SERVER-SIDE WEB PROGRAMMING UNIT2: PROGRAMMING BASED ON EMBEDDED LANGUAGE

CFGS - DAW 2

JESUITAS
Sagrado Corazón
Jesuitas · Logroño

Index

4. Arrays

- An array is a collection of data values organized as an ordered collection of key-value pairs:

Key	Value
corn	yellow
beet	red
carrot	orange
pepper	green
broccoli	green

4.1. Indexed Versus Associative Arrays

- There are two kinds of arrays in PHP: ***indexed and associative.***
  - *Indexed*: The keys are integers, beginning at 0. Indexed arrays are used when you identify things by their position.
  - *Associative arrays*: Thet have strings as keys and behave more like two-column tables. The first column is the key, which is used to access the value. (Hashmap!)

4.2. Identifying Elements of an Array

- Before we look at creating an array, let's look at the structure of an existing array.
- You can access specific values from an existing array using the array variable's name, followed by the element's key, or index, within square brackets:

```
$age['fred'];
$shows[2];
$shows['2'];
```

4.3. Storing Data


```php
$addresses[0] = "spam@cyberpromo.net";
$addresses[1] = "abuse@example.com";
$addresses[2] = "root@example.com";

$price['gasket'] = 15.29;
$price['wheel'] = 75.25;
$price['tire'] = 50.00;
```

```php
$addresses = array("spam@cyberpromo.net", "abuse@example.com", "root@example.com");
$price = array(
    'gasket' => 15.29,
    'wheel' => 75.25,
    'tire' => 50.00
);
$days = ['gasket' => 15.29, 'wheel' => 75.25, 'tire' => 50.0];
```

4.3. Storing Data

- To construct an empty array, pass no arguments to array():

```
$addresses = array();
```

- You can specify an initial key with => and then a list of values:

```
$days = array(1 => "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun");
// 2 is Tue, 3 is Wed, etc.
$whoops = array('Fri' => "Black", "Brown", "Green");
// same as
$whoops = array('Fri' => "Black", 0 => "Brown", 1 => "Green");
```

4.3. Storing Data

- Adding Values to the End of an Array:

```
$family = array("Fred", "Wilma");
$family[] = "Pebbles"; // $family[2] is "Pebbles"

$person = array('name' => "Fred");
$person[] = "Wilma"; // $person[0] is now "Wilma"
```

- Assigning a Range of Values:
  - The range() function creates an array of consecutive integer or character values between and including the two values you pass to it as arguments.

```
$numbers = range(2, 5); // $numbers = array(2, 3, 4, 5);
$letters = range('a', 'z'); // $letters holds the alphabet
$reversedNumbers = range(5, 2); // $reversedNumbers = array(5, 4, 3, 2);
```

4.3. Storing Data

- <u>Getting the Size of an Array:</u>
  - The *count()* and *sizeof()* functions are identical in use and effect:

```
$family = array("Fred", "Wilma", "Pebbles");
$size = count($family); // $size is 3
```

- <u>Padding an Array:</u>

```
$scores = array(5, 10);
$padded = array_pad($scores, 5, 0); // $padded is now array(5, 10, 0, 0, 0)
$padded2 = array_pad($scores, -5, 0); // $padded2 is now array(0, 0, 0, 5, 10);
```

4.3. Storing Data


```
<html>
    <head>
        <title>
        </title>
    </head>
    <body>
        <h1>First exercise with arrays: <br/></h1>
        <?php
            $a=$arrayName = array(8,9);
            foreach ($a as $value) {
                echo ($value."<br>");
            }
        ?>
    </body>
</html>
```

4.4. Multidimensional Arrays

- The values in an array can themselves be arrays:

```
$row0 = array(1, 2, 3);
$row1 = array(4, 5, 6);
$row2 = array(7, 8, 9);
$multi = array($row0, $row1, $row2);
$value = $multi[2][0]; // row 2, column 0. $value = 7
```

- <u>To interpolate a lookup of a multidimensional array, you must *enclose the entire array lookup in curly braces{} (also for a position on one dimesion arrays)*</u>:

```
echo("The value at row 2, column 0 is {$multi[2][0]}\n");
```

4.5. Extracting Multiple Values

- To copy all of an array's values into variables, use the list() construct:

```
$person = array("Fred", 35, "Betty");
list($name, $age, $wife) = $person;
// $name is "Fred", $age is 35, $wife is "Betty
```

- *The use of the list function is a common practice for picking up values from a database* selection where only one row is returned. This would automatically load the data from the simple query into a series of local variables.

4.5. Extracting Multiple Values

- To extract only a subset of the array, use the *array_slice()* function:

```
$people = array("Tom", "Dick", "Harriet", "Brenda", "Jo");
$middle = array_slice($people, 2, 2); // $middle is array("Harriet", "Brenda")
```

A.2.18. What sentence (just one) would you code in order to extract only "Dick" and "Harriet" to some variables called $second and $third?

4.5. Extracting Multiple Values

- <u>Keys and Values:</u>
  - The *array_keys()* function returns an array consisting of only the keys in the array in internal order:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$keys = array_keys($person); // $keys is array("name", "age", "wife")
```

  - PHP also provides a function to retrieve an array of just the values in an array, *array_values()*:

```
$values = array_values($person); // $values is array("Fred", 35, "Wilma");
```

4.5. Extracting Multiple Values

- <u>Checking Whether an Element Exists:</u>
  - To see if an element exists in the array, use the *array_key_exists()* function:

```
$person['age'] = 0;
if (array_key_exists('age', $person)) {
    echo "exists!\n";
}
```

4.5. Extracting Multiple Values

- <u>Removing and Inserting Elements in an Array</u>:
  - The *array_splice()* function can remove or insert elements in an array and optionally create another array from the removed elements:

```
$subjects = array("physics", "chem", "math", "bio", "cs", "drama", "classics");
$removed = array_splice($subjects, 2, 3);//You can assign the result or not
// $removed is array("math", "bio", "cs")
// $subjects is array("physics", "chem", "drama", "classics")
print_r($removed);
print_r($subjects);
$removed = array_splice($subjects, 2);
print_r($removed); //?
print_r($subjects);//?
```

4.5. Extracting Multiple Values

- To insert elements where others were removed, use the fourth argument:

```
$subjects = array("physics", "chem", "math", "bio", "cs", "drama", "classics");
$new = array("law", "business", "IS");
array_splice($subjects, 4, 3, $new);
// $subjects is array("physics", "chem", "math", "bio", "law", "business", "IS")
```

- To insert new elements into the array while pushing existing elements to the right, delete zero elements:

```
$subjects = array("physics", "chem", "math");
$new = array("law", "business");
array_splice($subjects, 2, 0, $new);
// $subjects is array("physics", "chem", "law", "business", "math")
```

4.6. Converting Between Arrays and Variables

- <u>Creating Variables from an Array</u>:
  - The *extract()* function automatically creates local variables from an array.
  - The indexes of the array elements become the variable names

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Betty");
extract($person);
echo "Name: {$name}, Age: {$age}, Wife: {$wife}";
```

4.6. Converting Between Arrays and Variables

- Creating an Array from Variables:
  - The *compact*() function is the reverse of *extract*().

```
$color = "indigo";
$shape = "curvy";
$floppy = "none";
$a = compact("color", "shape", "floppy");
```

4.7. Looping Through Arrays

1. <u>The foreach Construct:</u>

- The most common way to loop over elements of an array is to use the *foreach* construct.

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");
foreach ($addresses as $value) {
    echo "Processing {$value}\n";
}
```

- In order to access to the current key:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
foreach ($person as $key => $value) {
    echo "Fred's {$key} is {$value}\n";
}
```

4.7. Looping Through Arrays

A.2.19. Declare an array called *$meal* with these key-values: breakfast→coffee, snack→sandwich, lunch→pizza, dinner→omellete. After that create a new method called *print_table*, and making use of a foreach loop print the passed array into a pretty table.

4.7. Looping Through Arrays

2. The For Loop:
   - If you want to know what element you're on as you're iterating through a numeric array, use for() instead of foreach().

```
$dinner = array('Sweet Corn and Asparagus',
    'Lemon Chicken',
    'Braised Bamboo Fungus');
for ($i = 0, $num_dishes = count($dinner); $i < $num_dishes; $i++) {
    print "Dish number $i is $dinner[$i]\n";
}
```

4.7. Looping Through Arrays

- <u>Searching for Values:</u>
  - *in_array():*

```
$meals = array('Walnut Bun' => 1,
    'Cashew Nuts and White Mushrooms' => 4.95,
    'Dried Mulberries' => 3.00,
    'Eggplant with Chili Sauce' => 6.50,
    'Shrimp Puffs' => 0);
$books = array("The Eater's Guide to Chinese Characters",
    'How to Cook and Eat in Chinese');

if (in_array(3, $meals)) {
    print 'There is a $3 item.';
}

if (in_array('How to Cook and Eat in Chinese', $books)) {
    print "We have How to Cook and Eat in Chinese";
}

if (in_array("the eater's guide to chinese characters", $books)) {
    print "We have the Eater's Guide to Chinese Characters.";
}
```

A.2.20. *array_search()* is similar to *in_array()*, but what is the difference?

4.8. Sorting

- The functions provided by PHP to sort an array are:

Effect	Ascending	Descending	User-defined order
Sort array by values, then reassign indices starting with 0	sort()	rsort()	usort()
Sort array by values	asort()	arsort()	uasort()
Sort array by keys	ksort()	krsort()	uksort()

```
$names = array("Cath", "Angela", "Brad", "Mira");
sort($names); // $names is now "Angela", "Brad", "Cath", "Mira"
```

4.8. Sorting


```php
$logins = array(
    'njt' => 415,
    'kt' => 492,
    'rl' => 652,
    'jht' => 441,
    'jj' => 441,
    'wt' => 402,
    'hut' => 309,
);
arsort($logins);
$numPrinted = 0;
echo "<table>\n";
foreach ($logins as $user => $time) {
    echo("<tr><td>{$user}</td><td>{$time}</td></tr>");
    if (++$numPrinted == 3) {
        break; // stop after three
    }
}
echo "</table>";
```

4.8. Sorting

- <u>Randomizing Order:</u>
  - To traverse the elements in an array in random order, use the *shuffle()* function.
  - It replaces all existing keys (string or numeric) with consecutive integers starting at 0.

```
$weekdays = array("Monday", "Tuesday", "Wednesday", "Thursday", "Friday");
shuffle($weekdays);
print_r($weekdays);
```

4.9. Acting on Entire Arrays

- PHP has several useful functions for modifying or applying an operation to all elements of an array:

  $sum = **array_sum**(array);

  $merged = **array_merge**(*array1*, *array2* [, *array* ...]);

  $diff = **array_diff**(*array1*, *array2* [, *array* ...]);

4.9. Acting on Entire Arrays

A.2.21. What do you think this code is trying to do?

```
function arrayUnion($a, $b)
{
    $union = array_merge($a, $b); // duplicates may still exist
    $union = array_unique($union);
    return $union;
}
$first = array(1, "two", 3);
$second = array("two", "three", "four");
$union = arrayUnion($first, $second);
print_r($union);
```