PHP 101 (part 4): The Food Factor

A Big Mistake

Having spent lots of time travelling around the outer landscape of PHP – learning all about control structures, operators and variables – you're probably bored. You might even be thinking of dropping out right now, and instead spending your time more constructively (or so you think) in front of the idiot box.

That would be a big mistake. And when I say big, I mean humongous.

You see, if you forego this segment of the tutorial for the dubious charms of Ally McBeal, you're going to miss out on one of PHP's coolest variable types. It's a little thing called an array, and I'm not exaggerating when I tell you that once you're on speaking terms with it, you're never going to look at a PHP script the same way again. But hey, don't take my word for it... toss that remote aside and come see for yourself!

Fruity Pizza

Thus far, the variables we've discussed contained only a single value, such as:

```
<?php

$i = 5;
```

?>

However, array variables are a different kettle of fish altogether.

An array is a complex variable that allows you to store multiple values

in a single variable (which

is handy when you need to store and represent

related information). Think of the array

variable as a “container”

variable, which can contain one or more values. For example:

```
<?php
// define an array
$pizzaToppings = array('onion', 'tomato', 'cheese', 'anchovies', 'ham','pepperoni');
print_r($pizzaToppings);
?>
```

Here, `$pizzaToppings` is an array variable, which contains the values

`'onion'`, `'tomato'`, `'cheese'`, `'anchovies'`, `'ham'` and `'pepperoni'`.

(Array variables

are particularly useful for grouping related values together.)

`print_r()` is a special function that allows you to take a sneak peek

inside an array. It’s more useful for debugging (finding out why your

script doesn’t work) than it is for display purposes, but I’ll use it

here so you can see what’s going on under the surface. You do have

your server running and your browser open, right?

The various elements of the array are accessed via an index number, with the first element starting at zero. So, to access the element `'onion'`, you would use the notation `$pizzaToppings[0]`, while `'anchovies'` would be `$pizzaToppings[3]` – essentially, the array variable name followed by the index number enclosed within square braces.

PHP also allows you to replace indices with user-defined “keys” (即相當於 Map), in order to create a slightly different type of array. Each key is unique, and corresponds to a single value within the array.

```
<?php
// define an array
$fruits =
array('red' => 'apple', 'yellow' => 'banana', 'purple' =>'plum', 'green' => 'grape')
;
print_r($fruits);
?>
```

In this case, `$fruits` is an array variable containing four key-value pairs. (The `=>` symbol is used to indicate the association between a key and its value.) In order

to access the value `'banana'`, you would use the notation `$fruits['yellow']`, while the value `'grape'` would be accessible via the notation `$fruits['green']`.

This type of array is sometimes referred to as a “hash” or “associative array”. If you’ve ever used Perl, you’ll see the similarities to the Perl hash variable.

Eating Italian

The

simplest was to define an array variable is the `array()` function.

Here’s how:

```
<?php
// define an array
$pasta = array('spaghetti', 'penne', 'macaroni');
?>
```

The rules for choosing an array variable name are the same as those for any other PHP variable: it must begin with a letter or underscore, and can optionally be followed by more letters, numbers and underscores.

Alternatively, you can define an array by specifying values for each element in the index notation, like this:

```
<?php
// define an array
$pasta[0] = 'spaghetti';
$pasta[1] = 'penne';

$pasta[2] = 'macaroni';
?>
```

If you’re someone who prefers to use keys rather than default numeric indices, you might
prefer the following example:

```
<?php
// define an array
$menu['breakfast'] = 'bacon and eggs';

$menu['lunch'] = 'roast beef';
$menu['dinner'] = 'lasagna';
?>
```

You can add elements to the array in a similar manner. For example, if you wanted to add
the element ‘`green olives'` to the `$pizzaToppings` array, you would

use something like this:

```
<?php
// add an element to an array
$pizzaToppings[3] = 'green olives';
?>
```

In order to modify an element of an array, simply assign a new value to the corresponding scalar variable. If you wanted to replace `'ham'` with `'chicken'`, you'd use:

```
<?php
// modify an array
$pizzaToppings[4] = 'chicken';
?>
```

You can do the same using keys. The following statement modifies the element with the key 'lunch' to a different value:

```
<?php
```

```
// modify an array
$menu['lunch'] = 'steak with mashed potatoes';
?>
```

Push And Pull

You can also add an element to the end of an existing array with the `array_push()` function:

```
<?php
// define an array
$pasta = array('spaghetti', 'penne', 'macaroni');
// add an element to the end

array_push($pasta, 'tagliatelle');
print_r($pasta);
?>
```

And you can remove an element from the end of an array using the interestingly-named `array_pop()` function.

```
<?php
```

```
// define an array
$pasta = array('spaghetti', 'penne', 'macaroni');
// remove an element from the end

array_pop($pasta);
print_r($pasta);
?>
```

If you need to pop an element off the top of the array, you can use the `array_shift()` function:

```
<?php
// define an array
$pasta = array('spaghetti', 'penne', 'macaroni');
// take an element off the top

array_shift($pasta);
print_r($pasta);
?>
```

And the `array_unshift()` function takes care of adding elements to the beginning of the array.

```
<?php
// define an array
$pasta = array('spaghetti', 'penne', 'macaroni');
// add an element to the beginning

array_unshift($pasta, 'tagliatelle');
print_r($pasta);
?>
```

The `array_push()` and `array_unshift()` functions don’t work with associative arrays; to add elements to these arrays, it’s better to use the $arr[$key] = $value notation to add new values to the array.

The `explode()` function splits a string into smaller components, based on a user-specified delimiter, and returns the pieces as elements as an array.

```
<?php
// define CSV string
$str = 'red, blue, green, yellow';
// split into individual words

$colors = explode(', ', $str);
```

```
print_r($colors);
?>
```

To do the reverse, you can use the `implode()` function, which creates a single string from all the elements of an array by joining them together with a user-defined delimiter. Reversing the example above, we have:

```
<?php
// define array
$colors = array ('red', 'blue', 'green', 'yellow');
// join into single string with 'and'

// returns 'red and blue and green and yellow'

$str = implode(' and ', $colors);
print $str;
?>
```

Finally, the two examples below show how the `sort()` and `rsort()` functions can be used to sort an array alphabetically (or numerically), in ascending and descending order respectively:

```
<?php
// define an array
$pasta = array('spaghetti', 'penne', 'macaroni');
// returns the array sorted alphabetically

sort($pasta);

print_r($pasta);

print "<br />";
// returns the array sorted alphabetically in reverse
rsort($pasta);

print_r($pasta);
?>
```

Looping the Loop

So that takes care of putting data inside an array. Now, how about getting it out?

Retrieving data from an array is pretty simple: all you need to do is access the appropriate element of the array using its index number. To read an entire array you simply loop over it, using any of the loop constructs you learned about

in Part Three of this tutorial.

How about a quick example?

```
<html>

<head></head>

<body>

My favourite bands are:

<ul>

<?php

// define array

$artists = array('Metallica', 'Evanescence', 'Linkin Park', 'Guns n Roses');

// loop over it and print array elements

for ($x = 0; $x < sizeof($artists); $x++) {

    echo '<li>'.$artists[$x];

}

?>

</ul>

</body>
```

`</html>`

When you run this script, here's what you'll see:

My favourite bands are:

- *Metallica*
- *Evanescence*
- *Linkin Park*
- *Guns n Roses*

In this case, I've defined an array, and then used the `for()` loop to: run through it, extract the elements using the index notation, and display them one after the other.

I'll draw your attention here to the `sizeof()` function. This function is one of the most important and commonly used array functions. It returns the size of (read: number of elements within) the array. It is mostly used in loop counters to ensure that the loop iterates as many times as there are elements in the array.

If you're using an associative array, the `array_keys()` and `array_values()` functions come in handy, to get a list of all the keys and values within the array.

```
<?php
// define an array
$menu = array('breakfast' => 'bacon and eggs', 'lunch' => 'roast
```

```
beef','dinner' => 'lasagna');
/* returns the array ('breakfast', 'lunch', 'dinner') with numeric indices */

$result = array_keys($menu);

print_r($result);

print "<br />";
/* returns the array ('bacon and eggs', 'roast beef', 'lasagna') with numeric indices
*/

$result = array_values($menu);

print_r($result);
?>
```

What's That Noise?

There is, however, a simpler way of extracting all the elements of an array. PHP 4.0 introduced a spanking-new loop type designed specifically for the purpose of iterating over an array: the `foreach()` loop. (It is similar in syntax to the Perl construct of the same name.) Here's what it looks like:

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

```
    do this!

}
```

A `foreach()` loop runs once for each element of the array passed to it as argument, moving forward through the array on each iteration. Unlike a `for()` loop, it doesn't need a counter or a call to `sizeof()`, because it keeps track of its position in the array automatically. On each run, the statements within the curly braces are executed, and the currently-selected array element is made available through a temporary loop variable.

To better understand how this works, consider this rewrite of the previous example, using the `foreach()` loop:

```
<html>

<head></head>
<body>

My favourite bands are:

<ul>
```

```
<?php
// define array

$artists = array('Metallica', 'Evanescence', 'Linkin Park', 'Guns n Roses');
// loop over it

// print array elements

foreach ($artists as $a) {

    echo '<li>'.$a;

}
?>
</ul>


</body>


</html>
```

Each time the loop executes, it places the currently-selected array element in the temporary variable `$a`. This variable can then be used by the statements inside the loop block.

Since a `foreach()` loop doesn't need a counter to keep track of where it

is in the array, it is lower-maintenance and also much easier to read than a standard `for()` loop. Oh yeah… and it also works with associative arrays, with no extra programming needed.

Music for the Masses

In addition to their obvious uses, arrays and loops also come in handy when processing forms in PHP. For example, if you have a group of related checkboxes or a multi-select list, you can use an array to capture all the selected form values in a single variable, to simplify processing. Consider the following example, which illustrates this:

```
<html>

<head></head>
<body>
<?php

// check for submit
if (!isset($_POST['submit'])) {

    // and display form
```

```
    ?>
        <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
        <input type="checkbox" name="artist[]" value="Bon Jovi">Bon Jovi

    <input type="checkbox" name="artist[]" value="N'Sync">N'Sync

    <input type="checkbox" name="artist[]" value="Boyzone">Boyzone

    <input type="checkbox" name="artist[]" value="Britney Spears">Britney
Spears

    <input type="checkbox" name="artist[]" value="Jethro Tull">Jethro Tull

    <input type="checkbox" name="artist[]" value="Crosby, Stills &
Nash">Crosby, Stills & Nash

    <input type="submit" name="submit" value="Select">
        </form>
<?php

    }

else {
```

```
    // or display the selected artists

    // use a foreach loop to read and display array elements

    if (is_array($_POST['artist'])) {

        echo 'You selected: <br />';
            foreach ($_POST['artist'] as $a) {

          echo "<i>$a</i><br />";

           }

        }
        else {

         echo 'Nothing selected';

    }

}

?>
</body>
```

```
</html>
```

When the above form is submitted, PHP will automatically create an array variable, and populate it with the items selected. This array can then be processed with a `foreach()` loop, and the selected items retrieved from it.

You can do this with a multi-select list also, simply by using array notation in the select control's "name" attribute. Try it out for yourself and see… and make sure you tune in for the next PHP 101 tutorial, same time, same channel.