Chapter 6

Manipulating Arrays

PHP Programming with MySQL
2nd Edition

Lecture 1

Objectives

In this chapter, you will:

- Declare and initialize associative arrays
- Iterate through an array
- Find and extract elements and values
- Sort, combine, and compare arrays
- Understand multidimensional arrays
- Use arrays in Web forms

Adding and Removing Elements from the Beginning of an Array

- The `array_shift()` function removes the first element from the beginning of an array
  - Pass the name of the array whose first element you want to remove
- The `array_unshift()` function adds one or more elements to the beginning of an array
  - Pass the name of an array followed by comma-separated values for each element you want to add

Adding and Removing Elements from the Beginning of an Array (continued)

```
$TopSellers = array(
     "Chevrolet Impala",
     "Chevrolet Malibu",
     "Chevrolet Silverado",
     "Ford F-Series",
     "Toyota Camry",
     "Toyota Corolla",
     "Nissan Altima",
     "Honda Accord",
     "Honda Civic",
     "Dodge Ram");
array_shift($TopSellers);
array_unshift($TopSellers, "Honda CR-V");
echo "<pre>\n";
print_r($TopSellers);
echo "</pre>\n";
```

Adding and Removing Elements from the Beginning of an Array (continued)

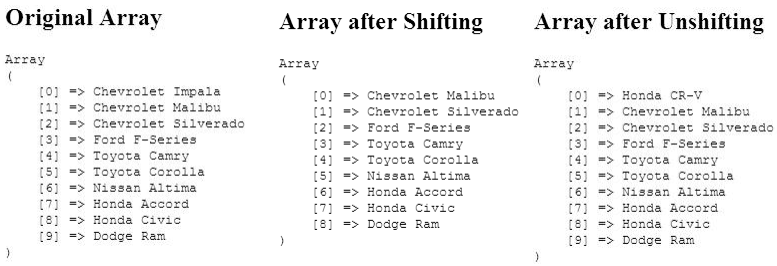

Figure 6-3 Output of an array modified with the `array_shift()` and `array_unshift()` functions

Adding and Removing Elements from the End of an Array

- The `array_pop()` function removes the last element from the end of an array
  - Pass the name of the array whose last element you want to remove
- The `array_push()` function adds one or more elements to the end of an array
  - Pass the name of an array followed by comma-separated values for each element you want to add

Adding and Removing Elements from the End of an Array (continued)

```
$HospitalDepts = array(
     "Anesthesia",
     "Molecular Biology",
     "Neurology",
     "Pediatrics");
array_pop($HospitalDepts); // Removes "Pediatrics"
array_push($HospitalDepts, "Psychiatry", "Pulmonary
   Diseases");
```


Adding and Removing Elements Within an Array

- The `array_splice()` function adds or removes array elements
- The `array_splice()` function renumbers the indexes in the array
- The syntax for the `array_splice()` function is:

```
array_splice(array_name, start,
  characters_to_delete, values_to_insert);
```

Adding and Removing Elements Within an Array (continued)

- To add an element within an array, include a value of 0 as the third argument of the `array_splice()` function

```
$HospitalDepts = array(
     "Anesthesia",          // first element (0)
     "Molecular Biology",   // second element (1)
     "Neurology",           // third element (2)
     "Pediatrics");         // fourth element (3)
array_splice($HospitalDepts, 3, 0, "Ophthalmology");
```


Adding and Removing Elements Within an Array (continued)

- To add more than one element within an array, pass the `array()` construct as the fourth argument of the `array_splice()` function
- Separate the new element values by commas

```
$HospitalDepts = array(
   "Anesthesia",          // first element (0)
   "Molecular Biology",   // second element (1)
   "Neurology",           // third element (2)
   "Pediatrics");         // fourth element (3)
array_splice($HospitalDepts, 3, 0, array("Opthalmology",
   "Otolaryngology"));
```

Adding and Removing Elements Within an Array (continued)

- Delete array elements by omitting the fourth argument from the `array_splice()` function

```
$HospitalDepts = array(
     "Anesthesia",          // first element (0)
     "Molecular Biology",   // second element (1)
     "Neurology",           // third element (2)
     "Pediatrics");         // fourth element (3)
array_splice($HospitalDepts, 1, 2);
```


Adding and Removing Elements Within an Array (continued)

- The `unset()` function removes array elements and other variables
- Pass to the `unset()` function the array name and index number of the element you want to remove
- To remove multiple elements, separate each index name and element number with commas

```
unset($HospitalDepts[1], $HospitalDepts[2]);
```

Removing Duplicate Elements

- The `array_unique()` function removes duplicate elements from an array
- Pass to the `array_unique()` function the name of the array from which you want to remove duplicate elements
- The `array_values()` and `array_unique()` functions do not operate directly on an array
- The `array_unique()` function does renumber the indexes after removing duplicate values in an array

Removing Duplicate Elements (continued)

```
$TopSellers = array(
     "Ford F-Series", "Chevrolet  Silverado", "Toyota Camry",
     "Honda Accord", "Toyota Corolla", "Ford F-Series", "Honda
  Civic",
     "Honda CR-V", "Honda Accord", "Nissan Altima", "Toyota
  Camry",
     "Chevrolet Impala", "Dodge Ram", "Honda CR-V");
echo "<p>The 2008 top selling vehicles are:</p><p>";
$TopSellers = array_unique($TopSellers);
$TopSellers = array_values($TopSellers);
for ($i=0; $i<count($ TopSellers); ++$i) {
     echo "{$TopSellers[$i]}<br />";
}
echo "</p>";
```

Removing Duplicate Elements (continued)

Figure 6-4 Output of an array after removing duplicate values with the `array_unique()` function

Declaring and Initializing Associative Arrays

- With associative arrays, you specify an element's key by using the array operator (=>)
  - The syntax for declaring and initializing an associative array is:

```
$array_name = array(key=>value, ...);
```

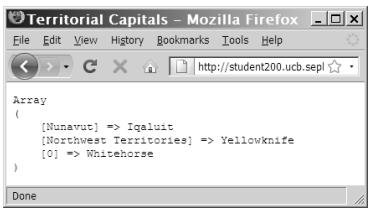

Figure 6-5 Output of array with associative and indexed elements

Declaring and Initializing Associative Arrays (continued)

```
$Territories[100] = "Nunavut";
$Territories[] = "Northwest Territories";
$Territories[] = "Yukon Territory";
echo "<pre>\n";
print_r($Territories);
echo "</pre>\n";
echo '<p>The $Territories array consists of ',
     count($Territories), " elements.</p>\n";
```

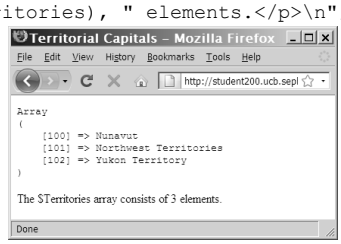

Figure 6-6 Output of an array with a starting index of 100

Finding and Extracting Elements and Values

- One of the most basic methods for finding a value in an array is to use a looping statement to iterate through the array until you find the value
- Rather than write custom code to find a value, use the `in_array()` and `array_search()` functions to determine whether a value exists in an array

Determining if a Value Exists

- The `in_array()` function returns a Boolean value of true if a given value exists in an array
- The `array_search()` function determines whether a given value exists in an array and:
  - Returns the index or key of the first matching element if the value exists, or
  - Returns `FALSE` if the value does not exist

```
if (in_array("Neurology", $HospitalDepts))
     echo "<p>The hospital has a Neurology department.</p>";
```


Determining if a Key Exists

- The `array_key_exists()` function determines whether a given index or key exists
- You pass two arguments to the `array_key_exists()` function:
  - The first argument represents the key to search for
  - The second argument represents the name of the array in which to search

Determining if a Key Exists (continued)

```
$ScreenNames["Dancer"] = "Daryl";
$ScreenNames["Fat Man"] = "Dennis";
$ScreenNames["Assassin"] = "Jennifer";
if (array_key_exists("Fat Man", $ScreenNames))
     echo "<p>{$ScreenNames['Fat Man']} is already
     'Fat Man'.</p>\n";
else {
     $ScreenNames["Fat Man"] = "Don";
     echo "<p>{$ScreenNames['Fat Man']} is now
     'Fat Man'.</p>";
}
```


Returning a Portion of an Array

- The `array_slice()` function returns a portion of an array and assigns it to another array
- The syntax for the `array_slice()` function is:

```
array_slice(array_name, start, characters_to_return);
```

Returning a Portion of an Array (continued)

```
// This array is ordered by sales, high to low.
$TopSellers = array("Ford F-Series", "Chevrolet Silverado",
   "Toyota Camry", "Honda Accord", "Toyota Corolla", "Honda
   Civic", "Nissan Altima", "Chevrolet Impala", "Dodge Ram",
   "Honda CR-V");
$FiveTopSellers = array_slice($TopSellers, 0, 5);
echo "<p>The five best-selling vehicles for 2008
   are:</p>\n";
for ($i=0; $i<count($FiveTopSellers); ++$i) {
     echo "{$FiveTopSellers[$i]}<br />\n";
}
```


Returning a Portion of an Array (continued)

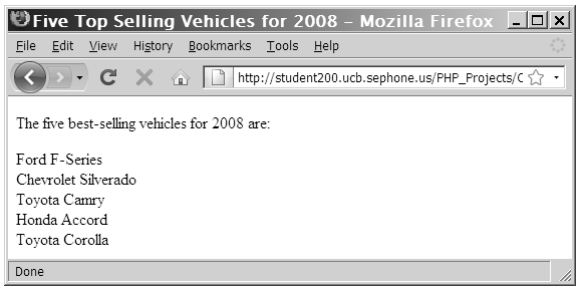

Figure 6-11 Output of an array returned with the `array_slice()` function

Sorting Arrays

- The most commonly used array sorting functions are:
  - `sort()` and `rsort()` for indexed arrays
  - `ksort()` and `krsort()` for associative arrays

Sorting Arrays (continued)

Function	Description
`array_multisort(array[, array, ...])`	Sorts multiple arrays or multidimensional arrays
`arsort(array[, SORT_REGULAR | SORT_NUMERIC | SORT_STRING])`	Sorts an array in descending order (largest to smallest) by value and maintains the existing keys for an associative array
`asort(array[, SORT_REGULAR | SORT_NUMERIC | SORT_STRING])`	Sorts an array in ascending order (smallest to largest) by value and maintains the existing keys for an associative array
`krsort(array[, SORT_REGULAR | SORT_NUMERIC | SORT_STRING])`	Sorts an array in descending order by key and maintains the existing keys for an associative array
`ksort(array[, SORT_REGULAR | SORT_NUMERIC | SORT_STRING])`	Sorts an array in ascending order by key and maintains the existing keys for an associative array
`natcasesort(array)`	Performs a case-sensitive natural order sort by value and maintains the existing keys for an associative array
`natsort(array)`	Performs a case-insensitive natural order sort by value and maintains the existing keys for an associative array

Table 6-2 Array sorting functions *(continues)*

Sorting Arrays (continued)

(continued)

Function	Description
`rsort(array[, SORT_REGULAR | SORT_NUMERIC | SORT_STRING])`	Sorts an array in descending order by value, removes any existing keys for an associative array, and renumbers the indexes starting with 0
`sort(array[, SORT_REGULAR | SORT_NUMERIC | SORT_STRING])`	Sorts an array in ascending order by value, removes any existing keys for an associative array, and renumbers the indexes starting with 0
`uaksort(array[, comparison_function])`	Sorts an array in ascending order by value using a comparison function and maintains the existing keys for an associative array
`uksort(array[, comparison_function])`	Sorts an array in ascending order by key using a comparison function and maintains the existing keys for an associative array
`usort(array[, comparison_function])`	Sorts an array in ascending order by value using a comparison function, removes any existing keys for an associative array, and renumbers the indexes starting with 0

Table 6-2 Array sorting functions

Sorting Arrays (continued)

- If the `sort()` and `rsort()` functions are used on an associative array, the keys are replaced with indexes

Sorting Arrays (continued)

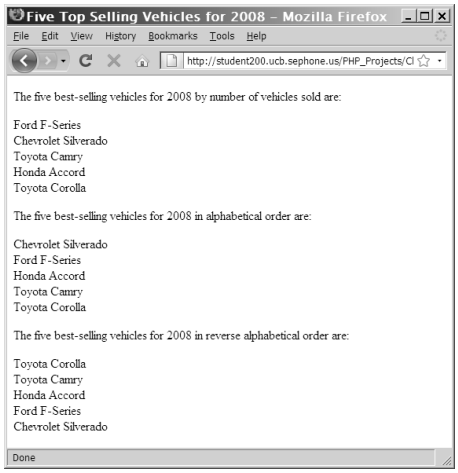

Figure 6-12 Output of an array after applying the `sort()` and `rsort()` functions

Sorting Arrays (continued)

Figure 6-13 Output of an associative array after sorting with the `sort()` function

Sorting Arrays (continued)

Figure 6-14 Output of an associative array after sorting with the `asort()` function

Sorting Arrays (continued)

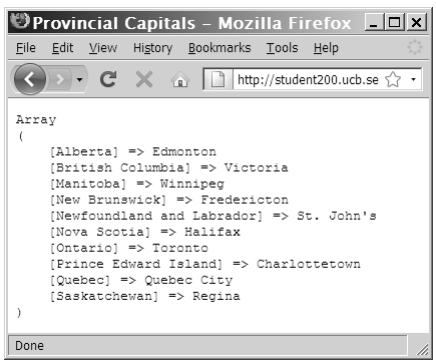

Figure 6-15 Output of an associative array after sorting with the `ksort()` function

Combining Arrays

- To append one array to another, use the addition (+) or the compound assignment operator (+=)
- To merge two or more arrays use the `array_merge()` function
- The syntax for the `array_merge()` function is:

```
new_array = array_merge($array1, $array2,
$array3, ...);
```


Combining Arrays (continued)

```
$Provinces = array("Newfoundland and Labrador",
  "Prince Edward Island", "Nova Scotia", "New
  Brunswick", "Quebec", "Ontario", "Manitoba",
  "Saskatchewan", "Alberta", "British
  Columbia");
$Territories = array("Nunavut", "Northwest
  Territories", "Yukon Territory");
$Canada = $Provinces + $Territories;
echo "<pre>\n";
print_r($Canada);
echo "</pre>\n";
```

Comparing Arrays

- The `array_diff()` function returns an array of elements that exist in one array but not in any other arrays to which it is compared
- The syntax for the `array_diff()` function is:

```
new_array = array_diff($array1, $array2,
$array3, ...);
```

- The `array_intersect()` function returns an array of elements that exist in all of the arrays that are compared

Comparing Arrays (continued)

```
$ProvincialCapitals = array("Newfoundland and Labrador"=>"St.
   John's", "Prince Edward Island"=>"Charlottetown", "Nova
   Scotia"=>"Halifax", "New Brunswick"=>"Fredericton",
   "Quebec"=>"Quebec City", "Ontario"=>"Toronto",
   "Manitoba"=>"Winnipeg", "Saskatchewan"=>"Regina",
   "Alberta"=>"Edmonton", "British Columbia"=>"Victoria");
$TerritorialCapitals = array("Nunavut"=>"Iqaluit", "Northwest
   Territories"=>"Yellowknife", "Yukon Territory"=>"Whitehorse");
$CanadianCapitals = $ProvincialCapitals + $TerritorialCapitals;
echo "<pre>\n";
print_r($CanadianCapitals);
echo "</pre>\n";
```

Comparing Arrays (continued)

- The syntax for the `array_intersect()` function is:

```
new_array = array_intersect($array1,
$array2, $array3, ...);
```


Comparing Arrays (continued)

```
$Provinces = array("Newfoundland and Labrador",
  "Prince Edward Island", "Nova Scotia", "New
  Brunswick", "Quebec", "Ontario", "Manitoba",
  "Saskatchewan", "Alberta", "British
  Columbia");
$Territories = array("Nunavut", "Northwest
  Territories", "Yukon Territory");
$Canada = array_merge($Provinces,
  $Territories);
```