Topic Support Guide

Cambridge International AS & A Level

Cambridge International AS & A Level Computer Science

9608

For examination from 2017

Topic 1.2.1 Networks – client-server model
Topic 1.2.3 Client- and server-side scripting

Cambridge Advanced

CAMBRIDGE International Examinations

Version 1.1
Updated: 10.03.16

A DIVISION OF
CAMBRIDGE ASSESSMENT

Contents

Introduction

How to use this guide

The aim of this guide is to facilitate your teaching of the client-server content of Cambridge International AS and A Level Computer Science – the two learning objectives on the client-server model in topic 1.2.1 Networks, and syllabus topic 1.2.3 Client- and server-side scripting. These are introductory topics that are part of 1.2 Communication and Internet technologies. The guidance and activities in this resource are designed to help teachers devise programmes of study which provide teaching time devoted to theory work as well as activities that consolidate learning.

Section 1 lists some key terms used in this topic and their definitions. Section 2 introduces the client-server computing model, client-side and server-side scripting. It then works through a web application implemented in HTML, PHP and JavaScript, that uses client-side and server-side scripting for validating data input to a simple registration web page and updating the details into a server database. Section 3 lists some online resources that will help you familiarise yourself with and understand both the theoretical and practical aspects of this topic. Section 4 gives ideas for class and homework activities.

Learning objectives

Using this document should help you guide learners in the following syllabus learning objectives:

1.2.1 Networks

- explain the client-server model of networked computers
- give examples of applications which use the client-server model

1.2.3 Client and server-side scripting

- describe the sequence of events executed by the client computer and web server side when a web page consisting only of HTML tags is requested and displayed by a browser
  - Client-side
    - recognise and identify the purpose of some simple JavaScript code
    - describe the sequence of events executed by the client computer and web server when a web page with embedded client-side code is requested and displayed by a browser
    - show understanding of the typical use of client-side code in the design of an application
  - Server-side
    - recognise and identify the purpose of some simple PHP code
    - describe the sequence of events executed by the client computer and web server when a web page with embedded server-side code is requested and displayed by a browser
    - show understanding that an appropriately designed web application for accessing database data makes use of server-side scripting

Prior knowledge

Before you begin teaching this topic you should:

- be familiar with the client-server model of computing and its applications
- understand the sequence of client and server events when a web page is requested and displayed, for client-side scripting
- understand the sequence of client and server events when a web page is requested and displayed, for server-side scripting
- have a working knowledge of JavaScript and PHP code
- understand the use of server-side scripting for accessing database data

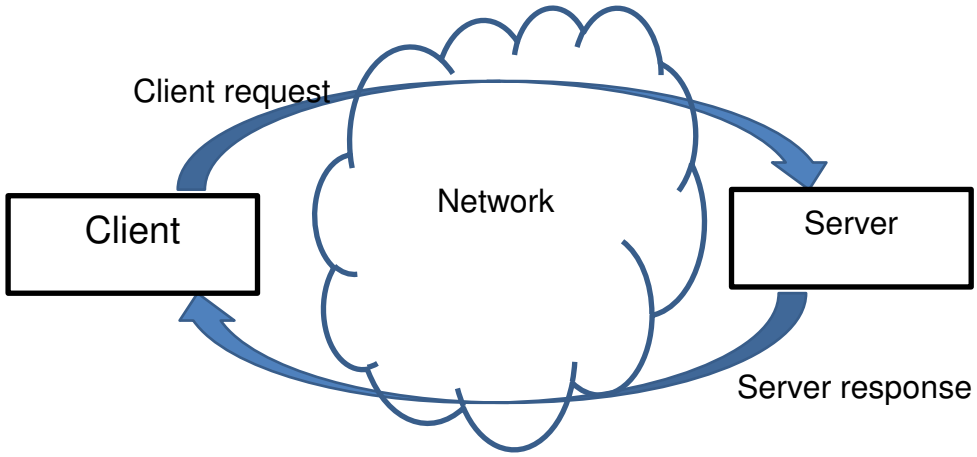
1. Key terms

Word/phrase	Meaning
cascading style sheets (CSS)	Style language that defines the appearance of HTML documents, such as font, colour, page layout.
client	A computer that requests the services provided by a server
server	A computer that provides a service requested by a client
client-server model	A system in which some of the computers (clients) request services provided by the other computers (servers)
scripting language	A high-level programming language (set of instructions) that is 'interpreted' rather than compiled (e.g. when a web page is displayed)
JavaScript	A common client-side scripting language used for interactive parts of a web page
PHP	A common server-side scripting language
HTML	A mark-up language that determines the appearance of web pages
SQL	Structured Query Language – a language used to manage/manipulate a relational database
MySQL	An SQL database implementation
Common Gateway Inteface (CGI)	General term for a program that runs on a web server and interacts with a browser (e.g. a program that queries data from a database, and returns it to be displayed in the browser)

2. Theory and worked example

2.1 What is the client-server model?

In the client-server model of computing, servers are computers on a network that run specialist software which is available to non-specialist computers ('clients'). Client computers send requests (e.g. a browser application requests a web page) across the network to the appropriate server. The server sends back the response (e.g. the web page to be displayed) to the client device.



There can be many different servers on a network that perform specialised tasks, for example:

Type	Description
File server	Stores network users' data files
Print server	Manages the printers that are connected to the network and the printing of user documents on the network printers
Communications / web server	Handles many common communications functions for the network, such as e-mail, remote access, firewalls or internet services
Application server	Shares network-enabled versions of common application software and eliminates the need for software to be installed on each workstation
Database server	Manages common databases for the network, handling all data storage, database management and requests for data

2.2 What are client-side scripting and server-side scripting?

The following web page provides a clear description of these two terms, as well as some typical uses:

http://www.yourwebskills.com/clientserver.php

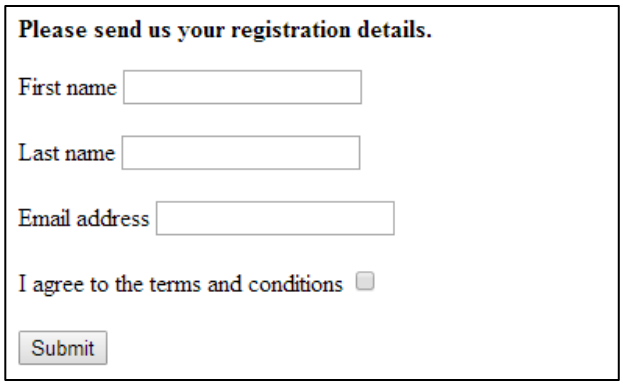
2.3 Worked example – registration web page connecting to a database on the server

Teaching note: Learners need to understand the concepts associated with the process of transferring data from a form into a database but are not expected to be able to write the code required. Example files are provided, but you will need to set up the MySQL database with XAMPP (or equivalent) to make the system work.

The example sets up a web page for submitting customer registration details and then adds the customer details to a database.

2.3.1 Creating a registration form in HTML

The HTML code below creates the form on the right which is used to collect a customer's First name, Last name, Email address and check that they agree to the terms and conditions.



Enquiries.html

```
1  <html>
2  <head><title>Enquiry Form</title></head>
3  <body>
4  <p><b>Please send us your registration details.</b></p>
5  <form name="enquiry" action="process.php" method = "post">
6  First name
7  <input type="text" name="fname" value=""><br>
8  Last name
9  <input type="text" name="lname" value=""><br>
10 Email address
11 <input type="text" name ="email" value=""><br>
12 I agree to the terms and conditions
13 <input type="checkbox" name="terms"><br>
14 <input type="submit" value="Submit">
15 </form>
16 </body>
17 </html>
```

The form is named *Enquiries1.html*. The <input> tags contain the field names: *fname, lname, email,* and *terms*. The *action* event calls the *process.php* script. Note that if the Submit button is clicked at this stage, an error will be generated since there is no *process.php* script.

This form is available as the resource *Enquiries1.html*

2.2 Adding client side JavaScript to validate the input

Now modify the *Enquiries1.html* form by adding JavaScript code to validate the input. Three checks are performed:

- a presence check on the First name and Last name fields
- a format check on the Email field
- a value check to make sure the Terms and conditions checkbox has been ticked

```
<head><title>Enquiry Form</title></head>
<script>
function validateForm() {
    var fname_present = document.forms["enquiry"]["fname"].value;
    if (fname_present == null || fname_present == "") {
        alert("First name must be filled out");
        return false;
    }
    var lname_present = document.forms["enquiry"]["lname"].value;
    if (lname_present == null || lname_present == "") {
        alert("Last name must be filled out");
        return false;
    }

    var email = document.forms["enquiry"]["email"].value;
    if (!/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(email)) {
        alert("You have entered an invalid email address!")
        return (false)
    }
    var terms = document.forms["enquiry"]["terms"].checked;
    if (!terms) {
        alert("You must agree to the terms and conditions!");
        return false;
    }
}


</script>
```

The file *enquiries2.html* contains the modified code to validate the form fields on the client side. The JavaScript is written between the <script> and </script> tags. The form tag also needs to be updated to add the *onsubmit* event attribute:

```
<form name="enquiry" action="process.php" onsubmit="return validateForm()" method = "post">
```

This calls the *validateForm()* JavaScript function that will return either true or false. If false is returned the *action* will not be processed, so the client side validation checks have to be passed before the server side *process.php* script is executed. This means that the server load is reduced if validation tests detect errors on the client side.

This form is available as the resource *Enquiries2.html*.

2.3 Server-side scripting

For the next part of this task, you need a webserver such as XAMPP. The HTML files need to be placed in the *htdocs* folder in the XAMPP installation. You can then open them by typing in the URL (e.g. *localhost/Enquiries1.html*).

Test the file *Enquiries1.html* again by running it with invalid data (e.g. leave the forename field blank) to see that errors are picked up by the server-side code. (Do not use *Enquiries2.html,* as this will have validated the form data already if JavaScript was enabled in the client browser.).

The PHP script is added between the <?PHP and ?> tags. It is called when the registration form is submitted. The PHP interpreter on the server will interpret the code and return the results in the form of an HTML page.

This script is available as the resource *process.php* This should be placed in the same folder *htdocs* as the other files on the server.

Enquiries2.html | Process.php | Enquiries1.html

```
<html>
<body>

<H1>Your registration has passed to the server for processing.</H1>

<?php
//validation of the form data
$invalid = False;

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    //fname and lname presence check
    if (empty($_POST["fname"]) or empty($_POST["lname"])){
        print "You must enter the both the first name and last name<br>";
        $invalid = True;
    } else {
    $fname = $_POST["fname"];
    $lname = $_POST["lname"];
    }
    //email presence check and format check
    if (empty($_POST["email"])) {
        print "You must enter the email address<br>";
        $invalid = True;
    } else {
        $email = $_POST["email"];
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            print "Invalid email format<br>";
            $invalid = True;
        }
    }
    //check terms box
    if (!isset($_POST["terms"])) {
        print "You must agrees to the terms and conditions<br>";
        $invalid = True;
    }
}
if ($invalid == True) {
    exit("Due to errors the data will not be added to the system<br>");
}

?>

</body>
</html>
```

The POST event in the form submission allows the named form fields to be accessed in PHP.

It is recommended that you look at a tutorial on basic PHP syntax, although the code given should be fairly intuitive if you have experience programming in another high level language.

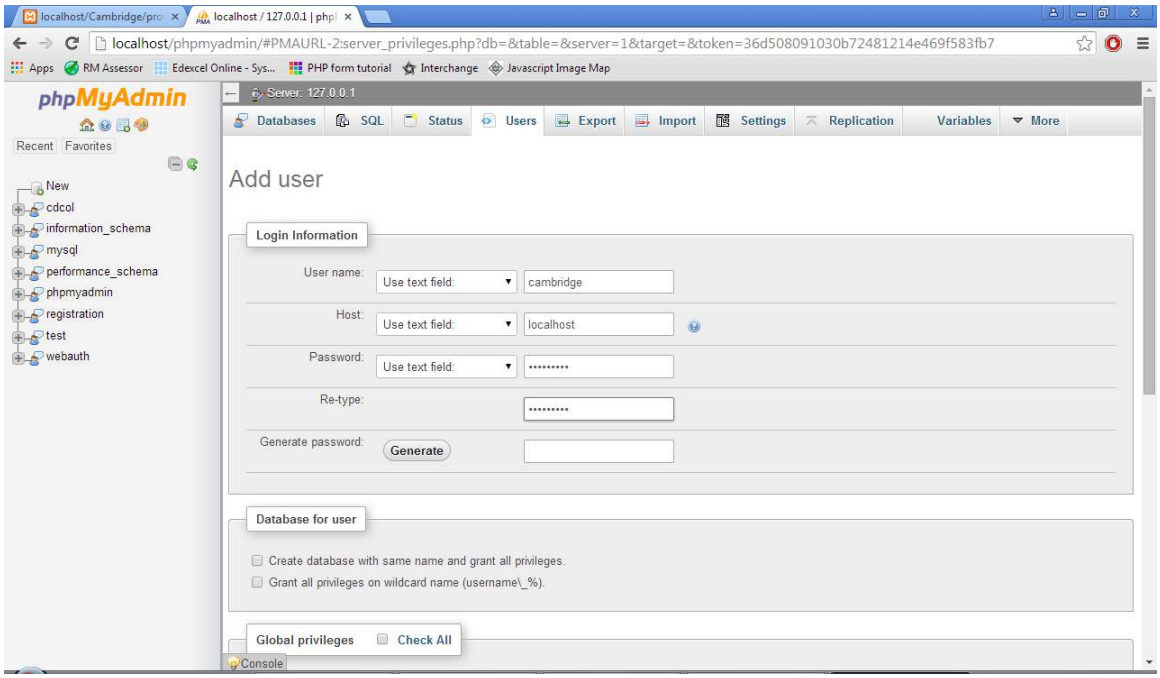
When the form data has been validated on the server side it is ready for adding to the database. Step 2.4 creates this database.

2.4 Creating a MySQL database

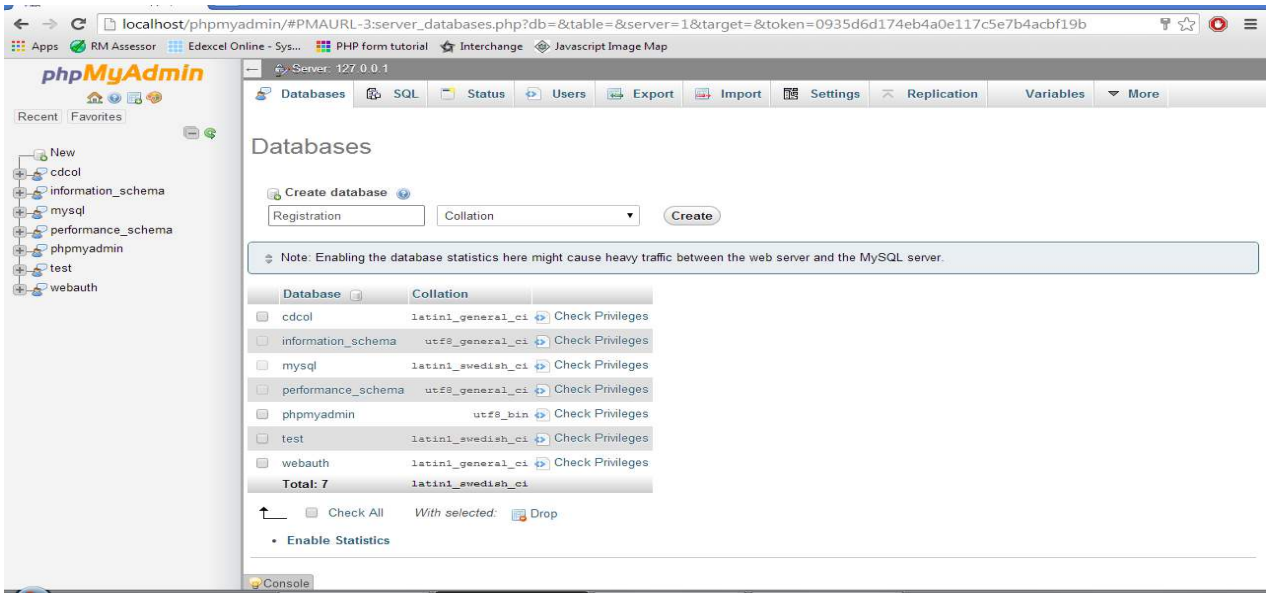
This step sets up a simple database with four fields: Customer ID (auto incrementing key field), First name, Last name and Email in a MySQL database.

XAMPP has MySQL built in and you can use this to create a database with one table.

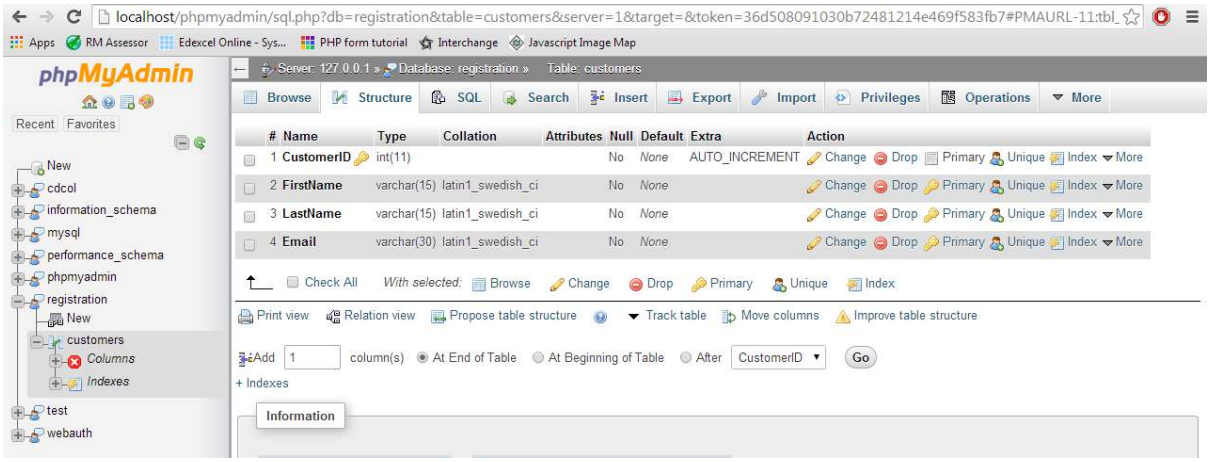
First, create a new user for the database – for this example the username and password have been set to *cambridge* and the host is *localhost.* This user should be granted all permissions, so tick all the boxes underneath.



Then create a database. This database was called *Registration*.



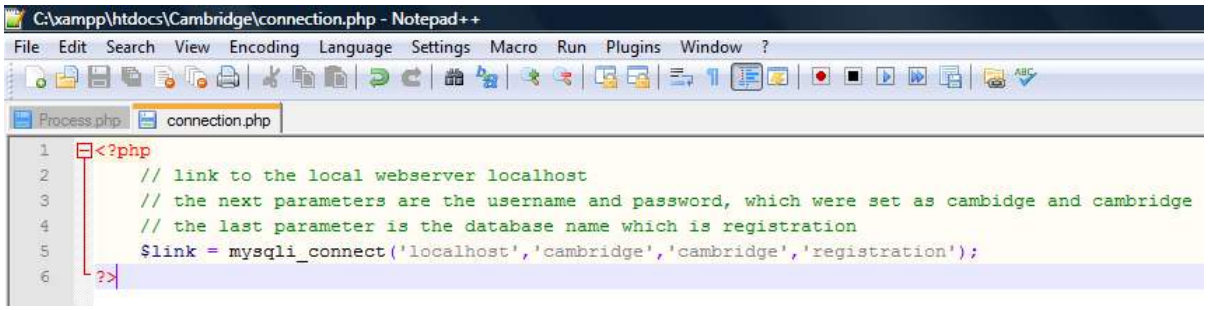
You then need to create a table called *customers*. Create four columns, *CustomerID, FirstName, LastName and Email.* Use the columns option to turn the *CustomerID* field into an auto increment field. This means that when an INSERT SQL query is run a value for the *CustomerID* field will not be required since it will auto increment.



2.5 Modify the server-side code to connect to the database that has been created

The script *connection.php* contains code to make the connection to the database on the webserver. The advantage of it being a separate script is that it can be called from any web page on the site and connecting to the database only has to be coded once.

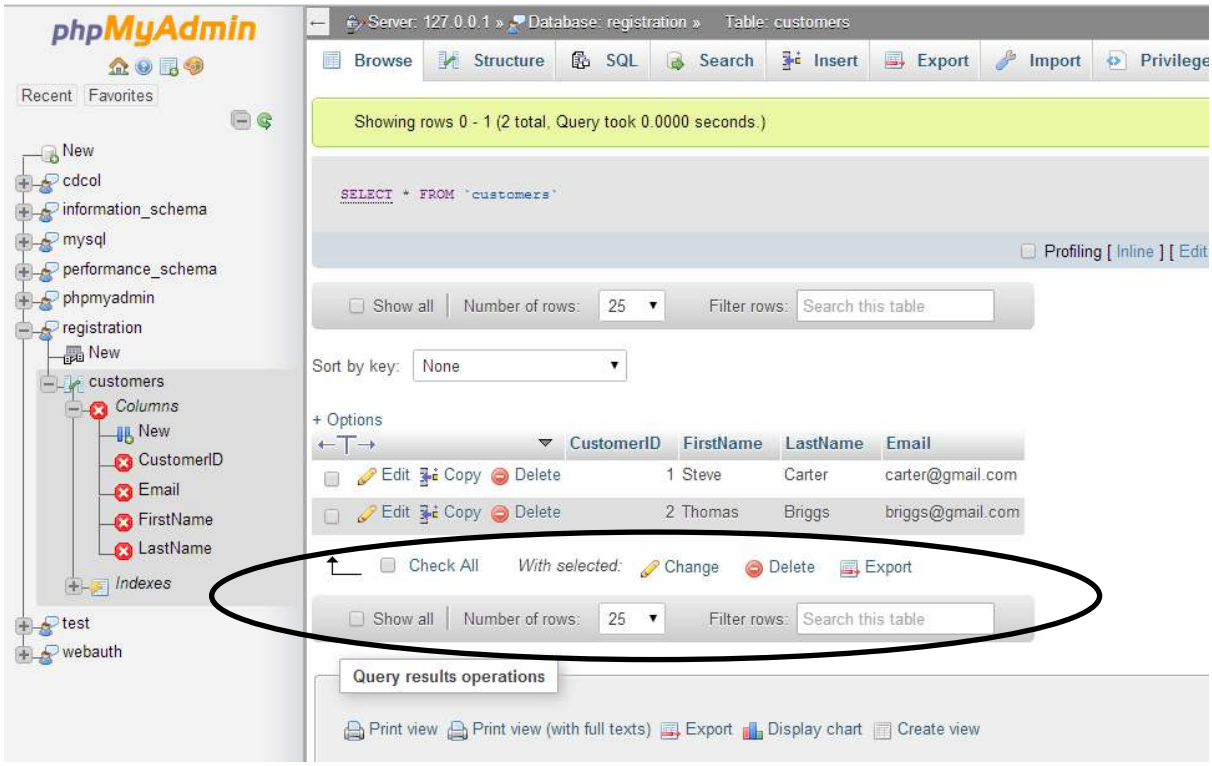
This script is available as a resource *connection.php*. This should be placed in the same folder *htdocs* as the other files on the server.



Add the following code to the *process.php* script to call the connection script, and to run the SQL query to INSERT the form fields into the fields in the database.

```
<?php
    // this runs the script in connection.php which makes the connection to the database
    require_once("connection.php");
    // an SQL query is written
    // this query is to INSERT the three fields into the customers table
    // there is no need to add the CustomerID field from the table as its properties were set as a key field
    $query = "INSERT INTO customers(FirstName,LastName,Email)"."VALUES"."('$fname','$lname','$email')";
    // execute the SQL string on the database or die if there is an error
    $result = mysqli_query($link,$query) or die(mysqli_error($link));
?>
```

When this is implemented the data from the form is inserted into the *customers* table in the database as can be seen below:



3. Online resources

The following are useful resources for understanding client- and server-side scripting as well as familiarising yourself with the languages used.

The content of websites is dynamic and constantly changing. Schools are strongly advised to check each site for content and accessibility prior to using it with learners. Cambridge International Examinations is not responsible for the accuracy or content of information contained in these sites. The inclusion of a link to an external website should not be understood to be an endorsement of that website or the site's owners (or their products/services).

3.1 Websites

A clear introduction to client-side and server-side scripting; outlines the process that occurs when a web page is requested and displayed by a browser, for embedded client-side and server-side code:

http://www.yourwebskills.com/clientserver.php

BBC Bitesize introduction to web pages, the uses and applications of client and server-side scripting and the different languages used:

http://www.bbc.co.uk/education/guides/znkqn39/revision/1

Describes client-side and server-side scripting and illustrates some uses of server-side scripting in PHP code:

https://www.washington.edu/accesscomputing/webd2/student/unit5/module1/beyondhtml.html

Useful slide presentation outlining the difference between client-side and server-side scripting and showing the sequence of events involved:

http://slideplayer.com/slide/4906328/

PHP and MySQL basics. This tutorial assumes knowledge of HTML, CSS and programming in a high level language. It shows clearly how a database can be used to generate content that is then fed into a client web page. It is a very useful step-by-step guide for learners written especially with A Level Computing projects in mind:

http://community.computingatschool.org.uk/resources/349

W3Schools.com contains numerous tutorials and references for HTML, CSS, PHP, SQL and JavaScript:

http://www.w3schools.com/

JavaScript interactive tutorial course:

http://www.codecademy.com/tracks/javascript

3.2 Videos

12-minute tutorial on creating a simple form data presence check in JavaScript:

https://www.youtube.com/watch?v=y5UEXujzSag

10-minute tutorial on creating a login script in PHP for a username and password that is matched against records in a MySQL database:

https://www.youtube.com/watch?v=5XpBzLVHkPY

3.3 Software

XAMPP server is free and includes PHP and MySQL:

https://www.apachefriends.org/download.html

Notepad++ is a useful text editor:

https://notepad-plus-plus.org/download/v6.7.9.html

4. Class and homework activities

4.1 Homework questions

1. Describe two advantages of a client-server model.

2. Describe the advantages and disadvantages of client-side scripting.

3. Explain how server-side scripting is processed by a web server.

4. The following questions refer to this code which is stored in *enquiry.html*:

```
01 <html>
02 <head><title>Enquiry Form</title></head>
03 <script>
04 function validateForm() {
05 var fname_present = document.forms["enquiry"]["fname"].value;
06     if (fname_present == null || fname_present == "") {
07         alert("First name must be filled out");
08         return false;
09 }
10 }
11 </script>

12 <body>
13 <p><b>Please send us your registration details.</b></p>
14 <form name="enquiry" action="process.php" onsubmit="return validateForm()"
   method="post">
15 First name
16 <input type="text" name="fname" value=""><
17 <input type="submit" value="Submit">
18 </form>
19 </body>
20 </html>
```

a. Identify the line numbers where JavaScript has been inserted.
b. Explain what the JavaScript code in lines 05 to 09 does.
c. Explain the parameters in the <form> tag in line 14.

5. The following code is stored in *process.php* and is called from the *enquiries.html* form in question4.

```
01  <html> <body>
02  <?php
03  //validation of the form data
04  $invalid = False;

05  if ($_SERVER["REQUEST_METHOD"] == "POST") {
06  if (empty($_POST["fname"])){
07     print "You must enter the first name<br>";
08     $invalid = True;
09  } else {
10  $fname = $_POST["fname"];
11  }
12  }
13  if ($invalid == True) {
14     exit("Due to errors the data will not be added to the system<br>");
15  }
16  ?>
17  </body> </html>
```

a. Explain what happens when this code is called when no value has been entered into the fname field in the form.

6. The following code is then inserted after line 16 of the code in question 5.

```
17  <?php
18  $link = mysqli_connect ('localhost','user','password','database');
19  $query = "INSERT INTO customers (FirstName)"."VALUES"."('$fname')";
20  $result = mysqli_query($link,$query);
21  ?>
```

a. Explain what happens when the code in lines 19 and 20 is executed.

4.2 Homework solutions

1. Two advantages of the client-server model are:

   - Resources (such as files) are centralised and thus access is possible from any client on the network.
   - Central management of resources gives greater security when compared to each client having to patch software or update virus checking software individually, that is, these issues are taken away from the responsibility of each individual user.

2. Client-side scripting

Advantages:

- If processing is performed on a client it reduces the processing load on a server since many errors can be eliminated before data is transmitted to the server.
- Web pages can be made more interactive.

Disadvantages

- Not all clients will run the scripting language, so there is no guarantee that a script will have been run. This means that server-side scripting will still be required for form validation and this leads to a longer development time if both client- and server-side scripting is implemented.
- Client-side scripts may be turned off for security reasons since scripts can run automatically.

3. When a server receives a processing request from a client it will look at the file extension being called. A file named *process.php* has the file extension *.php*. The server will then pass the script to the appropriate interpreter for processing. If the file extension had been *.py* it would be a Python type file, so would be passed to a Python interpreter for processing. The CGI (Common Gateway Interface) on the server determines the mappings of the file types to the appropriate interpreters and it deals with the transfer of variables (e.g. POST / GET methods) to relevant scripts that are to be run. The results of this processing are then output as an HTML page that can then be sent back to the client and rendered by the client browser.

4.
   a. The <script> </script> tags indicate that the server-side script has been included between lines 3 and 11.
   b. Line 5 picks up the value of the fname field from the form (defined in line 16) and then tests to see if the value is null or empty in line 6. If it is, an error message is displayed (line 7) and the value False is returned (line 8).
   c. The <form> tag has a number of attributes:

- The *name* attribute gives the form a name by which it can be identified.
- The *action* attribute gives the name of the script to be executed on the server when the submit button is clicked.
- The *onsubmit* attribute calls the validation route client-side. If false is returned the submit is cancelled.
- The *method* attribute determines how the data variables will be transmitted to the script (POST / GET).

5. The code is using the POST method to pick up the variable that was submitted by the form. The $_POST["fname"] code picks up the value in the field fname. The code checks to see if it contains an empty value (line 6), and, if so, generates an error message (line 7). If the error message is generated it uses the print command to output the message – this is then displayed in the final HTML page that is output. The variable *$invalid* is then set to true in line 8. This is checked in line 13 where an additional error message will be printed because the value of *$invalid* is true.

6. Line 19 sets up the value of $*query*. This is set to an SQL string that contains the code to insert the value of the *fname* variable into *FirstName* field in the *customers* table. Line 20 then calls the MySQL database with the *$query*. The output is stored in *$result*. This may return values or an error message, but in this case, if all works successfully, it will not output anything since it is inserting data into a table rather than selecting it.