Episode 10

Arrays

Introduction

An array is a data type that can hold more than one value. Each value is called an element. There are two types of arrays in PHP: indexed and associative. In this episode, we are going to learn how to create an array, how to add and delete elements from an array and how to use the foreach loop to more easily access elements in an array.

Goals

In this section of the course your goals are:

- [ ] To learn how to create an array.
- [ ] To learn how to add and delete elements from an array.
- [ ] To learn how to create an associative array.
- [ ] To learn how to loop through an indexed array.
- [ ] To learn how to use foreach loops to access elements in an associative array.

Creating an array

You can create an array with a single statement:

```
$friends = array('Bella', 'Diana', 'Rosa', 'Emily');
```

Or you can create an array with multiple statements:

```
$friends = array();
$friends[0] = 'Bella';
$friends[1] = 'Diana';
$efriend[2] = 'Rosa';
$friends[3] = 'Emily';
```

In both cases we use the **array** keyword to create an array. Let's find the length of an array using the **count( )** function.

```
echo count($friends)."<br>";
print_r($friends);
```

The count( ) function returns the number of elements in an array. The print_r( ) function is an easy way to see what elements you have in your array during the development process.

Adding and deleting elements

You can add an element to an array by specifying its index:

```
$friends[4] = 'Ashley';
```

Or by leaving the index field empty:

```
$friends[] = 'Jazmin';
```

If you omit the index inside the brackets PHP will add it to the end of the array. Specifying an index that is higher than the size of the array will add the element leaving gaps in between. If in our example we had coded `$friends[6] = 'Ashley'` that would have left a gap with two NULL values.

To delete an element from an array we use the **unset( )** function. Let's see how that works:

```
unset($friends[2]);
```

Using the unset( ) function will remove an element from an array, but it will not close the gap left. Instead, a NULL value will replace the removed element. If we now try to use `$friends[2]` we will get an error. In order to reindex the array we need to use the **array_values( )** function. Let's see how this works:

```
$friends = array_values($friends);
```

Now, any NULL values in the array are removed.

Associative Arrays

The main difference between 'regular' arrays and associative arrays is that rather than having a numerical index, associative arrays have named keys that you assign to them. Let's create one:

```
$ages = array("Bella"=>"22", "Diana"=>"26", "Rosa"=>"22", "Emily"=>"27");
```

As with indexed arrays, associative arrays can also be created in the following way:

```
$ages = array();
$ages["Ashley"] = "26";
$ages["Jazmin"] = "28";
```

Adding elements and deleting elements works the same way:

```
$ages["Ashley"] = "26";
$ages["Jazmin"] = "28";
```

Elements are added to the end of the array. Since associative arrays are not ordered by an index, deleting an element will leave no gap. In our associative array for example if we were to delete “Diana”, it wouldn’t be intuitive to look for that name after “Bella”. In an indexed array it would be natural to look for 3 after 2.

Looping through an indexed array

Let’s look at the following example and see how it works:

```
$neStates = array("Maine", "Vermont", "New Hampshire", "Massachusetts", "Rhode Island", "Conne

$size = count($neStates);

echo "<h4>New England States:</h4>";
echo "<ul>";
for ($i = 0; $i < $size; $i++){
    echo "<li>".$neStates[$i]."</li>";
}
echo "</ul>";
```

First, we initialized the array $neStates. Next, we found its size by using the count( ) function. Then we added a heading and started an unordered list. Now for the heart of the matter: in the body of the for loop, we go through each of the elements by accessing its index. We make sure that we don’t go outside the array by checking our counter against the size of the array.
You can also use a **foreach loop** to go through the array:

```
echo "<h4>New England States:</h4>";
echo "<ul>";
foreach($neStates as $value){
    echo "<li>$value</li>";
}
echo "</ul>";
```

Using our $neStates we can see that using the foreach loop is a little simpler. There is no need to find the size of the array or have to worry about initializing a counter and remembering to increase it. Inside the parentheses, we include the name of the array, the keyword **as** and a variable that will be used to loop through the array.

foreach loop with associative arrays

Associative arrays work in a similar manner with foreach loops:

```
$neStateCapitals = array("Maine"=>"Augusta", "Vermont"=>"Montpelier", "New Hampshire"=>"Concor

echo "<table>";
echo "<tr><th>State</th>";
echo "<th>Capital</th></tr>";
foreach($neStateCapitals as $key => $value){
    echo "<tr><td>$key</td>";
    echo "<td>$value</td></tr>";
}
echo "</table>";
```

The only difference is that inside the parentheses we can include two variables, one each for the key and the value. If only one variable is included then only the values will be displayed.

Debugging Code

Examine the following code. Find and fix the errors.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Arrays</title>
    <style>
        body{margin: 30px;}
        th, td {padding: 5px;}
    </style>
</head>
<body>

<?php
    $squares = array();

    echo "<table>";
    echo "<tr><th>Number</th>";
    echo "<th>Square</th>";
    for($i = 1; i <= 20; $i+) {
        $squares[$i] = $i*$i;
        echo "<tr><td> $i </td>";
        echo "<td> $square[$i] </td></tr>;
    }
    echo "</table>";
?>
</body>
</html>
```

Lab Exercises

1.
    - Create two arrays: one called $squares and one called $cubes.
    - Write a for loop that will initialize the two arrays with the squares and cubes of numbers from 1 to 20.
    - Write a foreach loop that will generate a table similar to:

Number	Square	Cube
1	1	1
2	4	8
3	9	27
…	…	…
19	361	6859

Number	Square	Cube
20	400	8000

2. The rand( ) function generates a random integer. When used with two arguments: rand($min, $max) it returns an integer between $min and $max inclusive.
    - Using a for loop use the rand( ) function to generate 500 integers between 1 and 10.
    - Create an array to keep track of how many times each integer is generated.
    - In a table, display the integer and the number of times each integer was generated. Your table should look similar to:

Number	Times Generated
1	46
2	51
3	55
4	52
5	39
6	60
7	43
8	54
9	48
10	52

3. Create a form with ten input fields: 5 for names and 5 for ages. Using appropriate labels and a submit button, store the information entered in an associative array and display back the data as an HTML table.