built in function array_pop() and array_push(). To remove the last element in the array, use the array_pop() function. To add an element to the end of an array, use the array_push() function.

Syntax: array_pop($arrayName);

```php
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_pop($stack);
print_r($stack);
```

```
Output: Array
(
    [0] => orange
    [1] => banana
    [2] => apple
)
```

Syntax: array_push($arrayName);

array_push() treats array as a stack, and pushes the passed variables onto the end of array. The length of array increases by the number of variables pushed.

```php
$stack = array("orange", "banana");
array_push($stack, "apple", "raspberry");
print_r($stack);
```

Output:

```
Array
(
    [0] => orange
    [1] => banana
    [2] => apple
    [3] => raspberry
)
```

The syntax below will have the same effect as the array_push() function. A new element will automatically be placed at the end of the array.

```php
<?php
$array[] = $var;
?>
```

Adding and Removing Elements Within an Array -

Modifying the order or elements in middle of the array will at times be required. By using the array_splice() function it will remove selected elements from an array and replaces it with new elements. The function also returns an array with the removed elements.

Syntax: array_splice(array,start,length,array)

array	Required. Specifies an array
start	Required. Numeric value. Specifies where the function will start removing elements. 0 = the first element. If this value is set to a negative number, the function will start that far from the last element. -2 means start at the second last element of the array.
length	Optional. Numeric value. Specifies how many elements will be removed, and also length of the returned array. If this value is set to a negative number, the function will stop that far from the last element. If this value is not set, the function will remove all elements, starting from the position set by the start-parameter.
array	Optional. Specifies an array with the elements that will be inserted to the original array. If it's only one element, it can be a string, and does not have to be an array.

Sample Code:

```php
$input = array("red", "green", "blue", "yellow");
array_splice($input, 2);
// $input is now array("red", "green")

$input = array("red", "green", "blue", "yellow");
array_splice($input, 1, -1);
// $input is now array("red", "yellow")

$input = array("red", "green", "blue", "yellow");
array_splice($input, 1, count($input), "orange");
// $input is now array("red", "orange")

$input = array("red", "green", "blue", "yellow");
array_splice($input, -1, 1, array("black", "maroon"));
// $input is now array("red", "green",
//           "blue", "black", "maroon")

$input = array("red", "green", "blue", "yellow");
array_splice($input, 3, 0, "purple");
// $input is now array("red", "green",
//           "blue", "purple", "yellow");
```

Sample Code:

```php
// append two elements to $input
array_push($input, $x, $y);
array_splice($input, count($input), 0, array($x, $y));

// remove the last element of $input
array_pop($input);
array_splice($input, -1);

// remove the first element of $input
array_shift($input);
array_splice($input, 0, 1);

// insert an element at the start of $input
array_unshift($input, $x, $y);
array_splice($input, 0, 0, array($x, $y));

// replace the value in $input at index $x
$input[$x] = $y; // for arrays where key equals offset
array_splice($input, $x, 1, $y);
```

To add more than one element within an array, pass the array() construct as the fourth argument to the array_splice() function. Within the array() construct, include the new element values separated by commas, as if you were creating a new array. The following example shows how to add two new elements, “rat” and “rabbit,” between the “mouse” and “snake” elements:

```php
$pets = array(
    "dog",  // first element (0)
    "cat", // second element (1)
    "mouse", // third element (2)
    "snake"); // fourth element (3)

array_splice($pet, 3, 0, array("rat", "rabbit"));
```

Unset function

To remove an element in an array, another option is to use the built in unset() function. The unset() function will remove an element anywhere in the array, but will move move and renumber the other elements in the array. Leave an empty element in the array. The unset() function destroys a given variable.

Syntax: unset ($varName)

```php
function foo()
{
    unset($GLOBALS['bar']);
}
```

```
$bar = "something";
foo();
```

To renumber an array, to remove the empty element use the function array_values() and assignment the return values to a new array without the empty element.

```
unset($pet[2]); //remove cat
$pet = array_values($pets);
//assign a new array without the cat element to the same variable
name.
```

Removing Duplicate Elements

To remove duplicated elements in the array, to ensure all elements in the array are unique. For example, if your company wants to display a book listing, it would be unhelpful to repeat the same book in multiple locations. The PHP built in function array_unique() will remove any duplicated elements in the array. If two or more array values are the same, the first appearance will be kept and the other will be removed.

Syntax: array_unique(array)

```
$a=array("a"=>"red","b"=>"green","c"=>"red");
print_r(array_unique($a));
```

Output

Array ([a] => red [b] => green)

Associative Arrays

Associative array uses a key value format instead of a basic array with an index integer value determined by the placement in the array. A element value in array is reference it's location by the key name instead of a number value.

$arrayName = array(key1 => value2, key2 => name2, …);

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";

//Alternative Syntax

$age["Peter"]="35";
$age["Ben"]="37";
$age["Joe"]="43"
echo "Peter is " . $age['Peter'] . " years old.";
```

Array Pointer

An internal array pointer is a variable that points to the currently select element in an array. Internal array pointers are useful when looping through each element in an array. By using the current()

function, you can simply get a return value of the array element that's currently being pointed to by the internal pointer. It does not move the pointer in any way. If the internal pointer points beyond the end of the elements list or the array is empty, current() returns FALSE.

Related functions

end() - moves the internal pointer to, and outputs, the last element in the array

next() - moves the internal pointer to, and outputs, the next element in the array

prev() - moves the internal pointer to, and outputs, the previous element in the array

reset() - moves the internal pointer to the first element of the array

each() - returns the current element key and value, and moves the internal pointer forward

```php
<?php
$transport = array('foot', 'bike', 'car', 'plane');
$mode = current($transport); // $mode = 'foot';
$mode = next($transport);    // $mode = 'bike';
$mode = current($transport); // $mode = 'bike';
$mode = prev($transport);    // $mode = 'foot';
$mode = end($transport);     // $mode = 'plane';
$mode = current($transport); // $mode = 'plane';

$arr = array();
var_dump(current($arr)); // bool(false)

$arr = array(array());
var_dump(current($arr)); // array(0) { }
?>
```

Loop through Array

The most efficient method to iterate through every element in an array is to use a loop. Normally the ***foreach*** or ***for*** loop are the best options. Using the foreach loop does not move the internal array point with each loop iteration. The next() function should be used to move the internal array pointer. Use the next() function is most important when iterate through a associative array where you might need the key string as well as the element value. By using a loop, with an if statement as seen in the sample code below, elements can be found and extract from the array.

```php
$array = array(
    'fruit1' => 'apple',
    'fruit2' => 'orange',
    'fruit3' => 'grape',
    'fruit4' => 'apple',
```

```
    'fruit5' => 'apple');

// this cycle echoes all associative array
// key where value equals "apple"
foreach ($array as $fruit_name) {
    if ($fruit_name == 'apple') {
        echo key($array).'<br />';
    }
    next($array);
}
```

Search array

There are two function to search an array, the in_array() and array_search() functions. The array_search will search for a value in the array and if there is a element value that match, the index location is returned. For the in_array() function, a TRUE or FALSE value will be returned.

Syntax: in_array(search,array,type)

Parameter	Description
search	Required. Specifies the what to search for
array	Required. Specifies the array to search
type	Optional. If this parameter is set to TRUE, the in_array() function searches for the search-string and specific type in the array.

Sample Code in_array():

```
$people = array("Peter", "Joe", "Glenn", "Cleveland");

if (in_array("Glenn", $people))
{
    echo "Match found";
}
else
{
    echo "Match not found";
}
```

Syntax: array_search(value,array,strict)

value	Required. Specifies the value to search for
array	Required. Specifies the array to search in
strict	Optional. If this parameter is set to TRUE, then this function will search for identical elements in the array. Possible values: • true • false - Default When set to true, the number 5 is not the same as the string 5

Sample Code array_search():

```
$a=array("a"=>"5","b"=>5,"c"=>"5");
echo array_search(5,$a,true); //Output b
```

Find a key

To determine if a key exist in an associative array use the array_key_exists() function.

Syntax: array_key_exists(key,array)

key	Required. Specifies the key
array	Required. Specifies an array

Sample Code:

```
$a=array("Volvo"=>"XC90","BMW"=>"X5");
if (array_key_exists("Volvo",$a))
{
    echo "Key exists!";
}
else
{
    echo "Key does not exist!";
}
```

Remove Array Sections

The array_slice() function returns selected parts of an array.

Syntax: array_slice(array,start,length,preserve)

array	Required. Specifies an array
start	Required. Numeric value. Specifies where the function will start the slice. 0 = the first element. If this value is set to a negative number, the function will start slicing that far from the last element. -2 means start at the second last element of the array.
length	Optional. Numeric value. Specifies the length of the returned array. If this value is set to a negative number, the function will stop slicing that far from the last element. If this value is not set, the function will return all elements, starting from the position set by the start-parameter.
preserve	Optional. Specifies if the function should preserve or reset the keys. Possible values: • true - Preserve keys • false - Default. Reset keys

Sample Code:

```php
$a=array("red","green","blue","yellow","brown");
print_r(array_slice($a,2));
```

//Output: Array ([0] => blue [1] => yellow [2] => brown)

Rearrange Arrays

PHP has many built in function to reorder array base on element value. The arrays can be sorted in alphabetical order ascending and descending, numerical order from largest to smallest or smaller to largest. Array can also be sort by key value and number element value.

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

Sample Code: //Ascending order

```php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);

$clength = count($cars);
for($x = 0; $x < $clength; $x++) {
    echo $cars[$x];
```

```
    echo "<br>";
}
```
//Output

BMW

Toyota

Volvo

Sample Code: //Descending order

```
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);

$clength = count($cars);
for($x = 0; $x < $clength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
```

//Output

Volvo
Toyota
BMW

Merging Arrays

The built in PHP function array_merge() will merge one or more arrays together. If two or more array elements have the same key, the last one overrides the others.

Syntax: `array_merge(array1,array2,array3...)`

array1	Required. Specifies an array
array2	Optional. Specifies an array
array3,...	Optional. Specifies an array

Sample code:

```
$a1=array("a"=>"red","b"=>"green");
$a2=array("c"=>"blue","b"=>"yellow");
print_r(array_merge($a1,$a2));
```

Arrays Difference

If two arrays need to be compared to each other, PHP offers a few different function to compare arrays. The array_diff() and array_intersect() function. The array_diff() function compares the values of two (or more) arrays, and returns the differences. The array_diff() function compares the values of two (or more) arrays, and return an array that contains the entries from array1 that are not present in array2 or array3, etc. The array_intersect() function will compares the values of two (or more) arrays, and returns the matches. This function compares the values of two or more arrays, and return an array that contains the entries from array1 that are present in array2, array3, etc.

Syntax: array_diff(array1,array2,array3...);

array1	Required. The array to compare from
array2	Required. An array to compare against
array3,...	Optional. More arrays to compare against

Sample Code:

```
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$a2=array("e"=>"red","f"=>"black","g"=>"purple");
$a3=array("a"=>"red","b"=>"black","h"=>"yellow");

$result=array_diff($a1,$a2,$a3);
print_r($result);
//Output: Array ( [b] => green [c] => blue )
```

Syntax: array_intersect(array1,array2,array3...);

array1	Required. The array to compare from
array2	Required. An array to compare against
array3,...	Optional. More arrays to compare against

Sample Code:

```
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$a2=array("e"=>"red","f"=>"black","g"=>"purple");
$a3=array("a"=>"red","b"=>"black","h"=>"yellow");

$result=array_intersect($a1,$a2,$a3);
print_r($result);
//Output: Array ( [a] => red )
```

Multidimensional Arrays

Elements in an array can also be an array, allowing for multidimensional arrays. Making it an array of arrays, known as a Multidimensional Array. Multidimensional Arrays can either be index arrays or key value associative arrays.

Dimensional Indexed Arrays

Two dimensional arrays are the most common multidimensional array. The easiest way of thinking of a two dimensional array is to think of a table with row and column headers. Where the row would be the first index to reference and the column would be the second using the bracket notation.

Table[gallon][row];

To create a multidimensional array first consider each row in a table and create a basic array. Do that for each row. Now wrap an array() declaration around all the basic arrays separated by a comma. Now this is an array of arrays or a multidimensional array.

Sample Code // multidimensional index array :

```php
$myArray = array(
    array( "one", "two", "three" ),
    array( "four", "five", "six" )
);

// To get access to the elements of the
// $myArray array we must point to the two
// indices (row and column).
// Remember arrays start at zero

// Displays "six"
echo $myArray[1][2];
```

Sample Code:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

```php
$cars = array
(
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),
    array("Land Rover",17,15)
);
```

```
echo $cars[0][0].": In stock: ".$cars[0][1].", sold:
".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold:
".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold:
".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold:
".$cars[3][2].".<br>";


for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
```

Creating a multidimensional associative array in essentially the same as creating a multidimensional index array but now the key has to be specified.

Sample Code // multidimensional associative/index array:

```
$movies = array(
    array(
        "title" => "Rear Window",
        "director" => "Alfred Hitchcock",
        "year" => 1954
    ),
    array(
        "title" => "Full Metal Jacket",
        "director" => "Stanley Kubrick",
        "year" => 1987
    ),
    array(
        "title" => "Mean Streets",
        "director" => "Martin Scorsese",
        "year" => 1973
    )
);
echo "The title of the first movie:<br />";
echo $movies[0]["title"] . "<br /><br />";

echo "The director of the third movie:<br />";
echo $movies[2]["director"] . "<br /><br />";

echo "The nested array contained in the first element:<br />";
print_r( $movies[0] );
echo "<br /><br />";
//Output
```

```
The title of the first movie:
Rear Window

The director of the third movie:
Martin Scorsese

The nested array contained in the first element:
Array ( [title] => Rear Window [director] => Alfred Hitchcock [year]
=> 1954 )
```

Sample Code // multidimensional associative array:

```php
$cars = array(
    "car1" =>
        array("make" => "Toyota","colour" => "Green","year" =>
1999,"engine_cc" => 1998),
    "car2" =>
        array("make" => "BMW","colour" => "RED","year" =>
2005,"engine_cc" => 2400),
    "car3" =>
        array("make" => "Renault","colour" => "White","year" =>
1993,"engine_cc" => 1395),
)
```

Iterate through multidimensional Array

To iterate through each dimension of a multidimensional array it is easiest to use nested foreach() or for() loops. Add a nested foreach() or for() loop for each dimension in the multidimensional array.

```php
$mealPlan = array( array('chicken', 'beef', 'pork'),
    array('potato','tomato', 'broccoli') );

// count($mealplan) return 2, outter for loop, will loop 2 times
for ($i = 0, $numspecials = count($mealPlan); $i < $numspecials; $i++)
{
    // $numsub is the number of elements in the sub arrays
individually
    for ($m = 0, $numsub = count($mealPlan[$i]); $m < $numsub; $m++) {
        print "Element [$i][$m] is " . $mealPlan[$i][$m] . "\n";
    }
};
```

// Output: Element [0][0] is chicken Element [0][1] is beef Element [0][2] is pork Element [1][0] is potato Element [1][1] is tomato Element [1][2] is broccoli