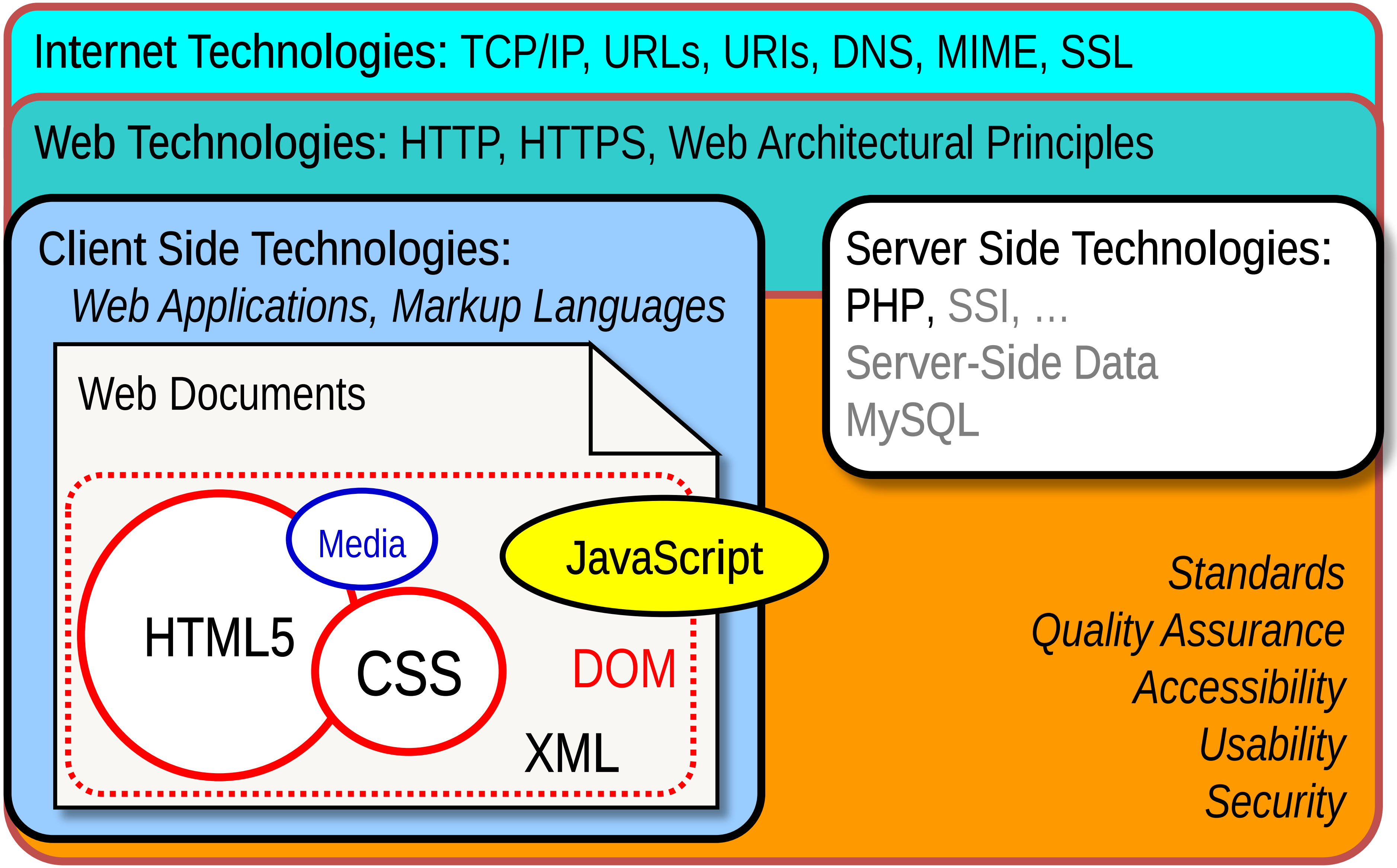
SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10011 Creating Web Applications

Lecture 9 – Server-side Programming PHP: Part 2

Acknowledgement: The contents in this document was adopted from learning material prepared by Alan Colman (Convener, SUT)

Unit of Study Outline


Internet Technologies: TCP/IP, URLs, URIs, DNS, MIME, SSL
Web Technologies: HTTP, HTTPS, Web Architectural Principles
Client Side Technologies:
Web Applications, Markup Languages
Web Documents
Media
HTML5
CSS
JavaScript
DOM
XML
Server Side Technologies:
PHP, SSI, …
Server-Side Data
MySQL
Standards
Quality Assurance
Accessibility
Usability
Security

Last Week

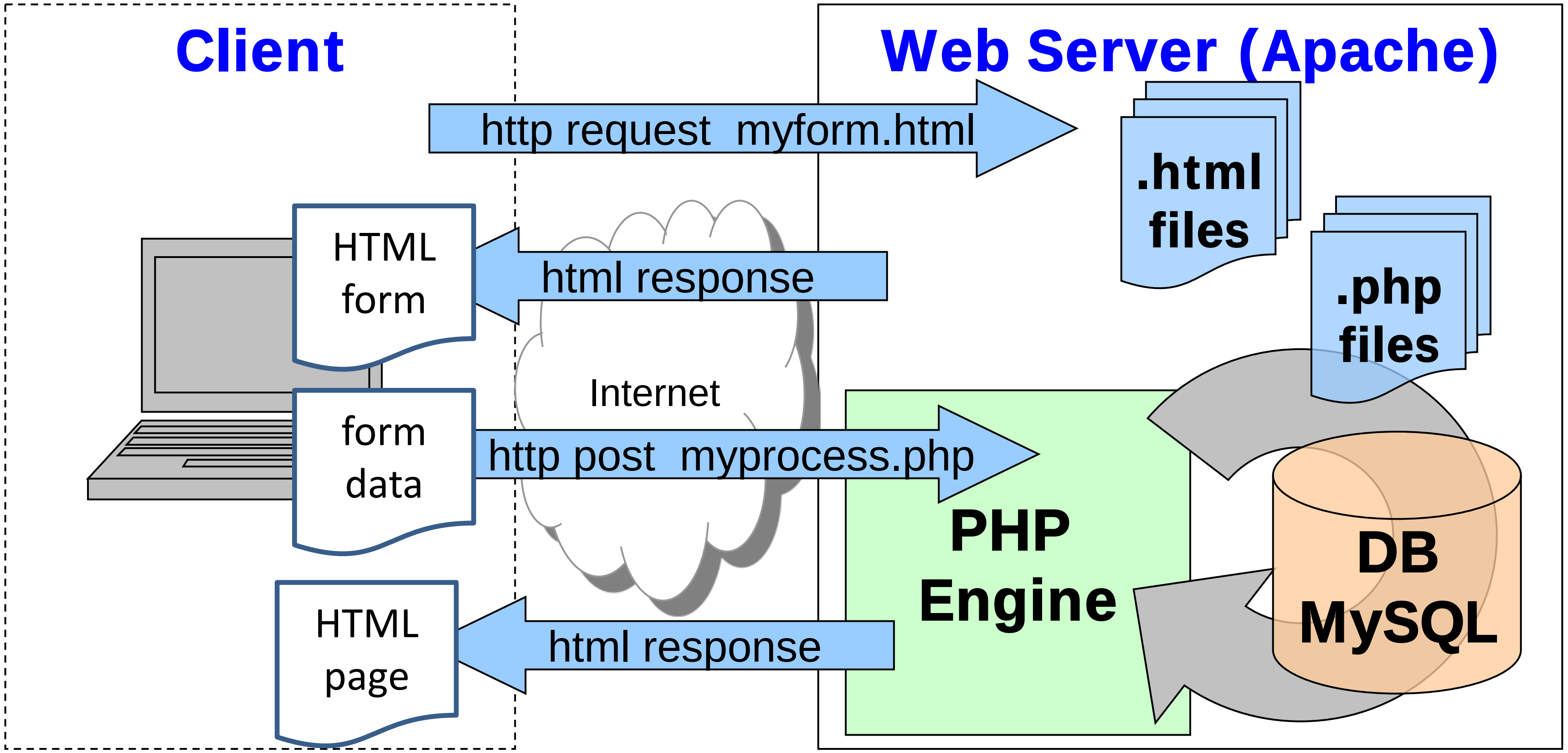
- Client/Server Architecture
- PHP Scripting
- PHP Variables and Constants
- Data Types
- Arrays
- Expressions
- Functions and Scope
- Control Flow

This week - Outline

- Form Data Processing
  - Form and process files
  - superglobal variables
- Input validation
- Includes
- Managing 'state' between client and server (hidden fields, query strings, sessions)
- Managing Page Flow (hidden inputs, self call, redirection)

Server-Side Scripting and PHP

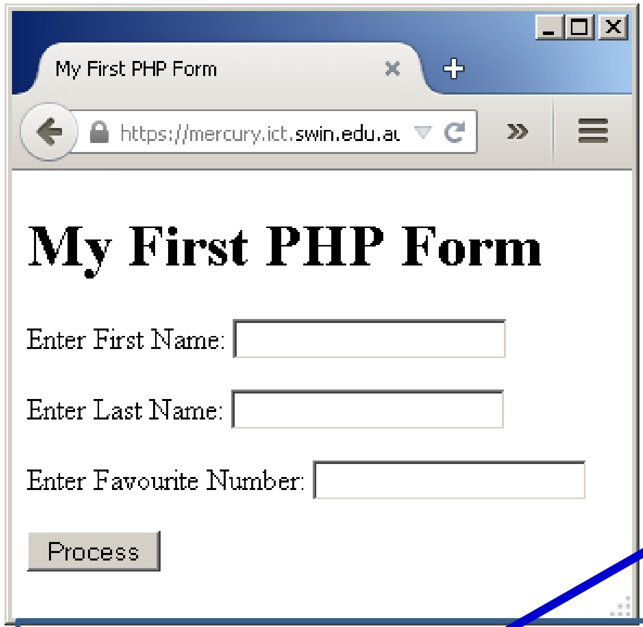
Apache/PHP/MySQL example



FORM DATA EXTRACTION AND SUPERGLOBALS

http://php.net/manual/en/language.variables.scope.php

Form data extraction



My First PHP Form

Enter First Name:

Enter Last Name:

Enter Favourite Number:

Process

must match

```
....
<form method="post" action="php_process1.php">
    <p>Enter First Name:
    <input type="text" name="fname" />
    ...
    <input type="submit" value="Process" />
    </p>
</form>
...
```

myfirst_phpform.html

php_process1.php

```
<?php
   ...
//Transfer form data to variables
   $fname = $_POST["fname"];
   $lname = $_POST["lname"];
   $num = $_POST["num"];

   ...
   echo "<h1>Welcome
            $fname!</h1>";
   ...
?>
```

superglobals

Form Data Extraction Using Superglobals

- $_GET and $_POST **superglobals** (or autoglobals) are to read and array of name-value pairs submitted to the PHP script
- Superglobals are **associative arrays** – arrays whose elements are referred to with an alphanumeric key instead of an index number

  e.g. `$studentId = $_POST["studentId"];`

  "Key" instead of index number

- Are always accessible, regardless of scope

See ***Predefined Variables, Superglobals and examples:***
http://php.net/manual/en/reserved.variables.php

Using Superglobals (continued)

- `$_GET` is the default method for submitting a form
- `$_GET` and `$_POST` allow you to access the values sent by forms that are submitted to a PHP script
- **GET method** appends form data as one long string to the URL specified by the **action** attribute
  - typically used for ***get*** information from a resource e.g. getting a record from a database
- **POST method** sends form data in the body of the HTTP request, not visible in the URL
  - typically used for ***creating*** a resource e.g. creating a new record in a database

More Superglobals

- Superglobals contain client, server, and environment information that you can use in your scripts

See ***Predefined Variables, Superglobals and examples:***
http://php.net/manual/en/reserved.variables.php

Using Superglobals (continued)

```
echo "This script was executed with the
  following server software: ",
  $_SERVER["SERVER_SOFTWARE"], "<br />";
echo "This script was executed with the
  following server protocol: ",
  $_SERVER["SERVER_PROTOCOL"], "<br />";
```

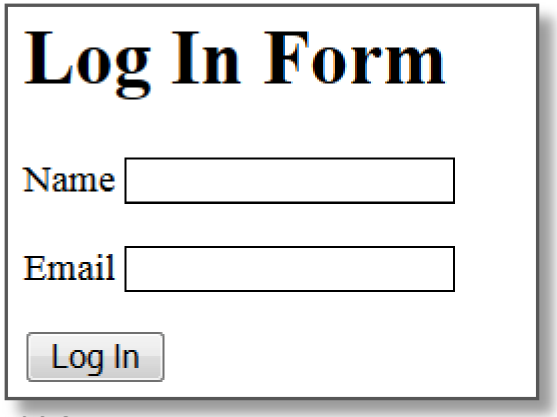
Associative array of pre-defined elements (in capitals)

Using Superglobals (Example 2)

- Given the following registration form

```
<body>
<h1>Log In Form</h1>
<form method="post" action="storeName.php">
  <p><label for="uname">Name</label>
  <input id="uname" type="text" name="uname"></p>
  <p><label for="email">Email</label>
  <input id="email" type="text" name="email"></p>
  <p><input type="submit" value="Log In" /></p>
</form>
</body>
```

form control name values will become index name for the superglobal associative array



Using Superglobals (Example 2)

- In the file **storeName.php**, data are extracted via superglobal **$_POST**, given form method="post"

Any preferred variable name

Name from the input form

```
...
$uname = $_POST['uname'];
$email = $_POST['email'];
echo "<p>User name: $uname<br/>";
echo "Email: $email</p>";
...
```

FORM DATA CHECKING USING PHP

Checking Form Data at the Server

- Essential to validate incoming data:
  - Maintain integrity of the server data
  - Help prevent malicious attack – e.g. SQL injection
- Checking GET or POST has been entered
- Validating data formats
- Cleansing input data

Example

See:

http://www.w3schools.com/php/php_form_validation.asp

Checking GET or POST data exists

- Use the **`isset()`** function to ensure that a variable is set before you attempt to use it

```
<?php
  if (isset ($_POST["fname"]))
      $fname = $_POST["fname"];
  else
      echo "Error: Please enter data in the
            <a href=\"php_form1.php\">form</a>";
?>
```

Assign data to local variable name if it exists

Validating data formats – e.g. strlen

```
<$php
    if (isset ($_POST["fname"])) {
        $fname = $_POST["fname"];
        $err_msg = ""; // validate data by assuming all is correct
        if (strlen ($fname) == 0 ) {        // Look for data that is wrong
            $err_msg .= "<p>Error: enter first name.</p>";
        }
        if ($err_msg == "") {               // Proceed if nothing is wrong
            echo "<h1>Welcome $fname!</h1>";
        } else {        // Display error message, if data validation fails
            echo $err_msg;                  }
    } else
        echo "Error: Please enter data";
?>
```

Same approach as used in JavaScript

Validating data formats – RegExp

```
<$php
    if (isset ($_POST["fname"])) {
        $fname = $_POST["fname"];
        $err_msg = "";
         if (!preg_match("/^[a-zA-Z ]*$/",$fname)) {
        $err_msg .=
                "<p>Only letters and spaces allowed.</p>";
        }
        …
    }
    } else
        echo "Error: Please enter data";
?>
```

PHP function

Same pattern as used in JavaScript

Regular expressions in PHP

int preg_match (string $pattern , string $subject)

- Performs regular expression match
- Returns 1 if the pattern matches given subject, 0 if it does not, or FALSE if an error occurred.

- For more complex forms of the function see http://php.net/manual/en/function.preg-match.php

Validating using the filter_var function

- filter_var() filters a variable predefined filters
- Returns the filtered data, or FALSE if the filter fails, e.g.

```
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $err_msg .=  "Invalid email format";
}
```

Pre-defined filter

- Predefined filters for validating
  - email, types, ip addresses, URLS, …
- Filters also available for sanitising data

http://php.net/manual/en/function.filter-var.php

Sanitising data

- Because code can be mixed with HTML, form data are vulnerable to 'code injection'.
- Making sure there are no control characters in the data sent to a PHP script can help prevent this.
- You can write a small function like:

```
function sanitise_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
```

Remove leading or trailing spaces

Remove backslashes in front of quotes

Converts HTML control characters like **<** to the HTML code **&lt;**

Ex: Sanitising data before processing

```
<$php
    function sanitise_input($data) {
        $data = trim($data);
        $data = stripslashes($data);
        $data = htmlspecialchars($data);
        return $data;
    }
    if (isset ($_POST["fname"])) {
        $fname = $_POST["fname"];
         $fname = sanitise_input($fname) {
            if (!preg_match("/^[a-zA-Z ]*$/",$fname)) {
... } ...?>
```

PHP INCLUDES

PHP Includes

- Facilitates the reuse of PHP code at the files level
- Useful for including recurring functionality or content e.g. menus
- [Optional] See Server-side Includes in the Extras section on Blackboard

PHP include example

```
<!DOCTYPE html>
<html lang="en">
<head>
        …
</head>
<body>
        <?php
                include_once ("php_menu.php");
        ?>
        <!-- Web page starts here -->
        <h1>Input checking using input values</h1>
        …
</html>
```

include_once ensures that the code is only included once

Whatever text is in the file php_menu.php will be inserted at this point

PHP include and require

```
<!DOCTYPE html>
<html lang="en">
<head>
        …
</head>
<body>
        <?php
                require ("php_menu.php");
        ?>
        <!-- Web page starts here -->
        <h1>Input checking using input values</h1>
        …
</html>
```

Same as include by will produce a fatal error if the file is missing

MANAGING STATE

SWINBURNE UNIVERSITY OF TECHNOLOGY

Managing State

Techniques for **maintaining state** information with PHP include:

- Hidden form fields
- Query strings
- Sessions

Understanding State Information

- HTTP was originally designed to be **stateless** – Web browsers store no persistent data about a visit to a Web site
- We need techniques to **maintaining state**: i.e. store persistent information about Web site visits, that can be passed backwards and forwards between the client and the server.
- We have previously used Web Storage and Cookies to store information locally on the client
- Information about individual visits to a Web site also needs to be maintained on the server

Understanding State Information (cont)

Some reasons why a web application may need to **maintain state** information:

- Temporarily store information for a user as a browser navigates within a multipart form
- Allow a user to create bookmarks for returning to specific locations within a Web site
- Customize individual Web pages based on user preferences
- Provide shopping carts that store order information
- Store user IDs and passwords
- Use counters to keep track of how many times a user has visited a site

Using Hidden Form Fields to Save State

- Use hidden form fields to temporarily store data that needs to be sent to a server that a user does not need to see
- Examples include the result of a calculation
- Create hidden form fields with the **`<input />`** element using **`type="hidden"`**

```
<input type="hidden" ... />
```

- Hidden form field attributes are **`name`** and **`value`**

Using Hidden Form Fields to Save State

- When submitted to a PHP script, access the values submitted from the form with the **`$_GET[]`** and **`$_POST[]`** Superglobals
- To pass form values from one PHP script to another PHP script, store the values in hidden form fields

Using Hidden Form Fields to Save State

```
<form action="toolLoans.php" method="get">
. . .
<p>
<input type="hidden" name="toolID"
          value="<?php echo $tool_id ?>" />
<input type="submit" value="Hire Tool" />
</p>
</form>
```

Note: The hidden value will be visible if you "view page source" on the client.

Using Query Strings to Save State

- A **query string** is a set of name=value pairs appended to a target URL
- A **query string** consists of a single text string containing one or more pieces of information
- Any forms that are submitted with the **GET** method automatically add a question mark (?) and append the **query string** to the URL of the server-side script

Using Query Strings to Save State

- To pass information from one Web page to another using a query string,
  - add a question mark (?) immediately after the URL
  - followed by the query string containing the information in name=value pairs, and
  - separate the name=value pairs within the query string by ampersands (&)

```
<a href="page2.php?firstName=John&lastName=Smith
&occupation=singer">John Smith</a>
```

Using Query Strings to Save State

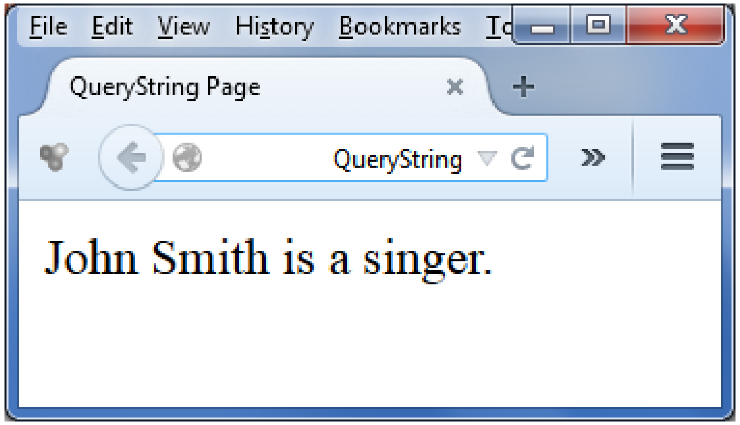
- To pass query string information from one PHP script to another PHP script, echo the values in the script

```
<a href="page2.php?firstName="<?php echo $fname; ?>
"&lastName="<?php echo $lname; ?>
"&occupation="<?php echo $occ; ?>">
<?php echo $fname, $lname; ?></a>
```

Note: The values will be visible in the query string.

Using Query Strings to Save State

```
echo "{$_GET['firstName']} {$_GET['lastName']}
 is a {$_GET['occupation']}. ";
```



John Smith is a singer.

Output of the contents of a query string

Using Sessions to Save State

- A **session** refers to a period of activity when a PHP script stores *state information on a Web server*
- **Sessions** allow you to maintain state information *even when clients disable cookies in their Web browsers*

Starting a Session

```
<?php
session_start();
...
?>
<p><a href='<?php echo
  "occupation.php?PHPSESSID="
    . session_id() ?>'>Occupation</a></p>
```

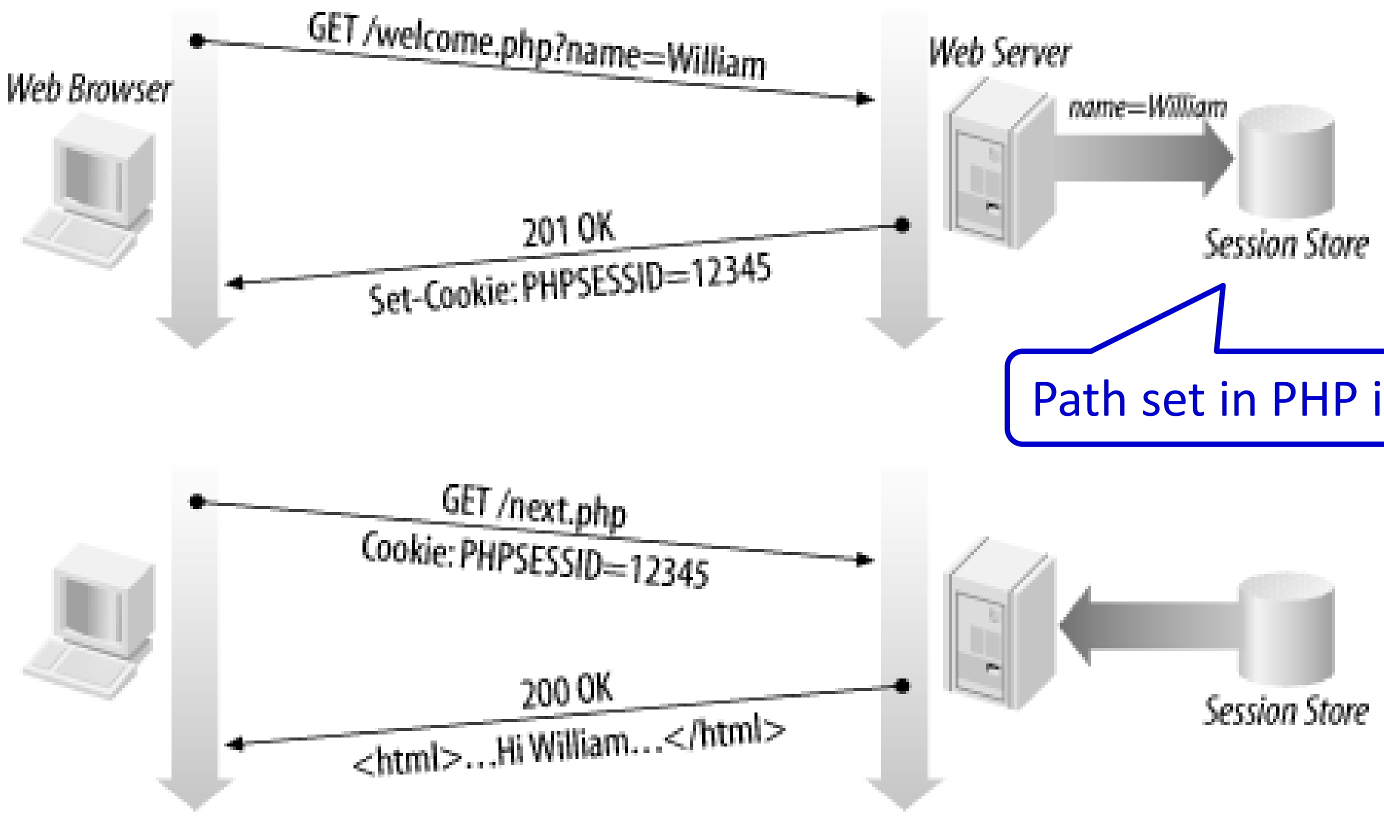
Starting a Session

- The **`session_start()`** function starts a new session or continues an existing one
- The **`session_start()`** function generates a unique session ID to identify the session
- A **session ID** is a random alphanumeric string that looks something like:

  ```
  7f39d7dd020773f115d753c71290e11f
  ```

- The **`session_start()`** function creates a text file on the Web server that is the same name as the session ID, preceded by **`sess_`**

Session interaction



Path set in PHP ini.

Starting a Session (continued)

- Session ID text files are stored in the Web server directory specified by the **`session.save_path`** directive in your php.ini configuration file
- The **`session_start()`** function does not accept any functions, nor does it return a value that you can use in your script

```
<?php
session_start();
...
```

Starting a Session (continued)

- You must call the **`session_start()`** function ***before*** you send the Web browser any output
- If a client's Web browser is configured to accept cookies, the session ID is assigned to a temporary cookie named `PHPSESSID`
- Pass the session ID as a query string or hidden form field to any Web pages that are called as part of the current session

Working with Session Variables

- Session state information is accessed using the **`$_SESSION`** superglobal
- When the **`session_start()`** function is called, PHP either initializes a new **`$_SESSION`** superglobal or retrieves any variables for the current session (based on the session ID) into the **`$_SESSION`** superglobal

Working with Session Variables (continued)

```
<?php
session_set_cookie_params(3600);
session_start();
$_SESSION['firstName'] = "John";
$_SESSION['lastName'] = "Smith";
$_SESSION['occupation'] = "singer";
?>
<p><a href='<?php echo "Occupation.php?"
 . session_id() ?>'>Occupation</a></p>
```

Sets the “lifetime” argument to 3600 seconds

Working with Session Variables (continued)

- Use the **isset()** function to ensure that a session variable is set before you attempt to use it

```
<?php
session_start();
if (isset($_SESSION['firstName']) &&
   isset($_SESSION['lastName'])
      && isset($_SESSION['occupation']))
    echo "<p>" . $_SESSION['firstName'] . " "
          . $_SESSION['lastName'] . " is a "
          . $_SESSION['occupation'] . "</p>";
?>
```

Deleting a Session (continued)

```
<?php
session_start();
$_SESSION = array();
session_destroy();
?>
```

Step 1

Step 2: Use the array() construct to reinitialize the $_SESSION superglobal

Step 3: Delete the session

This is the code often used for a “Log-out” script, or the code that is included in a “Registration” / “Log In” page, so that it deletes any existing user sessions whenever a user opens it.

SWINBURNE UNIVERSITY OF TECHNOLOGY

- http://phpcodechecker.com/

SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10011 Creating Web Applications

What's Next?

- Server-side Data
- PHP and MySQL

Acknowledgement: The contents in this document was adopted from learning material prepared by Alan Colman (Convener, SUT)