WEB 306 – PHP: Databases and Framework

Day 3 – Superglobals

1. Templates and Includes

1.1 Requiring and Including Files

PHP allows us to store PHP code in separate files which are imported into our main files. These files are called templates or includes depending on their purpose. Template and include files are imported using one of 4 keywords:

1. `require`
2. `require_once`
3. `include`
4. `include_once`

The difference between `require` and `include` is that files which are required will stop the page from functioning if they can not be found while files which are included just don't get included if they can not be found. You would just use `require_once` and `include_once` to prevent a file from being required or included more than once.

PHP Resources

1. PHP Website
2. XAMPP
3. Documentation
4. W3Schools
5. CSS Tricks
6. Scotch.io
7. Stack Overflow

1.2 Template Files

Since it was originally designed as a templating language we can use PHP for HTML templating. PHP template files are frequently saved in a `/templates` folder, generally include ***mostly HTML*** code and are sometimes named using the conventions `filename.tpl.php` or `filename.html.php` to indicate that they are template files that include primarily HTML.

To create a template file cut the code for a piece of your template from your HTML and save it in a new PHP file. Then import it using one of the 4 keywords. For headers and footers I would normally use `require_once`.

```
<?php require_once 'templates/header.tpl.php'; ?>
<div class="content">
  <?php include_once 'templates/superglobals.tpl.php'; ?>
</div>
<?php require_once 'templates/footer.tpl.php'; ?>
```

1.3 Include Files

PHP files saved in the /includes folder generally include ONLY PHP code and are sometimes named **`filename.inc.php`** to indicate that they are an includes file. Typically these files are used for form and data processing. It is also common to have one include file which contains all of your custom function definitions.

When you are using an include file, you would have an opening PHP tag at the beginning of the file but because you are only writing PHP code, you do not need to use a closing PHP tag. It is actually considered the best practice to exclude the closing PHP tag because it can prevent redirects to other pages from working.

```
<?php
// You do not need a closing PHP tag if you are using an include file
```

To import an include file use one of the 4 keywords. For function includes I would normally use **`include`** because it wouldn't usually be necessary to shut down the entire page if our functions couldn't be found.

```
<?php include 'includes/functions.inc.php'; ?>
```

2. Variable Scope

Variable scope determines which parts of your code your variables can be accessed from. If you try to use a variable which is not available within the current scope of a block of code, it will result in an error. Depending on where variables are defined, they are given different scopes. There are two types of variable scopes in PHP: ***global*** and ***local***.

2.1 Global Scope

Variables which are defined ***outside*** of a function have a ***global scope*** and by default, can only be accessed outside of a function. Variables with a global scope are accessible accross required and included files as well.

```
<?php
// Variables defined outside functions are global
$special_chars = '/([<>!@#$%^&])/';
```

In order for global variables to be accessed inside of a function, they must either be prefaced with the **`global`** keyword (see ***2.3 The global Keyword***) or be accessed through the **`$GLOBALS`** associative array superglobal variable (see ***3.1 $GLOBALS***).

> **"Global variables, as the name implies, are variables that are accessible globally, or everywhere throughout the program. Once declared, they remain in memory throughout the runtime of the program. This means that they can be changed by any function at any point and may affect the program as a whole."**
>
> **— From What is a Global Variable?, Techopedia**

2.2 Local Scope

Variables which are defined ***inside*** of a function have a ***local scope*** and can only be accessed inside of that function. Parameters of functions are also considered local variables.

```
function sanitize_string($string, $special_chars = null) {
   // Variables defined inside functions are local
   $strip_tags = strip_tags($string);
   $remove_chars = preg_replace($special_chars, '', $strip_tags);
   $sanitized = htmlentities($remove_chars, ENT_QUOTES, 'UTF-8');

   return $sanitized;
}
```

2.3 The global Keyword

The `global` keyword is used to allow functions to access global variables from within the function. To use it, simply preface the global variable you wish to use in the function with the keyword `global`.

```
// Variables defined outside functions are global
$special_chars = '/([<>!@#$%^&])/';

function format_name($name) {
   global $special_chars; // Import global variable to function

   // global variable is used in the function
   $sanitized = sanitize_string($name, $special_chars);
   $lowercase = strtolower($sanitized);
   $capitalized = ucwords($lowercase);

   return $capitalized;
}
```

If you are familiar with JavaScript, `global` is the equivalent of `var` in JavaScript but is used more sparingly. Global variables can also be accessed in functions through the `$GLOBALS` associative array superglobal variable (see ***3.1 $GLOBALS***).

2.4 The static Keyword

Normally after a function is executed its local variables are deleted from the program's memory. The `static` keyword is used when you want the value of a variable to be stored in the program's memory the next time the function is used. To use it, simply preface the local variable you want to make static with the keyword `static`.

```
function counter() {
   static $x = 0;
   echo $x;
   $x++;
}
```

3. Superglobals

Superglobals are ***predefined global variables*** in PHP which can be accessed from anywhere in your code, hence why they are called ***"superglobals"***. They are ***associative arrays*** which hold large amounts of data which is frequently added to them automatically based on user interactions and input. Superglobals are written in ALL CAPS to diffrentiate them from other variables. There are ***9 Superglobal variables***, each of which serves a unique purpose.

3.1 $GLOBALS

$GLOBALS is used to access an array of every global variable in a PHP script. This is useful when you need to access a global variable inside of a function.

```
$title = 'PHP Superglobals';

function show_title() {
   echo '<h2>' . $GLOBALS['title'] . '</h2>';
   // This is the same as $title but as an associative array item
}

show_title();
```

Global variables can also be accessed in functions by being prefaced with the **global** keyword (see ***2.3 The global Keyword***).

3.2 $_SERVER

$_SERVER is a variable for an array of server data such as information about paths and script locations.

3.2.1 PHP_SELF

PHP_SELF would be used to get the path to the current PHP file.

```
echo '<p>' . $_SERVER['PHP_SELF'] . '</p>';
```

Using the **basename()** function in combination with PHP_SELF gets just the name of the PHP file.

```
echo '<p>' . basename($_SERVER['PHP_SELF']) . '</p>';
```

Use **htmlspecialchars()** to sanitize data. Never use **PHP_SELF** in a form action without sanitizing it to prevent security risks.

```
$this_file = basename(htmlspecialchars($_SERVER['PHP_SELF'], ENT_
QUOTES, 'UTF-8'));
```

3.2.2 SERVER_ADDR

SERVER_ADDR gets the IP address of the server under which the current script is executing.

```
echo '<p>' . $_SERVER['SERVER_ADDR'] . '</p>';
```

3.2.3 SERVER_NAME

`SERVER_NAME` gets the name of the server host under which the current script is executing.

```
echo '<p>' . $_SERVER['SERVER_NAME'] . '</p>';
```

3.2.4 SERVER_SOFTWARE

`SERVER_SOFTWARE` gets the server identification string, given in the headers when responding to requests.

```
echo '<p>' . $_SERVER['SERVER_SOFTWARE'] . '</p>';
```

3.3 $_GET

3.3.1 Sending $_GET Data With Links

Data can be sent to the URL as key/value pairs in the format:

```
http://website.com?key=value&next_key=value
```

We can send this data to the URL by simply linking to URLs with this format.

```
<a href="<?php echo $this_file; ?>?friend=birds">Birds</a>
<a href="<?php echo $this_file; ?>?friend=dog">Dog</a>
```

Any key/value pairs in the URL are stored as an array in the `$_GET` superglobal variable. Therefore, `$_GET` can be used to collect data from the URL.

```
// Run this code if friend is not empty
if (!empty($_GET['friend'])) {
   // GET the data from URL
   $friend = $_GET['friend'];
   // Set the image name to the value of friend
   echo '<p><img src="img/' . $friend . '.jpg" alt="Elephant and ' .
$friend . '" class="elephant"></p>';
} else {
   echo '<p><img src="img/beach.jpg" alt="Elephant"
class="elephant"></p>';
}
```

3.3.2 Sending $_GET Data With Forms

An HTML `<form>` tag with a `method="get"` attribute sends the form data to the URL as key/value pairs. We can then use `$_GET` to collect this data from the URL. The action attribute of the `<form>` tag points to a file which will process the data which is submitted from the form. The PHP script for this form is in this PHP file so we will set the action to `$this_file`.

```
<form action="<?php echo $this_file; ?>" method="get">
   <label for="elephant_name">Elephant Name:</label>
   <input type="text" name="elephant_name" id="elephant_name">
   <button type="submit" name="button">Send to URL and $_GET</button>
</form>
```

The `name` attribute is very important. It is what is used as the key in the key/value data which is submitted from the form. Finally let's get our form data, sanitize and format it, and display it.

```
// Run this code if elephant_name is not empty
if (!empty($_GET['elephant_name'])) {
   // GET the user input from URL
   $elephant_name = $_GET['elephant_name'];
   // Sanitize and format input
   $elephant_name_formatted = format_name($elephant_name);
   // Display input
   echo '<h3>The elephant is ' . $elephant_name_formatted . '</h3>';
} else {
   echo '<h3>You did not provide an elephant name!</h3>';
}
```

Because data from `$_GET` forms is displayed in the URL, they are completely useless for transmitting any kind of sensitive data like usernames or passwords. That's what `$_POST` is for.

3.4 $_POST

`$_POST` is used to collect data after submitting an HTML form with a `method="post"` attribute.

```
<form action="<?php echo $this_file; ?>" method="post">
   <label for="friend_name">Friend Name:</label>
   <input type="text" name="friend_name" id="friend_name">
   <button type="submit" name="button">Send to $_POST</button>
</form>
```

The data is then saved in the `$_POST` associative array and ***nowhere*** else... ***except*** for in the `$_REQUEST` associative array which we'll talk about later. When collecting sensitive data you should ALWAYS use the `$_POST` variable instead of `$_GET`.

```
// Run this code if friend_name is not empty
if (!empty($_POST['friend_name'])) {
   // Retreive the user input from the $_POST array
   $friend_name = $_POST['friend_name'];
   // Sanitize and format input
   $friend_name_formatted = format_name($friend_name);
   // Display input
   echo '<h3>The friend name is ' . $friend_name_formatted . '</h3>';
} else {
   echo '<h3>You did not provide a friend name!</h3>';
}
```

3.5 $_COOKIE

Cookies are used to store data about users in the browser. To set a cookie in PHP, use the `setcookie()` function. The `setcookie()` function sets a browser cookie. It ***MUST*** be used before the opening `<html>` tag.

```
<?php
```

```
$cookie_name = 'chocolate_chip';
$cookie_value = 'img/chocolate_chip.png';

if (isset($_GET['cookie']) && $_GET['cookie'] == 'set') {
   setcookie($cookie_name, $cookie_value);
} else {
   // Set the expiration date to one hour ago
   setcookie($cookie_name, '', time() - 3600);
}
?>
<!DOCTYPE html>
<html lang="en">
```

Once browser cookies have been set they are stored in the `$_COOKIE` associative array. We can then access cookie data using the `$_COOKIE` variable. We can check how many cookies our browser has.

```
echo '<h3>There are ' . count($_COOKIE) . ' cookies left</h3>';
```

We can use `$_GET` to set to create links which will set and clear our cookie.

```
<a href="<?php echo $this_file; ?>?cookie=set">Set</a>
<a href="<?php echo $this_file; ?>?cookie=clear">Clear</a>
```

We can check if the cookie is set and display data based on what the value of the cookie is.

```
if (isset($_COOKIE['chocolate_chip'])) {
   echo '<img src="' . $_COOKIE['chocolate_chip'] . '" alt="Cookie">';
}
```

3.6 $_FILES

When a user uploads files to a form they are stored in the `$_FILES` associative array. We can then access them using the `$_FILES` superglobal variable.

3.6.1 File Uploads

To allow users to upload files, you ***MUST*** use a `method="post"` form. You must also include an attribute of `enctype="multipart/form-data"` for added encryption. We will point this action to a separate PHP include file due to the complexity of the form processing. Set the input type to "file" and make sure to set the `name` attribute

```
<form action="includes/process-upload.inc.php" method="post"
enctype="multipart/form-data">
   <label for="elephant_img">Elephant Image:</label>
   <input type="file" name="elephant_img" id="elephant_img">
   <button type="submit" name="button">Upload Image</button>
</form>
```

3.6.2 Processing File Uploads

Specify the directory where the file is going to be placed.

```
$target_dir = '../uploads/';
```

Specify the path of the file to be uploaded.

```
$target_file = $target_dir . basename($_FILES['elephant_img']['name']);
```

Check if the image file is an actual image or fake image. If the file is not a fake image try to move the uploaded file to specified directory and redirect back to the previous page using the `header()` function.

```
$check = getimagesize($_FILES['elephant_img']['tmp_name']);

if ($check !== false) {
   if (move_uploaded_file($_FILES['elephant_img']['tmp_name'], $target_
file)) {
        $file_name = basename($_FILES['elephant_img']['name']);
        header('Location: ../index.php?upload=success&file=' . $file_
name);
        exit();
   } else {
        header('Location: ../index.php?upload=error');
        exit();
   }
} else {
   header('Location: ../index.php?upload=not_image');
   exit();
}
```

3.6.3 The header() Function

The `header()` function is used to redirect the user to a page and add in `$_GET` data.

3.6.4 The exit() Function

The `exit()` function kills a script. It is used to make sure that after `header()` redirects, none of the code which comes after it tries to run and interrupts the redirect.

3.7 $_REQUEST

The `$_REQUEST` variable just contains all the data from `$_GET`, `$_POST` and `$_COOKIE`. This means that we can access data which has been sent to any of those associative arrays from `$_REQUEST` as well. Let's use `$_REQUEST` to send a success message and display our image if it uploads successfully and an error message if it does not. Check if `$_GET['upload']` is set and if it is equal to 'success'.

```
<?php
if (isset($_REQUEST['upload']) && $_REQUEST['upload'] == 'success') {
?>
<p>The file <?php echo $_REQUEST['file']; ?> has been uploaded.</p>
<p><img src="uploads/<?php echo $_REQUEST['file']; ?>"
alt="Elephant"></p>
<?php
```

```
} else if (isset($_REQUEST['upload']) && $_REQUEST['upload'] ==
'error') {
?>
<p class="error">Sorry, there was an error uploading your file.</p>
<?php
} else if (isset($_REQUEST['upload']) && $_REQUEST['upload'] == 'not_
image') {
?>
<p class="error">File is not an image.</p>
<?php
}
?>
```

3.8 $_SESSION

A session is a way to store information to be used across multiple pages. Unlike a cookie, the information is not stored on the users computer.

3.8.1 Starting Sessions

A session is started with the `session_start()` function. It would normally be used at the top of the page, before the opening `<html>` tag.

```
<?php
session_start();
?>
<!DOCTYPE html>
<html lang="en">
```

Once the session has started we can use the `$_SESSION` associative array to store session variables.

3.8.2 Setting and Accessing Session Data

```
if (!empty($_GET['elephant_name'])) {
  $elephant_name = $_GET['elephant_name'];
  $_SESSION['elephant_name'] = format_name($elephant_name);
}
```

Then on the next page we can use the `$_SESSION` associative array to display our session variables.

```
if (!empty($_SESSION['elephant_name'])) {
  echo '<h3>' . $_SESSION['elephant_name'] . '</h3>';
}
```

3.8.3 Unsetting and Destroying Sessions

To remove all session variables use the `session_unset()` function.

```
session_unset();
```

To destroy the session use the `session_destroy()` function.

```
session_destroy();
```

3.9 $_ENV

`$_ENV` is used to store environment variables. Environment variables normally include sensitive data like database credentials and API keys. You can set `$_ENV` data like this:

```
$_ENV['host'] = 'localhost';
$_ENV['user'] = 'root';
$_ENV['password'] = 'secret123';
$_ENV['database'] = 'elephants';
```