Advanced Higher

HTML Forms and PHP

Continuation from Higher

The Higher Computing Science course defined the use of form elements and input types to include validated input for text, numeric, and restricted-choice entry (select and radio). There are no additional input methods or validation in the Advanced Higher course.

The focus of web content in the Advanced Higher course is the use of PHP to integrate with an SQL database. This includes server-side processing of HTML form code introduced at Higher.

Form action and method attributes

To process the contents of an HTML form, an action and method must be initiated when the form's 'submit' button is clicked. These are coded as attributes of the <form> element.

```
<form action="registerStudent.php" method="POST">
```

The action shown above states the file "registerStudent.php" will be opened when the form is submitted. As this is a PHP file, the web server where it is stored will automatically execute the PHP code contained in the file.

You can submit a form using one of two HTTP methods:

- GET
- POST

The submission process for both methods begins the same way, with the browser constructing a form data set.

GET

If you submit a form with `method="GET"`, the browser constructs a URL by taking the value of the action attribute, appending a ? to it, then appending the form data set. It then processes this URL as if following a link. The browser divides the URL into parts and recognises a host, then sends a GET request to that host, with the rest of the URL as an argument.

Advantages of using GET:

- If security is not an issue, the URL can be bookmarked, allowing it to be re-used without having to complete and submit the original form.
- If there is a network connection issue when a form is submitted, the browser will automatically resend the form, as it assumes it does not contain sensitive data.
- GET submissions can usually be cached. If the same submission is used regularly (for example form data used to generate the same database query), this could have a significant effect on efficiency.

Disadvantages of using GET:

- The form data in the constructed URL is visible and so less secure.
- URLs can only contain ASCII codes, which will cause issues if the form data contains non-ASCII characters.
- The URL constructed will be stored in the user's web browsing history, making it inappropriate for sensitive data.
- URLs have a limited number of characters, which limits the form data submitted.

Recommended use
GET is usually used when non-sensitive data (like the parameters of a database query) is sent to a server.

POST

When you submit a form using the POST method, the form data set is encoded within a message that is sent to the server.

Advantages of using POST:

- The submitted form data is not visible and so more secure than GET.
- Non-ASCII characters can be submitted within the form data set.
- There is no URL character limit, so form data can be much larger.

Disadvantages of using POST:

- The submitted form cannot be bookmarked for later use.
- If there is a network issue while the form is being submitted, the browser will ask the user to resubmit the form.

Recommended use
POST is usually used when sensitive data (like personal information) is sent to a server.
A database update would usually be initiated with the POST method.

POST can also be used for non-sensitive data: if the submitted data is likely to contain non-ASCII characters or the length is over the limit of a URL.

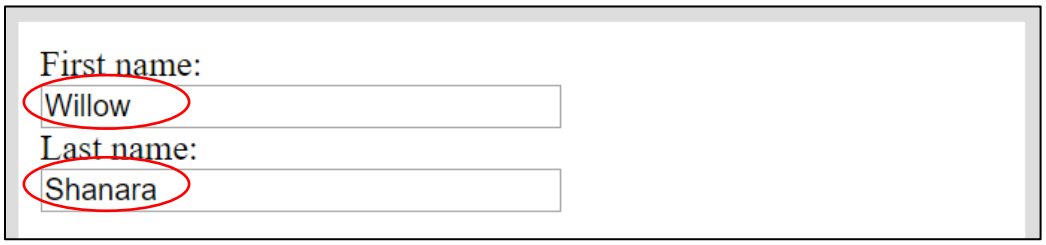
Name and value attributes for form element and input types

The form data set is comprised of key/value pairs, where key is a declared attribute of the form input called `name` and `value` is the data entered by the user.

In the case of text and numeric input, the `name` attribute is defined within the input type. These attributes are used when processing the form and must match the attributes used when the submitted form data is assigned to server-side variables.

```
First name:<br>
<input type="text" name="firstname" size="30" maxlength="15" required>
<br>

Last name:<br>
<input type="text" name="lastname" size="30" maxlength="15" required>
<br><br>
```

The `value` can be a number, a character or a string that the user types into the form's field or one that has been defined in the HTML form.



In the case of a drop-down menu or a radio button input, both the `name` and `value` are defined in the HTML code.

```
Select play:
<select name="play">
  <option value="hamlet">Hamlet</option>
  <option value="godo">Waiting for Godo</option>
  <option value="brothers">Blood Brothers</option>
  <option value="curious">Curious Incident</option>
</select>
<br><br>
```

```
Choose your age:<br>
<input type="radio" name="age" value="12 to 14" checked> 12-14
<input type="radio" name="age" value="15 or 16"> 15,16
<input type="radio" name="age" value="over 16"> over 16
```

PHP form processing (WDD)

The Advanced Higher course requires candidates to execute server-side code to:

- process HTML form data, using PHP
- store submitted form data within a database table, using SQL and PHP
- query a database, using SQL and PHP
- display the results of a query within HTML table elements, using PHP

Database and web servers

To execute PHP files, you need a database server (connected to a database to store or retrieve data) and a web server. Although we usually think of a server as hardware, a web and/or database server setup is a collection of software technologies that may be:

- installed on and run from a local PC or hardware server
- installed on and run from a USB flash drive
- installed on and run from an external PC or hardware server across the World Wide Web

There are many ways to install the required software. These range from builds of individual components (which requires knowledge, expertise and time), to prebuilt, simple installations that require a single install such as XAMPP, WampServer or EasyPHP.

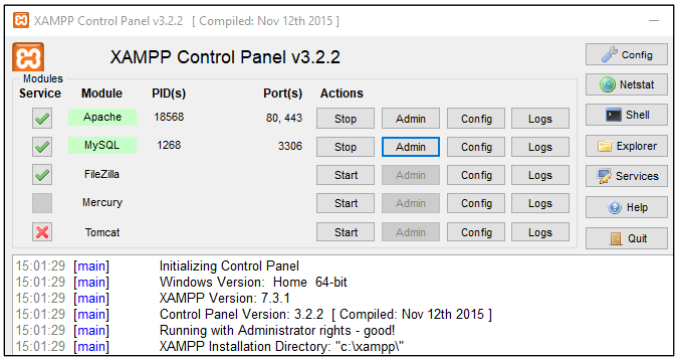
Executing PHP files

You must have the following to execute .php files:

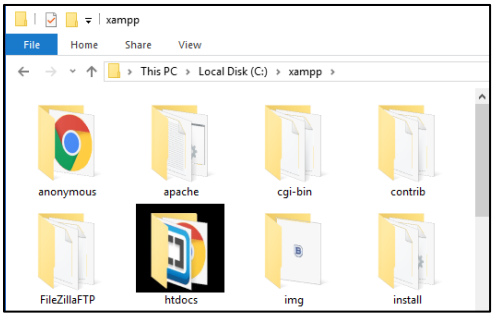
- web server software installed and running
- .php files saved to a specific folder within the installed server folders

The following examples demonstrate this for an XAMPP setup.

XAMPP control panel showing both Apache (web server) and MySQL (database server) applications running



XAMPP folder htdocs where .php files are located



What is a PHP file?

PHP files are text files that can contain HTML, CSS, JavaScript, and PHP code.

When a .php file is executed on a server, the PHP code it contains can:

- collect form data
- add, delete and modify data in your database
- generate dynamic page content

When a .php file is executed, the results are returned to the browser as a plain HTML file.

The following examples use code taken from the Advanced Higher example website. You can download the example website from SQA's secure site.

HTML forms

The Drama page on the Advanced Higher example website contains the following form.

```
<form action="registerStudent.php" method="POST">

  First name:<br>
  <input type="text" name="firstname" size="30" maxlength="15" required>
  <br>

  Last name:<br>
  <input type="text" name="lastname" size="30" maxlength="15" required>
  <br><br>

  Select play:
  <select name="play">
    <option value="hamlet">Hamlet</option>
    <option value="godo">Waiting for Godo</option>
    <option value="brothers">Blood Brothers</option>
    <option value="curious">Curious Incident</option>
  </select>
  <br><br>

  Number of Tickets Required (between 1 and 3):
  <input type="number" name="tickets" value="1" min="1" max="3">
  <br><br>

  Choose your age:<br>
  <input type="radio" name="age" value="12 to 14" checked> 12-14
  <input type="radio" name="age" value="15 or 16"> 15,16
  <input type="radio" name="age" value="over 16"> over 16

  <br><br>

  If required, please delete and state any special requirements:<br>
  <textarea name="message" rows="6" cols="30">none</textarea>
  <br><br>

  <input type="submit" value="Register">
</form>
```

To process a form, the server requires the following:

- `action=""`

  This contains the name of the file to be executed when the form is submitted. This can be the current file or a different file.

- `method=""`

  The method used to submit the form can be GET or POST:

  — GET — the submitted data is visible to the user and therefore not secure

  — POST — the submitted data is hidden from the user (forms are almost always submitted using the POST method)

- `name="" and value=""`

  When the data in the form is submitted to the server, it is converted into an array of key/value pairs (where key is the `name` of the form controls and `value` is the data entered by the user).

  Note: with <select> and radio input, the values are defined in the form code.

PHP form processing

When the form on the Drama web page is submitted, the following file is executed:

`registerStudent.php`

A .php file may contain HTML, CSS and JavaScript, so you must identify any PHP code by placing it inside a PHP script.

```
<?php
    // PHP code goes here
?>
```

Assign form data to server-side variables

The values in the array passed from the submitted form are assigned to separate variables in the lines below. Each of these lines uses the `$_POST[" "]` method to extract values from matching variables first declared in the form.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
      $forename = $_POST["firstname"];
      $surname = $_POST["lastname"];
      $play = $_POST["play"];
      $tickets = $_POST["tickets"];
      $age = $_POST["age"];
      $requirements = $_POST["message"];
}
```

Note: these assignments are placed within a conditional statement, which checks that the form has been submitted:

If a form is submitted using the $_GET method, POST would be changed to GET as shown below.

```
if ($_SERVER["REQUEST_METHOD"] == "GET") {
      $forename = $_GET["firstname"];
      $surname = $_GET["lastname"];
      $play = $_GET["play"];
      $tickets = $_GET["tickets"];
      $age = $_GET["age"];
      $requirements = $_GET["message"];
}
```

Checking for unwanted characters

Note: this is not a requirement of the Advanced Higher course, but may be useful for project work.

For security purposes, you should check the form data for unwanted characters. The function `test_input()` has been created to pass the user's inputted data through three predefined PHP functions and return the result. The functions `trim()`, `stripslashes()` and `htmlspecialchars()` remove different unwanted characters from a string.

```
 8     if ($_SERVER["REQUEST_METHOD"] == "POST") {
 9     $forename = test_input($_POST["firstname"]);
10     $surname = test_input($_POST["lastname"]);
11     $play = test_input($_POST["play"]);
12     $tickets = $_POST["tickets"];
13     $age = test_input($_POST["age"]);
14     $requirements = test_input($_POST["message"]);
15     }
16
17     function test_input($data) {
18       $data = trim($data);
19       $data = stripslashes($data);
20       $data = htmlspecialchars($data);
21       return $data;
22     }
```

Open and close connection to database

To connect to a database, you need to define the following parameters. You can enter these directly into the connection function or store them in variables, as shown below.

```
3       $servername = "localhost";
4       $username = "root";
5       $password = "";
6       $dbname = "studentList";
```

Line 3 defines the host name of the server. Line 4 defines the username used to connect to the server. "root" is a default value that is usually set with administration rights for the server.

For security purposes, server access is usually password protected (for example `$password="hsd56XC89"`). For teaching purposes, the password string can be left empty as shown in line 5. The name of the database the script will connect to is stored in line 6.

The function `mysqli_connect()` is used to connect to the server:

```
$conn = mysqli_connect($servername, $username, $password);
```

or to connect to a database stored on the server:

```
$conn = mysqli_connect($servername, $username, $password, $dbname);
```

This function returns a Boolean True if the connection is successful. You can use the Boolean value to:

- ensure the script only proceeds when a proper connection is made
- return an error message if the function returns false
- kill the script using `die()` if a connection is not made, as shown below

```
24      // Create connection
25      $conn = mysqli_connect($servername, $username, $password);
26
27      // Check connection
28      if (!$conn) {
29          die("Connection failed");
30      }
```

`$conn` stores a single instance of a connection.

Connections should be closed using the function `mysqli_close()` at the end of a script.

```
73      mysqli_close($conn);
74  ?>
```

Executing an SQL query to insert submitted form data into a database table

After the submitted form data has been assigned to PHP variables and a connection to the database has been established, you can use SQL to add the form data to a database table.

The function `mysqli_query()` is used to execute an SQL statement, as shown below.

```
61      $sql = "INSERT INTO studentData (forename, surname, play, tickets, age, requirements)
62      VALUES ('$forename', '$surname', '$play', '$tickets', '$age', '$requirements')";
63
64      if (mysqli_query($conn, $sql)) {
65         echo '<script type="text/javascript">alert("Sucessfully Registered");</script>';
66      } else {
67         echo "Error inserting data";
68      }
```

The function requires two parameters:

- the connection ($conn) used to identify the connection to the database being used
- the SQL statement including the PHP variables, that now store the form data

Line 67 shows the use of echo to output a message. The echo statement is often used in PHP coding to output HTML code, which is then interpreted by the browser and displayed.

```
echo "<p>Hello world</p>";
```

Additional notes:

1 You can write the SQL statement directly into the function, but it is common practice to assign the SQL statement to a variable, which is then used in the function. This makes the code more readable.

2 For an SQL INSERT statement, the function returns a Boolean value (True = success, False = failed). This can then be used to return messages.

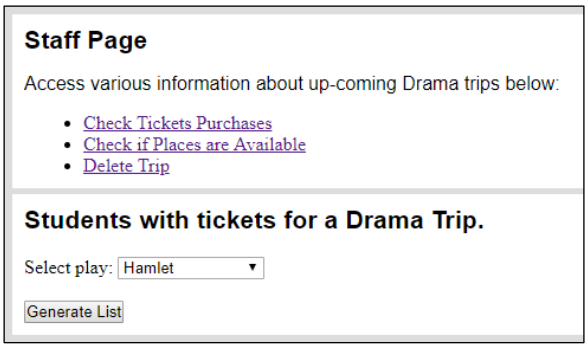
3 The example above uses a JavaScript alert to inform the user that their drama trip details have been successfully added to the database. This is not a requirement of the Advanced Higher course, but may be a useful tool to visually demonstrate the success of the `mysqli_query()` function, without using echo.

Executing an SQL query and displaying formatted results using PHP

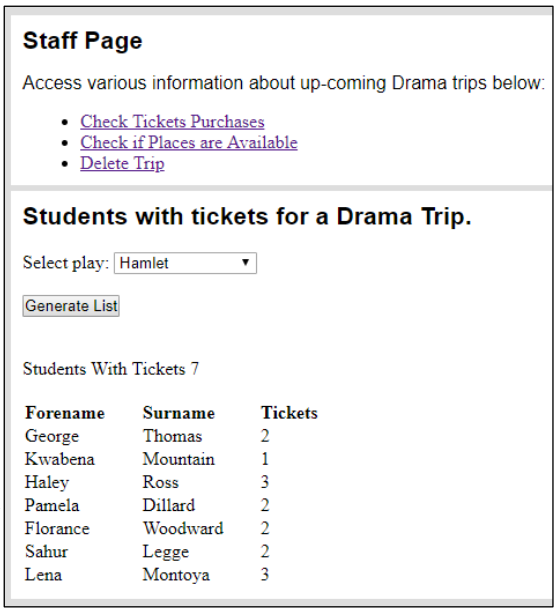
The staff page on the Advanced Higher example website includes two further examples of web and database integration.

Check ticket purchases

This example uses a simple form to input the name of a play. The web page outputs a list of students, with the number of tickets each student has purchased.



When the 'Generate List' button is clicked, the page is reloaded, with the query output displayed in a table.



The results of an SQL SELECT statement returns output in the form of an array. The following code was used to display the returned data.

```php
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $play = $_POST["play"];

    $conn = mysqli_connect($servername, $username, $password, $dbname);
    if (!$conn) {die("Connection failed");}

    $sql = "SELECT forename, surname, tickets FROM studentData WHERE play = '$play'";
    $result = mysqli_query($conn, $sql);

    echo "<br><br>Students With Tickets ".mysqli_num_rows($result)."<br><br>";

    if (mysqli_num_rows($result) > 0) {

        echo "<table>";
        echo "<tr><th style='width:100px;text-align:left'>Forename</th> <th
        style='width:100px;text-align:left'>Surname</th> <th style='width:50px;text-
        align:left'>Tickets</th></tr>";
        // output data of each row
        while($row = mysqli_fetch_array($result)) {
            echo "<tr><td>".$row["forename"]."</td> <td>".$row["surname"]."</td>
            <td>".$row["tickets"]."</td></tr>";
            }
        echo "</table>";

    } else {
    echo "0 results";
    }
```

When the ‘Generate List’ button is clicked, line 63 assigns the selected play to the PHP variable `$play`.

An SQL statement is used with `mysqli_query()` to query the database for students who have tickets for the selected play.

```
SELECT forename, surname, tickets FROM studentData WHERE play =
'$play'
```

The PHP function `mysqli_num_rows()` is used in line 71, to display the number of rows returned by the query — which is the number of students found.

Lines 75 to 81 use PHP to display an HTML table. This output is built in three stages:

- the static top part of the table
- the dynamic middle part of the table, where the number of rows displayed will depend on the query result
- the static bottom part of the table

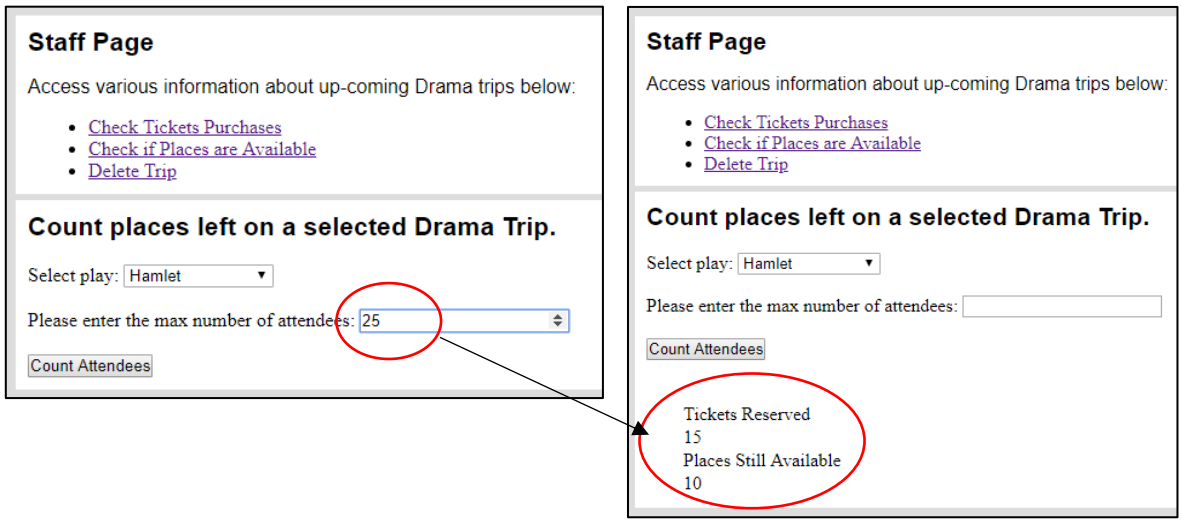
A `while` loop on line 78 uses the function `mysqli_fetch_array()` to extract each row returned by the SQL select statement in turn. Each extracted row is stored as an associative array. The contents of the array are concatenated with the HTML table elements; this is required to create a single row of a three-column table.

```
<tr>   <td></td>  <td></td>  <td></td>   </tr>
```

Note: the first row of the table is displayed as a header row using <th> in place of <td>. Also, `mysqli_num_rows()` is used to ensure the table is only displayed when >0 rows are returned by the query (line 73). If zero rows are returned, `"0 results"` is displayed instead of the table.

Check if places are available

In addition to the name of the play, this form also includes numerical input. The web page counts the total number of tickets purchased for the selected play and calculates the number of places remaining.



This is achieved using the code below.

```
72        $play = $_POST["play"];
73        $maxTickets = $_POST["maxTickets"];
```

```
81        // Create SQL statement returns the number of rows matching a play name.
82        $sql = "SELECT SUM(tickets) FROM studentData WHERE play = '$play'";
83
84        // The query is passed to the server and the result stored as a MySQL result object
85        $result = mysqli_query($conn, $sql);
86        $countArray = mysqli_fetch_array($result);
87        $placesLeft = $maxTickets - $countArray[0];
```

```
90        echo "<br><br><table style='margin-left:30px'>";
91        echo "<tr><td>Tickets Reserved</td></tr>";
92        echo "<tr><td>".$countArray[0]."</td></tr>";
93        echo "<tr><td>Places Still Available</td></tr>";
94        echo "<tr><td>".$placesLeft."</td></tr>";
95        echo "</table>";
```

Note: the first element of the array returned by `mysqli_fetch_array()` stores the numeric, aggregate result of the query.

`$countArray[0]`

Building web pages generated by PHP code

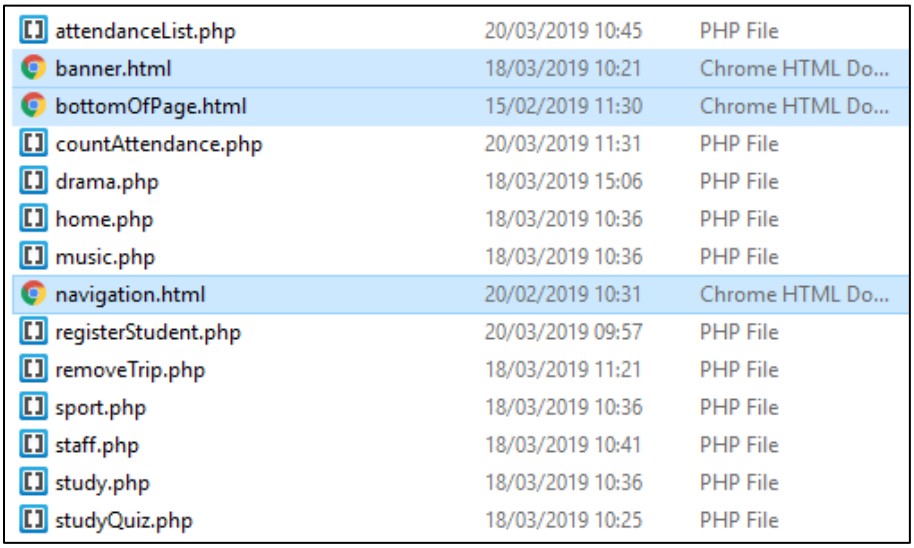
You can generate the HTML returned to a browser by a .php file in the following ways:

- If the PHP script is contained within the same page as a submitted form, then the entire page will be reloaded when the GET or POST script is executed. Any output produced by the script will be included according to the position of the script within the HTML. This is the simplest solution if you wish to stay on the same web page when a form is submitted.
- If you want to generate a completely different page, then the form should load a different .php file. In this case, the PHP file will have to contain all the HTML elements required to build the new page.

The PHP include function

This is not a requirement of the Advanced Higher course, but is an efficient way to build pages without repeating lots of code.

The Advanced Higher example website separates out the <header>, <nav> and <footer> elements of each page, storing them in separate HTML files.



attendanceList.php	20/03/2019 10:45	PHP File
banner.html	18/03/2019 10:21	Chrome HTML Do...
bottomOfPage.html	15/02/2019 11:30	Chrome HTML Do...
countAttendance.php	20/03/2019 11:31	PHP File
drama.php	18/03/2019 15:06	PHP File
home.php	18/03/2019 10:36	PHP File
music.php	18/03/2019 10:36	PHP File
navigation.html	20/02/2019 10:31	Chrome HTML Do...
registerStudent.php	20/03/2019 09:57	PHP File
removeTrip.php	18/03/2019 11:21	PHP File
sport.php	18/03/2019 10:36	PHP File
staff.php	18/03/2019 10:41	PHP File
study.php	18/03/2019 10:36	PHP File
studyQuiz.php	18/03/2019 10:25	PHP File

You can include these elements in each page using the PHP function `include`.

```
14  <body>
15
16  <?php
17  include 'banner.html';
18  include 'navigation.html';
19  ?>
```

In addition to substantially reducing the amount of code in each page, this also makes maintenance of these three elements easier, as their contents are stored in a single location and not repeated across every page of the website.

PHP sessions (WDD)

Definition and use

When a browser loads a new web page, it forgets all the information from the previous page. A PHP session is a way of storing information within a website, so that it can be retained and used across multiple pages.

Sessions work in a similar way to a program. The website code opens (starts) the session. Information is generated, stored and sometimes changed. The website code then closes (destroys) the session to end it.

Examples of session use are:

- retaining selected items in a shopping cart, as the user navigates from page to page
- displaying a user's id on multiple pages, following a successful login
- retaining values, such as a user's quiz Score, when each new question page loads

Starting a session

The following PHP function is used to start a session. This should be placed at the top of a page, before any HTML code. If data is being passed between multiple pages, each page that requires access to the session should contain the PHP code below.

```
<?php
// Start a new session
     session_start();
?>

<!DOCTYPE html>
<html>
<head>
```

When a new session starts, a user key is stored on the user's computer. The `session_start()` function looks to see if a user key exists. If it does, the current session is continued. If no user key exists, a new session is started.

Session variables

Session variables are assigned values, as shown below.

```
$_SESSION['staffLogin']="False";
```

A PHP file that contains `session_start()` has access to any session variables previously created.

Ending a session

The PHP function `session_destroy()` is used to end a session. On the Advanced Higher example website, clicking the 'Log Out' button calls the file `logout.php`. The code below destroys the session and then reloads the staff page.

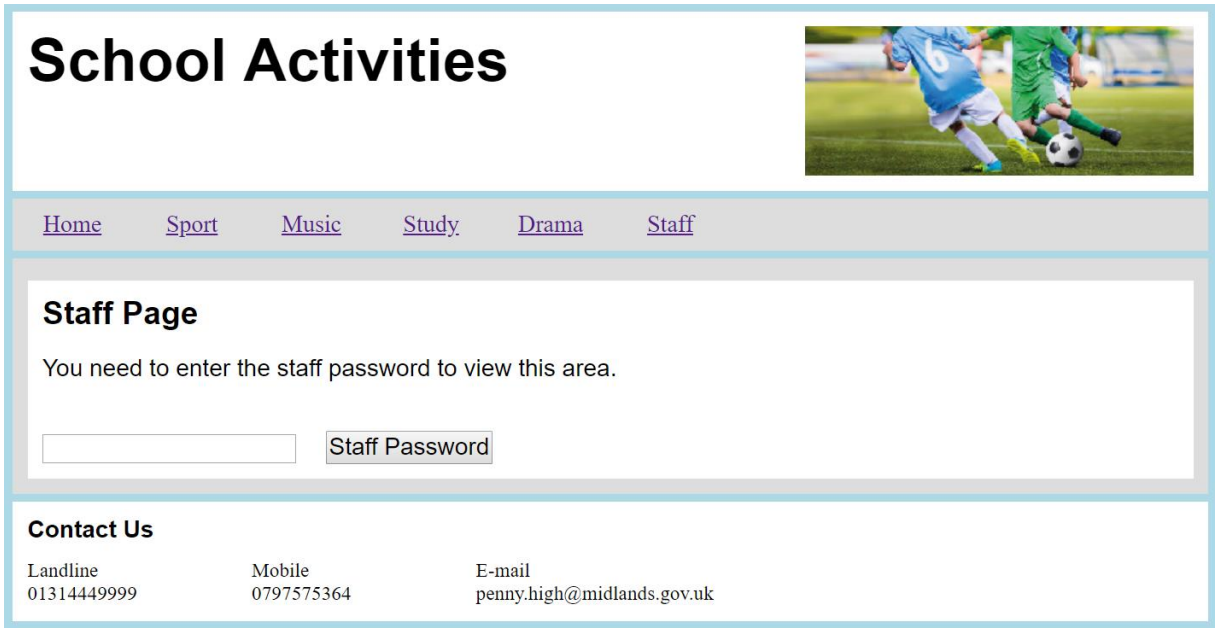
```php
<?php
// Start the session
    session_start();
?>

<?php
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        session_destroy();
        include 'staff.php';
        die();
        }
?>
```

Note: this page must also include `session_start()`, as the current session must be continued before it can be destroyed.

Worked example

The following examples use code taken from the Advanced Higher example website. You can download the example website from SQA's secure site.

To view the staff page, a password is required. The original page content remains hidden until the correct password is entered. On the example website, the staff password is 'password' and has been implemented using session variables.



The page initially hides the content, and instead displays a simple form. The form calls the PHP file `login.php` when the 'staff password' button is clicked.

```
<div>
    <h2>Staff Page</h2>
    <p>You need to enter the staff password to view this area.</p>
    <form action="login.php" method="POST">
        <input class="signInGap" type="text" name="staffpass"
        value="">
        <input class="signButton" type="submit" value="Staff
        Password">
    </form>
</div>
```

When executed, the `login.php` file:

1 connects to the current session
2 compares the user's password (from the form) with the string `"password"`
3 sets the session variable `staffLogin` to `True` , if the user's input and the string match
4 uses the PHP function `include 'staff.php'` to display the staff page

```php
<?php
// Start the session
    session_start();
?>

<?php
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $staffpass = $_POST["staffpass"];
        if (empty($staffpass)) {
            include 'home.php';
            die();
        } else {
            if ($staffpass == "password") {
                $_SESSION['staffLogin']="True";
                include 'staff.php';
                die();
                }
            else {
                include 'staff.php';
                die();
                }
            }
        }
?>
```

The above code contains alternative outcomes if the user's password field is empty or the password is incorrect.

Note: the `empty()` function used in this example is not a requirement of the Advanced Higher course.
This example could be extended to retrieve users' names and passwords stored within a database. PHP and SQL could be used to retrieve and then compare a stored password to the user's login attempt.

When the `staff.php` file is reloaded, the session variable `staffLogin` now stores `True`, indicating the user has successfully logged in. The staff page uses the value stored in the session variable to determine if the login form or the page content is displayed.

```
if ($_SESSION['staffLogin']=="True") {

        displays the page content

} elseif ($_SESSION['staffLogin']=="False") {

        displays the login form
}
```

The full code can be viewed using the Advanced Higher example website. You can download the example website from SQA's secure site.

Additional notes

The example website includes two PHP functions that are not included in the Advanced Higher course, but may be useful to candidates completing web or database projects.

The `staff.php` file begins with the code shown below.

```php
<?php
// Start the session
    if(session_status() !== PHP_SESSION_ACTIVE) {
        session_start();
        // If the session variable staffLogin does not exist then create one.
        if (!isset($_SESSION['staffLogin'])) {
            $_SESSION['staffLogin']="False";
        }
    }
?>
```

Session_Status()

The use of `include` within the login script, results in `session_start()` being called twice. Once at the top of the `login.php` file and then again when `staff.php` is included within the same page. This generates an error, as two sessions cannot be started within the same page.

The function `session_status()` prevents an error, by only calling the `session_start()` function when no other session is active.
Note: this could be avoided by including the login script within the same page.

Isset()

As stated earlier, the session variable `'staffLogin'` is used within a conditional statement to show or hide content within the staff page. If `staffLogin` had not yet been created, an error is generated by a condition that refers to a variable that does not exist.

The `isset()` function checks to see if the variable exists when the page loads. If it does not, it initialises the session variable to prevent the error from occurring.