COMP 3015

Introduction to PHP

Lesson 03

Arrays

An array is a list – a group – of (hopefully) related data. There are many array functions built into PHP. The complete details can be found here http://www.php.net/manual/en/ref.array.php.

The variables we have seen so far store one single value. An array is a datatype that can store many values. Think it as a row of boxes and each box can hold its own value.

```
$countries:  ["Canada"] ["USA"] ["Japan"] ["England"]
```

An array is made up of a bunch of *elements*. Each element has a **value** (eg. The string “Canada”) and an **index** (eg. The first box is at index 0, the second is at 1). In the above example, Canada is at index 0, USA is at index 1.

Initializing Arrays

Simply assign values to the array variable. Every time you do this, a new element is created and its value is set. The square brackets `[ ]` indicate that this is an array.

```
$countries[] = "Canada";  // Canada at 0
$countries[] = "Fiji";    // Fiji is at 1
$countries[] = "England"; // England is at index 2
```

As indicated above, the first element is index 0 by default. So this code is equivalent to method

```
$countries[0] = "Canada"; // explicitly set an index
$countries[1] = "Fiji";
$countries[2] = "England";
```

You can also use any indexes

```
$countries[7] = "Canada";
$countries[40] = "Fiji";
$countries[22] = "England";
```

You could also do this all on one statement, letting PHP provide the indexes

```
$countries = array("Canada", "Fiji", "England");
```

Or specify the indexes yourself, in one statement

```
$countries = array(
    7 => "Canada",
    40 => "Fiji",
    22 => "England"
);
```

Counting Elements in Arrays

The `count()` function tells the number of elements in the specified array.

```
echo count($countries);
```

prints `3` for our array.

This code will print out all of the countries in the array `$countries`

```
$countries[] = "Canada";
$countries[] = "Fiji";
$countries[] = "England";

$total = count($countries);

for ($index = 0; $index < $total; $index++)
{
    echo $countries[$index] . "<br/>";
}
```

Deleting from Arrays

The entire `$countries` array can be deleted using unset

```
unset($countries);
```

We can use the `unset()` function to delete individual elements from an array. However, this may lead to unexpected results, like this

```
echo "Before deleting:<br/>";
$countries[] = "Canada";
$countries[] = "Fiji";
$countries[] = "England";
$total = count($countries);

for ($index = 0; $index < $total; $index++)
{
    echo $countries[$index] . "<br/>";
}

unset ($countries[1]); // remove Fiji
```

```
echo "<hr/>After deleting: <br/>";
$total = count($countries);

for ($index = 0; $index < $total; $index++)
{
    echo $countries[$index] . "<br/>";
}
```

Run the above. What happens?

The "Fiji" element was deleted. But why wasn't "England" printed? "England" was not printed because it is still element number two. However, our loop only prints out elements zero and one. In other words, even though the second element was removed, the array was not re-indexed.

We can fix this problem several ways. One way is to loop up to element two. However, there is a better solution. The `foreach` construct offers a perfect solution. This is the final loop type which we will learn. The `foreach` construct works only with arrays.

foreach

The `foreach` construct looks like this

```
foreach($array as $value)
{
    // ...do something with $value...
}
```

When the `foreach` construct gets executed the first time, it begins at the first element of the array. It creates a copy of the array and operates on the copy, so if the `foreach` loop changes things inside the array, the original is not changed. Only the copy is changed, the original is not altered.

For example

```
echo "Before deleting:<br/>";

$countries[] = "Canada";
$countries[] = "Fiji";
$countries[] = "England";

foreach($countries as $country)
{
    echo $country . "<br/>";
}

unset($countries[1]); // remove Fiji

echo "<hr/>After deleting:<br/>";
foreach($countries as $country)
{
    echo $country . "<br/>";
}
```

Sorting Arrays

`sort()` http://www.php.net/manual/en/function.sort.php

The most basic built-in sorting function is `sort()`. `sort()` arranges the array elements into alphabetical order: numbers first, then punctuation, then letters. The array indices are also rearranged to reflect the new array order: the first element is now element 0, and so on.

`asort()` changes the order of the elements but does not change the indices.

`rsort()` works like `sort()`, but sorts the array in reverse alphabetical order (and rearranges the keys).

`arsort()` works like `asort()`, but sorts the array in reverse alphabetical order (and does not rearrange the keys).

There are also more complicated sorting functions such as `ksort()`, and `krsort()`. Try them out; here's a sample

```
$countries[0] = "Canada";
$countries[1] = "Fiji";
$countries[2] = "England";

sort($countries);

// Array is now in order Canada, England, Fiji
// Canada is still index 0
// England is now index 1
// Fiji is now index 2
```

Forms

HTML files can contain forms which obtain data from users. This data can be used by PHP scripts; for example PHP can validate it, store it, and manipulate it.

HTML forms can use several "methods" to send data PHP scripts. The most common methods are `GET` and `POST`.

$_GET

When a PHP script is invoked, the variables `$_GET` and `$_POST` are defined automatically by PHP and contain information sent to by the form via `GET` and `POST` methods. `$_GET` and `$_POST` are arrays. To access the values sent, we use the `name` attribute from the form as a key.

Consider this HTML form

```
<form action="search.php" method="GET">
    Type your first name:
    <input type="text" name="firstname">
    <br/>
    Type your last name:
    <input type="text" name="lastname">
    <br/>
    <input type="submit">
</form>
```

Form data is sent to the file `search.php` which is located in the same directory as the html file. It is sent via the `GET` method, so it will be accessible inside the `search.php` script using the `$_GET` array variable.

Inside `search.php`, we can access the values that were sent from the form by addressing it with the name attribute as keys

```
echo $_GET["firstname"] . " " . $_GET["lastname"];
```

Notice that the keys in the [ ] must match the names from the form.

When the `GET` method is used to transfer data to a script, the data is transmitted via the URL, visible in the address bar.

In fact, you could call the `search.php` script with names simply by typing them in the address bar of the browser, like this

http://localhost/search.php?firstname=john&lastname=doe

This could be useful, for instance if user "john doe" wanted to bookmark this page, they could bookmark it with their name directly as part of the URL, and they would not have to start at the html page and re-type their name each visit. This URL-method of specifying data is not possible using the `POST` method.

$_POST

Consider this HMTL form, which uses the `POST` method instead of `GET`

```
<form action="search.php" method="POST">
    Type your first name:
    <input type="text" name="firstname">
    <br />
    Type your last name:
    <input type="text" name="lastname">
    <br/>
    <input type="submit">
</form>
```

This data will not be accessible to the `$_GET` variable. Use the `$_POST` variable instead

```
echo $_POST["firstname"] . ' ' . $_POST["lastname"];
```

Because form data is encapsulated in the variables `$_GET` and `$_POST` as arrays, we can use the `foreach` construct to loop through them to work with form data

```
foreach ($_POST as $formdata)
{
    echo $formdata . "<br/>";
}
```

Using $_GET and $_POST with non-text form data

Other than the `text` input type as shown above, you can use `checkbox`, `dropdowns` and `radio` as well.

Here is an HTML file which gets user input with a variety of form input types

```
<form action = "signup.php" method="POST">

<span>check which school you attend:</span>
<input type="radio" name="school" value="bcit">bcit<br/>
<input type="radio" name="school" value="ubc">ubc<br/>
<input type="radio" name="school" value="sfu">sfu<br/>

<span>check here if human: </span>
<input type="checkbox" name="human" value="human">
<br/>

birthplace:
<select name="birthplace">
    <option value="bc">bc</option>
    <option value="alberta">alberta</option>
    <option value="Saskatchewan">saskatchewan</option>
</select>
<input type="submit"/>
</form>
```

This input is sent to `signup.php`, which can be accessed via the $_POST array

```
echo $_POST["school"] . "<br/>";
echo $_POST["human"] . "<br/>";
echo $_POST["birthplace"] . "<br/>";
```

More on Functions - include/require

You can define functions anywhere in your PHP files but make sure they are defined before they are used (i.e. they are higher in the code). Another good idea is to put all of your functions into external files. This helps organize your code and improves maintaiblity.

Let's create a PHP file to contain the two functions we created: `doubleTheNumber` and `arrayToSelect`. Here is the entire file

```
<?php

/**
 * The following function converts an array into a
 * string that represents an HTML select element:
 */
function arrayToSelect($theArray)
{
    $theSelect = "<select name=\"mySelect\">";

    foreach($theArray as $element)
    {
        $theSelect = $theSelect . "<option>".
        $element ."</option>";
    }

    $theSelect = $theSelect . "</select>";
    return $theSelect;
}
```

```
/**
 * This function receives one parameter,
 * $someNumber, doubles it,
 * and returns the doubled value
 */
function doubleTheNumber ($someNumber)
{
    return $someNumber * 2;
}

?>
```

Now, we can use these two functions in other PHP files. But first, we must let PHP know where the functions are. We can use the `require()` or the `include()` built-in functions.

require

When you `require()` a file, you are essentially taking its contents and putting them into your script. If the file you are requiring produces a failure, your PHP script will halt.

The `include()` function works the same as the `require()` function, but if the file you are including produces a failure, your PHP script will continue (with only a warning).

Assuming you saved the above functions file as `myfunctions.php`, to require it in a separate file you simply

```
require("myfunctions.php");

$number = 17;
echo doubleTheNumber($number);
```

There's also a function called `include_once()` and `require_once()`. Calling `include_once()`/`require_once()` means if the file has already been included, don't include it again. If the file has not yet been included, then include it now. This causes a slight performance overhead but it does ensure you don't include/require the same functions twice, resulting in redefining the functions which causes PHP to error.