

WORKSHOP – LAMP SESSION 02

By Eddo Rotman



Now where were we?

- We separated client side from server side
- We were talking about the flow of a web application, using a request-response mechanism
- For the client side we wrote in HTML, JavaScript and CSS
- On the server side we were talking about PHP
- We talked about PHP as a scripting language and about its syntax
- PHP is loosely typed

So, what's the plan for today?

- More PHP Syntax
 - Strings
 - Arrays
- Passing data
- Saving data between requests
- Classes & Objects

Parsing Strings

- How to handle variables in strings?
- Strings in double quotes will be parsed, strings in single quotes won't

```
$name1 = 'fred';
$name2 = "wilma";
$name3 = "barney";
$name4 = "betty";

echo 'Morning ' . $name1;
echo "Good morning $name4";
echo "Is {$name3} home?";
echo "Am I responsible for all the {$name3}s in the world???" ;
echo 'I\'ll send ' . $name2 . ' over';
echo "[ $name2 yelling] {$name1}! Tell {$name4} I want my \"Harry
Potter\" book back!";
```

Arrays

```
$foo = array(1, 2, 3);

$bar[] = 1;
$bar[] = 2;
$bar[] = 3;

$baz = array(0=>1, 1=>2, 2=>3);

$moe = array(2=>3, 1=>2, 0=>1);

print ($bar == $foo) ? 'Yes' : 'No';           // Yes
print ($bar == $baz) ? 'Yes' : 'No';         // Yes
print ($bar == $moe) ? 'Yes' : 'No';         // Yes

print ($bar === $foo) ? 'Yes' : 'No';        // Yes
print ($bar === $baz) ? 'Yes' : 'No';        // Yes
print ($bar === $moe) ? 'Yes' : 'No';        // No
```

Arrays

- PHP arrays are not really arrays but actually ordered maps of *elements* which are *key => value* pairs
- Keys can be integers or strings only
- Values can be any data type
- In the same array, keys don't have to be of the same type
- In the same array, values don't have to be of the same type

```
<?php
$foo = array(
    1          => 'a',
    'foo'     => 1876,
    'c',
    8         => array('w', 'x'),
    new stdClass()
);
```

Arrays

- If no key is given for the element, PHP will give it an integer key. Counting from 0 and always giving the next smallest available integer key (maximal key + 1)
- Because arrays are maps, giving an existing key again will override the previous element
- Key value automatic casting
 - Double or boolean will be converted to integer
 - NULL will evaluate to an empty string
 - String will be casted to integer only if it is exactly the decimal notation of the integer
- Array elements are ordered by the order their keys were added

Pop quiz

- What will be the content of \$foo?

```
<?php
$foo = array('a', 4 => 'b', 'john' => 'doe', false => 'c', 'e');
```

```
array(0 => 'c', 4 => 'b', 'john' => 'doe', 5 => 'e');
```

- And the output of the following code?

```
<?php
echo count(array('fred', false => 'wilma', '0' => 'barney',
                '' => 'betty', 0 => 'dino'));
```

```
1
```

Comparing arrays

- Comparison operators

- The result of a comparison operation is always boolean
- == equivalence – evaluates to true if the two arrays have the same paired elements, regardless of the order
- === identity – evaluates to true if the two arrays have the same paired elements in the same order
- != not equivalent – evaluates to true if the two arrays don't have the same paired elements
- !== not identical – evaluates to true if the two arrays don't have the same paired elements or if the elements are not in the same order

Arrays

- Functions
 - <http://php.net/manual/en/ref.array.php> has over 75 array specific functions
 - There are more which are not just for arrays
- Array iteration
 - for
 - foreach
 - while
 - do-while
 - array pointer iteration

Arrays

```
$foo = array(9=>'what', 0=>'a', 2=>'wonderful', 'small'=>'world');
foreach($foo as $key => $value) {
    echo $value, ' ';
    $value = 4;
}
echo PHP_EOL;

reset($foo);
while(null !== key($foo)) {
    echo current($foo), ' ';
    next($foo);
}
echo PHP_EOL;

for($i=0; $i<count($foo); $i++) {
    echo $foo[$i], ' ';
}
echo PHP_EOL;
```

```
what a wonderful world
what a wonderful world
a wonderful
```

Arrays

- Arrays are very common in PHP as any data containers and can be used as
 - Stacks (LIFO)
 - Queues (FIFO)
 - Sets (Arbitrary order)
- For each option we have functions to help us handle it
 - *array_push()*
 - *array_pop()*
 - *array_shift()*
 - *array_unshift()*

Superglobal Arrays

- `$GLOBALS` – a reference to every variable which is currently available in the global scope of the current script
- `$_SERVER` – variables set by the web server or otherwise directly related to the execution environment of the current script
- `$_GET` – variables provided to the script via URL query string
- `$_POST` - variables provided to the script via HTTP POST
- `$_COOKIE` - variables provided to the script via HTTP cookie

Superglobal Arrays

- `$_FILES` - variables provided to the script via HTTP post file uploads
- `$_ENV` - variables provided to the script via the environment
- `$_REQUEST` - variables provided to the script via the GET, POST and COOKIE
- `$_SESSION` - variables which are currently registered to the script's session

Before we go on...

- Directives - recommended values (php.ini)

Directive	Development	Production
display_errors	On	Off
error_reporting	E_ALL E_STRICT	E_ALL
short_open_tag	Off	Off
register_globals	Off	Off
magic_quotes_gpc	Off	Off

- Setting the recommended values for development mode will make your application less server dependent. In many cases you can't control the production server

GETting and POSTing data

- In order for PHP to get information from the client's request, we use the PHP built in global arrays - `$_POST`, `$_GET`, `$_REQUEST`, `$_COOKIE` & `$_FILES`
- Always remember that data sent from the client is tainted, unless proven otherwise and it is up to the developer to make certain of that
- There are two methods for the client to explicitly send data to the server's global arrays
 - GET
 - POST

GETting and POSTing data

- POST and GET are two forms of transferring data from the client to the server
- The main difference between the two methods is that POST allows you to send along a data payload to the server, and not just the URI string the GET can
 - Therefore, the GET may be limited in size, depending on the client
 - Some form elements can only be sent through POST (such as file uploads)
- Security-wise the two methods are the same
- It is considered a good practice to use GET for cases you want to retrieve data from the server, and use POST for cases you want to send data to the server

Forms

- Forms are HTML elements which can handle data transfer to the server

```
<form action="index.php" method="GET">  
  <input type="hidden" name="user" value="karnaf" />  
  <input type="submit" value="send" />  
</form>
```

- When a form is submitted through GET, its values are encoded directly into the URI. In the example above, clicking 'send' will cause the browser to call

```
http://localhost/index.php?user=karnaf
```

- On the server side, the submitted value will be available through the `$_GET` superglobal

```
echo $_GET['user']; // will output 'karnaf'
```

Forms

- Similarly, transferring the data to the server using POST will be done through

```
<form action="index.php" method="POST">  
  <input type="hidden" name="user" value="karnaf" />  
  <input type="submit" value="send" />  
</form>
```

- When a form is submitted through POST, its values will be sent through the request's data payload, and not through the URI
- On the server side, the submitted value will be available through the `$_POST` superglobal
`echo $_POST['user'];// will output 'karnaf'`

So far...

- POST & GET Summary (so far)
 - Use POST for sending data to the server and GET to retrieve data from the server
 - Remember that there are cases that require the use of POST
 - Multipart forms (incl. file uploads)
 - The data sent should be specifically encoded
 - Large amount of data are sent
 - Whenever the request is done with a URL that has attributes in its query string, those attributes can be accessed through the `$_GET` superglobal
- As a best practice, `$_REQUEST` should be avoided

Headers

- When the server sends its response to the client, it will start with a block of header information
- We can affect the headers by calling the *header()* function
 - This can only be done before any output is echoed
 - Mostly used to redirect the page

Headers

- Cookies

- Cookies are small text files which are saved on the client side, usually 4-6KB
- Since they are on the client side, they may not be considered safe, and of course, may be deleted by the user or his / her system
- In order to set a cookie, you can use the *header()* function, but it is better to use *setcookie()*
- *setcookie(\$name [, \$value [, \$expire [, \$path [, \$domain [, \$secure [, \$httponly]]]]])*
- Unless a cookie gets an expiration date, it will be valid only until the browser session ends. *\$expire* argument is passed as a UNIX timestamp (number of seconds passed from January 1st, 1970). In the above example it is for one day

Headers

- There is no option to cause the user's client to delete a cookie, however, giving it a new value and setting its expiration date to the past, will cause it to be deleted when the client's session ends

```
setcookie('preferred_lang', NULL, -3600);
```
- When the client sends a request to the server, it will send with the request all the relevant cookies it has
- Accessing the cookies can be done through the `$_COOKIE` superglobal array
- Note that assigning values to `$_COOKIE` from the server side will have no effect on the client side. That is, it can't replace `setcookie()`
- Since the cookies mechanism is a two-stage process, where you set the cookie on the client side and then get it back on the server side, it will only be available on the next request

Saving the state

- Sessions

- HTTP is a stateless protocol, meaning that it does not save data or state between requests. Each request is therefore unique and has its own context and scope
- Sessions are an option to maintain data between requests, by keeping it on the server, usually as a file
- This is done by passing a unique session identifier between requests, usually using cookies
- Since *session_start()* actually uses cookies, it must be called before any output is returned to the client
- It is considered a good practice to call *regenerate_session_id()* whenever a change in a user's access or privileges is done. This is to avoid session fixation problem

Saving the state

- Accessing the session data on the server side can be done *only* after calling `session_start()` (if the session isn't started automatically) and through the `$_SESSION` superglobal array
- Saving things to `$_SESSION` will keep them across requests

Recap

Client Side

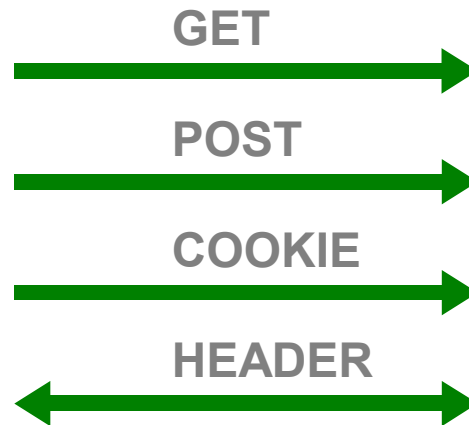


Cookie file

Server Side



Session



Questions?