University of Nottingham
UK | CHINA | MALAYSIA

G51DBI

PHP (2)

Tim Brailsford

HTML forms

- HTML forms are used to submit user data to PHP scripts
- The action attribute is the name of the script
- The method attribute is the HTTP protocol used to transmit the data (either PUT or GET).
- Data from the for is accessed using PHP superglobals

Form example (GET)

php-demo15.html

```
<html>
<body>

<form action="hello.php" method="GET">
  What is your name?
  <input type="text" name="personName">
  <input type="submit">
</form>
</body>
</html
```

hello.php

```
<html>
<body>
<h2>Hello
  <?php
     echo $_GET["personName"];
  ?>
</h2>
</body>
</html
```

Form example (POST)

php-demo16.html

```
<html>
<body>

<form action="hello2.php" method="POST">
  What is your name?
  <input type="text" name="personName">
  <input type="submit">
</form>
</body>
</html
```

hello2.php

```
<html>
<body>
<h2>Hello
  <?php
     echo $_POST["personName"];
  ?>
</h2>
</body>
</html
```

GET vs POST

- HTML forms are used to submit user data to PHP scripts
- Different HTTP request methods
- GET passes data on the URL
- POST passes data from the server to the browser internally
  - GET requests can be cached, bookmarked and remain in the browser history POST does not
  - GET requests have length restrictions, POST does not
  - GET requests are visible to everyone and so easy to reverse engineer
  - POST requests are invisible so more secure

Be aware of security issues

- There are potential PHP "exploits" - especially "Cross Site Scripting" (XSS)
- Form Validation makes this much less likely
- Various PHP functions can be used to process user input and prevent attacks
  - Use regular expressions to validate data items
  - htmlspecialchars() returns a string with special HTML characters (e.g. <) converted to entities (e.g. $lt;).
  - trim() removes unnecessary whitespace
  - stripslashes() removes backslashes
- See: https://www.w3schools.com/php/php_form_validation.asp

Session Variables and Cookies

- Both store variables
  - Can be used by the same or other pages, at some point in the future.
- Session variables
  - Stored in memory on the server
  - Persist for the duration of the "session" 24 minutes or less if the session is explicitly ended.
  - Browser agnostic
- Cookies
  - Stored in a file on the client
  - Persist until expiration (seconds to months)
  - Browser specific

Setting a cookie

```
<?php
 $cookie_name = "person";
 $cookie_value = "Tim";
 $cookie_time = 20;
 setcookie($cookie_name, $cookie_value, time()+$cookie_time);
?>
```

- Cookies must be set before the <html> tag
- Time is usually in days (e.g. 86400 * 2 is 2 days)

Reading a cookie

```
<?php

  $cookie_name = "person";

  if(!isset($_COOKIE[$cookie_name]))
  {
     … cookie not set
  }
     else
  {
     echo "Value: " . $_COOKIE[$cookie_name];
  }

?>
```

Deleting a cookie

```
<?php

 $cookie_name = "person";

 setcookie($cookie_name, null, -1);

?>
```

Cookie issues

- Intended for long term storage of information (e.g. personal preferences or user profiles)
- Not compatible across browsers
- Cookies can be disabled by browsers
  - Before using cookies, check that they are working!
- Moderately but not totally secure
  - It is bad practice to store sensitive information (e.g. passwords) in cookies

Session variables

- Use session_start() function to start a new session
  - NB you can only have one session at once
  - This will last for 24 minutes, unless actively destroyed
  - session_start must be executed in each PHP file that uses session variables
- Use the $_SESSION superglobal to assign and read variables
  - This is an associative array - the session variable 'myVar' is thus $_SESSION['myVar']
- A session can be ended using session_destroy()
  - This destroys all session variables
  - NB you must do a session_start() before you can do a session_destroy()

Create a new session

```
<?php

 session_start();

 $_SESSION["person"] = "Tim" ;
 $_SESSION["email"] = "tim.brailsford@nottingham.ac.uk";

?>
```

Accessing a session variable

```
<?php

 session_start();

 echo "My name is $_SESSION['person']";

?>
```

Deleting session variables

- Delete an individual session variable

```
<?php
 session_start();
 unset ($_SESSION['person']);
?>
```

- Destroy the entire session

```
<?php
 session_start();
 session_destroy();
?>
```

Session variable issues

- Intended for communicating between web pages
- Non-persistent
  - 24 minute maximum
- No browser issues and cannot be disabled
- Much more secure than cookies
- There can be problems on congested networks
  - Disciplined coding
  - Delete session variables when you have finished with them and check that they are set before you use them

Still to come in PHP...

- Database stuff (mysqli)