Web Technology II - BIM 4th SEMESTER

Unit 4: Working with Arrays

Array Basics

An array stores multiple values in one single variable. A variable is a storage area holding a number or text. The problem is that a variable will hold only one value. An array is a special variable, which can store multiple values in one single variable. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

$cars1="Saab";

$cars2="Volvo";

$cars3="BMW";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array! An array can hold all your variable values under a single name. And you can access the values by referring to the array name. Each element in the array has its own index so that it can be easily accessed. In PHP, there are three kind of arrays:

1. Numeric array - An array with a numeric index
2. Associative array - An array where each ID key is associated with a value
3. Multidimensional array - An array containing one or more arrays

Numeric Arrays

A numeric array stores each array element with a numeric index. There are two methods to create a numeric array. In the following example the index are automatically assigned (the index starts at 0):

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

In the following example we assign the index manually as:

$cars[0]="Saab";

$cars[1]="Volvo";

$cars[2]="BMW";

$cars[3]="Toyota";

Example

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
// To Print all we can use all do loop, for loop, while loop and for each loop as:
foreach($cars as $value)
{
echo $value;
}
?>
The code above will output:
Saab and Volvo are Swedish cars.
```

Associative Arrays

An associative array, each ID key is associated with a value. When storing data about specific named values, a numerical array is not always the best way to do it. With associative arrays we can use the values as keys and assign values to them.

Example 1:

In this example we use an array to assign ages to the different persons:

$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);

Example 2

This example is the same as example 1, but shows a different way of creating the array:

$ages['Peter'] = "32";

$ages['Quagmire'] = "30";

$ages['Joe'] = "34";

The ID keys can be used in a script:

Example 3

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
echo "Peter is " . $ages['Peter'] . " years old.";
?>
The code above will output:
Peter is 32 years old.
```

Example 4: Using loop

```
<?php
$movie = array( "title" => "Rear Window",
                "director" => "Alfred Hitchcock",
                "year" => 1954,
                "minutes" => 112 );

echo "<dl>";

foreach ( $movie as $key => $value ) {
  echo "<dt>$key:</dt>";
  echo "<dd>$value</dd>";
}
echo "</dl>";
?>
```

Output:

```

title:
    Rear Window
director:
    Alfred Hitchcock
year:
    1954
minutes:
```

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

$families = array("Griffin"=>array("Peter","Lois","Megan"),"Quagmire"=>array("Glenn"),

"Brown"=>array("Cleveland","Loretta","Junior"));

The array above would look like this if written to the output:

Array ([Griffin] => Array([0] => Peter [1] => Lois [2] => Megan)

[Quagmire] => Array ([0] => Glenn)

[Brown] => Array ([0] => Cleveland [1] => Loretta [2] => Junior))

```
<?php
$families = array("Griffin"=>array("Peter","Lois","Megan"),"Quagmire"=>array("Glenn"),
"Brown"=>array("Cleveland","Loretta","Junior"));
echo "Is " . $families['Griffin'][2] . " a part of the Griffin family?";
?>
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

Example 2:

```
<?php
$families = array("Griffin"=>array("Peter","Lois","Megan"),"Quagmire"=>array("Glenn"),
"Brown"=>array("Cleveland","Loretta","Junior"));
for($i=0;$i<=2;$i++)
{
echo $families ['Griffin'][$i];
}
```

Output:

```
Peter Lois Megan
// Using foreach
foreach($families as $key=>$value)
{
echo $families[$value];
}
?>
```

Example 3:

```
<?php
//we create this array

$people= array(

array(
"name" => "Jennifer Kimbers",
"email" => "abc@gmail.com",
"city" => "Seattle",
"state" => "Washington"),

array(
"name" => "Rodney Hutchers",
"email" => "def@gmail.com",
"city" => "Los Angeles",
"state" => "California"),

array(
"name" => "Robert Smith",
"email" =>  "ghi@gmail.com",
"city" => "Michigan",
"state" => "Missouri")

);


$num=0;
```

```
foreach ($people as $person)
{
$num++;
echo "<br><b># $num</b><br>";
foreach ($person as $key=>$value)
{
echo "$key: $value <br>";

}}
?>
```

Actual PHP Output

```
# 1
name: Jennifer Kimbers
email: abc@gmail.com
city: Seattle
state: Washington

# 2
name: Rodney Hutchers
email: def@gmail.com
city: Los Angeles
state: California

# 3
name: Robert Smith
email: ghi@gmail.com
city: Michigan
state: Missouri
```

Looping through Arrays

1. While Loop

The while loop is probably the most popular because of the recognizable and meaningful name. I always like to think of the while loop as the following.

PHP Code

```
<?php
$CodeWallTutorialArray = array("Eggs", "Bacon", "HashBrowns", "Beans", "Bread", "RedSauce");
$arrayLength = count($CodeWallTutorialArray);
        $i = 0;
        while ($i < $arrayLength)
        {
            echo $CodeWallTutorialArray[$i] ."<br />";
            $i++;
        }?>
```

Output

```
Eggs
Bacon
HashBrowns
Beans
Bread
RedSauce
```

2. For Loop

It uses the very same concept for looping and picking out information from the array. Three parameters are needed for the for loop and they are as follows –

1. An initial counter set to a certain value, usually zero.
2. A Boolean test, usually involving the initial counter.

PHP Code

```
<?php
$CodeWallTutorialArray = aarray("Eggs", "Bacon", "HashBrowns", "Beans", "Bread", "RedSauce");

        for ($i = 0; $i < count($CodeWallTutorialArray); $i++)
{
        echo $CodeWallTutorialArray[$i] ."<br />";
 }
?>
```

Output

```
Eggs
Bacon
HashBrowns
Beans
Bread
RedSauce
```

3. Foreach Loop

It just simply pass in the array and do what you want with it. It’s both easy to use, understand and comes in handy for many use-cases. There isn’t a mandatory rule to use a numeric index to pick out data values.

PHP Code

```
<?php
$foodArray = array("Eggs", "Bacon", "HashBrowns", "Beans", "Bread");


     foreach ($foodArray as $food)  {
        echo $food ."<br />"   }
?>
```

Output

```
Eggs
Bacon
HashBrowns
Beans
Bread
```

4. Do While Loop

With the do while loop, you will need to create a test within the while operator, just as you would in a pure while loop case.

PHP Code

```
<?php
$foodArray = array("Eggs", "Bacon", "HashBrowns", "Beans", "Bread");
    $i = 0;

    do {
      echo $foodArray[$i] . "<br />";
      $i++;
    }
    while ($i < count($foodArray));
?>
```

Output

```
Eggs
Bacon
HashBrowns
Beans
Bread
```

Modifying Arrays

This can be used to modify the value of array item by using predefined function **array_map** function but generally programmer can modify the items as below:

```
<?php
        $fruits[0] = "pineapple";
        $fruits[1] = "pomegranate";
        $fruits[2] = "tangerine";

        $fruits[2] = "watermelon";

        $fruits[] = "grapes";

        for ($index = 0; $index < count($fruits); $index++){

          echo $fruits[$index], "<BR>";          }
     ?>
Output:
pineapple
pomegranate
watermelon
grapes
```

Sorting Arrays

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.

PHP array sort functions are as below:

- 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

Sort Array in Ascending Order - sort()

The following example sorts the elements of the $cars array in ascending alphabetical order:

Example 1:

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

The following example sorts the elements of the $numbers array in ascending numerical order:

Example 2:

```
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
?>
```

Sort Array in Descending Order - rsort()

The following example sorts the elements of the $cars array in descending alphabetical order:

Example 1:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);
?>
```

The following example sorts the elements of the $numbers array in descending numerical order:

Example 2:

```
<?php
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers);
?>
```

Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
?>
```

Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
?>
```

Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
?>
```