Programming via PHP- Arrays

13.1. Basics

An **array** in PHP is a powerful structure for a script to remember a conglomeration of data. You should think of an as a collection of associations between *keys* and their corresponding *values*. For example: I might want to have an array that associates people's names with their addresses. We might choose the address to be the key, and the name to be the value associated with each key. Thus, associated with the key 221B I would find the value Sherlock Holmes. (If you happen to have seen the*array* notion in a different programming language, forget what you learned: PHP's notion of array is rather unusual.)

An array is referenced using a single variable, such as `$residents`. To retrieve the value associated with a key, place the value into brackets following the variable name. Thus, given `echo $residents["221B"];` PHP will echo Sherlock Holmes.

We place a value into the array using this same bracket notation, but now on the left sign of an assignment statement. Thus, it would be legal to write`$residents["220A"] = "Watson";`. If `$residents` wasn't yet referring to anything, this would create an array. If the key `220A` didn't already exist in the array, the key would be inserted with Watson as the associated value. And if there were already a value associated with `220A`, that person would be evicted in favor of Watson.

In fact, PHP automatically constructs some built-in variables referring to arrays. The `$_POST` variable is one major example: When a browser sends a POST request to a PHP script, PHP sets `$_POST` to be an array whose keys correspond to the names of the input controls, and whose values are the values sent by the browser for each input control. In fact, we've been avoiding `$_POST` by using the `import_request_variables` function; this function conveniently creates variables corresponding to each array entry.

In our above example, the keys and values are both strings. But the keys and values can be of any type, and indeed they can mix types. In an example below using the`explode` function, we'll see an array where it happens that the keys are all integers, and the values are all strings.

13.2. Example

Arrays show up in several of PHP's library functions. One such is `explode`, which splits a string into several pieces. It takes two parameters: The first is a string describing what separates the pieces into which it is to be divided, and the second is the string that should be divided. The function returns an array, with the keys being the integers 0, 1, 2,..., with the first piece associated with 0, the second piece associated with 1, and so on.

For example, suppose we have a Web form where the user types a telephone number in the form 501-340-1300 and for some reason we want to extract the area code and exchange from the telephone number. The following PHP code accomplishes this.

```
$phone_parts = explode("-", $form_phone);
$area_code = $phone_parts[0];
$exchange = $phone_parts[1];
```

13.3. Array presence

Sometimes we'll want to determine whether an array has any value associated with a particular key. We can do this using **`isset`**. For example, if I have a PHP script that is supposed to be invoked from a form with a field named *`userid`*, and I want to verify that the PHP script was indeed sent a value for *`userid`*, I can write the following.

```
if(isset($_POST["userid"])) {
    echo "<html><head><title>Error</title></head><body>\n";
    echo "<h1>Error</h1><p>Sorry, you must enter your user ID.</p>\n";
    echo "</body></html>\n";
    exit;
}
```

(The above example also uses the PHP function **`exit`**, which terminates the execution of the current script. In this case, we wouldn't want it to continue because we have already sent the HTML response reporting the problem with the POST request.)

If for some reason you want to delete a key and its associated value from an array, you can use **`unset`**, as in **`unset`**`($residents["221B"]);`.

Both **isset** and **unset** can also be applied in contexts that have nothing to do with arrays. For example, if we were still using the `import_request_variables` function, then we could also have written **if**(**isset**(`$form_userid`)) {... to see whether the browser has sent us any information from of the form's `userid` blank.

(Though they look like functions, technically **isset** and **unset** are part of the base PHP language rather than part of its function library. This is a technical detail, though. You are free to think of them as functions, if you like.)

Source: http://www.toves.org/books/php/ch13-arrays/index.html