

## PHP – Back to Basics 3: Arrays

So far in this series, we've looked at variables, which can only hold a single value, either a piece of text or a number for example. Arrays on the other hand allow you to store a number of variables in one place, and are widely used in PHP, in a number of different areas. For example, when you read from a database the resulting records are returned in the form of an array, or when a user submits a form, again the results are returned in the form of an array.

An array allows you to store multiple values in a single variable. You could always define a separate variable for each piece of data you want to store, but what happens when you don't know how many pieces of data you need? An array can contain 4 items or 40 items, and you don't need to know in advance exactly how many pieces of data need to be stored. It also makes the data much easier to work with, in that you can work through every item in the array, or sort the items in numerical or alphabetical order for example.

Each piece of data in an array is called an **element**. Each element in an array has a unique number, allowing you to refer to a specific element which is called an **index**. The index number is assigned as each item is added to the array, in numerical order, starting from 0. Alternatively, you can assign a unique piece of text, called a **key**, to each element in the array to refer to that particular element.

### Creating an Array

To create an array, you can either use an array construct, or by using the array operator and we'll look at each of these methods now.

### Creating Arrays with the array construct

The array construct is used when you want to create an array manually, i.e. you want to add values manually to an array. As an example, we'll create a simple array containing some names using the array construct, as shown below:

```
$names = array("Gareth", "Bruce", "Nancy", "George");
```

This creates an array containing 4 elements, which can be referred to by their index numbers, as shown in the table below:

Index Number	Element Value
0	Gareth
1	Bruce
2	Nancy
3	George

As an example, to access the third element of the array, you could use the following code:

```
echo $names[2];
```

This would output "Nancy" to the browser. Remember, that because the index number starts at 0, rather than 1, to refer to the third element we use the index number 2.

## Creating an array with the array operator

The array construct, shown previously, is fine if you know exactly how many elements you want to add. If you don't however, you can create an array, or add values to an existing array using the array operator. The array operator is simply a set of square brackets e.g. [], with no value in between. As an example, we can recreate the `$names` array from the previous example as shown below.

```
$names[] = "Gareth";  
$names[] = "Bruce";  
$names[] = "Nancy";  
$names[] = "George"
```

This code creates an identical array to the previous one, and we can refer to the elements in the same way using their index number. For example, to refer to the last element in the array you could use:

```
echo $names[3];
```

This would output "George" to the browser.

Using the array operator, you can also add an element to the end of an existing array. For example, add the following line of code to the previous example:

```
$names[] = "Fred";
```

The value "Fred" is then added to the end of the array, and so can be referred to by the index number 4. This also works to add values to an array created by the array construct seen previously.

## Associative Arrays

The previous arrays that we've looked at mean that the elements are referred to in the order in which they were added, and as we've seen these are referred to by an index number. Associative arrays are so called because each element has a name associated with it, so you don't actually need to know the position of a particular element in the array.

## Creating an associative array with the array construct

Creating an associative array with the array construct is very similar to creating a standard array, and an example is shown below:

```
$person = array("name" => "Gareth", "age" => 26, "occupation" => "writer");
```

This can also be written as shown below, which makes it easier to read:

```
$person = array(  
    "name" => "Gareth,
```

```
"age" => 26,  
"occupation" => "writer"  
);
```

In the above example, we have created a name for each element, which is referred to as a *key*. As a key refers to a particular element, each key must be unique otherwise you would overwrite the previous value.

To refer to a particular element in the array, we use the key, as shown in the example below:

```
echo $person["age"];
```

This would output 26 to the browser for the example array shown above.

You can also store the key value in a variable, and refer that to reference a particular element, as shown in the following example:

```
$key = "occupation";  
echo $person[$key];
```

This would output "writer" to the browser.

## Adding to an associative array

To add to an associative array, we just need to use the array operator (`[]`), but specifying a key as shown below.

```
$person['location'] = "UK";
```

This would add a new element to the `$person` array, referenced by the key "location".

You can also create an array from scratch using the array operator. To create our person array, we could use the following code:

```
$person["name"] = "Gareth";  
$person["age"] = "26";  
$person["occupation"] = "writer";
```

And this would create an identical array to the one we created using the array construct.

## Multidimensional Arrays

Arrays don't just have to have a single value for each element; an element can also hold another array itself allowing you to build quite complex data structures. As an example, have a look at the code below:

```
$pets = array(  
    array(  
        "name" => "fido",  
        "animal" => "dog",  
        "age" => 6  
    ),  
);
```

```
        array(
            "name" => "horace",
            "animal" => "hamster",
            "age" => 2
        ),
        array(
            "name" => "dibs",
            "animal" => "cat",
            "age" => 4
        )
    );
```

In the code above, we create an array consisting of three elements, using the standard array construct that we looked at earlier. However, each element *also* consists of an array, again created using the array construct, which again consists of three elements.

We can refer to any element in the three arrays with the standard array notation. For example, to find out what animal "Horace" is, we would use the code:

```
echo $pets[1]["animal"];
```

When this code is run it would output "hamster" to the browser, as it refers to the "animal" key of the second element in the array (index 1).

Again you could also use the array operator as shown below:

```
$pets[] = array(
    "name" => "fido",
    "animal" => "dog",
    "age" => 6
);

$pets[] = array(
    "name" => "horace",
    "animal" => "hamster",
    "age" => 2
);

$pets[] = array(
    "name" => "dibs",
    "animal" => "cat",
    "age" => 4
);

echo $pets[1]["animal"];
```

Again, this would output "hamster" to the browser.

## How are Multidimensional Arrays Used?

A typical example of multidimensional arrays is when you read from a database, which we'll be looking at in this series. When you execute an SQL query against a MySQL database, a multidimensional array is contained, with each element containing an array which represents a single database record. Each element contains an associative array, where the key is the field name from the database table.

## Working with Arrays

So far, we've looked at creating arrays, using both the array construct and the array operator. We're now going to look at how we can work with the arrays once they have been created.

### Finding the number of elements in an array – count()

As we've seen before, we can access a particular element in an array by using the elements index number, for example with the code shown below:

```
echo $pets[2]["name"];
```

However, because arrays are so versatile, often you won't know beforehand how many elements there actually are in an array.

To find the number of elements in an array, we use the PHP **count()** function to which we pass the name of the array we want to query. The `count()` command is really simple to use, as shown in the example code below.

```
$names = array("Gareth", "Bruce", "Nancy", "George");  
echo count($names);
```

If you run this code, you'll see **4** output to the browser, as there are 4 elements in the array. However, remember that because the index number starts at 0, the last element is referred to by the value of `count($names) - 1`.

### Accessing the last element in an array – end()

If you want to access the last element in an array, you can use the PHP **end()** function. Again, this is passed the name of the array that you are interested in, and it returns the value of the last element in the array, as shown in the example code below.

```
$names = array("Gareth", "Bruce", "Nancy", "George");  
echo end($names);
```

When this code is run, it will output "George" to the browser, as this is the value in the last element of the array.

### Outputting all the values in an array – var\_dump()

One extremely useful command for debugging is the PHP **var\_dump()** command. This outputs the contents of an array, and is used as shown below:

```
$names = array("Gareth", "Bruce", "Nancy", "George");  
echo "<pre>";  
var_dump($names);  
echo "</pre>";
```

As you can see, you simply pass the array you want to view to the `var_dump()` command. We have also wrapped the `var_dump()` command with HTML `<pre></pre>` tags, which gives us nicely formatted output. When this code is run, you'll see output as shown below:

```
array(4) {
  [0]=>
  string(6) "Gareth"
  [1]=>
  string(5) "Bruce"
  [2]=>
  string(5) "Nancy"
  [3]=>
  string(6) "George"
}
```

Although this is rarely used to output details to the end user of a web site, it's extremely useful for debugging your code to see the actual contents of an array as you are creating your code.

## Looping through all values in an Array – foreach()

Often, especially when working with results from a database, you'll want to loop through all the values in an array, for example to output the record data to the screen. There are two methods of looping through an array, depending on whether you're working with an indexed array or an associative array, and we are going to look at each method.

### Looping through an indexed array

To loop through an indexed array, we use the PHP **foreach()** command, which runs once for every value in the array. The format for the `foreach()` command is shown below:

```
foreach($array as $temp){
}
```

Where **\$array** is the name of the array, and **\$temp** contains the value of each element in the array, starting from the first element and working through each element in turn.

The easiest way to see how this is used is with a simple example, as shown below.

```
$names = array("Gareth", "Bruce", "Nancy", "George");
foreach($names as $name){
    echo $name;
    echo "<br>";
}
```

When this code is run, it will loop from the first element of the array to the last, and each turn round the loop the current element is held in `$name`, and we output this to the browser, which produces the output shown below:

```
Gareth
Bruce
```

```
Nancy  
George
```

## Looping through an Associative Array

Now that we've looked at looping through an indexed array, we're going to look at looping through an associative array. This still uses the `foreach()` command, but with a slightly different format as shown below:

```
foreach($array as $key=>$value){  
}
```

Again, this loops through each element in the array, but as well as the value of the element being placed in `$value`, the name of its associated key is also placed in the variable `$key`.

You can see a simple example of how this is used in the example code below.

```
$pet = array(  
    "name" => "horace",  
    "animal" => "hamster",  
    "age" => 2  
);  
  
foreach($pet as $key=>$value){  
    echo $key . " - " . $value;  
    echo "<br>";  
}
```

In the code above, we create a simple associative array, and then use the `foreach()` command to loop through each element in the array. Each time round the loop, the name of the elements key is placed in **`$key`**, and the elements value is held in **`$value`**, and we output these to the browser. When the code is run, the following output will be seen.

```
name           -           horace  
animal         -           hamster  
age - 2
```

## Looping through a Multidimensional Array

To loop through a multidimensional array, such as the results from a database query, we again use the `foreach()` command, but this time we use it twice. As a multidimensional array is an array of arrays, we use the `foreach()` command once to loop through each element, and as each element is an array we again use the `foreach()` command to loop through that particular array.

This is demonstrated with the code below:

```
$pets = array(  
    array(  
        "name" => "fido",  
        "animal" => "dog",  
        "age" => 6  
    ),  
);
```

```
        array(
            "name" => "horace",
            "animal" => "hamster",
            "age" => 2
        ),
        array(
            "name" => "dibs",
            "animal" => "cat",
            "age" => 4
        )
    );

foreach($pets as $pet){
    echo "<p>";
    foreach($pet as $key=>$value){
        echo $key . " - " . $value;
        echo "<br>";
    }
    echo "</p>";
}
```

First, we setup a multidimensional array as usual with the `array` construct. Next, we use a `foreach()` loop to loop through each element in the `$pets` array, which returns the current element in `$pet`. Because the element is also an array, we use the `foreach()` command again to loop through that array and output the values to the browser.

When this code is run, you'll see the output shown below.

```
name          -          fido
animal        -          dog
age - 6

name          -          horace
animal        -          hamster
age - 2

name          -          dibs
animal        -          cat
age - 4
```

And you can see each element in the array has been output.

## Joining two Arrays – `array_merge()`

Sometimes, you may need to combine two arrays into a single array, which you can do with the `array_merge()` command. This command takes two arrays as parameters and returns a new array containing the elements from both arrays. It's important that if you do this with associative arrays, each array has different names for the element keys, otherwise the data will be overwritten where the two keys are identical.

As an example of the `array_merge()` command, try the code shown below.

```
$firstArray = array("Gareth", "Bruce");
```

```
$secondArray = array("Nancy", "George");  
$newArray = array_merge($firstArray, $secondArray);  
  
foreach($newArray as $element){  
    echo $element;  
    echo "<br>";  
}
```

Here, we create two arrays, `$firstArray` and `$secondArray`. We then use the `array_merge()` command to merge the two arrays into a new array called `$newArray`. We then use the `foreach()` command to loop through each element in the array and output it to the browser.

When the code is run, the following is output to the browser.

```
Gareth  
Bruce  
Nancy  
George
```

You can see that the two arrays have been merged together into one, in the order in which they appeared in the `array_merge()` command.

## Sorting Arrays

In this next section, we're going to look at sorting the values in an array, using the built in PHP commands.

### Sorting an Indexed Array – `sort()`

The `sort()` command sorts an array into numerical or alphabetical order. The function doesn't actually return a new array, instead it transforms the data into the array passed to it, as shown in the example code below.

```
$array = array("Gareth", "Bruce", "Nancy", "George");  
sort($array);  
  
foreach($array as $element){  
    echo $element;  
    echo "<br>";  
}
```

When this code is run, the following output is sent to the browser.

```
Bruce  
Gareth  
George  
Nancy
```

First, we create an array of names in no particular order. We then use the `sort()` command, passing it the array in `$array`. Finally, we use a `foreach()` loop to output each element in the array to the browser. You can see that the results have been sorted into alphabetical order.

We can also sort an array in numerical order, as shown in the code below.

```
$array = array(10,4,6,2);  
sort($array);  
  
foreach($array as $element){  
    echo $element;  
    echo "<br>";  
}
```

When this code is run, the following output is sent to the browser, and you can see that the numbers have been sorted in numerical order.

```
2  
4  
6  
10
```

If an array contains both numerical and alphabetical values, then the array is sorted first into alphabetical order and then numerical order, as shown in the code below.

```
$array = array("Gareth",10,4,"Bruce",6,2,"Nancy");  
sort($array);  
  
foreach($array as $element){  
    echo $element;  
    echo "<br>";  
}
```

When the code is run, the following output is sent to the browser.

```
Bruce  
Gareth  
Nancy  
2  
4  
6  
10
```

Note that its important that you don't pass an associative array to the `sort()` command. If you do this, the values will be sorted into alphabetical or numerical order, however they will no longer be associated with the relevant keys, and so the results will not be as expected.

## Sorting an Associative Array by Value – `asort()`

With the `sort()` command, we saw that you couldn't sort an associative array because, although the values are sorted, their association with their relevant keys is not maintained. In order to sort the values in the array, and maintain the association between the keys and the values we use the PHP `asort()` command. This is used in a similar method to the `sort()` command, as shown in the code below.

```
$array = array(  
    "key1" => 10,
```

```
        "key2" => 4,  
        "key3" => 6,  
        "key4" => 12  
    );  
  
    asort($array);  
  
    foreach($array as $key=>$value){  
        echo $key . " - " . $value;  
        echo "<br>";  
    }
```

Here, we first create an array containing values in no particular order. We then pass this array to the `asort()` command to sort the array into numerical order. Finally, we use the `foreach()` command to output the key and value for each element to the browser. When the code is run you will see the output shown below.

key2	-	4
key3	-	6
key1	-	10
key4 - 12		

Here you can see that the values have been sorted into numerical order, but the association between each value and its key has been maintained.

Again, if the array contains values which are both alphabetical and numerical, the values will be sorted first into alphabetical order, and then into numerical order as shown by the code below.

```
$array = array(  
    "key1" => "Gareth",  
    "key2" => 4,  
    "key3" => "Bruce",  
    "key4" => 1  
);  
  
asort($array);  
  
foreach($array as $key=>$value){  
    echo $key . " - " . $value;  
    echo "<br>";  
}
```

When this code is run, the following output is sent to the browser.

key3	-	Bruce
key1	-	Gareth
key4	-	1
key2 - 4		

And you can see that again the values in the array have been correctly sorted.

## Sorting an Associative Array by Key – `ksort()`

The PHP `ksort()` command is very similar to the previous `asort()` command, but instead of sorting the array in order by the values, the array is sorted in order of the keys.

Again, this can be demonstrated with a simple example, as shown in the code below.

```
$pet = array(
    "name" => "horace",
    "animal" => "hamster",
    "age" => 2
);

ksort($pet);

foreach($pet as $key=>$value){
    echo $key . " - " . $value;
    echo "<br>";
}
```

First, we create an array where the arrays keys are in no particular order. We then use the `ksort()` command to sort the array into order by the key. We then output the elements to the browser with the `foreach()` command.

When this code is run, the following results are displayed.

```
age          -          2
animal       -          hamster
name - horace
```

And you can see that the elements have now been order in alphabetical order by each elements key, and the association between the keys and values are still maintained.

This completes this tutorial on arrays. In a future article in this series, we will be looking at working with HTML forms, and how to work with the data entered by the user. When a form is submitted using the POST method, the data is returned in the form of an array, and we will be using the techniques learnt in this tutorial to work with the data.

## Summary

In this tutorial, we first looked at what an array is, and how to create an array using the array construct and the array operator, and how to add data to the end of an existing array. We then looked at creating multidimensional arrays, which are arrays where each element contains another array.

We then looked at how we could work with the arrays that we created, so that we could count the number of elements in the array, and loop through every value in an array so we could work with each element in turn, as well as merging the contents of two arrays into a single array.

Finally, we looked at how we could sort an array into alphabetical and numerical order, sorting by value or by key, and how we could sort an associative array and still maintain the association between the arrays keys and values.